DevOps on Z PoT

Proof of Technology

DevOps Solution for Z Development

David Hawreluk
dhawrel@us.ibm.com

Vanessa Callaghan
Vanessa.r.callaghan@ibm.com

Wilbert Kho
wilbert@us.ibm.com

Sept-23-2021

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 concepts
- Awareness (not necessarily proficiency) of basic z/OS System Programming tasks
- Basic familiarity with JCL concepts (JCL skills NOT required)
- No Java, COBOL or PL/I skills are required
- Developers are welcome

Proof of Technology Objectives

The objective of this session is to demonstrate through hands-on workshops, the power of collaborative development and delivery enabled by the Enterprise DevOps solutions.

Application teams can learn how they can modernize their application portfolio using modern technology and tools. We will focus on code and build, automate testing and deploy.

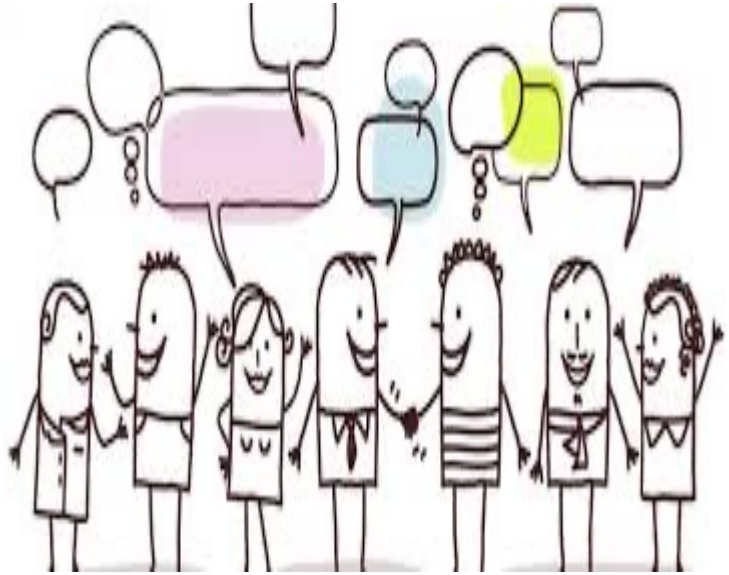
Production teams can learn how they can extend and automate their existing build and deployment solutions for enterprise applications

Areas covered:

- Enterprise analytical tool for application understanding and modernization (ADDI)
- Integrated application development and problem analysis (ADFz)
- Managing your source code using modern technology (including Git)
- Building automation (DBB) for COBOL without a specific source code manager but using a pipeline automation tool (including Jenkins)
- Using a unit testing framework (zUnit) for COBOL as part of the development and build environment.
- Using IBM Z Virtual Test Platform (VTP) for application Integration Testing of a COBOL/CICS/DB2 application
- Running Integration tests using Galasa framework
- Application deployments automation to many environments (UCD)
- Expose Mainframe legacy applications via RESTful APIs (z/OS Connect)
- Measure and report on how system resources are used by mainframe applications to improve performance (APA)

Few introductions (15 min)…

- Introduce yourself and your team
- What's your role within your organization ?
  (Architect? Developer ? DBA? Manager? ????)
- Current development tools set in use?
  (IDz/TSO/SCLM/Changeman/Endevor)
- What do you hope to get out of this session?



1 minute Please….

Agenda - Day 1

2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)

3:00 LAB 1 - Working with z/OS using COBOL and DB2

or

LAB7 - IBM DBB with Git, Jenkins and UCD on Z

5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

2:00 zUnit and VTP Overview (Wilbert)

2:15 Demo: Scenario using Z DevOps solutions including **ADDI**, **zUnit**, **Git** and **Jenkins** (David/Regi)

3:00 – 4:50 Choose one Optional lab:

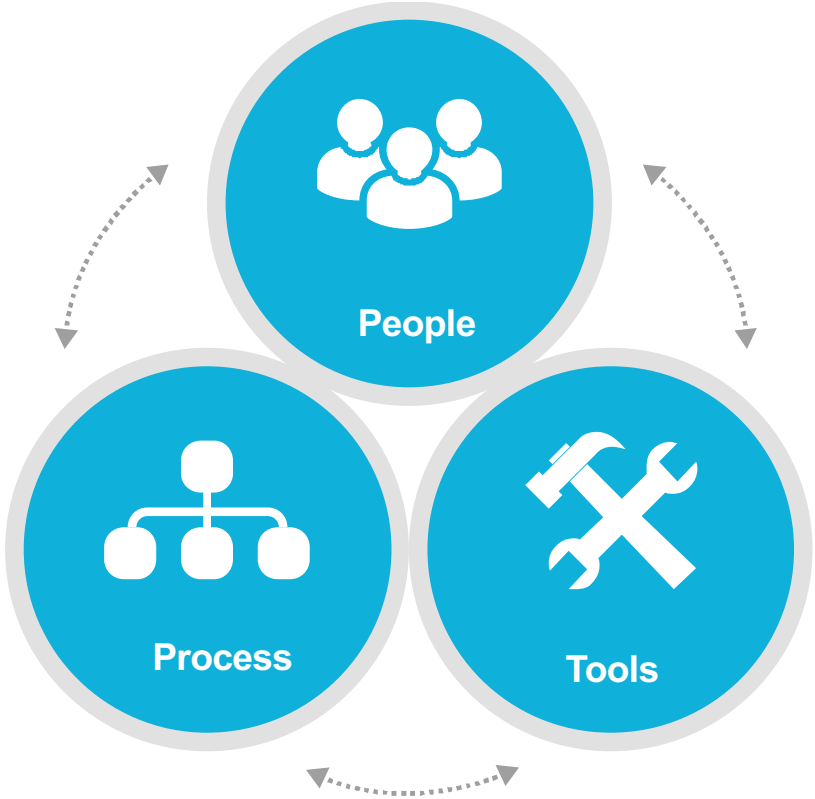
- → LAB 7 - IBM **DBB** with **Git**, **Jenkins** and **UCD** on Z
- → LAB 6B : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Remote Assets**
- → LAB 6C : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Local Assets**
- → LAB 6D : Using **IBM zUnit** to Unit Test a COBOL/DB2 **batch** program
- → 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**)
- → LAB 9 - Using **IBM Z VTP** to test a COBOL /CICS / DB2 transaction
- → LAB 10 - Deploying COBOL/CICS/DB2 application using a **GitLab CI** Pipeline
- → LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- → LAB 12 – Using **ADDI** to find CICS programs to be converted to API's
- → LAB 13 – Running Integration Tests using **Galasa**

5:00 Checkpoint - Comments and Closing Day 2 (David/Ron/Wilbert/Regi)

Lecture Labs

DevOps is not one of these things…

It’s all of them!



DevOps is a set of practices that combines software development (Dev) and IT operations (Ops).
It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.

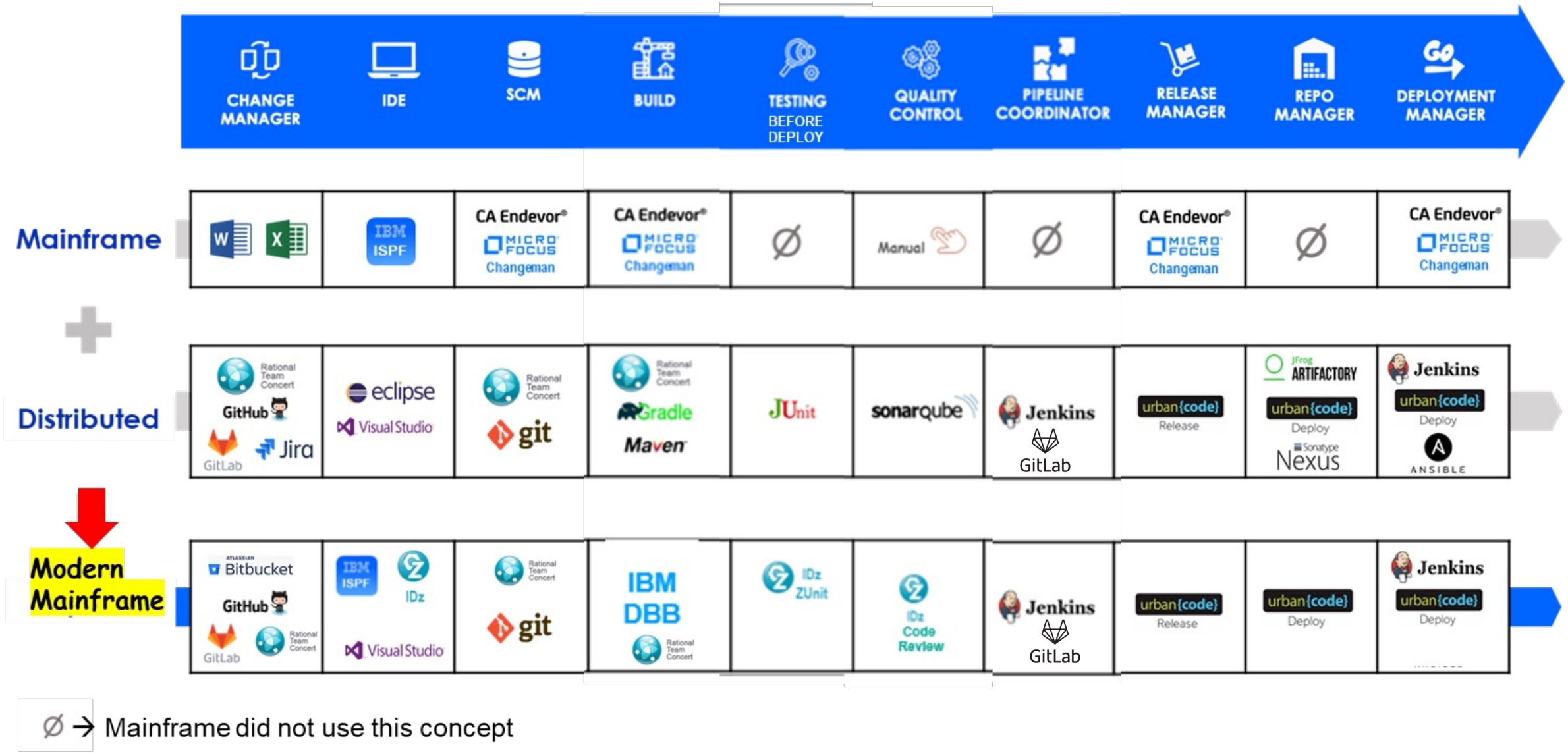
Why DevOps for z/OS?

- Client Value through increased quality
  - Automated test (at any level) at code delivery and
  - Code reviews will include results of build, package and tests

- Modernizing to industry standard tools
  - Enables z/OS to leverage relevant open source tooling. E.g *Jenkins* and *Git*
  - Unlocks an ecosystem not currently available to z/OS dev *
  - Decreased time to market

- Retention and Skills
  - New hires start out knowing some of the DevOps tools
  - Using homegrown, hard to maintain, proprietary tools is a problem
  - Lack of automation increases the time until a new team member is contributing fully
  - Current tools support not sustainable

Imagine a unified Pipeline on Z/OS



How to justify DevOps?
The value of early and extensive testing

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

During the Coding or Unit Testing phases

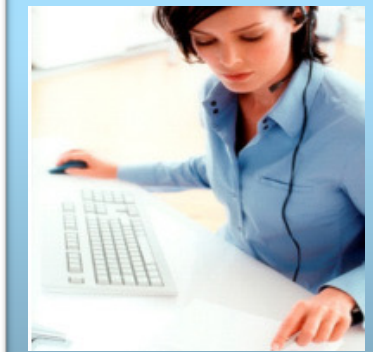
$80/defect

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.

What is Git ?

- A distributed, open source version control system
  - Runs on all platforms (z/OS, Linux, Windows, Mac)
    - z/OS uses Rocket® Open Source implementation on USS
  - Command line interface
  - Community supported

- The de-facto standard for software development
  - Integrated into many tools
    - Jenkins, Slack, Urban Code Deploy, Ansible, Artifactory, Jira, JUnit, Maven, Gradle, Make, DBB, Azure and many more.

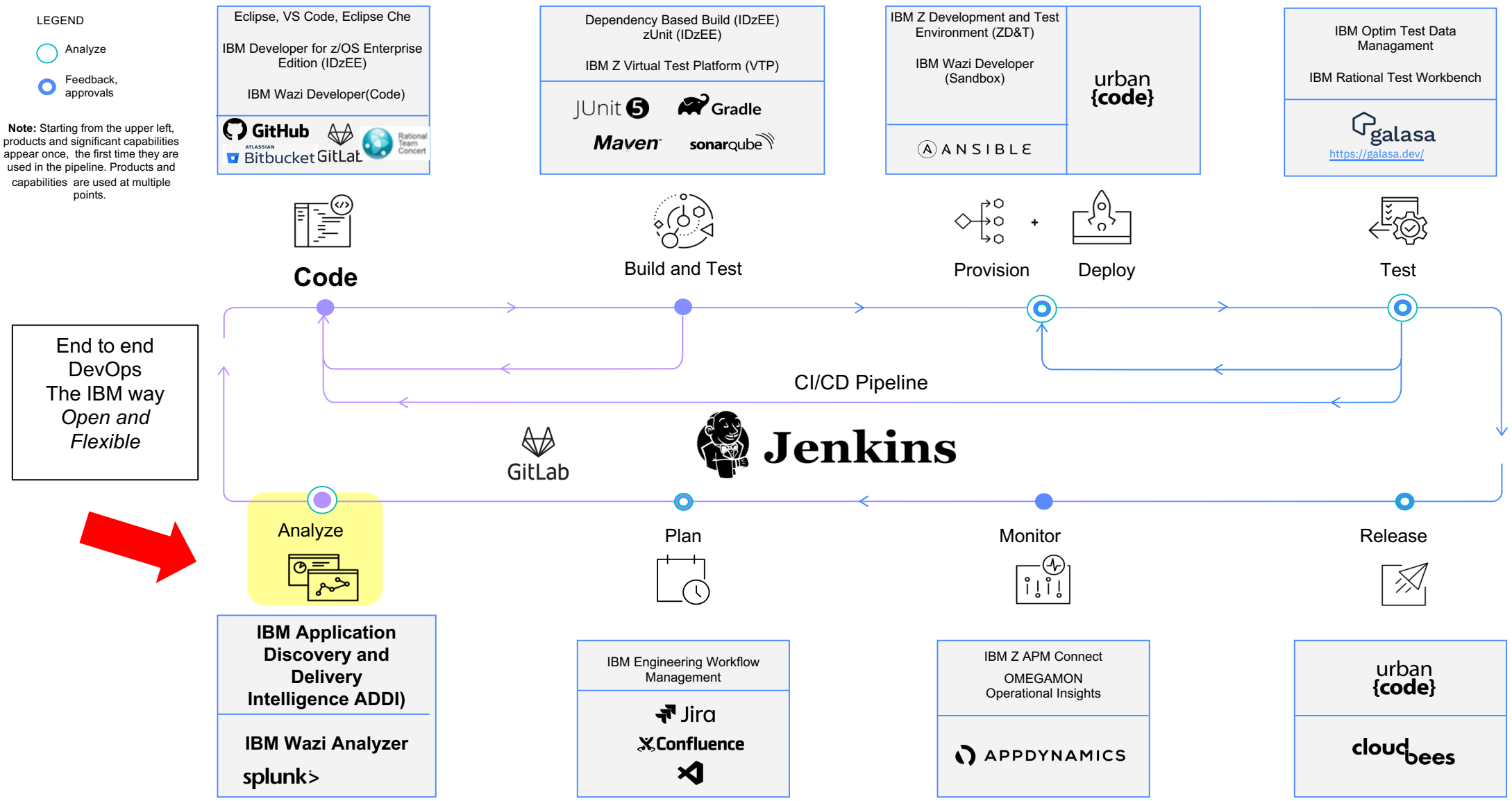
What is Jenkins ?

“Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.”

Ref: https://www.jenkins.io/doc/

Note: Jenkins is not the only orchestrator. You can use Azure DevOps, TeamCity, Bambo ...

Z DevOps Pipeline



AD and **ADDI** Extensions

AD
"Application Discovery Browser "

Call graphs | Resource Usage | Reports | Impact Analysis

Source Code

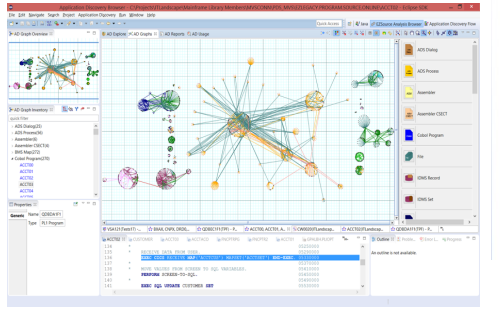
IBM Application Discovery Build Client

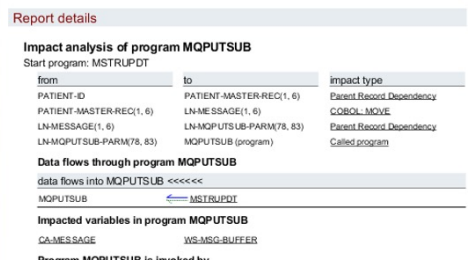
Static code analysis

Incremental update

AD Discovers "Business Keywords" used by/within business Terms in COBOL

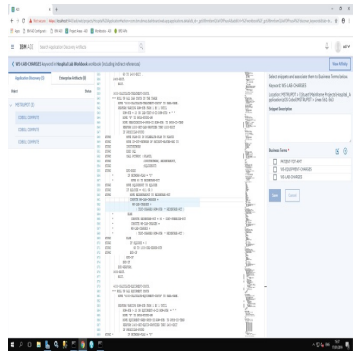
AD Repository

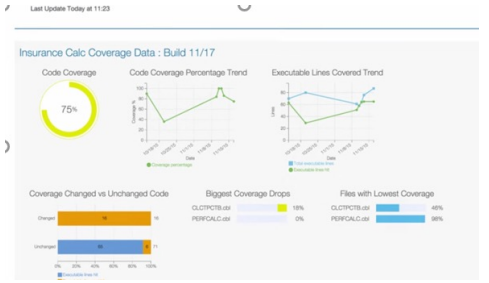




ADDI Extensions
"WEB Browser "

ADDI Extensions Repository





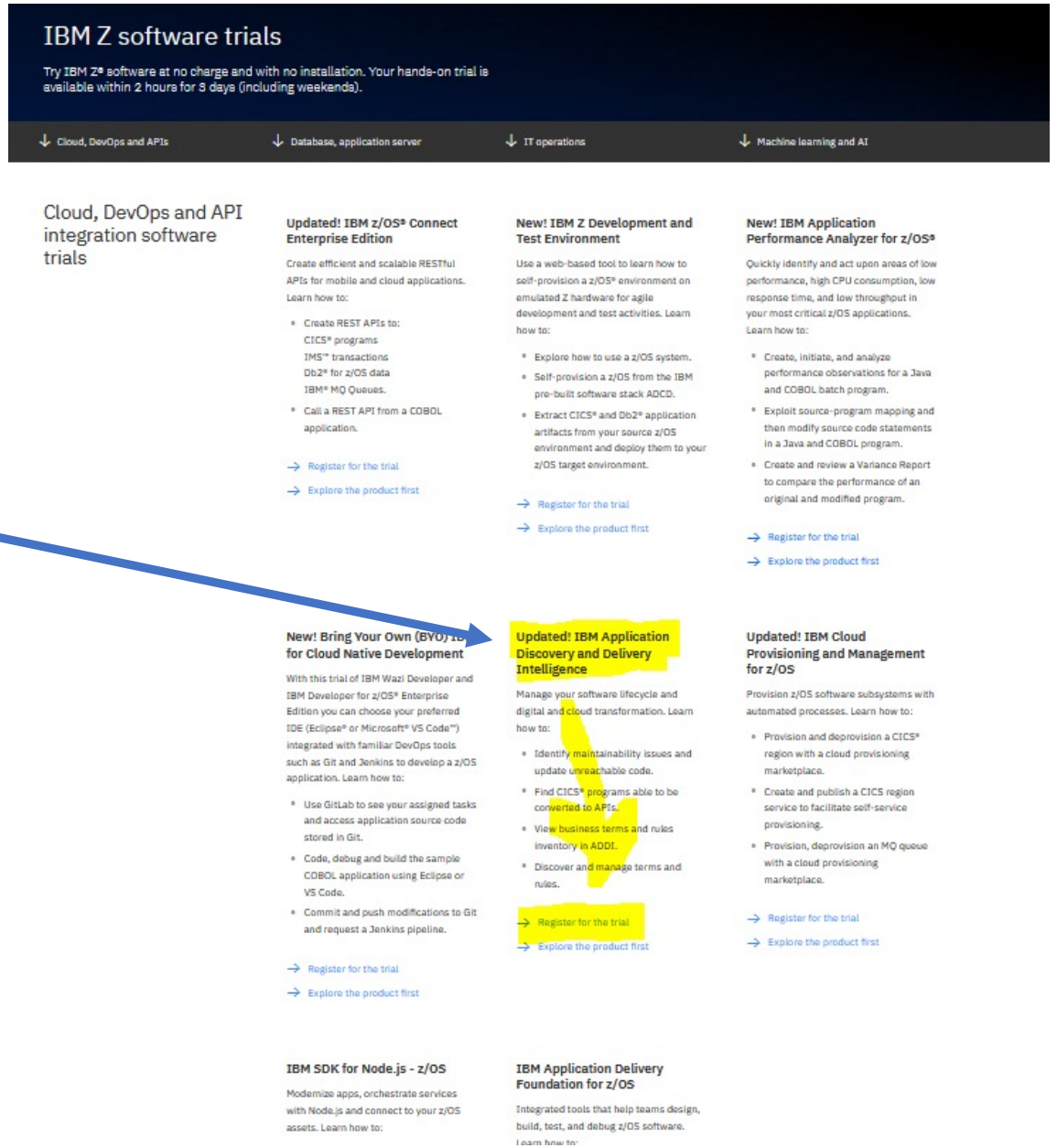
"ADDI helps you focus brain power on the business challenges rather than memorizing code"

ADDI Extensions Manages
Business Keywords, Terms, Rules

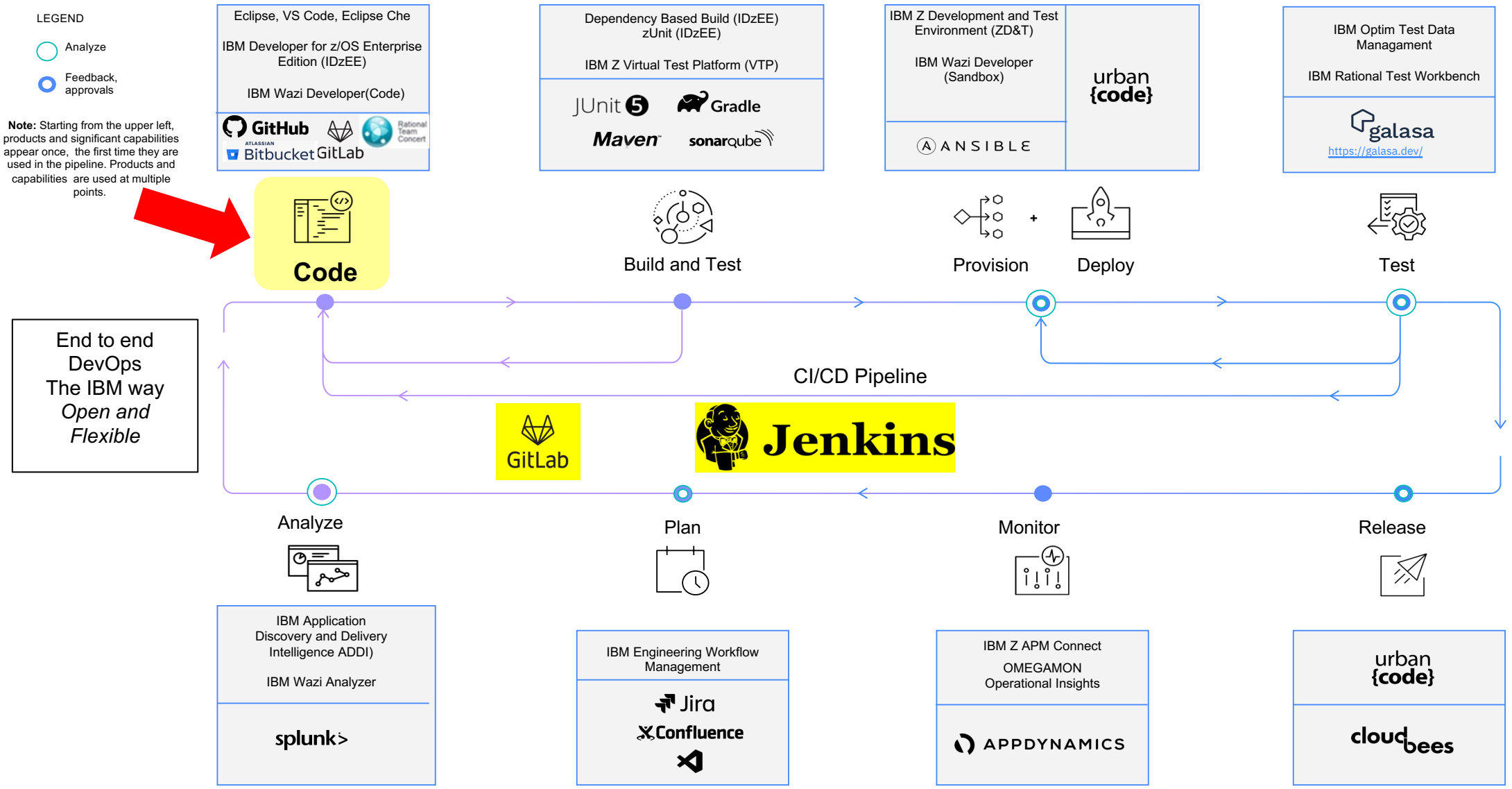
AD = Application Discovery
ADDI = Application Discovery and Delivery Intelligence

If interested in doing the ADDI Lab tomorrow

- You MUST request an ADDI zTrial instance
  1. **Open IBM zTrial registration page** https://www.ibm.com/it-infrastructure/z/resources/trial
  2. Click on **Register for the trial**
  3. Login or Create a new IBMID
  4. Select the Application Discovery and Delivery Intelligence zTrial
  5. Look for emails for when your instance is ready



Z DevOps Pipeline

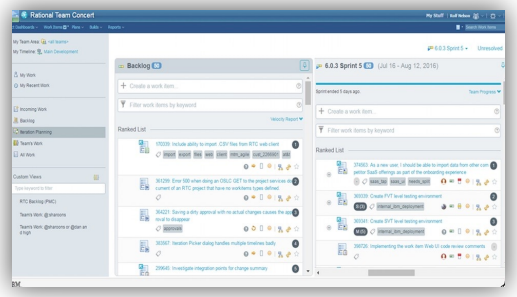


z/OS and subsystems

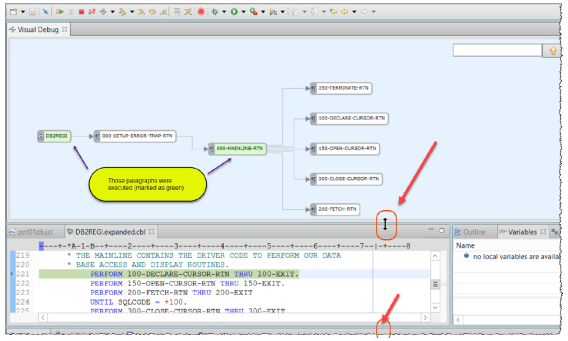
IBM Developer for z/OS Enterprise Edition- the developer's cockpit!

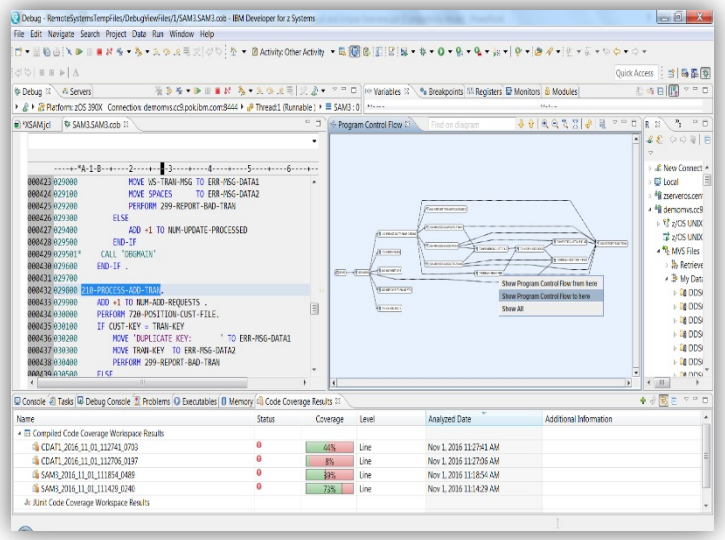
Build and SCM

Integration with Git, RTC and Endevor for Lifecycle and Source Management

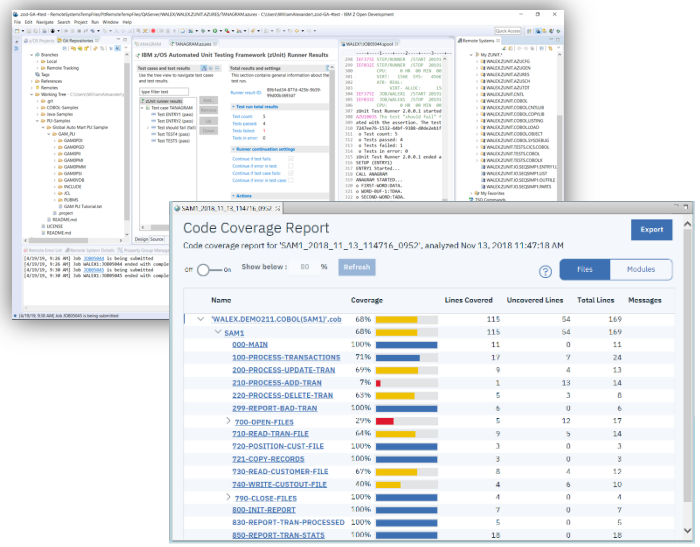


Debug and code coverage

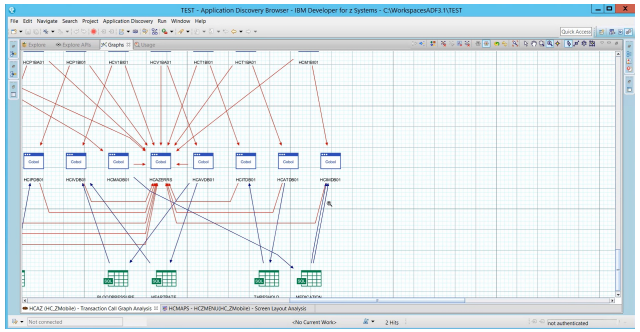




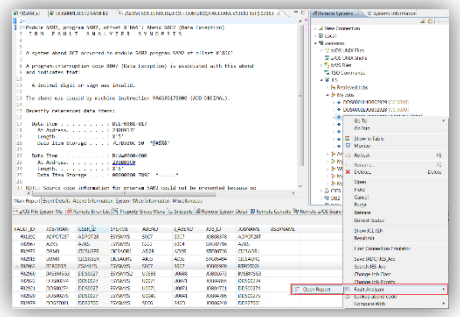
Unit test and code coverage



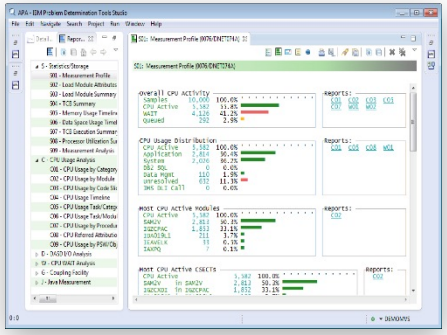
Application Discovery



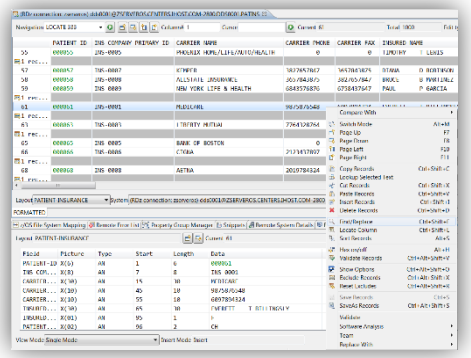
Abend analysis



Performance analysis



File Management



Analyze (program understanding)

Outline view

– Show high level abstraction of source

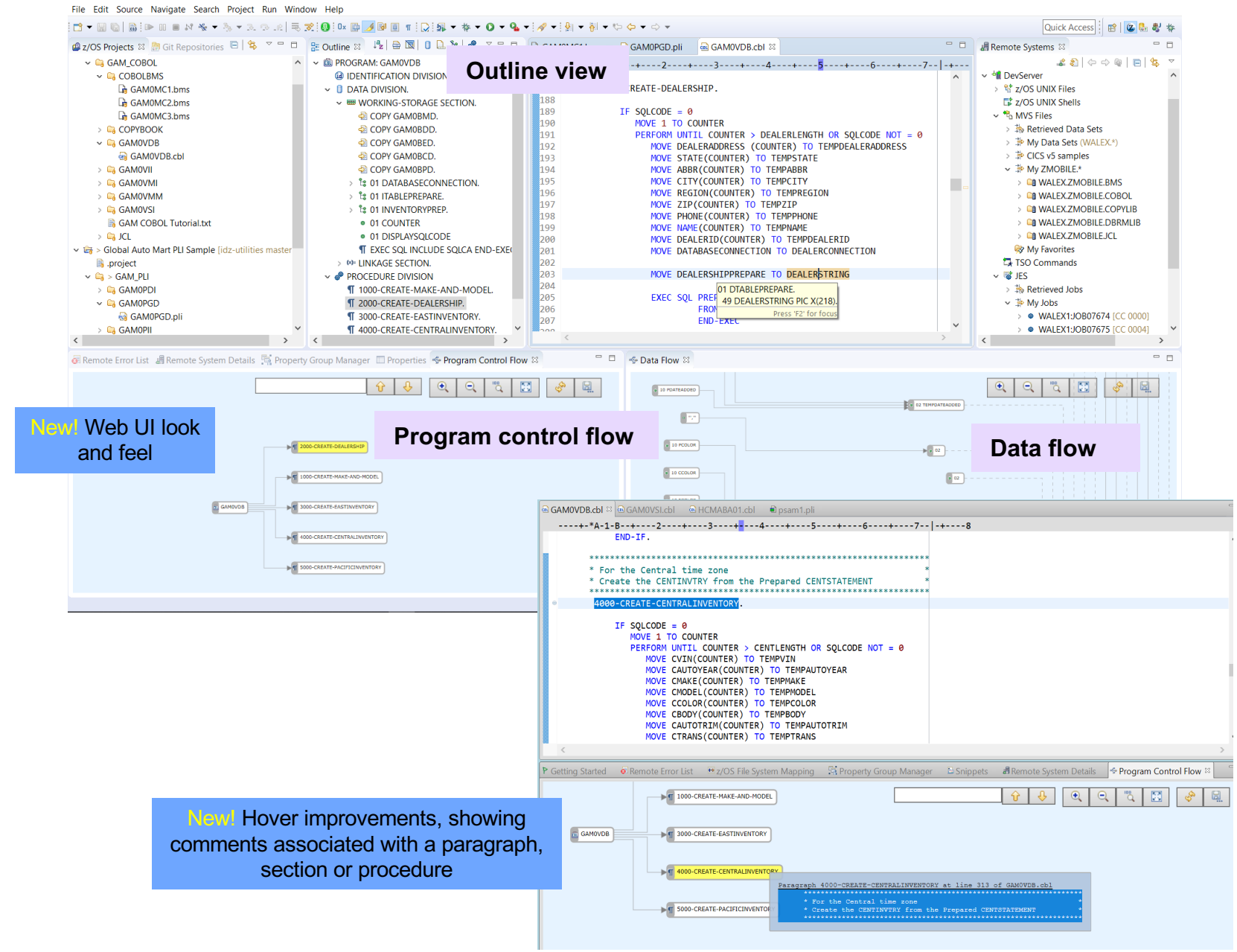
Program control flow

– Graphical view of program logic

Data flow diagram

– Graphical view of data movement throughout the program

Powerful local and remote search capabilities



Analyze (code rules and statistics)

Code rules

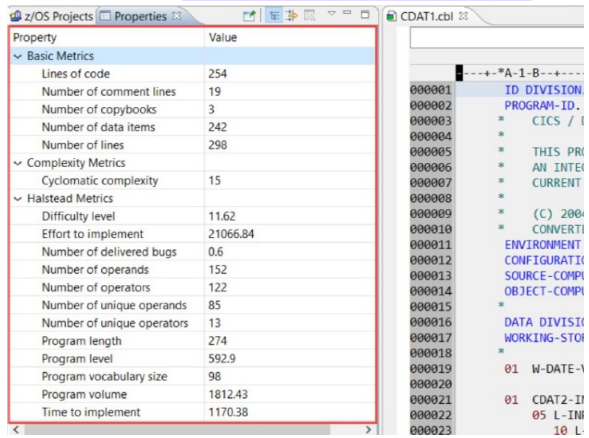
- Electronic desk checking
- Can be run in batch as part of a pipeline
- Customizable

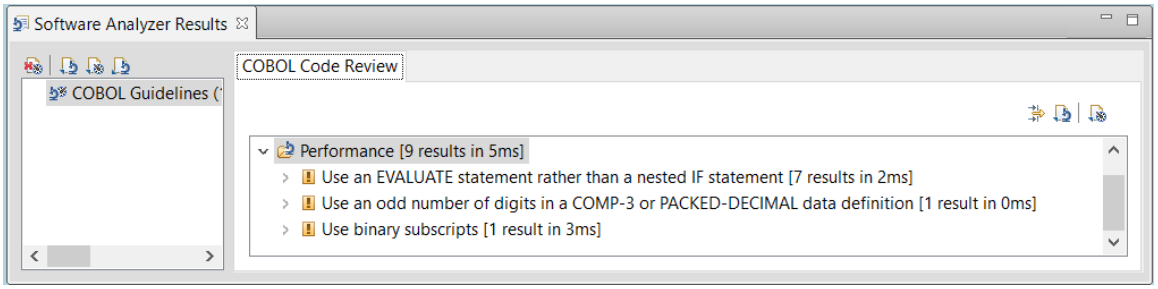
Program level statistics

- Base metrics
- Complexity
- Halstead

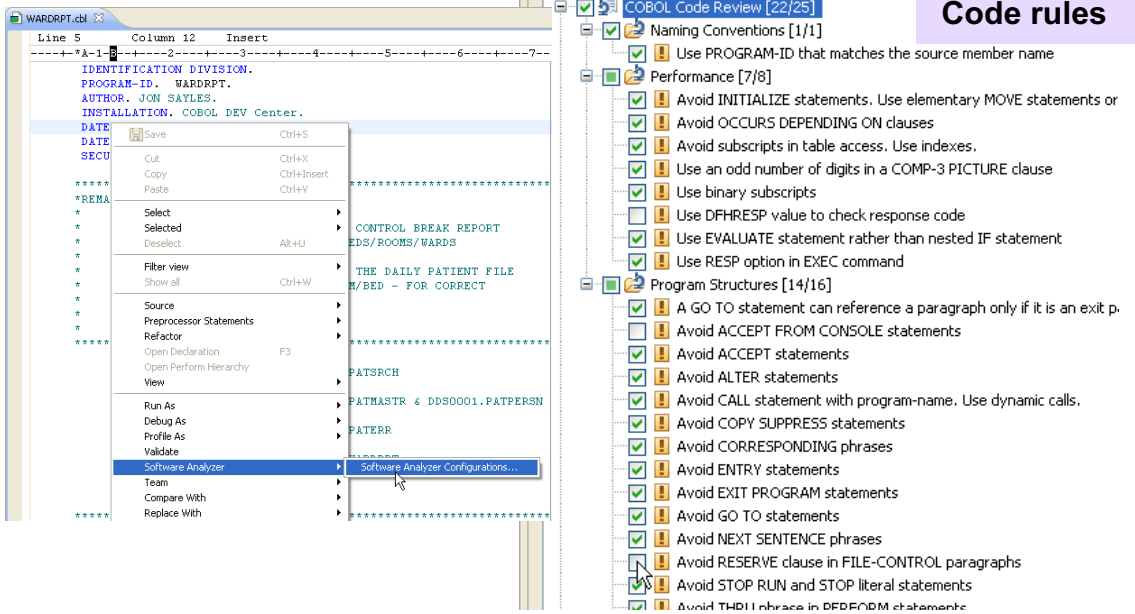
SonarLint integration

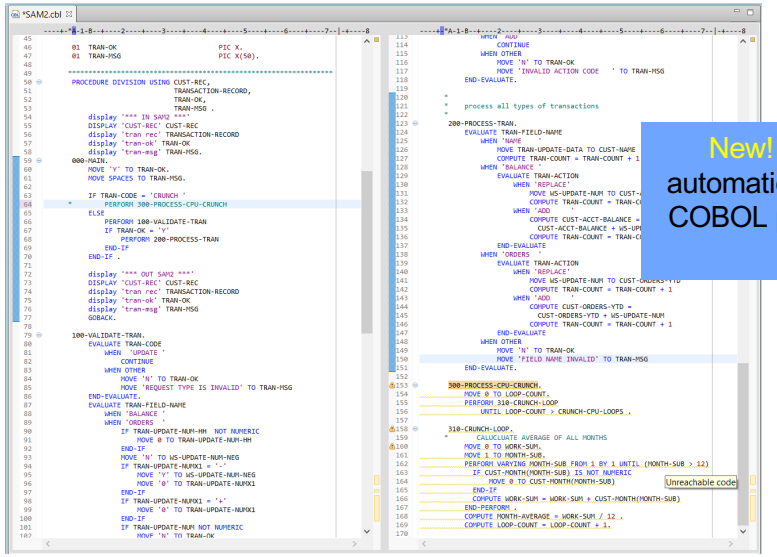
Program statistics





Code rules



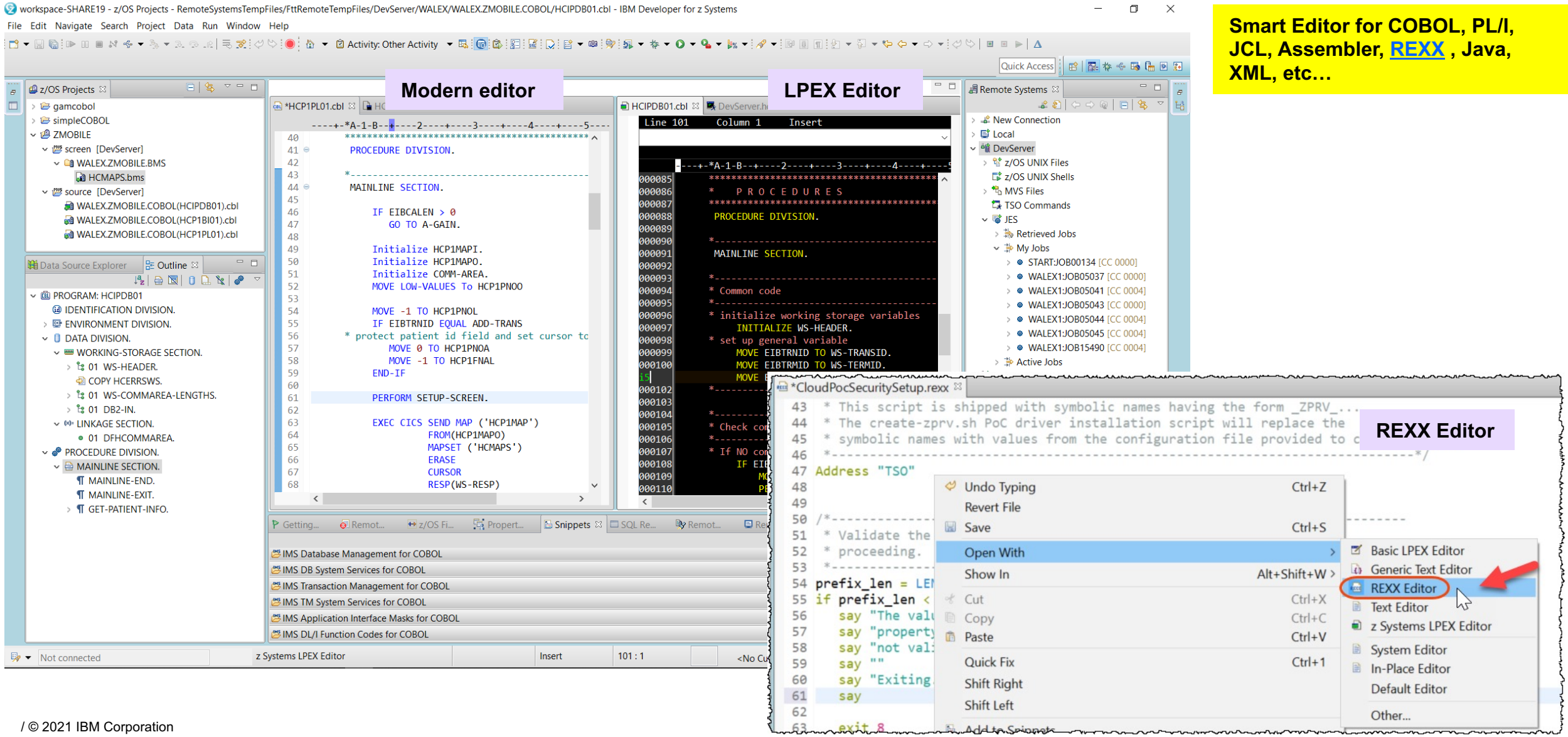


New! Unreachable code automatically identified when a COBOL program is opened for edit

Could be used in a Pipeline (like Jenkins) via Batch execution to enforce standards

Edit

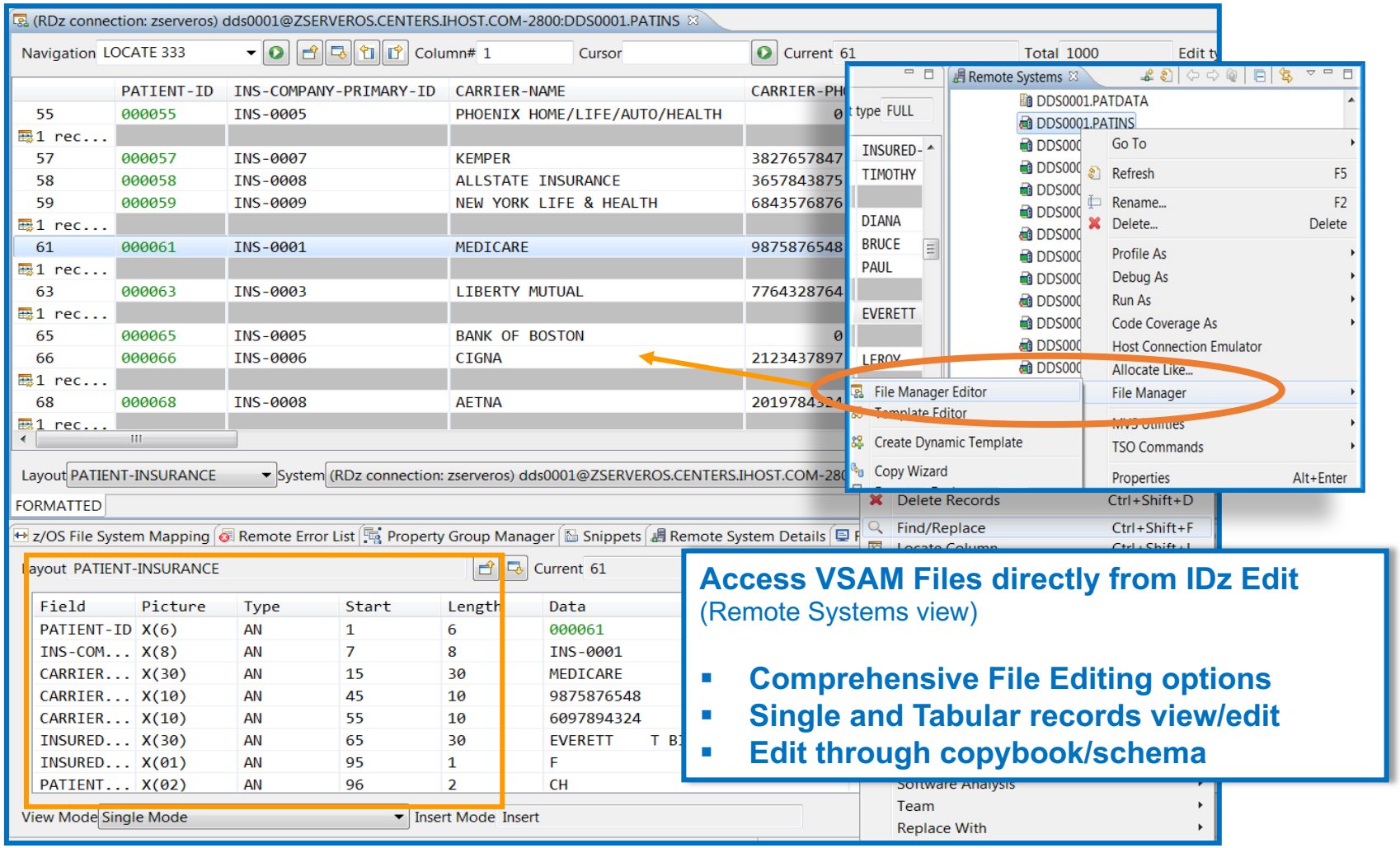
Traditional Eclipse style editors, plus ISPF-like LPEX editor with a command line and prefix area



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



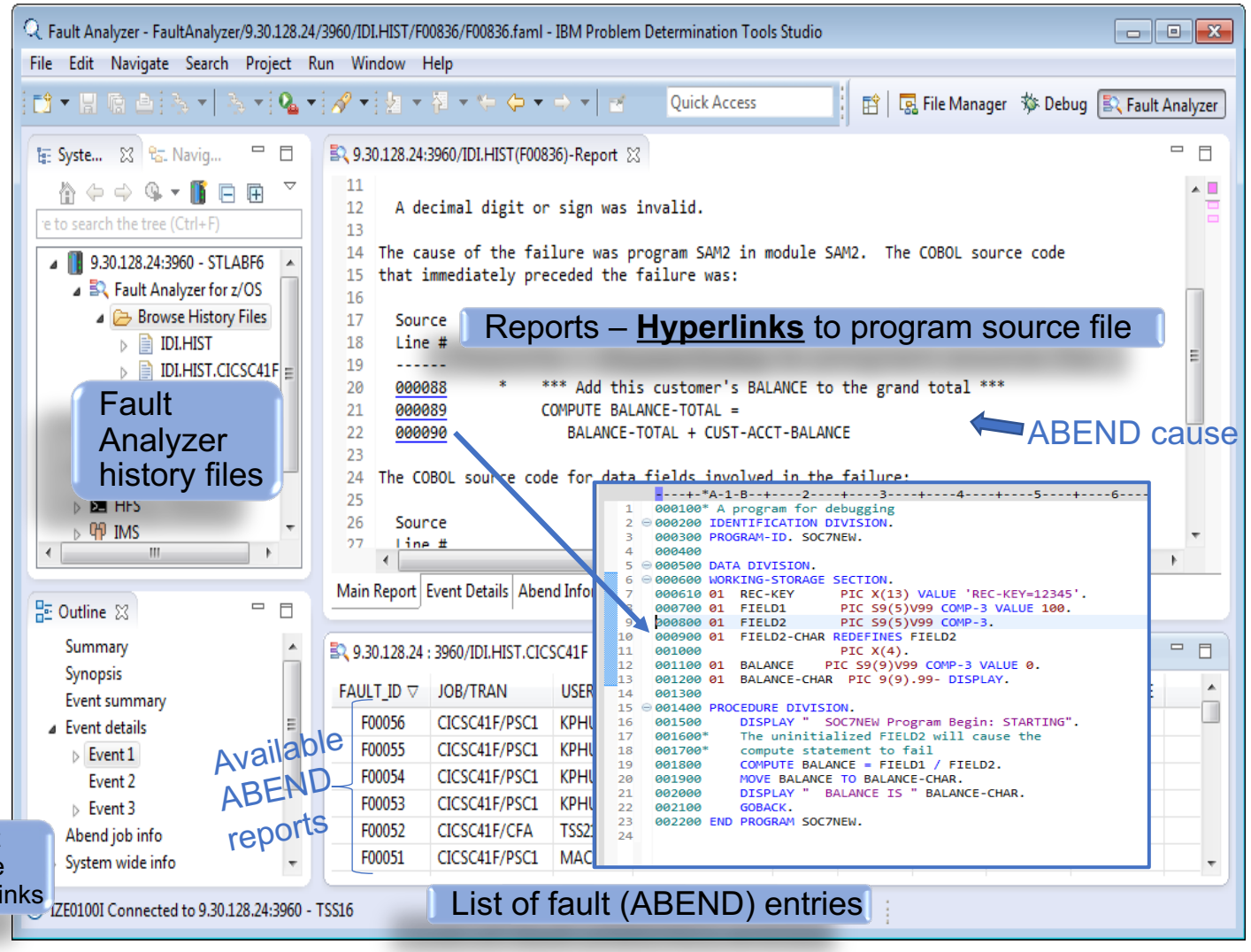
Access VSAM Files directly from IDz Edit
(Remote Systems view)

- **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)

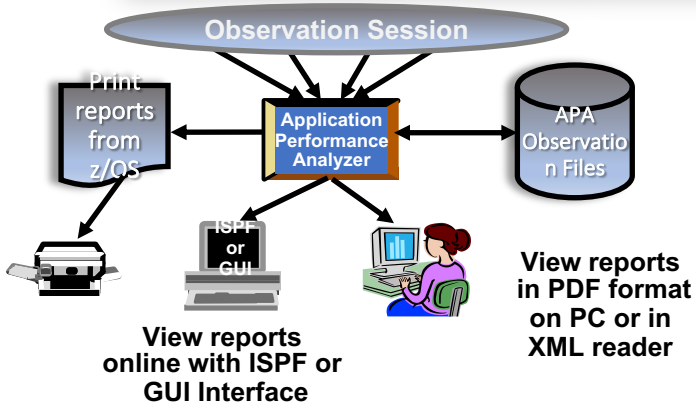
- Automatic program abend capture and reporting
- Program source-level reporting
- Provides a detailed report about program failures to help resolve them quickly
- Enables you to track and manage application failures and fault reports
- Integration with ADFz and 3270-based Interface



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

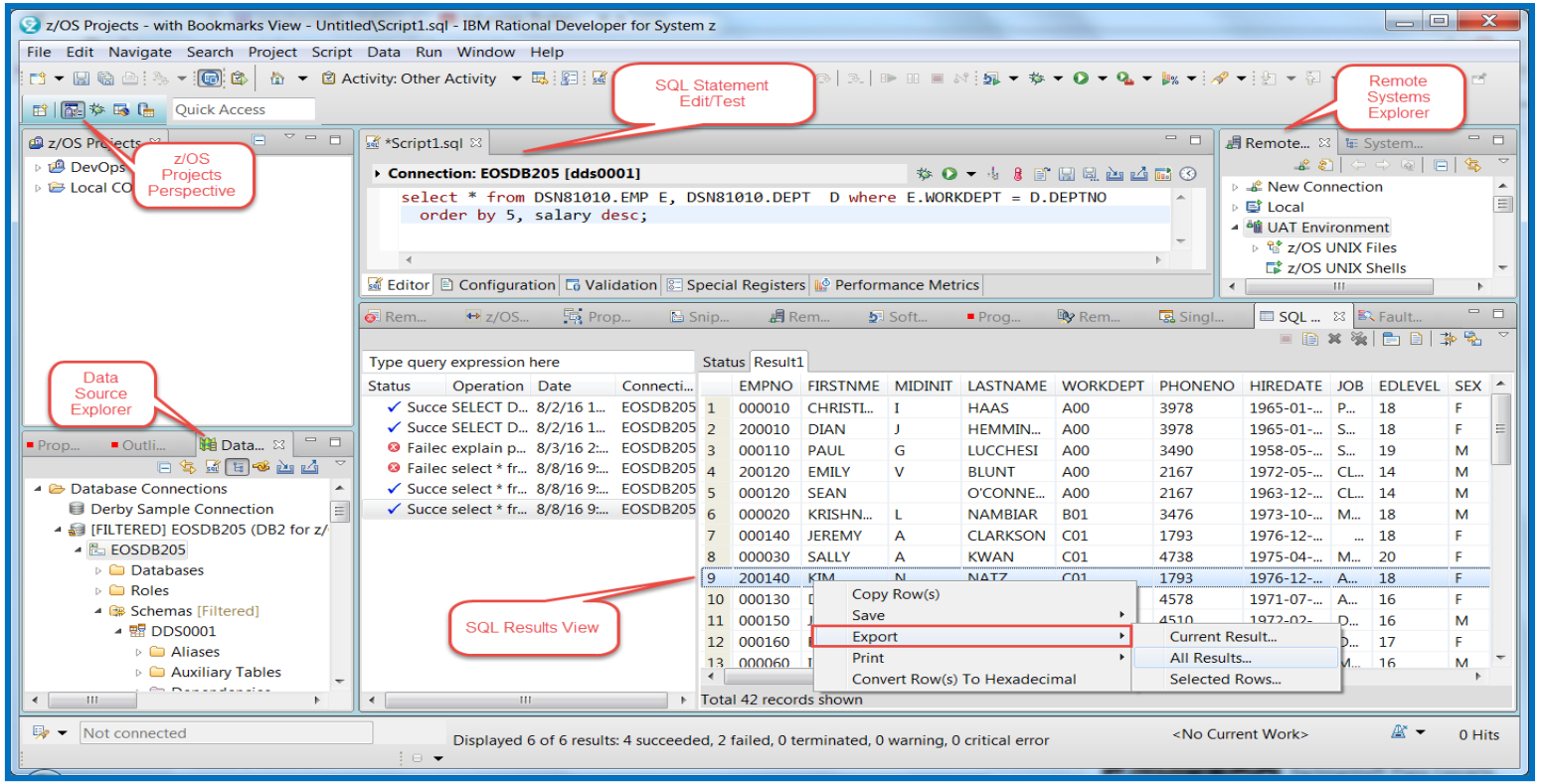
- Specify **number of times** APA monitors a job's performance when **job or program becomes active**
- **Invoke the monitoring capability** from other programs such as IBM Tivoli® OMEGAMON
- **Compare two observation reports** to see the relevant differences, to supplement threshold monitoring
- Assembler, 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

Integrated with ADFz and 3270-based Interface

Data Studio : DB2 Development Tooling

A flexible graphical interface to DB2 and SQL

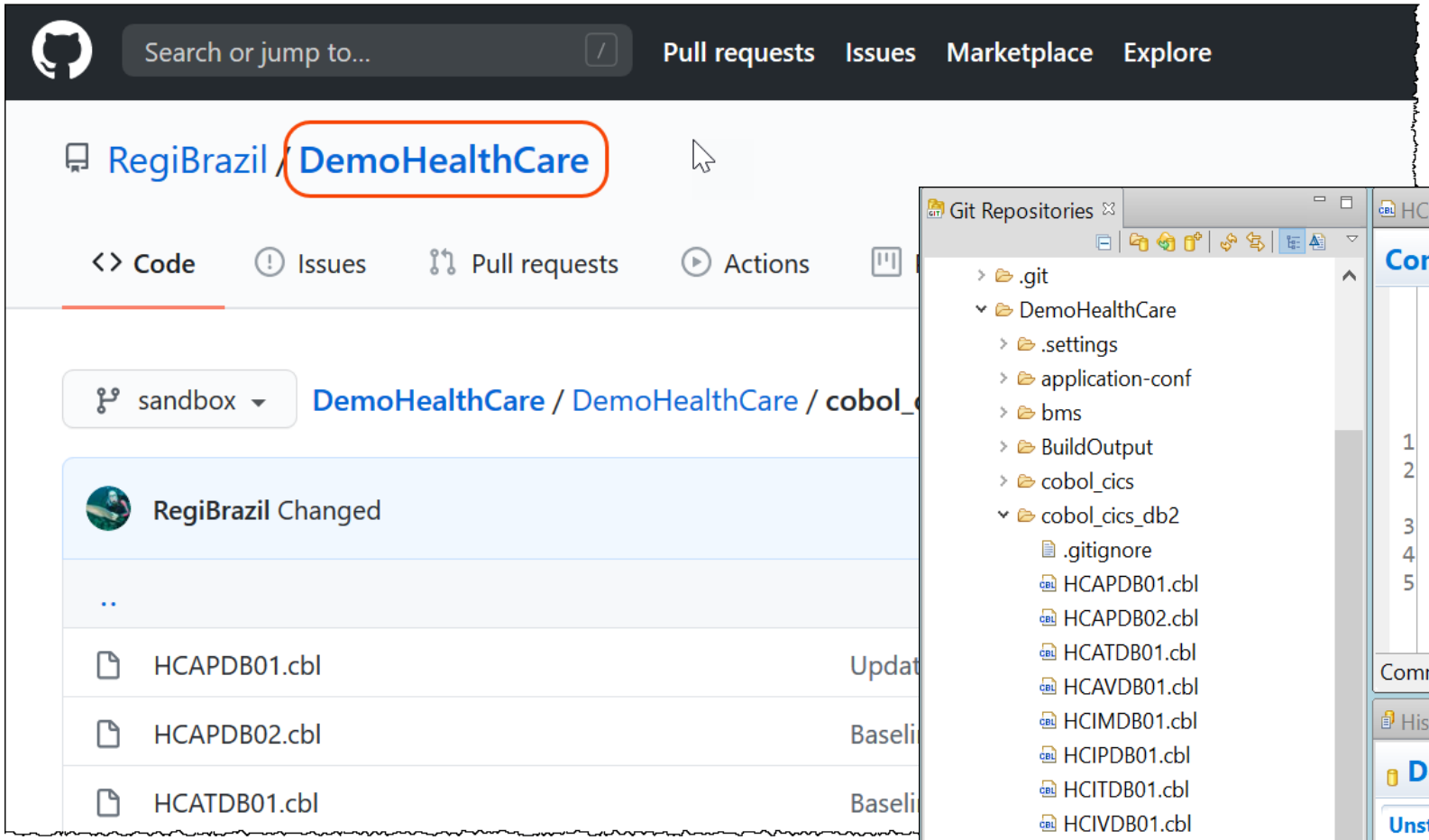
- DB2 Table/View/Index Analysis
- DB2 Data Model and for Test Data Management and manipulation:
  - CRUD
  - Simple Table row/column sub-setting (sampling)
  - Test/Run/Tune SQL directly from COBOL or PL/I programs
  - Run existing SPUFI files (DDL, DML)
- SQL code and Test
  - Code embedded SQL directly 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

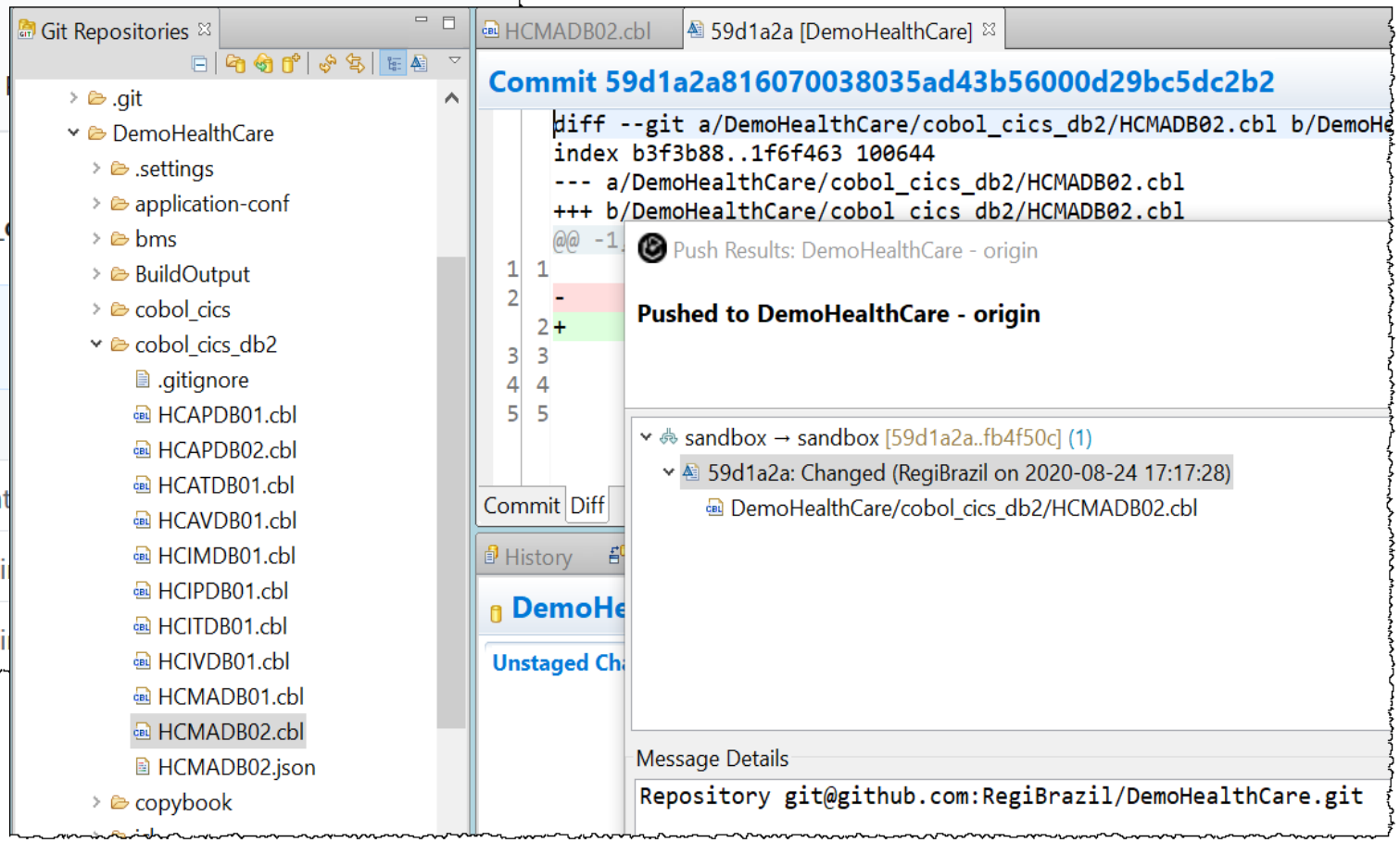


Source Code Management

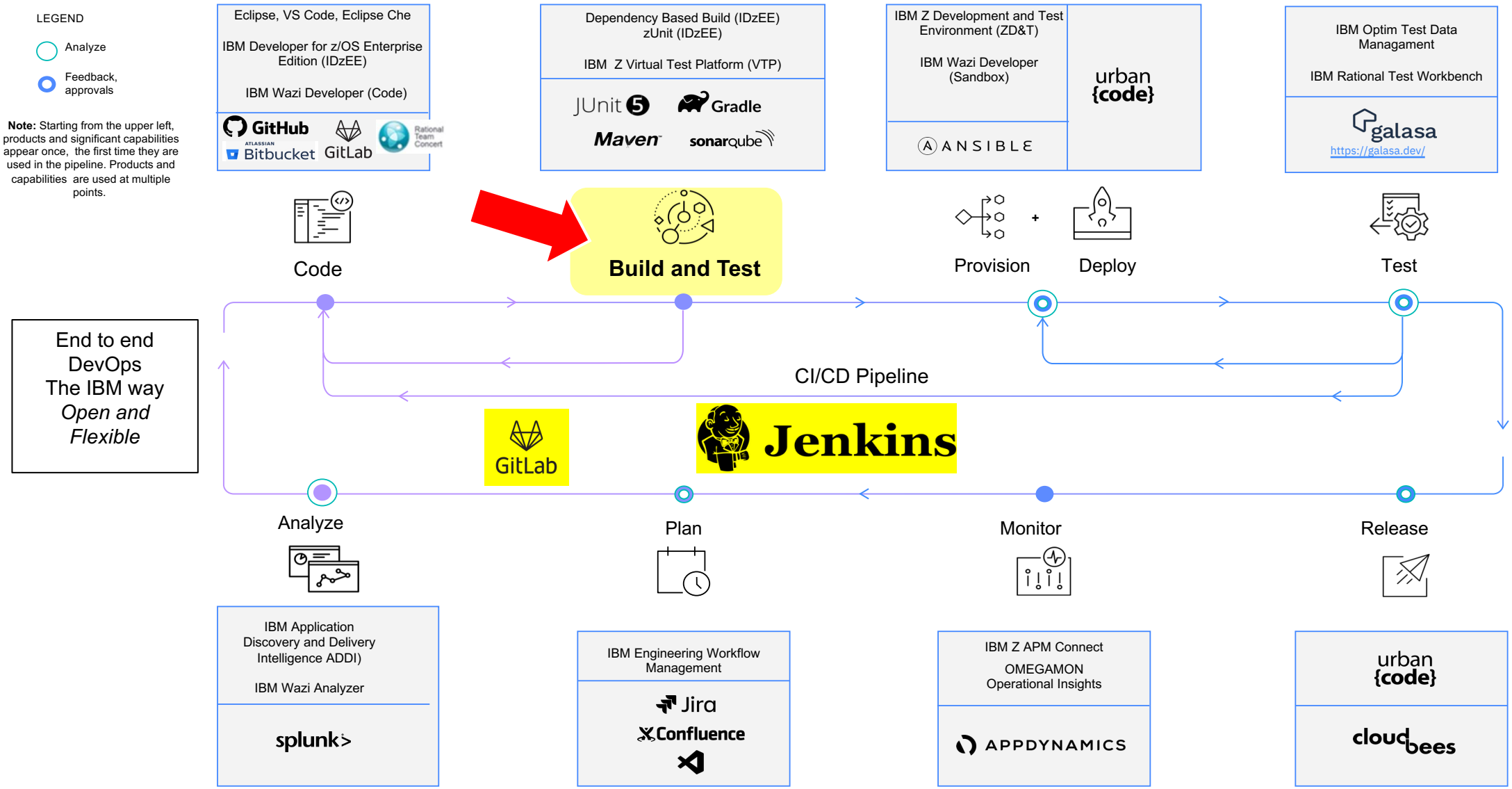
EGit included in IDz

IBM **IDz** offers out-of-the-box **integration** for the IBM Rational Team Concert, CA **Endevor** and IBM SCLM.



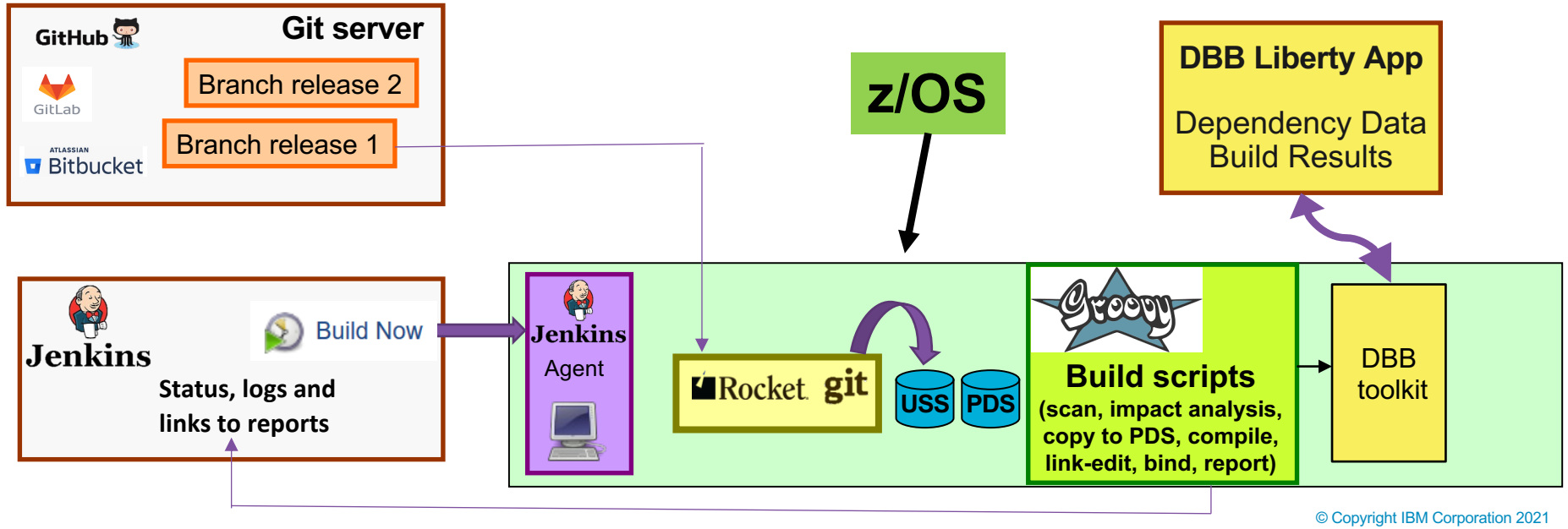


Z DevOps Pipeline



What is IBM Dependency Based Build (DBB)?

- Provides a modern scripting languagebased automation capability that can be used on z/OS.
  - The DBB API can be called by Java based scripting languages such as **Groovy**, JRuby, Jython, Ant, Maven, Python, etc.
- 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.



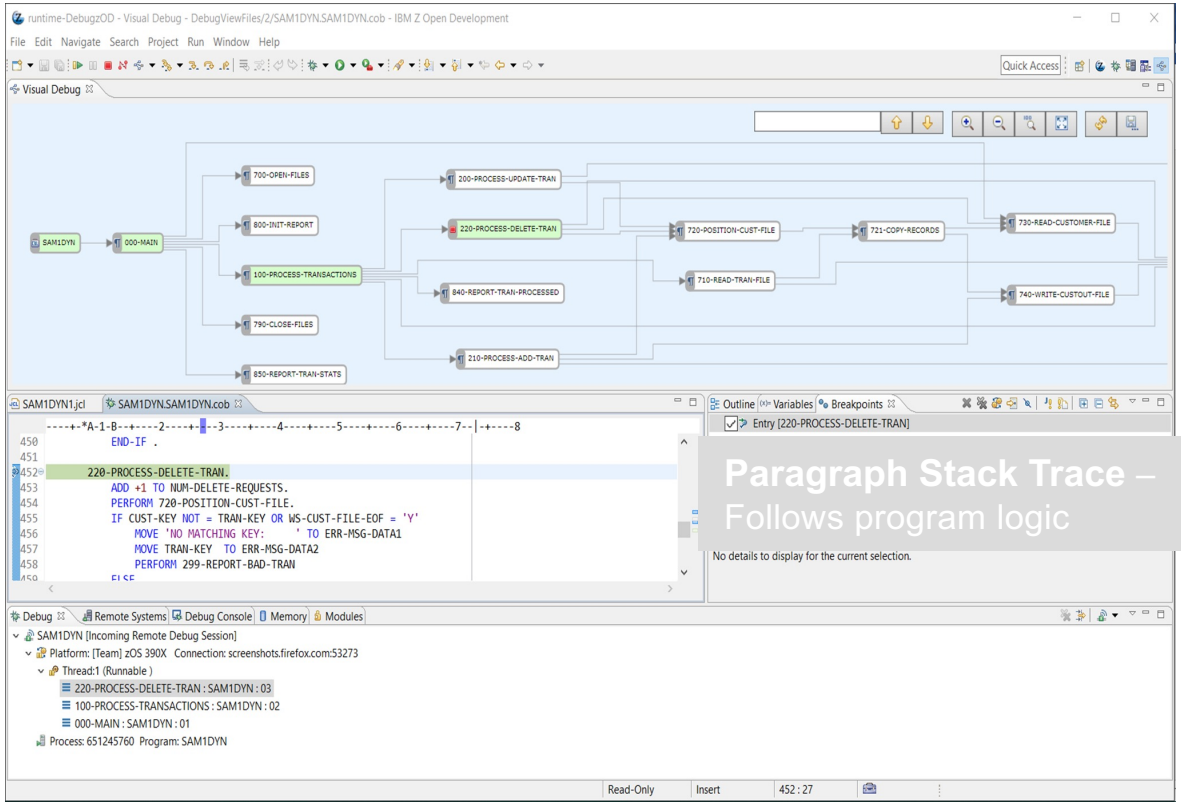
Diagnose (z/OS Debugger)

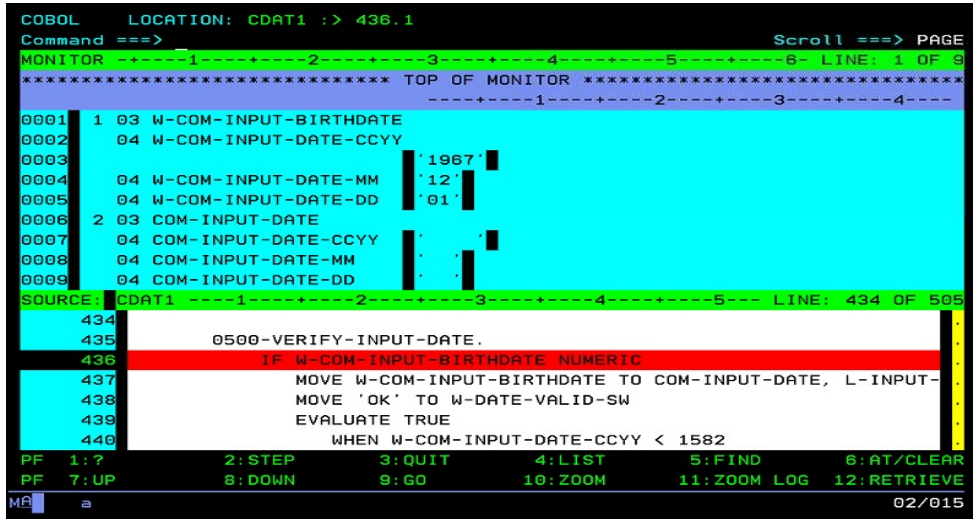
Program testing and analysis

Helps you examine, monitor, and control the execution of application programs on z/OS

With:

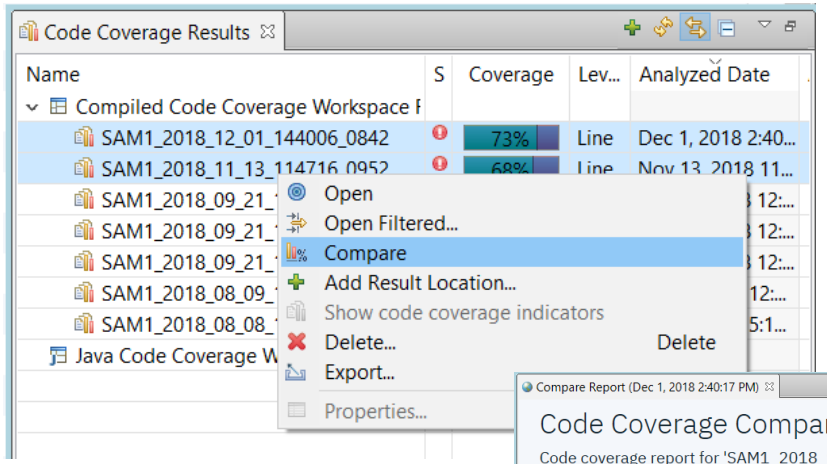
- Visual debug
- Code coverage facilities
- Eclipse GUI
- 3270 UI
  (Debug for z/OS and IDZ-EE)





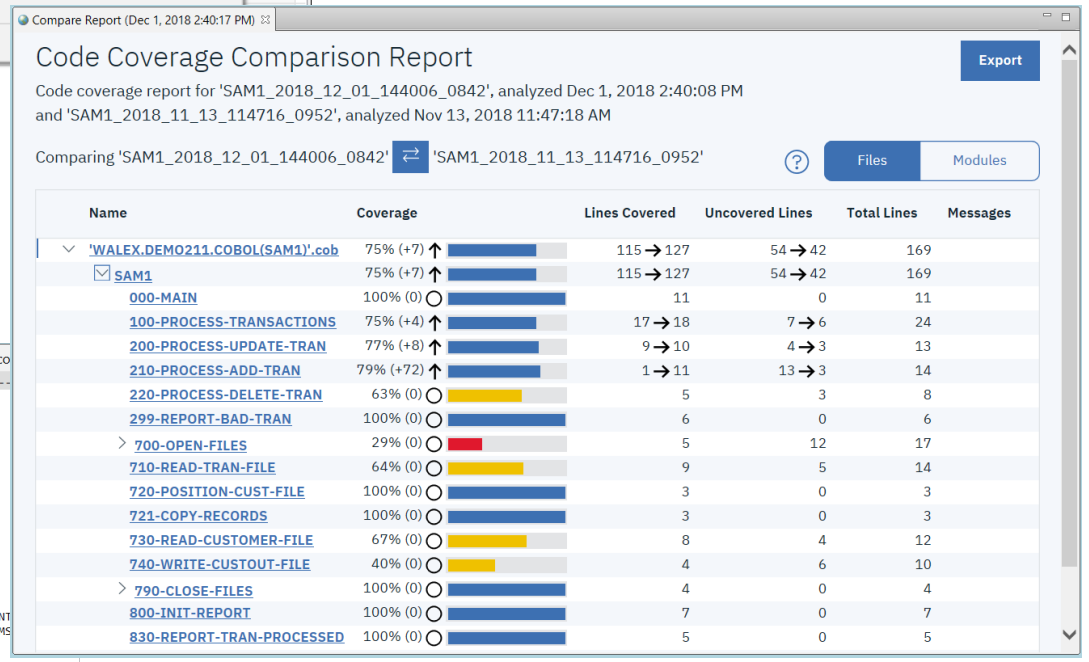
Test (code coverage)

- Tracks lines of code that have been executed during test
- Improves application testing quality
- Focuses testing resource usage
- Numerous reports and options to assess testing and trends
  - Source view
  - Summary
  - Comparisons
  - Merging
- Supports: Batch, CICS and IMS
- Integration with ADDI and SonarQube
- Run in batch as part of your DevOps pipeline, with RESTful APIs for Debug profile creation

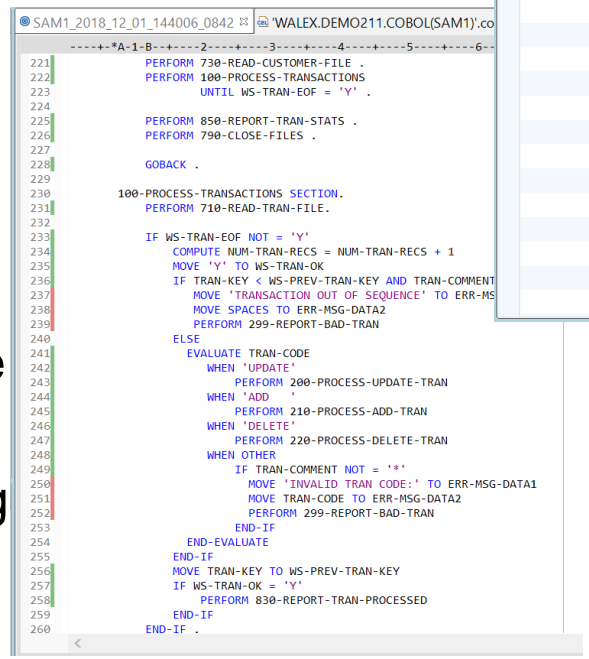


Compare results

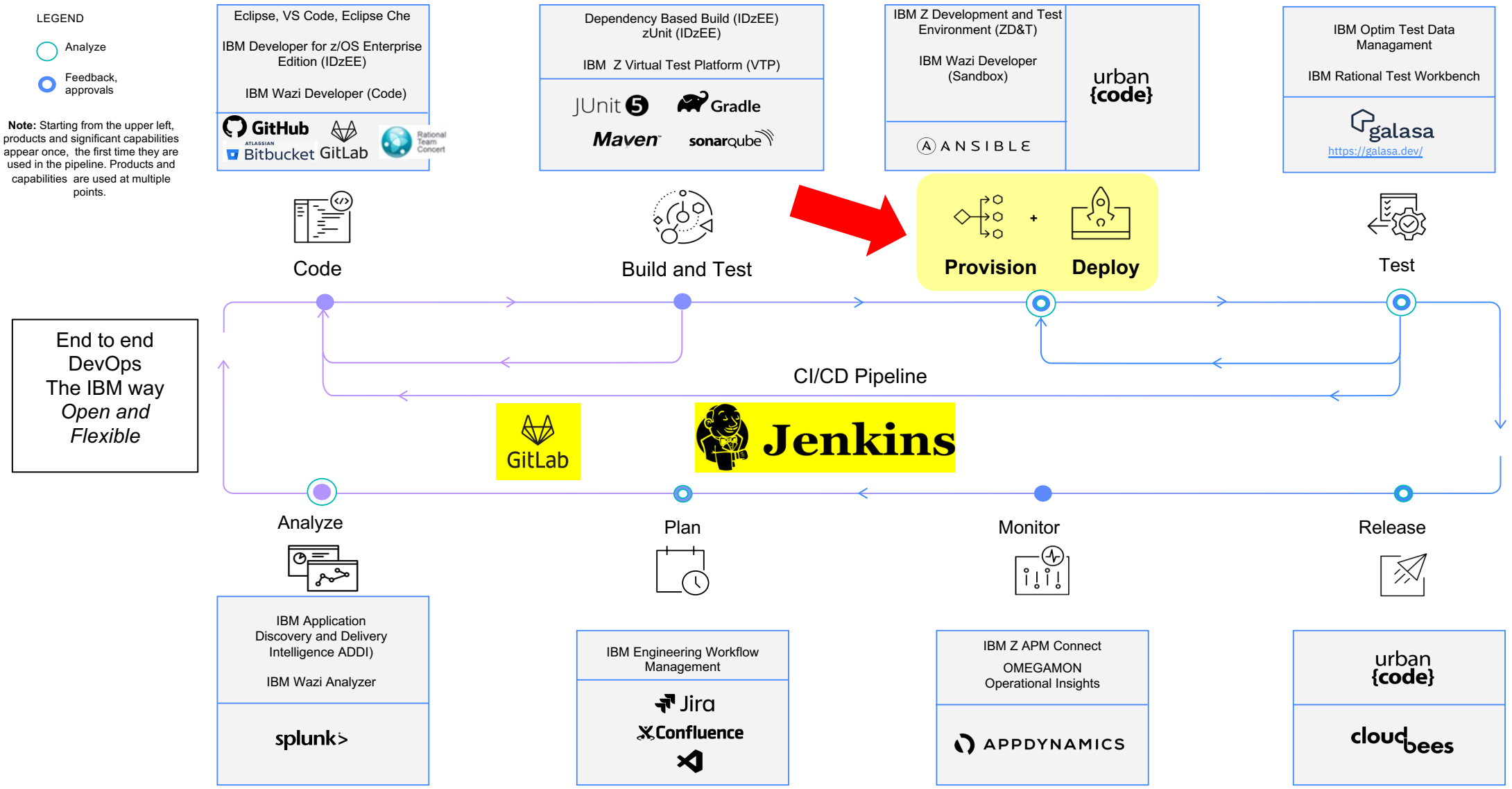
New! Web UI look and feel with easier filtering, grouping and comparisons, plus PDF output (14.1.5)



Coverage



Z DevOps Pipeline

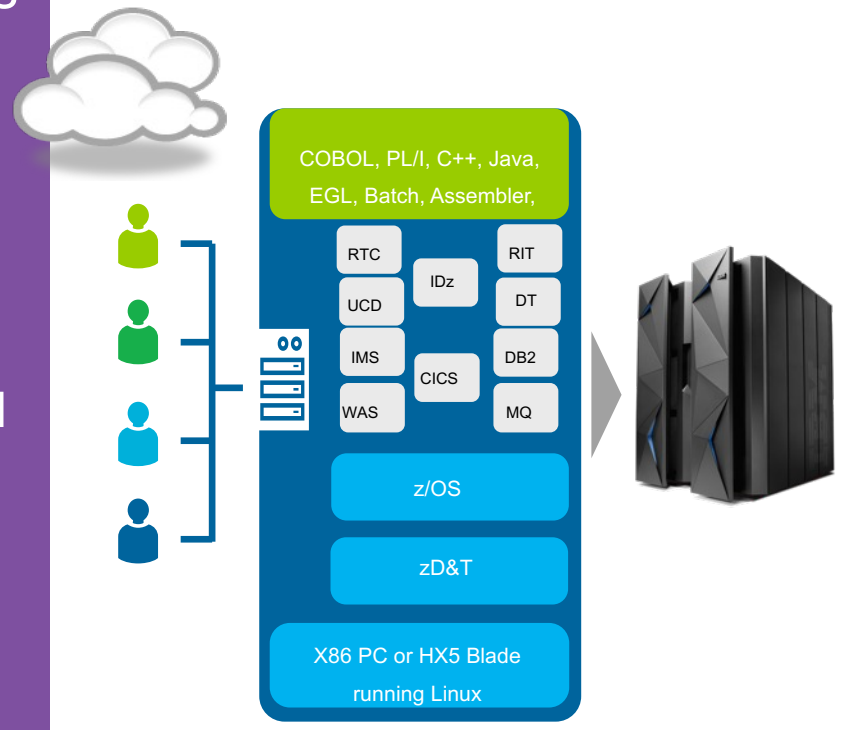


Provisioning

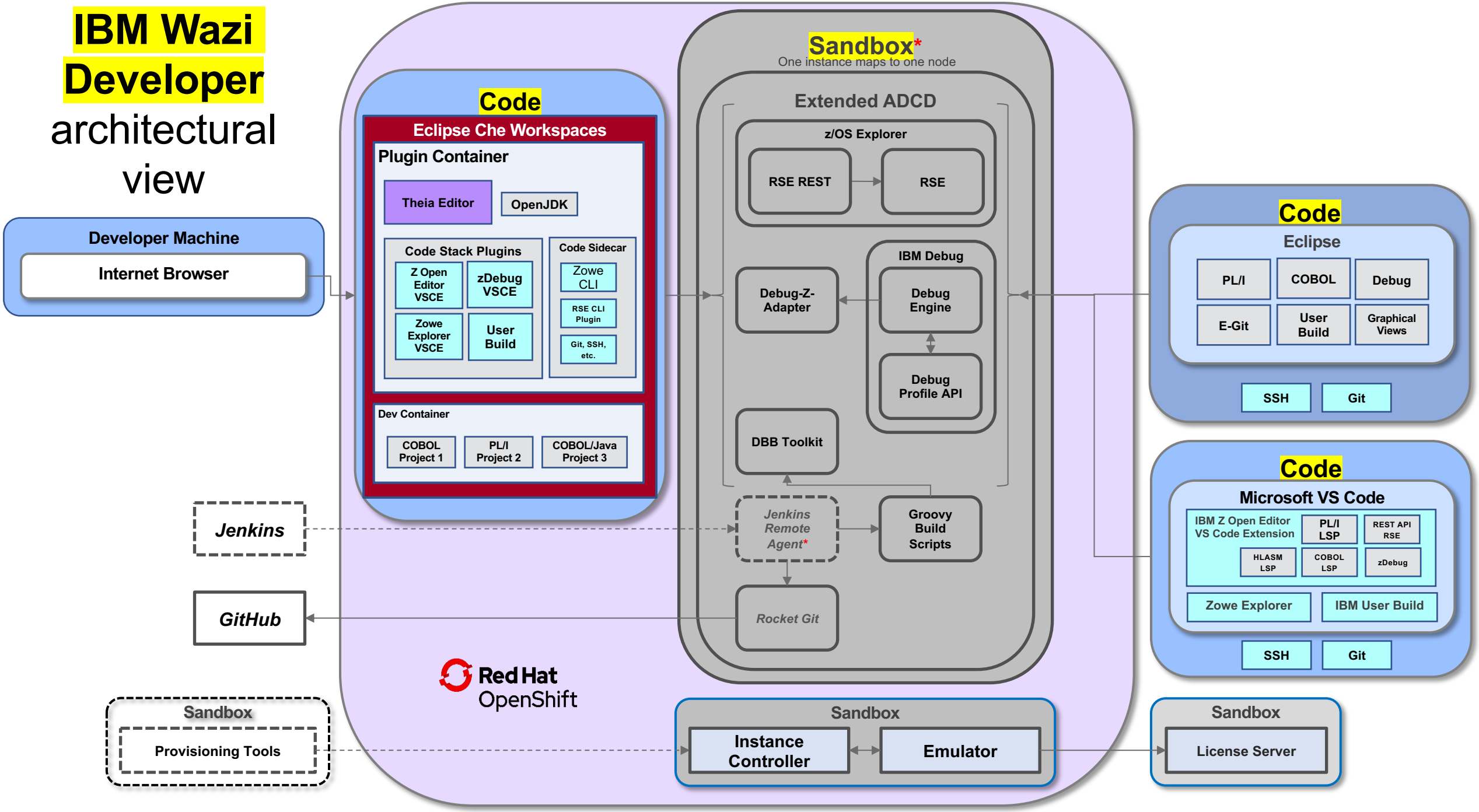
—Shift left with **Z Development and Test Environment**

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 IBM Cloud)
- Latest z/OS 2.4 software and middleware
- ***Developer autonomy just like distributed, web and mobile developers!***

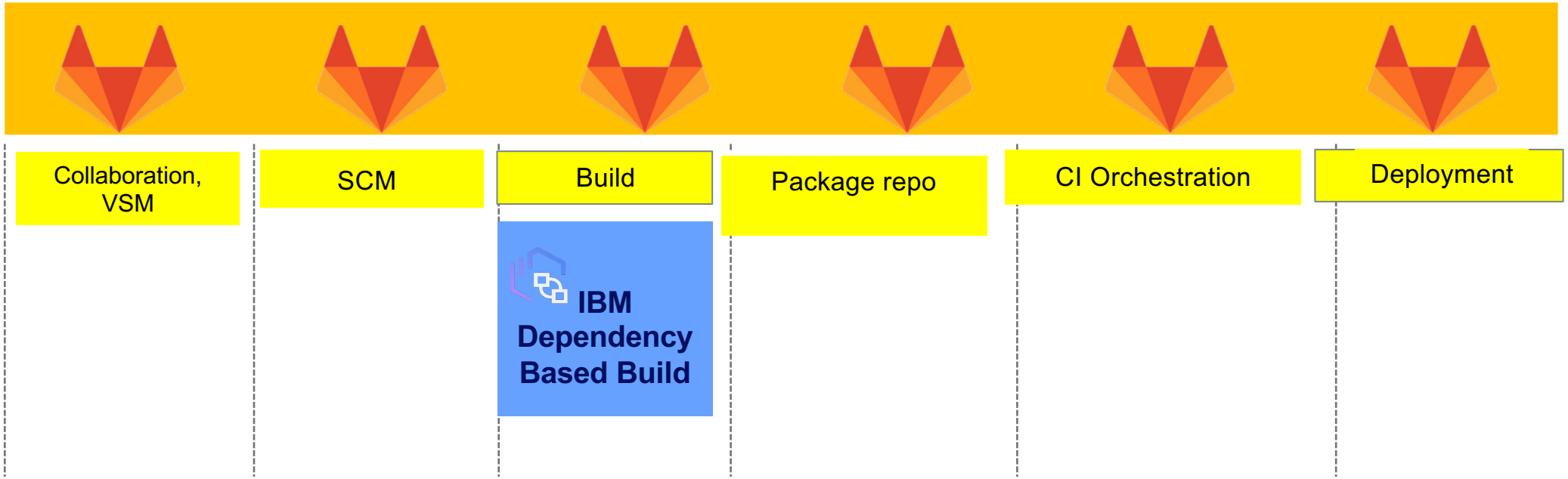


IBM Wazi Developer architectural view



DevOps mainframe pipeline with **GitLab Ultimate for z/OS**

One solution provides: Project Management, SCM, CI/CD, issue tracking, container registries, logging, dependency scanning and license management. Many IBM, 3rd party and open source tools integrate.



Extend GitLab with Specialists

Jira

GitHub

Bitbucket

JFrog Artifactory

Jenkins

urban{code} Deploy

GALAXY

IBM Rational Test

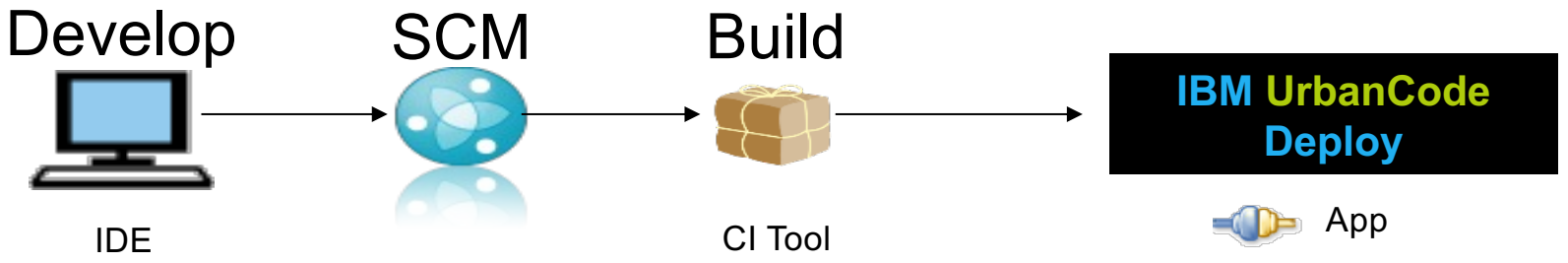
urban{code}

Velocity (VSM, data, pipeline orchestration)

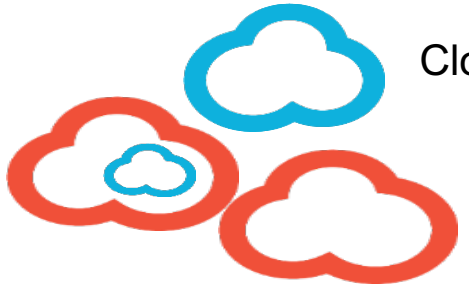
UrbanCode for Deployment automation (UCD)

urban{code}

FROM MAINFRAME TO I-SERIES TO DISTRIBUTED TO CLOUD-NATIVE



Mobile Device



Cloud: Public / Private / Hybrid

amazon web services

Microsoft Azure

Traditional

Mainframe

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, IMS, Db2, MQ, across multiple environments

Integrates with Jenkins

Manages artifact versions via an in-built repository called codestation

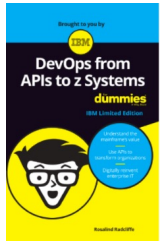
IBM

Jenkins

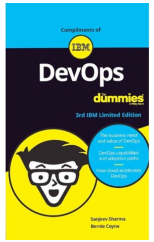
Additional Resources

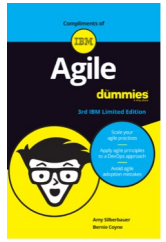
— Mainframe CI/CD with an open toolchain:
- https://www.linkedin.com/pulse/mainframe-cicd-open-toolchain-its-real-spectacular-minaz-merali/

— Mainframe Dev Center:
- Downloads - Mainframe DEV (ibm.github.io)



— 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?

Agenda - Day 1

2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)

3:00 LAB 1 - Working with z/OS using COBOL and DB2

or

LAB7 - IBM DBB with Git, Jenkins and UCD on Z

5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

2:00 zUnit and VTP Overview (Wilbert)

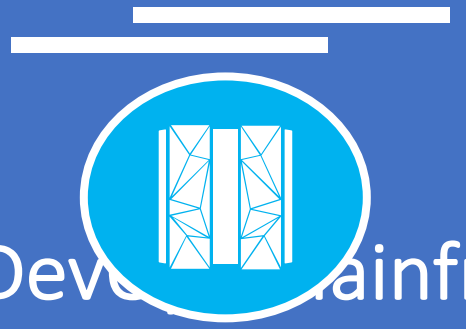
2:15 Demo: Scenario using Z DevOps solutions including **ADDI**, **zUnit**, **Git** and **Jenkins** (David/Regi)

3:00 – 4:50 Choose one Optional lab:

- → LAB 7 - IBM **DBB** with **Git**, **Jenkins** and **UCD** on Z
- → LAB 6B : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Remote Assets**
- → LAB 6C : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Local Assets**
- → LAB 6D : Using **IBM zUnit** to Unit Test a COBOL/DB2 **batch** program
- → 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**)
- → LAB 9 - Using **IBM Z VTP** to test a COBOL /CICS / DB2 transaction
- → LAB 10 - Deploying COBOL/CICS/DB2 application using a **GitLab CI** Pipeline
- → LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- → LAB 12 – Using **ADDI** to find CICS programs to be converted to API's
- → LAB 13 – Running Integration Tests using **Galasa**

5:00 Checkpoint - Comments and Closing Day 2 (David/Ron/Wilbert/Regi)

Lecture Labs



Dev ainframe Tooling - Labs

Labs

Depending on the LAB…

1. As IDE you will use either **IDz** version 14.2.4 or **IDz** V 15.0.1.

2. Use the correct z/OS user id: **`EMPOT01`** or **`EMPOT05`** or **`IBMUSER`** indicated on the exercise.

3. Each time that you see the symbol ⏭ it means that you have to "*do something*" on your computer – not merely read the explanations.

If you could not complete the lab don't get frustrated… nothing here will go to production ☺

Topology used on the labs

zD&T on Cloud

z/OS

DBB

Jenkins Agent

UCD-Agent

DBB App server

Jenkins

UCD Server urban{code}

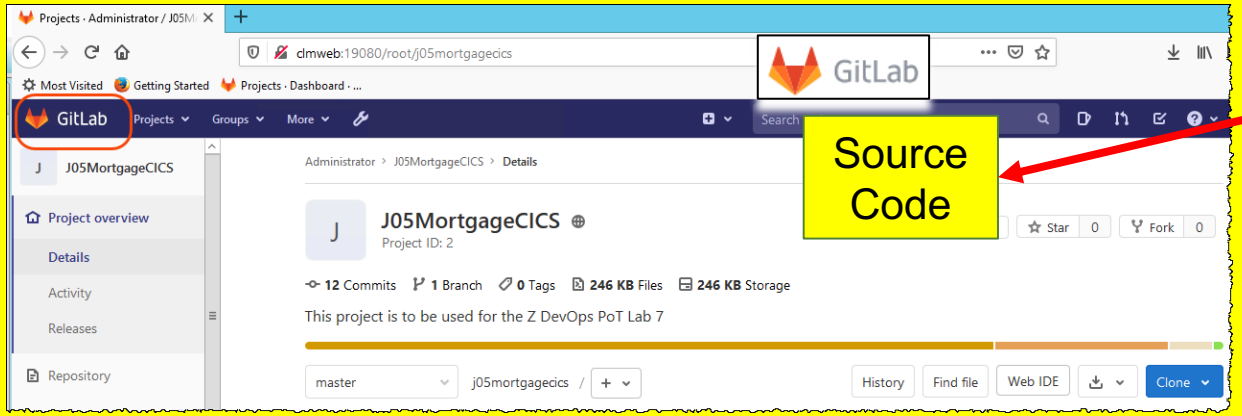
Executables

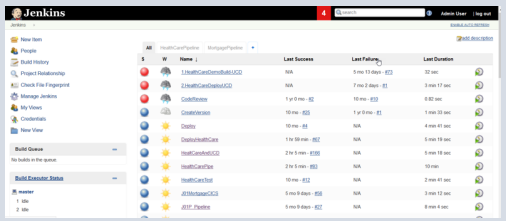
Windows

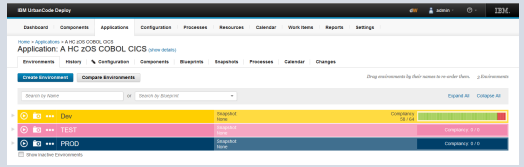
Developer using ADFz

Git

Source Code







Linux

Lab 1: Working with mainframe using COBOL and DB2

Duration: 90 Minutes

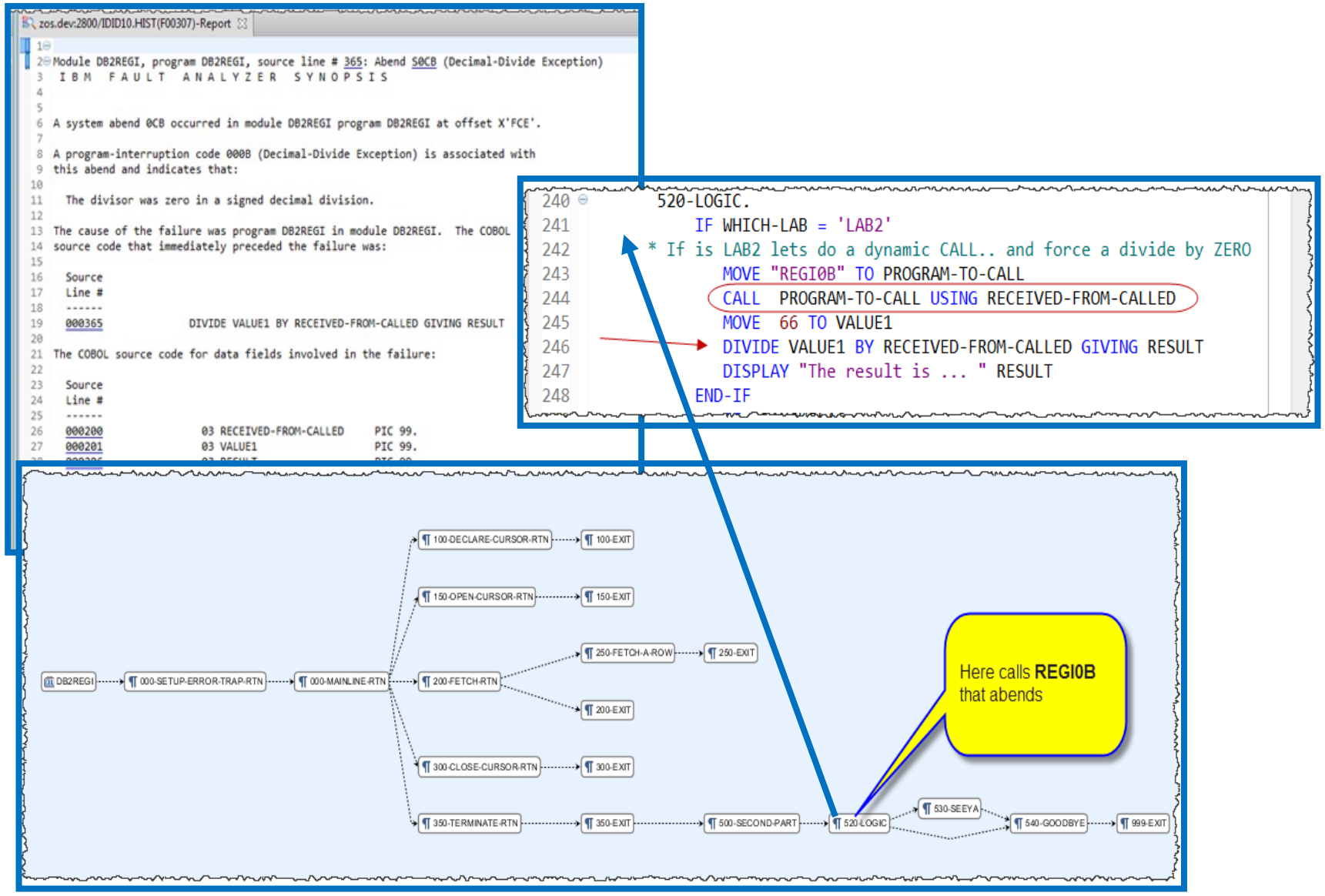
Objective

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.**
   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



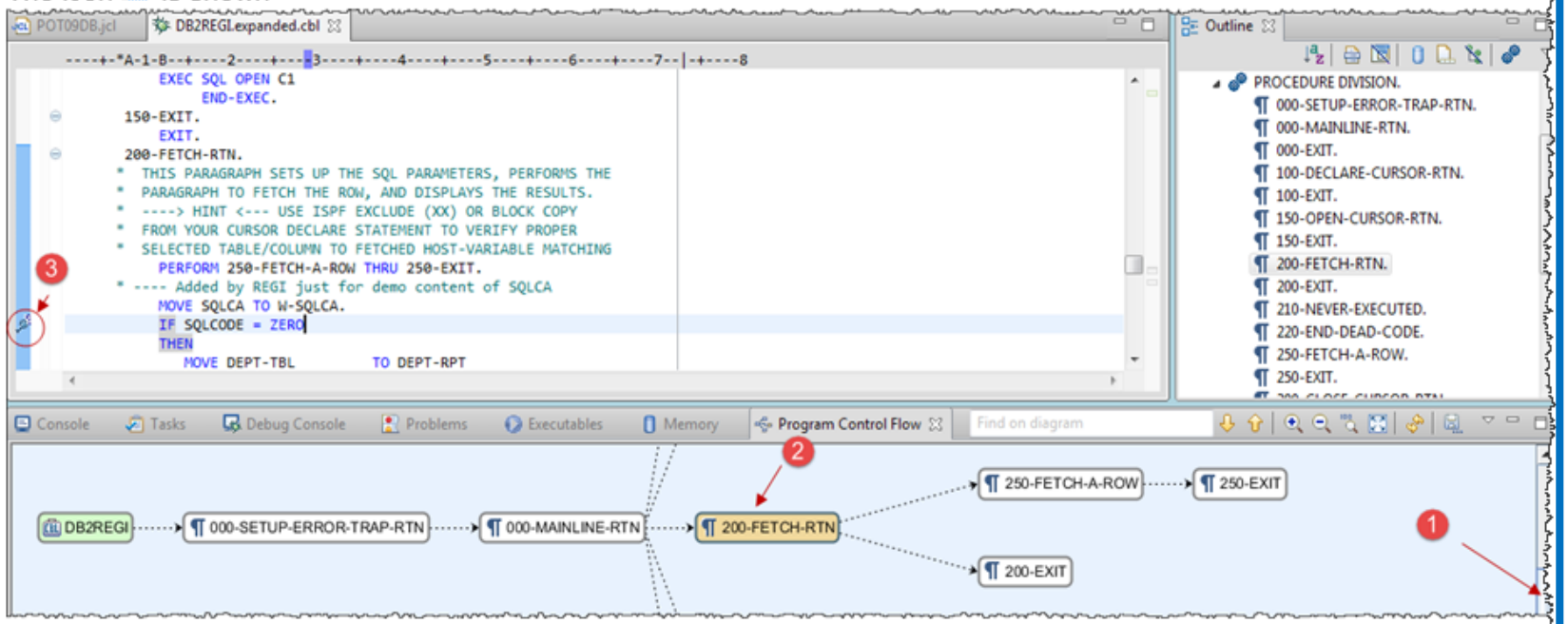
Lab 1: Don’t Miss… The Debug… Page 22

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.

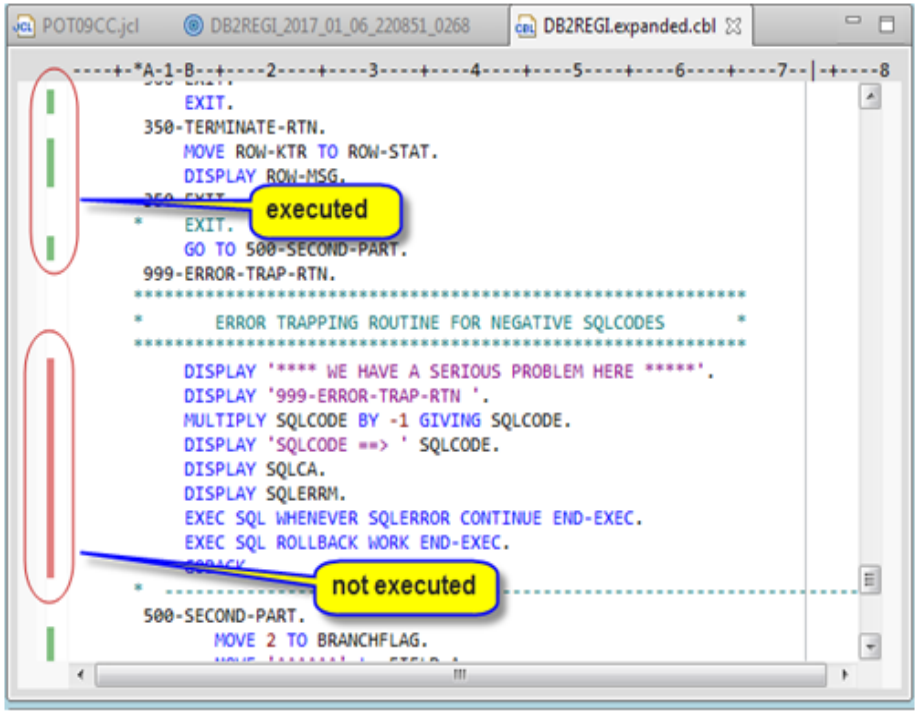
The icon is shown



Lab 1: Don’t Miss… Code Coverage …Page 48

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

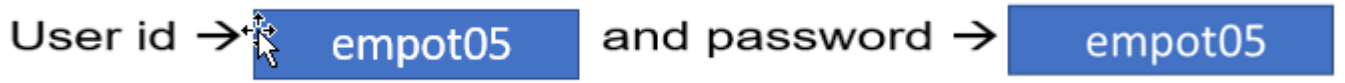
Scroll down and you will see the lines executed in **green** and the lines not executed in **Red**.



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 GitLab, 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/link 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 - Day 1

2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)

3:00 LAB 1 - Working with z/OS using COBOL and DB2

or

LAB7 - IBM DBB with Git, Jenkins and UCD on Z

5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

2:00 zUnit and VTP Overview (Wilbert)

2:15 Demo: Scenario using Z DevOps solutions including **ADDI**, **zUnit**, **Git** and **Jenkins** (David/Regi)

3:00 – 4:50 Choose one Optional lab:

- → LAB 7 - IBM **DBB** with **Git**, **Jenkins** and **UCD** on Z
- → LAB 6B : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Remote Assets**
- → LAB 6C : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Local Assets**
- → LAB 6D : Using **IBM zUnit** to Unit Test a COBOL/DB2 **batch** program
- → 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**)
- → LAB 9 - Using **IBM Z VTP** to test a COBOL /CICS / DB2 transaction
- → LAB 10 - Deploying COBOL/CICS/DB2 application using a **GitLab CI** Pipeline
- → LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- → LAB 12 – Using **ADDI** to find CICS programs to be converted to API's
- → LAB 13 – Running Integration Tests using **Galasa**

5:00 Checkpoint - Comments and Closing Day 2 (David/Ron/Wilbert/Regi)

Lecture Labs

Agenda - Day 1

2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)

3:00 LAB 1 - Working with z/OS using COBOL and DB2

or

LAB7 - IBM DBB with Git, Jenkins and UCD on Z

5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

2:00 zUnit and VTP Overview (Wilbert)

2:15 Demo: Scenario using Z DevOps solutions including **ADDI**, **zUnit**, **Git** and **Jenkins** (David/Regi)

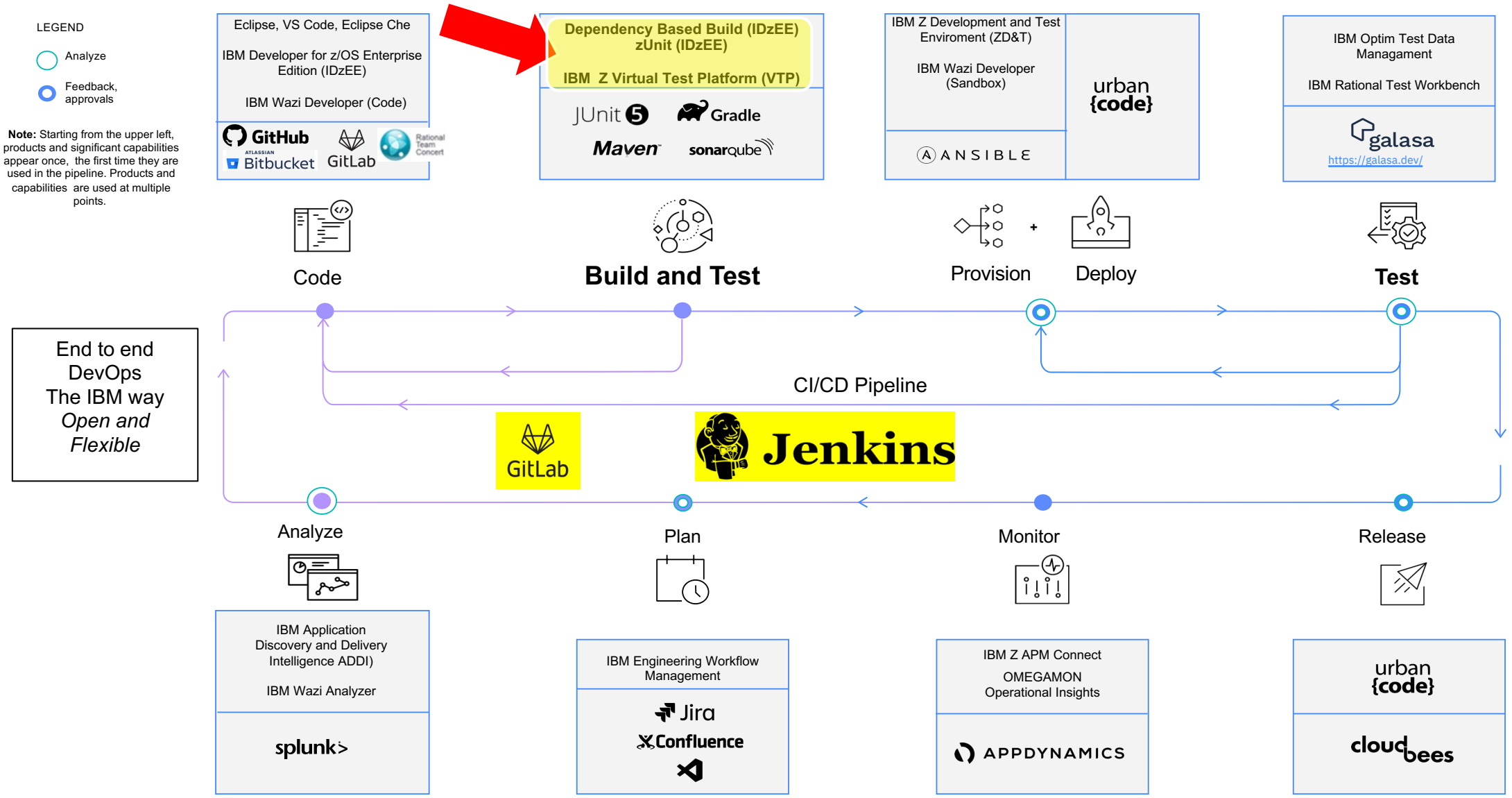
3:00 – 4:50 Choose one Optional lab:

- → LAB 7 - IBM **DBB** with **Git**, **Jenkins** and **UCD** on Z
- → LAB 6B : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Remote Assets**
- → LAB 6C : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Local Assets**
- → LAB 6D : Using **IBM zUnit** to Unit Test a COBOL/DB2 **batch** program
- → 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**)
- → LAB 9 - Using **IBM Z VTP** to test a COBOL /CICS / DB2 transaction
- → LAB 10 - Deploying COBOL/CICS/DB2 application using a **GitLab CI** Pipeline
- → LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- → LAB 12 – Using **ADDI** to find CICS programs to be converted to API's
- → LAB 13 – Running Integration Tests using **Galasa**

5:00 Checkpoint - Comments and Closing Day 2 (David/Ron/Wilbert/Regi)

Lecture Labs

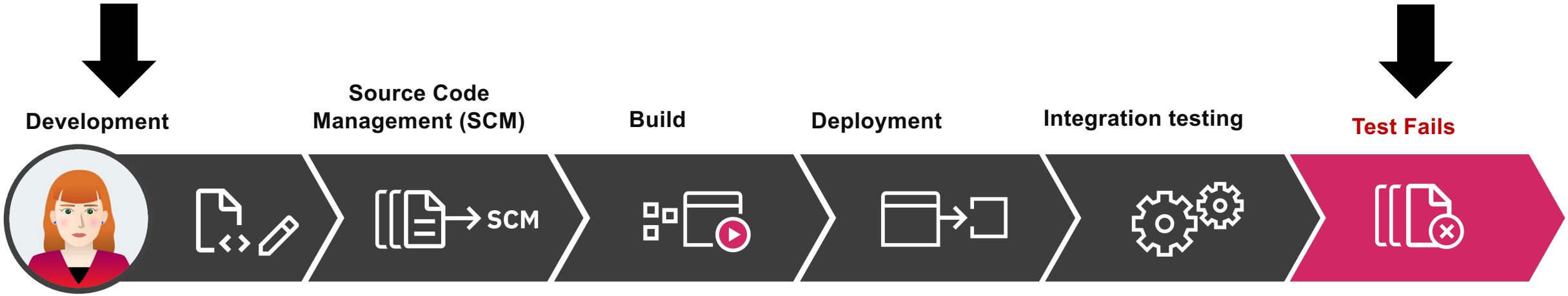
Z DevOps Pipeline



Shift Left testing

Start your testing here

Don’t wait until here



Deb, The Developer

Makes a code change for program HCIPDB01 as part of transaction **HCAZ**

Checks in code

Runs build

Deploys changes to the target environment

Full integration test

Shift Left

The Testing Pyramid

What tools and practice do IBM offer to fit in with The Test Pyramid?

User Acceptance Test

Actual users test the software to ensure the change meets business requirement

System Test

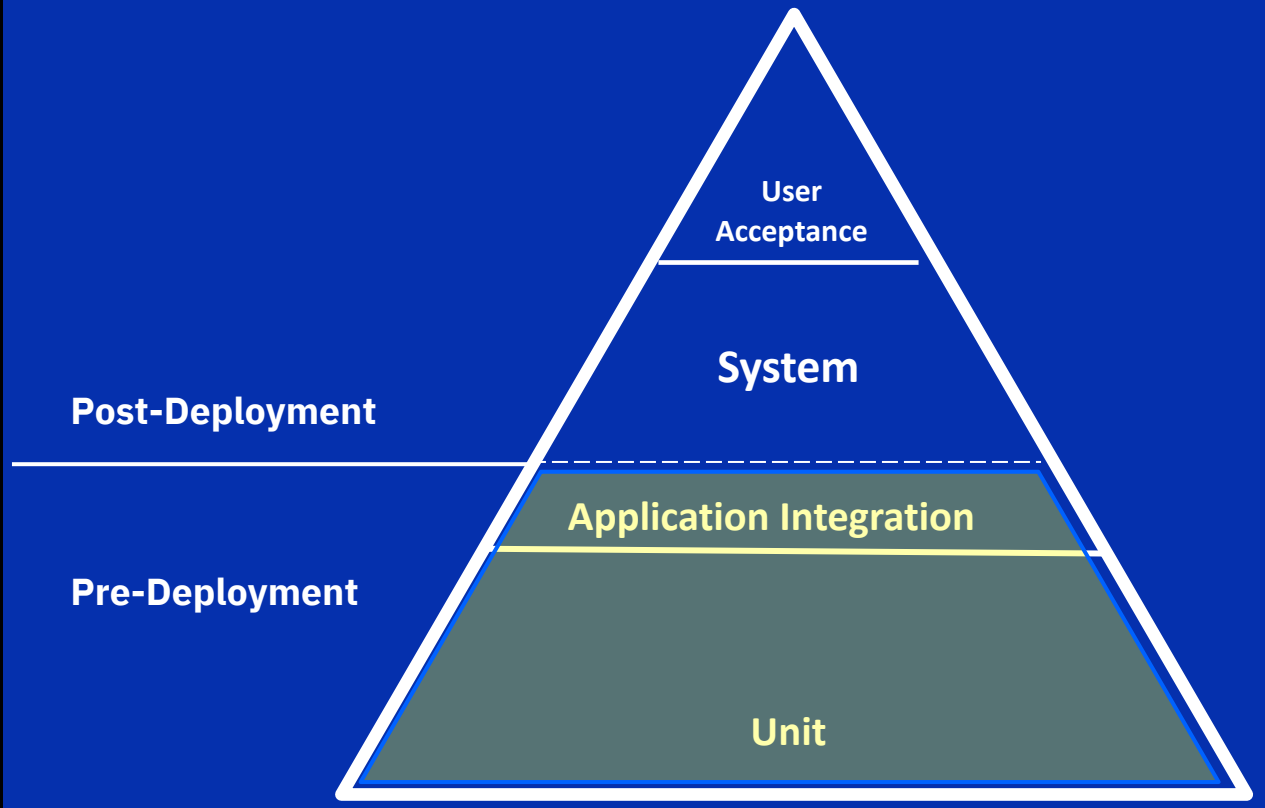
RTW (Rational Integration Tester/Rational functional tester/Test Virtualization Server)

Application Integration test

IBM Virtual Test Platform

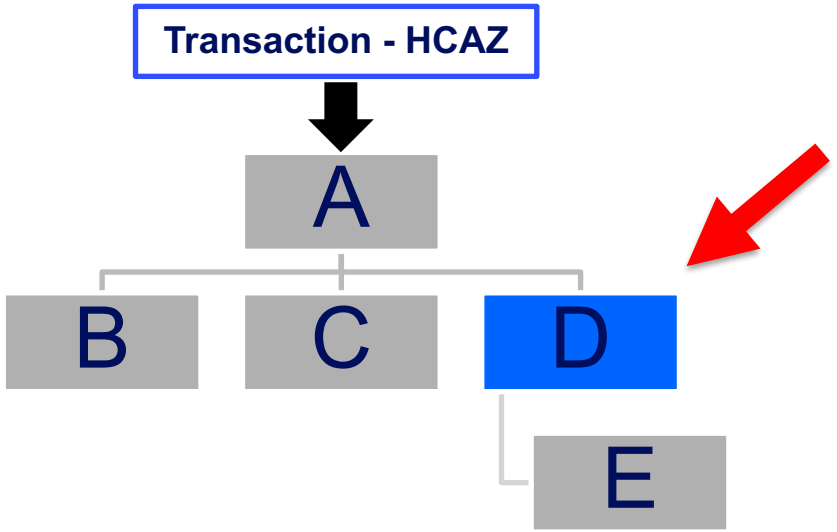
Unit test

zUnit (IBM Developer for z/OS)



Unit Testing (zUnit)

Scope - Program



A unit is the smallest testable part of software – meaning testing an individual unit of source code, like a single source file which contains a list of methods, procedures.

Our current “unit” for z/OS Applications is a Program

Shift Left – unit testing with **ZUnit**

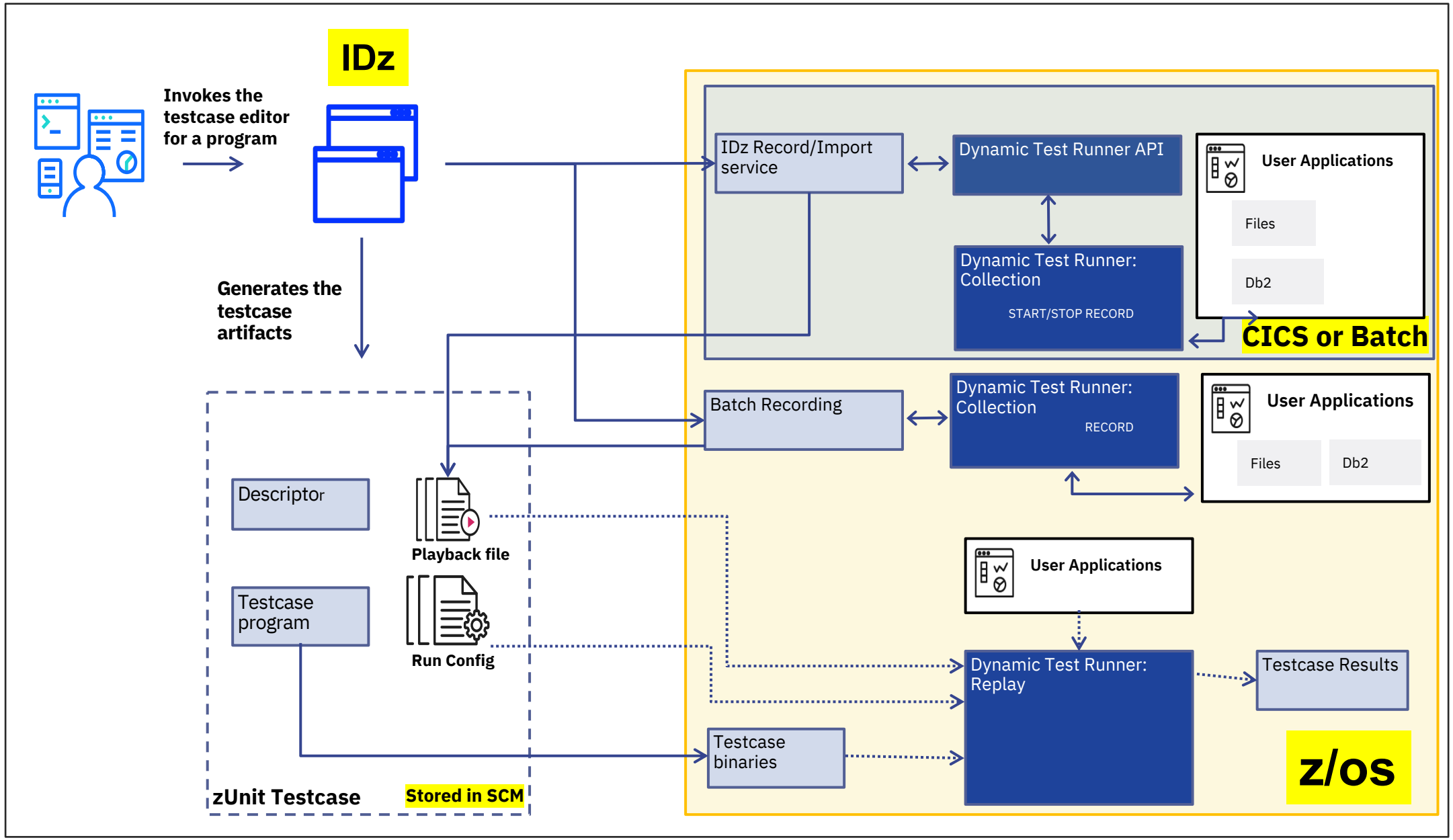
A way of testing the smallest piece of code or program that can be practically isolated

This is done at the **program level**

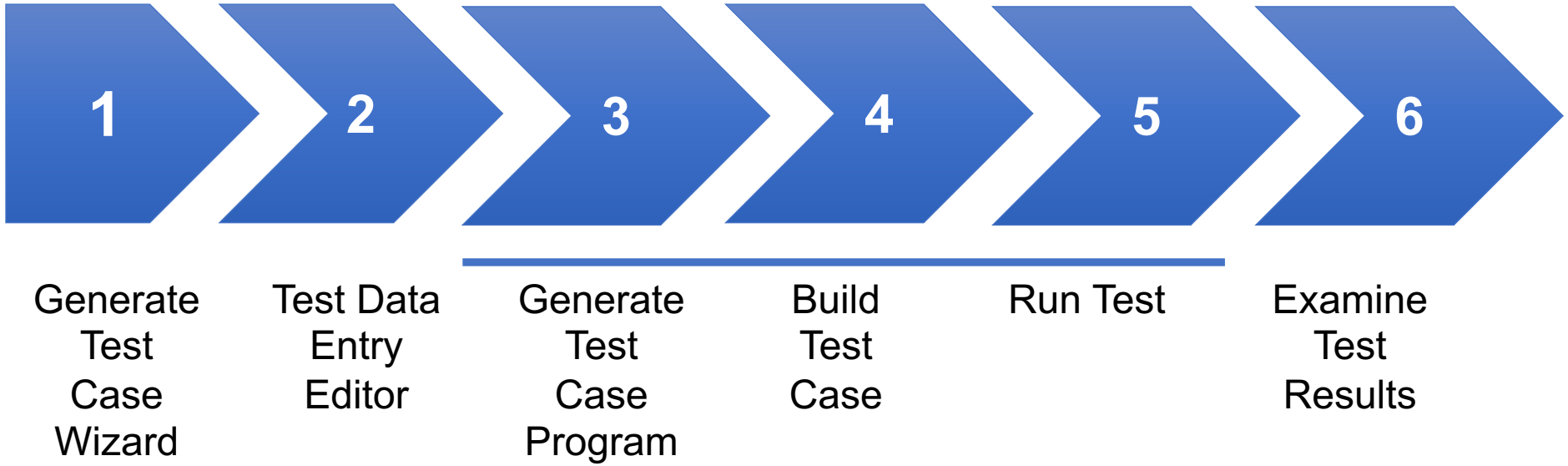
```
1  //IBMUSER1 JOB ,
2  // MSGCLASS=H,MSGLEVEL=(1,1),REGION=0M,COND=(16,LT)
3  //* Action: Run Test Case... NO CICS or DB2 required
4  //* TEST CASE Module at: IBMUSER.GIT.ZMOBILE.LOAD(THCIPDB0)
5  //         SET FELJOB=ZUNITGO
6  //RUNNER EXEC PROC=BZUPPLAY,
7  // BZUCFG=IBMUSER.ZUNIT.BZUCFG(THCIPDB0),
8  // BZUCBK=IBMUSER.GIT.ZMOBILE.LOAD,
9  // BZULOD=IBMUSER.GIT.ZMOBILE.LOAD,
10 //  PRM='STOP=E,REPORT=XML'
11 //REPLAY.BZUPLAY DD DISP=SHR,
12 // DSN=IBMUSER.ZUNIT.PB.HCIPDB01  *PLAYBACK File
13 //REPLAY.BZURPT DD DISP=SHR,
14 // DSN=IBMUSER.ZUNIT.BZURES(THCIPDB0)
15 //* Below is where is the program to be TESTED
16 //STEPLIB  DD DISP=SHR,DSN=IBMUSER.GIT.ZMOBILE.LOAD
```

No CICS and DB2 required

ZUnit High Level Architecture

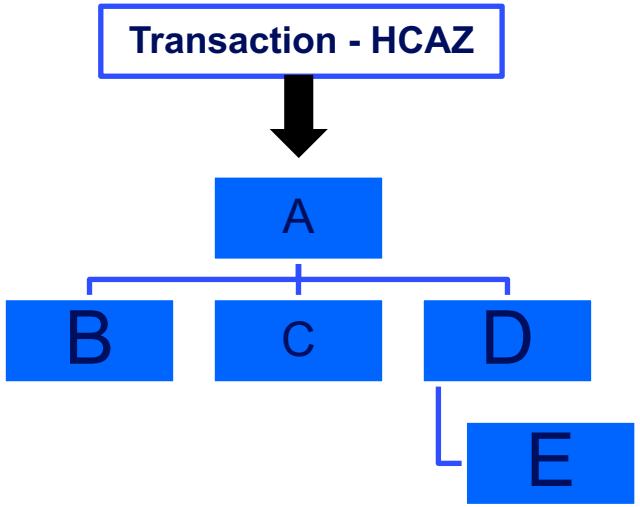


zUnit Basic Workflow



Application Integration test (IBM Z VTP)

Scope - Transaction



Application integration test is the next level of testing after Unit Testing. Its purpose is to confirm that the changes in a unit (programs) works with the interfaces to other units (programs), and ensure no unexpected impacts to other units

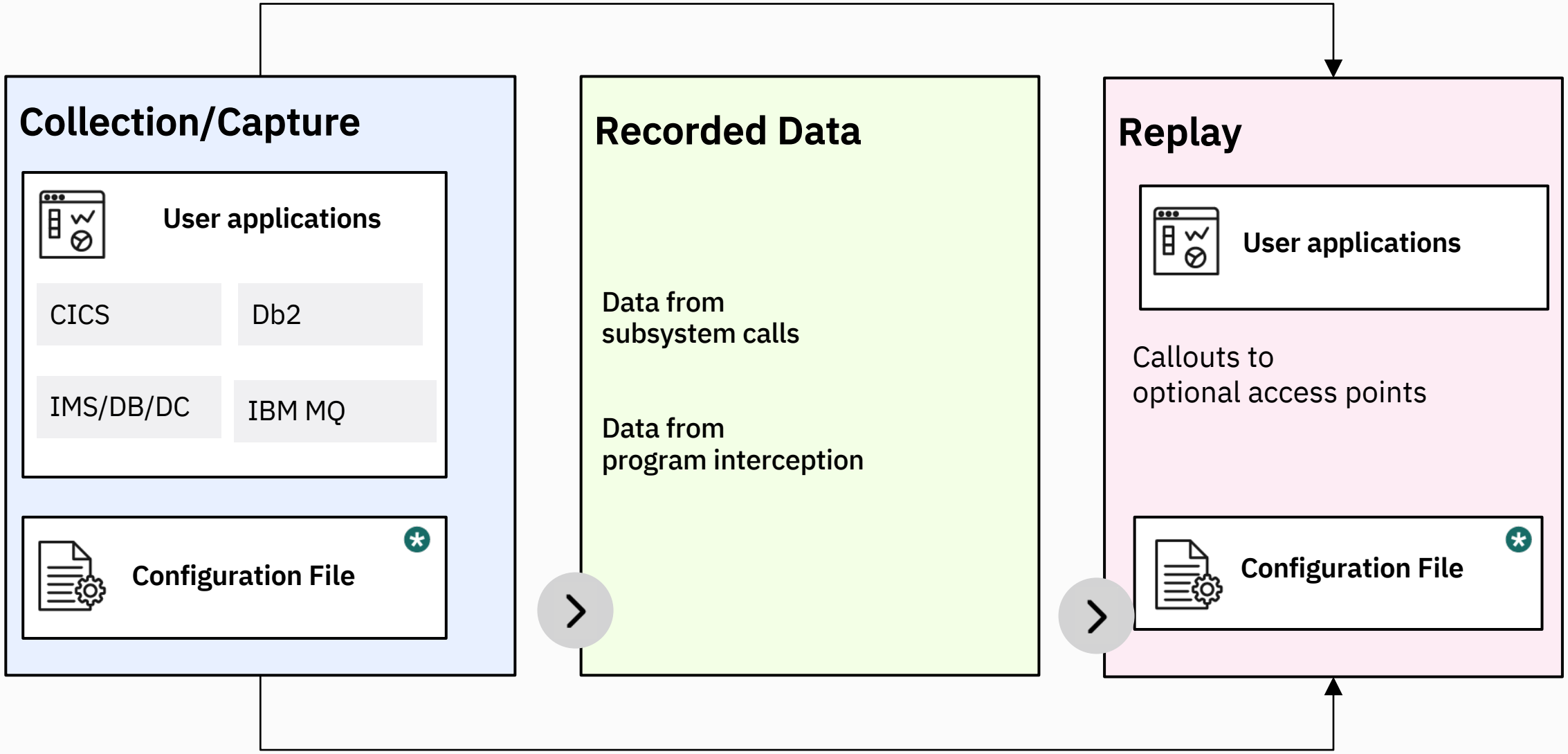
Transaction and batch jobs

```
//HCAZPLAY  JOB ,NOTIFY=&SYSUID,REGION=0M
//*  Run IBM Z VTP for HCAZ   Application
//PLAYBK    EXEC PGM=BZUPLAY,PARM='TRACE=N'
//STEPLIB   DD DISP=SHR,DSN=BZU100.SBZULOAD
//* Below are the programs to be tested
//          DD DISP=SHR,DSN=EMPOT.ZMOBILE.TEST.LOAD
//*
//BZUPLAY   DD DISP=SHR,DSN=BZU100.ZUNIT.PLAYBACK.DEMO  *playback file
//SYSOUT    DD SYSOUT=*
//BZUMSG  DD SYSOUT=*
```

Testing with Z Virtual Test Platform

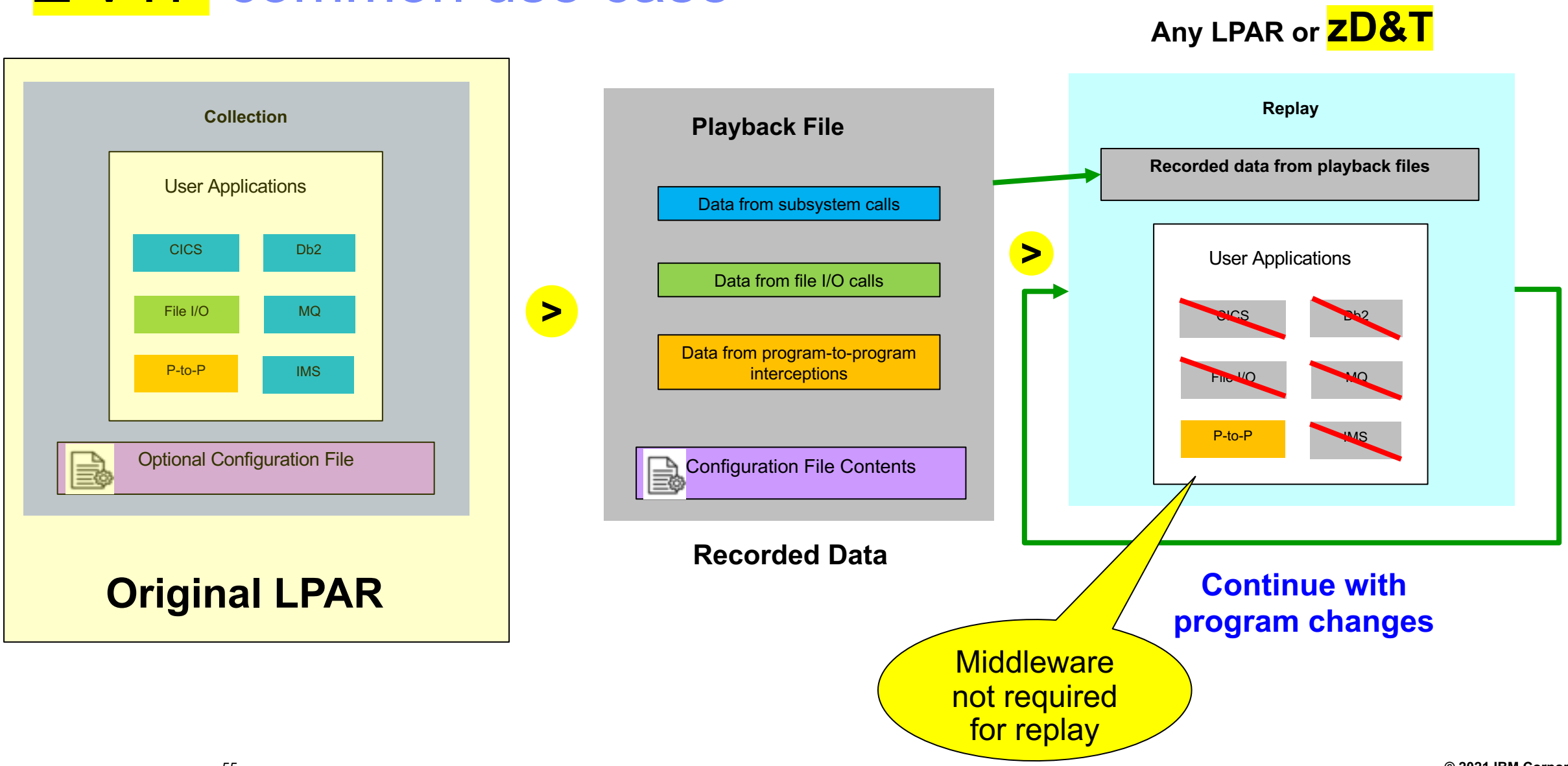
Part 1 – Capture or Collection

Part 2 – Replay



Optional

Z VTP common use case



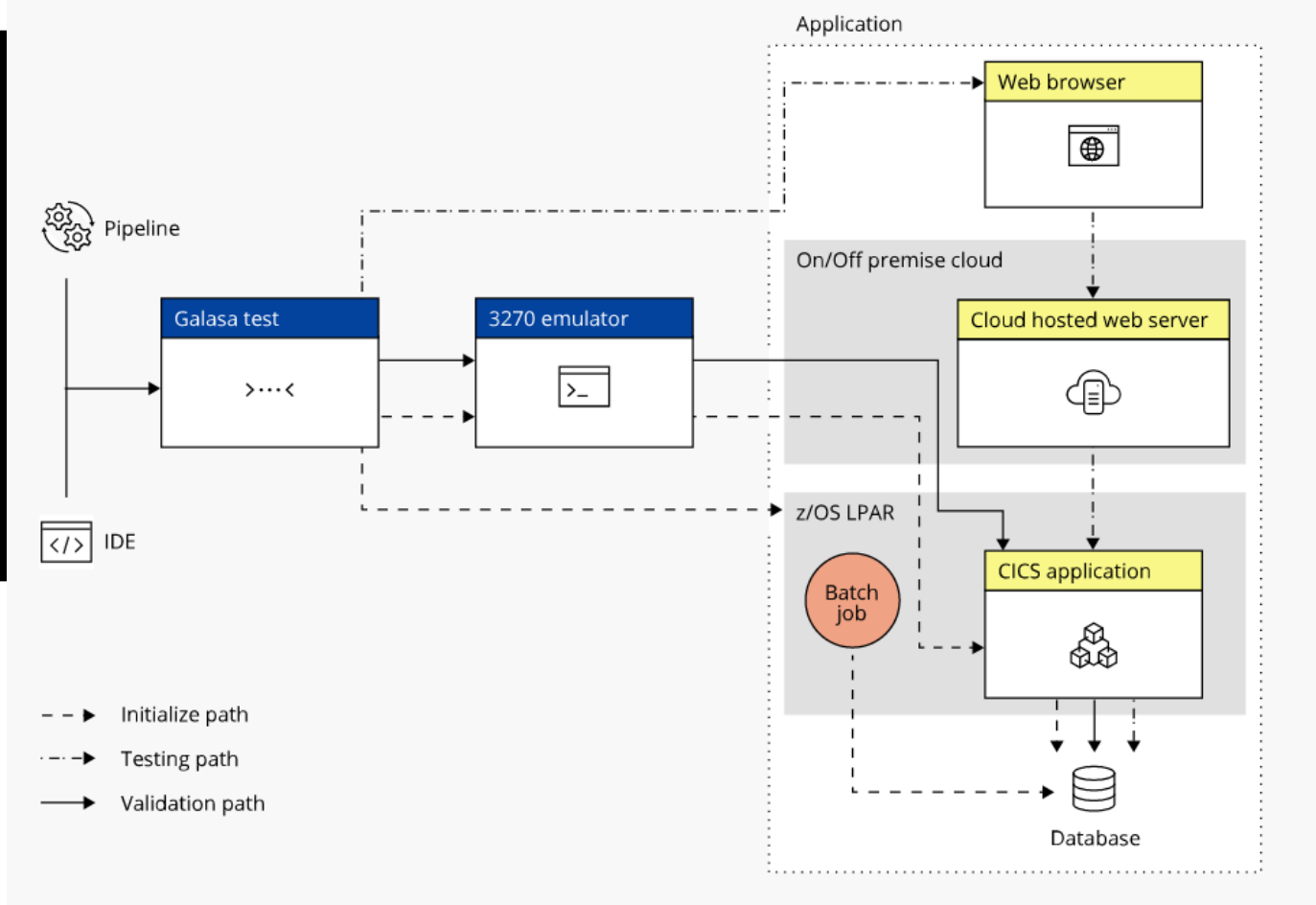
Galasa test automation framework

Run repeatable, reliable, automated tests and deliver software changes faster, with confidence

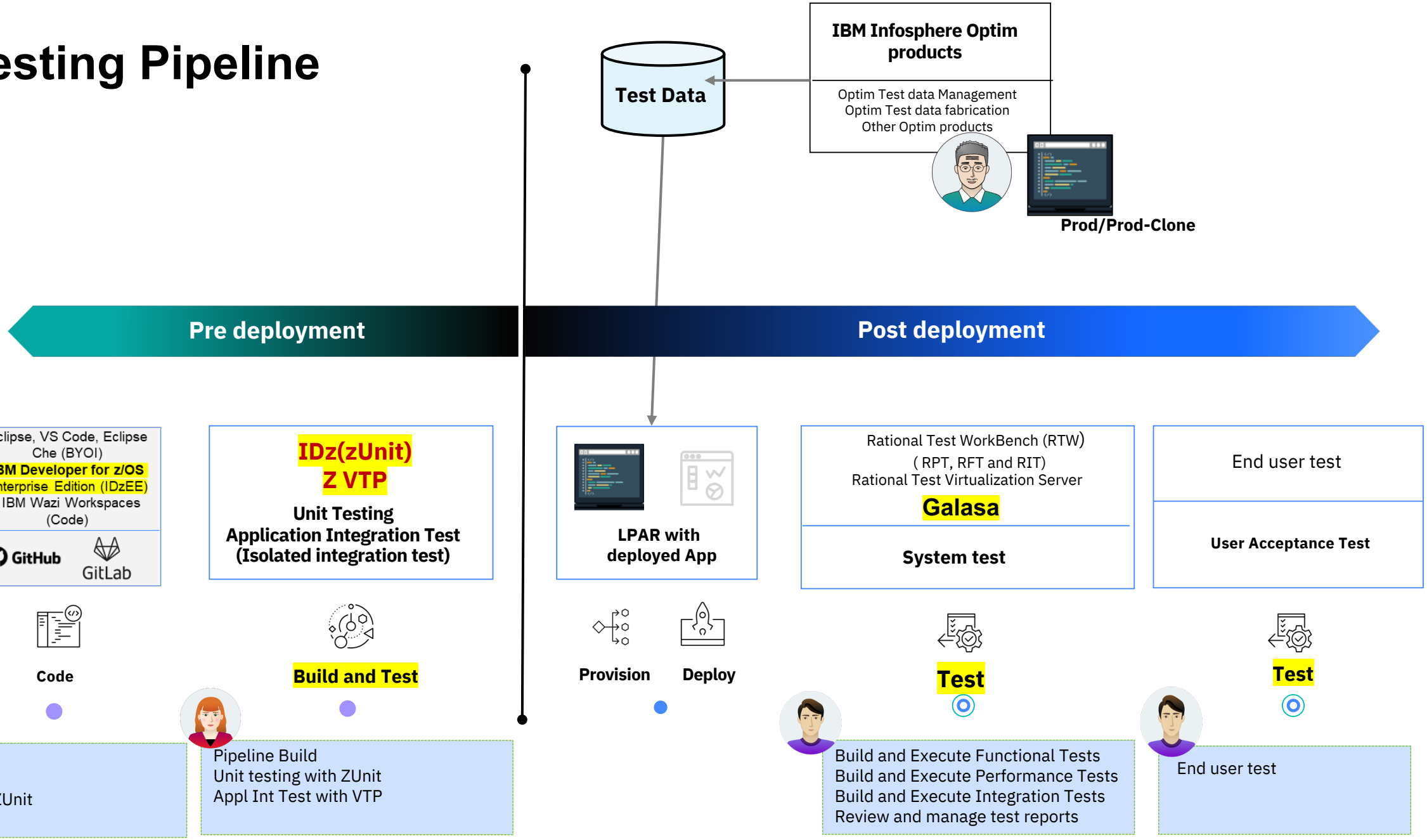
Test hybrid cloud applications leveraging first class integration with z/OS

https://galasa.dev

- Provides scalable test scheduling, execution and reporting
- Offers deep integration with z/OS, IBM Middleware and a range of open-source test tools
- Integrates with common CI/CD pipeline products
- Helps eliminate repetitive manual and semi-automated testing
- Based on the technology that several internal IBM teams have used to test IBM products



The Testing Pipeline




QUESTIONS?

Agenda - Day 1

2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)

3:00 LAB 1 - Working with z/OS using COBOL and DB2

or

LAB7 - IBM DBB with Git, Jenkins and UCD on Z

5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

2:00 zUnit and VTP Overview (Wilbert)

2:15 **Demo**: Scenario using Z DevOps solutions including **ADDI**, **zUnit**, **Git** and **Jenkins** (David/Regi)

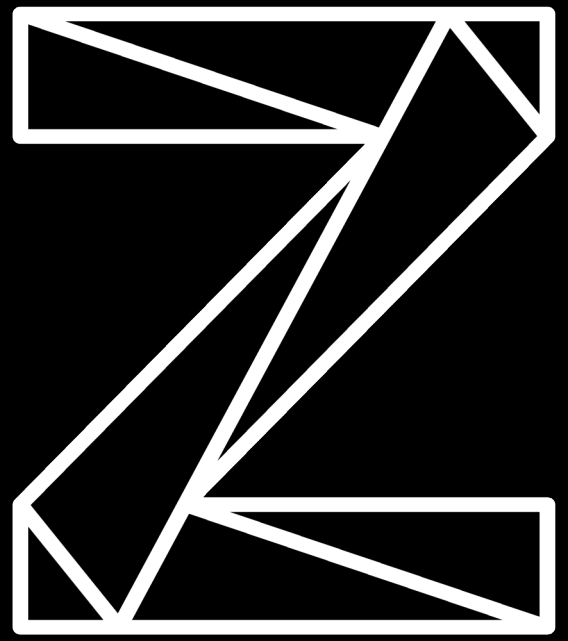
3:00 – 4:50 Choose one Optional lab:

- → LAB 7 - IBM **DBB** with **Git**, **Jenkins** and **UCD** on Z
- → LAB 6B : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Remote Assets**
- → LAB 6C : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Local Assets**
- → LAB 6D : Using **IBM zUnit** to Unit Test a COBOL/DB2 **batch** program
- → 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**)
- → LAB 9 - Using **IBM Z VTP** to test a COBOL /CICS / DB2 transaction
- → LAB 10 - Deploying COBOL/CICS/DB2 application using a **GitLab CI** Pipeline
- → LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- → LAB 12 – Using **ADDI** to find CICS programs to be converted to API's
- → LAB 13 – Running Integration Tests using **Galasa**

5:00 Checkpoint - Comments and Closing Day 2 (David/Ron/Wilbert/Regi)

Lecture Labs

IBM Z DevOps Tools in action



Demonstration

Session Objectives

- We will demonstrate how IBM DevOps and open source tools like Git and Jenkins help the continuous integration, continuous testing, continuous delivery on the z/OS space.

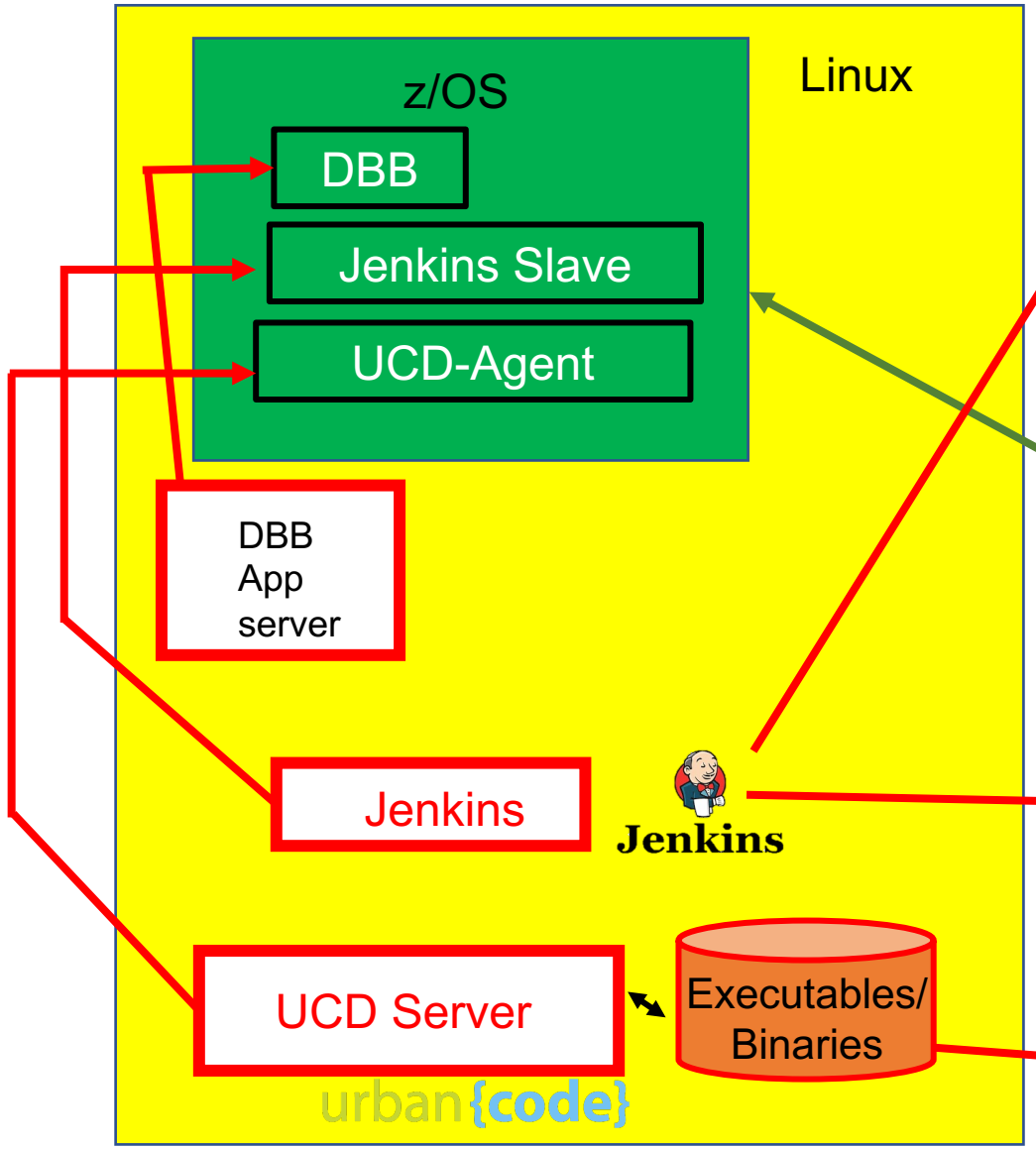
- Our scenario consists of a “broken” COBOL/CICS/DB2 application. The developer will use IBM DevOps tools to understand the application (ADDI), locate and identify the faulty code, make the appropriate fixes and compile/link/bind using a new build automation tool (DBB) that integrates with Git and Jenkins pipeline on z/OS. Once the code is fixed, the developer will push the changes to a Git repository..

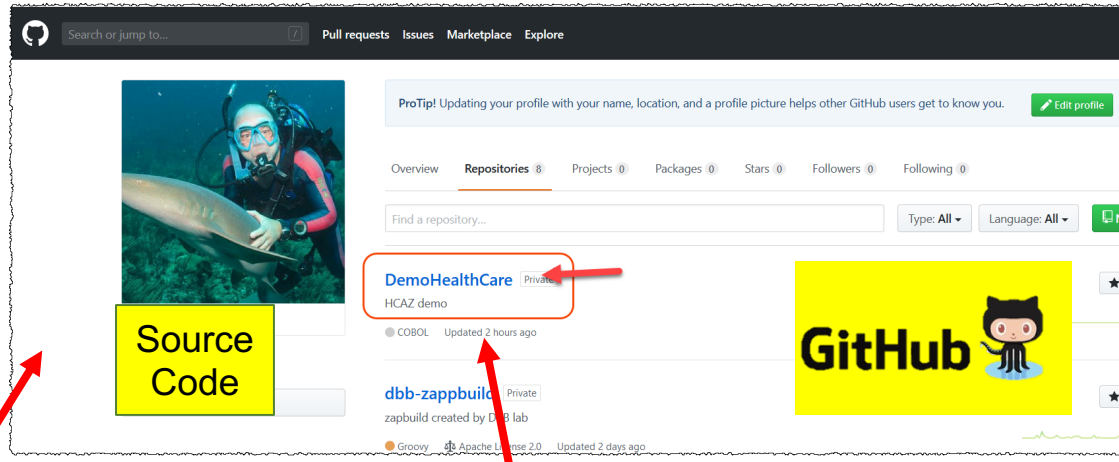
- In order to demonstrate continuous delivery capabilities, this session will showcase a Jenkins pipeline integrated with IBM tools for the build process, unit testing(using the new zUnit functionality), and deployment of code to a CICS environment (using UrbanCode Deploy).

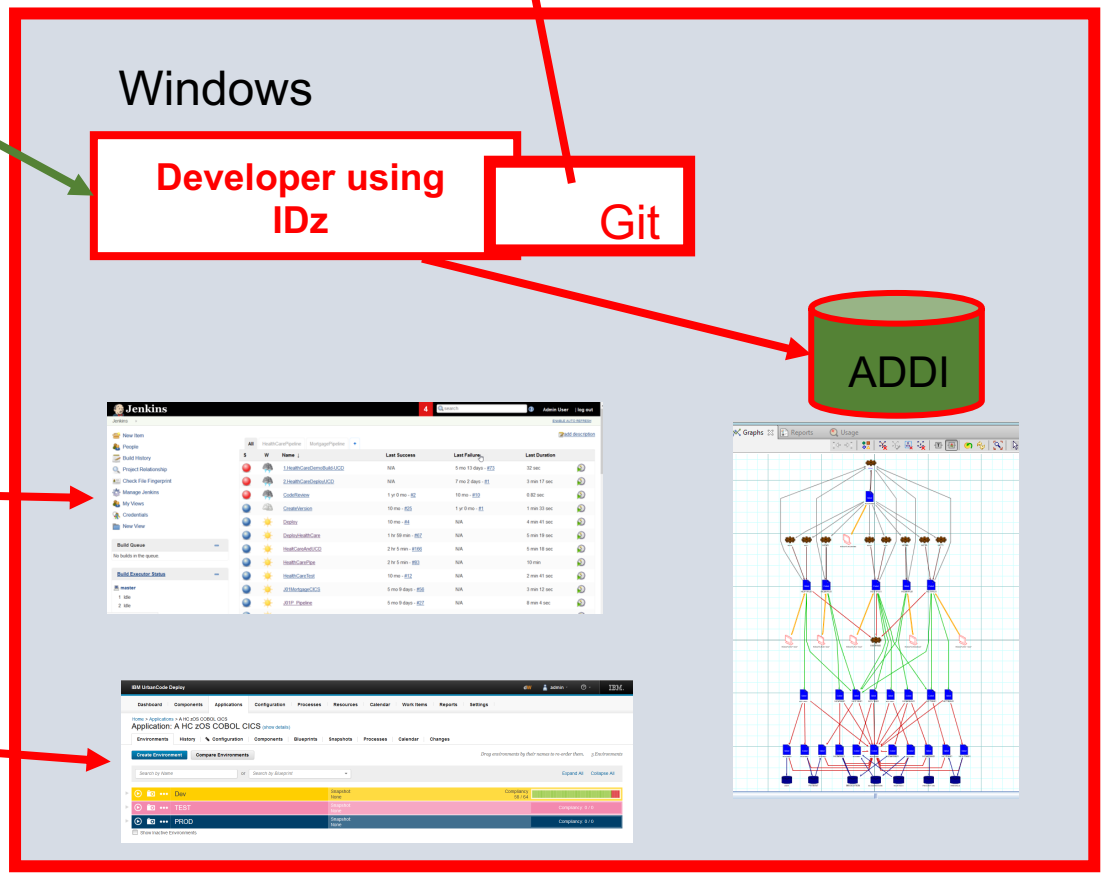
- While our scenario uses a COBOL/CICS/DB2 program as an example, the process is similar to other environments or languages (such as Batch, IMS, or PL/1

Topology used on this Demo

zD&T on VMWARE



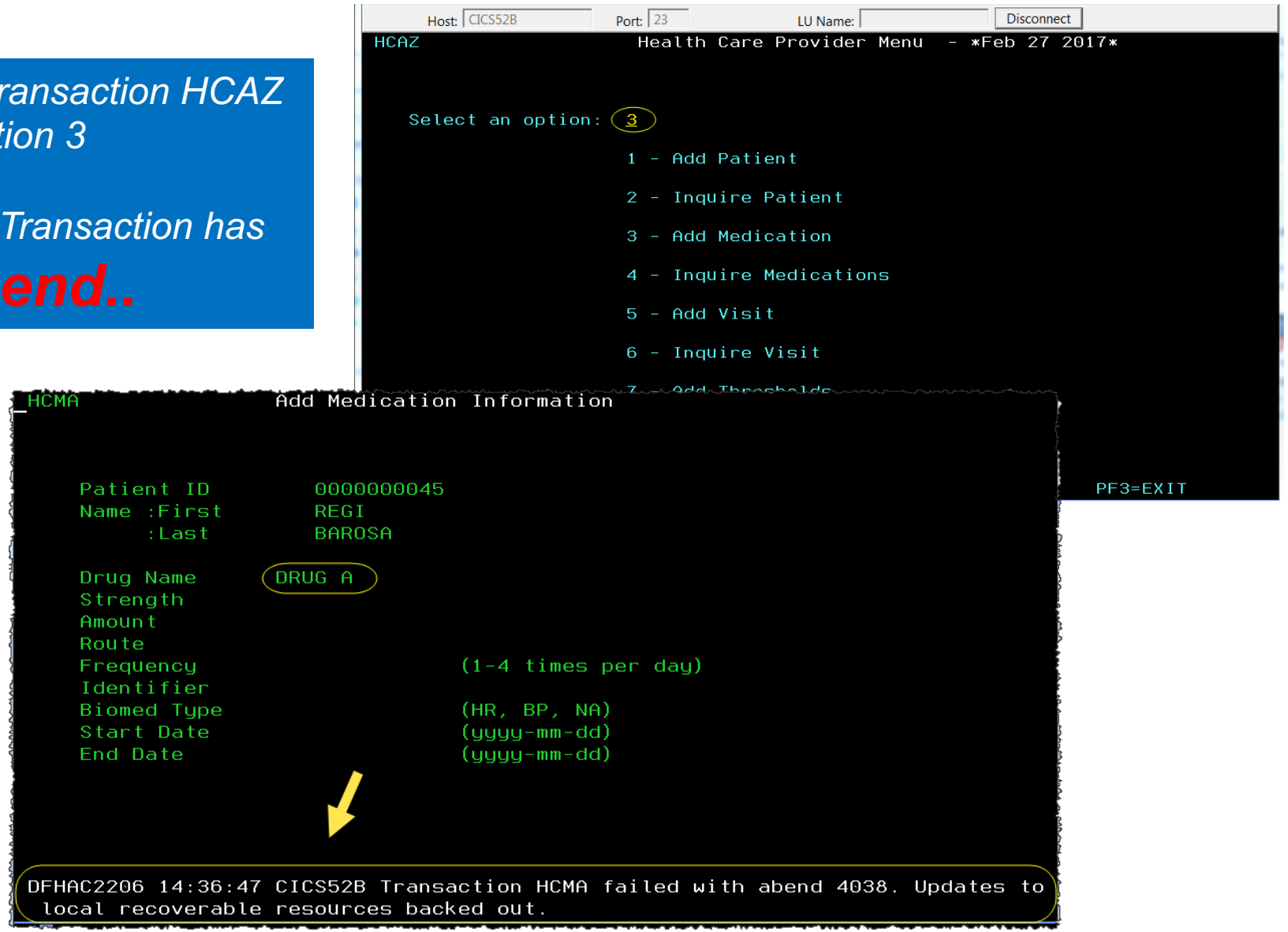




Scenario: CICS Transaction ABEND

Using transaction HCAZ and option 3

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



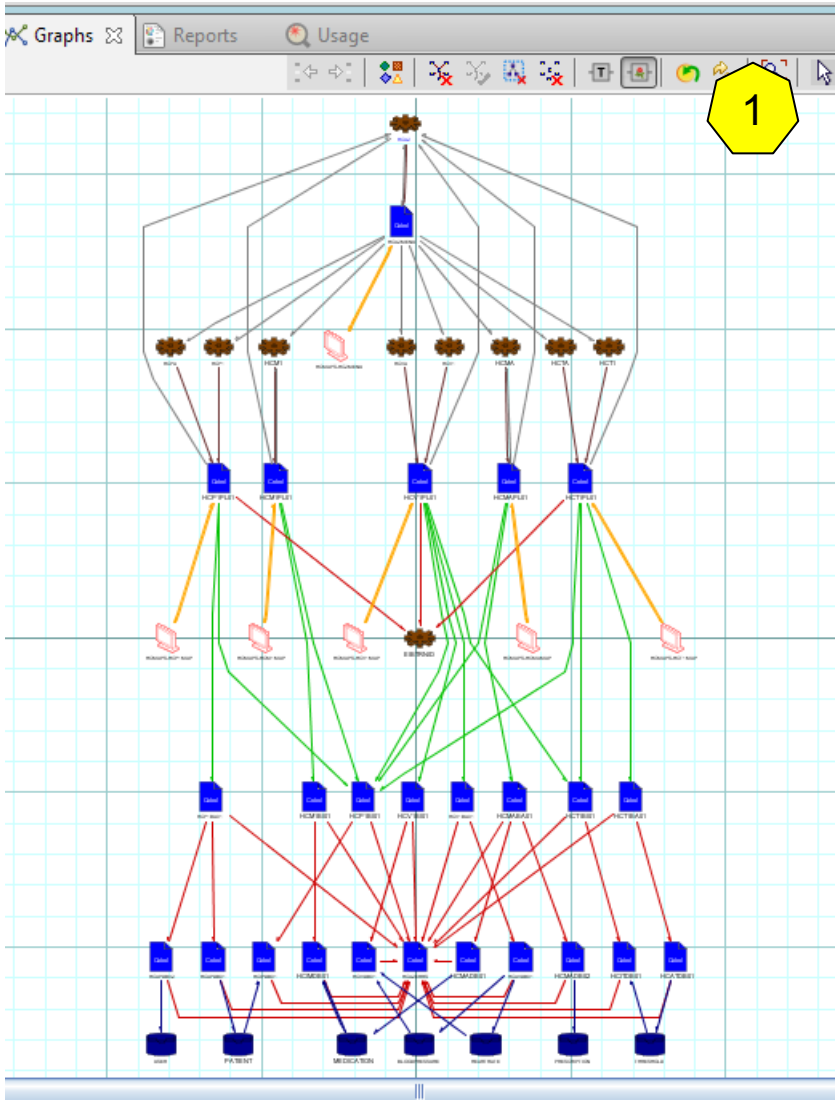
PART #1 - Understand the Application Components

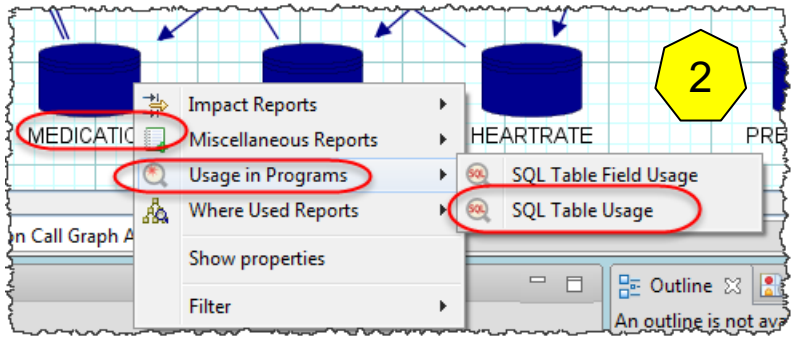
Objective: Fix the CICS Abend

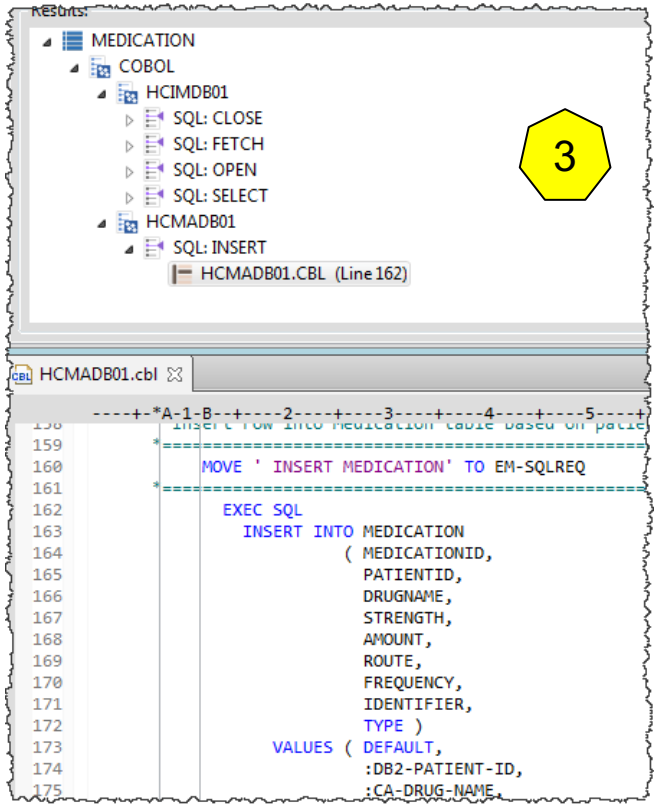
Alex - Project Manager
Performs application analysis

1. Uses **Application Discovery (ADDI)** to understand the **HCAZ** application.

Uses **Application Discovery (ADDI)** to understand the application (**HCAZ**)







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 **IDz/Fault Analyzer** to identify what is causing the abend
3. Uses **IDz/Debug** to debug the program and verify the error
4. Uses **IDz** to update the COBOL program loaded from **Git**
5. Uses **IDz/DBB** to perform a "*User Dependency Build*", run the Unit Test (**zUnit**), **Code Coverage** and verify that the abend is fixed
6. Uses **IDz/Git** to **Commit and Push** the corrected code to Repository

Bob starts CICS transaction HCAZ to verify the abend

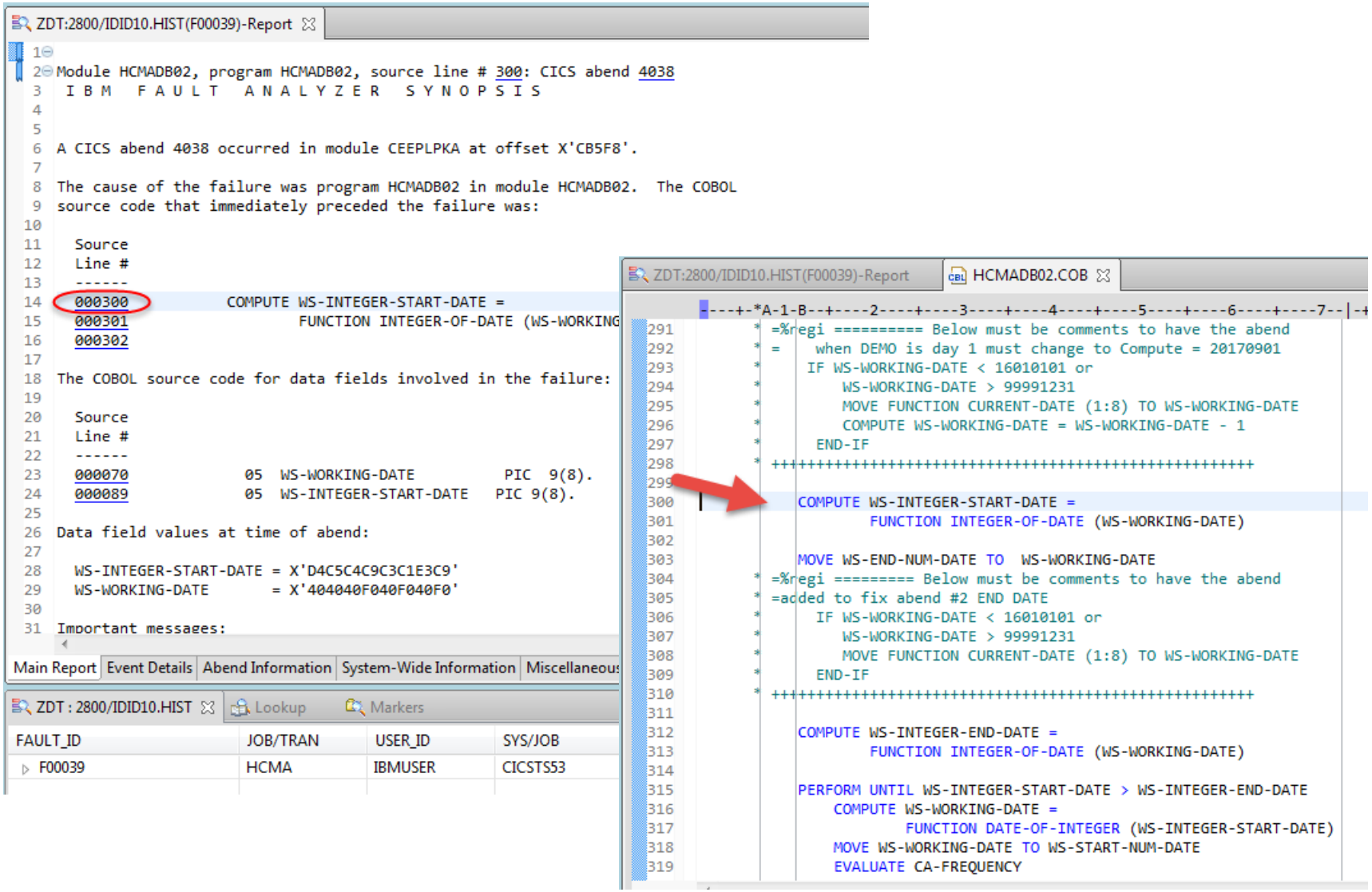
PART #2 - Developer see the CICS abend cause and fix it.

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 **IDz/Fault Analyzer** to identify what is causing the abend
3. Uses **IDz/Debug** to debug the program and verify the error
4. Uses **IDz** to update the COBOL program loaded from **Git**
5. Uses IDz/DBB to perform a “User Dependency Build”, run the Unit Test (zUnit), Code Coverage and verify that the abend is fixed
6. Uses **IDz/Git** to **Commit and Push** the corrected code to Repository

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



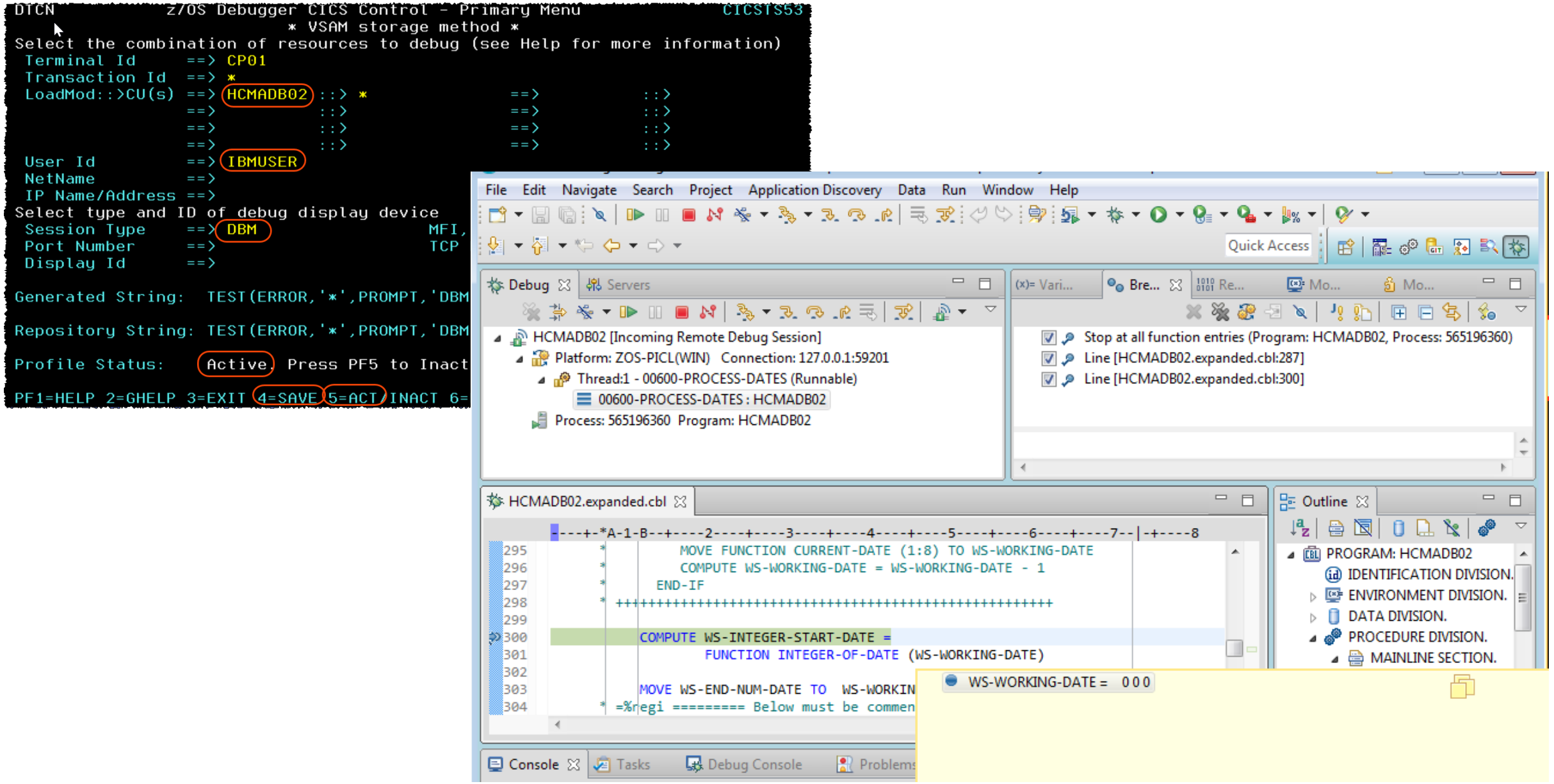
PART #2 - Developer see the CICS abend cause and fix it.

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 **IDz/Fault Analyzer** to identify what is causing the abend
3. Uses **IDz/Debug** to debug the program and verify the error
4. Uses **IDz** to update the COBOL program and Debug the code loaded from **Git**
5. Uses IDz/DBB to perform a “User Dependency Build”, run the Unit Test (zUnit), Code Coverage and verify that the abend is fixed
6. Uses **IDz/Git** to **Commit and Push** the corrected code to Repository

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



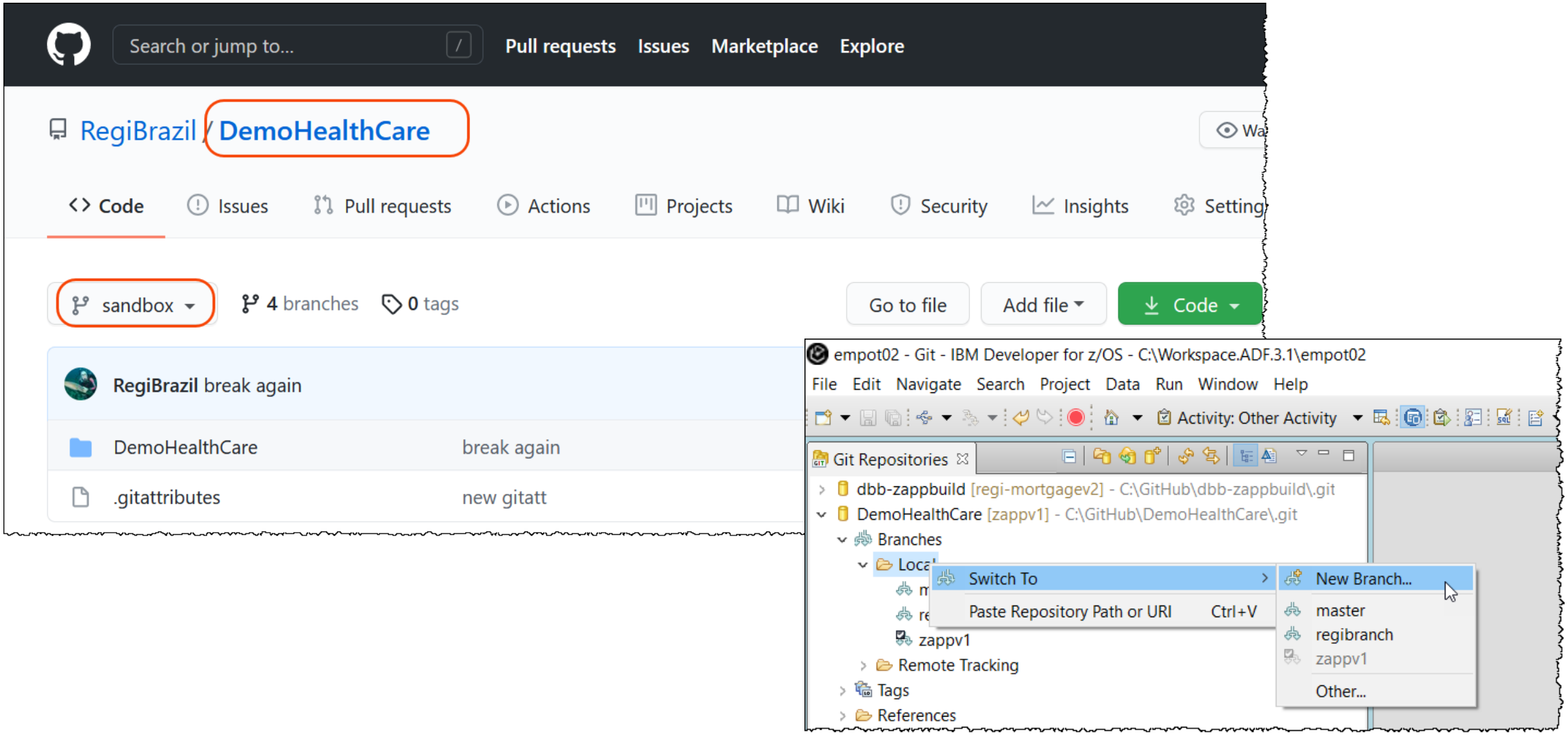
PART #2 - Developer see the CICS abend cause and fix it.

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

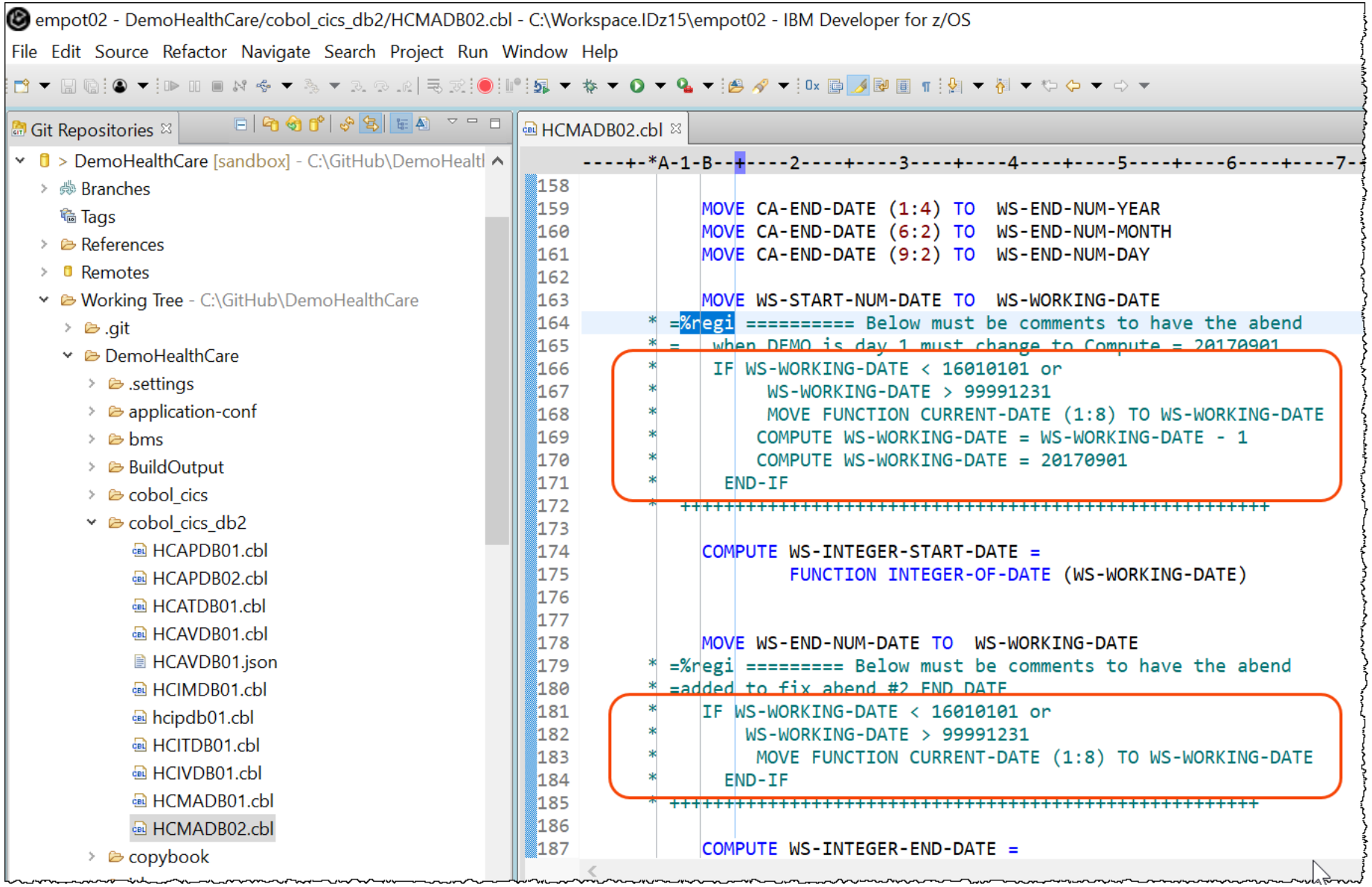
Objective: Fix the CICS Abend

1. Uses **CICS** to verify the issue.
2. Uses **IDz/Fault Analyzer** to identify what is causing the abend
3. Uses **IDz/Debug** to debug the program and verify the error
4. Uses **IDz** to update the COBOL program loaded from **Git**
5. Uses IDz/DBB to perform a “User Dependency Build”, run the Unit Test (zUnit), Code Coverage and verify that the abend is fixed
6. Uses **IDz/Git** to **Commit and Push** the corrected code to Repository

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



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



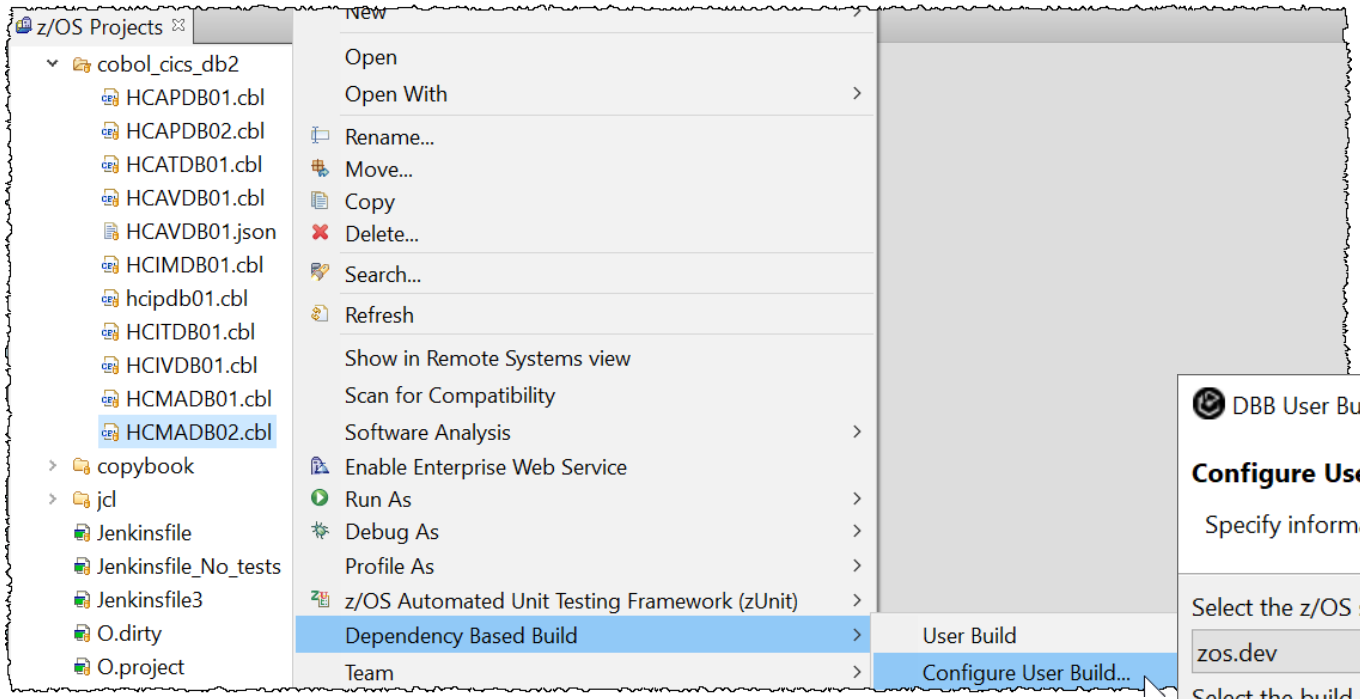
PART #2 - Developer see the CICS abend cause and fix it.

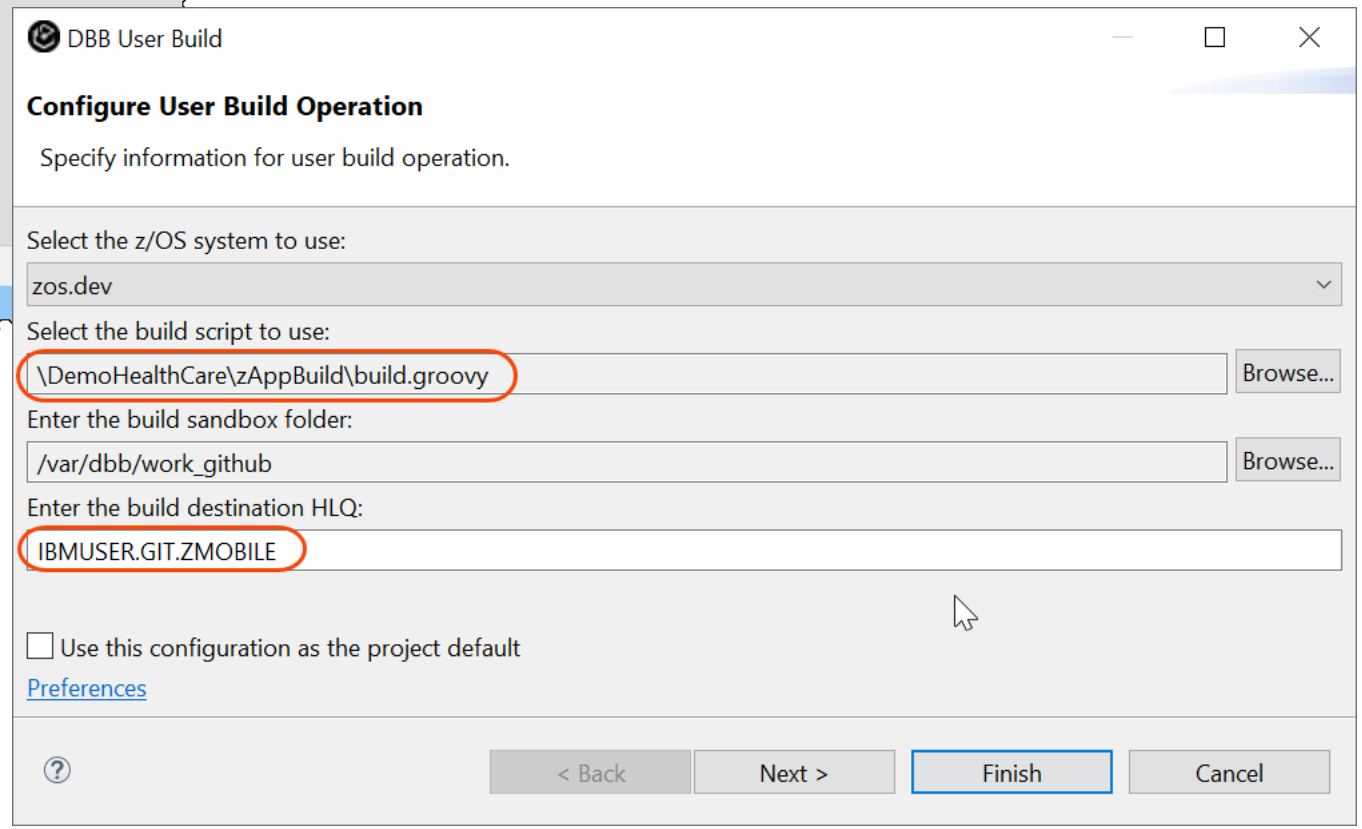
Bob – Mainframe Developer
Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

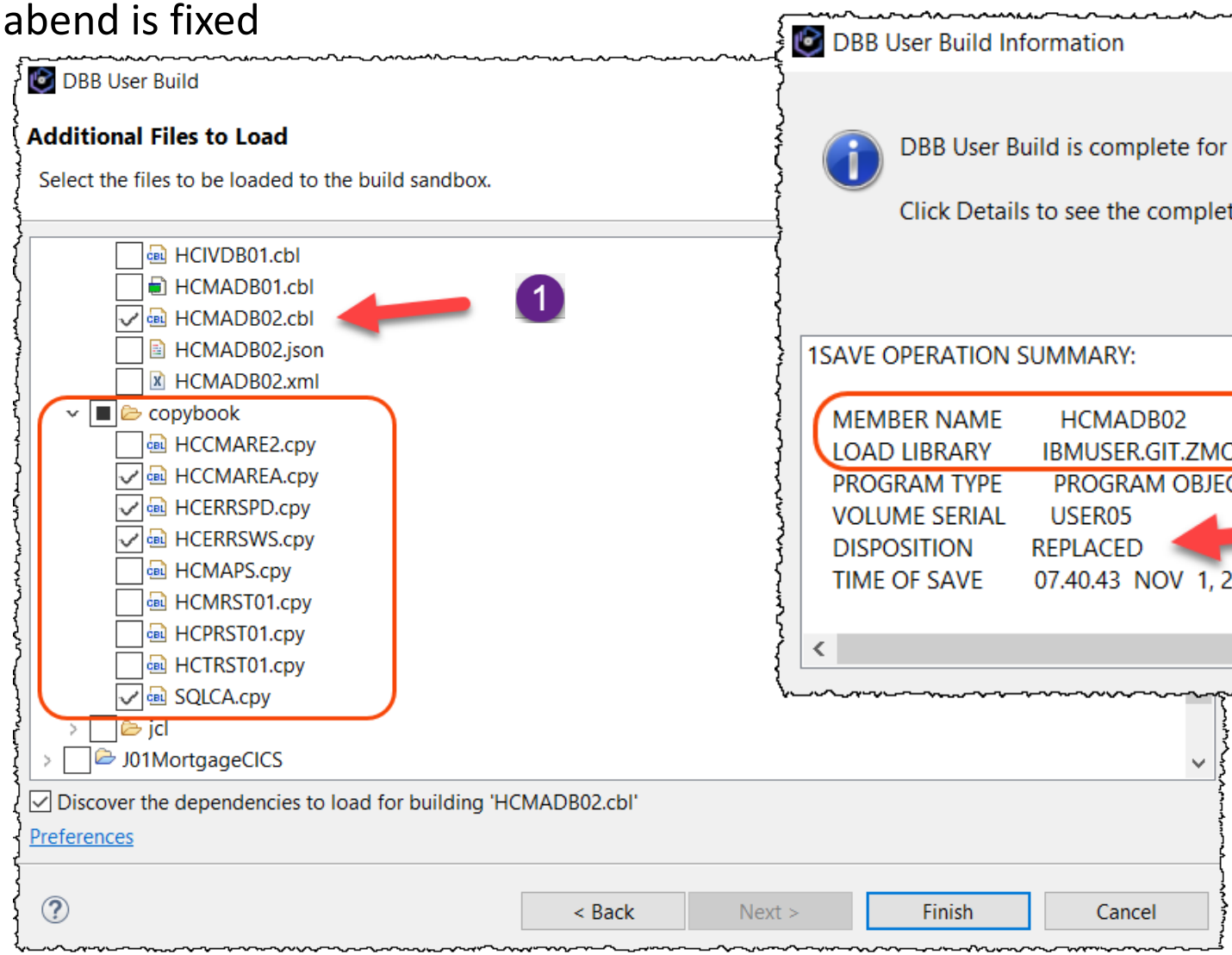
1. Uses **CICS** to verify the issue.
2. Uses **IDz/Fault Analyzer** to identify what is causing the abend
3. Uses **IDz/Debug** to debug the program and verify the error
4. Uses **IDz** to update the COBOL program loaded from **GIT**
5. Uses **IDz/DBB** to perform a "*User Dependency Build*", run the Unit Test (**zUnit**), **Code Coverage** and verify that the abend is fixed
6. Uses **IDz/Git** to **Commit and Push** the corrected code to Repository

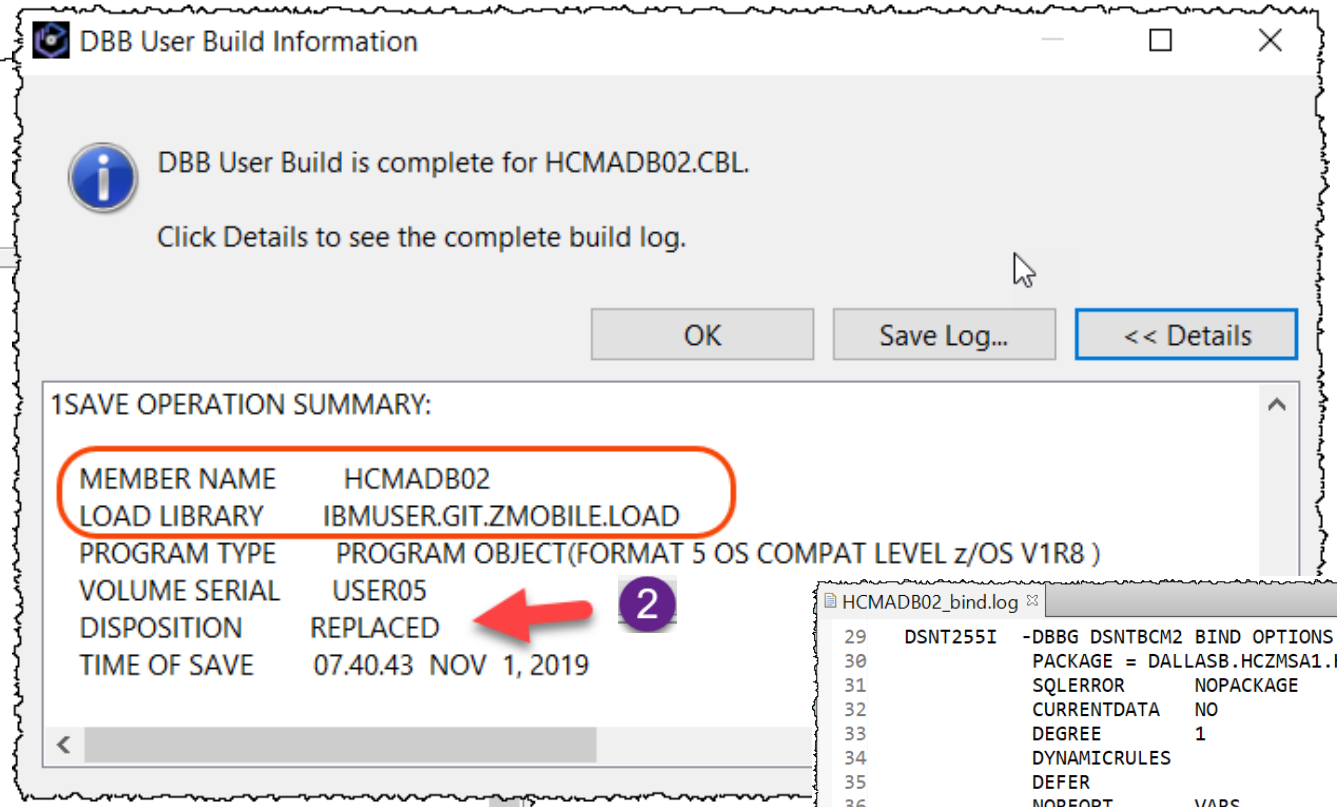
Uses IDz/DBB to perform a "**User Dependency Build**", run the Unit Test (zUnit), Code Coverage and verify that the abend is fixed

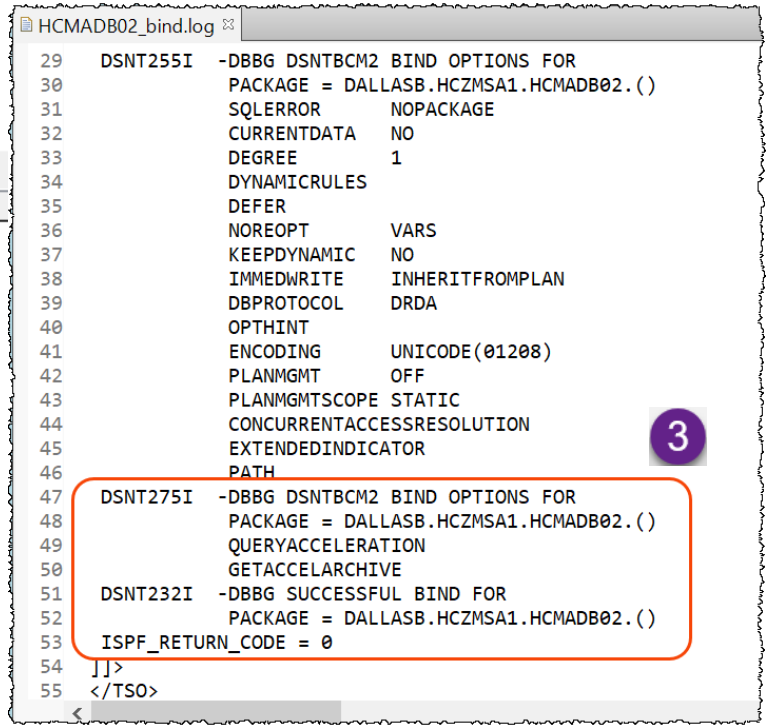


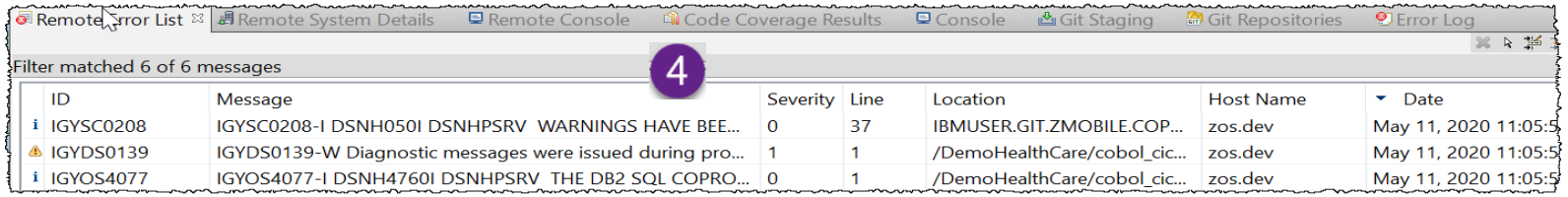


Uses IDz/DBB to perform a "**User Dependency Build**", run the Unit Test (zUnit), Code Coverage and verify that the abend is fixed



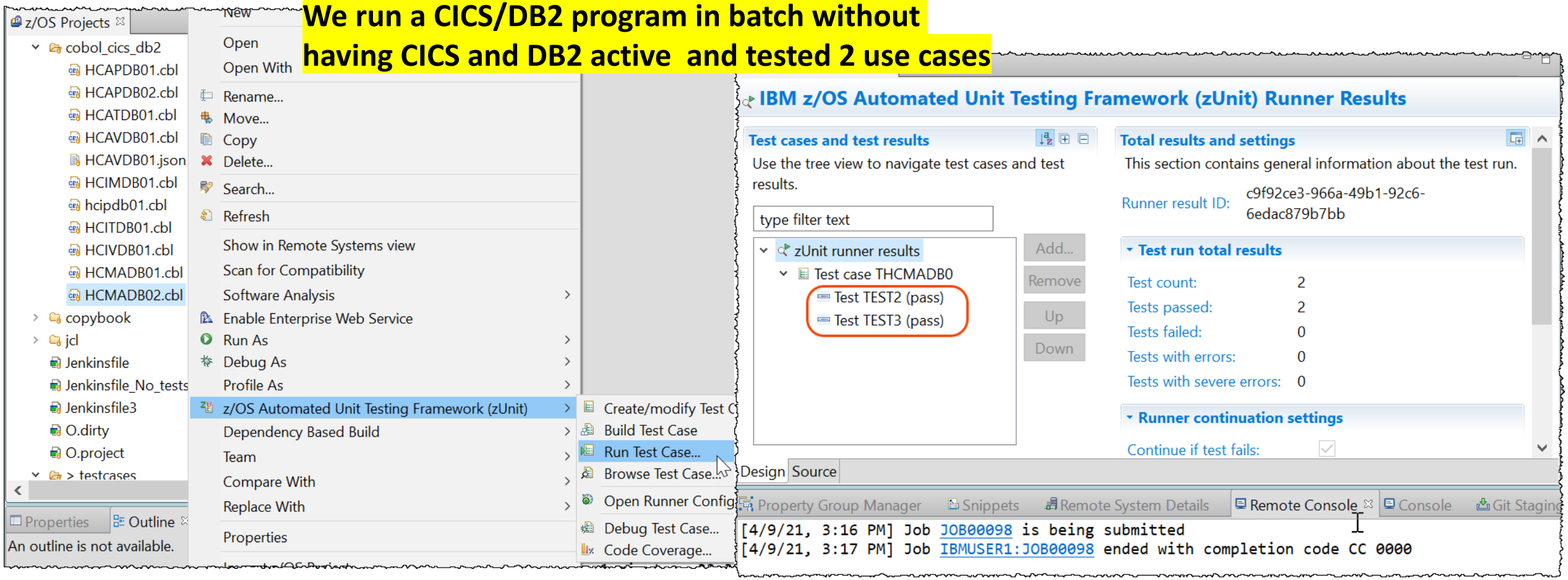




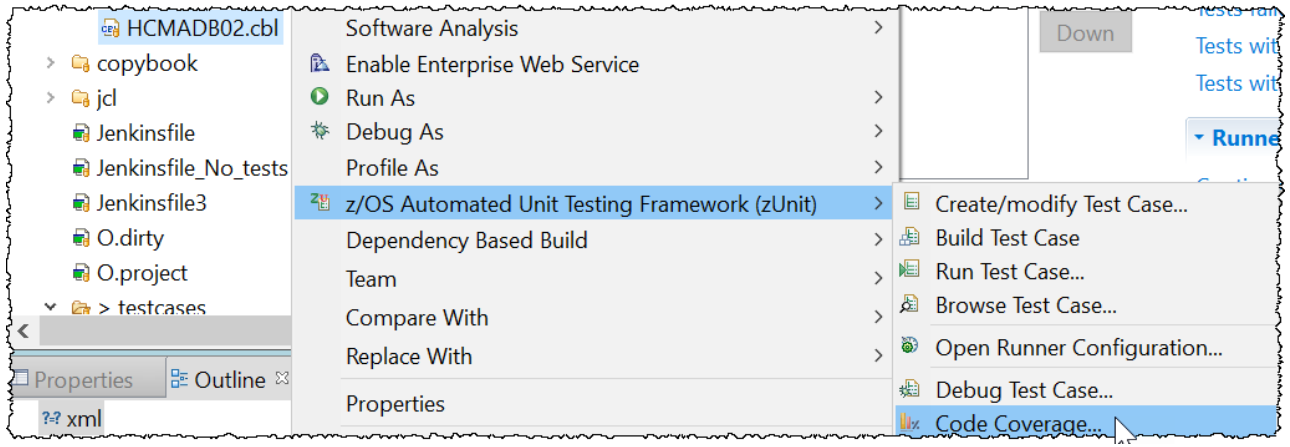


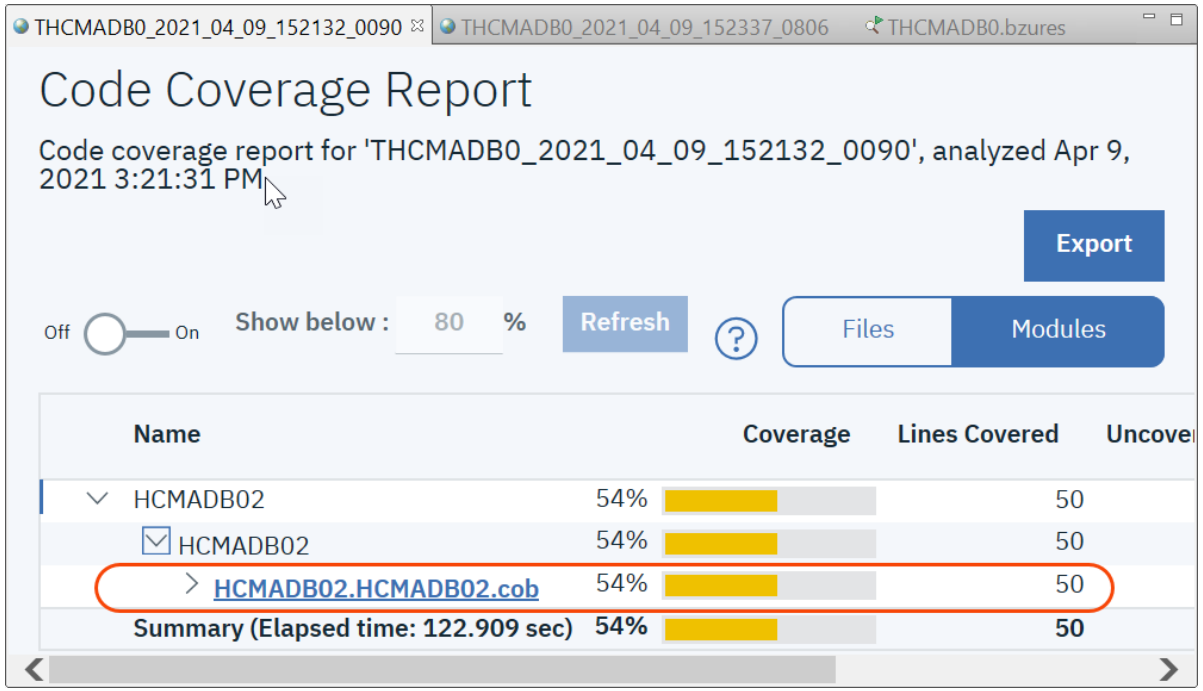
Uses IDz/DBB to perform a “User Dependency Build”, run the **Unit Test (zUnit)**, Code Coverage and verify that the abend is fixed

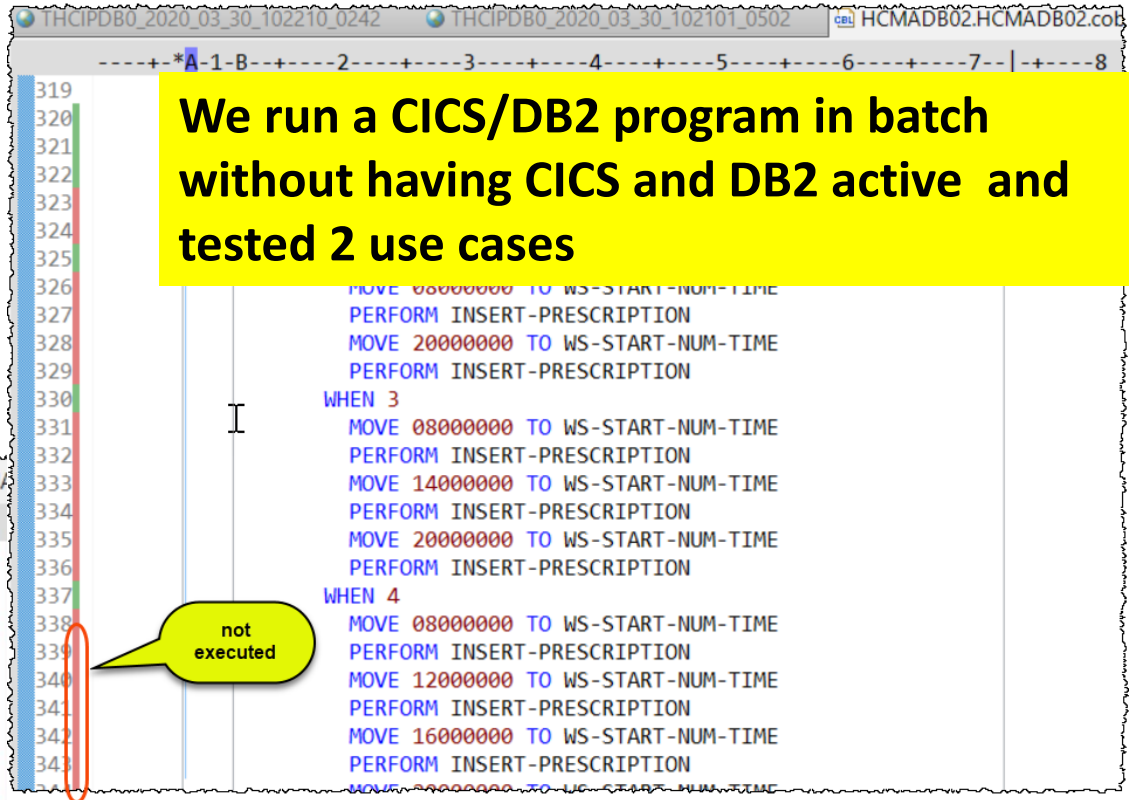
We run a CICS/DB2 program in batch without having CICS and DB2 active and tested 2 use cases

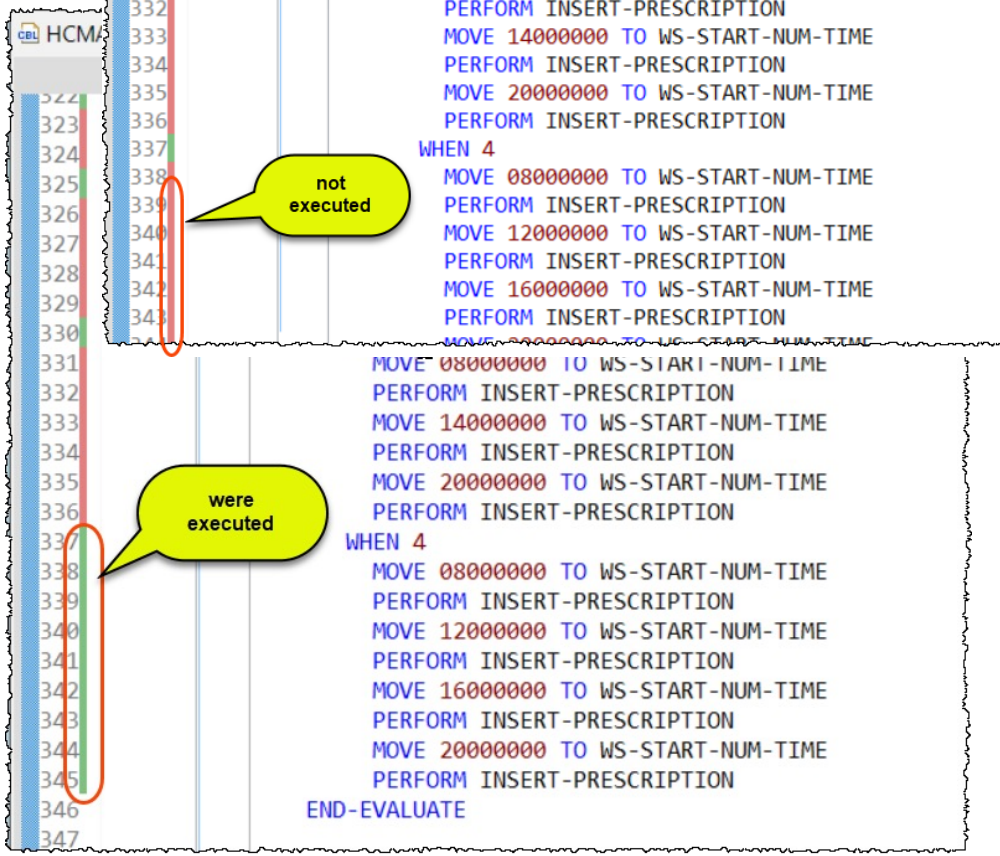


Uses IDz/DBB to perform a “User Dependency Build”, run the Unit Test (zUnit), **Code Coverage** and verify that the abend is fixed

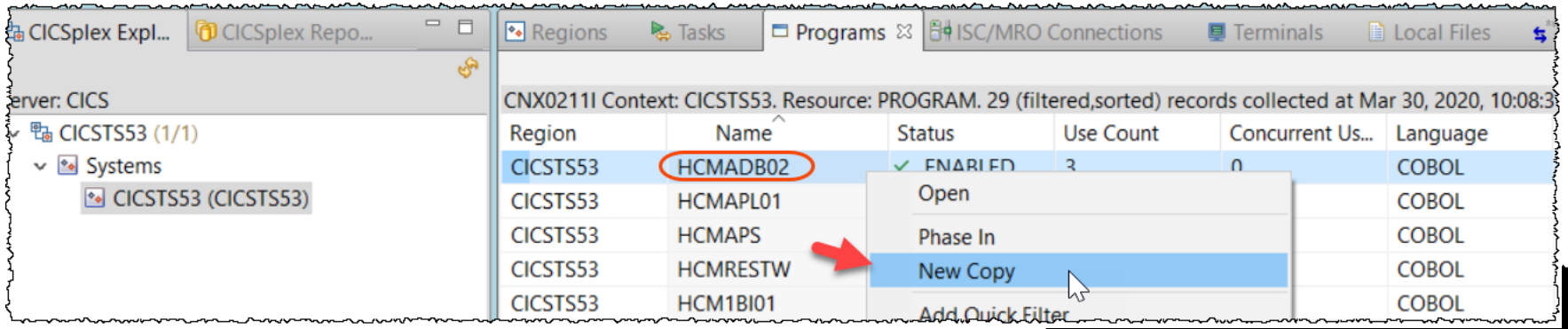


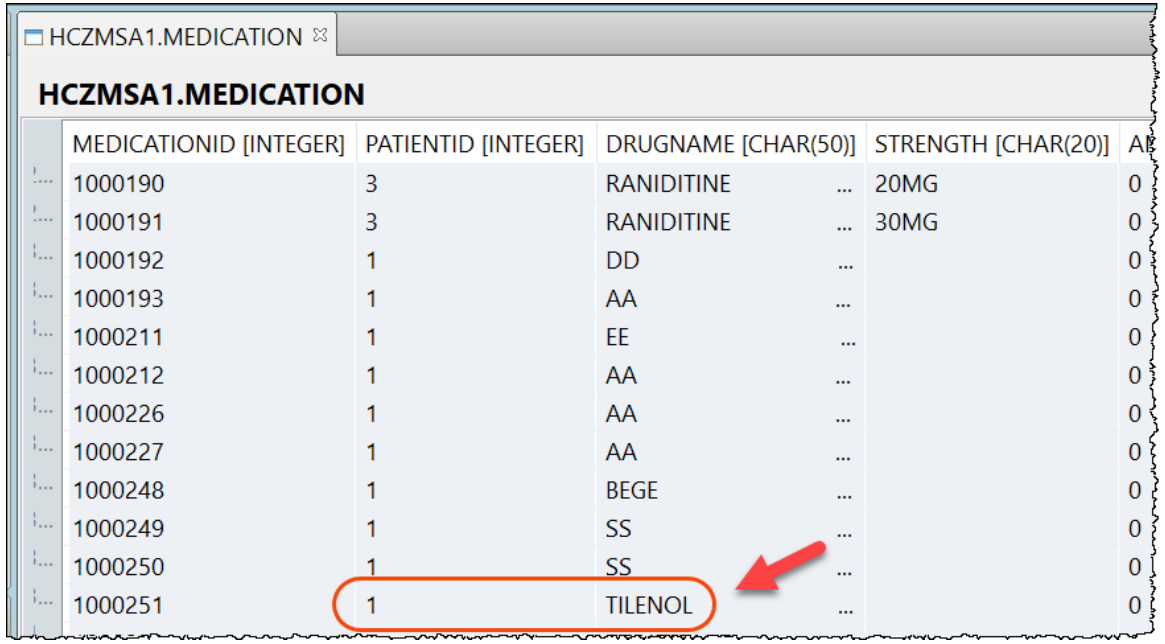


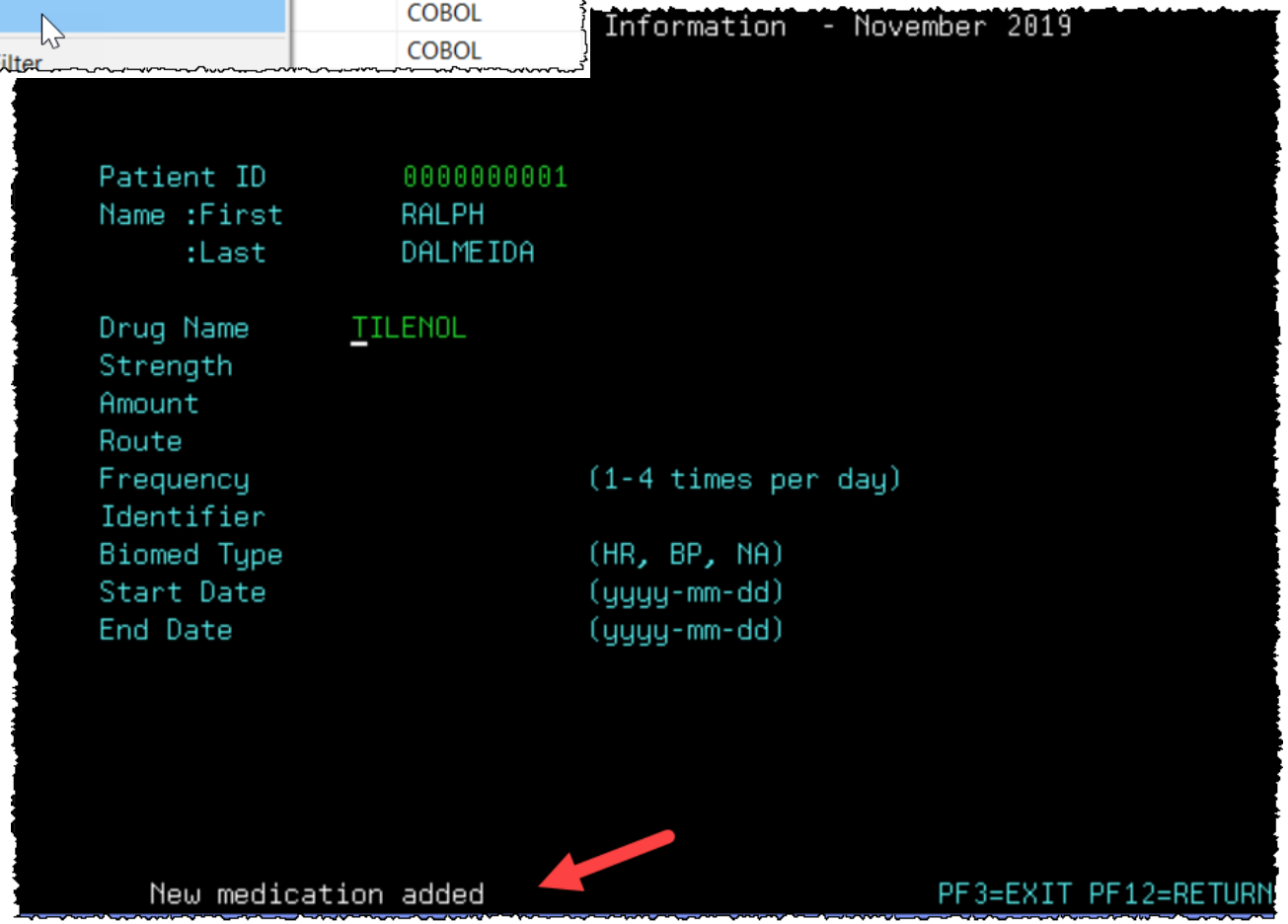




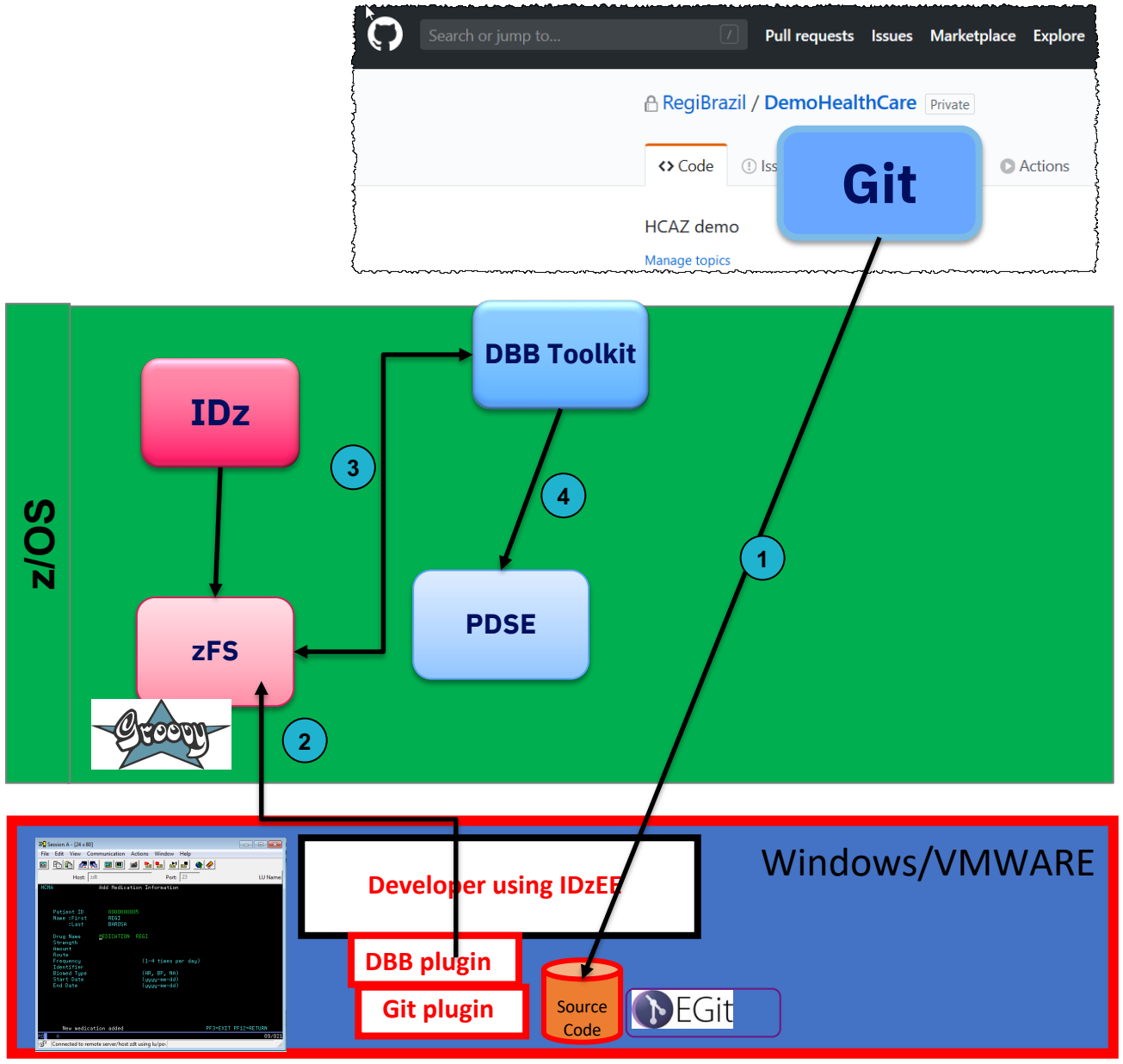
Uses IDz/DBB to perform a “User Dependency Build”, run the Unit Test (zUnit), Code Coverage and **verify that the abend is fixed**







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.

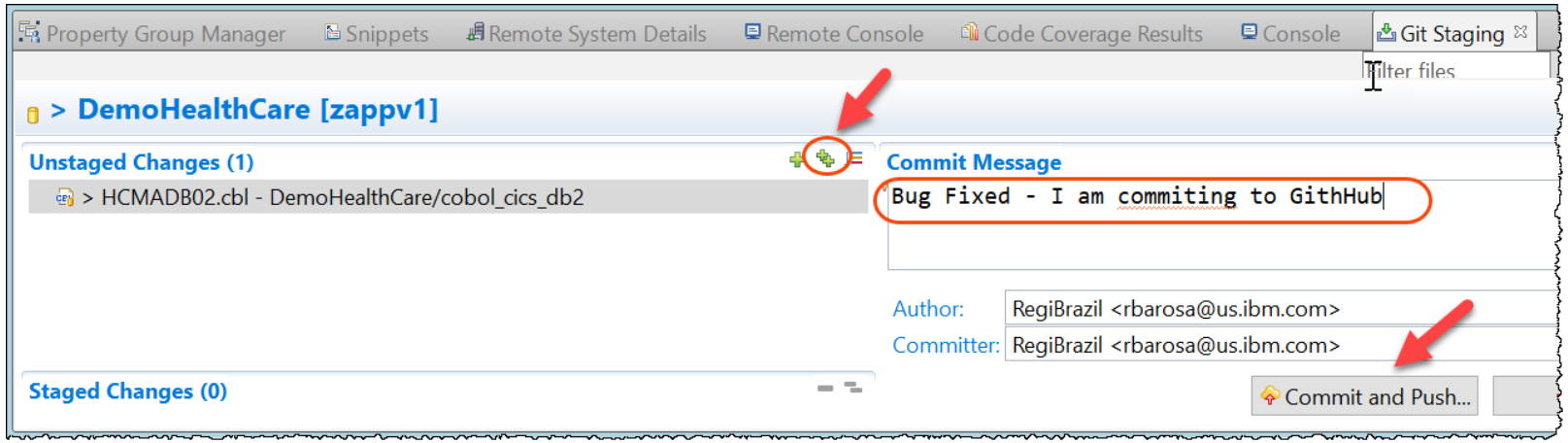
PART #2 - Developer see the CICS abend cause and fix it.

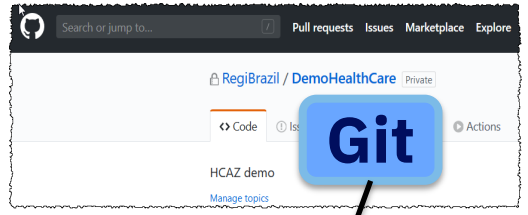
Bob – Mainframe Developer
Find the CICS issue, performs the COBOL updates, test the fix

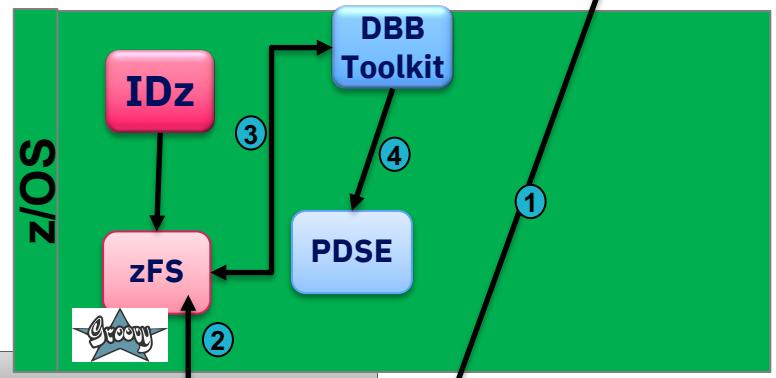
Objective: Fix the CICS Abend

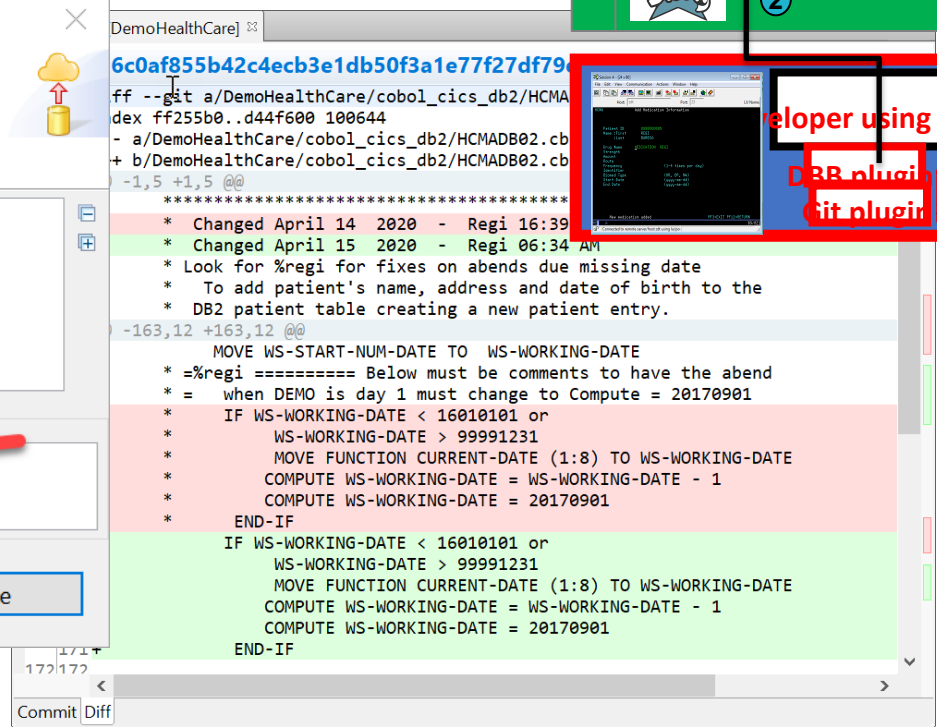
1. Uses **CICS** to verify the issue.
2. Uses **IDz/Fault Analyzer** to identify what is causing the abend
3. Uses **IDz/Debug** to debug the program and verify the error
4. Uses **IDz** to update the COBOL program loaded from **GIT**
5. Uses **IDz/DBB** to perform a "*User Dependency Build*", run the Unit Test (**zUnit**), **Code Coverage** and verify that the abend is fixed
6. Uses **IDz/Git** to **Commit and Push** the corrected code to Repository

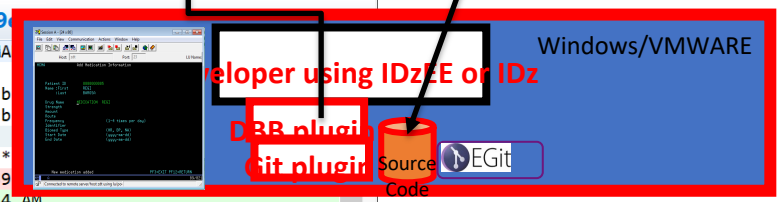
Uses **GIT** to Commit and Push the corrected code to Repository









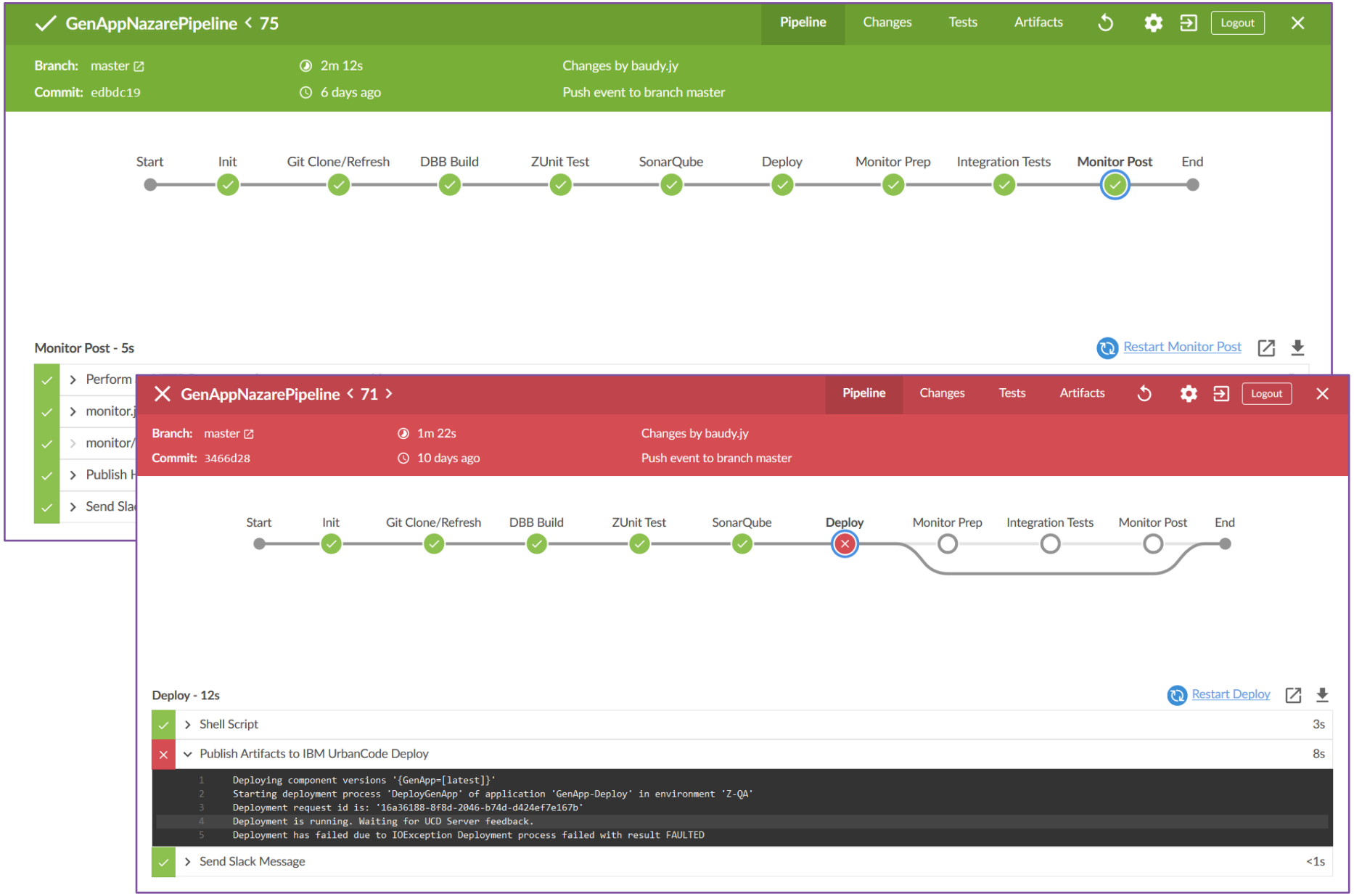


PART #3 - Release Engineer Deploy to z/OS and Mobile.

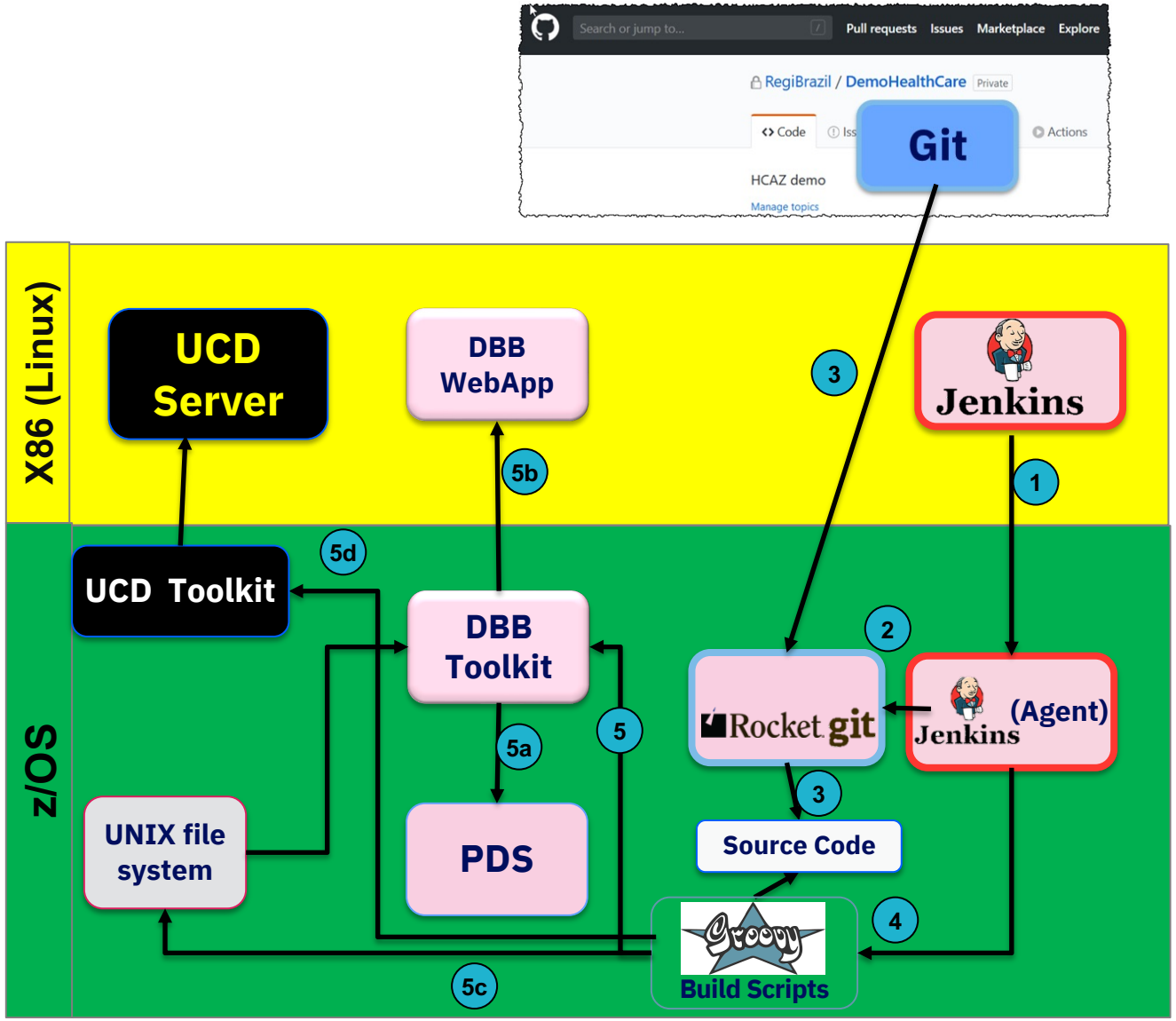
Rebecca – Release Engineer
Executes the Team Building using UCD and Jenkins

1. Uses **Jenkins** to perform the “team” building Unit Test and deploy using **UrbanCode Deploy (UCD)**.
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



High Level Demo Diagram using **Jenkins-Git-DBB-UCD**(UrbanCode Deploy)



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

[2] **Jenkins** agent issues **Git pull** command to update **zOS Rocket Git** repository (get latest updates).

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

[4] **Jenkins** agent invokes build scripts containing **DBB** APIs using zOS Git repository (Rocket).

[5] **DBB Toolkit** provides Java APIs to:
[5a] Create datasets, copy source from **zFS** to **PDS**, invoke zOS Compiler/Linkage Editor.

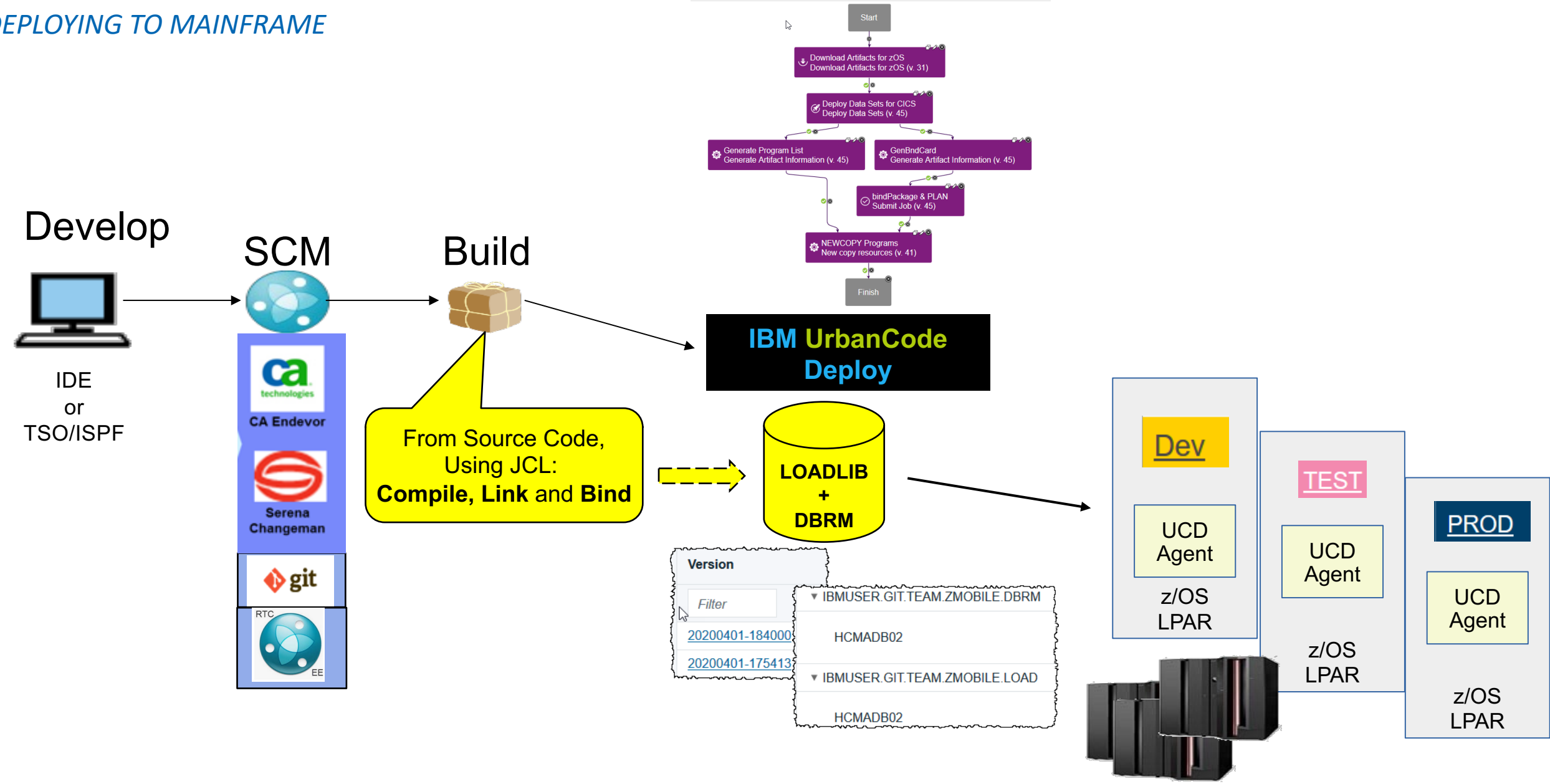
[5b] **DBB** scan and store dependency data from source files, perform dependency and impact analysis, store build results on DBB WebApp .

[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)

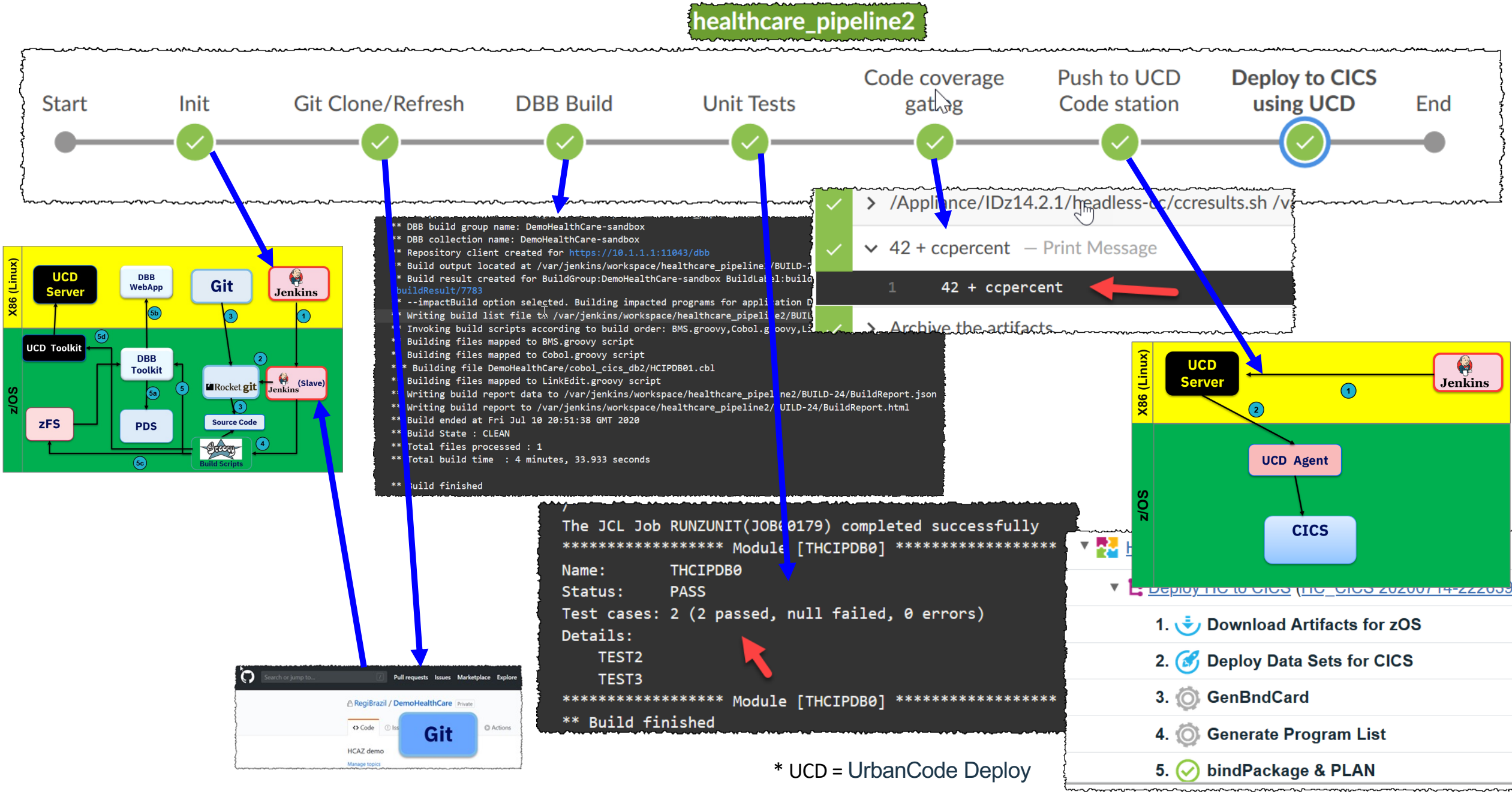
URBANCODE DEPLOY FOR DEPLOYMENT AUTOMATION

DEPLOYING TO MAINFRAME



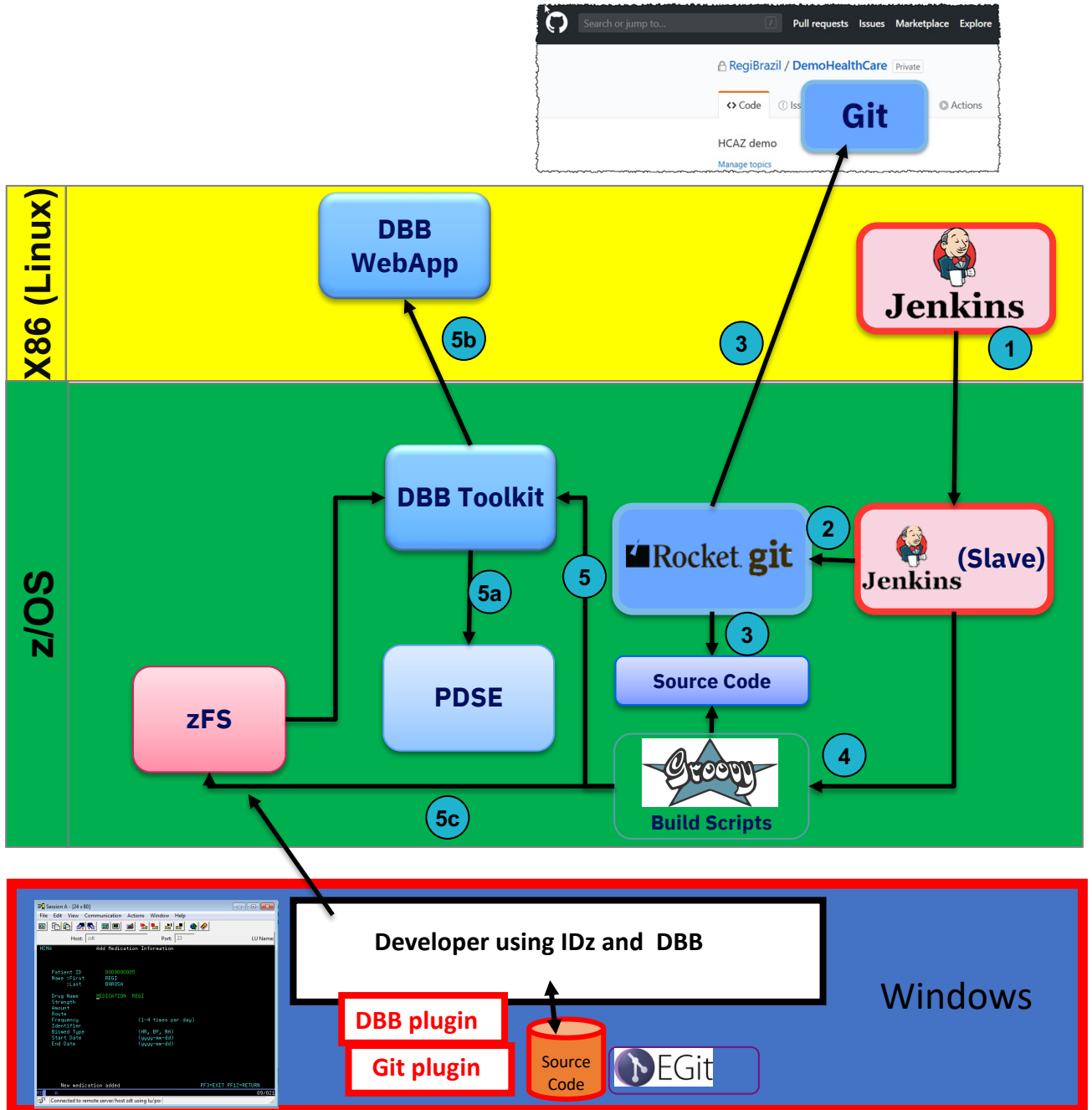
* RTC → IBM® Engineering Workflow Management

Demo High Level Diagram using Jenkins-Git-UCD



* UCD = UrbanCode Deploy

Solution High Level Diagram using Jenkins-Git-DBB



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

[2] **Jenkins** agent issues **Git pull** command to update **zOS Rocket Git** repository (get latest updates).

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

[4] **Jenkins** agent invokes build scripts containing **DBB** APIs using zOS Git repository.
Example scan changed **DBB WebApp** to find dependencies (5b))

[5] **DBB** Toolkit provides Java APIs to:
[5a] Create datasets, copy source from zFS to PDS, invoke zOS programs, issue ISPF and TSO commands, submit JCL.
[5b] **DBB** scan and store dependency data from source files, perform dependency and impact analysis, store build results on DBB WebApp.
[5c] Copy logs from PDS to zFS, generate build reports (can be saved in build result).

Agenda - Day 1

2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)

3:00 LAB 1 - Working with z/OS using COBOL and DB2

or

LAB7 - IBM DBB with Git, Jenkins and UCD on Z

5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

2:00 zUnit and VTP Overview (Wilbert)

2:15 Demo: Scenario using Z DevOps solutions including **ADDI**, **zUnit**, **Git** and **Jenkins** (David/Regi)

3:00 – 4:50 Choose one Optional lab:

- → LAB 7 - IBM **DBB** with **Git**, **Jenkins** and **UCD** on Z
- → LAB 6B : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Remote Assets**
- → LAB 6C : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Local Assets**
- → LAB 6D : Using **IBM zUnit** to Unit Test a COBOL/DB2 **batch** program
- → 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**)
- → LAB 9 - Using **IBM Z VTP** to test a COBOL /CICS / DB2 transaction
- → LAB 10 - Deploying COBOL/CICS/DB2 application using a **GitLab CI** Pipeline
- → LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- → LAB 12 – Using **ADDI** to find CICS programs to be converted to API's
- → LAB 13 – Running Integration Tests using **Galasa**

5:00 Checkpoint - Comments and Closing Day 2 (David/Ron/Wilbert/Regi)

Lecture Labs

Agenda - Day 1

2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)

3:00 LAB 1 - Working with z/OS using COBOL and DB2

or

LAB7 - IBM DBB with Git, Jenkins and UCD on Z

5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

(Wilbert)

2:15 Demo: Scenario using Z DevOps solutions including **ADDI**, **zUnit**, **Git** and **Jenkins** (David/Regi)

3:00 – 4:50 Choose one Optional lab:

- → LAB 7 - IBM **DBB** with **Git**, **Jenkins** and **UCD** on Z
- → LAB 6B : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Remote Assets**
- → LAB 6C : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Local Assets**
- → LAB 6D : Using **IBM zUnit** to Unit Test a COBOL/DB2 **batch** program
- → 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**)
- → LAB 9 - Using **IBM Z VTP** to test a COBOL /CICS / DB2 transaction
- → LAB 10 - Deploying COBOL/CICS/DB2 application using a **GitLab CI** Pipeline
- → LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- → LAB 12 – Using **ADDI** to find CICS programs to be converted to API's
- → LAB 13 – Running Integration Tests using **Galasa**

5:00 Checkpoint - Comments and Closing Day 2 (David/Ron/Wilbert/Regi)

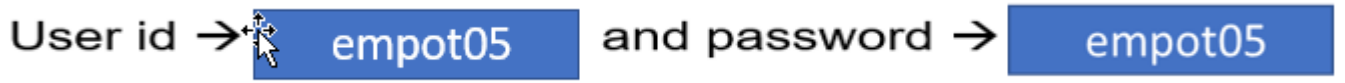
Lecture Labs

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 **J05P** 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/link 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 6B: Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Remote Assets (z/OS)

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.

.

Code on z/OS PDS
Using *traditional JCL* to compile/link

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 **HCAZ** 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. **Introduce a bug in 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.

6. **Run the unit test from a batch JCL** .
   You will run the unit test from a Batch JCL and observe a similar test case result.

Lab 6C: Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Local Assets

In this lab you will record interaction with a COBOL CICS/DB2 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.

.

Code on developer Workspace
Using *IBM DBB* to compile/link

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 **HCAZ** to become familiar with the Application that you will be recording.

2. **Record interaction with the application.**
   → You will record an interaction with the COBOL CICS/DB2 program.

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

4. **Introduce a bug in 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.

5. **Run the unit test from a batch JCL** .
   You will run the unit test from a Batch JCL and observe a similar test case result.

Lab 6D: Using IBM zUnit to Unit Test a COBOL/DB2 batch program

In this lab you will record interaction with a COBOL/DB2 program via JCL execution 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 COBOL/DB2 program, and rerun the unit test.

Overview of development tasks

Using *IBM DBB* to compile/link/bind

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

PART #1 – Unit test on COBOL/DB2 program DB2BATCH and introduce a bug

1. **Run the COBOL/DB2 batch program using JCL**
   → You will submit a JCL to execute a COBOL/DB2 program that calls a COBOL subprogram to print a small report.

2. **Use zUnit to record the batch execution.**
   → You will record the batch execution using the COBOL/DB2 program and subprograms.

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

4. **Modify the COBOL/DB2 program (introduce a bug) and rerun the unit test**
   → You will modify the COBOL/DB2 program introduce a bug, rerun the unit test, and observe the failure of the test case

PART #2 – Fix COBOL/DB2 program DB2BATCH and re-run the Unit test

5. **Run the batch program using the provided JCL and verify the bug .**
   → You will run the Batch JCL and observe the bug on the printed report..

6. **Use IDz to fix the bug, recompile the COBOL/DB2 program**
   → The bug is fixed using IDz, program fixed is rebuilt

7. **Rerun the zUnit and verify that the bug is eliminated**
   → When the zUnit test case is executed, you can verify that the program is fixed.

Lab 9: Using IBM Z VTP to test a COBOL /CICS / DB2 transaction

In this lab you will record interaction with a CICS COBOL/DB2 program via IBM Z VTP and save the recorded data into a playback file. You will then introduce a bug to the program and rerun the test to expose the bug.

Overview of development tasks

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

Using *IBM DBB* to compile/link

PART #1 – Record existing COBOL/CICS/DB2 transactions and introduce a bug

1. **Use VTP to record the CICS/DB2 application in action**
   → You will record the CICS application in action. This application starts multiple CICS transactions using COBOL programs with and without DB2 access.

2. **Run a JCL to execute the recorded replay sequence**
   → You will submit a JCL to execute the sequence of interactions that you recorded and verify the results.

3. **Modify one COBOL/DB2 program (introduce a bug) and rerun the VTP JCL**
   → You will modify the COBOL/DB2 program to introduce a bug, rerun the replay JCL, and observe the failure due the introduced bug.

PART #2 – Fix the program bug and re-run the test

4. **Run the CICS transaction and verify the bug**
   → You will run CICS application and observe the bug introduced.

5. **Use IDz to fix the bug, recompile the COBOL/DB2 program**
   → The bug is fixed using IDz, program fixed is rebuilt.

6. **Rerun the VTP JCL and verify that the bug is eliminated**
   → When running the JCL recorded replay you can verify that the program is fixed.

Lab 10: Deploying COBOL/CICS/DB2 application using a GitLab CI Pipeline

This lab will take you through the steps of using GitLab CI along with DBB, ZUnit and UrbanCode Deploy (UCD) on z/OS.On this lab you will fix a bug of an existing COBOL/CICS application stored in GitLab.
You will use IDz to change the code and perform a personal test for later delivery and commit to GitLab and then use GitLab CI for the final build and continuous delivery. The updated code will be deployed to CICS using UrbanCode Deploy (UCD)
.

Overview of development tasks

1. **Review the GitLab issue and verify the bug using the 3270 terminal**
   → You will start GitLab, verify the issue and using a 3270 emulation execute a transaction named **SSC1** to become familiar with the Application that you intend to modify.. When using customer number 1 notice that the DOB is 0000-00-00. This is a bug that needs to be fixed

2. **Load the source code from GitLab to the local IDz workspace**
   → using IDz you will clone the GitLab loading all COBOL source code to your windows client.

3. **Use IDz to run the zUnit test case and verify the error**
   Running zUnit you will verify that the DOB is incorrect,

4. **Modify the COBOL/CICS/DB2 program that has the bug using IDz.**
   → Using *IDz* you will modify the COBOL program LGICDB01 to fix the bug.

5. **Use IDz *DBB User Build* to compile/link and run the zUnit**
   → You will compile and link the modified code using the *DBB User Build* Function.
   When complete you will run zUnit generated test case and verify that the bug is fixed.

6. **Push and Commit the changed code to GitLab .**
   → You will commit the changes to *Git*. This will initiate the GitLab CI pipeline

7. **Verify the GitLab CI Build execution.**
   → You will use Gitlab to build the new changed code, run zUnit and push the executables to be deployed using *UCD*. .

8. **Verify the GitLab CI Package and Deploy to UCD and test the CICS transaction again using 3270**
   → You will verify the results after the final deploy to *CICS* using *UCD*

Lab 11 – Migrating partitioned data sets (PDS) to a distributed Git repository.

Most DBB users will install the Rocket Software port of the Git client to z/OS as part of their DevOps solution.

To facilitate the move to Git as an SCM for z/OS development, you can use a DBB migration tool that copies source code from partitioned data sets (PDS) to a local Git repository on the z/OS File System (zFS) to be committed and pushed to a distributed Git server.
More details at https://www.ibm.com/docs/en/adfz/dbb/1.1.0?topic=migrating-data-sets-git

Overview of development tasks

.

1. **Logon to IDz and prepare the necessary directories and script migration file on zFS.**
   → You will start IDz and prepare some folders and the script migration file required for the migration on z/OS UNIX Files.

2. **Migrating using DBB migration tool to a local Rocket Git client repository on zFS**
   → You will use the DBB migration tool, and the Rocket Git client installed on USS system and create a client git.

3. **Push the migrated files to Git server (Gitlab)**
   → When all source files have been copied to the local Rocket Git repository, you are ready to push the files to the Git server as follows,

4. **Load the source code from Git to the local IDz workspace and start working with the migrated assets**
   → Once the migration is complete on Git server you may clone using IDz and start working with the migrated assets.

Lab 13 – Running Integration Tests using Galasa

This lab will take you through the steps needed to execute some sample Integration tests using **Galasa** against a sample mainframe application using IDz as IDE.
In this lab we will use the **SimBank** example that is provided through the Galasa eclipse plugin. This example provides a **sample mainframe style application** as well as a set of tests that can be executed to test the application.

Overview of development tasks

PART #1 –
Explore the *SimBank* application and execute the Installation Verification Test (IVT)

1. Start the example application SimBank from within IBM Developer for zOS
2. Use IBM Personal Communications to log onto the sample application
3. Run the IVT test to validate that Galasa can connect to the SimBank Application
4. Open the Test Result Editor

PART #2 –
Execute a web service and 3270 integration test

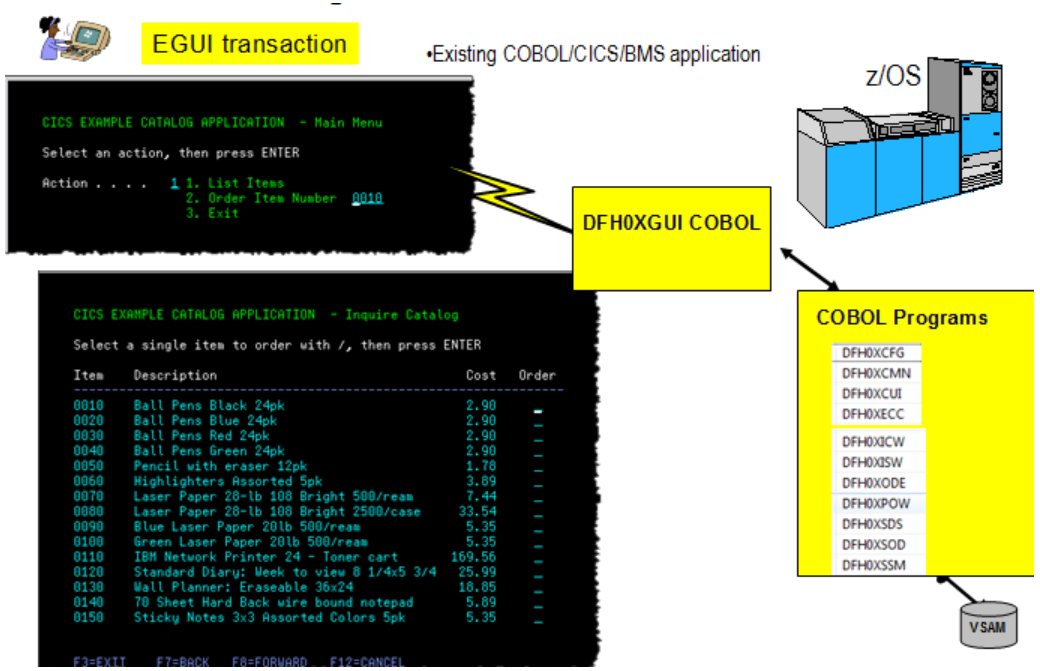
1. Execute a mixed web service and 3270 test within the Galasa framework
2. Examine the source code to see how the Galasa test works
3. Examine the Run Editor to see the stored artifacts
4. Change the test to log the request and response for the web service

PART #3 –
Execute a Batch test

1. Execute a batch test within the Galasa framework
2. Examine the source code to see how the Galasa test works
3. Examine the Run Editor to see the stored artifacts

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:

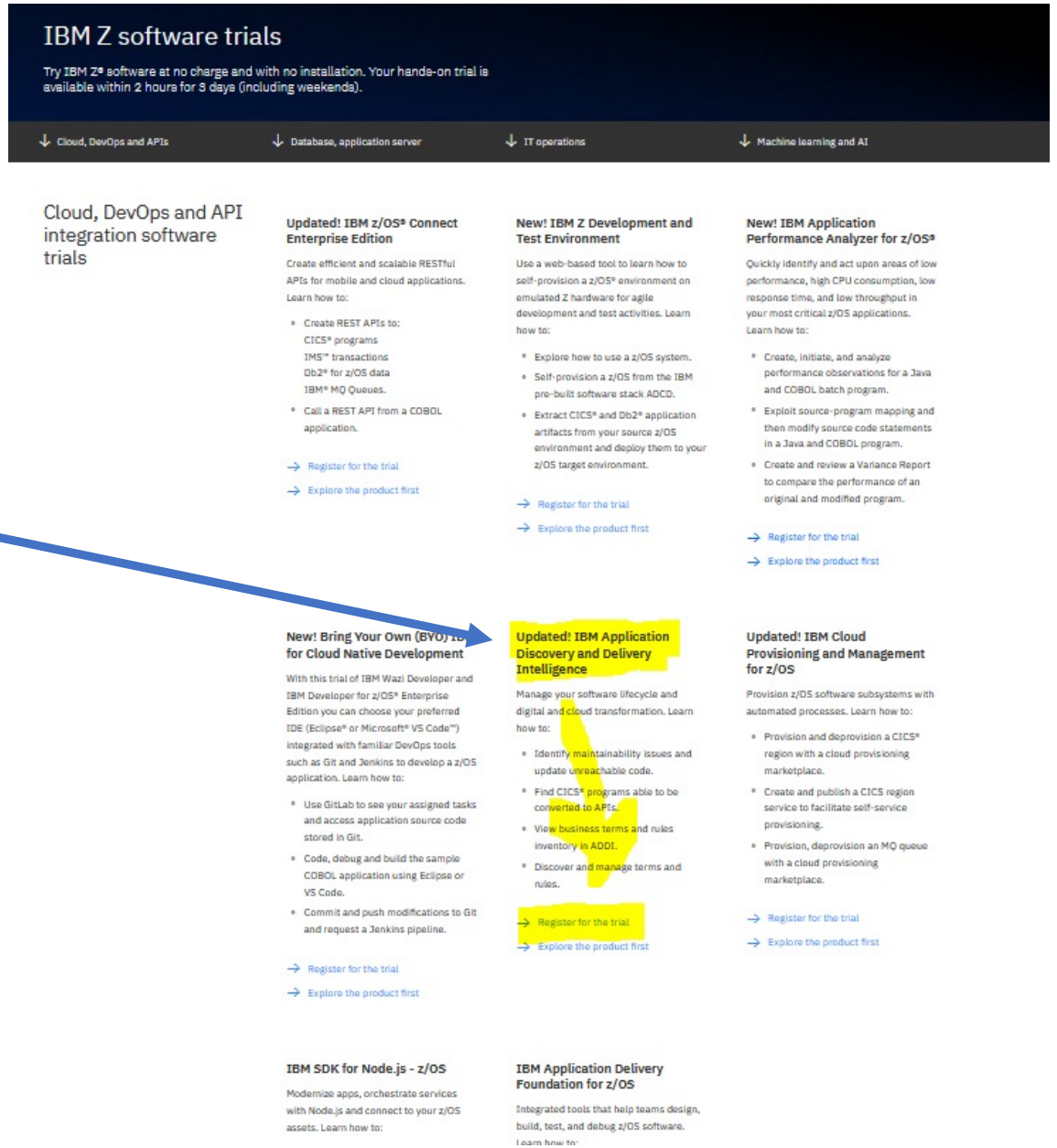
1. **Create a new observation request for a job that is not running yet**
   → Using *ADFz* you will create an *APA* observation request.

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.

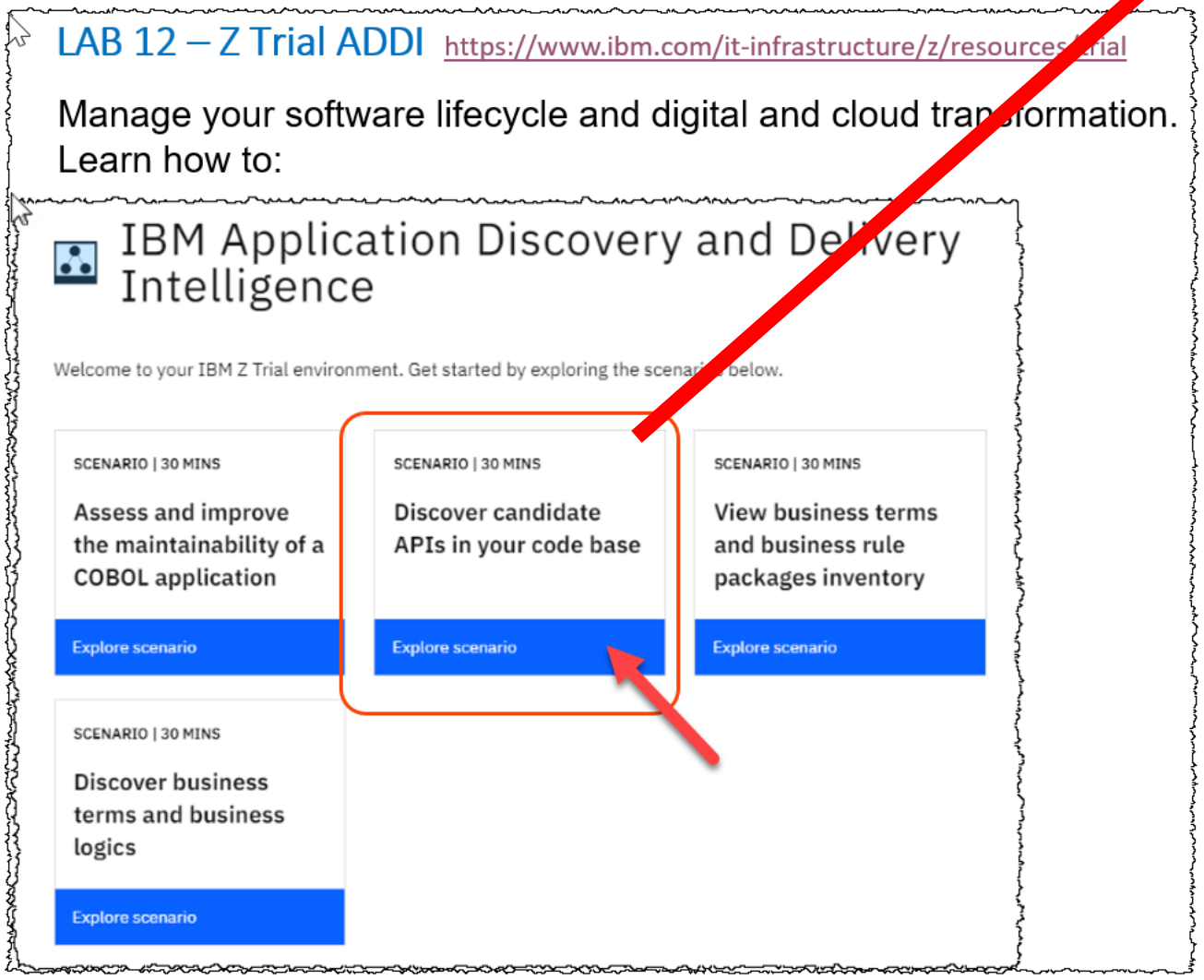
If interested in doing the ADDI Lab (LAB 12)

- You MUST request an ADDI zTrial instance
  1. **Open IBM zTrial registration page**
     https://www.ibm.com/it-infrastructure/z/resources/trial
  2. Click on **Register for the trial**
  3. Login or Create a new IBMID
  4. Select the Application Discovery and Delivery Intelligence zTrial
  5. Look for emails for when your instance is ready



LAB 12 – Z Trial ADDI https://www.ibm.com/it-infrastructure/z/resources/trial

Follow workbook instructions..



Exposing and leveraging APIs has become a critical component of the modern IT business model.
The first step in this process is discovering what potential API candidates exist in your codebase. This scenario guides you through the discovery process in roughly 30 minutes. By the end of the session, you will know how to perform the following tasks:

- Use IBM **ADDI** to understand the architecture of a CICS application.
- Create graphs to find individual components of the application that would be good API candidates.
- View a copybook to understand parameters for mapping the API.

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 “Free education”

https://ibm.github.io/mainframe-downloads/Training/idzrdz-remote-training.html

Overview

This year we are presenting several iterations of the complete (no-charge) IDz Entry Level Training courses, along with Experienced User topics such as **Code Review, Code Coverage, Integration with Open Source DevOps tooling, etc.**

The **Entry Level Training** consists of multiple hour-long hands-on-workshop modules that are delivered through **interactive, instructor-led web-conferences** ... an extremely effective training venue utilizing open Q&A, flexible class delivery, remote-mentoring, greater absorption of the material, etc.

These sessions cover all of the skills & knowledge needed to master IDz, enabling you for high-productivity z/OS COBOL application work with the tools. Please note that there is no need to register for any of the classes. Simply follow the **"Getting Started Notes"** instructions below.

Also note that you may attend any session you'd like - from any iteration at any time. **However the Entry Level classes are designed as sequenced educational content, with materials that build upon previous topics.** It is recommended that you proceed through the courses from Module-1 thru Module-9 (see details below on the individual module topics).

No-Charge IDz/ADFz Training offerings

https://ibm.github.io/mainframe-downloads/Training/idzrdz-remote-training.html

Home › IDz/RDz Remote Training

IDz/RDz Remote Training

Entry-Level Training - Module Content Details

+ Module 1: The RDz Workbench and introduction to Eclipse for ISPF Developers
+ Module 2: Editing Program Source
+ Module 3: Analyzing COBOL Programs
+ Module 4: Introduction to RDz Remote Systems' Features
+ Module 5: Dataset Access and Organization
+ Module 6: ISPF 3.x Options, Batch Job Submission & Management
+ Module 7: MVS Subprojects
+ Module 8: The DB2 and SQL Data Tools
+ Module 9: Debugging z/OS COBOL Applications using IDz/RDz

IDz/ADFz/ADDI Index of Product Training, Video and Blog Resources

https://community.ibm.com/community/user/ibmz-and-linuxone/viewdocument/idzadfzaddi-index-of-product-trai-2?CommunityKey=b0dae4a8-74eb-44ac-86c7-90f3cd32909a&tab=librarydocuments

This page contains links to the following kinds of IDz, ADFz, ADDi and z/DevOps standalone enablement resources:

- Video Recordings (classes, one-off topics, seminars/presentations, etc.)
- Links to Blogs
- PDFs and other deep-dive learning content
- New release announcements

Information about the IBM (no-charge) public training classes

Consider adding a link to this page to your Workspace. As more blogs, PDF
updated to include them.

IDz and Eclipse - Source Editing, Options and Techniques

— Overview - Terms and Concepts

- Overview of the ADFz and IDz Solution - Feature/Function PDF
- IDz Workbench Overview - A short product intro-video
- Introduction to IDz - Click for Click Workshops - "Labs.PDF"
- Workspace Design - Part of the IDz Product Rollout Framework

\+ Application Development and IDz Navigation Tools and Techniques

\+ ISPF Techniques for Editing and Browsing Source Files

\+ Courses for PL/I, Assembler, C, Java and Preprocessing 4GL Code for use with IDz

Questions / Comments

Agenda - Day 1

2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)

3:00 LAB 1 - Working with z/OS using COBOL and DB2

or

LAB7 - IBM DBB with Git, Jenkins and UCD on Z

5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

view (Wilbert)

2:15 Demo: Scenario using Z DevOps solutions including **ADDI**, **zUnit**, **Git** and **Jenkins** (David/Regi)

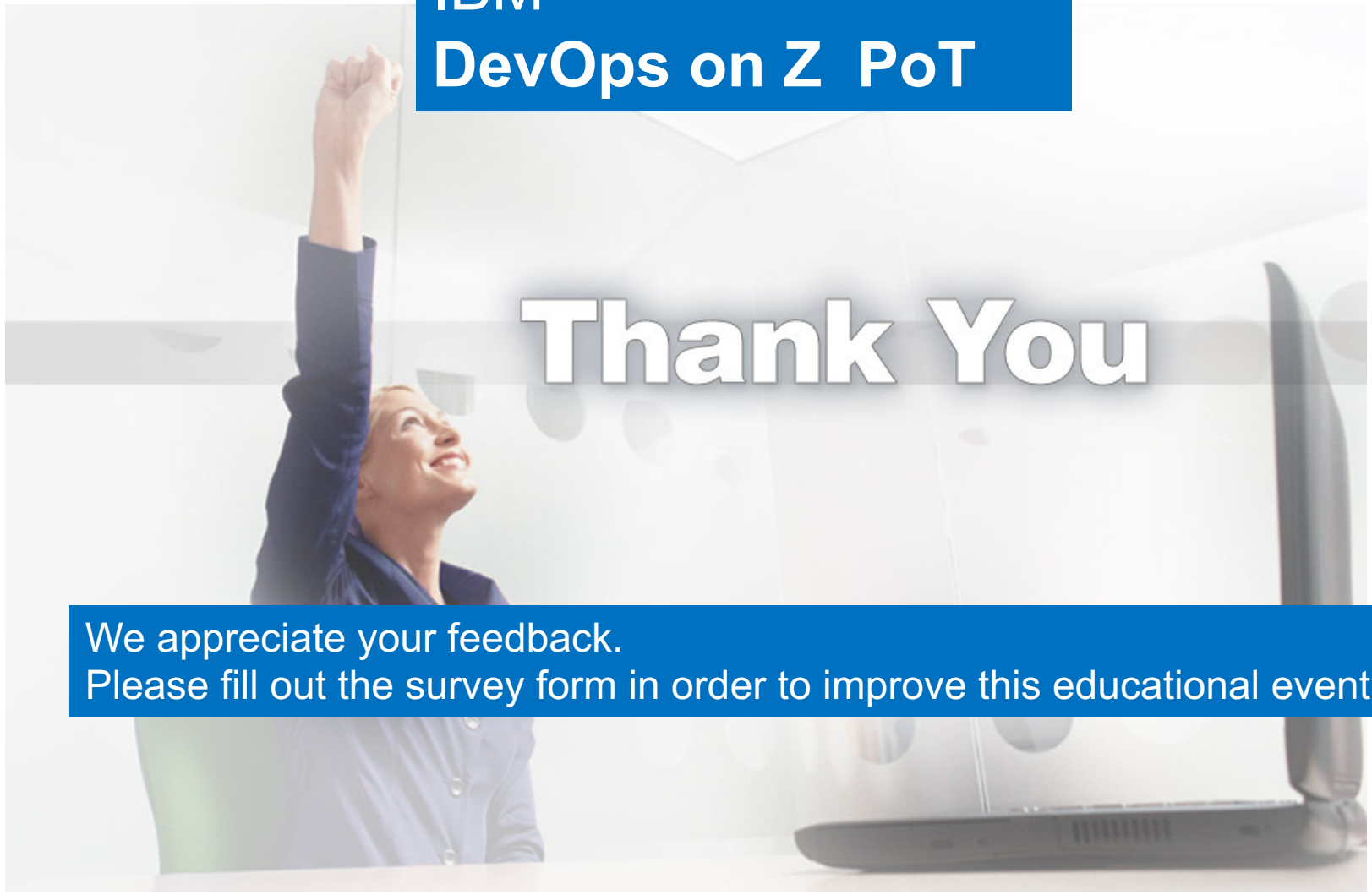
3:00 – 4:50 Choose one Optional lab:

- → LAB 7 - IBM **DBB** with **Git**, **Jenkins** and **UCD** on Z
- → LAB 6B : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Remote Assets**
- → LAB 6C : Using **IBM zUnit** to Unit Test a COBOL/**CICS**/DB2 program using **Local Assets**
- → LAB 6D : Using **IBM zUnit** to Unit Test a COBOL/DB2 **batch** program
- → 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**)
- → LAB 9 - Using **IBM Z VTP** to test a COBOL /CICS / DB2 transaction
- → LAB 10 - Deploying COBOL/CICS/DB2 application using a **GitLab CI** Pipeline
- → LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- → LAB 12 – Using **ADDI** to find CICS programs to be converted to API's
- → LAB 13 – Running Integration Tests using **Galasa**

5:00 Checkpoint - Comments and Closing Day 2 (David/Ron/Wilbert/Regi)

Lecture Labs

IBM
DevOps on Z PoT



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

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.

Why Git

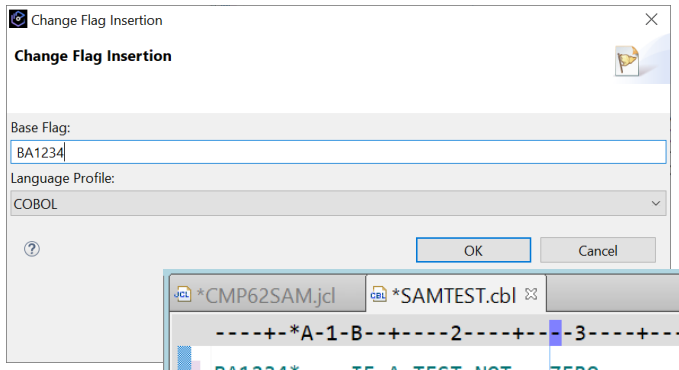
- Git is a fast and modern implementation of version control
  - Very lightweight
  - Extremely fast - The tool just gets out of the way
- Git is Distributed
  - Every developer has full copy, with history
  - No single point of failure
- Git facilitates collaborative changes to files
  - Enables branch-based workflows for multiple, parallel branches of development
  - Feature Branching capabilities that allow isolated environments for every change to codebase
  - Ensures that the master branch always contains production-quality code
  - Facilitates collaboration of changes and bringing those changes together into a single file

- Git provides a history of content changes
  - History for all of the files that make up a project, not just those of individual users.
  - Files could be Graphics, Designs, Documentation, Code, etc.

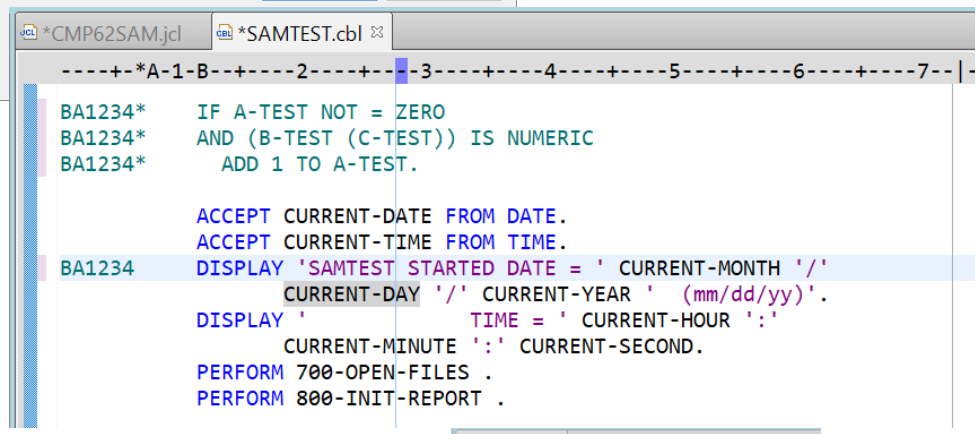
- Git is easy to use for any type of knowledge worker
  - Native environment is a Command Line terminal
  - There are UI extensions that can provide a pleasant user experience for version control.
  - Ex:
    - IBM Developer for z/OS (IDz) - eGit Perspective
    - Visual Studio Code (VSCode)

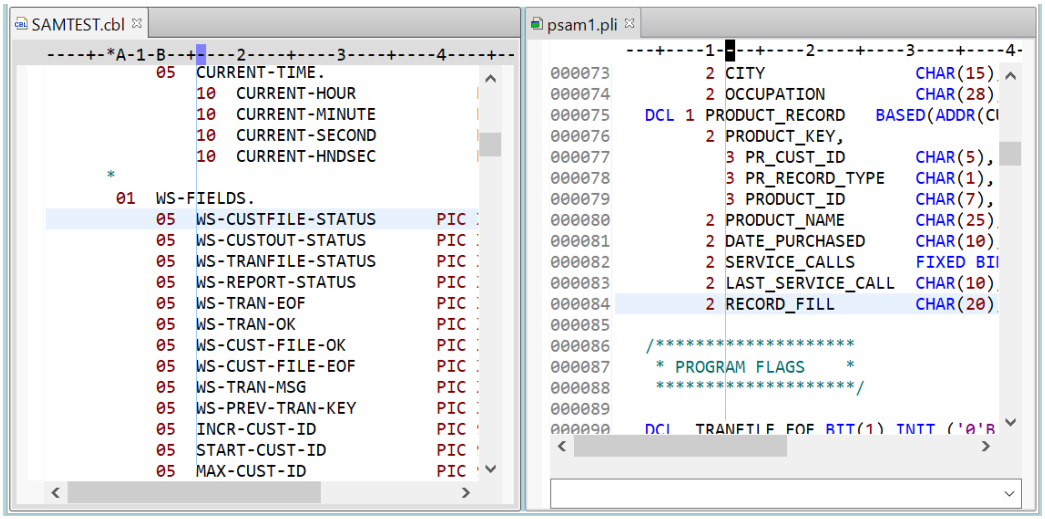
New editing capabilities

Analyze and Edit

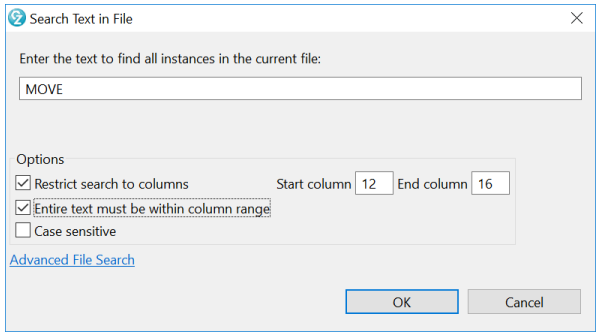


New! COBOL and PL/I editors now support Autocomment (14.2)

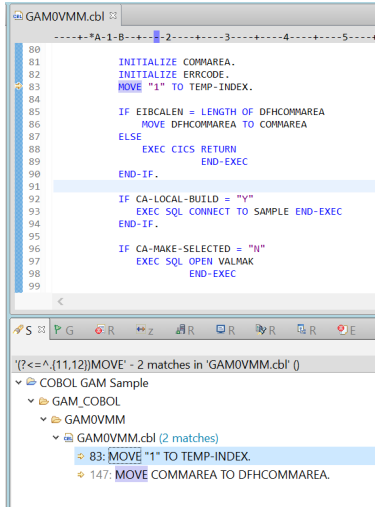


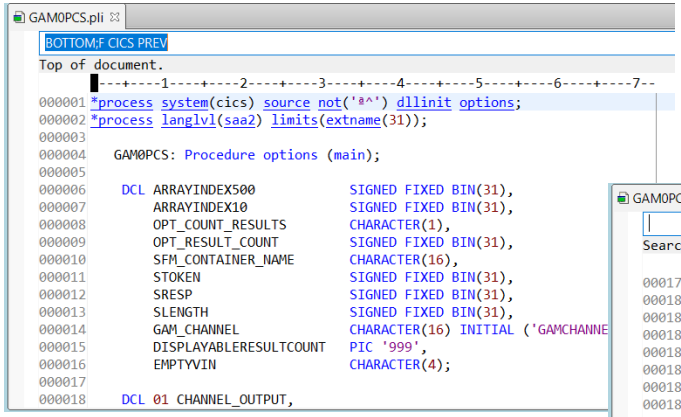


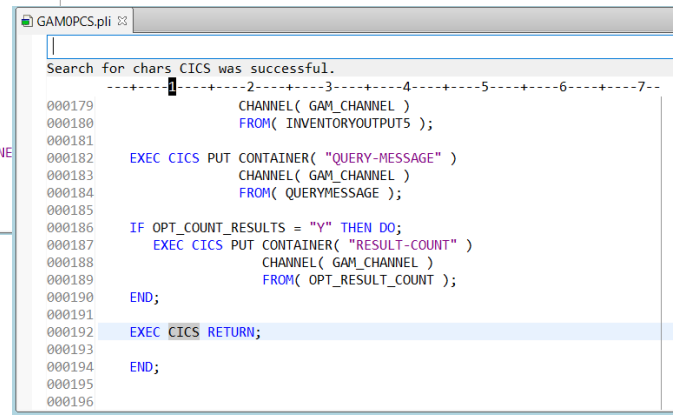
New! COBOL and PL/I editors offer a shortcut to toggle a vertical bar on/off in the editor (14.2)



New! Search file Text- specify columns

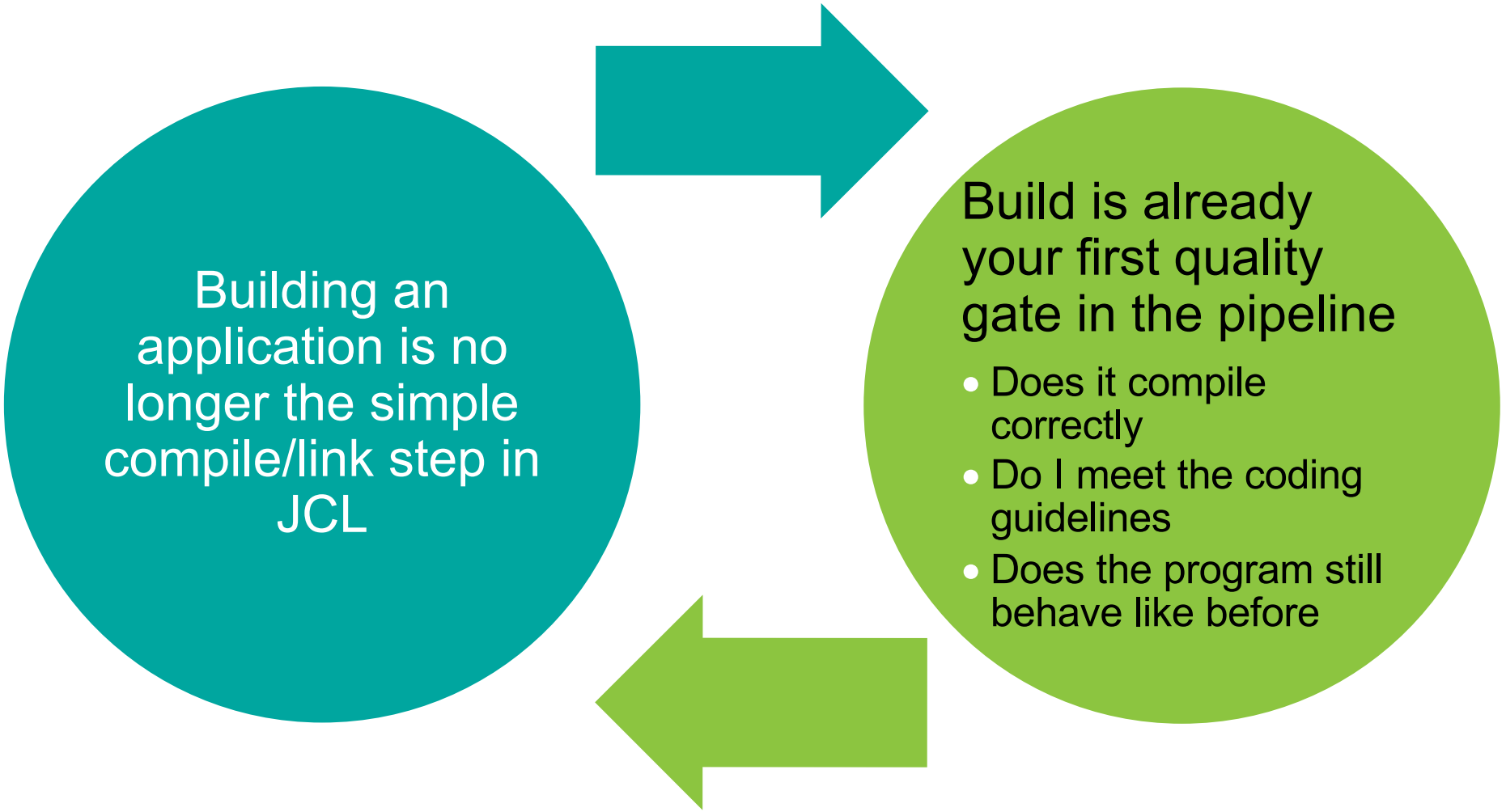






New! LPEX editor command line handles no space with multiple commands- similar to ISPF

Building vs. Compile/Link



Topology used on LAB 12

zD&T on Cloud

z/OS

DBB

DBB App server

Rocket git

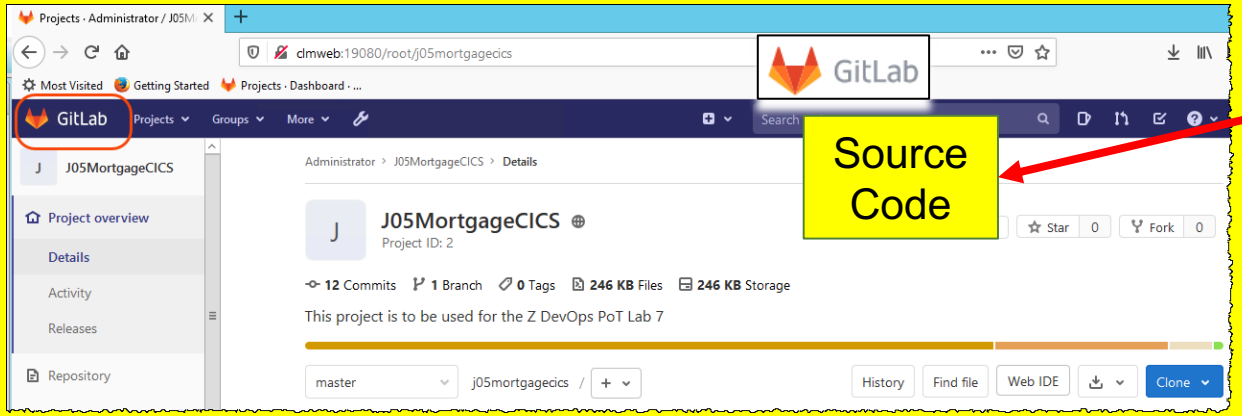
USS

Windows

Developer using ADFz

Git

Source Code



Linux