Z Open Development PoT

Proof of Technology

DevOps Solution for Z Development

Mark Boyd
mboyd@us.ibm.com

Regi Barosa
rbarosa@us.ibm.com

Mark Garcia
msgarcia@us.ibm.com

Ronald Geraghty
Ronald.Geraghty@ibm.com

Timothy Han
timothy.han@ibm.com

December 2019

Audience

- This Proof of Technology is targeted to, but not limited to Architects, Technical Specialists or Developers
- Non-technical attendees with an understanding of application lifecycle management are welcome

Pre-requisites

- Familiarity with z/OS (MVS) concepts
- Awareness (not necessarily proficiency) of basic z/OS System Programming tasks
- Basic familiarity with JCL concepts (JCL skills NOT required)
- No Java skills are required
- COBOL, PL/I and 4GL programmers are welcome

Proof of Technology Objectives

This Proof of Technology (PoT) will demonstrate some of the IBM DevOps Solution for z Systems Development:

- Introduces DevOps Tools for z/OS:
  - Application Discovery (AD)
  - Application Delivery Foundation for z Systems (ADFz – IDz + PDTools)
  - UrbanCode Deploy (UCD)
  - z Systems Development and Test Environment z (zD&T – formerly RD&T)
- Provides basic skills and introductory hands-on exposure to AD, ADFz and UCD
- No prerequisite programming skills are required
- The lab exercises are structured to provide a guided walk-through of tools capabilities

Agenda

9:00 Introductions
9:20 DevOps for Z

10:00 Break

10:15 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z

11:45 Lunch

12:30 zUnit Overview

12:40 Demo: Scenario using Z DevOps solutions including zUnit, Git and Jenkins

1:30 z/OS DevOps Testing Automation & Virtualization

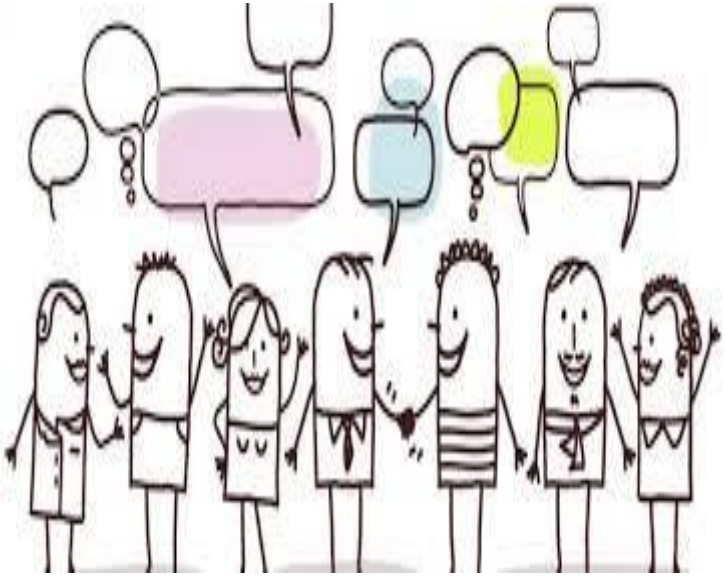
2:00 – 4:30 Choose one Optional lab:

- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6 : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → LAB 2C Application Discovery : Find a candidate API from Catalog Manager Application
- → LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- → LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
- → LAB 8 - Using Application Performance Analyzer (APA)

Lecture Labs Breaks

Introductions

- Introduce yourself
- Name and organization
- Current integration technologies/tools in use
- Experience with Eclipse base products?



What do you need to get out of attending this PoT workshop?

Mainframe DevOps

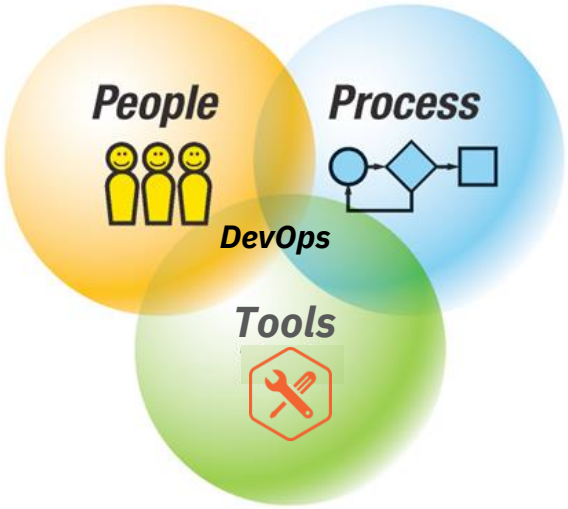
Z DevOps → The Technical Value Proposition is Strong

ENTERPRISE WIDE STANDARDIZATION

Platform neutral, continuous integration and deployment pipeline across the enterprise. Uniform enterprise wide agile delivery processes and standards facilitates transferable skills

MODERN DEVELOPMENT PRACTICES

Advanced Application Intelligence, Modern development IDEs to empower the next generation of developers.
20-50% productivity benefits for typical development tasks[1]



Successful DevOps transformation results in **7x lower software delivery change failure rate[3]**

AUTOMATION

Powerful and automated unit testing as part of a continuous delivery pipeline
50-90% time savings on test cycles[2]

SKILLS AVAILABILITY

DevOps Engineer is **#2 on Glassdoor's 50 best Jobs** in America Rankings and **one of the Top 5 Technical Skills Growth areas**

Strong Integration with Open Source → 84% of developers prefer to use open source over closed or proprietary tools[4]

git Jenkins JIRA

OPEN TOOL CHAIN

ANSIBLE openstack™ sonarqube

What is DevOps ?

An approach to lean and agile software delivery that promotes closer collaboration between lines of business, development, quality assurance and IT operations.

Outcome : Deliver customer business application requirements, more iteratively, faster, with improved quality and at a lower cost.

Accelerate software delivery – *for faster time to value*

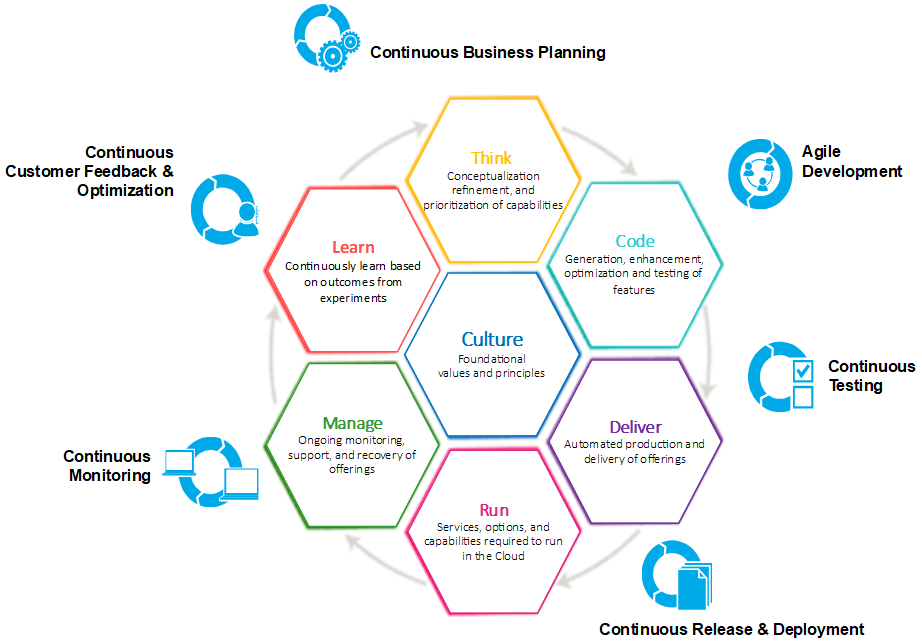
Balance speed, cost, quality and risk – *for increased capacity to innovate*

Reduce time to customer feedback – *for improved customer experience*

Process

Culture

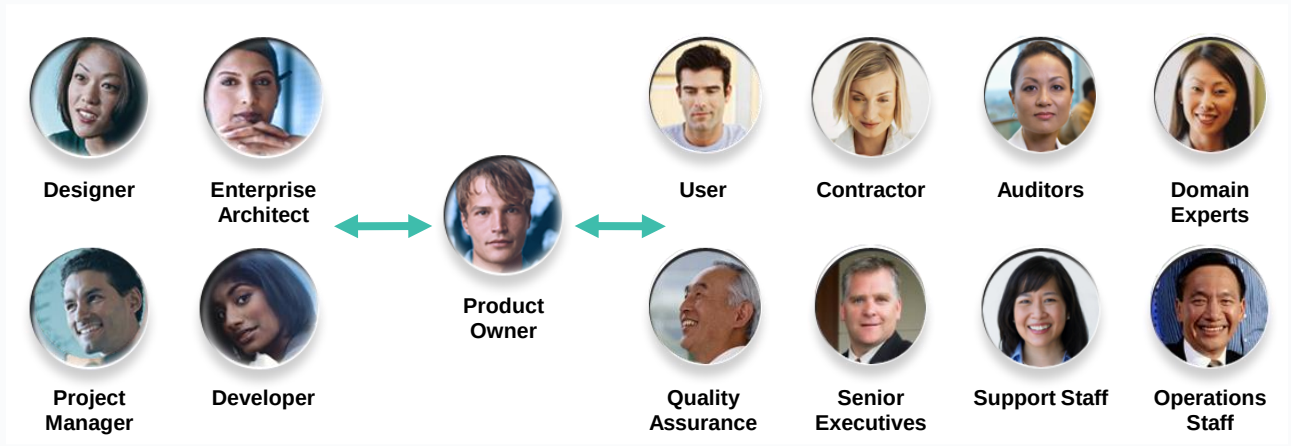
Technology



Lean and Agile principles

Building a DevOps culture

- **Don’t under-estimate the value of training and skills enablement!**
  - Agile practices (Stand-ups, Sprints, Playbacks, Retrospectives), Design Thinking, new Tools
- **Everyone is responsible for Delivery**
  - Successful teams communicate well, respect each others opinions, allow everyone to contribute



It’s all about the people

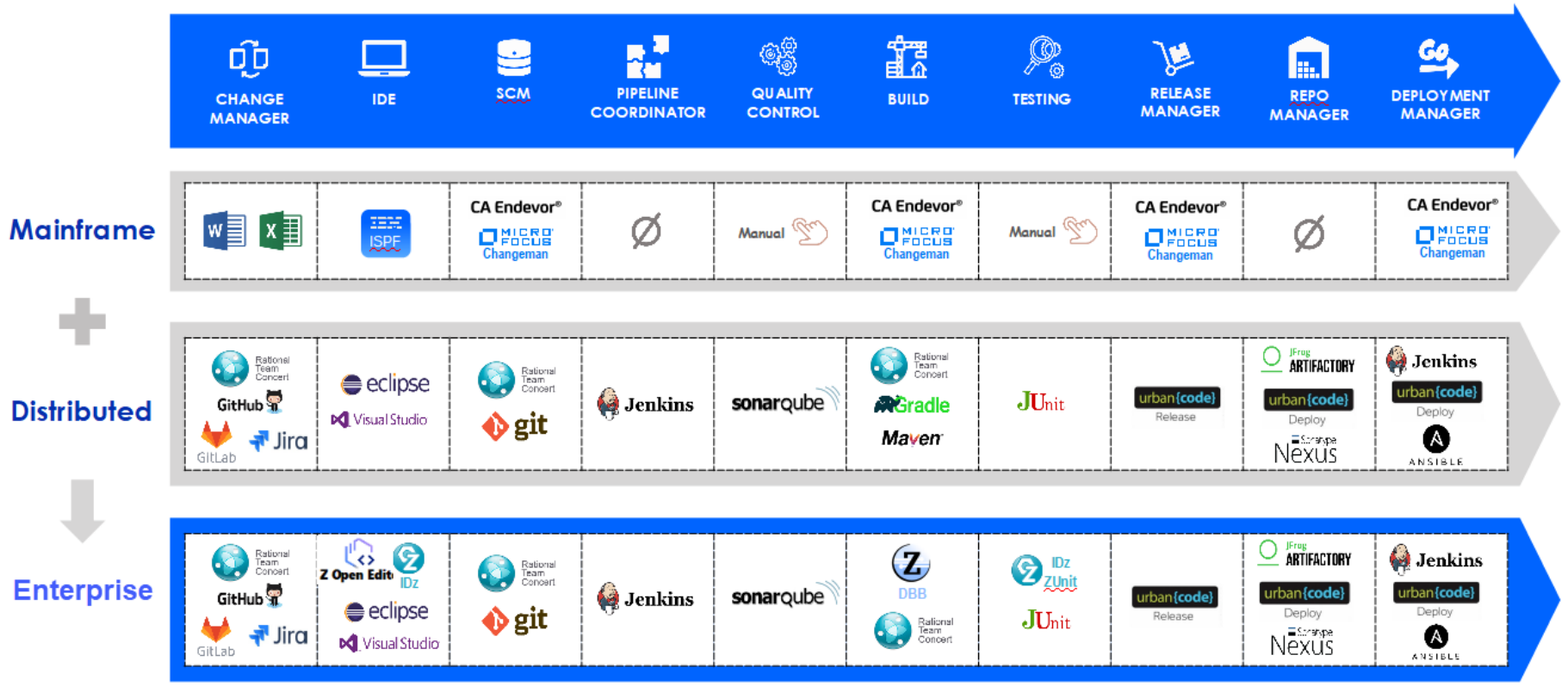
Often changing the culture is the hardest part

It is a journey
- Management commitment is a key to success
- Survive short term decrease in productivity for long term gains
- Be sure to promote winning examples

Educate all team members on the new…
- Goals
- Processes
- Tools

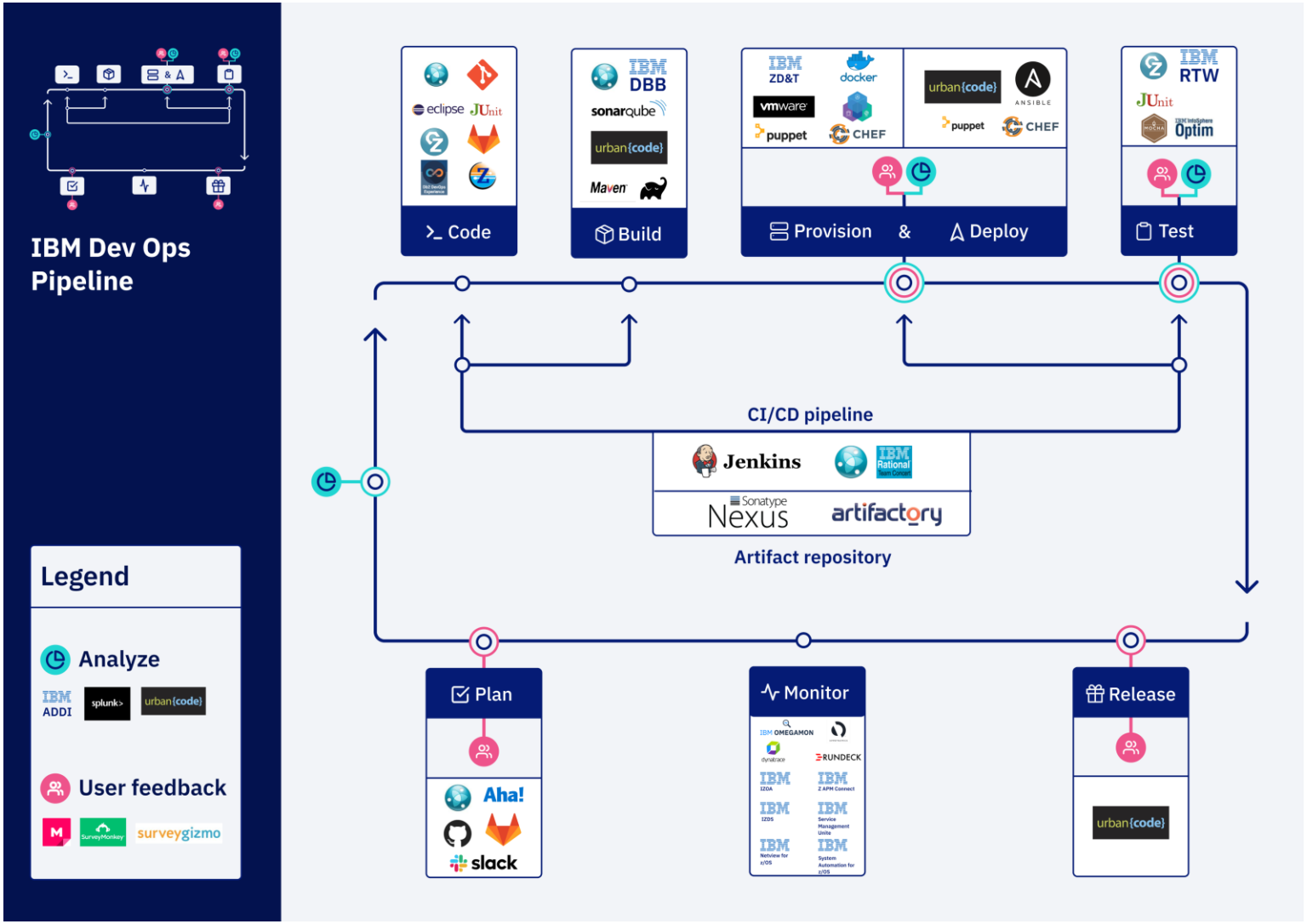
Understand where your bottlenecks are
- Planning, Development, Testing, Deployment
- Continue to look for ways to improve them
- Standardize and Automate wherever possible

Enterprise DevOps – Consistent tools across the enterprise



Enterprise DevOps – The Benefits

- Consistent tools across the enterprise.
  - Better collaboration.
  - Developer flexibility.
- Distributed teams have already figured out much of the process & culture issues.
- Easier on-boarding of new z developers.
- Better integration using open source.
- Elimination of administrative work.


IBM Dev Ops Pipeline
Code
Build
Provision & Deploy
Test
CI/CD pipeline
Jenkins
Sonatype Nexus
artifactory
Artifact repository
Plan
Monitor
Release
Legend
Analyze
User feedback

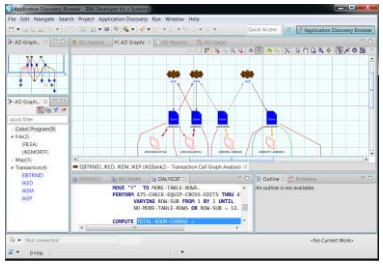
IBM Developer for z Systems:

An Integrated Development Environment for System z

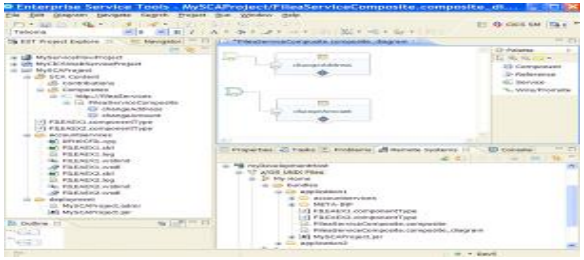
Integration with Team Concert for Lifecycle and Source Management

Access to typical System z sub-system functionality in z/OS, CICS, IMS, DB2, WAS

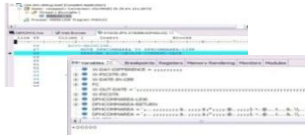
Integration with Application Discovery



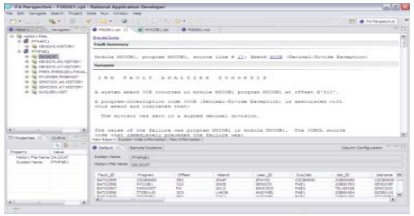
IBM Developer for System z



Out of the Box debugger and code coverage capabilities

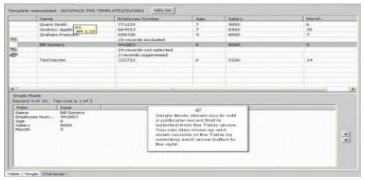


Integration with Fault Analyzer for Dump Analysis



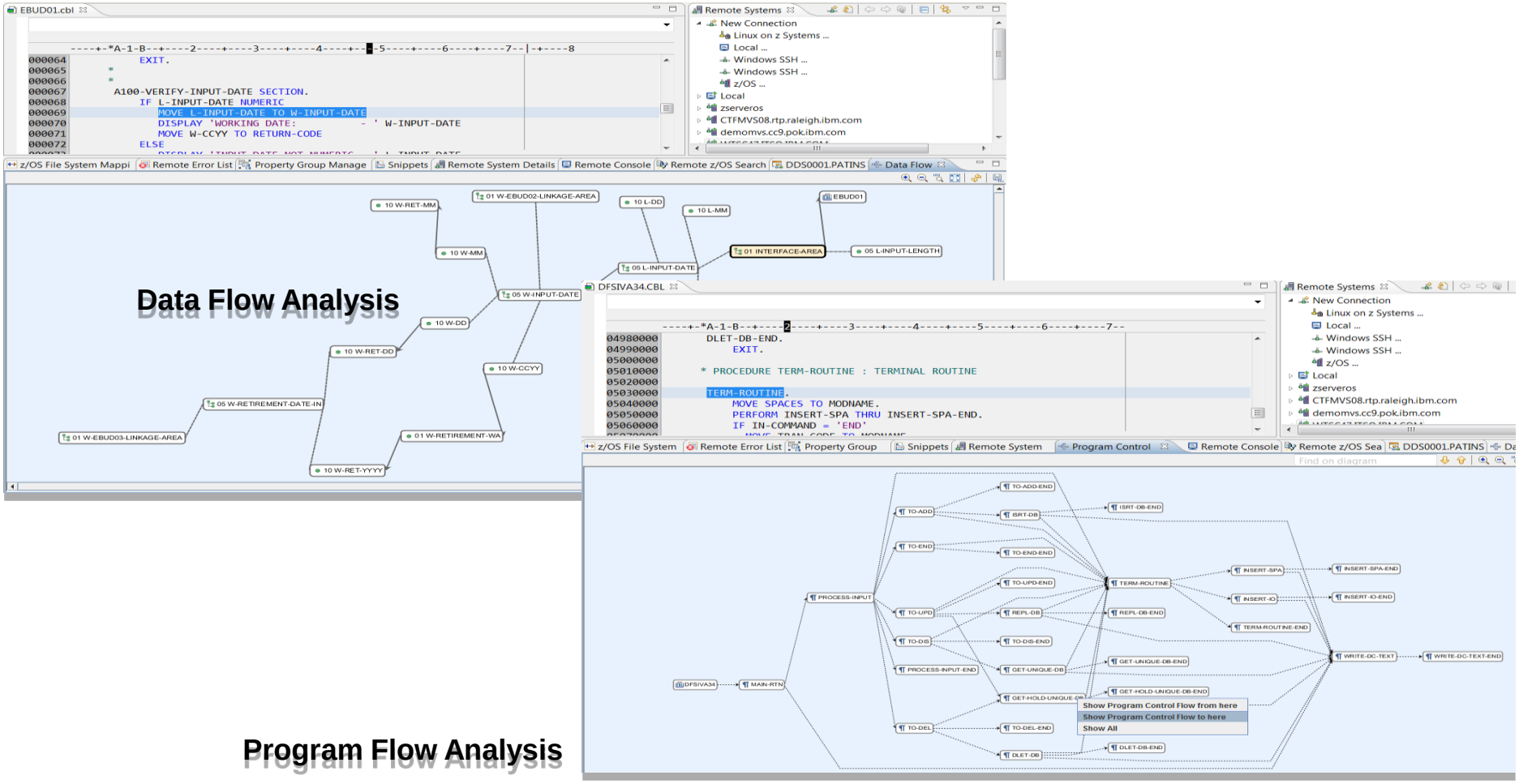
A modern IDE for productive development of cross-platform applications written in COBOL, PL/I, ASM, Java, EGL or C/C++ in System z CICS, IMS, DB2, Batch applications

Integration with File Manager for file and test data handling

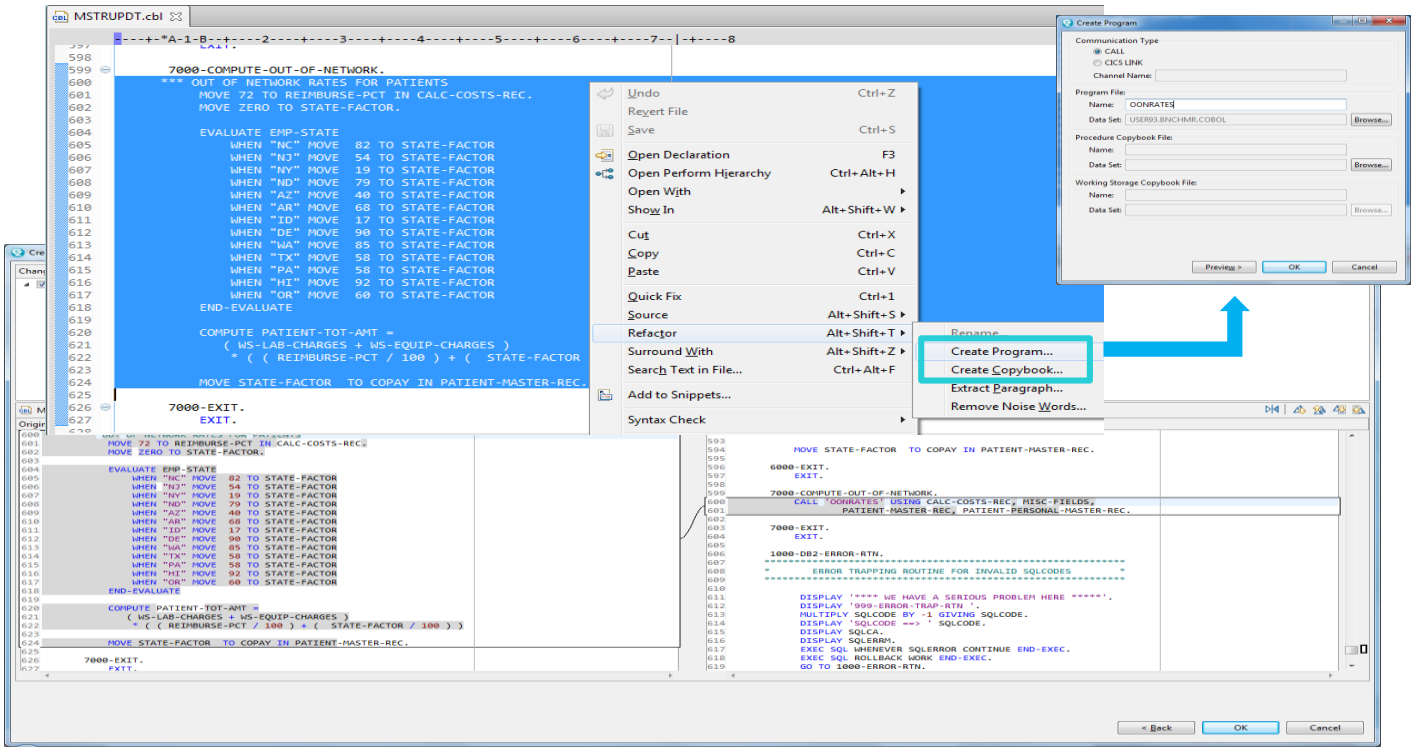


Integration with zD&T for flexible access to System z environment

IDz Program Analytics



COBOL Program Refactoring

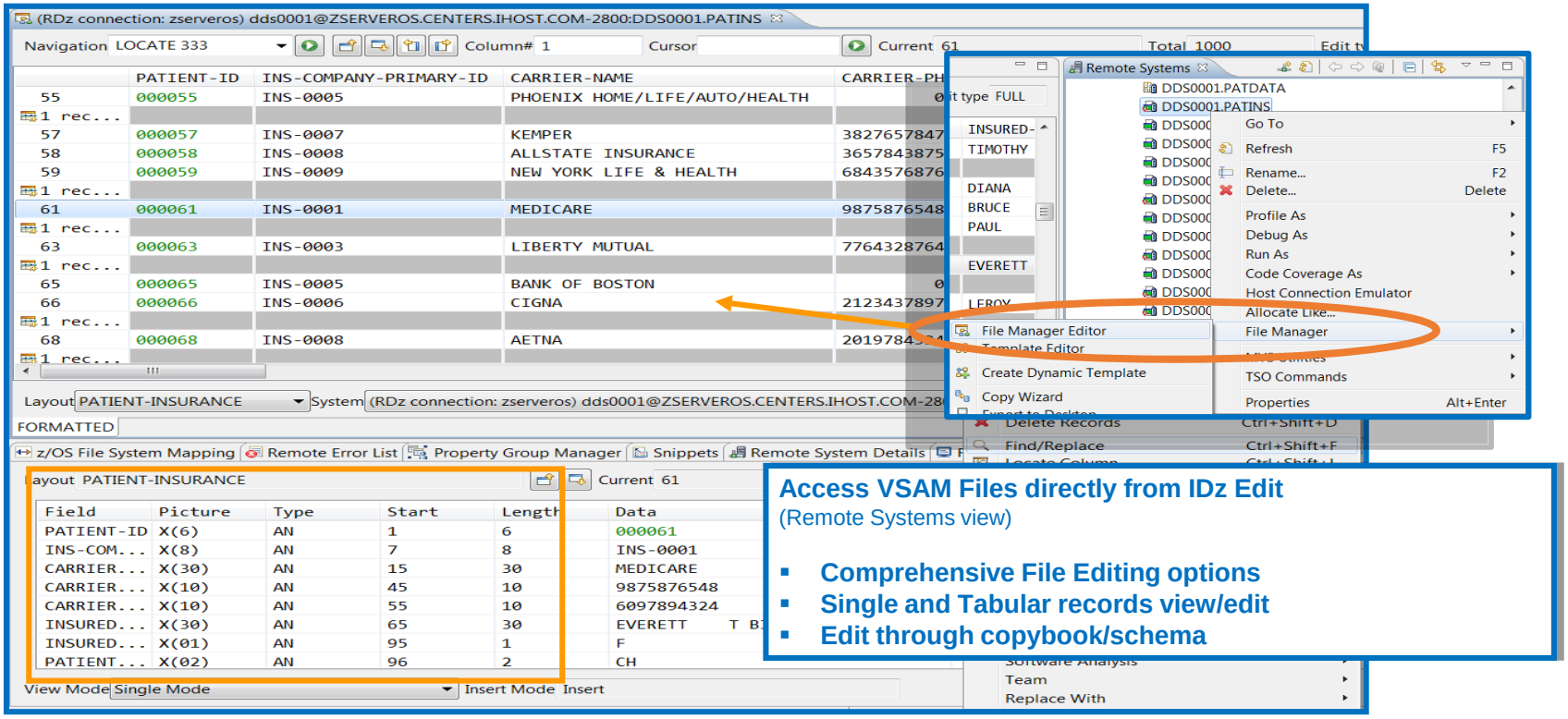


- **Modularization**
  - Increase application testability
    - Easier to create automated unit test cases
  - Accelerate development and deployment
    - Separate presentation logic from business logic and data access logic
    - Allows for concurrent development
- **Maintainability**
  - Improve readability of existing applications making them easier to maintain
  - Decrease complexity of application logic

File Manager (FM) - Edit file types

- Copy/Field records
- Different record types in dataset
- Sample
- Sort
- Fixed, Variable

- HFS files
- Sequential (PDS and PDSE
- VSAM (including IAM)
- Websphere MQ messages on a queue
- CICS: TS or TD queues



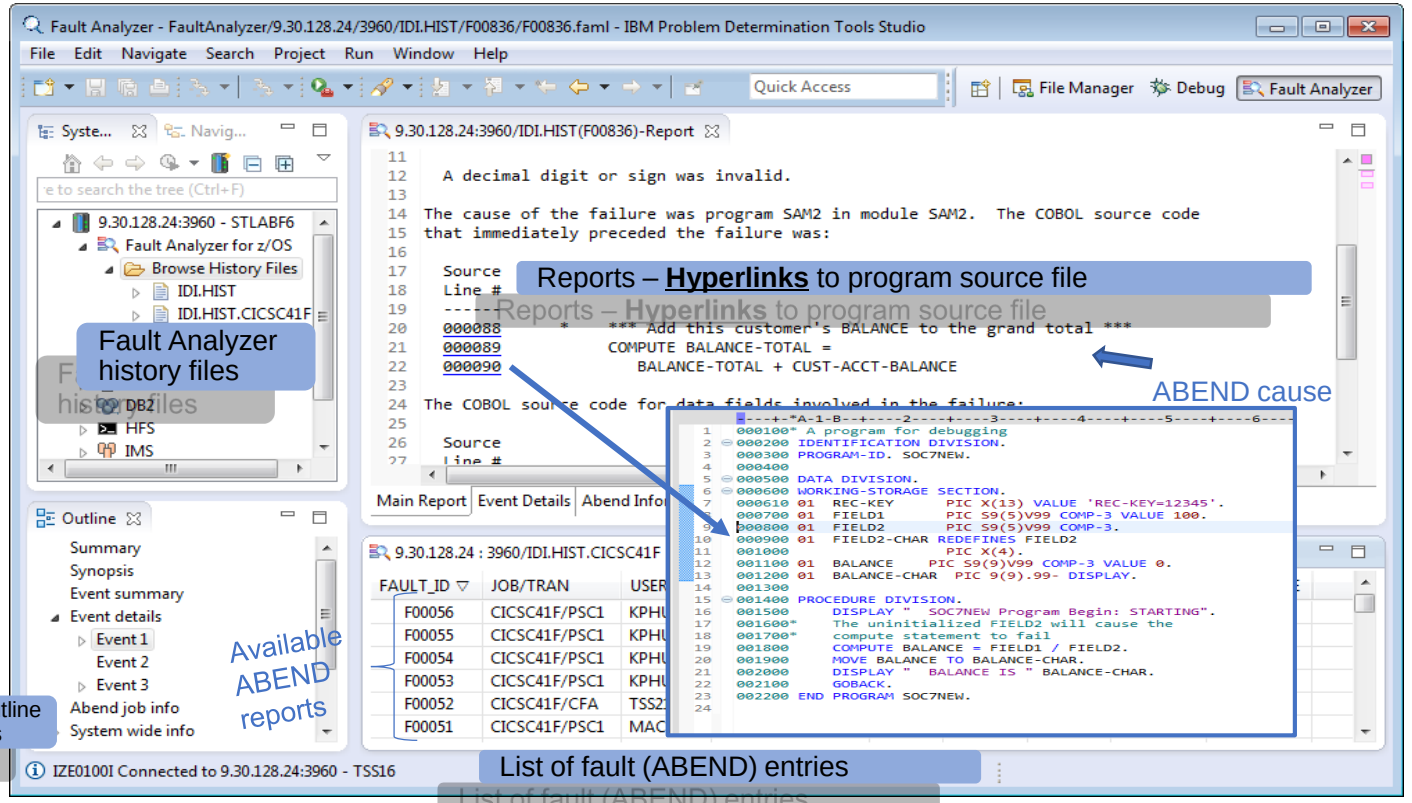
Access VSAM Files directly from IDz Edit
(Remote Systems view)

- **Comprehensive File Editing options**
- **Single and Tabular records view/edit**
- **Edit through copybook/schema**

Fault Analyzer (FA) for Program Failures

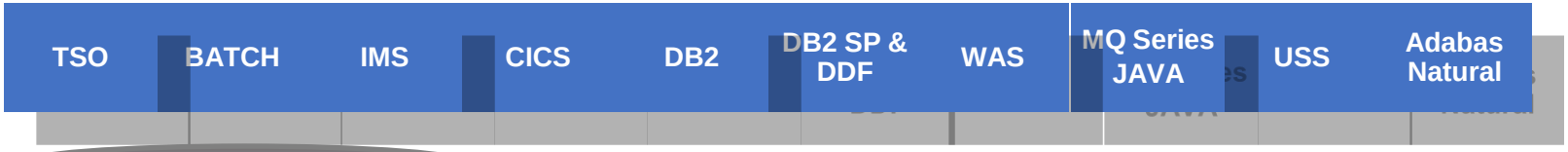
IBM Fault Analyzer improves developer productivity and decreases deployment costs by helping to analyze and correct application failures quickly (CICS,DB2,IMS,MQ,COBOL,PLI,ASM, C/C++,JAVA)

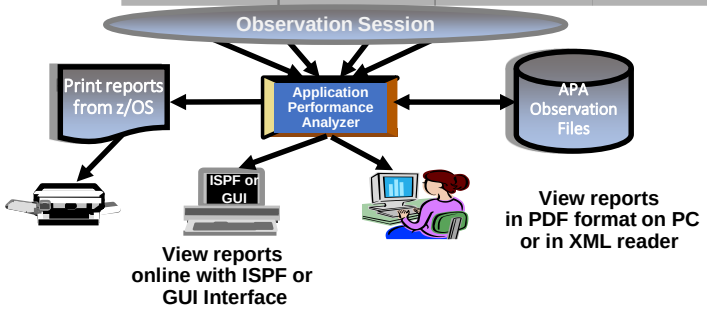
- Automatic program abend capture and reporting
- Program source-level reporting
- Provides a detailed report about program failures to help resolve them quickly
- Enables you to track and manage application failures and fault reports
- Integration with ADFz and 3270-based Interface



Application Performance Analyzer (APA) for z/OS

Provides rapid pin-pointing of enterprise application bottlenecks





- Displays overall system activity, enabling you to check job execution online and select which active job to monitor
- Automatically starts to monitor job performance when the job or program becomes active
- Provides multiple summary reports to assist in identifying key areas of performance bottlenecks

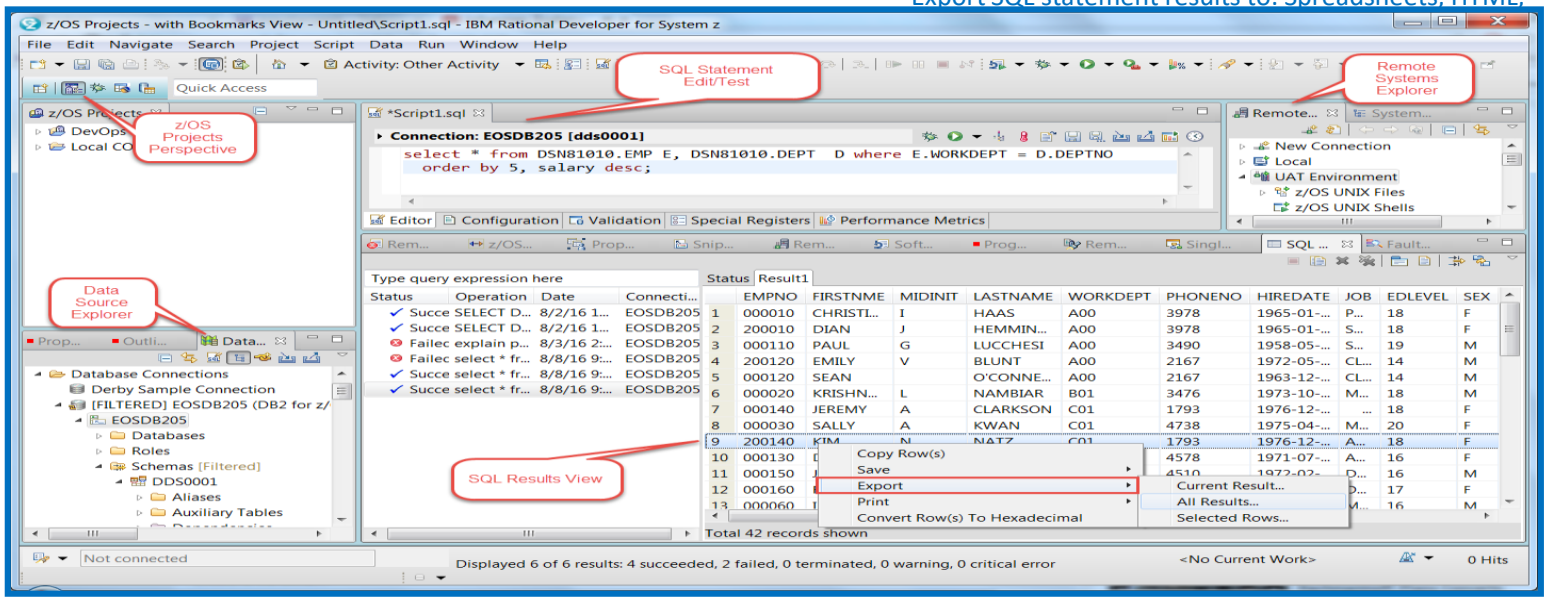
Integrated with ADFz and 3270-based Interface

- Specify number of times APA monitors a job's performance when **job or program becomes active**
- **Invoke the monitoring capability** from other programs such as IBM Tivoli® OMEGAMON
- **Compare two observation reports** to see the relevant differences, to supplement threshold monitoring
- Assembler, COBOL and PL/I **statement usage** within each module or disassembly for modules without source
- **DASD & Processor statistics**
- IMS/CICS transaction performance relationships to database
- **DB2 z/OS; Stored Procedures**, SQL, DDF detailed **DB2 delay information**
- IBM WebSphere **MQ** queue information

Data Studio (plug-in): DB2 Development Tooling

A flexible graphical interface to DB2 and SQL

- DB2 Table/View/Index Analysis
- DB2 Data Model and for Test Data Management and manipulation:
  - CRUD
  - Simple Table row/column sub-setting (sampling)
  - Test/Run/Tune SQL directly from COBOL or PL/I programs
  - Run existing SPUFI files (DDL, DML)
- SQL code and Test
  - Code embedded SQL directly within COBOL, PL/I and Assembler programs
  - Statement content assist from the DB2 Catalog
- Code SQL with graphical tooling
- Interactively Code/Test/Tune SQL statements
  - Export SQL statement results to: Spreadsheets, HTML,



IBM Developer for z Systems – also known as IDz (previously known as RDz)

https://developer.ibm.com/mainframe/products/ibm-developer-for-z-systems-enterprise-edition/

— IBM Developer for z Systems is a comprehensive solution for creating and maintaining z/OS applications efficiently. The rich set of COBOL, PL/I, C/C++, High Level Assembler (HLASM), JCL and Java development tools, and optimized tooling for Batch, CICS, IMS and DB2 runtimes provide z/OS application developers a modern development environment for enhanced productivity.

— Features include...
- z/OS Explorer
- CICS Explorer
  - CICS Resource management
  - Bundle editor & Event processing
- Smart editors for COBOL, PL/I and JCL
- LPEX editor
  - ISPF look & feel (command line, prefix area)
  - COBOL, PL/I, JCL, Assembler, C/C++, REXX
- Program understanding tools
- Software Analyzer
  - IDE and Batch support
- IBM z/OS Debugger

— Authorized User licenses, Floating licenses

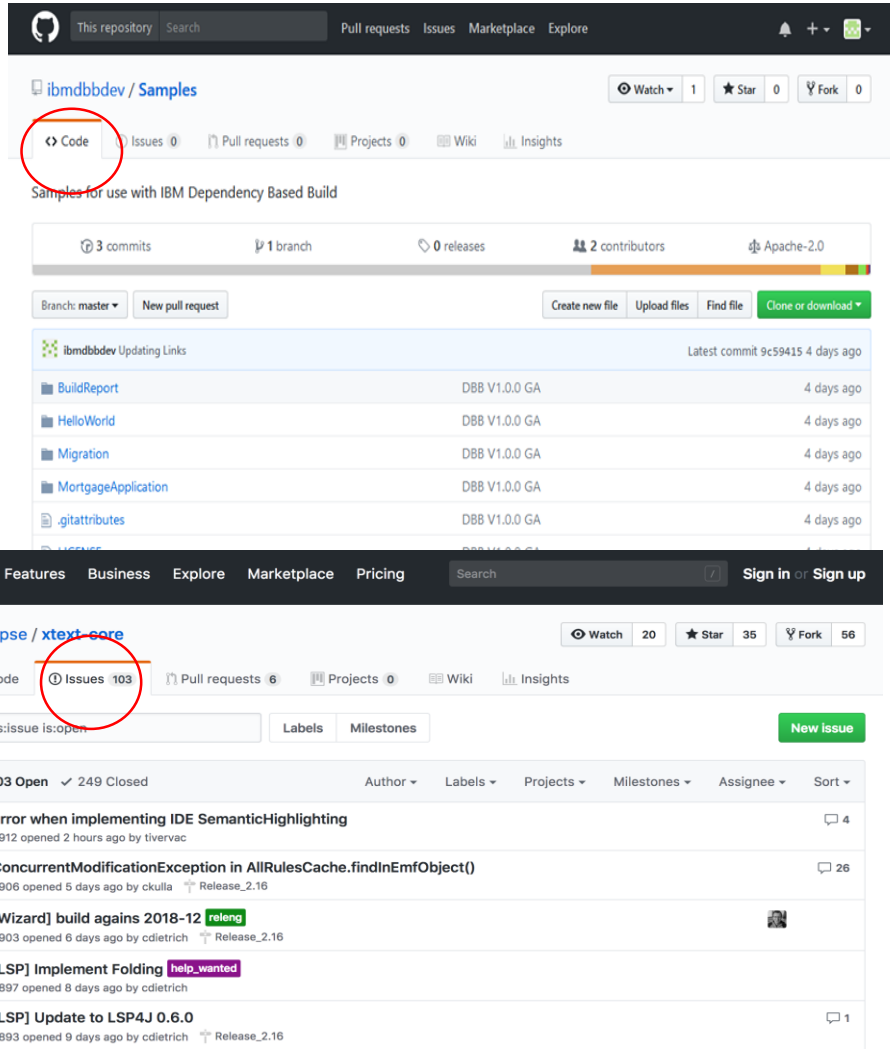
— Features include...
- Test Case generation
- Run tests from the IDE or CI/CD pipeline
- Stubbing subprograms and subsystems
- Automated Pass/Fail checking
- Data recording
- File handling
- Code coverage results
- Integration with dashboards
- Refactoring tools

— Features include...
- Git/DBB Integration
- CARMA framework
  - CA Endevor integration
- Menu Manager
  - Extensibility
  - Edit Macro support
  - Interactive REXX
- Enterprise Service Tools
  - SOAP/XML
  - JSON
- Data Tools
- Host Emulator

Source Code Management

What is Git?

- Open Source SCM initially built to manage Linux development
- Distributed Source Code Management
  - Full copy of the repository locally for each user their own private copy on their own machine
  - Uses a commit ID instead of a version
  - Forking the repository is normal
- Used with other open source tools such as Jenkins, Ant, Maven, Gradle for Continuous Integration / Continuous Delivery pipeline
- Strengths
  - Simple, cheap and easy to deploy
  - Supports true parallel development
  - Pervasive in Open Source development where there is no effective competition
- GitHub, GitLab, and BitBucket are common Git based web hosted repositories.
- In 2017 Rocket Software released a port of Git for z/OS
  https://www.rocketsoftware.com/product-categories/mainframe/git-for-zos



IBM Dependency Based Build

https://developer.ibm.com/mainframe/products/ibm-dependency-based-build/

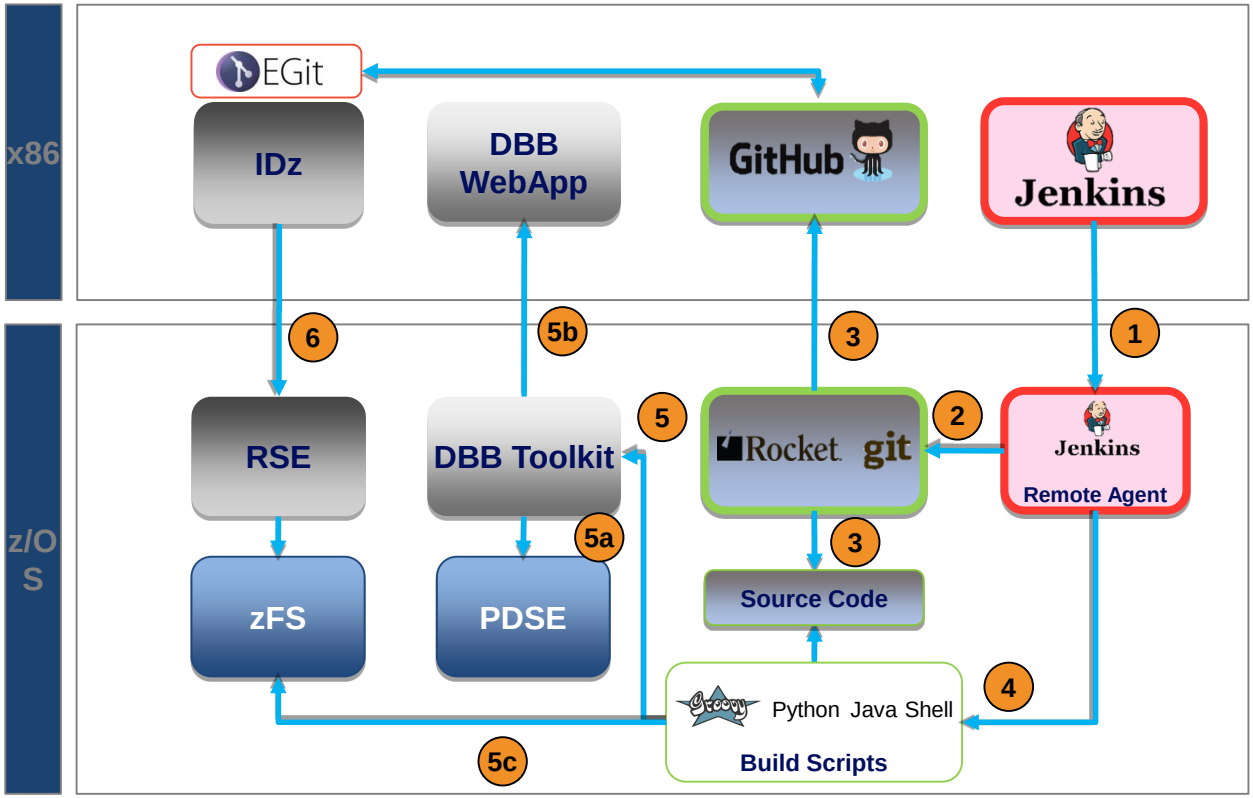
Supports general automation task support:

- Use the shell, Python, Java or groovy instead of JCL

Contains a Dependency Scanner to understand dependencies

- Dependencies are stored in database for use by builds
- Application Server provides REST based access to information stored
- Build reports can also be stored in database

Dependency Based Build typical setup



[1] Jenkins server sends build commands to remote agent.

[2] Jenkins agent issues Git pull command to update local Git repository.

[3] Rocket's Git client automatically converts source from UTF-8 to EBCDIC during pull.

[4] Jenkins agent invokes build scripts containing DBB APIs in local Git repository.

[5] DBB Toolkit provides Java APIs to:

[5a] Create datasets, copy source from zFS to PDS, invoke zOS programs, issue ISPF and TSO commands.

[5b] Scan and store dependency data from source files, perform dependency and impact analysis, store build results.

[5c] Copy logs from PDS to zFS, generate build reports (can be saved in build result).

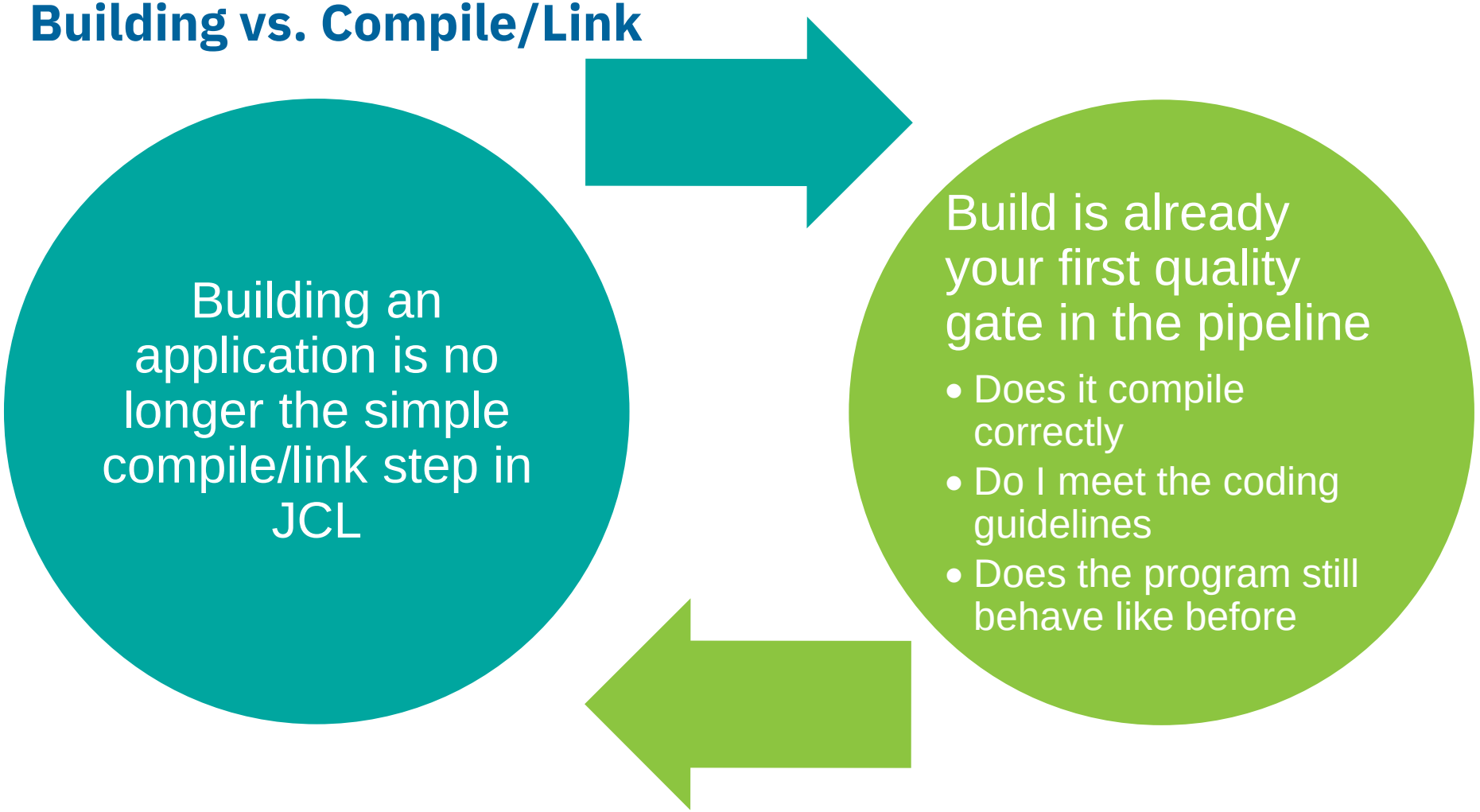
[6] IDz allows developers to test code changes before committing by running a user build.

Example Compile JCL vs Groovy+DBB APIs

```
//COBOL  EXEC PGM=IGYCRCTL,REGION=0M,
//       PARM='LIB'
//*
//STEPLIB  DD DISP=SHR,DSN=IGY.SIGYCOMP
//*
//SYSIN    DD DISP=SHR,DSN=USER1.BUILD.COBOL(HELLO)
//SYSLIN   DD DISP=SHR,DSN=USER1.BUILD.OBJ(HELLO)
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT2   DD DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT3   DD DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT4   DD DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT5   DD DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT6   DD DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT7   DD DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
```

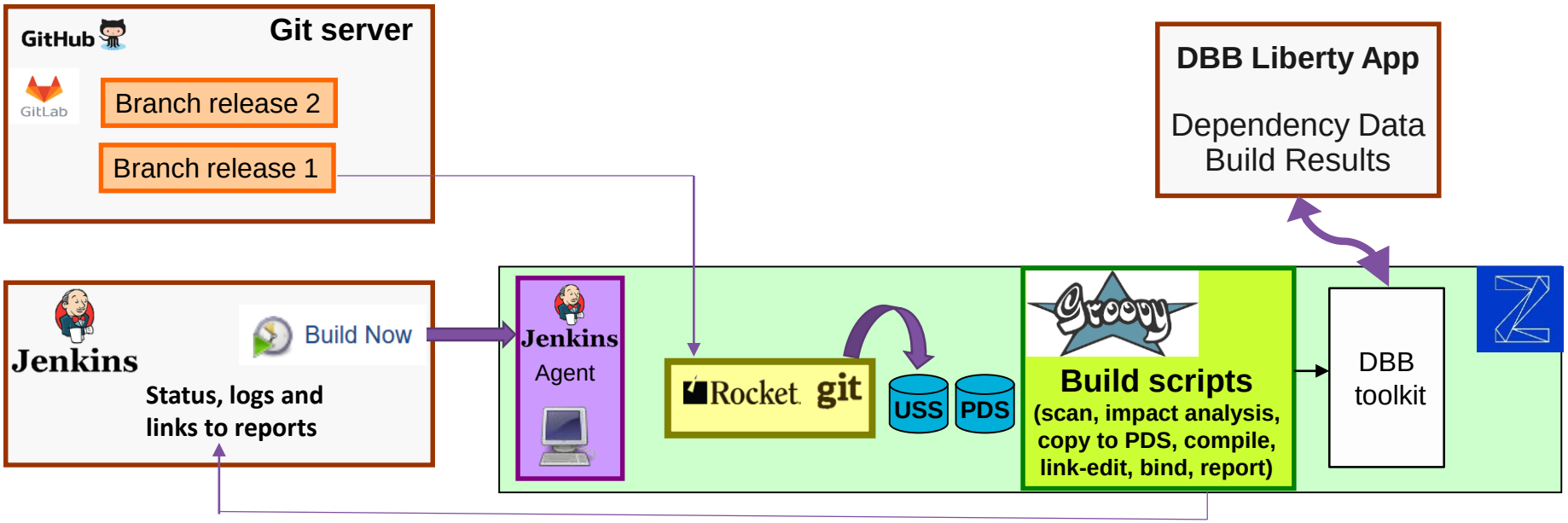
```
import com.ibm.dbb.build.*

println("Copying source from zFS to PDS . . .")
def copy = new CopyToPDS().file(new File("/u/usr1/build/helloworld.cbl"))
                          .dataset("USR1.BUILD.COBOL").member("HELLO")
copy.execute()


println("Compiling . . .")
def tempOps = "tracks space(5,5) unit(vio) blksize(80) lrecl(80) recfm(f,b) new"
def compile = new MVSExec().pgm("IGYCRCTL").parm("LIB")
compile.dd(new DDStatement().name("TASKLIB").dsn("IGY.SIGYCOMP").options("shr"))
compile.dd(new DDStatement().name("SYSIN").dsn("USR1.BUILD.COBOL(HELLO)")
                            .options("shr"))
compile.dd(new DDStatement().name("SYSLIN").dsn("USR1.BUILD.OBJ(HELLO)")
                            .options("shr"))
compile.dd(new DDStatement().name("SYSPRINT").options(tempOps))
compile.dd(new DDStatement().name("SYSUT1").options(tempOps))
compile.dd(new DDStatement().name("SYSUT2").options(tempOps))
compile.dd(new DDStatement().name("SYSUT3").options(tempOps))
compile.dd(new DDStatement().name("SYSUT4").options(tempOps))
compile.dd(new DDStatement().name("SYSUT5").options(tempOps))
compile.dd(new DDStatement().name("SYSUT6").options(tempOps))
compile.dd(new DDStatement().name("SYSUT7").options(tempOps))
compile.copy(new CopyToHFS().ddName("SYSPRINT")
                            .file(new File("/u/usr1/build/helloworld.log")))
def rc = compile.execute()

if (rc > 4)
    println("Compile failed!  RC=$rc")
else
    println("Compile successful!  RC=$rc")
```

Building vs. Compile/Link

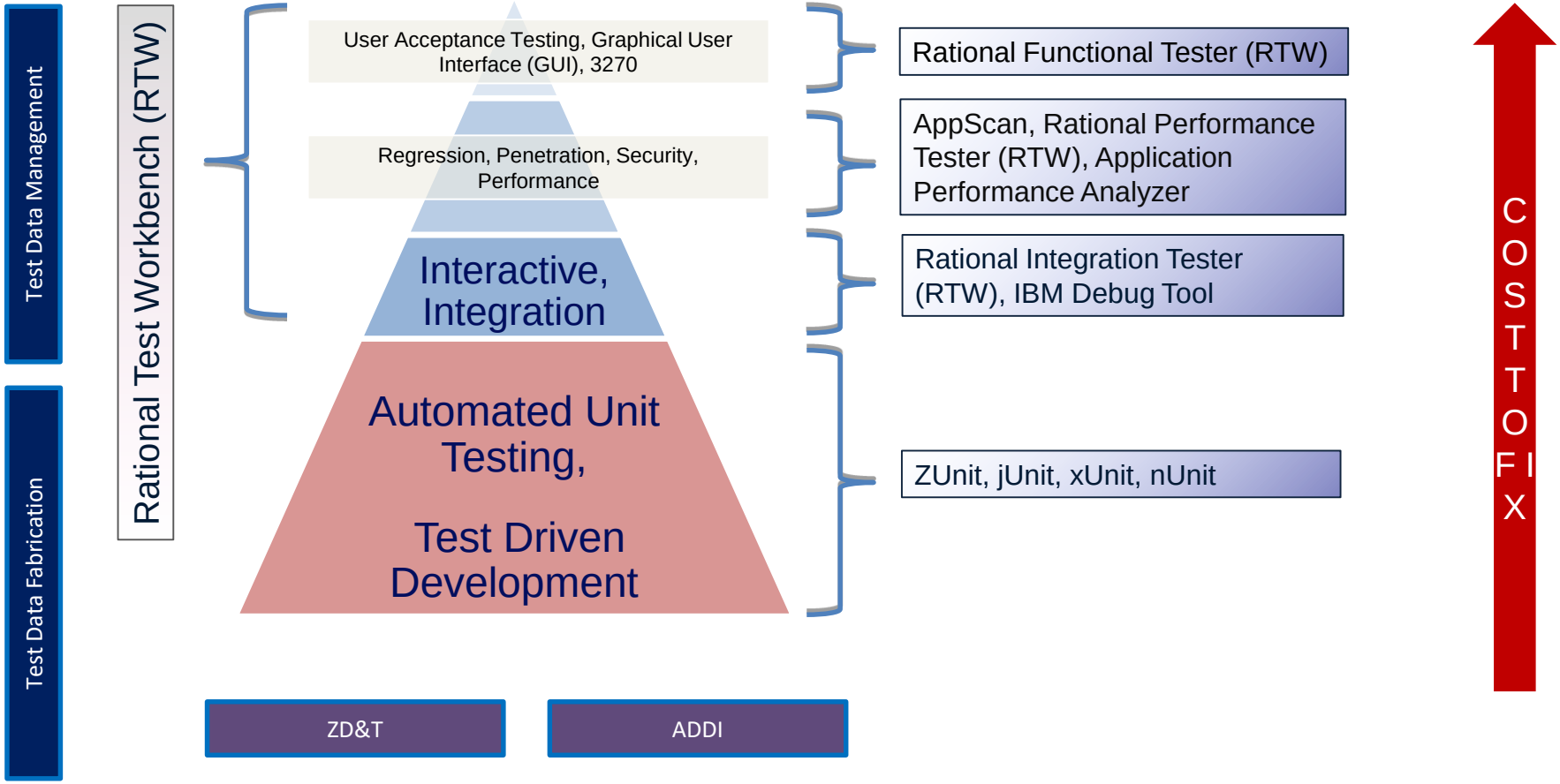


IBM Dependency Based Build (DBB)

- Intelligent build environment for applications
  - ***Groovy*** driven build environment for **compiling**, **linking** and **processing** z/OS programs and applications
  - Dependency identification and understanding
    - Automatically identify program dependencies , build intelligently based on dependencies
  - Integration with any CI tooling (e.g. *Jenkins*)

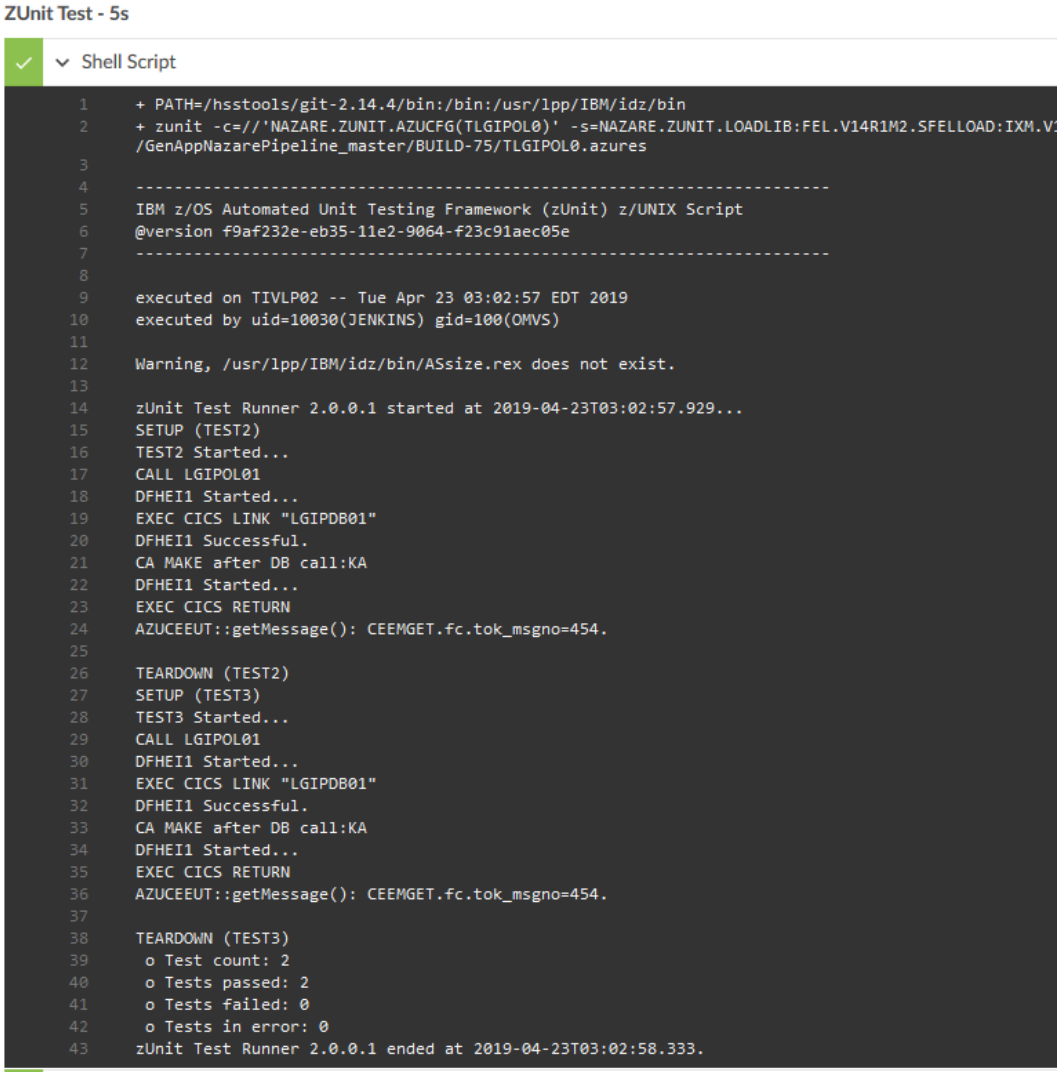


There are many types of testing



Automated Unit Tests

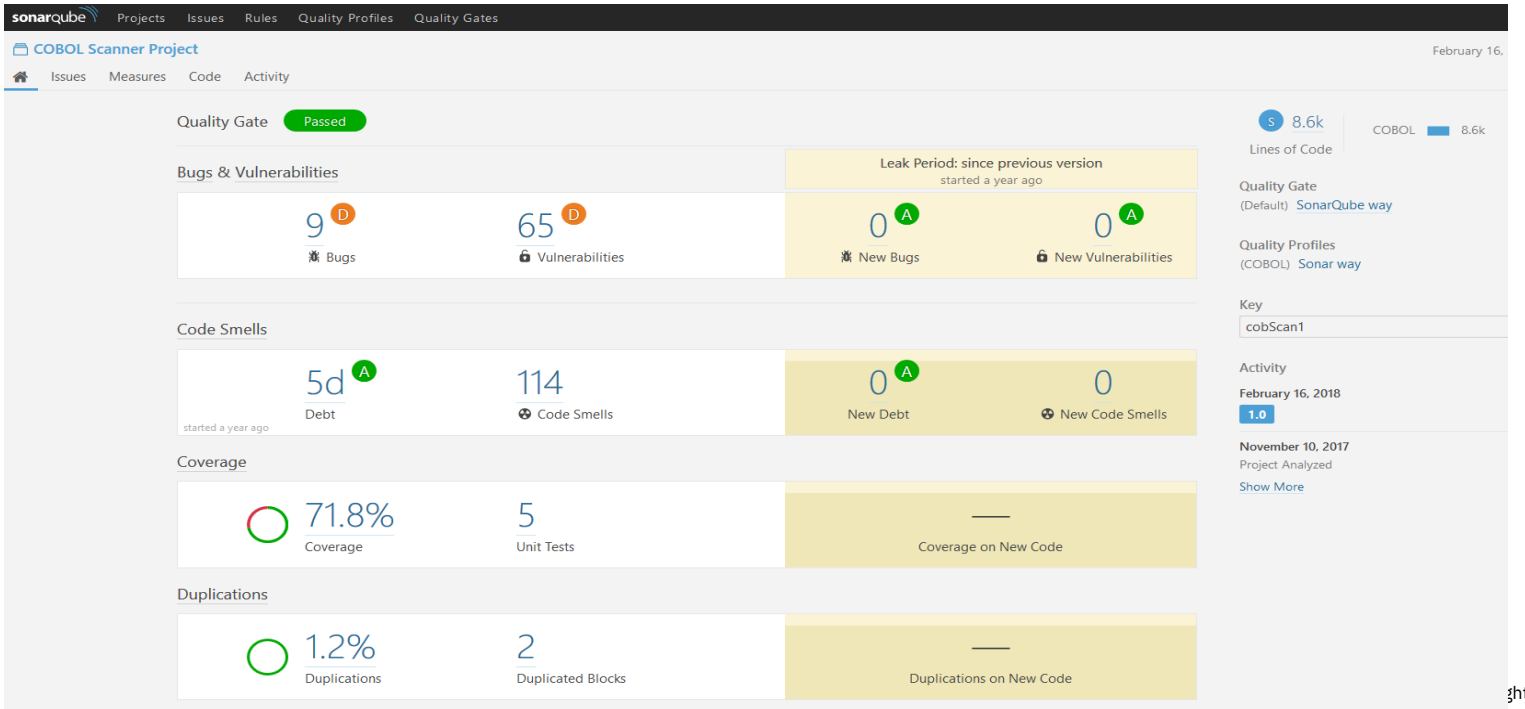
- Availability of isolated environments is a key to Early Testing
- Developers to perform unit tests without interference from other teams
- Elimination of shared data issues
- zUnit is the framework
  - Creating and running COBOL and PL/I test cases
  - The same test cases can be run from the IDE and a CI pipeline
- Enables Continuous Integration for z/OS



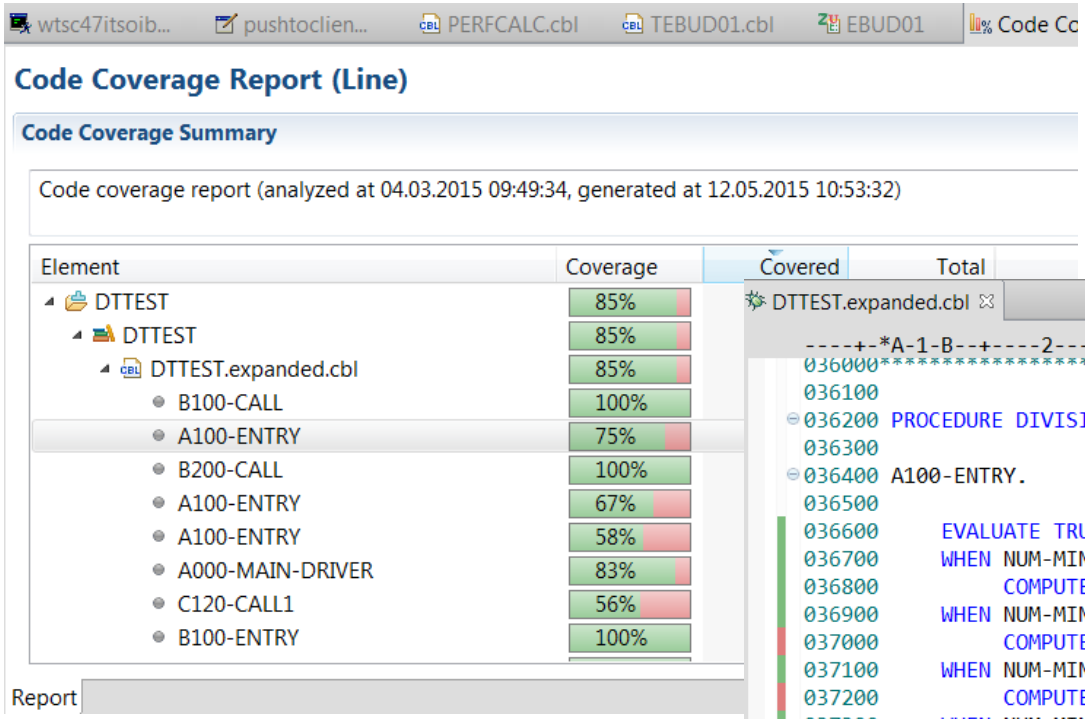
Code Quality Dashboards

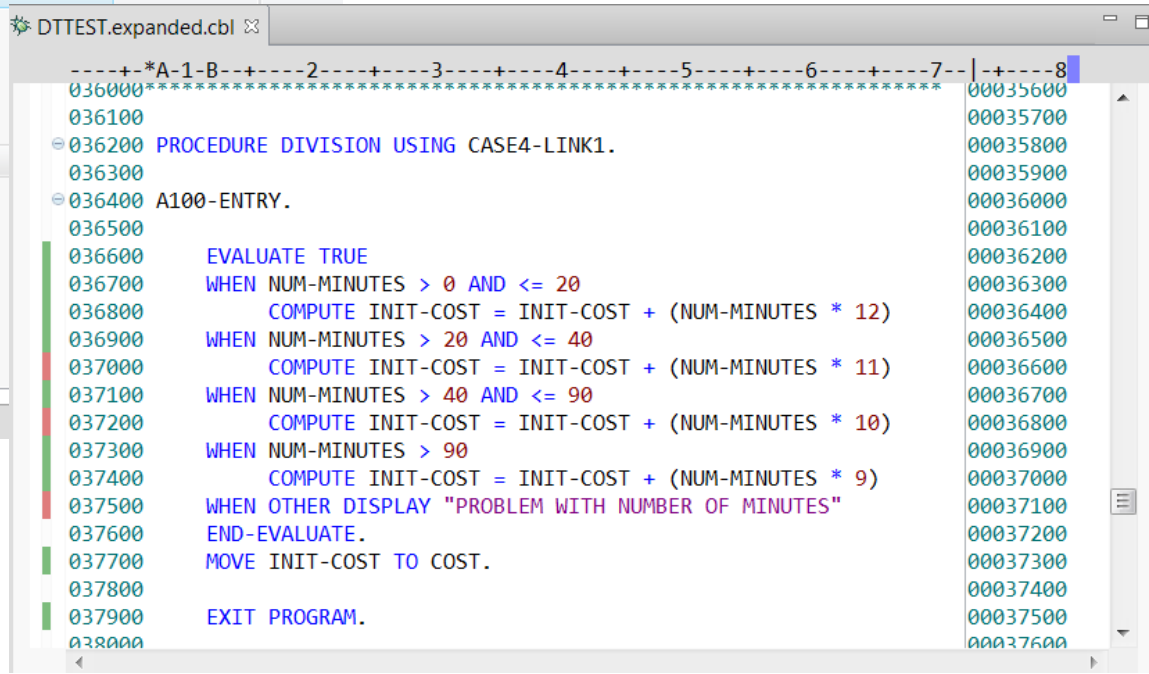
— Report on coding standard violations, Unit test results, and Code Coverage results
- SonarQube is a widely used Enterprise Dashboard
- Support for COBOL & PL/I is licensed and NOT open source

— IBM Application Discovery and Delivery Intelligence (ADDI) is another option



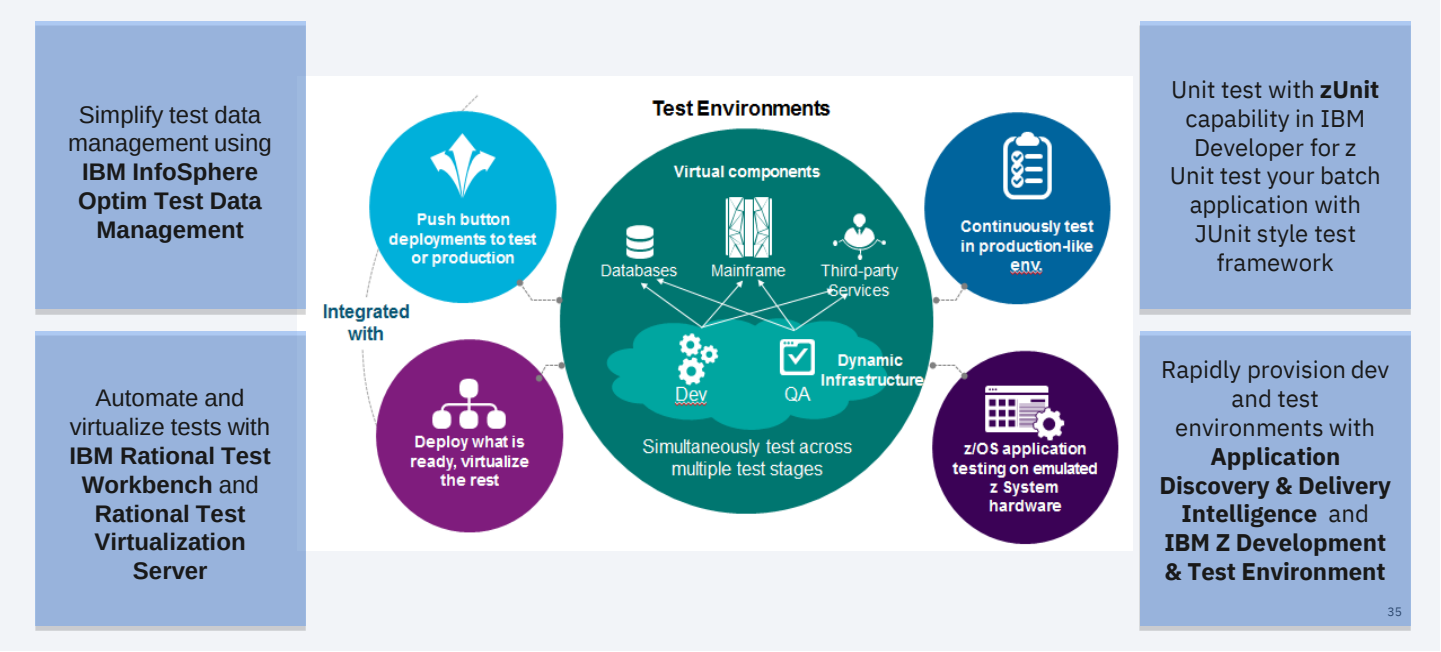
Tests linked with Code Coverage = Test Coverage!





Additional Testing

- Multiple Tools exist for Integration, Functional Testing
- Rational Test Workbench can be used for Functional, Regression, terminal-based tests, Test Virtualization, API and Performance tests
- Rational Integration Tester used for testing APIs

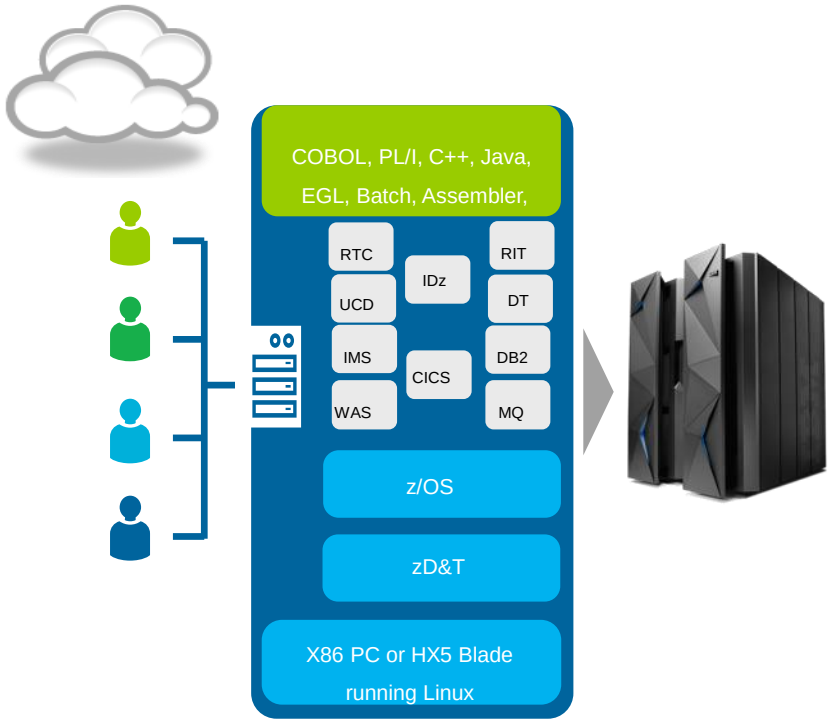


Provisioning

— Shift left with **Z Development and Test Environment**

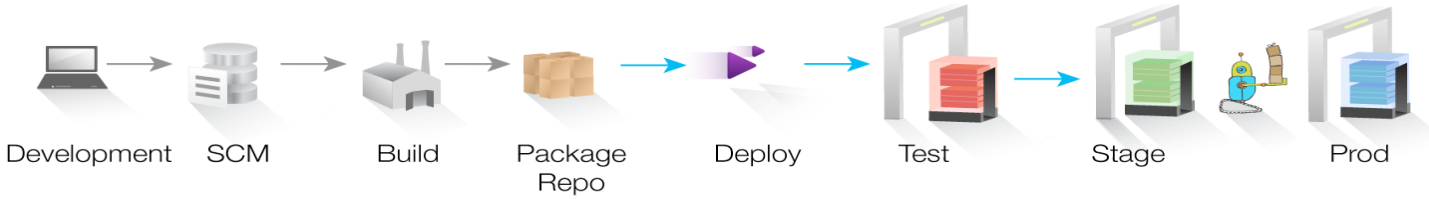
Develop and test IBM Z applications anywhere, anytime with z/OS optimized for x86 hardware

- **zD&T tools to accelerate provisioning and image management**
- Cloud friendly, software-based licensing for enterprise customers (managed service on Softlayer) NEW
- Latest z/OS 2.2 software and middleware
- ***Developer autonomy just like distributed, web and mobile developers!***



UrbanCode for Deployment automation

Enabling clients to more rapidly deliver mobile, cloud, big data and traditional applications with high quality and low risk



Drive down cost
Reduce amount of manual labor, resource wait-time, and rework by eliminating errors & providing self-service environments

Speed time to market
Increase frequency of software delivery through automated, repeatable deployment processes across development, test and production

Reduce risk
Robust configuration management, coordinated release processes, audits, and traceability

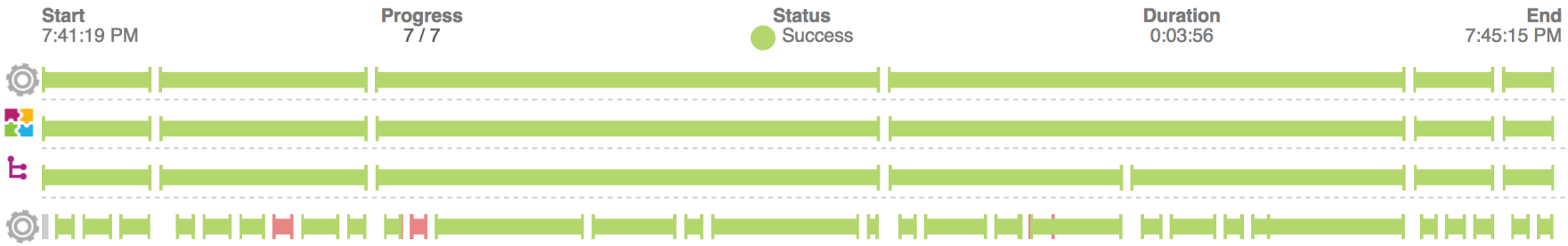
IBM UrbanCode Deploy automates the deployment of applications, databases and configurations into development, test and production environments, helping to drive down cost, speed time to market with reduced risk.

Deployment orchestration for Distributed and z/OS Applications – including CICS, Db2, MQ, across multiple environments

Integrates with Jenkins

Manages artifact versions via an in-built repository called codestation

Web UI for checking deployment status



Repeat Request | Download All Logs

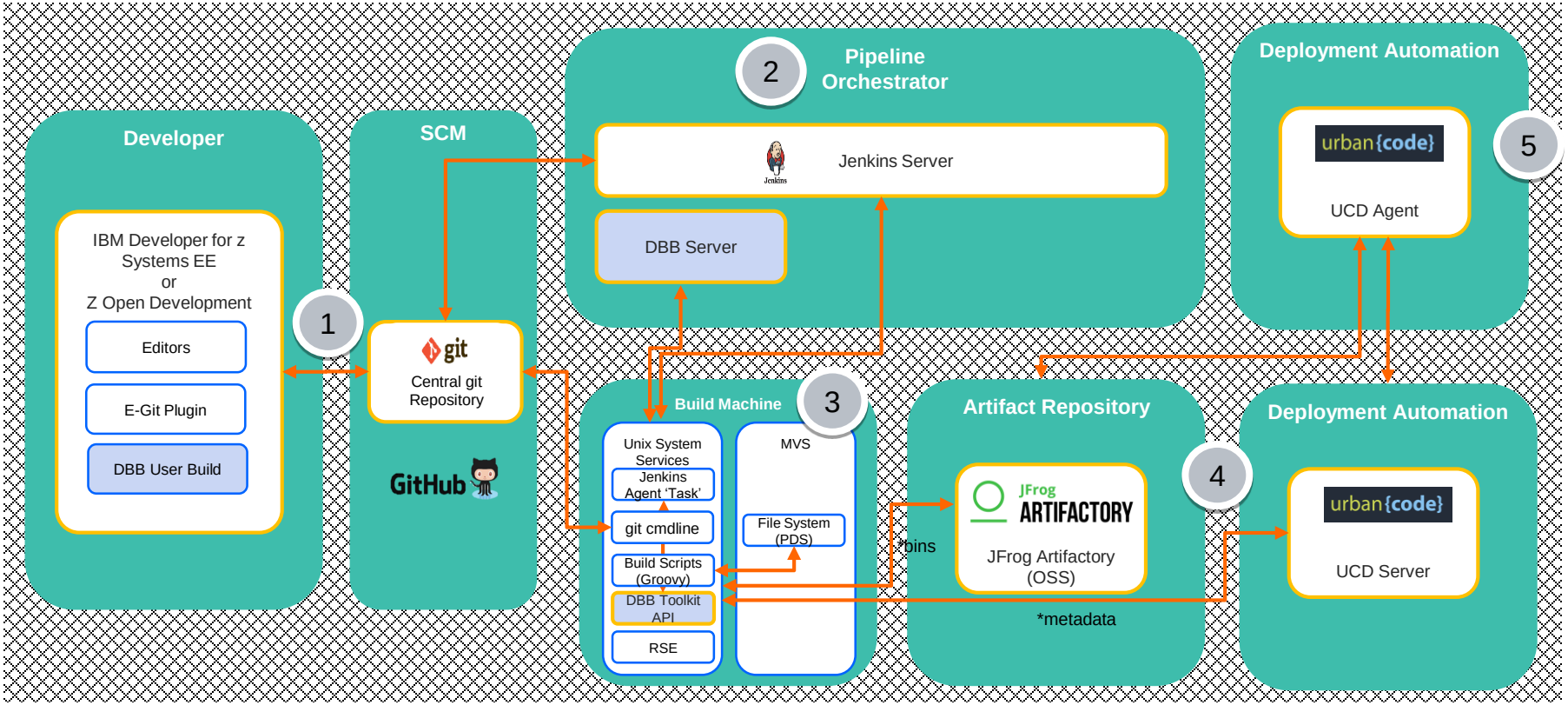
Expand All | Collapse All

Step	Progress	Start Time	Duration	Status
▸ 1. Install Insurance-Database-Config	1 / 1	7:41:19 PM	0:00:18	Success
▸ 2. Install Insurance-Database	1 / 1	7:41:37 PM	0:00:33	Success
▸ 3. Install Insurance-CICS	1 / 1	7:42:10 PM	0:01:19	Success
▸ 4. Install Insurance-Mainframe	2 / 2	7:43:30 PM	0:01:21	Success
▸ 5. Install Insurance-Liberty-Config	1 / 1	7:44:52 PM	0:00:13	Success
▸ 6. Install Insurance-Backend-Interface	1 / 1	7:45:06 PM	0:00:09	Success
Total Execution	**7 / 7**	**7:41:19 PM**	**0:03:56**	**Success**

2018

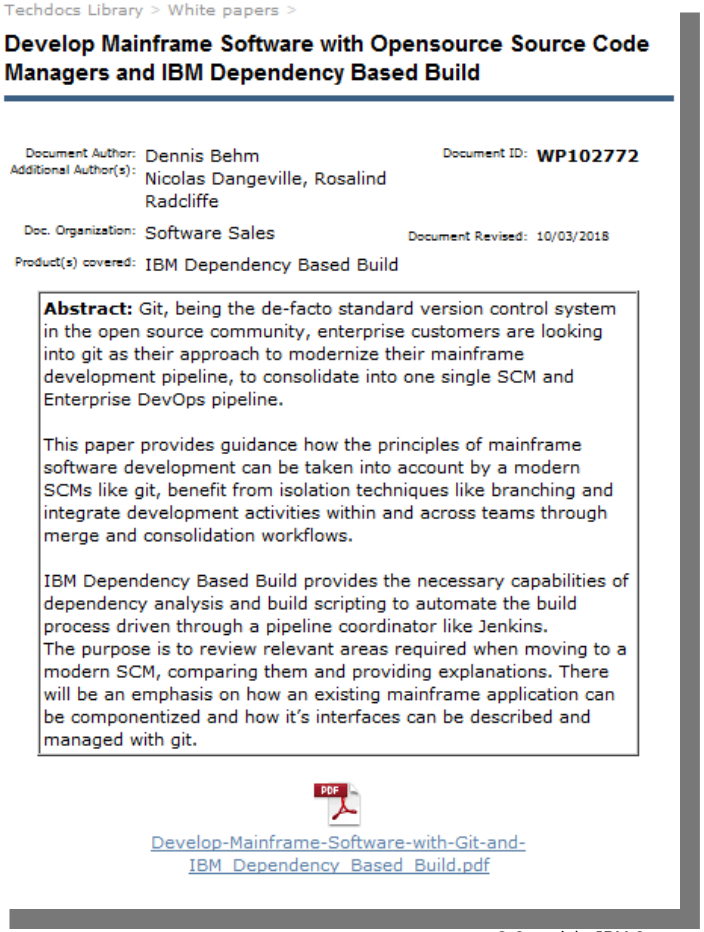
Open Pipeline build scenario – Overview

1 Edit & check-in → 2 Pipeline → 3 Build → 4 Publish → 5 Deploy & Test



Develop Mainframe Software with Opensource Source Code Managers and IBM Dependency Based build

— https://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102772



Techdocs Library > White papers >

Develop Mainframe Software with Opensource Source Code Managers and IBM Dependency Based Build

Document Author: Dennis Behm
Additional Author(s): Nicolas Dangeville, Rosalind Radcliffe
Document ID: **WP102772**
Doc. Organization: Software Sales
Document Revised: 10/03/2018
Product(s) covered: IBM Dependency Based Build

Abstract: Git, being the de-facto standard version control system in the open source community, enterprise customers are looking into git as their approach to modernize their mainframe development pipeline, to consolidate into one single SCM and Enterprise DevOps pipeline.

This paper provides guidance how the principles of mainframe software development can be taken into account by a modern SCMs like git, benefit from isolation techniques like branching and integrate development activities within and across teams through merge and consolidation workflows.

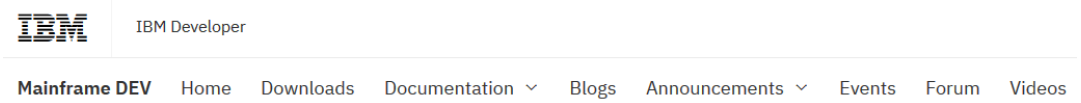
IBM Dependency Based Build provides the necessary capabilities of dependency analysis and build scripting to automate the build process driven through a pipeline coordinator like Jenkins.
The purpose is to review relevant areas required when moving to a modern SCM, comparing them and providing explanations. There will be an emphasis on how an existing mainframe application can be componentized and how it's interfaces can be described and managed with git.

Develop-Mainframe-Software-with-Git-and-IBM_Dependency_Based_Build.pdf

Additional Resources

— Mainframe CI/CD with an open toolchain:
- https://www.linkedin.com/pulse/mainframe-cicd-open-toolchain-its-real-spectacular-minaz-merali/

— Mainframe Dev Center:
- https://developer.ibm.com/mainframe/



— IBM DevOps for Enterprise Systems:
- https://www.ibm.com/it-infrastructure/z/capabilities/enterprise-devops

— For Dummies books:
- https://www.ibm.com/ibm/devops/us/en/resources/dummiesbooks/

— IBM Z Trial:
- https://ibm.biz/z-trial

Questions / Comments

Agenda

9:00 Introductions
9:20 DevOps for Z

10:00 Break

10:15 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z

11:45 Lunch

12:30 zUnit Overview

12:40 Demo: Scenario using Z DevOps solutions including zUnit, Git and Jenkins

1:30 z/OS DevOps Testing Automation & Virtualization

2:00 – 4:30 Choose one Optional lab:

- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6 : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → LAB 2C Application Discovery : Find a candidate API from Catalog Manager Application
- → LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- → LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
- → LAB 8 - Using Application Performance Analyzer (APA)

Lecture Labs Breaks

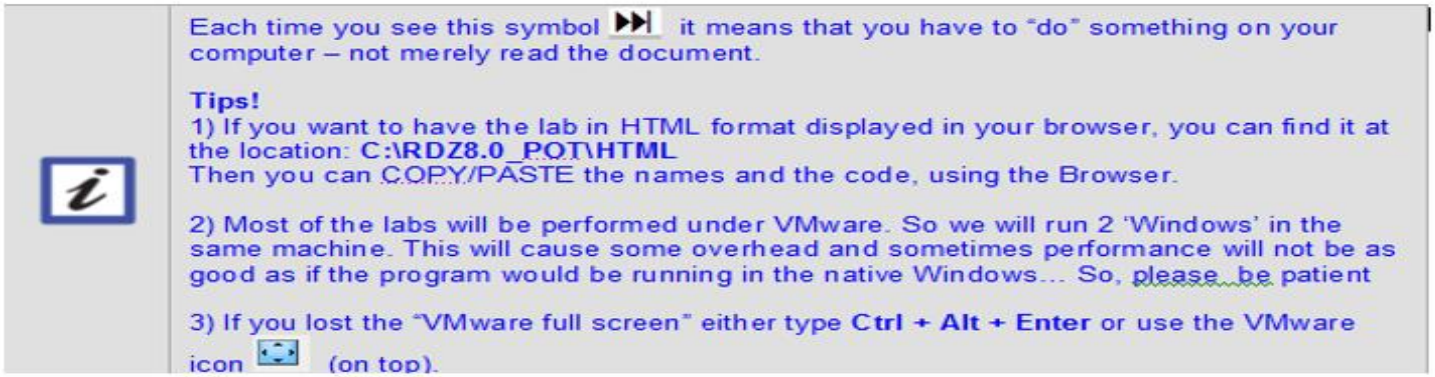
DevOps Mainframe Tooling - Labs

Labs

Labs using VMware …

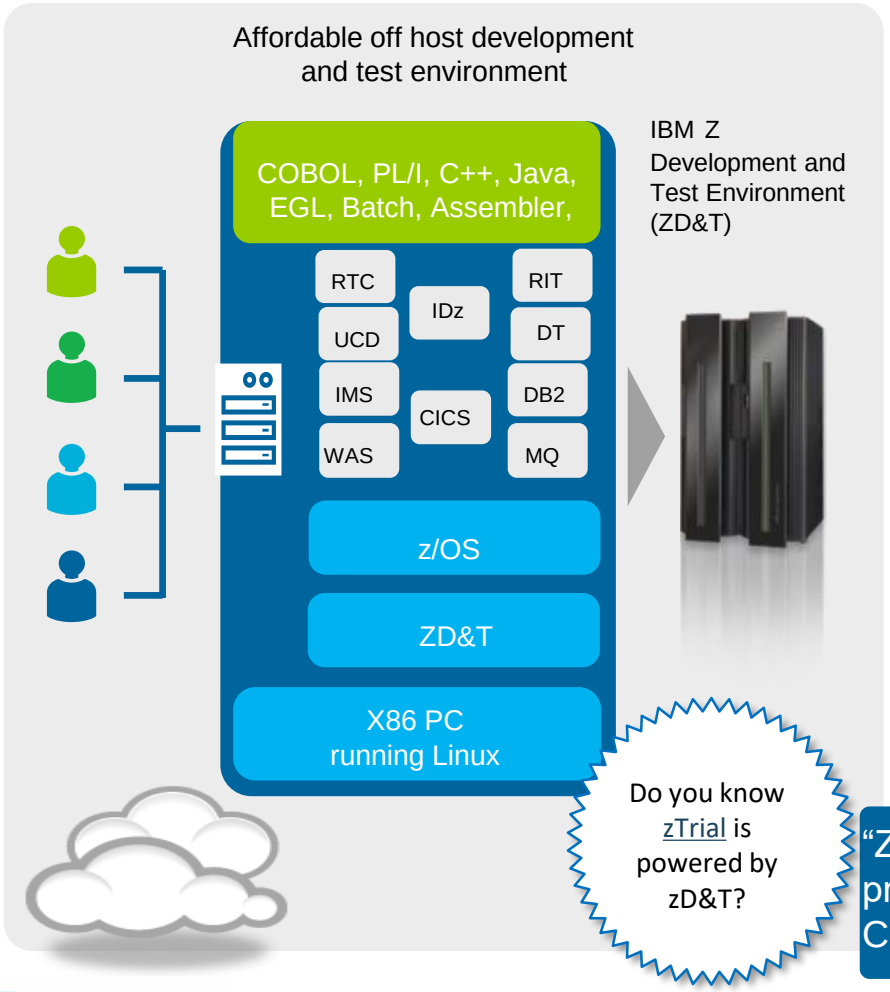
1. User id → empot01 and password → empot01

2. Follow exactly what is described in the lab



3. If you could not complete the lab don't get frustrated…

ZD&T → Dev/Test Environment



- Develop and test z/OS applications anywhere, anytime
- Free up mainframe development MIPS for production workload
- Eliminate costly delays by reducing burden on existing IT operations staff
- Reduce time to value and minimize ongoing administration and capital expense with zD&T Cloud Managed DevOps
- Exploit the z14 hardware capability, including z14 pervasive encryption
- Comprehensive z/OS 2.3 software distribution:
  - z/OS plus major subsystems
  - Underpinned by the z/OS components of DevOps for the Enterprise development, test, and deployment tooling
  - CICS subsystem pre-provisioned

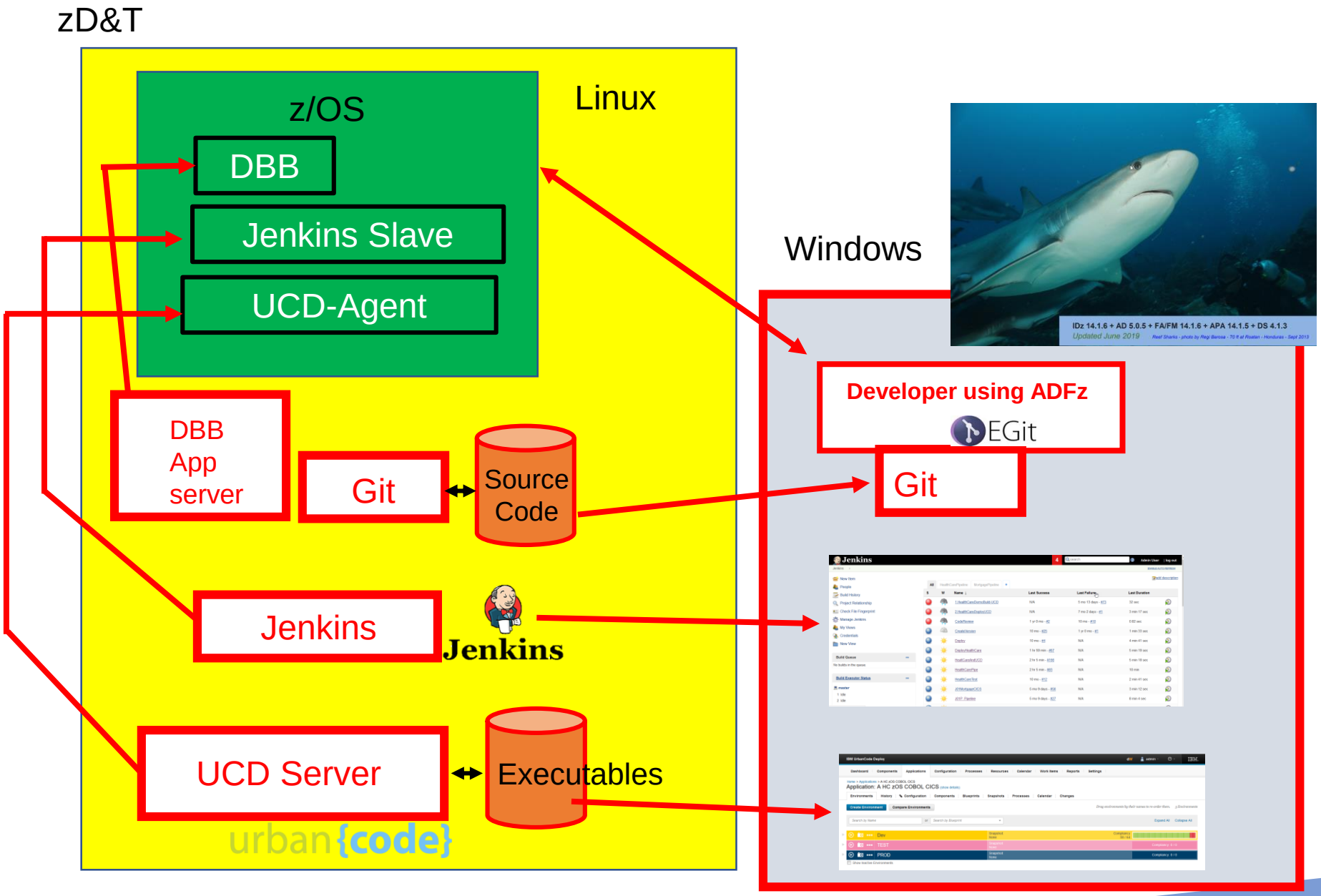
New in V12: Cloud Integration & Deployment Automation

"ZD&T improved our development and testing timeline and provided stability and quality" Developer, Large Enterprise Computer Services Company

https://www.techvalidate.com/tvid/C99-3E2-1ED

Note: This Program is licensed only for development and test of applications that run on IBM z/OS. The Program may not be used to run production workloads of any kind, nor more robust development workloads including without limitation production module builds, pre-production testing, stress testing, or performance testing.

zD&T on Cloud environment



Lab 1: Working with mainframe using COBOL and DB2

Duration: 90 Minutes

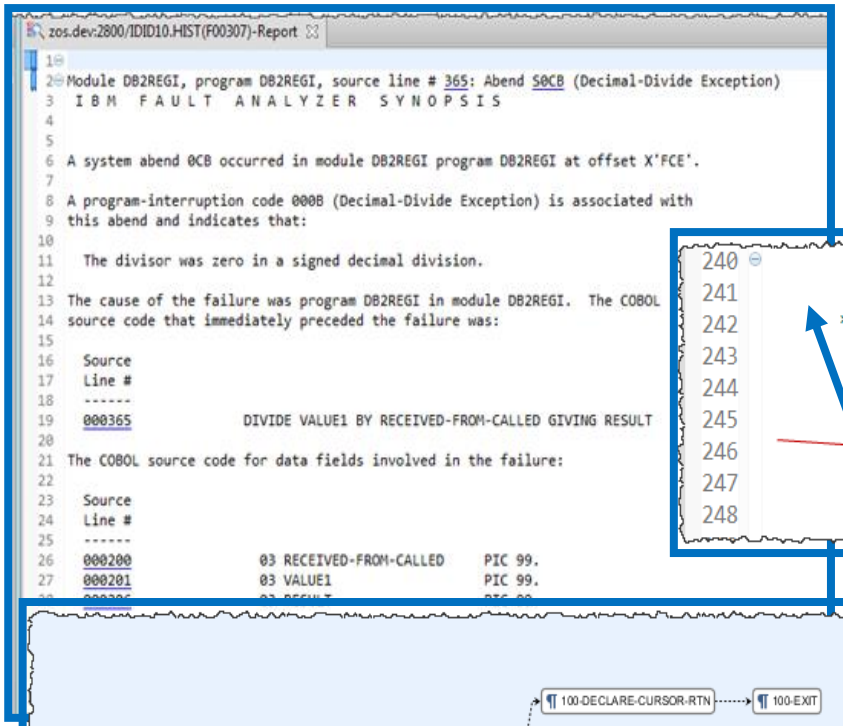
Objective

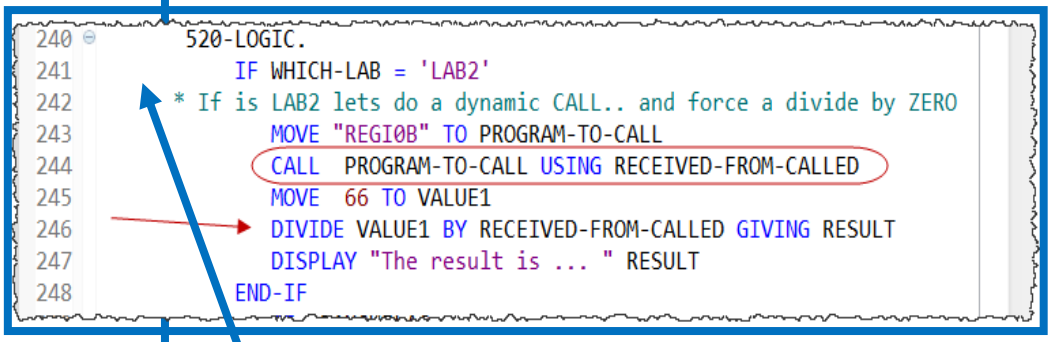
This lab will take you through the steps of using the **Application Delivery Foundation for z Systems (ADFz)** to work with a z/OS system. It will familiarize you with some of the capabilities of this product using a DB2 COBOL batch program that is ABENDing.

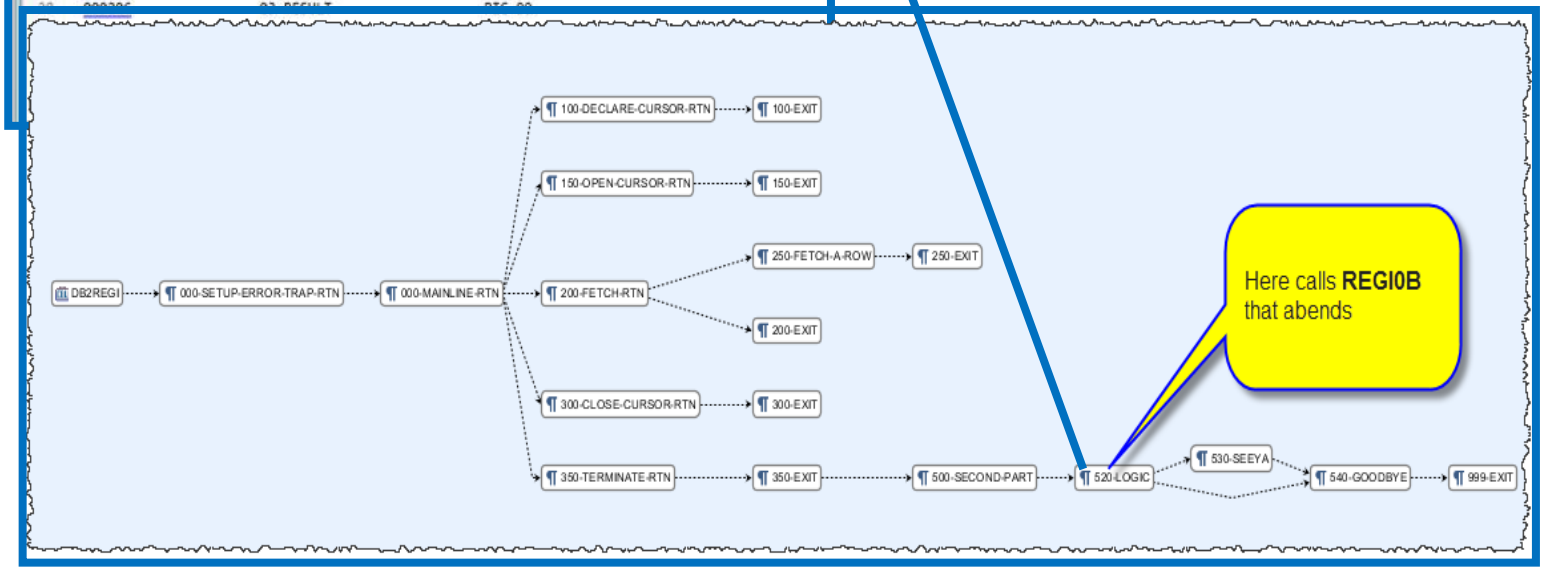
Key Activities

1. **Connect to a z/OS System.**
   Use your Workspace to connect to the z/OS system. Each student will have an unique z/OS userid.
2. **Execute the DB2/COBOL batch program and verify the ABEND.**
3. **Use Fault Analyzer to identify the cause of the ABEND**
4. **Use the IBM Debug for a temporary fix**
   You will modify the field content to bypass the bug
5. **Modify the COBOL code to fix the bug.**
6. **Use Code Coverage**
7. **(Optional) Execute SQL statement when editing the program.**
   5. While editing the COBOL/DB2 program you will be able to execute SQL statements and verify the results.
8. **(Optional) Using File Manager**
   An example of using File Manager against a VSAM file

Lab 1: Working with mainframe using COBOL and DB2







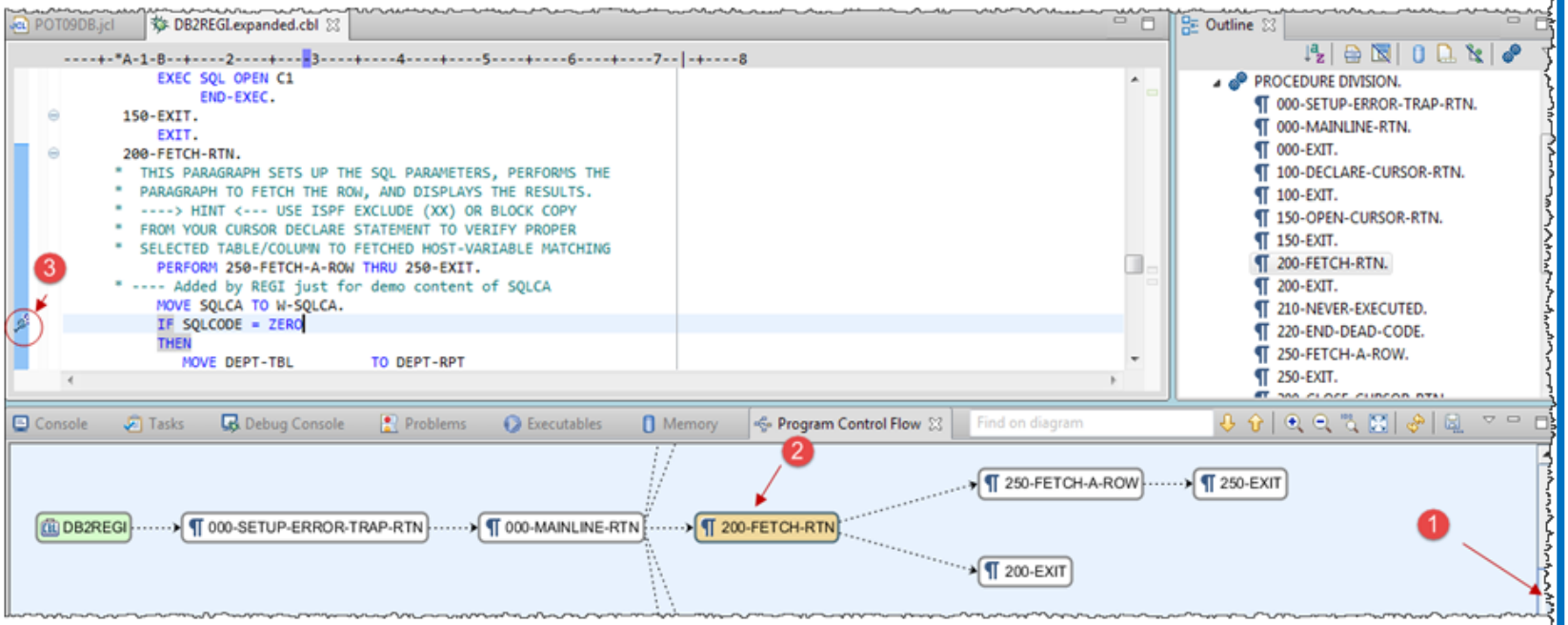
Lab 1: Don’t Miss… The Debug… Page 20

6.2.5 The *Program Control Flow* diagram opens

⏭ On *Program Control Flow* view, (1) **scroll down** and click on (2) **200-FETCH-RTN**

⏭ On the COBOL editor move the mouse to the blue column on left and (3) **double click** on the line `IF SQLCODE = ZERO` to create a breakpoint.

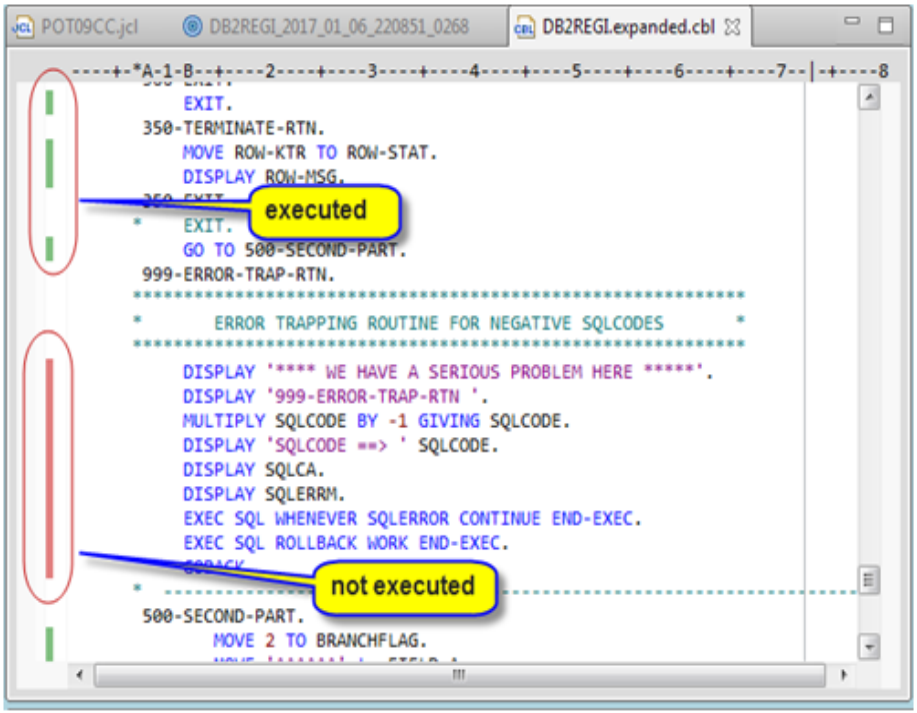
The icon is shown



Lab 1: Don’t Miss… Code Coverage …Page 47

8.2.4 Double click on **DB2REGI.expanded.cbl**

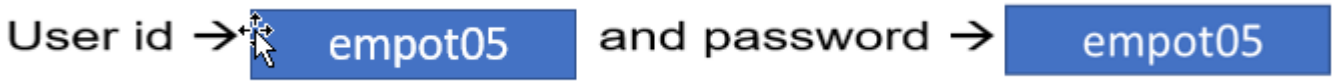
Scroll down and you will see the lines executed in **green** and the lines not executed in **Red**.



Lab 7: Using IBM Dependency Based Build with Git, Jenkins and UCD on z/OS

This lab will use IBM Dependency Based Build (DBB) along with Git, Jenkins and UrbanCode Deploy (UCD) on z/OS.
You will modify an existing COBOL/CICS application stored on Git.
You will use ADFz to change the code and perform a personal test for later delivery and commit to Git and then use Jenkins for the final build and continuous delivery.
The updated code will be deployed to CICS using UrbanCode Deploy (UCD)
.

Overview of development tasks



1. **Get familiar with the application using the 3270 terminal**
   → You will start a 3270 emulation and execute a transaction named **JxxP** to become familiar with the Application that you intend to modify.

2. **Load the source code from Git to the local IDz workspace**
   → You will load the COBOL code that is stored on Linux to your windows client to be modified.

3. **Modify the COBOL code using IDz.**
   → Using IDz you will modify the COBOL code to have a different message in a CICS dialog.

4. **Use IDz DBB User Build to compile/bind and perform personal tests.**
   → You will compile and link the modified code using the DBB User Build function available on IDz EE or ADFz. When complete you will run the code using CICS for a personal test and verify that the change is correctly implemented.

5. **Push and Commit the changed code to Git .**
   → You will commit the changes to Git.

6. **Use Jenkins with Git plugin to build the modified code**
   → You will use Jenkins pipeline to build the new changed code and push the executables to be deployed using UCD.

7. **Use Jenkins and UCD plugin to deploy results and test the code again**
   → You will verify the results after the final deploy to CICS using UCD

8. **(Optional) Understanding DBB Build Reports**
   → You will understand the reports generated by DBB during the build

Agenda

9:00 Introductions
9:20 DevOps for Z

10:00 Break

10:15 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z

11:45 Lunch

12:30 zUnit Overview

12:40 Demo: Scenario using Z DevOps solutions including zUnit, Git and Jenkins

1:30 z/OS DevOps Testing Automation & Virtualization

2:00 – 4:30 Choose one Optional lab:

→ LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
→ LAB 6 : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
→ LAB 2C Application Discovery : Find a candidate API from Catalog Manager Application
→ LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
→ LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
→ LAB 8 - Using Application Performance Analyzer (APA)

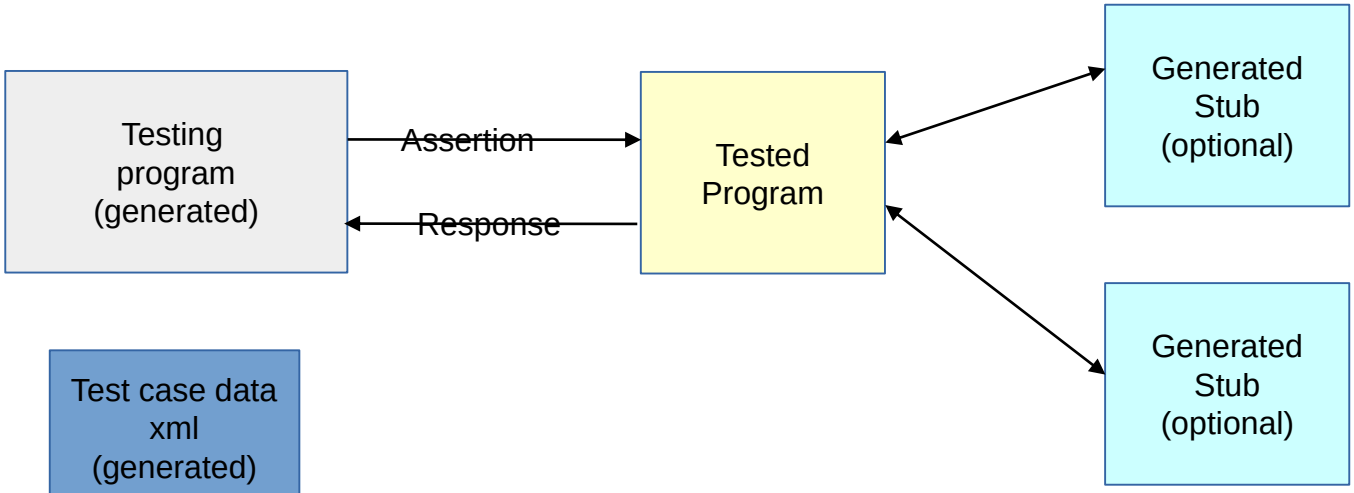
Lecture Labs Breaks

What is Unit Testing?

- A Unit is the smallest practical piece of software to be tested
  - Object oriented languages may treat a Class or Method as a unit
  - Procedural languages may treat a program as a unit

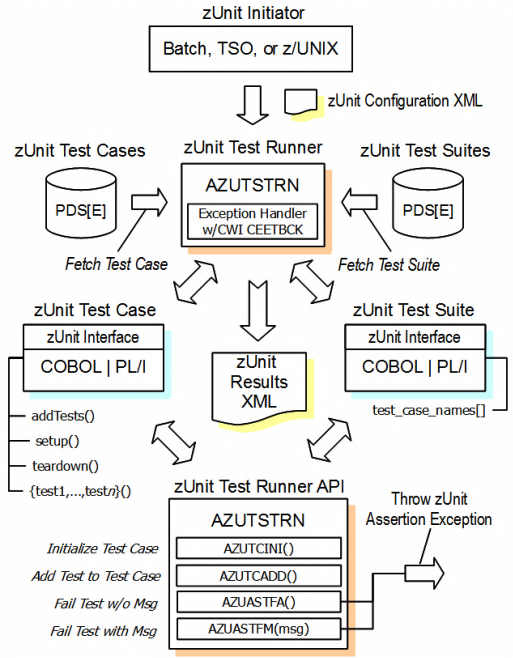
- Unit Testing is a method of testing software units to determine if they are fit for use
  - Unit Tests are usually created by developers, using development tools and unit testing frameworks
  - Unit Tests are expected to run quickly, so they can be run repeatedly
  - Some developers prefer that unit tests isolate the unit to be tested from other units and resources

IDz zUnit Basics



Test case data xml (generated)

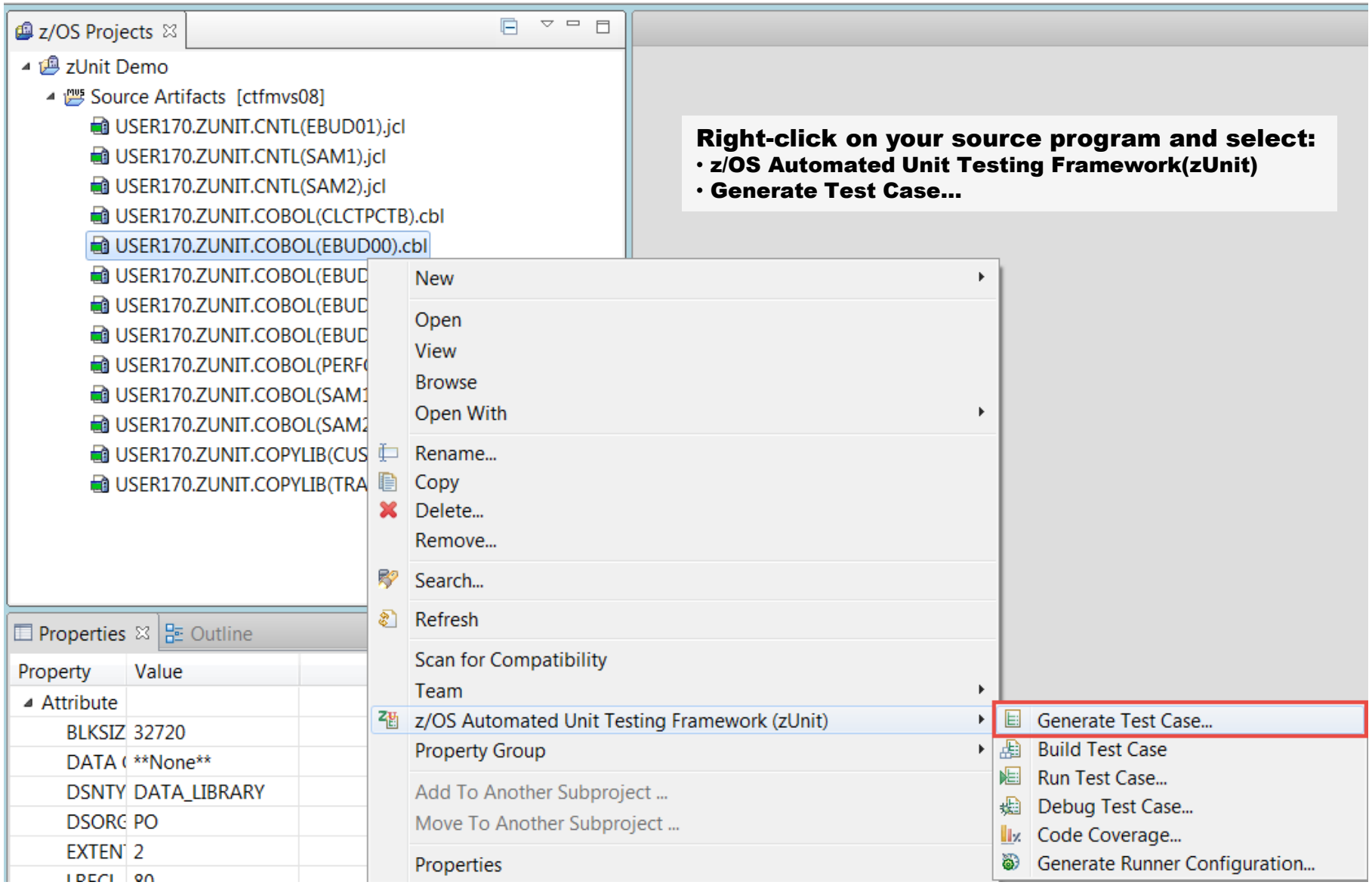
Test case results xml (generated)



zUnit Basic Workflow

1	2	3	4	5	6
Generate Test Case Wizard	Test Data Entry Editor	Generate Test Case Program	Build Test Case	Run Test	Examine Test Results

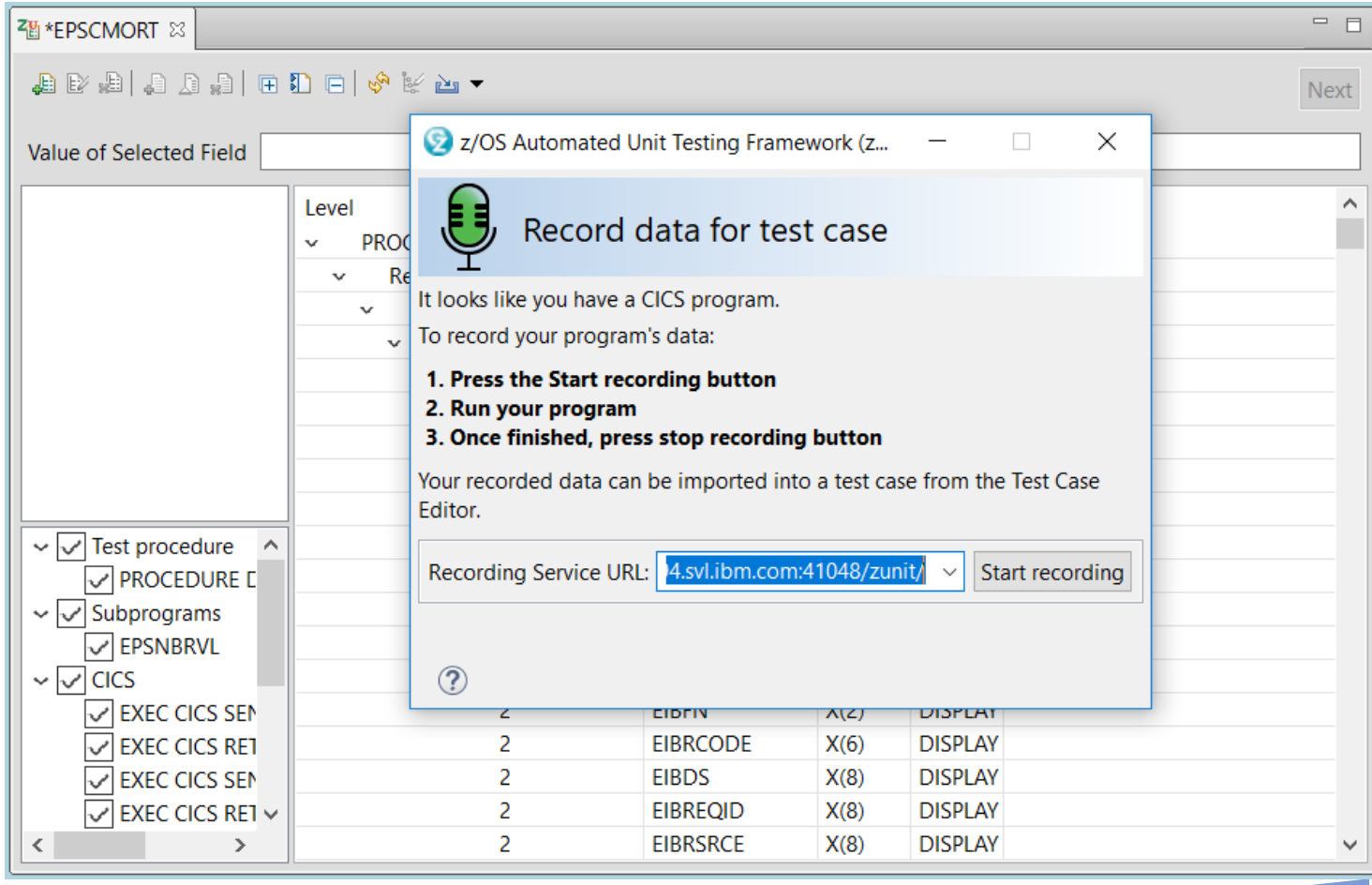
Generating the Test Case - Using the IDz Wizard



IBM zUnit for CICS programs

- Start Recording
- Execute the program in CICS
- Stop Recording
- Import recorded data in Test Case
- Generate Test Case
- Build Test Case
- Run Test Case
  - Outside of CICS

Currently COBOL only



Agenda

9:00 Introductions
9:20 DevOps for Z

10:00 Break

10:15 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z

11:45 Lunch

12:30 zUnit Overview

12:40 Demo: Scenario using Z DevOps solutions including zUnit, Git and Jenkins

1:30 z/OS DevOps Testing Automation & Virtualization

2:00 – 4:30 Choose one Optional lab:
- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6 : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → LAB 2C Application Discovery : Find a candidate API from Catalog Manager Application
- → LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- → LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
- → LAB 8 - Using Application Performance Analyzer (APA)

Lecture Labs Breaks

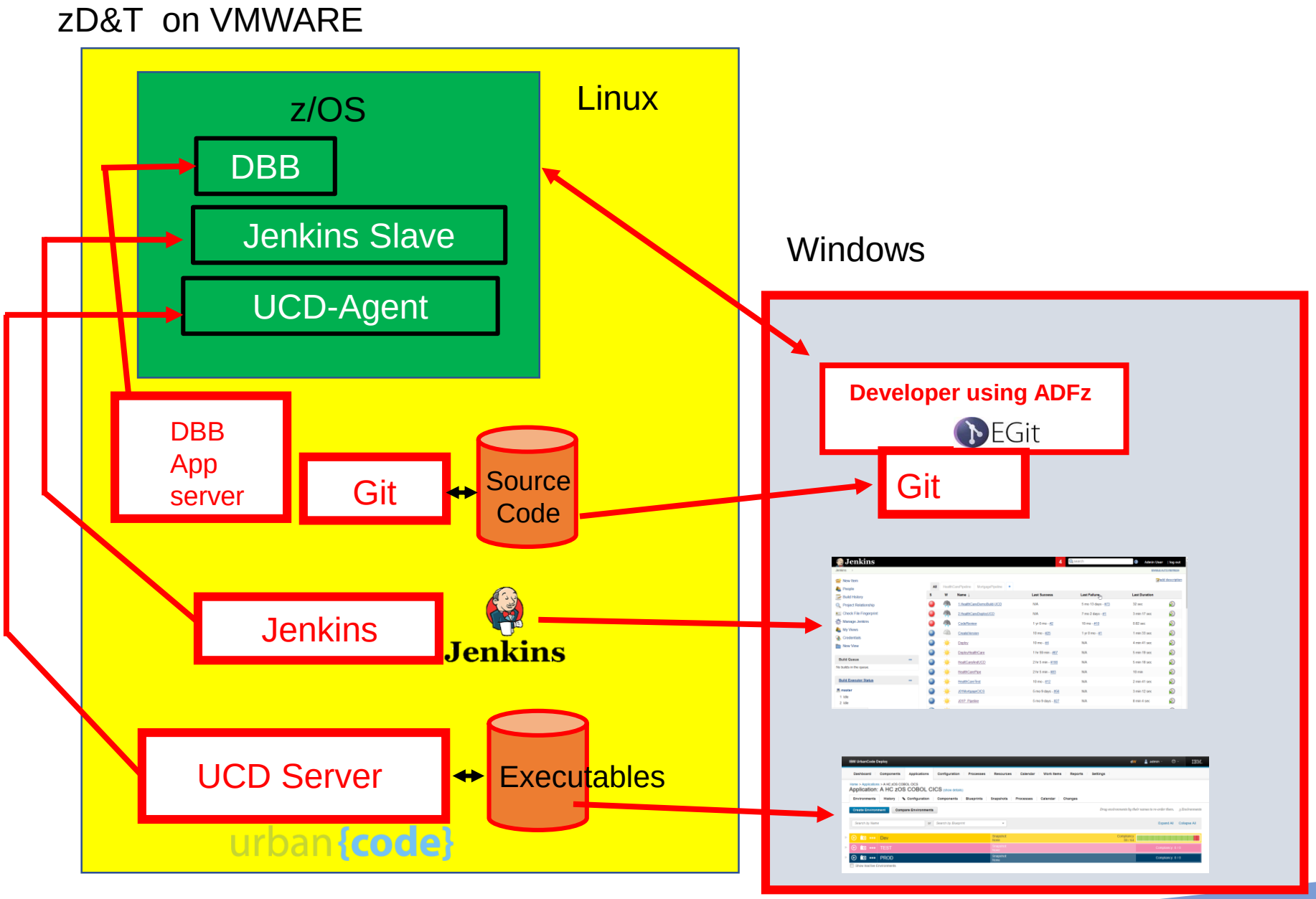
Z DevOps Tooling

Demonstration

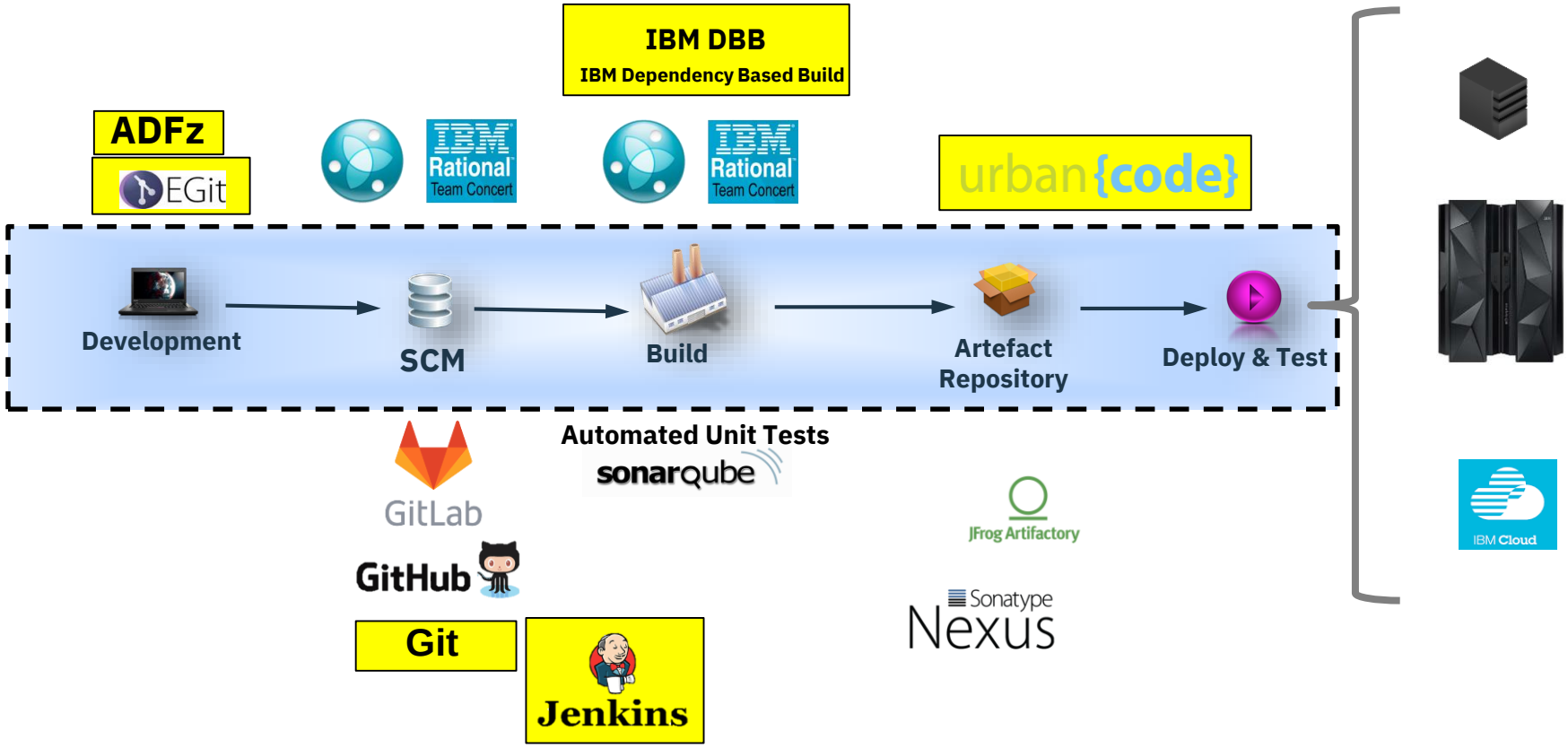
PART 1 - Z Unit Test using IDz (zUnit)

Demonstration

zD&T Topology for ADFz/Git/Jenkins/UCD



Open Pipe Line across the Enterprise



AD = Application Discovery
ADFz = Application Delivery Foundation for z Systems
IBM DBB = IBM Dependency Based Build
urban{code} = IBM UrbanCode Deploy (UCD)

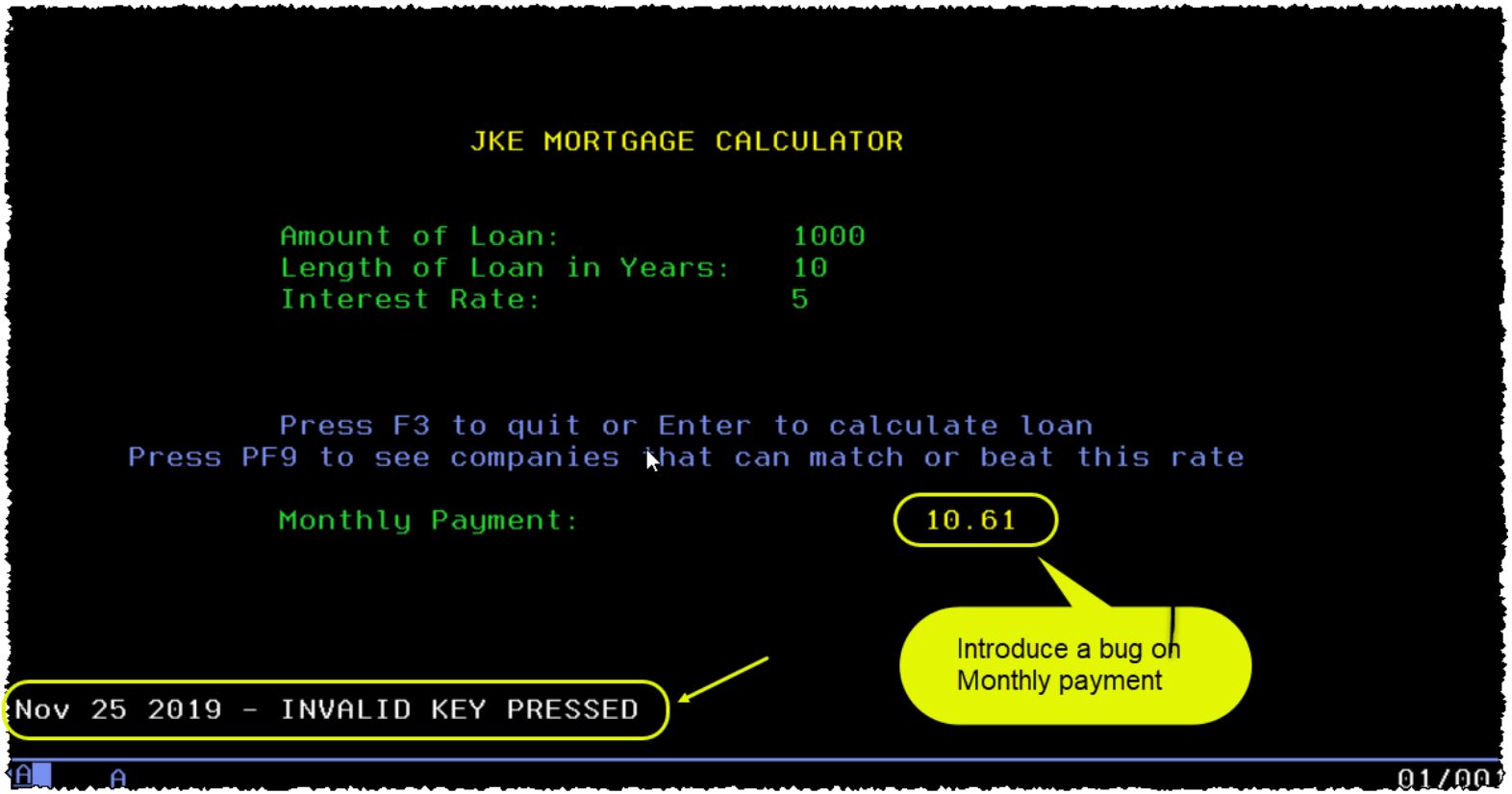
Tools/technology used on this demo

Scenario: COBOL CICS Transaction

*Using transaction **J05P** and pressing **enter** and then **F1***

A payment is calculated and a message is displayed by program **J05CMORT**

➔ **Developer will work on this program and introduce a bug**



Unit test on COBOL/CICS program J05CMORT

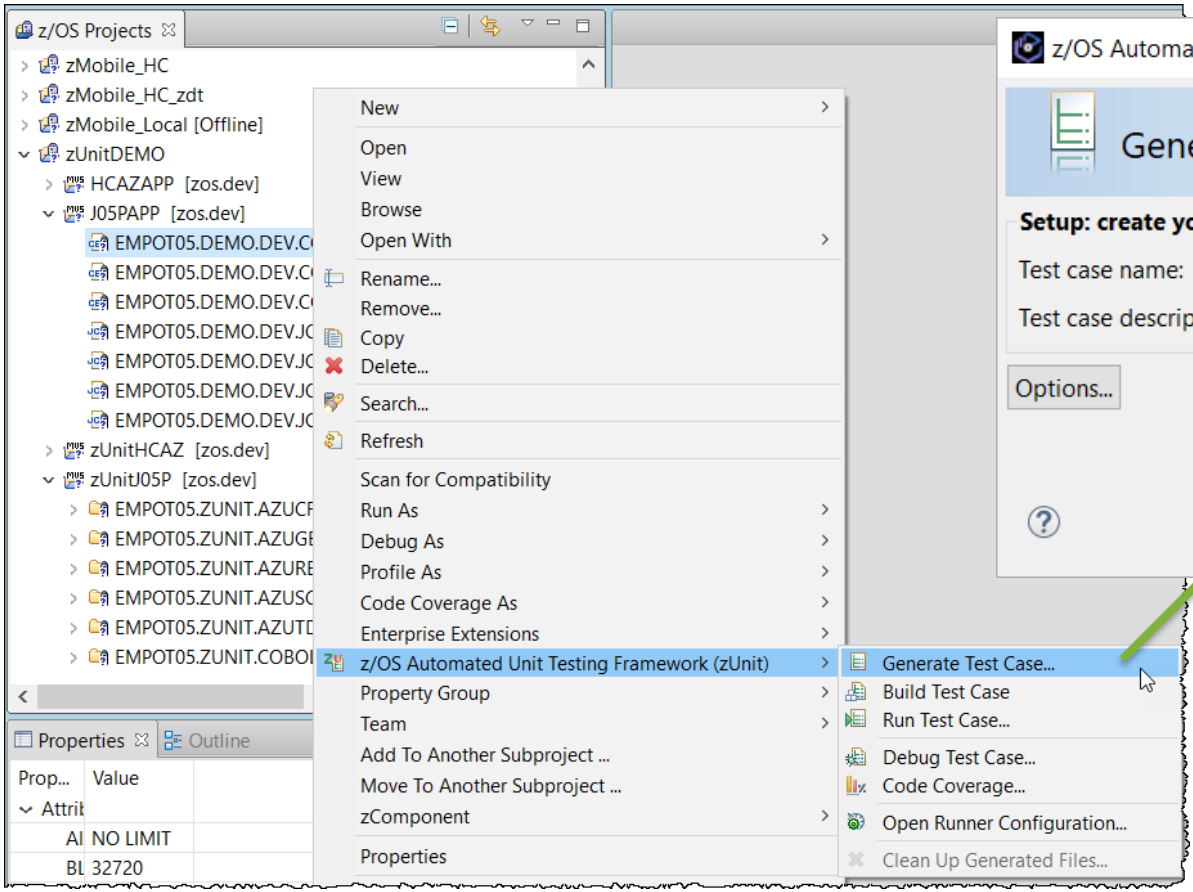
1. Before any change, developer
   record CICS program interactions using **IDz** and **CICS**.

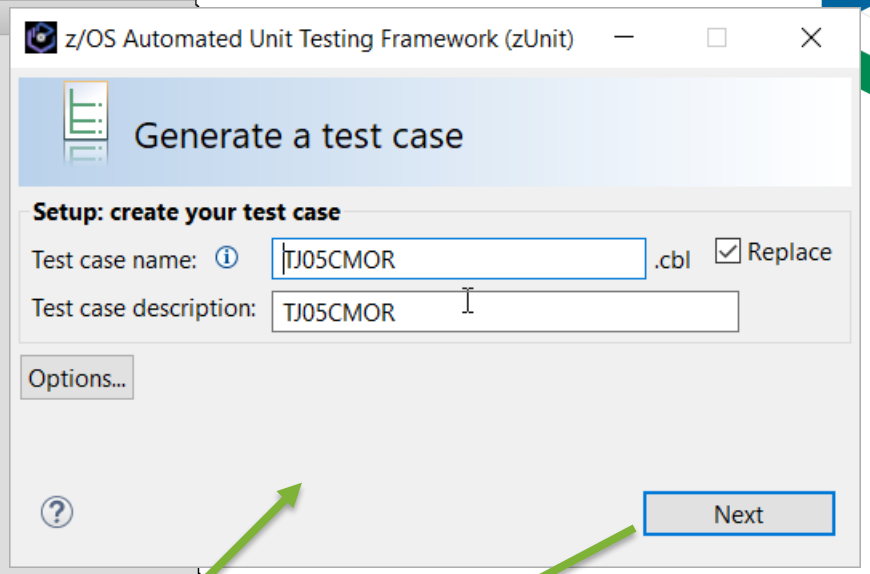
2. Generate, build and run the Z unit test
   Compile and link-edit the generated unit test programs, followed by running the unit test.

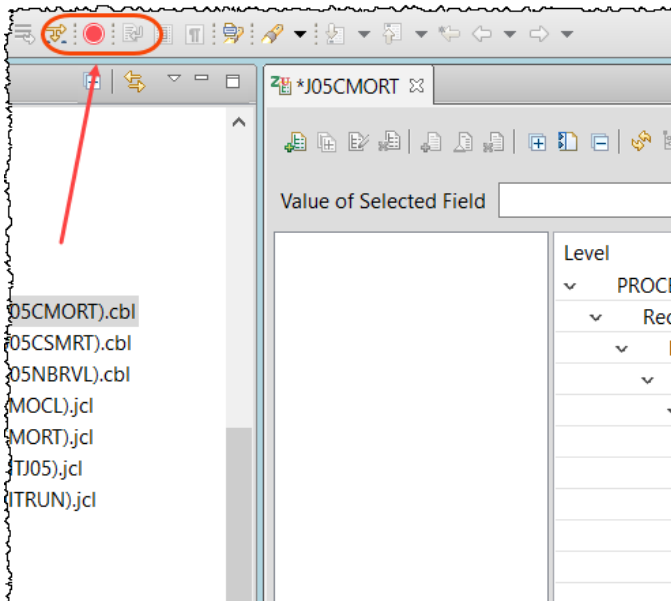
3. Modify the program and rerun the unit test
   Modify the program under test and introduce a bug (bad mortgage value),
   Rerun the unit test, and observe the failure of the test case.
   All done in batch.. No need to start CICS..

4. Use CICS and see the error showing up
   The new value is showing incorrectly (bug introduced).

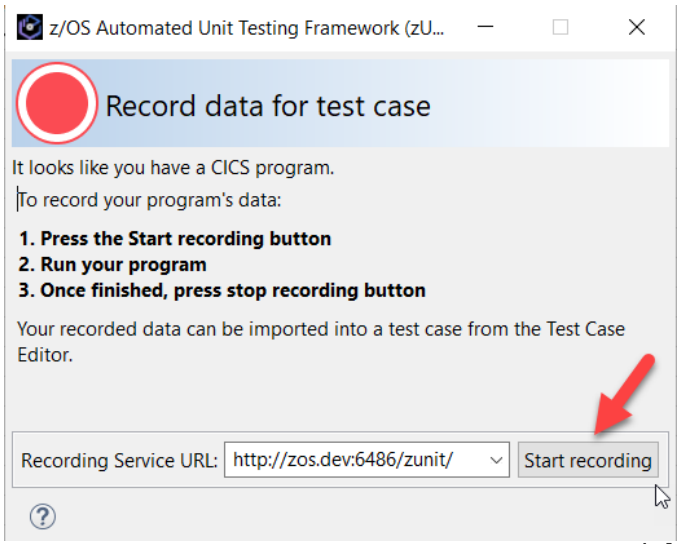
1. Record CICS program interaction using IDz …

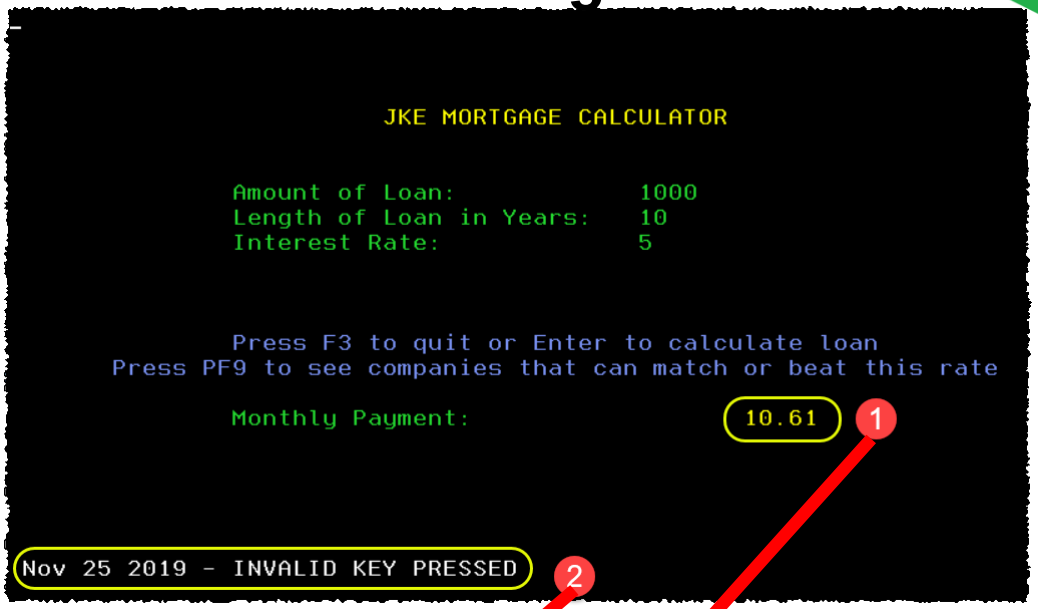


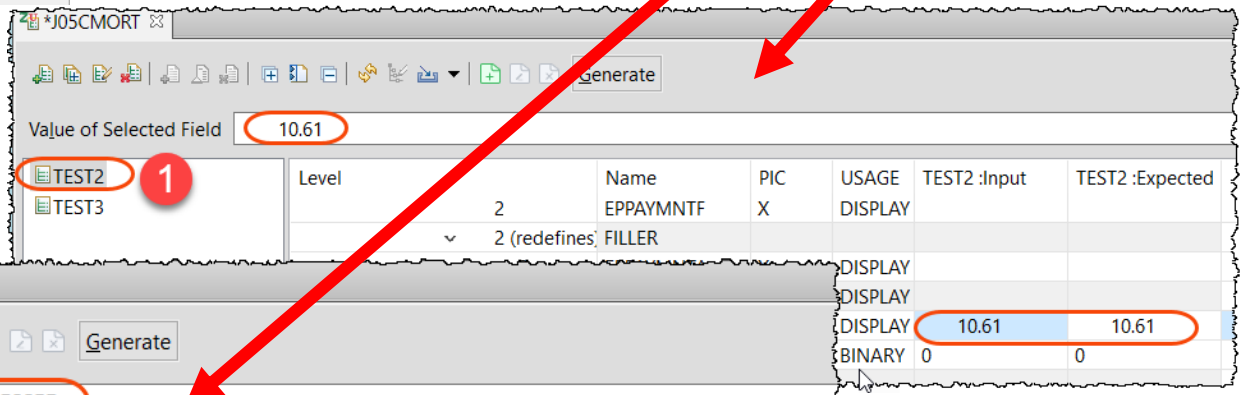


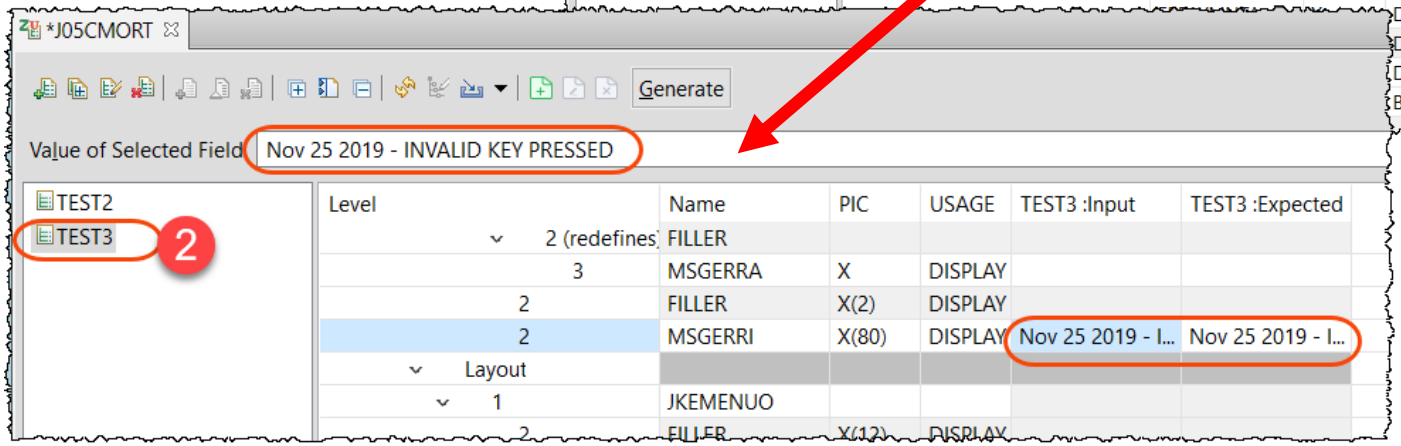


1. Record CICS program interaction using IDz









Unit test on COBOL/CICS program J05CMORT

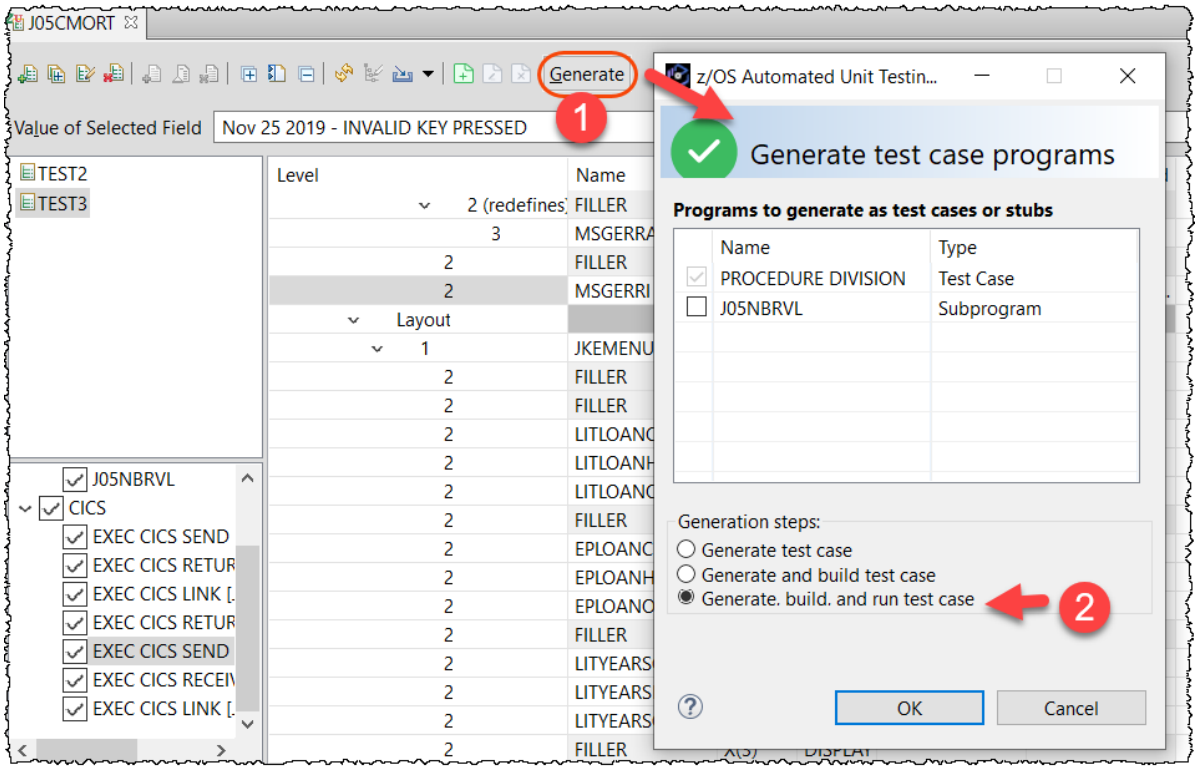
1. Before any change, developer
   record CICS program interactions using **IDz** and **CICS**.

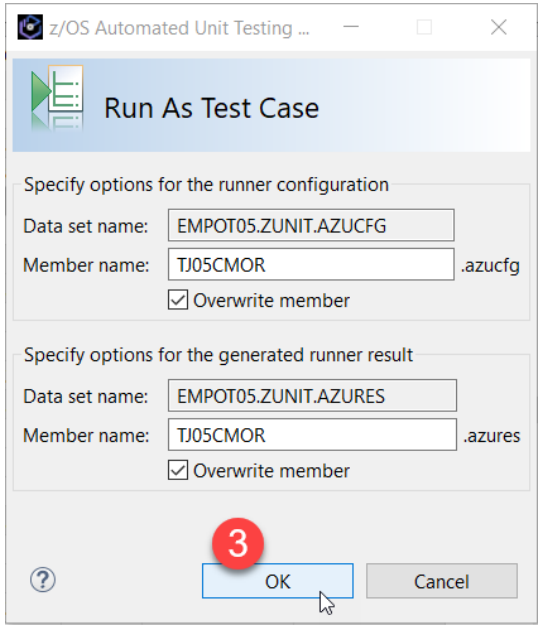
2. Generate, build and run the Z unit test
   Compile and link-edit the generated unit test programs, followed by running the unit test.

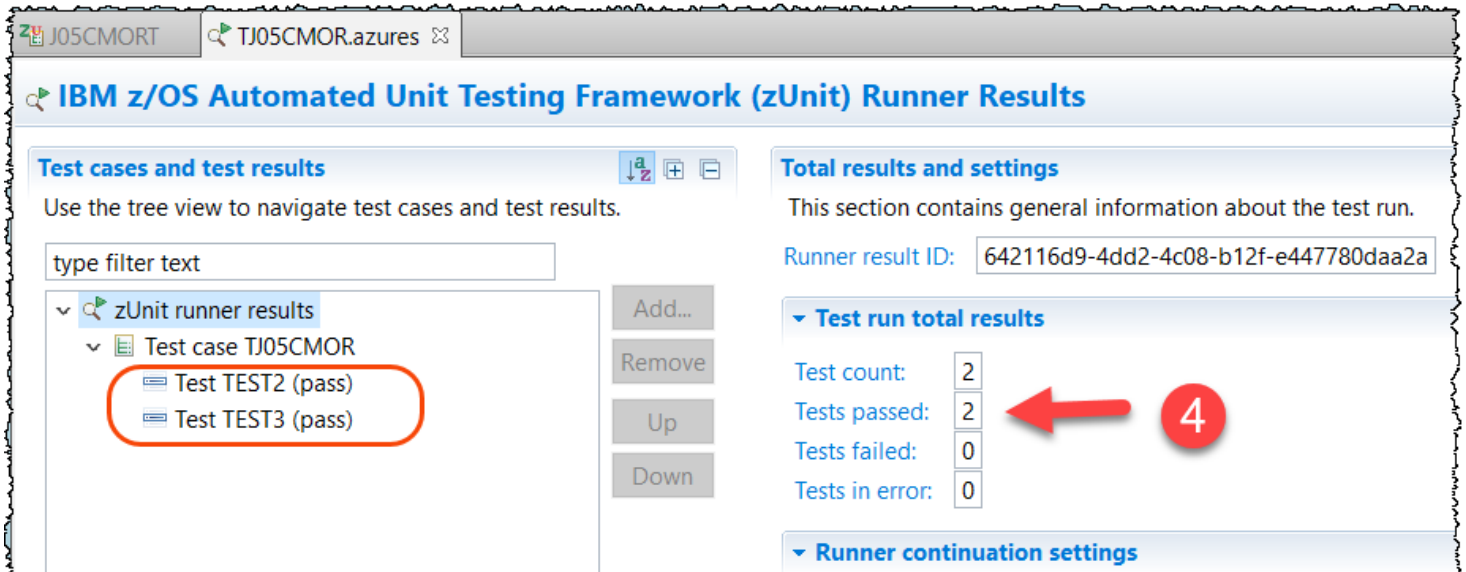
3. Modify the program and rerun the unit test
   Modify the program under test and introduce a bug (bad mortgage value),
   Rerun the unit test, and observe the failure of the test case.
   All done in batch.. No need to start CICS..

4. Use CICS and see the error showing up
   The new value is showing incorrectly (bug introduced).

2. Generate, build and run the unit test







Unit test on COBOL/CICS program J05CMORT

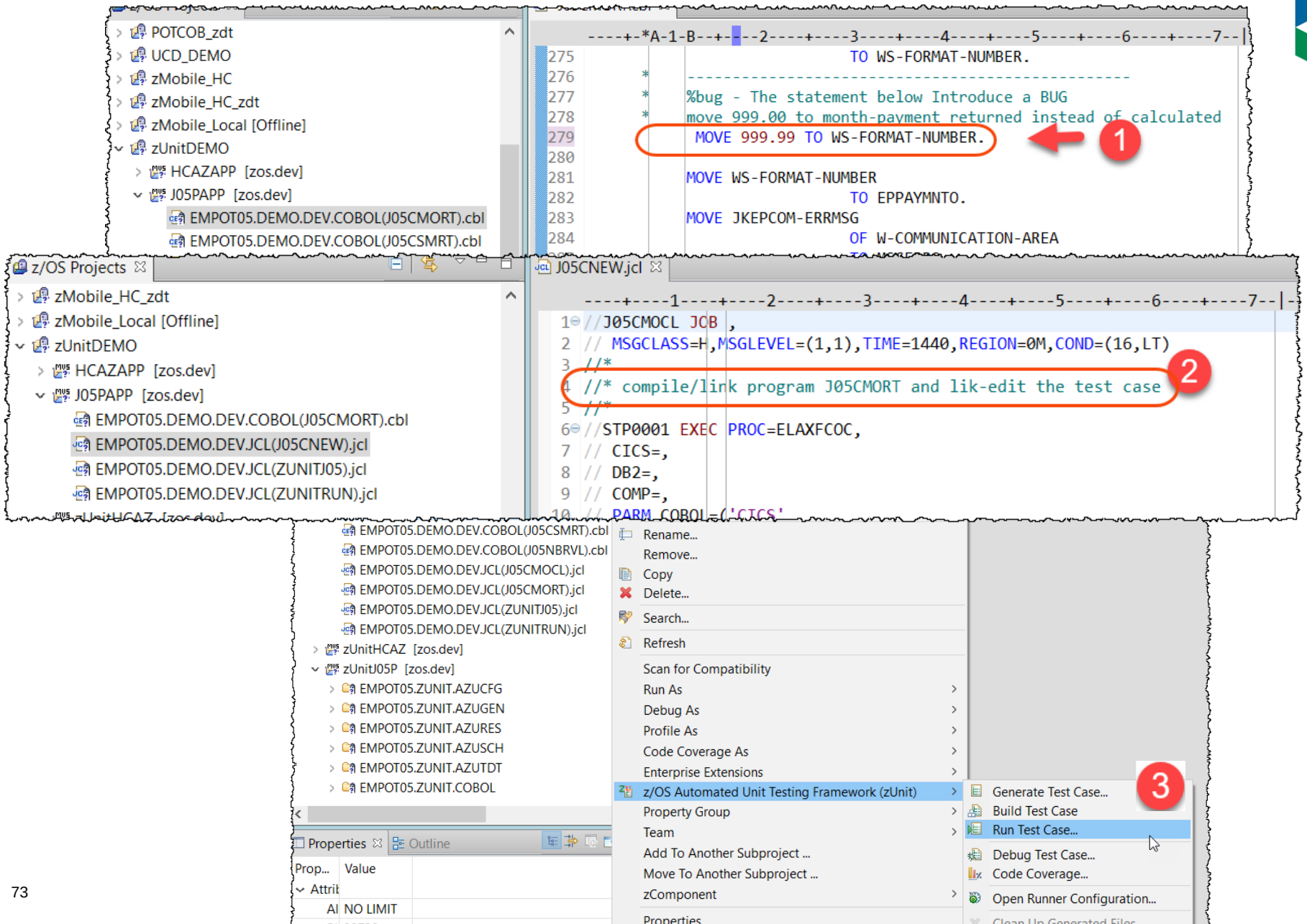
1. Before any change, developer
   record CICS program interactions using **IDz** and **CICS**.

2. Generate, build and run the Z unit test
   Compile and link-edit the generated unit test programs, followed by running the unit test.

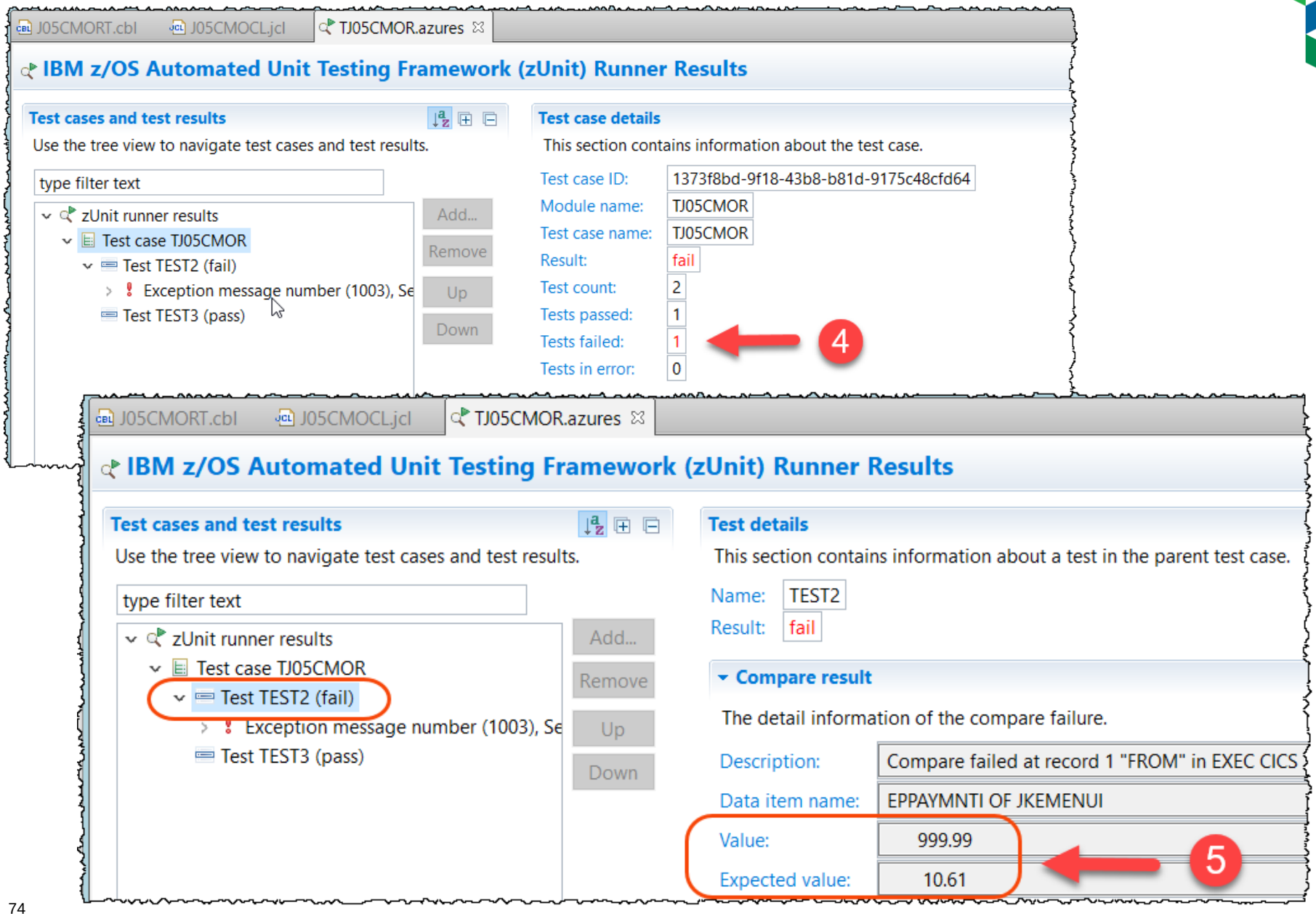
3. Modify the program and rerun the unit test
   Modify the program under test and introduce a bug (bad mortgage value),
   Rerun the unit test, and observe the failure of the test case.
   All done in batch.. No need to start CICS..

4. Use CICS and see the error showing up
   The new value is showing incorrectly (bug introduced).

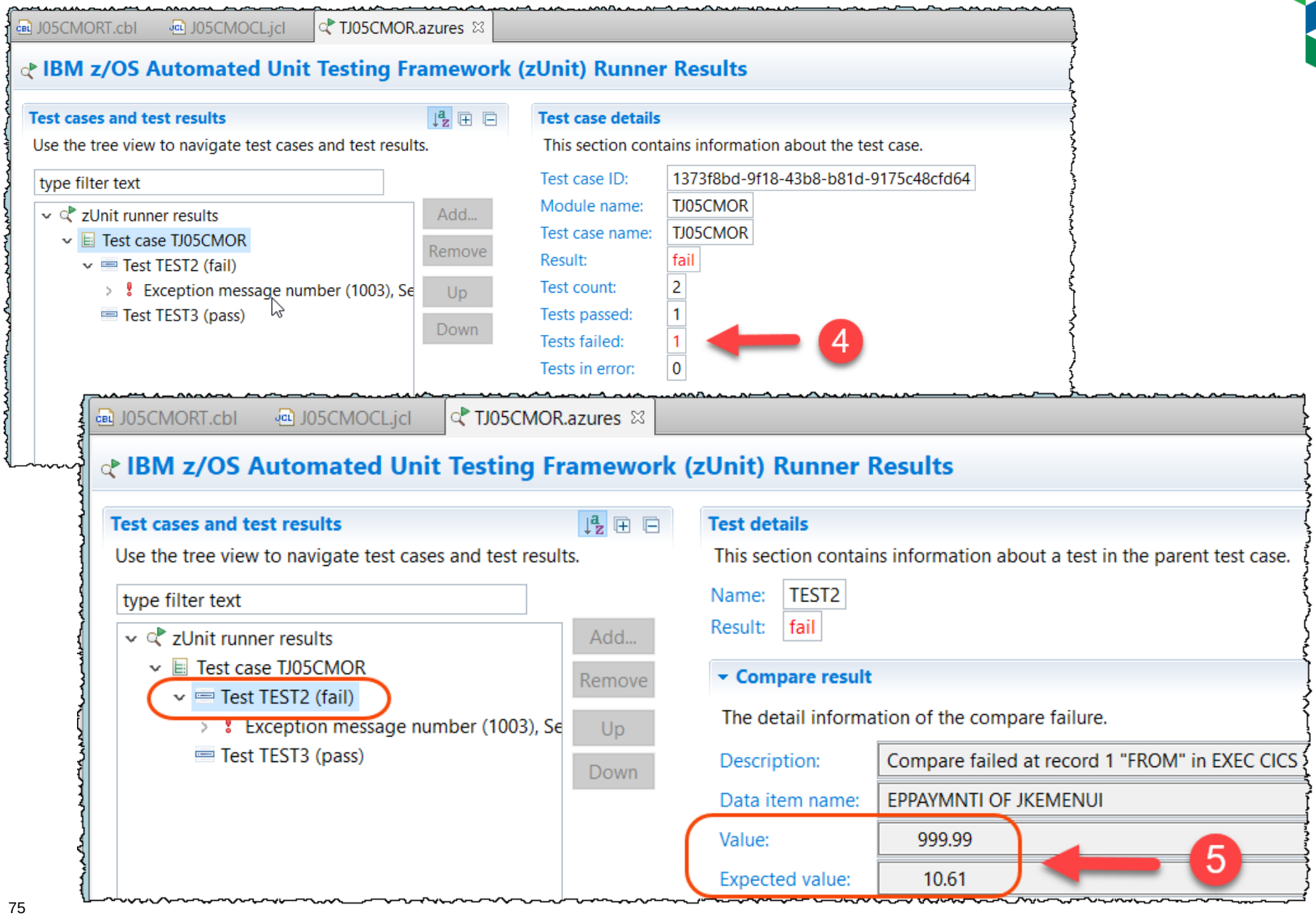
3. Modify the program and rerun the unit test…



3. Modify the program and rerun the unit test



3. Modify the program and rerun the unit test



Unit test on COBOL/CICS program J05CMORT

1. Before any change, developer
   record CICS program interactions using **IDz** and **CICS**.

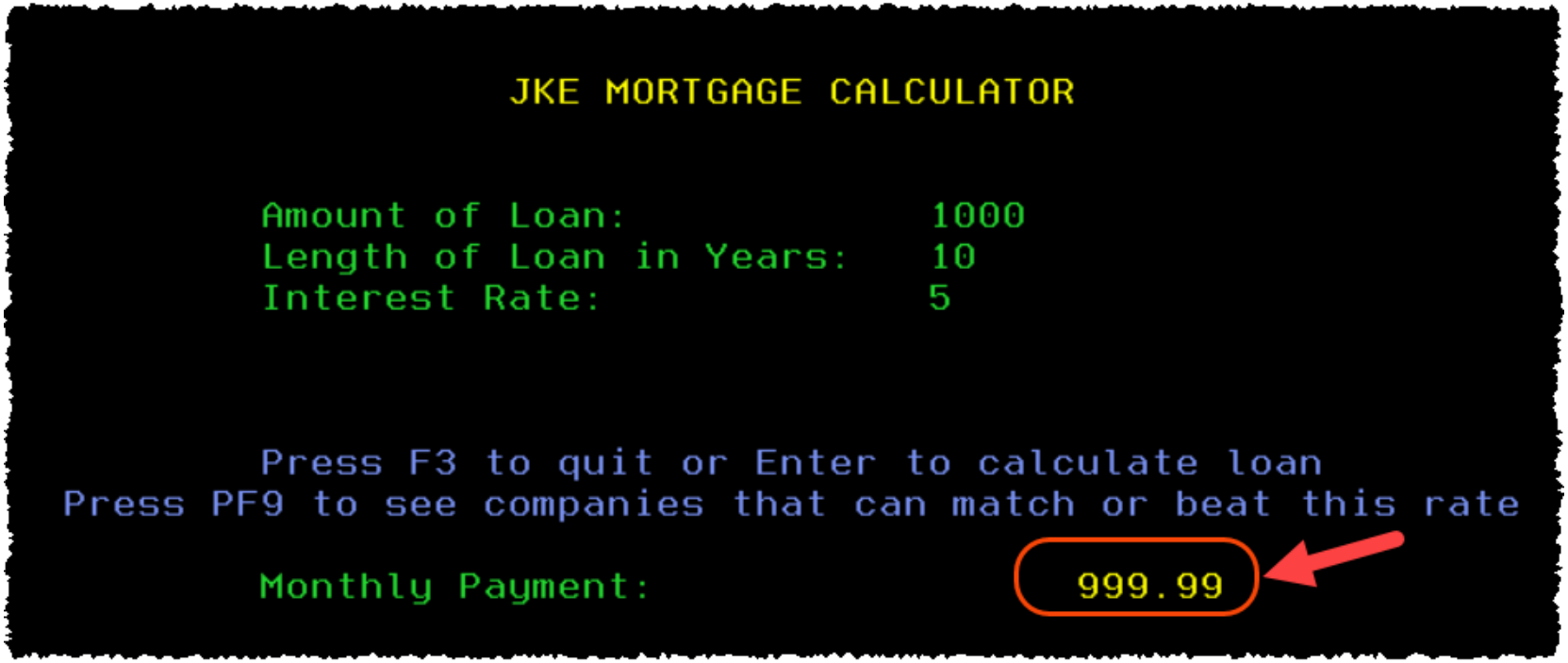
2. Generate, build and run the Z unit test
   Compile and link-edit the generated unit test programs, followed by running the unit test.

3. Modify the program and rerun the unit test
   Modify the program under test and introduce a bug (bad mortgage value),
   Rerun the unit test, and observe the failure of the test case.
   All done in batch.. No need to start CICS..

4. Use CICS and see the error showing up
   The new value is showing incorrectly (bug introduced).

Developer **A** is frustrated.. Push back to Git and go home..

4. Use CICS and see the error showing up



Have done CICS NEWCOPY?

DevOps Mainframe Tooling

Demonstration

Part 2 - Git/DBB/Jenkins/UCD

Demonstration

1

- What is *IBM Dependency Based Build* (**DBB**)?

2

- General architecture of the solution

4

- Git/DBB/Jenkins/UCD in action (demo)

What is IBM Dependency Based Build?

- **IBM Dependency Based Build (DBB)** is a tool to build traditional z/OS applications such as COBOL and PL/I as part of a continuous integration pipeline.
- Provides a modern scripting language based automation capability that can be used on z/OS.
  - The DBB API is written in Java and can be called by Java applications as well as Java based scripting languages such as **Groovy**, JRuby, Jython, Ant, Maven, etc.
- **Not tied to a specific source code manager** or **pipeline automation tool.** (We will use *Git* and *Jenkins* on the demo)
- Consists of a build toolkit (Java API, Groovy Installation) installed on USS (z/OS) and a Liberty application server that hosts build metadata (dependency data, build results) installed on Linux.

DBB Sample Groovy compile step versus JCL

```
import com.ibm.dbb.build.*

println("Copying source from zFS to PDS . . .")
def copy = new CopyToPDS()
             .file(new File("/u/usr1/build/helloworld.cbl"))
             .dataset("USR1.BUILD.COBOL").member("HELLO")
copy.execute()

println("Compiling . . .")
def compile = new MVSExec().pgm("IGYCRCTL").parm("LIB")
compile.dd(new DDStatement().name("TASKLIB")
             .dsn("IGY.SIGYCOMP").options("shr"))
compile.dd(new DDStatement().name("SYSIN")
             .dsn("USR1.BUILD.COBOL(HELLO)").options("shr"))
compile.dd(new DDStatement().name("SYSLIN")
             .dsn("USR1.BUILD.OBJ(HELLO)").options("shr"))
compile.dd(new DDStatement().name("SYSPRINT")
             .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT1")
             .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT2")
             .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT3")
             .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT4")
             .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT5")
             .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT6")
             .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT7")
             .options("tracks space(5,5) unit(vio) new"))
compile.copy(new CopyToHFS().ddName("SYSPRINT")
             .file(new File("/u/usr1/build/helloworld.log")))
def rc = compile.execute()

if (rc > 4)
    println("Compile failed!  RC=$rc")
else
    println("Compile successful!  RC=$rc")
```

```
//COBOL  EXEC PGM=IGYCRCTL,REGION=0M,
//       PARM='LIB'
//*
//STEPLIB  DD DISP=SHR,DSN=IGY.SIGYCOMP
//*
//SYSIN    DD DISP=SHR,DSN=USER1.BUILD.COBOL(HELLO)
//SYSLIN   DD DISP=SHR,DSN=USER1.BUILD.OBJ(HELLO)
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT2   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT3   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT4   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT5   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT6   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT7   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//            BLKSIZE=80,LRECL=80,RECFM=FB
```


1

- What is *IBM Dependency Based Build* (**DBB**)?

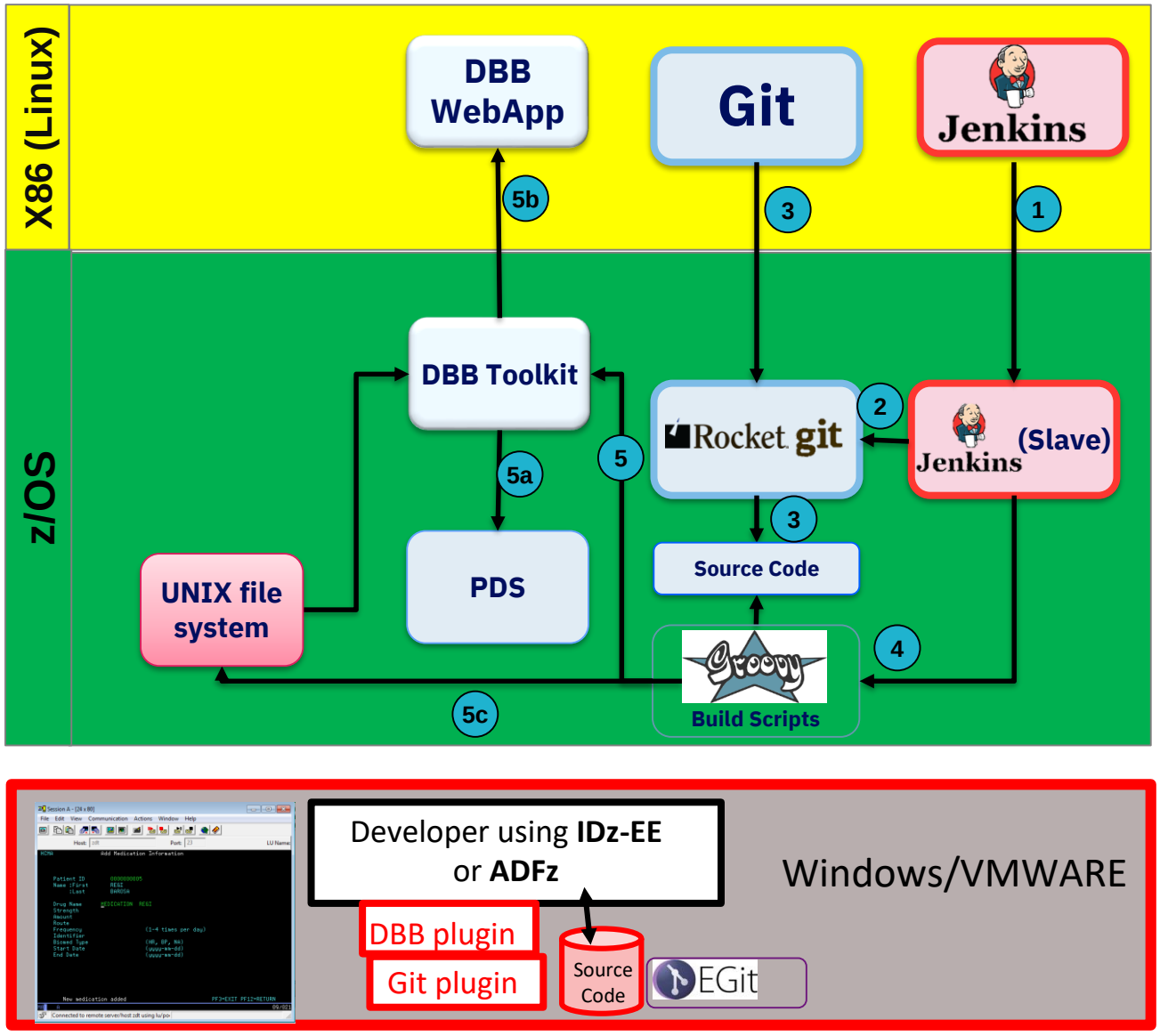
2

- General architecture of the solution

4

- Git/DBB/Jenkins/UCD in action (demo)

Solution High Level Diagram



[1] Jenkins server sends build commands to remote agent.

[2] Jenkins agent issues Git pull command to update local Git repository.

[3] Rocket's Git client automatically converts source from **UTF-8** to **EBCDIC** during pull.

[4] Jenkins agent invokes build scripts containing DBB APIs in local zOS Git repository.

[5] DBB Toolkit provides Java APIs to:

[5a] Create datasets, copy source from **zFS** to **PDS**, invoke zOS programs, issue ISPF and TSO commands.

[5b] Scan and store dependency data from source files, perform dependency and impact analysis, store build results.

[5c] Copy logs from **PDS** to **zFS**, generate build reports (can be saved in build result).

1

- What is *IBM Dependency Based Build* (**DBB**)?

2

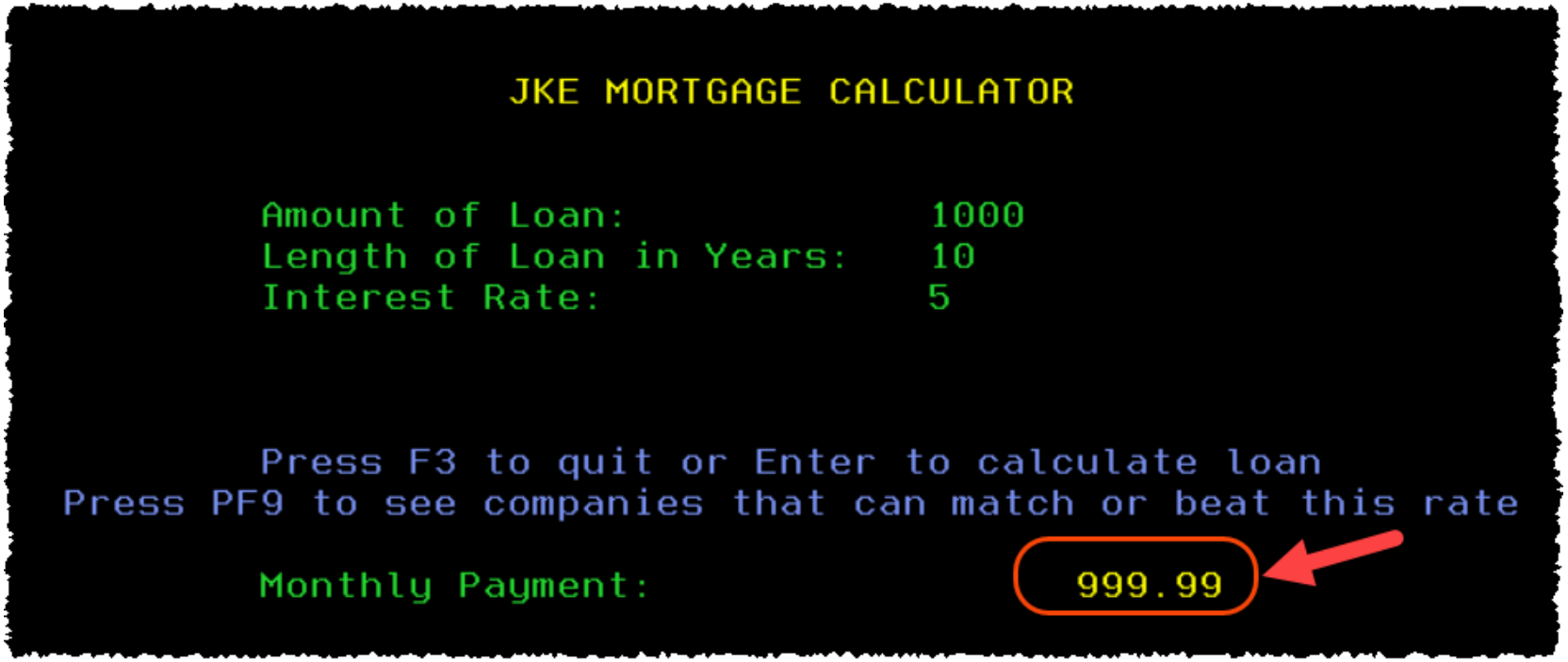
- General architecture of the solution

4

- Git/DBB/Jenkins/UCD in action (demo)

Scenario: Developer will use Git/DBB/Jenkins to fix the bug

*Type transaction **J05P** and pressing enter it shows incorrect bad monthly payment*



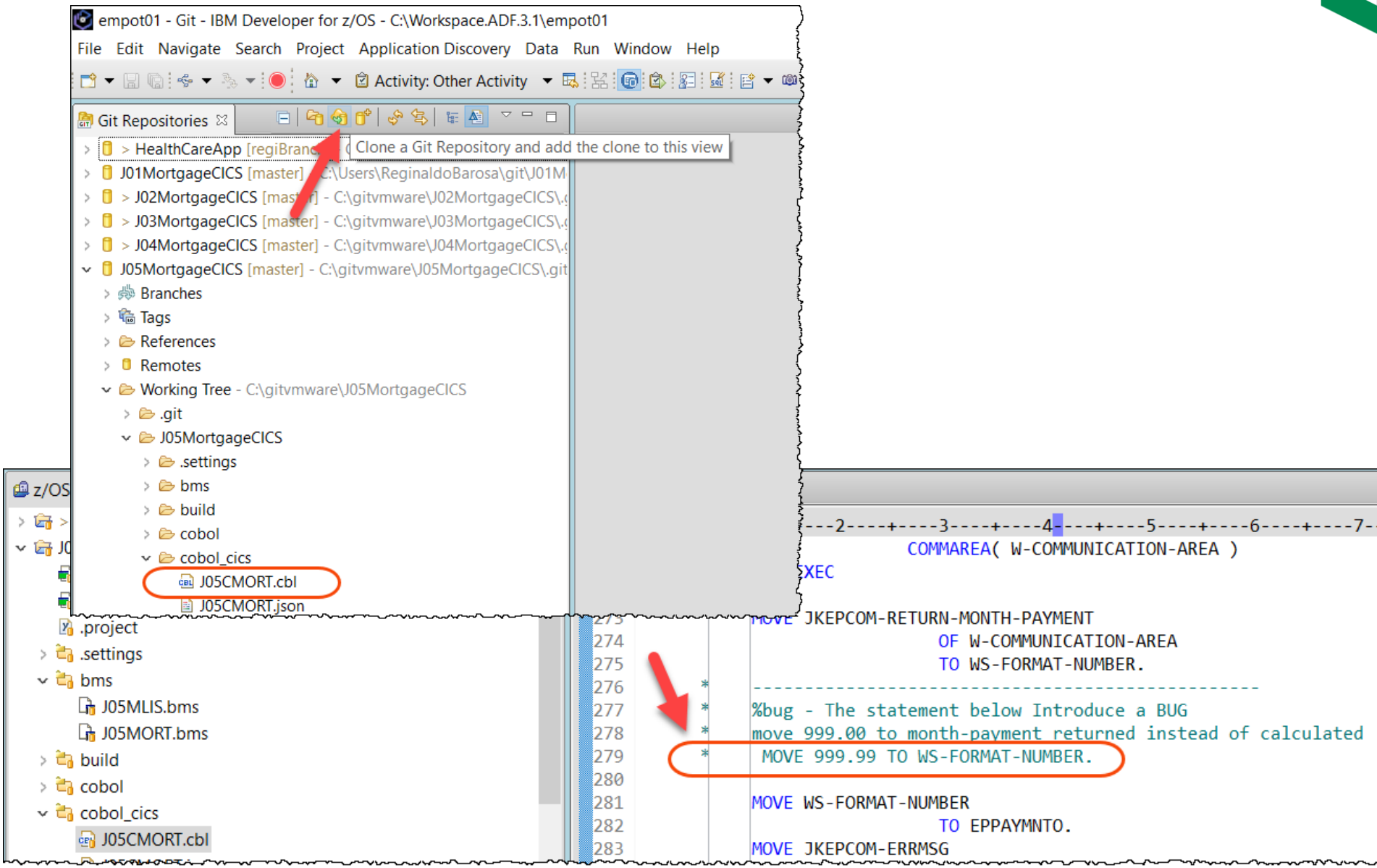
Bug was created by developer **A**.

PART #1 – Fix COBOL/CICS `J05CMORT` broken program

Developer **B** will fix the issue created by developer **A**

1. Uses **IDz** to update the COBOL program stored in **Git**
2. Uses **DBB** to perform a User Dependency Build and verify the code is fixed
3. Uses **Git** to **Commit and Push** the corrected code to Repository

1. Uses IDz to update the COBOL program stored on Git

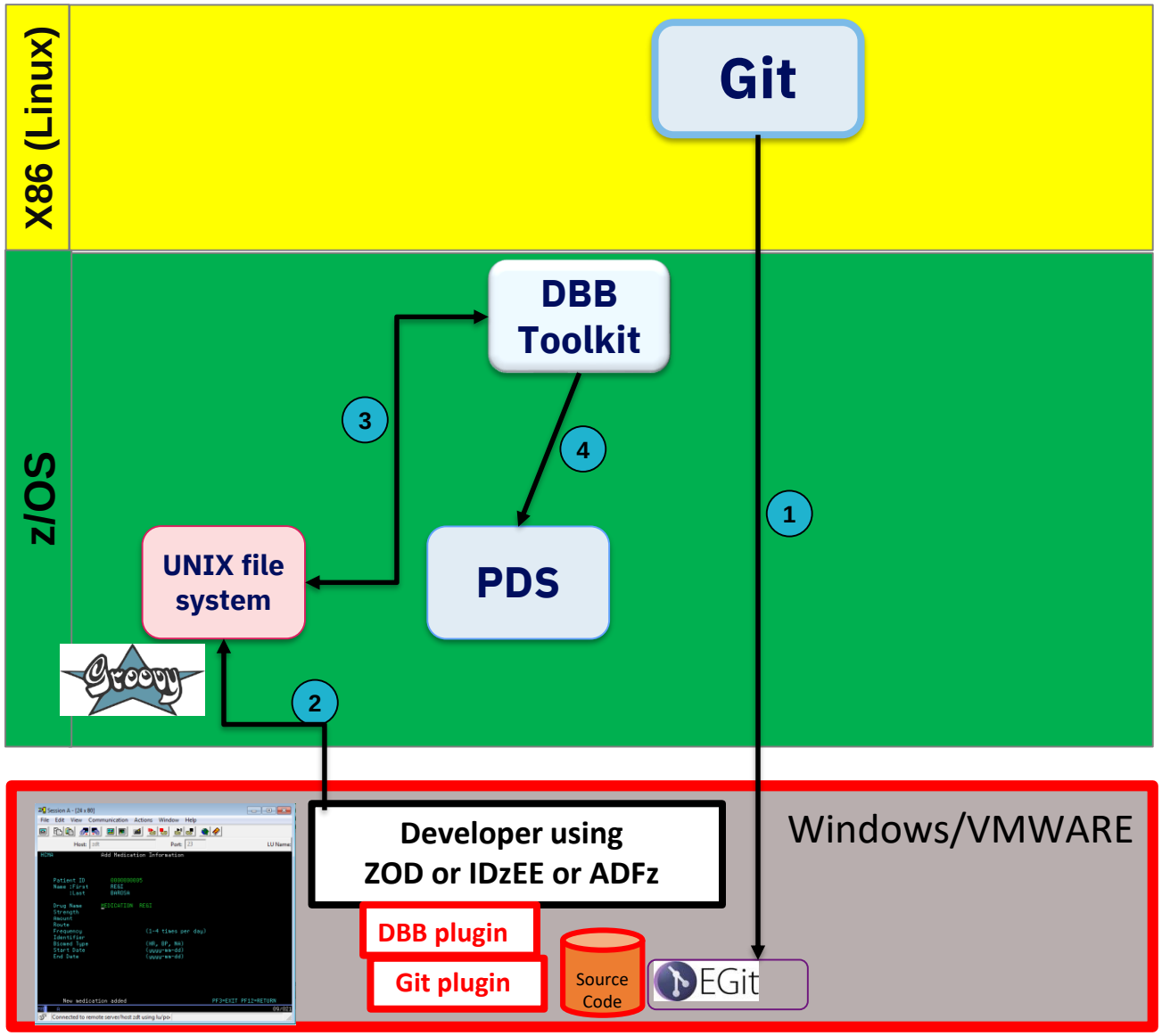


PART #1 – Fix COBOL/CICS `J05CMORT` broken program

Developer **B** will fix the issue created by developer **A**

1. Uses **IDz** to update the COBOL program stored in **Git**
2. Uses **DBB** to perform a User Dependency Build and verify the code is fixed
3. Uses **Git** to **Commit and Push** the corrected code to Repository

Using **DBB User Build** (Personal test)



[1] **eGit** plugin issues Git pull command to update local Git repository.

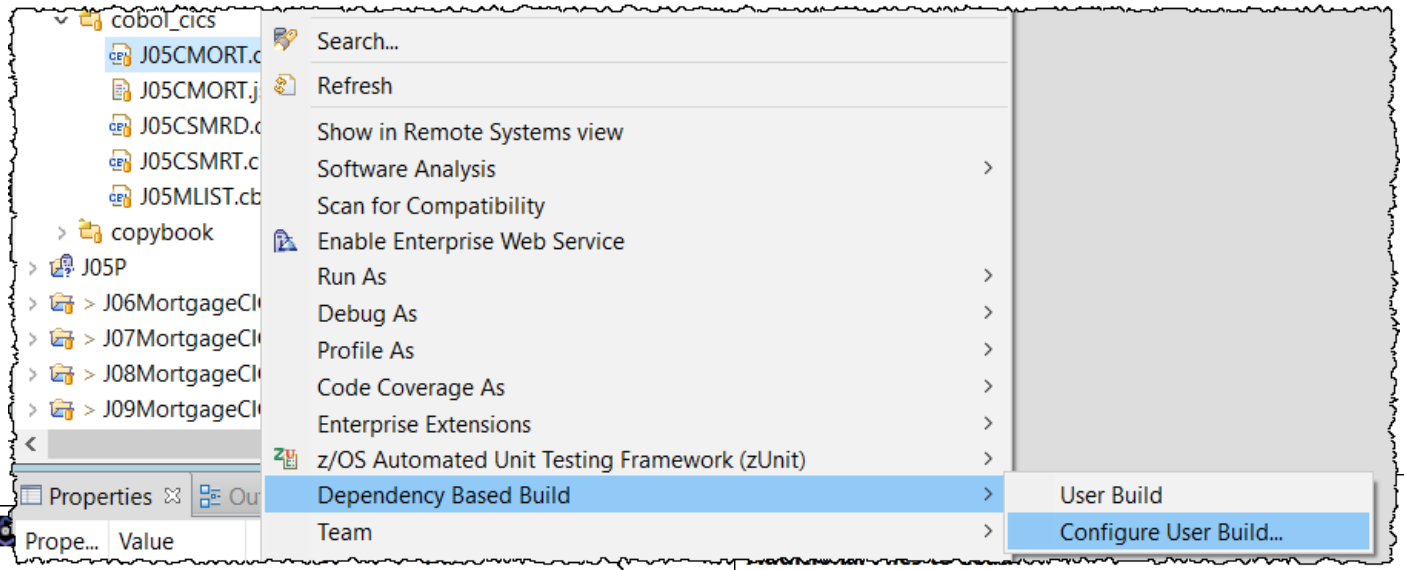
[2] **DBB User Build** moves COBOL source/copybooks from local IDz workspace to **zFS** files, invokes Groovy build scripts containing DBB APIs.

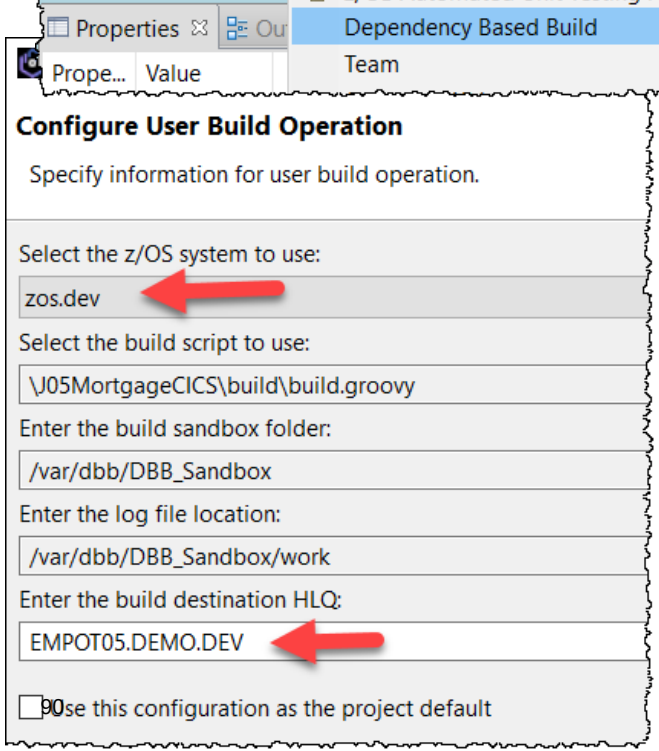
[3] DBB Toolkit provides Java APIs to:

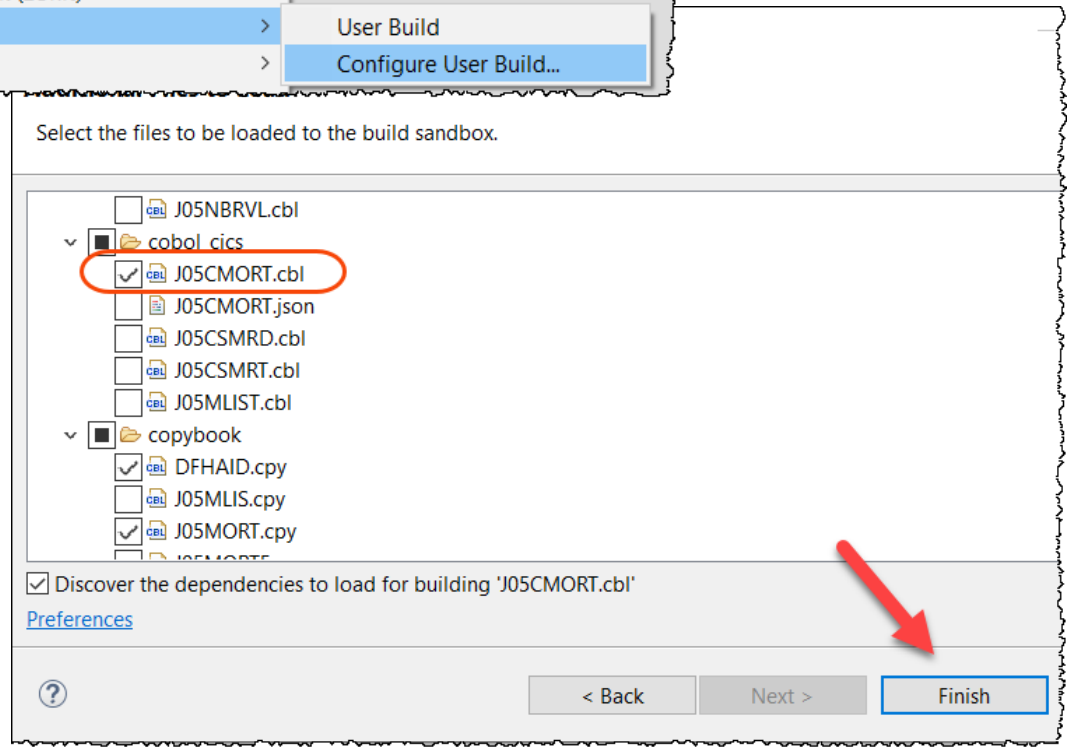
[4] Create datasets, copy source from **zFS** to **PDS**, invoke zOS programs (Compiler/Linkage Editor/ REXX for DB2 Bind).

[4a] Copy logs from **PDS** to **zFS**, generate build reports and display a Console output log.

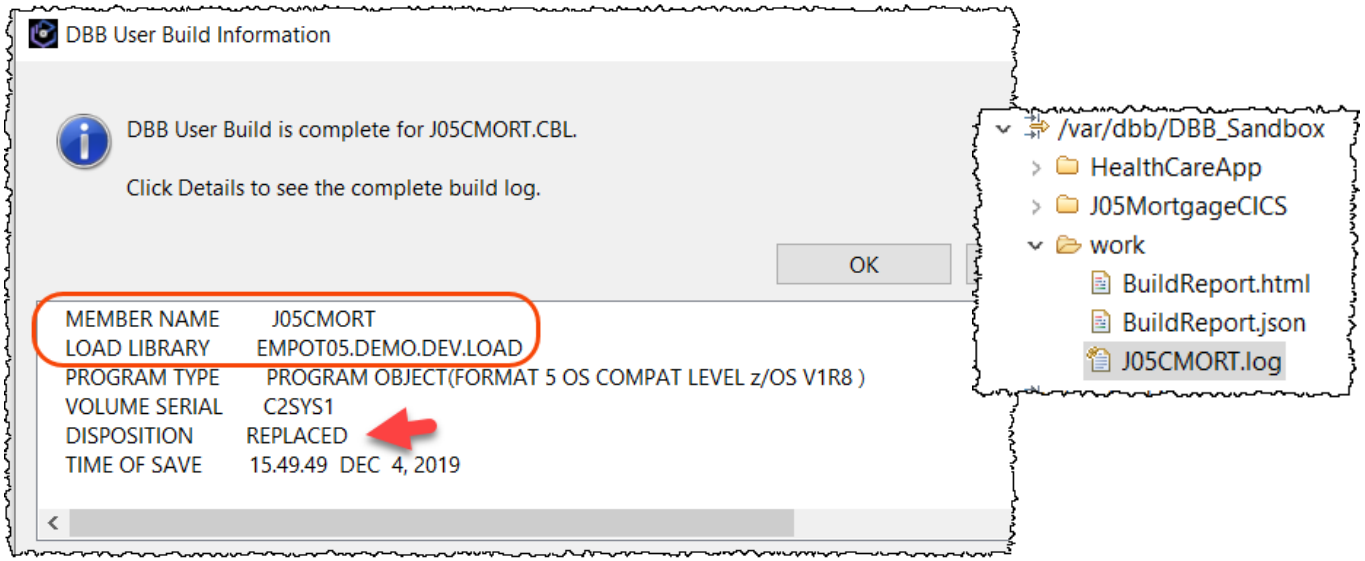
2. Uses DBB to perform a User Dependency Build and verify the code is fixed…

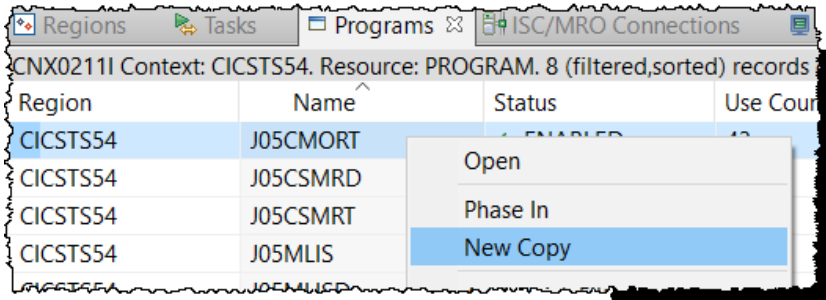


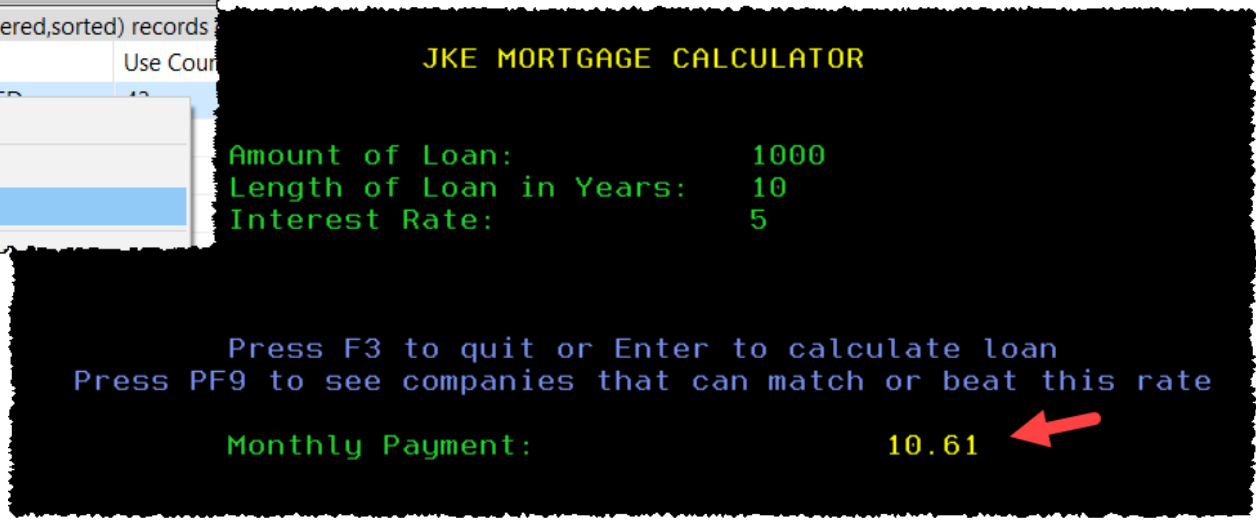




2. Uses DBB to perform a User Dependency Build and verify the code is fixed





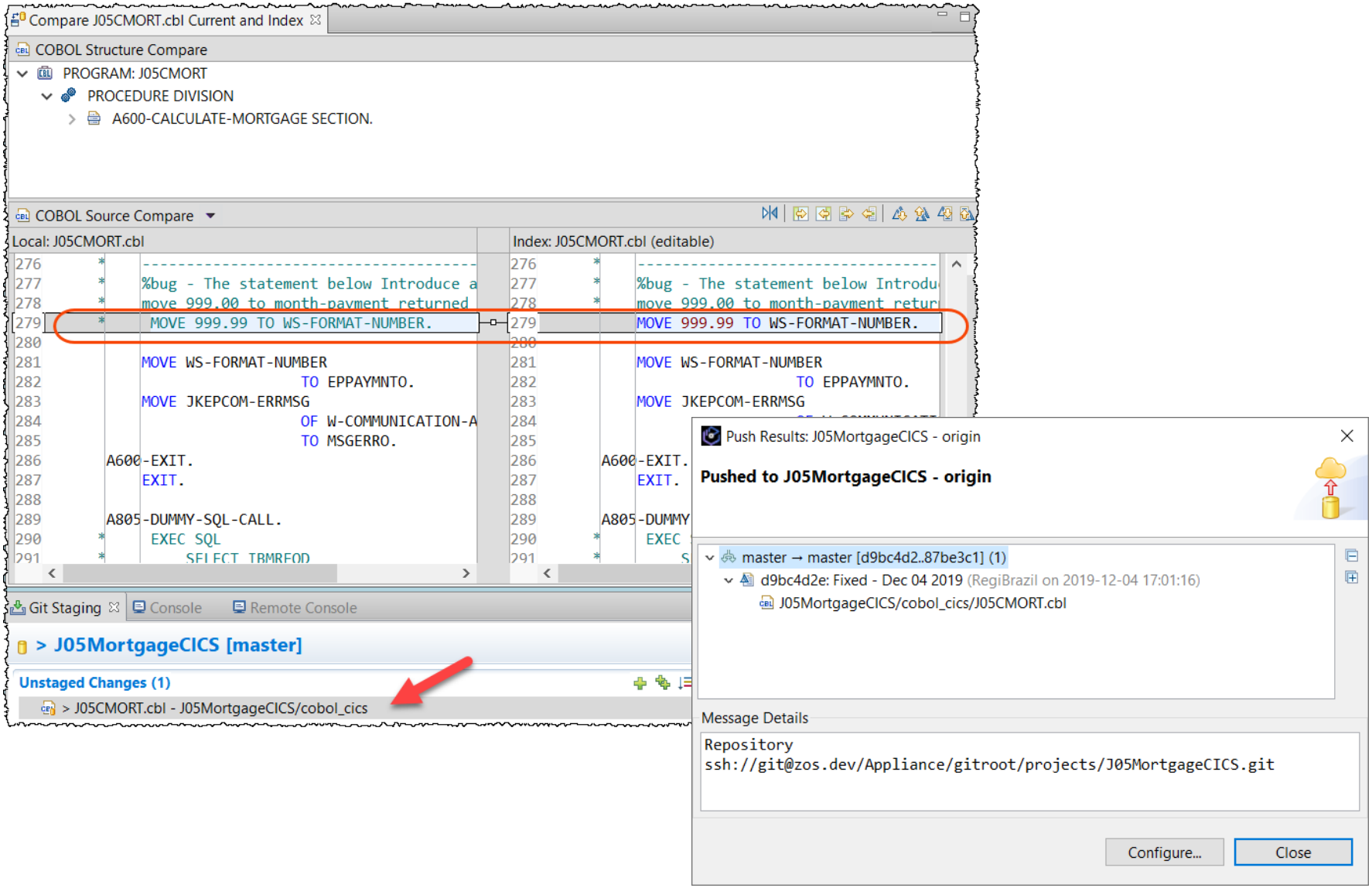


PART #1 – Fix COBOL/CICS `J05CMORT` broken program

Developer **B** will fix the issue created by developer **A**

1. Uses **IDz** to update the COBOL program stored in **Git**
2. Uses **DBB** to perform a User Dependency Build and verify the code is fixed
3. Uses **Git** to **Commit and Push** the corrected code to Repository

3. Uses GIT to Commit and Push the corrected code to Repository

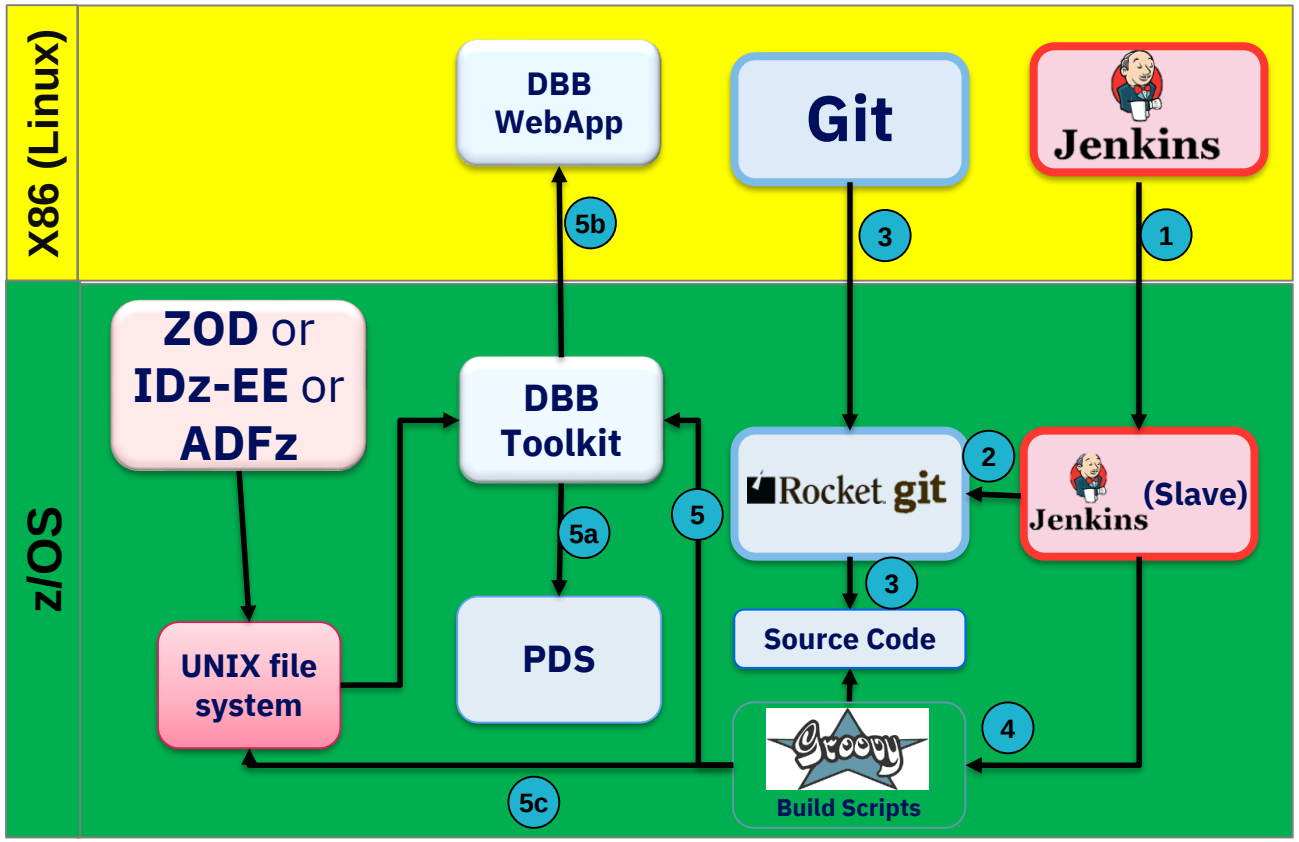


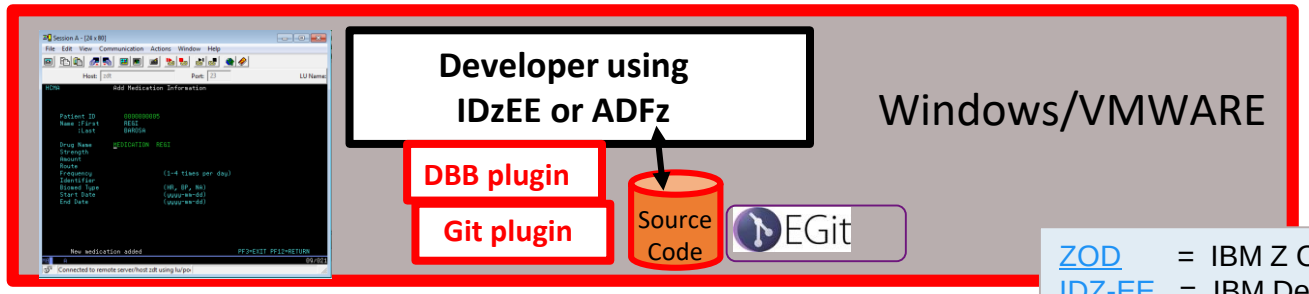
PART #2 - Release Engineer Deploy to z/OS and Mobile.

Rebecca – Release Engineer
Executes the Team Building using Jenkins

1. Uses **Jenkins** to perform the “team” building and **UrbanCode Deploy (UCD)** deploy.
2. Verify the Build results using **DBB** browser
3. Verify the Deploy results using **UCD** browser
4. Run the new deployed CICS application

Solution High Level Diagram





[1] Jenkins server sends build commands to remote agent.

[2] Jenkins agent issues Git pull command to update local Git repository.

[3] Rocket's Git client automatically converts source from UTF-8 to EBCDIC during pull.

[4] Jenkins agent invokes build scripts containing DBB APIs in local Git repository.

[5] DBB Toolkit provides Java APIs to:

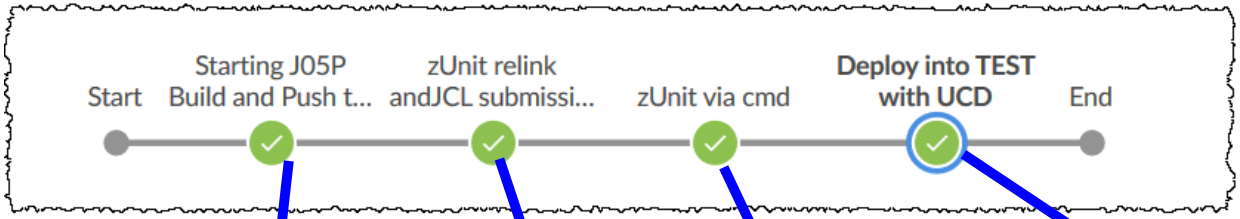
[5a] Create datasets, copy source from zFS to PDS, invoke zOS programs, issue ISPF and TSO commands.

[5b] Scan and store dependency data from source files, perform dependency and impact analysis, store build results.

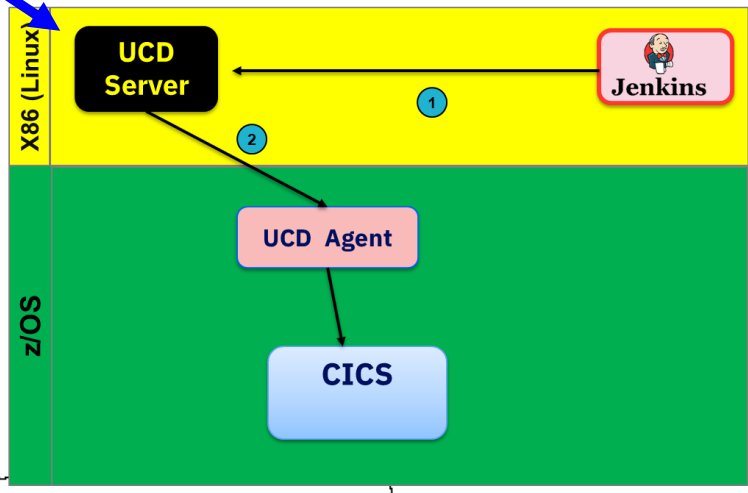
[5c] Copy logs from PDS to zFS, generate build reports (can be saved in build result).

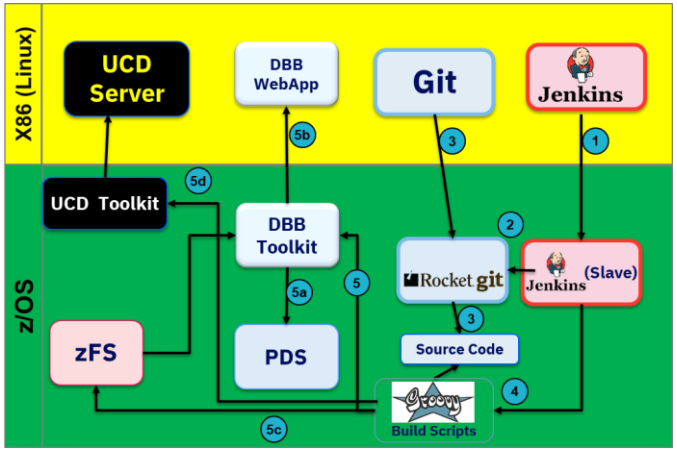
ZOD = IBM Z Open Development
IDZ-EE = IBM Developer for z Systems Enterprise Edition
ADFz = Application Delivery Foundation for z Systems
DBB = IBM Dependency Based Build

Demo High Level Diagram using Jenkins/Git/UCD



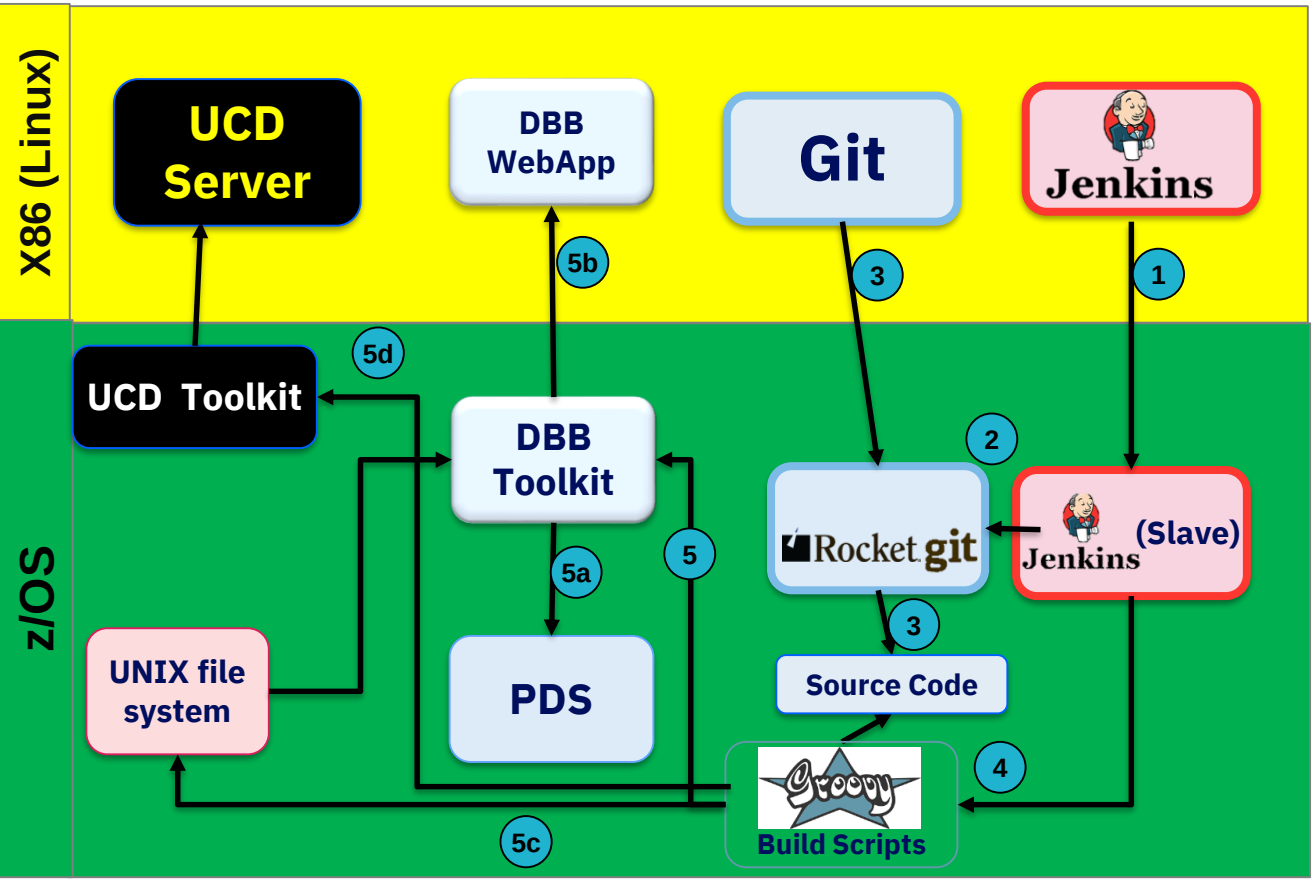

```
Running JCL 'EMPOT05.DEMO.DEV.JCL(J05CNEW)'
The JCL Job J05CNEW(JOB00155) completed successfully
Finished: SUCCESS
```






```
TEARDOWN (TEST4)
  o Test count: 2
  o Tests passed: 2
  o Tests failed: 0
  o Tests in error: 0
zUnit Test Runner 2.0.0.1 ended at 2019-12-04T20:
Finished: SUCCESS
```

High Level Demo Diagram with **Jenkins** and **UCD** (UrbanCode Deploy)



[1] Jenkins server sends build commands to remote agent.

[2] Jenkins agent issues Git pull command to update local Git repository.

[3] Rocket's Git client automatically converts source from **UTF-8** to **EBCDIC** during pull.

[4] Jenkins agent invokes build scripts containing DBB APIs in local Git repository.

[5] DBB Toolkit provides Java APIs to:

[5a] Create datasets, copy source from **zFS** to **PDS**, invoke zOS Compiler/Linkage Editor/ REXX for DB2 Bind.

[5b] Scan and store dependency data from source files, perform dependency and impact analysis, store build results.

[5c] Copy logs from **PDS** to **zFS**, generate build reports (can be saved in build result).

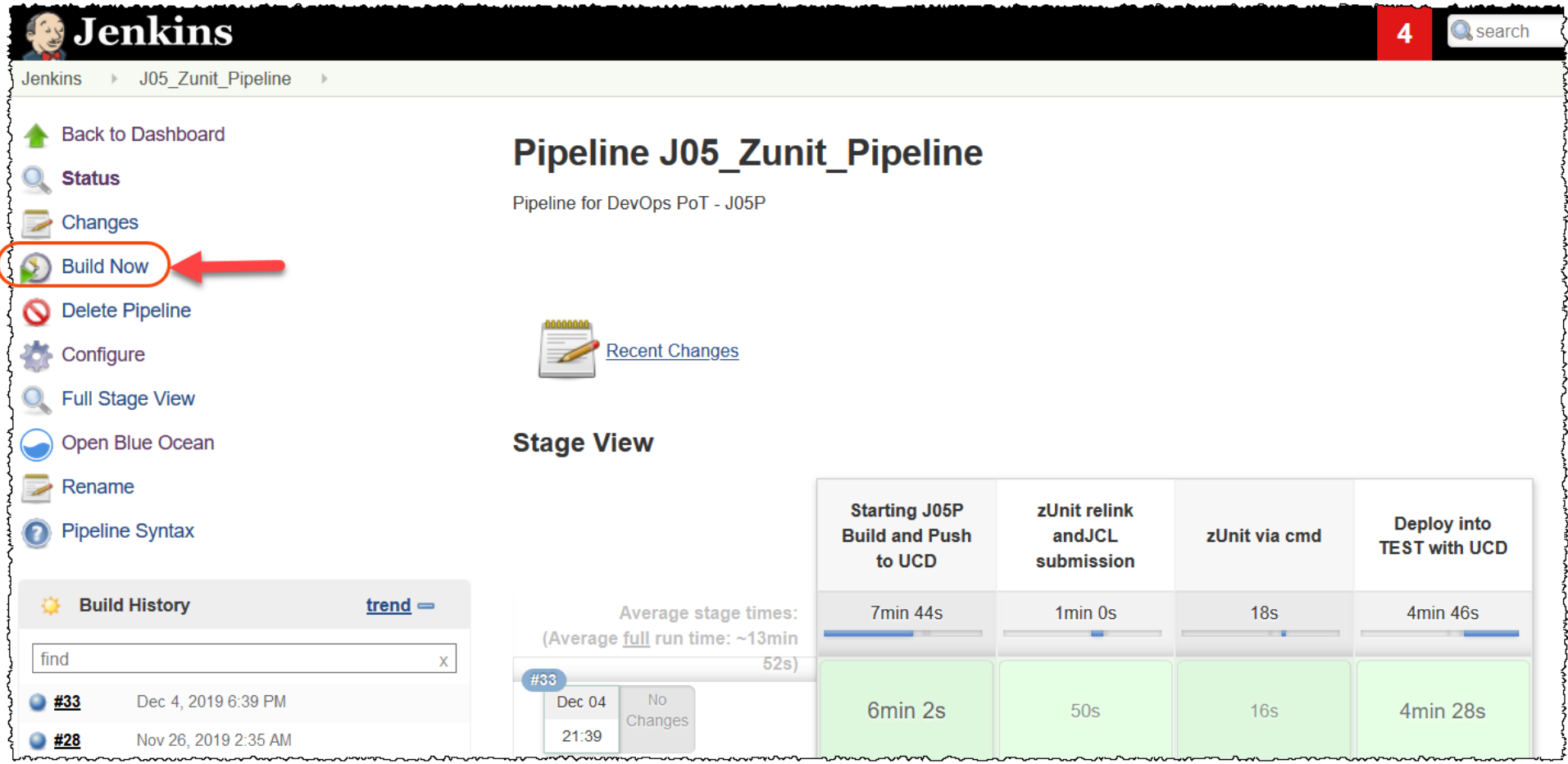
[5d] Invoke UCD toolkit to create a deployable version at UCD Server (Linux)

PART #3 - Release Engineer Deploy to z/OS and Mobile.

Rebecca – Release Engineer
Executes the Team Building using Jenkins

1. Uses **Jenkins** to perform the “team” building and **UrbanCode Deploy (UCD)** to deploy.
2. Verify the Build results using **DBB** browser
3. Verify the Deploy results using **UCD** browser
4. Run the new deployed CICS application

1. Uses Jenkins to perform the “team” building and **UrbanCode Deploy** to deploy



What is Git?

- **Git** is a free and open source distributed version control system.
  - Version Control System is a system that records changes to a file or set of files over time so that you can recall specific versions later
  - Distributed means that there is no main server and all of the full history of the project is available once you cloned the project.
- **GitHub**, **GitLab** and **Bitbucket** are code collaboration and version control tools offering repository management based on Git.
  - They each have their share of fans, though **GitHub** is by far the most-used of the three.
  - Of the three, only **GitLab** is open source, though all three support open source projects.

Why Git?

- Distributed
  - Full local repo, with history
  - Work offline
- Lightweight branching, users collaborate better and spend less time managing version control and more time developing code.
- De-facto standard

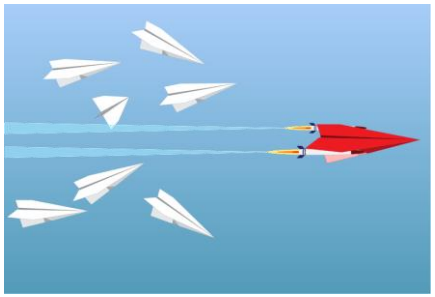
GitHub

GitLab

ATLASSIAN Bitbucket

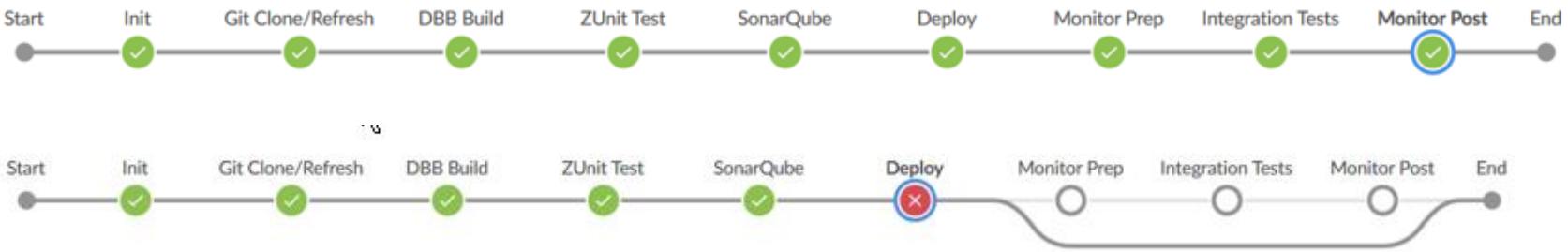
Git vs. traditional SCM

- True parallel development
  - Branches on demand
  - Freedom from pre-defined streams
- Continuous Improvement to build processes
  - Use generic artifact repositories
  - Easy integration with CI/CD Pipelines (Jenkins)



What is Jenkins?

- Jenkins is an automation engine which supports a number of automation patterns
- Jenkins can invoke scripts for builds and tests continuously and monitors the execution and status of remote executions.



Why Jenkins

- Used everywhere
- Continuous Integration leads to Continuous Deployment allowing us to deliver software more rapidly.

PART #3 - Release Engineer Deploy to z/OS and Mobile.

Rebecca – Release Engineer
Executes the Team Building using Jenkins

1. Uses **Jenkins** to perform the “team” building and **UrbanCode Deploy (UCD)** deploy.
2. Verify the Build results using **DBB** browser
3. Verify the Deploy results using **UCD** browser
4. Run the new deployed CICS application

2. Verify the Build results using **DBB browser**

DBB Build Result

Field	Value
id	3077
group	J05MortgageCICS
label	build.20191205.024250.042
Build Report	view

Build Summary

Number of files being built: 2

	File	Commands	RC	Data Sets	Outputs	Deploy Type	Logs
1	J05MortgageCICS/bms/J05MORT.bms	ASMA90	0	EMPOT05.DEMO.DEV.BMS(J05MORT)	EMPOT05.DEMO.DEV.COPYBOOK(J05MORT)		J05MORT.log
		ASMA90	0				
		IEWBLINK	0		EMPOT05.DEMO.DEV.LOAD(J05MORT)	MAPLOAD	
2	J05MortgageCICS/cobol_cics/J05CMORT.cbl • DFHAID COPY • J05MortgageCICS/copybook/J05MTINP.cpy COPY • J05MortgageCICS/copybook/J05MTOUT.cpy COPY • J05MortgageCICS/copybook/J05MTCOM.cpy COPY • J05MortgageCICS/copybook/J05NBRPM.cpy COPY • J05MortgageCICS/copybook/J05MORT.cpy COPY • J05NBRVL CALL • J05MLIST LINK Hide Dependencies	IGYCRCTL	4	EMPOT05.DEMO.DEV.COBOL(J05CMORT)			J05CMORT.log
		IEWBLINK	0		EMPOT05.DEMO.DEV.LOAD(J05CMORT)	LOAD	

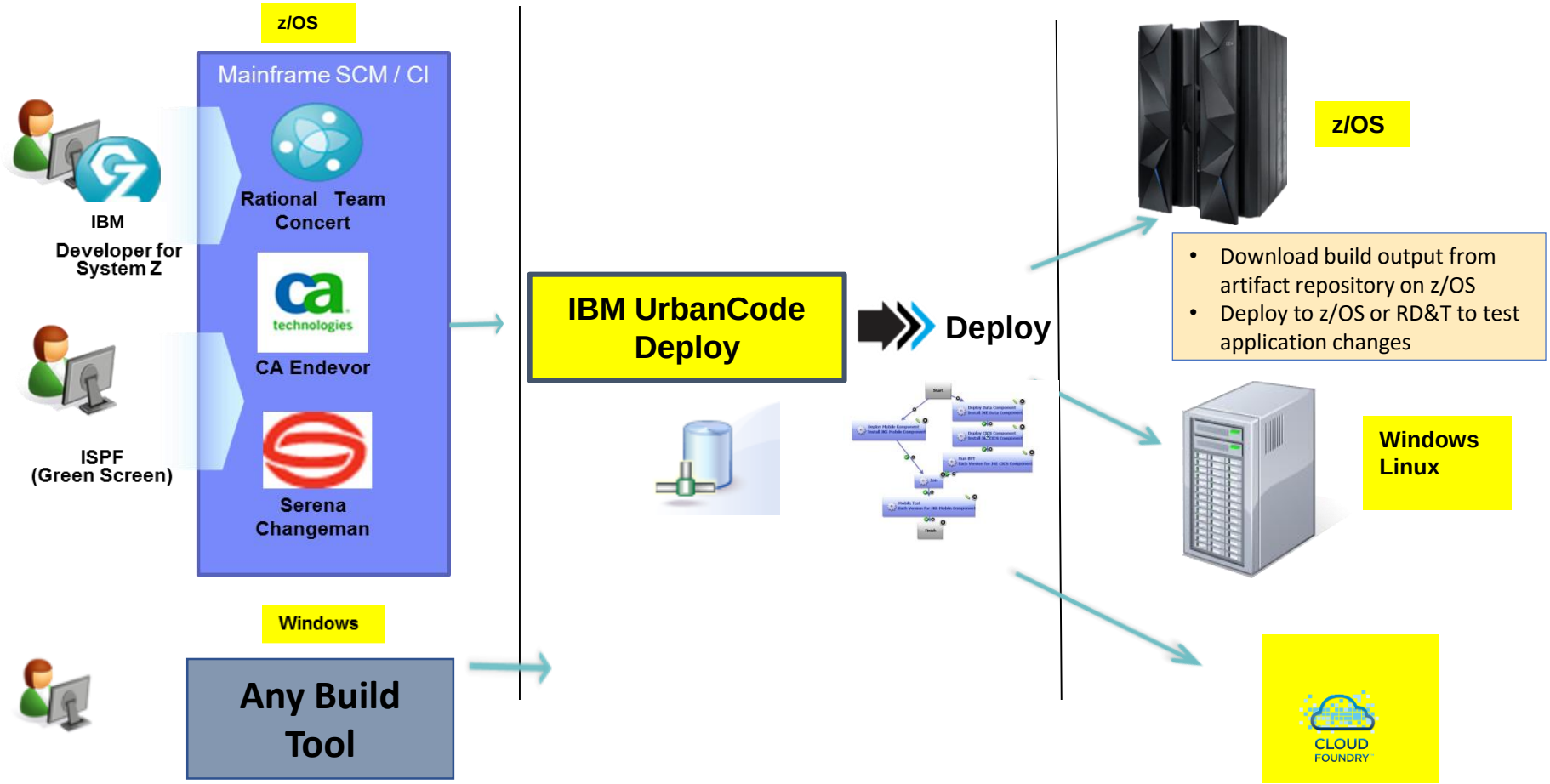
PART #3 - Release Engineer Deploy to z/OS and Mobile.

Rebecca – Release Engineer
Executes the Team Building using Jenkins

1. Uses **Jenkins** to perform the “team” building and **UrbanCode Deploy (UCD)** deploy.
2. Verify the Build results using **DBB** browser
3. Verify the Deploy results using **UCD** browser
4. Run the new deployed CICS application

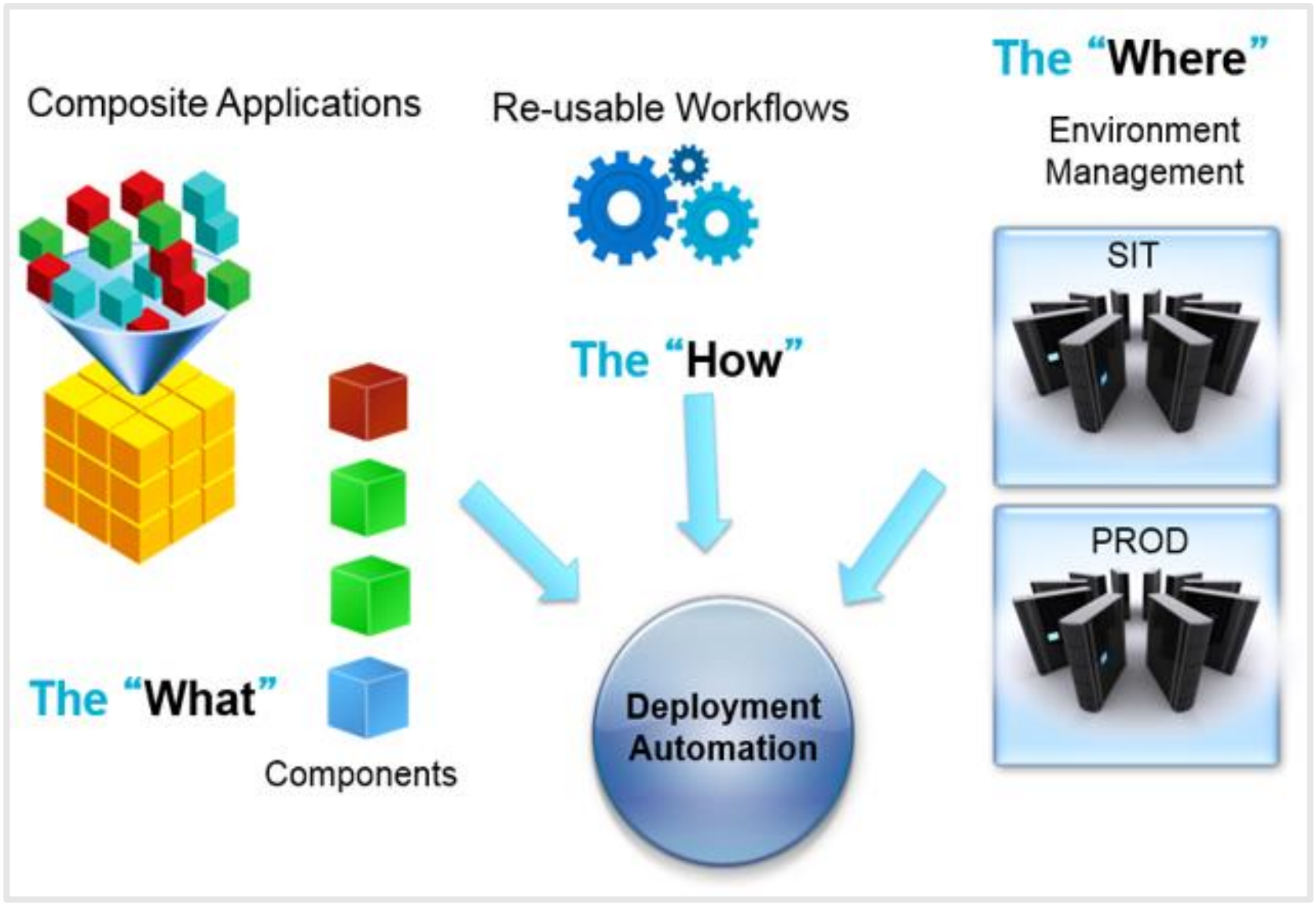
Continuous Delivery for the Mainframe and Others

Capabilities to speed delivery of interdependent, multi-platform applications

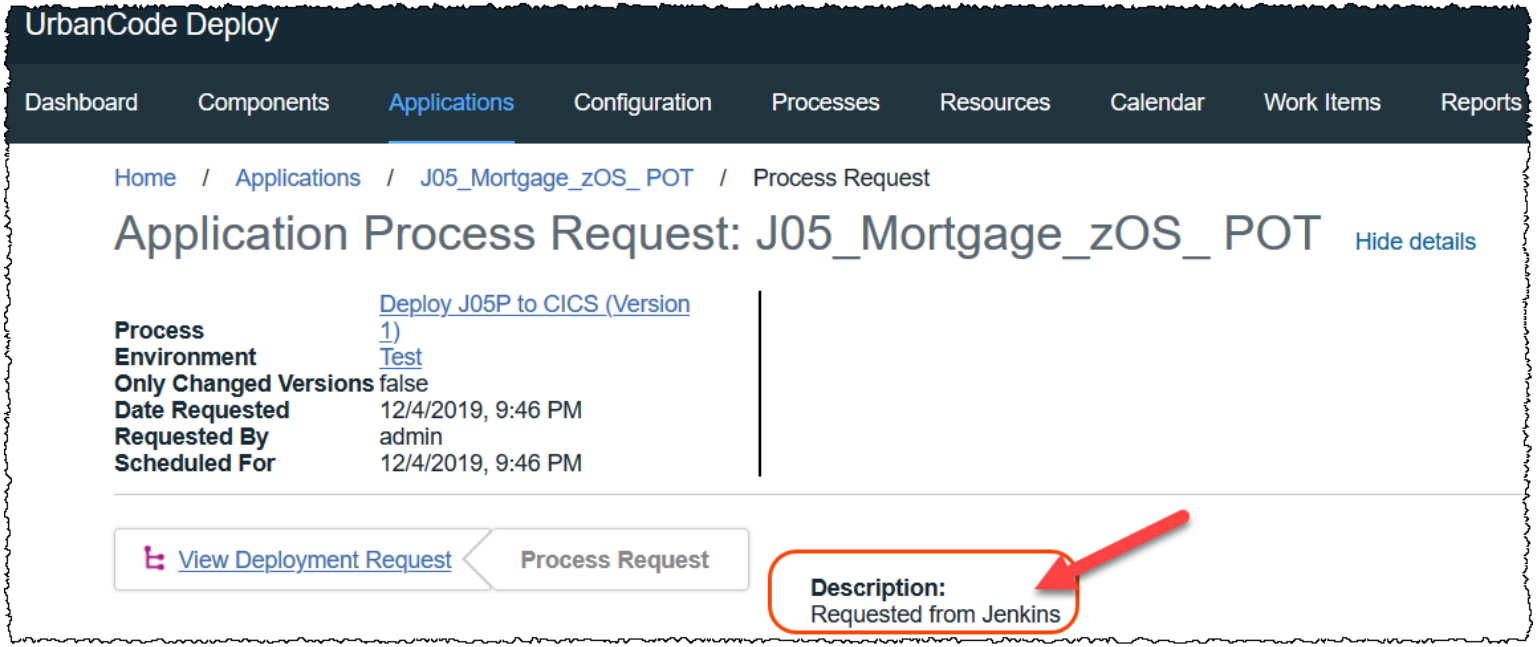


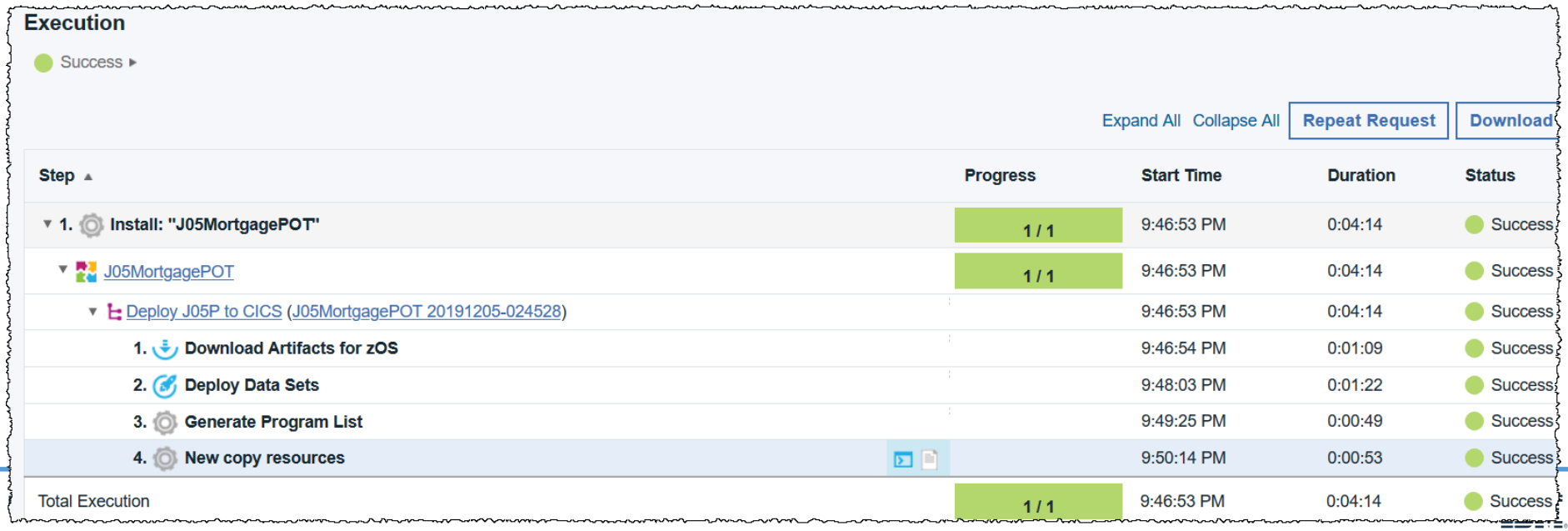
- Provides a **unified solution** for continuous delivery of heterogeneous enterprise applications
- **Accelerate delivery** and reduces cycle time to develop/test multi-tier applications across heterogeneous environments and platforms
- **Reduce costs** and eliminate delays for delivering mainframe applications
- **Minimize risk** and improve productivity across disparate teams with cross-platform release planning

UrbanCode Deploy: What makes it different?



3. Verify the Deploy results using **UCD browser**



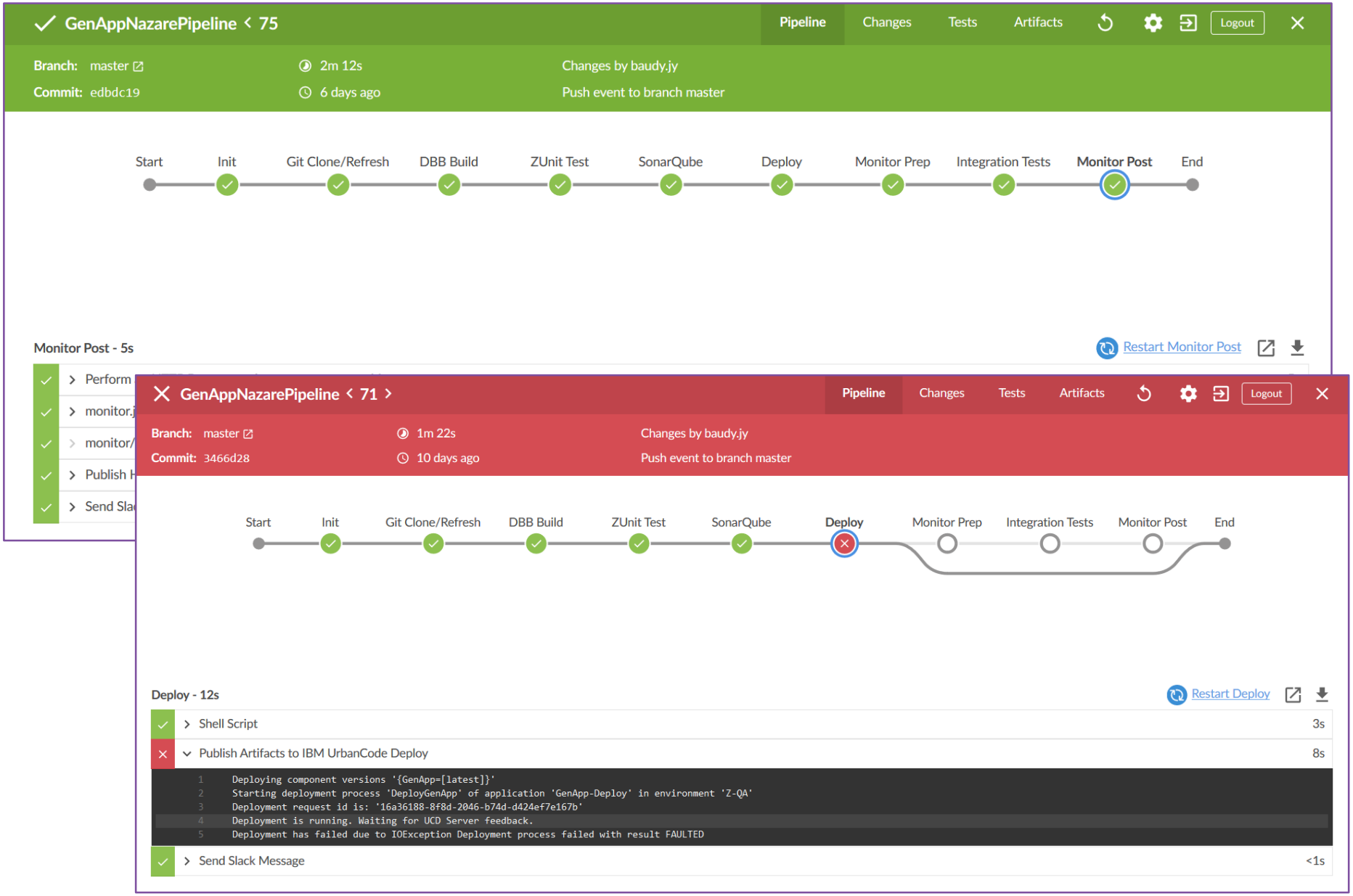


PART #3 - Release Engineer Deploy to z/OS and Mobile.

Rebecca – Release Engineer
Executes the Team Building using Jenkins

1. Uses **Jenkins** to perform the “team” building and **UrbanCode Deploy (UCD)** deploy.
2. Verify the Build results using **DBB** browser
3. Verify the Deploy results using **UCD** browser
4. Run the new deployed CICS application

Example Pipeline Results



Agenda

9:00 Introductions
9:20 DevOps for Z

10:00 Break

10:15 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z

11:45 Lunch

12:30 zUnit Overview

12:40 Demo: Scenario using Z DevOps solutions including zUnit, Git and Jenkins

1:30 z/OS DevOps Testing Automation & Virtualization

2:00 – 4:30 Choose one Optional lab:
- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6 : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → LAB 2C Application Discovery : Find a candidate API from Catalog Manager Application
- → LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- → LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
- → LAB 8 - Using Application Performance Analyzer (APA)

Lecture Labs Breaks

Agenda

9:00 Introductions
9:20 DevOps for Z

10:00 Break

10:15 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z

11:45 Lunch

12:30 zUnit Overview

12:40 Demo: Scenario using Z DevOps solutions including zUnit, Git and Jenkins

1:30 z/OS DevOps Testing Automation & Virtualization

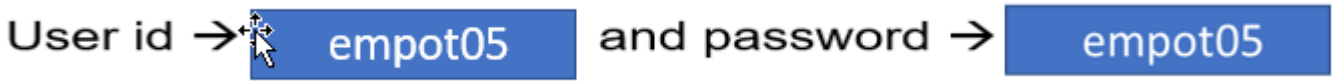
2:00 – 4:30 Choose one Optional lab:
- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6 : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → LAB 2C Application Discovery : Find a candidate API from Catalog Manager Application
- → LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- → LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
- → LAB 8 - Using Application Performance Analyzer (APA)

Lecture Labs Breaks

Lab 7: Using IBM Dependency Based Build with Git, Jenkins and UCD on z/OS

This lab will use IBM Dependency Based Build (DBB) along with Git, Jenkins and UrbanCode Deploy (UCD) on z/OS.
You will modify an existing COBOL/CICS application stored on Git.
You will use ADFz to change the code and perform a personal test for later delivery and commit to Git and then use Jenkins for the final build and continuous delivery.
The updated code will be deployed to CICS using UrbanCode Deploy (UCD)
.

Overview of development tasks



1. **Get familiar with the application using the 3270 terminal**
   → You will start a 3270 emulation and execute a transaction named **JxxP** to become familiar with the Application that you intend to modify.

2. **Load the source code from Git to the local IDz workspace**
   → You will load the COBOL code that is stored on Linux to your windows client to be modified.

3. **Modify the COBOL code using IDz.**
   → Using IDz you will modify the COBOL code to have a different message in a CICS dialog.

4. **Use IDz DBB User Build to compile/bind and perform personal tests.**
   → You will compile and link the modified code using the DBB User Build function available on IDz EE or ADFz. When complete you will run the code using CICS for a personal test and verify that the change is correctly implemented.

5. **Push and Commit the changed code to Git .**
   → You will commit the changes to Git.

6. **Use Jenkins with Git plugin to build the modified code**
   → You will use Jenkins pipeline to build the new changed code and push the executables to be deployed using UCD.

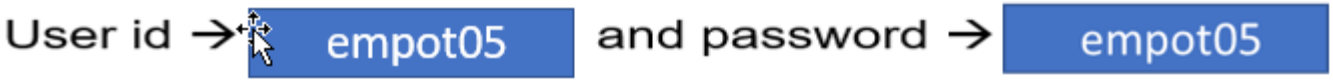
7. **Use Jenkins and UCD plugin to deploy results and test the code again**
   → You will verify the results after the final deploy to CICS using UCD

8. **(Optional) Understanding DBB Build Reports**
   → You will understand the reports generated by DBB during the build

Lab 6: Using IBM zUnit to Unit Test a COBOL CICS application

In this lab you will record interaction with a COBOL CICS program (program under test) and import the recorded data into the unit test case generation process. You will build and run the unit test, make a change to the CICS COBOL program, and rerun the unit test.

.



Overview of development tasks

To complete this tutorial, you will perform the following tasks:

1. **Get familiar with the application using the 3270 terminal**
   → You will start a 3270 emulation and execute a transaction named **J05P** to become familiar with the Application that you will be recording.

2. **Import a z/OS project**
   → You will import a z/OS project with the required resources added to the project.

3. **Record interaction with the application.**
   → You will record an interaction with the COBOL CICS program.

4. **Generate, build and run the unit test**
   → You will compile and link-edit the generated unit test program, followed by running the unit test.

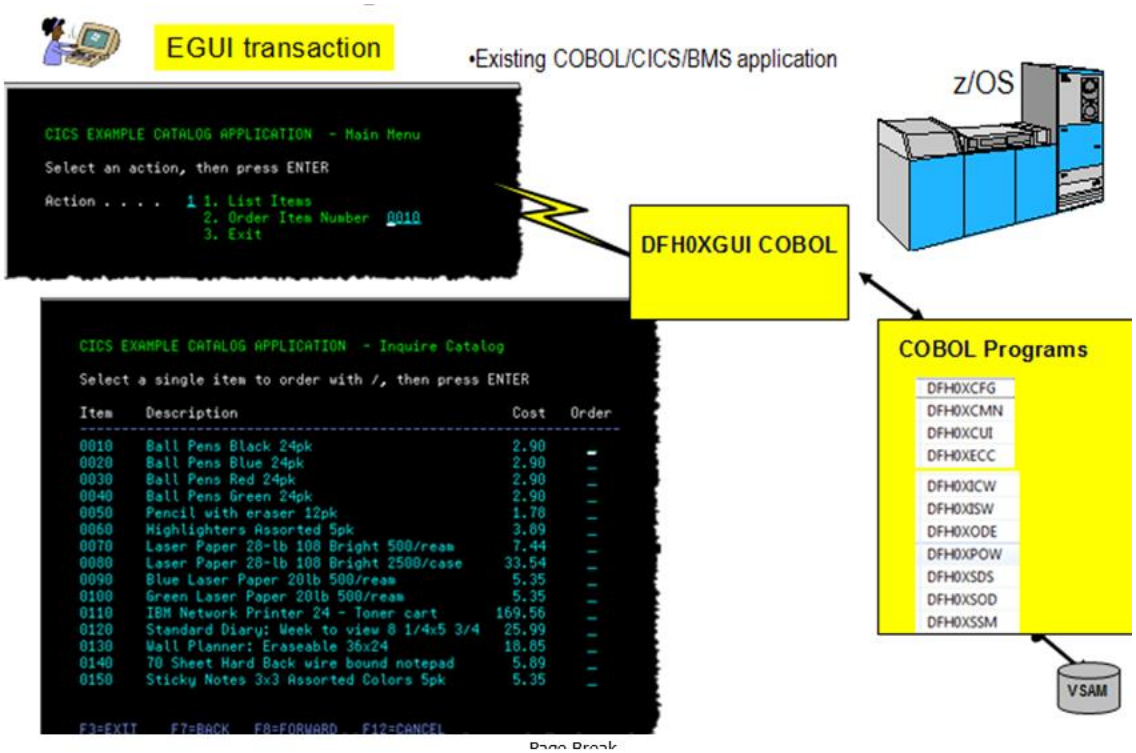
5. **Modify the program and rerun the unit test**
   → You will modify the program under test, rerun the unit test, and observe the failure of the test case.

Run the unit test from a shell script.

→ You will run the unit test from a shell script and observe a similar test case result

Lab 2C – AD: Find a candidate API from Catalog Application

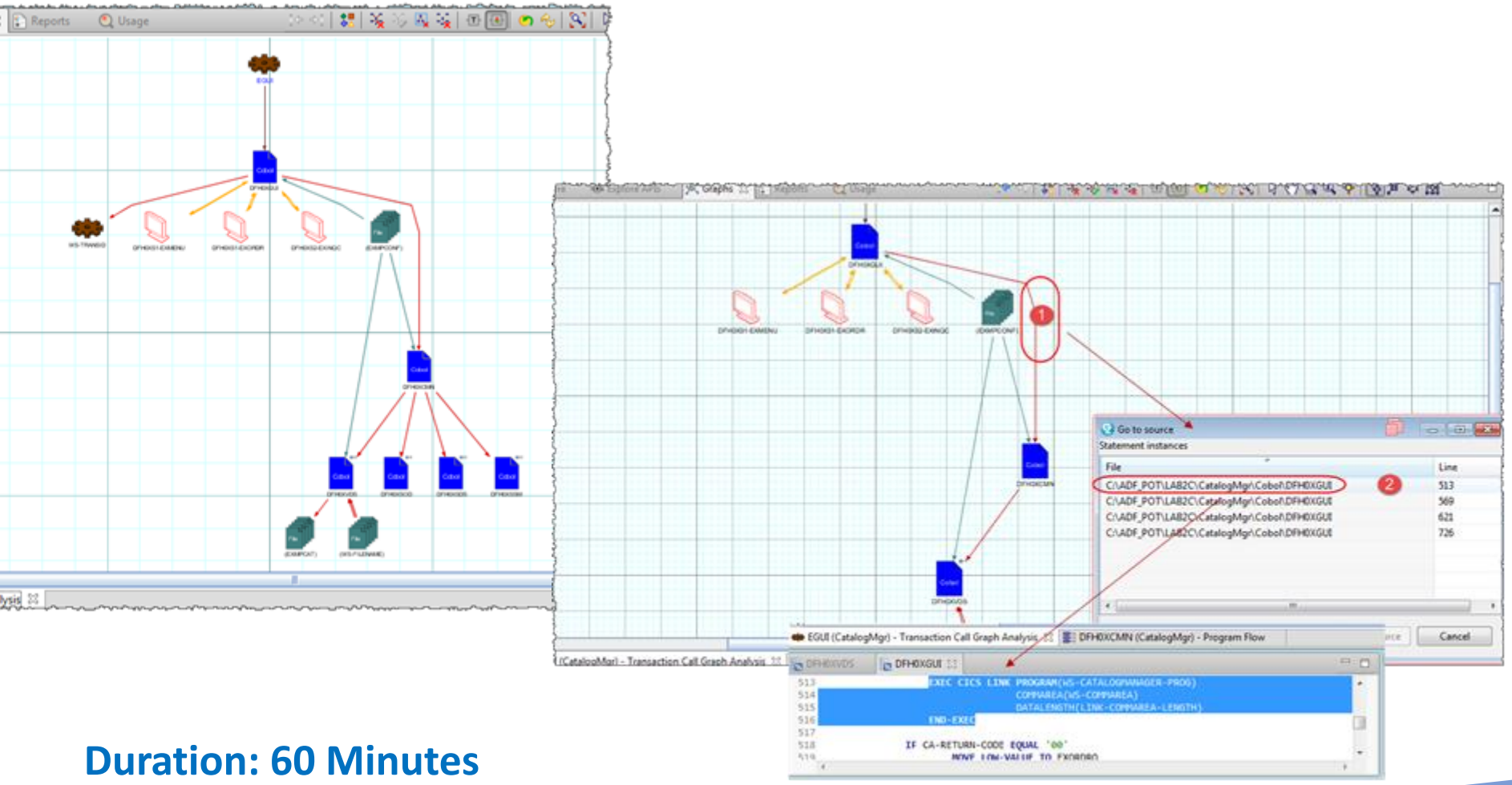
Objective: This lab you will go through the process of discovering a possible API from existing the Catalog Manager Application (EGUI transaction).



Duration: 60 Minutes

Lab 2C – AD: Find a candidate API from Catalog Application

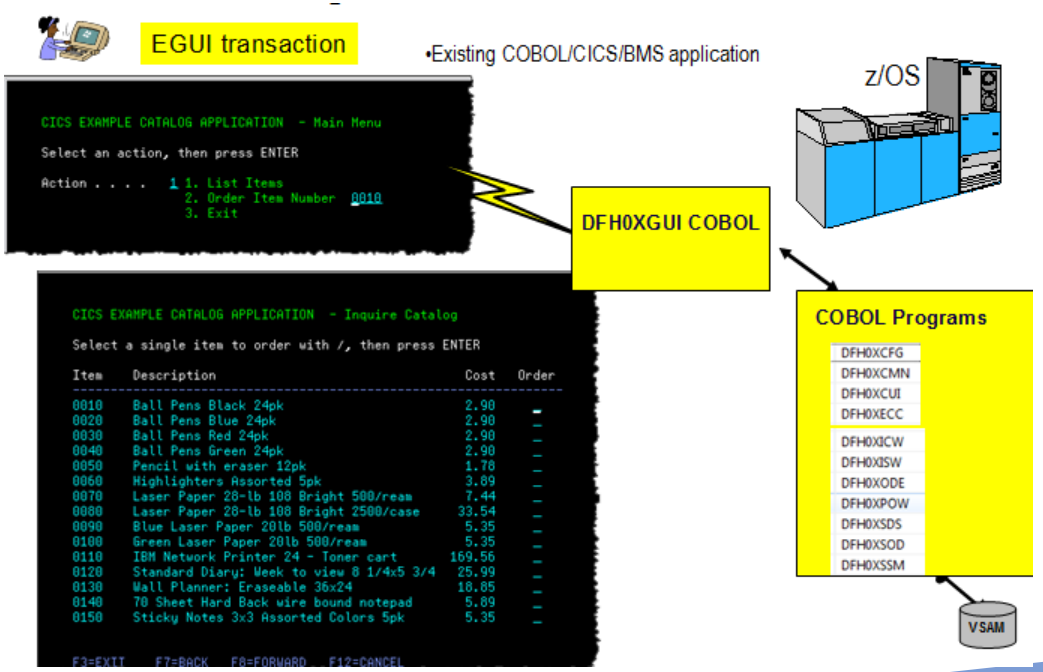
Objective: This lab you will go through the process of discovering a possible API from existing the Catalog Manager Application (EGUI transaction).



Duration: 60 Minutes

Lab 2D: **z/OS Connect EE Toolkit** : Create an API from Catalog Manager Application (EGUI)

- In this lab you will go through the process of creating an API that allows REST clients to access the application. The different steps of the lab are:
-
- 1. Create your team services using the z/OS Connect EE API Toolkit
- 2. Create your team API using the z/OS Connect EE API Editor
- 3. Test your team API using a REST client:.



Duration: 60 Minutes

Lab 5: UrbanCode Deploy → Focus on Deploy

Abstract:

You will create and deploy an existing COBOL CICS application using UrbanCode Deploy to z/OS

Part 1 – Create the UrbanCode application infrastructure.

In this section you will design the actual deployment process, you need to prepare the application infrastructure in UCD. First, you will create a component and add component version, then you will define resources and map them to an environment, and finally you will create an application and add environment and component to it

Part 2 - Create the UrbanCode deployment processes.

On this part you will you create a deployment process for your component and the application process that uses the component process to deploy the component

Part 3 – Deploy the application to z/OS CICS

On this part you will deploy the Application to the z/OS CICS.

Lab 8: Using Application Performance Analyzer (APA)

This lab will take you through the steps of using Application Performance Analyzer (APA) integrated with Application Delivery Foundation for z (ADFz).

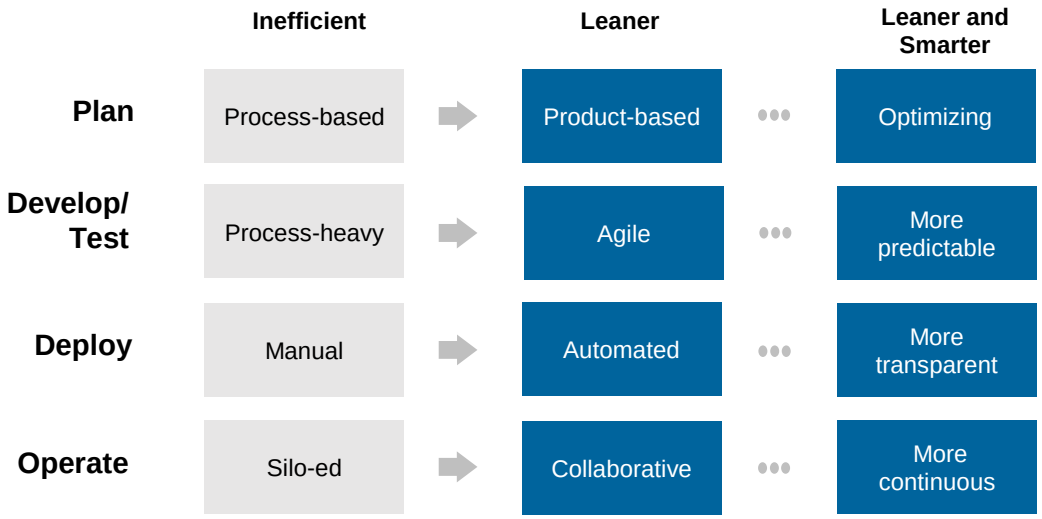
On this lab you will measure an existing COBOL/DB2 program running in batch.

Overview of development tasks

To complete this tutorial, you will perform the following tasks:

1. **Create a new observation request for a job that is not running yet**
   → Using *ADFz* you will create an *APA* observation request.
2. **Run a sample batch job to collect performance data**
   → You will submit a JCL that will execute a COBOL/DB2 batch program and will collect performance data.
3. **Review some of the reports created.**
   → You will analyze some of the reports created.

Start your transformation today with an IBM DevOps Workshop



Contact us about a no-charge DevOps assessment workshop

Workshop Objectives

- Define business drivers for DevOps,
- Identify existing or planned DevOps initiatives
- Determine the top inhibitors within current software lifecycle
- Create an adoption roadmap for DevOps practices

Overview

- Led by IBM DevOps Solution Architects
- Audience:
  - LOB Executives
  - Senior IT
  - Managers in Application Development and IT Operations

IBM Z Trial Program

Try the latest IBM Z capabilities today at zero cost, with no installation.

No charge environment

No set up, no install

Hands-on tutorials

Available now:

- IBM z/OS Connect Enterprise Edition
- IBM Db2 Utilities Solution for z/OS
- IBM Cloud Provisioning and Management for z/OS
- IBM Information Management System (IMS)
- IBM Machine Learning for z/OS
- IBM Dependency Based Build
- IBM Application Discovery and Delivery Intelligence (ADDI)
- IBM Application Delivery Foundation for z Systems
- IBM Z Development and Test Environment
- IBM Db2 Administration Solution for z/OS
- IBM Operational Decision Manager for z/OS
- IBM Open Data Analytics for z/OS
- IBM OMEGAMON for JVM on z/OS
- IBM CICS Transaction Server
- IBM SDK for Node.js – z/OS

Coming soon:

- IBM IMS Administration Tool for z/OS
- IBM z/OS Management Facility
- IBM DB2 Performance Solution for z/OS

Register now at ibm.biz/z-trial

No-Charge IDz/ADFz Training offerings for the fall

https://developer.ibm.com/mainframe/idzrdz-remote-training/

December 11, 9:00 AM EST	**IDz Entry-Level Training Module 6 - the ISPF 3.x Data Set Utilities** This module instructs on the data set utility features of IDz that are available from the ISPF 3.x dialogs: **File allocation, File rename, File delete, Library compress, VSAM File create, GDG Model and GDG Data Set create** . Batch Job management and interactive CLIST/REXX support is also part of Module 6. An introduction to Menu Manager is presented
December 17, 9:00 AM EDT	IDz Experienced Training: Code Review Learn how to improve application code quality by running programs thru a static analysis tool that detects statements which break corporate policy for: Naming Convention, COBOL 5, Performance efficiency and "Maintain-ability". Learn how to run Code review; Interactively, against an entire PDS (in one operation), in Batch/JCL - and using a Baseline.Time-permitting, learn the initial techniques for writing custom code review rules.
December 18, 9:00 AM EDT	**IDz Entry-Level Training Module 7 - MVS Subprojects** Learn how to use this convenient & powerful IDz feature to organize and manage individual PDS members and individual QSAM files on a per-project bases. MVS Subprojects are also utilized with SCM access and Embedded SQL code/test/tune.

Overview

Our learning catalog offers you the training you need, featuring on-demand courses and live remote instructor-led training. Our focus is to equip you with the skills required to make a DevOps transformation successful.

Training

Remote Instructor-led Training

Please click the button below to see the schedule.

Learn more

IDz/RDz Self-Paced Learning

Learn more

IBM Dependency Based Build

Self-paced learning

Coming soon

Questions / Comments