Z Open Development PoT

Proof of Technology

DevOps Solution for Z Development

Mark Boyd
mboyd@us.ibm.com

Regi Barosa
rbarosa@us.ibm.com

Timothy Han
timothy.han@ibm.com

Thuy-Mi Le
Thuy-mi.le@ibm.com

November 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 on z Systems Introduction
9:45 Application Delivery Foundation (ADFz):
- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break

10:30 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 z Open Unit Test (Zunit) : Overview & Demonstration

1:00 Day in the Life - Using Application Discovery, DBB/Git/Jenkins and UCD

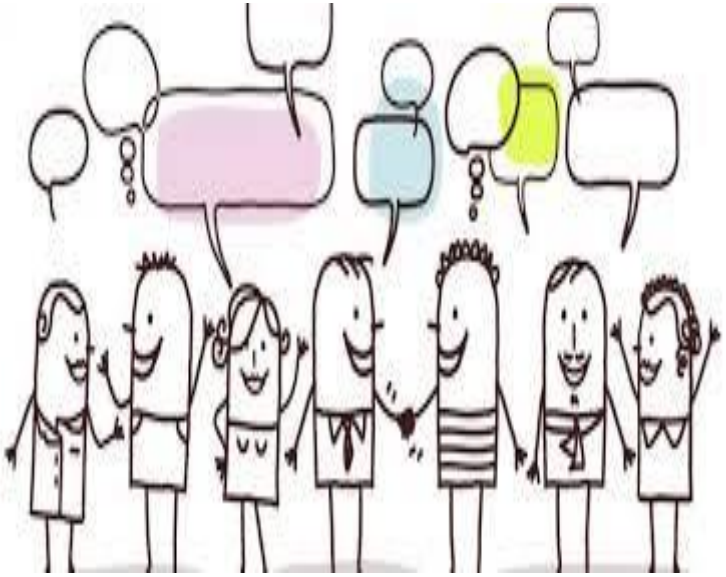
2:00 – 4:30 Choose one Optional lab:

→ **LAB 0 - Unit Testing for COBOL CICS** (1 hour)
→ LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (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

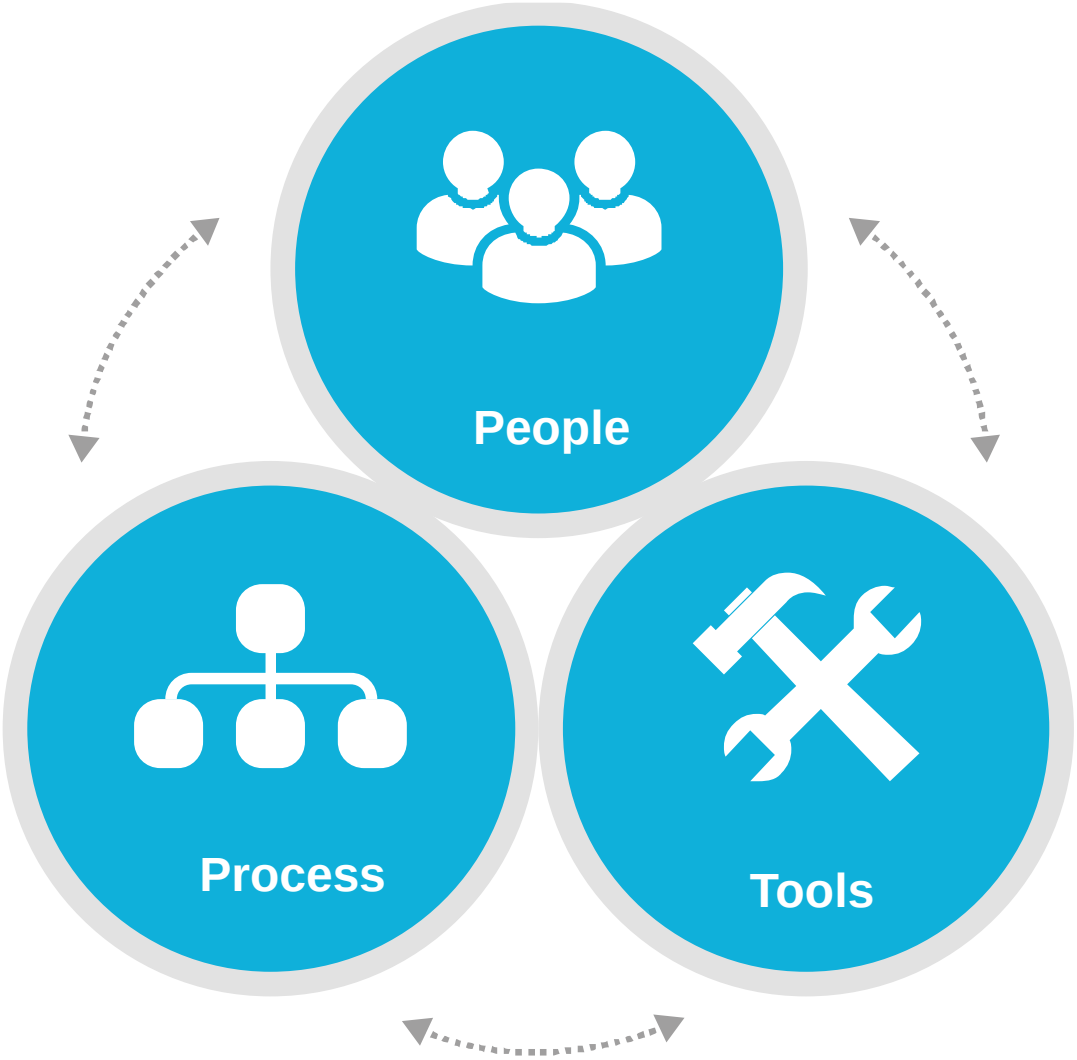
Introductions

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

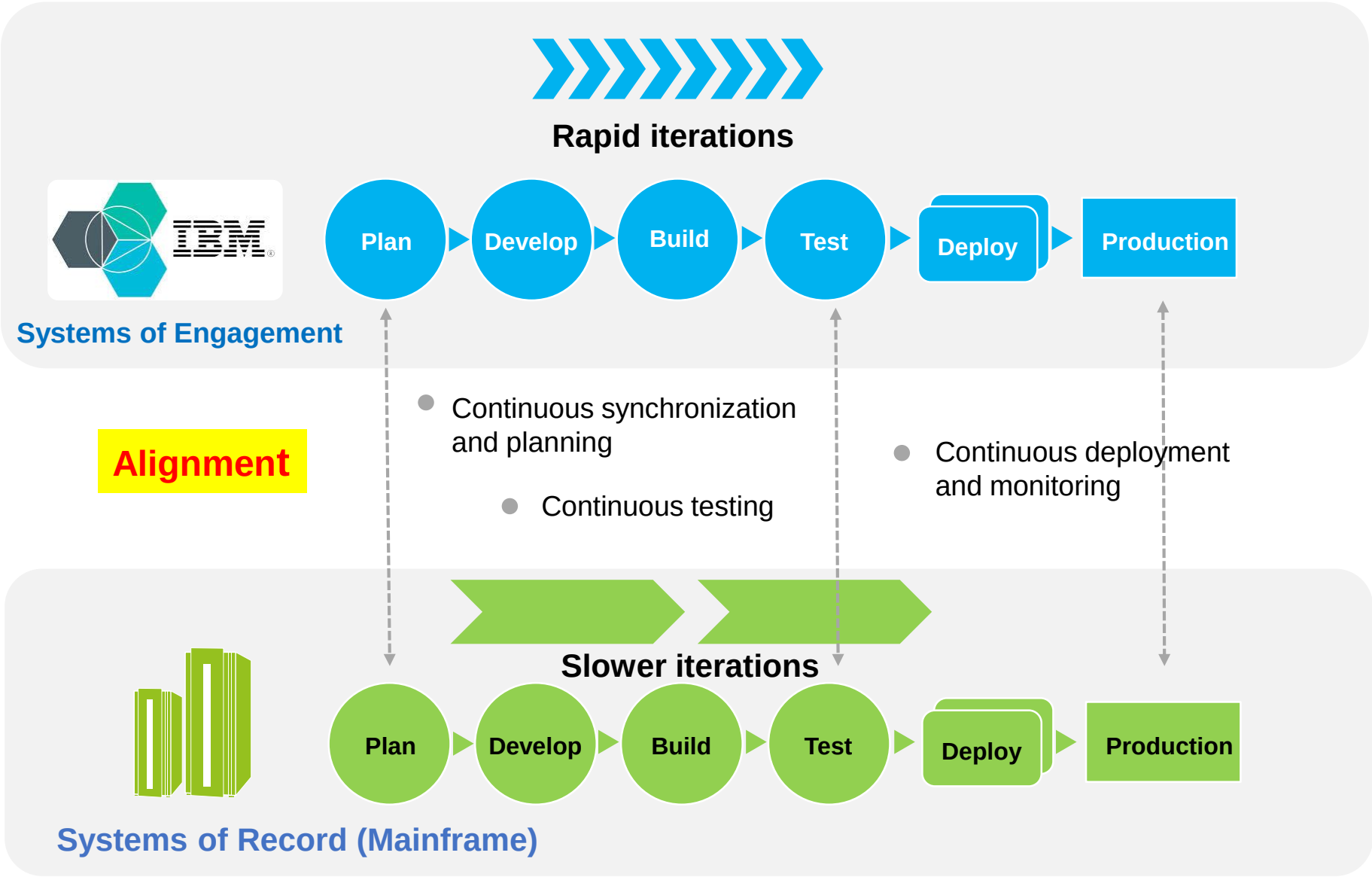


What do you need to get out of attending this PoT workshop?

DevOps is not one of these things… It's all of them!

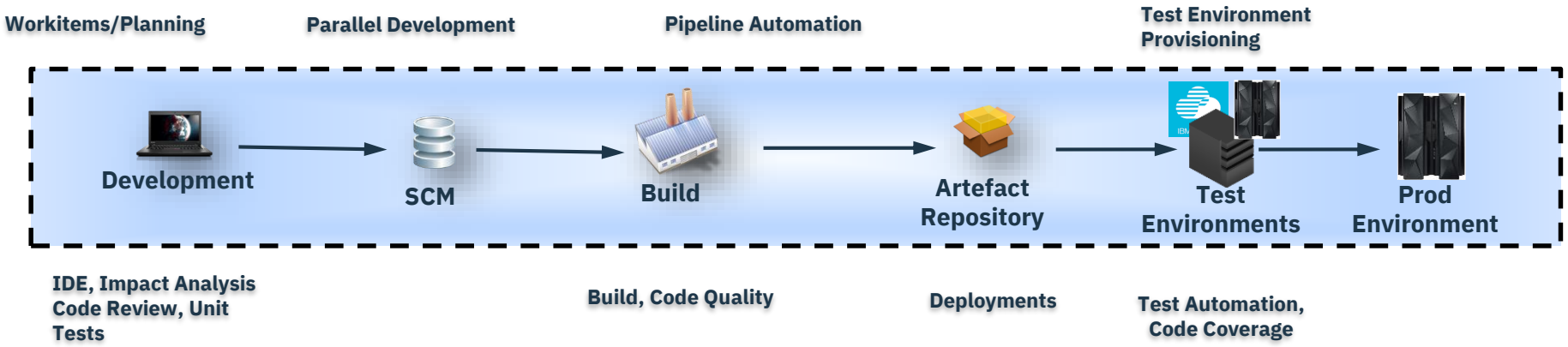


Applications and teams move at *variable speeds*…

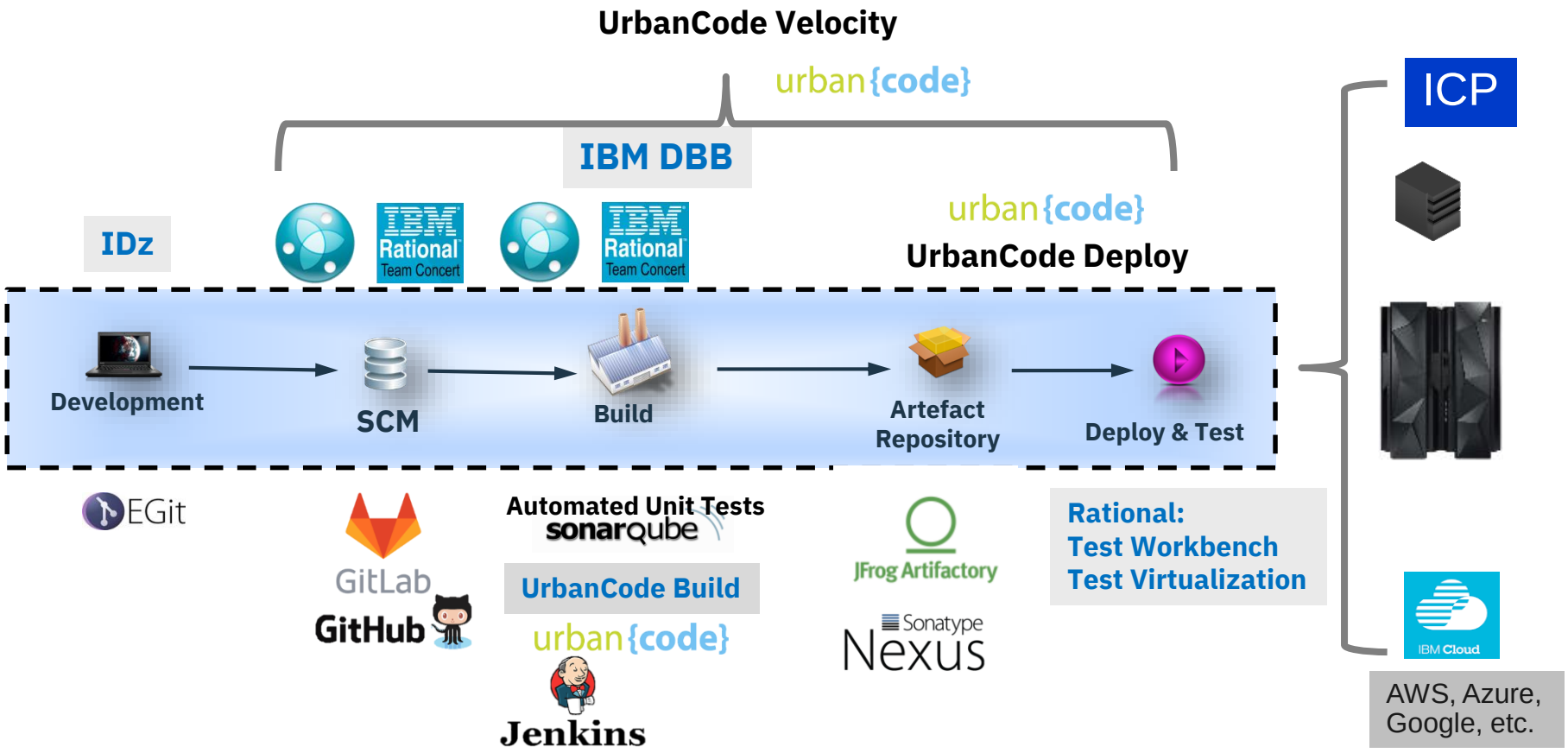


End to End Strategy

- End to End strategy sets direction for steps along the way
- Single DevOps pipeline or as close as possible improves collaboration and reduces friction between teams
- Removing the differences that provide no value from z/OS to allow it to be part of any pipeline



Open DevOps Pipe Line across the Enterprise – z/OS & Distributed

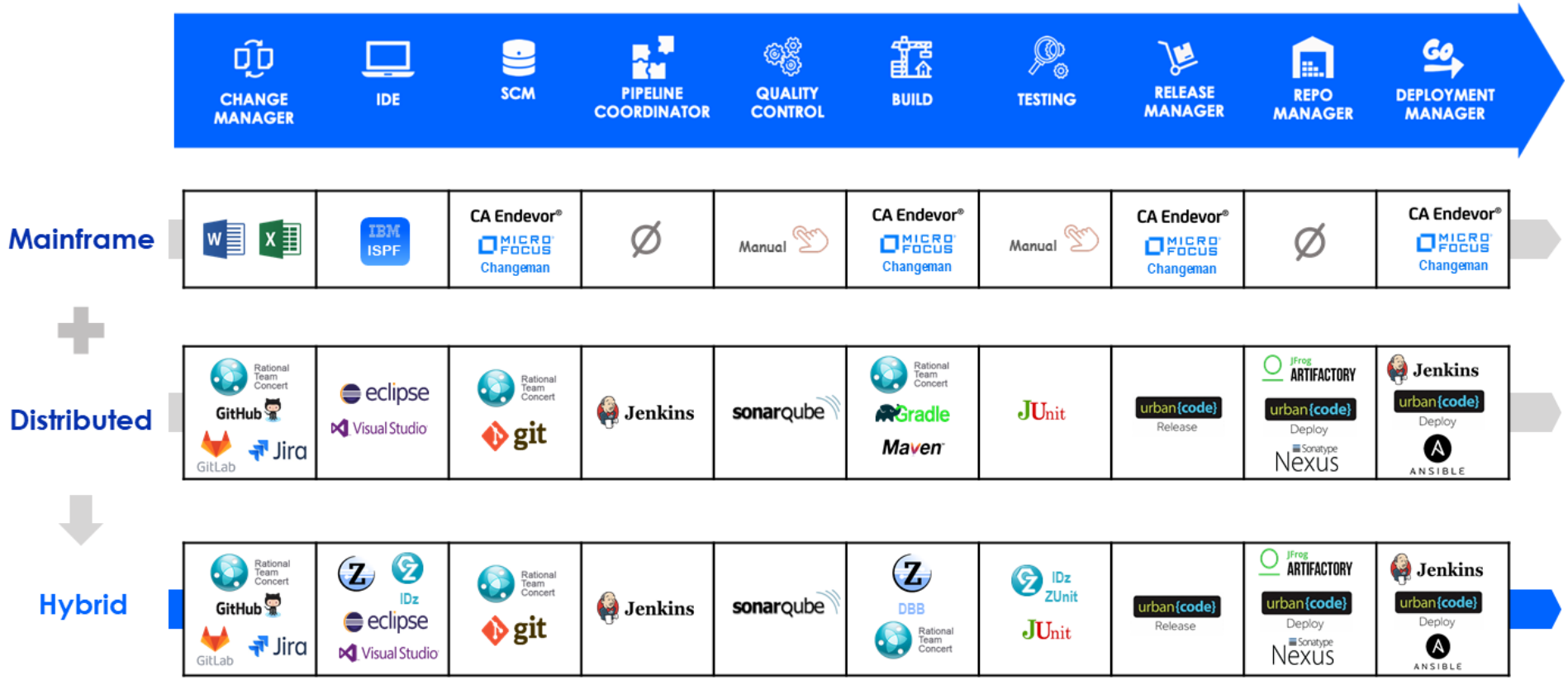


Leveraging what is being used elsewhere that works for all platforms, streamline software development processes **Build on your current pipeline .....**

Why do you force mainframe developers to use tools you would not make the distributed teams use?

	Mainframe	Distributed
Pipeline Coordination	Jenkins	
Release Mgmt	UrbanCode Release	
Deployment Mgmt	Jenkins or UrbanCode or Ansible	
Artifact Repo	Artifactory, Nexus or UrbanCode	
Requirements/Tasks	Issues or Rational Team Concert or Jira	
Quality Control	SonarQube	
Testing	Multiple, JUnit, ZUnit	
Source Control	Rational Team Concert or Git	
Build	Maven, Gradle, Make, IBM DBB	
Development	Eclipse (including IDz) , Visual Studio	

Imagine a unified pipeline on Z



The value of early and extensive testing

"80% of development costs are spent identifying and correcting defects" **

During the Coding or Unit Testing phases

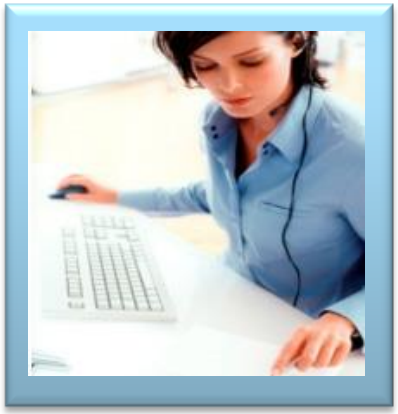
$80/defect

During the BUILD phase

$240/defect

During Quality Assurance or the System Test phases

$960/defect



Once released into production

$7,600/defect

+

Law suits, loss of customer trust, damage to brand

****National Institute of Standards & Technology**

Source: GBS Industry standard study

Defect cost derived in assuming it takes 8 hours to find, fix and repair a defect when found in code and unit test.
Defect FFR cost for other phases calculated by using the multiplier on a blended rate of $80/hr.

DevOps for Enterprise Systems

Key Takeaways

1. **DevOps is about transforming application development and delivery in order to accelerate digital innovation.**
   So DevOps is a topic for both business and IT roles in the organization.

2. **You don't buy DevOps, you do DevOps.** *DevOps is an approach, a mindset – a combination of culture, process and technology (including infrastructure, tools and services).*

3. **DevOps is not only about the hand-off between Development and Operations.** *DevOps is about applying lean and agile principles across the application delivery lifecycle (biz-dev-test-deploy-operate) to achieve continuous delivery of digital innovation.*

Key concepts: automation, feedback loops.

Questions / Comments

Agenda

9:00 Introductions
9:20 DevOps on z Systems Introduction
9:45 Application Delivery Foundation (ADFz):
- **IBM Developer for z Systems (IDz)**
- **Problem Determination Tool (PD Tools)**

10:15 Break

10:30 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 z Open Unit Test: Overview & Demonstration

1:00 Day in the Life - Using Application Discovery, DBB/Git/Jenkins and UCD

2:00 – 4:30 Choose one Optional lab:

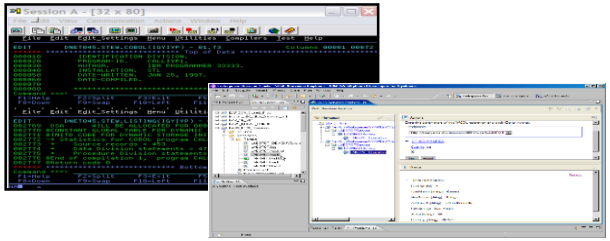
→ LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
→ LAB 2C Application Discovery : Find a candidate API from Catalog Manager Application
→ LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
→ LAB 5: **UCD** – Create UrbanCode Deploy infrastructure and deploy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

Lecture Labs Breaks

Hardware has been improved.. How about our Software?

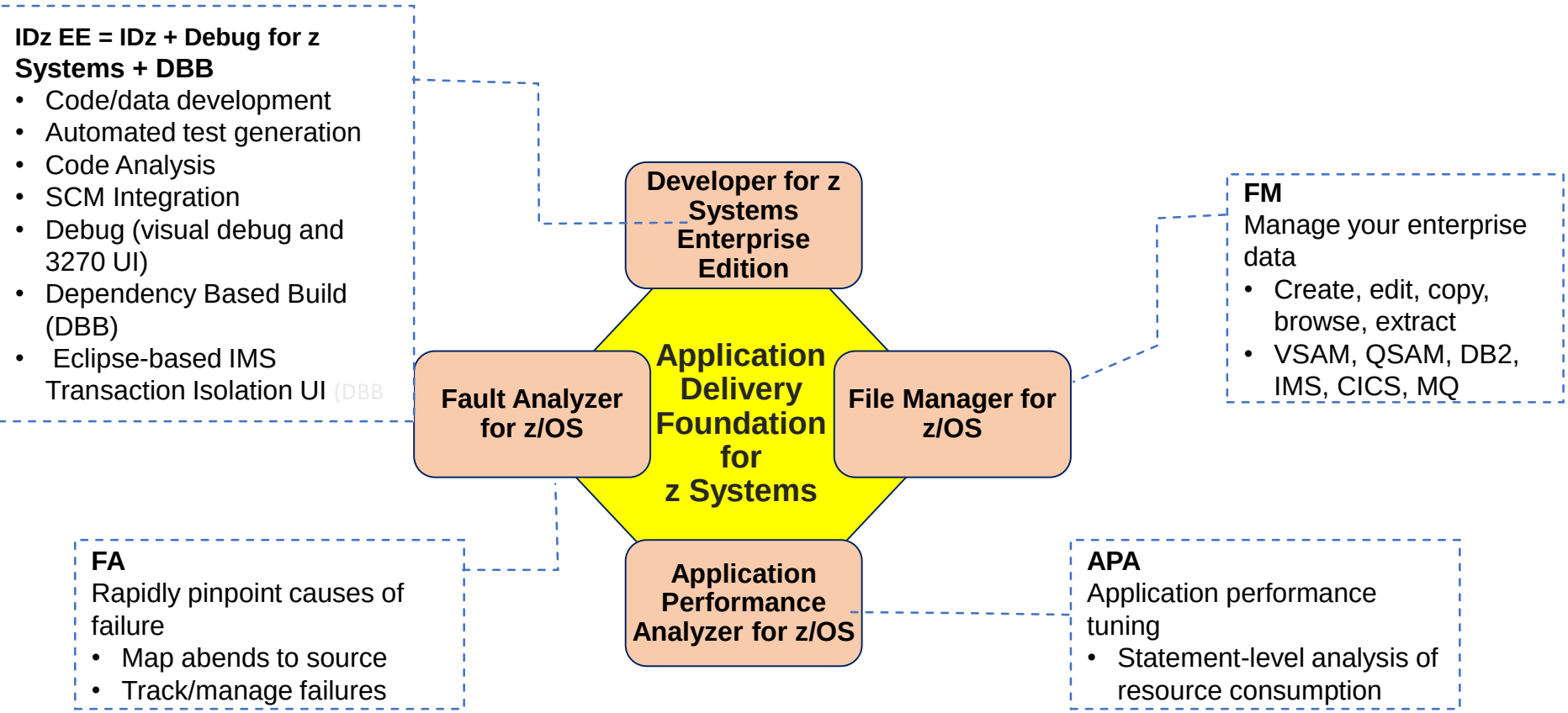
In September 1956, IBM launched the 305 RAMAC, the first computer with a hard disk drive (HDD). That HDD weighed over a ton and stored 5MB of data.

Makes you appreciate your 64 GB jump drive, doesn't it?

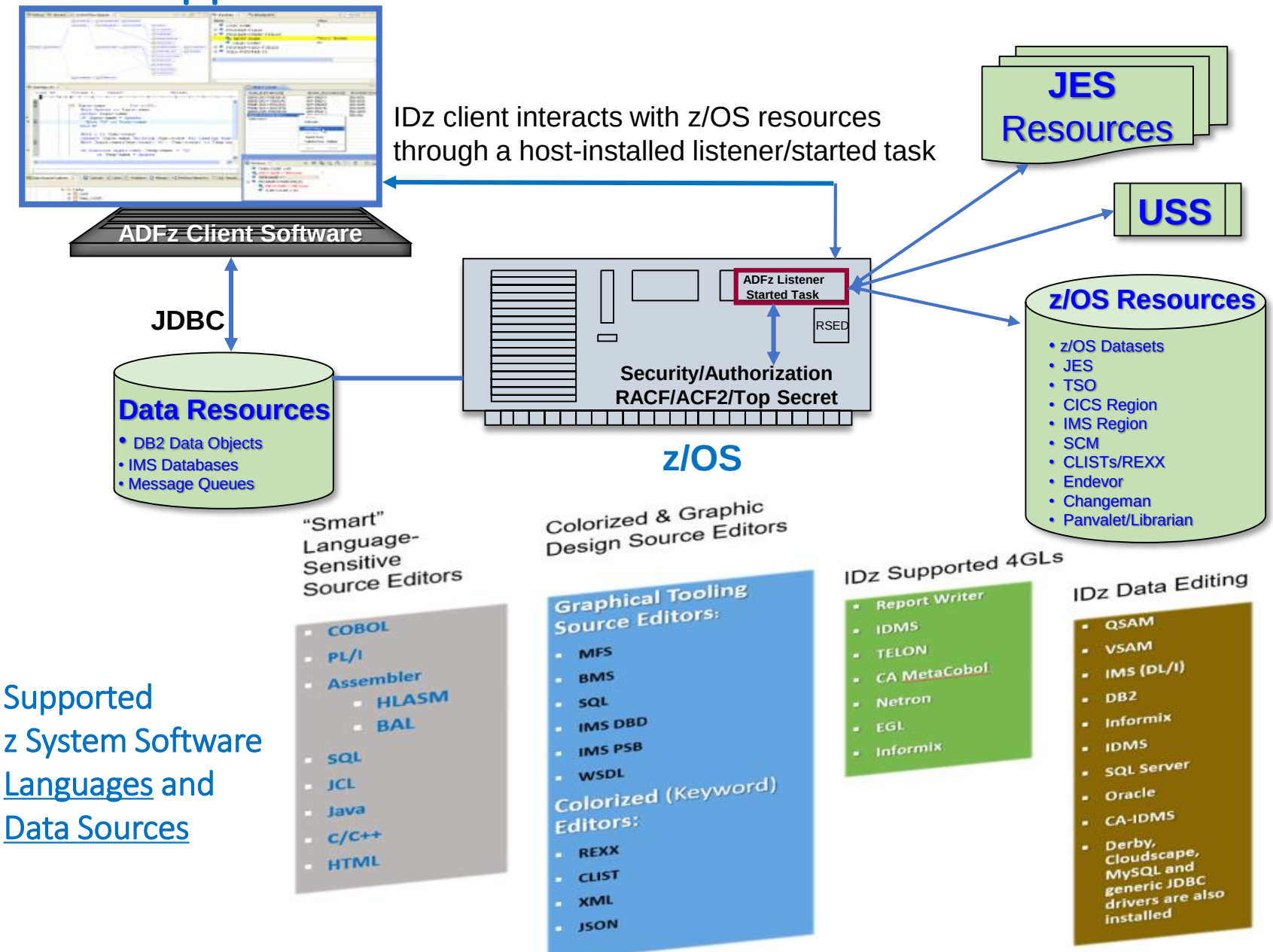


→ Software also needs Improvements….

What is Application Delivery Foundation for z Systems (ADFz)?



IDz: Supported Environments and Architecture

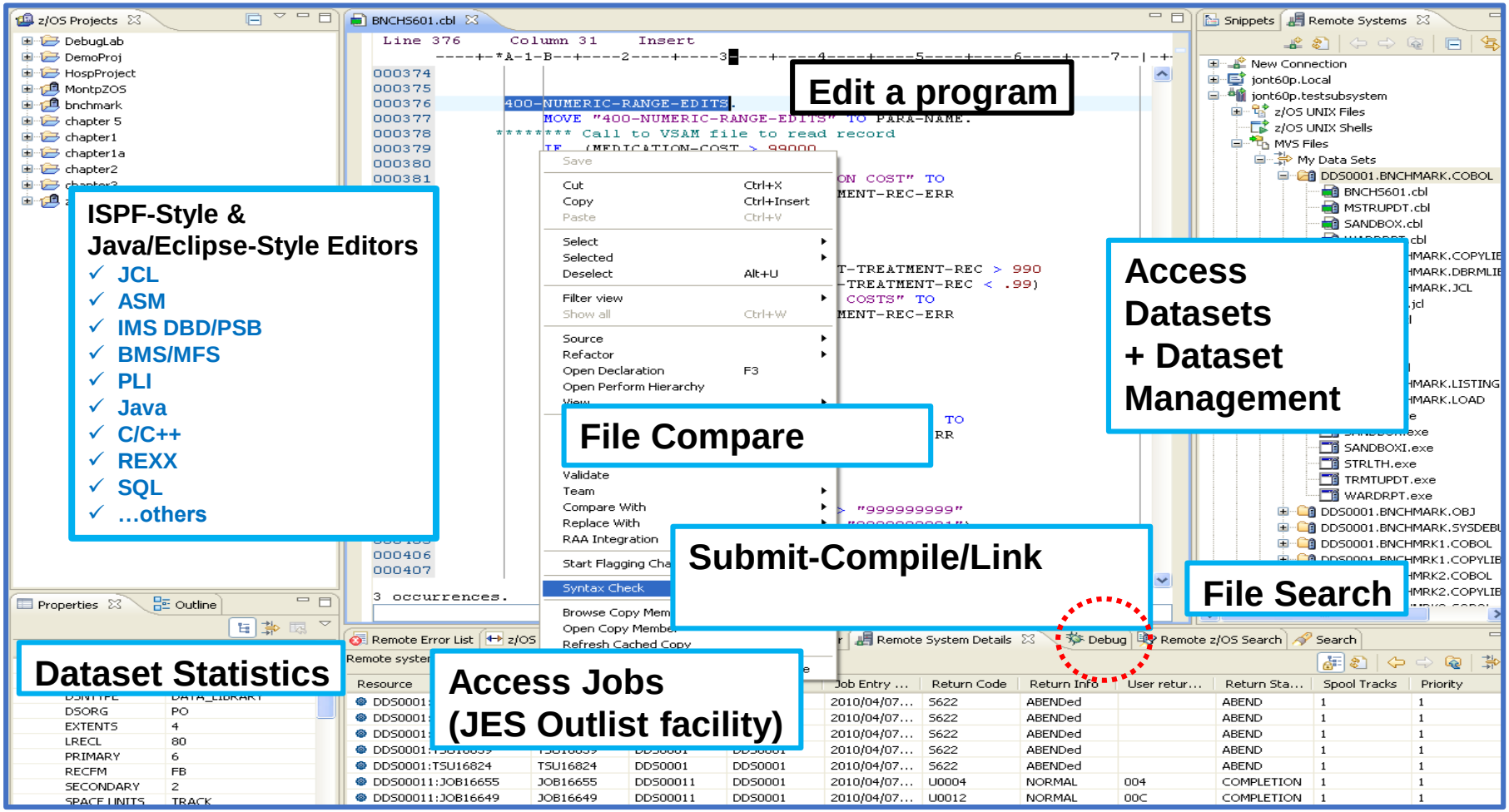


Supported z System Software Languages and Data Sources

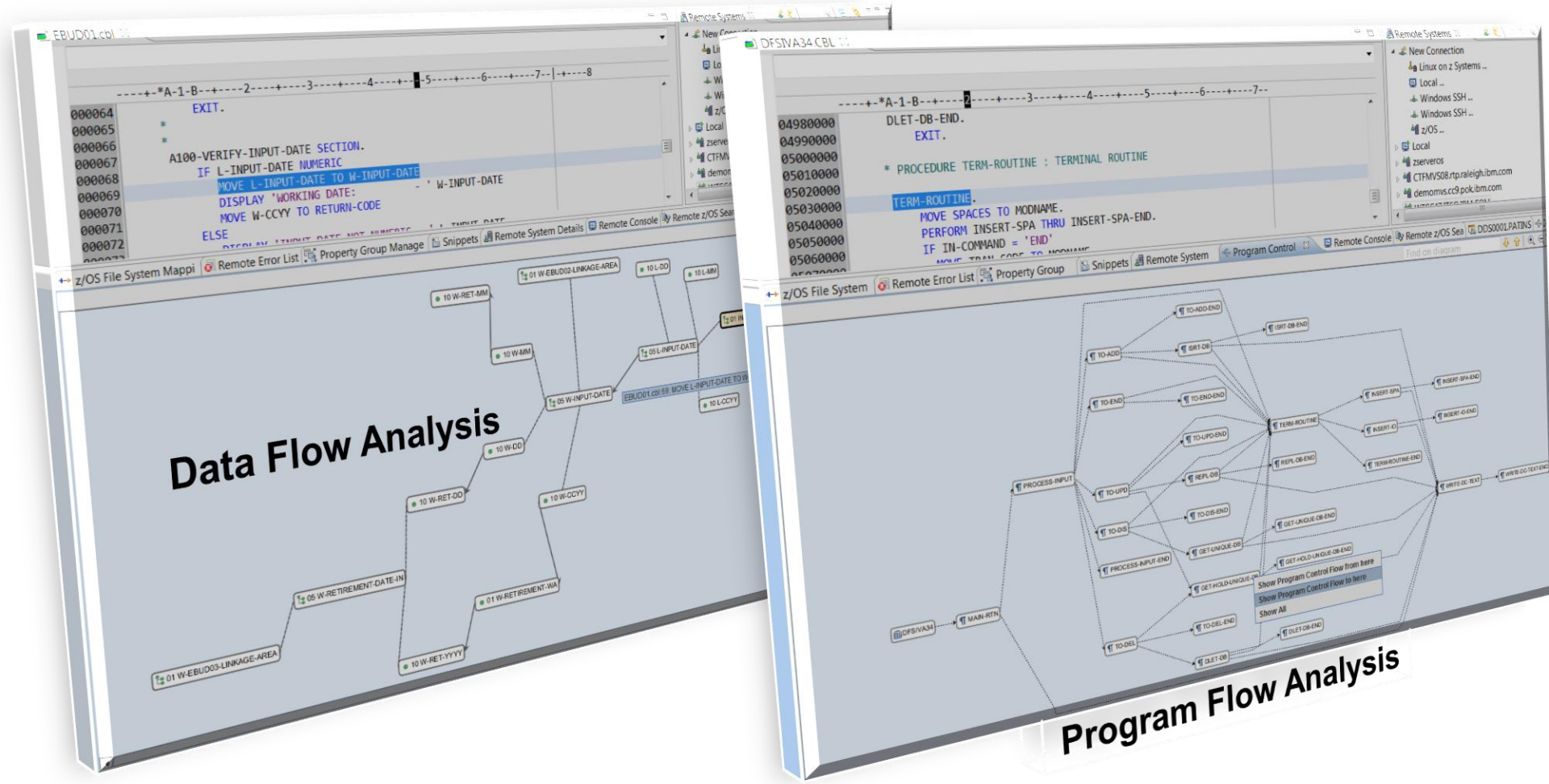
Using **IDz** for Development & Testing

IDz is the ideal DevOps Entry-Point !!!

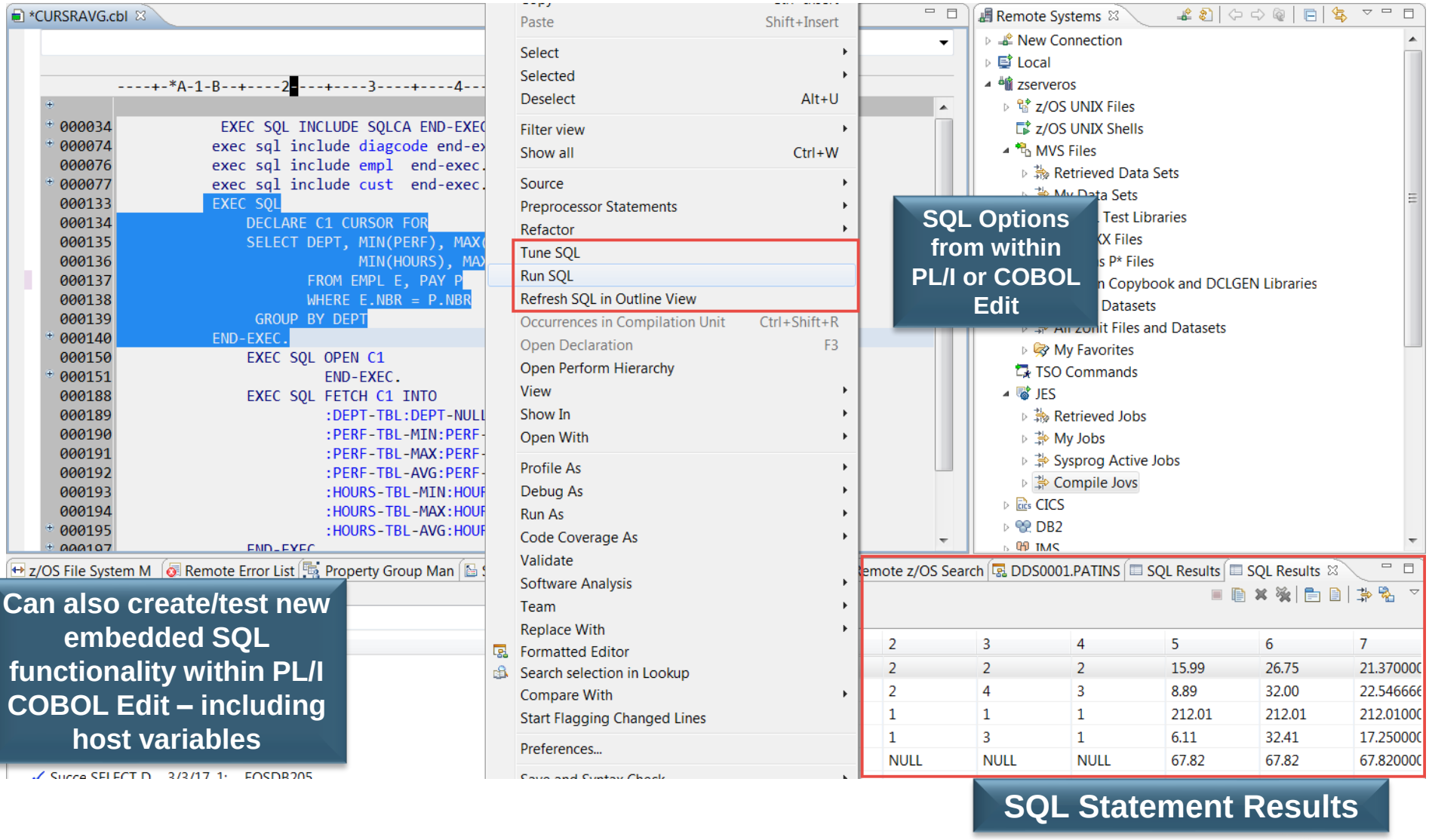
Instead of maneuvering across ISPF panels and working ***sequentially***, in IDz the functionality you need is always in-focus … you work ***concurrently***



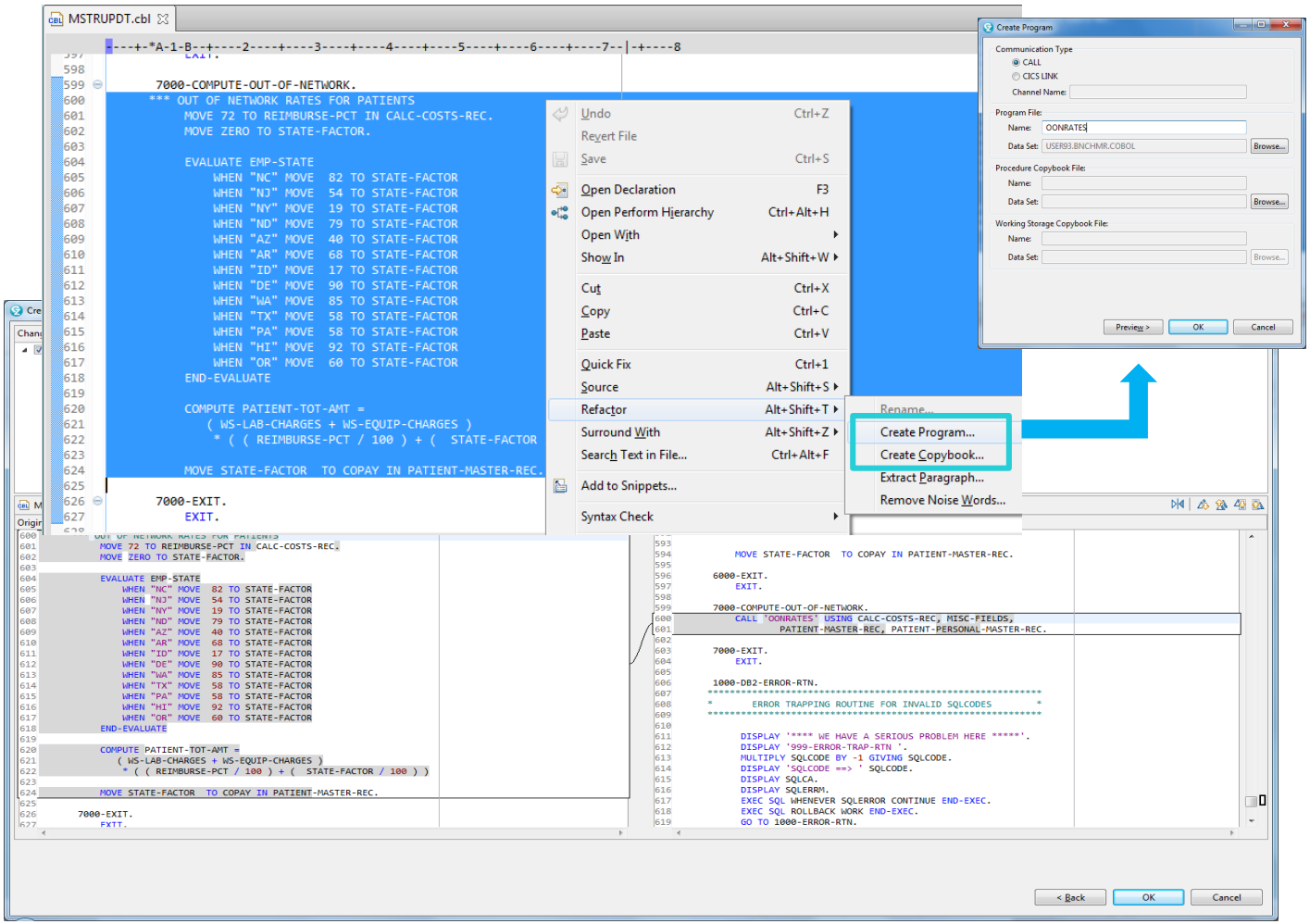
Program Analytics



SQL Coding within IDz editor



COBOL Program Refactoring

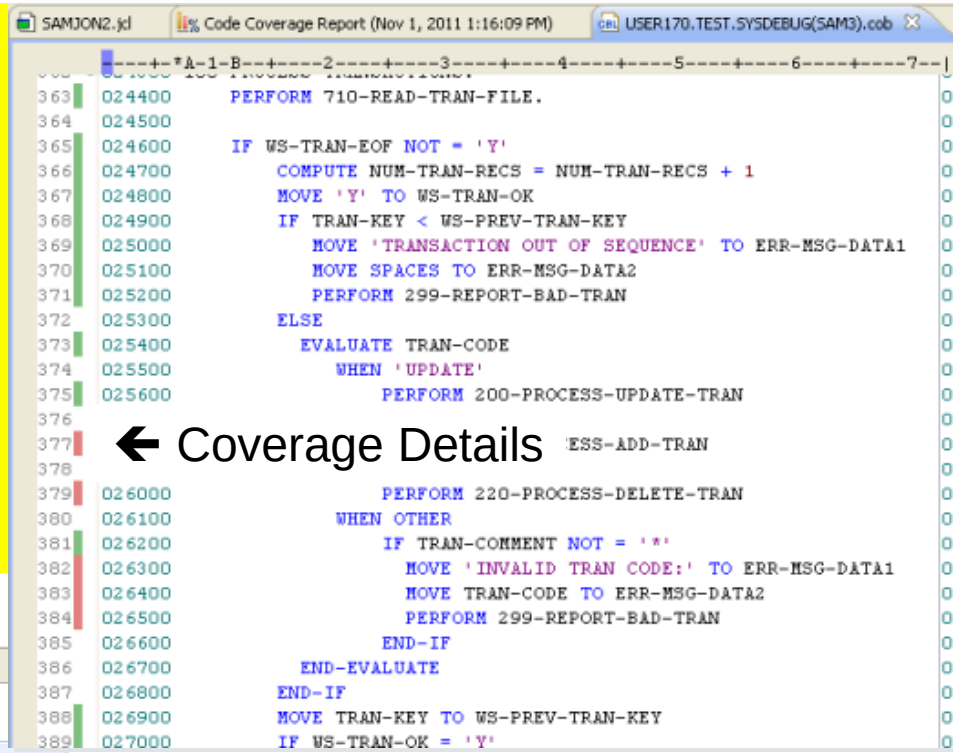


- **Modularization**
  - Increase application testability
    - Easier to create automated unit test cases
  - Accelerate development and deployment
    - Separate presentation logic from business logic and data access logic
    - Allows for concurrent development
- **Maintainability**
  - Improve readability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

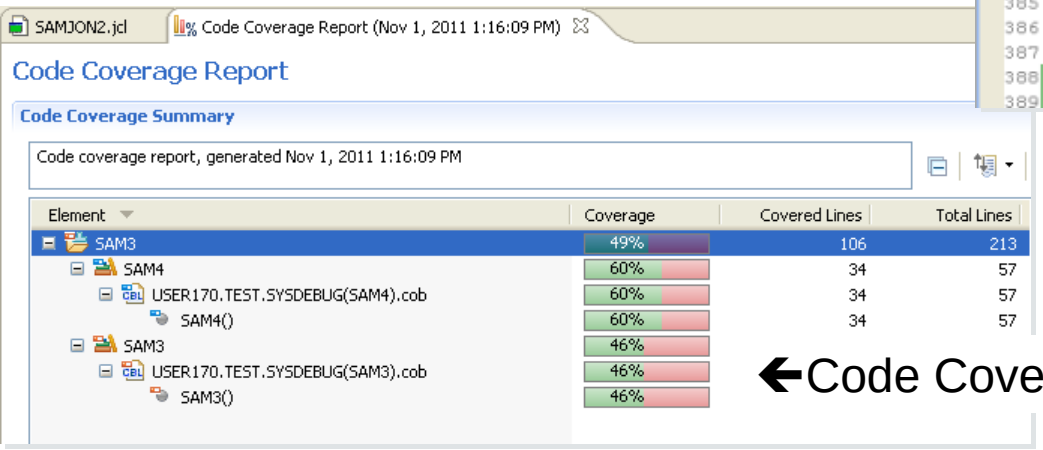
Code Coverage

Extension to Testing & Debugging:

- Measures testing quality
  - Coverage % - based on input data values from screens, files and databases
- Tracks tested lines of code
- Improves application quality
- Focuses testing resource usage
- Reports on tested code and trends
- Supports: Batch, CICS and IMS TM



← Coverage Details



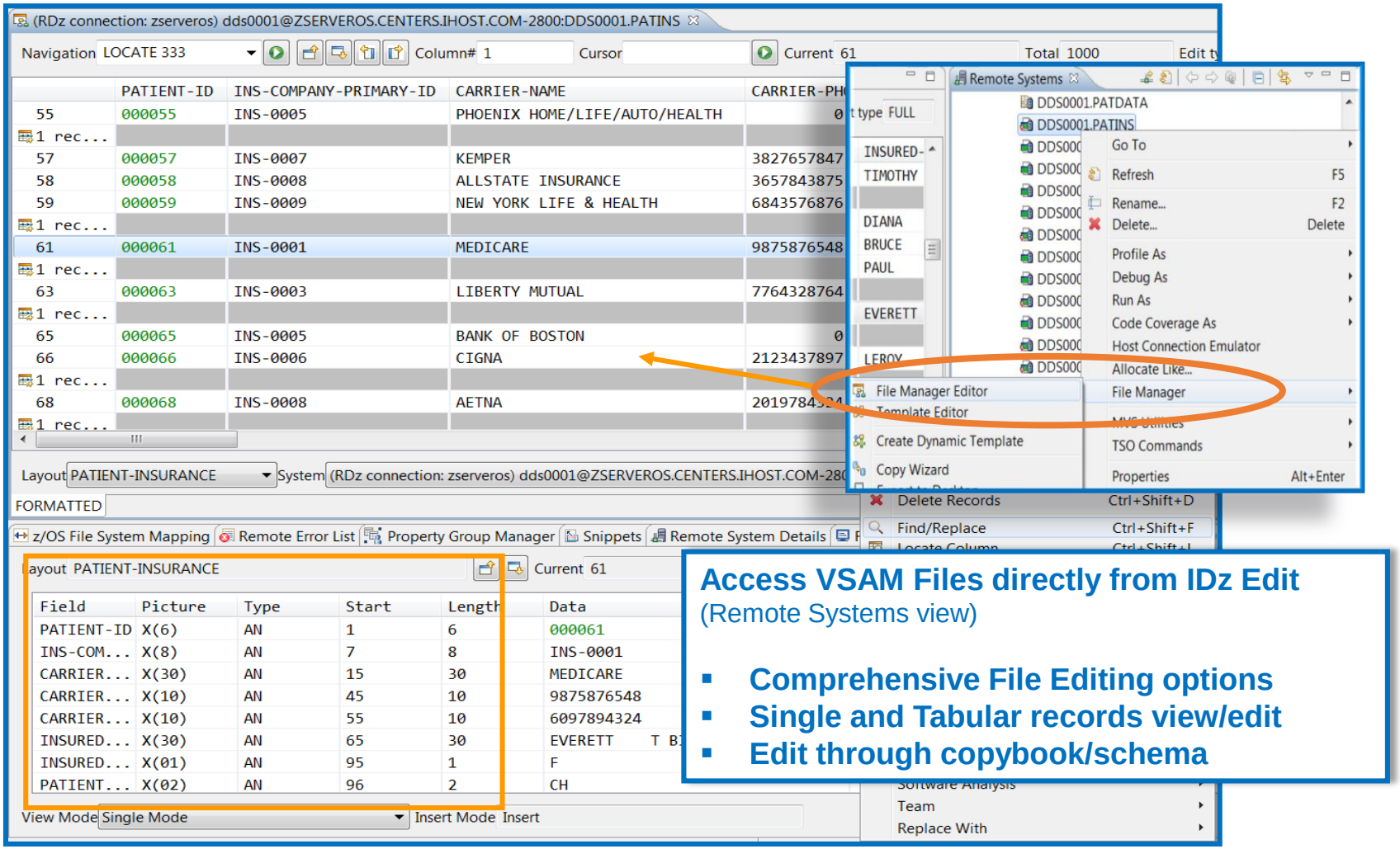
←Code Coverage Report

Code Coverage central to DevOps Continuous Testing

ADFz: “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



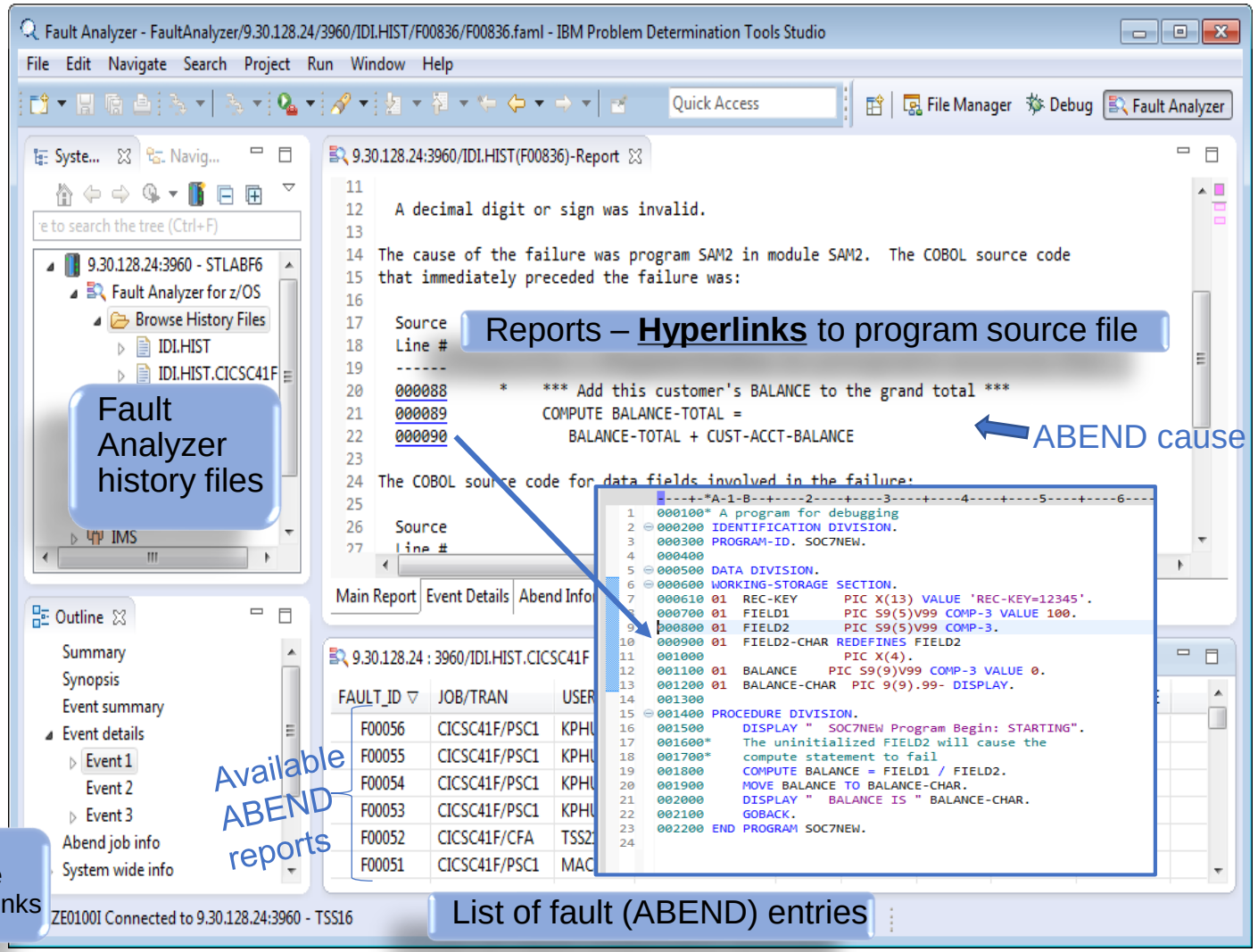
Access VSAM Files directly from IDz Edit
(Remote Systems view)

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

ADFz: “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)

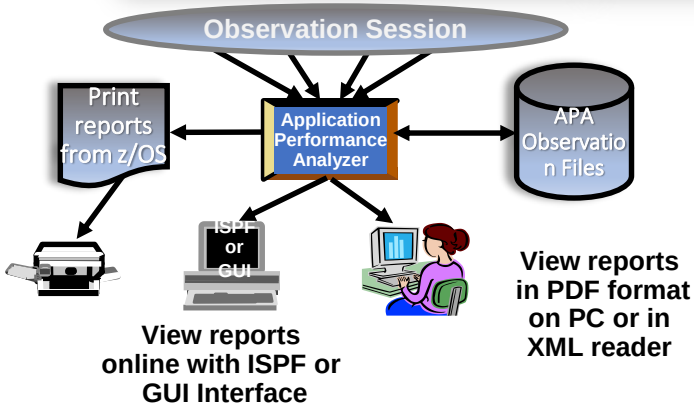
- Automatic program abend capture and reporting
- Program source-level reporting
- Provides a detailed report about program failures to help resolve them quickly
- Enables you to track and manage application failures and fault reports
- Integration with ADFz and 3270-based Interface



ADFz: Application Performance Analyzer (APA) for z/OS

Provides rapid pin-pointing of enterprise application bottlenecks

TSO | BATCH | IMS | CICS | DB2 | DB2 SP & DDF | WAS | MQ Series JAVA | USS | Adabas Natural



- 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

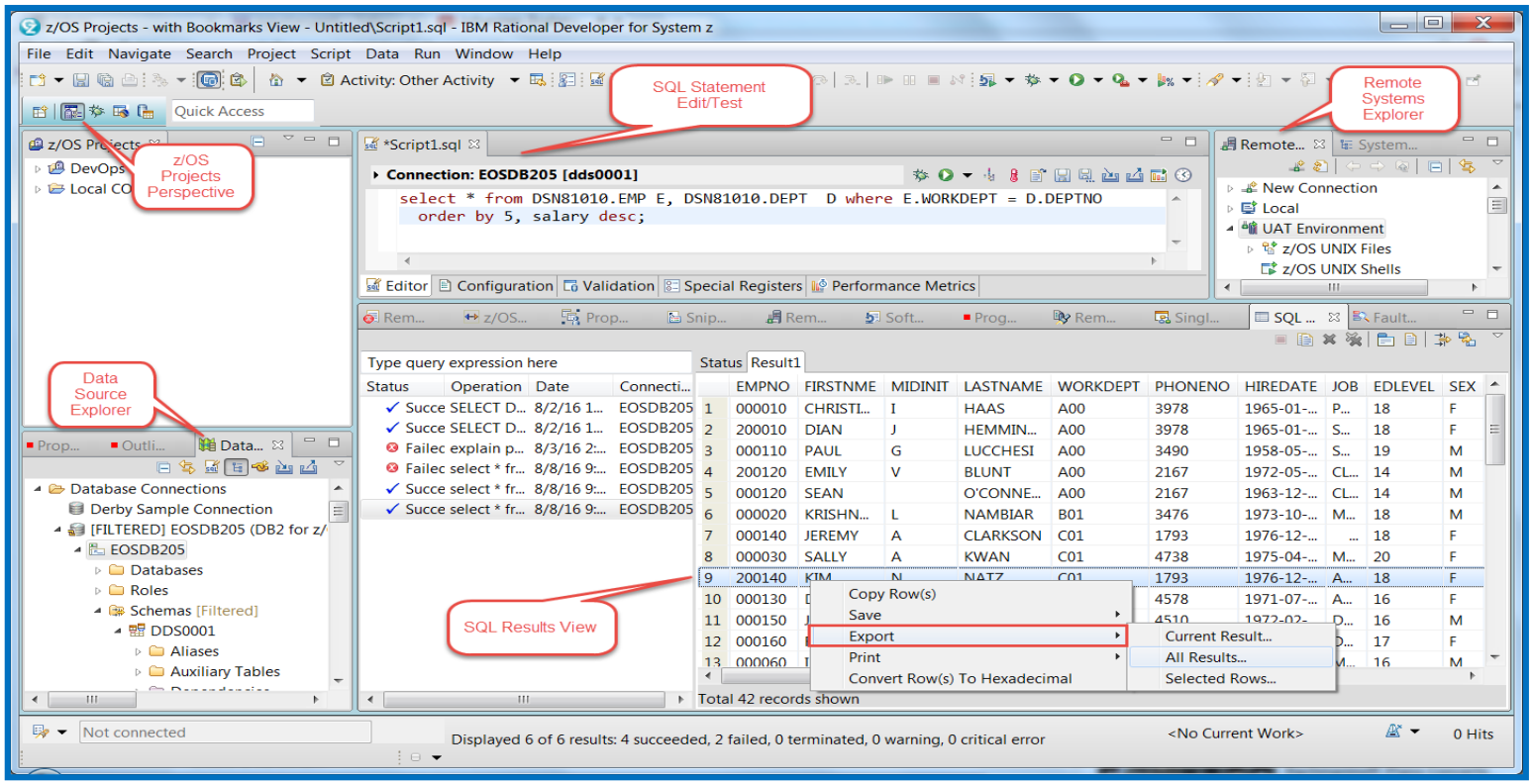
Integrated with ADFz and 3270-based Interface

- 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

IDz: Data Studio (plug-in): DB2 Development Tooling

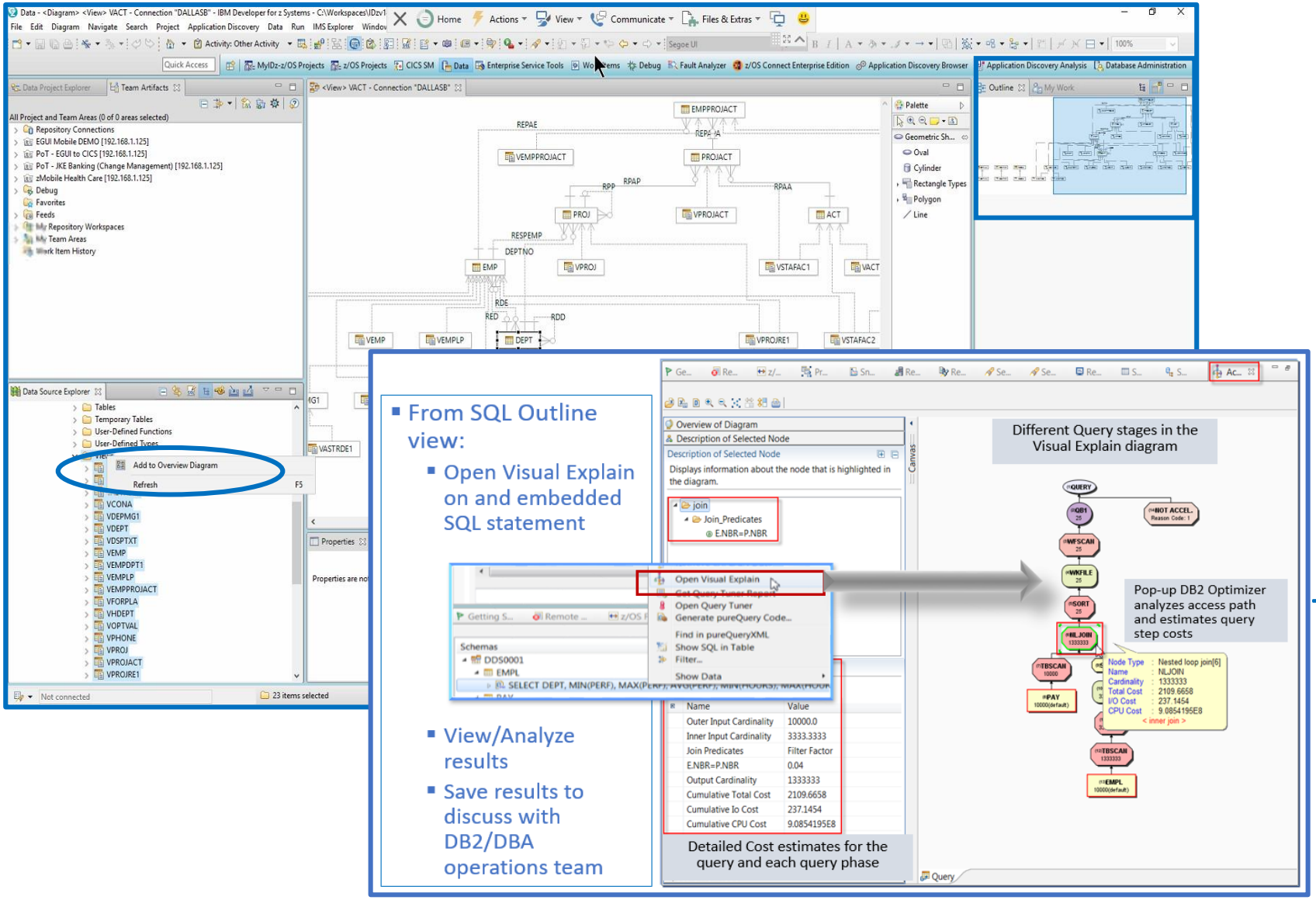
A flexible graphical interface to DB2 and SQL

- 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



IDz: ER-Diagram - Analyze DB2 Table/View Relationships

Using “Data” perspective



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 (CI) 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** (most common → Git) or **pipeline automation tool** (most common → Jenkins) **.**
- Consists of a build toolkit (Java API, Groovy Installation) installed on USS (UNIX Systems Services) and a Liberty application server that hosts build metadata (dependency data, build results) installed on Linux.

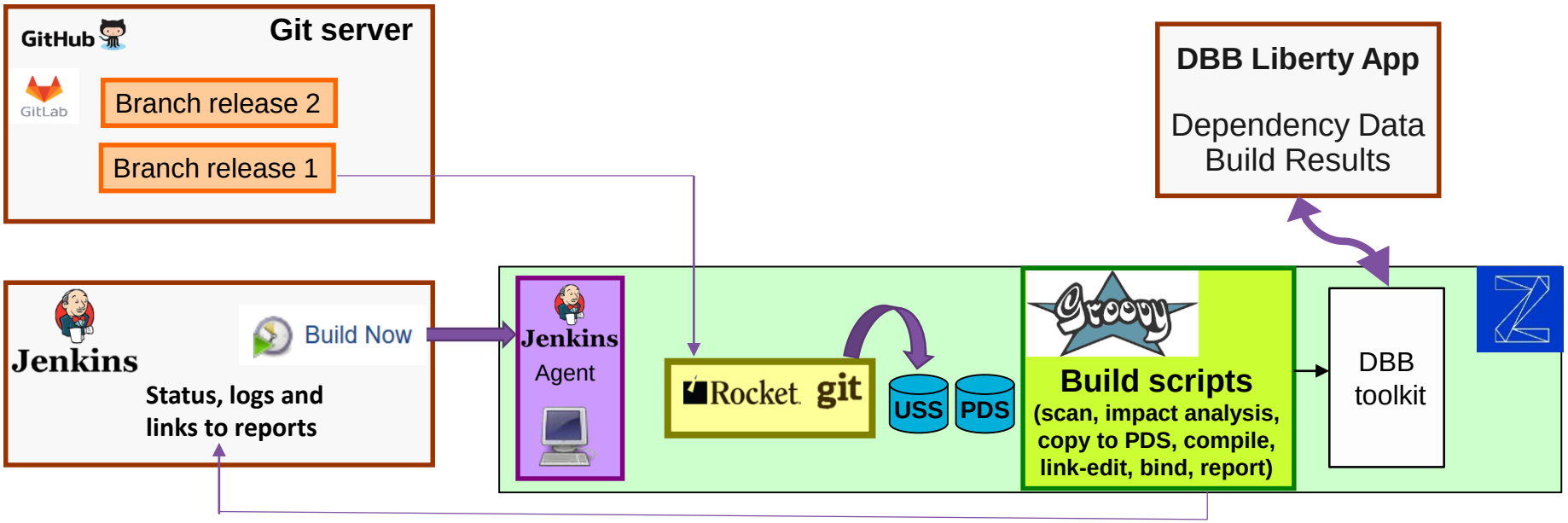
DBB Sample Groovy compile step versus JCL

```
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
```


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*)



Questions / Comments

Agenda

9:00 Introductions
9:20 DevOps on z Systems Introduction
9:45 Application Delivery Foundation (ADFz):
- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break

10:30 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 z Open Unit Test: Overview & Demonstration

1:00 Day in the Life - Using Application Discovery, DBB/Git/Jenkins and UCD

2:00 – 4:30 Choose one Optional lab:
- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (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)

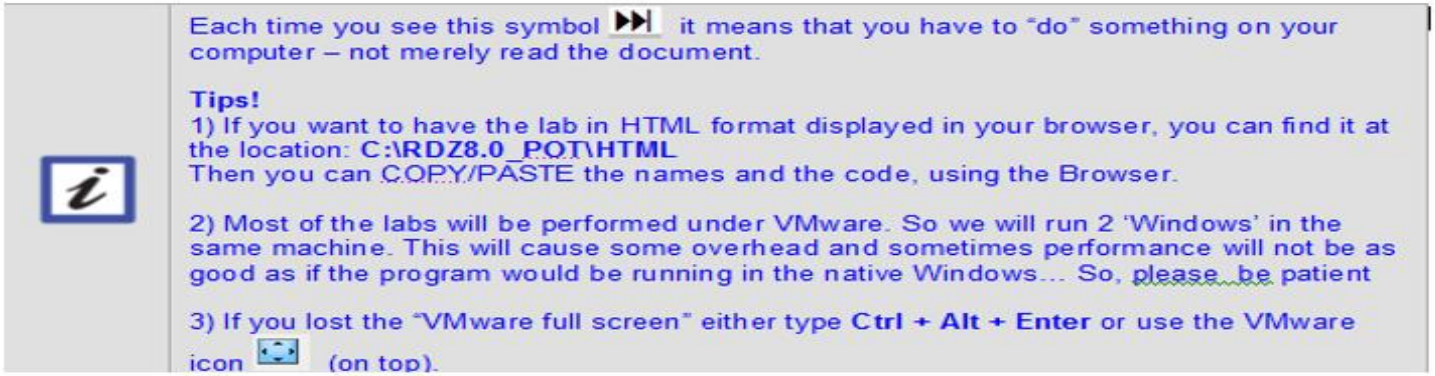
Lecture Labs Breaks

DevOps Mainframe Tooling - Labs

Labs

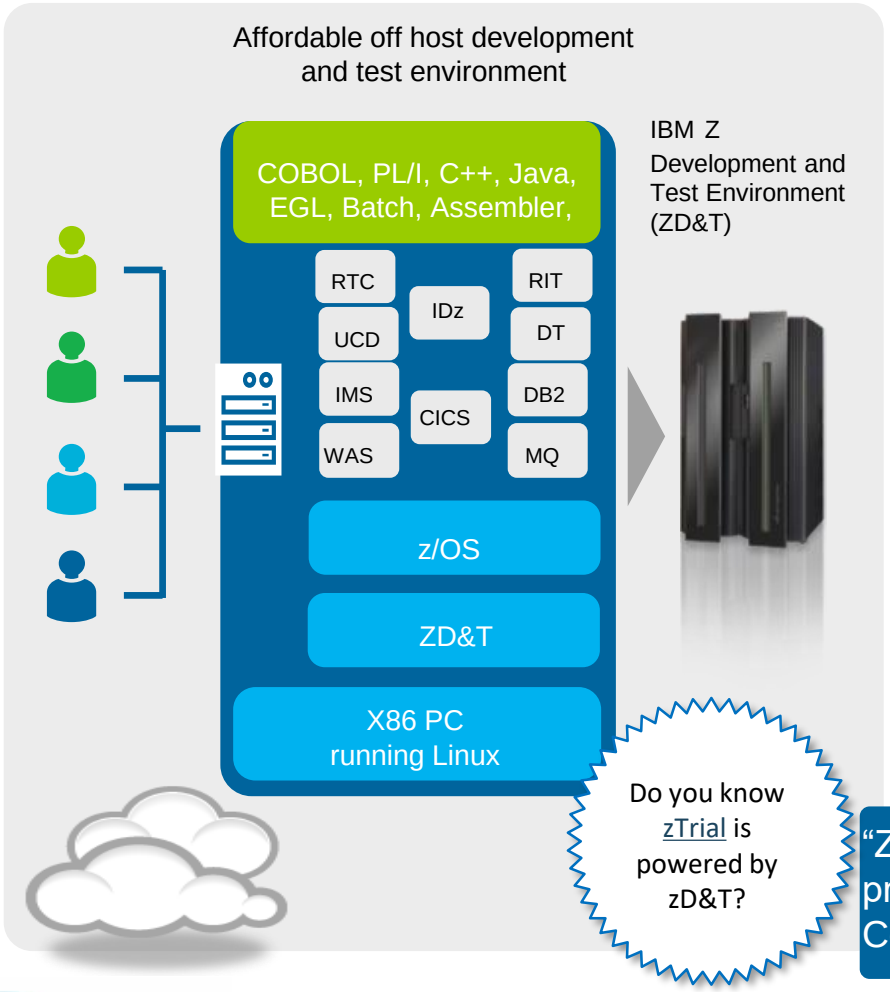
Labs using VMware …

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

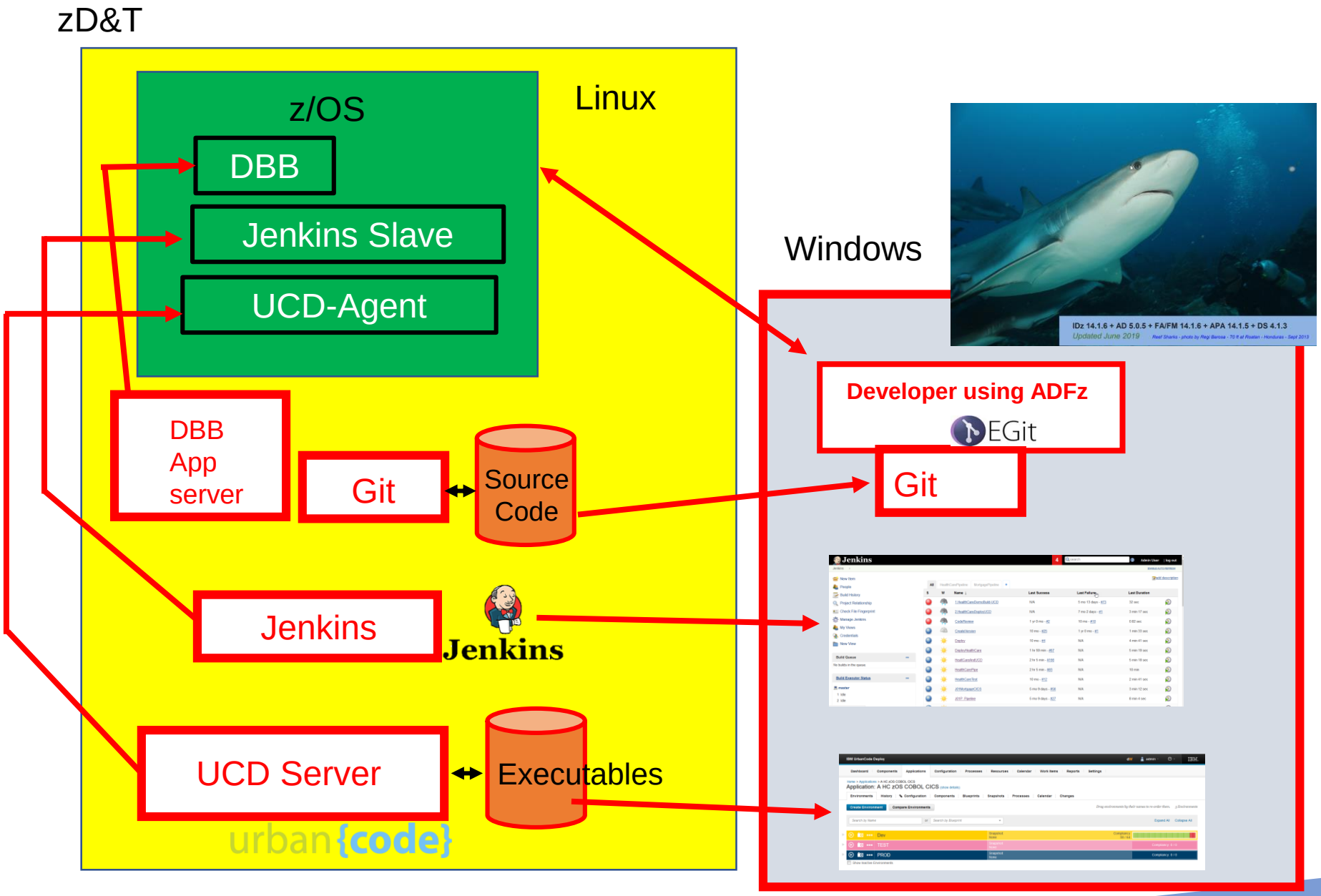
New in V12: Cloud Integration & Deployment Automation

"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

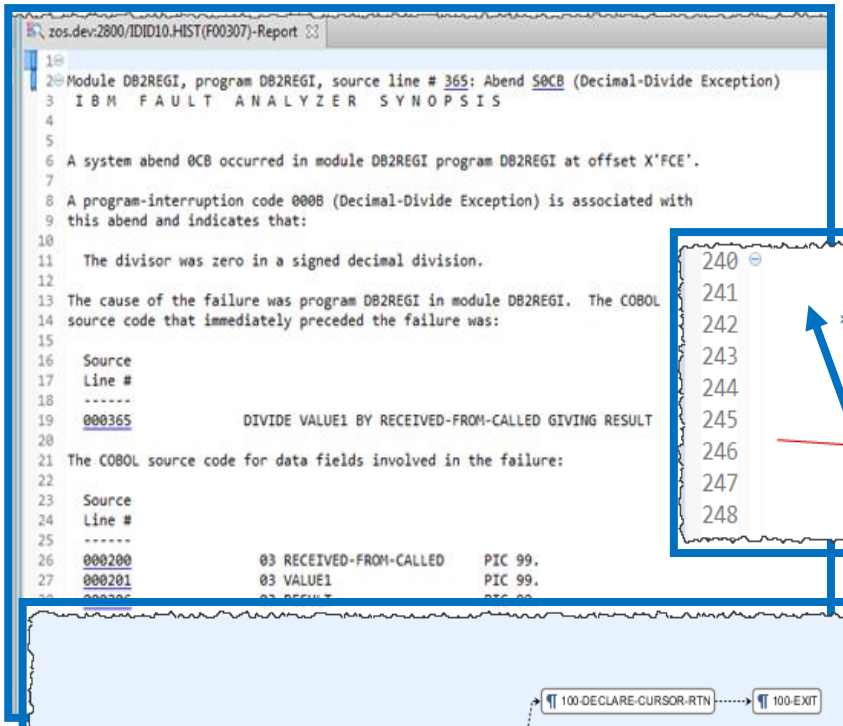
Objective

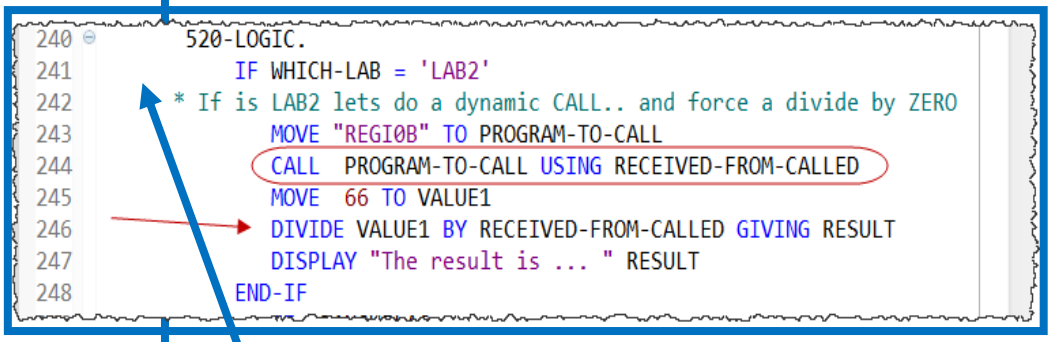
This lab will take you through the steps of using the **Application Delivery Foundation for z Systems (ADFz)** to work with a z/OS system. It will familiarize you with some of the capabilities of this product using a DB2 COBOL batch program that is ABENDing.

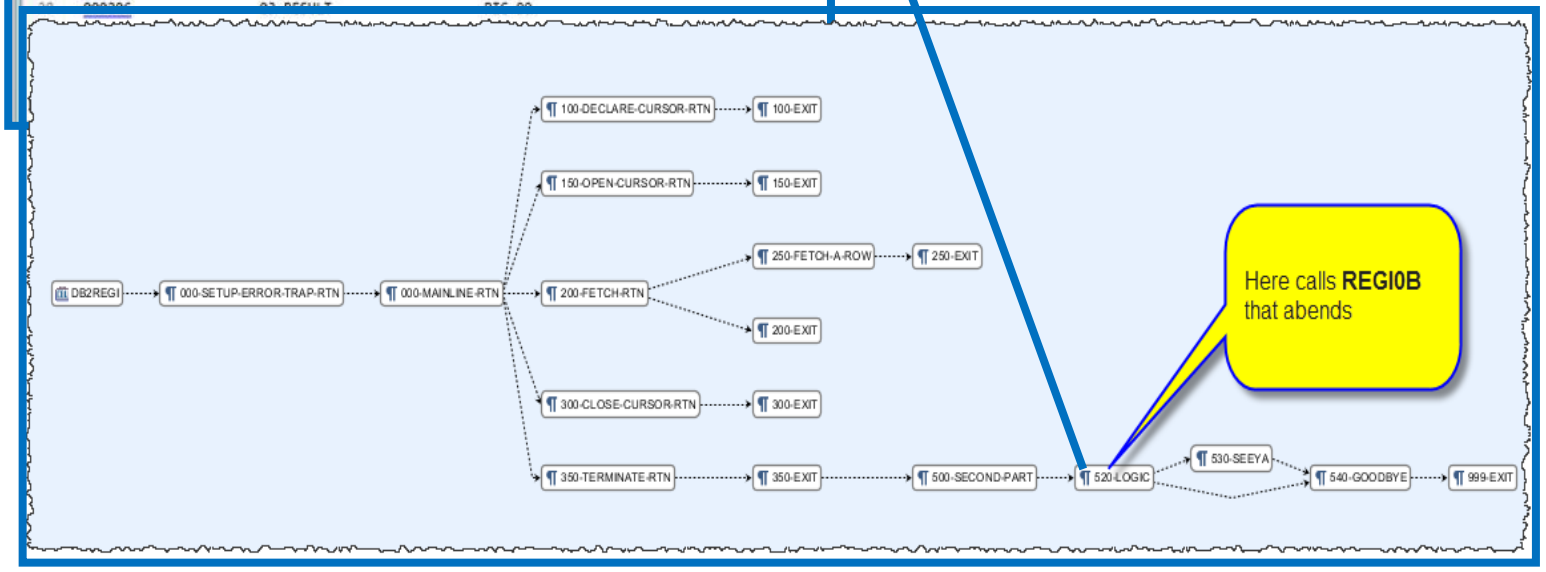
Key Activities

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







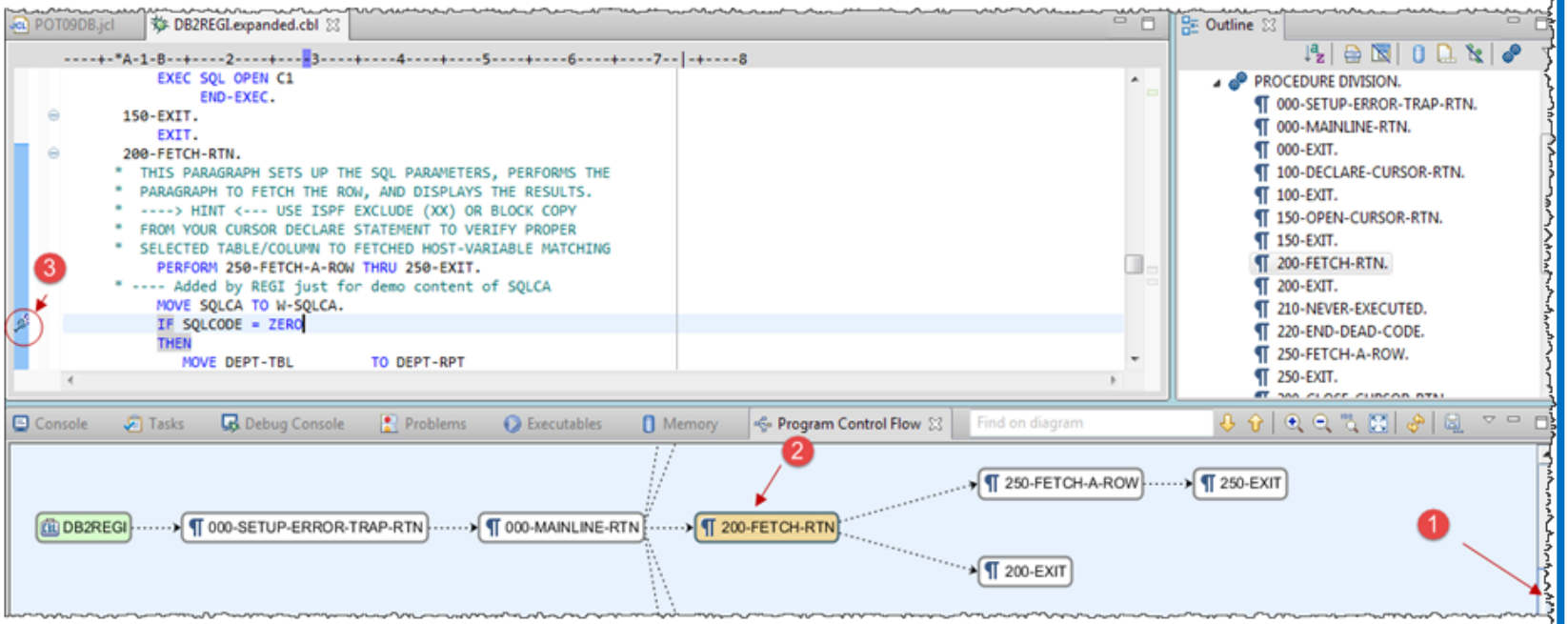
Lab 1: Don’t Miss… The Debug… Page 20

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.

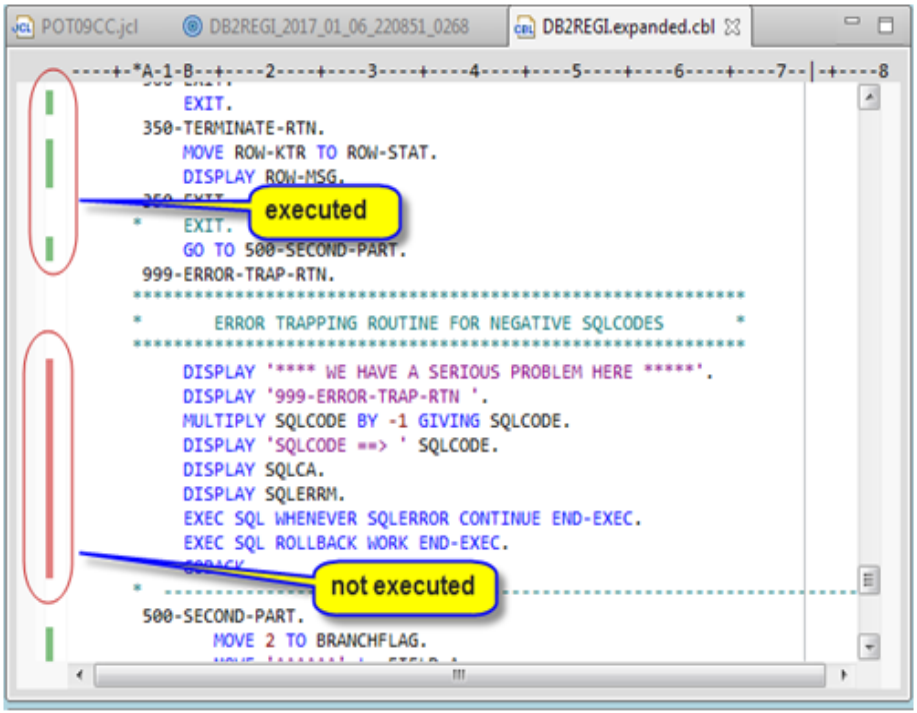
The icon is shown



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

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

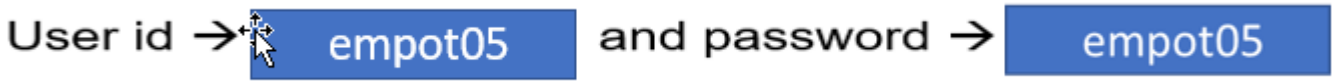
⏭ Scroll down and you will see the lines executed in **green** and the lines not executed in **Red**.



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 on z Systems Introduction
9:45 Application Delivery Foundation (ADFz):
- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break

10:30 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 z Open Unit Test: Overview & Demonstration

1:00 Day in the Life - Using Application Discovery, DBB/Git/Jenkins and UCD

2:00 – 4:30 Choose one Optional lab:
- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (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)

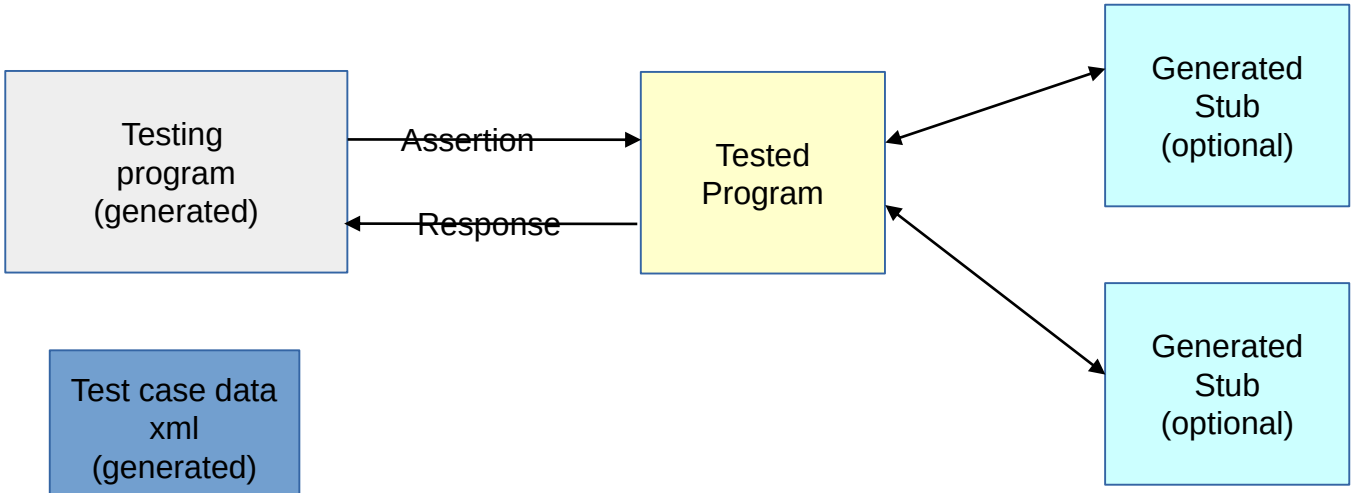
Lecture Labs Breaks

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

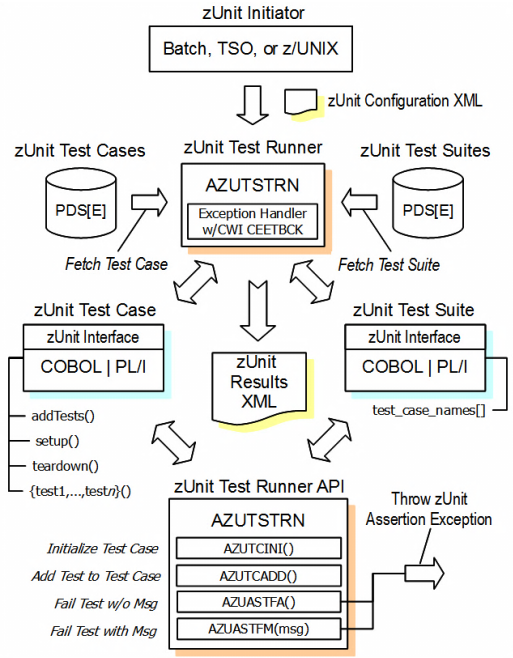
- Unit Testing is a method of testing software units to determine if they are fit for use
  - Unit Tests are usually created by developers, using development tools and unit testing frameworks
  - Unit Tests are expected to run quickly, so they can be run repeatedly
  - Some developers prefer that unit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

IDz zUnit Basics



Test case data xml (generated)

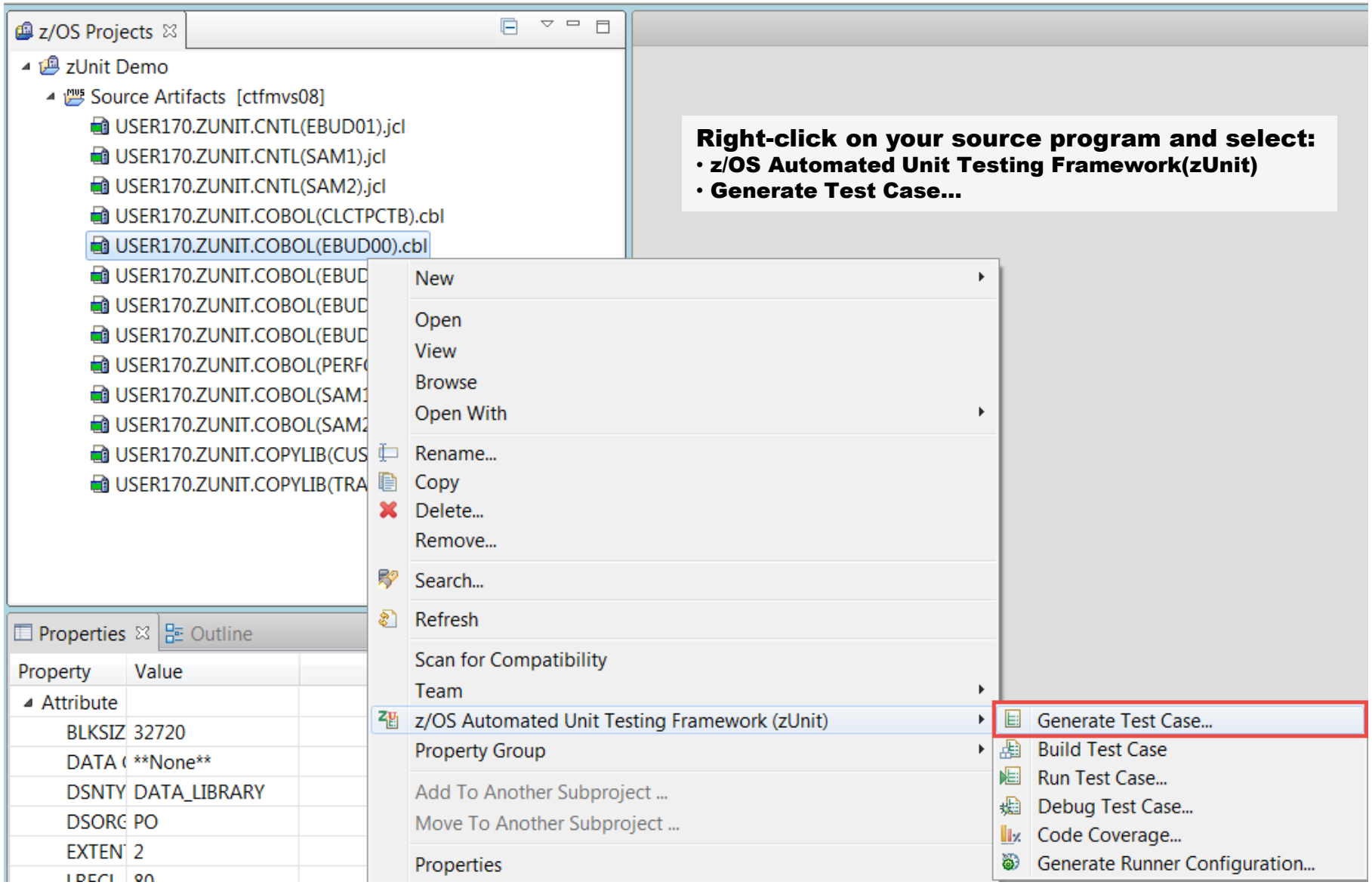
Test case results xml (generated)



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

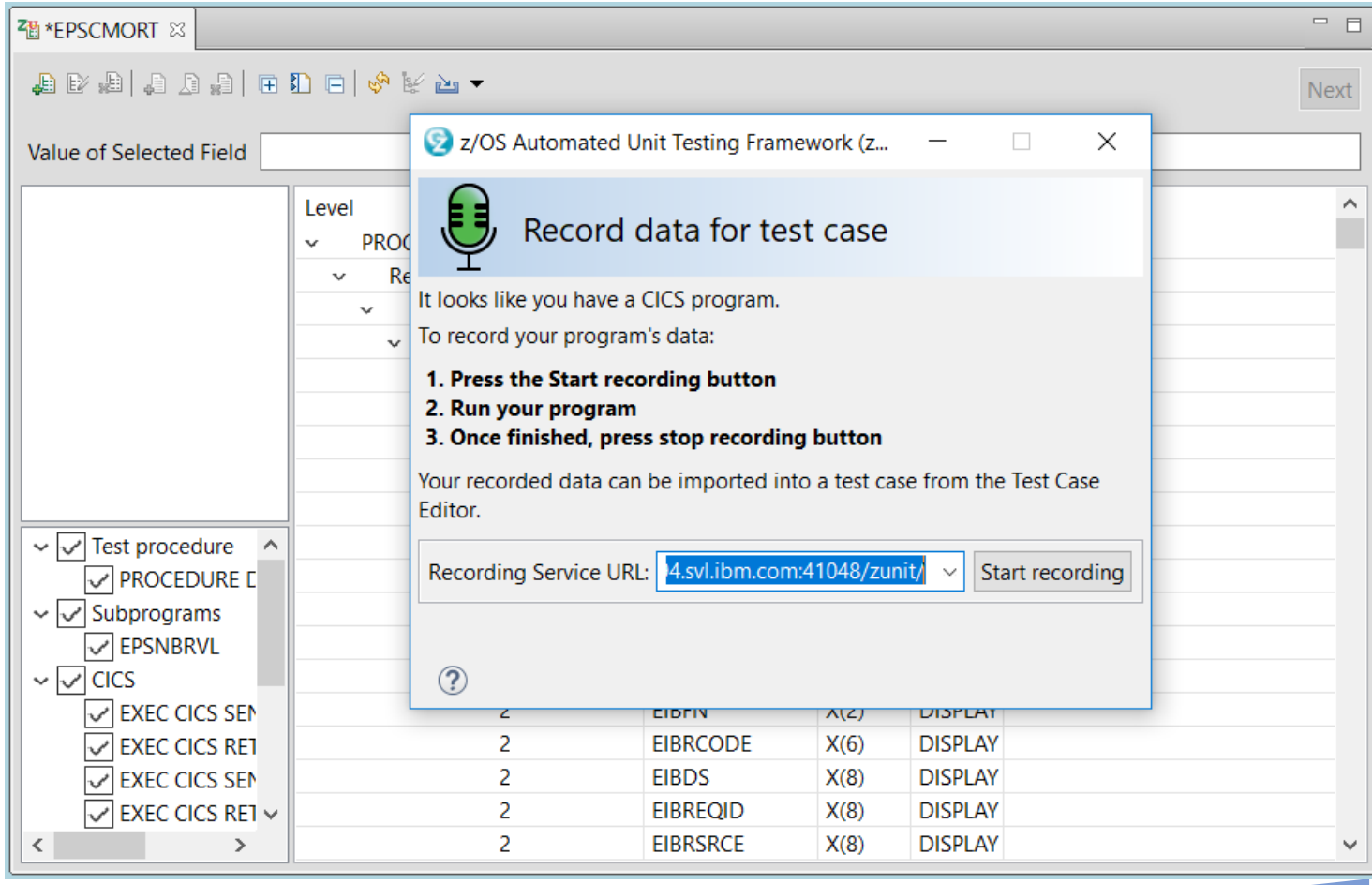
Generating the Test Case - Using the IDz Wizard



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



DevOps Mainframe Tooling

Demonstration

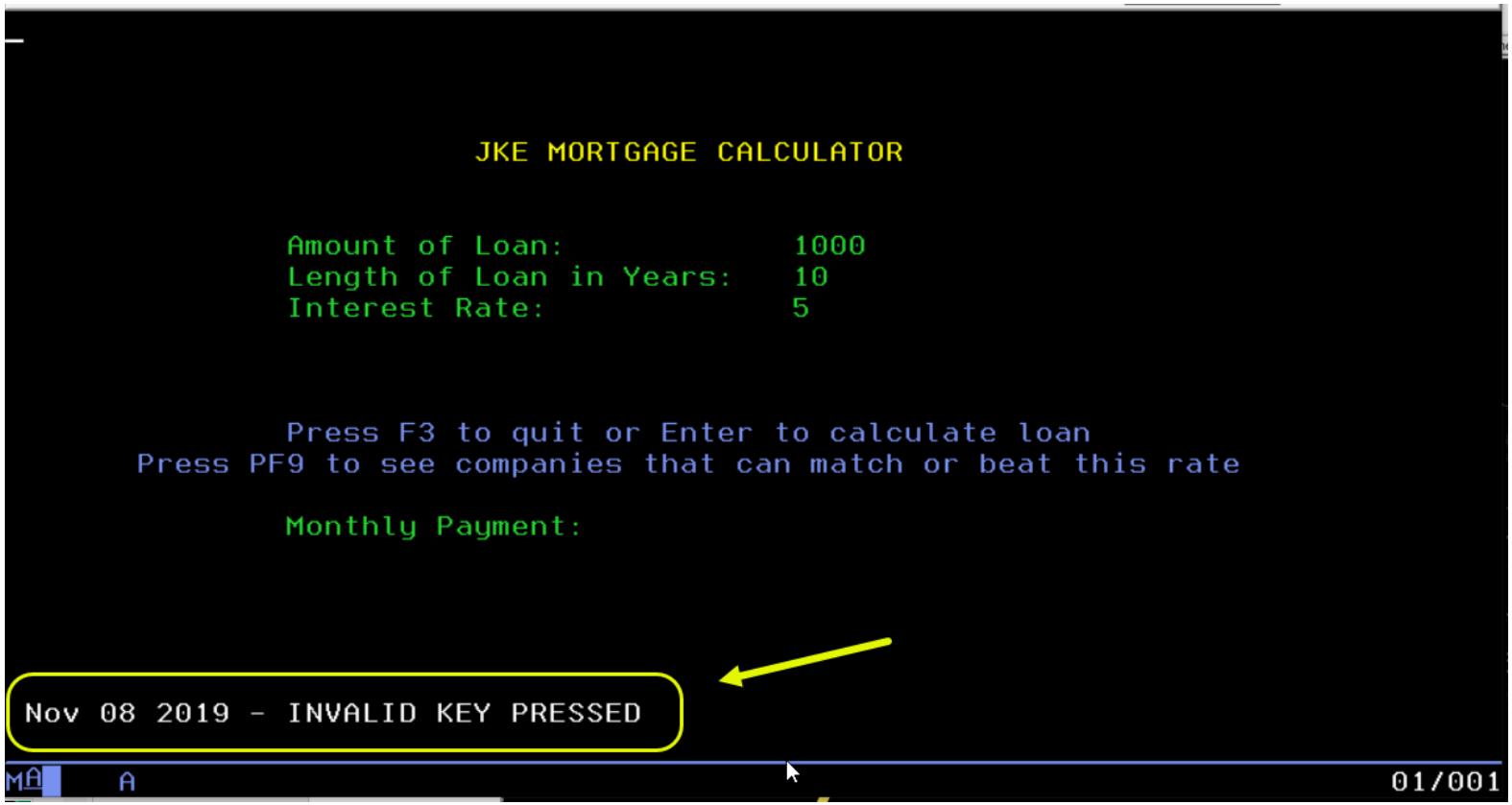
Z Unit Test using IDz (Zunit)

Demonstration

Scenario: COBOL CICS Transaction

*Using transaction **J05P** and pressing **F1***

message is displayed



Unit test on COBOL/CICS program J05CMORT

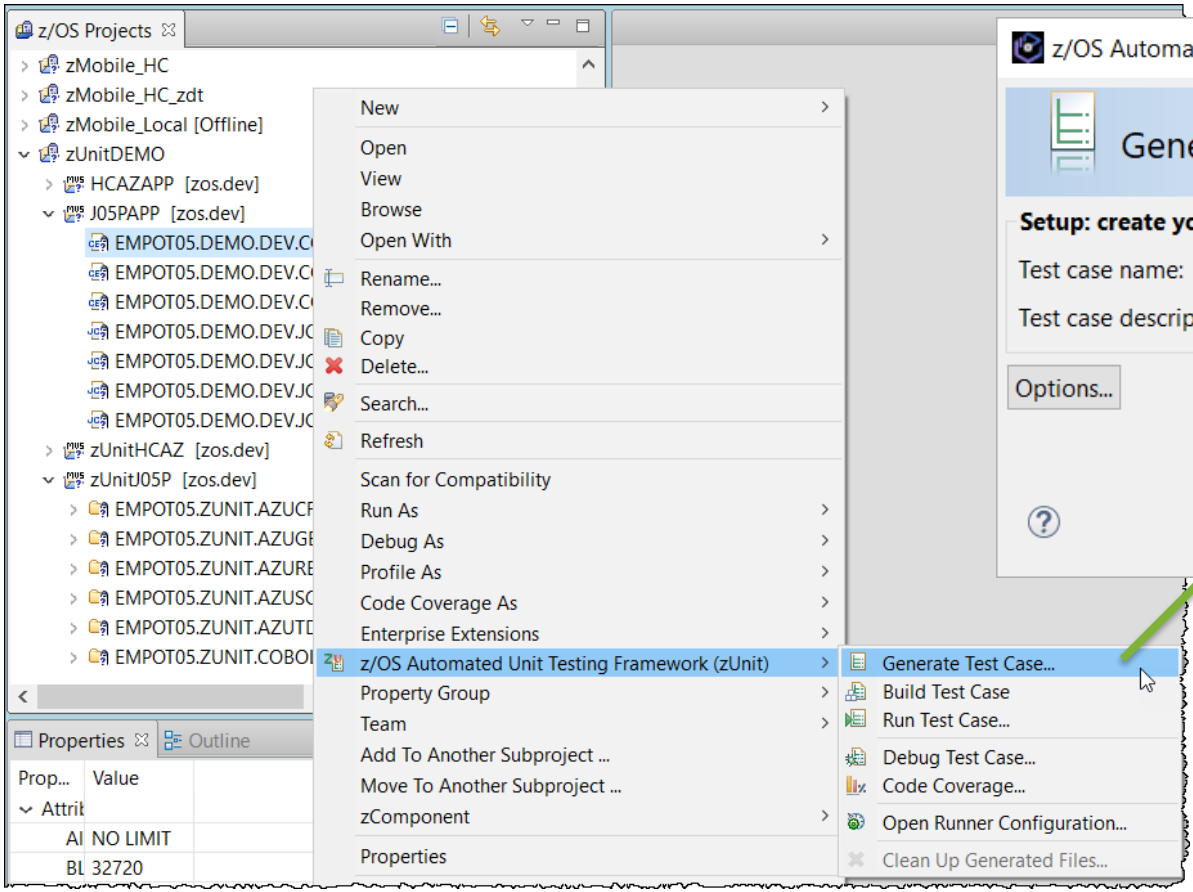
1. Record CICS program interaction using **IDz** .

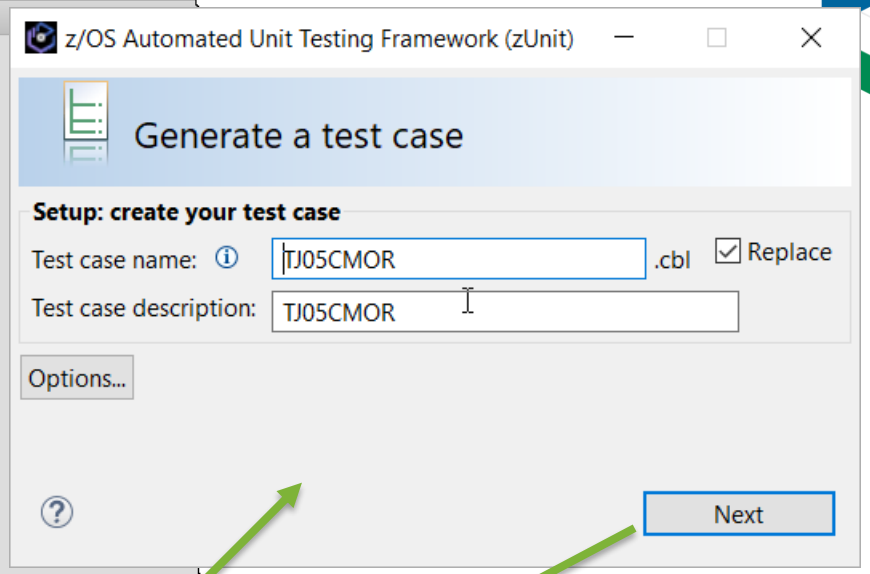
2. Generate, build and run the unit test
   Compile and link-edit the generated unit test program, followed by running the unit test.

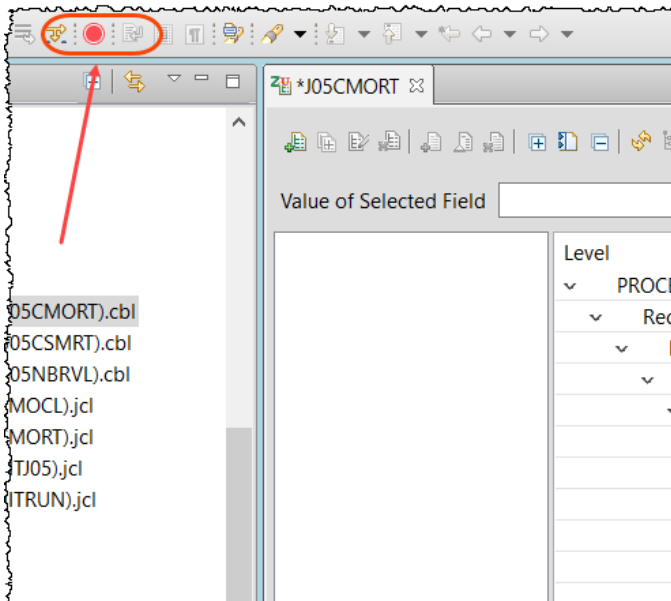
3. Modify the program and rerun the unit test
   Modify the program under test, rerun the unit test, and observe the failure of the test case.

4. Uses Jenkins to perform the Build, ZUnit and Deploy
   The Z unit test will run from a Z/OS shell script and deploy only if does not fail.

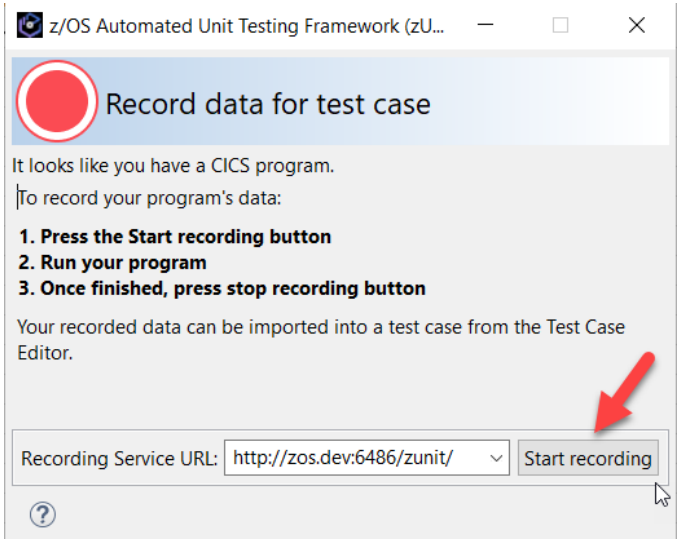
1. Record CICS program interaction using IDz …

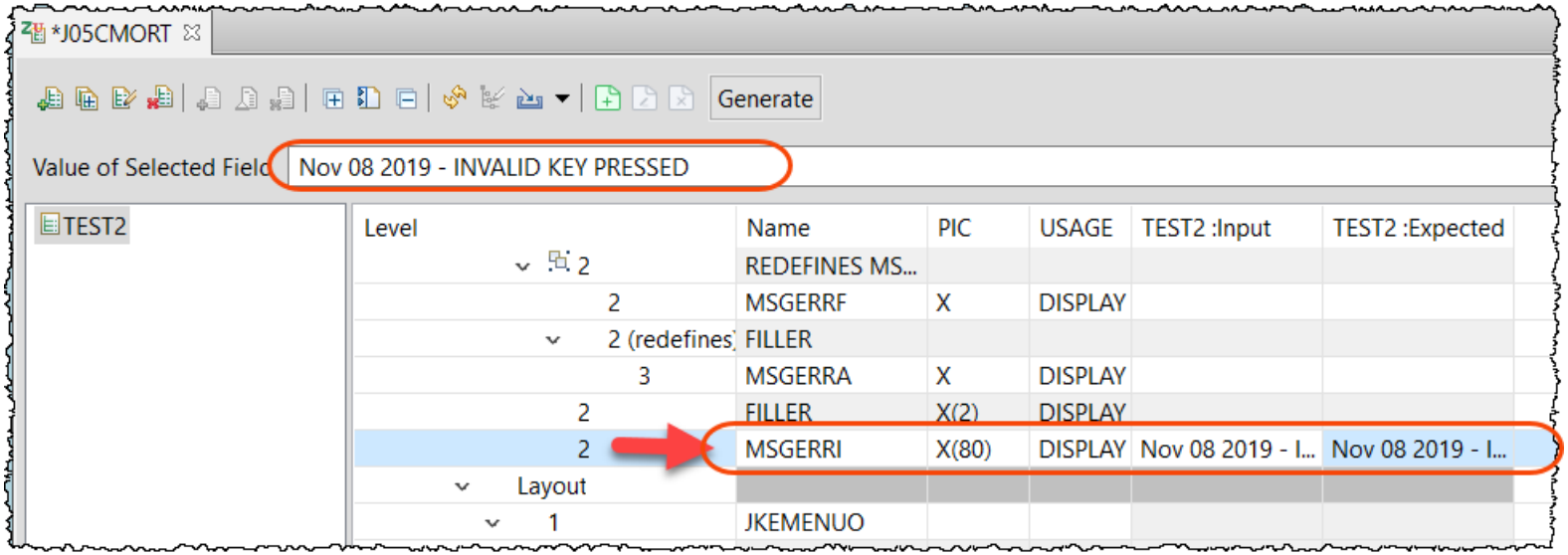






1. Record CICS program interaction using IDz

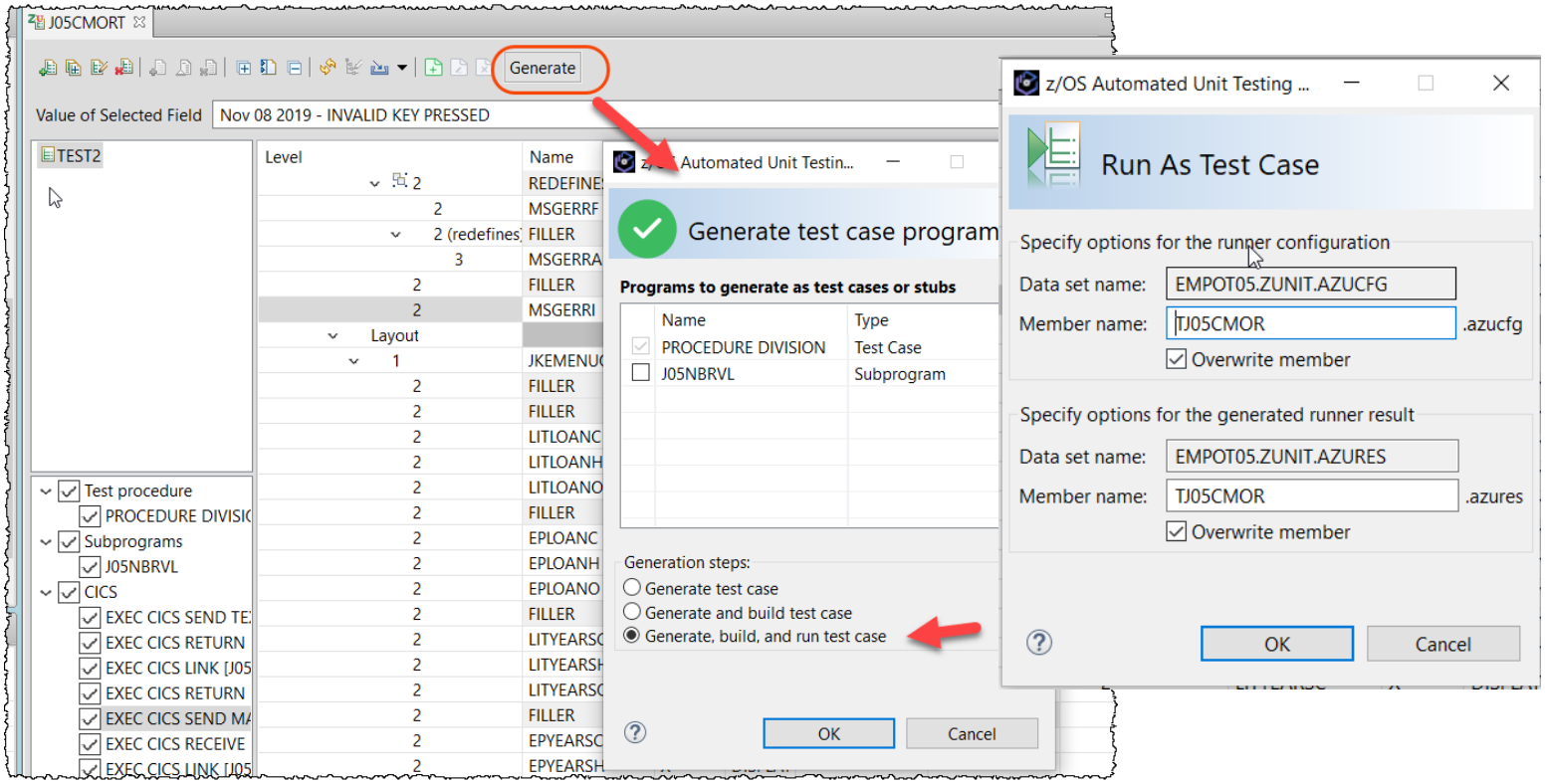


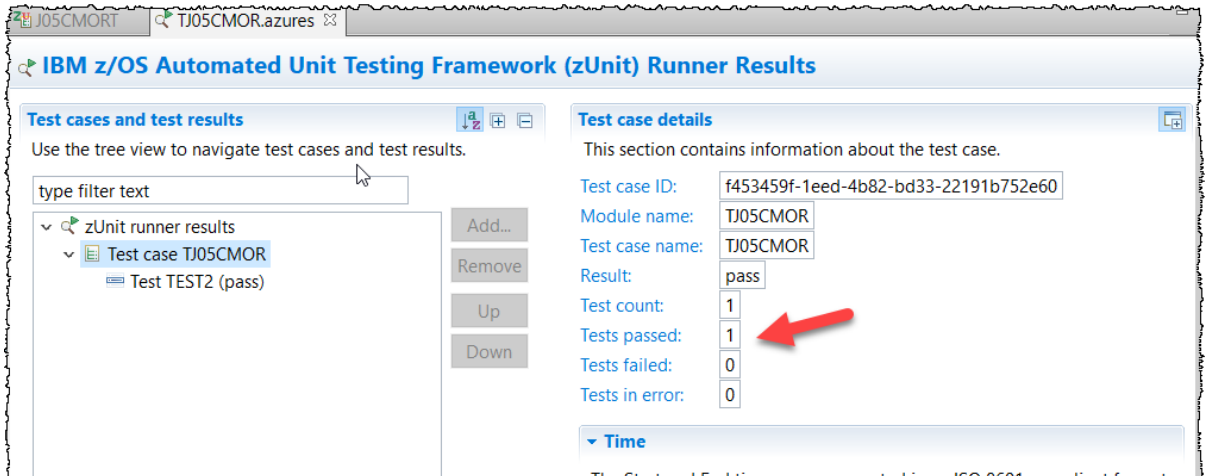


Unit test on COBOL/CICS program `J05CMORT`

1. Record CICS program interaction using **IDz** .
2. Generate, build and run the unit test
   Compile and link-edit the generated unit test program, followed by running the unit test.
3. Modify the program and rerun the unit test
   Modify the program under test, rerun the unit test, and observe the failure of the test case.
4. Uses Jenkins to perform the Build, ZUnit and Deploy
   The Z unit test will run from a Z/OS shell script and deploy only if does not fail.

2. Generate, build and run the unit test





Unit test on COBOL/CICS program J05CMORT

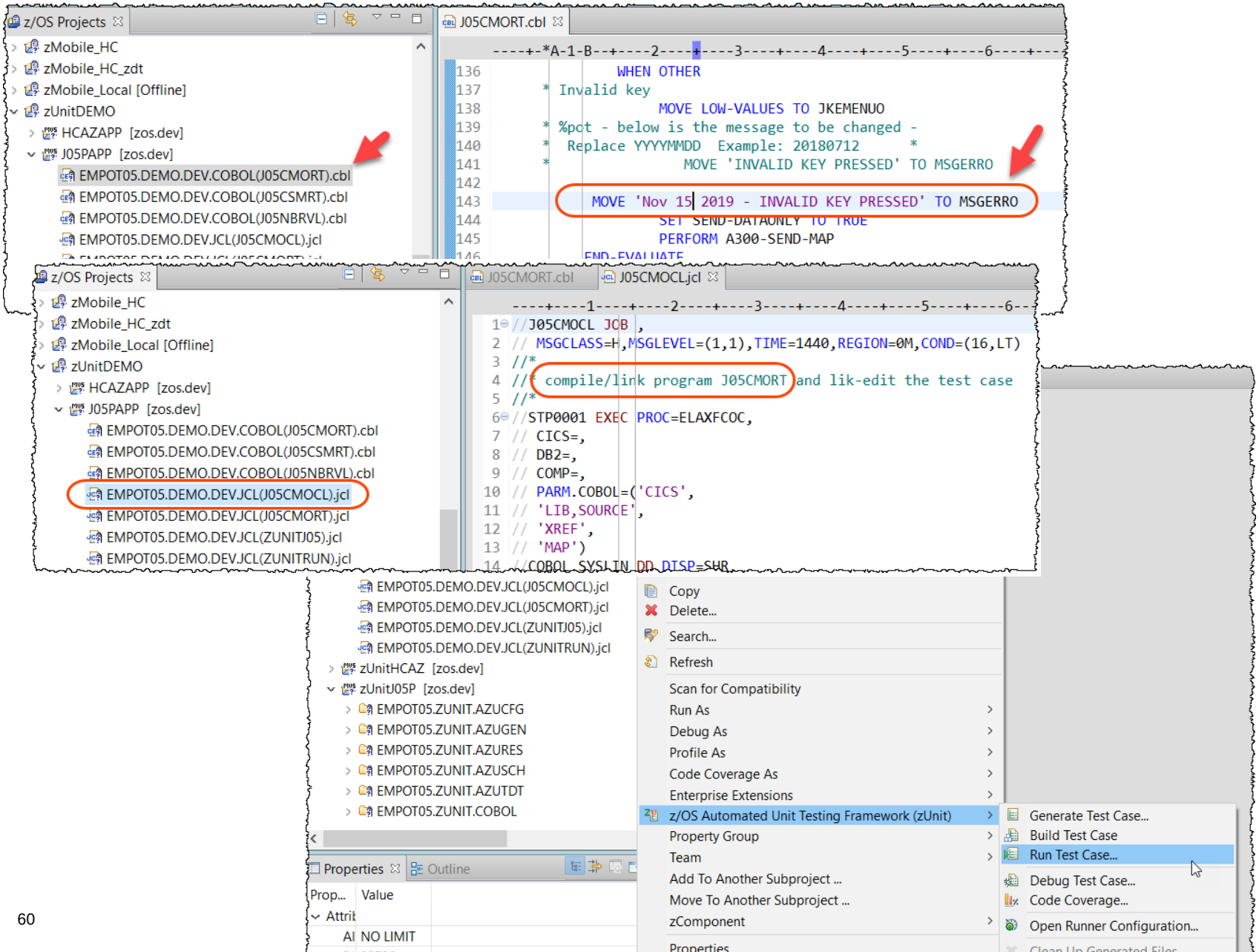
1. Record CICS program interaction using **IDz** .

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

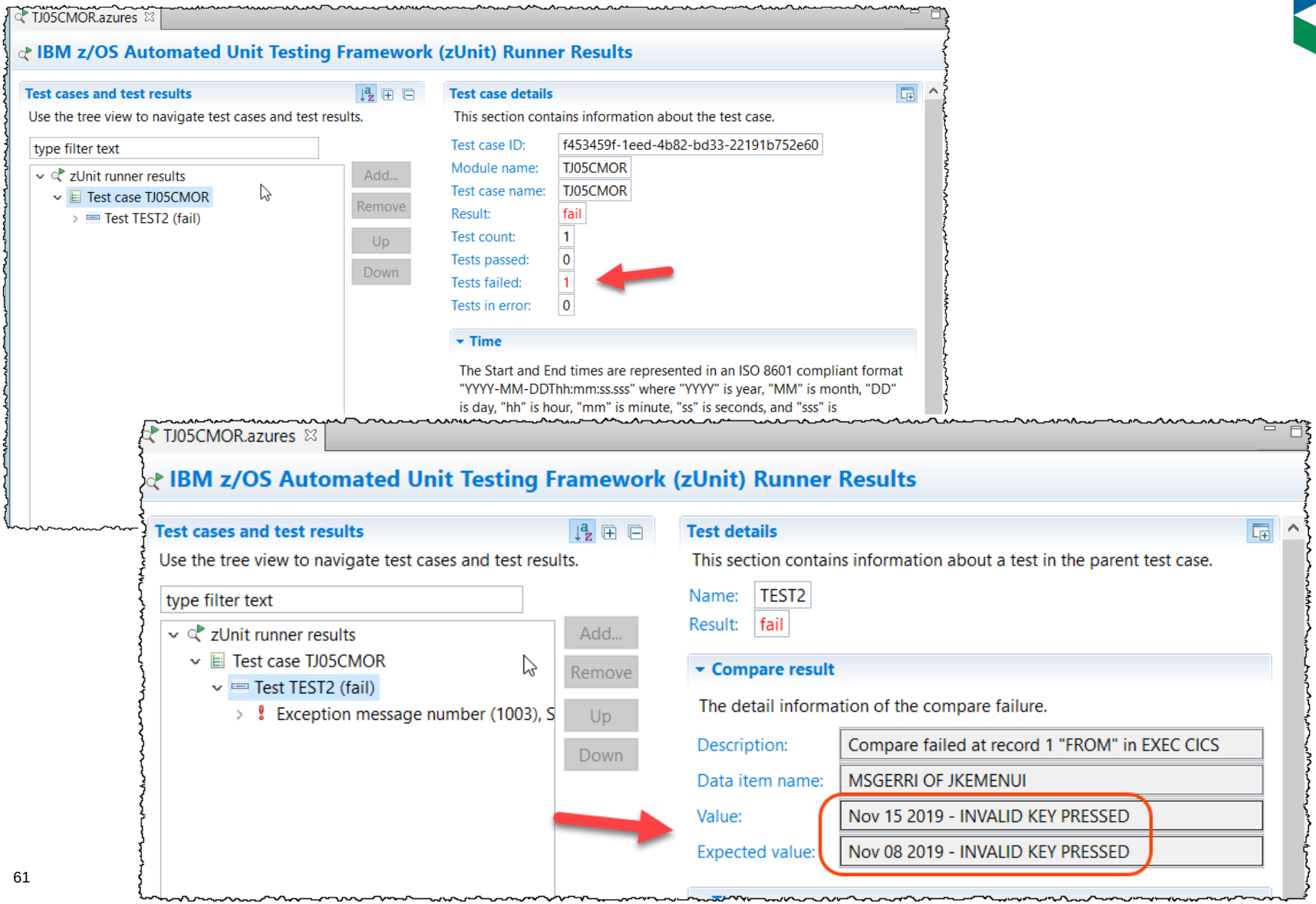
3. Modify the program and rerun the unit test
   Modify the program under test, rerun the unit test, and observe the failure of the test case.

4. Uses Jenkins to perform the Build, ZUnit and Deploy
   The Z unit test will run from a Z/OS shell script and deploy only if does not fail.

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`

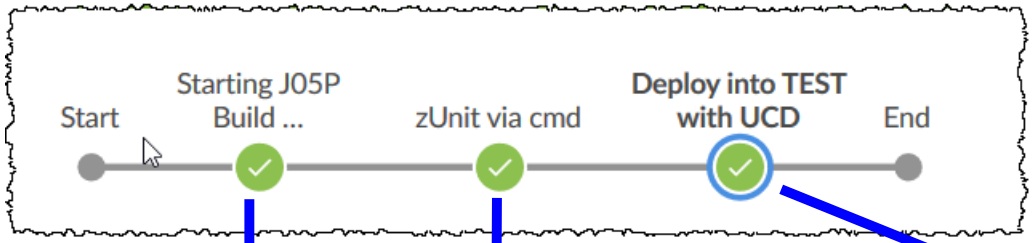
1. Record CICS program interaction using **IDz** .

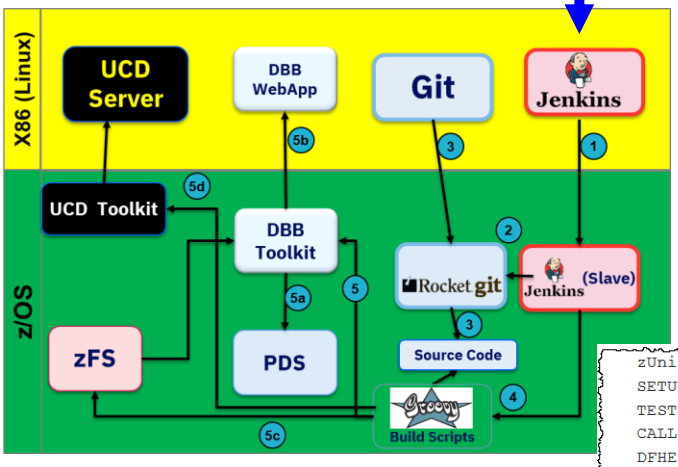
2. Generate, build and run the unit test
   Compile and link-edit the generated unit test program, followed by running the unit test.

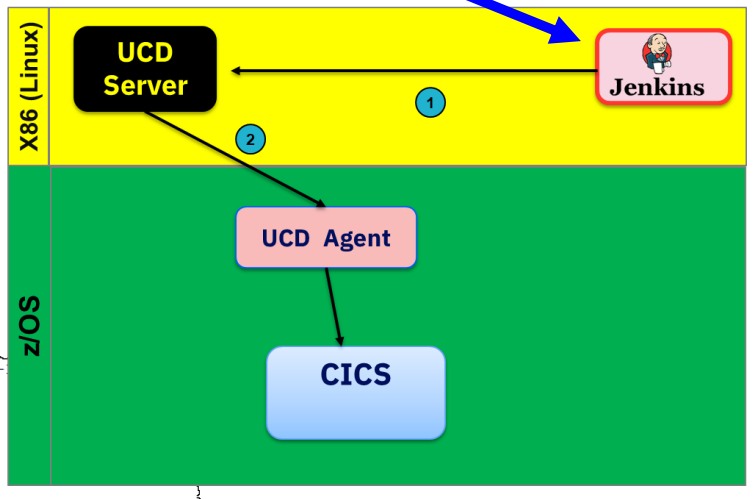
3. Modify the program and rerun the unit test
   Modify the program under test, rerun the unit test, and observe the failure of the test case.

4. Uses Jenkins to perform the Build, ZUnit and Deploy
   - Developer push changes to Git
   - The Z unit test will run from a Z/OS shell script and deploy only if it does not fail.

High Level Diagram using Jenkins/Git/Zunit/UCD








```
zUnit Test Runner 2.0.0.1 started at 2019-11-
SETUP (TEST2)
TEST2 Started...
CALL J05CMORT
DFHEI1 Started...
EXEC CICS SEND MAP "JKEMENU" L=00425
DFHEI1 Successful.
DFHEI1 Started...
EXEC CICS RETURN X'E000' L=00375
AZUCEEUT::getMessage(): CEEMGET.fc.tok_msgno=454.

TEARDOWN (TEST2)
 o Test count: 1
 o Tests passed: 1
 o Tests failed: 0
 o Tests in error: 0
zUnit Test Runner 2.0.0.1 ended at 2019-11-19T11:29:01.932.
Finished: SUCCESS
```

Agenda

9:00 Introductions
9:20 DevOps on z Systems Introduction
9:45 Application Delivery Foundation (ADFz):
- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break

10:30 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 z Open Unit Test: Overview & Demonstration

1:00 Day in the Life - Using Application Discovery, DBB/Git/Jenkins and UCD

2:00 – 4:30 Choose one Optional lab:
- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (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

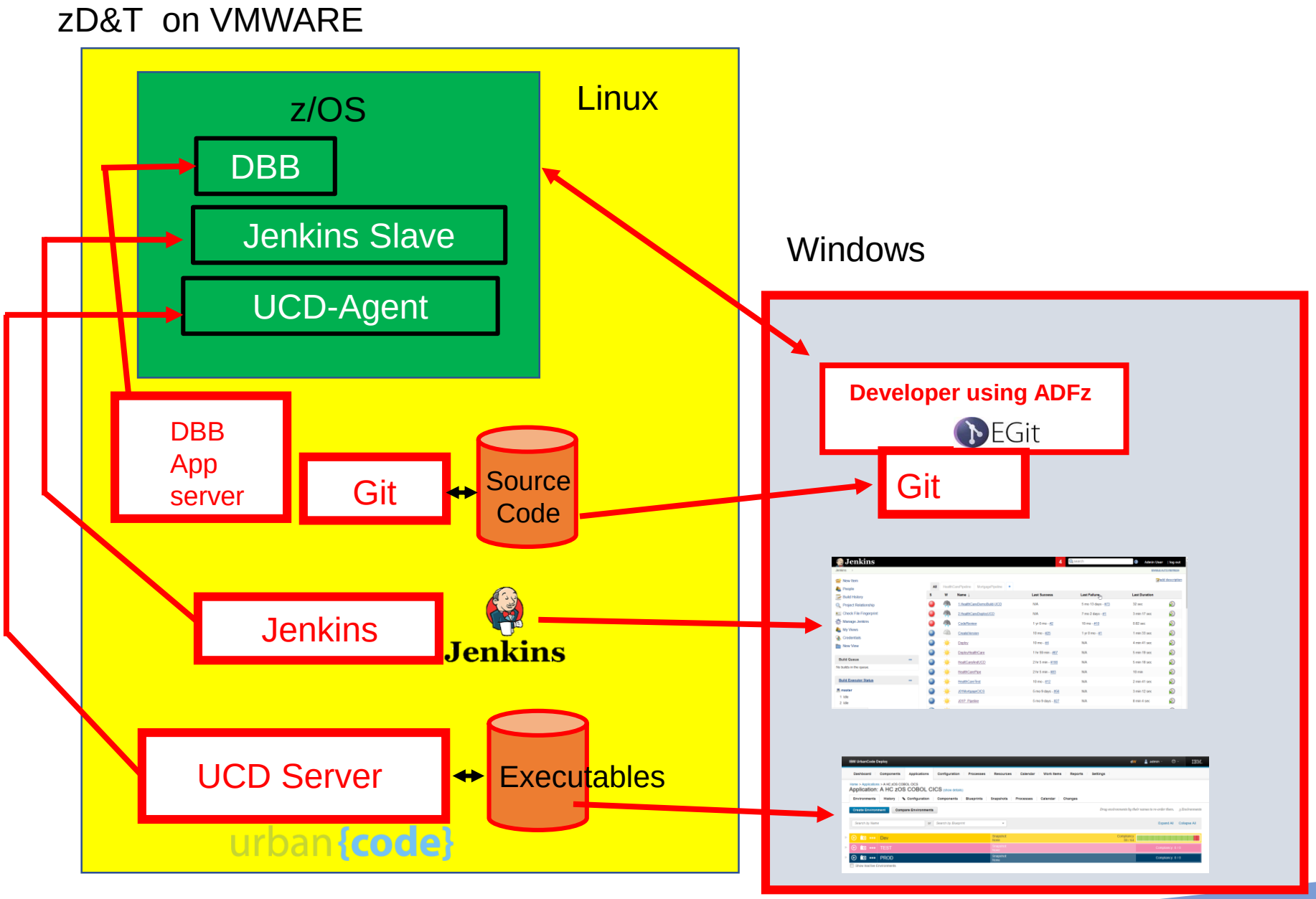
Demonstration

zD&T: AD - ADFz – DBB – UCD

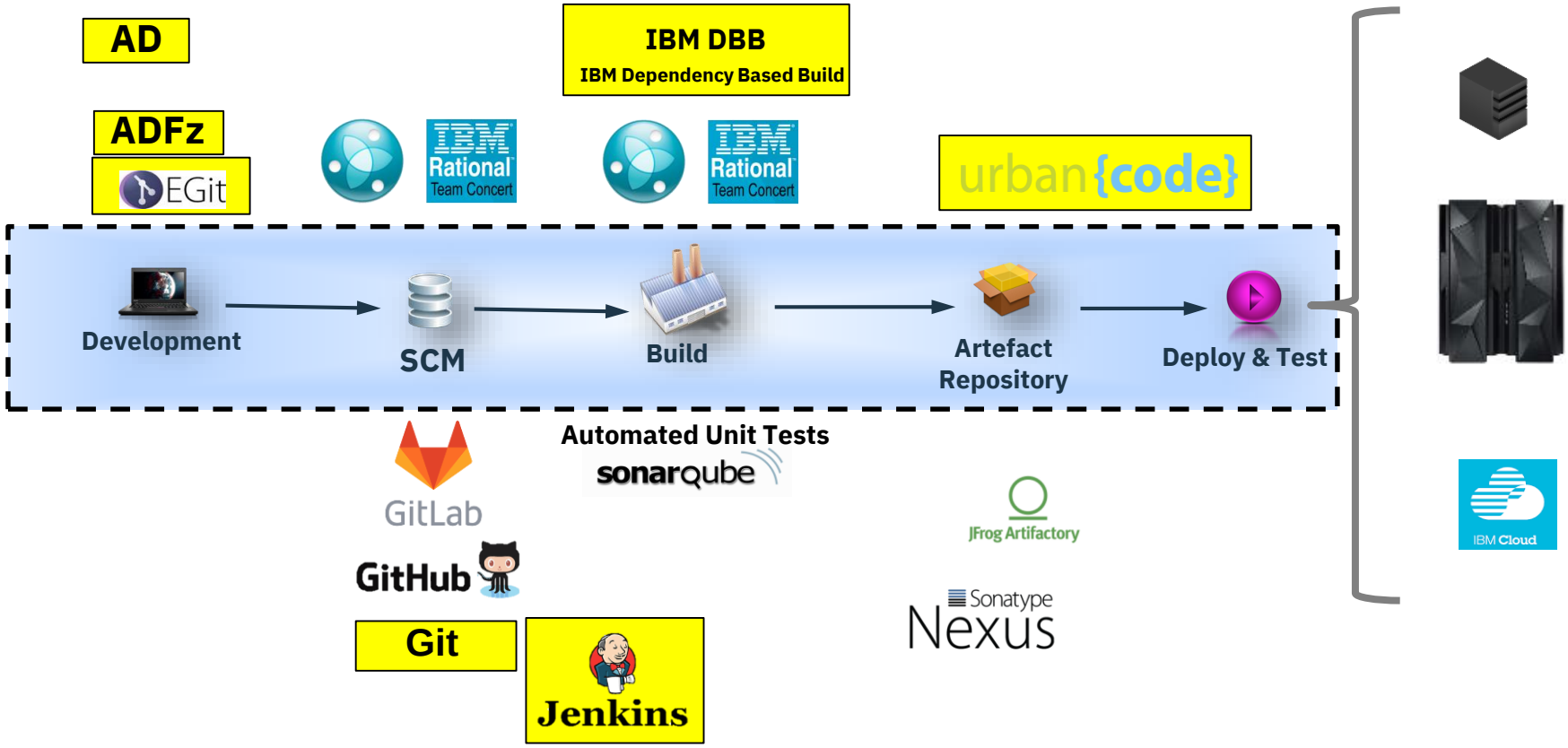
Using Git and Jenkins

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

Agenda

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

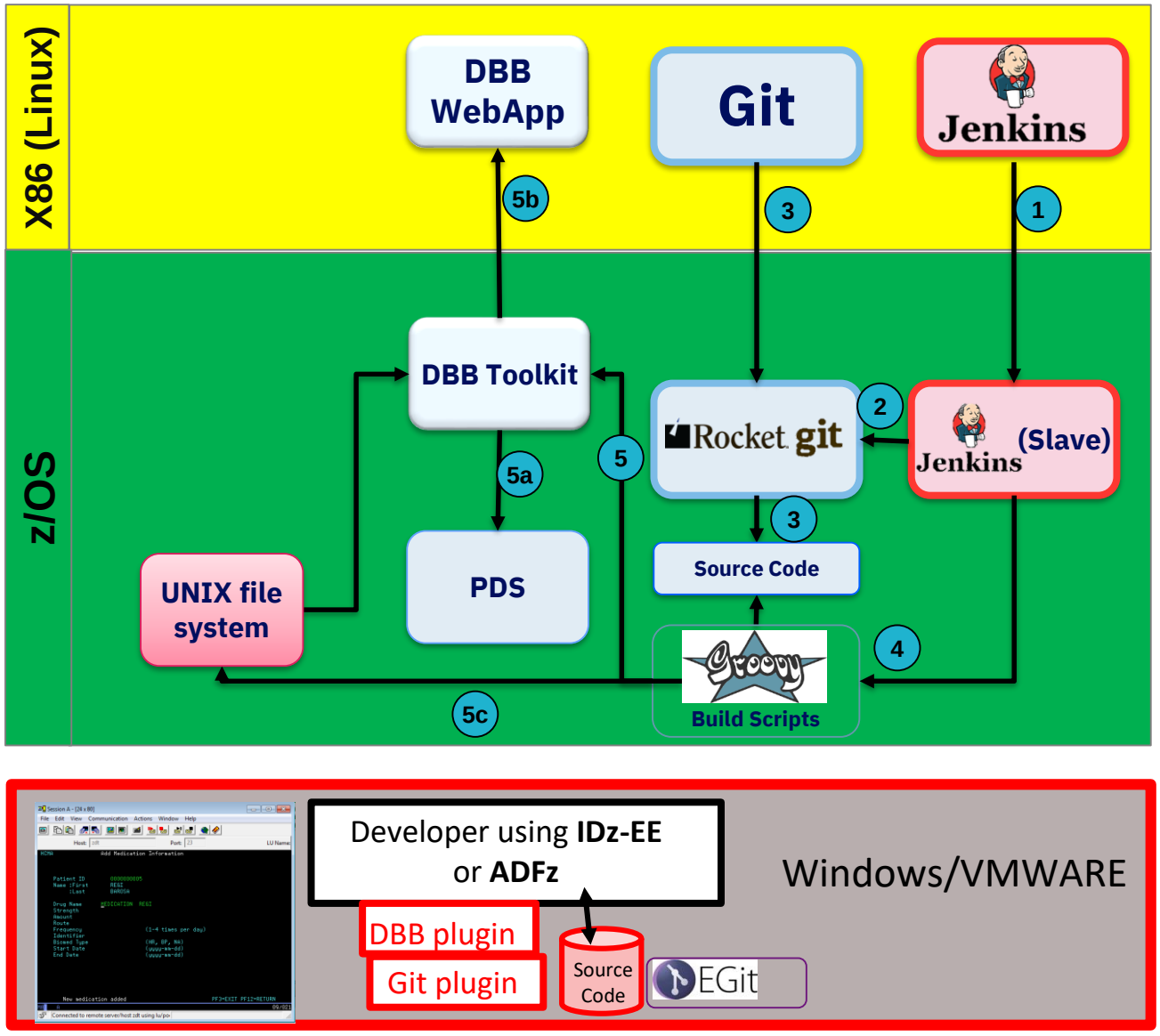
```
import com.ibm.dbb.build.*

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
```


Agenda

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).

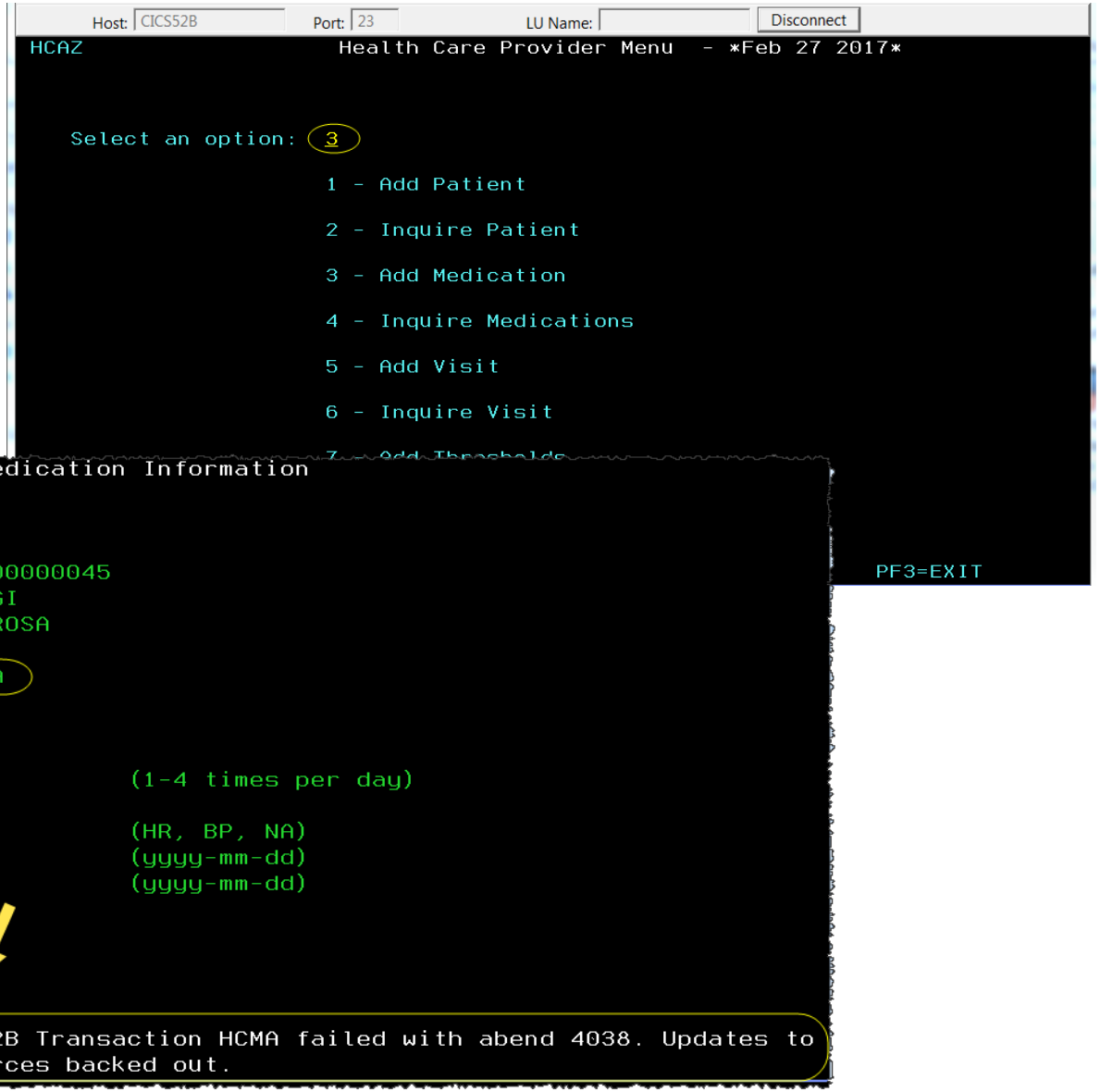
Agenda

1. • What is *IBM Dependency Based Build* (**DBB**)?
2. • General architecture of the solution
4. • Git/DBB/Jenkins/UCD in action (demo)

Scenario: CICS Transaction ABEND

Using transaction HCAZ and option 3

HCMA Transaction has an ***abend..***



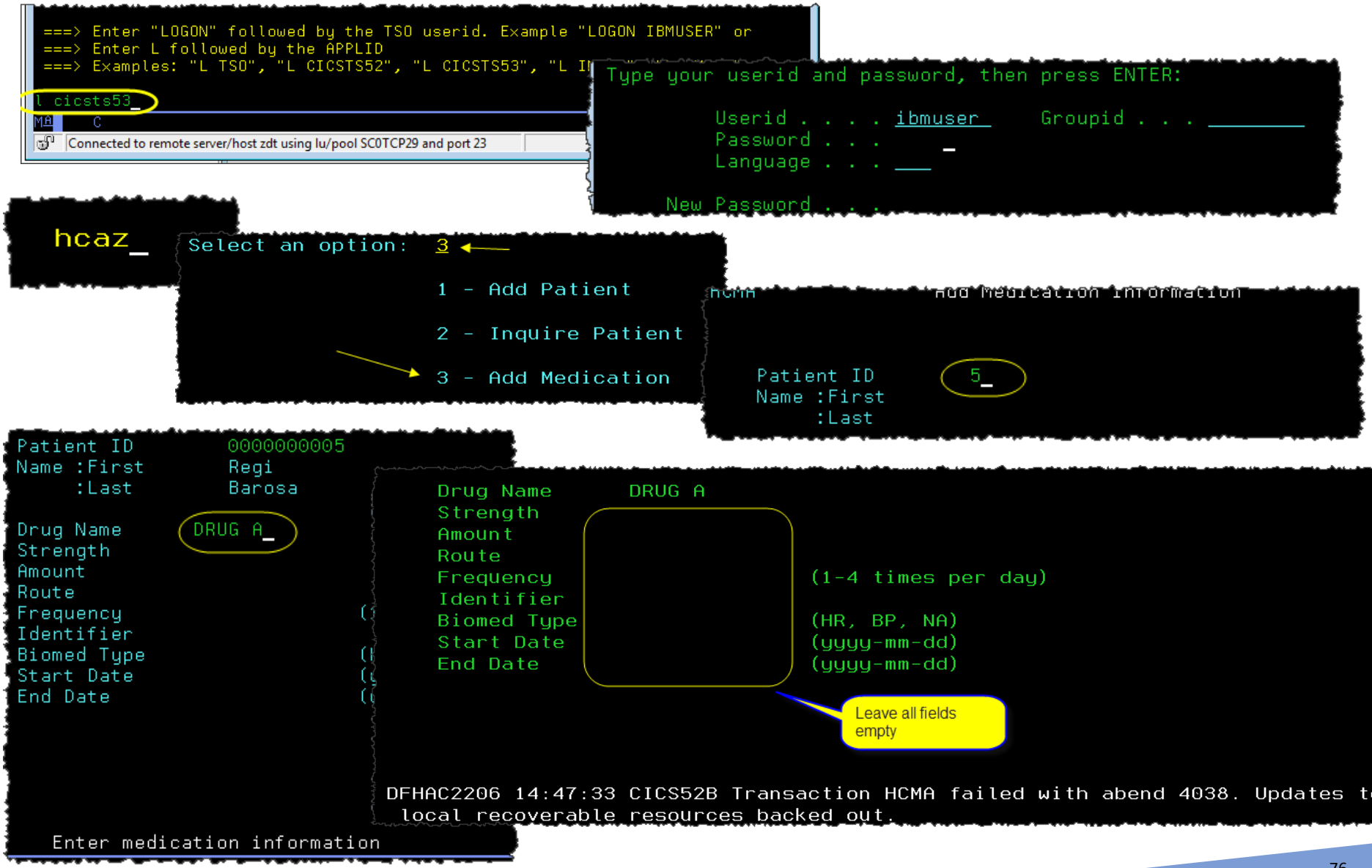
PART #1 - Verify the CICS Abend and understand the Application Components

Objective: Fix the CICS Abend

Alex - Project Manager
Performs application analysis

1. Executes CICS transaction **HCAZ** to verify the abend.
2. Uses **Application Discovery (AD)** to understand the application.

Alex starts CICS transaction HCAZ to verify the abend



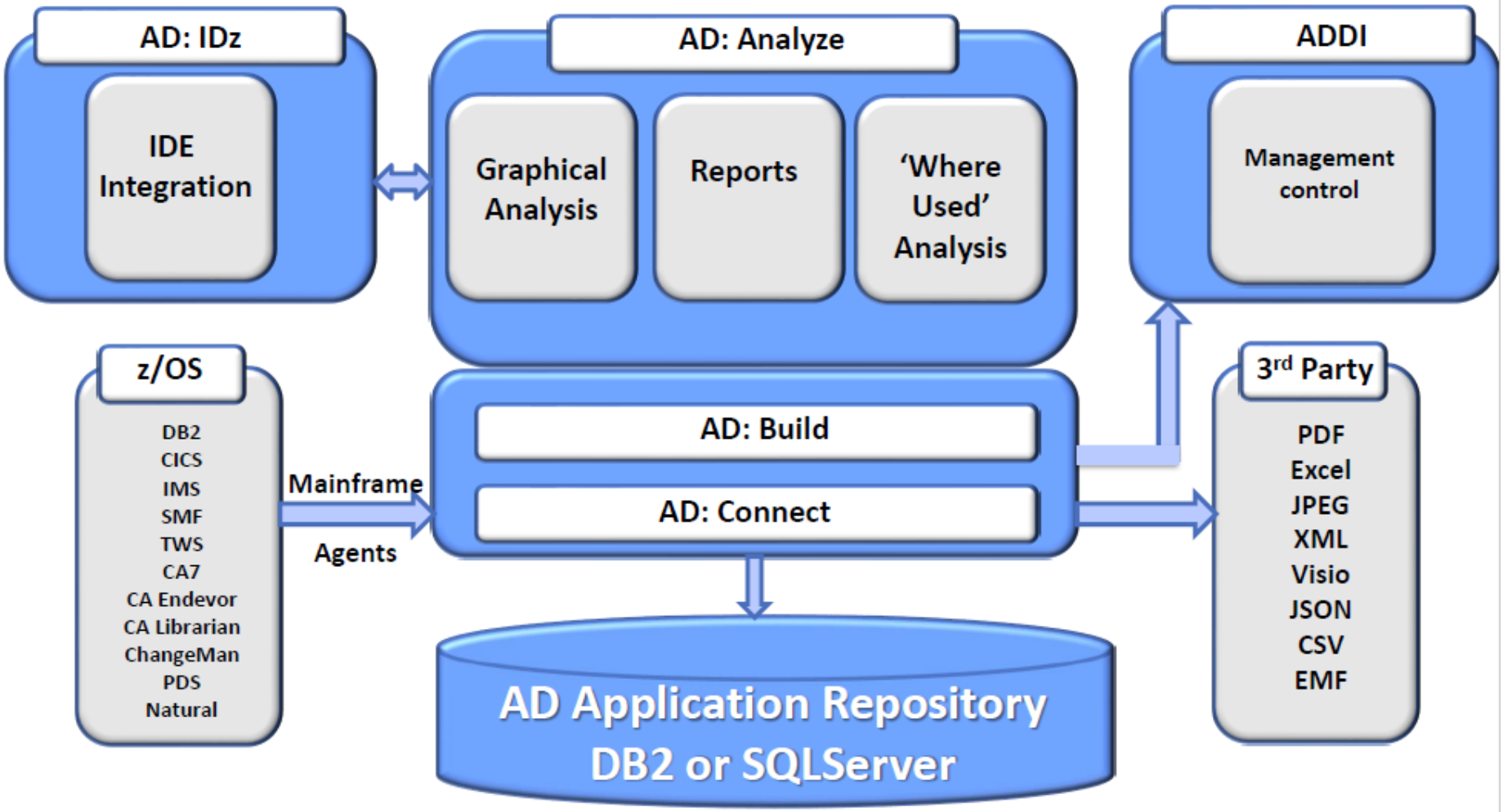
PART #1 - Verify the CICS Abend and understand the Application Components

Objective: Fix the CICS Abend

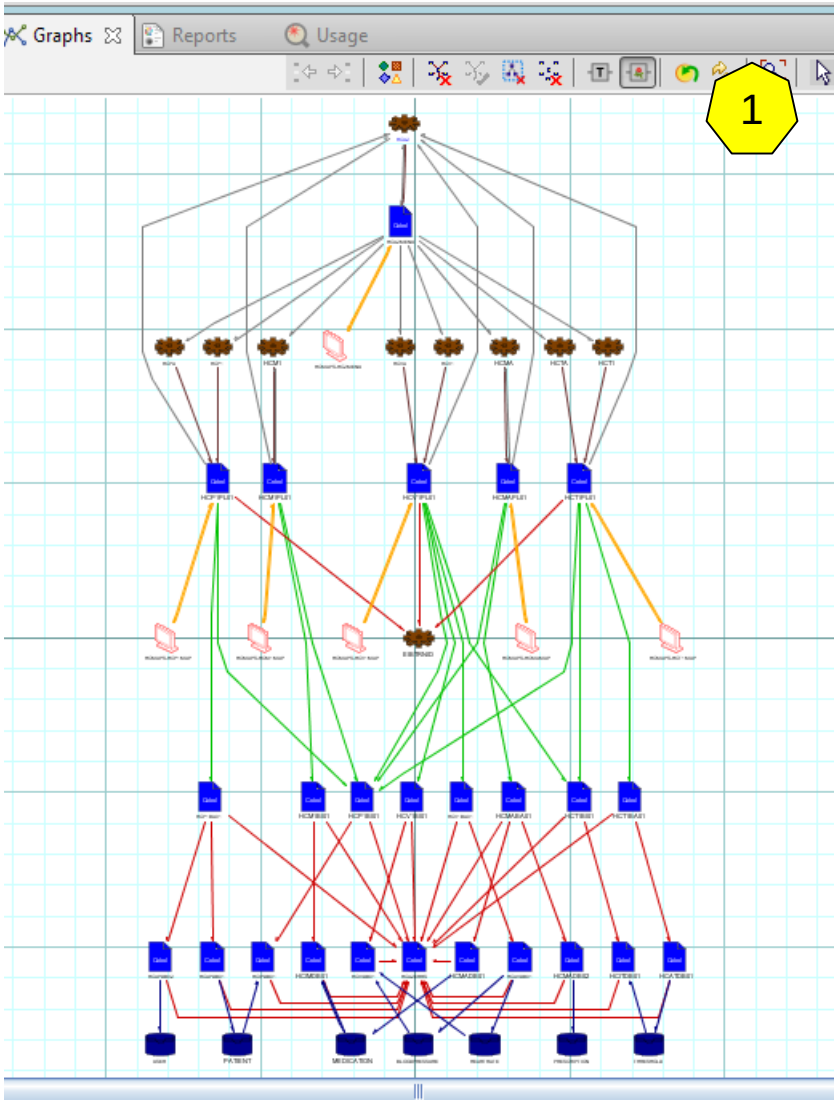
Alex - Project Manager
Performs application analysis

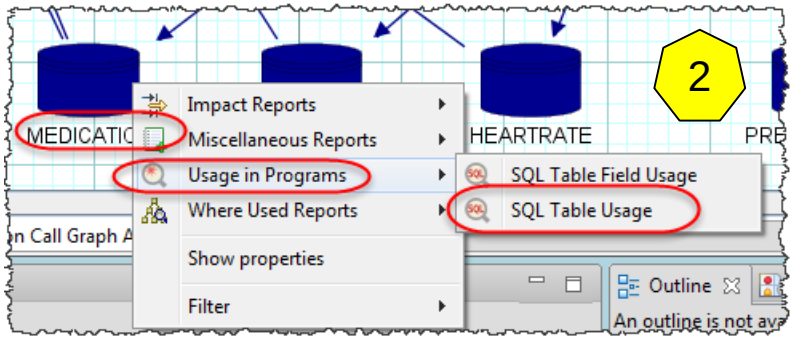
1. Executes CICS transaction **HCAZ** to verify the abend.
2. Uses **Application Discovery (AD)** to understand the application.

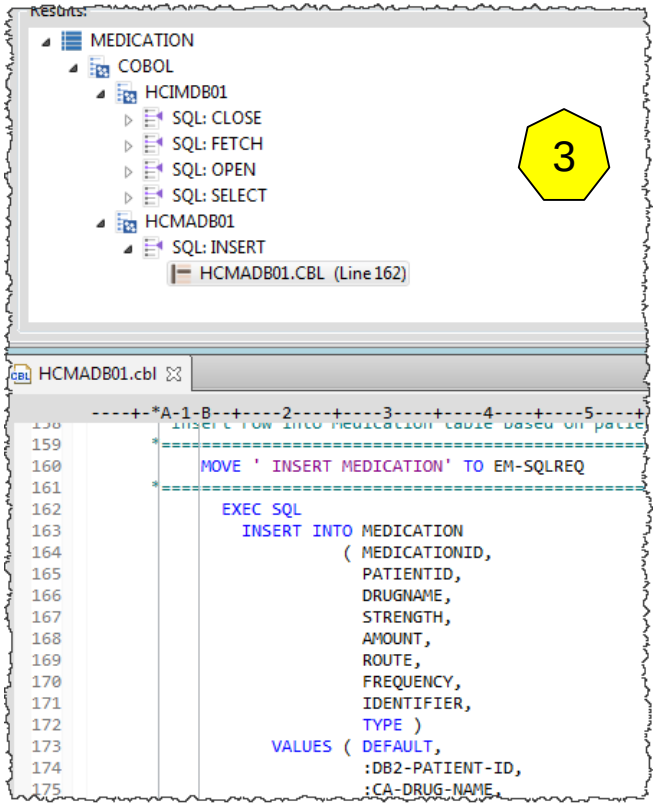
IBM Application Discovery – Big Picture



Uses **Application Discovery (AD)** to understand the application







PART #2 - Developer see the CICS abend cause and fix it.

Bob – Mainframe Developer
Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

1. Uses **CICS** to verify the issue.
2. Uses **ADFz/Fault Analyzer** to identify what is causing the abend
3. Uses **ADFz/Debug** to debug the program and verify the error
4. Uses **ADFz** to update the COBOL program loaded from **GIT**
5. Uses **DBB** to perform a User Dependency Build and verify that the abend is fixed
6. Uses **GIT** to **Commit and Push** the corrected code to Repository

Uses **ADFz/Fault Analyzer** to identify what is causing the abend

ZDT:2800/IDID10.HIST(F00039)-Report

```
Module HCMADB02, program HCMADB02, source line # 300: CICS abend 4038
 I B M   F A U L T   A N A L Y Z E R   S Y N O P S I S

A CICS abend 4038 occurred in module CEEPLPKA at offset X'CB5F8'.

The cause of the failure was program HCMADB02 in module HCMADB02.  The COBOL
source code that immediately preceded the failure was:

  Source
  Line #
  ------
  000300          COMPUTE WS-INTEGER-START-DATE =
  000301                  FUNCTION INTEGER-OF-DATE (WS-WORKING
  000302

The COBOL source code for data fields involved in the failure:

  Source
  Line #
  ------
  000070             05  WS-WORKING-DATE          PIC  9(8).
  000089             05  WS-INTEGER-START-DATE   PIC 9(8).

Data field values at time of abend:

  WS-INTEGER-START-DATE = X'D4C5C4C9C3C1E3C9'
  WS-WORKING-DATE       = X'404040F040F040F0'

Important messages:
```

Main Report | Event Details | Abend Information | System-Wide Information | Miscellaneous

ZDT : 2800/IDID10.HIST | Lookup | Markers

FAULT_ID	JOB/TRAN	USER_ID	SYS/JOB
F00039	HCMA	IBMUSER	CICSTS53

ZDT:2800/IDID10.HIST(F00039)-Report | HCMADB02.COB

```
291        * =%regi ========== Below must be comments to have the abend
292        * =    when DEMO is day 1 must change to Compute = 20170901
293        *     IF WS-WORKING-DATE < 16010101 or
294        *         WS-WORKING-DATE > 99991231
295        *         MOVE FUNCTION CURRENT-DATE (1:8) TO WS-WORKING-DATE
296        *         COMPUTE WS-WORKING-DATE = WS-WORKING-DATE - 1
297        *      END-IF
298        * ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
299
300           COMPUTE WS-INTEGER-START-DATE =
301                   FUNCTION INTEGER-OF-DATE (WS-WORKING-DATE)
302
303           MOVE WS-END-NUM-DATE TO  WS-WORKING-DATE
304        * =%regi ========= Below must be comments to have the abend
305        * =added to fix abend #2 END DATE
306        *      IF WS-WORKING-DATE < 16010101 or
307        *         WS-WORKING-DATE > 99991231
308        *         MOVE FUNCTION CURRENT-DATE (1:8) TO WS-WORKING-DATE
309        *      END-IF
310        * ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
311
312           COMPUTE WS-INTEGER-END-DATE =
313                   FUNCTION INTEGER-OF-DATE (WS-WORKING-DATE)
314
315           PERFORM UNTIL WS-INTEGER-START-DATE > WS-INTEGER-END-DATE
316               COMPUTE WS-WORKING-DATE =
317                       FUNCTION DATE-OF-INTEGER (WS-INTEGER-START-DATE)
318               MOVE WS-WORKING-DATE TO WS-START-NUM-DATE
319               EVALUATE CA-FREQUENCY
```

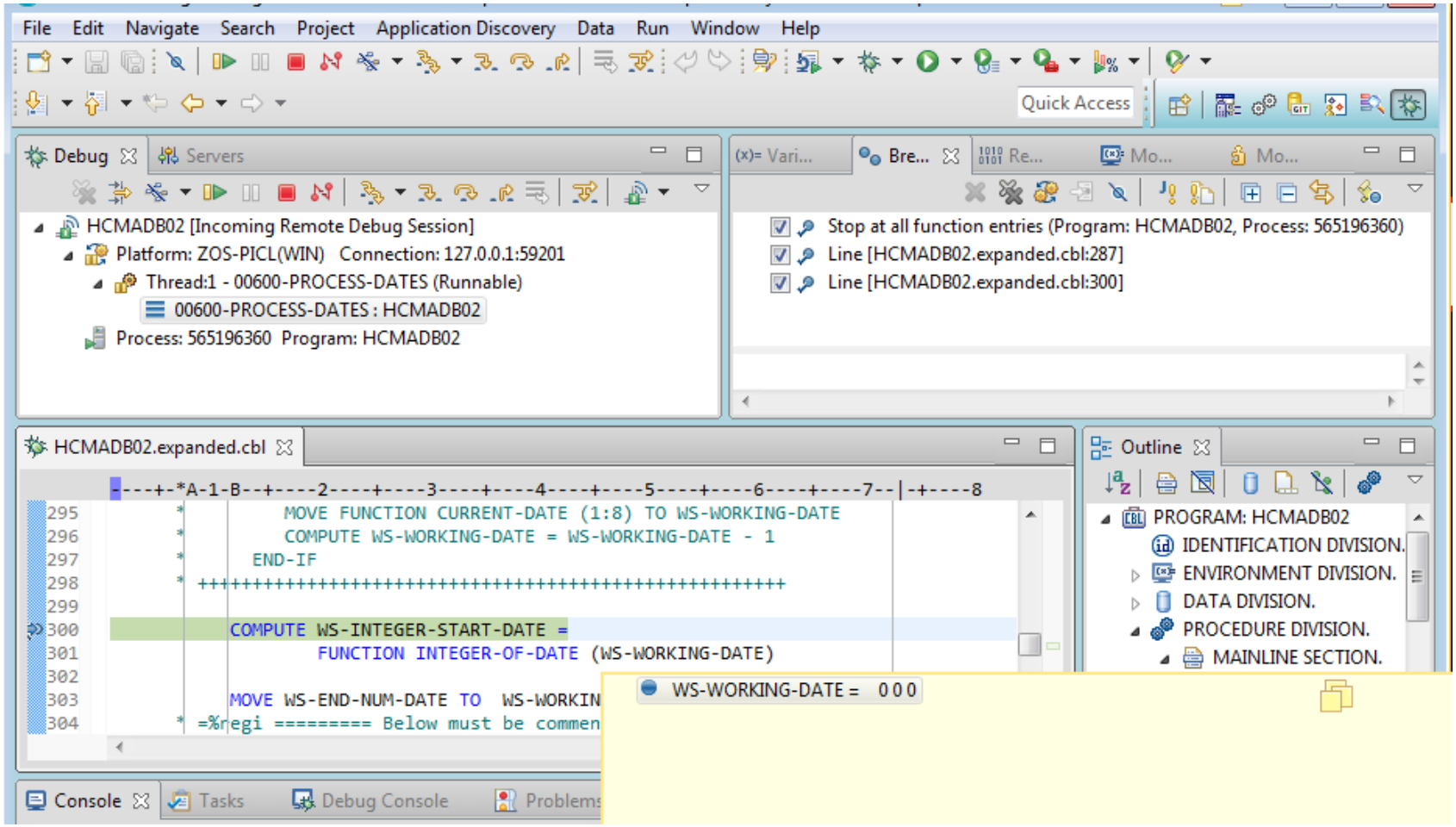
PART #2 - Developer see the CICS abend cause and fix it.

Bob – Mainframe Developer
Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

1. Uses **CICS** to verify the issue.
2. Uses **ADFz/Fault Analyzer** to identify what is causing the abend
3. Uses **ADFz/Debug** to debug the program and verify the error
4. Uses **ADFz** to update the COBOL program and Debug the code loaded from **GIT**
5. Uses **DBB** to perform a User Dependency Build and verify that the abend is fixed
6. Uses **GIT** to **Commit and Push** the corrected code to Repository

Uses **ADFz/Debug** to debug the program and verify the error



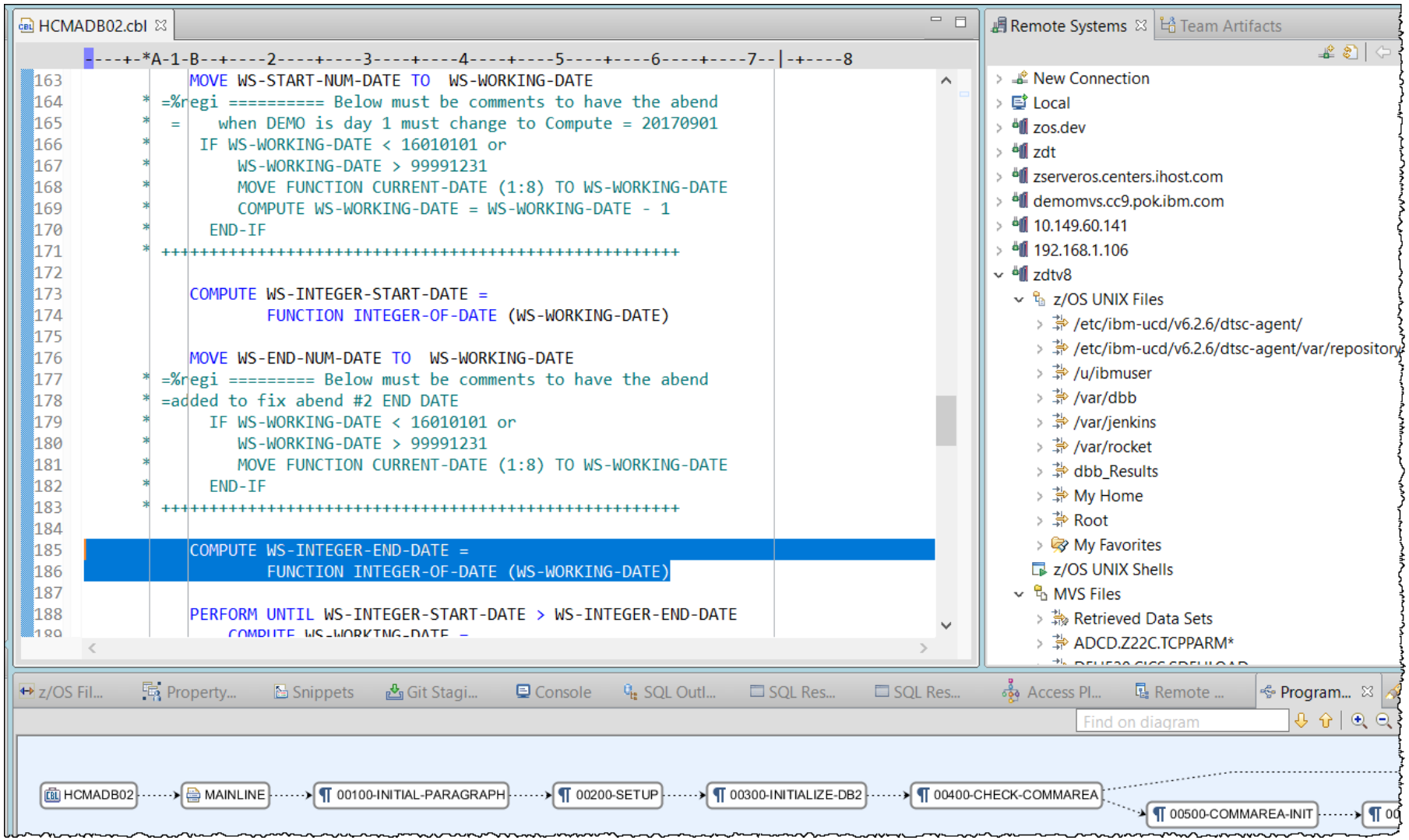
PART #2 - Developer see the CICS abend cause and fix it.

Bob – Mainframe Developer
Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

1. Uses **CICS** to verify the issue.
2. Uses **ADFz/Fault Analyzer** to identify what is causing the abend
3. Uses **ADFz/Debug** to debug the program and verify the error
4. Uses **ADFz** to update the COBOL program loaded from **GIT**
5. Uses **DBB** to perform a User Dependency Build and verify that the abend is fixed
6. Uses **GIT** to **Commit and Push** the corrected code to Repository

Uses **ADFz** to update the COBOL program loaded from GIT



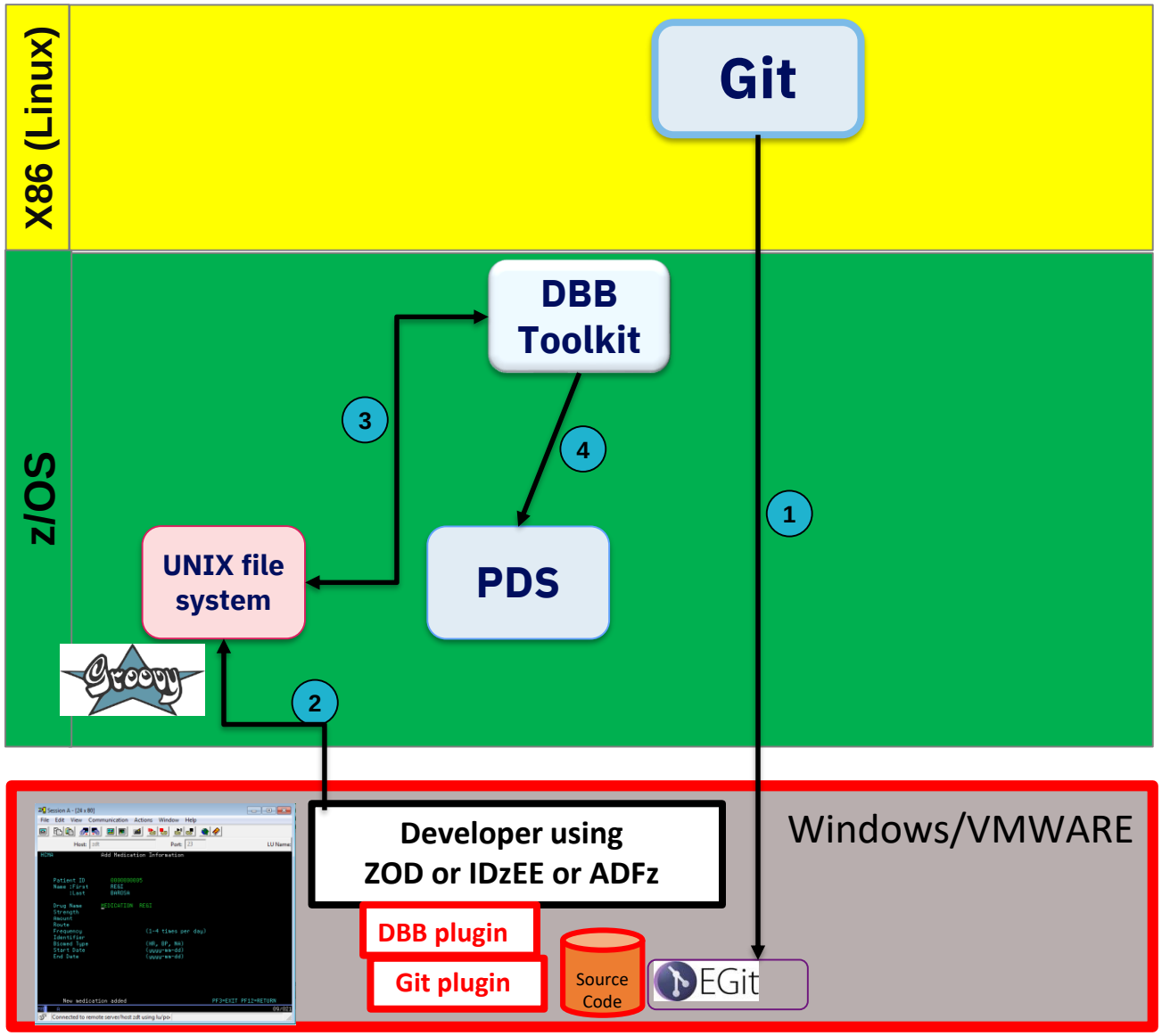
PART #2 - Developer see the CICS abend cause and fix it.

Bob – Mainframe Developer
Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

1. Uses **CICS** to verify the issue.
2. Uses **ADFz/Fault Analyzer** to identify what is causing the abend
3. Uses **ADFz/Debug** to debug the program and verify the error
4. Uses **ADFz** to update the COBOL program loaded from **GIT**
5. Uses **DBB** to perform a User Dependency Build and verify that the abend is fixed
6. 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.

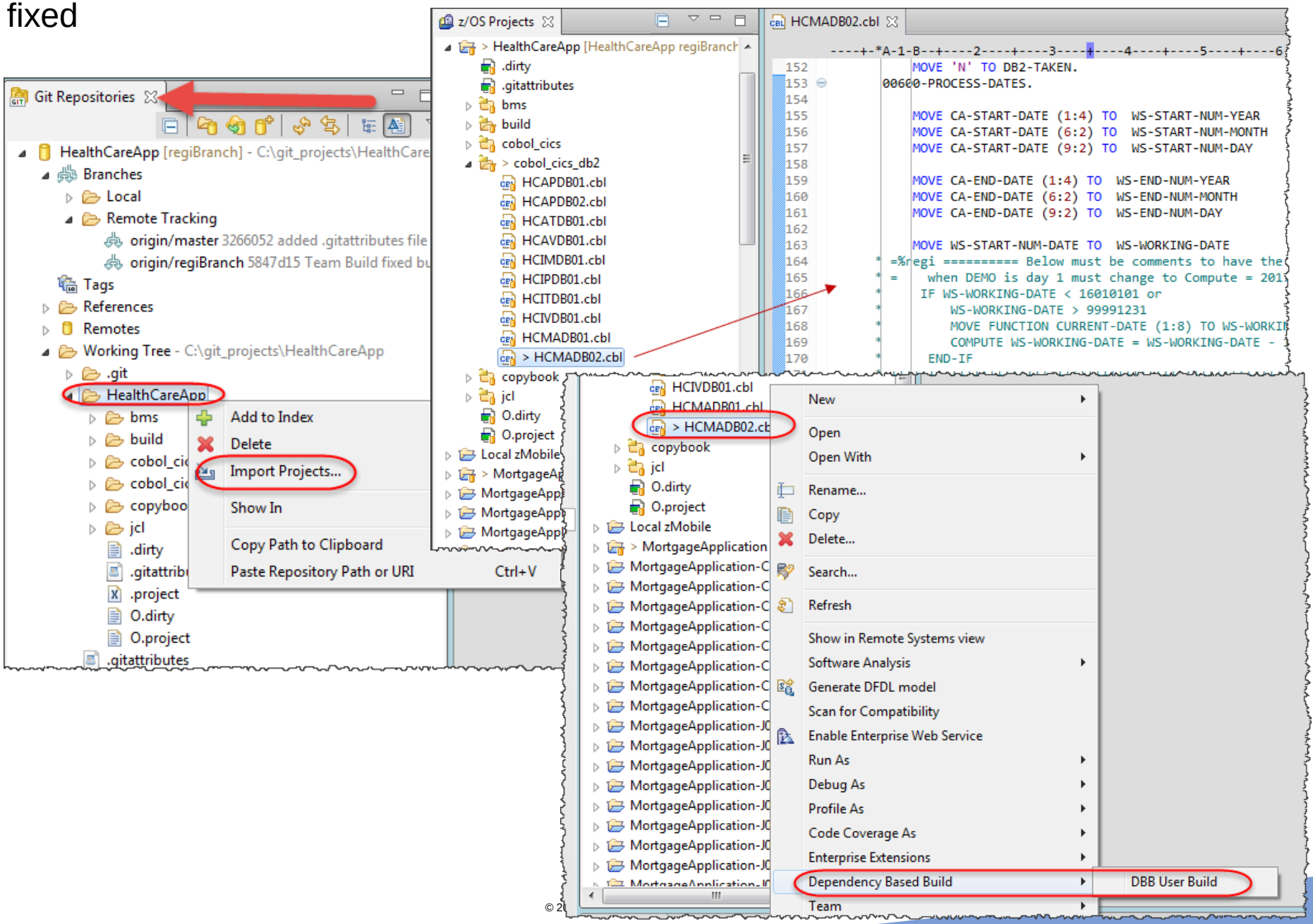
[2] **DBB User Build** moves COBOL source/copybooks from local IDz workspace to **zFS** files, invokes Groovy build scripts containing DBB APIs.

[3] DBB Toolkit provides Java APIs to:

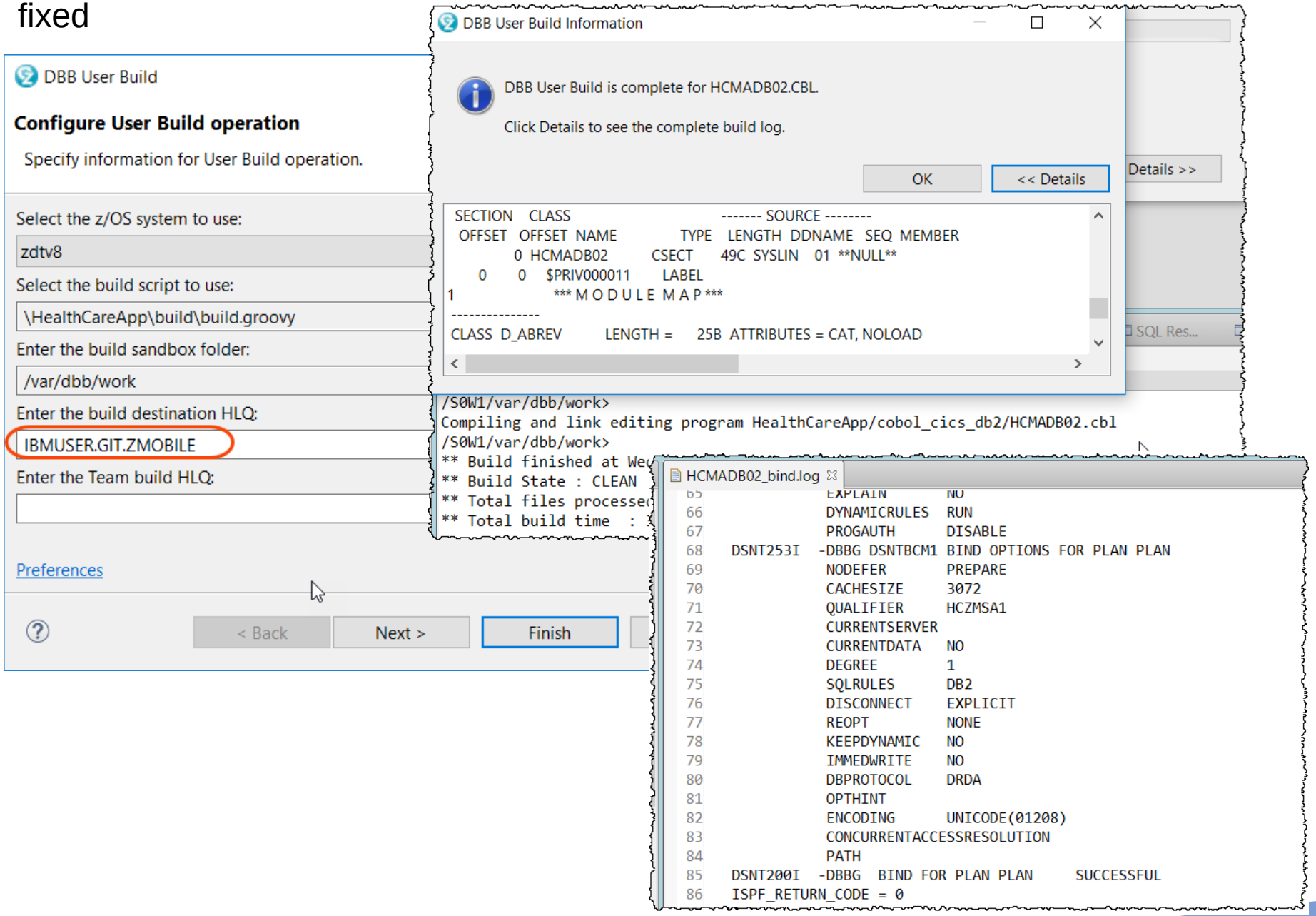
[4] Create datasets, copy source from **zFS** to **PDS**, invoke zOS programs (Compiler/Linkage Editor/ REXX for DB2 Bind).

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

Use **DBB User Build** to perform a **Dependency Build** and verify that the abend is fixed



Use **DBB User Build** to perform a **Dependency Build** and verify that the abend is fixed



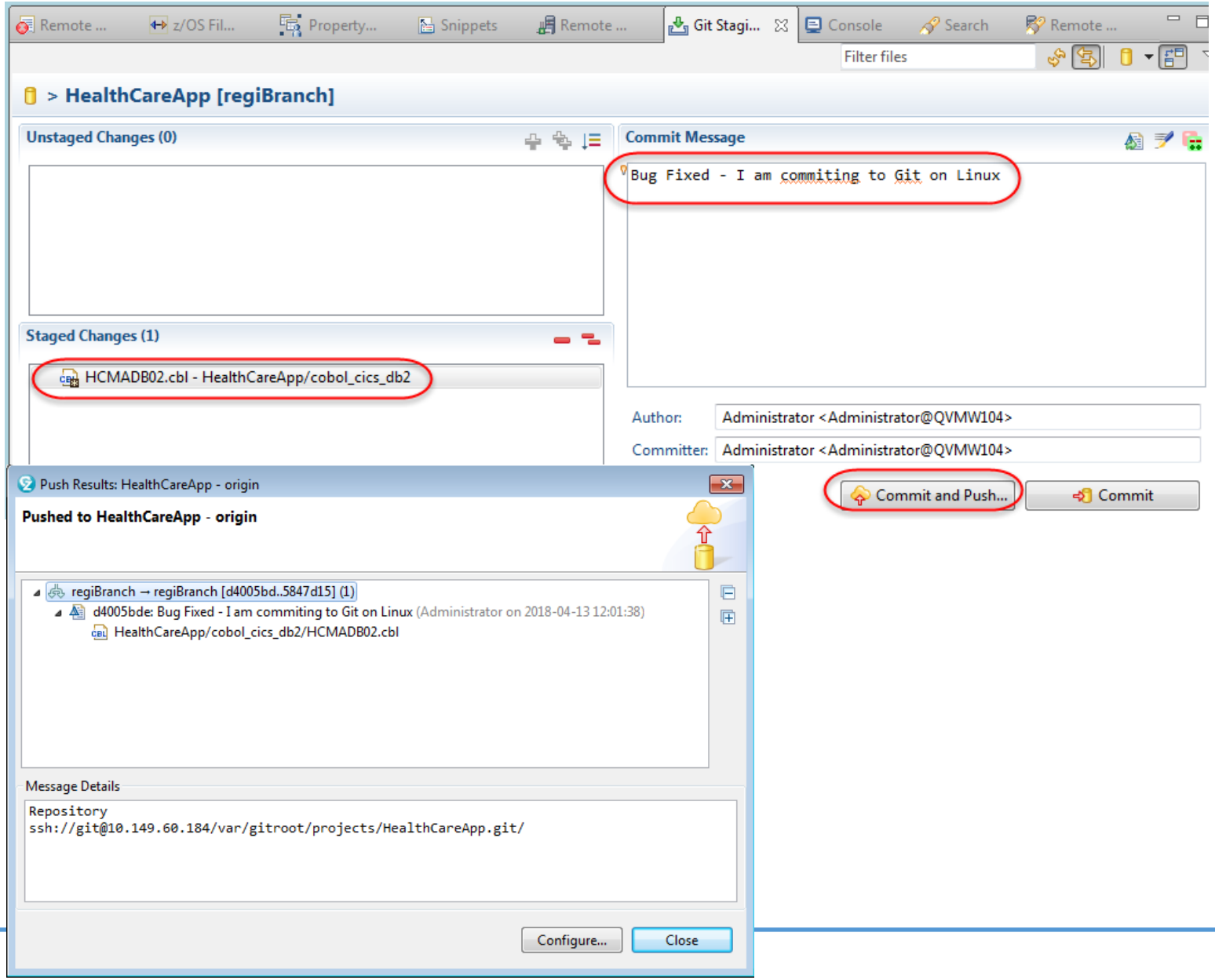
PART #2 - Developer see the CICS abend cause and fix it.

Bob – Mainframe Developer
Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

1. Uses **CICS** to verify the issue.
2. Uses **ADFz/Fault Analyzer** to identify what is causing the abend
3. Uses **ADFz/Debug** to debug the program and verify the error
4. Uses **ADFz** to update the COBOL program loaded from **GIT**
5. Uses **DBB** to perform a User Dependency Build and verify that the abend is fixed
6. Uses **GIT** to **Commit and Push** the corrected code to Repository

Uses **GIT** to Commit and Push the corrected code to Repository

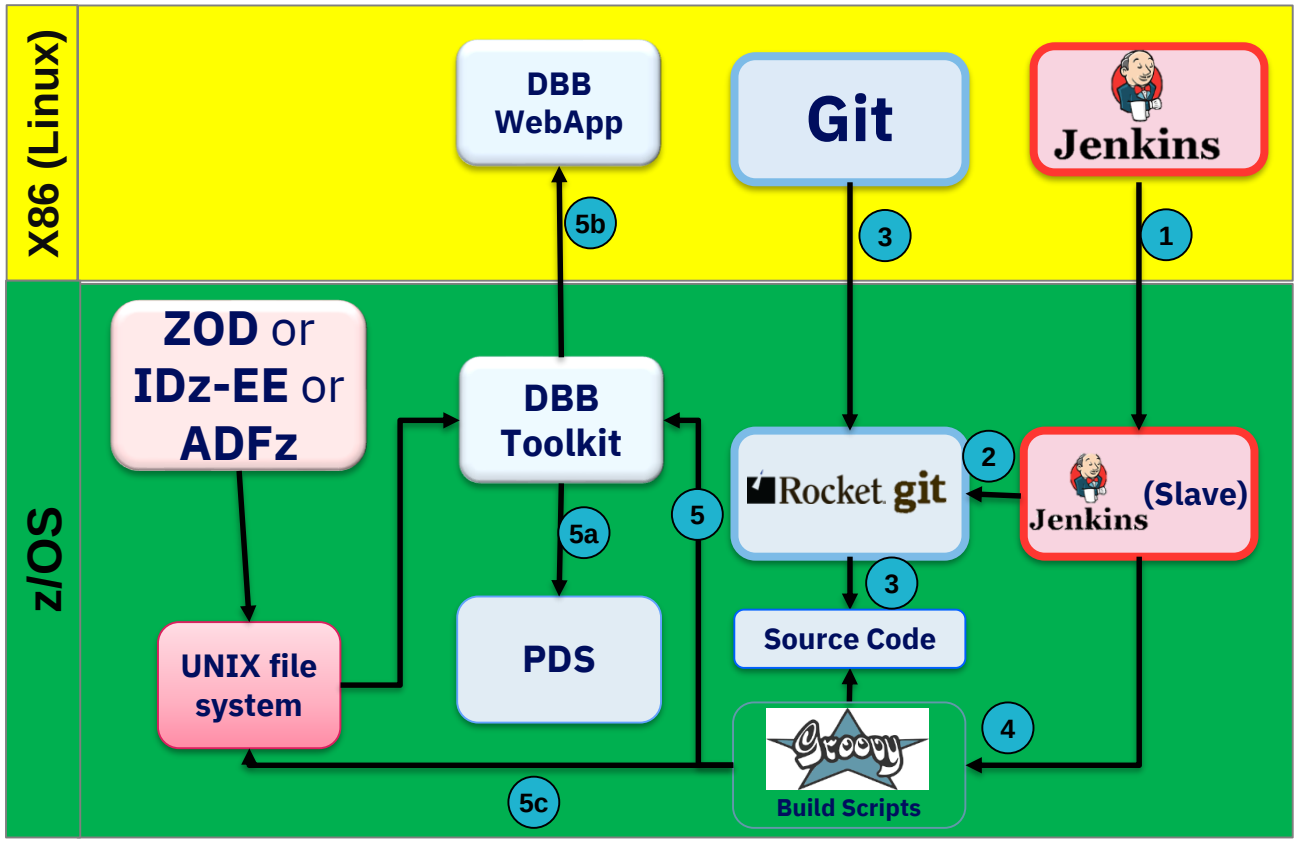


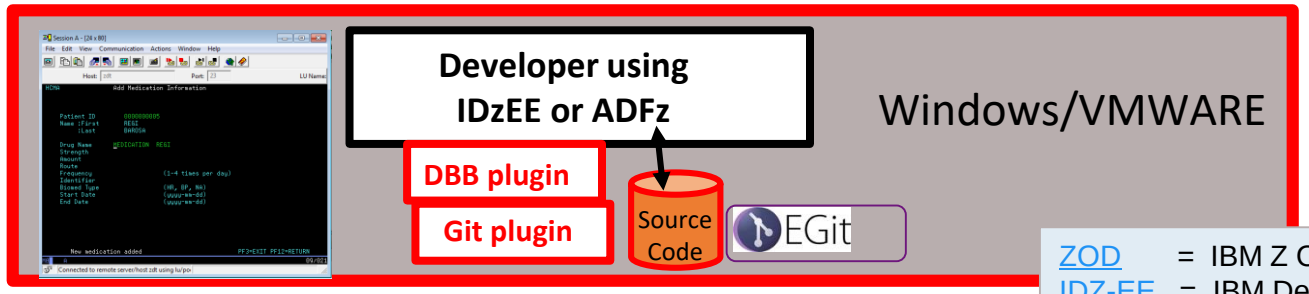
PART #3 - Release Engineer Deploy to z/OS and Mobile.

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

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:

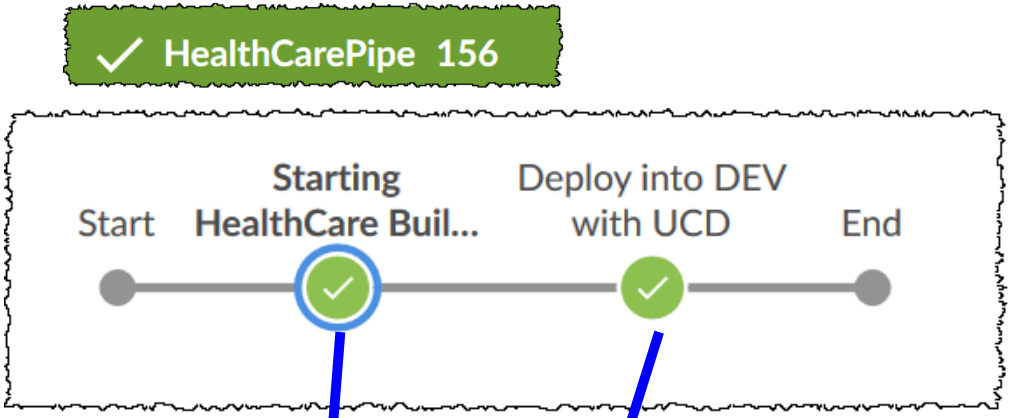
[5a] Create datasets, copy source from zFS to PDS, invoke zOS programs, issue ISPF and TSO commands.

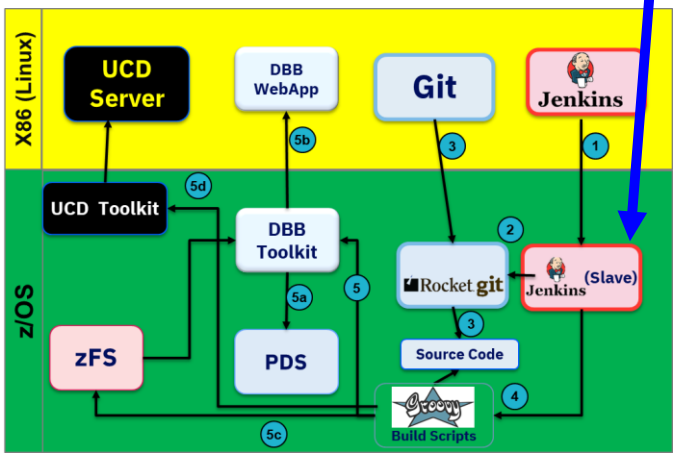
[5b] Scan and store dependency data from source files, perform dependency and impact analysis, store build results.

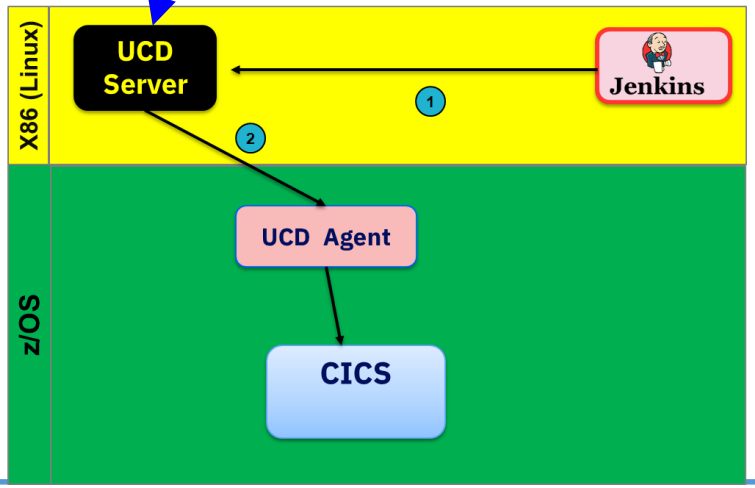
[5c] Copy logs from PDS to zFS, generate build reports (can be saved in build result).

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

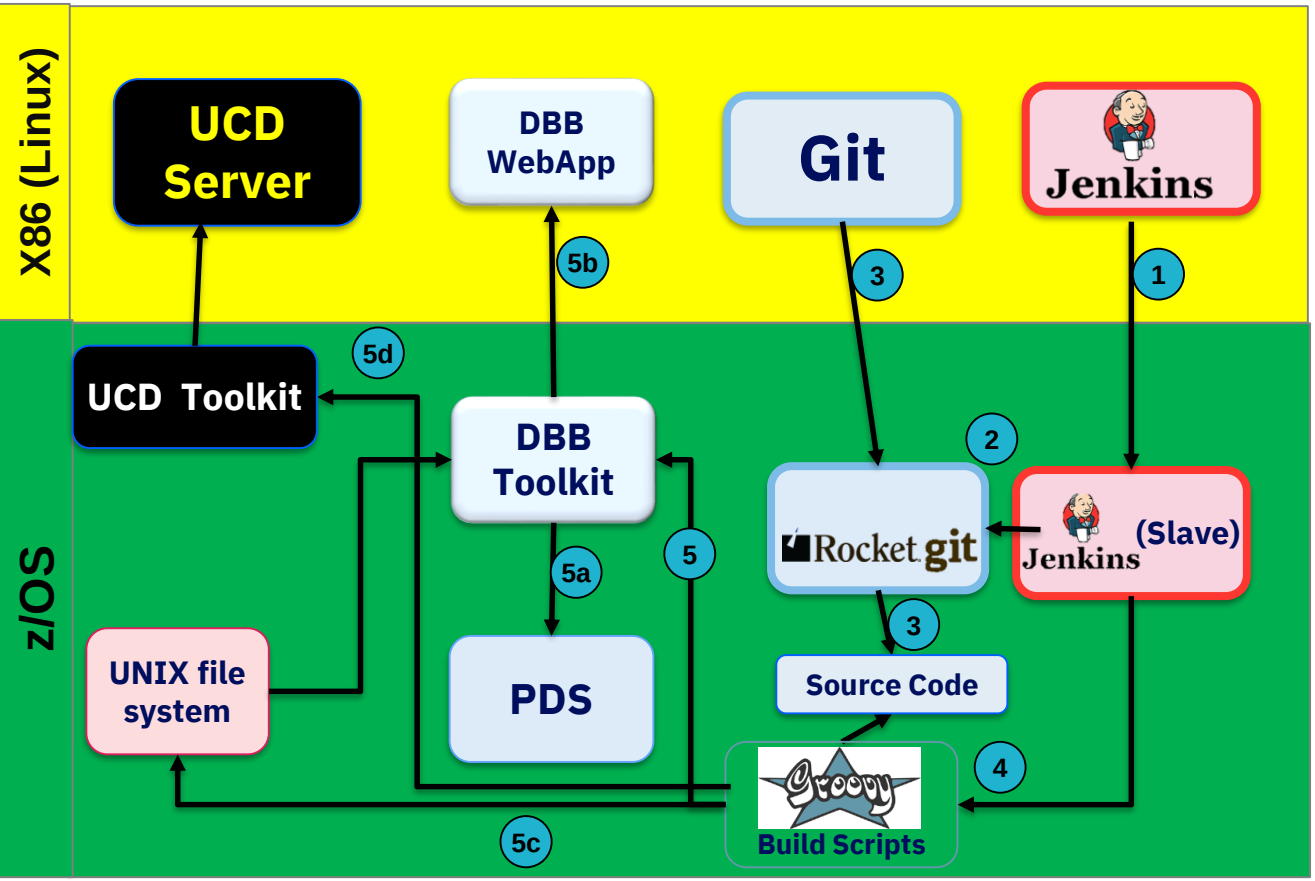
Demo High Level Diagram using Jenkins/Git/UCD







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:

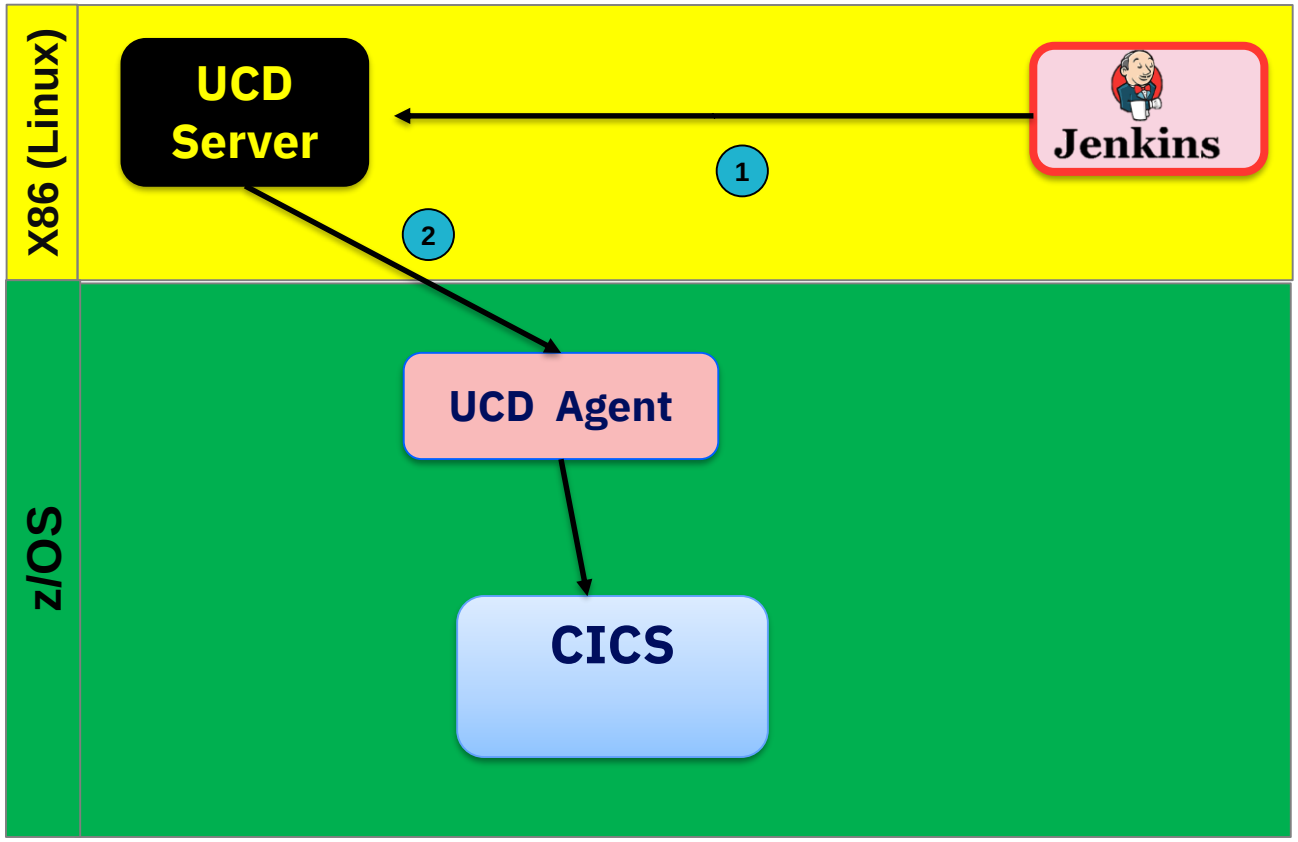
[5a] Create datasets, copy source from **zFS** to **PDS**, invoke zOS Compiler/Linkage Editor/ REXX for DB2 Bind.

[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)

High Level Demo Diagram with Jenkins and UCD (2)



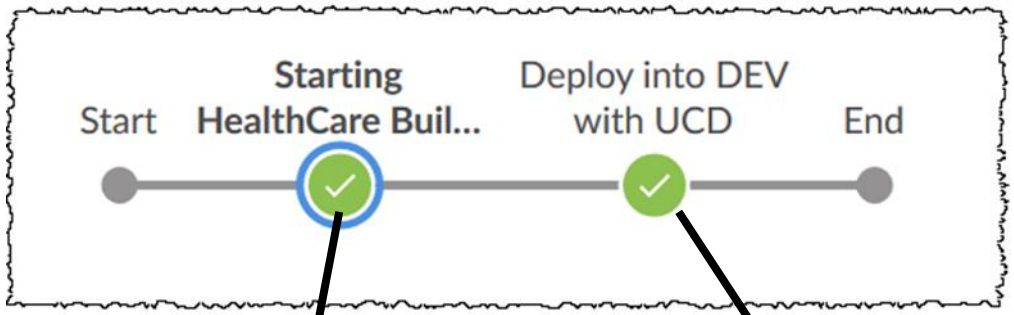
[1] Jenkins agent invokes UCD Deploy using UCD/Jenkins plugins.

[2] UCD Server uses the z/OS UCD agent to execute the deploy activities (coping datasets, issuing CICS Newcopy, etc)
.

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




```
** Invoking build scripts according to build order: Compile, LinkEdit, CobolCompile, JCL
* Building HealthCareApp/cobol_cics_db2/HCMADB02.cbl using CobolCompile.groovy script
Copying /var/jenkins/workspace/HealtCareAndUCD/HealthCareApp/cobol_cics_db2/HCMADB02.cbl to
IBMUSER.GIT.TEAM.ZMOBILE.COBOL(HCMADB02)
Resolving dependencies for file HealthCareApp/cobol_cics_db2/HCMADB02.cbl and copying to
IBMUSER.GIT.TEAM.ZMOBILE.COPYBOOK
Compiling and link editing program HealthCareApp/cobol_cics_db2/HCMADB02.cbl
** Build finished at Fri Apr 13 15:34:35 GMT 2018
** Build State : CLEAN
** Total files processed : 1
** Total build time  : 1 minutes 17.527 seconds
```

```
Started by user Admin User
Building on master in workspace /var/lib/jenkins/workspace/DeployHealthCare
Deploying component versions '{HC CICS=[latest]}'
Starting deployment process 'Deploy HC to CICS + MF to Windows' of application 'A HC zOS COBOL CICS' in
environment 'Dev'
Deployment request id is: '1ad41013-f10c-48ff-950c-4ee6a20641bf'
Deployment is running. Waiting for UCD Server feedback.
Finished the deployment in 361 seconds
The deployment result is SUCCEEDED. See the UrbanCode Deploy deployment logs for details.
Finished: SUCCESS
```

```
Create version and store package:
....Version artifacts stored to UCD server CodeStation
....Version:20180413-153539 created
Elapsed time 59.0 seconds.

** buztool output properties
   version.url -> https://ucdserver:18443//#version/5527f4c7-d7b6-41d6-9755-545908b58677
   version.repository.type -> CODESTATION
   version.name -> 20180413-153539
   version.id -> 5527f4c7-d7b6-41d6-9755-545908b58677
   component.name -> HC_CICS
   version.shiplist -> /var/jenkins/workspace/HealtCareAndUCD/BUILD-19/shiplist.xml
Warning: you have no plugins providing access control for builds, so falling back to legacy behavior of
permitting any downstream builds to be triggered
Triggering a new build of DeployHealthCare
Finished: SUCCESS
```

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

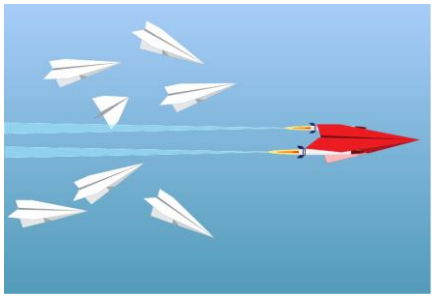
GitHub

GitLab

ATLASSIAN Bitbucket

Git vs. traditional SCM

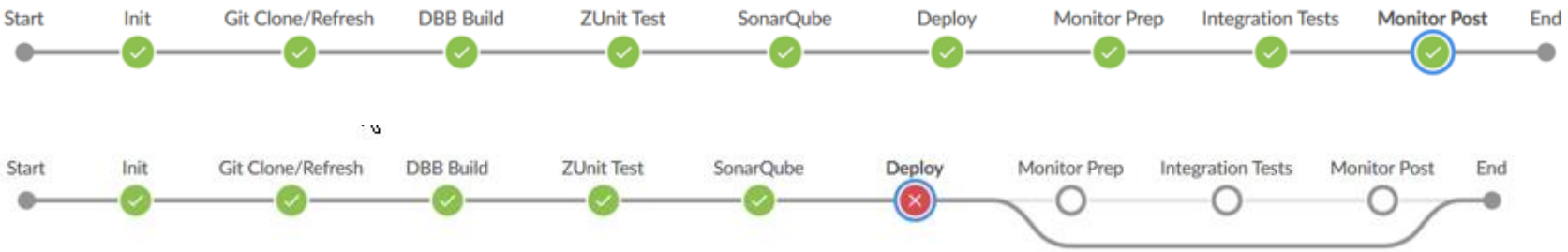
- True parallel development
  - Branches on demand
  - Freedom from pre-defined streams



- 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

Verify the Build results using **DBB browser**

Collections Build Results

DBB Build Result

Field	Value
id	427
group	HealthCareApp
label	build.20180413.033317.033
Build Report	view
buildReportData	view
state	2 (COMPLETE)
status	0 (CLEAN)

Toolkit Version:

Version:	0.9.1
Build:	96
Date:	16-Feb-2018 21:49:21

Build Summary

Number of files being built: 1

	File	Commands	RC	Data Sets	Outputs	Logs
1	HealthCareApp/cobol_cics_db2/HCMADB02.cbl Show Dependencies	IGYCRCTL	4	IBMUSER.GIT.TEAM.ZMOBILE.COBOL(HCMADB02)	IBMUSER.GIT.TEAM.ZMOBILE.DBRM(HCMADB02)	HCMADB02.log
		IEWBLINK	0		IBMUSER.GIT.TEAM.ZMOBILE.LOAD(HCMADB02)	
		exec 'IBMUSER.GIT.ZMOBILE.REXX(BINDHC2)'	0			HCMADB02_bind.log

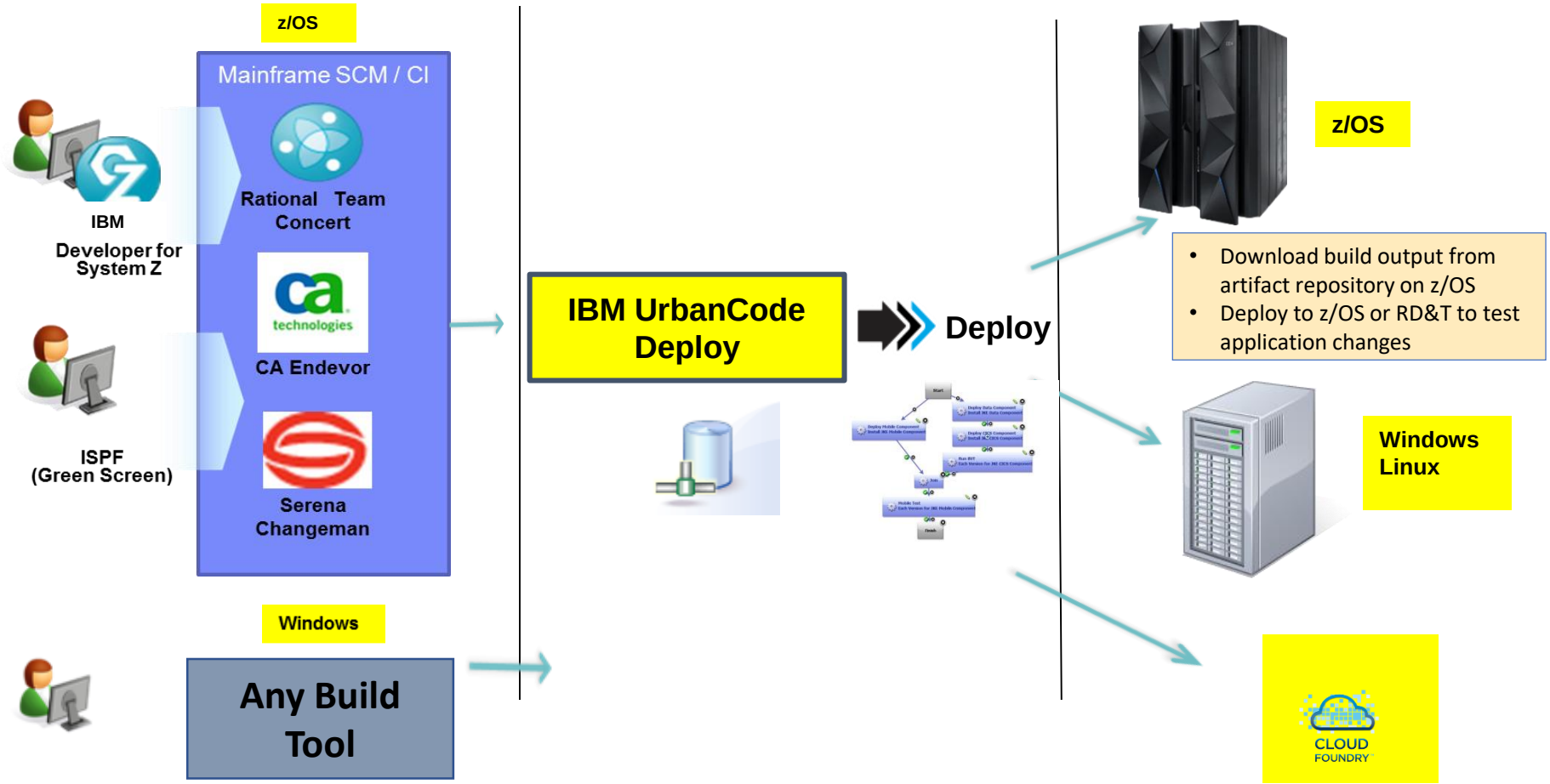
PART #3 - Release Engineer Deploy to z/OS and Mobile.

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

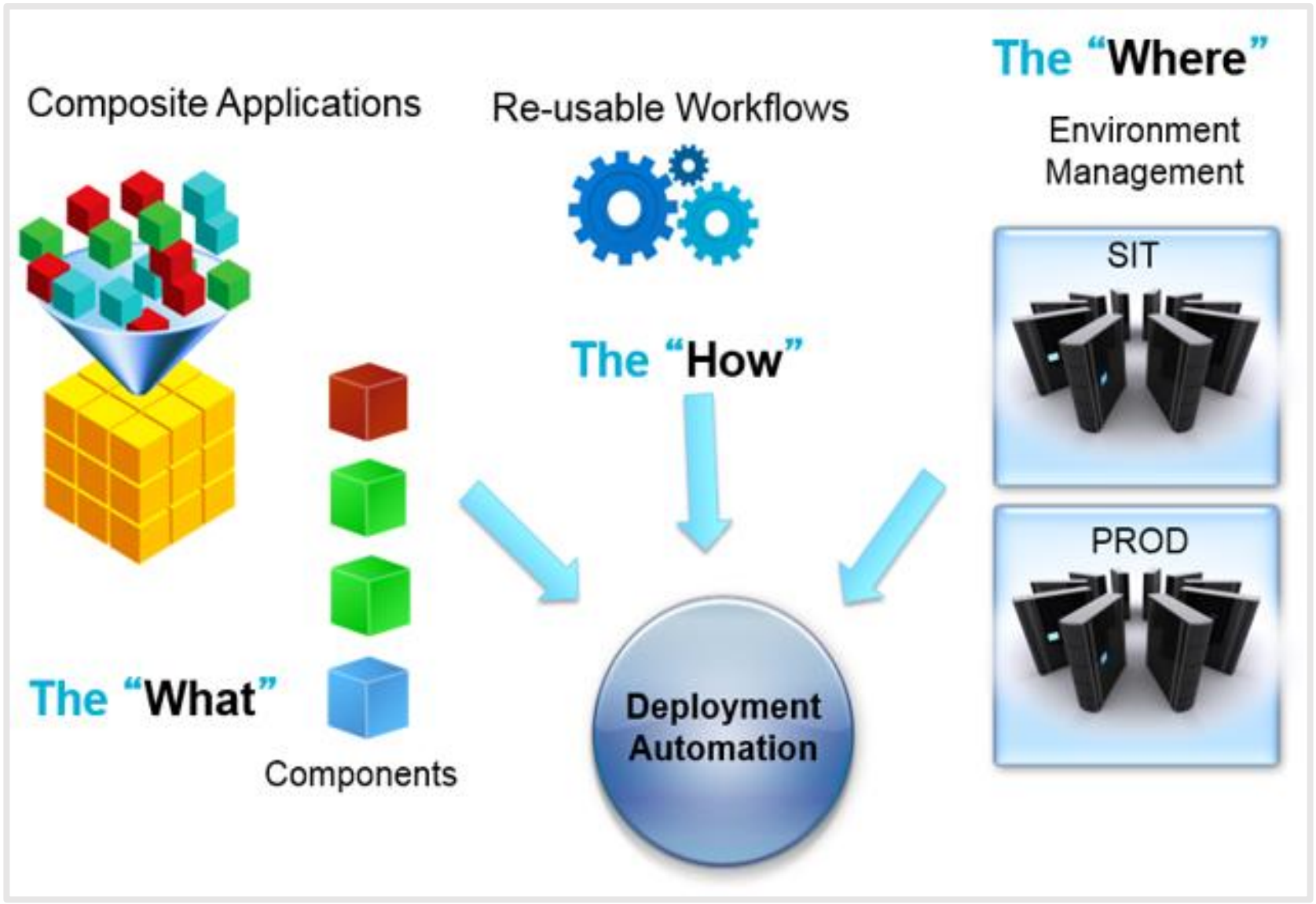
Continuous Delivery for the Mainframe and Others

Capabilities to speed delivery of interdependent, multi-platform applications

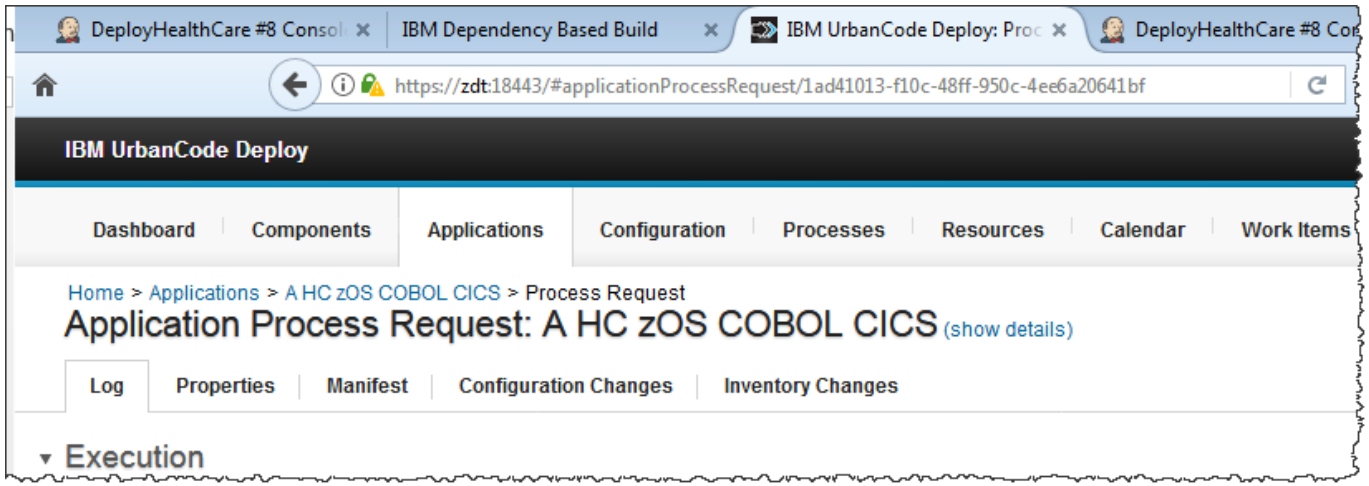


- Provides a **unified solution** for continuous delivery of heterogeneous enterprise applications
- **Accelerate delivery** and reduces cycle time to develop/test multi-tier applications across heterogeneous environments and platforms
- **Reduce costs** and eliminate delays for delivering mainframe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

UrbanCode Deploy: What makes it different?



Verify the Deploy results using **UCD browser**



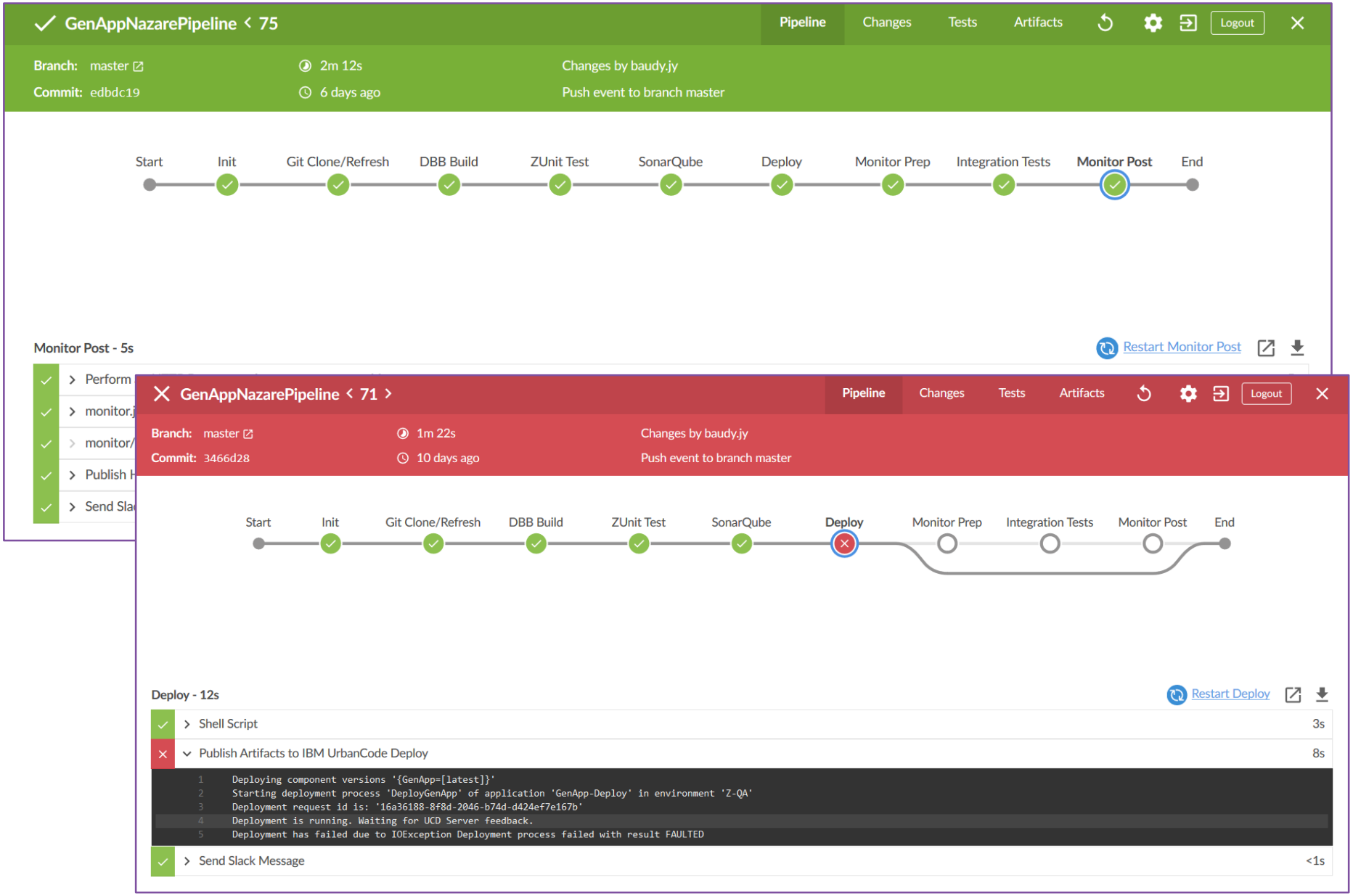
1. Install JKEMobileWindows7		11:36:35 AM	0:00:00	Not Mapped
2. Install HC_CICS	1 / 1	11:36:35 AM	0:05:54	Success
HC_CICS	1 / 1	11:36:36 AM	0:05:54	Success
Deploy HC to CICS (HC_CICS 20180413-153539)		11:36:36 AM	0:05:54	Success
1. Download Artifacts for zOS		11:36:38 AM	0:01:08	Success
2. Deploy Data Sets for CICS		11:37:47 AM	0:01:16	Success
3. Generate Program List		11:39:03 AM	0:01:30	Success
4. GenBndCard		11:39:03 AM	0:01:31	Success
5. bindPackage & PLAN		11:40:35 AM	0:00:59	Success
6. NEWCOPY Programs		11:41:34 AM	0:00:53	Success
3. Install HCMobileWindows7		11:36:35 AM	0:00:00	Not Mapped

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 on z Systems Introduction
9:45 Application Delivery Foundation (ADFz):
- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break

10:30 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 z Open Unit Test (Zunit) : Overview & Demonstration

1:00 Day in the Life - Using Application Discovery, DBB/Git/Jenkins and UCD

2:00 – 4:30 Choose one Optional lab:

→ **LAB 0 - Unit Testing for COBOL CICS** (1 hour)
→ **LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (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 0: Unit Testing for COBOL CICS

In this exercise, you will use IBM Developer for Z Systems to create and execute automated unit tests for a program using CICS. This process includes recording or capturing data from a live CICS Application run and importing data for playback/stubbing so that you can run your unit tests without deploying.
.

You will need ANOTHER z/OS instance (ask for help)

Userid: **IBMUSER** password: **sys1**

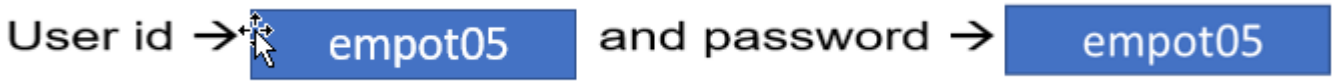
Overview of development tasks

- Use ZUNIT from IBM Developer for Z systems to create a unit testing test case from a recording.
- Run the test
- Modify a program and view the failure of a test case

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.

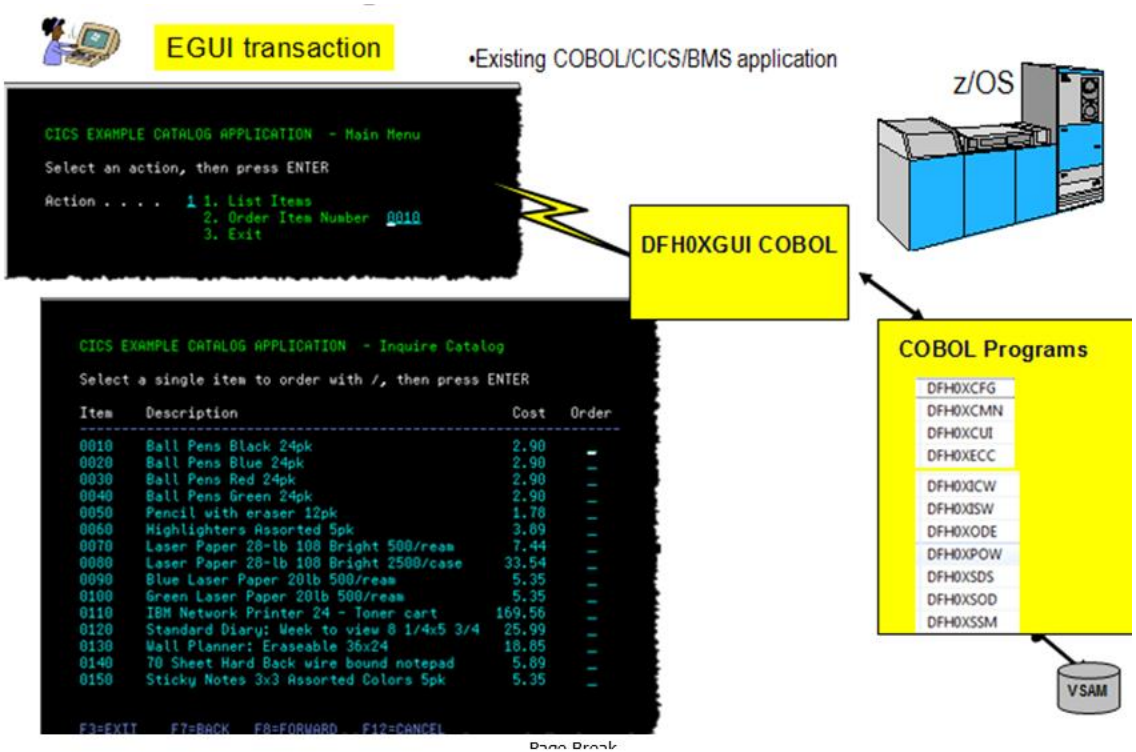
6. **Use Jenkins with Git plugin to build the modified code**
   → You will use Jenkins pipeline to build the new changed code and push the executables to be deployed using UCD.

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

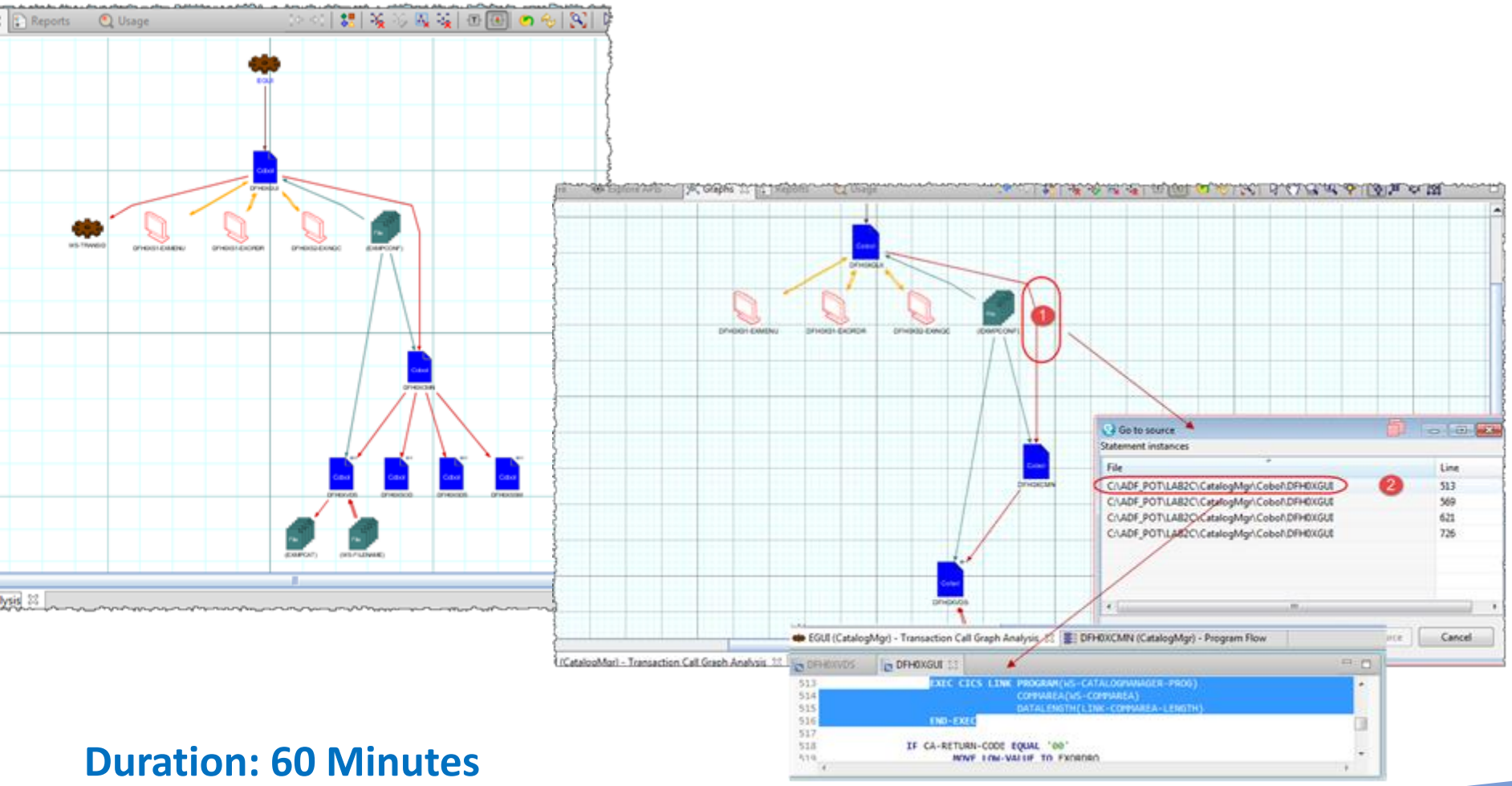
Objective: This lab you will go through the process of discovering a possible API from existing the Catalog Manager Application (EGUI transaction).



Duration: 60 Minutes

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

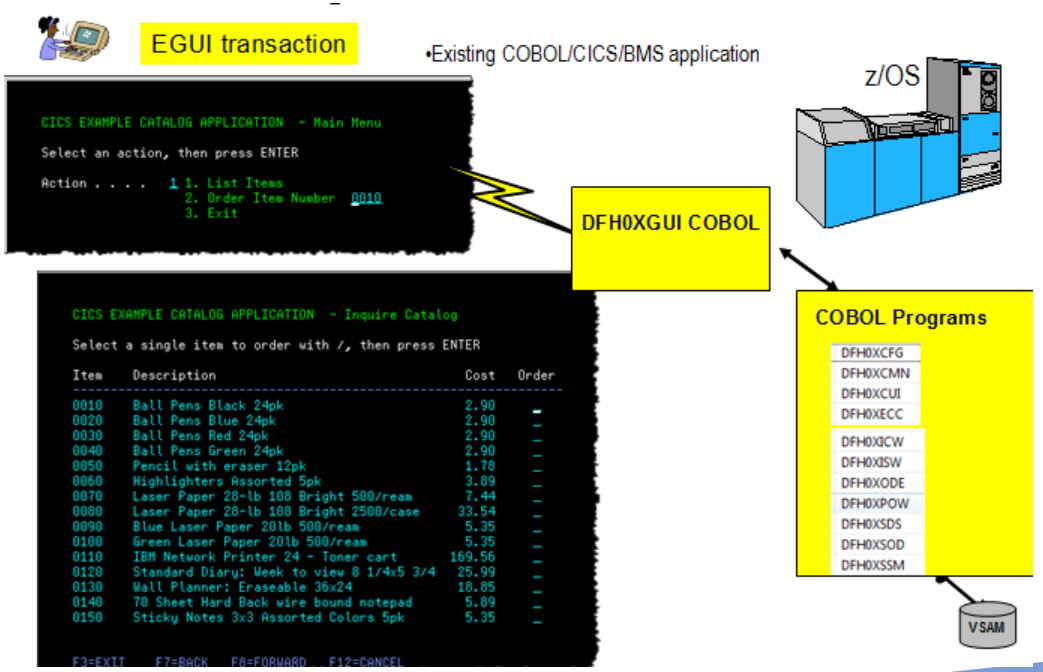
Objective: This lab you will go through the process of discovering a possible API from existing the Catalog Manager Application (EGUI transaction).



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

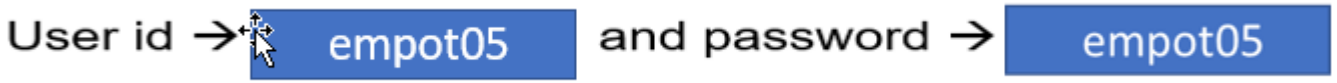
Part 3 – Deploy the application to z/OS CICS

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

Lab 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

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

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).

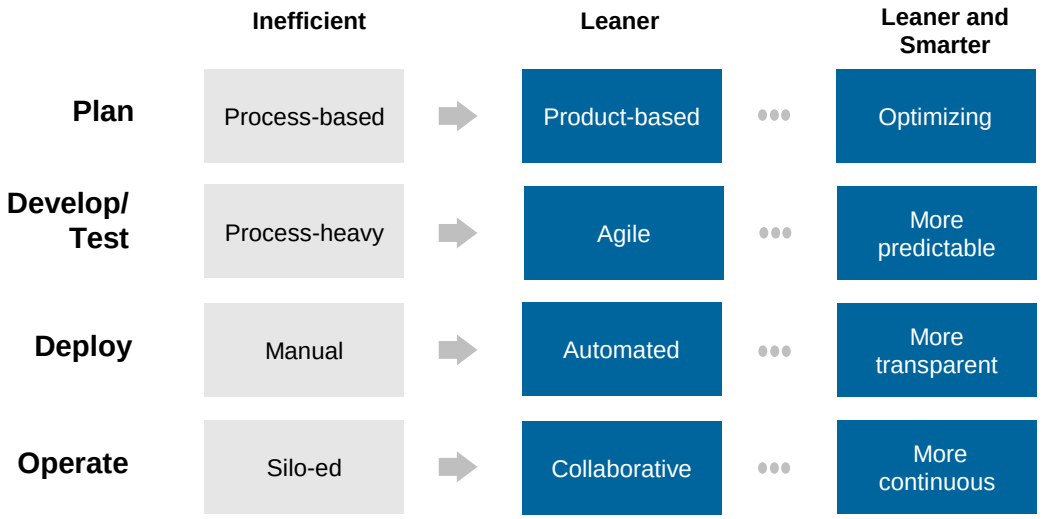
On this lab you will measure an existing COBOL/DB2 program running in batch.

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

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

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

No-Charge IDz/ADFz Training offerings for the fall

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

IDz/ADFz/RDz Entry-Level and Experienced User Training Schedule

Date & Time	Topic
September 11 - 9:00 AM EDT	**IDz Entry-Level Training Module 6** This module instructs on the dataset utility features of IDz that are utilized for ISPF 3.x tasks: File allocation, File rename, File delete, Library compress, VSAM File create, GDG Model create. Batch Job management and interactive CLIST/REXX support is also part of Module 6.
September 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.
September 25 - 9:00 AM EDT	**IDz Entry-Level Training Module 8 - DB2/SQL** IDz's world-class features for understanding your logical and physical data model (entities & their relationships), modifying DB2 test table values and coding/testing SQL: Using SPUFI files, Interactive SQL, and coding Embedded SQL statements directly into COBOL and PL/I programs, then running and using Visual Explain on the statements.
September 30 - 9:00 AM EDT	** Next Iteration of IDz Entry Level Training Starts - Monday Sessions ** **Module 1 - IDz/Eclipse Introduction for ISPF Developers** Class overview, Workspace installation & customization, IDz Terms & Concepts & Eclipse Navigation for ISPF Developers, DevOps Tooling overview, Split-Screen, Use of multi-window'd views, Program Control Flow, Bookmarks for ISPF-Labels and more.

No-Charge IDz/ADFz Training offerings for the fall

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

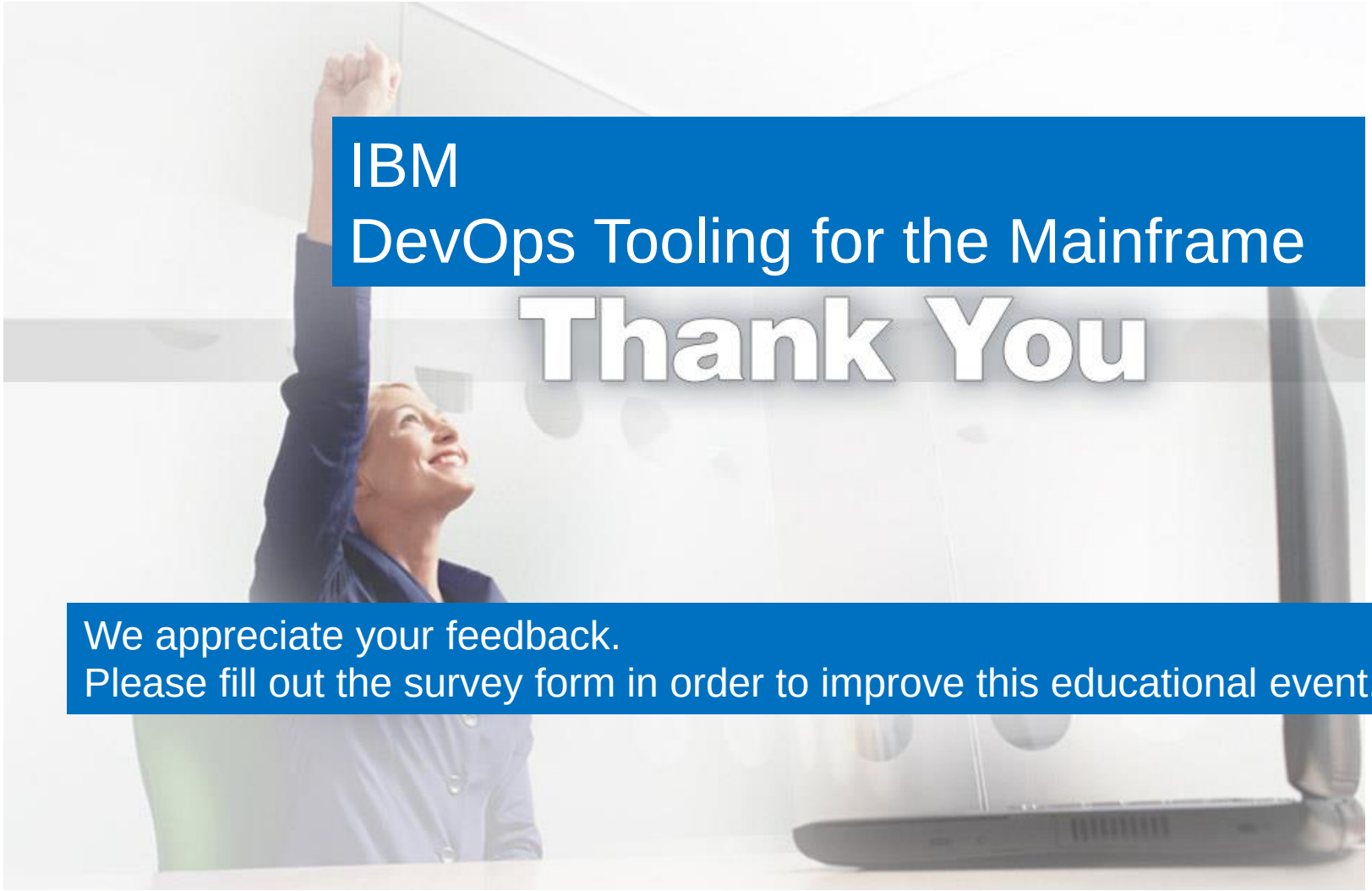
September 30 - 9:00 AM EDT	** Next Iteration of IDz Entry Level Training Starts - Monday Sessions ** **Module 1 - IDz/Eclipse Introduction for ISPF Developers** Class overview, Workspace installation & customization, IDz Terms & Concepts & Eclipse Navigation for ISPF Developers, DevOps Tooling overview, Split-Screen, Use of multi-window'd views, Program Control Flow, Bookmarks for ISPF-Labels and more.
October 1 - 9:00 AM EDT	IDz Experienced Training: File Manager Learn how to use the File Manager "GUI" plug-in with IDz to browse and edit QSAM (Sequential) and VSAM file records - both in unformatted and in formatted (Copybook overlay) mode. Also learn how to connect to File Manager and use the tools such as Queries and Filtering Sorting & Excluding records. Time-permitting DB2 will be covered.
October 2 - 9:00 AM EDT	**IDz Entry-Level Training Module 9 - Debug & Code Coverage** Batch & Online debugging features and techniques, including JCL to launch Debug, program animation features, break points, variable monitors/changing variable values dynamically, Record & Playback and an introduction to Code Coverage.
October 7 - 9:00 AM EDT	**IDz Entry-Level Training Module 2 - IDz Editing and Code Development Tools & Techniques** Overview of IDz code editors & productivity techniques, ISPF emulation: Command line commands/Prefix area commands, Hot Key definitions, Editing source with embedded Hex characters, JCL editing, IDz's source download and program model, and Content Assist ("look-ahead typing")
October 14, 9:00 AM - 10:00 AM EDT	**IDz Entry Level Training - Module 3 - Program Analysis Tools** ISPF Search techniques (review), IDz graphical tools for: Logic Flow and program execution analysis, Graphical & declarative tools for Data Flow and impact analysis (within a single program).

No-Charge IDz/ADFz Training offerings for the fall

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

October 30, 9:00 AM EDT	** Next Iteration of IDz Entry Level Training Starts - Wednesday Sessions ** **IDz Module 1 - IDz/Eclipse Introduction for ISPF Developers** Class overview, Workspace installation & customization, IDz Terms & Concepts & Eclipse Navigation for ISPF Developers, DevOps Tooling overview, Split-Screen, Use of multi-window'd views, Program Control Flow, Bookmarks for ISPF-Labels and more.
November 4, 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
November 6, 9:00 AM EST	**IDz Entry-Level Training Module 2 - IDz Editing and Code Development Tools & Techniques** Overview of IDz code editors & productivity techniques, ISPF emulation: Command line commands/Prefix area commands, Hot Key definitions, Editing source with embedded Hex characters, JCL editing, IDz's source download and program model, and Content Assist ("look-ahead typing")
November 11, 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.
November 12, 9:00 AM EDT	IDz Experienced Training: Code Coverage Learn how to create Code Coverage statistical reports that analyze program lines executed during a batch or online (CICS) test run. Topics include: The ENVAR statement, Including/Excluding Programs, Running in batch

Questions / Comments



IBM
DevOps Tooling for the Mainframe

We appreciate your feedback.
Please fill out the survey form in order to improve this educational event.

Z Open Development (ZOD)	IDz Enterprise Edition (IDZ-EE)	IBM Developer for z Systems (IDz)
Entry level tools to link z/OS development seamlessly with an established, open DevOps toolchain	*All capabilities and more under one license for your enterprise, supporting all phases of your digital transformation*	*Comprehensive, modern toolset for developing and maintaining z/OS applications using DevOps or traditional software delivery practices*
Dependency Based Build	Dependency Based Build	
Modern SCM integration (Git or Rational Team Concert)	Modern and traditional SCM integration (Git, Rational Team Concert, and CA Endevor) plus the CARMA framework for additional SCM integration and customization	Modern and traditional SCM integration (Git, Rational Team Concert, and CA Endevor) plus the CARMA framework for additional SCM integration and customization
Modern Eclipse style editors for COBOL, PL/I, Java, JCL. BMS and MFS editor.	Choice of editors (Eclipse or ISPF style) for COBOL, PL/I, Java, JCL, HLASM, C/C++, and REXX. BMS and MFS editor	Choice of editors (Eclipse or ISPF style) for COBOL, PL/I, Java, JCL, HLASM, C/C++, and REXX. BMS and MFS editor.
Program understanding	Program understanding	Program understanding
Visual debug support for COBOL and PL/I. Remote debug support for COBOL, PL/I and C/C++.	Visual debug support for COBOL and PL/I. Remote and 3270 debug support for COBOL, PL/I, C/C++, Assembler, and ABO optimized modules. IMS Transaction Eclipse UI. Load Module Analyzer. IBM COBOL and CICS Command Level Conversion Aid for OS/390 & MVS & VM.	Visual debug support for COBOL and PL/I. Remote debug support for COBOL, PL/I, C/C++, Assembler, and ABO optimized modules.
	Code coverage	Code coverage
Client side code rules, SonarLint integration	Code rules (client-side and host-based automated collection), SonarLint integration	Code rules (client-side and host-based automated collection), SonarLint integration
	zUnit for automated unit test	zUnit for automated unit test
	Refactoring	Refactoring
	Traditional access, with host connection emulator and menu manager, to integrate legacy tools and processes	Traditional access, with host connection emulator and menu manager, to integrate legacy tools and processes
	Enterprise Service Tools and J2C	Enterprise Service Tools and J2C
Deep integration with CICS Explorer, IMS Explorer, ADDI, FA, FM, and APA	Deep integration with Data Studio, ZD&T, CICS Explorer, IMS Explorer, ADDI, FA, FM, and APA	Deep integration with Data Studio, ZD&T, CICS Explorer, IMS Explorer, ADDI, FA, FM, and APA
Licensed by user (authorized)	Licensed by capacity (Value Unit)- unlimited users	Licensed by user (authorized and floating)
Available in PPA, trade up path to IDz Enterprise Edition	Available in Shopz	Available in PPA, trade up path to IDz Enterprise Edition