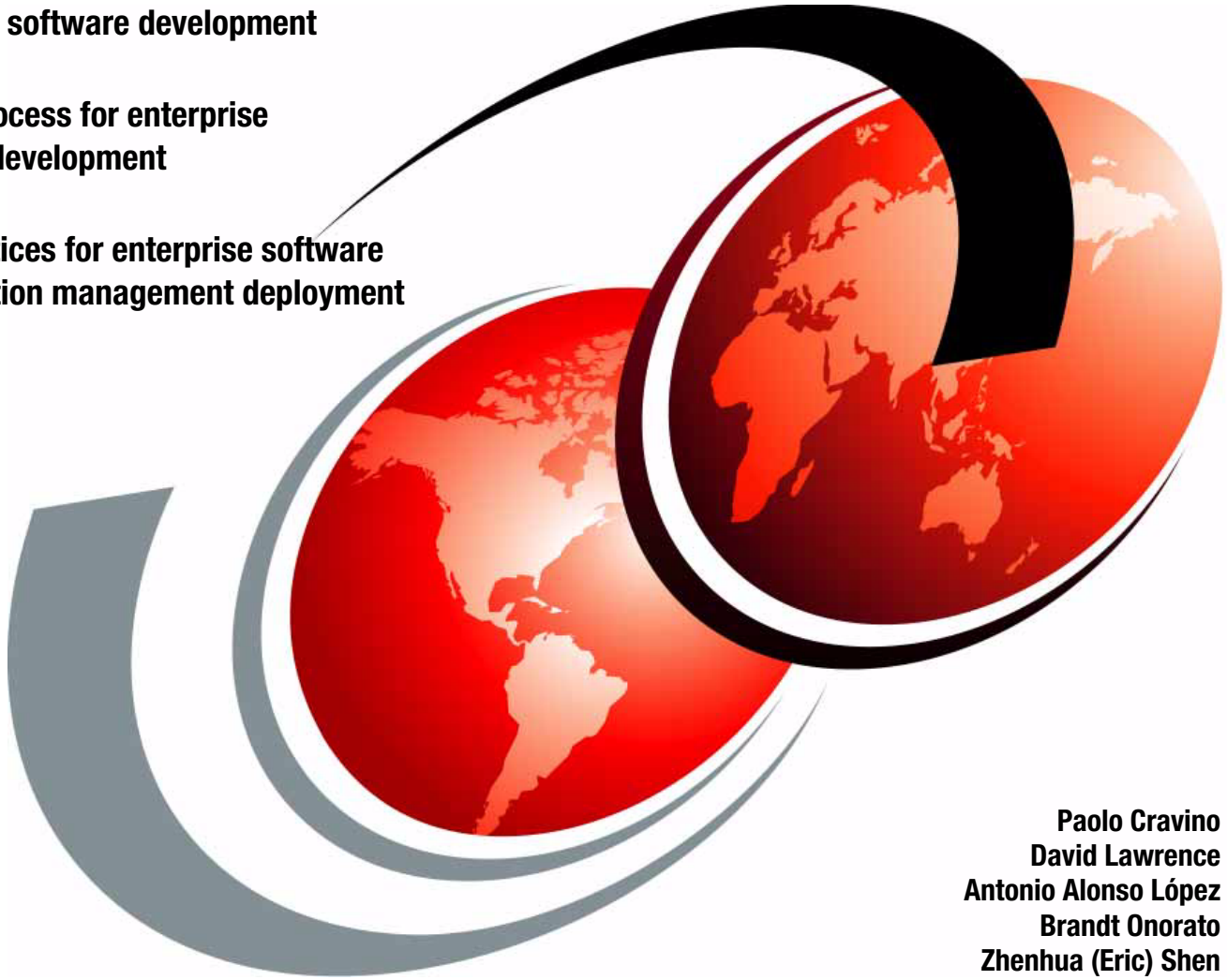
Rational. software

IBM

Enterprise Software Configuration Management Solutions for Distributed and System z

- Application lifecycle management for enterprise software development
- Unified process for enterprise software development
- Best practices for enterprise software configuration management deployment



Paolo Cravino
David Lawrence
Antonio Alonso López
Brandt Onorato
Zhenhua (Eric) Shen

Redbooks

ibm.com/redbooks

International Technical Support Organization

Enterprise Software Configuration Management Solutions for Distributed and System z

January 2009

SG24-7598-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (January 2009)

This edition applies to Rational Developer for System z: 7.1.1.2, Rational ClearCase: 7.0.1.0, Rational ClearQuest: 7.0.1.0, Rational Build Forge: 7.0.2, and WebSphere Studio Asset Analyzer: 5.1.1.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Build Forge®
CICS®
ClearCase®
ClearQuest®
DB2®
IBM®
IMS™
MVS™
OS/390®
RACF®
Rational®
Redbooks®
Redbooks (logo) ®
REXX™
System i®
System z®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

Snapshot, and the NetApp logo are trademarks or registered trademarks of NetApp, Inc. in the U.S. and other countries.

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Visual Studio, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

In this IBM® Redbooks® publication, we describe how you can implement an application lifecycle management solution with Rational® Developer for System z®, Rational ClearQuest®, Rational ClearCase® with the z/OS® extensions, and Rational Build Forge®.

Our target audience includes java and COBOL developers, architects, team leads, and release and production management staff who are responsible for managing the life cycle of cross-platform or enterprise-wide applications that are composed of host z/OS and distributed components.

In this book, we present an approach that we call Enterprise Software Configuration Management (ESCM) and discuss the problems with releasing cross-platform applications in a secure, robust, and reversible manner.

In chapter 1, we explain the rationale for producing this book and present a few case studies.

In chapter 2, we describe the challenges that we observed in many IT enterprises as they struggle to integrate the distributed and mainframe sides of the house. We also discuss both the costs and benefits of implementing an ESCM process and the risk of not doing so.

In chapter 3, we describe the idealized ESCM model, specifically, what a successful ESCM looks like from an organizational and functional perspective (as opposed to from a technical perspective). We present some suggested (and effective) best practices.

In chapter 4, we present a detailed implementation with several alternative strategies that we use in this book in the context of an overall ESCM life cycle, where Rational Developer for System z is positioned as the primary developer interface. We show how we implemented Rational ClearCase with the z/OS extensions to run native z/OS builds and describe how we use the integrations between Rational Developer for System z, Rational ClearQuest, Rational Build Forge, and their associated integrations to implement an end-to-end ESCM life cycle.

In chapter 5, we discuss what to consider as you begin the implementation of an ESCM solution and prepare you to structure your teams, assign responsibilities, and plan for general tasks and activities.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Figure 1 From left, Antonio Alonzo Lopez, Brandt Onorato, David Lawrence, Paolo Cravino, and center, Zhenhua (Eric) Shen

Paolo Cravino is a Software Pre-sales Senior IT Specialist on the Rational brand team in Milan, Italy. Before joining Rational Software in 1999, he was a Configuration Manager for a major Italian banking group. In Rational, he started as a Software Engineering Specialist working with several customers in the software configuration management (SCM) area on both distributed and mainframe environments, then moved to a Service Manager responsibility, and last a Technical Sales Manager. Now Paolo is a Senior IT Specialist Solution Leader for Italy and SouthWest Integrated Operating Team.

David Lawrence is an Enablement Engineer in the Enterprise Tools and Compilers Ecosystem team. He initially joined Rational Software in 1996 after several years as a customer of Rational ClearCase. He was a consultant and trainer on Rational SCM solutions to many customers in most industries (banks, telecoms, retail, and so on). He led the first installation and configuration of the ClearCase z/OS extensions in a large European Bank in 2001. He is now responsible for enabling IBM Rational Sales teams and strategic customers in implementations of Rational solutions for system z.

Antonio Alonso López is the IBM Rational Account Manager in Spain. He brings 10 years of experience in the software engineering field. Prior to joining Rational in 2002 he was the Configuration Manager in a major industry software automation company. At Rational he spread the Rational configuration management tools in large corporations with a particular focus in the Finance, Insurance, and Retail sectors where mainframe platforms are very common.

Brandt Onorato has been with Rational Software for the last three years providing mentoring and technical expertise with the enterprise-wide ClearCase Change Management Solution. Prior to Rational Software, Mr. Onorato spent 20 years assisting clients with implementing access controls, installing and configuring operating system software, and establishing software change controls. Mr.Onorato held positions as a z/OS Systems Programmer, Data

Security Specialist, and for the last 10 years Software Configuration Management consultant. During the last three years he provided technical guidance and expertise with the integration of mainframe assets into the ClearCase Change Management Solution, which included, requirements analysis, methodology and software life cycle design, scope of inventory, source migration, and design and coding of build and deploy processes. Mr. Onorato is a published author, and he creates technical manuals, instructional technical classes, executive briefs, and functional specifications.

Zhenhua (Eric) Shen is an Enablement Engineer in Rational IBM who focuses on SCM solutions for system z and i. Eric has 10 years of software development and enablement experience. Prior to joining Rational in 2002 he was the Software Developer Lead in a major industry software automation company. Eric has a masters of science degree in computer engineering from Northeastern University and a MBA degree in general management from Babson College.

Thanks to Joe DeCarlo, Manager for Special Projects at the ITSO in San Jose, CA and the following IBM colleagues for their contributions to this project.

Beng Chiu	Omkar Nimbalkar
Bob Kennedy	Ray Green
Charles Chu	Reginaldo Barosa
Dale Newby	Sanjay Chandry
Dudley Thomas	Scott Laningham
Duncan Upton	Stephen Hunt
Emeka Igbokwe	Steve Borriello
Jack Verstappen	Steve Rubin
Jim Hildner	Steve Seifert
John Casey	Thomas Pietrcollo
Larry Cox	Tony Lee
Louis Gentry	Tsuneo (Sam) Horiguchi
Manual Moreno Villares	

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Why this book?

In most companies, business software evolves over time, by design or through acquisitions, into complex, multiplatform environments. Frequently, applications use components that run on both distributed and mainframe operating platforms. Using existing assets in modern application architectures means organizations must modernize software development practices by transitioning from a multiple-silo mentality to a more collaborative mind set. Technology and customer demand are driving deeper and more complex interdependencies between the traditional mainframe data center and distributed, network application middleware and front ends. Because changes to one part of an application on one platform can affect other parts of the application on another platform, effectively supporting application change requires an enterprise scale to synchronize activities that relate to software development, management, and release.

In this book, we describe an approach to a single unified process for software development across multiple platforms, using a single repository for enterprise software configuration management. With this approach, a single configuration management solution manages all software development across the enterprise.

In this chapter, we discuss some of the reasons why a single configuration management solution across platforms has become necessary in today's modern information technology (IT) environments. We also:

- Review an example scenario
- Introduce the IBM Rational approach to solve the business problem with an example scenario
- Discuss the benefits of a single unified software configuration management (SCM) solution
- Explain the impact of not having a single point-of-control for change and release management

1.1 Business scenarios

It is a fact of life that mature IT systems exist on multiple hardware and software platforms, and that business software continues to be developed on each unique platform. What are the reasons for this complicated state of affairs?

The first reason has to do with the mantra from about fifteen years ago: “The mainframe is dead.” This theme, throughout the IT industry, led to many efforts to move software off of the mainframe onto other platforms, which includes UNIX®, System i® (formerly i Series), and Windows®. Certainly the popularity of these platforms grew naturally without “The mainframe is dead” movement, but this movement did contribute to the greater utilization of other platforms, which, in turn led to new software being developed on multiple platforms leaving, for a time, the tried- and-true mainframe applications in a state of maintenance only.

The second reason was the dot.com boom that both nurtured and lived off of the maturing of the Internet. At the beginning, everyone wanted their bit of the Web real-estate, from Fortune 500 companies to the mom and pop businesses in our local communities. Web sites at first seemed to be a whim, but soon businesses realized how effective the Internet could be as a way to interact with customers, business partners, and internal users. Our on demand world was born, which led to the rapid development of new software and tools that could utilize the Internet and the platforms that supported it.

The advance of the Internet and the way information was being delivered led to the advances in architecture with existing platforms, such as the mainframe and AS400. With the majority of the raw data still residing on these platforms, the need to interact with other platforms became paramount.

This rebirth and birth of technologies on all platforms led us to this multiplatform reality, where each platform has its strength and businesses need to exploit each platform to its maximum advantage.

1.2 Configuration management solutions

As the above software development dynamic was evolving, so too were *configuration management* (CM) solutions. In fact, each CM solution evolved completely independent of others, according to the needs of its unique platform and the perceived way software was developed and deployed on that platform.

Many companies created home-grown CM tools to meet their own specific needs. Software vendors also responded with a plethora of CM solutions, most with a niche in a single platform. Businesses responded by selecting one of these solutions or creating their own to meet the needs of the moment for their current architecture.

Meanwhile, either by acquisitions or responding to business need, companies were busy adding platforms to their software architecture, which meant that companies inherited additional CM tools, needed to acquire an additional tool to meet the needs of the new platform, or had to customize their home grown system to meet their new requirements. So as business grew across platforms, so did the number of configuration management solutions that are part of businesses’ software portfolios.

Naturally, many companies experienced a crisis of multiple CM solutions. In some cases, not only are there different CM tools for each platform, but even multiple CM tools for a single platform, combinations of vendor packages, and home grown solutions. This particular

phenomenon usually occurred after acquisition of other companies and when the cost of migration and retraining of development teams was perceived as too great.

In addition, some companies also faced a customization factor with their CM tooling, for example, a vendor package was selected and the CM administrators, in response to both business drivers and usability issues, customized and enhanced the solution to their specific usage model over time. This customization factor ties the company and staff resources to the unique tooling for the foreseeable future. Yet, as staff attrition occurs and as the business model accelerates, they cannot modify the tool to keep pace with the business needs. Development teams that are tied to the CM tool cannot keep up with the changing business landscape.

Those scenarios are the most common historical reasons for development teams following different processes, with cross-platform dependencies. Now add today's IT needs, for example, compliance mandates, such as Sarbanes-Oxley, governance, and management of geographically distributed development teams, and we have a very complicated software development process and a complex business model to navigate.

1.3 Example scenario

In this section, we introduce the example scenario.

1.3.1 Introduction

By design or through acquisitions, management in the example scenario had business software that evolved over time into multiplatform environments, which means that parts of their business applications ran under Windows, UNIX, System z (z/OS), and System i. Most all of these applications communicated with each other either by feeding information to one another or as an interface to the user, which is a condition that is common in mature IT systems, known as *cross-platform dependency*. Configuration management solutions must be able to coordinate these type of applications across platforms.

Each platform solves a true business need and performs its tasks as directed; therefore, the solution is not to consolidate applications onto one platform. Rather, the goal is to understand a business's working technology, how it is evolving in the modern workplace, and how the business uses this technology.

In this example scenario, each platform had a different process in place for migrating code through its life cycle and onto its production environment, and each platform had its own configuration management systems in place to aid with this migration. Coordinating these different configuration management systems is difficult at best, which was true in both of these examples.

1.3.2 Example scenario: Transportation

The example scenario relates to the transportation industry. They designed their current configuration management system in-house. Although it served them well for years, compliance issues arose that pertained specifically to Sarbanes Oxley. Their home-grown system did not actively include version control, so they could not accurately reproduce artifacts that were run against their production system. They also could not accurately track the deployment of their artifacts through their development life cycle. Their business applications were evolving into this cross-platform architecture.

There were environment changes that were also factors, after acquisitions that had its own development staff. Also, the management of the example scenario wanted to outsource some of their System z software development, which meant that development would occur at two separate locations; therefore, they needed to maintain the same process in both locations and coordinate the releases of these cross-platform applications.

1.4 Introduction to our approach with the example scenario

The company in this example scenario needed a way to manage their build, compile, and deploy processes for their z/OS artifacts. The z/OS extensions feature of ClearCase offers a way to interface with the mainframe while using ClearCase as the single repository for both the mainframe and distributed artifacts, which allows for a single point-of-control and a single process of development for both platforms.

In this chapter, we describe how to obtain build management using native ClearCase with the z/OS extension feature. In subsequent chapters, we will describe integrating the software development process with ClearQuest, adopting Build Forge to manage both System z and distributed builds, and using WebSphere® Asset Analyzer to determine cross-platform dependencies for system z.

The build and deployment interface between the ClearCase host and the target z/OS system is a combination of PERL, Remote build commands, and BCL.[1] The combination of these three components can essentially run any mainframe job or utility from the centralized location of the ClearCase repository, which we describe in more detail in Chapter 5.

1.5 Details of our approach

The migration or assignment of the element-level attributes is not complicated. If the migration is from an existing configuration management tool, whether it is a home grown tool or a third-party tool, the data should be available within the tooling, usually through a simple report. As part of the migration, we assign the element level attributes as we import the source files into the ClearCase repository. In the example scenario described earlier, we deliver the information in spreadsheet form, read it into the import process as a text file, and generate the commands to assign the attributes.

If there are no metadata or element level build attributes as part of the existing system, we must perform an inventory analysis, usually on the mainframe, using the amblist utility. From that output, we can determine the components.

The design of the ClearCase architecture relates to the current z/OS infrastructure or current life cycle design. It can be architected within ClearCase, either as a one-to-one relationship that is identical to what exists or modified to suit client needs. In the case of the example scenario, we implement the ClearCase architecture as a one-to-one relationship to their current z/OS architecture.

After the design is laid out, and as projects begin, the ClearCase administrator can enter the project attributes and essentially define the migration path of the individual project as it relates to the z/OS supporting infrastructure and life cycle. We can define the life cycle in a serial fashion, meaning that every project follows the same life cycle, or that projects can have the

[1] BCL stands for "Build Control Language," which is proprietary to ClearCase z/OS Extensions. It is similar to normal Job Control Language with some additional features specific to ClearCase.

flexibility to point to different supporting environment infrastructures based on the requirements of the individual projects.

1.6 Benefits of a unified ESCM solution with our example

In this section, we provide the benefits, of using a unified ESCM solution.

For the example scenario, the ClearCase Z/OS Extensions solution positioned them to be able to exploit other rational solutions, such as ClearCase Multisite. The example scenario management planned to off-shore some of their z/OS development while maintaining their process of development in both locations. The unified ESCM solution also allowed for very low administration with regard to their build process because after a build engine is in place, the only maintenance is to provide for new target libraries should the z/OS infrastructure be changed, such as an upgrade to the compilers, in which case they must add new libraries. Should they add a new element type to the build process, they only need to modify the build engine if a new development language is added to their z/OS development process.

Both implementations saw additional value by allowing z/OS development teams to utilize graphical user interface (GUI) tools to assist in their development process, utilizing ClearCase diff/merge utilities and code regression testing as they moved through their life cycle.

Using project level attributes also allowed flexibility within the life cycle. Although in the example scenario case management wanted to have a fixed life cycle, the project level attributes allow them to add z/OS infrastructure to their life cycle, for example, if they want to add quality assurance (QA) capabilities, they can easily change their migration path dynamically with project level attributes. The only restriction is in creating the z/OS infrastructure to support it.

1.7 Additional benefits

In addition to the benefits that we discussed in 1.6, “Benefits of a unified ESCM solution with our example” on page 5, additional benefit opportunities might exist for most enterprises. Each of these benefits vary from scenario-to-scenario, depending on architecture, usage model, and the development tools that are used:

- Reduced software license costs
- Reduced CPU utilization from host by moving some development off of the host
- Sunset variety of development tools across different business units to support change and release management
- Simplified process for delivering code changes into production
- Simplified coordination of code deployment for cross-platform applications
- Simplified consolidation of developer training that is required to maintain various tools across platforms
- Common metrics to measure software quality
- Common project structure across platforms
- Cross platform consolidation and sharing of skill sets
- Common repository for all development artifacts, documentation, and approval process linked to a common change request

1.8 Summary

In our example scenario, we highlight how an ESCM solution can solve software development issues. Further observations of our customer landscape also reveal a need to collaborate, consolidate, and modernize development teams. Overcoming the obstacles to cross-platform application development and modernization requires an enterprise-wide approach to change and release management. The best approach provides a consistent process paradigm and common tools to reduce cost, minimize risk exposure, and improve development agility, which helps organizations to ensure that the right versions of the right applications are available at all times, and enables them to provide an audit trail of changes across the software life cycle to prevent application failures and to help meet increasingly stringent regulatory requirements. Figure 1-1 on page 7 shows an illustration of an ESCM architecture.

An effective ESCM solution also enables organizations to automate and enforce best-practice development processes that enhance collaboration and productivity across distributed and mainframe development teams at every stage of the application life cycle. The ESCM solution also provides controlled, highly-secure access to the information that practitioners in various roles need to create, update, build, deliver, and reuse business-critical software assets, no matter where they are located.

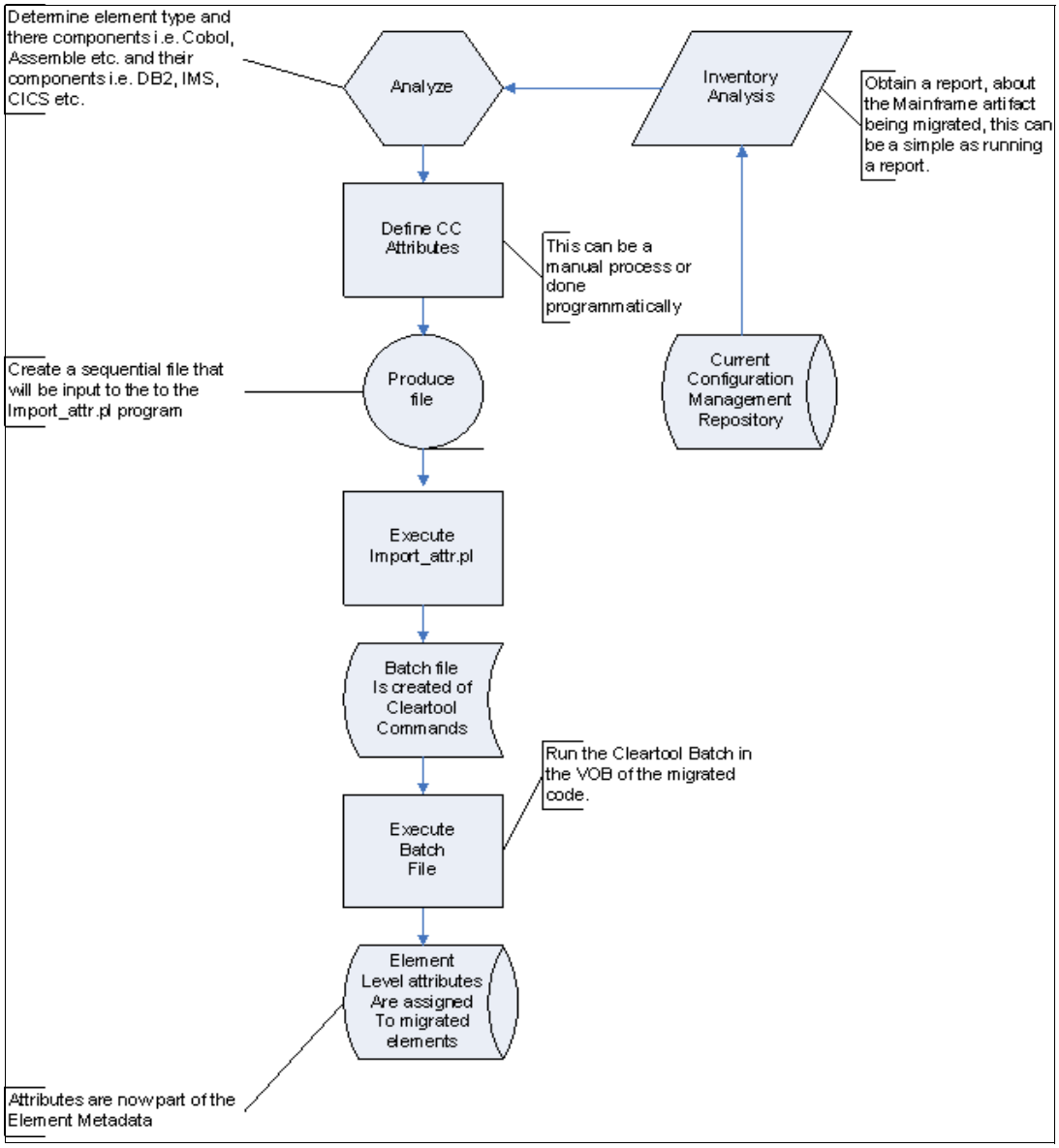


Figure 1-1 ESCM supporting architecture

By providing an enterprise approach to managing assets with consistent processes, common reporting, and common project management across z/OS, UNIX, Windows, and Linux® platforms, these Rational products bridge the gap between mainframe and distributed development. As a result, organizations can increase project collaboration, improve release coordination, and unify development and modernization efforts for heterogeneous IT environments.

2

An Enterprise Software Configuration Management approach

In this chapter, we discuss the importance of defining the right software configuration management (ESCM) process in large enterprises. We explain the negative impact of having disconnected and platform-dependent configuration management processes and introduce the distributed versus mainframe typical practices. Organizations will obtain advantages when they adopt an unified configuration management process, but they will also face some challenges. At the end of this chapter, we discuss benefits and return of investment.

2.1 Why have an ESCM process?

Enterprises with a significant software development activity usually have a large number of applications to maintain and to evolve. They frequently launch new projects and new and aged code pieces are integrated. A large development community is in charge of these activities. Some organizations have thousands of developers who are often spread in multiple development sites in different cities, states, or countries.

Commercial configuration management tools, such as IBM Rational ClearCase, help these organizations to manage change effectively and provides support for large scale development projects. Configuration management tools usually help to effectively manage the evolution of a change request from its identification to its implementation through the software development life cycle. Most development organizations define life cycle stages as analysis, development, testing, and deployment.

Still, each organization has its own culture and tools that need to be adapted to the way that the organization works, which is dependent on its needs. It is a common practice to define what is called the configuration management process and adapt the tool to fit that process with the goal of developing efficiently.

Defining a software configuration management (SCM) process means to:

- Define roles and responsibilities in the development life cycle: Developers, Integrators, Project Managers, Testers, and so on.
- Define the activities through the life cycle, for example, Deliver, Promotion, Rollback, and Baseline publications.
- Establish the framework for parallel development support:
  - Do developers edit same source code concurrently?
  - Does the development team have to fix defects on the production release and work in parallel toward a new release?
- Reuse: Is there any need for teams of developers to reuse code?
- Define the change request approval workflow.
- Build management process definition:
  - Daily build procedure.
  - Build reporting and notification system.

Often, configuration management tools' deployment initiatives fail due to a lack of thought and effort to analyze and implement the right process. Defining the right process in an enterprise-scale configuration management deployment is critical. There are a lot of challenges, also but the invested effort will benefit the efficiency of the development process.

2.2 Current enterprise challenges

Software development is a critical business process in many organizations, especially in software development organizations. Also large financial, insurance, and government institutions have significant software development teams that maintain and create applications to respond to business demands.

There are no two identical companies in terms of configuration management needs. However we can make the following statements as generic characteristics of large corporations:

- Multiple execution platforms exist. IBM traditional mainframe applications coexist with new running platforms that UNIX, Linux, and Windows support.
- Multiple language environments coexist. Traditional languages coexist with new trends languages: Cobol, PL1, Java™, C, C++, and Assembly.
- Applications are multiplatform. It is seldom that a business application is built in one single language and runs in a single-execution platform, for example, the user Interface layer is built in Java or HTML, and the application logic is built in Cobol.
- Development department structure usually is platform dependent. The mainframe team and the distributed team goals are to support each other's technology expertise. Communication and skills barriers exist and are difficult to overcome.
- Software development support tools, which includes configuration management tools, are usually platform dependent. Most companies have in common the:
  - Need to respond fast to market imperatives.
  - Development occurs in different sites that are geographically dispersed.
  - Need to evolve to more flexible architectures, following the service-oriented architecture (SOA) trends.
  - Need to establish control of governance systems in the development process to respond to regulatory compliance and auditabilty demands.

It is a very common situation that configuration management process are also platform independent (mainframe and distributed), while, as previously stated, the application is a unique entity. Very often, both configuration management processes are disconnected with huge implications in the software development efficiency and quality.

2.2.1 Disconnected processes

Having a different configuration management process for the distributed and mainframe platform brings many challenges to development teams. Each process is normally supported by either a commercial tool or an inhouse solution, but it solves issues that are related to the execution platform that was meant to be. Having configuration management processes disconnected means that change requests and the affected source code follows different paths depending on a language and build platform. Here are some challenges that organizations face because of this situation:

- Deployment and build process synchronization: A change request that affects to both mainframe and distributed code is build in an isolated way in each platform, which makes it difficult to obtain the build of materials of the complete build and to manage situations of partial build failure. Synchronization mechanisms are difficult to implement without the help of tools that can manage diverse environments.
- Reverse traceability: What version do I have in production? This is a very common question in the every day life of a developer or software project manager. Build and deployment processes are usually automated, but because processes are isolated per platform, information about build results and contributions to an executable are never recorded in the system. It is very difficult to guarantee the correct access to the production version or to the source code in both platforms.
- Automated build and deployment processes coexist with manual-urgent processes: Most of the organizations independently solved the configuration management in each platform. It is also very common to have manual processes outside of the tool's control to manage the urgent corrective actions. The result is usually an inconsistent process that is difficult

to register and to track changes and the traceability between the source code and executables.

Disconnected processes result in an inefficient development process and poor quality in the final software product release. Through the understanding of typical mainframe and distributed practices, it is our goal to purpose a unified *platform-independent* process to fit different scenarios with IBM Rational tools.

2.2.2 Typical distributed practices

There are plenty of tools in the market of software configuration management that provide configuration management support for the distributed development environment. Practices vary from organization to organization, but as a generic, we can state that:

- In the distributed area, there is a large percentage of new development activities that must coexist with maintaince activities.
- Parallel development is required:
  - Several developers concurrently share and edit source code.
  - The tool must support maintaince and ongoing development.
- Version control and version history support is required.
- SCM tools are programing language agnostic. Tools store elements or artifacts but have no knowledge of language construction rules.
- Source code is often stored in the SCM product database. The tools usually follow a checkout, checking paradigm control to manage access control.
- SCM distributed tools are usually not in charge of build and deployment processes, which leaves this responsibility to a third-party system that quite often is a set of scripting.
- Either commercial or freeware SCM tools are widely used with few development shops with a home made SCM for distributed area.

2.2.3 Typical mainframe practices

On the mainframe area, SCM practices are diverse but consistent at the same time. There is not two equal organizations, but the set of requirements and needs are quite similar for all of them:

- In the mainframe area, maintaince activities are much more predominant than new application development activities.
- Often, parallel development support is not an issue and not required.
- SCM mainframe tools understand the programming language to determine the information that is needed for build and impact analysis.
- Source code is often stored (not always) in the native OS/390® library system and follows a copy-based (promotion) model to manage change.
- SCM mainframe tools are in charge of managing builds and deploying software to the execution platform.
- There are shops with commercial SCM tools implemented, but there are others with home made solutions to manage change on the mainframe area.

Adopting a unified process to manage change at the application level independently of the platform is a great challenge. Different philosophies, skill sets, history, and technologies are impacted; however, the advantages and benefits are also great.

2.3 Challenges of adopting a unified processes

Having a unified software configuration management process enterprise wide brings many benefits and advantages, but to get to that point organizations will face many challenges. The configuration management process is in the core of the software development process of the organizations and therefore impacts the way that software developers do their job. Large shops with hundreds or thousands of software developers must put special attention on the way that they manage the journey to a unified process adoption. Careful project management technology and project management skills are needed, but more important, strong leadership skills are key to manage the revolution to a new way of developing and delivering software applications.

Understanding organizational, technological, and tool related issues helps to put in place the right project management framework to adopt the unified process as a key part of the ESCM.

2.3.1 Organizational differences

It seems obvious that if the goal is to have a unified SCM process, that it is important to have a unique software development department, regardless of the language. This is very far from the truth in many corporations. Very often development teams are divided into *open-distributed systems* and *mainframe systems*, where each team does not know the practices of the others. The mainframe team usually brings a very large background of developing software, in some cases more than 30 years of experience; whereas, the distributed team is new to many organizations. Companies hire new college graduates to develop new components in Java or C++ while COBOL developers have been in business for decades. There is a lack of interest in college and university programs to teach their students mainframe technology, which we unfortunately think is a reality world wide.

Distributed platform developers are very used to change and adopting new technologies, while mainframe developers are reluctant to change because the mainframe platform has remained stable for many years.

Organizations must carefully analyze these organizational differences when they want to adopt a unified change management system. They must put in place the correct project management and risk management framework to accomplish an initiative that impacts the way that the organization develops software.

2.3.2 Platform technology differences

Of course there are many differences between the mainframe platform and the distributed platform, based in UNIX or Windows. This section could be; however, in the context of configuration management we are just interested in the differences from the software development perspective. Surprisingly the deeper we get into this discussion we realize that development practices are quite similar in both platforms.

Let us start from the programming language perspective with a Cobol versus Java example. Both languages have different but similar concepts. Both of them are structured-oriented languages and both reuse libraries. Java is an object-oriented language and Cobol is not, but generally speaking the language syntax is not the reason why these two languages are so different.

Obviously there are differences in the way that you access the source code. In distributed there are Integrated Development Environments (IDE) in the market that provide a fast environment to the programmer, such as IBM Rational Application Developer. In the

mainframe development, although there are also some IDEs it is more common to see developers using the traditional operating system facilities, such as ISPF or customizations, over it.

In the context of configuration management, the critical point is the way you access the source code. In distributed, we can access the code in the local workstation, while in mainframe, the developers are logged to the host machine, and they share CPU for different activities. This sharing of resources makes the configuration management practices quite different from one platform to another. Configuration management systems in the mainframe concentrate on access control and build management due to the share of resources, while in the distributed, the focus is more in parallel development capabilities. The reality is that build management and access control are also important in the distributed area in the same way that parallel development is important for mainframe departments. At the end of the day, development is what it is and needs are similar for either platform.

2.3.3 Toolset and implementation differences

Generally, we can state that in the mainframe configuration area we found much more customizations and ad hoc implementations. In the distributed area, there is a trend to use the tools in an out-of-the box fashion. You must tailer mainframe configuration management tools to fit the execution platform, and build capabilities and functionality are incorporated into the tools. An external IT provider usually performs the implementation on the mainframe, and in some cases, the system remains unchanged for years. Maintenance on the mainframe systems has a high cost for the nature of the platform and for the lack of support and improvements by some providers.

In the distributed platform, commercial tools, such as IBM Rational ClearCase, are very extended, the same as other freeware tools. In some cases, there is a layer upon these tools that was created to fit the needs of the organization. We found that organizations with large customizations were mostly done using Perl scripting, but the vast majority uses the tool out-of-the box. The systems were implemented very few years ago, and providers constantly deliver changes and new functionality in their tools.

2.4 Impact of adopting a unified process

Adopting a unified configuration management approach brings clear advantages to organizations by improving application quality and providing the ability to respond faster to market demand. However, it is not easy to get there, and the unified change management approach affects to the core of how the company develops software. In this section, our goal is to introduce the impact of adopting the unified approach in an organization.

2.4.1 Organization and people

Resistance to change is the main challenge that the organization faces. The reality is that if an organization is thinking about adopting a unified process, it is because they want to improve the actual configuration management process and not to state the way that it is. On the other hand, it is also true that companies have delivered business applications for years, so any change to the actual process is always a matter of discussion.

Organizations were structured as new technologies were released. Some technology trends made the creation of departments and development teams. We find teams named as “Java Development Team” or “Open System Development Team”. Mainframe developers and open distributed development are often split and report to different managers. ESCM is about

unifying the configuration management process so that both teams are equally affected. The impact is unavoidable because the organization is going to change the operations of their development practice, and the people who are affected tend to resist this change.

Here are some recommendations to mitigate the impact and to align people with the new process:

- Create a *Unified Configuration Manager Board* with equal representation on both sides. If possible, add well respected engineers to the team in both areas.
- Make both sides the owners of the project, and be the sponsors of the solution. Do not let one team lead the process, while others simply follow.
- Provide project funding and leadership from top management.

We could state that in 90% of the cases, there is a natural trend to try to replicate heritage processes and use cases in the new system. Although IBM Rational solution is flexible and customizable, it is based in commercial tools with a particular vision and nature. Stubbornness is dangerous. We have seen cases of customers trying to customize the tools in such a high degree that the maintaince of the system is more complex than building a home made system from scratch. Organization and teams have to reasonably adapt thinking to the tooling and at the same time the tools must be flexible enough to meet user requirements and expectations.

The ESCM is a cross department vision because not only are development and architecture teams affected, but other teams and departments are important stakeholders of the solution. Interfacing with these teams usually brings conflicts and discussion that the Unified Configuration Manager Board must manage. Here are some comments about relationships with other departments:

- Relationship with the Development department: Although the development department is the most affected because they are the end users, the relationship is usually good because they have a clear understanding of the goal of ESCM. Because they are the main users, the solution must meet their expectations and solve the issues that are related to changing their activities. Open communication and following the same goal is the key.
- Relationship with the Methodology group: Often organizations have a methodology that has the goal of standardizing software development processes. In many cases, the ESCM is the responsibility of this Methodology team; however, in some other organizations, the Methodology team simply approves the new process. Consistent communication and willingness to move forward is important between the Methodology team and the Unified Configuration Management Board because it is a best practice to have a representative from this department on board.
- Relationship with the Production department. Production is normally in charge of deploying applications to production activities. We strongly believe that deployment to production is part of the ESCM process and redefined, if necessary. This last issue is a matter of discussion in many Production departments, which obviously represents a big challenge, but it is the only way to guarantee reverse traceability. The effort is worthy, which is why we consider it necessary to include a Production stakeholder in the Unified Configuration Management Board. Production should also own the process.

ESCM is an IT-wide process, not just development. Many departments are affected, it is mandatory to have support from top management and to have the right project management framework in place; otherwise, the risk of failure is high.

2.4.2 Resources and training

In the last section, we described the need to build a Unified Configuration Management Board to be in charge of the ESCM process implementation, which must provide the foundation to the team that supports the system when it is released.

The Unified Configuration Management Board should have at least the following roles:

- Project Manager: Senior Project Manager with access to the CIO level.
- Chief Configuration Manager Architecture: Senior Architect who is well respected in the organization.
- Mainframe Configuration Manager Advisor: Expert in the mainframe configuration management area.
- Distributed Configuration Manager Advisor: Expert in the distributed configuration management area.

The Unified Configuration Management Board should be trained in Rational tooling: Rational ClearCase, ClearQuest, and BuildForge. IBM Rational provides a set of courses that we recommend for the Unified Configuration Management Board to provide the skills foundation that are necessary to implement the right process. You can view the courses:

http://www-306.ibm.com/software/rational/education/

2.4.3 Investment and return on investment

Every organization has its particular situation that can make a difference in terms of the investment to make and the benefits that are expected, such as the number of different languages to migrate, the need to adapt to an existing methodology, the amount of code to migrate, or how fast can a software development team adapt to a new tool, are factors to consider as part of a return on investment (ROI) study.

There are four categories of investments to be made:

- Funds to purchase the software licenses to implement the ESCM. This cost is often compared with savings that are provided by avoiding the high maintaince fees that the CM mainframe tools usually have.
- Funds to hire the expert consultants to define the right ESCM process and to adapt the toolset.
- Internal effort made by the company with the resources that are dedicated to the ESCM project: Project Manager, CM personal, experts in each platform, and personal responsibility to migrate data.
- Roll out and training to the software development team in the ESCM.

Return of this investment is diverse and interesting to consider. IBM Rational professional services and experts can do detailed ROI studies for you. These studies are difficult to generalize and should be done with a good knowledge of the organization, the actual CM practice, and the objectives of ESCM that they want to achieve.

We can say that there are two types of return of investment. The first type is related to savings in development cost. ESCM provides a great amount of automation, team awareness, and workspace stability, which means that the developer is more concentrated on the creative activities than on the configuration management activities, which saves development time.

There are multiple parameters to measure but some examples are:

- Developers save time because it is easier to find the right version of the code with ESCM.
- Developers save time communicating with the team because of the advanced comunication and ream awareness capabilities of ESCM.
- Time to deploy to production is decreased.
- Stability of ESCM usually means less integration testing.

These parameters are easily measured and can be part of a detailed ROI study.

The second type of return of investment parameters are business oriented. They are related to the impact in the business of having a low application quality. Poor configuration management systems usually allow defects and bugs go to production. Some critical bugs can mean a decrease in revenue for some businesses. Another business factor to consider is that decreasing development time means that the company can be more flexible to adapt applications to the market needs and can provide a competitive advantage. This parameter can be measured, but the business department and not IT must perform this type of analysis.

2.5 Benefits

In this section, we discuss the benefits of an ESCM process.

2.5.1 Governance

ESCM is about process automation in the software development life cycle. With this system in place, we can define what we want to do, how to do it, and being able to prove it. ESCM provides a governance framework that gives development teams the ability to define, implement, and track development projects efficiently.

The first key feature of ESCM is that the system links the activity management system with the actual changes in the software elements that need to be modified to implement the activity, as illustrated in Figure 2-1 on page 18.

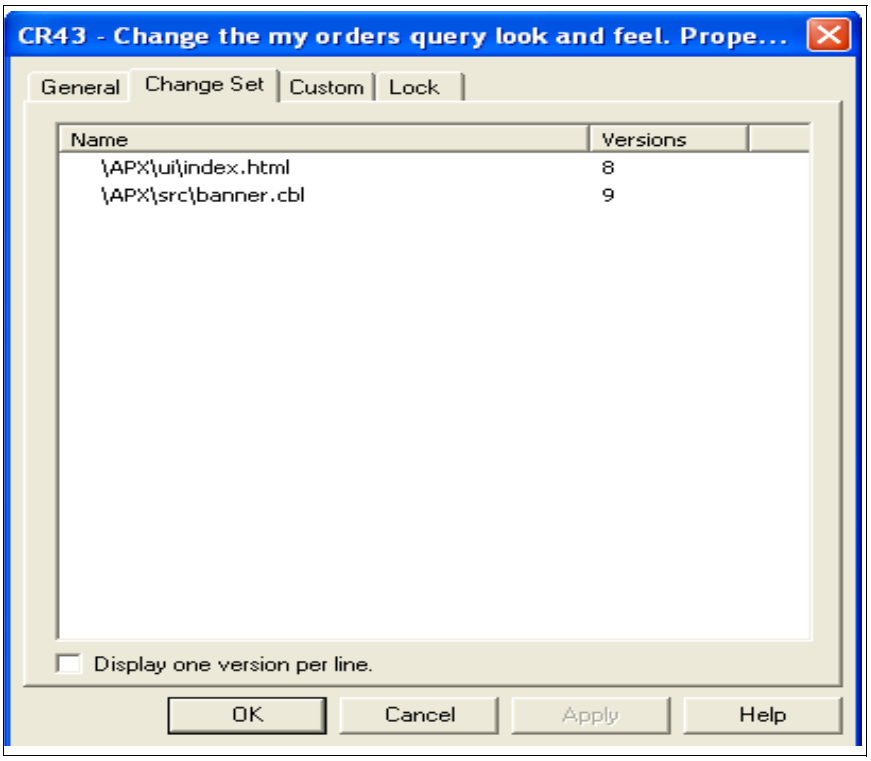


Figure 2-1 The activity CR43 links with an HTML and Cobol source file

You can track, organize, and prioritize activities according to the software development process that is defined. You can define different kinds of activities, for example, you can manage defects, enhancement requests, urgent maintaince requests, and so on.

Governance is challenging, in terms of the deployment to production process because it is often uncontrolled or “manually controlled”. In many configuration management systems, this is the weakest area and brings confusion to the development team in cases of failed deployments and the actual content of deployment packages. ESCM has a centric and unified deployment system that controls change and builds from development to production. With our ESCM solution, we provide a unique point-of-control for each package that we want to promote, regardless of their platform. With ESCM, we have reporting and diagram capabilities to access promotion information from a project or multiproject perspective, and we can consult the status of projects from the change management perspective.

Example of a governance use case

An example of a governance use case are Trend Diagram releases. The activity management is centralized and there different ways of accessing this information. We can obtain real-time information about activities progression, status, source code affected, and build results from different perspectives according to the role played in the project. In Figure 2-2 on page 19, we can check the number of releases on a weekly bases and the status of these releases.

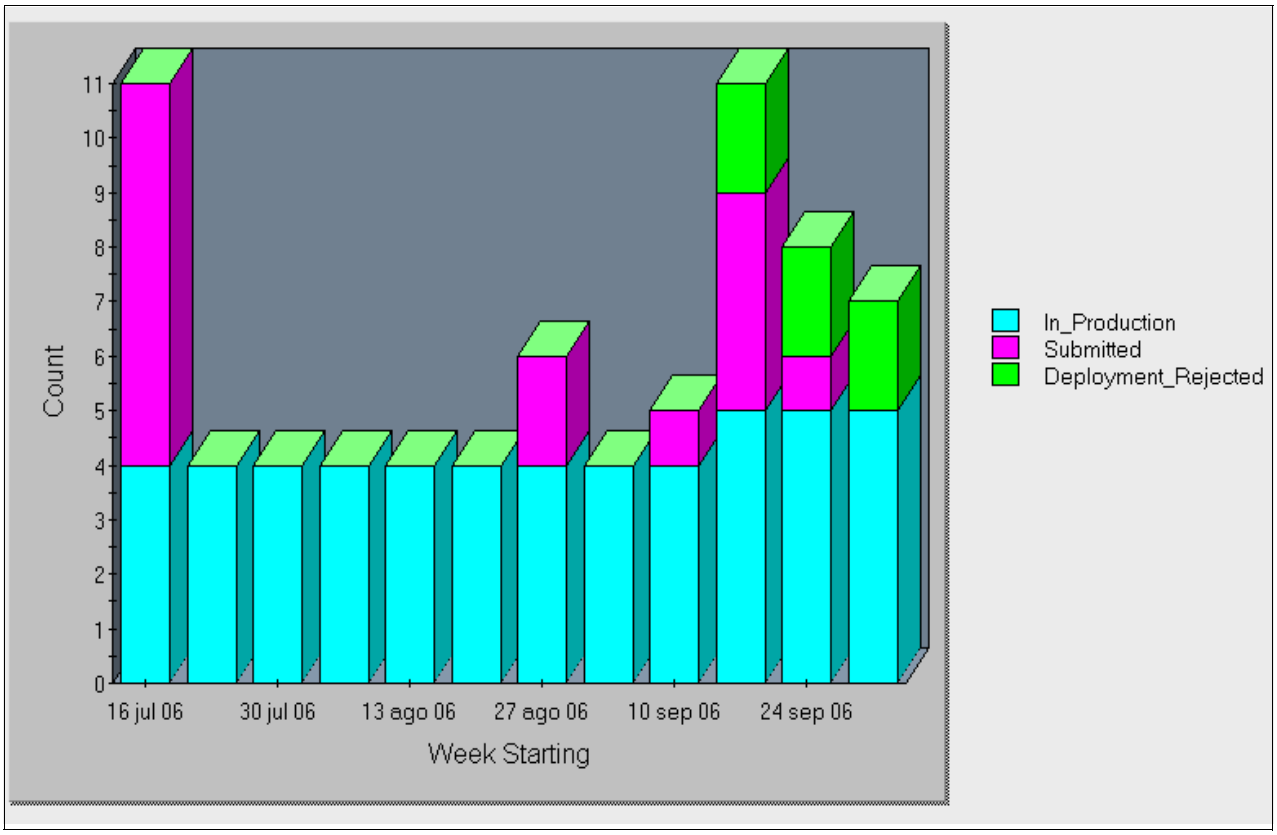


Figure 2-2 Number or releases and status on a weekly basis

2.5.2 Auditabilty

With IBM Rational tools, you can record every change to a software element through the life cycle. Every event is recorded in the configuration management database, and you can consult it afterwords.

Examples:

- History on changes to an element. The ESCM system can record when, why, and by whom an element was changed, as illustrated in Figure 2-3 on page 20.

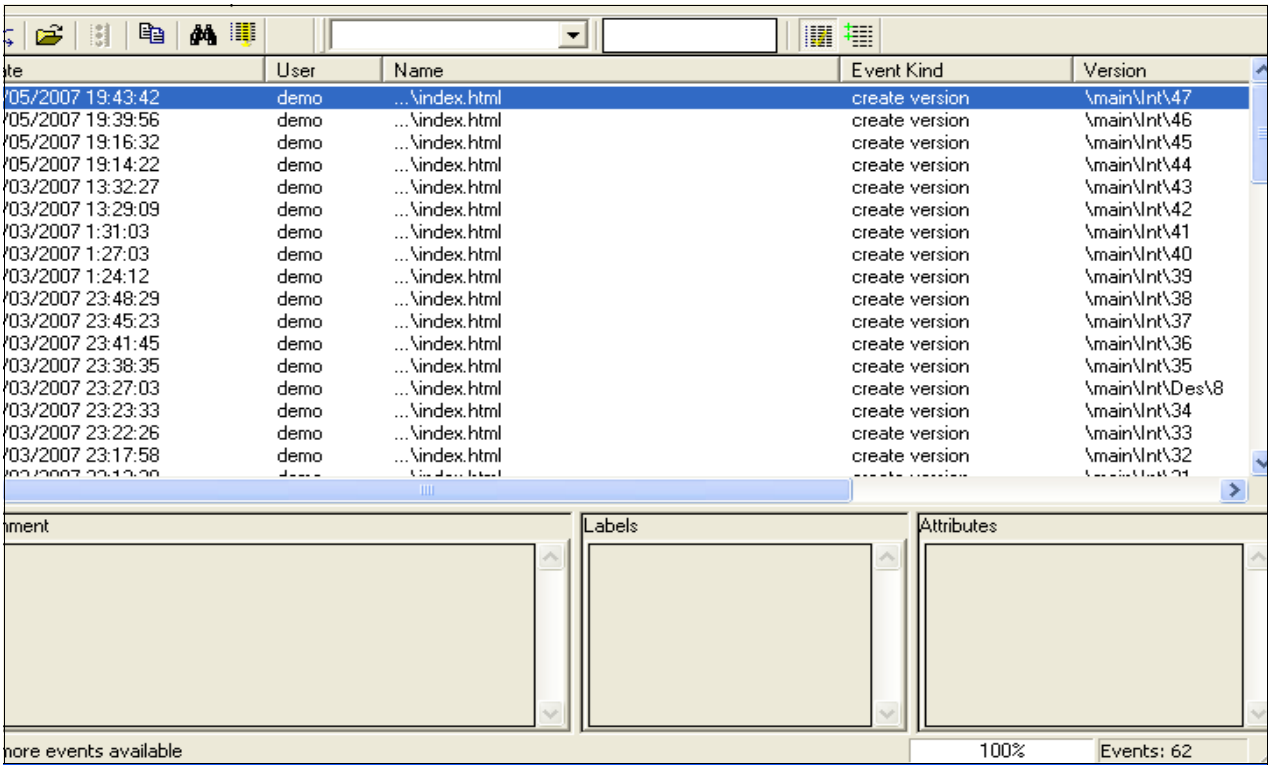


Figure 2-3 History of events recorded on changes in a particular file.

- Changes on a change request or change package, for example, CQ record history, as illustrated in Figure 2-4 on page 21.

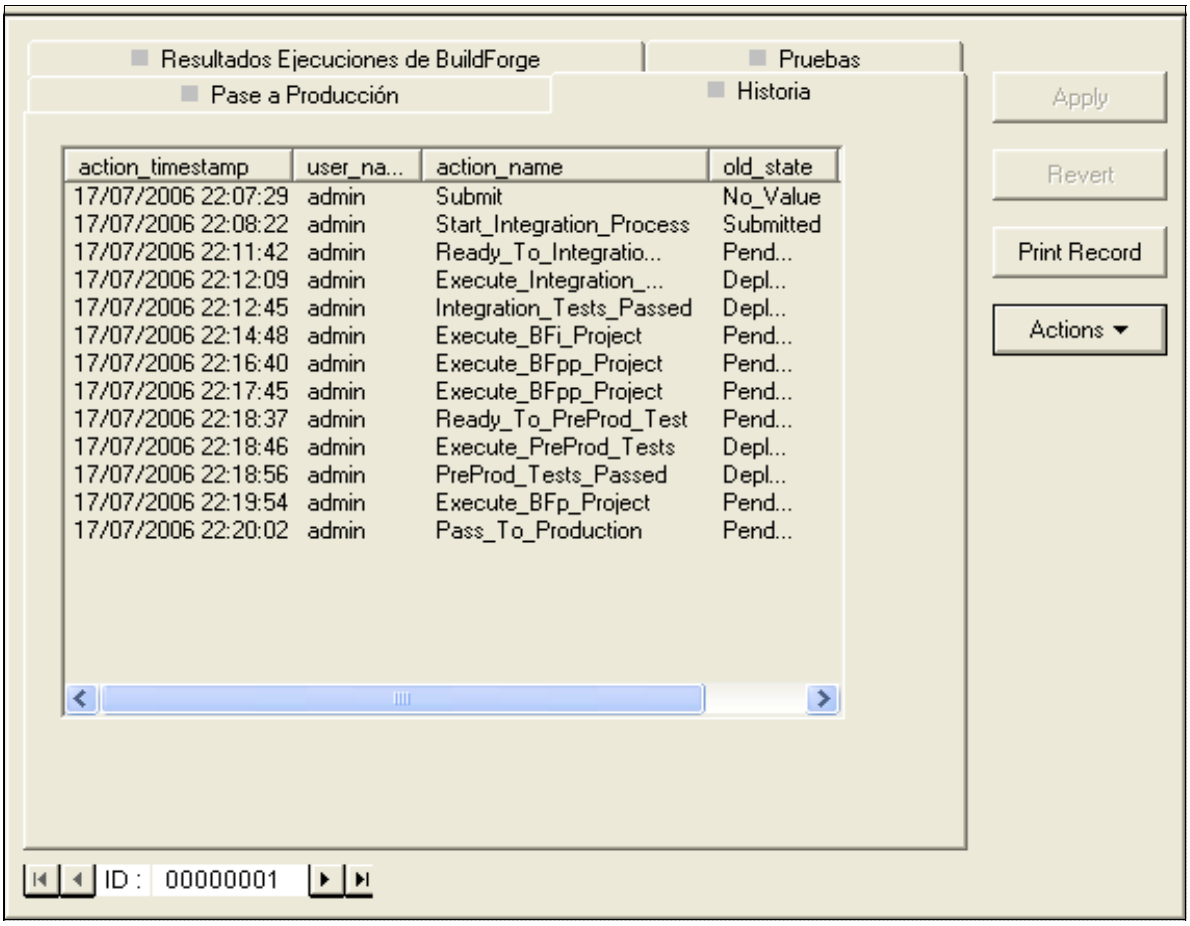


Figure 2-4 History of actions of a change request

- Source changes that are included in a particular deployment. It responds to the question “What changes on my code were included in the last production build”. ESCM provides powerful “Bill of Materials” authoring capabilities and availability to the system, as illustrated in Figure 2-5 on page 22.

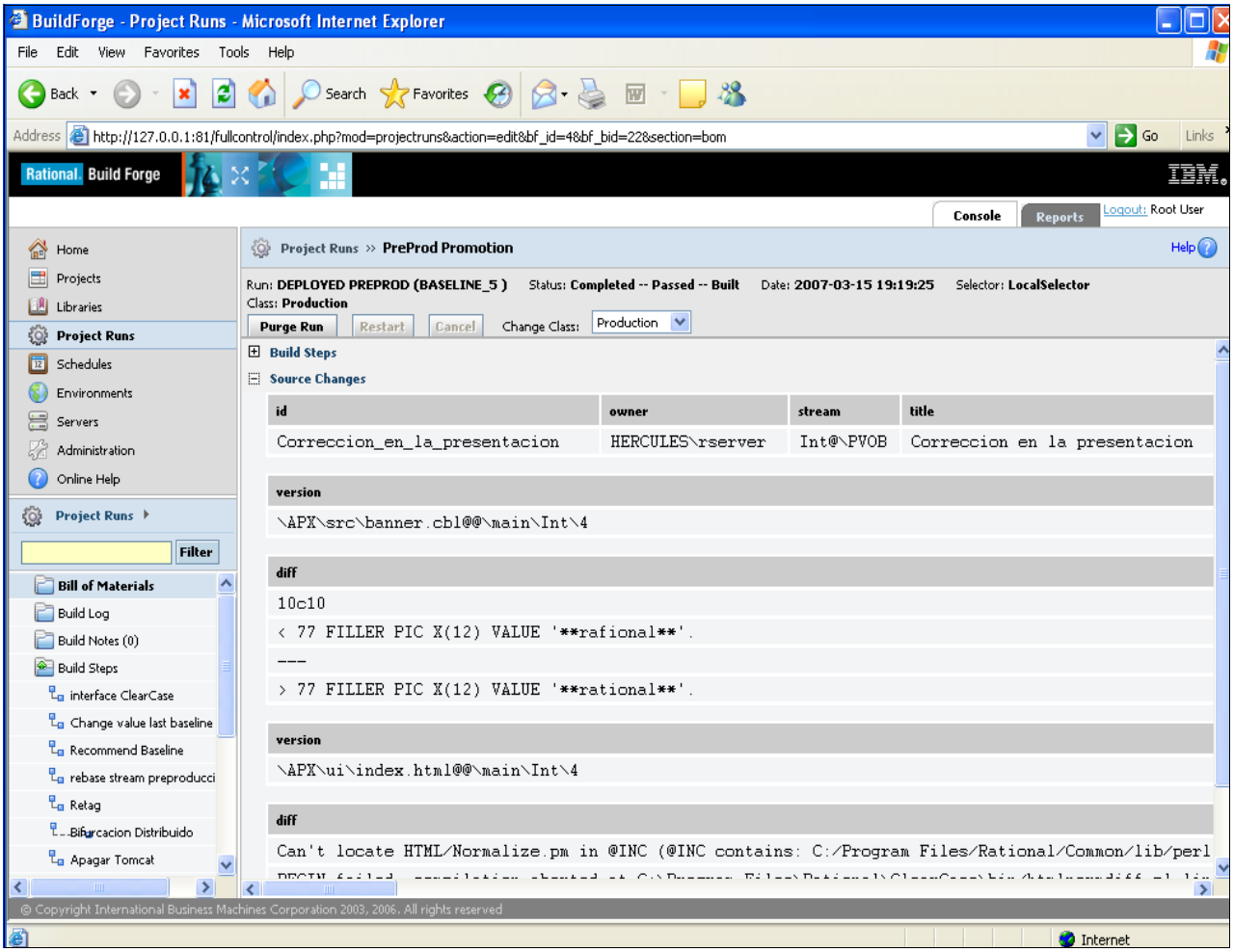


Figure 2-5 Bill of Materials of Baseline 5 on PreProdcution Environment

2.5.3 Compliance, regulations, and certifications

Complying with government or industry regulations is a mandatory requirement for many organizations. Because the development process is quite spread across the interest of organizations, many organizations seek to achieve capability maturity model integration (CMMI) certification. Configuration and change management is a process area on level 2 on CMMI.

In the configuration management process area, there are three specific goals that are related to ESCM. Let us describe how the ESCM solution supports these CMMI goals:

- Specific Goal 1 (SG 1): Baseline of identified work products are established and maintained.

  IBM Rational ClearCase is the core of ESCM, and provides the concept of baseline as a metadata of the system. Baselines are created using the command cleartool mkbl or with ClearCase interface GUI operations. A baseline identifies changes that are included in the work products using the ClearCase concept of activity. Each activity includes changes that occur in the source elements, documents, scripts, deliverables, and so on. When you create the baseline, you can select the activities to include. The baseline remains part of the system, which you can maintain and consult afterwards, as illustrated in Figure 2-6 on page 23.

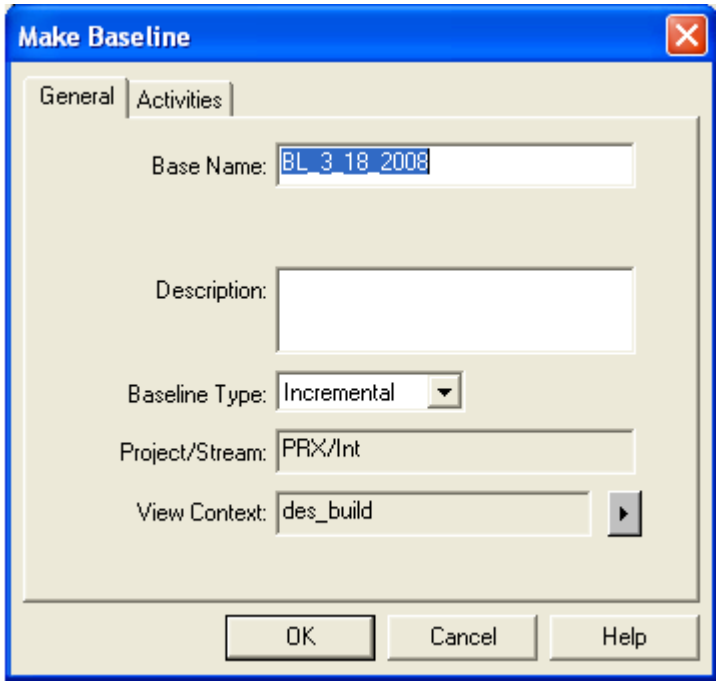


Figure 2-6 Make Baseline dialog box.

- Specific Goal 2 (SG 2): Changes to the work products under configuration management are tracked and controlled.

  IBM Rational tooling control stores all changes that are done to a work product. Metadata, such as, owners, history, time stamp, and changes in the control units that are included in a work product, is stored by default, as illustrated in Figure 2-7 on page 24 and Figure 2-8 on page 25.

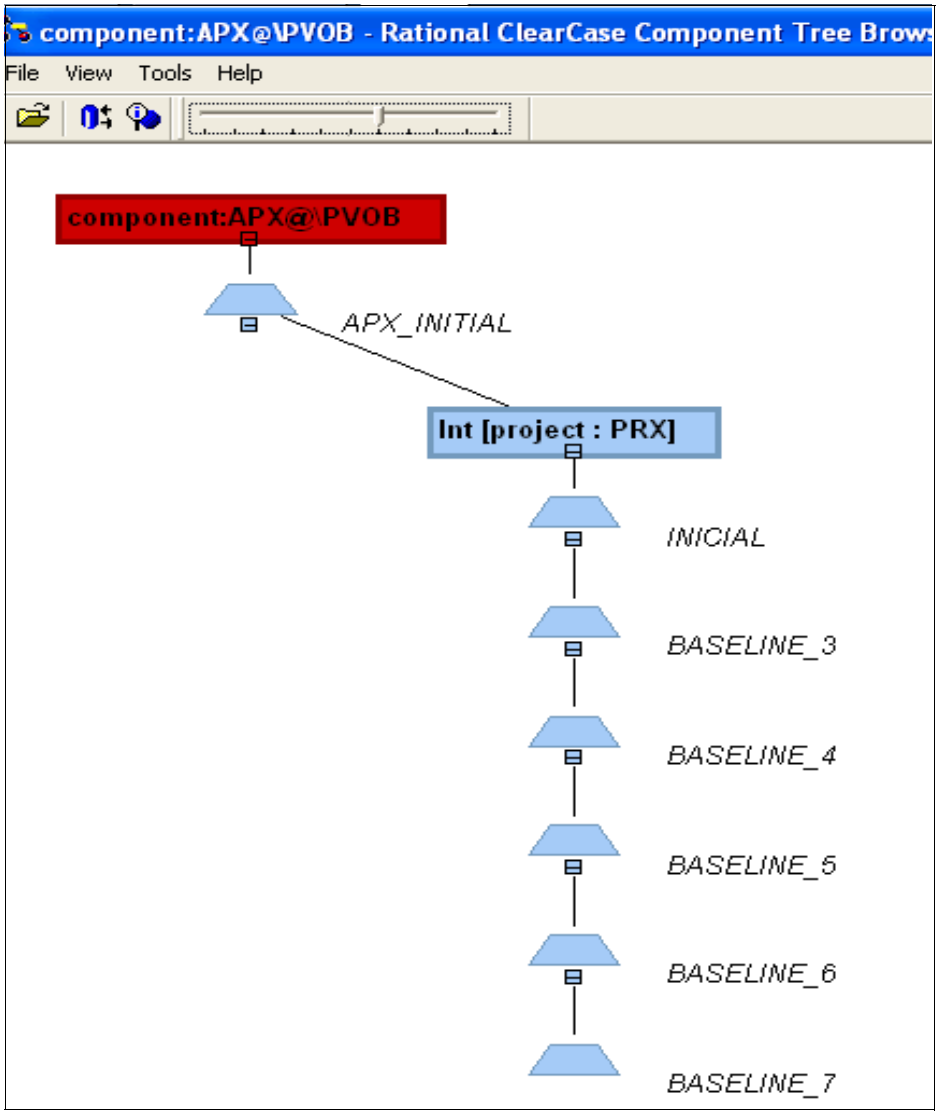


Figure 2-7 Component tree browser

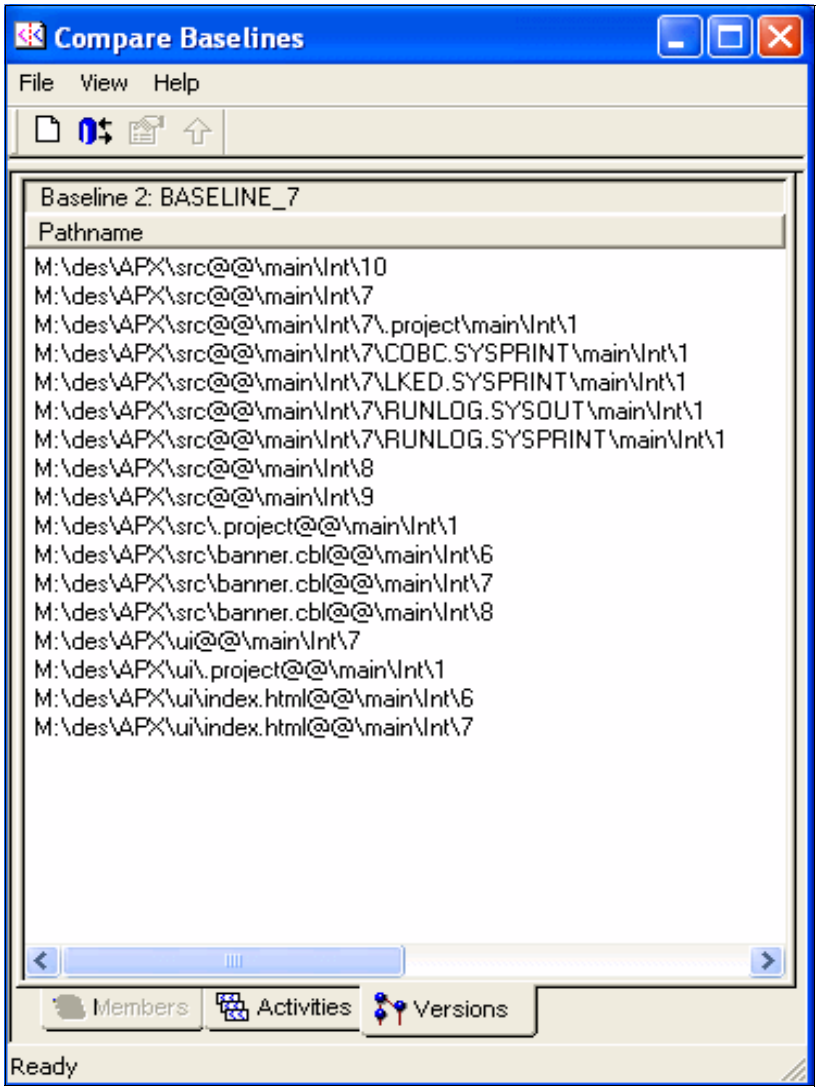


Figure 2-8 Compare baselines

- Specific Goal 3 (SG 3): The integrity of baselines is established and maintained.

  Baseline integrity is guaranteed. You cannot change baseline content in terms of work products that are included. The tools provide a way to represent the quality of a baseline, but the baseline content remains untouchable. IBM Rational ClearCase also provides a way to make parent baselines of children baselines using the concept of composed baselines. We have strong capabilities on baseline management but always under the control of the tool that provides the baseline integrity, as illustrated in Figure 2-9 on page 26.

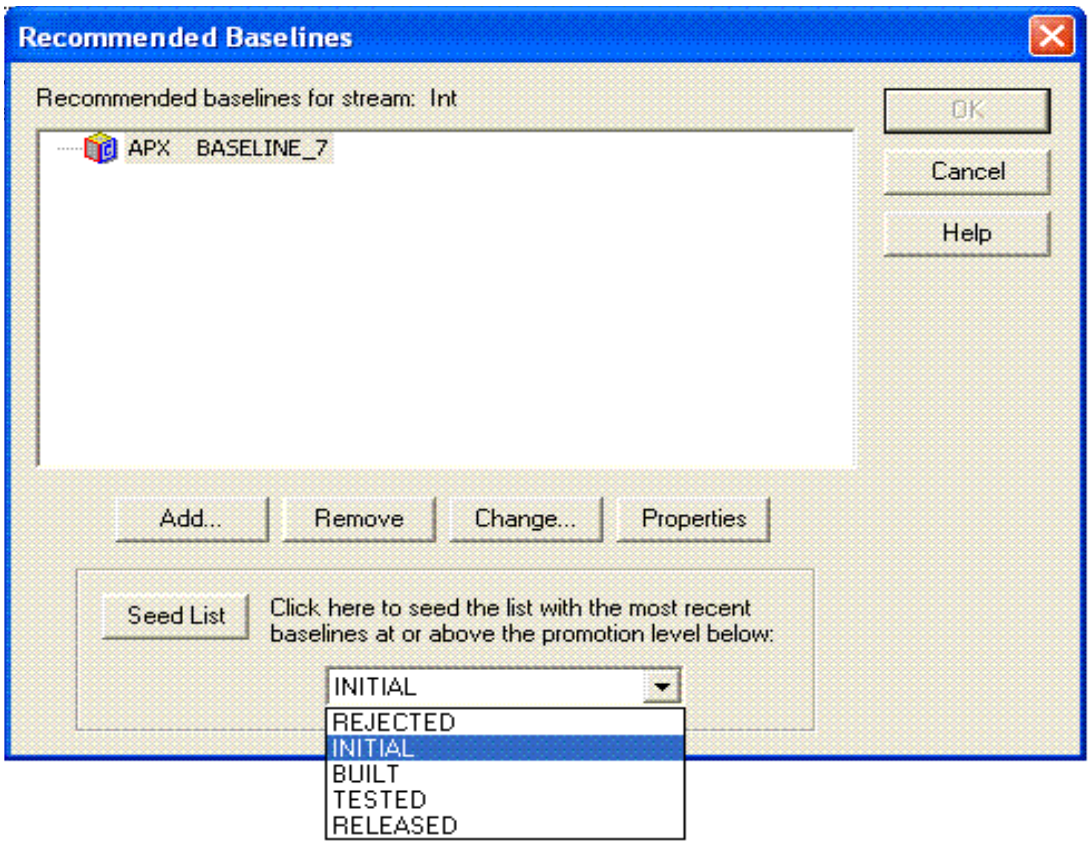


Figure 2-9 Baseline Promotion Levels indicates the quality of the baseline

The IBM Rational tools provide the necessary automation to help to achieve configuration management process area requirements for CMMI level 2. In addition, to assist to achieve the certification level, using the tools makes this certification efficient from the development team perspective because it provides automation in the development process.

2.5.4 Support for geographically distributed development

In today's global world, more often enterprises have developers spread in different sites. Very often these sites are in different cities, countries, even continents. In some cases, overseas sites belong to a subcontracting company, and the change management system has to support different sets of situations in what IBM Rational calls the *Geographically Distributed Development (GDD) strategy*. ESCM supports this global environment. The underneath tools of the ESCM solution provides two types of support for remote development based in:

- WAN support for every set of the ESCM.
- Replication System for high intense development activities in case the WAN does not provide the right performance.

ESCM provides accurate time information of development activities, depending on the hourly frame where a user is located. It also gives the right environment for team collaboration in world-wide development projects.

2.5.5 Productivity

Using any configuration management system gives many advantages to organizations. Most of them are related to team support, team comunication, parallel development, security, and auditabilty. However, ESCM provides unique advantages because of the unification to one SCM system to manage diverse, running application platforms that no other solution can provide.

This list of productivity improvements are unique to ESCM and its unified approach:

- Use modern tooling: Enterprise development IDE, such as, IBM Rational Developer for System z, gives strong edition and debugging capabilities for Cobol and Web development. Modern IDEs increase productivity versus using classic ISPF panels for development on a mainframe.
- Build automation: Build execution is automated with a centralized reporting and consulting system for mainframe and distributed builds through the IBM Rational Build Forge Console.
- Parallel development support capabilities: With ESCM, we support different parallel development user scenarios for maintenance and evolution, component sharing, concurrent development, and product diversity. The tools manage the parallel development complexity, which enables the development teams to concentrate on value activities.
- Reverse traceability: ESCM integrates the deployment flow from development to production and records the bill of materials of the version that is deployed in production. Teams feel safe about what they deliver to production and what particular changes are included in the build. Reverse traceability enhances productivity and increases the trust level among development team members.

3

The Enterprise Software Configuration Management implementation model

In this chapter, we describe the way you can implement and use an Enterprise Software Configuration Management (ESCM) architecture. Given that there is not a single usage model, it is important to clearly understand all of the requirements (related to processes, environments, workflows, roles, interfaces, assets, and so on), and try to design and implement the best scenario to fit your needs.

We also describe and discuss some of the usage models and provide suggestions about how to adapt them to different situations.

3.1 Introduction to an ESCM architecture usage model

In this section, we discuss the key pieces of an ESCM architecture usage model:

- Unified Change Management (UCM) development models
- Processes

It is clear that an ESCM architecture/implementation is not only a matter of installing and configuring IBM or third-party products. Most of the work is related to the way these products are implemented and in some cases it is required to extend or integrate them using custom interfaces or procedures.

Any activity that aims to define the best model to use or implement must start from a complete analysis that is performed with the right team of people from Configuration Managers to Project Managers, from Developers to Testers, Release Managers, System and Database Administrators, consultants (if any), and mainframe and distributed environment experts. You want to clearly understand exactly requirements and the way that they want the ESCM architecture implemented and delivered to end-users. In this phase, it is key to put on the table all of the different aspects (not only related to products) that can impact the final model. Some of them (for example, processes and development scenarios) are critical for the success of the project and can be difficult to understand and then implement.

During the first phase, involve key stakeholders in the discussion and allow all important roles to provide their view of all aspects, with clear informations on the “as-is” model and clear requirements that are related to the “desired/to-be” model. In our experience, we noticed that if these points are unclear, problems occur during the critical phase and during the delivery of the solution, which in turn, requires you to review and correct or re-write some or most of the components that are already developed or implemented. Furthermore, this is not always a problem that is only related to the technology (products), it can also affect how the products are used.

Overall, when we consider ESCM, we must consider different:

- Tools (integrated together)
- Platforms (used to store data or to access them)
- User interfaces (for both processes and changes)
- Locations (with geographical distributed development or physical environments)

Putting all of these pieces together, in a working environment, with fluent processes where all of the involved roles work together with minimal overhead, collaborating and sharing information, is often challenging, which is why it is important to clearly state, in the beginning, all of the objectives, all of the constraints, all of the limits (there are many), and all of the phases with related milestones (because it is almost impossible to think to a one-phase only implementation).

Our main objective in this chapter is to help you to better understand all of these aspects and to provide some examples of how you can implement an ESCM solution, based on real cases and experiences. At the end of this chapter, you will be able to identify:

- A list of tasks to put into the project plan
- The areas where you must invest your time
- The priorities and phases of the project
- The areas to further investigate before the project analysis starts
- The areas to further invest to prepare an effective environment that facilitates the project implementation

3.2 Best practices and scenarios

When you work in an Enterprise Software Configuration Management system, it is normal to find heterogeneous platforms and environments that developers use to connect to perform changes to software artifacts. The development platforms that are most involved in the scenario are:

- UNIX
- Linux
- Windows
- zLinux
- System z
- System i

Development can occur on different platforms, as shown in Figure 3-1 on page 32, and can involve several developers working on the same artifact or in parallel on different sets of files. Projects can share code and also have objects synchronized together. This synonymous development can occur when there are multiple development parts that are involved on different environments, in a single project. The typical scenario is when a project shares Java code on distributed environments and COBOL/CICS/DB2® on System z.

Development that is related to mainframe assets can occur in two different ways:

- Developers can continue to access code on System z ISPF panels, and perform operations, such as check out and check in, from a common central repository. In this case, developers continue to work on their environment without any impact on the way they normally perform common operations (for example, build, debug, test, and so on).
- Developers can move to Rational Developer for System z on Windows and Linux platforms, and work locally or remotely on the System z instance. The impact in this case is higher because they learned a "new way" to work, but the benefits are higher than continuing to work on mainframe. At the same time, they gain integration between IDE tools and SCM tools and work on a single workspace, interface, or prospective.

Typically UNIX, Linux, and System z platforms remain environments for deployment of builds, while development occurs on Windows and Linux platforms. The integration between SCM tools and tools to perform Release Management, allow you to maintain synchronized development, test, and production environments. The different roles that are involved in the software development process, access the correct artifact in the correct environment, without the risk of getting wrong versions of code. Also, deployment on heterogeneous target environments (for example, System z, UNIX, and Linux) occur simultaneously, which is a full guarantee of software consistency between all of the objects that are present into the software application release. Figure 3-1 on page 32 provides an illustration of a development environment.

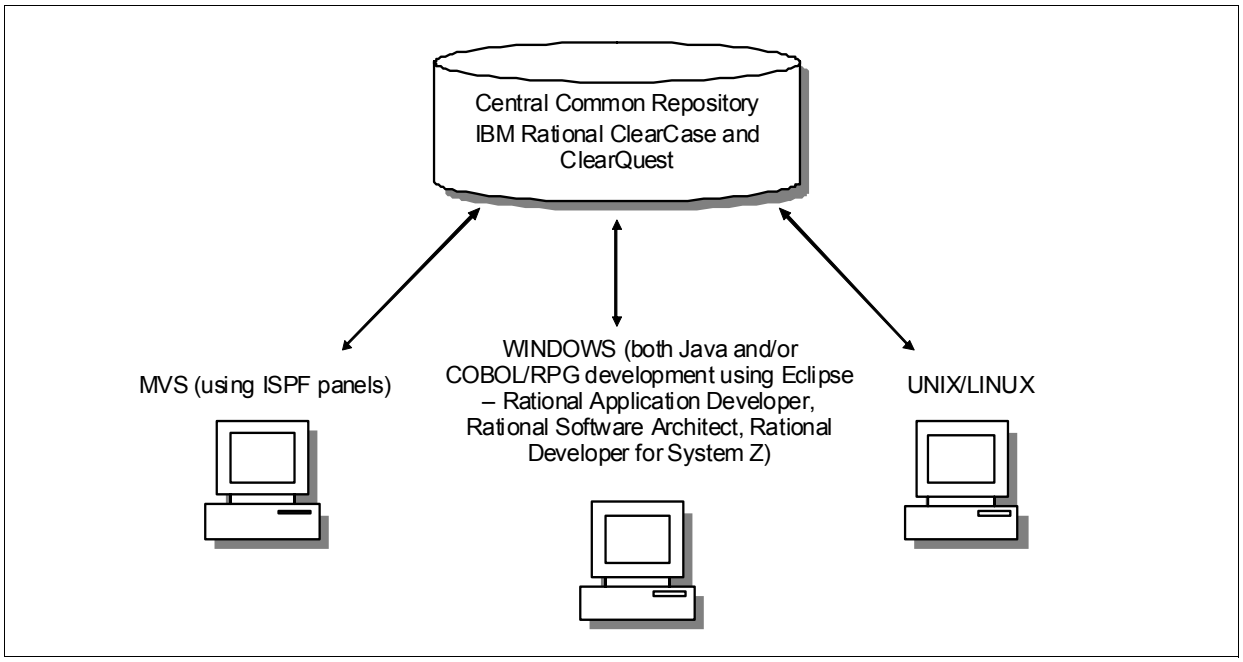


Figure 3-1 Development environment diagram

One of the key points is to guarantee access to repository resources from heterogeneous platforms. Developers need interfaces to work on artifacts without worrying about where they are stored, and they need to be able to perform all operations that are associated with the various roles involved in the process.

Another factor to keep in mind is the possibility to use interfaces, such as a browser, for roles that typically do not need to have native clients locally installed, for example, Project Managers, Testers, and Release Managers. Also, in case of development on a mainframe, users want this type of interface because they do not have to learn new tools to work in the ESCM process.

The available interfaces are:

- Graphical interfaces to control ESCM processes
- Graphical interfaces to access artifacts and to perform changes
- Web interfaces to reduce maintenance costs and end-user training
- Multiple Virtual Storage (MVS™) interfaces to help developers remain in their native environment
- Custom interfaces (both integrated or stand-alone) to extend the native functionalities that the standard interface provides

3.3 Unified Change Management development model

The Unified Change Management (UCM) is a specific change management process that IBM Rational developed in conjunction with some of our customers. UCM supports software project teams in managing the production and modification of files, directories, components and systems. Change Management processes can consist of two separate disciplines:

- Software Configuration Management (SCM): SCM deals with version control, workspace management, parallel development, software integration, and software builds.
- Change and Release Management (CRM): CRM deals with the processes and procedures by which defects, enhancement requests or maintenance requests, and new features are submitted, evaluated, implemented, verified, and completed. CRM also encompasses software deployment and release processes.

Rational has three tools that support these two disciplines, respectively:

- Rational ClearCase: automates SCM-related processes.
- Rational ClearQuest: automates Change Management-related processes.
- Rational BuildForge: automates the Release Management-related processes.

By using these three tools together, you can automate UCM. Actually, you can automate almost any change management process using ClearCase, ClearQuest, and BuildForge, but if you want out-of-the-box support for change management, then UCM is your best choice.

Rational ClearCase, Rational ClearQuest, and Rational BuildForge provide extensive flexibility for configuring processes that manage change in software development. When used together, these powerful tools simplify change management by clearly identifying associations between any type of change request, whether enhancement request or bug fix, with related changes in application code or Web content.

Unified Change Management takes the integration of Software Configuration Management and Change and Release Management by providing a predefined process that organizes work around activities and artifacts. Based on the Rational best practices of managing change, Rational ClearCase and Rational ClearQuest enable Unified Change Management. They accelerate development with an out-of-the-box process model that you can turn on or off, based on the specific needs of your software development team.

Unified Change Management simplifies change management across the software development life cycle by automating the management of all types of software development artifacts, which can include requirements documents, design models, testing artifacts, application code (both distributed and mainframe, for example, Java code, net code, COBOL, RPG, and so on), and HTML and XML content.

As development organizations grow and release cycles accelerate, the need for tools and processes that streamline the management of software change becomes essential. Unified Change Management delivers this process with a tooling infrastructure that Rational ClearCase, Rational ClearQuest, and Rational BuildForge provide. Powered by Rational ClearCase and Rational ClearQuest, UCM is the IBM Rational activity-based process for change management.

UCM provides defined processes for five specific areas:

- Insulating and collaborating of shared and common code artifacts for developers
- Collecting groups of related artifacts that are developed, integrated, and released together into UCM components

- Creating component baselines at project milestones and promoting baselines as complete sets, according to established quality standards
- Organizing changes into change sets
- Integrating activity and artifact management

3.3.1 Developer insulation and collaboration

Developers require insulated work environments to isolate their work-in-progress from the potentially destabilizing changes of other team members. Rational ClearCase provides two options for accessing the correct versions of artifacts and for working on them in private workspaces. These include *snapshot views* and *dynamic views*, which are optimized, respectively, for either local or network use.

Snapshot™ views provide team members with the flexibility to work disconnected from the network and easily synchronize their work with mainline development. Dynamic views are implemented with a unique virtual file system that enables team members to transparently access the correct versions of the correct artifacts, without copying them to their hard drive. Regardless of how they are created, these private work areas allow team members to work simultaneously on multiple releases, for example, a developer who works on release 2 can also fix a defect in release 1 without fear of altering code that is unrelated to the developer's assigned activities. In another example, one developer can fix a defect in release 2 while another developer works on a new feature addition. Because Rational ClearCase enables each developer to perform their respective activities in their own private work areas, none of the individual needs to worry that their work is impacted by the other's changes.

While insulation from destabilizing changes is critical for minimizing error, the integration of all changes into a common work area that is accessible to all team members is an essential step in team development. Today's component-based development methodologies, along with the increasing frequency and pace of code changes, require teams to integrate work often.

With Rational ClearCase, teams can implement a variety of project policies to insulate and enable collaboration, simultaneously, for example, a small team can use a serial methodology that allows just one developer at a time to make changes to an artifact. In contrast, this type of policy hinders the productivity of large teams that require multiple developers to collaborate on common artifacts.

Rational ClearCase supports parallel development on a large scale with rich branching and merging capabilities. In UCM, the branches that enable parallel development are viewed at a more abstract level, such as *streams*. A stream represents a workspace that can be private or shared. Streams define consistent configurations of project versions and provide the balance between isolation and sharing that enables efficient collaboration in a UCM project. To achieve this balance, Rational ClearCase configures private streams for individual team members along with public integration streams where all work is delivered, as shown in Figure 3-2 on page 35.

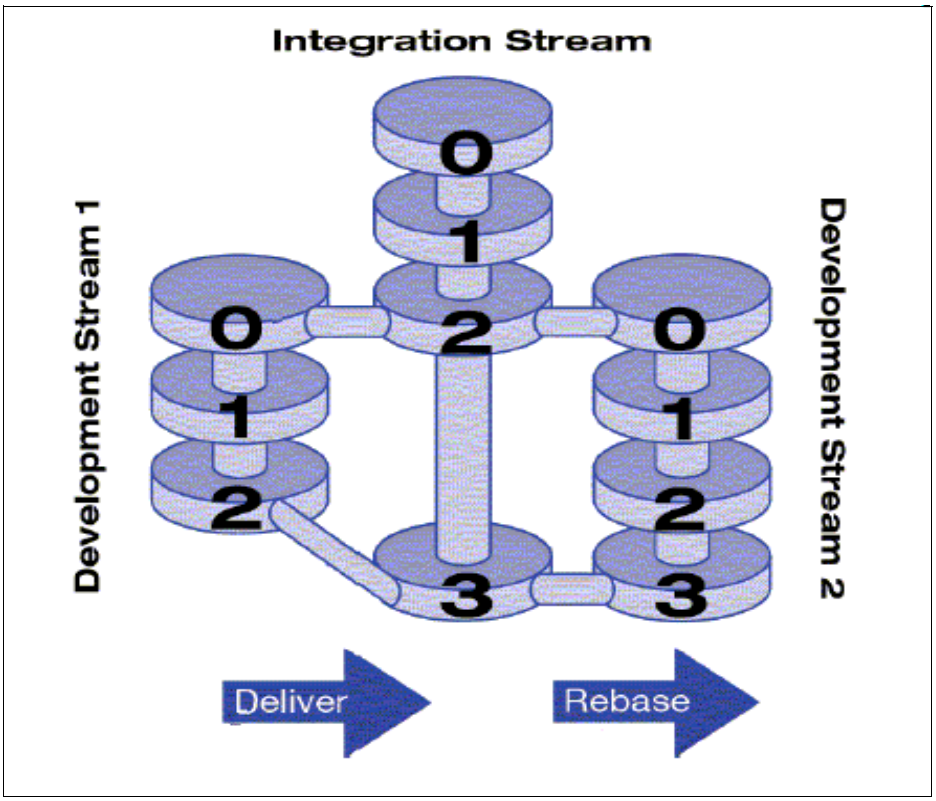


Figure 3-2 UCM provides public integration and private work areas

Private streams provide developers with insulated workspaces that are initialized with the public artifacts that meet established quality standards. Developers use these private workspaces to make changes and to build and test artifacts. When developers are satisfied with their changes, they deliver them to the public integration stream. To keep up-to-date on project changes that other team members make, developers can also *rebase* their private work streams with the most recent, stable baselines from the project's public integration stream. With UCM, developers choose when to deliver and when to rebase.

The public integration streams serve as the coordinating point for all project changes that are made by all team members. UCM introduces the concept of a *baseline* as a measure of progress. A baseline, which is defined as an abstract representation of a build or configuration, is a version of a *component*. A component is a collection of related artifacts.

3.3.2 Activities and artifacts

As software systems and teams grow in size and complexity, it becomes increasingly important for teams to logically organize activities around periodic releases and efficiently manage the artifacts that they use to implement these activities. An *activity* can involve fixing a defect or building an enhancement for an existing product. An *artifact* is anything that evolves over the course of the development life cycle, such as a requirements document, source code, design model, or test script. Teams perform activities and produce artifacts, as shown in Figure 3-3 on page 36. UCM integrates the activity management capabilities that Rational ClearQuest provides with the artifact management functions of Rational ClearCase.

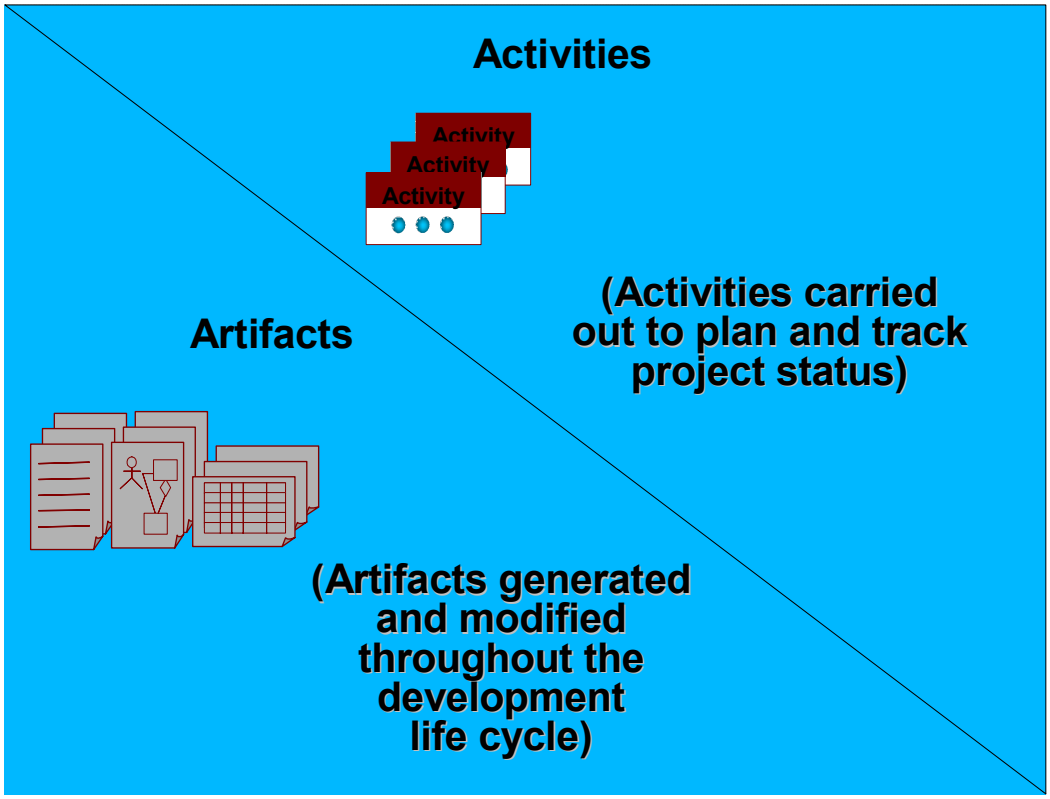


Figure 3-3 UCM integrates Rational ClearQuest's activity management capabilities with the artifact management functions of Rational ClearCase

Activity management

In UCM, Rational ClearQuest manages activity management. Rational ClearQuest is a highly flexible and scalable defect and change tracking system that captures and tracks all types of change requests, such as defects and enhancement requests. In UCM, these changes are viewed as activities.

Rational ClearQuest offers a customizable workflow for tracking and managing activities, which enables teams to more easily:

- Assign activities to specific individuals
- Identify priorities, status, and other information that is associated with activities
- Automatically generate queries, reports, and charts

You can adapt the ClearQuest workflow engine based on your team or procedural requirements. Teams without customization needs or that already have a predefined change management process can use ClearQuest's out-of-the-box process, forms, and associated rules for fast implementation, as shown in Figure 3-4 on page 37. Teams that require extensive customization can use Rational ClearQuest to tailor just about every aspect of their change management system, which includes defect and change request state transition life cycles, database fields, user interface layouts, reports, charts, and queries.

An effective workflow for managing and tracking defects and other changes throughout the development process is essential for meeting today's high-quality standards and tight production deadlines. UCM raises the level of abstraction of these changes to view them in terms of activities. It then leverages the Rational ClearQuest workflow engine to link activity management to associated development artifacts.

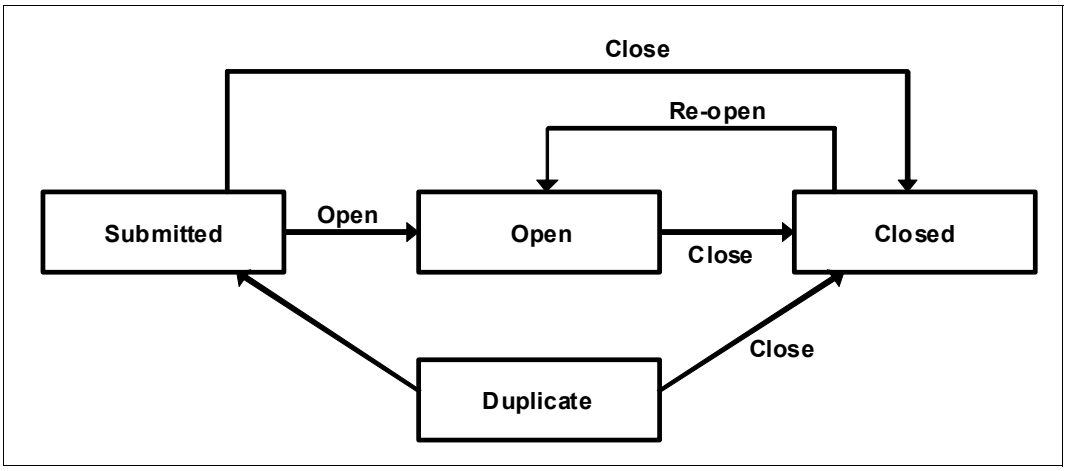


Figure 3-4 Rational ClearQuest's out-of-the-box process framework for defect and change tracking

Artifact management

Rational ClearCase provides the software configuration management infrastructure that UCM leverages to automate the management of artifact development across the project life cycle. Rational ClearCase offers:

- Secure digital artifact storage and versioning
- A parallel development infrastructure with unlimited branching capabilities coupled with sophisticated merge functions
- Workspace management for selection of the correct versions of artifacts
- Complete scalability from small, co-located project workgroups to large, globally distributed teams

Rational ClearCase also provides a flexible infrastructure for implementing SCM. With embedded metadata, triggers, and hyperlinks, teams can use ClearCase to implement custom SCM processes that are based on the level of process enforcement that is required, for example, to meet auditing and maintenance requirements, teams can attach *comments* metadata on check-in. They can enforce policies that manage the use of these comments to insure that they exist, meet minimum length standards, and are spelled accurately. Other teams might find it sufficient to simply 'recommend' policies for completing comments.

With Rational ClearCase, organizations benefit from the flexibility of using different policies for different teams or projects. With UCM, they get an out-of-the-box process that automatically implements these project policies based on proven, successful development processes.

3.3.3 Change requests, features, and packages

Working in a change management environment requires that developers identify activities (the why) to perform a set of changes to a set of files.

Activities (in a Software Development Life Cycle) are usually generated from a need, which could be a new feature to implement, a new set of functionalities to add, a bug fix that is related to a problem that is discovered in the production or test environment, a change that is related to a business or compliance need, and so on.

One of the most critical phases of software development is the ability to trace and control the status of changes and each person's responsibilities. Project Managers and Team Leaders need to assign tasks to developers and verify the related status. The roles that are involved need to act at the right time depending on how work is performed and completed. Also, each role must be focused on the right information to manage, without risks of performing the wrong operation on the wrong objects, which is why in a change management process you must introduce the concept of *Change Set* (assets modified for a particular activity), *Change Request* (or activity), *Feature* (set of Change Requests) or *Package* (set of Features), as shown in Figure 3-5, to help better organize the work of all roles and to have the ability to control each phase of development and automate (where possible) operations and procedures.

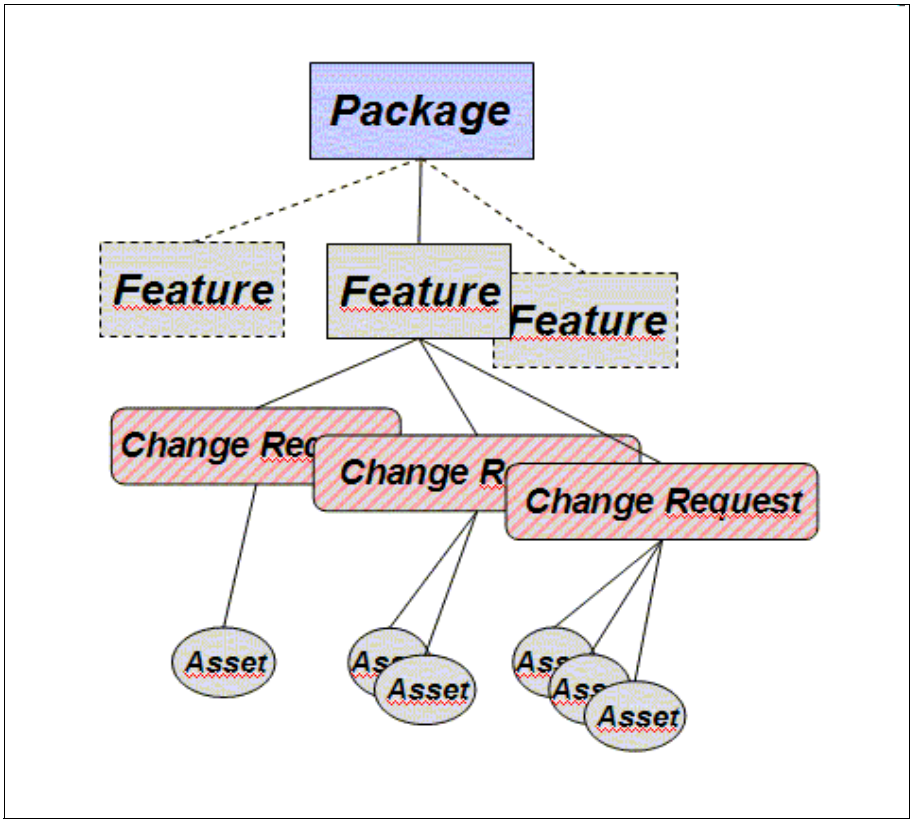


Figure 3-5 Relationships between changes

Without this ability, software development becomes complete chaos, where developers and other roles operate in a chaotic environment and nobody knows exactly what to do, with known related problems.

Using the IBM Rational product line (ClearQuest, ClearCase, and BuildForge) these concepts are built-in and very easy to use, implement, or customize.

ClearQuest can store and manage Change Requests (related to a change management process that is defined internally) with all of the information that helps to better identify its characteristics (for example, Headline, Owner, Dates, Priority, Severity, Description). It is really easy to customize forms and fields depending on the organization's needs and requirements.

Also, the possibility to relate and connect different types of change requests using the parent-child concept is a native functionality that ClearQuest provides.

Using these kinds of relationships, you can define procedures to perform operations that relate to a common set of activities, for example, during a promotion from development to integration, from integration to pre-production, or a synchronization of release activities on different platforms (for example, the front-end on an Application Server and the back-end on a z/OS environment). The same applies to operations that promote or authorize the release of new change requests that must be treated as a complete set of changes instead of single, independent files to release.

3.3.4 Parallel development

If you are a programmer, then you are already familiar with the concepts of serial and parallel development. *Serial development* occurs when a single release of software is developed, and only one person is working on a given file at a time. Defects that are fixed during software maintenance are included in the next release. *Parallel development*, as shown in Figure 3-6, occurs when individuals or teams that are working on different releases make changes to the same files at the same time. The changes that developers make in parallel are later merged, along with fixed defects from maintenance.

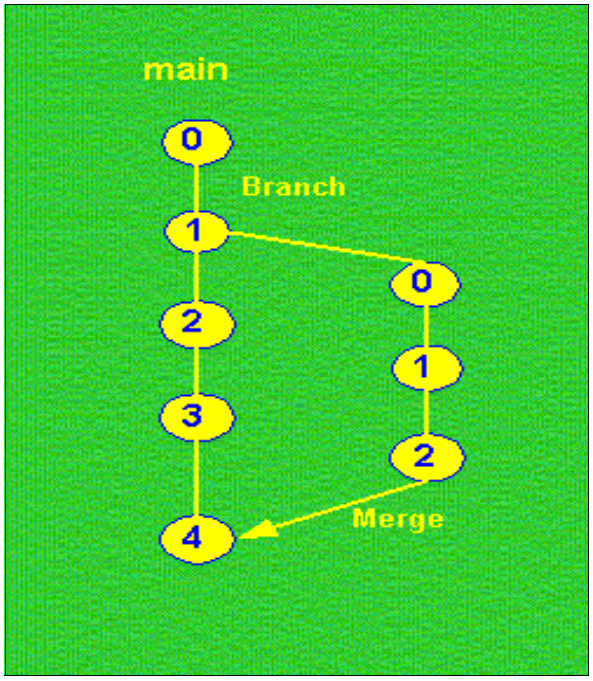


Figure 3-6 Serial development

Many software projects, especially in their early stages, follow a strictly linear development progression where each successive version of the software is derived from, and increments, the previous version. The configuration management of such a project is straightforward because in general there is just one "latest and greatest" version of the software, which forms the context in which all development work takes place.

Parallel development occurs when there is a need for separate development paths to diverge from a common starting point, so that there is no longer a single "latest and greatest" version; instead, there are two or more concurrent "latest" configurations where new development is carried on. Also, implicit is the potential need for the divergent development paths to converge again, which means that any strategy for development branching should also take into account the process for merging.

There are many reasons why parallel development might occur, which includes:

- Need to maintain a software release while new development is in progress
- Urgent bug fixing in production while new features are under development
- Variants of standard configurations
- Isolation of individual work from other developers
- Need to separate development from integration, from test and production
- Staging or temporary areas

3.4 Workflow management

Another fundamental requirement for the implementation of an ESCM solution on heterogeneous environments is related to the process. Attempting to manage a Software Configuration Management system and a Change Request Management system without establishing a formal process, without defining the roles that participate in this process, and without determining responsibilities and deadlines, usually leads to a situation of total chaos.

The process that you implement needs to allow individuals to carry out a number of activities (bug fixing, enhancements or maintenance) in parallel and ensures that the work that the different roles perform are involved in the process and are:

- Authorized
- Secure
- Guaranteed to produce a coherent and consistent final result

To this end, you must establish working environments to encapsulate the work of the different roles and to allow each role to perform his or her assigned functions in complete autonomy.

In an ESCM process, there are usually various types of activities that are linked to the type of modifications to make to the software. You can implement the following types of activities and associate different workflows to them:

- Bug fix
- Enhancement
- Maintenance

Bug fix activities aim to debug or resolve issues with released software. This type of activity usually involves a fast release cycle because little time might be needed to resolve bugs due to blocking mechanisms in the production phase.

Enhancement or maintenance activities are oriented towards implementing new functions or improving new application releases. This type of activity involves a more rigid release cycle with more testing phases because it is necessary to make sure that new software releases conform to previous releases.

You must integrate both activities into the production process to prevent, for example, the repetition of previously corrected errors in subsequent releases.

The logical workflow to implement could also entail parallel lines that lead to a single production environment. Figure 3-7 on page 41 shows an example.

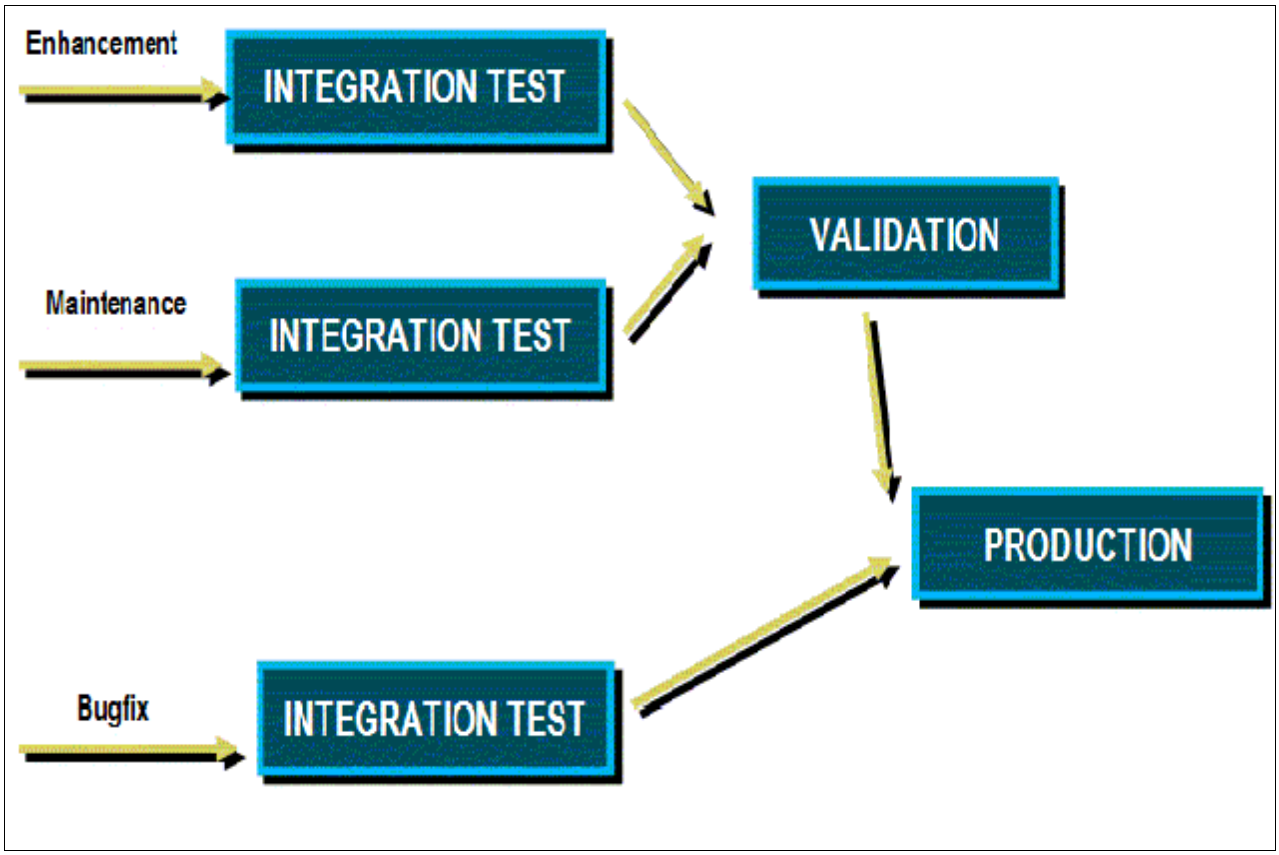


Figure 3-7 Software development life cycle provides support for different types of change requests

There are work areas upstream of each flow line where you make modifications using the described interfaces or IDE development tools. These areas are where you perform unit tests and conflict resolution operations whenever parallel modifications are made to the same objects.

You can further make the enhancement and maintenance activities parallel (enhance1 and enhance2) because the release times for this type of request can be very long. This process allows developers to group functions to be inserted in the new release and to reserve other functions for a later release.

The production environment (usability) coincides with the environment where the baselines are generated for software to be released. After a release is sent into production, it becomes the starting point for further developments and is also aligned with other development and testing environments.

3.4.1 Change management

As stated in the previous paragraph, the only way to control an ESCM environment is to implement a process that allows users to access artifacts and to perform changes according to a need. Of course, all information is related to operations that are performed, who did what, when and how, so that traceability is kept under control to guarantee that the system is organized, roles work in the same way, information is shared and passed through them, authorizations and phases are respected, and synchronizations are guaranteed.

It is part of this area that you must define processes that are flexible and that allow administrators to fit the change management process to the business needs without creating overhead to end-users, while simultaneously maintaining complete control of the system.

Designing a process or a series of processes that fit these requirements is not easy at all, which is why, during the first phases of the analysis, it is important to collect all of the information and to understand exactly what model to implement.

Also, different projects might need to have different processes, which depends on the:

- Phase the projects are in (inception, elaboration, construction, transition)
- Number of developers involved
- Need to have parallel or serial development
- Number of environments (logical and physical)
- Need to manage different type of change requests:
  - Bug fix
  - Maintenance
  - Enhancements
  - Parent-child relationships

You want to have flexible processes that you can assign to projects and then review during the project's life cycle, for example, one project that starts from scratch might require only configuration management functionalities without the need to immediately trace all change requests. This is because the first phase is basically made of development (shared by all developers) with no integration.

Using IBM Rational ClearQuest as a central repository where you store all information that is related to projects, characteristics, attributes, associated processes, and activation flags, can help in the identification of all projects' information and changes.

Let us see a possible situation that we can implement, which is an extension of the standard model that we defined in previous chapters and that can help organizations to better fit their needs.

First phase: Identify project and define related information

When a new project starts from scratch, the first requirement to cover is related to the configuration management area; therefore, the you must have the ability to keep changes to the code, history, and versions under control. At the same time, users need to collaborate and share code, while they work on the same or latest version. There is no need for parallel development, merges, or integrations at this time. At the same time, the system must be easy to use and adopt without overhead for the roles (developers, project managers) that are involved.

Although maximum flexibility is required, we strongly suggest that you maintain central control of these projects that start and evolve during ensuing months to ensure that all information that resides at the corporate level is in sync and checked.

IBM Rational ClearQuest can help at this juncture with "stateless" tables, where we can define applications, components, roles, profiles, and so on. The application table, as shown in Figure 3-8 on page 43, can store all of the information that is related to projects, such as type, associated processes and workflows, and activation flags.

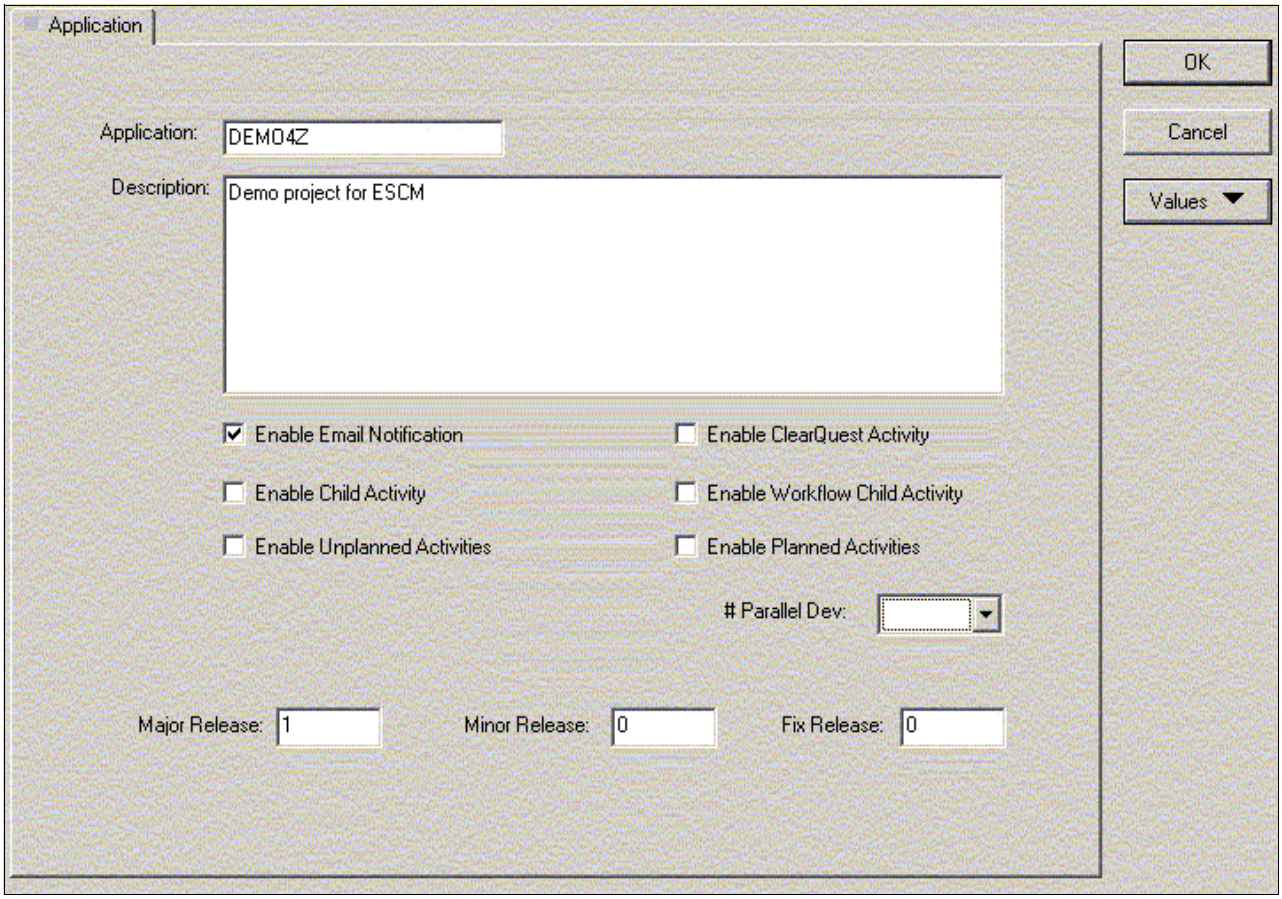


Figure 3-8 Using Rational ClearQuest to store information

Using such parameters, we can define and enforce different types of processes and scenarios. At the same time, we are sure that all applications are stored in a single table and are accessible to be verified and modified. A Change Management process relies completely on the information that the administrator manages.

When a new project starts, flags, such as “Enable ClearQuest Activity”, “Enable Child Activity”, “Enable Workflow Child Activity”, “Enable Unplanned Activity” and “Enable Planned Activity” are disabled. The ClearCase repository uses this information to define an environment (based on ClearCase-base model - no UCM) where all developers can work in the main line branch, as shown in Figure 3-9 on page 44.

This working model, gives the following advantages:

- No need to create a complex UCM environment
- No need to perform UCM operations, such as delivery or rebase
- No need to create activities in the first phase. History, versions, traceability, and permissions are guaranteed
- No need to migrate the repository (export or import) when the process and model will evolve
- The possibility to create few views and workspaces in ClearCase and to share work between different developers (both on distributed and mainframe environment)
- The possibility to create labels to freeze the completed work and to set milestones

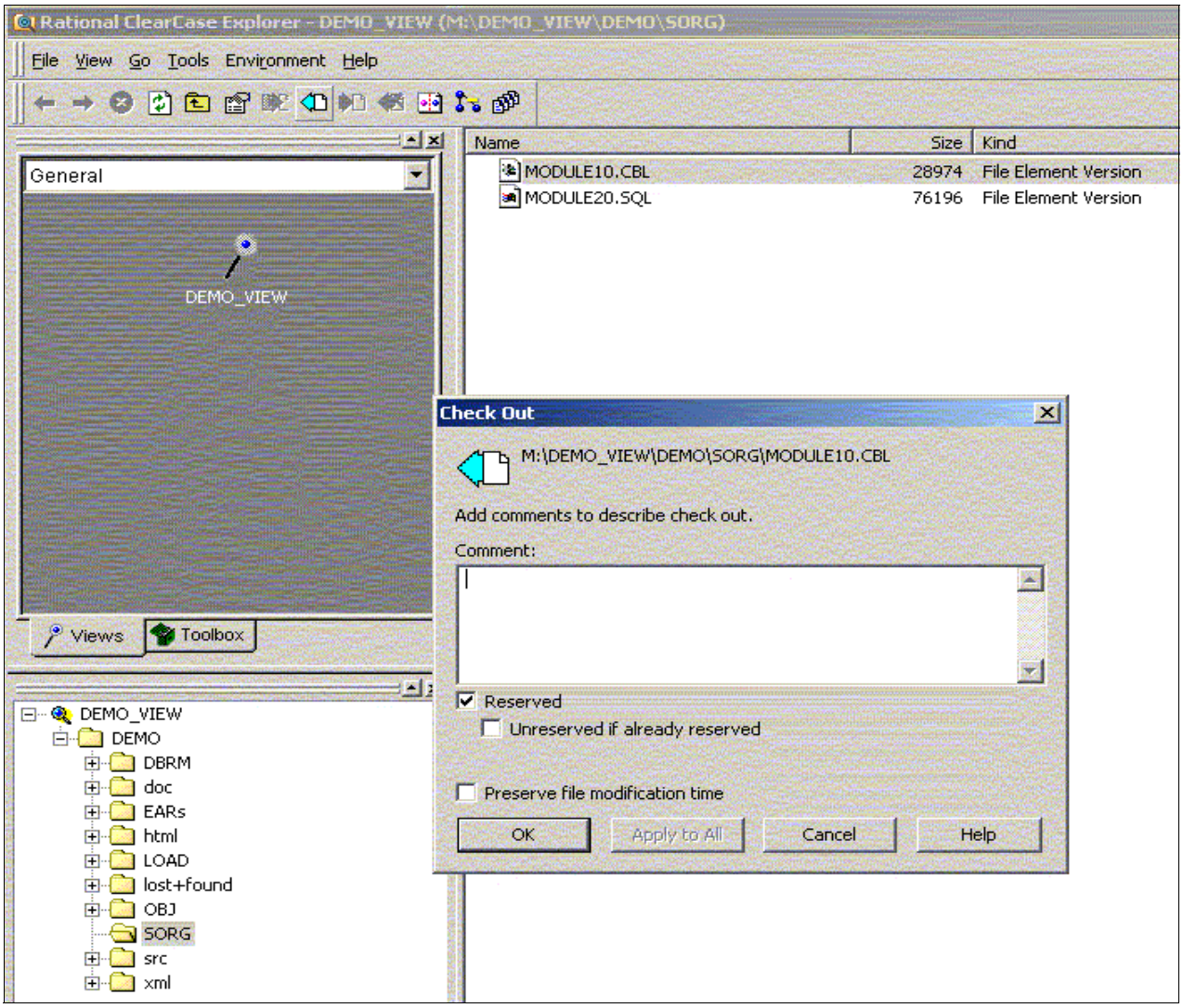


Figure 3-9 ClearCase base view to access source codes and share changes

Using this approach, the productivity of the team, the collaboration, and the reduction of number of operations to execute, are at the maximum level. Also, conflicts, on changes to the same file, are avoided. Checkout reserves are always performed and help developers to work on each file without needing to merge that file with other changes.

This phase typically requires a substantial amount of time to develop because it is the first iteration of the project to complete. When Team Leaders or Project Manager decide to freeze and promote work to the integration area, the second phase can start.

Second phase: Test and fix defects

During this second phase, artifacts are ready to be delivered into the integration area and deployed into the testing environment.

In this phase, recycles on fixes and changes occur to correct and re-deliver changes for the test activities, which is why development must now work on a separate branch to guarantee that changes occur in a separate environment and do not create confusion with test activities.

To enable these parallel activities, go back to the ClearQuest Application table, and we can activate the flag that is related to “ClearQuest Activities” and “Unplanned Activities”, as shown in Figure 3-10. Activating the first flag means that we are now moving from a ClearCase base model to the Unified Change Management model. Also, we start to use ClearQuest activities (record type) of type “unplanned” (defect or bug fix).

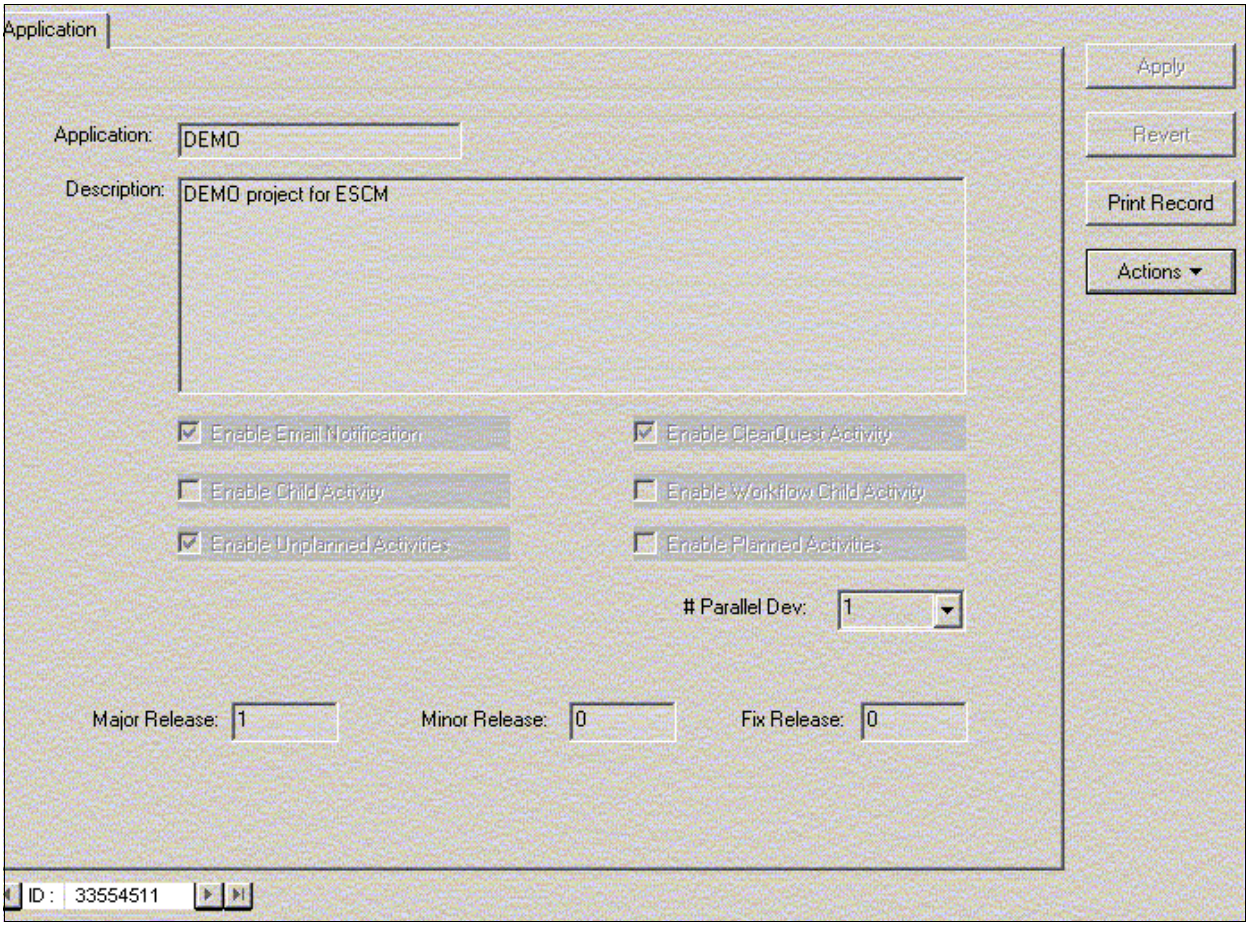


Figure 3-10 Flags are activated to move from a ClearCase base model to a UCM model

Here, you can use flags from external procedures to generate the UCM environment of the project. In particular, you can use Project Version Object Base (PVOB), components, streams, and baselines to govern changes and to trace activities during the entire software life cycle.

Accessing ClearQuest to extract this information and then work on ClearCase is pretty simple. Example 3-1 contains sample code that you can generate.

Example 3-1 Generated code

```
$cquserdb=@ARGV[0];
$cqtable=@ARGV[1];
$cqkeyname=@ARGV[2];
$cqkeyvalue=@ARGV[3];
$cqfieldname=@ARGV[4];

$CQSession = CQPerlExt::CQSession_Build();
$CQSession->UserLogon($cqlogin, $cqpasswd, $cquserdb, $cqschema);
$queryDefObj = $CQSession->BuildQuery($cqtable);
```

```
$queryDefObj->BuildField($cqfieldname);
$operator = $queryDefObj->BuildFilterOperator($CQPerlExt::CQ_COMP_OP_EQ);
$operator->BuildFilter($cqkeyname, $CQPerlExt::CQ_COMP_OP_EQ, [$cqkeyvalue]);
$resultSetObj = $CQSession->BuildResultSet($queryDefObj);
$resultSetObj->Execute();
$fetchStatus = $resultSetObj->MoveNext();
while ($fetchStatus eq "1") {
   $cq1 = $resultSetObj->GetColumnValue(1);
   print ("$cq1\n");
   $fetchStatus = $resultSetObj->MoveNext();
}
CQSession::Unbuild($CQSession);
```

The result of such flags, after the activation of this second phase, is the creation of the UCM environment of the information and objects that you need to work, as shown in Figure 3-11.

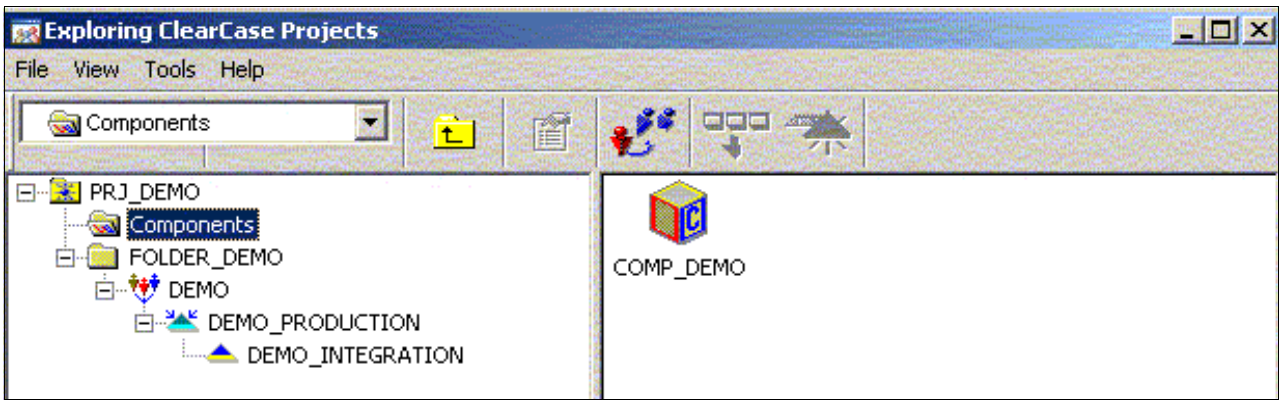


Figure 3-11 UCM is now activated to allow development and test environment to work isolated

From now on, all changes must be performed using a ClearQuest activity that must be opened and assigned to the developers, which helps to trace all requests (for new features or to fix errors that are discovered in the test environment). The Project Manager now has the complete picture of how things are going, who is working on what, and when and which files change.

Again, during this phase, parallel work between enhancement requests and bug fixing is not required. All changes are made to improve the version of the software that is deployed into the test environment. At this time, there is no need to have separate work.

The advantage for developers here is to have their own stream, or environment, where they can make changes and decide when these changes are ready to deploy into the integration area for test activities.

The duration of this process depends on the number of bug fixes that are discovered in the test environment. New change requests are generated to notify everyone of new issues, or the same activity can iterate several times until the problem is solved, as shown in Figure 3-12 on page 47.

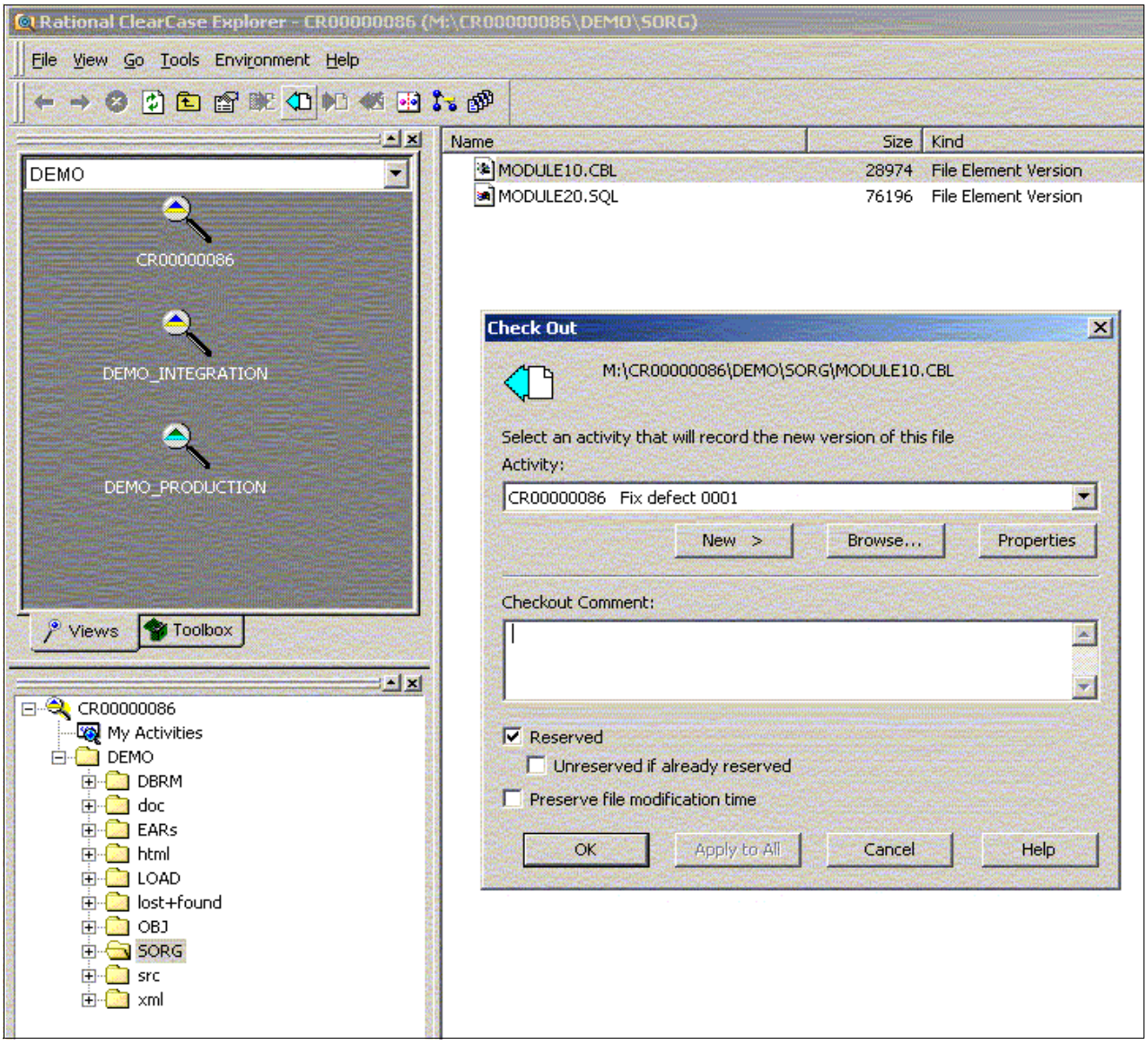


Figure 3-12 Using UCM, checkout operations are connected to ClearQuest activities

Another benefit of this second phase is the possibility to create automatically private branches when new activities are generated. UCM allows several ways for this to work. During the analysis phase, it is important to define and design the right model to use. Having separate branches for each ClearQuest activity, as shown in Figure 3-13 on page 48, has several advantages and disadvantages.

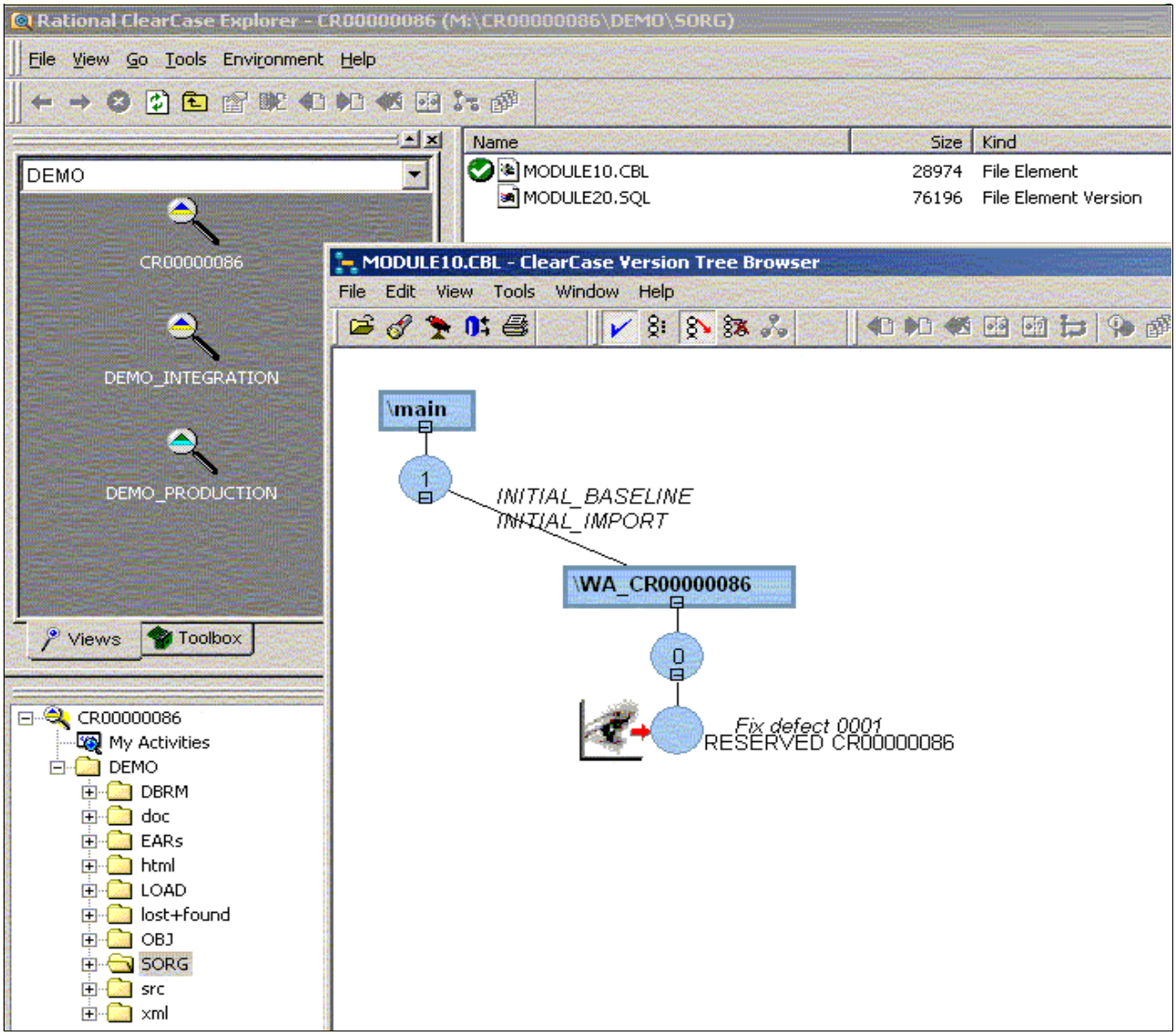


Figure 3-13 During the checkout operation, a dedicated branch is created to isolate the developer's work

Advantages:

- No conflicts during changes
- New change requests do not inherit changes that are not confirmed
- Parallel development
- Deliver and rollback several times

Disadvantages:

- Developers need to merge, in case of conflicts
- Delivered changes are not automatically inserted into the parallel development

After completing the test phase in the integration or test environment, you can promote the new release (set of files) to the production environment, as shown in Figure 3-14 on page 49. This activity is performed in ClearQuest, promoting all activities that are ready to be inserted into a new baseline.

At the end of this phase, we have our first new release or baseline in production, ready for end-users to use.

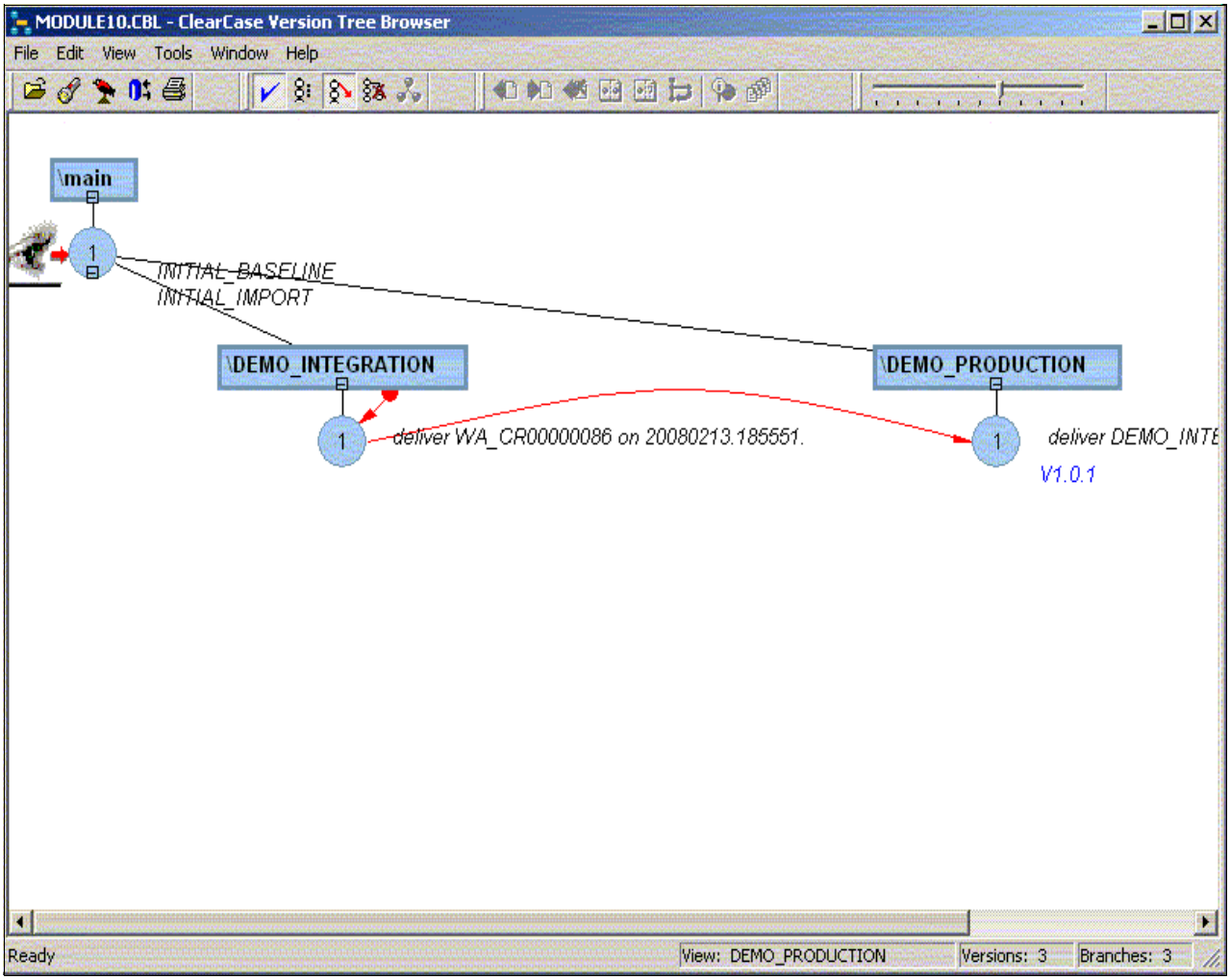


Figure 3-14 A new baseline is generated into the production environment

Third phase: Develop new enhancements and fix defects in parallel

The third (and last) phase corresponds to the final and complete environment where developers work in parallel to create new features, maintain applications, and fix defects that are discovered in production or in test. This is the configuration that is associated to applications that already exist in production and were implemented in the beginning, if the project did not start from scratch.

Working in parallel in this phase is one of the most important requirements because the development of new features and enhancements might require several weeks or months of work and resource investment. It is almost impossible to think to work on a single line for both enhancements and bug fixing activities and to collapse all developers' work in a single delivery.

In some situations, it is required to have several parallel development lines for enhancements with each line isolated from others and a separate physical/logical environment where testers can perform first integration tests. Then, parallel lines merge in an integration area (called pre-production or validation) before they are promoted to the production environment.

The enhancement development line and bug fixing development line continue to work in parallel until the production environment is ready. After a new set of files is deployed in

production, you must verify and merge (if required) files that are in conflict with the new release to avoid reintroducing a problem that got fixed in a previous delivery.

Also in this phase, ClearQuest flags help to set correct policies and processes, as shown in Figure 3-15. Each project can have its own flags, which gives the maximum flexibility to the ESCM implementation, while at the same time maintaining the complete control/govern of the ESCM process.

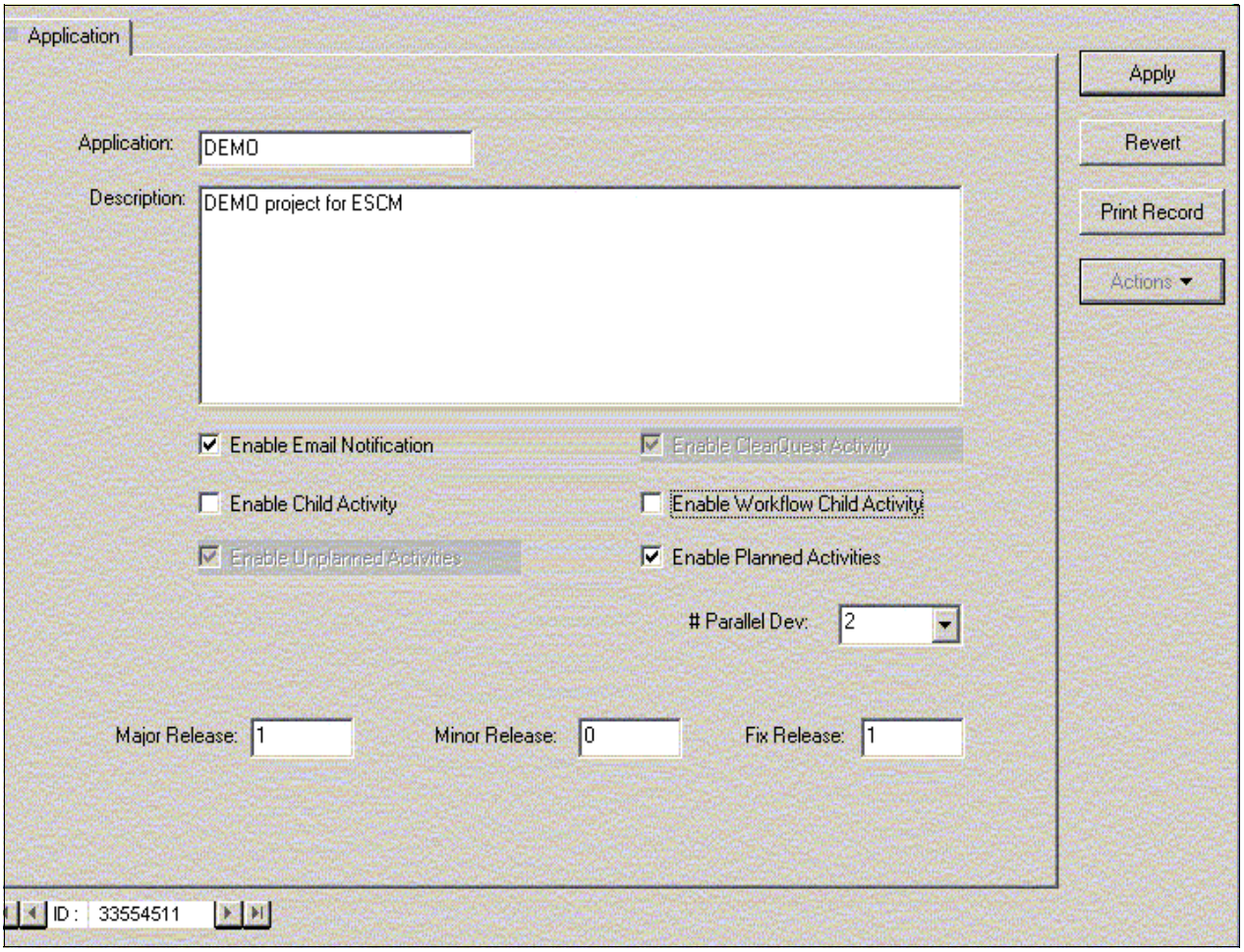


Figure 3-15 When all flags are activated, the ESCM process is fully enabled and configured

For the second phase, also in this environment, employees can work in parallel, as shown in Figure 3-16 on page 51 and Figure 3-17 on page 51 or share a common set of changes. In Figure 3-16 on page 51, each activity that is generated as a “planned” activity (could be an enhancement or a maintenance request) has its own branch where single developers work isolated. In Figure 3-17 on page 51, it is also possible to separate work between developers (everyone has his/her own activity to work), but all work belongs to the same branch or stream. This scenario is implemented by enabling “Child Activity”. The greatest benefit in this case is to synchronize the work of several developers before the complete set of changes is promoted into the integration area.

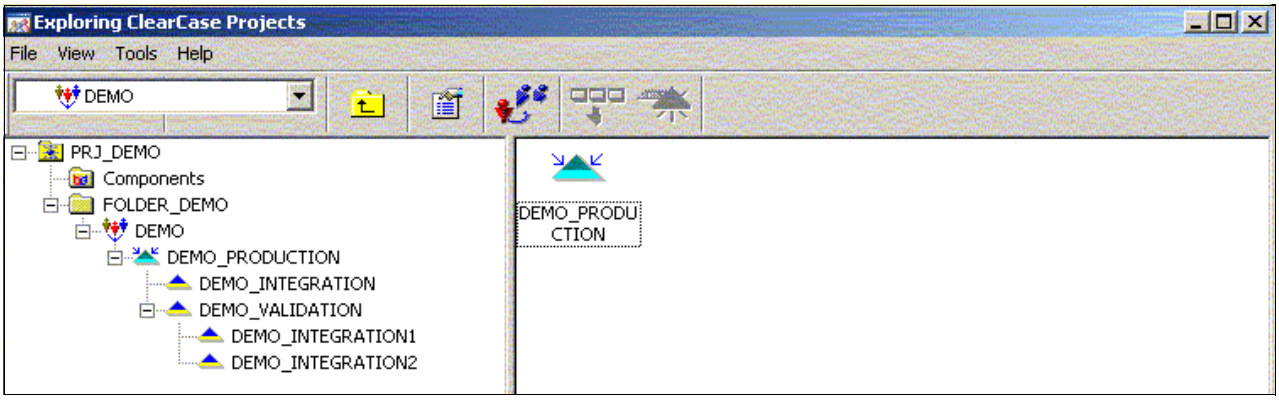


Figure 3-16 With UCM every project can be configured to fit development process and physical/logical environments

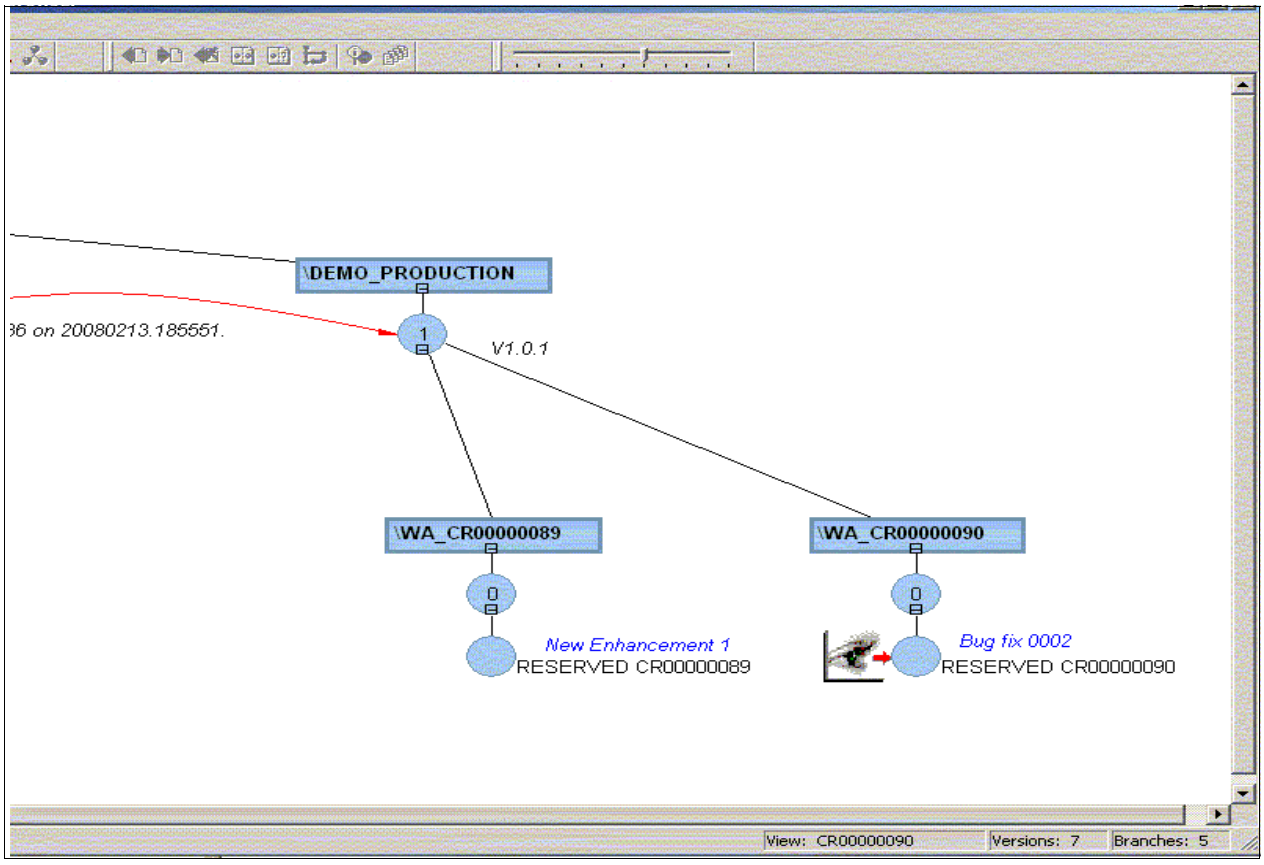


Figure 3-17 Developers can work in parallel or share work on a single common stream

Using “Child Activities” to perform changes allows you to maintain a parent-child relationship between the “master” activity and its “details”, which then allows us also to define different change workflow to control and assign children to developers. Because the developers can perform their work in different time frames, each of them has the flexibility to start and complete their work without affecting others. The owner of the “master” activity can control and verify the status of each child activity and decide how to operate (re-assign an unworked activity, move one child from one master to another, and so on).

The model we described in this section is only one of the several ways to work and implement a change management workflow. The key is to perform the correct analysis of workflows and implementations to put in place to reach the correct usage model for each organization.

4

The Rational Enterprise Software Configuration Management implementation

In this chapter, we present an approach that we refer to as the Enterprise Software Configuration Model (ESCM). In this chapter, we present the elements of the solution and how they are combined to support an enterprise-wide Application Lifecycle Management solution. We assume that you have in-depth familiarity with the administrative features of Rational ClearQuest, Rational ClearCase, and Rational Build Forge. You also need knowledge of Rational Developer for System z.

Figure 4-1 shows an overview of the Rational solutions that we use in this book.

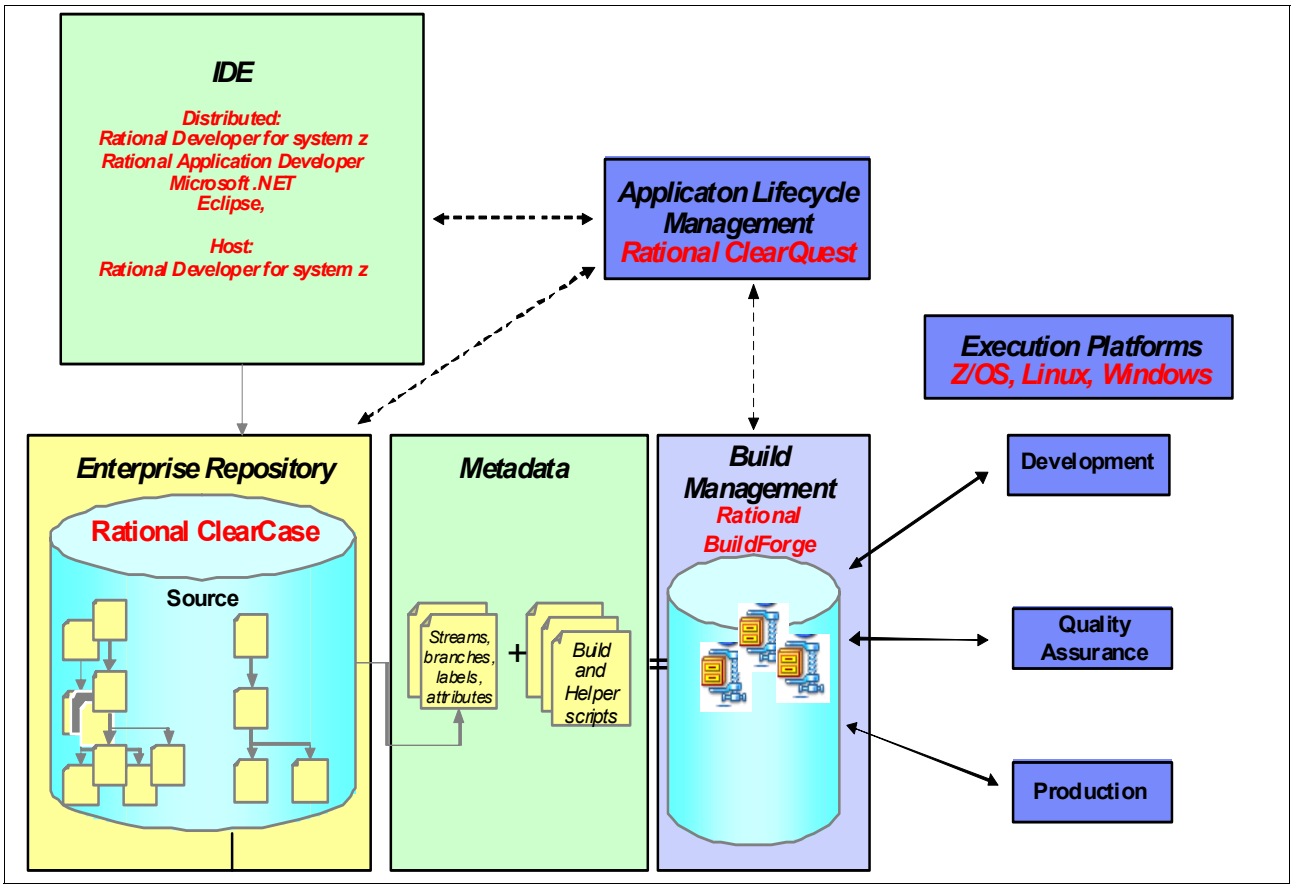


Figure 4-1 ESCM supporting architecture

4.1 Enterprise repository

If you implemented a robust and effective SCM solution with IBM Rational ClearCase, in a distributed environment, you know that there are more similarities between the mainframe and distributed environments than there are differences.

Most of the differences are in terminology, but after you identify and cross-reference those differences, the most successful implementations make use of some long-established, but little known features of ClearCase. The ClearCase Extensions for z/OS are a more recent addition to ClearCase (2003.06 and later), and as their name implies, they enable you to create an Enterprise SCM that supports native z/OS builds and provides direct access to a ClearCase repository from ISPF. In earlier chapters, we presented an ESCM process as a concept. In this chapter, we describe the use of IBM Rational solutions to implement an ESCM centralized repository and life cycle. In our implementation, we combine the change and application management capabilities of IBM Rational ClearQuest and the cross platform build management.

Regardless of the solution that you use, to support an enterprise-level SCM process, the centralized repository should offer the following capabilities:

- Scalability to host thousands of files, regardless of the host or target platform
- Support for identifying and organizing groups of files as independently managed components, regardless of the host or target platform
- Support and enforcement of a standardized application life cycle
- Support for role-based access
- Transparent access from host or distributed portals

These characteristics enable you to create a comprehensive application management infrastructure with the robustness to manage business critical applications and the flexibility of effective use in both the distributed and z/OS mainframe environments.

Let us now examine how to use the capabilities of IBM Rational ClearCase to create a centralized repository that provides robustness and flexibility. In this chapter, we assume that you have a working knowledge of IBM Rational Clearcase. We present a discussion of those features of ClearCase that are most relevant to deployment in an enterprise application SCM.

4.1.1 Managing builds

Building an enterprise application from distributed and host components must be efficient and reliable. A critical difference between a distributed application and building a z/OS application is the build process, particularly the methods that you use to configure the compile, link, and deploy steps.

Modern Java, C++ and .NET applications are built with the aid of sophisticated IDEs, such as IBM Rational Application Developer or Microsoft® Visual Studio®, in the context of a project. The IDE automates and abstracts the compile, link, and packaging steps from the developer. The developer performs these steps either on a developer workstation or a managed build server using information that can be stored in project files, makefiles, or environmental settings in the IDE.

z/OS applications are compiled, linked, and deployed with Job Control Language (JCL) build scripts, which are submitted to the host. The build script contains all of the information about the compile and link process. Modern IDEs, such as IBM Rational Application Developer for z, have features for automating the creation and submission of these scripts. This distinction is significant because the distributed application, build rules, and dependency relationships are maintained in the IDE and in local build scripts. In a ClearCase distributed environment, dependencies can be automatically inferred by using clearmake or by invoking a build script in a clearaudit shell. In the z/OS application, the build script is the only way to make this information available to the z/OS batch environment.

In the z/OS environment, you can configure the build process and set the compiler, pre-link, and link options by passing parameters to the build scripts or by invoking the command line invocation of a build script. In the distributed environment, you can do this by selecting build options in the IDE.

To deliver the benefit of a single repository, the standard process approach of an ESCM environment must be able to manage these two different approaches with a consistent paradigm.

In our project, we employed a straightforward approach.

4.1.2 Organizing and identifying assets

At one level, the addition of the z/OS mainframe assets to a ClearCase repository is straightforward. You can store practically any type of file in a *ClearCase Versioned Object Base* (VOB). You can store COBOL, JCL, CICS®-COBOL, and any other type of source member can in a VOB and access them from any desktop application and from ISPF.

ClearCase performs Extended Binary Coded Decimal Interchange Code (EBCDIC) to ASCII translation transparently, and enables both the Eclipse user and the ISPF user to work in parallel with the same physical repository. The ClearCase paradigm of making all of the elements visible, but read only unless checked out to the user, is extended to the ISPF user. The usual developer actions of select view, check out, and check in, all behave in similar or identical fashion, which means that the primary considerations in planning the implementation of an ESCM solution with ClearCase are architectural and organizational, not technical.

We begin by analyzing the existing enterprise software asset base and determining how to organize these assets into a ClearCase repository. There is a considerable amount of knowledge that is available to guide organizations in this process, There is no technical limit on the size of a Clearcase Versioned Object Base. Performance characteristics of the server and network and the nature of the access to the repository do impose some constraints.

Aside from the size constraints, there are no hard technical requirements from ClearCase that influence this process. Architectural and organizational factors dictate a logical structure. The build and release process is a significant factor. The significant point here is that there is no technical requirement from ClearCase, to either segregate or co-locate z/OS modules with other assets.

The main consideration is architectural. We start off with the simplest approach, which is to allocate a top-level directory for each subsystem. You can start off the same way. Assets that are shared across subsystems, such as copybooks, database access routines, and so on need special consideration. Changes to these shared assets can affect multiple subsystems; therefore, they require careful management, which can be a good argument for maintaining them in a separate VOB or directory where you can choose to employ more stringent access and approval levels.

4.1.3 Using ClearCase meta data in z/OS builds

We use ClearCase meta data and a set of helper scripts, or alternately, a set of trigger scripts to manage the parameterization of the z/OS builds. There is an example of a helper script in the appendix. For more information about Clearcase triggers and meta data, refer to the ClearCase Reference Manual.

When a build is invoked, the scripts use calls to the *ClearCase Automation Library* (CAL) to retrieve the meta data, set the relevant parameters, and construct a build script, which we describe in detail later in the chapter.

Table 4-1 on page 57 lists some of the project and element level attributes that we use to implement the components of the build process. The names and data types are arbitrary and relevant to the implementation at the time. Choose attribute names that are meaningful in your organization. There is no difference between project level and element level attributes as far as how they are created in ClearCase. The distinction simply designates what the attribute in question is used for.

Table 4-1 Project and element level attributes

Name	Type	Value	Function
PGMRNAME	string	SMITH.K	Annotates the programmer name
AUTHPARMS	string		
CICS	string		
COPYLIB1	string	TAX.CPY.INTRT	Specifies the required copybook1
COPYLIB2	string	DATE.CPY.YEAR	Specifies the required copybook2
COPYLIB3	string	N	
LOADLIB	string		
SRCLIB	string		
LANG	string	COBOL	Identifies the compiler
COBOL390	string		Invokes COBOL390 option
PGMDESC	string		Name or description of the program
DB2_BIND_MEMBER	string		
PGMSTAT	string		

Assigning element-level attributes

In this approach, we assign build information about a member-by-member basis, by assigning a set of attributes that vary depending on the characteristics of the source member. We also identify a set of what we refer to as project-level attributes that carry information that applies to the entire subsystem and z/OS infrastructure. Depending on the infrastructure, these attributes are related to syslib concatenation, which relates to copybooks and load modules, DB2 subsystem, CICS regions, IMS™ regions, and so on. Anything that is related to the deployment of artifacts to the proper z/OS environment, for example, in a simple life cycle based on Test, QA, and Production environments, the project-level attributes store the meta data to know each of the target libraries, CICS regions, and DB2 subsystems, which provide all the information that is needed when building or moving to any of the environments.

In the example scenario project, we created a REXX™ exec that analyzes the data on the mainframe and creates a similar set of CAL API calls, which we exported to the ClearCase host and ran as a batch file. Figure 4-2 on page 58 gives an overview of this process of assigning element-level attributes.

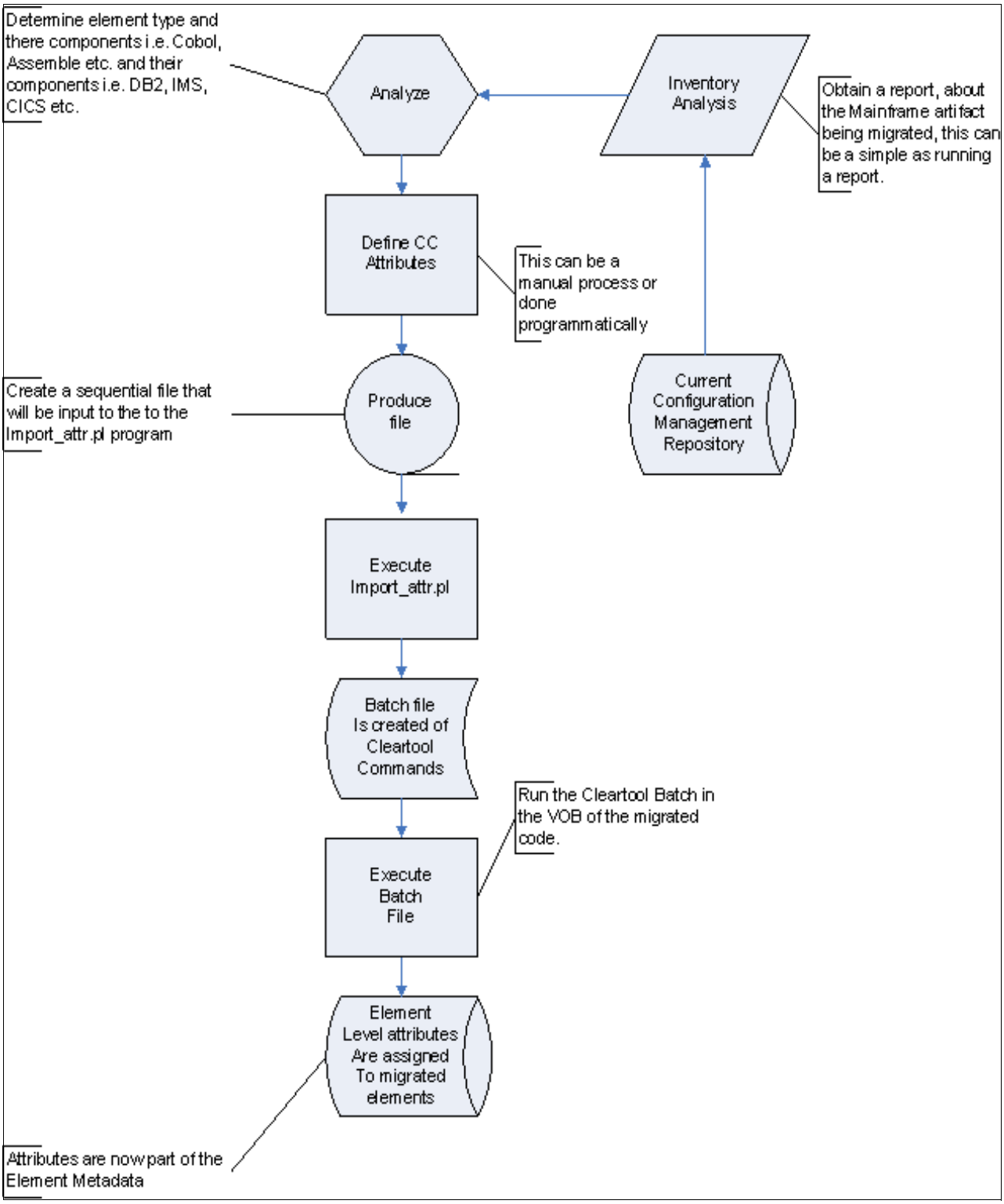


Figure 4-2 Schematic process for importing element-level attributes

Assigning project-level attributes

We use a single ClearCase attribute called ZCCENV to assign project-level attributes. We create a PERL script called SetZCCEnvAttr.pl, which sets the value of the attribute. The purpose of this attribute is to define and map the z/OS infrastructure and life cycle on the

mainframe. The build process queries this data and uses it to determine where and how the project will migrate through the life cycle. If a change is needed or a different migration path is necessary as development proceeds, the ClearCase administrator need only change the values with the ZCCENV definition for the project that uses the SetZCCEnvAttr.pl script. The build engine (described later) uses the new values and continues on in the new life cycle, or it moves to the new z/OS infrastructure for testing and QA. Figure 4-3 illustrates this process.

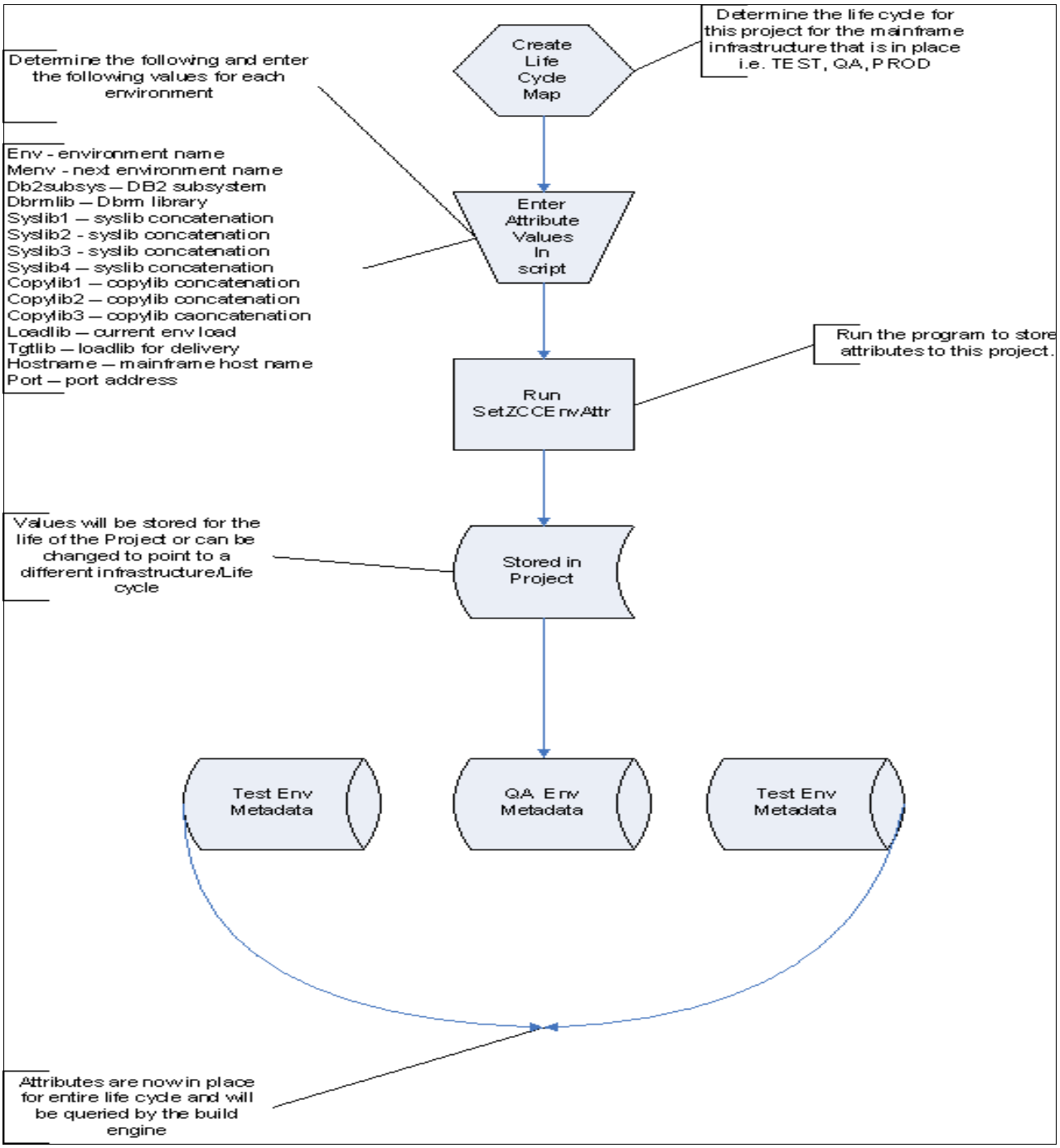


Figure 4-3 Process flow: creating and updating project-level attributes

Creating your repository

At this point, we described the points you need to consider to determine the preliminary structure of your ClearCase repository. We emphasize the word *preliminary*. In our experience, no matter how much effort you put into laying out the structure in advance, it remains fairly dynamic throughout use, changing requirements, and most importantly, as the organization begins to learn and optimize around the use of the solution, and you should plan for this change.

ClearCase is sufficiently flexible such that in most cases, you can implement a fairly simple model, with few controls initially, and iterate towards a more managed approach, adding appropriate controls and approvals in stages as the team gains experience. We found that the more successful organizations are those that use an iterative process adoption model.

With these considerations, be ready to begin to import assets into your new repository. You can use the ClearCase TSO Client to import a few files, as illustrated in Figure 4-4.

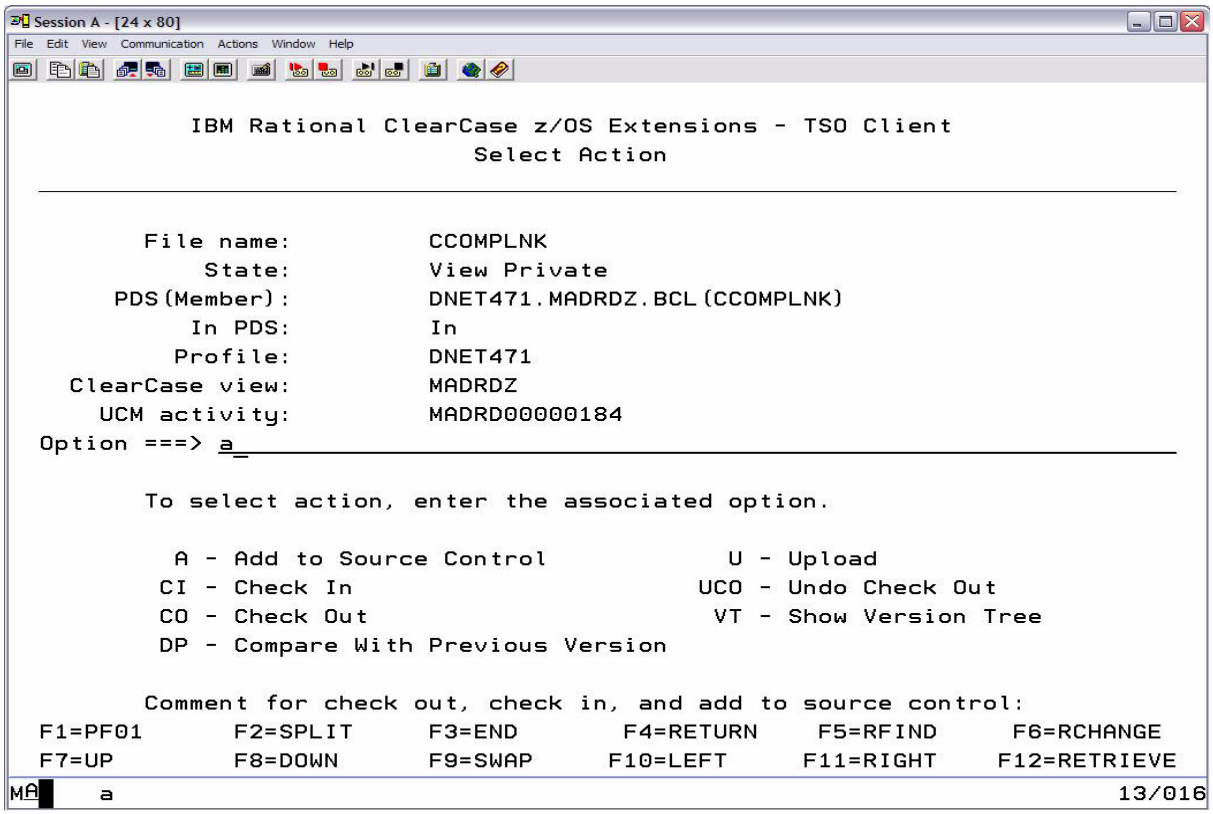


Figure 4-4 Adding PDS member DNET471.MADRDZ.BCL(CCOMPLNK) to source control

To add the PDS member to the source control:

1. Navigate to the member in question. The state view is private.
2. Select **A** to add the member to Clearcase. The window changes to look like Figure 4-5 on page 61.

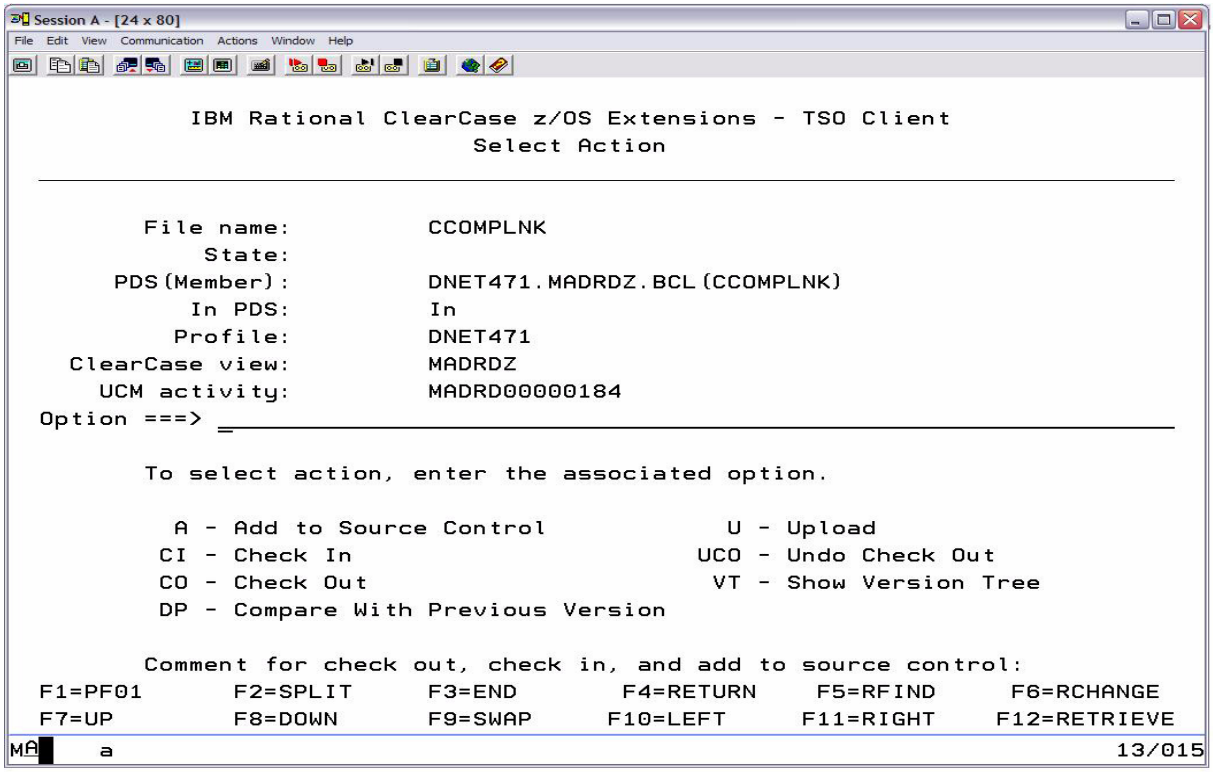


Figure 4-5 Member CCOMPLNK Added to source Control

Note that the State changes from view private to blank, which means that it is now a managed element.

In the more likely scenario where you importing hundreds (or thousands) of modules, you import z/OS assets in three steps:

1. Determine the file system hierarchy.
2. Transfer the assets from z/OS to the distributed file system in the environment where ClearCase is installed.
3. Import the assets into the ClearCase repository.

Determining the file system hierarchy

There are a host of references that can help you determine the file system hierarchy. If you are new to ClearCase, we recommend that you read *Managing Software Projects IBM Rational ClearCase*, which you can view at:

ftp://ftp.software.ibm.com/software/rational/docs/v2003/cc_family/ccase/doc/all/cc_proj/wwhelp/wwhimpl/js/html/wwhelp.htm

We address this topic in detail in Chapter 5., “Starting an Enterprise Software Configuration Management project” on page 101.

The factors to consider in your file system hierarchy are:

- Organization: Which teams are responsible for which subsystems?
- Architectures: What subsystems are interdependent and need to be built or managed together? What subsystems can or should be managed separately?

- Security and access: Who should you allow view or modify assets? Which assets do you allow them to modify?

Defining build and promoting strategy

Although many mainframe shops that we work with employ a build one time and promote strategy, there are those who build at each stage. From the perspective of ClearCase, this determines what host environments you deliver into, and how you want the action recorded.

The definition and configuration of your host environments should be what drives your ClearCase stream or branching strategy. Regardless of whether you build one time and promote the same binaries through the life cycle, or if you rebuild and rebind at each promotion step, we found that the most effective approach is to create a ClearCase integration branch or stream for each environment that you plan to deliver to.

If you build only one time, you identify a single stream or branch from which you deliver to your initial target environment and deliver to the initial target. Subsequent promotes can be performed on the host by the existing processes, or you can perform promotes in ClearCase with ClearCase deliver or merge actions on the load modules. You can then deliver into the target environment, for example, if you have environments called STG, STG2, PREPROD, and PROD, use those same names for your ClearCase streams or branches to make life much easier for everyone on the team. Figure 4-6 illustrates some of these alternatives. We will revisit this topic when we discuss build management in detail in 4.4.5, “Working with builds, migrations, and promotions” on page 88.

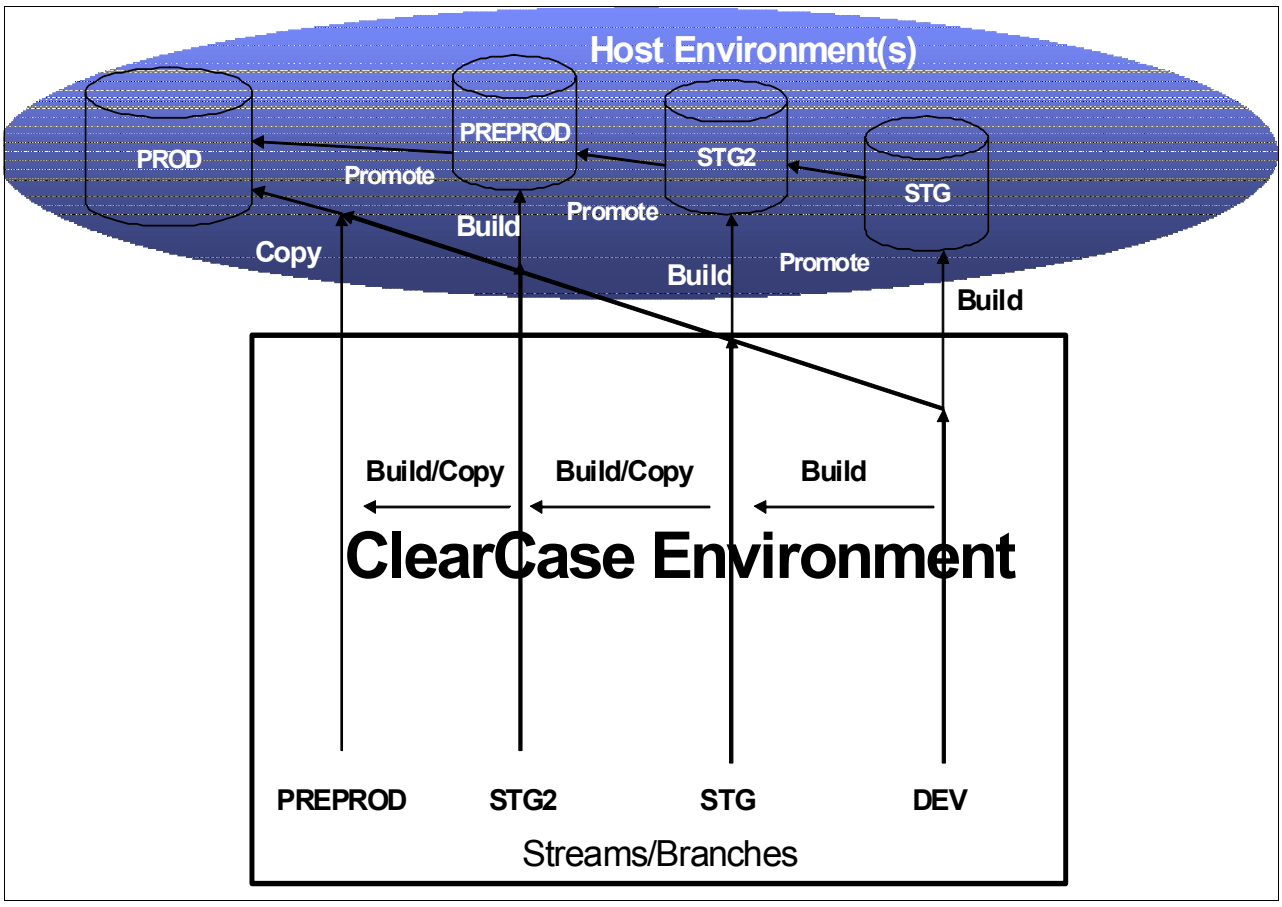


Figure 4-6 Stream strategies for delivery to z/OS

Transferring the assets

There are two steps to transferring the assets: physically transferring the assets and converting them from EBCDIC to ASCII, as shown in Figure 4-7. There are many solutions that are available to make this a simple process, for example, IBM Rational Developer for z (Release 7.1 and later) contains a remote synchronizer that you can use to automate transferring and synchronizing of members between the local Rational Developer for System z workspace and the host.

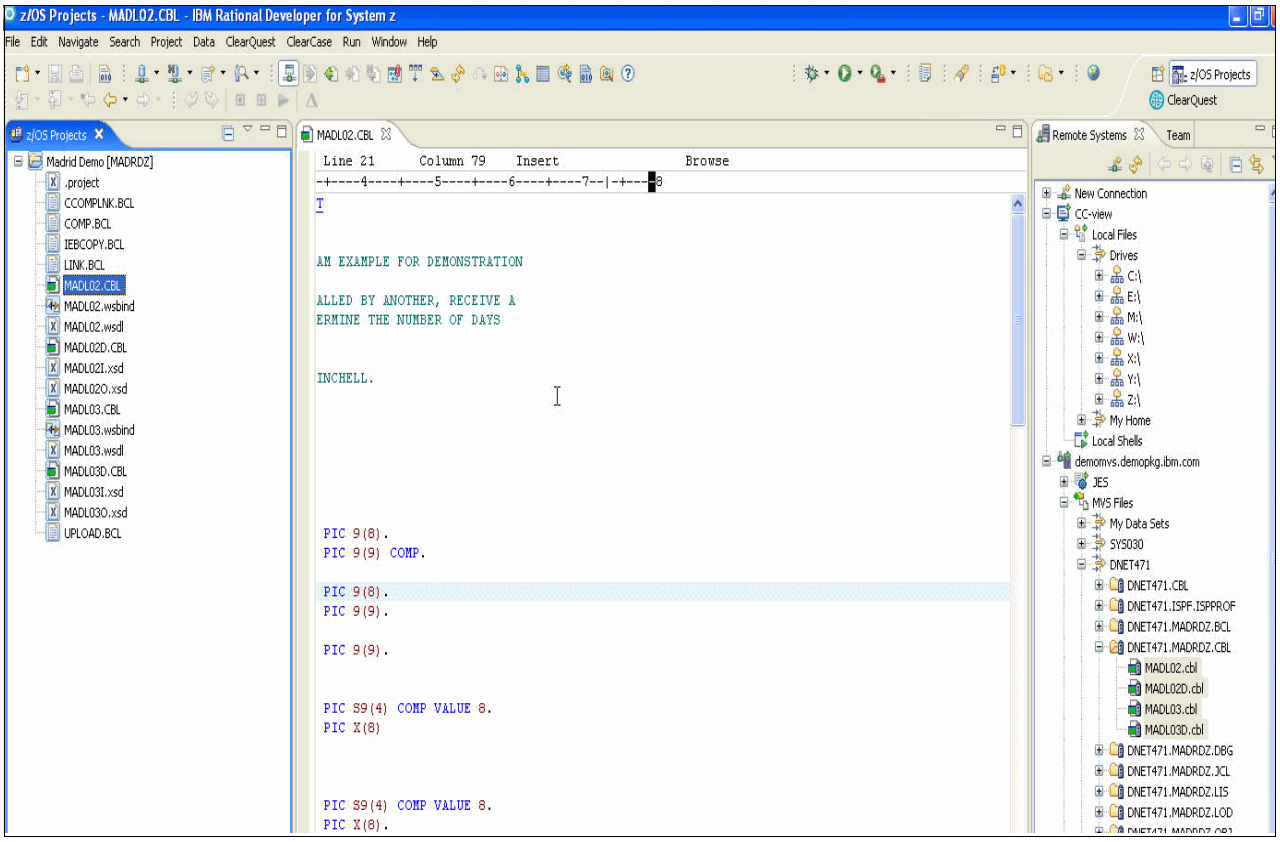


Figure 4-7 Transfer of assets

To ensure that binary files (object or load modules) are not corrupted, you might have to write a script that calls TSO XMIT.

Importing assets into IBM Rational ClearCase

After you transfer your assets to the ClearCase distributed environment and convert them to ASCII, you can import them into your ClearCase environments.

At this stage, you already defined and created your VOB layout. The next step is to create a view and import the assets.

Rational ClearCase includes several features to automate the importation of large numbers of file members. With the **`clearfsimport`** command, you can import a hierarchical file system into ClearCase in a single step and consult the ClearCase manual page entry for **`clearfsimport`** for more details.

You can use the ClearCase Explorer GUI to drag and drop small numbers of files into a ClearCase VOB.

There are more details about importing assets into ClearCase in the *ClearCase Administration Manual*, which is available at:

ftp://ftp.software.ibm.com/software/rational/docs/v2003/cc_family/ccase/doc/all/cc_admin/wwhelp/wwhimpl/js/html/wwhelp.htm

4.2 Managing your z/OS assets with ClearCase

To enable work with z/OS assets, Rational ClearCase includes a facility called Remote Build. Using Remote Build you can submit z/OS batch jobs directly from the desktop. You can use it to build assets that are managed by ClearCase or host-based assets. Regardless of where the assets are, using the ClearCase Remote Build features you can create ClearCase audit records of builds to run on the host. We go into significant depth on this topic in subsequent chapters.

Using the TSO client, you can perform basic ClearCase SCM operations, such as select activity, compare versions, and check out and in from a 3270 emulator. ClearCase manages a shadow PDS and populates it with the versions of assets that are selected by your view's config spec. You can work with both UCM and non-UCM views, snapshot or dynamic. As of ClearCase release 7.01, there is no support for Web or Eclipse-based views.

There is a separate manual, *Guide to Installing and Implementing the z/OS Extensions* that explains, in detail, how to install and configure the extensions. This manual is part of the ClearCase product documentation set, which we frequently refer to throughout this section. The document set is located at:

http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=GI11-6714-01

4.2.1 Using Remote Build

With the proper privileges, you can use Remote Build to submit build scripts to your host system. The remote build architecture is simple. There is a z/OS agent (or started task) that runs on the target z/OS system. Remote Build uses a secure socket and pipe protocol to communicate with and to transfer data to and from the host. Remote Build does not rely on either NFS or FTP; instead, it uses POSIX pipes and sockets to communicate with the ClearCase view server over a defined port.

Both z/OS and UNIX Systems Services versions of the remote build server are included as part of the standard ClearCase product. You can submit batch jobs directly to z/OS, and you can remotely run UNIX Systems Services scripts. For this book, we focus on the z/OS version.

In the z/OS environment, the ability to run any JCL in this manner would present a risk of unintentional consequences, so we instead support a JCL-like subset called *Remote Build Control Language* (BCL). *The Guide to Installing and Implementing the z/OS Extensions* contains a complete description of BCL.

In the distributed ClearCase environment, Remote Build is a standalone executable (rccbuild.exe) that is installed as part of a ClearCase fat-client install. In the simplest case, you can execute Remote Build from the command line in a ClearCase-managed directory. Remote Build manages the delivery of the source member and associated assets (build scripts, and so on) to the host. It also manages the return of the build products (listings, object, and load modules) to the repository where they are registered as derived objects.

You can use existing build scripts with BCL, but they will most likely need rework. You must expand and inline PROCs, for example, there are limits on how you perform concatenations.

Figure 4-8 is an example of a simple BCL script that does a compile and link of a COBOL module.

```
//COMP         EXEC PGM=IGYCRCTL,REGION=4M,
//                PARM=('EXIT(ADEXIT(ELAXMGUX)),
//                ADATA,LIB,TEST(NONE,SYM,SEP),
//                LIST,FLAG(I,I)')
//STEPLIB   DD DISP=SHR,DSN=DNET100.ADLAB.LOAD
//          DD DISP=SHR,DSN=SYS030.EPS.LOAD
//          DD DISP=SHR,DSN=CICSTS.V3R1.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=COBOL.V4R1.SIGYCOMP
//          DD DSN=COBOL.V3R4.SIGYCOMP,DISP=SHR
//          DD DSN=RATD4Z.V7R1.SFEKLOAD,DISP=SHR
//          DD DISP=SHR,DSN=CICSTS.V3R1.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=COBOL.V4R1.SIGYCOMP
//          DD DISP=SHR,DSN=DB2.V8R1.SDSNLOAD
//SYSLIB    DD DISP=SHR,DSN=SYS030.EPS.COBINC
//          DD DSN=CICSTS.V2R3.CICS.SDFHCOB,DISP=SHR
//          DD DSN=CICSTS.V2R3.CICS.SDFHMAC,DISP=SHR
//          DD DSN=CICSTS.V2R3.CICS.SDFHSAMP,DISP=SHR
//SYSIN DD DSN=&USR..MADRDZ.CBL(MADLO1),DISP=SHR,RCCEXT=CBL
//SYSLIN DD DSN=&USR..MADRDZ.OBJ(MADLO1),DISP=SHR,RCCEXT=OBJ
//SYSADATA DD DUMMY
//SYSPRINT DD RCCEXT=RCCOUT
//SYSUT1    DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT2    DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT3    DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT4    DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT5    DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT6    DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT7    DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSDEBUG DD DSN=&USR..MADRDZ.DBG(MADLO1),DISP=SHR,RCCEXT=DBG
//SYSXMLSD DD DUMMY
```

Figure 4-8 Example of using BCL to compile and link a COBOL program

There are more examples of BCL scripts in the *ClearCase Guide to Installing and Implementing z/OS Extensions,* which you can view at:

http://publib.boulder.ibm.com/infocenter/cchelp/v7r0m1/topic/com.ibm.rational.clearcase.help.doc/zos_extensions.pdf

In most implementations, Remote Build is almost never called directly. Depending upon what other solutions are being deployed, there are several ways to automate the invocation of Remote Build:

- ClearCase with Remote Build
- Build Forge with Remote Build
- ClearQuest with Remote Build
- Rational Developer for System z with Remote Build

We discuss each of these in following sections.

Using ClearCase with Remote Build on z/OS

ClearCase with Remote Build on z/OS is the most basic approach and is not dependent on any other solution, which is also the only means to have direct interaction between the host and a ClearCase repository, and to have ClearCase manage and record the build step.

Because this is frequently a requirement, the other solutions that we describe in the next sections usually depend on this step.

Managed builds: The simple scenario

The core of the process is a build script or a fragment of a build script that is similar to the BCL sample in Figure 4-8 on page 65. The nature of the build depends on many factors, such as the target environment, binding to a database or CICS region, and so forth. For simpler cases, you can manage the build process with variable parameters. The &USR variable in Figure 4-8 on page 65 is an example of this. You specify the value of &USR on the command line like this:

```
rccbuild -h hostname@port <rest of parameters . . > &USR=FRED
```

When this script executes, it expects to find a data set named FRED.MADRDZ.CBL, FRED.CBL.OBJ, and so forth.

Managed builds: The build engine

We came up with an approach to manage builds, where we take advantage of the meta-data management features of ClearCase to automate the management of the builds. We created a template PERL script that we adapt at each implementation. We refer to this script as a *build engine*, which is a PERL script that you run against each source element. For each member to be built, the engine reads the values of some predefined ClearCase attributes (which we create in advance) and values and uses them to determine element-level build properties, such as compiler settings and project level build properties, such as target load libraries, SYSLIB concatenations, DB2 subsystems, and so forth.

The attribute values are inserted into BCL statements, which are in turn assembled into build scripts that are submitted to the host with the ClearCase Remote Build Feature. Figure 4-9 on page 67 illustrates a high-level diagram.

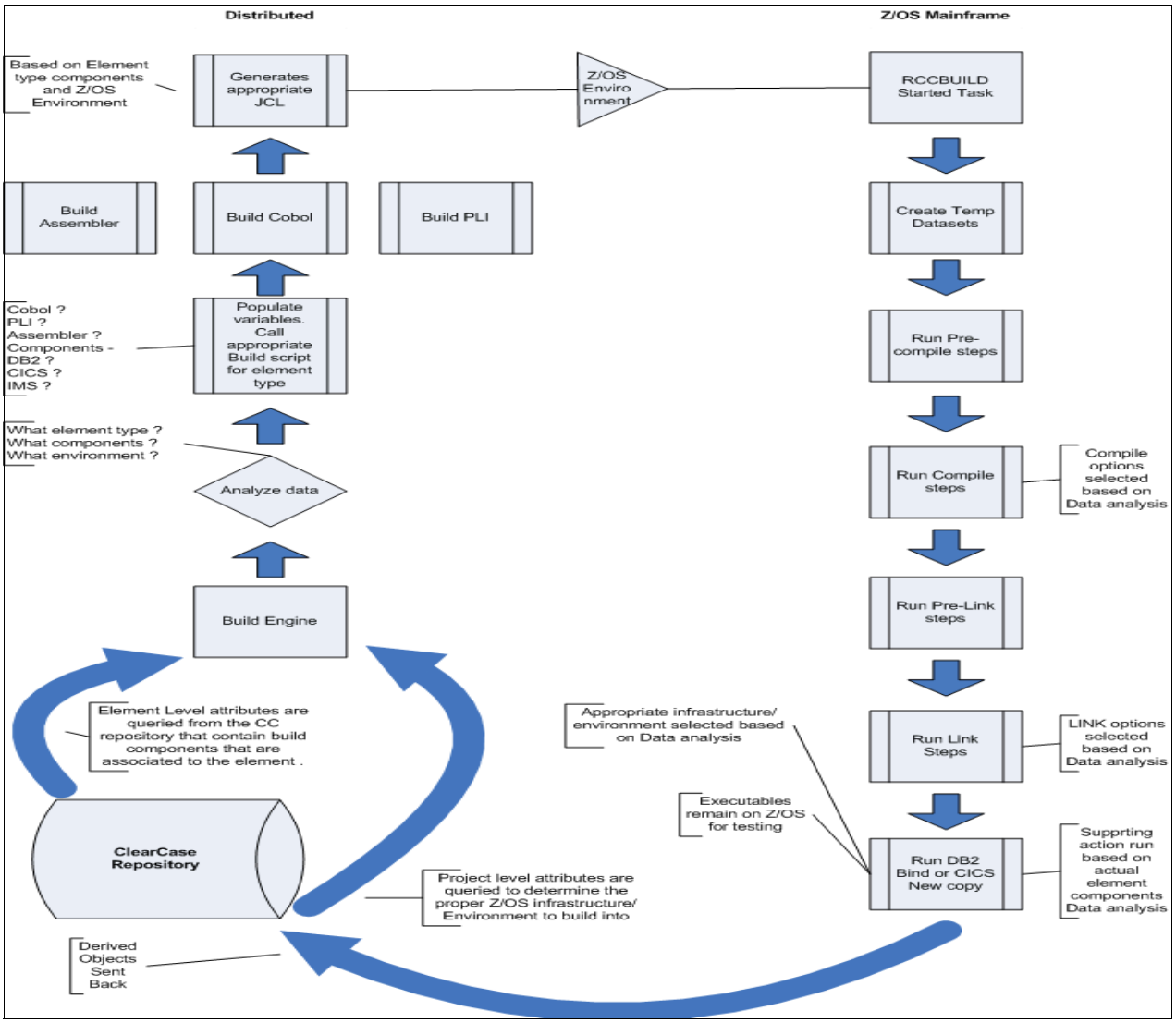


Figure 4-9 Build Engine: high-level overview

With this approach, the developers do not need to concern themselves with how to build or how to deploy. The build engine, using the information that is stored with the ClearCase element, knows this information. The script is called Host-Build.pl, which we provide an example of in Appendix A, “Source script for the build engine” on page 135.

For performance reasons, we use the ClearCase Automation Library to retrieve the project and element level attributes of the source member (ClearCase element). The code fragment in Example 4-1 illustrates this technique, where we retrieve element attributes from a ClearCase element name that is passed in by the calling program.

We could also pass the element name in from a ClearQuest hook, a ClearCase trigger, or from a customization of the ClearCase context sensitive menus.

Example 4-1 Retrieving an element attribute with the ClearCase Automation Library

```
my ($ccpn) = $ARGV[0];
my $CCApp= Win32::OLE->new ("ClearCase.Application") or die "Can't connect to
ClearCase object $!";
my $version = $CCApp->Element($ccpn) or die "Failed to retrieve element";
```

```
my $attlist = $version->Attributes;
my $attcount = $attlist->Count;
for ($i=1; $i<=$attcount; $i++) {
$myatt = $attlist->Item($i);
$myattname = $myatt->Type->Name;
$CCattr{$myattname} = $myatt->Value;
}
```

Messages are sent to STDOUT, which is echoed to a command window in the user's desktop environment. At the end of the build process, the window remains open so that the developer can review the outcome.

The full script is available as part of a field implementation kit. A complete version of it is available in Appendix A, "Source script for the build engine" on page 135.

Working with the Host-Build.pl script

You cannot just copy the build engine into your environment and run it. This version is a working prototype that runs in the IBM internal demonstration environment. We provide it as part of field service engagements to assist clients with this process. Much of the work is the task of modifying the script to your build processes and environments.

In planning the implementation of your process you should already know how you will be invoking this scrip: from a ClearQuest hook, from a ClearCase trigger, from a Build Forge step, from a Rational Developer for System z builder, or from some combination of them. You need to identify the build steps and any existing JCL that you intend to use.

Know what target environments you intend to deliver to. If you are delivering to environments that other mainframe SCM solutions manage, you need to ensure that the TSOID of the started task or (if you are running in authmode 2) of the user(s) who are running the build, has the appropriate authority to access or modify the target environments.

Know your build strategy, for example, will you build one time into an integration environment, and then promote to succeeding environments, or will you build at each promotion step?

As you define each of your build steps, or processes, you modify and extend the PERL subroutines in the sample to create subroutines with the appropriate build steps, data set definitions, and concatenations for your environment.

The build engine script is designed to be callable from a ClearCase trigger, a ClearQuest hook, or by customizing the ClearCase Explorer menu to include a "compile" or "build" option, as shown in Figure 4-10 on page 69 and Figure 4-11 on page 70.

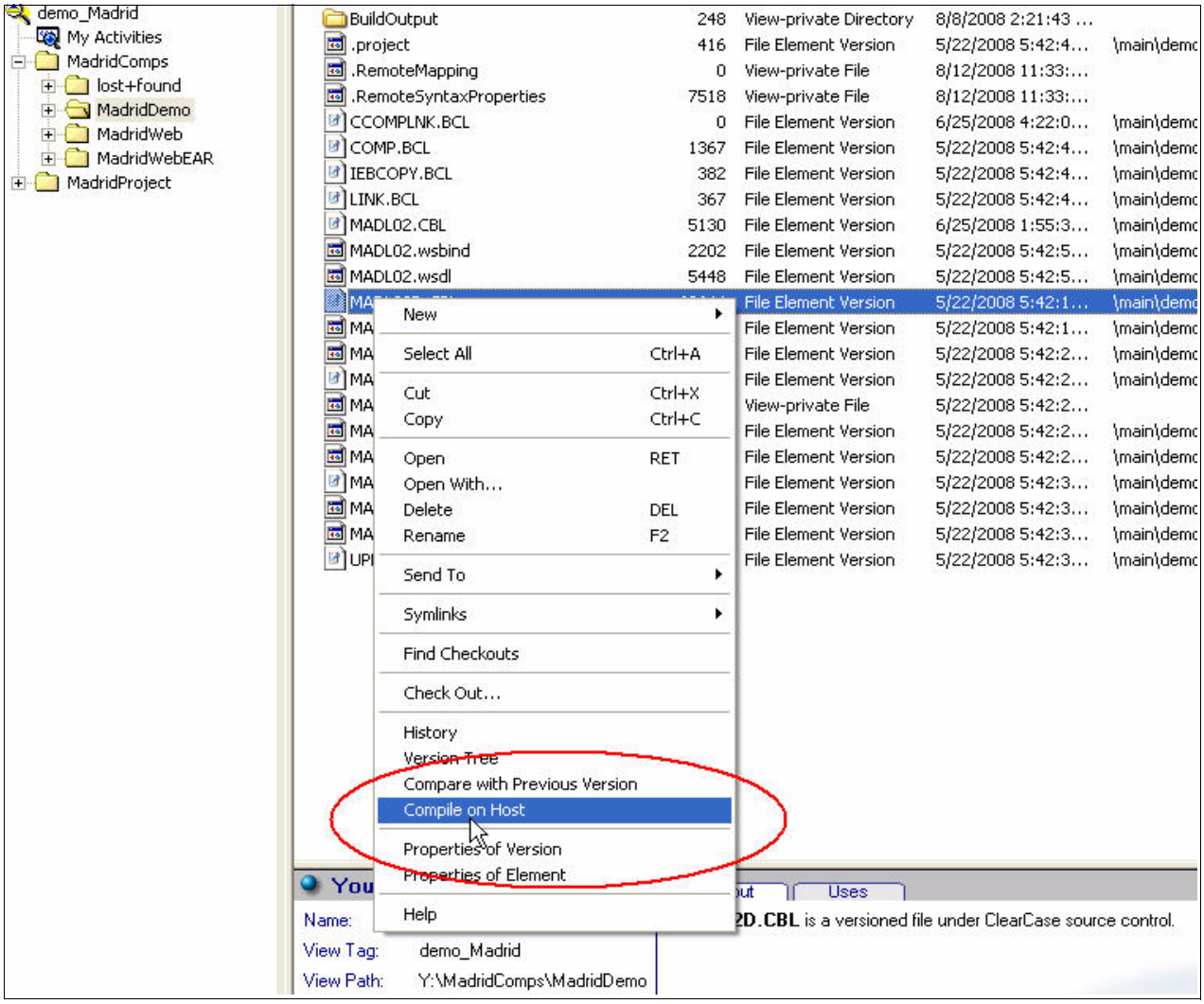


Figure 4-10 ClearCase Context menu customized to support z/OS compiles

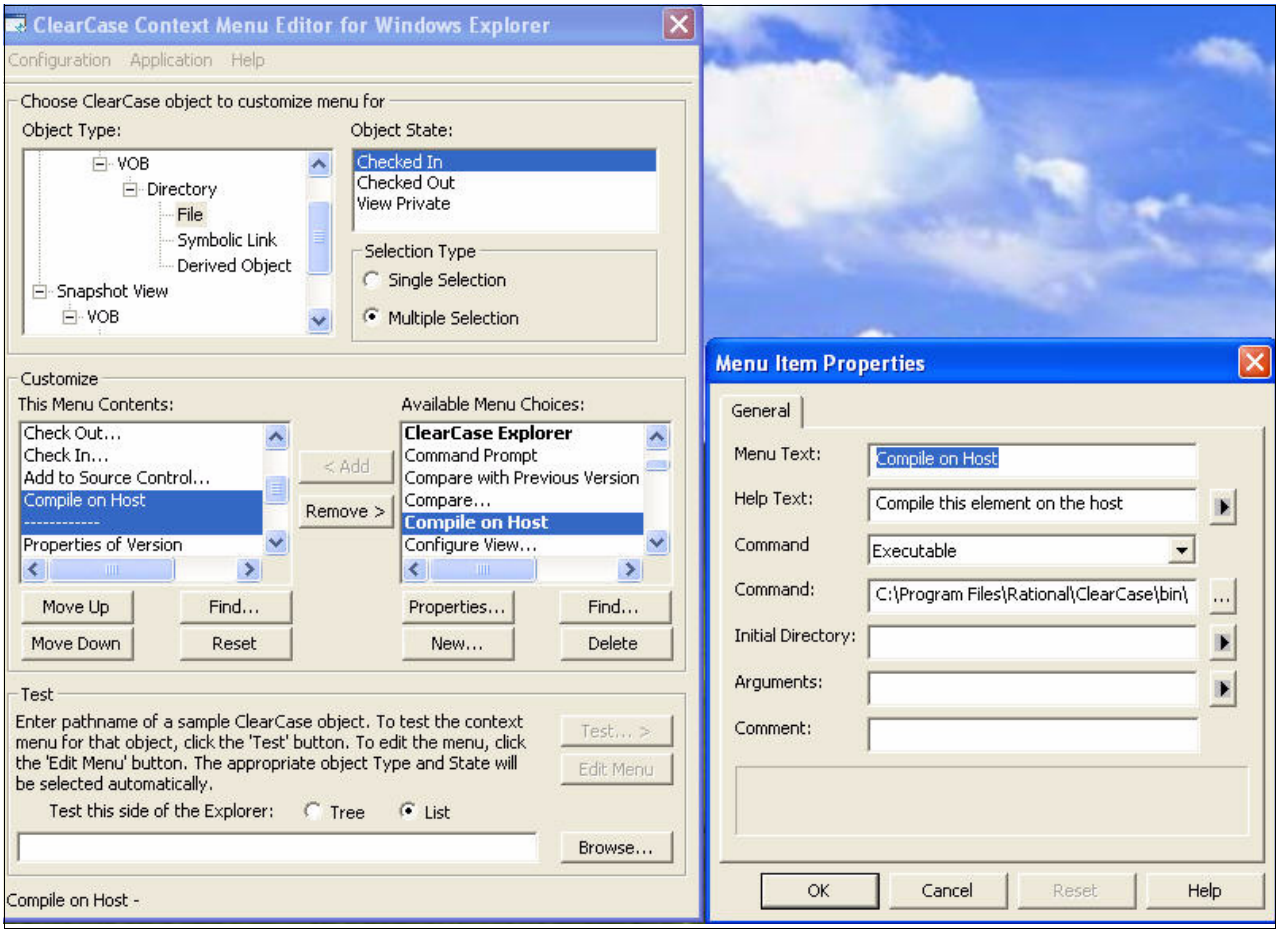


Figure 4-11 Customizing the ClearCase Context menu

This is the approach we take when there is a requirement that a user be able to perform builds of a selected member or set of members from the desktop.

Bear in mind that Remote Build is really a synchronization and transport mechanism. As such, you can extend this approach to build an entire subsystem or a cross-platform application depending on how you choose to configure your build scripts.

Using ClearCase with remote build on UNIX System Services

If configured, you invoke the UNIX System Services Remote Build server the same way that you invoke the z/OS server but the port number and possibly the host name are different.

Instead of passing BCL scripts, you pass REXX, PERL, or shell scripts that execute in the UNIX System Services shell environment.

Using Remote Build: hints, tricks, and security issues

Remote Build depends on a server task that runs on the host, which can be either a started task or a batch job. Consider the security implications when you configure the server task. This task must have authority to submit jobs and to access what ever host resources that are requested during the process.

You can run the server task in three security settings (authmodes), which are:

- 0: No user authentication. The user ID that starts the Remote Build server task on the host is used for build processes that are requested by all users.
- 1: The user ID and password, passed by the client, are optional. If supplied, they are verified with the security product (for example, RACF®, and so on).
- 2: The user ID and password, passed by the client, are required. The security product (for example, RACF, and so on) verifies them.

The user ID and password, passed by the client, are required. The security product (for example, RACF, and so on) verifies them, and an optional role can be supplied on the build request.

Carefully consider the privileges that you grant to the TSOID that runs the remote build server task. The remote build server task needs RACF authority to create some scratch PDSs in the installation area and to create members in the target PDS.

4.2.2 Using the ClearCase TSO client

ClearCase includes an ISPF client that enables the developer to work with ClearCase managed assets locally in a private PDS. The TSO client is similar in usage to the CC web client, as shown in Figure 4-12.

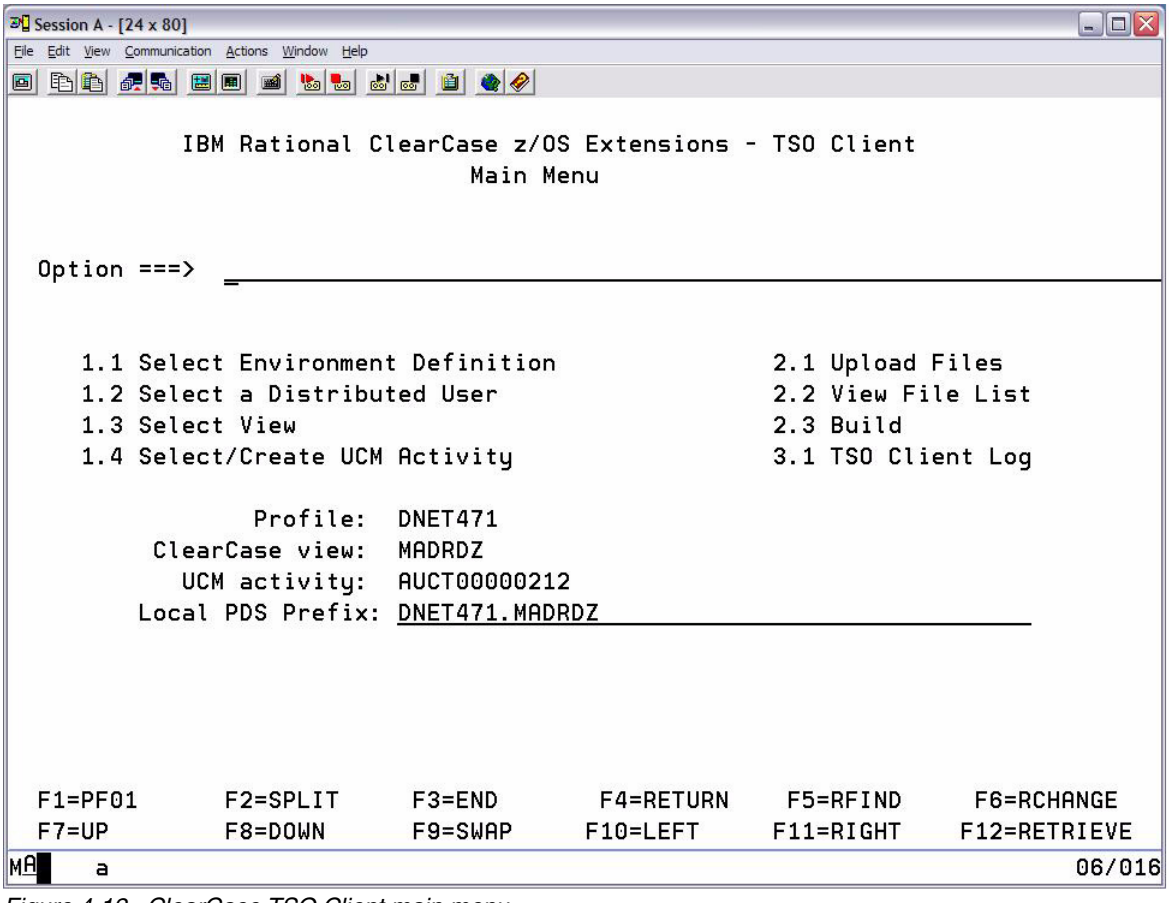


Figure 4-12 ClearCase TSO Client main menu

As with the Web client, you work with the TSO client by identifying a set of controlled members that you want to work with and uploading (or shadowing) them into a set local copy

area (A set of PDSs in this case). The versions you get are selected by the Clearcase view to which you attach your TSO client session.

The TSO client knows the state or the members that you upload. As with any other ClearCase portal, members are read-only until you check them out.

After your copy areas are populated, you perform developer level tasks, for example, check out, edit, and check in directly in the TSO client. ClearCase operations are immediately and transparently echoed back to your ClearCase view as you invoke them.

Considerations for using the TSO client

Consider the TSO client as another portal from which you can perform typical developer commands, such as adding members to source control, check out, check in, comparing different versions of a controlled element, and so forth.

You can also execute a build script, either locally (submitting a build script that resides in the local PDS on the host) or remotely (invoking remote build on the ClearCase view server to submit a build script from the ClearCase repository as though you were calling Remote Build from ClearCase). It is more likely that you will run builds from the ClearCase environment, with a combination of ClearQuest hooks, Build Forge steps, or Rational Developer for system z Builder steps.

The ability to perform administrative functions is also very limited. Your release engineering team must plan to use one of the distributed clients for administrative work.

Preparing to use the TSO client

Instructions for installing and configuring the host to use the TSO client is explained in the *Rational ClearCase Guide to Installing and Implementing z/OS Extensions* at:

http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=GI11-6714-00

After you install and configure the product, you must create the user workspaces on the host and corresponding ClearCase views. The usage model for the TSO client is similar to that for using ClearCase in general. In the same way that we recommend that you create separate views for each activity, you should maintain a separate view for exclusive users of the TSO client. Do not make any modifications to elements in this view except through the TSO Client.

After you have all of the components installed, you must make a few decisions about how you will work with TSO client. Each user must create, or have created for them, an environment definitions file like the example in Figure 4-13 on page 73.

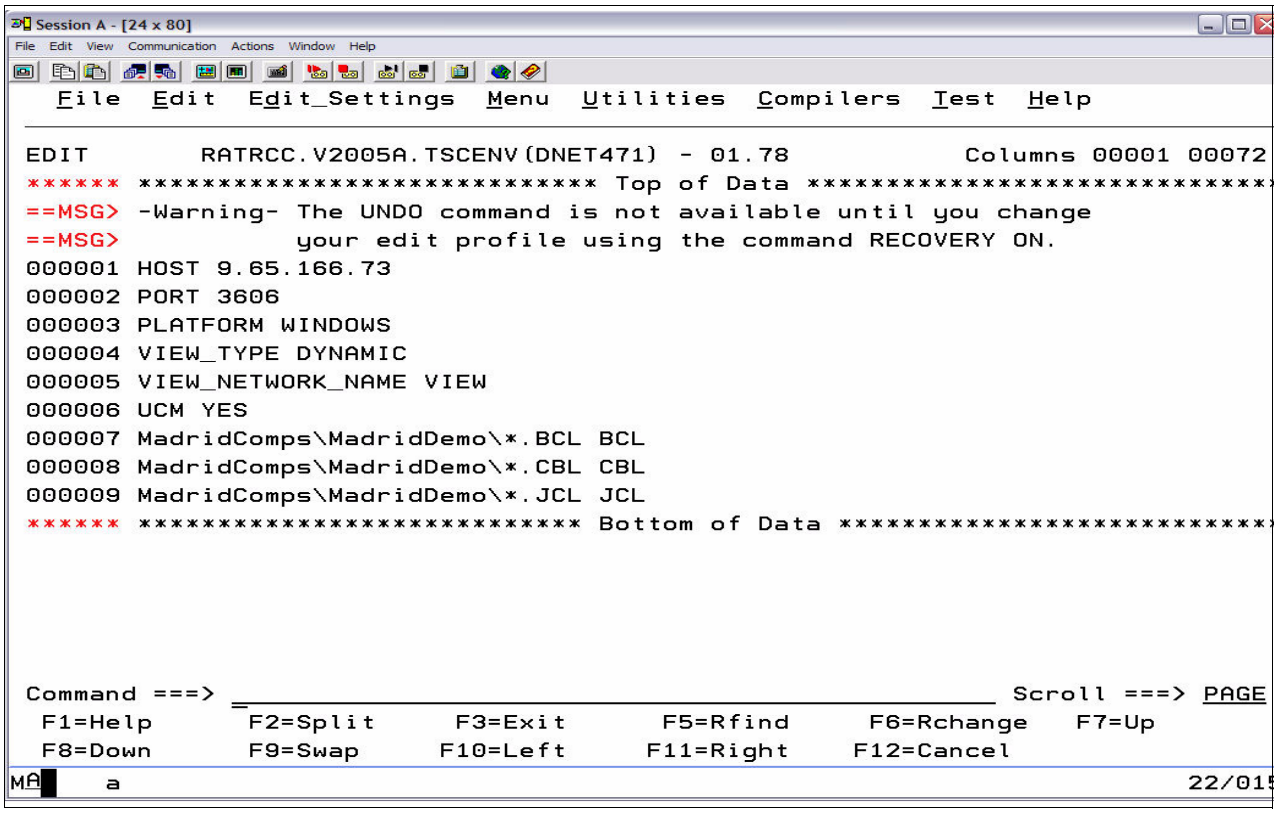


Figure 4-13 TSO client environment definitions

The fields in Figure 4-13 are explained in the z/OS Extensions Guide with the probable exception of the mapping statements (lines 7-9), so here is a quick explanation: The line contains two fields:

- The first field is the vob-extended path name of the element(s) that you are mapping. Use the file extension to determine which data set the element version is destined for. Any elements that you intend to work with using the TSO client must have file extension. Usually these are three letters and commonly understood, for example, CBL for COBOL or PLI for PL/1. You can use any extension that you want, but you must have one.
- The second field is the low-level qualifier of the data set to which the element is shadowed. The TSO client appends this field to the Local TSO Prefix, which we show in Figure 4-13. In Figure 4-13, Line 7 maps all of the elements in the ClearCase directory MadridComps\MadridDemo with an extension of BCL to a data set called DNET471.MADRDZ.BCL.

You can create the data sets in advance, or have the TSO client automatically create a PDS for you.

After you configure your TSO client environment, you can invoke the TSO client application by navigating to the installation PDS, which is usually named something, such as <HLQ>.<V200x>.TSCRX, and execute the member called RCCTSOR. Your sysprog might provide an alternate means for invoking the TSO client.

When the client starts, assuming your environment is configured correctly, you will see a window that looks like Figure 4-14 on page 74.

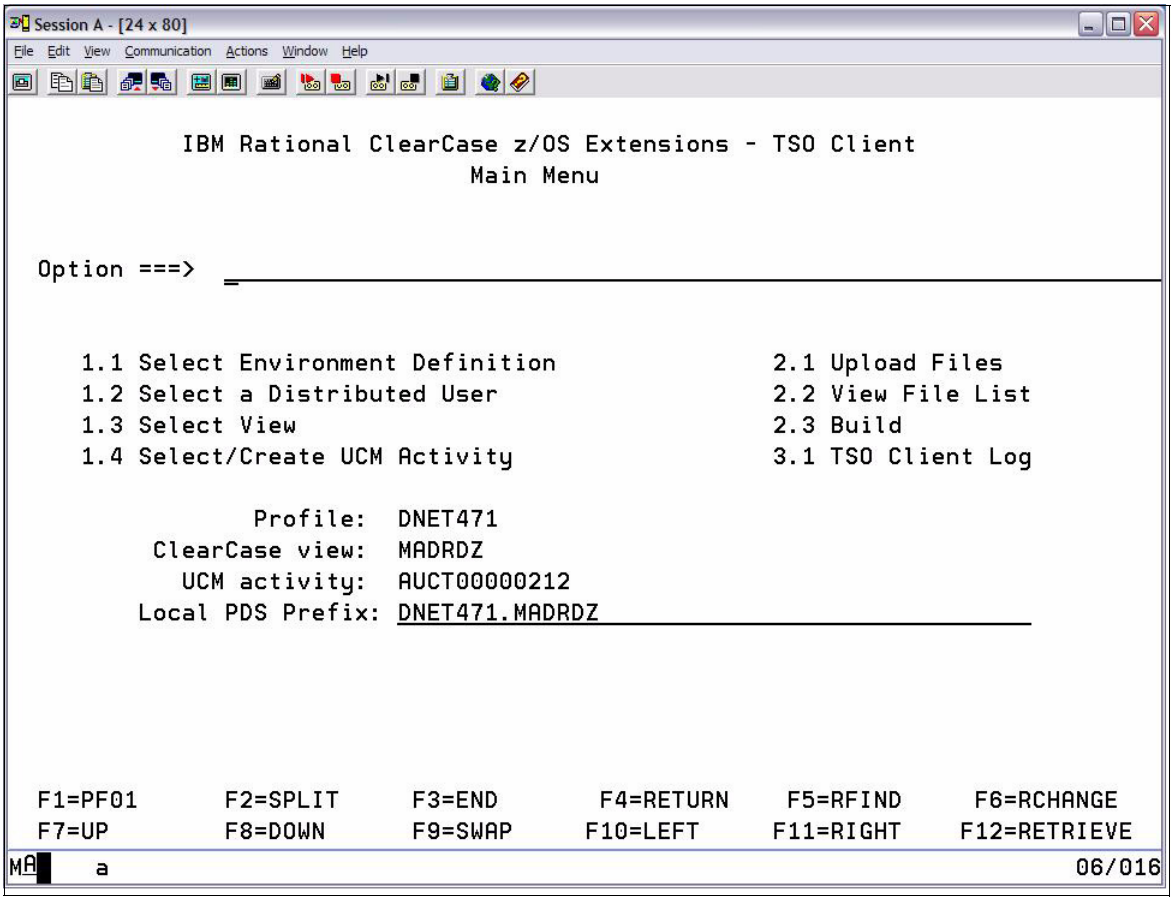


Figure 4-14 TSO Client Initial Screen

Before you can use the TSO client, you must complete the four fields in Figure 4-14: Profile, ClearCase view, UCM Activity, and Local PDS Prefix. Proceed through panels 1.1 through to 1.4 before you can begin work. In this example, because we set “UCM” to “YES” in the environment definitions file in figure Figure 4-13 on page 73, panel 1.4 requires us to select an activity. You can work in base ClearCase views by setting “UCM” to “NO”, and you will not be prompted for activity information.

The TSO client remembers all of this information, so it is not necessary to go through 1.1 to 1.4 at each invocation.

Most of your work is off of panels 2.2. To begin work with ClearCase controlled members, from panel 2.2, you are presented with a window to select the data set that you want to work in. Choose the data set, and the window is displayed, as shown in Figure 4-15 on page 75. In the next panel, the name of the controlled member and a set of options and commands that look familiar to an experienced Clearcase user, but may require some explanation to an ISPF user seeing ClearCase for the first time, are displayed. Of most significance are the CO and CI actions. You also use this panel to upload (for example, refreshing your data set version of the member with the version stored in the repository), to view the version history, and to compare differences.

Obviously, ISPF being what it is, you get a text-only version of the version tree and different output.

Nonetheless, the TSO client enables those who are most comfortable in a green window setting to use the same solution and repository as the rest of the team to perform normal day-to-day SCM functions.

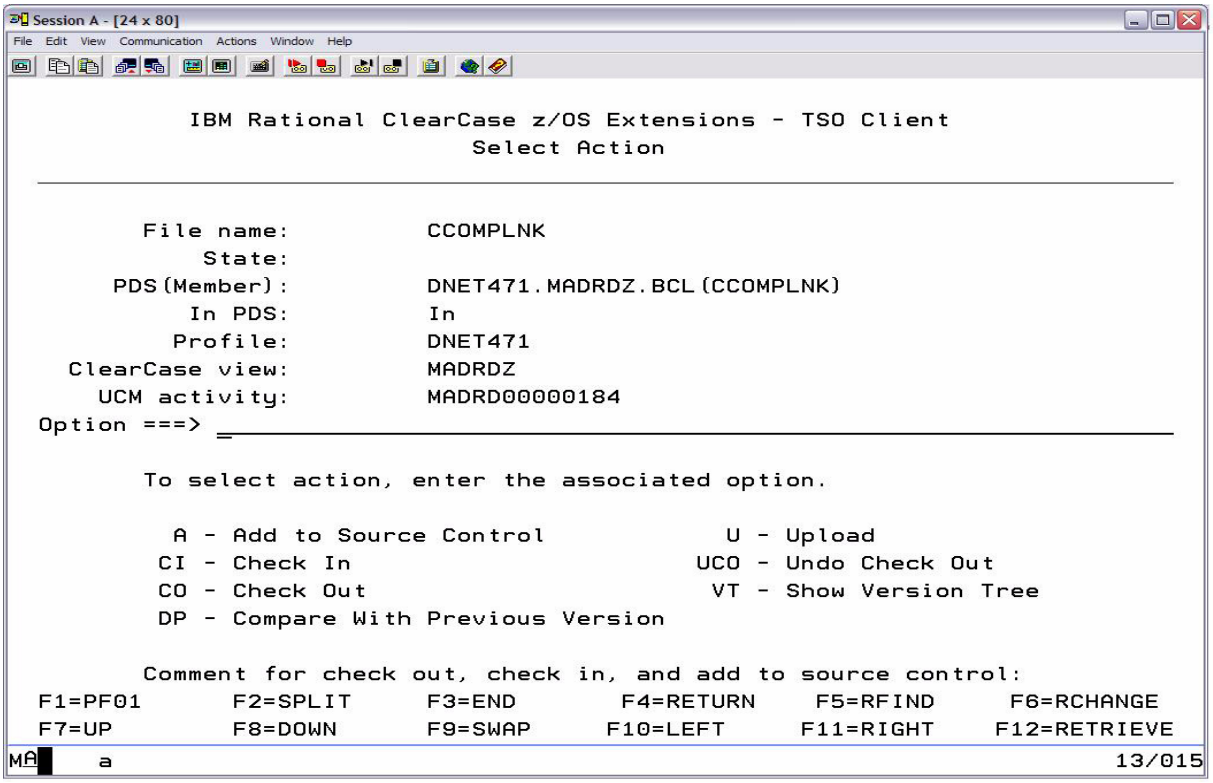


Figure 4-15 Green panel

4.2.3 Programmatic access to the ClearCase repository from z/OS

You do not need to use the TSO client to work with ClearCase-managed data from the host because there is a published Application Programming Interface (API). The API is described in the *Guide to Installing and Implementing the z/OS Extensions*.

Although it was not in the scope of the project for this book, we have a large financial customer who, for audit reasons, had a requirement for an off-host means of shadowing their z/OS production assets. They implemented this by inserting calls to the ClearCase TSO Client API into the windows of their existing host-based SCM solution. As their developers edit and deliver their z/OS assets, calls to the TSO-client API echo analogous commands to ClearCase and perform checkouts, additions to source control, check ins, and so on, which transparently creates and maintains a shadow ClearCase repository and helps them to close out a significant audit issue.

You can perform many of the typical user-level ClearCase commands on the distributed system, such as uploading and downloading members, adding to source control, check out and check in, and so forth. The complete set is described in detail the *Guide to Installing and Implementing the z/OS Extensions*.

Like the TSO client itself, consider the TSO-client API as another portal for performing typical developer functions.

If you intend to use the TSO-client API, there are some necessary preparatory steps.

TSO client: Hints and tricks

In this section, we discuss come TSO client hints and tricks.

Connection Issues

Communications between the TSO client and the ClearCase view server that runs on the distributed system is over secure POSIX pipes and sockets. Remote Build uses the same mechanism. In fact they are essentially two portals at either end of a “tunnel”.

For security reasons, the TSO client performs a series of tests to ensure that it is communicating with the intended ClearCase environment. The tests are fairly stringent, with the result that at times is difficult to establish a connection between the host and the distributed system.

Both the host and the ClearCase view server need to be known to each other, both by IP address and by host name. When you start the TSO client up, it pings the host name or IP address that is supplied in the host parameter in the environment definition file, as shown in the example on Figure 4-13 on page 73. For the TSO client to work, your distributed system must be “known” to the host by whatever name resolution service is employed, at the very least in the DS member TCP/IP.HOSTS.LOCAL or with what ever name resolution service is running.

It is likely that you might see the following message in the process if you encounter connection problems:

```
Error contacting TSO client server. See the log for more details.
```

Access the TSO client log from panel 3.1 off of the main TSO client menu. The latest entries are appended to the end of the file; therefore, you need to F8 to the end. Most of the traffic between the TSO client and the distributed system is logged here.

When you open the log, you will see a message similar to Example 4-2.

Example 4-2 Log message

```
The host name, <hostname>, for the RCCBLDS remote host
cannot be resolved.

 Recovery:

   - Verify that the remote host server's name and
     address are included in the current workstation's
     tcp/ip hosts file or the data files of the name server.
     If the host name and address appear in either of these
     files, the network or name server may be experiencing a
     temporary problem.

+++++End execution of testServer+++++
```

This error message means that the TSO client could not connect to the distributed server where the ClearCase view server is running, which is probably because either the IP address or the distributed host name were not known to the host. You can troubleshoot this error by issuing ping commands from both machines. Be sure to use both the IP address and the host name that you entered. For the TSO client to function, both the IP address and the host name of the view server and the z/OS host have to be known to each other.

Synchronization issues and recovery from errors

If you abnormally exit out of the TSO client, such as cancelling your session instead of exiting with the F3 key, it is possible that the PDS, which the TSO client is shadowing your managed files in, can get out of sync with the ClearCase view.

Manually removing the TSO client cache often fixes the out of sync problem. The cache is located in the installation PDS, which is in the TSCUSER directory. If you do not have authority to do this yourself, ask your sysprog to remove the file TSCUSER.<yourTSOIC>.

You can also safely remove the following members in your private area:

- <username>.TSC.CMD
- <username>.TSC.JCL
- <username>.TSC.LOG

These steps should clear up any residual issues from an improper exit from the TSO client. In the worst case, and as a last step, manually clear and recreate your local copy areas.

4.2.4 Using ClearQuest to manage the application life cycle

We often get requests to assist customers with implementing a life cycle management solution. The approach we take involves incorporating a build solution in an overall life cycle that ClearQuest manages.

If you plan to implement an activity-based application life cycle, you need to include ClearQuest in the solution. In an activity-based life cycle, the goal is to ensure that all work and modifications take place in the context of a defined activity, such as a change or enhancement request, production issue, or a defect. The *ClearQuest Unified Change Management schema* (UCM) implements activity-based life cycle management for software development.

You can implement an Enterprise SCM Life cycle in many ways, depending on your requirements.

We found that the most expedient approach is to begin with the UCM schema and augment it with a combination of the BuildTracker package or the CrossPlatform SCM package. To ensure adequate security and control over the turnover process, we strongly recommend that you incorporate the ClearQuest Electronic Signature package.

With Release 7.1 of ClearQuest, the *ClearQuest Application Lifecycle Management* (CQALM) package became available. The CQALM package is intended for comprehensive enterprise-level automation of an application life cycle. It is a significant transition from a software development-focused issue tracking solution to an enterprise-level workflow management solution.

There is a lot of documentation on CQALM that is available. There is a full discussion of CQALM on IBM DeveloperWorks at:

http://www.ibm.com/developerworks/rational/library/edge/08/mar08/pampino-pierce/index.html?S_TACT=105AGX15&S_CMP=EDU

We used the CQALM schema in the project for this book, which is illustrated in Figure 4-16 on page 78 with a shot of the ALM activity record that we created. We used ALM activities as the work items.

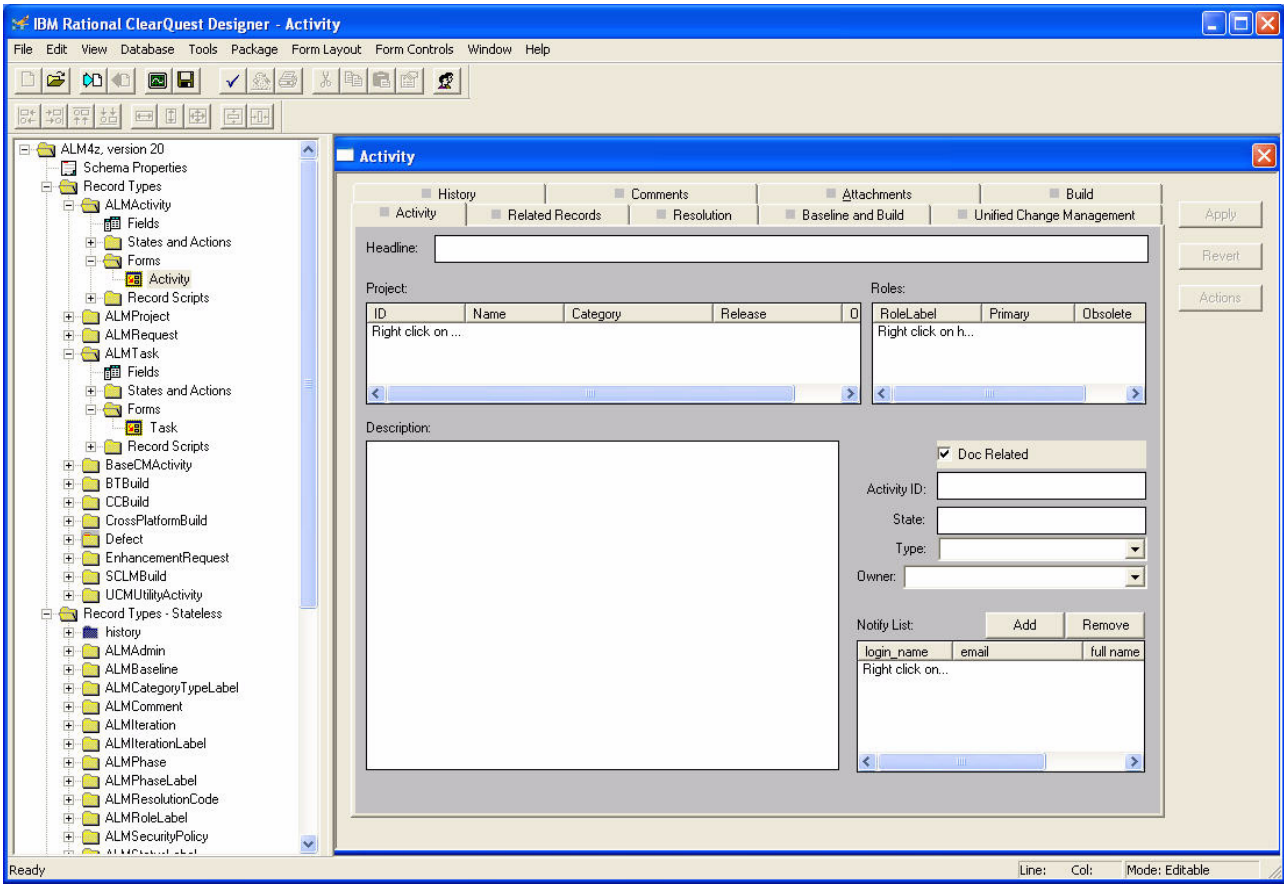


Figure 4-16 The CQALM schema and activity record

It took some effort to successfully adopt and implement a CQALM-based life cycle. You should plan for this, but the resulting benefit is a robust Enterprise SCM environment that is worth the effort. Review these sources, and decide whether the CQALM schema is appropriate for your organization.

You can download the CQALM Packages for ClearQuest from:

`http://www.ibm.com/services/forms/preLogin.do?lang=en_US&source=swg-ratcq`

There is no charge for the packages, but you must register to download them. The download includes the ALM Packages for ClearQuest, a sample database, and three tutorials to use with the sample.

Regardless of how comprehensive (or how simple) your ClearQuest life cycle is, if you want to incorporate an automated build as part of the life cycle, you can attach a hook to the “complete” action of the BTBuild record, to the “submit” action of a CrossPlatform Build request, or to whatever action and record type you identify for managing builds.

In one implementation, we created ClearQuest hook code, which uses the ClearQuest PERL API (CQPERL) to retrieve the changeset from ClearCase. It then iterates through the returned list and passes the version-extended path name to Host-Build.pl, which is illustrated in Example 4-3 on page 79.

This hook can also call Remote Build directly.

Example 4-3 ClearQuest hook code

```
#-----------------------------------------------------
# Get a ClearQuest Query Def object
#    We will query on All UCM Activities
#-----------------------------------------------------
my ($QueryDef) = $CQsession->BuildQuery("All_UCM_Activities");
#-------------------------------------------------------------
# Create the ClearQuest result set and execute
#-------------------------------------------------------------
my ($ResultSet) = $CQsession->BuildResultSet($QueryDef);
$ResultSet->Execute();
#-------------------------------------------------------------------
# loop through the CQ Query results.  Use an initial MoveNext to
#    get first record
#-------------------------------------------------------------------
$xlix = 1;
$records = 0;
$status = $ResultSet->MoveNext();
while ( $status == $AD_SUCCESS ) {
$records++;
$xlix++;

#-------------------------------------------------------------------
# Get the field values from the returned query result
#-------------------------------------------------------------------
$id = $ResultSet->GetColumnValue(1);
$state = $ResultSet->GetColumnValue(2);
$headline = $ResultSet->GetColumnValue(3);
$owner = $ResultSet->GetColumnValue(4);
write();
}
#-------------------------------------------------------------------
# Get the CC activity ID from the CQ "ucm_vob_object" field
#-------------------------------------------------------------------
$activity_id = $ResultSet->GetColumnValue(5);
#-------------------------------------------------------------------
# If the activity ID from CQ is null, there is no
#    change set info
#-------------------------------------------------------------------
if (! $activity_id ) {
print "     No ClearCase Activity info yet\n";
}
} else {
#-------------------------------------------------------------------
# If the CC activity ID is not null, then we need to use CAL
#    to extract the changeset info from the CC activity object
#-------------------------------------------------------------------
my ($CCApp) = Win32::OLE->new ("ClearCase.Application") or
            die "Can't create ClearCase application object via call to
Win32::OLE->new(): $!";
print "Processing Activity info\n" if $DEBUG;
#-------------------------------------------------------------
# Get an activity object from CAL
#-------------------------------------------------------------
      $myactivity = $CCApp->Activity($activity_id);
```

```
if (! $myactivity ) {
print "Can not resolve activity info in ClearCase\n";} else {
      $view = $myactivity->NameResolverView;
#-------------------------------------------------------------
# Get the activity's change set, which is a CCVersions collection.
#   Use the activity's "nameresolver view" for name resolution.
#-------------------------------------------------------------
      $ChangeSet = $myactivity->ChangeSet($view, "False");
      $CS_Entries = $ChangeSet->Count;
#-------------------------------------------------------------
# Loop through the CCVersions collection, collecting the names of
#     the versions for printing.
#-------------------------------------------------------------
$CS_Index = 1;
      while ($CS_Index <= $CS_Entries) {
         $Version = $ChangeSet->Item($CS_Index);
         $VersionPN = $Version->ExtendedPath;
            if ("$CS_Index" eq "1") {
               $cs_list = $VersionPN;
            } else {
               $cs_list = $cs_list . "\n" . $VersionPN;
            }
            $CS_Index++;
         }
      $status = $ResultSet->MoveNext();
```

The variable $VersionPN is what passed to Host-Buld.pl or to Remote Build, and should be the Clearcase version-extended path name of the element to be built.

There is a Developer works article that can guide you through using the ClearQuest PERL API at:

http://www.ibm.com/developerworks/rational/library/4702.html?S_TACT=105AGX15&S_CMP=EDU

4.3 Incorporating Build Forge with Remote Build

Build Forge automates many of the key steps of building an Enterprise application. If Build Forge is in your plans, you are probably considering how to use Build Forge to build the mainframe components of your applications. We used Build Forge as part of this project.

As with ClearQuest, your first decision is whether Build Forge fits into your strategy. You can implement a robust cross-platform build process without Build Forge, but like any tool, you must decide whether a home-grown solution can deliver the benefits, less expensively. We found that Build Forge is very easy to implement, with a few caveats.

If you are not already familiar with Build Forge, read this brief white paper that will get you started:

ftp://ftp.software.ibm.com/software/rational/web/whitepapers/RAW14005-USEN-00.pdf

4.3.1 Build Forge

There are two ways to build mainframe assets with BuildForge: use the BuildForge z/OS agent or use BuildForge to call ClearCase Remote Build.

Using the z/OS Build Forge agent

The advantage of the z/OS agent is that BuildForge manages the creation and submission of the z/OS builds.

The following white paper describes how to configure and use the Build Forge z/OS agent:

http://download.boulder.ibm.com/ibmdl/pub/software/dw/rational/pdf/BuildForge-SystemzUseCases.pdf

The z/OS agent runs in UNIX System Services, and you use REXX scripts in the HFS to run your build jobs.

Figure 4-17 is an example of a Build Forge step that runs a REXX script.

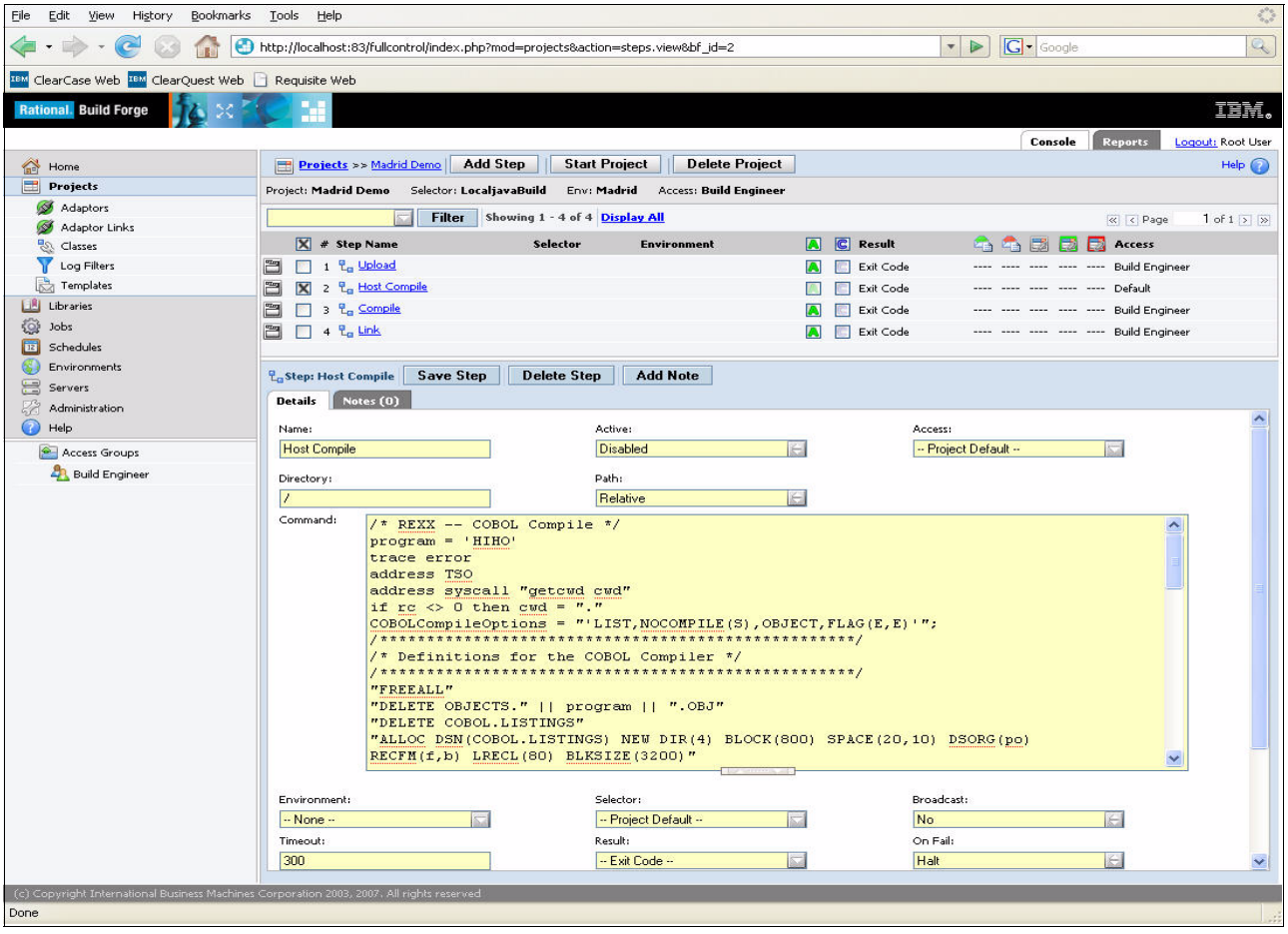


Figure 4-17 Using the Build Forge z/OS agent to run a host build from UNIX System Services

The agent can only work with artifacts that are already in a PDS or in an HFS. There is no means of transporting artifacts between the host and distributed environments. If you want to use the z/OS agent to build ClearCase managed artifacts, you need to transport them there by other means. One way that you can do this is to create an “UPLOAD” step in your Build Forge project, as shown in Figure 4-18 on page 82.

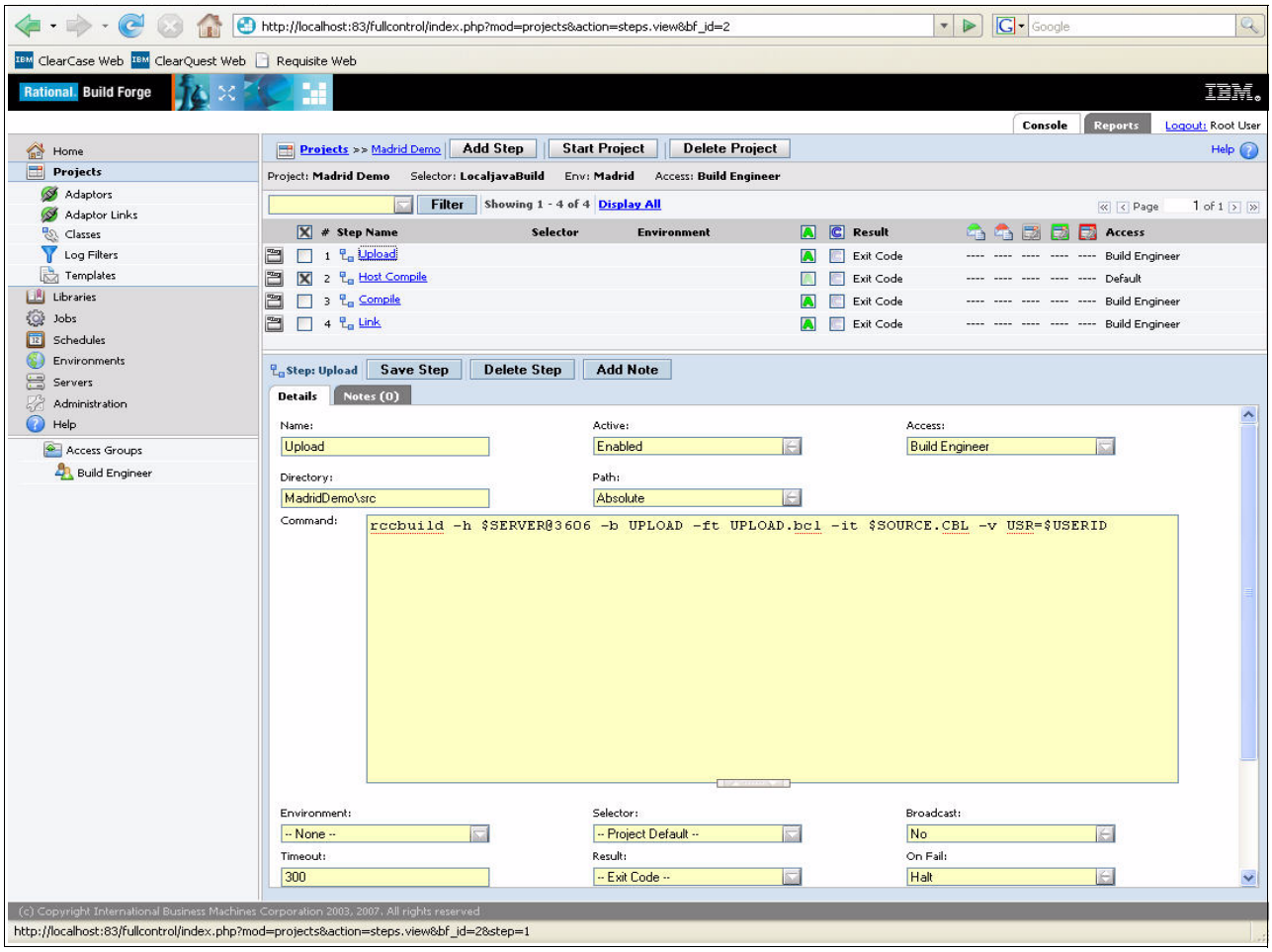


Figure 4-18 A Sample Build Forge step: uploading elements with ClearCase Remote Build

The step in Figure 4-18 calls a simple BCL script that ships the elements to the designated PDS on the host:

```
//UPLOAD EXEC PGM=IEFBR14,REGION=4M
//CBLSRC DD DSN=&USR..MADRDZ.CBL,DISP=SHR,RCCEXT=CBL
```

The next step is run on the host and contains the REXX steps to submit the compile job. In using the z/OS agent, the compile output and listings remain on the host by default. You must create an extra step to return them to the ClearCase repository.

ClearCase build auditing features cannot capture builds that you run with the Build Forge z Agent.

Building host applications with a Build Forge local agent

In our sample implementation, we tried an alternative approach. Instead of using the Build Forge z agent to submit host builds, we instead configured a local (desktop) Build Forge agent to use the ClearCase Remote Build feature to run the build job. We did this just by creating a different compile step, which like the upload step in Figure 4-18, calls ClearCase Remote Build to perform the compile, as shown in Figure 4-19 on page 83.

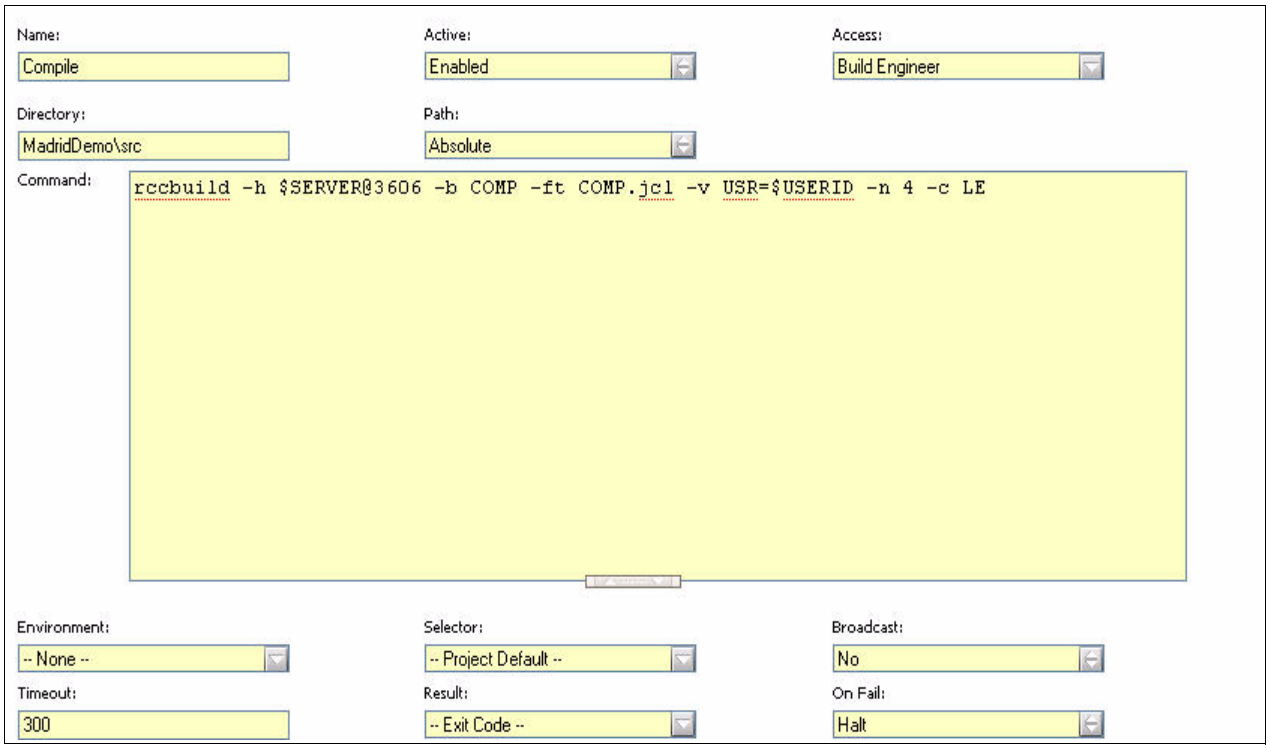


Figure 4-19 Build Forge step: invoking ClearCase Remote Build

If you choose to work this way, you do not need a separate upload step because Remote Build manages the transport of the files between the host and the distributed side.

4.4 Enterprise application life cycle management with Rational Developer for System z

In the previous sections, we showed you how to use combinations of ClearCase, ClearQuest, and Build Forge to build a mixed-workload application. In this section, we show you how we configure Rational Developer for system z to manage these steps as part of an Application Lifecycle Management.

Rational Developer for System z (in its original incarnation as WebSphere Studio Enterprise Developer) was created to equip the mainframe developer with a broader set of tools, a modern GUI, and the ability to off load some development tasks, such as syntax checks, to a less expensive environment.

When we started this book, our intent was to define the solution, or set of solutions, necessary to implement an Enterprise SCM. Our original assumption was that Rational ClearQuest would act as the ALMz “cockpit” or platform.

After some effort, it became clear that although it is entirely possible to construct a robust Application Lifecycle using ClearCase, ClearQuest, and Build Forge, what was needed was a means to more closely couple the host and distributed environments. So while we started the expectation of using Rational Developer for System z as the IDE, we found that although not mandatory, using Rational Developer for System z as the “control center” enabled us to build a more flexible and robust Application Lifecycle Management solution with less effort. There is substantial documentation, training and tutorials, to get you up to speed with Rational

Developer for system z. In this section, we simply cover those features of Rational Developer for System z that are relevant to Enterprise Software Configuration Management.

In this section, we assume that you are familiar with Rational Developer for System z. If you need to familiarize yourself with Rational Developer for System z, we suggest you start with the following Web page:

http://www.ibm.com/developerworks/rational/products/rdz/

4.4.1 Rational Developer for System z: A very brief overview

Rational Developer for System z is an Eclipse-based IDE for developing enterprise applications. It is a super-set of Rational Application Developer that offers all of the same benefits and additional features to support the development, debugging, and deploying of COBOL, PL/1, C, and C++ Web applications that are deployed to distributed and mainframe host environments. In addition to being a fully featured Java development environment, Rational Developer for System z includes a broad set of utilities to simplify and automate many of the tasks of an ISPF developer, and it enables off loading of some non-production tasks that are traditionally host-based, such as syntax checks, onto your desktop.

With Rational Developer for System z, you can browse a host PDS in the same way that you can browse a local desktop file system. You can drag and drop files between the two environments, and you can configure Rational Developer for System z to automatically synchronize the contents of your local and remote file systems.

For this book, we focus only on those features of Rational Developer for System z that are directly relevant to Enterprise SCM, although there is obviously much more you can do with it.

We focus on two primary activities: Working with ClearCase-managed artifacts and configuring and initiating cross-platform builds.

4.4.2 Working with ClearCase-managed artifacts

While we mostly use z/OS project's perspective in producing this book, you can use any perspective that you want to use. We found that the default window configuration in the z/OS project's perspective was the best suited to ESCM activities. In particular, with the z/OS project's window, you can work directly with ClearCase-managed artifacts by mapping your project directly to your ClearCase view, which we discuss in an upcoming section.

You can access ClearCase-controlled artifacts in two ways: through traditional ClearCase views (dynamic and snapshot) or through ClearCase Eclipse views.

You can use any kind of ClearCase view with Rational Developer for System z, which you determine by enabling either the ClearCase SCM adaptor or the Eclipse plug-in. Figure 4-20 on page 85 shows the steps for activating the ClearCase view.

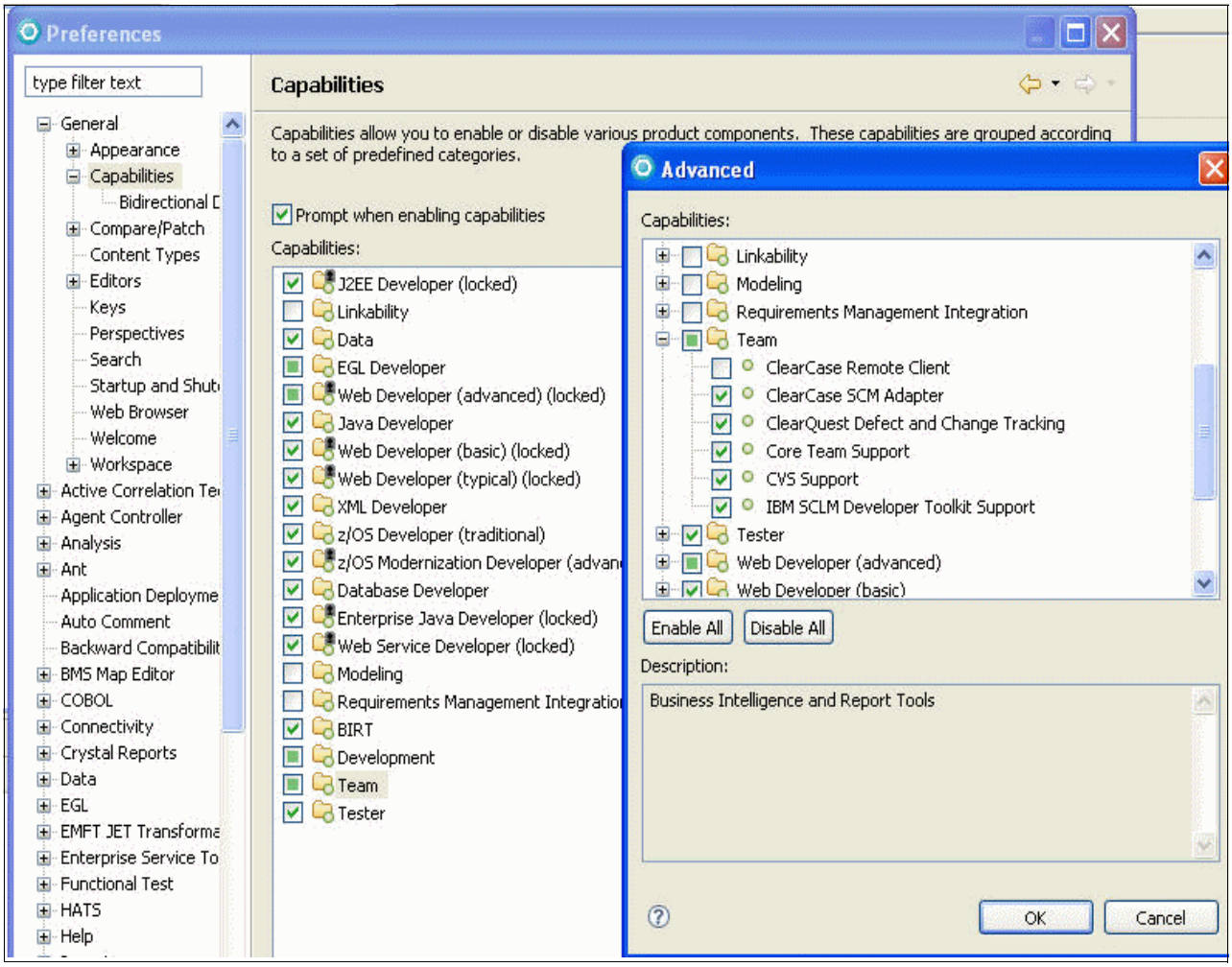


Figure 4-20 Activating the ClearCase SCM adaptor

The SCM adaptor is included with Rational Developer for System z. It assumes the availability of local ClearCase services, so you must have ClearCase locally installed on your machine to use it.

In Figure 4-20, we enabled the SCM adaptor by:

1. Select Window → General → Preferences → Capabilities, and then select **Team**.
2. In the lower-right corner of the Preferences window, click the **Advanced** button. The Advanced is displayed.
3. In the Advanced box, select **Team**. You will receive the choice of activating the ClearCase Remote Client (the Eclipse plug-in) or the ClearCase SCM adaptor.

If you are already running with the ClearCase fat client, using the SCM adaptor gives you the benefit of being able to run with ClearCase dynamic views inside of Rational Developer for System z. If you do not use the ClearCase fat client, use the ClearCase Remote Client (CCRC) instead.

The most significant difference between using the SCM adaptor and the CCRC is whether to load and synchronize your Web views (which is unnecessary with dynamic views). The rest of this section is equally applicable to either approach.

Important: You must be running Rational Developer for System z 7.1.1.2 and ClearCase 7.01 or later to use the SCM adaptor. If you are running earlier versions of either tool, use the Eclipse client.

4.4.3 Connecting to your repository

The usage model for Eclipse is based on the idea of a local workspace. When a developer starts work on an activity, they populate the workspace by copying (or checking out) a set of controlled artifacts from an external SCM repository, such as SubVersion or CVS, and then they copy them back in (or check them in).

One of the original strong selling points of ClearCase is that with ClearCase, the tedious task of manually copying artifacts back and forth between the repository and the work area is no longer necessary. ClearCase instead presents the developer with a “virtual” file system that projects a set of versions of managed artifacts as a native (Windows or Linux/UNIX) file system that is entirely private to the developer. With many other SCM systems, the rationale for a sandbox, or work area, is to enable developers to work in parallel and in isolation to avoid overwriting one another’s changes. With ClearCase, developers isolate themselves with these private file systems, which are referred to as “views”.

As mentioned above, this approach is the same whether you use local views and the ClearCase SCM Adaptor, or if you use the CCRC and an Eclipse view.

To work with web or Eclipse views, you use the ClearCase Eclipse plug-in. If you enable the Eclipse plug-in, you can work with your assets with the ClearCase Eclipse perspective, in addition to using the File menu in the z/OS projects view.

When a developer begins to work on an activity, they usually start by creating a new ClearCase view. The view appears as a private workspace to the developer, without requiring the developer to physically copy the artifacts to a different location. What this means with Rational Developer for System z (or any Eclipse-based solution) is that when you create or import a project, you do not need to physically copy the artifacts into your workspace. In fact, you should not; instead, you just map your workspace to your view drive or directory, as shown Figure 4-21 on page 87. If you do copy the artifacts into the workspace, you circumvent the value of ClearCase because if you copy the artifacts out of ClearCase, you are creating non-managed versions of them that at some point need to be reconciled with the repository version.

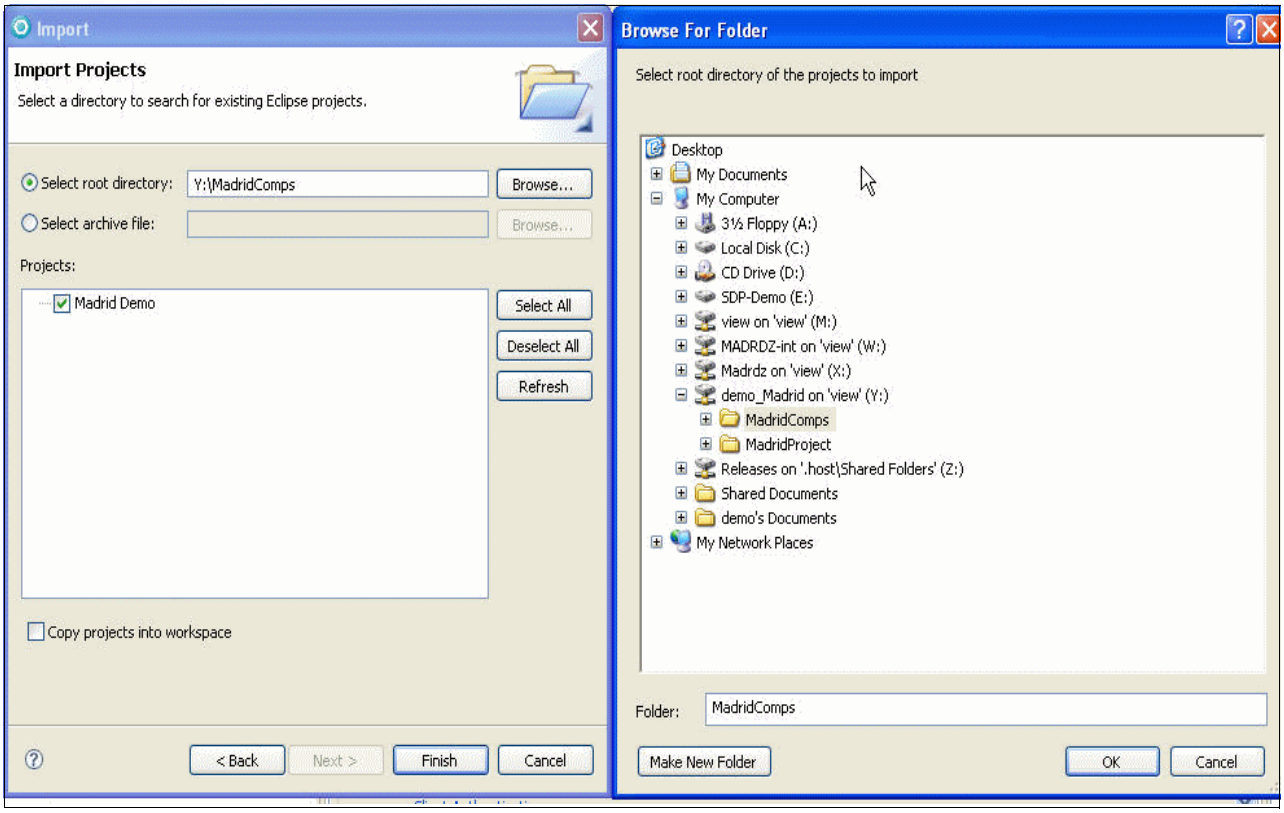


Figure 4-21 Importing a ClearCase-controlled Rational Developer for System z project

4.4.4 Working with controlled elements

There is no ClearCase perspective with the SCM adaptor. You access ClearCase operations by right-clicking the element or by customizing your perspective to add ClearCase buttons to your toolbar, as shown in Figure 4-22.

Figure 4-22 The ClearCase Eclipse toolbar

In Eclipse projects, certain files and directories need to be writable. You cannot work with the project until you, at the very least, make all of the directories writable. The project metadata files (.project and so on) need to be writable too, which you can do either by making them controlled elements and keeping them checked out or by creating them as view-private files.

After you import your project, the fact that you are working with ClearCase managed files is, or should be, transparent. Aside from having to check out elements to modify them, you should be able to build, test, and deploy in the same manner that you can with a native file system. What is different is that you do not need to copy assets in and out of a source repository to work with them. The only time that you might need to physically copy files around is to deploy them to test or production environments.

File-level ClearCase operations are displayed when you right-click the element, and select **Team**, as shown in Figure 4-23 on page 88.

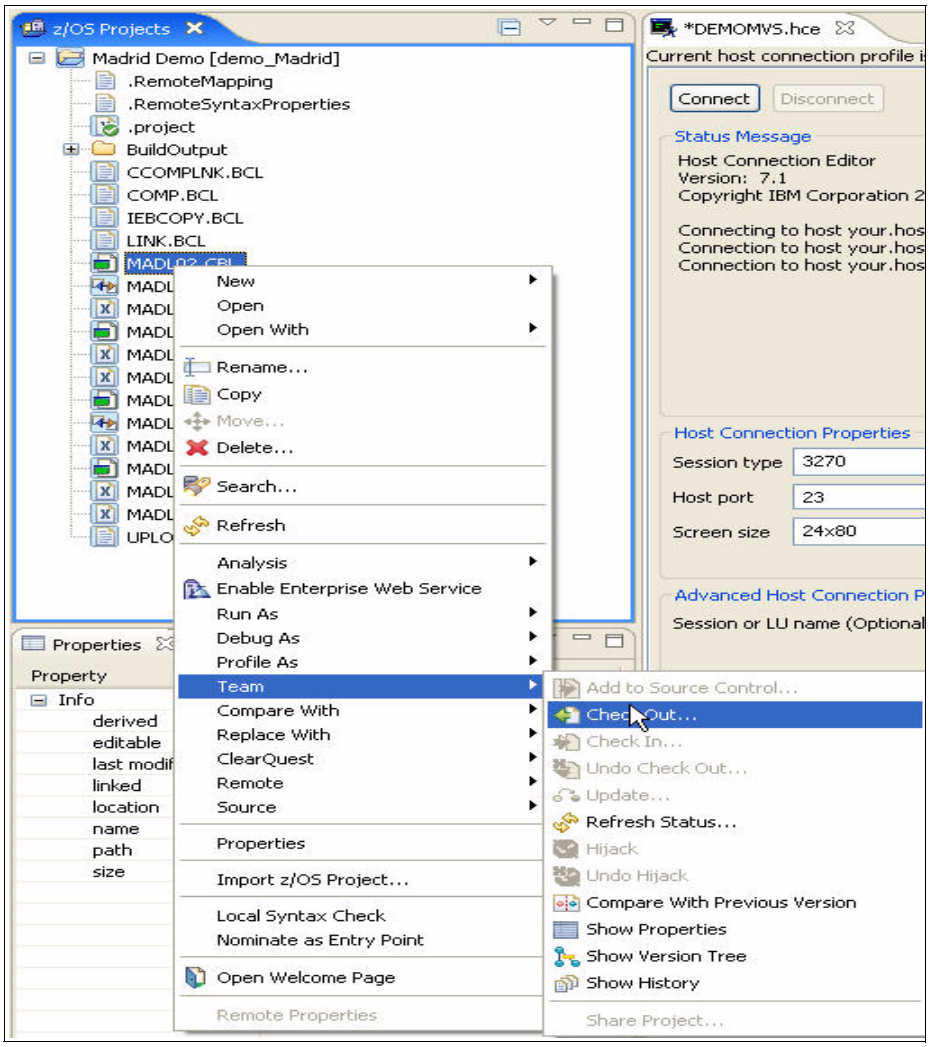


Figure 4-23 Checking out a ClearCase-managed element from Rational Developer for System z z/OS

4.4.5 Working with builds, migrations, and promotions

You can use any of the build solutions that we described in the section with Rational Developer for System z. Build Forge and ClearQuest all have fully functional Eclipse perspectives that enable you to use them from within Rational Developer for System z. These perspectives are well described in the documentation for the respective products.

Rational Developer for System z also includes its own build manager. As with Build Forge and the other solutions that we discussed in this section, you can build the same calls, to Remote Build or to a build engine script, into a Rational Developer for System z build step, as shown in Figure 4-24 on page 89 and Figure 4-25 on page 89.

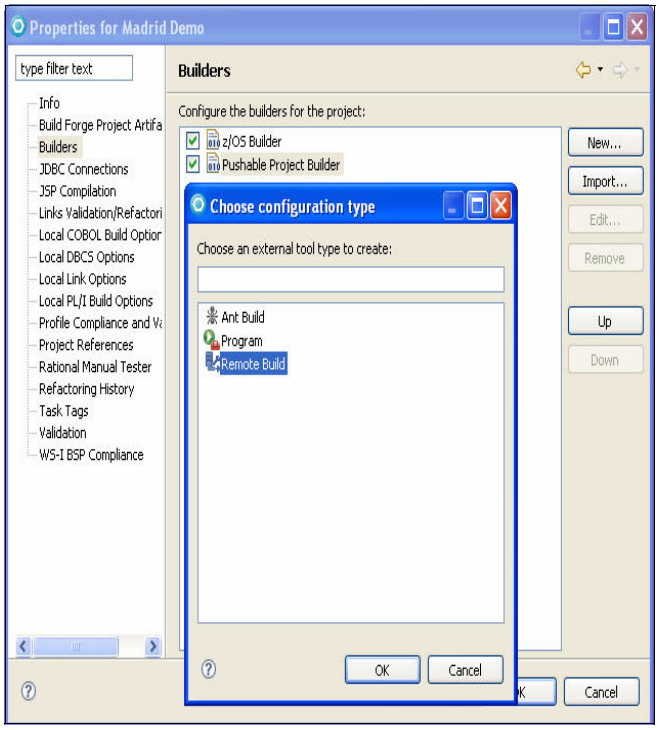


Figure 4-24 Configuring a Rational Developer for System z Builder for ClearCase Remote Build

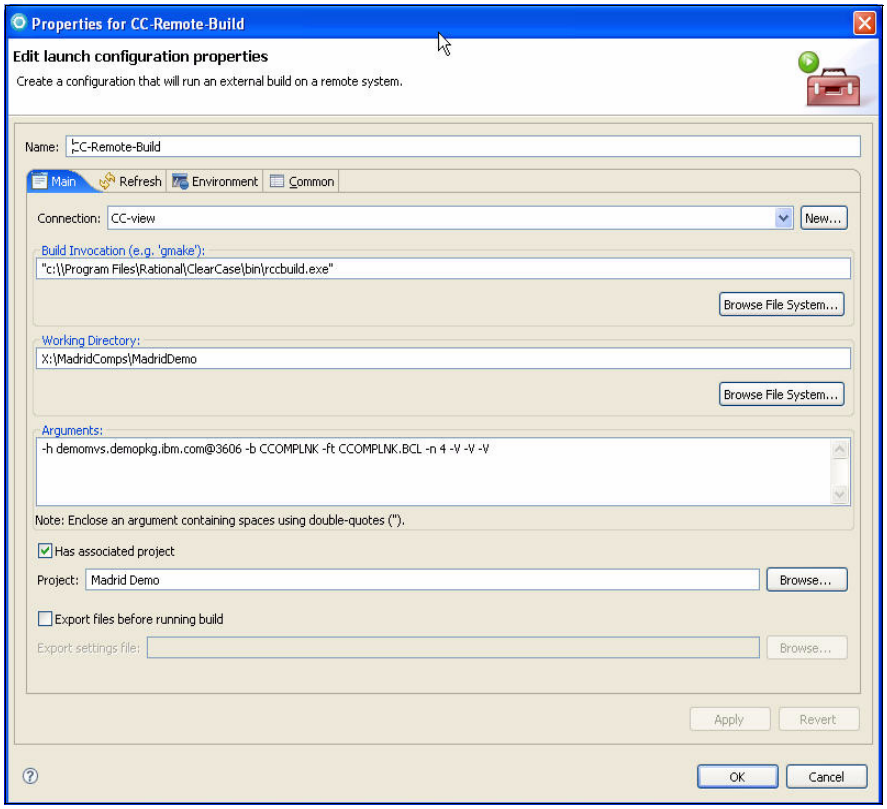


Figure 4-25 Creating a Rational Developer for System z Build Launcher to run ClearCase Remote Build

You can create additional launchers for each of your different build steps and options. After you create your launchers, you can build your project by selecting Project → Rebuild Project or by right-clicking the project itself.

When you run the build, Rational Developer for System z will display the results in a remote shell window.

As part of the proof of technology for this book, we also configured Rational Developer for System z builders to invoke ClearCase Remote Build directly and to invoke the Host-build.pl script, as shown in Figure 4-26.

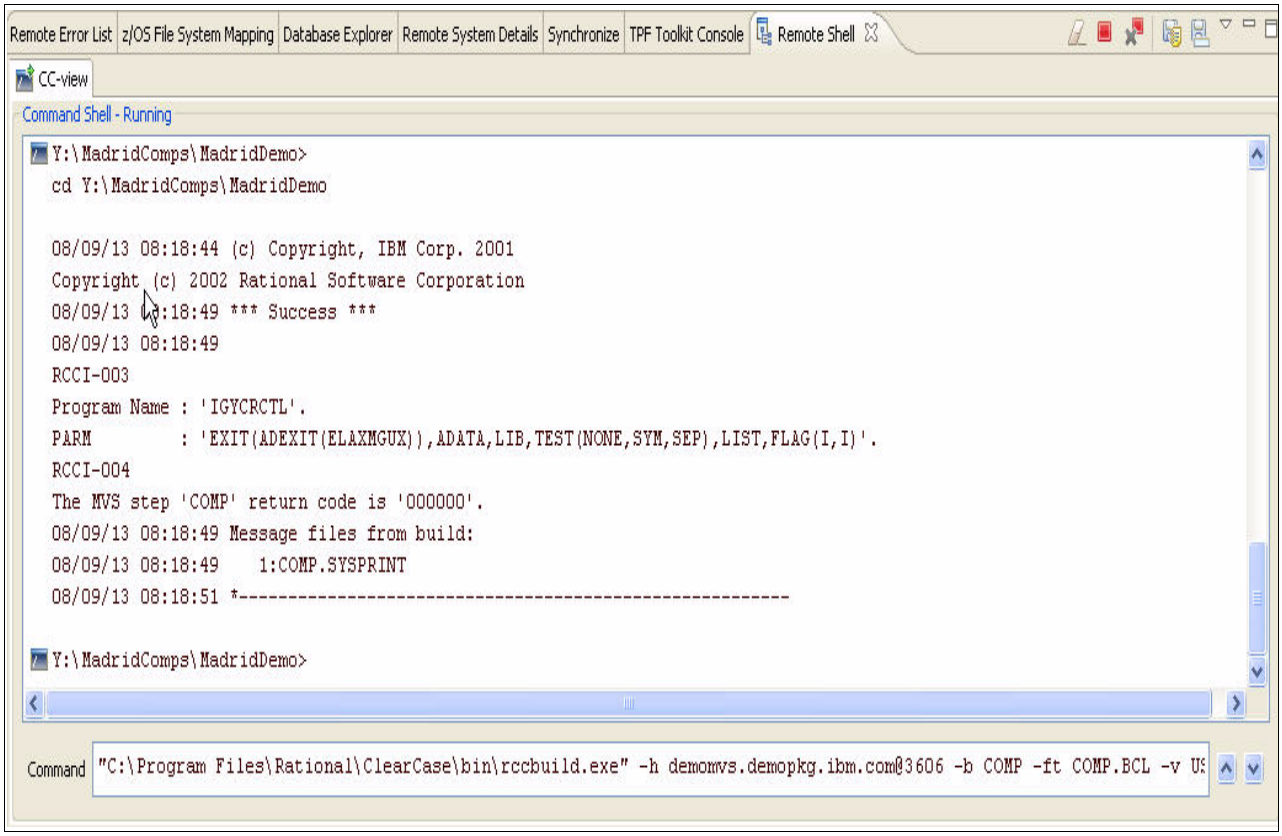


Figure 4-26 Remote Build output from the Rational Developer for System z Build Initiator

Regardless of which of the solutions actually invokes the build process, it is our experience that at some point in the process, there is a dependency on either Build Forge or the ClearCase Remote Build feature. The advantage of the Remote Build feature is that you can invoke it using both GUI and command-driven environments, which you can more easily invoke as part of an audited ClearCase build.

We found that you can construct a reasonable synchronized build solution just with ClearCase and Rational Developer for System z. Build Forge adds a layer of robustness and reporting that greatly simplifies this process, especially if you are confronted with audit or compliance issues.

As we developed the project for this book, we found that the following deployment model seemed to fit the best:

- Rational Developer for System z and ClearCase to developers
- Build Forge to lead developers and production control
- ClearQuest to project leads and Project Managers

As we saw earlier in this section, you can use a variety of build paths through these solutions. Figure 4-27 illustrates the alternatives. What you actually deploy depends on your organizational style, policies, and available solutions.

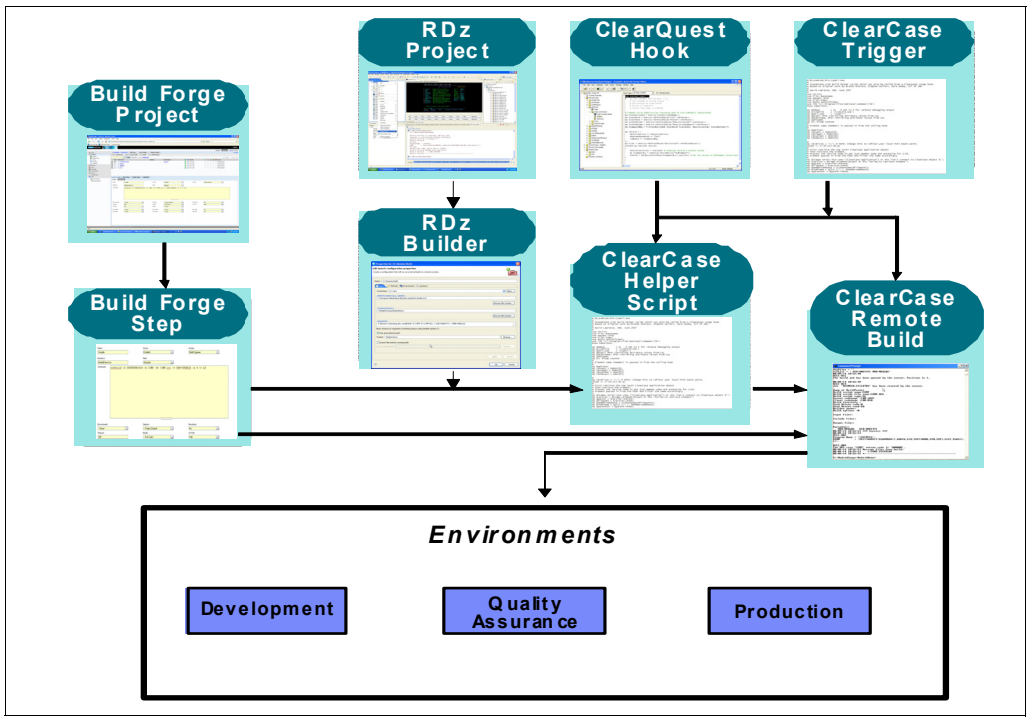


Figure 4-27 ESCM build paths with Rational Solutions

4.5 Using WebSphere Studio Asset Analyzer for impact analysis

In this section, we discuss how to use WebSphere Studio Asset Analyzer (WSAA) to help Project Managers, Programmer Analysts, Application Developers, Quality Assurance testers, and other users do application dependency analysis and impact analysis as part of the Application Lifecycle Management for System z. It is not the intention of this book to describe how to set up and use WSAA. More information about WSAA is at:

http://www.ibm.com/software/awdtools/wsaa

4.5.1 What is WebSphere Studio Asset Analyzer?

To start with, WSAA is a DB2 database that contains just over 100 tables. WSAA scans both mainframe and distributed software assets, takes the scanned information, and loads it into a DB2 repository that resides on the mainframe. These assets include PDS/PDSE data sets on mainframe and directories on Windows or UNIX. WSAA can also scan assets that Software Configuration Management systems manage. For distributed assets, WSAA supports scanning from IBM Rational Clearcase, CVS, and PVCS. For mainframe assets, SCLM is supported.

Figure 4-28 on page 92 shows the architecture of WebSphere Studio Asset Analyzer 5.1.

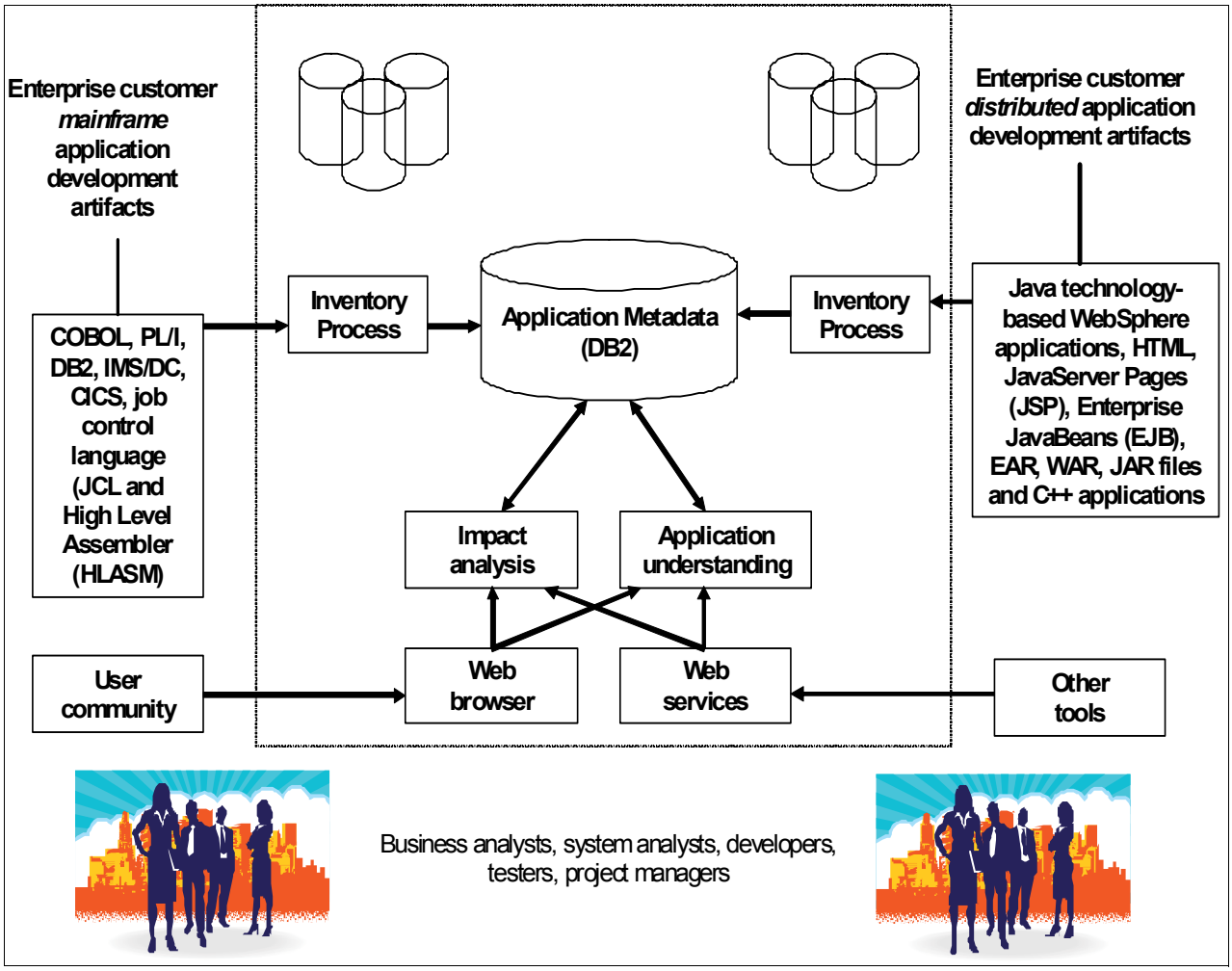


Figure 4-28 WSAA architecture diagram

WSAA is a WebSphere application. Typically, users access the application through a Web browser. The WSAA open architecture also offers user programmatic access through either direct *Structured Query Language* (SQL) queries or a Web Services application programming interface.

You can use WSAA in any phase of the application development process. However, in this section, we only discuss how to use WSAA to do application dependency and impact analysis as part of the Application Lifecycle Management for System z.

4.5.2 Identifying application dependency

Project Managers and Program Analysts can use WSAA to analyze application dependency and to define the scope of the business project before the project is assigned to work on. Using a Web browser, you can see your applications in both a general and detailed way. We use the Birthday application as the example to show you how to obtain the information. As we described in the earlier sections in this chapter, EPSL is a CICS application. A ClearQuest change request is submitted against this product defect.

Launch the WSAA with a Web browser. Typically you must login to WSAA depending on the set up of the WSAA system.

Figure 4-29 shows the WSAA main interface.

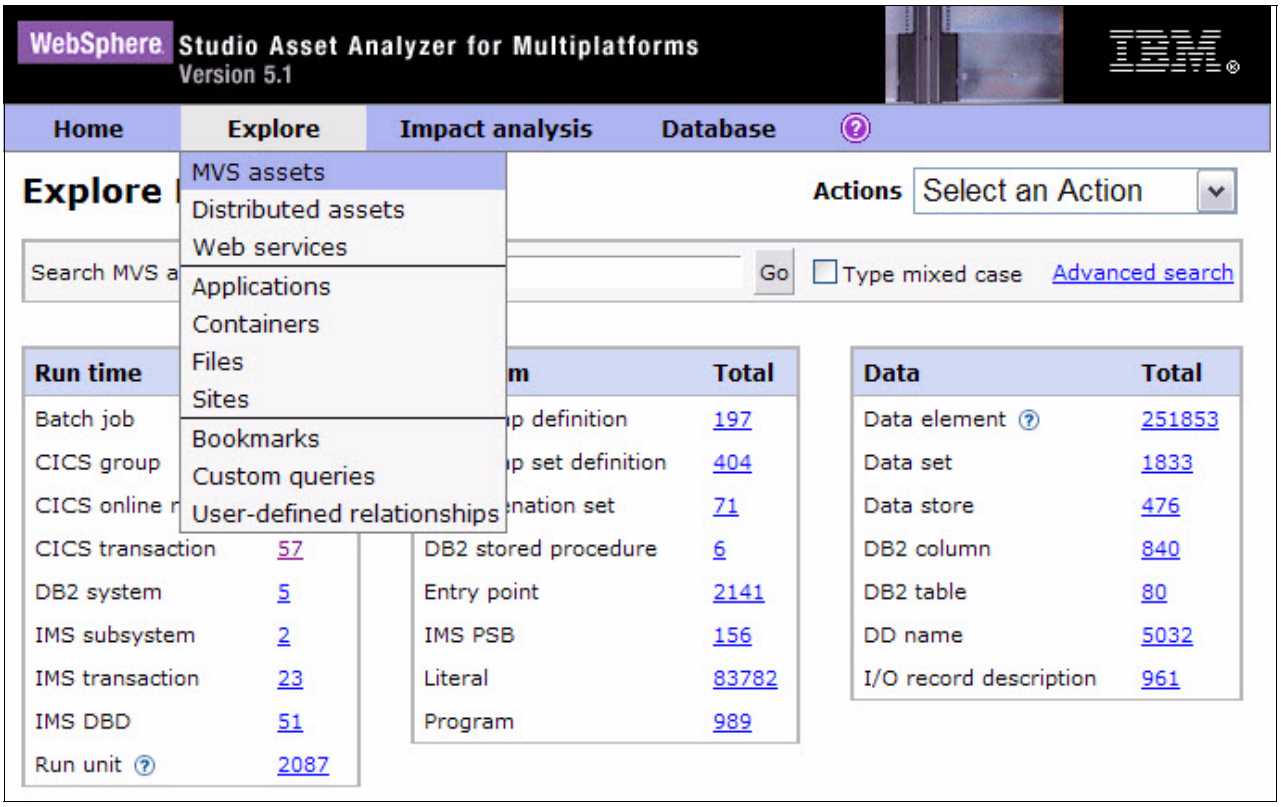


Figure 4-29 WSAA main interface

Because we know that the sample application, EPSL, is a CICS application, we select the Explore menu to list MVS assets. Using WSAA, you can search for particular applications too. For our example, we select EPSL from the CICS transaction category. Figure 4-30 on page 94 is the CICS transaction detail of the EPSL application. Other application information, such as online region, CICS group, main entry points, and other resolved entry points, are also displayed. WSAA also provides a variety of utilities for your convenience. You can display all of the application dependency information in a diagram, as shown in Figure 4-31 on page 95.

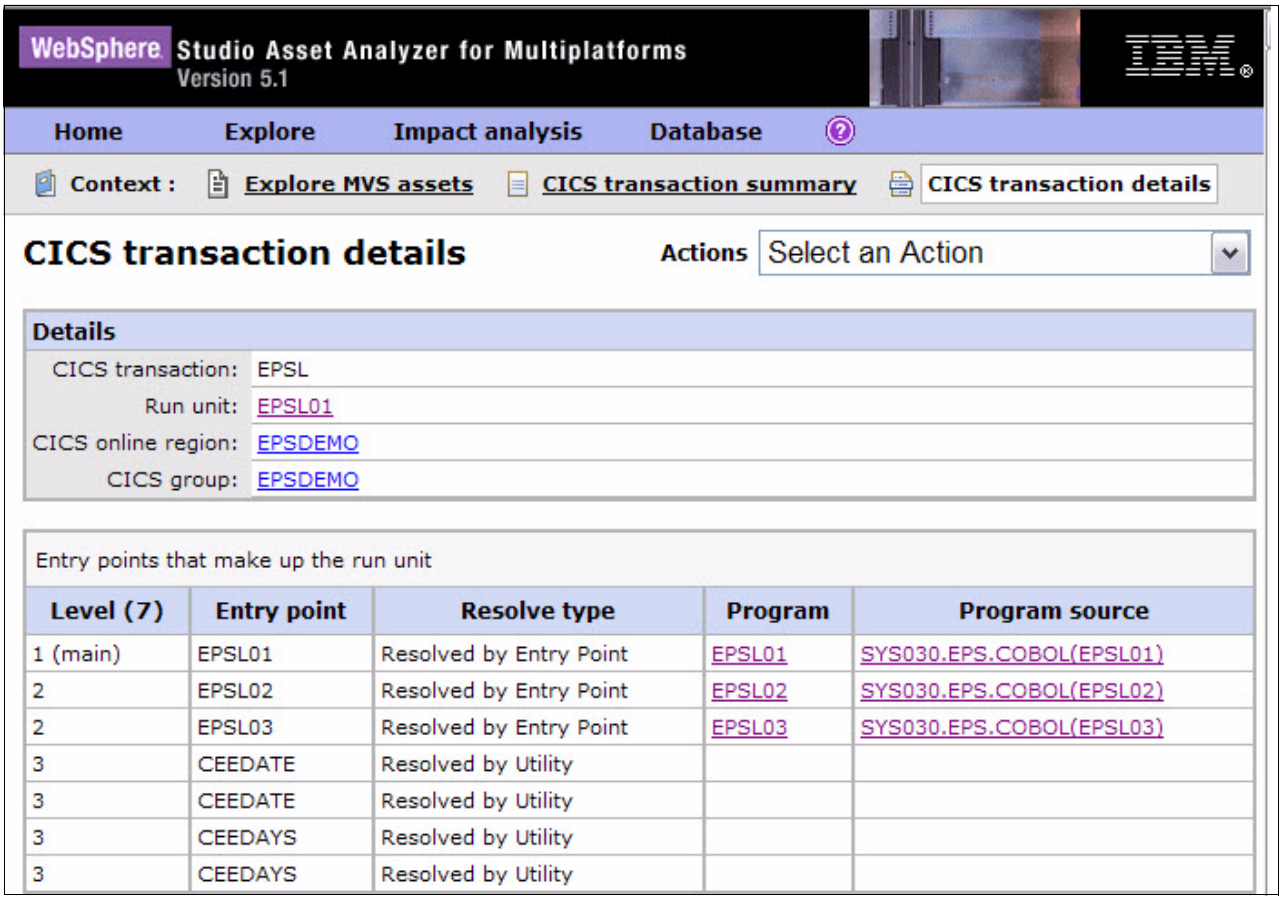


Figure 4-30 WSAA CICS transaction details diagram

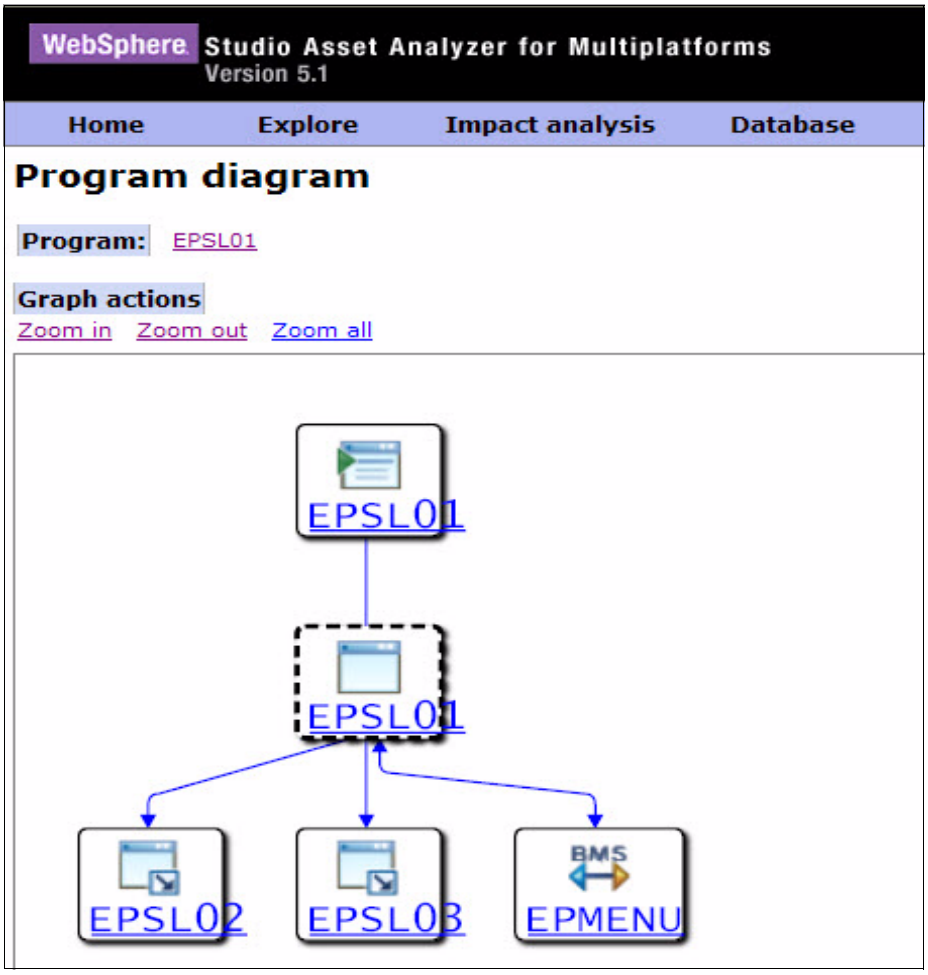


Figure 4-31 WSAA application dependency diagram

Figure 4-31 shows the high-level application dependency. You can further explore these assets with the easy to use functions that WSAA provides.

A structure diagram shows you more detailed information about the particular program. Figure 4-32 on page 96 is the program structure diagram of EPSL02. The structure diagram lists the procedures in program EPSL02. For our example, we know that the defect is about the calculation of the difference between today and the retirement date. The procedure “CALCULATE-DAY-DIFFERENCE” is the nearest area that developers should look at.

All program names and procedure names that are listed in the diagram are hyperlinks. Clicking a procedure in the program structure diagram brings up the source code of the selected procedure. Figure 4-33 on page 97 shows the source code of the procedure “A-200-CALCULATE-DAY-DIFFERENCE”.

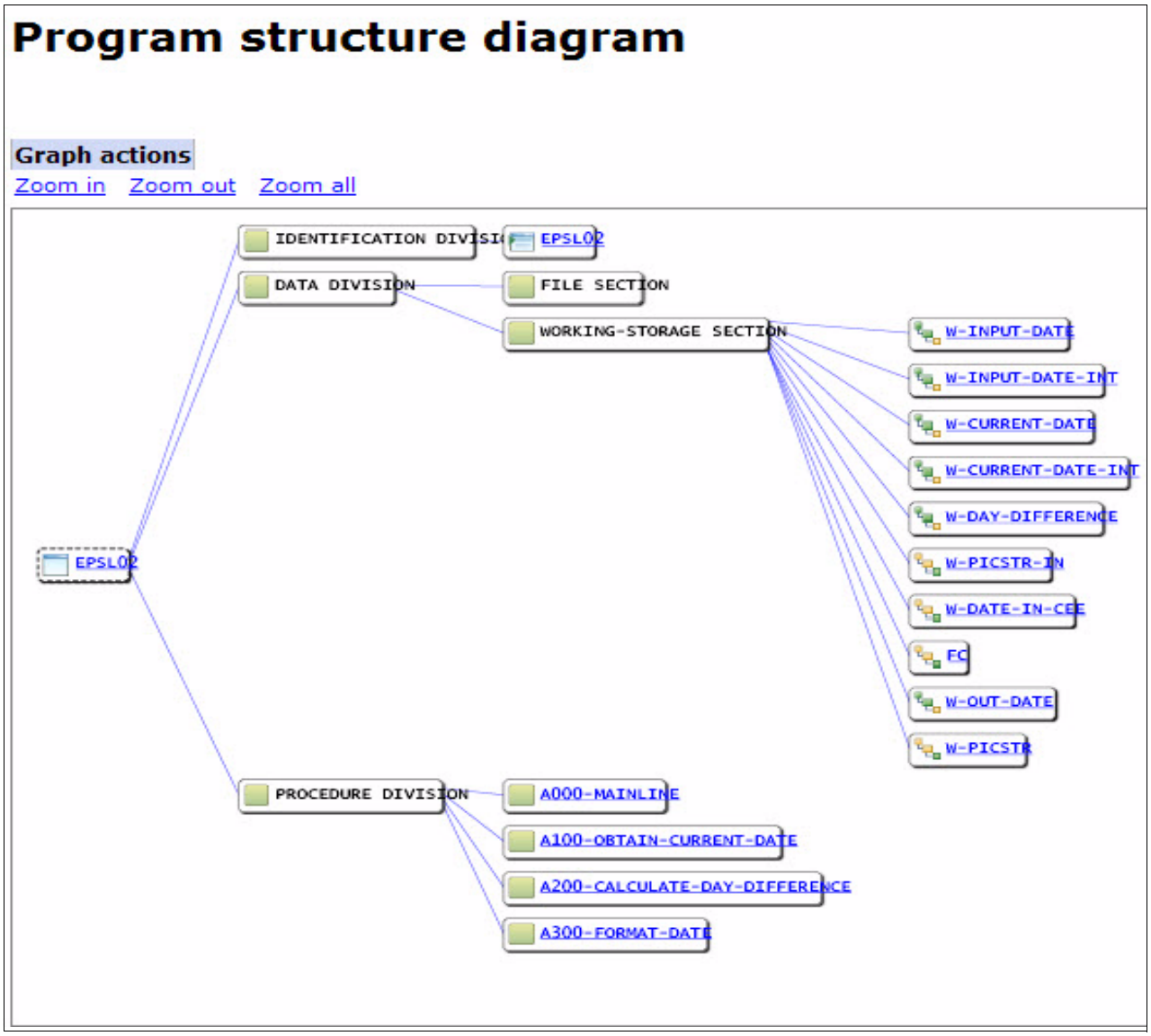


Figure 4-32 WSAA program structure diagram

```
81.             MOVE FUNCTION CURRENT-DATE(1:8) TO W-CURRENT-DATE
82.             COMPUTE W-CURRENT-DATE-INT = ,
83.                FUNCTION INTEGER-OF-DATE(W-CURRENT-DATE)
84.             .
85.          *
86.           A200-CALCULATE-DAY-DIFFERENCE.
87.             MOVE INPUT-DATE TO W-INPUT-DATE
88.
89.             COMPUTE W-INPUT-DATE-INT = ,
90.                FUNCTION INTEGER-OF-DATE(W-INPUT-DATE)
91.
92.             COMPUTE W-DAY-DIFFERENCE = ,
93.                W-CURRENT-DATE-INT - W-INPUT-DATE-INT
94.
95.             MOVE W-DAY-DIFFERENCE TO DAY-DIFFERENCE
96.             MOVE 0                TO COMM-PROGRAM-RETCODE
```

Figure 4-33 WSAA source code diagram

We just walked through how WSAA can help you to define the scope of the task with a simple example. The real application is more complicated, and it might take multiple iterations to define the right scope of the task. On the other hand, this is the reason that WSAA is useful because it provides an easy-to-use interface for you to explore and navigate from the higher-level structure diagram to source code.

4.5.3 Using the impact analysis

The impact analysis is one of the key features of WSAA. WSAA impact analysis helps you to evaluate the effect of particular code changes and to reduce the incoming impacts on assets. Figure 4-34 on page 98 shows the interface to create an impact analysis with WSAA.

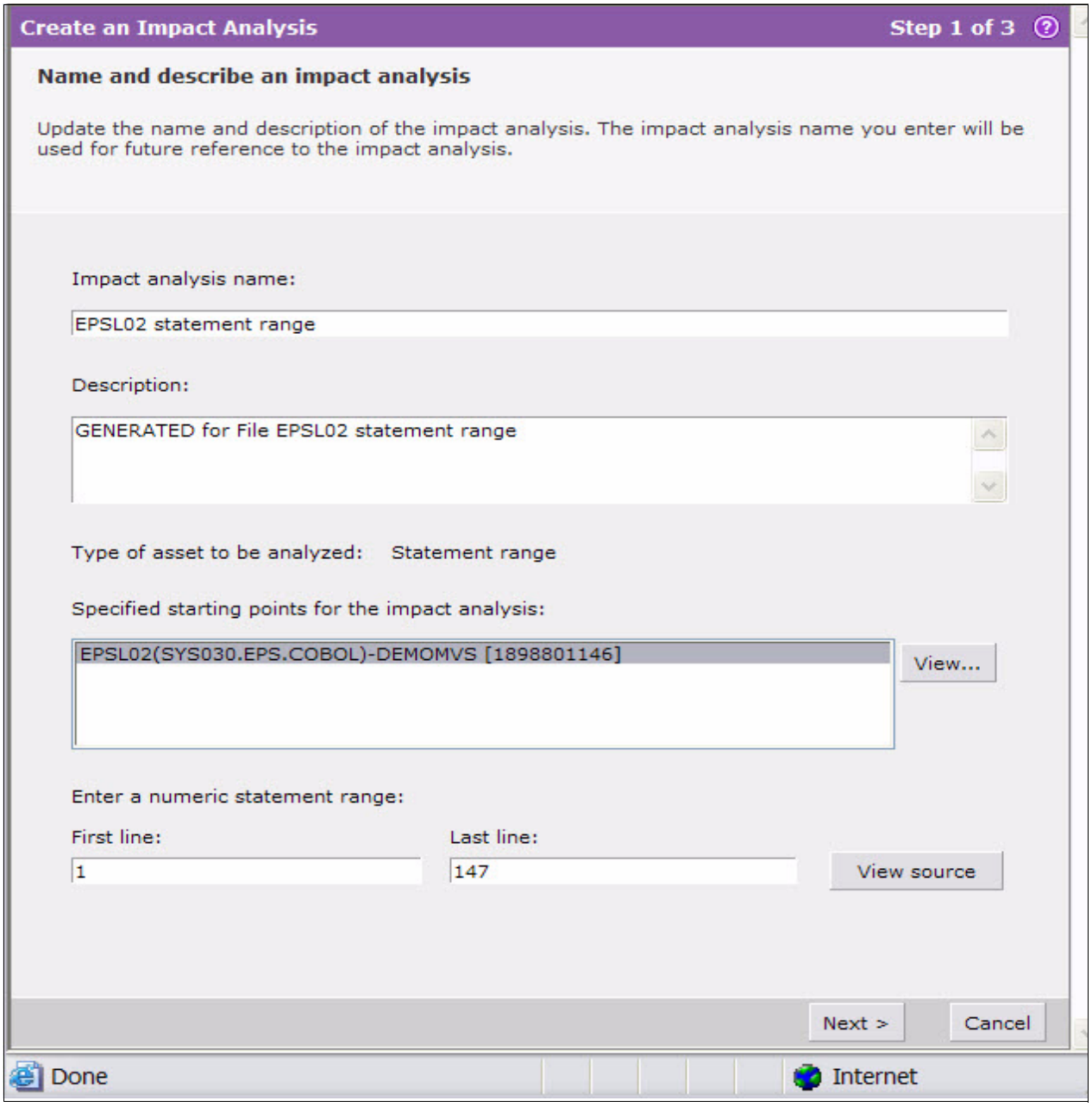


Figure 4-34 WSAA create impact analysis

Figure 4-35 on page 99 is the impact analysis diagram of the sample program EPSL02. On the right side of the diagram, WSAA shows the Indirect impacts, and on the left side of the diagram, WSAA shows the Direct impacts. The Direct impacts identify the number of data elements that are affected and the number of programs that use them. In the center of the diagram is the data set and data store information. Our simple example does not have any data sets.

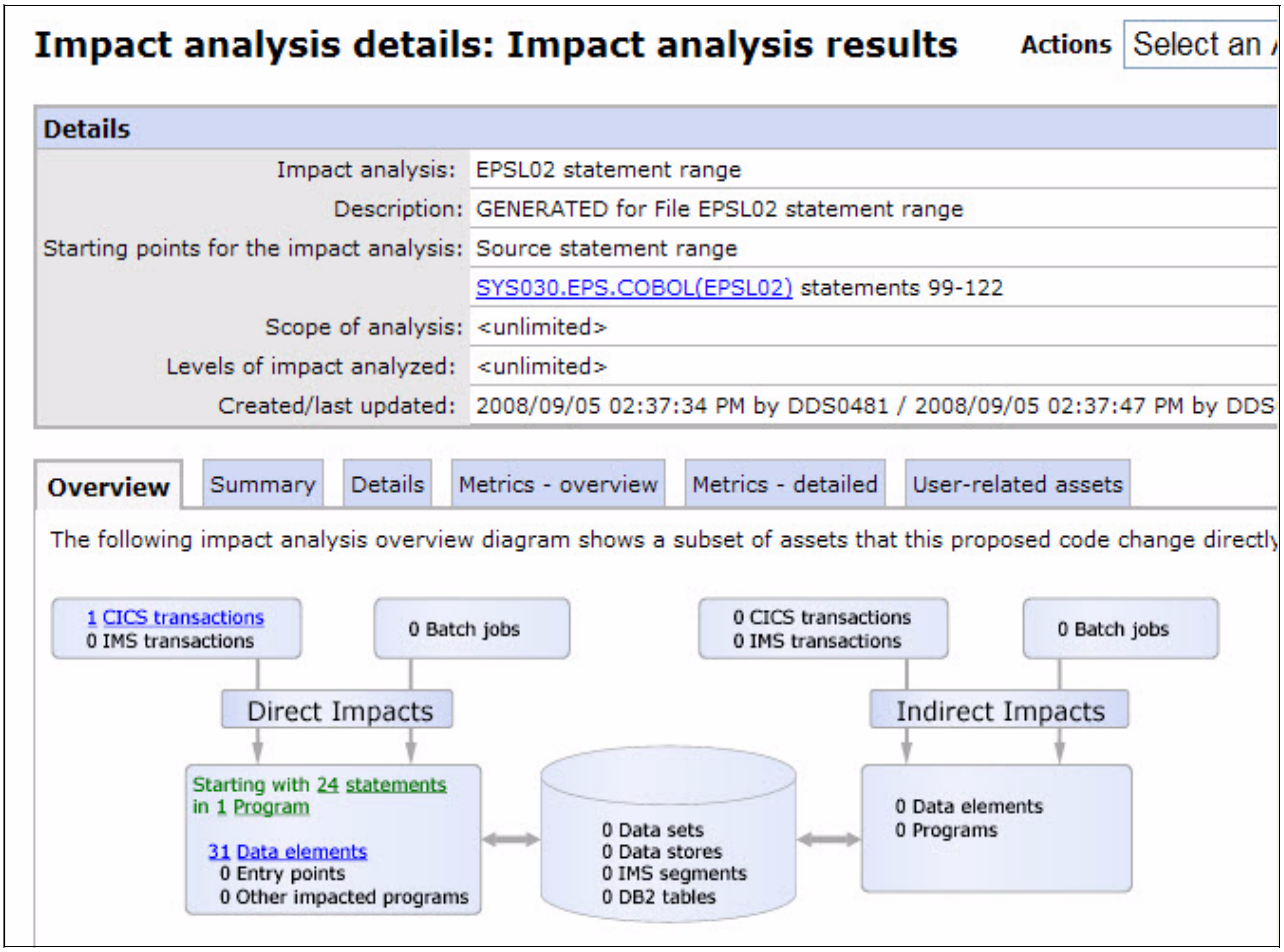


Figure 4-35 WSAA impact analysis diagram

Currently WSAA impact analysis is not automatically integrated into the managed build solution.

There are two ways to integrate WSAA impact analysis into the managed build process:

- Integrate using the WSAA programmatic access interface: Using the WSAA open architecture, you can access the DB2 database directly using SQL queries. So you can acquire the impact information and integrate it into the build process. You can also access the WSAA DB2 database with the Web Services API. Unfortunately, either way requires a certain amount of programming.
- Integrate using Clearcase and ClearQuest: In Section 3.3, we discussed the activity-based application life cycle. The goal is to ensure that all work and modifications take place in the context of a defined activity. All of the changed programs that are in the same activity are built together. If a program is affected by the changes in this activity and need to rebuild, you can simply add this program into the change set of the activity by checking out and checking in the same program with Clearcase. In our example, to fix the defect, we need to modify EPSL02 only. The change set of this activity has only one program, EPSL02. Although there is no code change that is required for EPSL01, WSAA impact analysis tells us that we might need to rebuild and deploy EPSL01. Checking out the EPSL01 and checking in the same file (without any code change) creates a new version with the same content that is in Clearcase; therefore, EPSL01 is added into the change set of this activity. When management build process builds and deploys the activity, both EPSL01 and EPSL02 are automatically built and deployed.

5

Starting an Enterprise Software Configuration Management project

In this chapter, we discuss detailed steps and proven best practices to start, plan, and implement a single-repository Enterprise Software Configuration Management (ESCM) solution. Implementing an SCM solution for even one platform has its challenges within an organization's development community and political infrastructure. When you add the development organizations of both the distributed and mainframe organizations, the challenge increases by at least a factor of two.

In this chapter, we first discuss the steps for organizing the project:

- The organizational aspects of implementing a single ESCM solution.
- Measurement of the organizations level of adoptability with a solution.
- A Core project team and roles within the organization.
- A Workshop approach and structure.
- An implementation strategy.
- Project milestones and measurements of success.

In the next sections of this chapter, we:

- Review and walk through the implementation steps in detail.
- Discuss how to execute the project tasks that we developed in the prior sections.

These tasks vary depending on your needs and requirements. We offer these implementation steps as examples that were used in actual implementations.

We also describe the following example migrations tasks and configurations:

- Data migration details
- ClearCase repository architecture
- Life cycle strategy

5.1 Project organization

In this section, we discuss the organizational aspects of implementing a single ESCM solution.

5.1.1 Organizational aspects of implementing a single ESCM solution

When starting an ESCM project, it is important to encourage communication among all affected developmental platforms, and make communication part of the project team. Include executive representation and technical representation from all developmental platforms as part of the communication and the project team.

In many instances, the original organizational contacts that were assigned or that started the ESCM project might have only owned one platform. Due to this fact, and due to silos of development across each respective platform, it is key to over communicate among the entire organization before starting the project. Communication might seem like a trivial point, but in most large organizations these software development, organizational, and political silos are prominent, and this is one of the primary reasons for moving towards a single, unified process for all software development.

Some consequences that can occur because of lack of or improper communication are:

- Project is delayed by architecture review from other technical organizations within the Enterprise.
- Difficulty in installing the solution pieces and parts on there respective platforms.
- Political entanglements with organizations within the Enterprise.
- Project overall cost is too high for adoption.
- Project gets cancelled.

5.1.2 Assessment of the organizations level-of-adoption of an ESCM solution

In the Table 5-1 on page 103 through Table 5-5 on page 106, we present questions to ask and the reasons why they are asked to gauge the level-of-adoption for an organization that wants to adopt a single, unified process-of-development. Ask these questions before the project starts, prior to any workshop, to each development platform, and compare the answers between their environments, which will identify differences and similarities between platforms, infrastructure environments. The organization's answers to these questions also gives a broad assessment of their environments, and how the organization adopts to change, for example:

- Do all platforms use the same naming conventions for there TEST,QA, and PROD environments, and are they referenced the same for any cross-platform applications?
- Do any cross-platform applications exist?
- Is parallel development an accepted practice between all platforms?

Table 5-1 on page 103 illustrates questions to ask and the reason for asking to what end result. These questions are a proven set that have been used to gauge the level-of-adoption and assessment.

Organization scope

Table 5-1 presents a list of questions to ask and the projected end result with regard to organization scope. These questions gauge the level-of-adoption and assessment.

Table 5-1 Organizational scope questions

Question	Objective(s)
How many software development areas exist within your organization?	Determines how many software development platform/ areas exist. Determines if the organization is aware of all development areas.
Is there an organizational chart that can be viewed?	Identifies any pockets of development that may exist. Determines communication paths for stake holders across platforms

Solution scope

Table 5-2 presents a list of questions to ask and the projected end result with regard to solution scope. These questions gauge the level-of-adoption and assessment.

Table 5-2 Solution scope questions

Questions	Objective(s)
Define your software development life cycle, for example, environments: Test, QA, Production?	Determines naming standard and life cycle standards across all platforms.
How many applications were newly developed over the last two years?	Identifies the trend or history of how many new applications are being developed on what platforms. Identifies the trend, from an architectural focus, of new developments from the Enterprise.
How many new applications will you develop over the next year?	Identifies the trending for a platform focus and an architectural focus.
What are your source code types?	Evaluates different source code that will be managed.
Are there different release levels in use for your source code, such as Cobol, C++, or .Net? Are different releases still being developed?	Determines how many levels need to be supported. Significant in determining the level of adoption to change within the development organization. If old levels are still being developed and not converted to the newer releases, or if there are no plans to do so, then it is a clear resistant to change.
Do you have duplicate named source code items with physically different sources that are related possibly to different customers or release levels?	Determines a release strategy, or identifies common source that is controlled by different development teams.
Do you have duplicate source files with different names?	Identifies a release strategy and common source that is controlled by different development teams, and it identifies reusability issues.

Questions	Objective(s)
What non-compile configuration items and versions (including other platforms) are maintained, such as, Word processing documents, project plans, requirement documents, JCL, parameters, database tables, compile options, Link cards, and so on?	Determines the development tools that are being used and where they are maintained. Also determines source code that is to be managed.
What concurrent development tools are being used, if any, such as Merging tools. Also identifies pockets of resistant to a single tool or development allowed to roll on your own instead of an Enterprise standard?	Determines parallel development tools, and how many tools are used by all development teams. Also identifies potential pockets of resistant to a single tool or development that is allowed to a roll your own, instead of an Enterprise standard.
What debugging tools are in place and used?	Determines how many development tools are in use. Also identifies a resistance from a single tool Enterprise standard.
How is work assigned to development teams?	Determines how work is decomposed, and assigned to development teams, the roles, and the process that is in place.
What metrics are in place to measure project health?	Determines what is measured and if measured, and if so how we can use this information with regard to our solution. In addition, with this and the next two questions, we can also determine CM maturity and possible measurements, if they do not exist.
How do you determine project success?	Determines what is measured and if measured, and if so, how we can use this information with regard to the solution.
What is measured to determine the delivered business value with regard to software development.	Determines what is measured and if measured, and if so, how we can use this information with regard to the solution.
What current reports are run and how often?	Gauges what they measure.

Users' scope

Table 5-3 has a list of questions to ask and the end result with regard to users' scope. These questions help to gauge the level of adoption and assessment.

Table 5-3 Users' scope questions

Questions	Objective(s)
How many software developers are in place today to support your application or systems?	Provides a sense of the number of developers and what they are supporting.
How many developers are remote?	Determines the architectural approach.
Is there any out-sourced development in place?	Determines the architectural approach.
If out-sourced development is in place, how is the software delivered?	Determines the process and use cases.
What software development roles are in place, and describe their responsibilities?	Determines how the software development process is defined, if at all.

Software promotion scope

Table 5-4 presents a list of questions to ask and the projected end result with regard to software promotion scope. These questions gauge the level-of-adoption and assessment.

Table 5-4 Software promotion scope questions

Questions	Objective(s)
Do you have applications that communicate with any other platforms?	Determines the cross-platform applications and processes that are in place to deploy the entire application to all owning platforms.
Do promotions or migrations through your life cycle require approvals?	Determines if there is an approval process in place.
In what environments of your life cycle are approvals required?	Identifies the approvals that are in place. If approvals are required for other environments, for example if the QA environment requires approval, there might be testing, coding and additional processes that are in place that need to be identified.
Are there any other products that interface with this approval process?	Determines if there is other tooling that controls the approval process and any other aspects of development.
Are Project IDs or change IDs required to change the source?	Determines if activity-based change is in place. Tests the level of the configuration management maturity of the Enterprise and the level-of-adoption to a full activity-managed system.
With promotions or migrations, through the life cycles, are the executables copied forward or are the recompiled from the source into the receiving environment?	Determines if there is a compile into the methodology or copy into the methodology.
Are migrations or promotions scheduled or adhoc?	Determines if there is a release strategy in place.
Are roll-back requirements for defects that are in production?	Determines the emergency fix process, if one exists.
Are there products in place that assist with promoting or installing code within the Enterprise, for example, install in CICS, DB2 tables, installation on servers?	Determines the product mix that is used for deployment and any processes that are in place that are used for deployment.

Audit scope

Table 5-5 presents a list of questions to ask and the projected end result with regard to software audit scope. These questions gauge the level-of-adoption and assessment.

Table 5-5 Audit scope questions

Questions	Objective(s)
Do you have a published software development life cycle policy that can be viewed?	Assesses the maturity level.
Are there any other published policies for software development?	Determines the maturity level of deploying or migrating through the life cycle.
When was your last audit conducted?	Determines if an internal audit does occur on the development environments.

Summary

The questions in Table 5-1 on page 103 through Table 5-5 can give you a general assessment of the client's environment and their Enterprise level-of-adoption. From an assessment perspective, this is a short list. Ask additional questions that relate to:

- Network:
  - Firewalls
  - Latency
  - Throughput
  - Domains
  - Protocols
  - VPN
- Standards:
  - Databases
  - Operating systems
  - Hardware
  - Security
- Web sever technology
- Architecture

5.1.3 Selecting a core project team

Selecting the core team is critical to the success of this type of project. Core team members must be comprised of an executive sponsor and technical leaders for each respective platform. All core team members participate in the projects from beginning to end, but some team members will contribute their time more than others.

The core team contributes valuable, needed input to the project structure, requirements, infrastructure architecture, design, and testing approach. The core team members should also participate in the initial workshop that we describe in 5.1.4, "Workshop approach and structure" on page 107.

Examples of Core team members are:

- Executive Sponsor: The executive that made the decision to move forward with the solution.
- Internal Auditor: Reviews any statutory requirements.
- Systems Administrator and Systems Programmer: Reviews install requirements and infrastructure support.
- Security Administrator: Reviews security requirements.
- One Lead Developer from each application that is affected, which includes all applications across all platforms.
- Configuration Management Administrators from all configuration management systems that are used within the Enterprise.
- One Project Manager.
- IBM.

You can add more team members, but your core team members must have deep technical knowledge of applications, supporting infrastructure, policies and procedures, and executive commitment.

Because the goal of the core team is to have technical and executive leaders that can make decisions around the solution, we recommend that you limit the size of the core team. The member descriptions in the list are the minimum requirements.

5.1.4 Workshop approach and structure

With the workshop approach, the core team (IBM, and Client staff) is assembled to meet for three-to-four full days. During this exchange, IBM educates and shares the technology around the solution and configuration management best practices. The Client, in turn, shares with the IBM team how they develop software, architecture, infrastructure, and the processes that are in place.

For both IBM and the Client, this exchange is the “I do not know yet what I do not know” meeting, which is essentially a more detailed assessment for both IBM and the Client about the solution and the Client environment. At the end of the workshop, both IBM and the Client will “know what they did not know”.

What we derive from this workshop is a project plan that both the IBM team and the Client core team develops. The next four subsections present an example format of this workshop. The workshop can run longer or shorter depending on the size of the Enterprise, but the three-to-four day workshop is proven successfully.

If additional communication and sharing is needed among other areas that where not part of the core team, we can set up additional short sessions based on relative concerns.

The following four subsections give you a general format of the workshop. The subject matter can change based on the Clients needs and experience.

Day 1 (Both IBM and Core team) client process and Rational tools

The schedule and activities for the first day of the workshop are:

- 9am: General discussion of current implementation:
  - How client developer do there work (offshore and Local)
  - Client release management process
  - Existing automation if any
  - Process and automation improvements
- Select Core team members, and assign roles and responsibilities
- Perform software development Analysis - How they work
- Workshop approach:
  - Selec Pilot application
  - Project approach discussion
- 1pm: General discussion Q/A Rational solution:
  - Short demonstrations
  - Concepts
  - Overview of how solution integrates with each platform

Day 2 (Both IBM and Client) usage model discussion

The schedule and activities for the second day of the workshop are:

- 9 am: Q/A session about previous days discussion
- 10 am: Repository architecture discussion
- Map current solution artifacts to repositories
- Analyze and review the software development
- 1 pm: Usage model discussion:
  - Review base ClearCase versus UCM
  - Review non-integrated use of ClearQuest and ClearCase
  - Determine current roles and process models
  - Review utilizing Clearquest to manage deployments

Day 3 IBM internal delivery planning

The schedule and activities for the third day of the workshop are:

- 9 am: Review workshop inputs
- Create project plan structure

Day 4 (IBM and Client) delivery planning

The schedule and activities for the fourth day of the workshop are:

- 9 am: Review plan and approach with Client:
  - Review issues and risks that were identified in the workshop
  - Modify the plan with Client advice and input
  - Define the deliverable documentation
- Quality Assurance: Review any items from entire workshop

5.1.5 Implementation strategy

Figure 5-1 illustrates a proven project structure. Some clients might want to use their own company standard project structure. We offer Figure 5-1 as an example that we used with success for numerous implementations.

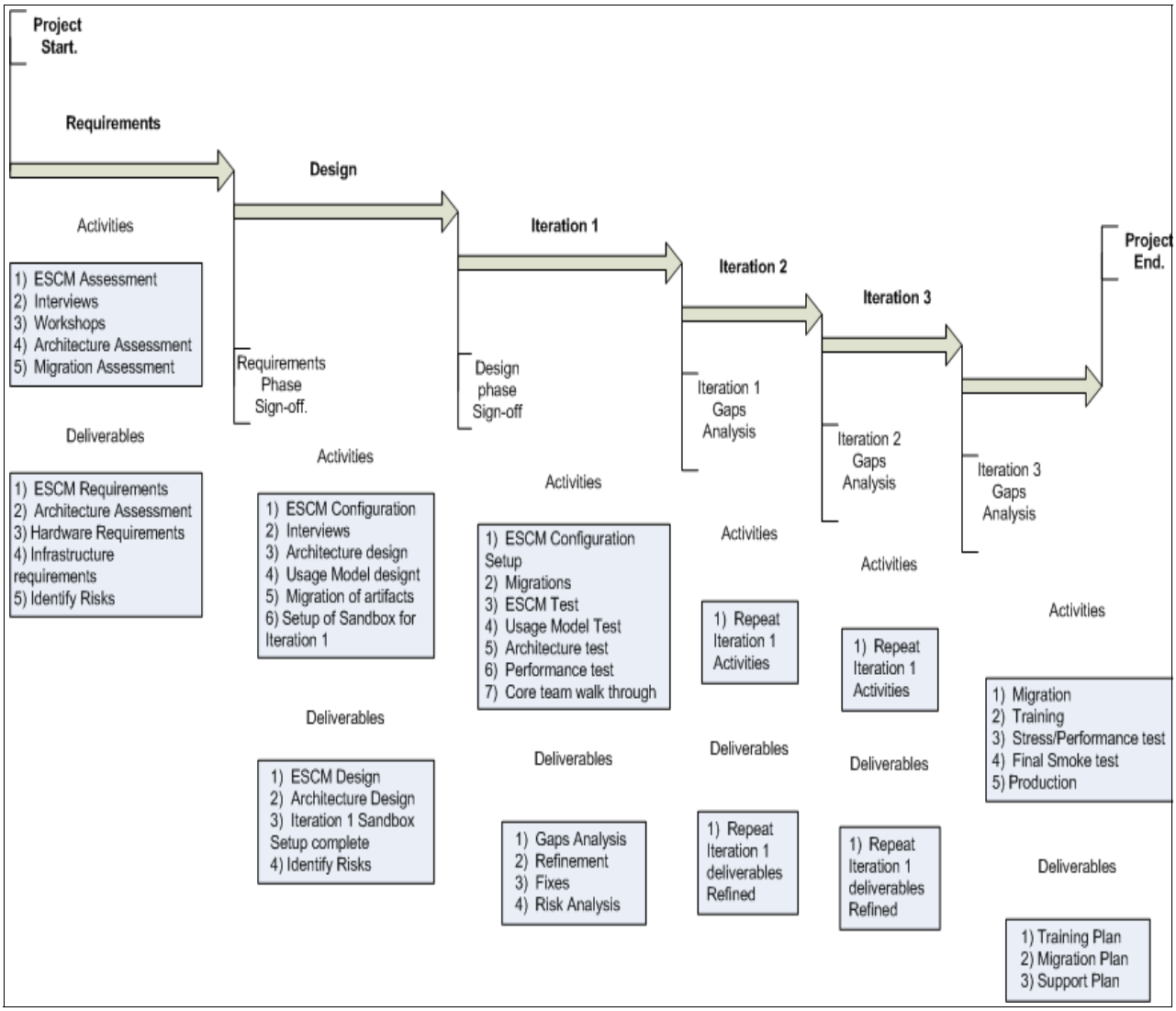


Figure 5-1 ESCM project structure

In Figure 5-1:

- The project structure is cut into five phases.
- The iteration phases are set to three, but you can expand them, if needed, for example, iteration nth. Typically three phases are enough if the requirements and designs phases are executed properly.

- This is not a serial path through the project. You can perform many tasks in different phases in parallel with others, for example, while in the Requirements phase, you can have hardware and migration set up to prepare for the Iteration one phase, and design aspects can be complete, with regard to the use cases.
- For the Requirements, Design, and the three Iteration phases, the requirement is that the core team sign-off or approve each phase as a group, before the end of each phase is considered completed. The sign off ensures that the team is in complete agreement and takes ownership of all requirements and design for the move to production environments after the iteration phases.

5.1.6 Project milestones and measurements of success

In this section, we discuss project milestones and measurements of success.

Project milestones

At the project start, determine significant milestones and measurements to report on the project's health at various points within each phase of the ESCM project. At the very least, the sign-off process for each phase should be a significant milestone.

Other example milestones for a project of this type can be:

Requirements phase:

- Deliver the first draft of the ESCM requirements document
- Deliver the first draft of the Architecture requirements document
- Determine the data migration requirements
- Identify pilot applications
- Identify job roles and responsibilities within the application life cycle
- Identify hardware requirements

Design phase:

- Deliver the first draft of the ESCM design document
- Deliver the first draft of the Architecture design
- Design the usage model, and select software development roles
- Acquire hardware
- Establish the infrastructure for testing the sandbox
- Install the sandbox, and prepare for preliminary design work

Iteration phases:

- Perform gap analysis for each iteration phase to determine if design and requirements are met. Determine any usability issues to address in subsequent phases
- Train the Core team
- Train development, and test the usage model
- Establish performance benchmarks
- Create quality assurance document (testing scripts)

Project measurements

Establishing project metrics is critical to determine project health and to validate why the ESCM project was even started. Because it can be challenging to consolidate different configuration management systems across platforms into a single repository with a single unified development process, you must establish metrics to measure the benefits of such an ESCM system.

This list contains examples of some project metrics that were used in similar projects:

- Cost of maintaining multiple configuration management systems compared to a single system.
- Cost of maintaining multiple software development tools compared to a single system.
- CPU utilization and usage of the existing system compared to a single repository. With this solution, the CPU should drop due to the off-host development.
- Complete control of cross-application deployments, and manage all components, both distributed and mainframe, with a single project, which ensures that all pieces and parts of the cross-application are deployed together.
- Common Software development governance across development platforms.
- Common traceability of projects across development platform.

Summary

The items in this section are just a few examples of project milestones and metrics that you can use, and we offer them only as a guide to the kind of tools that are available to monitor project health. There are many more, but the project core team should determine most of the measurements and metrics at the beginning of the ESCM project.

5.1.7 Migration tasks and configuration

In this section, we provide an example of a high-level overview of an ESCM project (project charter), which is then followed by a detailed task list. We used both of these examples for large ESCM deployments. Each ESCM project will have different characteristics depending on Client requirements.

Project charter

The purpose of a project charter is to clearly describe the executive sponsors' expectations of the project in terms of the following:

- The fundamental vision and justification of the project
- What is expected of the project
- The project evaluation criteria
- The project delivery team, context, and approach
- A high-level view of the proposed project schedule and effort

Business objective

The primary objective of this project is to migrate *client name* mainframe and distributed software development repositories and procedures from *current CM systems* respectively, to IBM Rational ClearCase and implement ClearQuest to support the release process.

As a result of this project, *the client* will have an operational ClearCase environment that can support their development teams' needs to access and modify code through remote access and their release teams' needs to promote deliverables from development into test and finally to production. In addition, ClearQuest will manage the release workflow, which allows automated management of the release process.

The solution includes the following products, which is just an example list because the actual list will depend on the Client's architecture:

- ClearCase v7.0.1:
  - ClearCase native client (CC)
  - ClearCase Remote Client (CCRC)
  - ClearCase Remote Build (RCCbuild)
- ClearQuest v7.0.1:
  - ClearQuest native client (CQ)
  - ClearQuest web (CQWeb)

Project objective

The scope of the project objectives are detailed in the project statement of work, and are included in the project charter for convenience. Normally the statement of work identifier is called out in this section. The scope and objectives are maintained in the statement of work and risks are maintained in weekly status reports.

Major deliverables and milestones

The following initial deliverables define the major milestones of the project. The tasks that are associated with milestones are in the project plan.

Deliverable milestone description

The following list contains deliverable milestones:

- Project Initialization
- Delivery of Project requirements
- Delivery of usage model and environment specification: Design
- Configured ClearCase environment in place
- Configured ClearQuest environment in place
- Mainframe CM system pilot migration completed
- Distributed CM system pilot migration completed
- Iteration One
- Gaps analysis
- Iteration nth
- Training

Project success indicators

The Project success indicators can be organized into a number of individual tasks that result in specific capabilities of the solution and use cases that define the features of the solution.

Use the following specifics capabilities of the solution as early success indicators of the project. These are just examples because there can be many more.

Initial technical success indicators

The following list contains initial success indicators:

- Installation and configuration of ClearCase
- Installation and configuration of ClearQuest
- Installation and configuration of CCRC
- Installation and configuration of CQWeb
- Configuration of LDAP for ClearQuest authentication
- Installation and configuration of RCCBuild
- Migration of pilot application artifacts
- Access to ClearCase assets

The following features of the solution define the final success indicators of the project:

- A ClearCase environment that is configured to represent the project's architecture such that it supports access and builds.
- Ability to access (checkout, modify, checkin) code from the ClearCase environment.
- Ability to promote ClearCase managed assets from development to test and then to production in the appropriate environments that support the release process for both mainframe and distributed artifacts.
- Ability to release and change between releases for reporting audit purposes.

Project boundaries

The following boundaries define the constraints that affect the successful outcome of the project. Additional boundaries might exist in the statement of work:

- The migration must be completed by *mm/dd/yy*.
- All element types must be identified by the *Client team*. The IBM Rational solution team creates the appropriate attributes in ClearCase.
- The *Client team* must identify all build and deployment processes, and allow the IBM Rational solution team to map them to the appropriate scripts in ClearCase.
- The *Client team* develops and delivers the user training

Assumptions

The following descriptions of the project assumptions are required but not part of the planning and execution of this project:

- Appropriate hardware and infrastructure is available and supported for the duration of the project.
- Pilot application-development schedules are such that asset migration can occur when the IBM Rational solution team and the *Client pilot team* are ready.

Dependencies

The following list contains descriptions of dependencies that the project team requires for successful completion of the project:

- Client subject matter experts must be available throughout the project to provide timely answers to technical questions.
- IBM subject matter experts will be required to identify element type attributes and map accurately current build processes to scripts within ClearCase

Risks

The following list contains descriptions of the initial risks that the Project team identified; however, the current risks of the project are maintained and communicated in weekly progress reports:

- Ability to display return code to the developer
- Ability to configure WSAA integration
- Ability to support out sourced development in India
- Ability to deploy to four separate logical partitions (LPARs)

Strategy overview

The project strategy is to first install and configure each product, and then verify the communication between each component of the solution. After each component is properly installed, evaluate higher-level technical capabilities of the solution using one or more

prototypes. Perform final migration of project code and validation of the solution after each of the individual capabilities of the solution are validated. This approach results in validating each of the *Initial Technical Success Indicators*, followed by validating the *Final Feature Success Indicators*.

Tradeoff matrix

Complete the tradeoff matrix, shown in Table 5-6, to capture executive management's priorities for the project. Throughout its life cycle, the project will encounter challenges that require timely decisions. The tradeoff matrix provides executive direction to the project to maximize business benefits. Any contradictions to the tradeoff matrix require escalation to the project sponsor.

In Table 5-6, the following definitions apply:

- Fixed: Critical need without flexibility
- Flexible: Secondary importance relative to other factors
- Accept: Least important relative to other factors

Table 5-6 Tradeoff matrix

Tradeoff matrix	Fixed (0-2 choices)	Flexible	Accept
Target organization (1 Choice)	**X**		
Scope (1 choice)		**X**	
Schedule (1 choice)	**X**		
Cost (1 choice)			**X**
Quality (1 choice)		**X**	
Resources (1 choice)			**X**

Project team

In this section, we discuss the project team.

Stakeholder team

Table 5-7 describes the Client team for this project.

Table 5-7 Stakeholder team

Name	Stakeholder Group/role	Responsibilities	Contact info
John Doe	Client sponsor	Executive authority for the project	
Jane Doe	Client Project manager	Project management authority	
Jack (lead dev) Ben (lead dev) John (CM admin)	Client Pilot Team	Development Tool pilot management	
Jay (CM admin) Don (MF admin) Jake (MF admin)	Client Solution team	Tool administration and mainframe infrastructures	

Name	Stakeholder Group/role	Responsibilities	Contact info
Pat	Client Change Manager	Software development process standards	
Ed	Client Internal Audit	Software project audit standards	
Art	Client information security	Software project security standards	
Lisa	Client DBA	Database standards	
Jack	IBM Client Exec		
Amy	IBM Software Sales Rep		
Tony	IBM Rational Software Sales Rep		
Dave	IBM Rational Services Mgr	Primary Contract authority Escalation of business issues	

IBM Rational services team

Table 5-8 describes the IBM Rational services team for this project.

Table 5-8 IBM Rational services team

Name	Role	Responsibilities	Contact Info
Barney	Solution Architect	Solution point of contact	
Beth	IBM Rational Project Mgr	Engagement point of contact	
Jack	IBM Rational Architect	Technical point of contact	
Jamie	IBM Rational Consultant	Implementation	

Proposed high-level schedule and rough effort estimates

Table 5-9 provides a high-level schedule and rough effort estimates for this project that are broken down into deliverable/milestones.

Table 5-9 Schedule and estimates

Deliverable/Milestone description	Target start date	Target end date
Inception Project Kickoff Workshop Installation		
Requirements Requirements documentation		
Design		
Elaboration		
Usage model		
Prototype		

Deliverable/Milestone description	Target start date	Target end date
Update usage model		
Design document		
Iteration 1		
Construction		
ClearQuest		
ClearCase		
Planning		
Migration		
Training		
Team walkthrough		
Gaps Analysis		
Iteration nth		

The Iteration phases are repeated, with tasks repeated for each iteration, and a gaps analysis occurs after every iteration. With this approach, the solution gains acceptance with each iteration until the final iteration when the migration is completed and accepted.

Signatures

Signing the signature block in Table 5-10 indicates that you read and accept the contents of the current version of this project charter.

Table 5-10 Content signatures

Name	Role	Date	Signature
Mike	Client Project manager		
Pat	Client Change Manager		
John	Client internal Audit		
Lisa	Client DBA		
Tony	Client information security		
Jake	IBM solution architect		
Paul	IBM Project Manager		

The signatures indicate acceptance to the approach of the ESCM project from the Client and IBM solution team (Core team).

Sample: Project plan

Table 5-11 on page 117 is a sample list of the tasks that are necessary to conduct a migration to the ClearCase/ClearQuest ESCM solution. These tasks are related to the migration of mainframe artifacts. Include additional for more platforms. This is an outline of tasks for you to incorporate into the project model step.

Table 5-11 Sample detailed project plan

Main Task	Subtasks	Days	Responsible	Status	Comment
I. Requirements Analysis					
Methodology/Life cycle	Interview software configuration management staff to determine the current build process				
	Interview software configuration management staff to determine current deployment process				
Scope of inventory					
Configuration requirements	IMS/DB2 requirements				
	IMS integration of executable				
	Infrastructure requirements				
	Audit/security requirements				
	CICS requirements				
	File naming standards and requirements				
Software promotion requirements	Promotion approval requirements				
	Operations/Production control requirements				
	Scheduling interface requirements				
	Tivoli/BuildForge/Data migration processing requirements				
	Tivoli/BuildForge/Data migration Back out processing requirements				
	Migration process review				
	Changes to policies and procedures (if any)				
Training requirements	Create TSO Client training Client specific				
	Create Tivoli/BuildForge/Data migration training				
	Create mentor schedule				
II. Technical Analysis					
Inventory Analysis	Identify compilation process jobs				
	Determine Link edit procedures/requirements (today)				

Main Task	Subtasks	Days	Responsible	Status	Comment
	Determine compile procedures/requirements (today)				
	Identify Link/Compile options				
	Compile Link options analysis				
	Identify element types				
	Classify inventory in ClearCase by application/type in repository (VOB)				
	Implementation/migration of code requirements				
	Contingency plan (initial)				
File naming standards /guidelines					
DASD estimates					
Acceptance testing criteria/ Test Plan					
Schedule QA classes					
III. Administration Set up					
Install ClearCase z/OS Extensions					
	Ensure current maintenance is applied				
Design Setup file allocation scheme					
Allocate files for testing and coding phases					
IV. Design					
Design Build Control language					
Design RCCBUILD scripts					
Design deploy scripts					
Design build and deploy started task scheme					
Design security rule strategy					
Design Build Engine/BuildForge					

Main Task	Subtasks	Days	Responsible	Status	Comment
Design deployment strategy					
Design Vob architecture					
Design Stream mapping					
Design workflow process and deployment ClearQuest					
Review Design - Usage Models					
V. Coding					**These tasks repeat for every element type or language used. This is just an example for Cobol**
Code scripts	Code COBOL BCL compile process				
	Code LLA/CICS new copy process				
	Code RCCBUILD scripts				
	Code BCL deploy scripts				
	Code BCL delete scripts				
	Code start and stop BCL related to DB2				
Create Build Engine	Create Build Engine for COBOL				
	Create Build Engine for JCL				
	Create Build Engine for copybooks				
	Create Build Engine for PARMS				
Create Link and compile option off load program (modify and review)					
Create utility job to migrate application to ClearCase					
Create IEBPUTPCH job					

Main Task	Subtasks	Days	Responsible	Status	Comment
Create job to transmit applications to ClearCase					
Create job to assign element-level attributes to migrated code					
Create Job to assign stream/project-level attributes					
Create ClearCase triggers					
Create ClearQuest triggers					
VI Testing					
Test scripts/triggers and processes	Test COBOL and COBOL build scripts				**These tasks repeat for every element type or language used. This is just an example for Cobol**
	Test deploy scripts				
	Test delete script				
	Test start and stop DB2 scripts				
Test Code Migration	Test and validate Link/compile options off load program				
	Test and validate migration job to migrate code to ClearCase				
	Test IEBPUTPCH job				
	Test transmit job to ClearCase host				
	Test and validate job to assign ClearCase attributes				

Main Task	Subtasks	Days	Responsible	Status	Comment
Test Build Engine	Test and validate COBOL				**These tasks repeat for every element type or language used. This is just an example for Cobol**
	Test and validate PARMS				
	Test and validate JCL				
	Test and validate copybooks				
Test usage model and flow					
VII. Acceptance Testing					
Inventory verification of load					
Users test compiles and deployments	Compile/Link options				
	Library concatenations				
	Listing information				
	Do start and stop procedures work				
	Do CICS newcopy and LLA refresh work				
	Do programs function				
Users test promotions	Promotion procedure and controls				
	Tivoli/BuildForge/Data migration package processing				
	Checkout processing				
	Checkin processing				
	Package processing				
	Security controls				
	Configuration set up and controls				
Acceptance testing and support	IBM support				
	Training on the solution				

Main Task	Subtasks	Days	Responsible	Status	Comment
VIII. Pre implementation					
Notify Disaster Recovery Team of new production files					
Create a development freeze date prior to implementation					
IX. On going tasks					
Modify documentation					
Training					
X. Implementation					
End user training					
Production migration					
Inventory verification					
Post implementing					
Contingency Plan preparation					
Total Days				100-150 on average	
Notes:					
The design, coding, and testing tasks are expanded based on the mix of element types					
Impact analysis design is modified depending on integration with WSAA or home-grown database or Native ClearCase					
Determine deployment strategy Tivoli/BuildForge/Data migration tool					

5.1.8 Physical data migration

Figure 5-2 graphically highlights the physical code migration process into a ClearCase repository. In this example, most of the preparation tasks are run on the mainframe, which includes options analysis, dependencies analysis, and preparing files that will be transmitted using file transfer protocol (FTP) to the ClearCase host. After the file(s) are on the ClearCase host, PERL scripts coupled with ClearCase commands are executed to import the mainframe artifacts into the ClearCase repository.

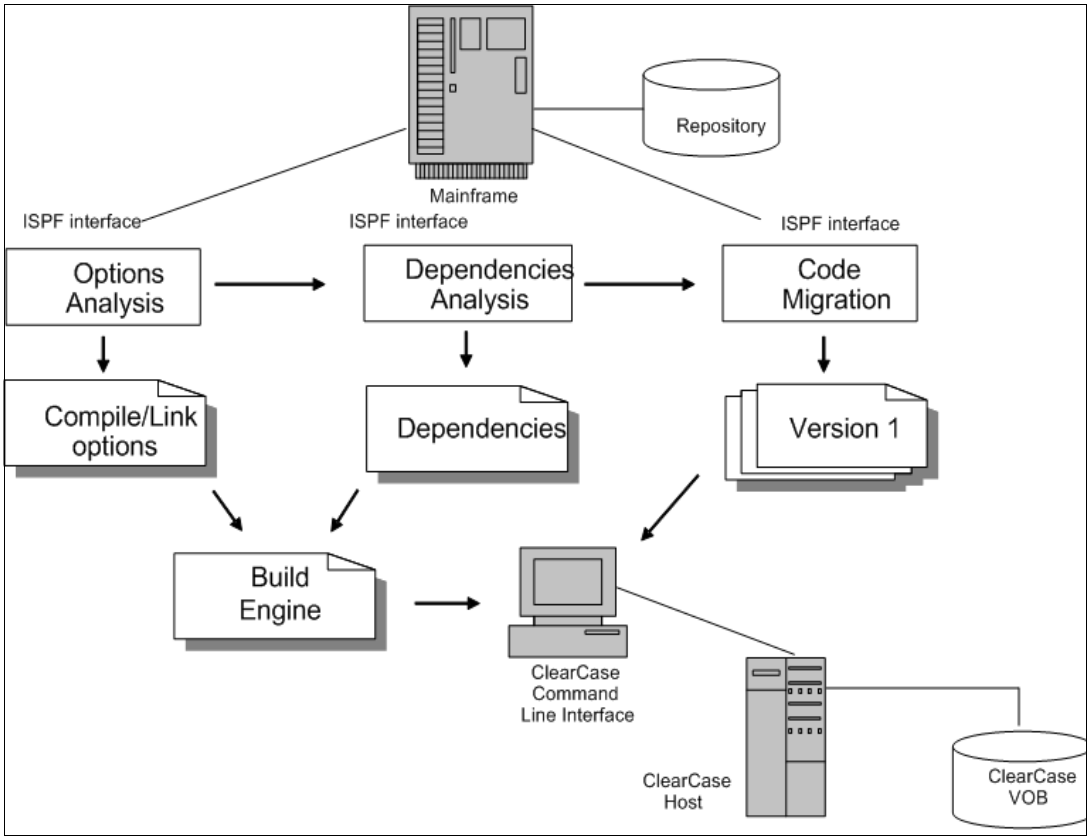


Figure 5-2 Migration process

Options analysis

Compile and link options are analyzed from the configuration management system that is being migrated to ClearCase. These options can be versioned elements within the configuration management repository that are stored as a selection list within the CM system or defaulted to as part of the configuration management system's managed build process.You must determine this in order to migrate the proper compile and link options of each element.

If the compile and link options are versioned elements within the repository, they are migrated along with the source code as versioned elements within ClearCase. There will be a one-to-one mapping from the old configuration management store to ClearCase.

If the compile and link options are stored as a selection list that developers can chose from and compile time, migrate these options to ClearCase as element-level attributes within ClearCase, which allows developers to make the same type of selection that they made within the old system, but instead of using ISPF to make the selection, they use ClearCase Explorer.

If the compile and link options are defaulted to within the managed build process of the old configuration management system, they are also carried forward when migrating the build process to the ClearCase managed build process.

Dependencies analysis

Dependencies analysis consist of four major tasks:

- Determine what type of dependencies exist within the application that was provided, which can be done by writing a scanning program using the search tools within ISPF. This process discovers any dependencies, such as copybooks, include files, subroutines, and any third-party tools. This process is also part of an inventory analysis, but you use it here to discover any issues with dependencies.
- Confirm that the inventory includes all dependencies that are provided with the inventory list, which you do to confirm that the listed inventory actually captured all of the true dependencies. In some cases, an inventory is provided, but dependencies exist outside of the configuration management system. You perform this step to confirm the actual inventory of the applications.
- Confirm if there are any called or linked programs that are not part of the inventory list that was provided. You do this to ensure that all dependencies are part of the configuration management system that will be migrated to ClearCase.
- Verify the concatenation structure within the managed build or build process of the CM system that is being migrated to ClearCase. Next verify that all dependencies are stored within that concatenated data sets, which you do to validate that all dependencies are accounted for.

Migrating the source to ClearCase

To migrate the source to ClearCase, you first isolate the source that you are migrating from the old configuration management system, which you do only if the configuration management system that is to be migrated to ClearCase has a proprietary access to its artifacts. If the configuration management system does not have a proprietary store and artifacts are stored in partition data sets, they can be accessed directly.

If there is a proprietary access method, run a utility that isolates or copies from the proprietary repository to a partitioned data set (PDS). Most configuration management system provide utilities to accomplish this.

After the inventory or artifacts are in a partitioned data set, use the IEBPTPCH utility, which creates a flat file of all members of the PDS. This is done in this example to transfer the flat file to the ClearCase host for migration. Figure 5-3 on page 125 is an example of the IEBPTPCH utility.

```
//JOBNAME JOB (2,3920O,PRINT-DEST),XXXXX,NOTIFY=USER,
//      MSGLEVEL=(1,1),MSGCLASS=9
//STEPOO   EXEC  PGM=IEBPTPCH
//*****************************************************************
//**   EXAMPLE OF IEBPTPCH FOR THE DUMP OF MEMBERS OF A
/**   PARTITIONED DATASET.
//**   MEMBER NAME PRINTS THEN THE MEMBER
//**   'PUNCH' PRINTS DATA IN CONTINUOUS STREAM (CARD)
//*****************************************************************
//SYSPRINT DD  SYSOUT=1
//SYSUT1   DD  DSN=DATA.VEROO.CBL.CNTL,DISP=SHR
//SYSUT2   DD  DSN=DATA.FLAT.FILE.COBOL.SRC,DISP=(,CATLG,DELETE),
//             DCB=(LRECL=81,RECFM=FB,BLKSIZE=8100),
//             SPACE=(CYL,(10,2)),UNIT=SYSDA
//SYSIN    DD    *
   PUNCH TYPORG=PO
```

Figure 5-3 Example of the IEBPTPCH utility

The next step is to transfer the flat file to the ClearCase host. Figure 5-4 is an example of a batch FTP that transfers the file to the ClearCase host system. It is an example of a batch job that will accomplish this task.

```
//JOBNAME JOB (2,3920O,PRINT-DEST),XXXXXX,NOTIFY=USER,
//      MSGLEVEL=(1,1),MSGCLASS=9
//STEPOO EXEC PGM=FTP,PARM='clearcase.host.name'(EXIT=8'
//SYSPRINT  DD  SYSOUT=*
//SYSIN     DD  *
userid pswd
put 'Data.flat.file.cobol.src' user.data.flat.file.cobol.src
quit
/*
```

Figure 5-4 FTP Batch Job for transfer to ClearCase Host

After the file is transferred to the ClearCase host, it is just a matter of running a PERL script to import the mainframe artifacts into the ClearCase repository. Figure 5-5 on page 126 through Figure 5-7 on page 128 show how to accomplish this task.

```
#!/usr/bin/perl
#
# mf_migrate.pl - migrate mainframe source code to ClearCase
#
#
# Rational software
# IBM Software Group
# 9/2/2004


# Source data from the mainframe will be copied to the ClearCase host
# as a large file containing the mainframe dataset members.  Each member
# will become a ClearCase element.  The file from the mainframe, we'll
# call it the transfer file, will be of the following form:
#
# VMEMBER NAME  FILENAME1
# V<file contents>
# VMEMBER NAME  FILENAME2
# V<file contents>
# ...
# <eof>
#
# The leading "V" on each line of the transfer file is a carriage control
# artifact from the mainframe.  It should be removed before each line is
# written to the Linux or Windows file.
#
# Each transfer file will contain only members of a particular type.  That is,
# the transfer file will contain only Cobol source files or only Cobol source
# files related to DB2, or only JCL files, etc.  All files of a particular
# type will be stored in a ClearCase directory appropriately named for the
# type of files (or mainframe members) it stores.
#
# No name changes are required or desired to the file names.  The member
# names on the mainframe will be identical to the ClearCase element names.
#
# The transfer file names will contain two digits to indicate the version
# of the members.  "00" will indicate the members that are currently in
# production on the mainframe.  "01" indicates the previous set of
# production members and so on.  Imports into ClearCase must be performed
# from the highest valued two digit number down to "00".
#
# In the future...
# The ClearCase Remote Build capability provided by the Mainframe Connector
# will be used to submit a job to the mainframe which will create the
# transfer file and copy it to the ClearCase host.
#
# There is no obvious way to relate the different versions of the source
# code on the mainframe to each other so we can't create historical
# baselines.  The only valid baseline will be "this is what is in
# production today" which is composed of the latest versions of all
# source code obtained from the mainframe.
#
```

Figure 5-5 PERL Script to import artifacts to ClearCase repository, Part 1 of 3

```
# Usage:  mf_migrate.pl [-h] [-v] -d <workdir> -t <targetdir> <file> ...
#
# This script silently assumes that it has been executed from within
# a ClearCase view context.  It will not work on Windows (path separators,
# system commands).


# get RACF id
# get RACF password
# get identification of dataset members to copy
# get number of iterations to migrate
# get target VOB tag
#
# get mapping between mainframe dataset members and ClearCase directories

use Cwd;
use Getopt::Std;

$usage = "Usage: $0 [-h] [-v] -d workdir -t targetdir file-to-migrate ...";
if (!getopts('hvd:t:') || $opt_h) {
   print "$usage\n";
   exit(0);
}

die "workdir is required\n$usage\nstopped" unless $work_dir_path = $opt_d;
die "VOB target directory is required\n$usage\nstopped" unless $VOB_target_path
= $opt_t;
$verbose = $opt_v;

$clearfsimport = "/opt/rational/clearcase/bin/clearfsimport";

$cwd = cwd();# save current working directory for use later

foreach $xfername (reverse sort @ARGV) {
   print "Processing $xfername...\n" if $verbose;
   system("rm -rf $work_dir_path");
   system("mkdir -p $work_dir_path");

   open(IN, "<${xfername}") || die "Can't open ${xfername}: $?";
   $file_count = 0;
   while (<IN>) {
      if (/^VMEMBER NAME  /) {
         s/^VMEMBER NAME  //;
         $name = $_;
         chomp($name);
         close(OUT) if ($file_count);
         open(OUT, ">${work_dir_path}/${name}") || die "Can't open
${work_dir_path}/${name}: $?";
         $file_count++;
         print "  extracting $name\n" if $verbose;
         next;
```

Figure 5-6 PERL script to import artifacts to ClearCase Repository, Part 2 of 3

```
print "Importing into ClearCase at $VOB_target_path\n" if $verbose;
   chdir($work_dir_path) || die "can't chdir to $work_dir_path: $?";
   system("$clearfsimport -recurse -rmname . $VOB_target_path");
   chdir($cwd) || die "can't chdir to $cwd: $?";
   print "Finished with $xfername\n\n" if $verbose;
}
```

Figure 5-7 PERL script to import artifacts to ClearCase Repository, Part 3 of 3

Summary

Figure 5-5 on page 126, Figure 5-6 on page 127, and Figure 5-7 is just one example of physically migrating the mainframe artifacts into a ClearCase repository. You can use additional methods, for example, from a PERL script and using FTP to the mainframe, can access the artifacts directly and migrate them into a ClearCase PDS at a time.

It is a very simple process to physically migrate the code from one repository to another. It is the organization of the artifacts that is the most important, which we discuss in the next section.

5.1.9 ClearCase repository architecture

Organization of the artifacts within the ClearCase repository must mirror how the client assigns work and develops their software. A simple example of this is if the client development teams work only on specific applications. One team works on the Finance system while another team only works on Payroll. In this example, the organization of the artifacts is that the Finance system is in one repository (VOB), while the Payroll system is in a separate repository (VOB). The separate repositories are also true if the client has security controls around there artifacts. If, for example, developers are only allowed to work on specific applications and the access to those artifacts is controlled, the ClearCase repository is organized to secure each application in its own repository (VOB).

Within the ClearCase repository, there is also an organization of the application itself, which is just a directory structure that organizes artifacts of the application. The directory structure is not unlike any file system that you can see on your desktop. Normally, within the application, if there are copybooks, include files, subroutine, and source, which are located in a directory leaf structure that identifies them as such.

The following list reflects an application structure within the repository (VOB):

FINANCE:

- COPYBOOKS:
  - ABC.CPY
  - XYZ.CPY
- JCL:
  - FIN001.JCL
  - FIN002.JCL
- INCLUDES
- SUBROUTINES
- COBOL SOURCE
- PL1 SOURCE

In the above simple application structure, the repository or application is organized to reflect the component parts of the application in an easy-to-find directory structure. In some cases, the organization of the repository is as simple as the previous example, but this is not always the case.

When determining how the applications will be migrated and stored in ClearCase, there are a few factors to consider:

- Look at the current naming conventions that store the artifacts in the current system:
  - Look at data set naming conventions to identify applications
  - Look at member naming conventions within the data sets
- Look at how the artifacts are organized in the current CM system that will be migrated:
  - Is there a project methodology?
  - Are there multiple system structures?
  - Is there a system-to-subsystem relationship?
- Determine how work is assigned to developers:
  - Are there on-going maintenance projects?
  - How many new project or application enhancements are there?

The next section contains some further examples of repository (VOB) architecture that we used in the past when we migrated to ClearCase.

Data set naming standard approach

Although this is example is very simple, this approach allows development to easily access their artifacts within ClearCase, based on their current naming convention standards.

The artifacts are organized within partitioned data sets by application category and are the programs where run, for example:

```
ACCTPAY.BATCH.SOURCE
ACCTREC.BATCH.SOURCE
ACTTPAY.CICS.SOURCE
ACCTREC.CICS.SOURCE
```

When these files are migrated to ClearCase, we first separate the applications into separate repositories (VOB), for example, ACCTPAY, ACCTREC, which retains the data set naming convention to organize the application in the same way as the data sets are organized:

ACCTPAY:

- BATCH.SOURCE:
  - PROG1.CBL
  - PROG2.CBL
  - PROG3.CBL
- CICS.SOURCE:
  - CPROG1.CBL
  - CPROG2.CBL
  - CPROG3>CBL

This organization allows developers to find elements based on the current naming convention of their data sets. The only difference is that now they are in a directory structure. Projects are at the application level within ClearCase.

System and subsystem approach

In this example, each system has subsystems that make up an entire system, and each subsystem is built to create the complete system. Subsystem artifacts are specifically related to and are owned by each system. The organization is at the System level with each subsystem being its own part of the overall system. Development teams can work on any subsystem within a System.

When migrating this example, each subsystem is a repository (VOB) with an architecture that reflects the artifacts within it, as in our previous examples:

SUBSYSTEM1:

- COPYBOOKS
- COBOL SOURCE
- DBRM
- JCL

SUBSYSTEM2:

- COPYBOOKS
- COBOL SOURCE
- DBRM
- JCL

SUBSYSTEM3:

- COPYBOOKS
- COBOL SOURCE
- DBRM
- JCL

The subsystems are ClearCase Projects, and developers make changes to different subsystems, which are eventually integrated into a system's project and built.

Project approach

This client uses a system or subsystem organization of their artifacts, although they use the System naming convention as a place holder, which means that the subsystems that belong to a system are not integrated into the overall system, which makes the system an integration point for all subsystems. Instead, subsystem changes are part of the system but are built and deployed independently of the system.

The repository architecture is at the System level, which means that the System is a single repository (VOB) with the subsystems being part of the System.

SYSTEM1

SUBSYSTEM1:

- COPYBOOKS
- COBOL SOURCE
- DBRM
- JCL

SUBSYSTEM2:

- COPYBOOKS
- COBOL SOURCE
- DBRM
- JCL

SUBSYSTEM3:

- COPYBOOKS
- COBOL SOURCE
- DBRM
- JCL

SYSTEM2

SUBSYSTEM1:

- COPYBOOKS
- COBOL SOURCE
- DBRM
- JCL

SUBSYSTEM2:

- COPYBOOKS
- COBOL SOURCE
- DBRM
- JCL

SUBSYSTEM3:

- COPYBOOKS
- COBOL SOURCE
- DBRM
- JCL

With this method, a subsystem is at the project level and development teams can select what artifacts they need to change from the entire system.

Summary

The examples in this section show how migrations and repository architectures (VOB) are done for clients. The key point is to make an effort to understand how the artifacts are stored, how development teams gain access to those artifacts, and how development makes changes to these artifacts. This knowledge allows the migration to go much smoother and is consistent with client software development processes.

5.1.10 Life cycle strategy

Determining a software life cycle design can be a very simple process when dealing with a mainframe infrastructure. The first step is to review with the client their current life cycle, and at the same time ask if there is the need for any improvements or changes that the client wants to have.

Software development life cycles can vary from client-to-client, but for our example, we use the most common that we have seen over the last few years. It is also a very simple model.

In most cases, there is a TEST, QA, EMER, and PROD environment, which can map to ClearCase stream structure. Figure 5-8 on page 132 is an example.

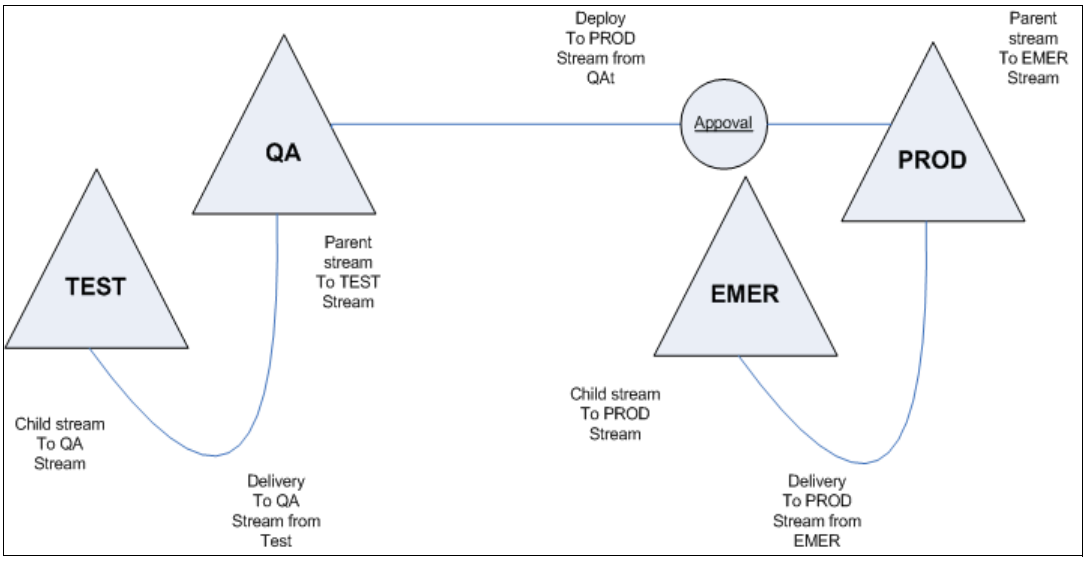


Figure 5-8 ClearCase Stream Structure to map mainframe life cycle

When analyzing the mainframe life cycle and environments to determine the proper ClearCase stream strategy to use, understand that in most cases there is a paring of environments. In this example, TEST and QA are paired, while EMER and PROD environments are paired. There can be additional environments but usually this pairing of environments is consistent.

How to pair the environments is determined by how developers move through the life cycle and the methodology behind that movement. In this example, developers initiate movement to test and build their code with their changes. After testing, the developers also move their code to the QA environment and rebuild and test the software changes. It is that developer-initiated movement and freedom to move in those environments that determines the pairing.

When a developer's changes are ready to be deployed to production, there is usually an approval process. When approvals are met, the deployment to production occurs automatically. This is also true with ClearCase and the model in Figure 5-8, where approval occurs within ClearQuest, and it is a ClearQuest Trigger that moves the deployable code onto production from the QA environment.

The EMER and PROD pairing is so that if emergency fixes are needed to an application in the the production environment, the developer can check out the problem element, make the appropriate fix, and initiate the deployment to production. There is also usually a process and controls around emergency fixes and deployment to production, but for this example, we are highlighting the paring of environments that determine the stream strategy.

When you create a stream strategy, try to determine the parent child relationship in ClearCase to the mainframe infrastructure that the physical movement of the code will occur. ClearCase logically stores and versions the code and maps the stream structure to the physical deployment on the mainframe.

5.2 Summary

In chapter one, we described the current modern business landscape and some of the true business problems that we saw with customers over recent years. We described an ESCM

solution and how it solves real business issues with regard to software configuration management and new software development architectures.

In chapter two, we drilled down deeper into the challenges that exist within the current multi platform software development environments that are part of a larger Enterprise. We also discussed the business challenges of adopting an ESCM solution and the ramifications of not adopting to an ESCM solution. Finally, we discussed in detail the benefits of adopting an ESCM solution.

In chapter three, we provided the best practices around the ESCM model, planing, and how to prepare for a collaborative software development environment. We also highlighted how and why to have all software development activity based, whether software changes are releases, major enhancements, or regular maintenance. Last, we discussed the impact of software development occurring in parallel development and the importance of workflow for gathering metrics to measure software quality.

In chapter four, we provided technical discussions about “how to” work with the different components of the IBM Rational ESCM solution. We discussed details of the integrations and how they all work together to form a collaborative software development environment for all platforms under a single, unified software development process. We gave many examples to ensure success in adopting an ESCM model.

In chapter five, we gave you examples of how to start and structure an ESCM project, along with real samples. Use the contents of this chapter as a guide through an ESCM project. The examples that we gave are from real implementations of ECSM projects in practice. Each client has unique needs, as it relates to their software development processes and how they manage projects, but you can customize these examples to those different models. In addition to project structure and tasks of an ESCM project, we provided a general discussion around the details of data migration, architecture of artifacts, projects, and life cycle.

Through the information in this book, we hope that adopting an ESCM solution now seems less complicated and helps you with project planning and implementing the solution.

In closing an ESCM solution offers:

- Cross-platform process management
- Cross-platform asset repository
- Heterogeneous build support
- Centralized build and release process
- Access to artifacts anytime from anywhere

A

Source script for the build engine

Example A-1 is the full source of a working prototype PERL script, which implements the build engine.

Example: A-1 Source for the Build Engine Script

```
#!$CLEARCASE/bin/ccperl.exe
#
# Standalone z/OS build helper script which can also be called from a ClearQuest
stage hook.
# Based on original work by Brandt Onorato, Stephen Seifert, Dale Newby, all of
IBM
#
# David Lawrence, IBM, June 2007
#
use strict;
use File::Basename;
use Getopt::Std;
use File::Copy;
use POSIX qw(strftime);
use lib "C:\\Program Files\Rational\commmon\lib";
#use CQPerlExt;

my $debug        = 0;   # Set to 1 for verbose debugging output
my $rccbuild     = 'rccbuild.exe';
my $perl_cmd     = "ccperl";
my $myatt; #for retrieving attribute values from CAL
my $myattname; #for retrieving attribute values from CAL
my $attname;
my $i; #loop counter
#
# Element name (member) is passed in from the calling hook
#
```

```
my %options;
my ($ccpn) = $ARGV[0];
my ($stage) = $ARGV[1];
my ($system) = ARGV[2];
my ($subsys) = ARGV[3];

#
my ($vdrive) = "v:"; # NOTE: Change this to reflect your local MVFS mount point.
$ccpn =~ s/\$(\w+)/$1/g;
#
# First retrieve the Top level ClearCase Application object
# Then retrieve the element.
# Process the version name to get just member name and extension for z/OS.
# element passed in from the hook and filter the name accordingly
#
my $CCApp= Win32::OLE->new ("ClearCase.Application") or die "Can't connect to
ClearCase object $!";
my $version = $CCApp->Element($ccpn) or die "Failed to retrieve element";
my $parent = $version->Parent;
my $filepath = $version->Path;
my $membernamebase = uc(basename($filepath));
my @leafname = split (/\./, $membernamebase);
my $parentdir = $parent->Path;
my $name         = uc( $ENV{'USERNAME'} );
my $mem          = uc( @leafname[@leafname-2]);
#
my $logdir       = $parentdir;
$logdir          = "$logdir/$name";
my $rccVerbosity = "-P $logdir/$mem.";
$rccVerbosity    = "$rccVerbosity -V -V -V" if $debug;
my $sysout       = 'RCCEXT=RCCOUT,DISP=(NEW,CATLG,DELETE),SPACE=(32000,(30,30))';
$sysout          = 'RCCEXT=RCCSTD,DISP=(NEW,CATLG,DELETE),SPACE=(32000,(30,30))'
if $debug;

cleanup();

my ( $env, $db2sys, $menv, $os390_hostname, $os390_port, $stage ) =
setEnvironmentVar();
my $jcl = "$mem.jcl";
my $tempDSN = "$env.$name.$mem";
my $OID     = '40 char CC Obj ID for use in load module';
my $ncpycnt = 7;    # number of CICS NewCopy steps
my $lcnt    = 0;    # loop counter
my $scnt    = 10;   # step counter

my $DLL = 'F';
if ( $mem =~ /^\w{4}[S]\w{2,3}$/ ) {    # 5th Char = 'S' (Subroutine)
    $DLL = 'U';
}

my $operation;   # from perspective of mainframe
my $pcFile;
my $indsn;
my $outdsn;
my $rccext;
```

```
my $binary;
my $loadlib;


my %CCattr = getCCattrs();

if ($stage =~ m/IT|ST|MIR1|PROD/) {
print "stage is $action \n";
     die 'Deletion error.'                  if ( s1delete() != 0 );
     die 'Allocation error.'                if ( s1alloc() != 0 );
     $operation = 'RECEIVE';
     $pcFile    = $ccpn;
     if ($ccpn =~ /\.cbl$/) {
        $outdsn = "$env.DSCC$stage.ALL.SOURCE";
        $rccext = 'CBL';
        $binary = 't';
        die 'Transfer error.'                  if ( Transfer() != 0 );
     }
     elsif ($ccpn =~ /\.DBM$/) {
        $CCattr{'DB2'} = 'Y';
        $outdsn = "$env.DSCC$stage.ALL.DBRMLIB";
        $rccext = 'DBM';
        $binary = 'b';
        die 'Transfer error.'                  if ( Transfer() != 0 );
        die 'Error generating Bind Statements' if ( GenBind() != 0 );
        die 'Bind Error'                       if ( BindProg() != 0 );
     }
     elsif ($ccpn =~ /\.DBG$/) {
        $outdsn = "$env.DSCC$stage.ALL.SYSDEBUG";
        $rccext = 'DBG';
        $binary = 'b';
        die 'Transfer error.'                  if ( Transfer() != 0 );
     }
     elsif ($ccpn =~ /\.LST$/) {
        $outdsn = "$env.DSCC$stage.ALL.LIST";
        $rccext = 'LST';
        $binary = 't';
        die 'Transfer error.'                  if ( Transfer() != 0 );
     }
     elsif ($ccpn =~ /\.BT$/) {
        $indsn   = "$tempDSN.BTLOAD";
        $outdsn  = "$env.DSCC$stage.ALL.BATCH.LOAD";
        $rccext  = 'BT';
        $binary  = 'b';
        $loadlib = 'l';
        die 'Transfer error.'                  if ( Transfer() != 0 );
     }
     elsif ($ccpn =~ /\.CS$/) {
        $CCattr{'CS'} = 'Y';
        $indsn   = "$tempDSN.CSLOAD";
        $outdsn  = "$env.DSCC$stage.ALL.CICS.LOAD";
        $rccext  = 'CS';
        $binary  = 'b';
        $loadlib = 'l';
        die 'Transfer error.'                  if ( Transfer() != 0 );
```

```
        die 'Error during CICS newcopy'        if ( cicsNewcopy() != 0 );
     }
     elsif ($ccpn =~ /\.SP$/) {
        $CCattr{'SP'} = 'Y';
        $indsn   = "$tempDSN.SPLOAD";
        $outdsn  = "$env.DSCC$stage.ALL.DB2.LOAD";
        $rccext  = 'SP';
        $binary  = 'b';
        $loadlib = 'l';
        die 'Transfer error.'                  if ( Transfer() != 0 );
        die 'Error during SP copy'             if ( storedProcCopy() > 4 );
     }
   s1delete() if (! $debug);
}
else {
   if  ( $ccpn =~ /\.cbl$/ ) {
      die 'Deletion error.'                  if ( s1delete() != 0 );
      die 'Allocation error.'                if ( s1alloc() != 0 );
   }
  %CCattr = genCCattrs() if (! %CCattr );

# copy source to mainframe
      $operation = 'RECEIVE';
      $outdsn    = "$env.DSCC$stage.ALL.SOURCE";
      $pcFile    = $filepath;
      $rccext    = 'CBL';
      $binary    = 't';
      die 'Transfer error.'                  if ( Transfer() != 0 );

# copy derived objects to pc
# CO/CI of derived objects always done in post-op (use workarea $logdir)
      $operation = 'XMIT';

      print "CICS Attribute is $CCattr{'CICS'} \n";
      if ( $CCattr{'CICS'} eq 'Y' ) {
              print "CICS XMIT, ATTR is $CCattr{'CS'} \n";
          die 'Translation error.'           if ( CicsTranslate() > 4 );
      }
      if ( $CCattr{'DB2'} eq 'Y' ) {
         die 'Precompile error.'             if ( DB2Precompile() > 4 );
         $outdsn = "$env.DSCC$stage.ALL.DBRMLIB";
         $pcFile = "$logdir\\$mem.DBM";
         $rccext = 'DBM';
         $binary = 'b';
         die 'Transfer error.'               if ( Transfer() != 0 );
      }
      die 'Compile error.'                   if ( CobCompile() > 4 );
      $outdsn = "$env.DSCC$stage.ALL.SYSDEBUG";
      $pcFile = "$logdir\\$mem.DBG";
      $rccext = 'DBG';
      $binary = 'b';
      die 'Transfer error.'                  if ( Transfer() != 0 );
      $outdsn = "$env.DSCC$stage.ALL.LIST";
      $pcFile = "$logdir\\$mem.LST";
      $rccext = 'LST';
```

```
      $binary = 't';
      die 'Transfer error.'                  if ( Transfer() != 0 );
      $outdsn = "$env.DSCC$stage.ALL.SIDEDECK";
      $pcFile = "$logdir\\$mem.SDK";
      $rccext = 'SDK';
      $binary = 't';
      die 'Transfer error.'                  if ( Transfer() != 0 );
      die 'Prelink Error.'                   if ( Prelink() > 4 );
      die 'Link Error.'                      if ( LinkProg() != 0 );
      if ( $CCattr{'DB2'} eq 'Y' ) {
         die 'Error generating Bind Statements' if ( GenBind() != 0 );
         die 'Bind Error'                       if ( BindProg() != 0 );
      }
      if ( $CCattr{SP} eq 'Y' ) {
         die 'Error during SP copy'          if ( storedProcCopy() != 0 );
      }
      if ( $CCattr{CS} eq 'Y' ) {
         die 'Error during CICS newcopy'     if ( cicsNewcopy() != 0 );
      }
   s1delete() if (! $debug);
}


exit(0);

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< writeFile >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub writeFile {
    my ( $outputfilename, $text ) = @_;

    print "OUTPUTFILE: $outputfilename \n" if $debug;
    open( FOUT, ">> $outputfilename" ) or die 'Unable to open output file';

    my $n = length($text);
    my $first = substr( $text, 0, 1 );
    my $last;
    if ( $n > 2 ) {
        $last = substr( $text, $n - 2, 2 );
        if ( $last eq "\n\n" ) {
            chop($text);
        }
    }
    if ( $first eq "\n" ) {
        $text = substr( $text, 1 );
    }
    print $text if $debug;
    print FOUT $text;

    close(FOUT);
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< getCCattrs >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub getCCattrs {
#
# Use CAL to retrieve Attributes
#
```

```
#my $elt = $version->Element;
my $attlist = $version->Attributes;
my $attcount = $attlist->Count;

for ($i=1; $i<=$attcount; $i++) {
   $myatt = $attlist->Item($i);
   $myattname = $myatt->Type->Name;
   $CCattr{$myattname} = $myatt->Value;
 }

    return %CCattr;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< genCCattrs >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub genCCattrs {
   my %CCattr;

   $CCattr{'DB2'}  = 'N';
   $CCattr{'CICS'} = 'N';
   $CCattr{'FLOW'} = 'N';
   $CCattr{'MQ'}   = 'N';
   $CCattr{'SP'}   = 'N';
   $CCattr{'CS'}   = 'N';
   $CCattr{'BT'}   = 'N';

   if ( substr( $mem, 5, 1 ) eq 'T' ) {
      $CCattr{'DB2'} = 'Y';
      $CCattr{'SP'}  = 'Y';
   }

my $version = $CCApp->Version($ccpn) or die "Failed to retrieve VOB element";
   open (IN, "$version") || die("Error Reading File: $ccpn $!");
   undef $/;
   my $infile = <IN>;
   close (IN) || die("Error Closing File: $ccpn $!");

   if ( $infile =~ / SQL / ) {
      $CCattr{'DB2'} = 'Y';
   }

   if ( $infile =~ / CICS / ) {
      $CCattr{'CICS'} = 'Y';
      $CCattr{'CS'}   = 'Y';
   }

   if ( $infile =~ /(SCRCODE|WFCODE)/ ) {
      $CCattr{'CICS'} = 'Y';
      $CCattr{'FLOW'} = 'Y';
   }

   if ( $infile =~ /MQOPEN/ ) {
      $CCattr{'MQ'} = 'Y';
   }

   if ( ($CCattr{'SP'} eq 'N') && ($CCattr{'CS'} eq 'N') ) {
```

```
      $CCattr{'BT'} = 'Y';
   }

#
# Now set the attributes from the hash created above
#
my $elt = $version->Element;
my $eltype = $elt->ElementType;
my $attlist = $elt->Attributes;
my $attcount = $attlist->Count;

for ($i=1; $i <=$attcount; $i++) {
   $myatt= $attlist->Item($i);
   $myattname = $myatt->Type->Name;
   $myatt->SetValue($CCattr{$myattname});
  }

   return %CCattr;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< setEnvironmentVar
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub setEnvironmentVar {
    my @env;    # $env, $db2sys, $menv, $os390_hostname, $os390_port, $stage
    if ($stage =~ /DEV/i ) {
   printf ("stage was - $stage\n") if $debug;
        @env = ( 'TEST', 'DBT1', 'KCT', '83.255.29.22', '3606', 'DEV' );
    }
    elsif ($stage =~ /IT/i) {
   printf ("stage was - $stage\n") if $debug;
        @env = ( 'CCA', 'DBD1', 'KCA', '83.255.29.22', '10003', 'IT' );
    }
    elsif ($stage =~ /ST/i) {
   printf ("stage was - $stage \n") if $debug;
        @env = ( 'CCS', 'DBD1', 'KCS', '83.255.29.22', '10003', 'ST' );
    }
    elsif ($stage =~ /REL/i) {
   printf ("stage was - $stage\n") if $debug;
        @env = ( 'CCE', 'DBP1', 'KCE', '83.255.29.22', '10003', 'REL' );
    }
    elsif ($stage =~ /MIR1/i) {
   printf ("stage was - $stage\n") if $debug;
        @env = ( 'CCP', 'DBP1', 'KCP', '83.255.29.22', '10003', 'MIR1' );
    )
    elsif ($stage =~ /PROD/i) {
   printf ("stage was - $stage\n") if $debug;
        @env = ( 'CCP', 'DBP1', 'KCP', '83.255.29.22', '10003', 'PROD' );

    }
    else {
        printf ("Unknown  stage - $stage\n") if $debug;
        exit -1;
    }

    printf( "Environment name (%s)\n", @env ) if $debug;
```

```
    return @env;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< cleanup >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub cleanup {
    mkdir $logdir;
    unlink <$logdir\\$mem.*>;
    unlink 'rccbuild.log',$jcl;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< s1delete >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub s1delete {
    writeFile(
        $jcl, "
//* ---------------------------------------------------------------------
//DELETE   EXEC PGM=IDCAMS
//*        ----
//*        ---- IDCAMS - Delete Temporary Files
//*        ----
//SYSPRINT DD   $sysout
//SYSIN    DD   *
 DELETE $tempDSN.* NONVSAM
 ;
 IF LASTCC LE 8 THEN SET MAXCC EQ 0
/*
//* ---------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b S1DELETE -ft $jcl
-n 0 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< s1alloc >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub s1alloc {
    writeFile(
        $jcl, "
//* ---------------------------------------------------------------------
//S1ALLOC  EXEC PGM=IEFBR14
//*        ----
//*        ---- IEFBR14 - Allocate Temporary Files
//*        ----
//TEMP1    DD  DSN=$tempDSN.DSNHOUT,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(800,(500,500))
//TEMP2    DD  DSN=$tempDSN.LINKPARM,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(400,(20,20),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
//TEMP3    DD  DSN=$tempDSN.PLKSET,
//             DISP=(NEW,CATLG,DELETE),
```

```
//             UNIT=SYSDA,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//TEMP4    DD  DSN=$tempDSN.LOADSET,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(800,(1500,1500),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//TEMP5    DD  DSN=$tempDSN.ELMOUT,RCCEXT=CBL,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(6160,(100,100),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//TEMP6    DD  DSN=$tempDSN.LNKLIST,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=3390,SPACE=(3630,500,500),RLSE),
//             DCB=(RECFM=FB,LRECL=121,BLKSIZE=3630)
//TEMP7    DD  DSN=$tempDSN.COBLIST,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=3390,SPACE=(6171,7000,7000),RLSE),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//TEMP8    DD  DSN=$tempDSN.PCLIST,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=3390,SPACE=(6171,(500,500),RLSE),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//TEMP9    DD  DSN=$tempDSN.LOADNAME,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(CYL,(5,5,5)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b S1ALLOC -ft $jcl -n
0 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< DB2Precompile >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub DB2Precompile {
    writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//PRECOMP  EXEC PGM=DSNHPC,REGION=4096K,
//  PARM=('HOST(IBMCOB),APOST')
//*        ----
//*        ---- DSNHPC - Precompile
//*        ----
//STEPLIB  DD  DISP=SHR,DSN=SYS1.DSN710.SDSNEXIT
//         DD  DISP=SHR,DSN=DSN710.SDSNLOAD
//SYSLIB   DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.&C1STAGE.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.&C1ST2.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.ST.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.REL.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.MIR1.BCOPY
```

```
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.PROD.BCOPY
"
    );
    if ( $CCattr{'STG1'} eq 'DEV' ) or ( $CCattr{'STG1'} eq 'IT' {
        writeFile(
            $jcl, "
//DBRMLIB  DD  DISP=SHR,DSN=ENDV.&C1SYSTEM.&C1SUBSYS.&C1STAGE.BDBRM
"
            );
        }
    if ( $CCattr{'STG1'} eq 'ST' ) or ( $CCattr{'STG1'} eq 'REL' {
        writeFile(
            $jcl, "
//DBRMLIB  DD  DISP=SHR,DSN=ENDV.NOVA.&C1SUBSYS.&C1STAGE.BDBRM
"
            );
        }
        writeFile(
            $jcl, "
//SYSIN    DD  DISP=SHR,DSN=$tempDSN.ELMOUT($mem)
"
        );
    }

    writeFile(
        $jcl, "
//SYSCIN   DD  DISP=SHR,DSN=$tempDSN.DSNHOUT
//SYSPRINT DD  $sysout
//SYSTERM  DD  $sysout
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b PREDB2 -ft $jcl -n
4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< CobCompile >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub CobCompile {
    writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//COMPILE  EXEC PGM=IGYCRCTL,REGION=4096K,
//  PARM=('LIB,APOST,SOURCE,LIST,TEST,RMODE(AMY),MAP')
"
    );

    writeFile(
```

```
        $jcl, "
//*        ----
//*        ---- Compile - COBOL compile
//*        ----
//STEPLIB  DD  DISP=SHR,DSN=IGY.SIGYCOMP
//SYSIN    DD  DISP=SHR,DSN=$tempDSN.DSNHOUT
"
    );
    writeFile(
        $jcl, "
//SYSLIN   DD  DISP=SHR,DSN=$tempDSN.LOADSET($mem)
//SYSLIB   DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.&C1STAGE.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.&C1STAGE.&C1ST2.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.ST.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.REL.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.MIR1.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.PROD.BCOPY
"
    );

    writeFile(
        $jcl, "
//SYSPRINT DD  DISP=SHR,DSN=$tempdsn.COBLIST,
//SYSTERM  DD  $sysout
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT3   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT4   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT5   DD  UNIT=SYSDA,SPACE=(3000,(500,500),,,ROUND)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT7   DD  UNIT=SYSDA,SPACE=(3000,(500,500),,,ROUND)
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b COMPILE -ft $jcl -n
4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}


# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< Prelink >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub Prelink {
    writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//PLKED    EXEC PGM=EDCPRLK,REGION=2048K
//STEPLIB  DD DSN=CEE.SCEERUN,DISP=SHR
"
    writeFile(
        $jcl, "
//*        ----
```

```
//*        ---- EDCPRLK - Prelink
//*        ----
//SYSIN    DD  DISP=SHR,DSN=$tempDSN.LOADSET($mem)
//         DD  DISP=SHR,DSN=CCT.DSCC.ALL.SIDEDECK($mem)
"
    );

    writeFile(
        $jcl, "
//SYSMSGS  DD  DISP=SHR,DSN=CEE.SCEEMSGP(EDCPMSGE)
//SYSLIB   DD  DUMMY
//SYSDEFSD DD  DUMMY
//SYSMOD   DD  DISP=SHR,DSN=$tempDSN.PLKSET
//SYSPRINT DD  DSN=$tempDSN.LNKLIST,DISP=SHR
//SYSOUT   DD  $sysout
//SYSTERM  DD  SYSOUT=*
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b PRELINK -ft $jcl -n
4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< LinkProg >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub LinkProg {
    my $status = 0;

    $OID = `cleartool lshistory -last 1 -fmt \"%1.400n\" $ccpn\@\@`;
    printf( "xtendedName (%s)\n", $OID ) if $debug;

    if ( $CCattr{'SP'} eq 'Y' ) {
       die 'LinkSP Error.'             if ( linkSP() != 0 );
       $indsn   = "$env.DSCC$stage.ALL.DB2.LOAD";
       $outdsn  = "$tempDSN.SPLOAD";
       $pcFile  = "$logdir\\$mem.SP";
       $rccext  = 'SP';
       $binary  = 'b';
       $loadlib = 'l';
       die 'Transfer error.'           if ( Transfer() != 0 );
    }
    if ( $CCattr{'BT'} eq 'Y' ) {
       die 'LinkBT Error.'             if ( linkBT() != 0 );
       $indsn   = "$env.DSCC$stage.ALL.BATCH.LOAD";
       $outdsn  = "$tempDSN.BTLOAD";
       $pcFile  = "$logdir\\$mem.BT";
       $rccext  = 'BT';
       $binary  = 'b';
       $loadlib = 'l';
       die 'Transfer error.'           if ( Transfer() != 0 );
    }
```

```
    if ( $CCattr{'CS'} eq 'Y' ) {
       die 'LinkCS Error.'             if ( linkCS() != 0 );
       $indsn   = "$env.DSCC$stage.ALL.CICS.LOAD";
       $outdsn  = "$tempDSN.CSLOAD";
       $pcFile  = "$logdir\\$mem.CS";
       $rccext  = 'CS';
       $binary  = 'b';
       $loadlib = 'l';
       die 'Transfer error.'           if ( Transfer() != 0 );
    }
    $loadlib = '';

    return $status;
}

#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< buildLoadname >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub buildLoadname {
    my $status = 0;

    $jcl = 'devprint.jcl';

    cofkymod::writeFile( $jcl, $compile_1, "
//* ---------------------------------------------------------------------
//LDNAME   EXEC PGM=IEBGENER
//*        ----
//*        ---- IEBGENER - Create Loadname entry
//*        ----
//SYSUT1   DD  *,DCB=BLKSIZE=80
  INCLUDE SYSLIB(SBCTDBC)
  NAME $mem(R)
/*
//SYSUT2   DD  DISP=SHR,DSN=$tempDSN.LOADNAME($mem)
//SYSPRINT DD  $sysout
//SYSIN    DD  DUMMY
//* ---------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b buildLoadname -ft
$jcl -n 4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}


# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< linkBT >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub linkBT {
    writeFile(
        $jcl, "
//* ---------------------------------------------------------------------
//LKEDBT   EXEC PGM=IEWL,
//    PARM='MAP,LIST,AMODE(31),RMODE(ANY)'
//*        ----
```

```
//*        ---- IEWL - Linkedit for Batch
//*        ----        USE DSNELI FOR TSO BATCH
//*        ----          , DSNRLI FOR SP
//*        ----          , DSNCLI FOR CICS DB2
//*        ----
//SYSLIB   DD  DISP=SHR,DSN=CEE.SCEELKED
//         DD  DISP=SHR,DSN=DSN710.SDSNLOAD
//         DD  DISP=SHR,DSN=ISP.SISPLOAD
//         DD  DISP=SHR,DSN=ENDV.NOVA.BASE.PROD.OLOAD
//         DD  DISP=SHR,DSN=ENDV.&C1SYSTEM.&C1SUBSYS.&C1STAGE.BLOAD
"
    );
    writeFile(
        $jcl, "
//SYSLIN   DD  DISP=SHR,DSN=$tempDSN.PLKSET
//         DD  DISP=SHR,DSN=$tempDSN.LOADNAME($mem)
"
    );
    writeFile(
        $jcl, "
//SYSLMOD  DD  DISP=SHR,DSN=ENDV.&C1SYSTEM.&C1SUBSYS.&C1STAGE.BLOAD($mem)
//SYSPRINT DD  DISP=SHR,DSN=$tempDSN.LNKLIST
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(50,50))
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b LINKBATCH -ft $jcl
-n 4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< linkCS >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub linkCS {
    writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//LKEDCS   EXEC PGM=IEWL,
//    PARM='RENT,MAP,AMODE(31),RMODE(ANY)'
//*        ----
//*        ---- IEWL - Linkedit for CICS
//*        ----        USE DSNELI FOR TSO BATCH
//*        ----          , DSNRLI FOR SP
//*        ----          , DSNCLI FOR CICS DB2
//*        ----
//SYSLIB   DD  DISP=SHR,DSN=SYS1.LINKLIB
//         DD  DISP=SHR,DSN=$env.DSCC.ALL.CICS.LOAD
//         DD  DISP=SHR,DSN=$env.DSCC.ALL.BATCH.LOAD
//         DD  DISP=SHR,DSN=MCS.$menv.CICS.LOAD
//         DD  DISP=SHR,DSN=MCS.$menv.BATCH.LOAD
//         DD  DISP=SHR,DSN=SYS1.CEE.SCEELKED
"
```

```
    );

    if ( $CCattr{'MQ'} eq 'Y' ) {
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=SYS2.MQSERIES.SCSQLOAD
"
        );
    }

    writeFile(
        $jcl, "
//         DD  DISP=SHR,DSN=SYS2.CICSTS13.SDFHLOAD
//*        DD  DISP=SHR,DSN=SYS2.CICSTS13.SDFHEXCI
//         DD  DISP=SHR,DSN=SYS1.CEE.SCEECICS
//         DD  DISP=SHR,DSN=SYS2.DEBUG.SEQAMOD
"
    );

    if ( $CCattr{'FLOW'} eq 'Y' ) {
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=SYS2.TST321G3.SFMCLOAD
//         DD  DISP=SHR,DSN=SYS2.MQWF321.UTILITY.SFMCLOAD
//         DD  DISP=SHR,DSN=SYSRUN.MQWF321.SFMCLOAD
"
        );
    }

    if ( $CCattr{'DB2'} eq 'Y' ) {
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=DB2.$db2sys.SDSNEXIT
//         DD  DISP=SHR,DSN=DB2.$db2sys.SDSNLOAD
"
        );
    }

    if ( $CCattr{'CICS'} eq 'N' ) {
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=SYS.PROD.LOAD
"
        );
    }

    writeFile(
        $jcl, "
//SYSLIN   DD  DISP=SHR,DSN=$tempDSN.PLKSET
"
    );

    if ( $CCattr{'DB2'} eq 'Y' ) {
        writeFile(
            $jcl, "
```

```
//         DD  DISP=SHR,DSN=SYS2.TCKCT.SYSEXEC(INDB2CIC)
"
        );
    }

    writeFile(
        $jcl, "
//         DD  DISP=SHR,DSN=SYS2.TCKCT.SYSEXEC(INCICSCM)
//         DD  DISP=SHR,DSN=SYS2.TCKCT.SYSEXEC(INCICSDB)
"
    );

    if ( $CCattr{'MQ'} eq 'Y' ) {
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=SYS2.TCKCT.SYSEXEC(INMQCICS)
"
        );
    }

    writeFile(
        $jcl, "
//         DD  *,DCB=BLKSIZE=80
 IDENTIFY $mem('$OID')
 NAME $mem(R)
/*
//SYSLMOD  DD  DISP=SHR,DSN=$env.DSCC$stage.ALL.CICS.LOAD
//SYSPRINT DD  $sysout
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(50,50))
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b LNKCICS -ft $jcl -n
4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< Transfer >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub Transfer {
   die "Unsupported operation" if (! ( $operation =~ /(XMIT|RECEIVE)/ ));
   my $cmd;

   if ($operation eq 'RECEIVE') {
      writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//XFERO$scnt EXEC PGM=IEFBR14
//*        ----
//*        ---- Clearcase upload -i
//*        ----
"
);
```

```
      if ($loadlib) {
         writeFile(
           $jcl, "
//DD1      DD  DISP=SHR,DSN=$indsn,RCCEXT=$rccext
//* ------------------------------------------------------------------
"
);
      }
      else {
         writeFile(
           $jcl, "
//DD1      DD  DISP=SHR,DSN=$outdsn,RCCEXT=$rccext
//* ------------------------------------------------------------------
"
);
      }
      $cmd =
"$rccbuild -h $os390_hostname\@$os390_port -b XFER -ft $jcl -i$binary $pcFile -n 4
-c LE $rccVerbosity";
   }

   if ($loadlib) {
      writeFile(
           $jcl, "
//* ------------------------------------------------------------------
//XFER1$scnt EXEC PGM=IKJEFT01
//*        ----
//*        ---- XMIT / RECEIVE
//*        ----
//INDD     DD  DISP=SHR,DSN=$indsn
//OUTDD    DD  DISP=SHR,DSN=$outdsn
//SYSPRINT DD  $sysout
//SYSTSPRT DD  $sysout
//SYSTSIN  DD  *
"
      );
      if ($operation eq 'XMIT') {
         writeFile(
           $jcl, "
 XMIT A.A DDNAME(INDD) MEMBER($mem) OUTDDNAME(OUTDD) NOLOG
"
         );
      }
      else {
         writeFile(
           $jcl, "
 RECEIVE INDDNAME(INDD) LOGDATASET($tempDSN.XFER.LOG)
"
         );
      }

      writeFile(
           $jcl, "
/*
//* ------------------------------------------------------------------
```

```
"
      );
   }

   if ($operation eq 'XMIT') {
      writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//XFER9$scnt   EXEC PGM=IEFBR14
//*        ----
//*        ---- Clearcase download -o
//*        ----
"
      );
      if ($loadlib) {
         writeFile(
           $jcl, "
//DD1      DD  DISP=SHR,DSN=$outdsn,RCCEXT=$rccext
"
         );
      }
      else {
         writeFile(
           $jcl, "
//DD1      DD  DISP=SHR,DSN=$outdsn($mem),RCCEXT=$rccext
"
         );
      }
      writeFile(
        $jcl, "
//* ------------------------------------------------------------------
"
      );
      $cmd = "$rccbuild -h $os390_hostname\@$os390_port -b XFER -ft $jcl -o$binary
$pcFile -n 4 -c LE $rccVerbosity";
   }
   my $status = system($cmd);
   printf( "Status was (%d)\n", $status ) if $debug;
   move( $jcl, "$logdir/$jcl.$scnt" );
   $scnt++;
   return $status;
}
```

Related publications

The publications that we list in this section are considered particularly suitable for a more detailed discussion of the topics that we cover in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 154. Note that some of the documents referenced here may be available in softcopy only.

- *Using WebSphere Studio Asset analyzer,* SG24-6065
- *Collaborative Application Lifecycle Management with Rational Products,* SG24-7622
- *Software Configuration Management: A Clear Case for IBM Rational ClearCase and ClearQuest UCM,* SG24-6399
- *IBM IT Governance Approach: Business Performance through IT Execution,* SG24-7517
- *Rational Business Driven Development for Compliance,* SG24-7244
- *Strategic Reuse with Asset-Based Development,* SG24-7529

Online resources

These Web sites are also relevant as further information sources:

- Using WebSphere Studio Asset Analyzer to find mainframe Web services

  http://www.ibm.com/developerworks/websphere/library/techarticles/0808_miller/0808_miller.html
- WebSphere Studio Asset Analyzer

  http://www.ibm.com/software/awdtools/wsaa
- The power of Unified Change Management

  http://www.ibm.com/developerworks/rational/library/701.html
- Unified Change Management from Rational Software: An Activity-Based Process for Managing Change

  http://www.ibm.com/developerworks/rational/library/2057.html
- Modernizing enterprise application development with integrated change, build and release management.

  http://www-07.ibm.com/my/events/baitsolution/download/RAW14001-USEN-00.pdf
- developerWorks Interviews: Build-and-release best practices

  http://www.ibm.com/developerworks/podcast/dwi/cm-int111307txt.html
- Rational ClearCase

  http://www-01.ibm.com/software/awdtools/clearcase/
- Rational ClearQuest

  http://www-01.ibm.com/software/awdtools/clearquest/

- Rational Build Forge

  http://www-01.ibm.com/software/awdtools/buildforge/
- Rational Developer for System z

  http://www-01.ibm.com/software/awdtools/rdz/

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Abbreviations and acronyms

BCL	Build Control Language
BF	IBM Rational BuildForge
CC	IBM Rational ClearCase
CCRC	IBM RAtional ClearCase Remote Client
CM	Configuration Management
CQ	IBM Rational ClearQuest
CQWeb	IBM Rational ClearQuest Web Client
CRM	Change and Release Management
ESCM	Enterprise Software Configuration Management
IDE	Integrated Development Environment
RDZ	IBM Rational Developer for z
SCM	Software Configuration Management
UCM	Unified Change Management
WSAA	IBM WebSphere Studio Asset Analyzer

Index

F

G

H

I

L

M

O

P

R

S

T

U

W

Z

(0.2”spine)
0.17”<->0.473”
90<->249 pages

Enterprise Software Configuration Management Solutions for Distributed and System z

Application lifecycle management for enterprise software development

Unified process for enterprise software development

Best practices for enterprise software configuration management deployment

In this IBM Redbooks publication, we describe how you can implement an application lifecycle management solution with Rational Developer for System z, Rational ClearQuest, Rational ClearCase with the z/OS extensions, and Rational Build Forge.

We present an approach called Enterprise Software Configuration Management (ESCM) and discuss the problems with releasing cross-platform applications in a secure, robust, and reversible manner.

In chapter 1, we explain the rationale for producing this book and present a few case studies.

In chapter 2, we describe the challenges that we observed in many IT enterprises as they struggle to integrate the distributed and mainframe sides of the house, which includes the cost and benefits of implementing an ESCM process and the risk of not doing so.

In chapter 3, we describe the idealized ESCM model, specifically, what a successful ESCM looks like.

In chapter 4, we present a detailed implementation with several alternative strategies, which we use in this book, in the context of an overall ESCM life cycle, where Rational Developer for System z is positioned as the primary developer interface.

In chapter 5, we discuss what to consider as you begin implementing an ESCM solution and prepare you to structure your teams, assign responsibilities, and plan for general tasks and activities.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7598-00 ISBN 0738431885