DevOps on Z PoT

Proof of Technology

DevOps Solution for Z Development

Regi Barosa
rbarosa@us.ibm.com

Ronald Geraghty
Ronald.Geraghty@ibm.com

Timothy Han
timothy.han@ibm.com

Wilbert Kho
wilbert@us.ibm.com

September 2020

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 skills are required
- COBOL, PL/I and 4GL programmers 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 unit testing and deploy.

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

Areas covered:

- Integrated application development and problem analysis (ADFz)
- Managing your source code using modern technology (including Git)
- Building automation (DBB) for COBOL or PL/I without a specific source code manager but using a pipeline automation tool (including Jenkins)
- Using a unit testing framework (zUnit) for COBOL or PL/I as part of the development and build environment.
- 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)

Agenda - Day 1

10:00 Introduction to DevOps on IBM Z - focus on open source tools (Ron)

10:40 LAB 1 - Working with z/OS using COBOL and DB2 (Tim)

or

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

12:50 Checkpoint - Comments and Closing Day 1 (Tim -Ron/Wilbert/Regi)

Agenda - Day 2

10:00 zUnit and Wazi VTP Overview (Wilbert)

10:15 Demo: Scenario using Z DevOps solutions including **zUnit**, **Git** and **Jenkins** (Regi)

11:00 – 12:50 Choose one Optional lab: (Tim)

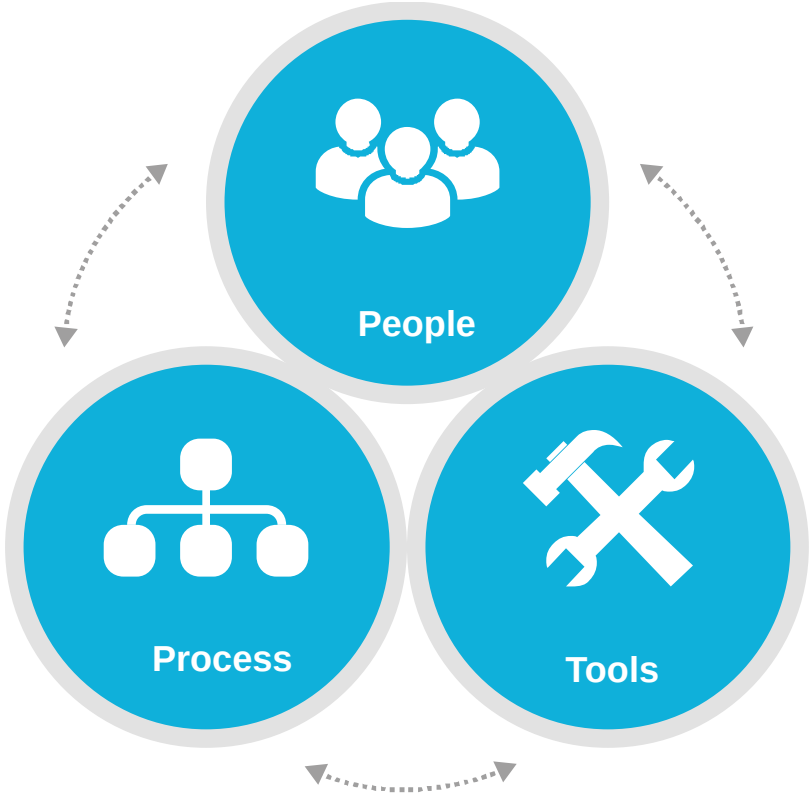
- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6B : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- → LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS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

12:50 Checkpoint - Comments and Closing Day 2 (Tim -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.

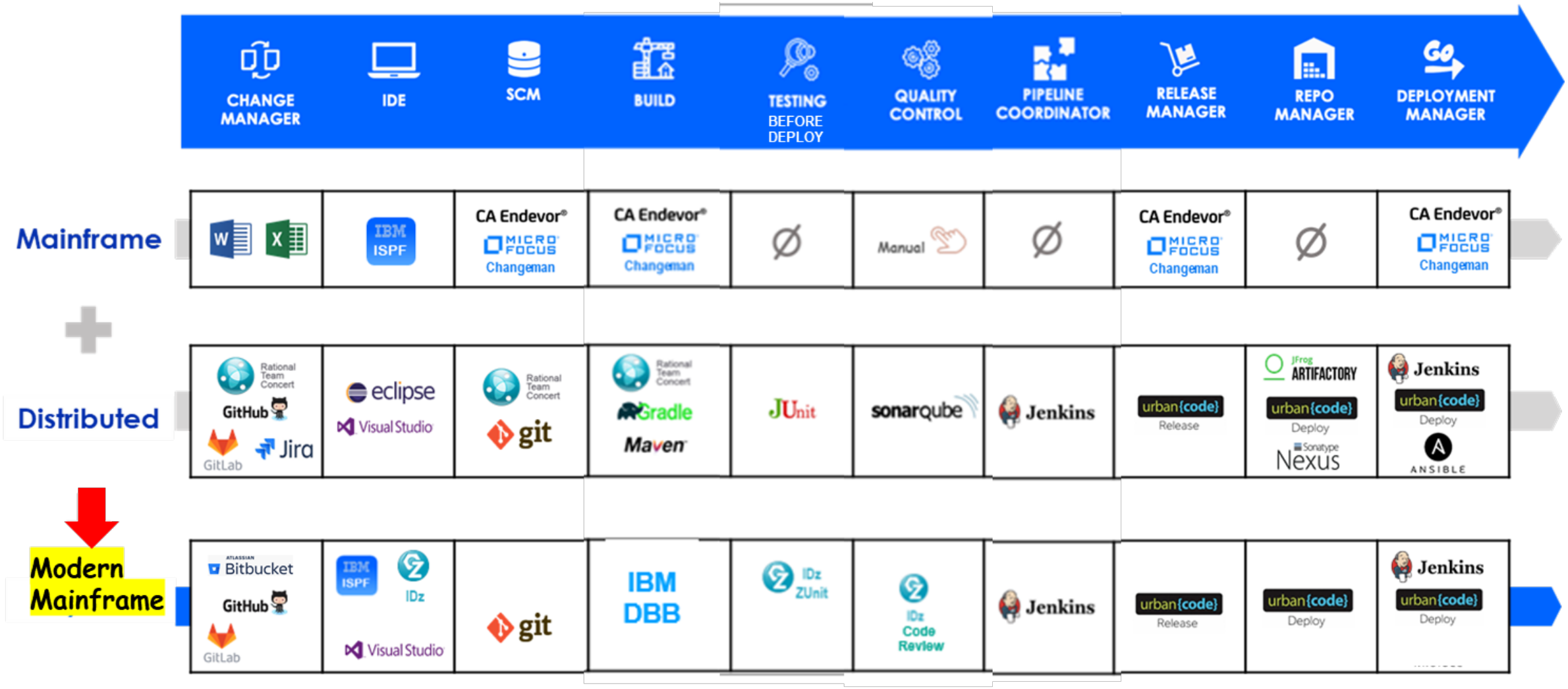
Why DevOps for z/OS?

- 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



∅ → Mainframe did not use this concept

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

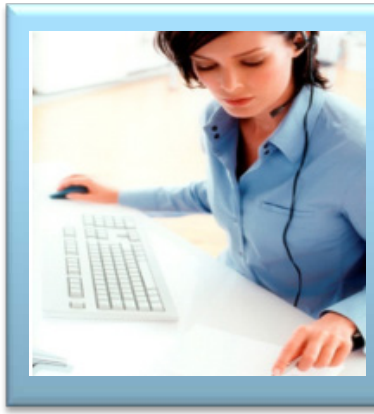
$80/defect

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.

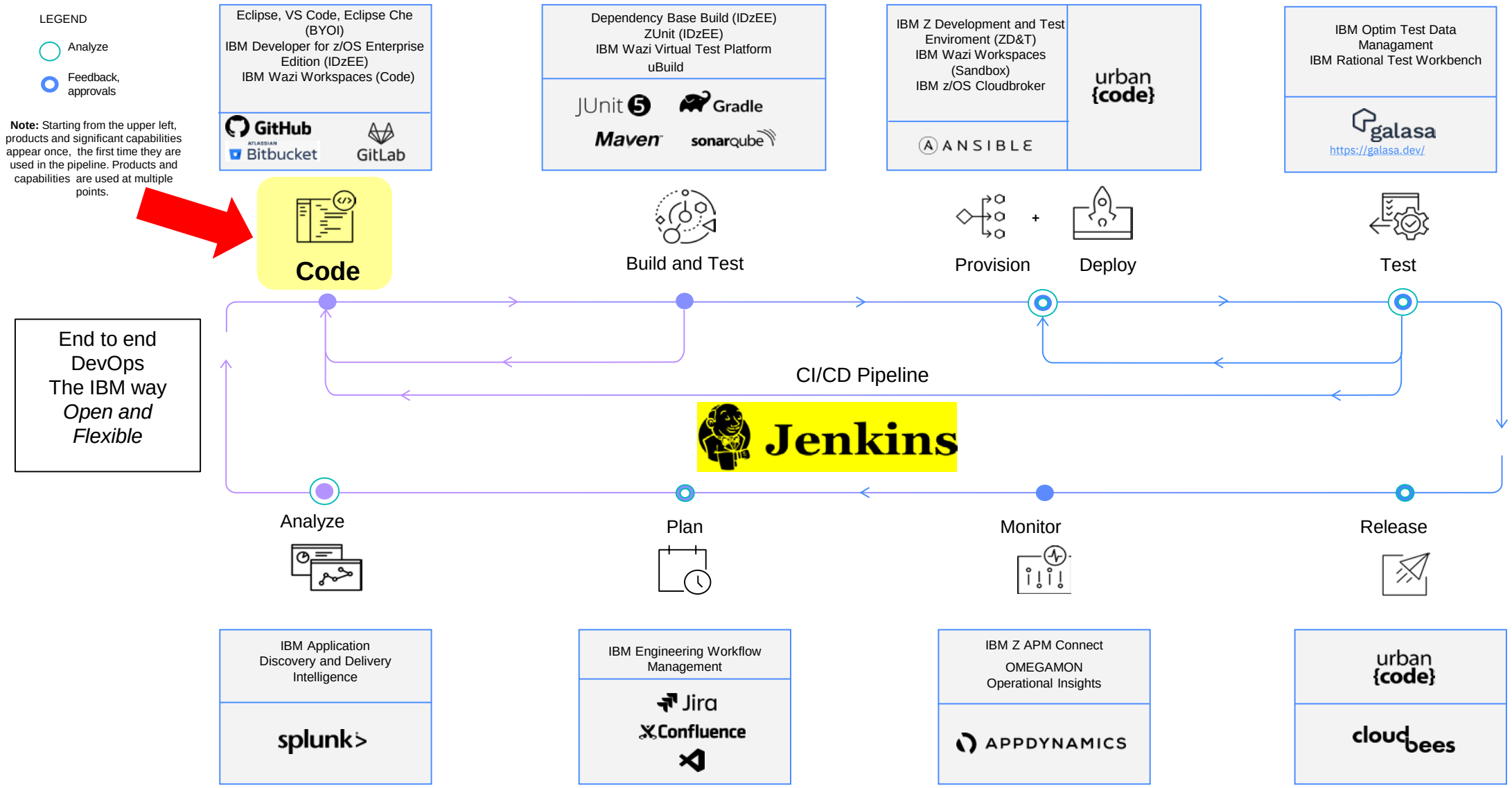
What is Jenkins?

"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



z/OS and subsystems

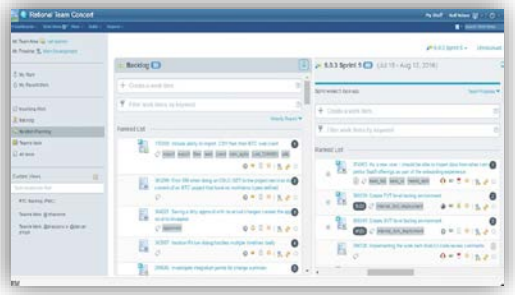
IMS

CICS

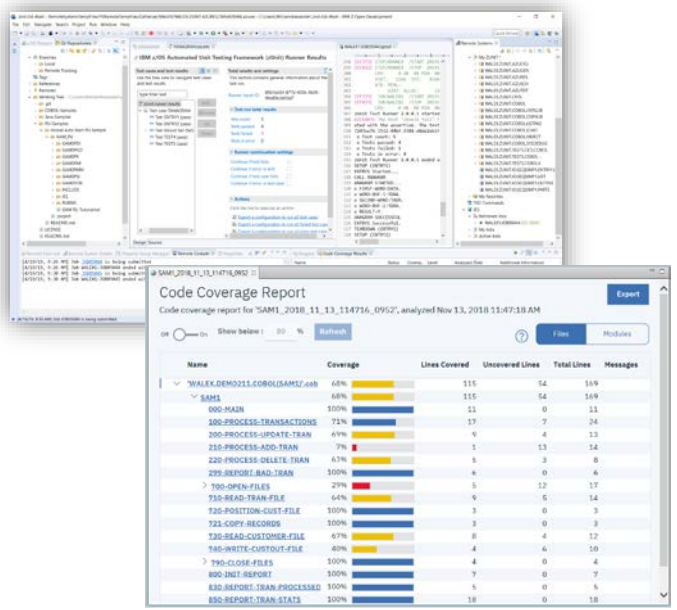
IBM Developer for z/OS Enterprise Edition- the developer's cockpit!

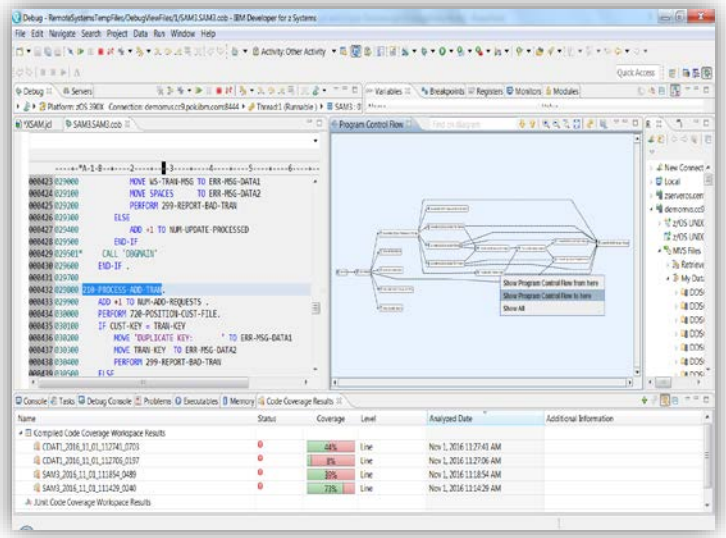
Build and SCM

Integration with Git for Lifecycle and Source Management

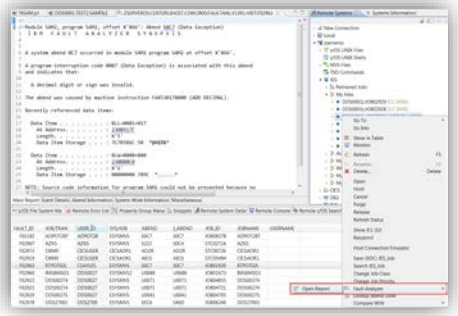


Unit test and code coverage

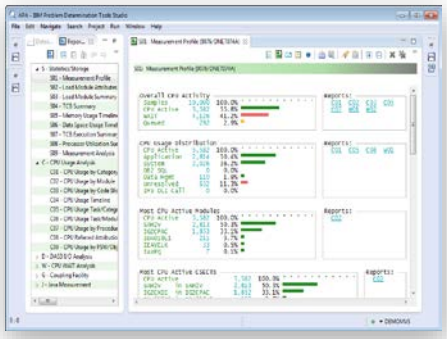




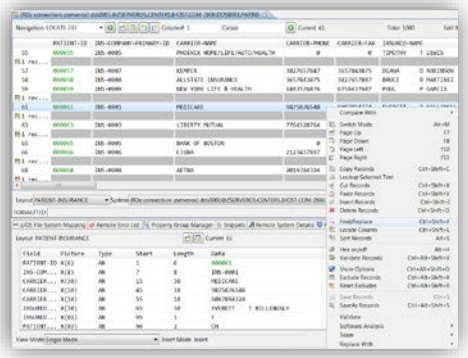
Abend analysis



Performance analysis



File Management



Analyze (program understanding)

Outline view

– Show high level abstraction of source

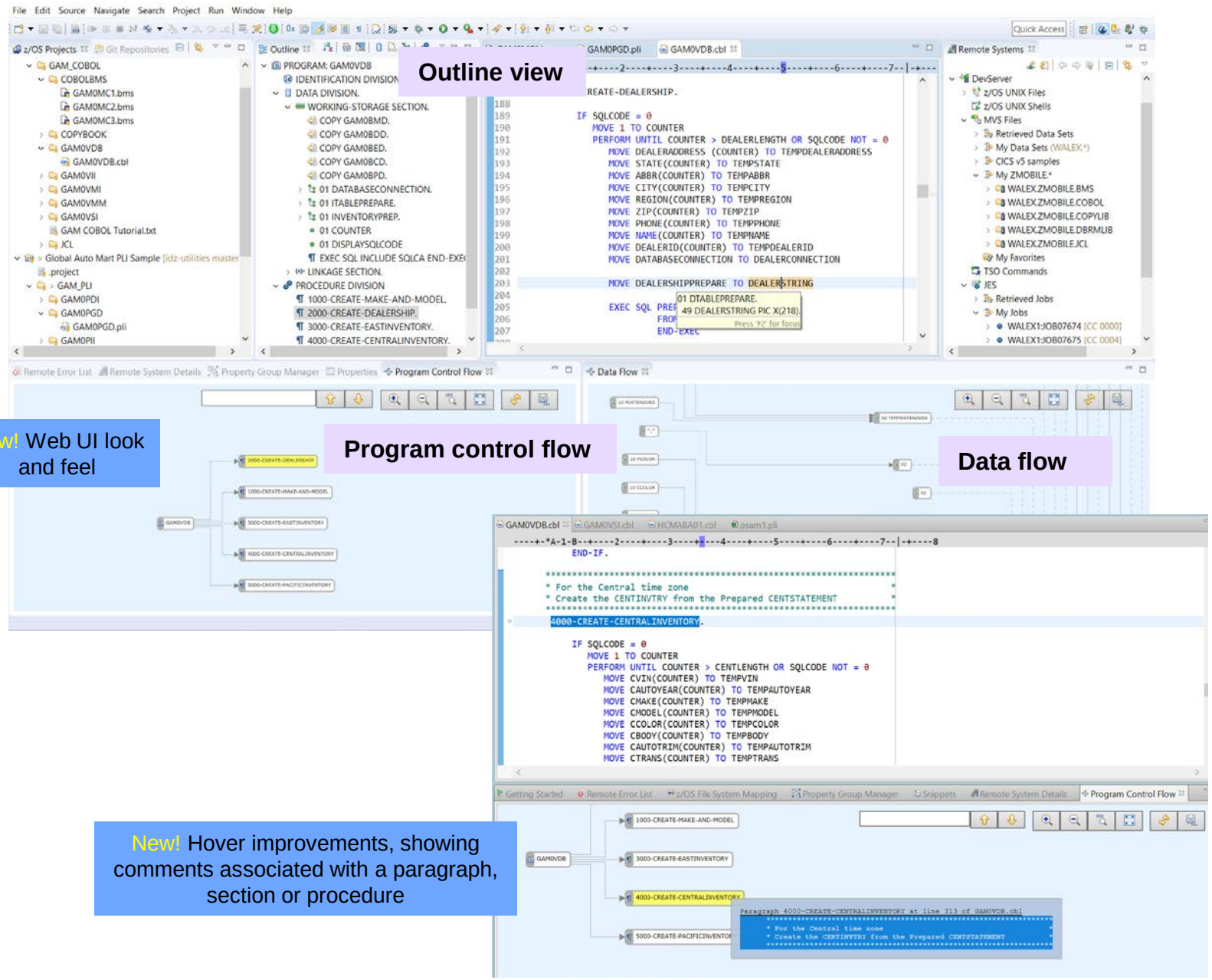
Program control flow

– 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

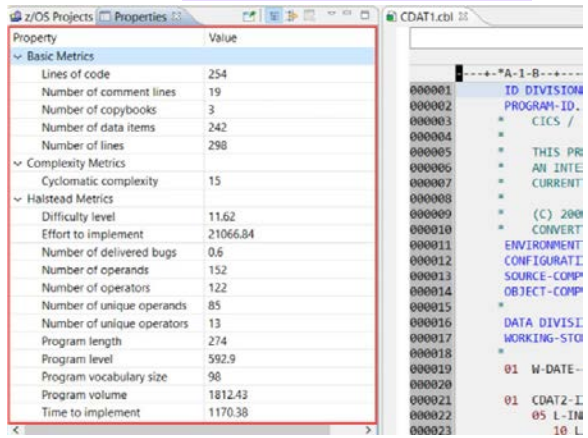
- Electronic desk checking
- Can be run in batch as part of a pipeline
- Customizable

Program level statistics

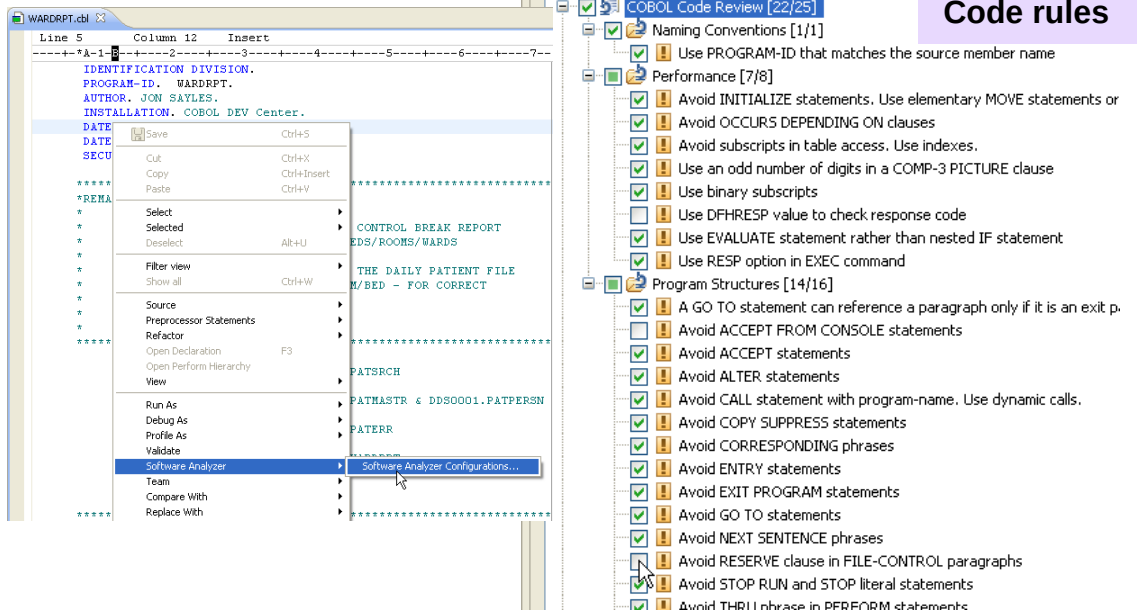
- Base metrics
- Complexity
- Halstead

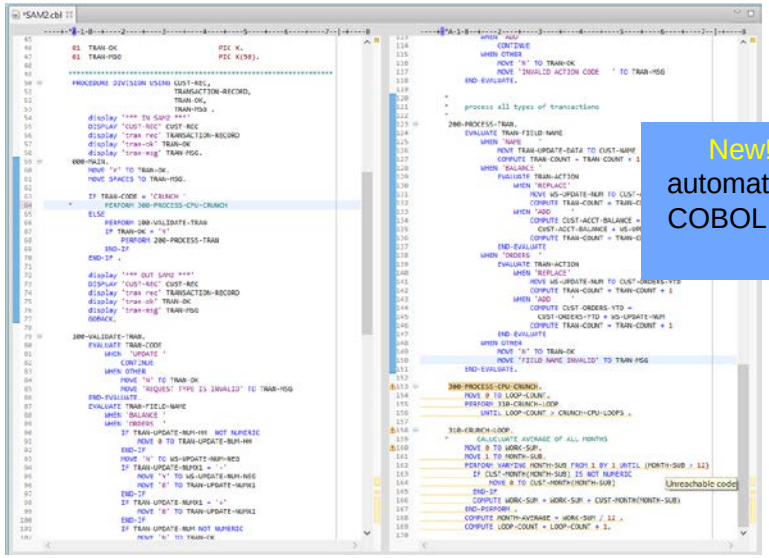
SonarLint integration

Program statistics

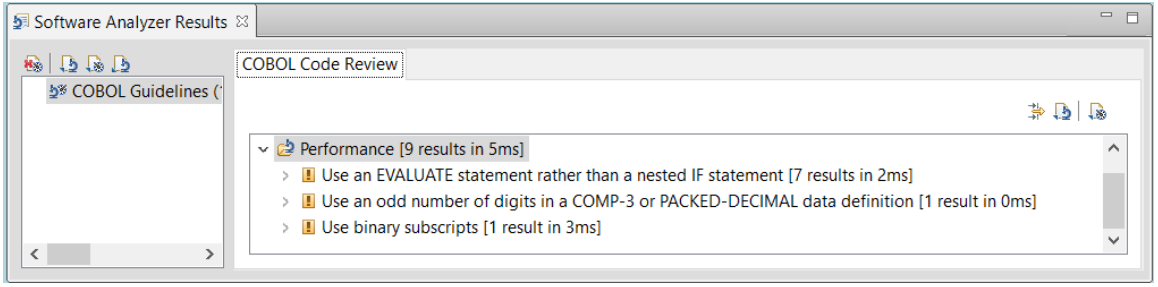


Code rules





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

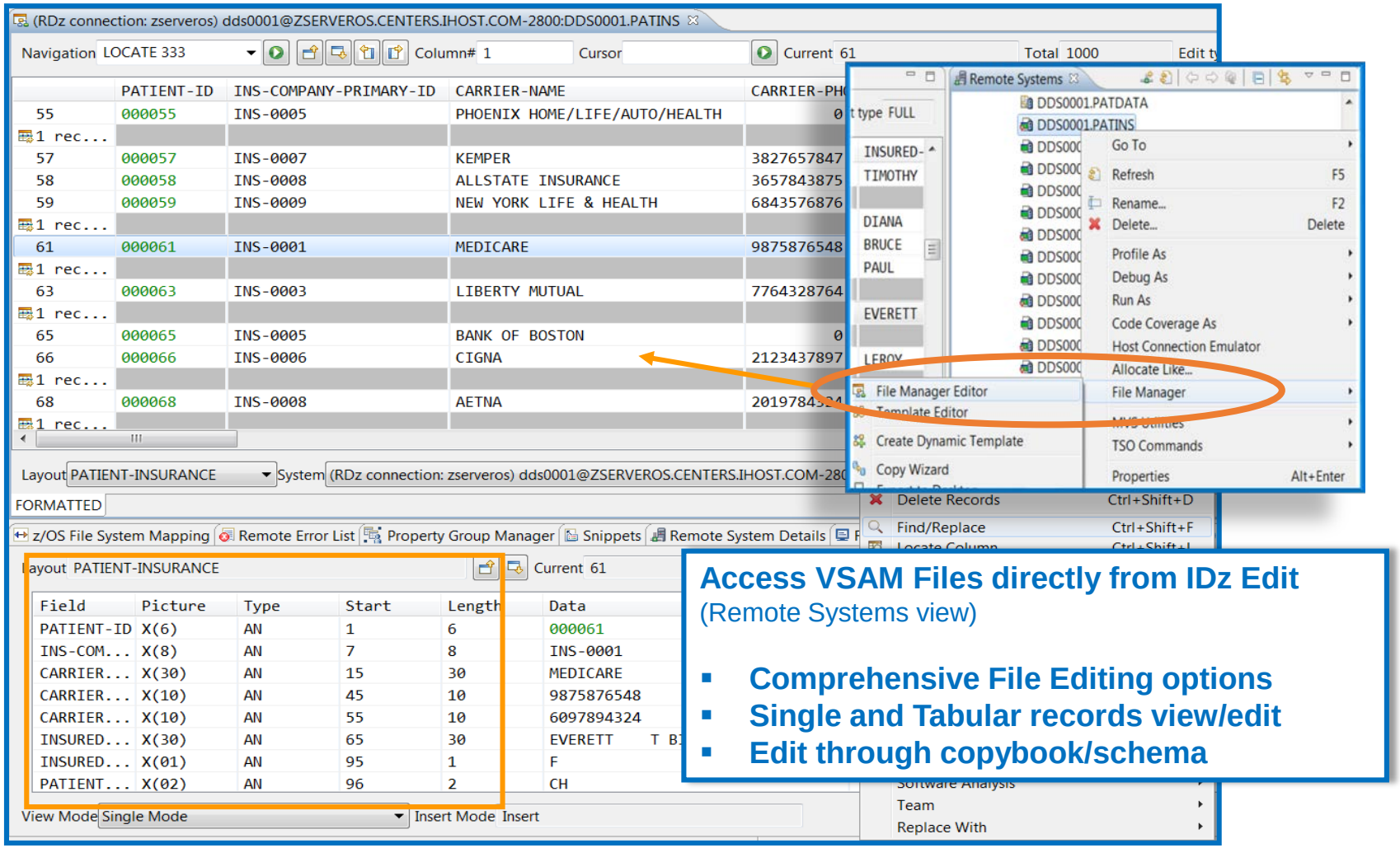


Could be used in a Pipeline (like Jenkins) via Batch execution to enforce standards

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

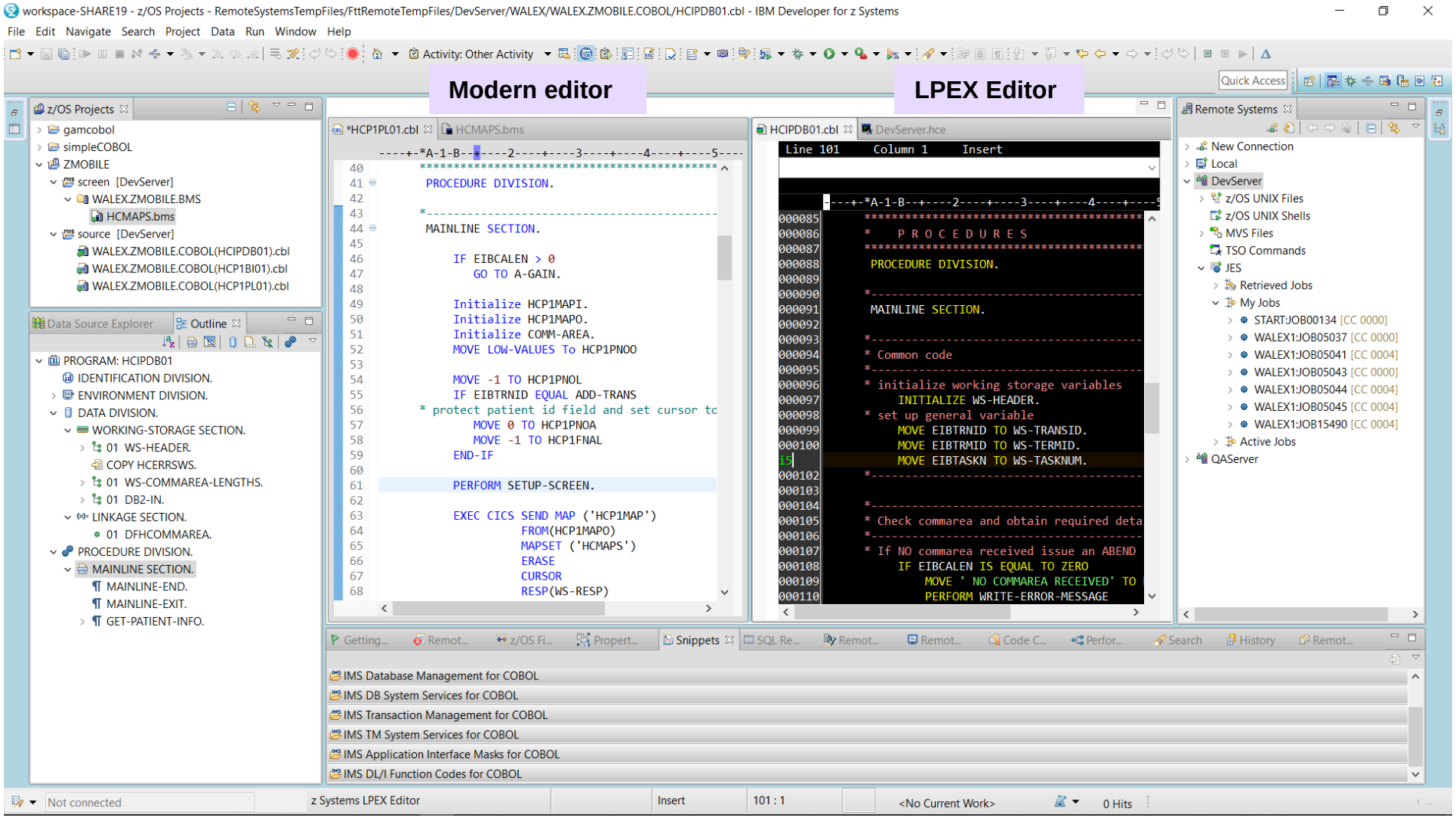


Access VSAM Files directly from IDz Edit
(Remote Systems view)

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

Edit

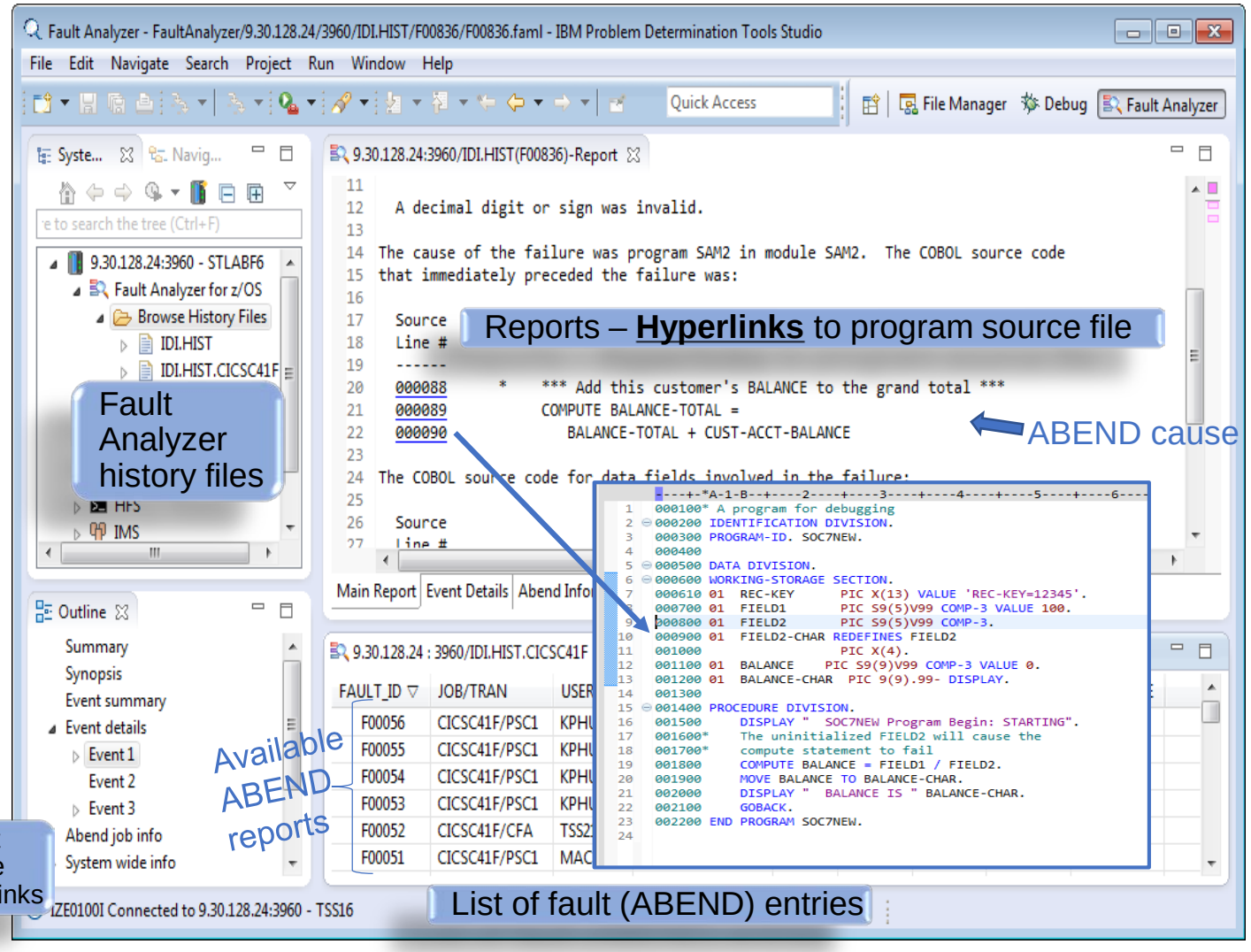
Traditional Eclipse style editors, plus ISPF-like LPEX editor with a command line and prefix area



ADFz: “Fault Analyzer (FA)” for Program Failures

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

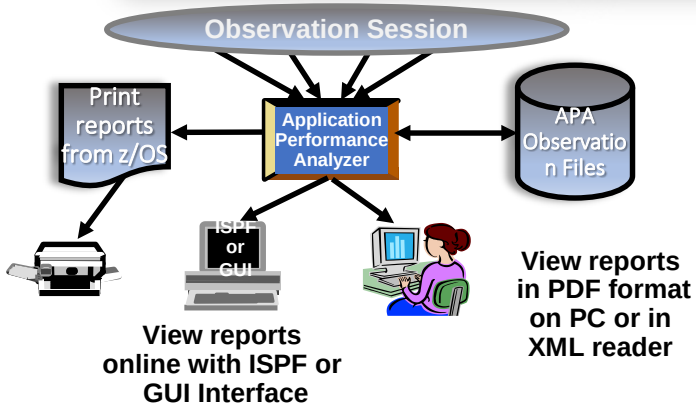
- Automatic program abend capture and reporting
- Program source-level reporting
- Provides a detailed report about program failures to help resolve them quickly
- Enables you to track and manage application failures and fault reports
- Integration with ADFz and 3270-based Interface



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

Provides rapid pin-pointing of enterprise application bottlenecks

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



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

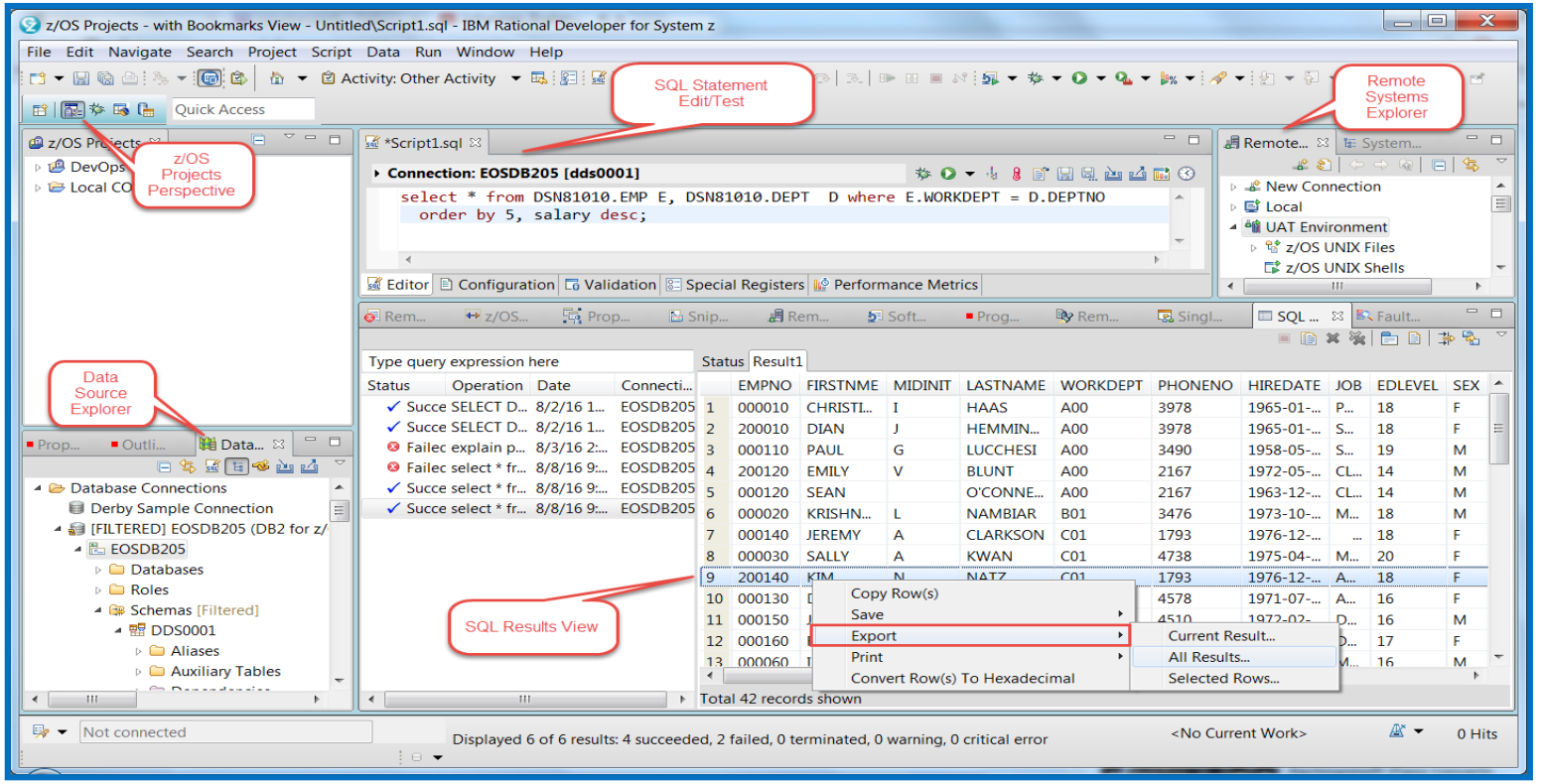
Integrated with ADFz and 3270-based Interface

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

Data Studio : DB2 Development Tooling

A flexible graphical interface to DB2 and SQL

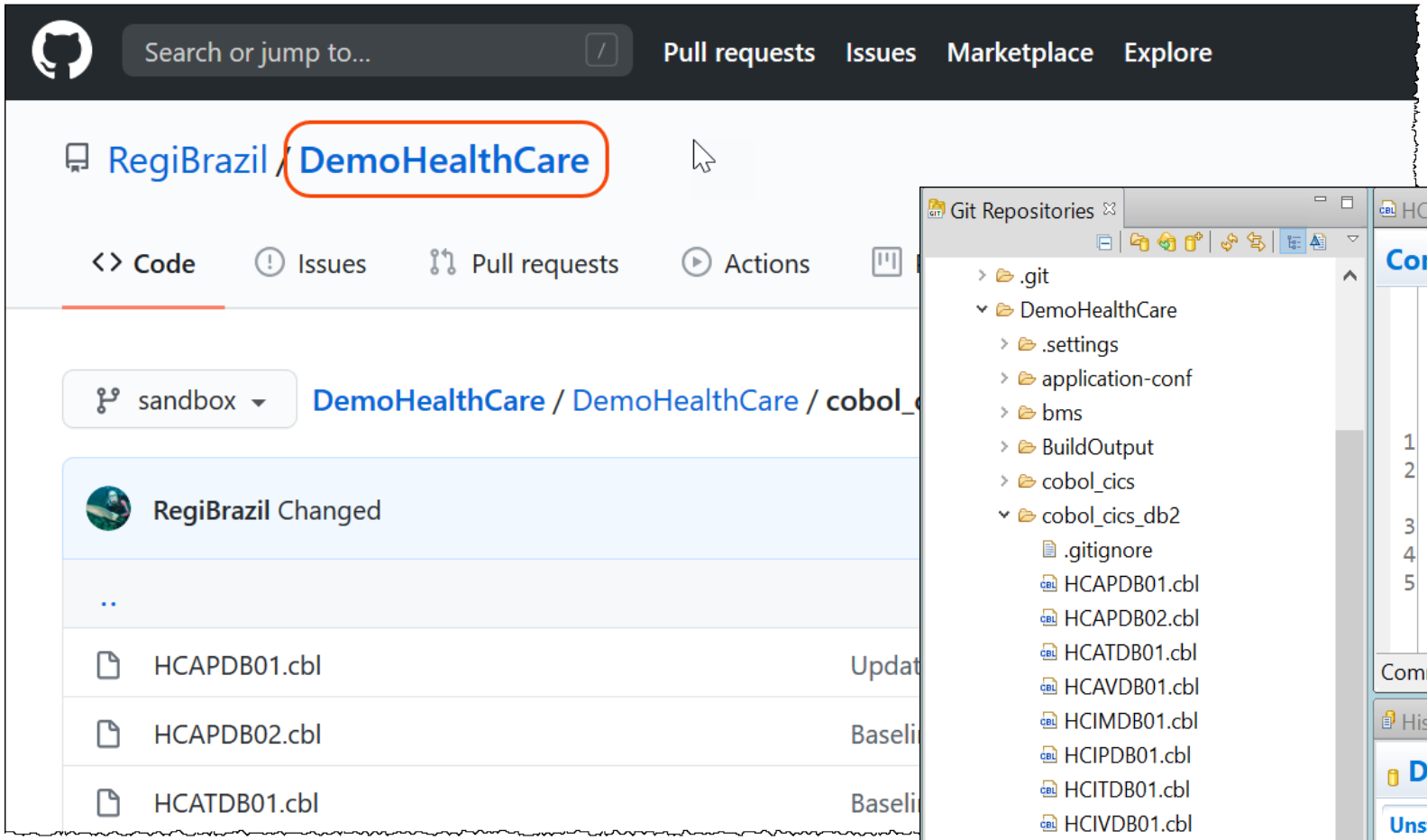
- DB2 Table/View/Index Analysis
- DB2 Data Model and for Test Data Management and manipulation:
  - CRUD
  - Simple Table row/column sub-setting (sampling)
  - Test/Run/Tune SQL directly from COBOL or PL/I programs
  - Run existing SPUFI files (DDL, DML)
- SQL code and Test
  - Code embedded SQL directly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

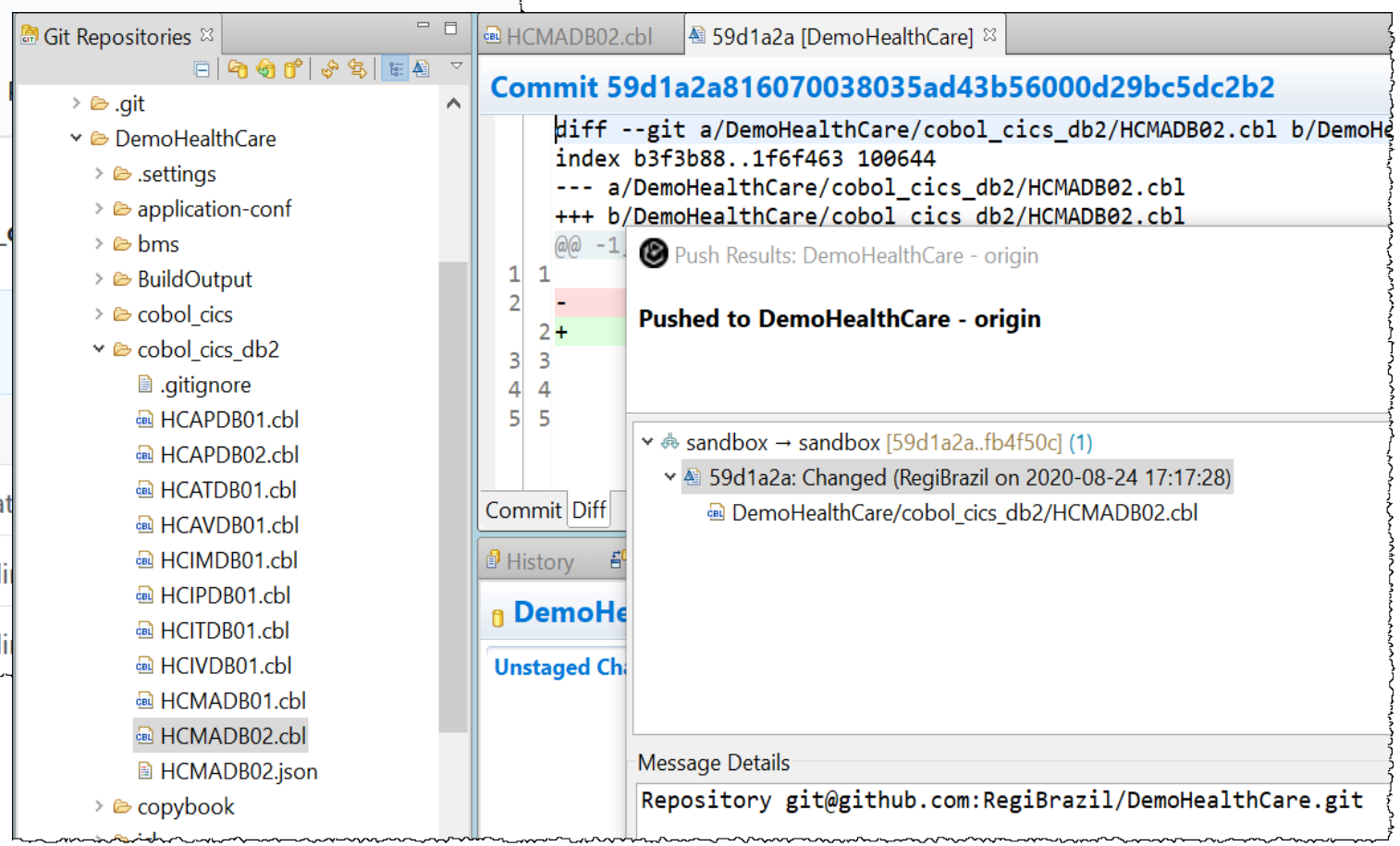


Source Code Management

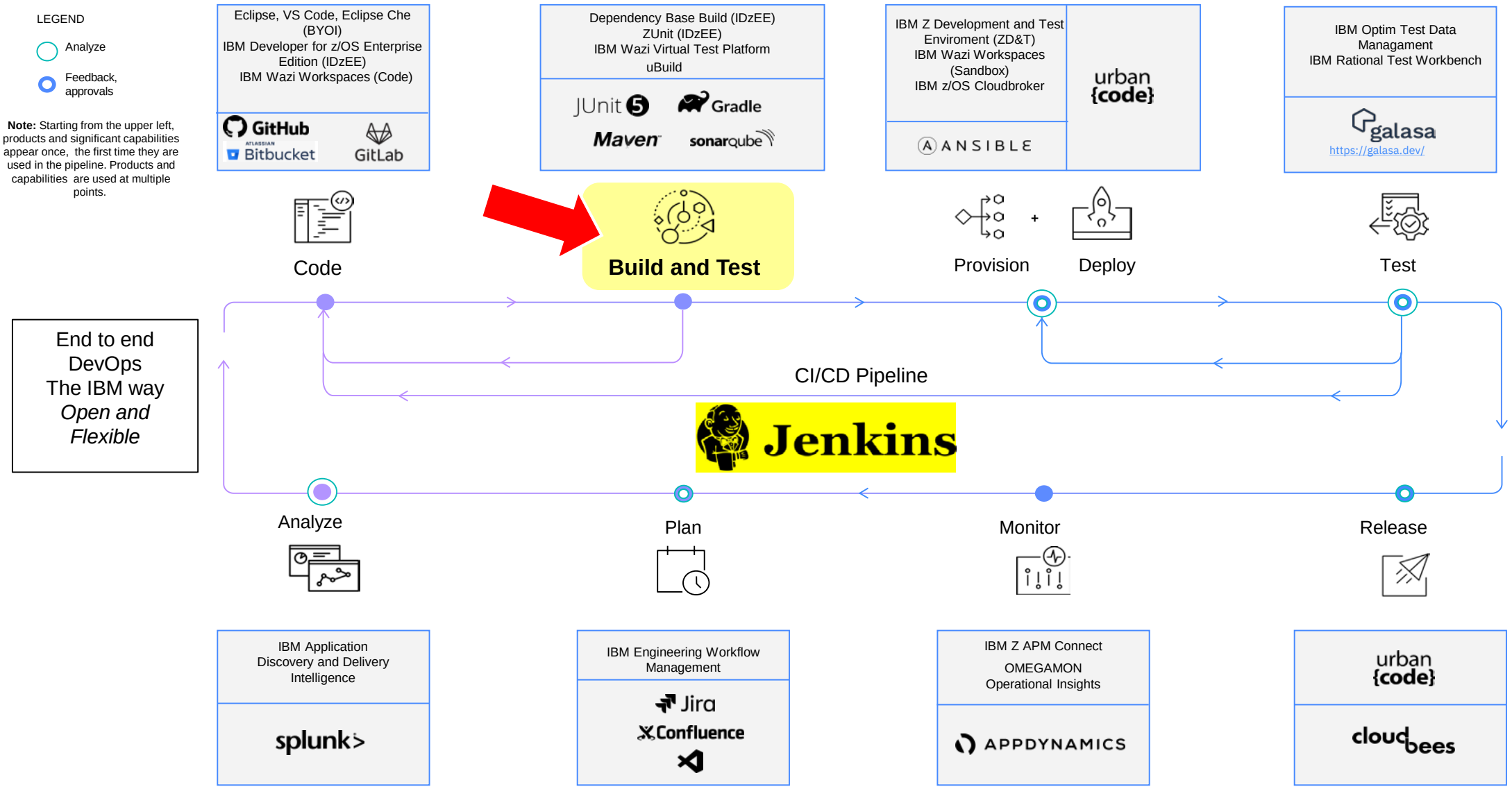
EGit included in IDz

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



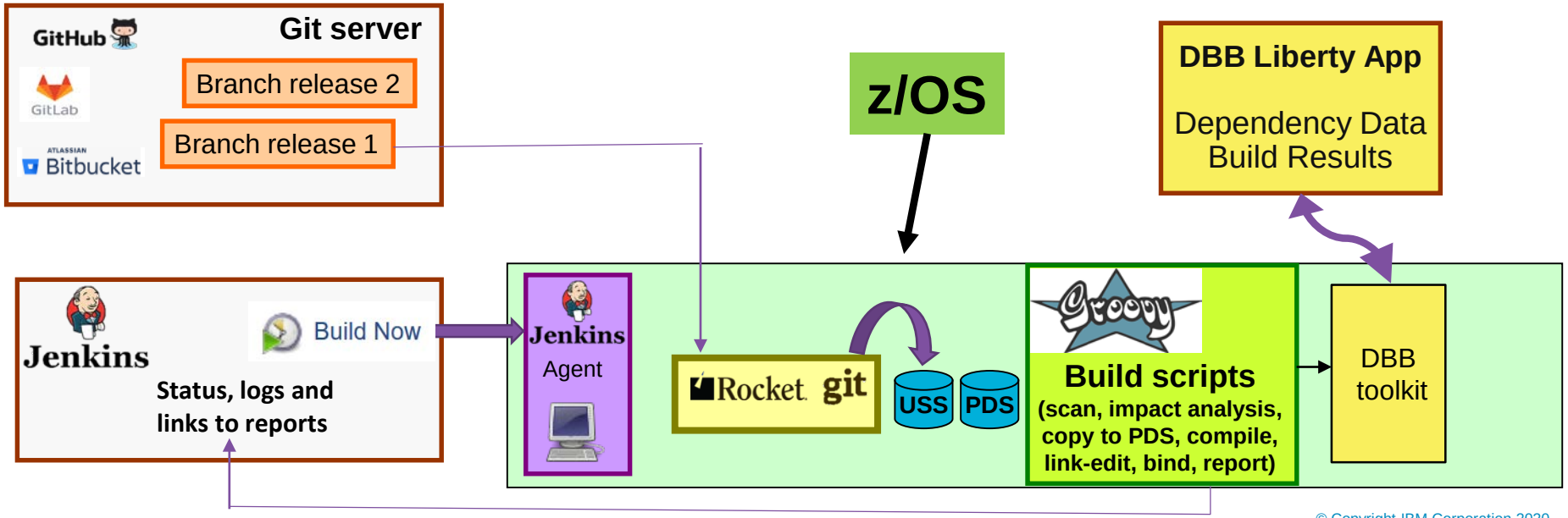


Z DevOps Pipeline



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)

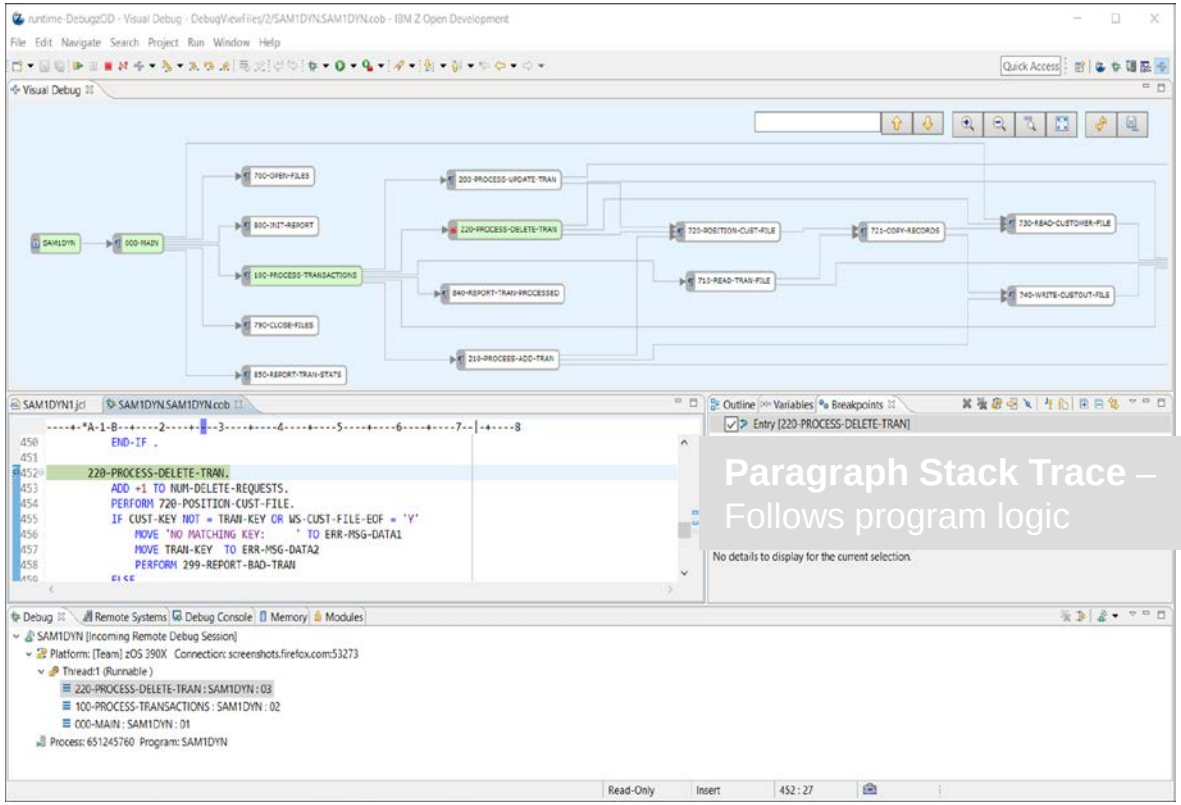
New! Web UI look and feel (14.1.5)

Program testing and analysis

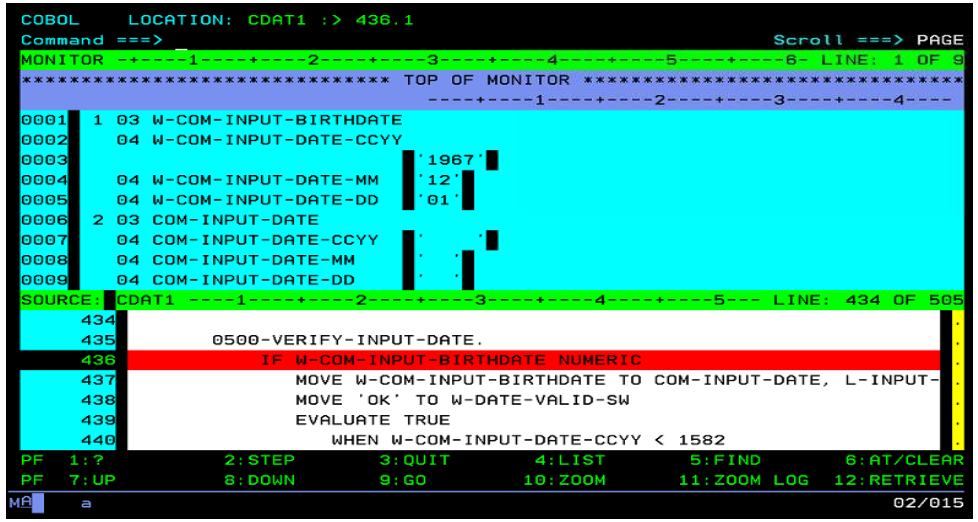
Helps you examine, monitor, and control the execution of application programs on z/OS

With:

- Visual debug
- Code coverage facilities
- Eclipse GUI
- 3270 UI (Debug for z/OS and IBM Developer for z/OS Enterprise Edition only)

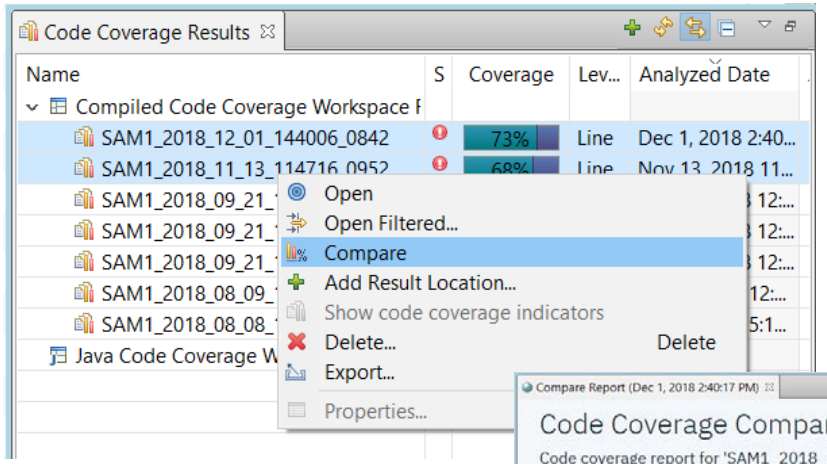


Paragraph Stack Trace – Follows program logic



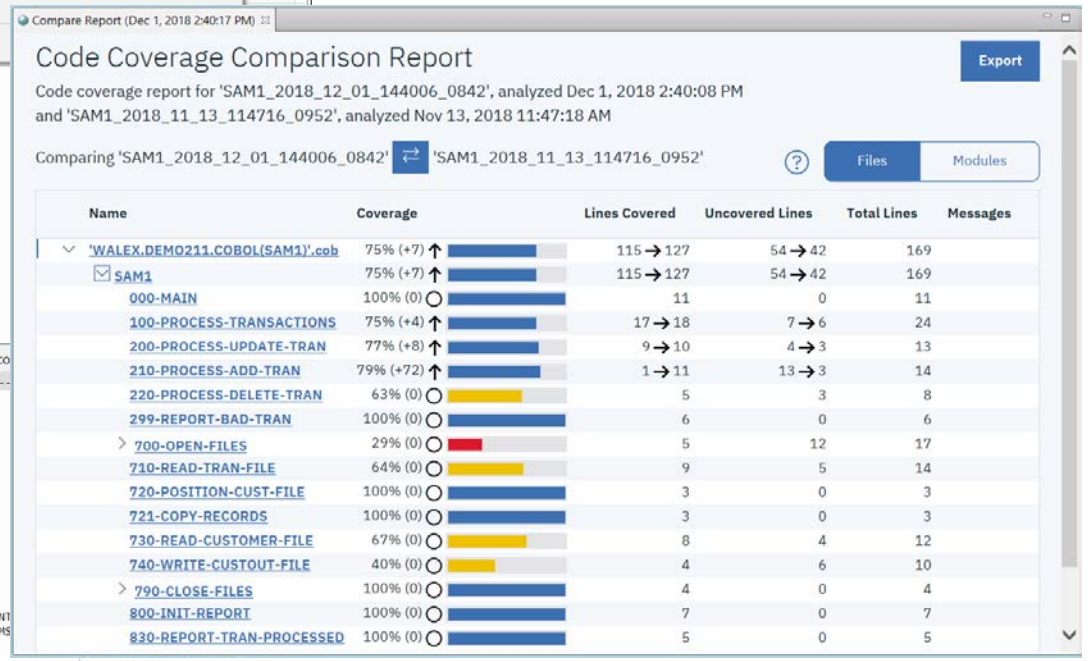
Test (code coverage)

- 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

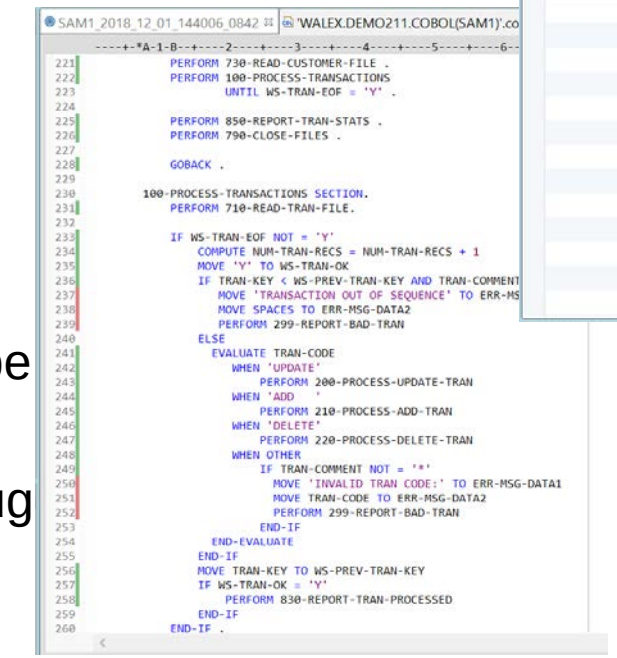


Compare results

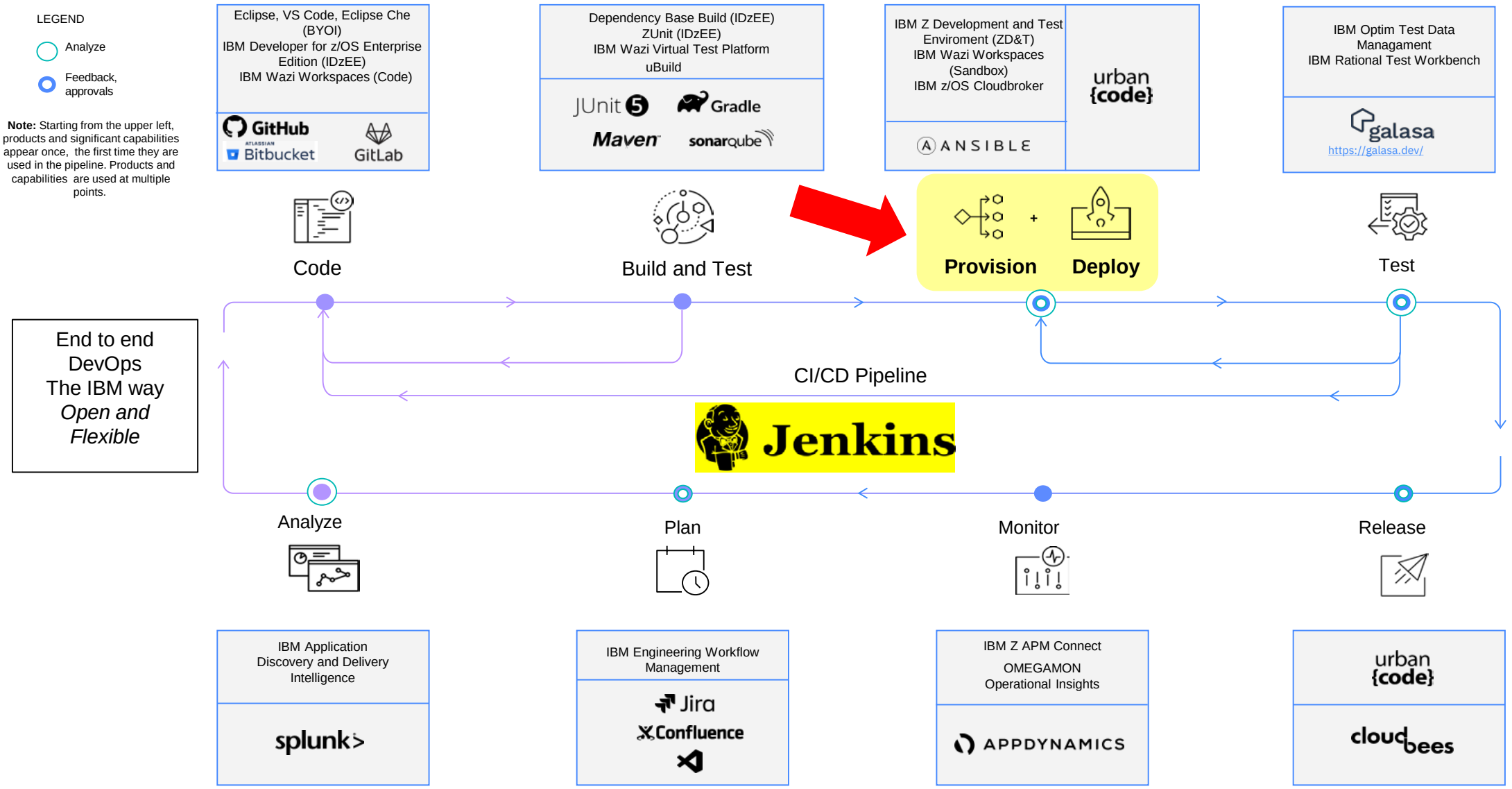
New! Web UI look and feel with easier filtering, grouping and comparisons, plus PDF output (14.1.5)



Coverage



Z DevOps Pipeline

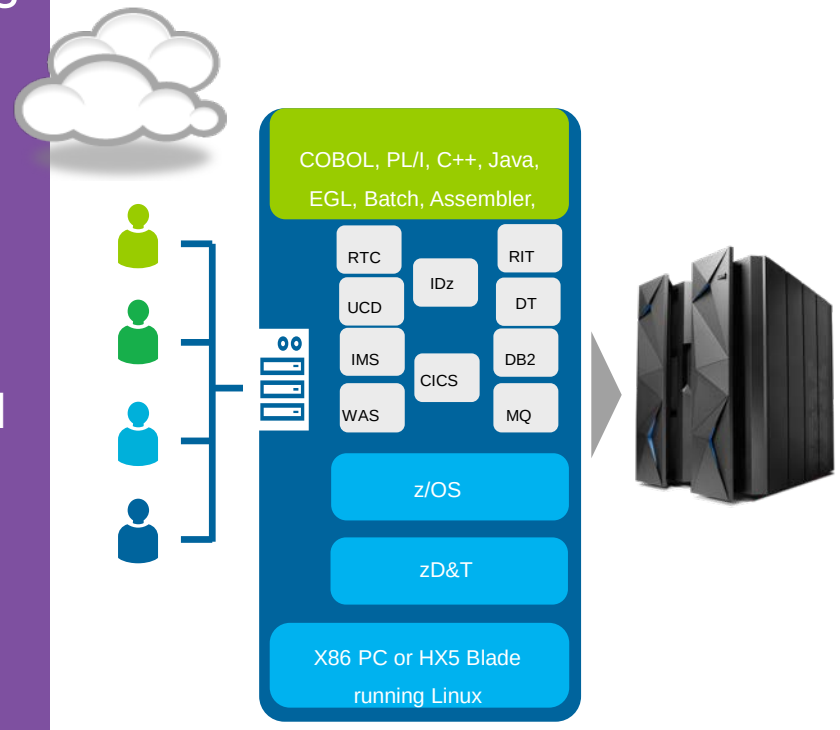


Provisioning

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

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

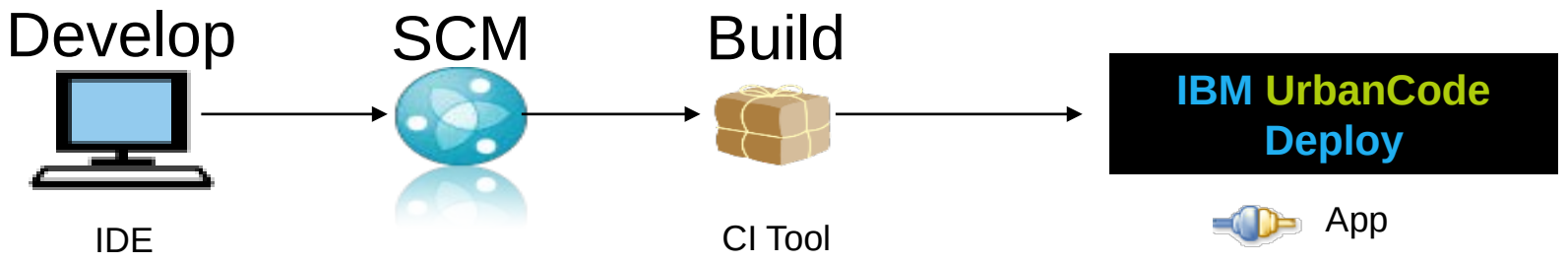
- **zD&T tools to accelerate provisioning and image management**
- Cloud friendly, software-based licensing for enterprise customers (managed service on Softlayer)
- Latest z/OS 2.3 software and middleware
- ***Developer autonomy just like distributed, web and mobile developers!***



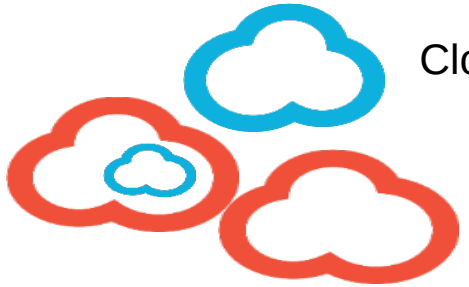
UrbanCode for Deployment automation (UCD)

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, Db2, MQ, across multiple environments

Integrates with Jenkins

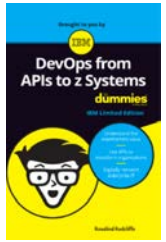
Manages artifact versions via an in-built repository called codestation

IBM

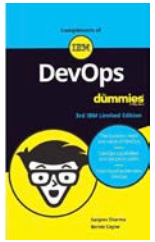
Additional Resources

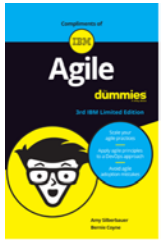
— Mainframe CI/CD with an open toolchain:
  - https://www.linkedin.com/pulse/mainframe-cicd-open-toolchain-its-real-spectacular-minaz-merali/

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



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





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

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

Questions / Comments

Agenda - Day 1

10:00 Introduction to DevOps on IBM Z - focus on open source tools (Ron)

10:40 LAB 1 - Working with z/OS using COBOL and DB2 (Tim)

or

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

12:50 Checkpoint - Comments and Closing Day 1 (Tim -Ron/Wilbert/Regi)

Agenda - Day 2

10:00 zUnit and Wazi VTP Overview (Wilbert)

10:15 Demo: Scenario using Z DevOps solutions including **zUnit**, **Git** and **Jenkins** (Regi)

11:00 – 12:50 Choose one Optional lab: (Tim)

- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6B : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → 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)

12:50 Checkpoint - Comments and Closing Day 2 (Tim -Ron/Wilbert/Regi)

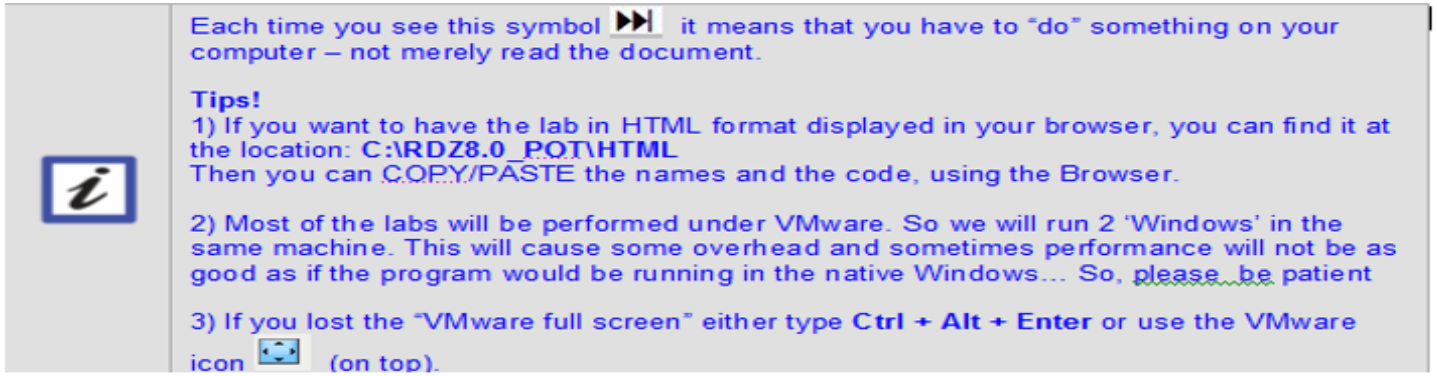
Lecture Labs

DevOps Mainframe Tooling - Labs

Labs

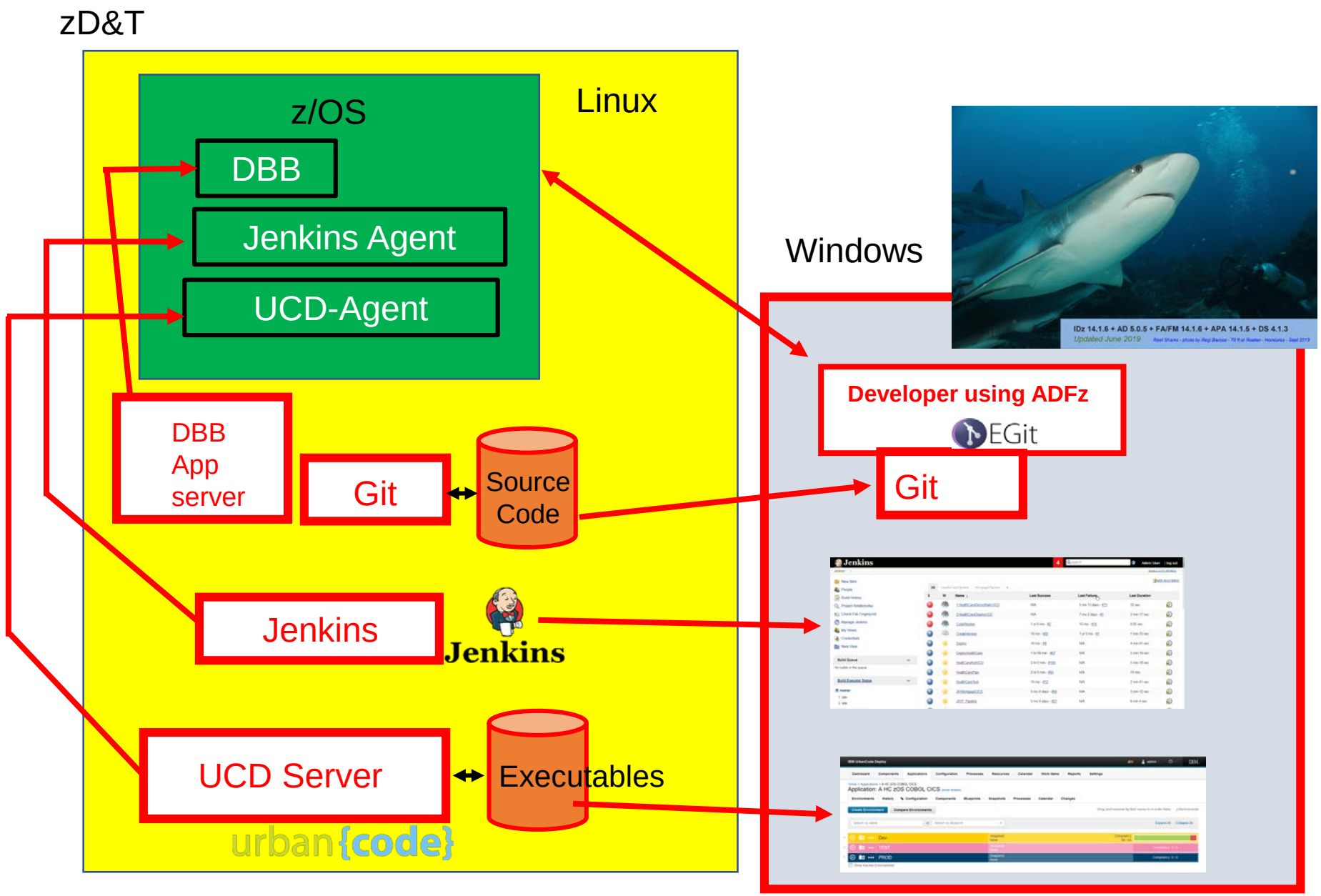
Labs using VMware …

1. User id → empot01 and password → empot01
2. Follow exactly what is described in the lab



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

zD&T on Cloud environment



Lab 1: Working with mainframe using COBOL and DB2

Duration: 90 Minutes

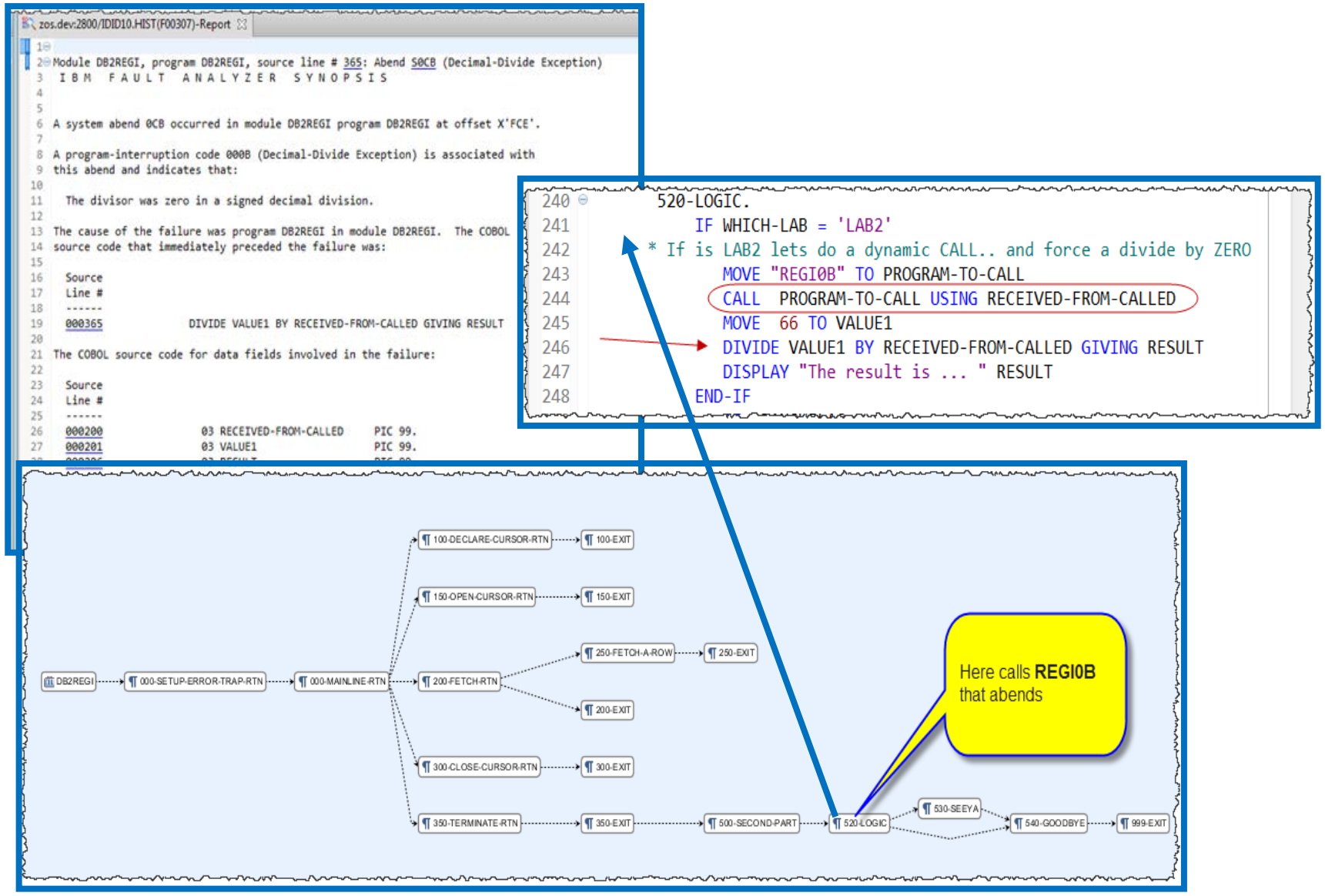
Objective

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

Key Activities

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

Lab 1: Working with mainframe using COBOL and DB2



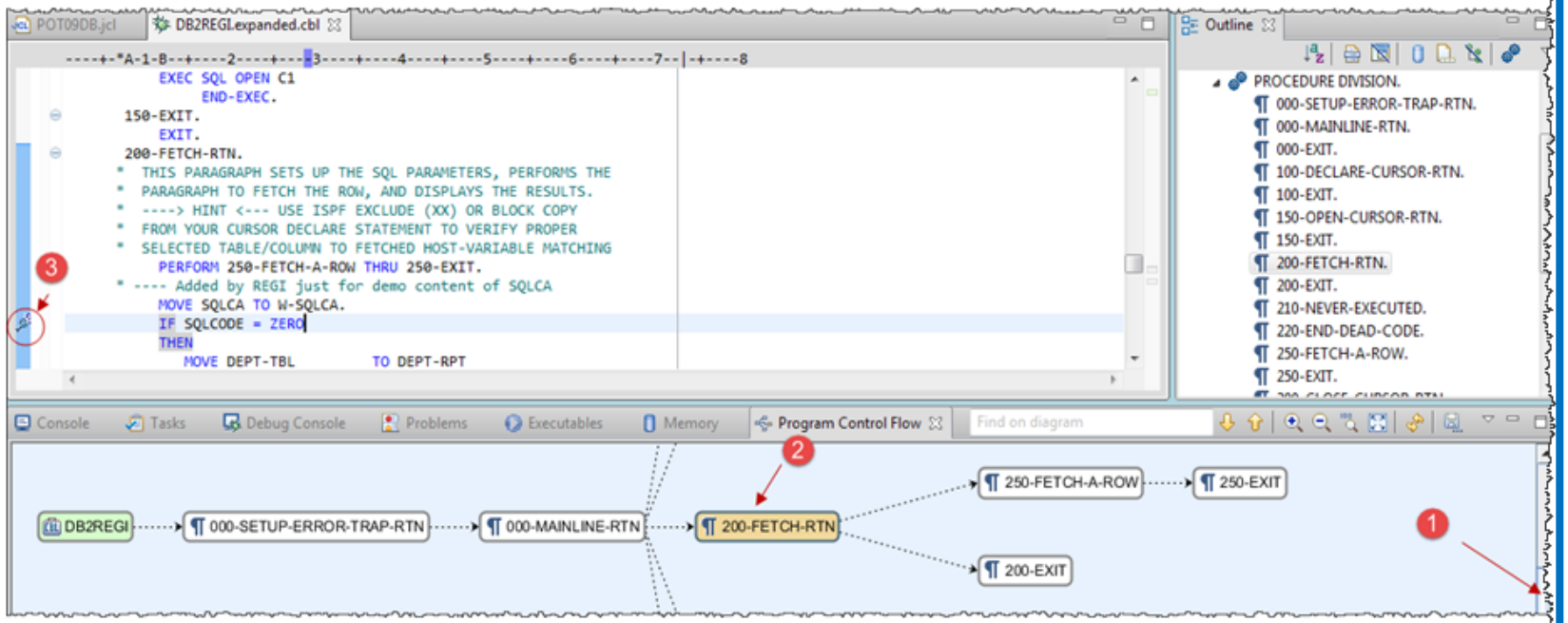
Lab 1: Don’t Miss… The Debug… Page 22

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.

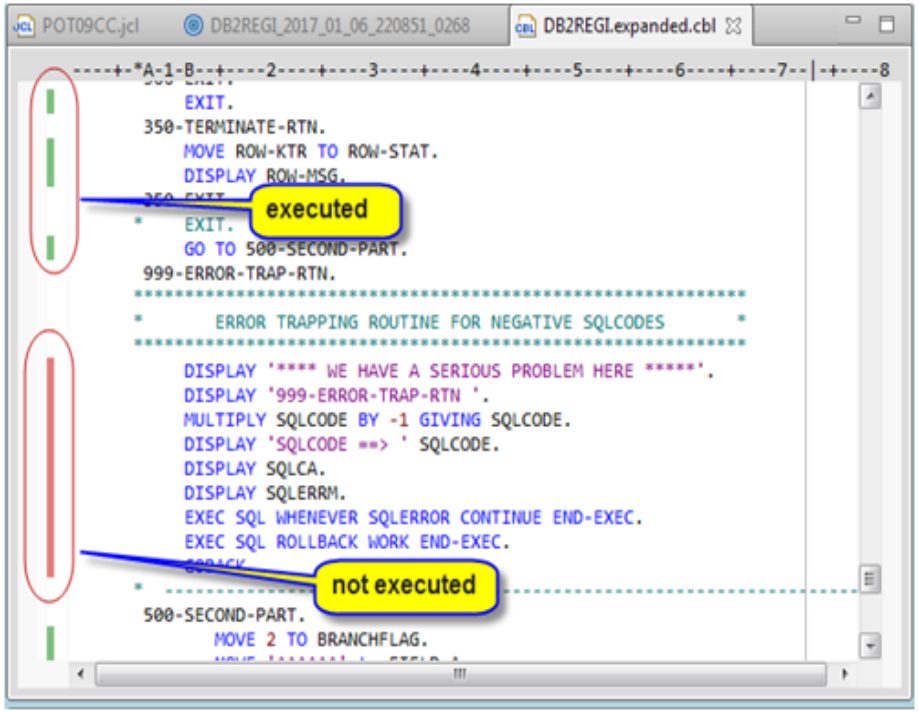
The icon is shown



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

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

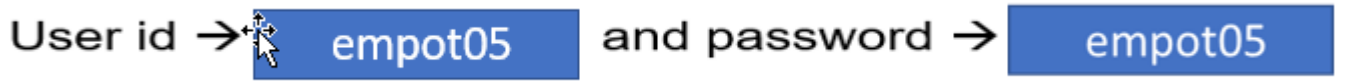
⏭ Scroll down and you will see the lines executed in **green** and the lines not executed in **Red**.



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

Overview of development tasks



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

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

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

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

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

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

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

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

Agenda - Day 1

10:00 Introduction to DevOps on IBM Z - focus on open source tools (Ron)

10:40 LAB 1 - Working with z/OS using COBOL and DB2 (Tim)

or

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

12:50 Checkpoint - Comments and Closing Day 1 (Tim -Ron/Wilbert/Regi)

Agenda - Day 2

10:00 zUnit and Wazi VTP Overview (Wilbert)

10:15 Demo: Scenario using Z DevOps solutions including **zUnit**, **Git** and **Jenkins** (Regi)

11:00 – 12:50 Choose one Optional lab: (Tim)

- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6B : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → 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)

12:50 Checkpoint - Comments and Closing Day 2 (Tim -Ron/Wilbert/Regi)

Lecture Labs

Agenda - Day 1

10:00 Introduction to DevOps on IBM Z - focus on open source tools (Ron)

10:40 LAB 1 - Working with z/OS using COBOL and DB2 (Tim)

or

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

12:50 Checkpoint - Comments and Closing Day 1 (Tim -Ron/Wilbert/Regi)

Agenda - Day 2

10:00 zUnit and Wazi VTP Overview (Wilbert)

10:15 Demo: Scenario using Z DevOps solutions including **zUnit**, **Git** and **Jenkins** (Regi)

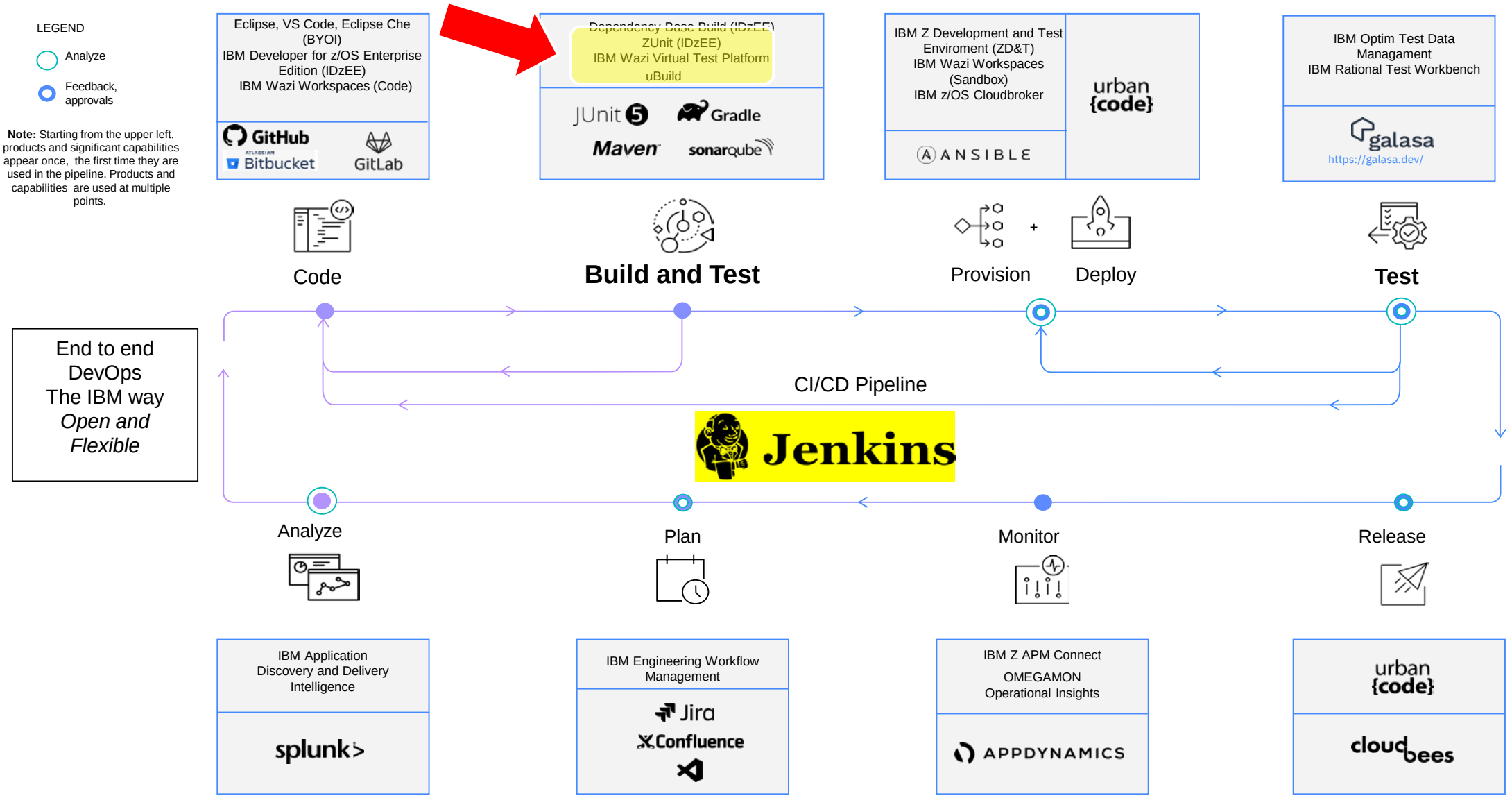
11:00 – 12:50 Choose one Optional lab: (Tim)

- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6B : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → 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)

12:50 Checkpoint - Comments and Closing Day 2 (Tim -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 HCMADB02 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

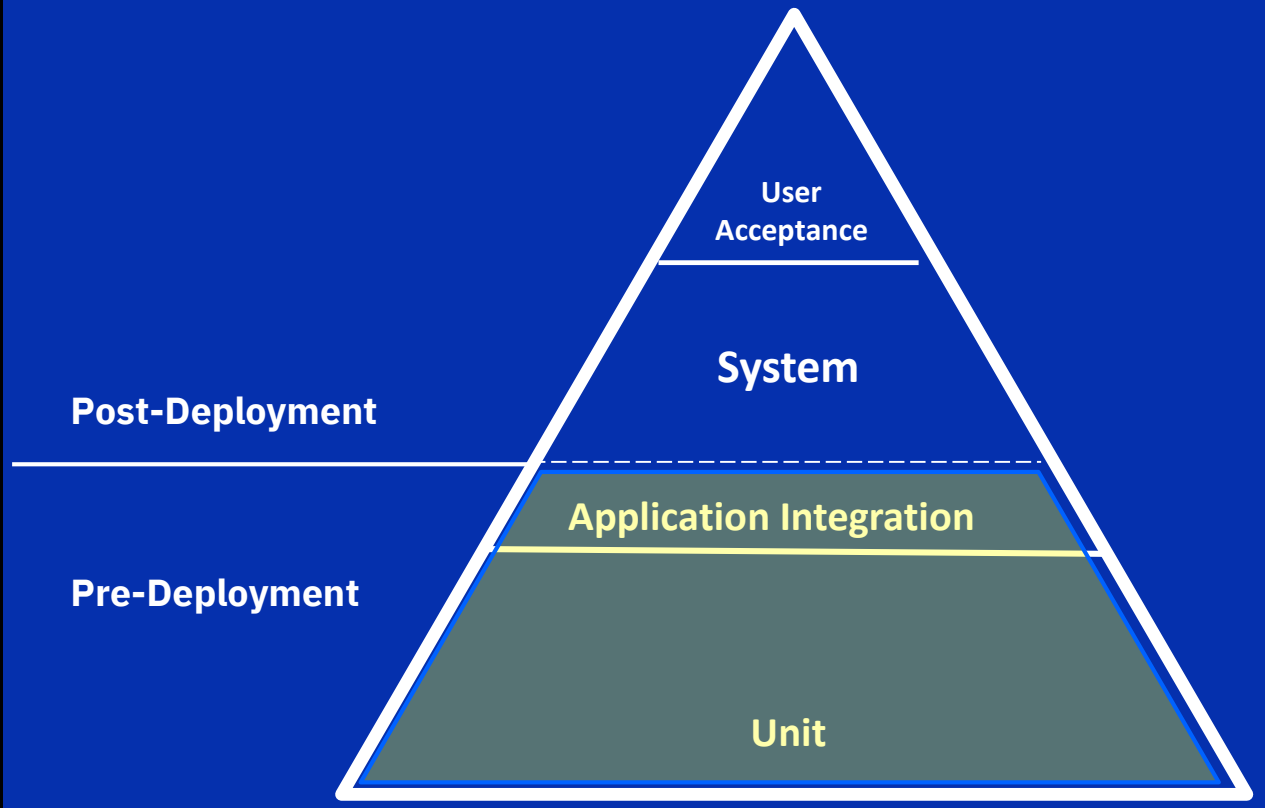
RTW (Rational Integration Tester/Rational functional tester/Test Virtualization Server)

Application Integration test

IBM Wazi Virtual Test Platform

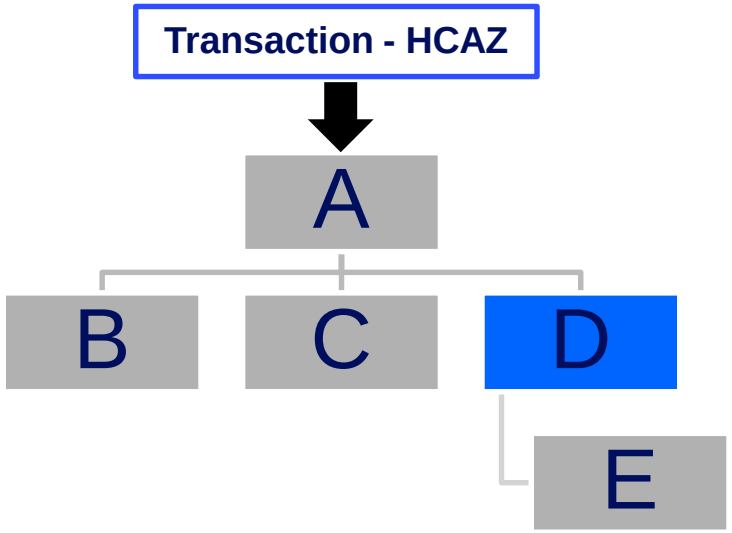
Unit test

ZUnit (IBM Developer for z/OS)



Unit Testing (zUnit)

Scope - Program



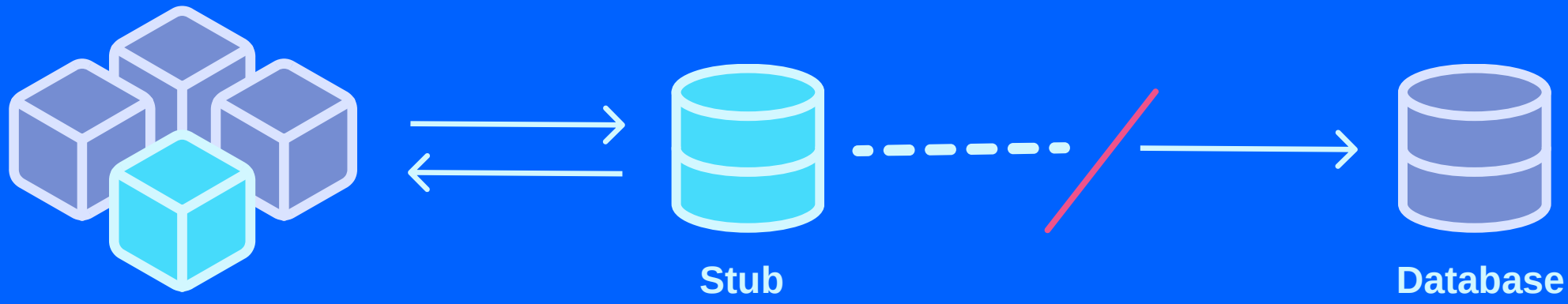
A unit is the smallest testable part of software – meaning testing an individual unit of source code, like a single source file which contains a list of methods, procedures.

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

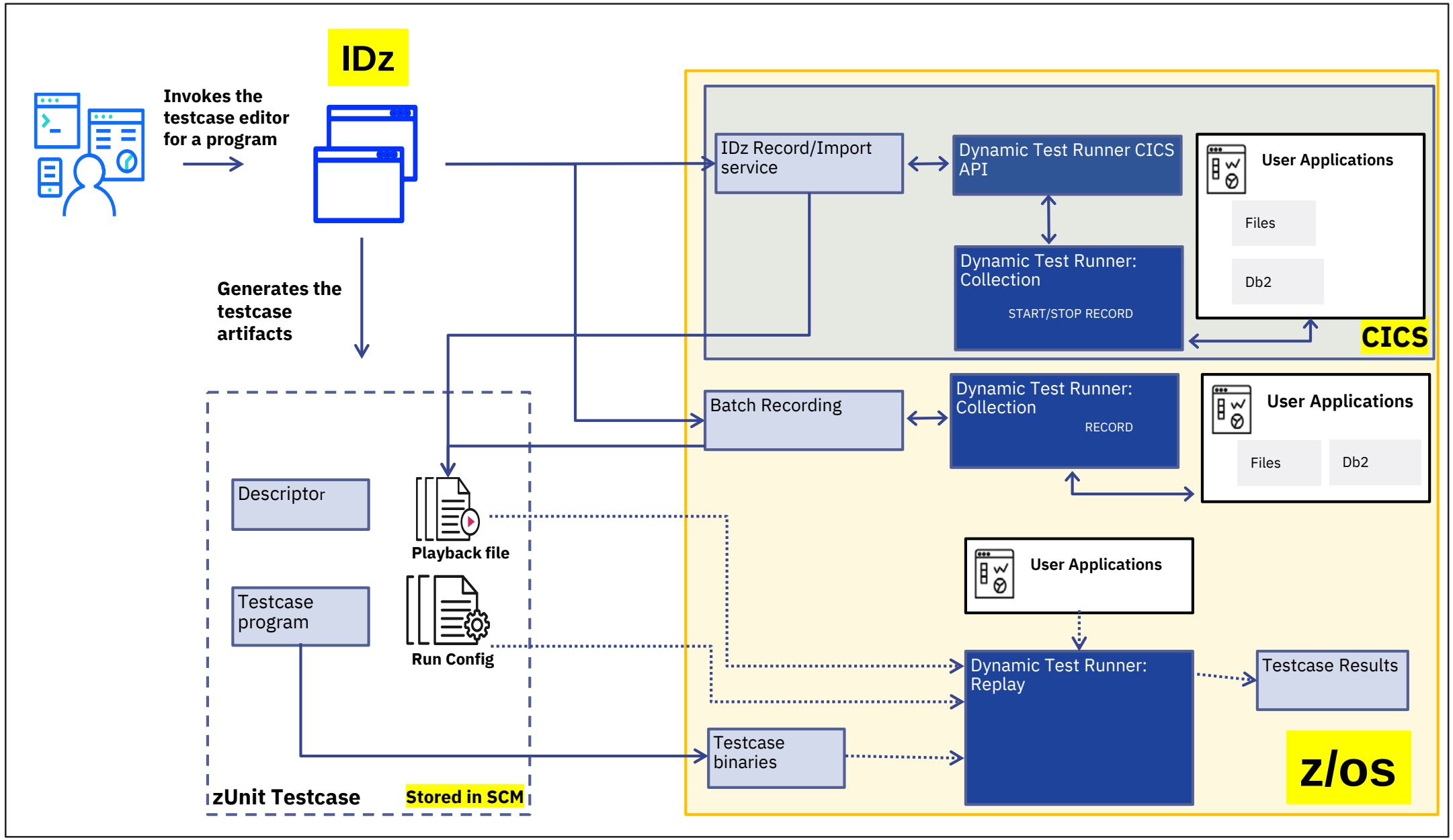
Shift Left – unit testing with ZUnit

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

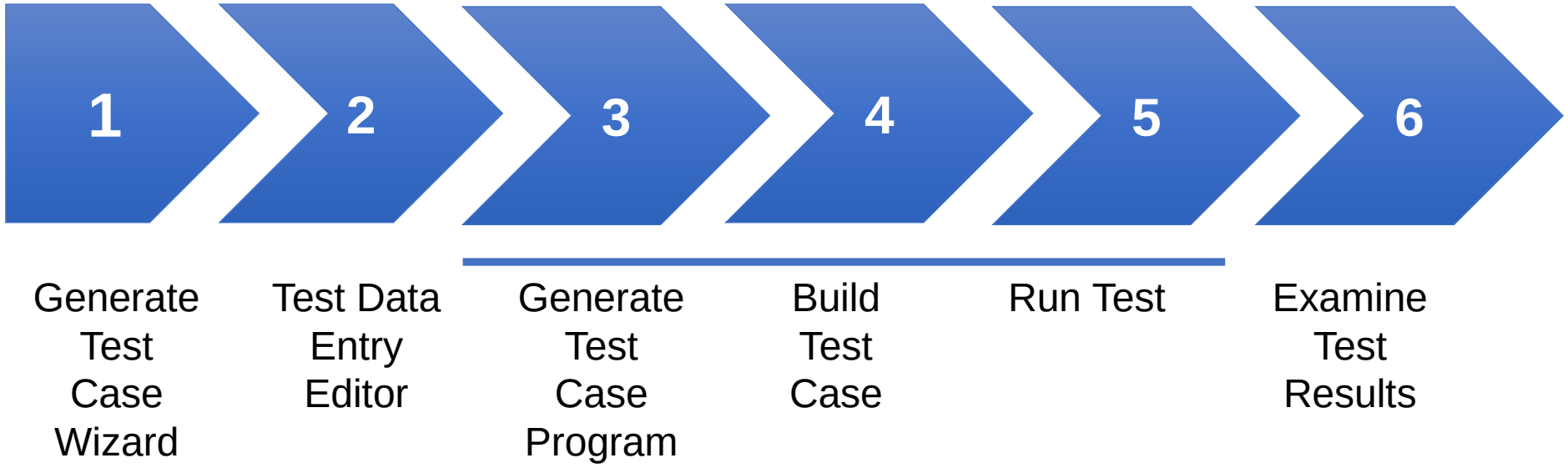
This is done at the **program level**



ZUnit High Level Architecture

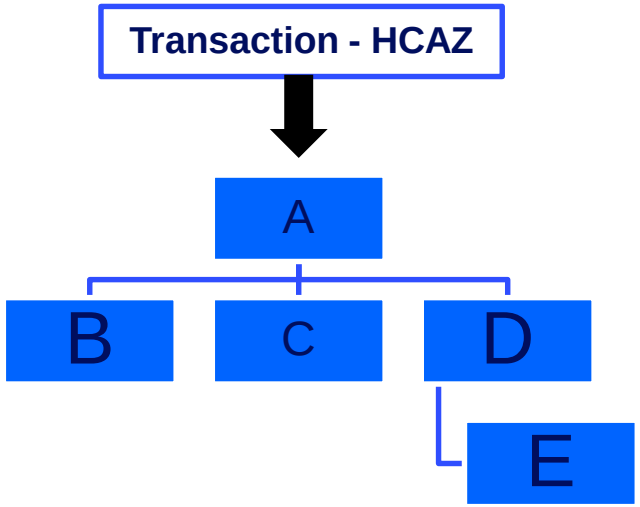


zUnit Basic Workflow



Application Integration test (**Wazi VTP**)

Scope - Transaction



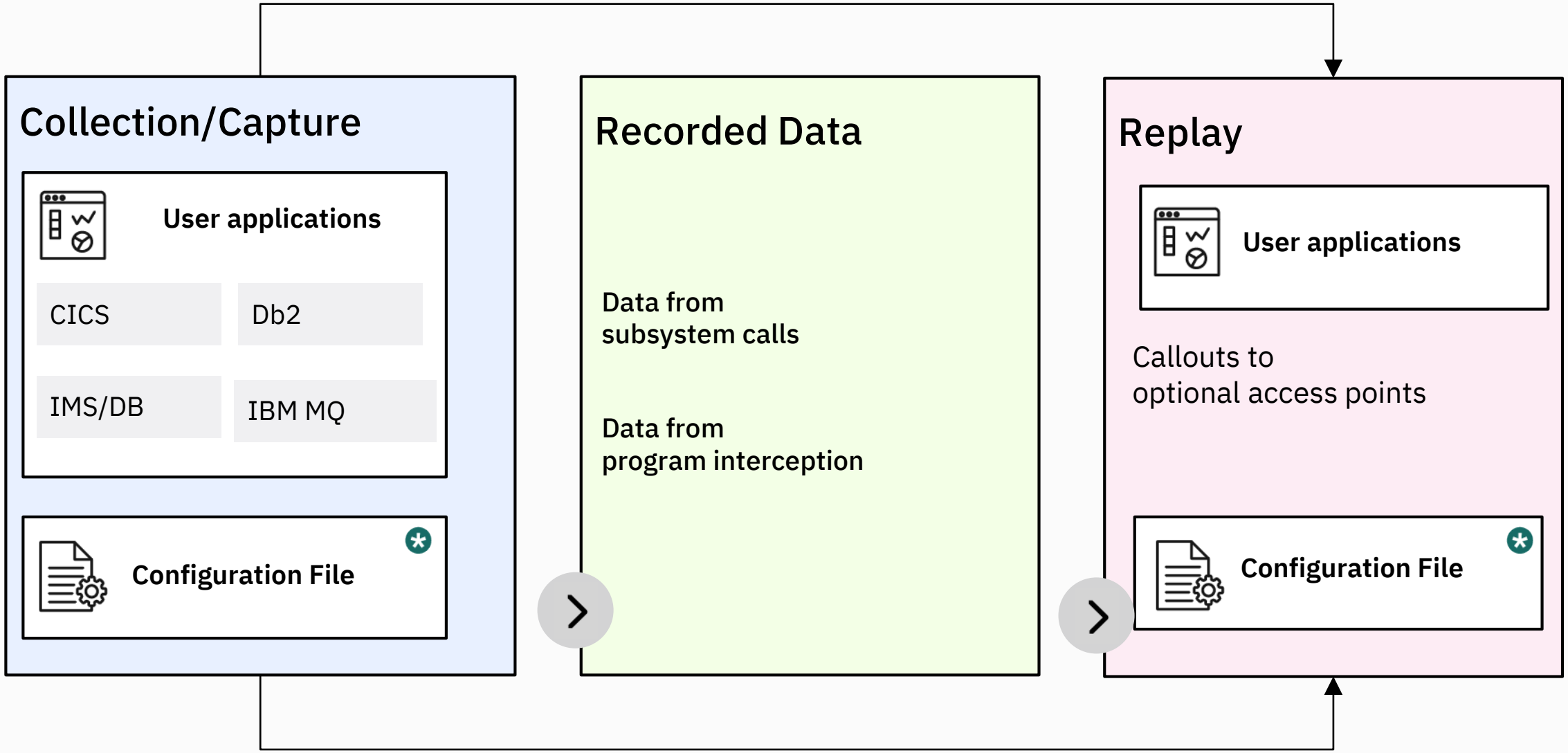
Application integration test is the next level of testing after Unit Testing. Its purpose is to confirm that the changes in a unit (programs) works with the interfaces to other units (programs), and ensure no unexpected impacts to other units

Transaction and batch jobs

Testing with Wazi Virtual Test Platform

Part 1 – Capture or Collection

Part 2 – Replay



Optional

The Testing Pipeline

Test Data

IBM Infosphere Optim products

Optim Test data Management
Optim Test data fabrication
Other Optim products

Prod/Prod-Clone

Pre deployment | **Post deployment**

Eclipse, VS Code, Eclipse Che (BYOI)
IBM Developer for z/OS Enterprise Edition (IDzEE)
IBM Wazi Workspaces (Code)

GitHub GitLab

Code

IDz(Zunit)
Wazi VTP

Unit Testing
Application Integration Test
(Isolated integration test)

Build and Test

Lpar with deployed App

Provision **Deploy**

Rational Test WorkBench (RTW)
(RPT, RFT and RIT)
Rational Test Virtualization Server

System test

Test

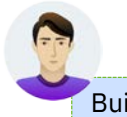
End user test

User Acceptance Test

Test

Code Change
Personal build
Unit test with ZUnit

Pipeline Build
Unit testing with ZUnit
Appl Int Test with VTP

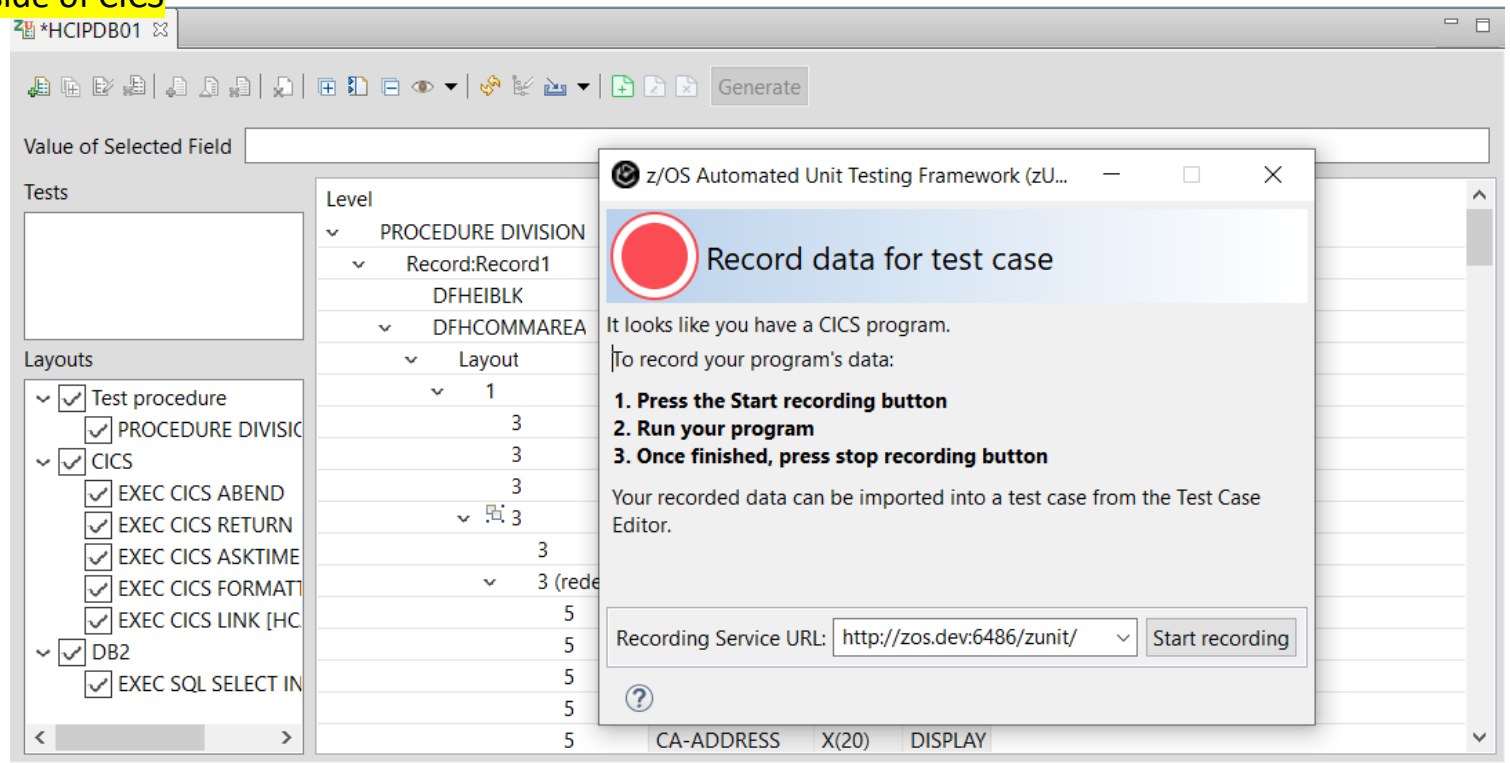


Build and Execute Functional Tests
Build and Execute Performance Tests
Build and Execute Integration Tests
Review and manage test reports

End user test

IBM zUnit for CICS/DB2 programs

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



Agenda - Day 1

10:00 Introduction to DevOps on IBM Z - focus on open source tools (Ron)

10:40 LAB 1 - Working with z/OS using COBOL and DB2 (Tim)

or

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

12:50 Checkpoint - Comments and Closing Day 1 (Tim -Ron/Wilbert/Regi)

Agenda - Day 2

10:00 zUnit and Wazi VTP Overview (Wilbert)

10:15 Demo: Scenario using Z DevOps solutions including **zUnit**, **Git** and **Jenkins** (Regi)

11:00 – 12:50 Choose one Optional lab: (Tim)

- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6B : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → 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)

12:50 Checkpoint - Comments and Closing Day 2 (Tim -Ron/Wilbert/Regi)

Lecture Labs

LEGEND

Analyze

Feedback, approvals

Note: Starting from the upper left, products and significant capabilities appear once, the first time they are used in the pipeline. Products and capabilities are used at multiple points.

End to end DevOps The IBM way *Open and Flexible*

Eclipse, VS Code, Eclipse Che (BYOI)
IBM Developer for z/OS Enterprise Edition (IDzEE)
IBM Wazi Workspaces (Code)
GitHub GitLab

Code

Dependency Based Build (IDzEE)
ZUnit (IDzEE)
IBM Wazi Virtual Test Platform
uBuild
JUnit 5 Gradle Maven sonarqube

Build and Test

IBM Z Development and Test Enviroment (ZD&T)
IBM Wazi Workspaces (Sandbox)
IBM z/OS Cloudbroker
ANSIBLE
urban {code}

Provision + Deploy

IBM Optim Test Data Managament
IBM Rational Test Workbench
galasa
https://galasa.dev/

Test

CI/CD Pipeline

GitHub Jenkins JFrog ARTIFACTORY Nexus

Analyze

Plan

Monitor

Release

IBM Application Discovery and Delivery Intelligence
splunk>

IBM Engineering Workflow Management
Jira
Confluence

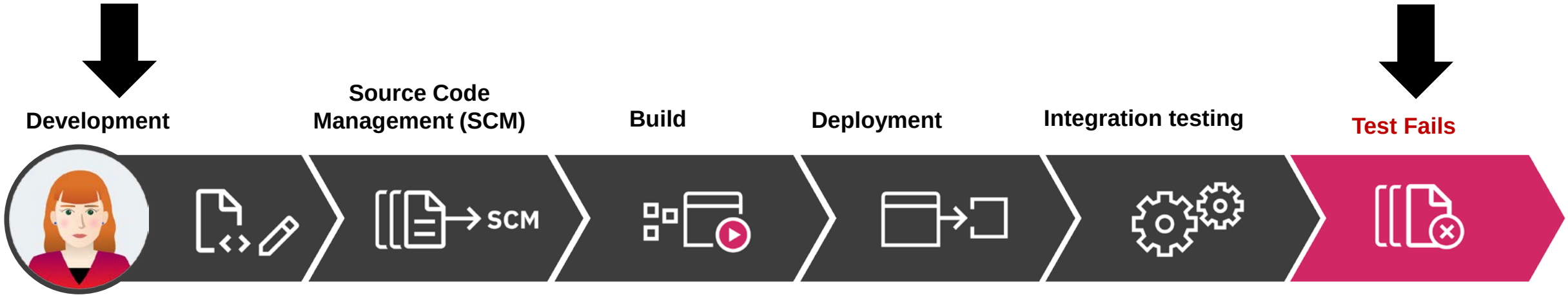
IBM Z APM Connect
OMEGAMON Operational Insights
APPDYNAMICS

urban {code}
cloudbees

Shift Left testing

Start your testing here

Don’t wait until here



Deb, The Developer

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

Checks in code

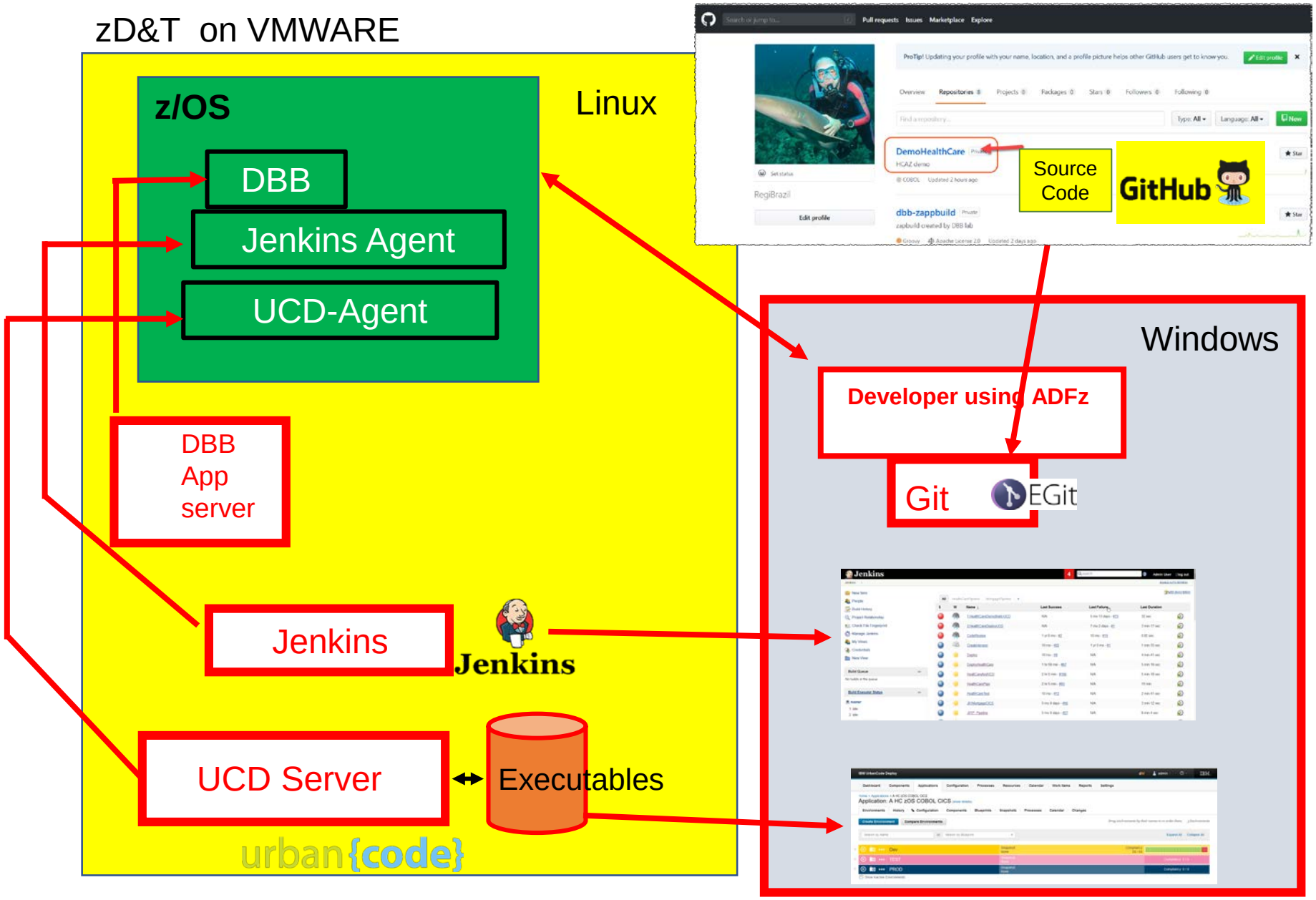
Runs build

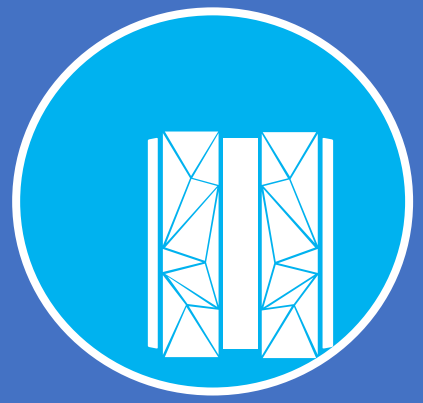
Deploys changes to the target environment

Full integration test

Shift Left

Topology used on this Demo





Z DevOps Tooling

Demonstration

COBOL/CICS/DB2 Program

PART 1 - Z Unit Test using IDz (zUnit)

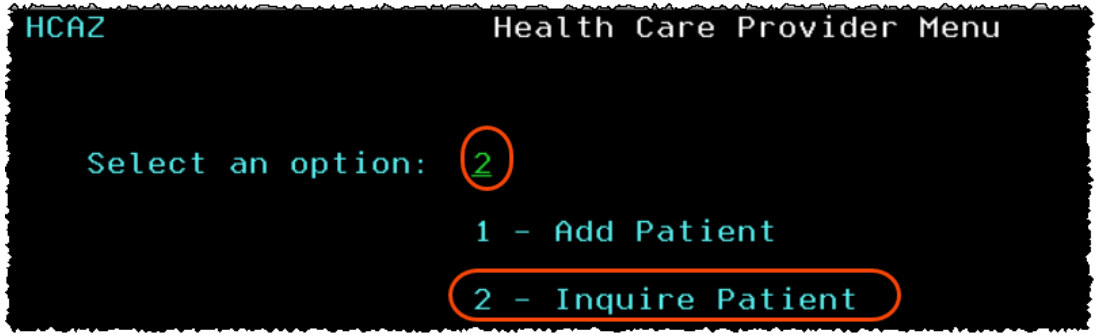
Demonstration

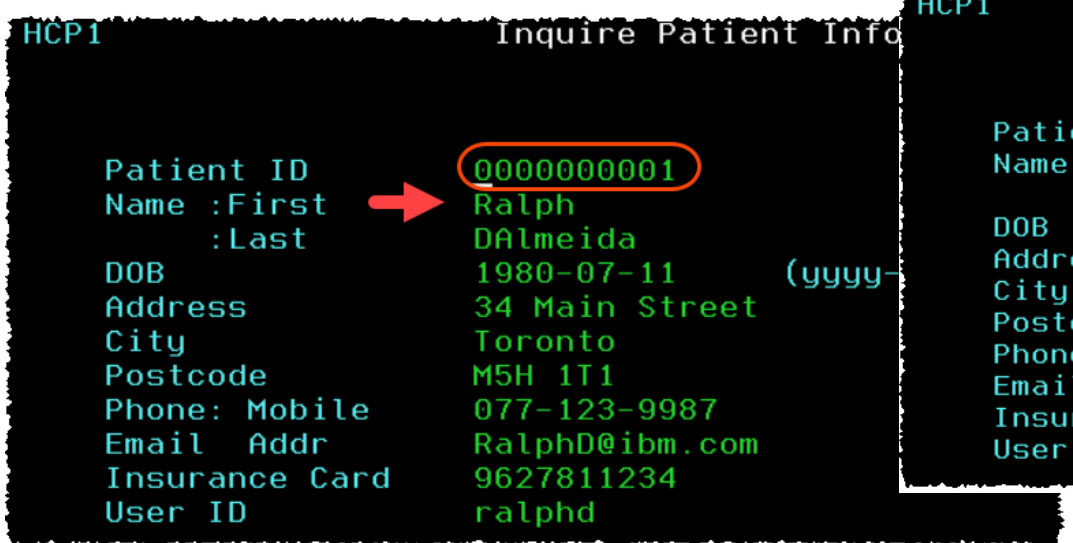
Scenario: COBOL CICS/DB2 Transaction

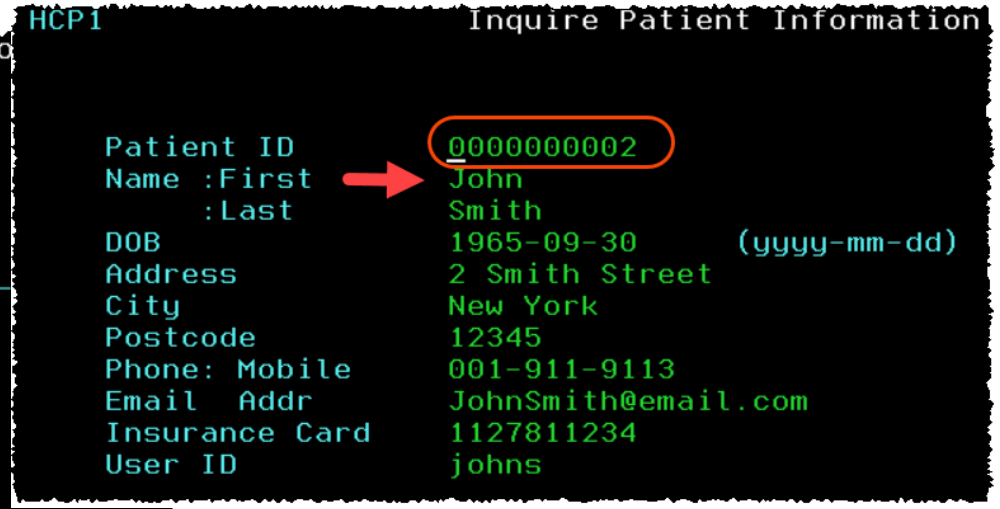
*Using transaction **HCAZ** option **2 (Inquire Patient)** and **Patient ID 1** press **PF12***
*Run second test case using **Patient ID 2***

***Patient information** is displayed by program* `HCIPDB01`

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





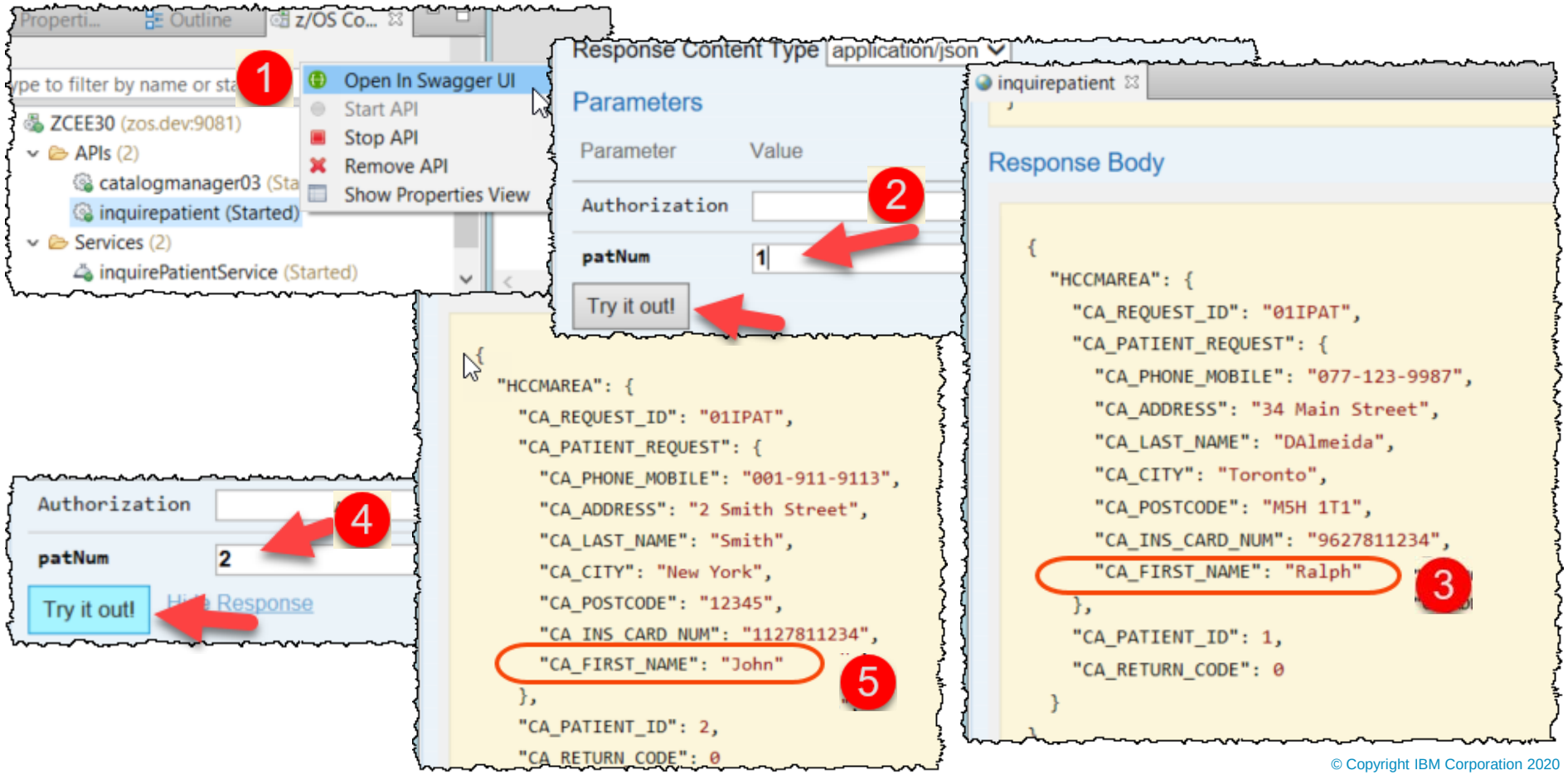


Scenario: COBOL CICS/DB2 Transaction using z/OS Connect

Invoking the z/OS Connect Restful API service ***inquirepatient*** *and* ***patNum 1***
Run second test case using ***patNum 2***

Patient information *is displayed by z/OS Connect using program* `HCIPDB01`

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



PART #1 – Unit test on COBOL/CICS program HCIPDB01

Developer A

1. Load the COBOL code from **Git** (*GitHub*)

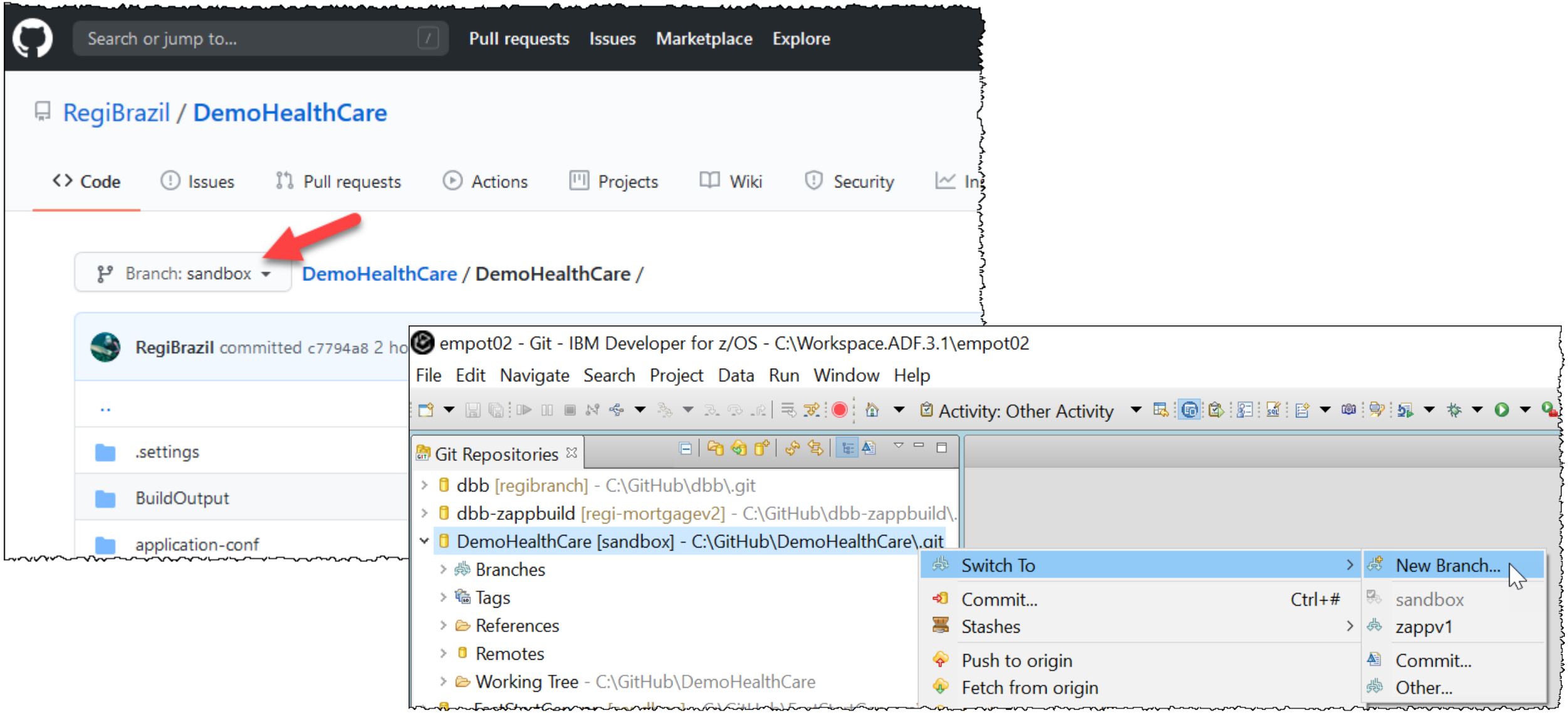
2. Before any change, developer records CICS program interactions using **IDz**

3. Generate, build and run the **zUnit** test
   Compile and link-edit the generated unit test programs using **IBM DBB**,
   followed by running the unit test.

4. Modify the program and rerun the **zUnit** test
   Modify the program under test and introduce a bug (bad name for Patient ID #1,
   Rerun the unit test, and observe the failure of the test case.
   All done in batch.. No need to start CICS.. No need of DB2

1. Load the COBOL code from Git (GitHub)

https://github.com/RegiBrazil/DemoHealthCare/tree/sandbox/DemoHealthCare

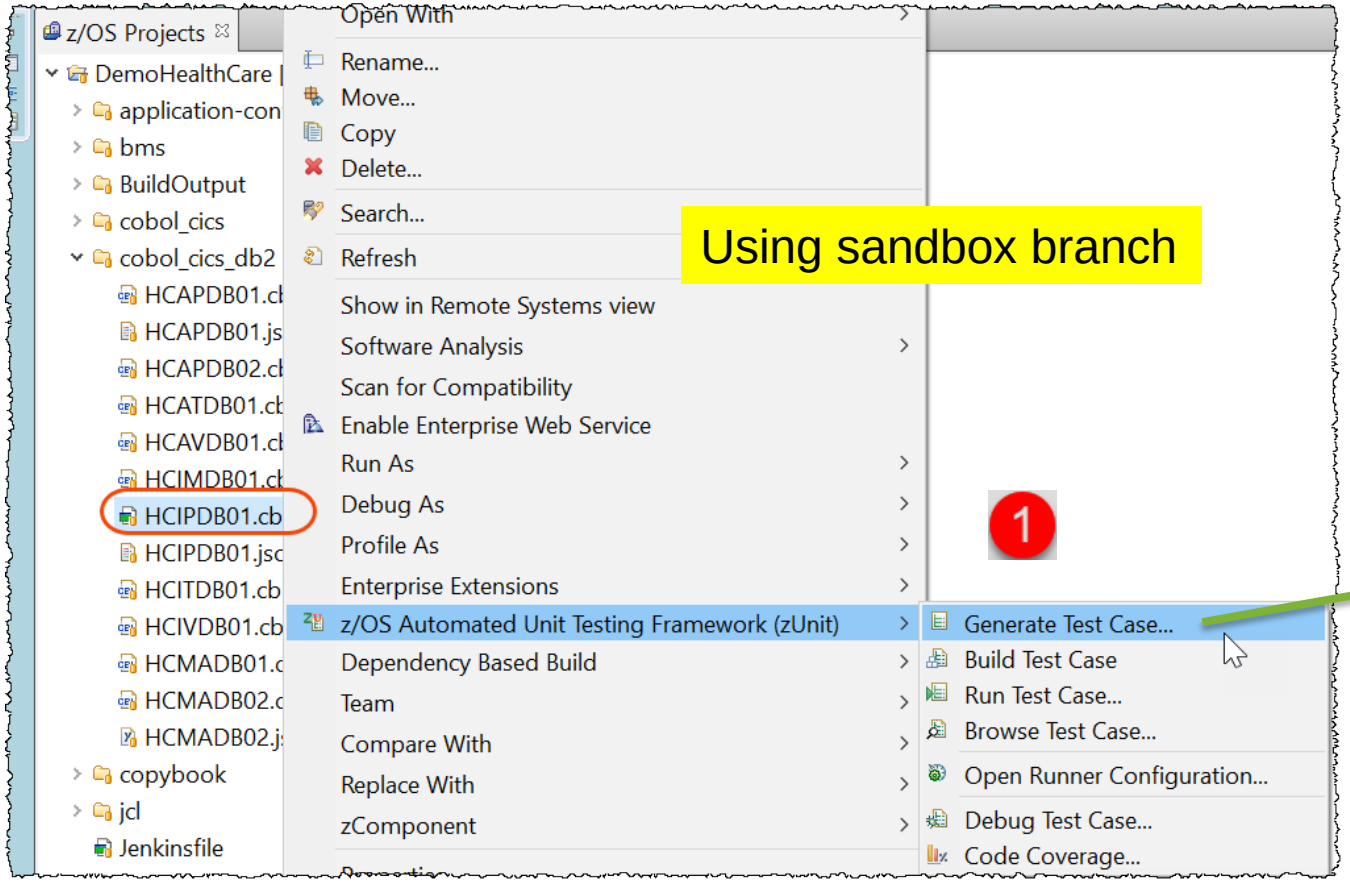


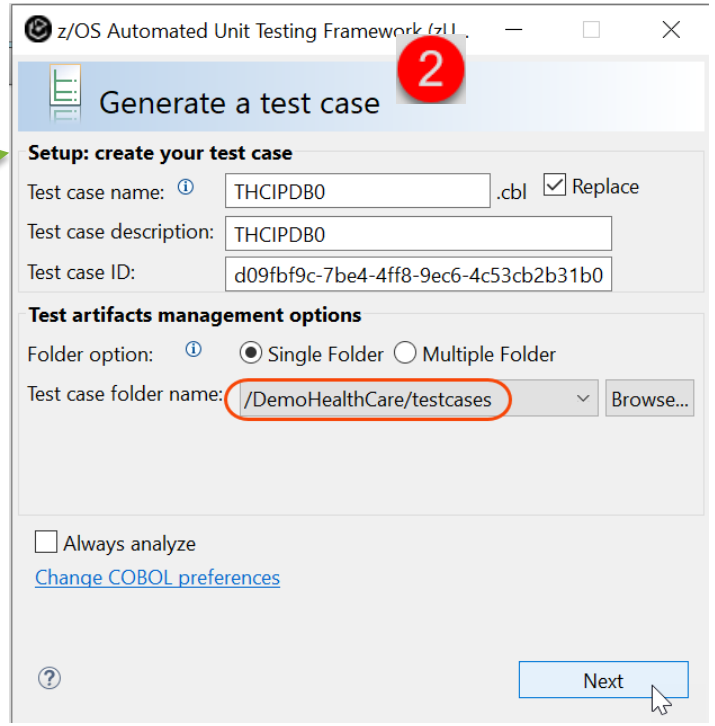
PART #1 – Unit test on COBOL/CICS program HCIPDB01

Developer A

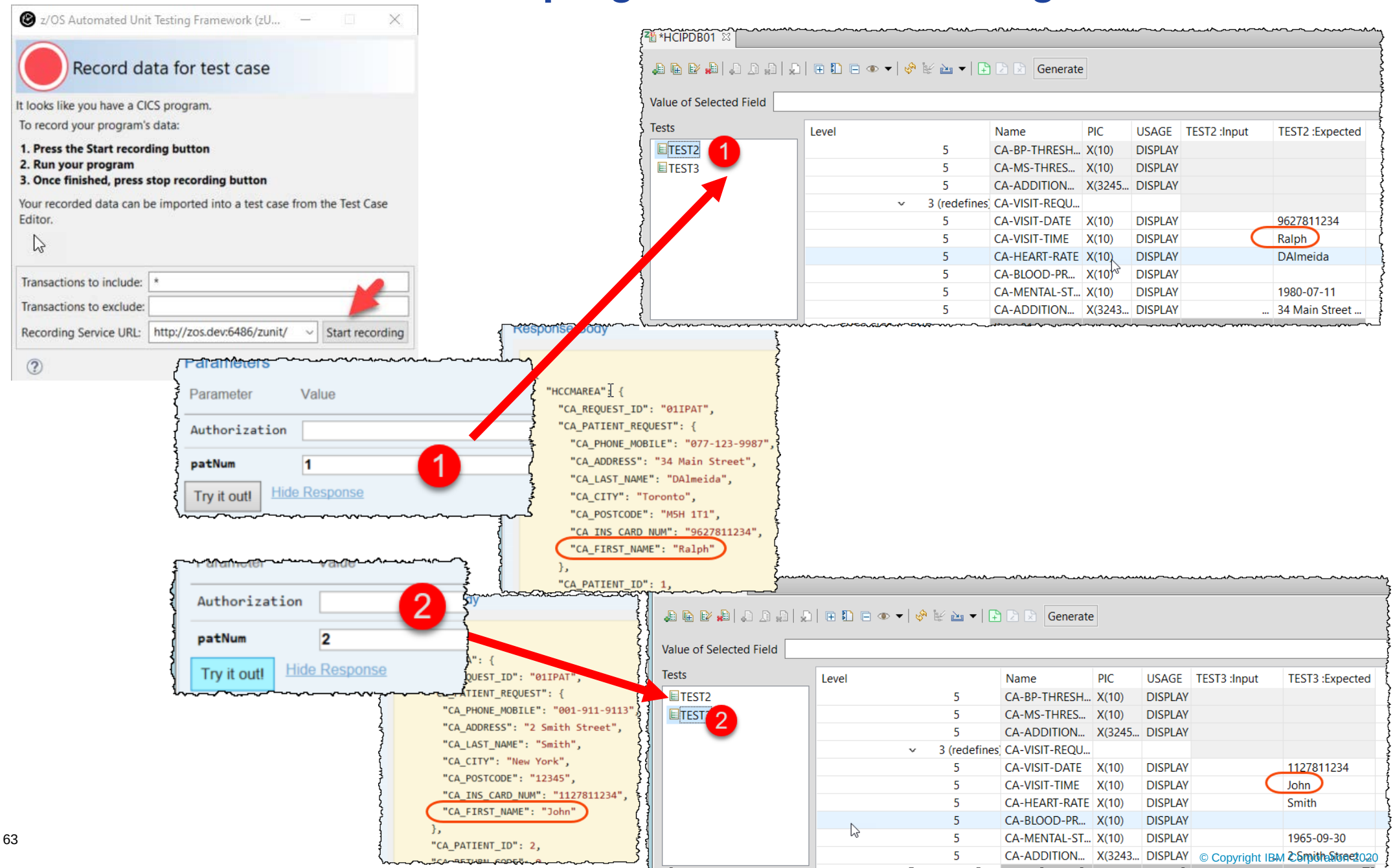
1. Load the COBOL code from **Git** (*GitHub*)
2. Before any change, developer records CICS program interactions using **IDz**
3. Generate, build and run the zUnit test
   Compile and link-edit the generated unit test programs using IBM DBB,
   followed by running the unit test.
4. Modify the program and rerun the zUnit test
   Modify the program under test and introduce a bug (bad name for Patient ID #1,
   Rerun the unit test, and observe the failure of the test case.
   All done in batch.. No need to start CICS.. No need of DB2

2. Record CICS program interaction using IDz …

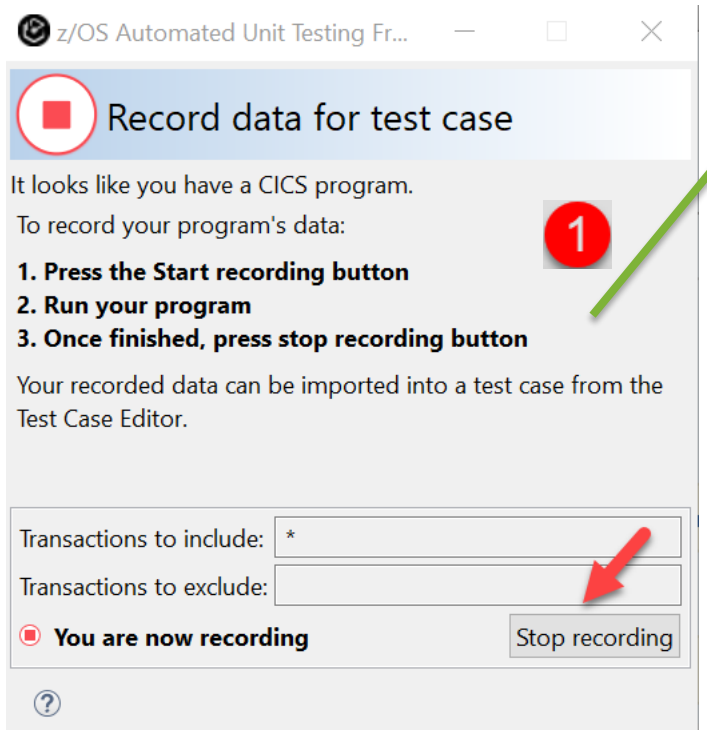


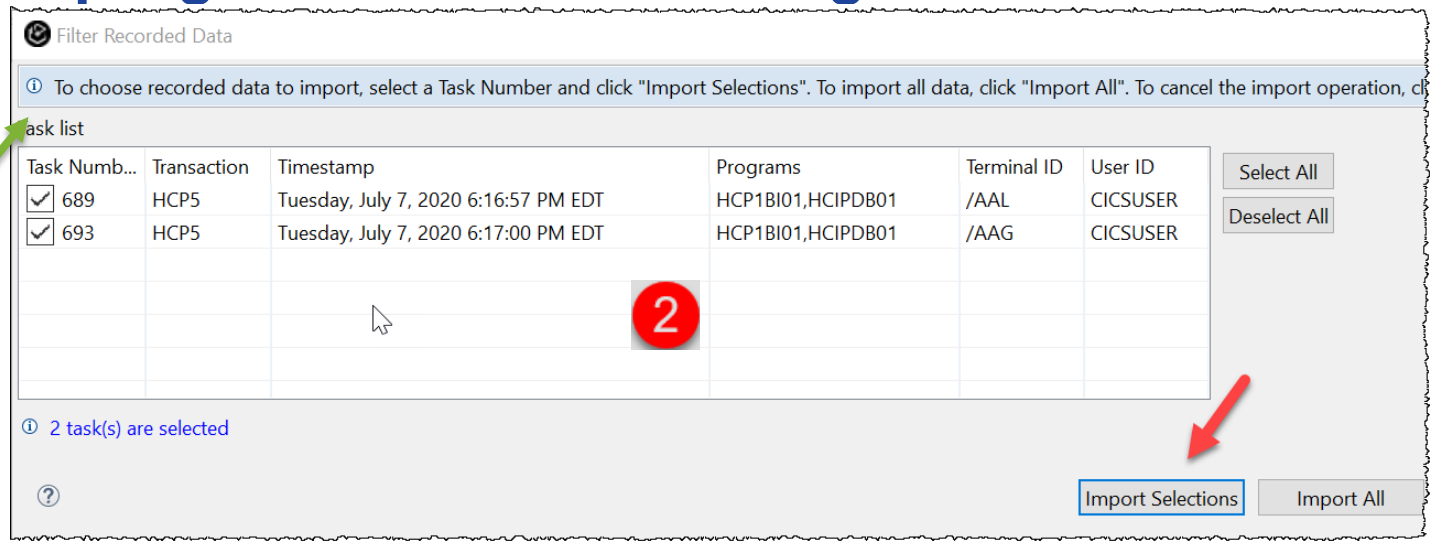


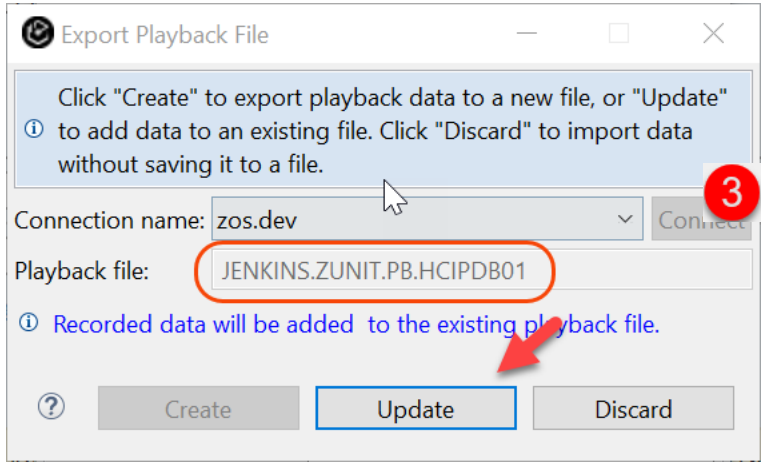
2. Record CICS program interaction using IDz

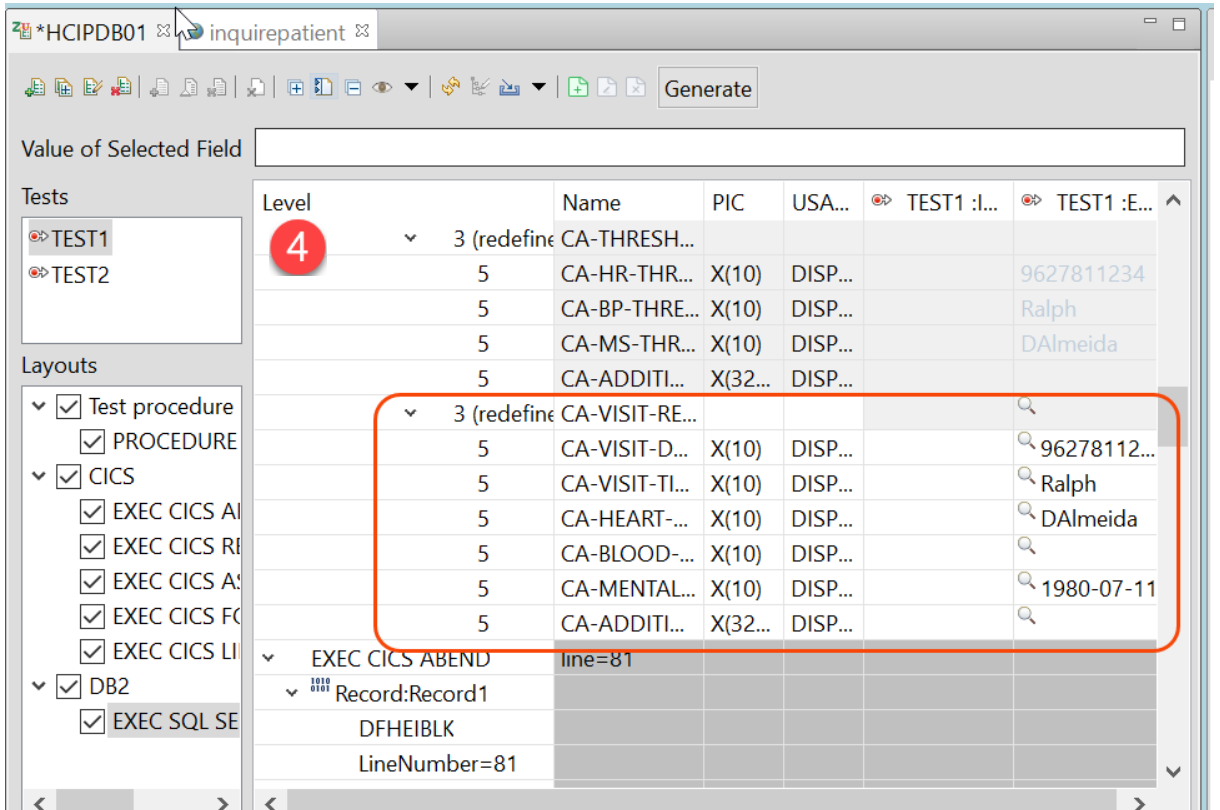


2. Record CICS program interaction using IDz …









PART #1 – Unit test on COBOL/CICS program HCIPDB01

Developer A

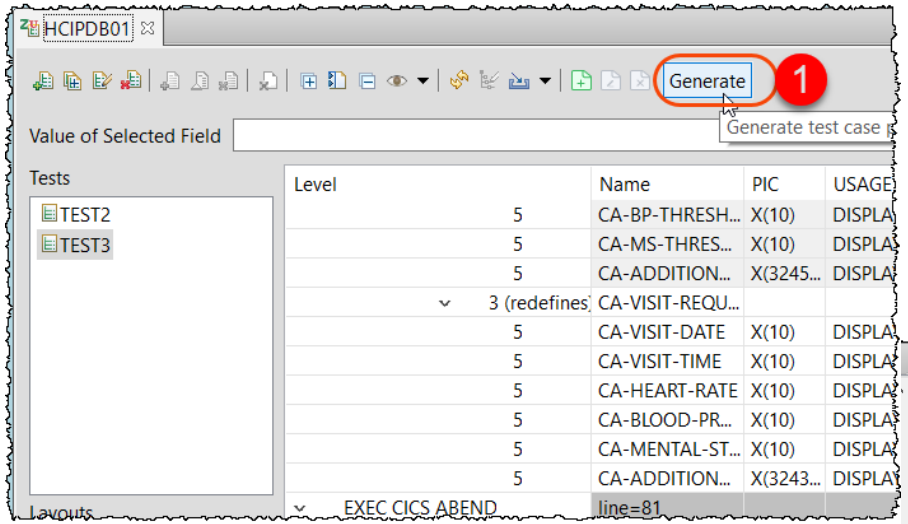
1. Load the COBOL code from **Git** (*GitHub*)

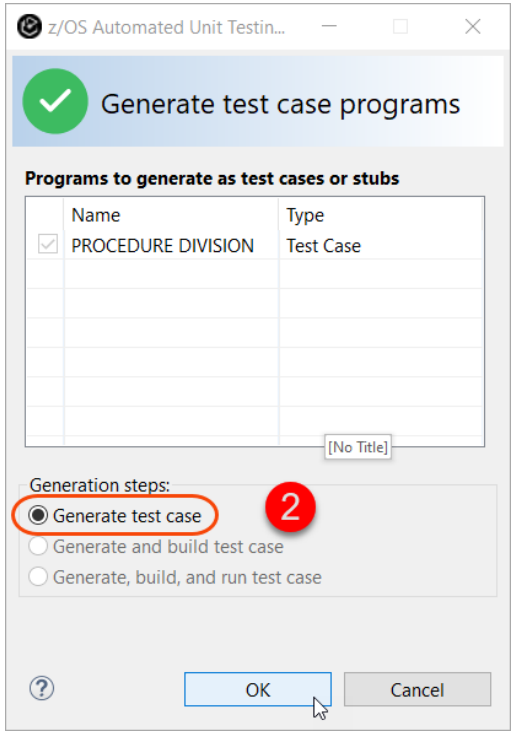
2. Before any change, developer records CICS program interactions using **IDz**

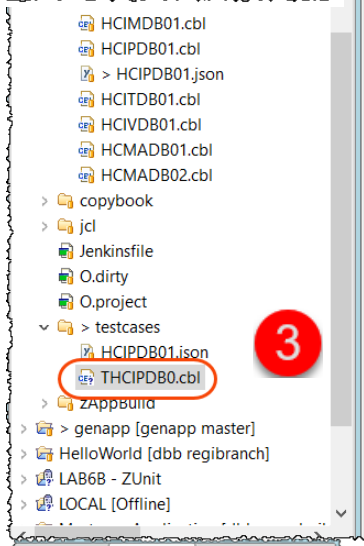
3. Generate, build and run the **zUnit** test
   Compile and link-edit the generated unit test programs using **IBM DBB**,
   followed by running the unit test.

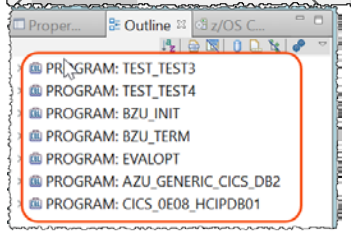
4. Modify the program and rerun the zUnit test
   Modify the program under test and introduce a bug (bad name for Patient ID #1,
   Rerun the unit test, and observe the failure of the test case.
   All done in batch.. No need to start CICS.. No need of DB2

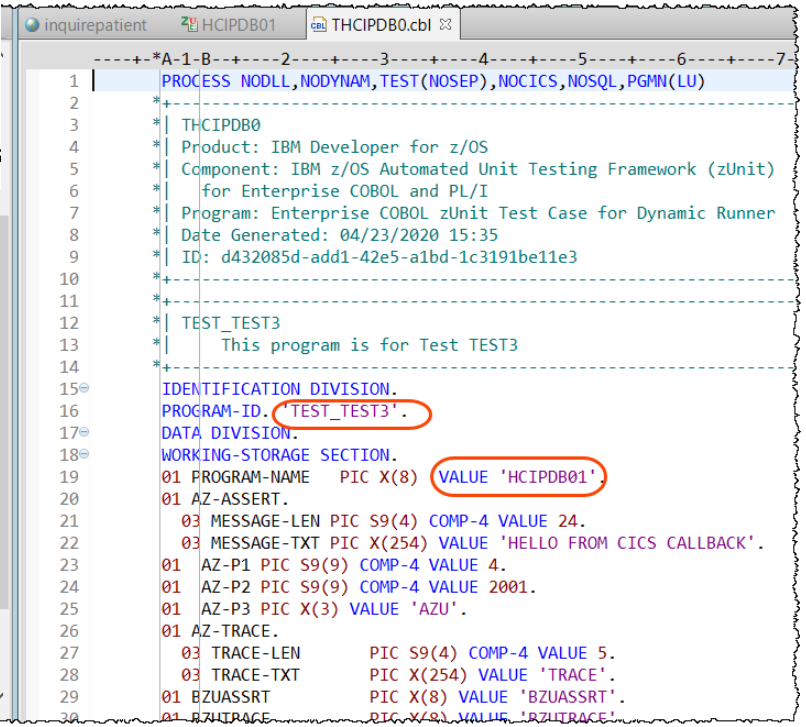
3. Generate, build and run the unit test..



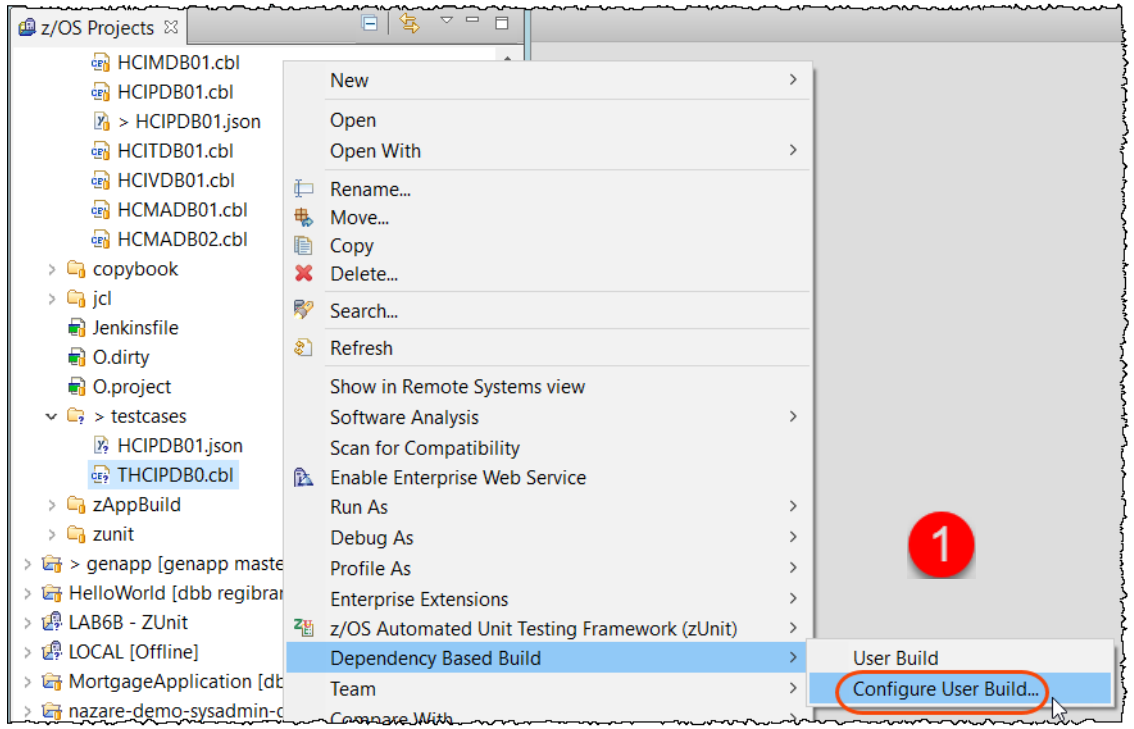


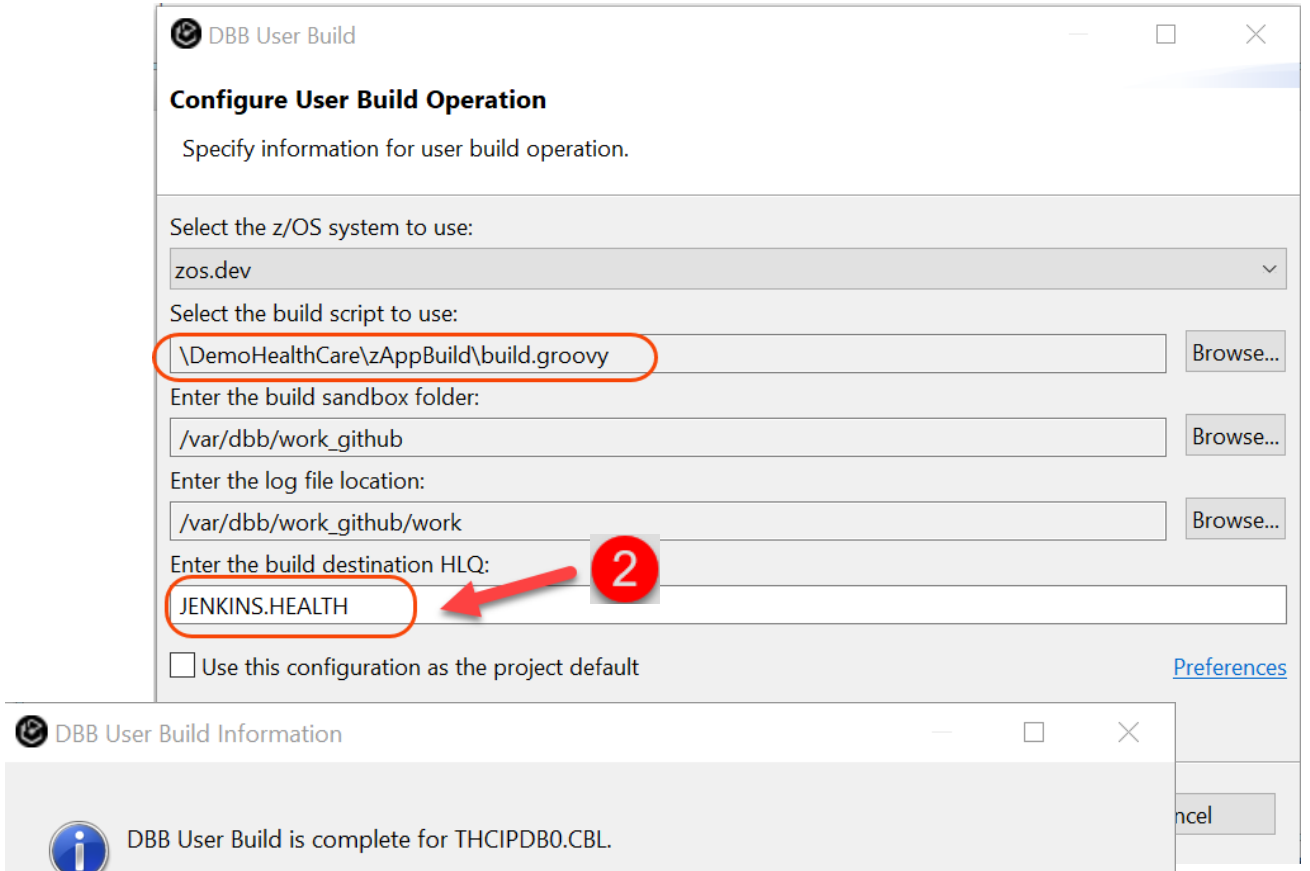


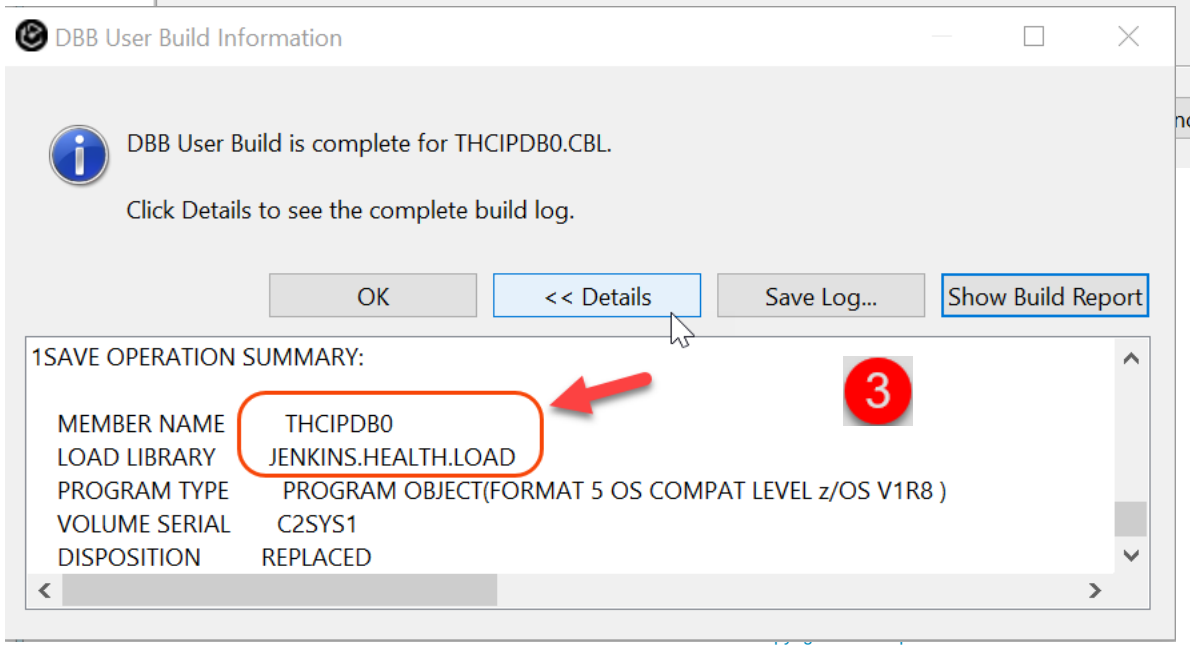




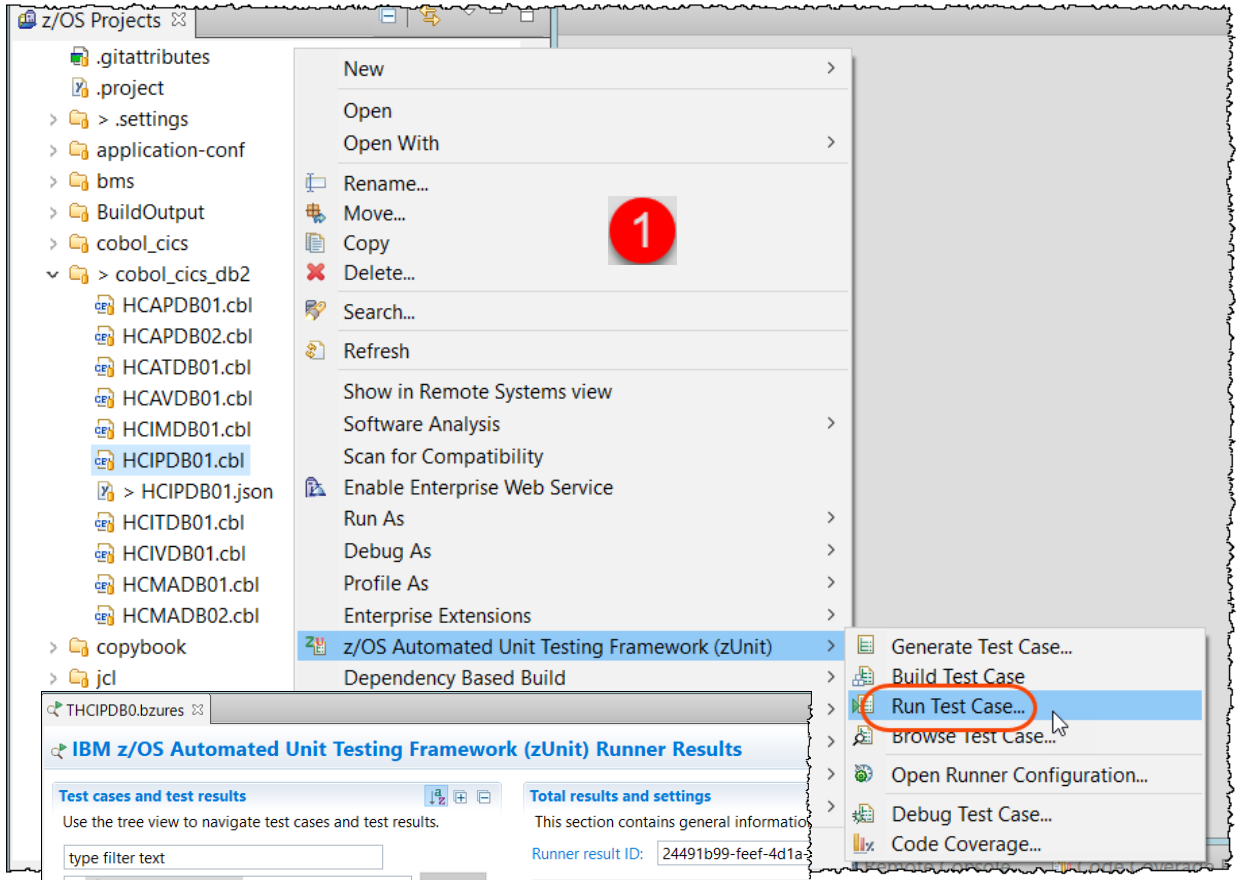
3. Generate, build and run the unit test…

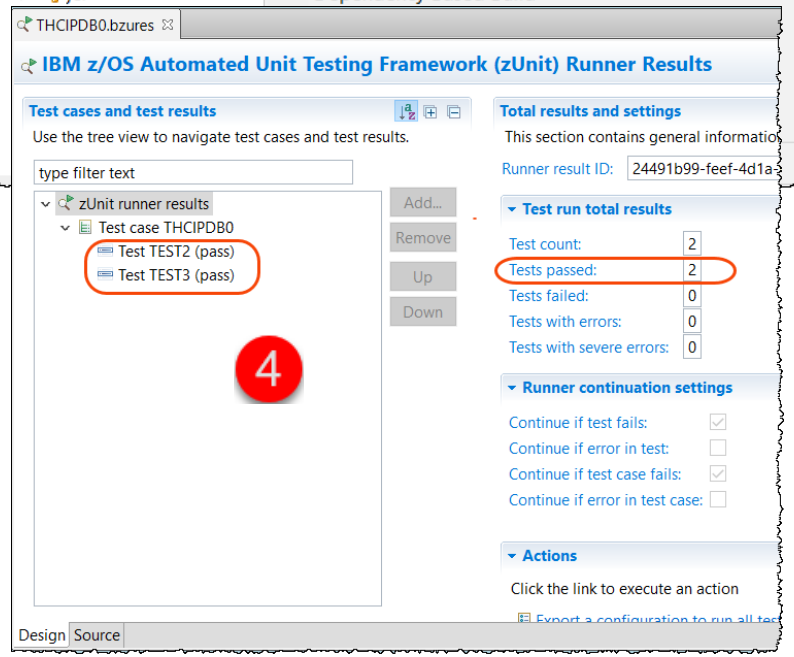


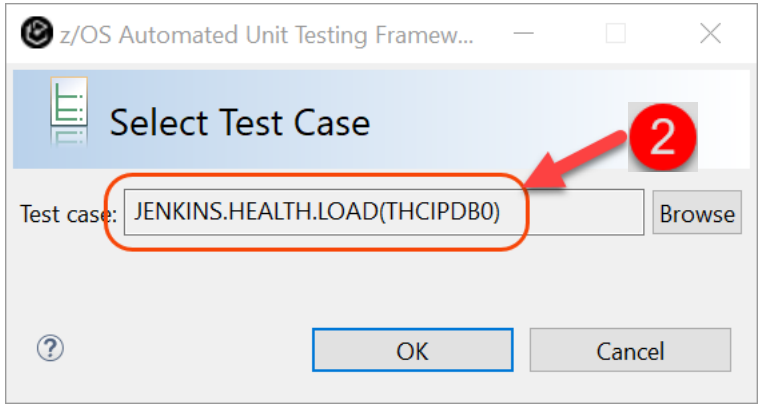


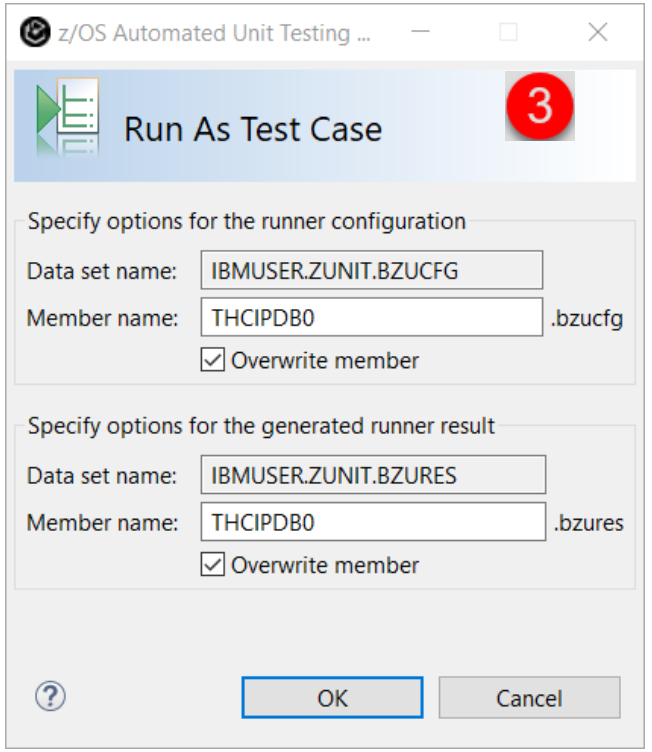


3. Generate, build and run the unit test









PART #1 – Unit test on COBOL/CICS program HCIPDB01

Developer A

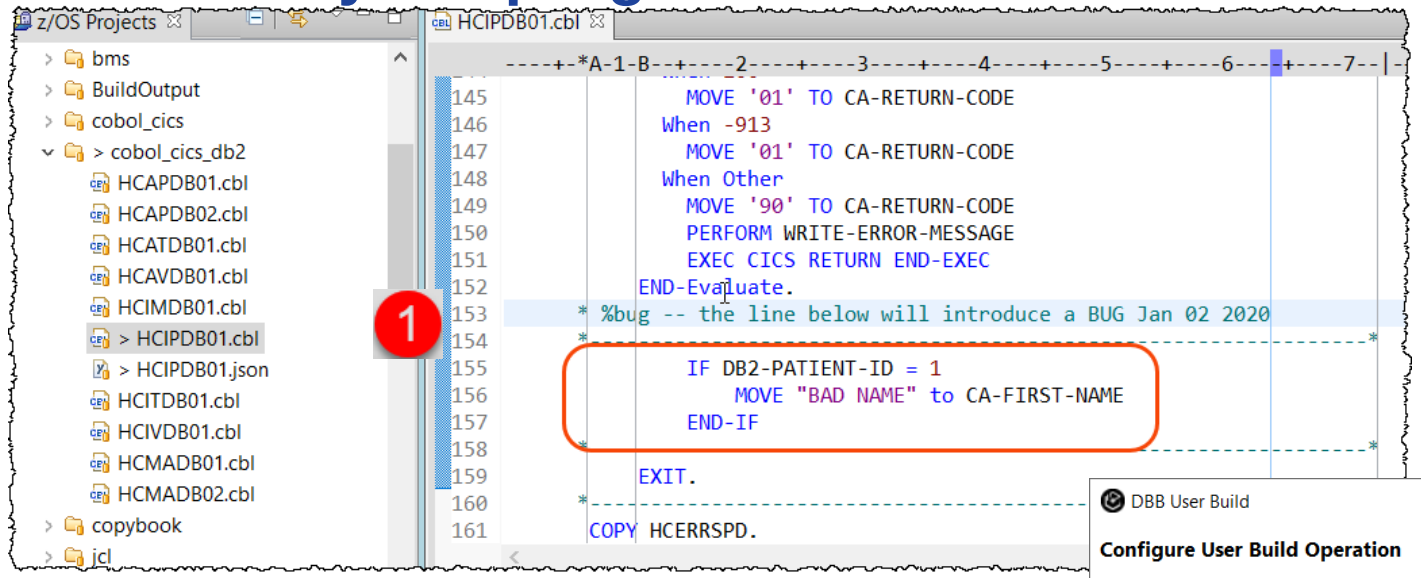
1. Load the COBOL code from **Git** (*GitHub*)

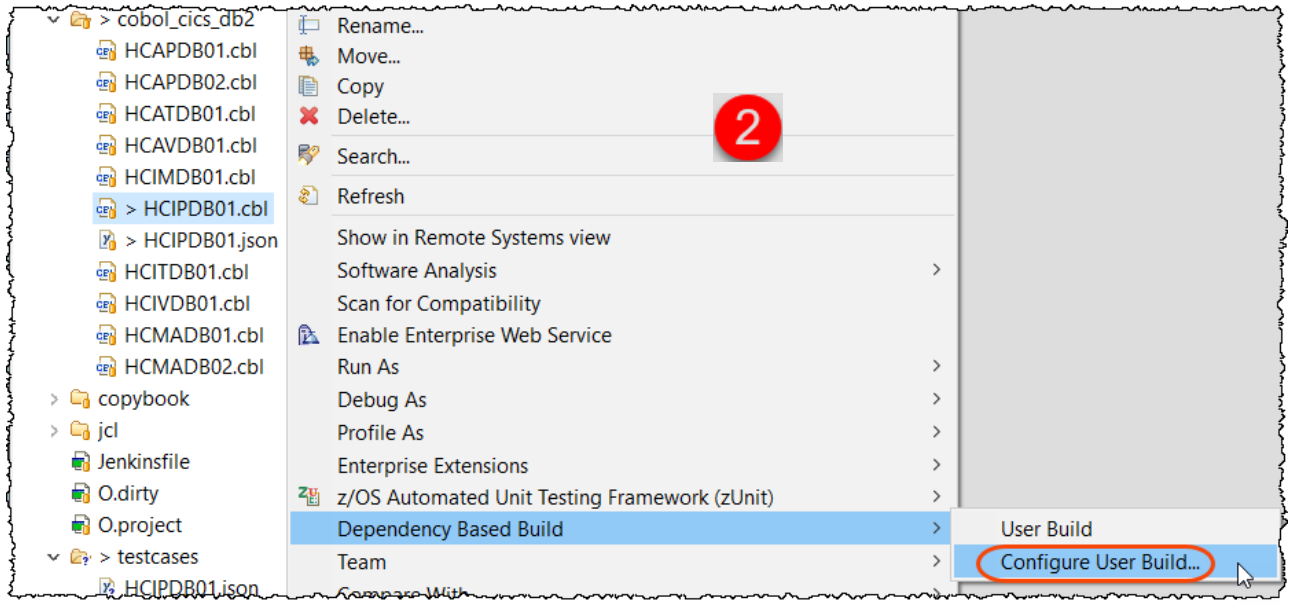
2. Before any change, developer record CICS program interactions using **IDz.**

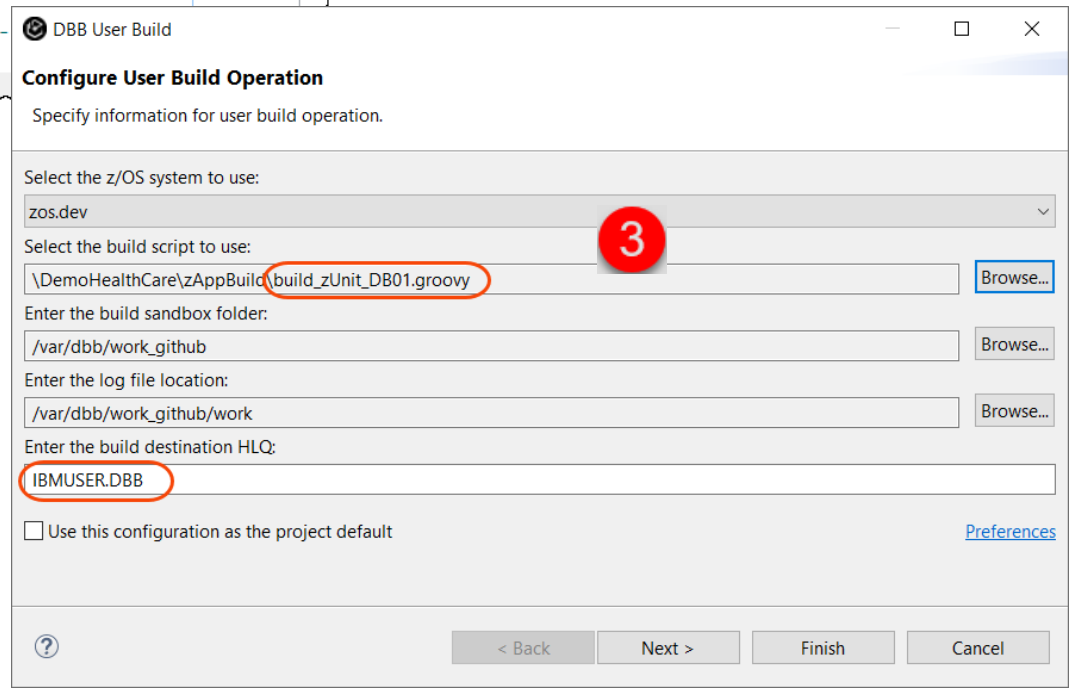
3. Generate, build and run the **zUnit** test
   Compile and link-edit the generated unit test programs using **IBM DBB**,
   followed by running the unit test.

4. Modify the program and rerun the **zUnit** test
   Modify the program under test and introduce a bug (bad name for Patient ID #1,
   Rerun the unit test, and observe the failure of the test case.
   All done in batch.. No need to start CICS. No need of DB2..

4. Modify the program and rerun the unit test…

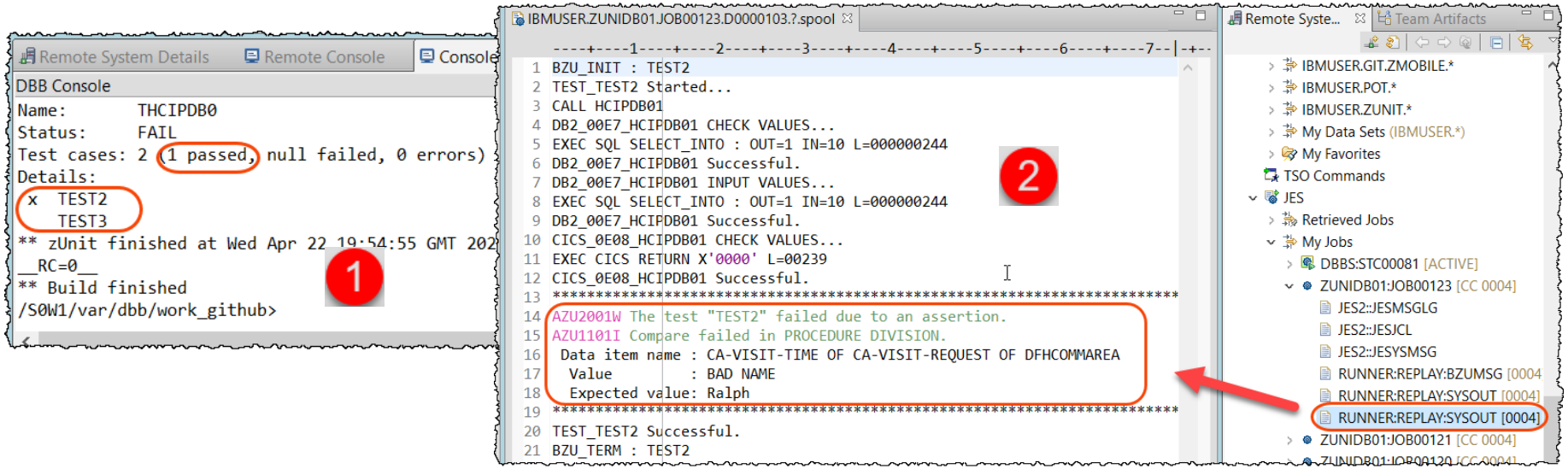




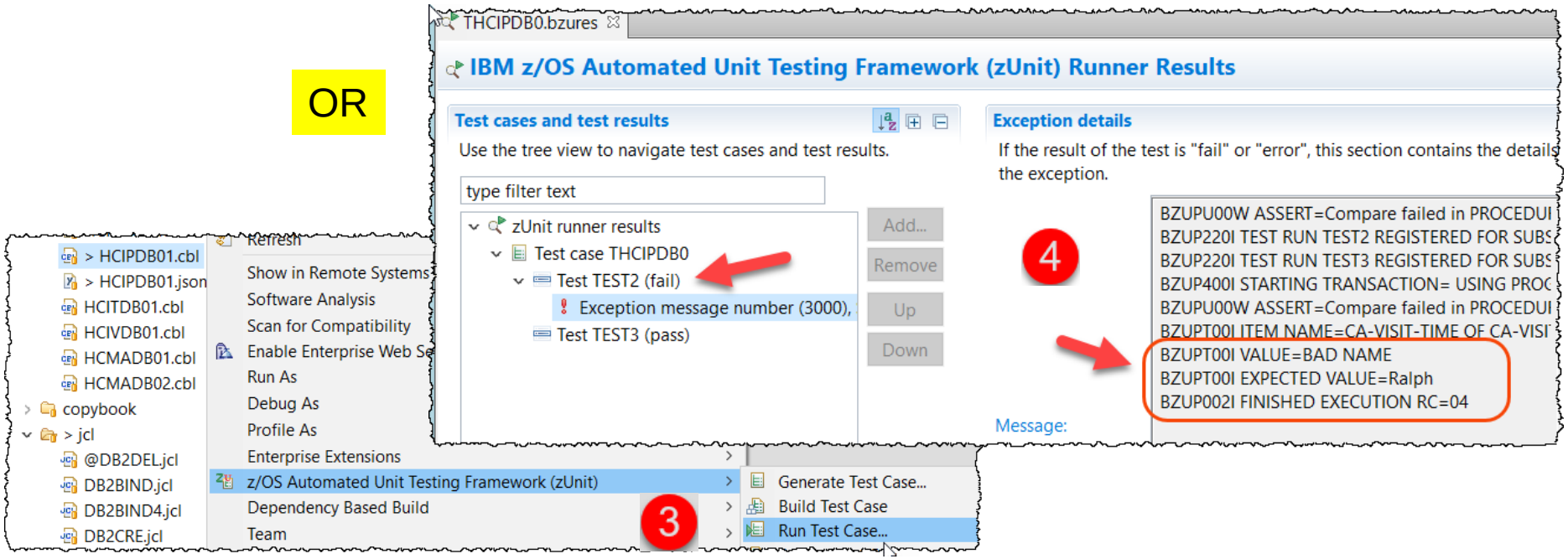



```
Using \DemoHealthCare\zAppBuild\build_zUnit_DB01.groovy
```

4. Modify the program and rerun the unit test



OR



PART #1 – Unit test on COBOL/CICS program HCIPDB01

Developer A

1. Load the COBOL code from **Git** (*GitHub*)

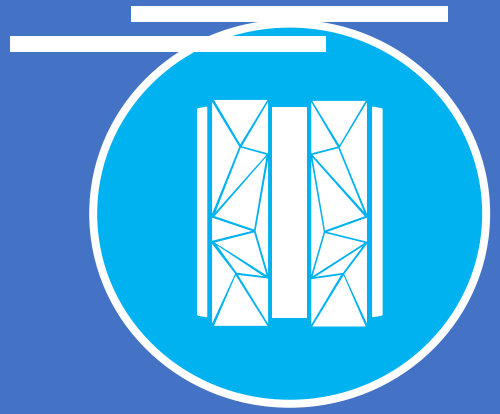
2. Before any change, developer records CICS program interactions using **IDz.**

3. Generate, build and run the **zUnit** test
   Compile and link-edit the generated unit test programs using **IBM DBB**,
   followed by running the unit test.

4. Modify the program and rerun the **zUnit** test
   Modify the program under test and introduce a bug (bad name for Patient ID #1,
   Rerun the unit test, and observe the failure of the test case.

   All done in batch.. No need to start CICS. No need of DB2..

➡ Developer **A** is frustrated.. and go home..



DevOps Mainframe Tooling

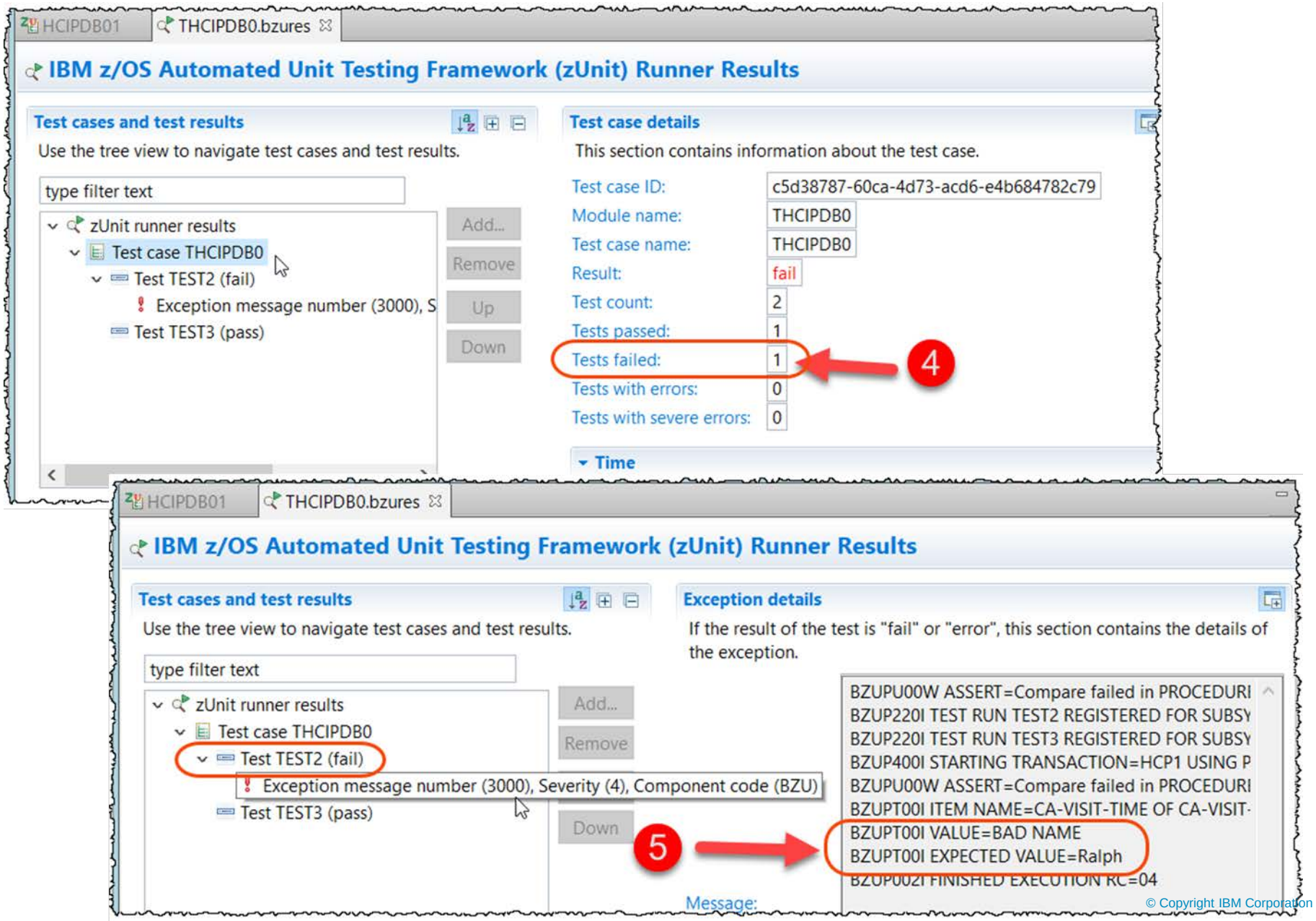
Demonstration

CICS/DB2 COBOL Program

Part 2 - Git/DBB/Jenkins/UCD

Demonstration

Scenario: Developer will use DBB to fix the bug

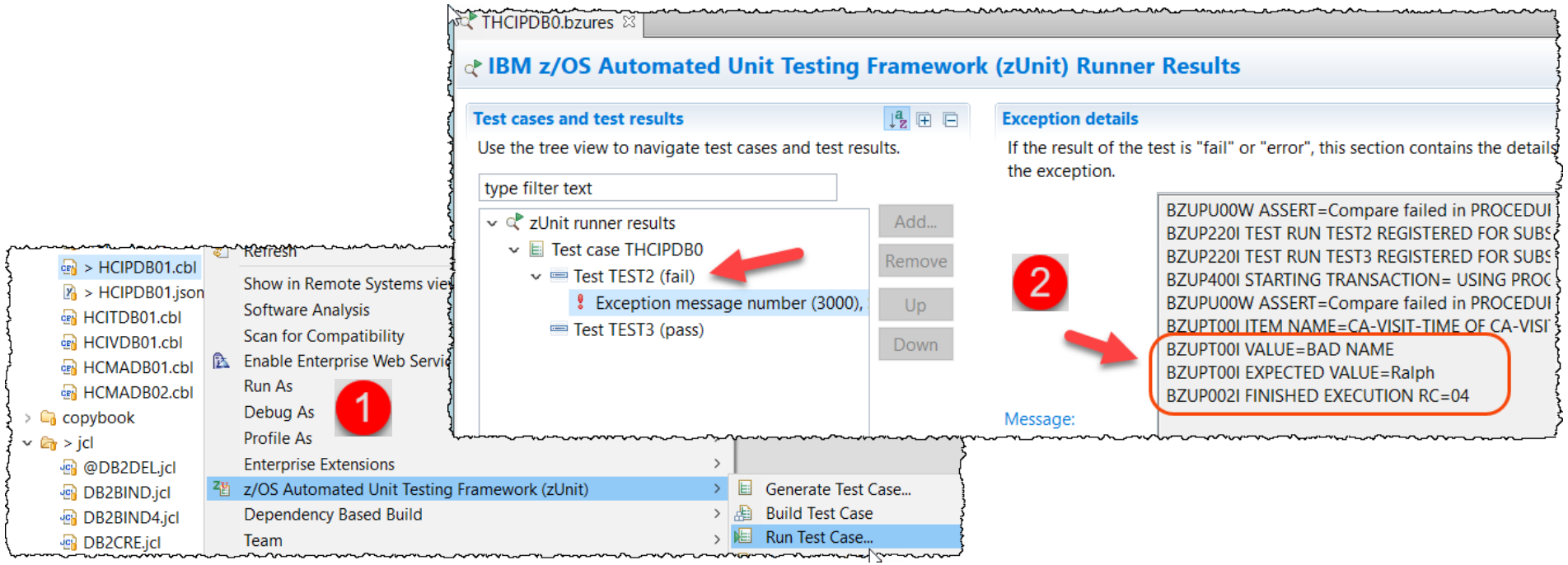


PART #2 – Fix COBOL/CICS HCIPDB01 broken program

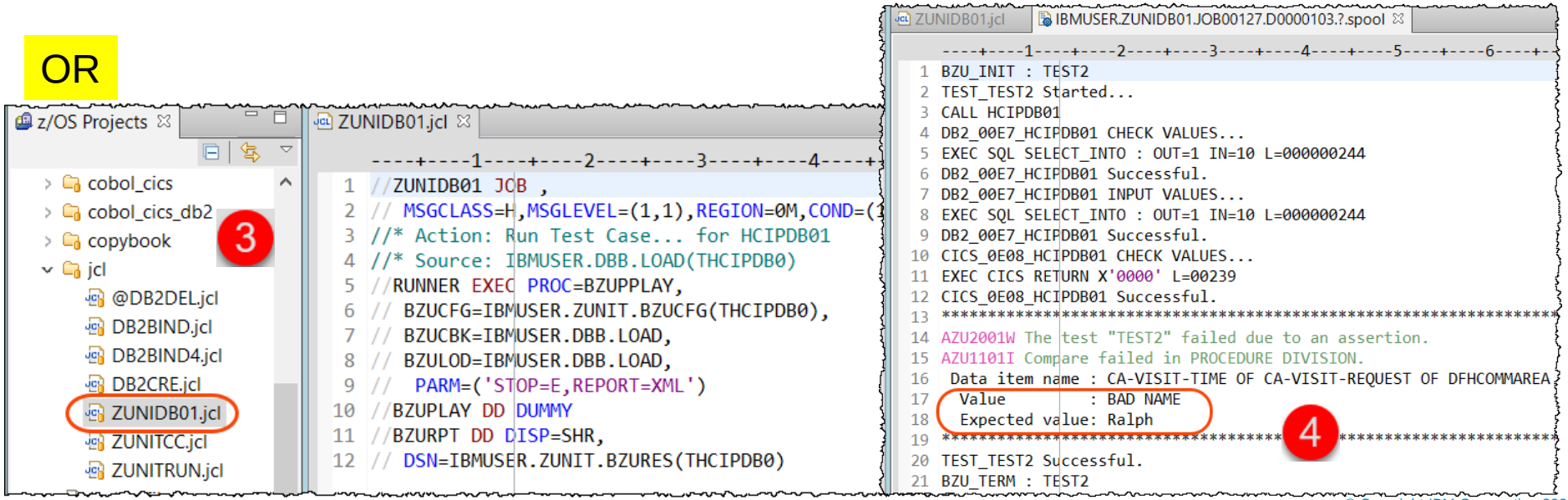
Developer **B** will fix the issue created by developer **A**

1. Developer **B** run the zUnit test case and verify the bug.
   Optionally he could use the Debug (in BATCH) to see the “bug
2. Uses **IDz** to update the COBOL program
3. Uses **DBB** to perform a “*User Build*”, run the unit test (**zUnit**), **Code Coverage** and verify that the abend is fixed
4. Optionally change **zUnit** test case for better **Code Coverage**
5. **Commit and Push** the corrected code to **Git** (*GitHub*)

1. Developer B run the zUnit test case and verify the bug…



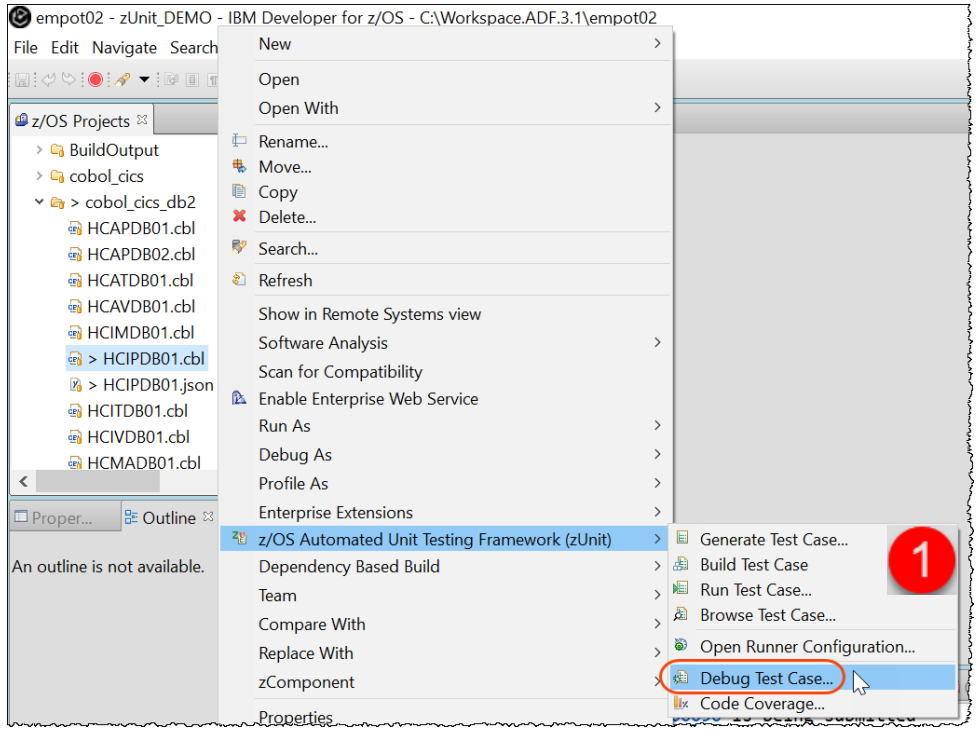
OR

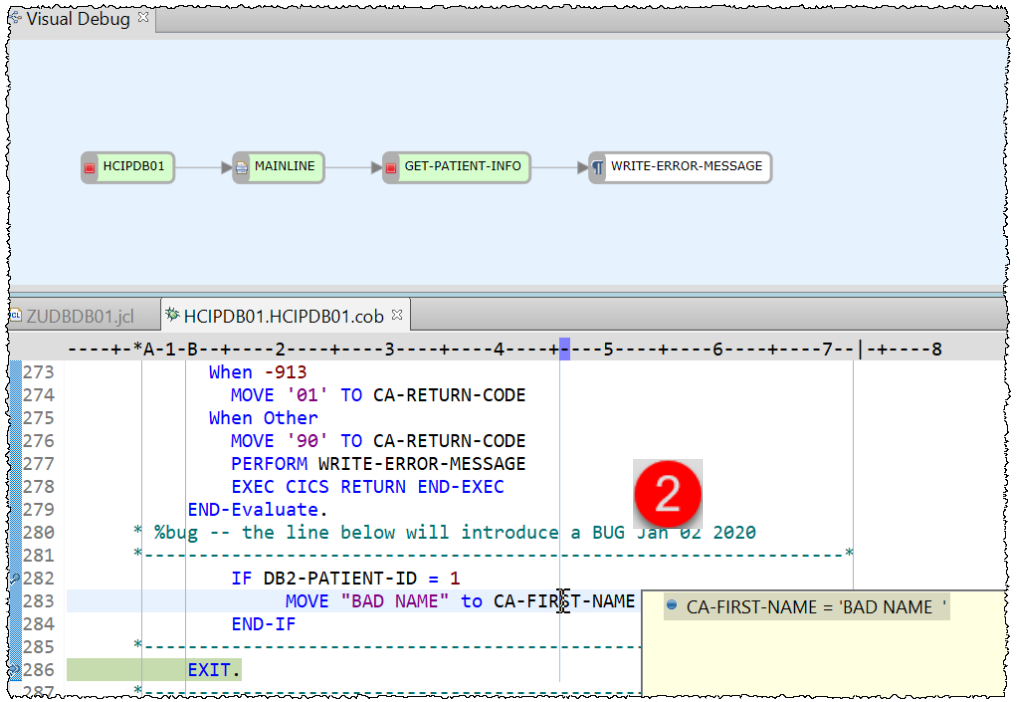


1. Developer B run the zUnit test case and verify the bug..

It could use the Debug (in BATCH) to see the “bug”

All done in batch.. No need to start CICS. No need of DB2..



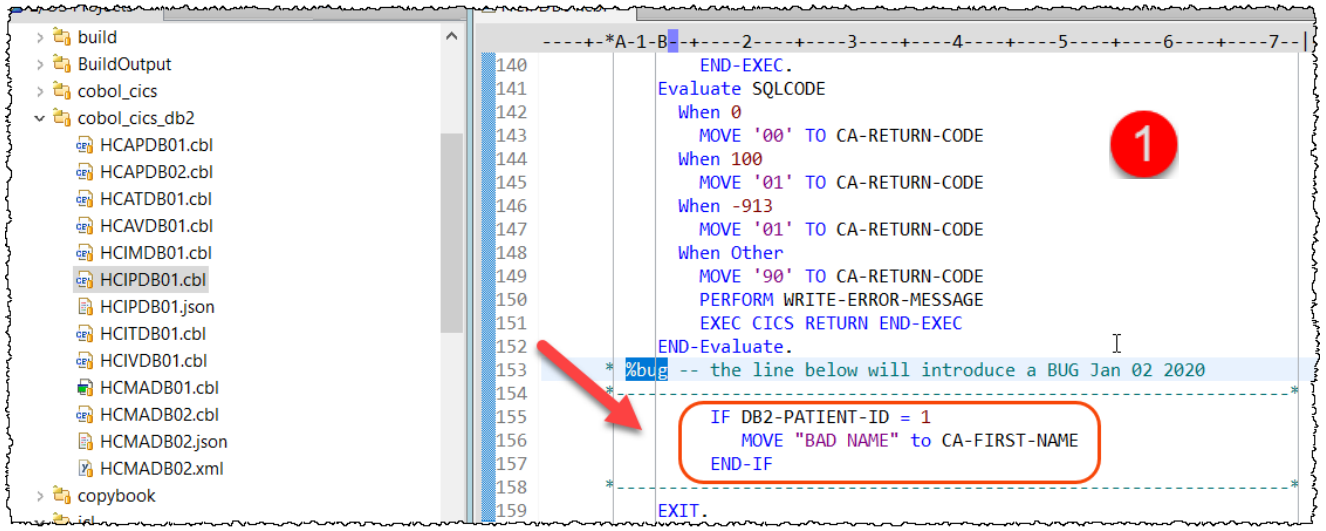


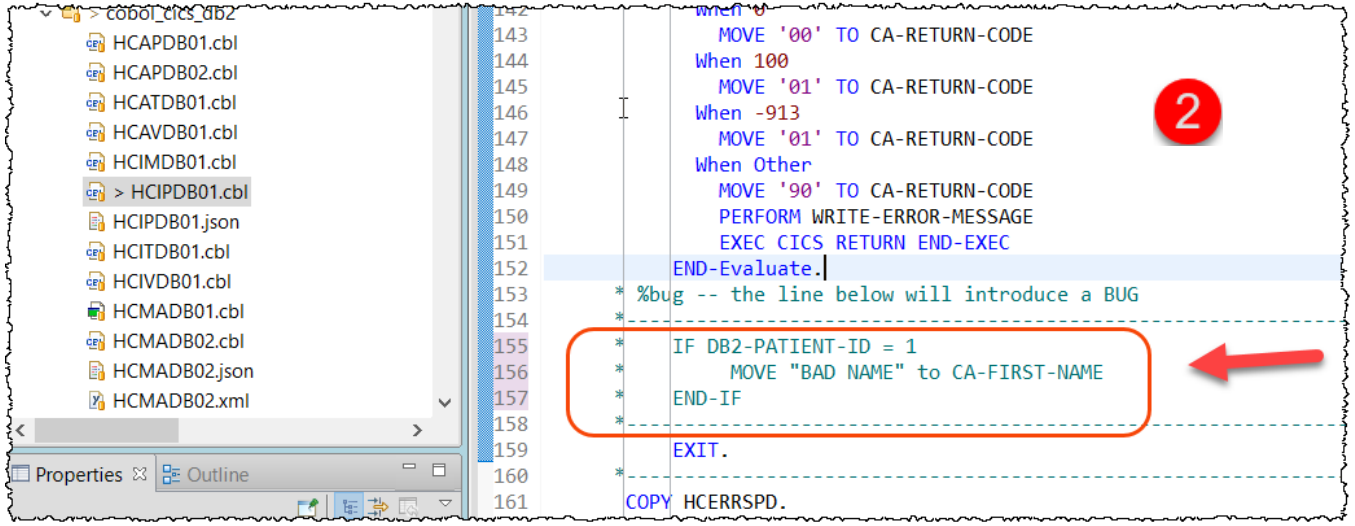
PART #2 – Fix COBOL/CICS HCIPDB01 broken program

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

1. Developer **B** run the zUnit test case and verify the bug.
   Optionally he could use the Debug (in BATCH) to see the “bug
2. Uses **IDz** to update the COBOL program
3. Uses DBB to perform a “User Build”, run the unit test (zUnit), Code Coverage and verify that the abend is fixed
4. Optionally change zUnit test case for better Code Coverage
5. Commit and Push the corrected code to Git (*GitHub*)

2. Uses IDz to update the COBOL program using local copy



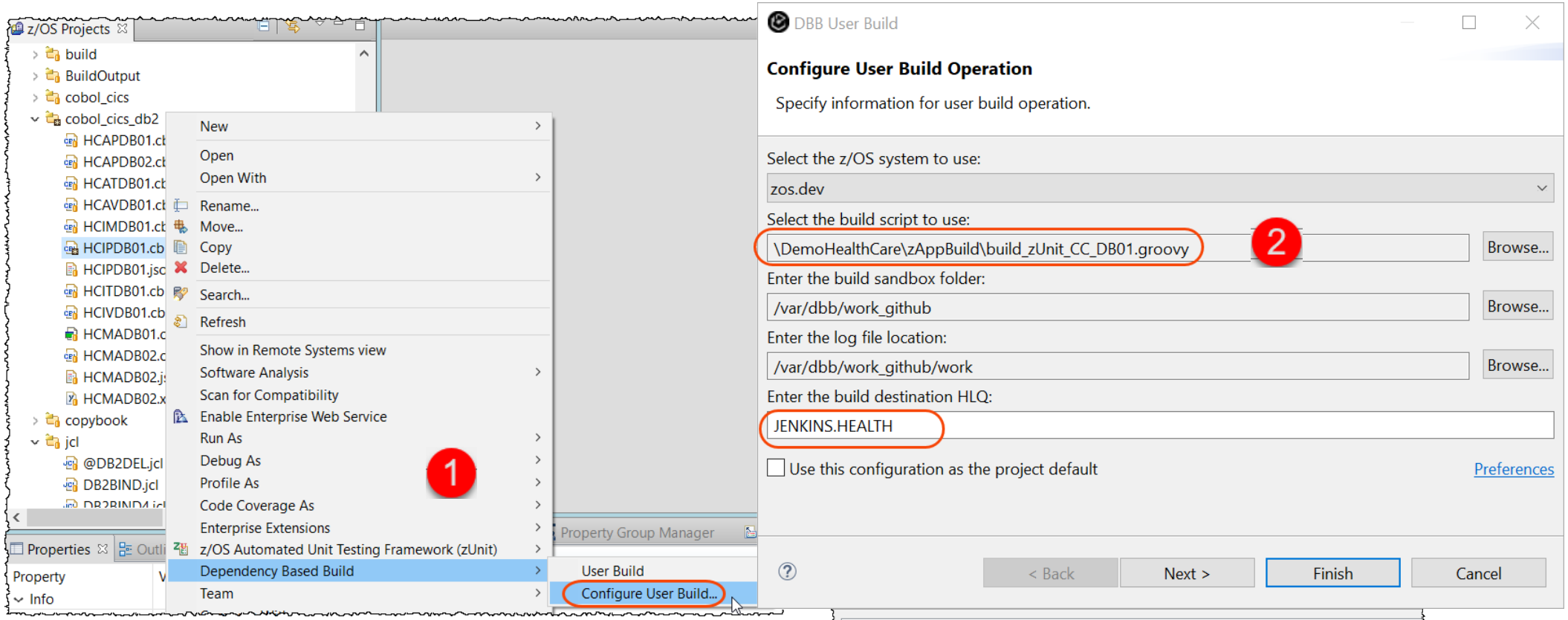


PART #2 – Fix COBOL/CICS HCIPDB01 broken program

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

1. Developer **B** run the zUnit test case and verify the bug.
   Optionally he could use the Debug (in BATCH) to see the “bug
2. Uses **IDz** to update the COBOL program
3. Uses **DBB** to perform a “*User Build*”, run the unit test (**zUnit**), **Code Coverage** and verify that the abend is fixed
4. Optionally change zUnit test case for better Code Coverage
5. Commit and Push the corrected code to Git (*GitHub*)

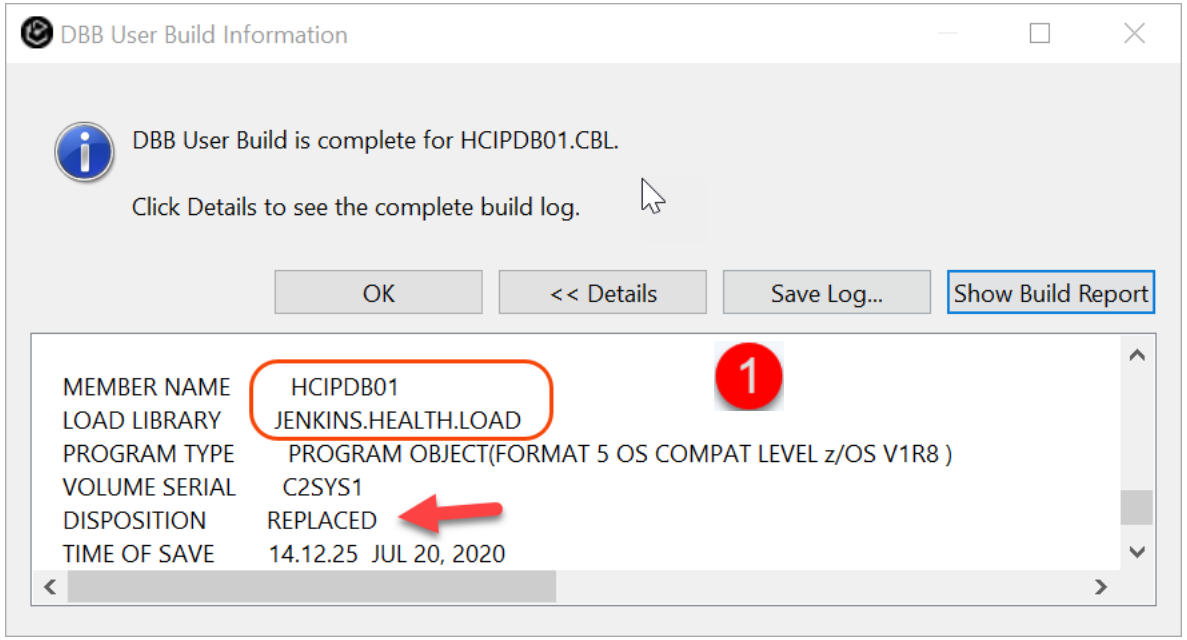
3. Uses DBB to perform a "*User Build*", run the Unit Test (zUnit), Code Coverage and verify that the abend is fixed

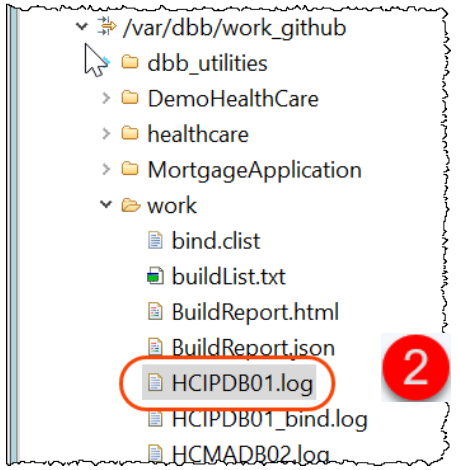


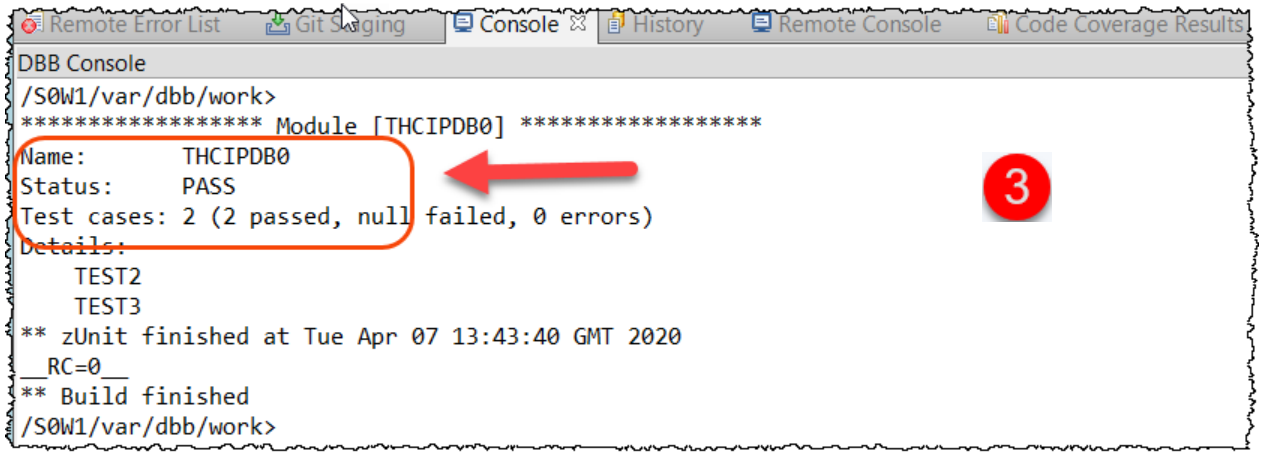
Uses **`\DemoHealthCare\zAppBuild\build_zUnit_CC_DB01.groovy`**

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

All done in batch.. No need to start CICS.. Or DB2

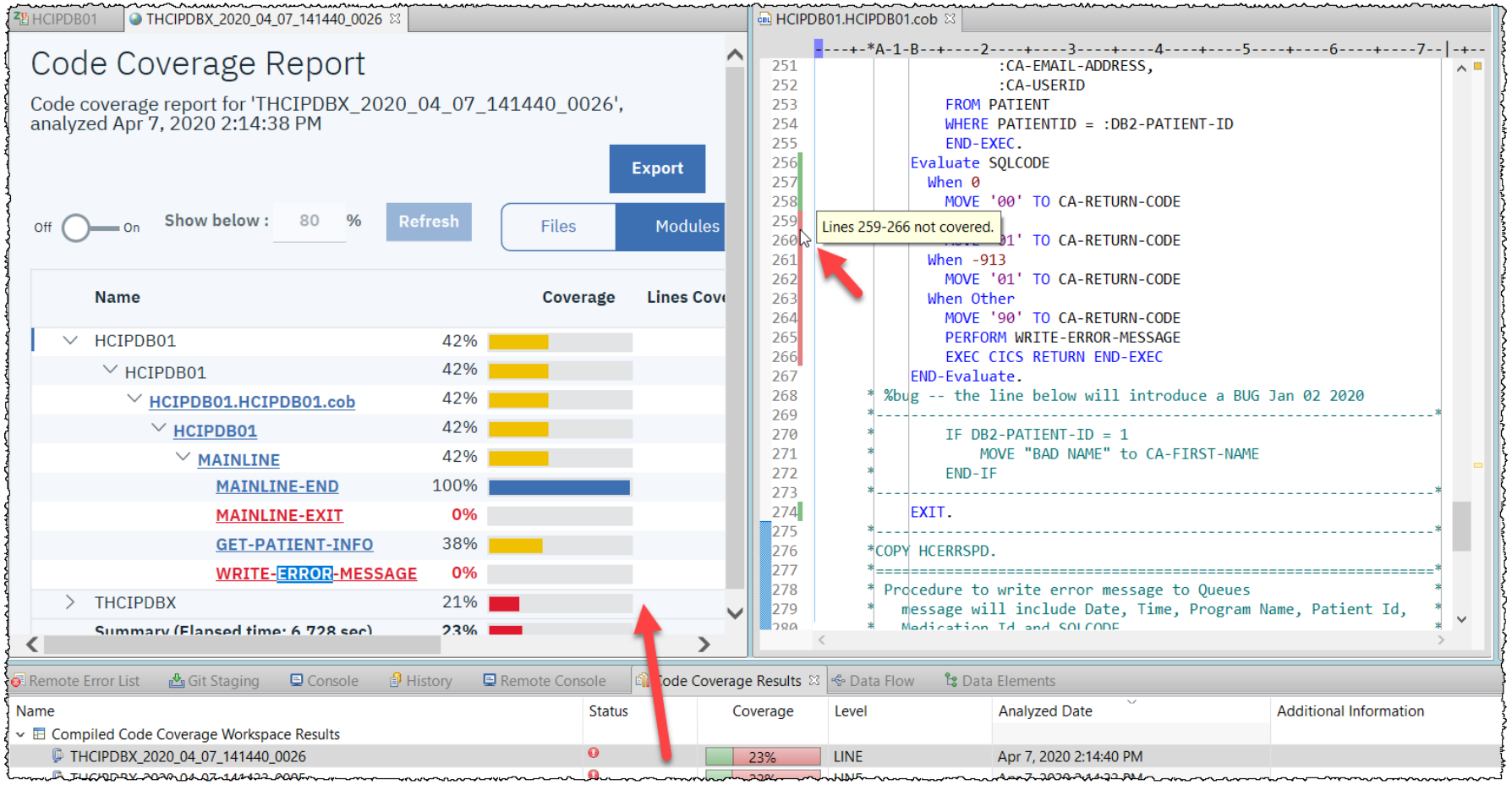






3. Uses DBB to perform a "*User Build*", run the Unit Test (zUnit), Code Coverage and verify that the abend is fixed

All done in batch.. No need to start CICS.. Or DB2

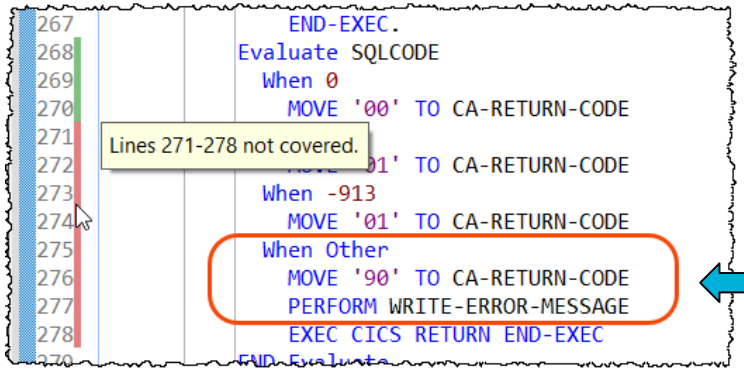


PART #2 – Fix COBOL/CICS HCIPDB01 broken program

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

1. Developer B run the zUnit test case and verify the bug.
   Optionally he could use the Debug (in BATCH) to see the “bug
2. Uses **IDz** to update the COBOL program
3. Uses DBB to perform a “User Build”, run the unit test (zUnit), Code Coverage and verify that the abend is fixed
4. Optionally change zUnit test case for better **Code Coverage**
5. Commit and Push the corrected code to Git (*GitHub*)

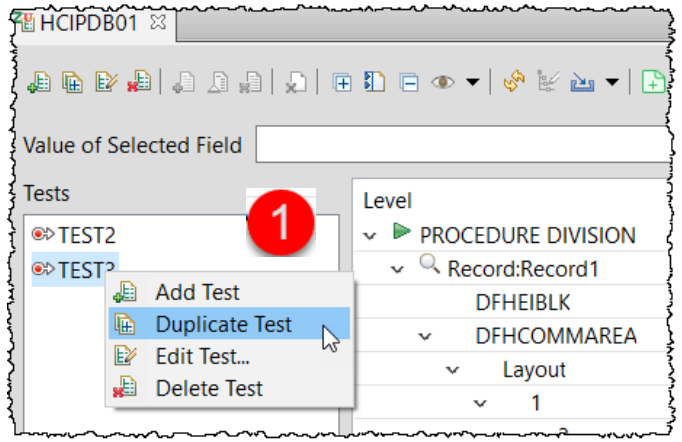
4. Optionally change zUnit test case for better Code Coverage…

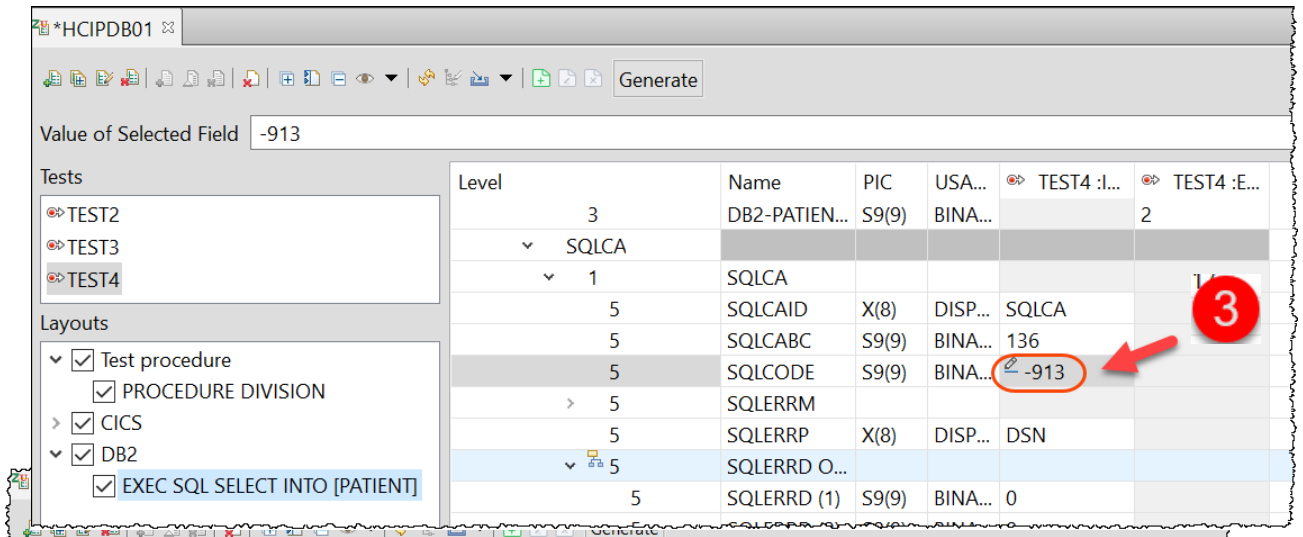


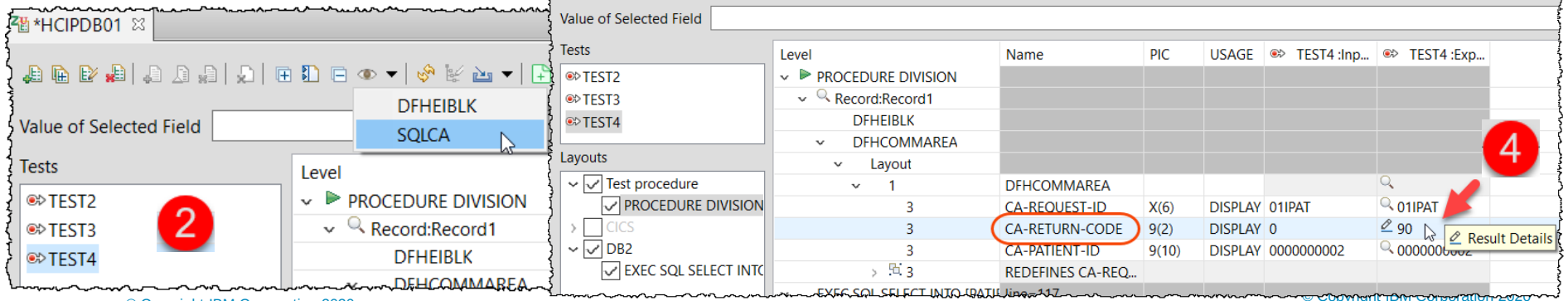
Could find “bugs” not ever tested before…
(shift to the left)

This condition is not tested

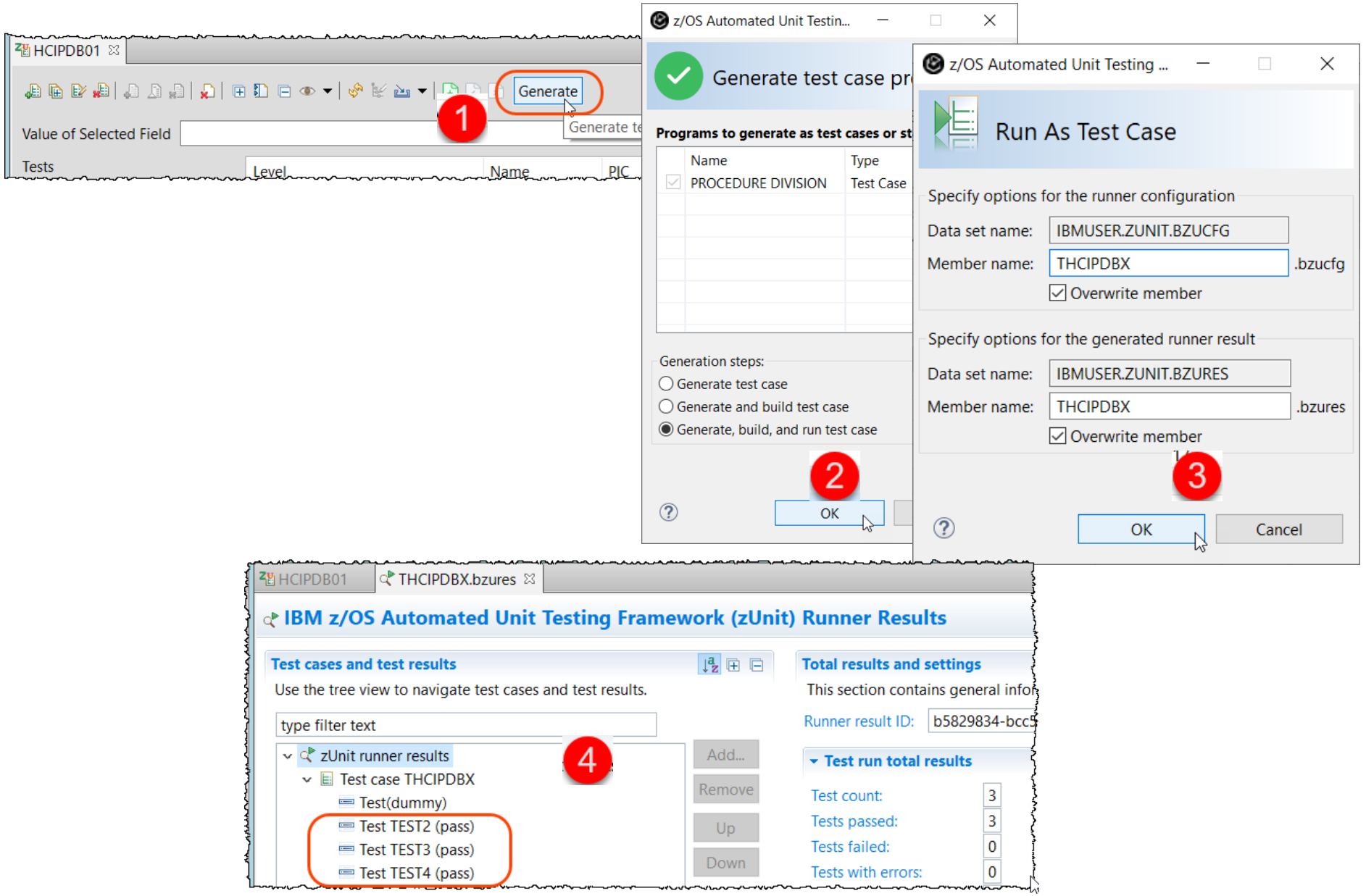
SQLCODE -913 must have **90** at **CA-RETURN-CODE**



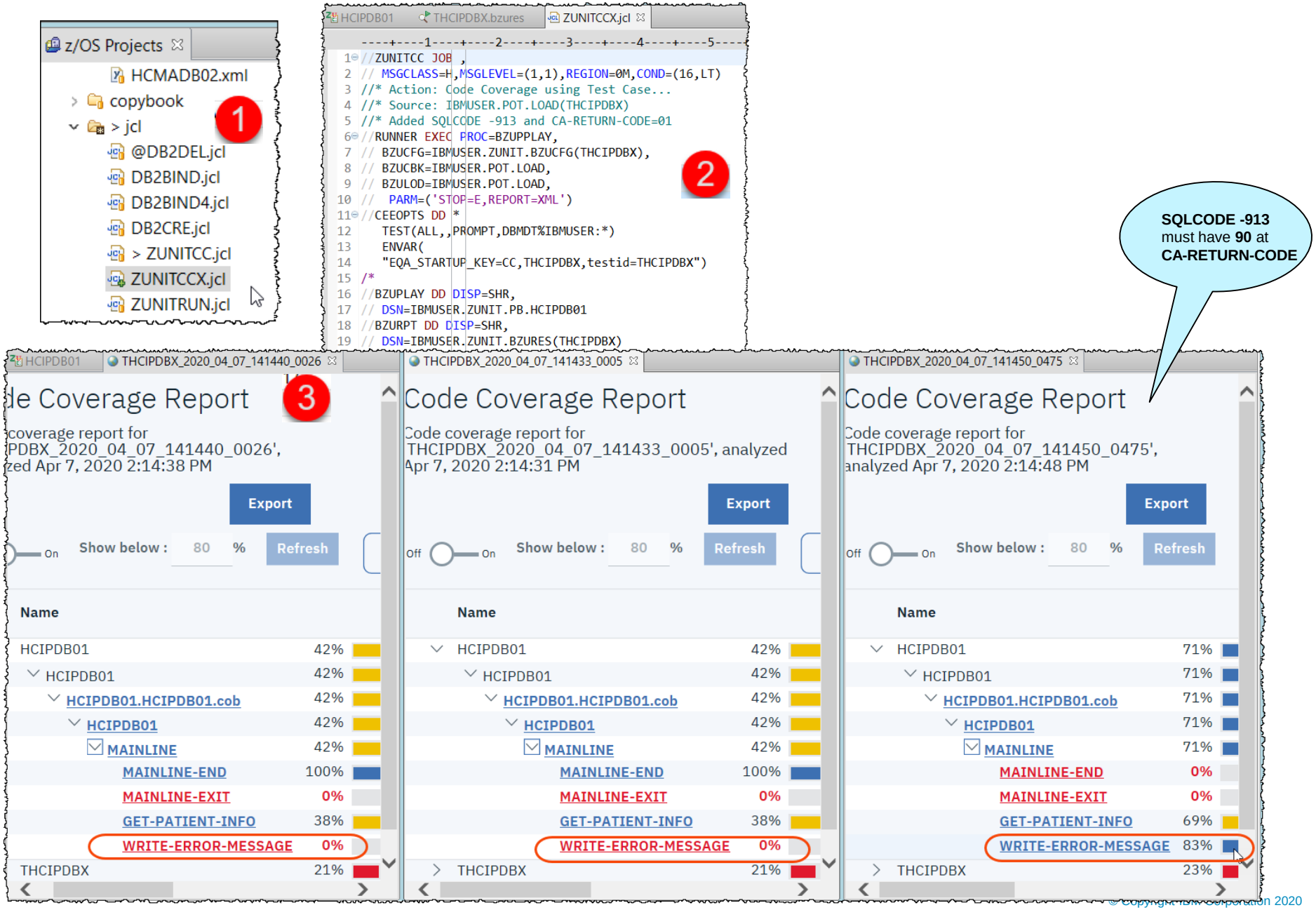




4. Optionally change zUnit test case for better Code Coverage…



4. Optionally change zUnit test case for better Code Coverage

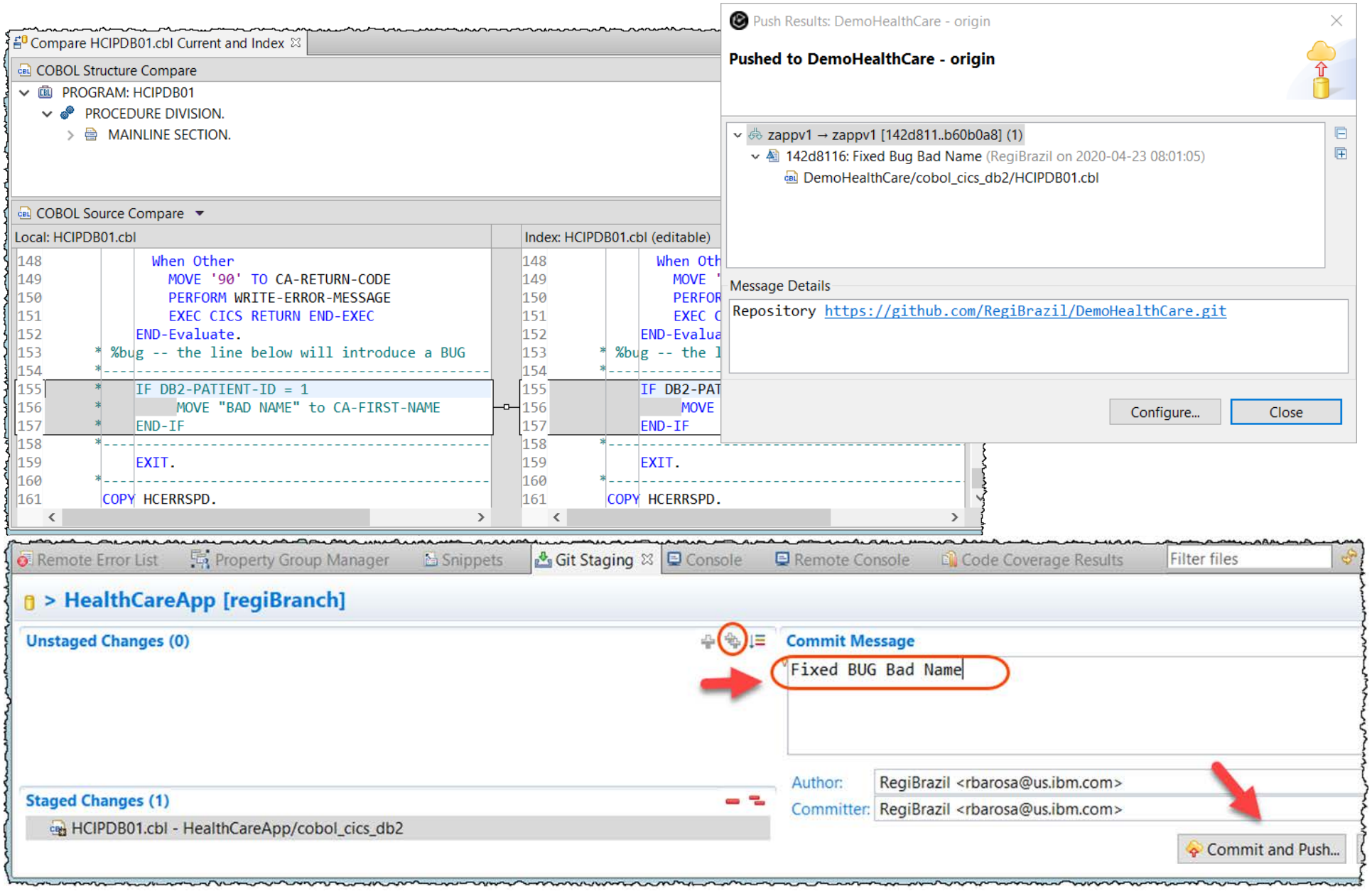


PART #2 – Fix COBOL/CICS HCIPDB01 broken program

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

1. Developer **B** run the zUnit test case and verify the bug.
   Optionally he could use the Debug (in BATCH) to see the “bug
2. Uses **IDz** to update the COBOL program
3. Uses **DBB** to perform a “User Dependency Build”, run the Unit Test (**zUnit**), **Code Coverage** and verify that the abend is fixed
4. Optionally change zUnit test case for better Code Coverage
5. **Commit and Push** the corrected code to **Git** (*GitHub*)

5. Commit and Push the corrected code to Git (GitHub)

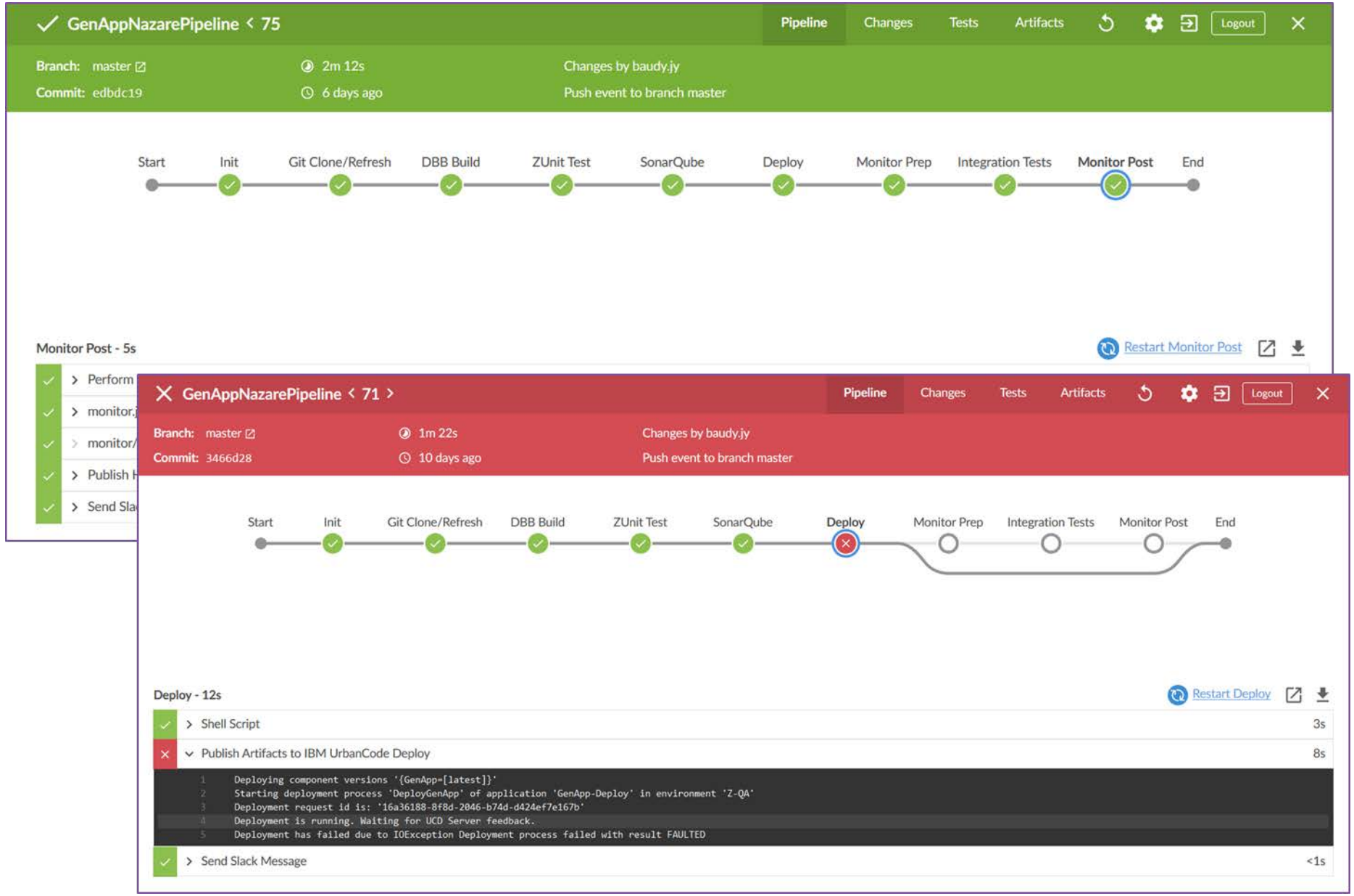


PART #3 - Release Engineer Deploy to z/OS and Mobile.

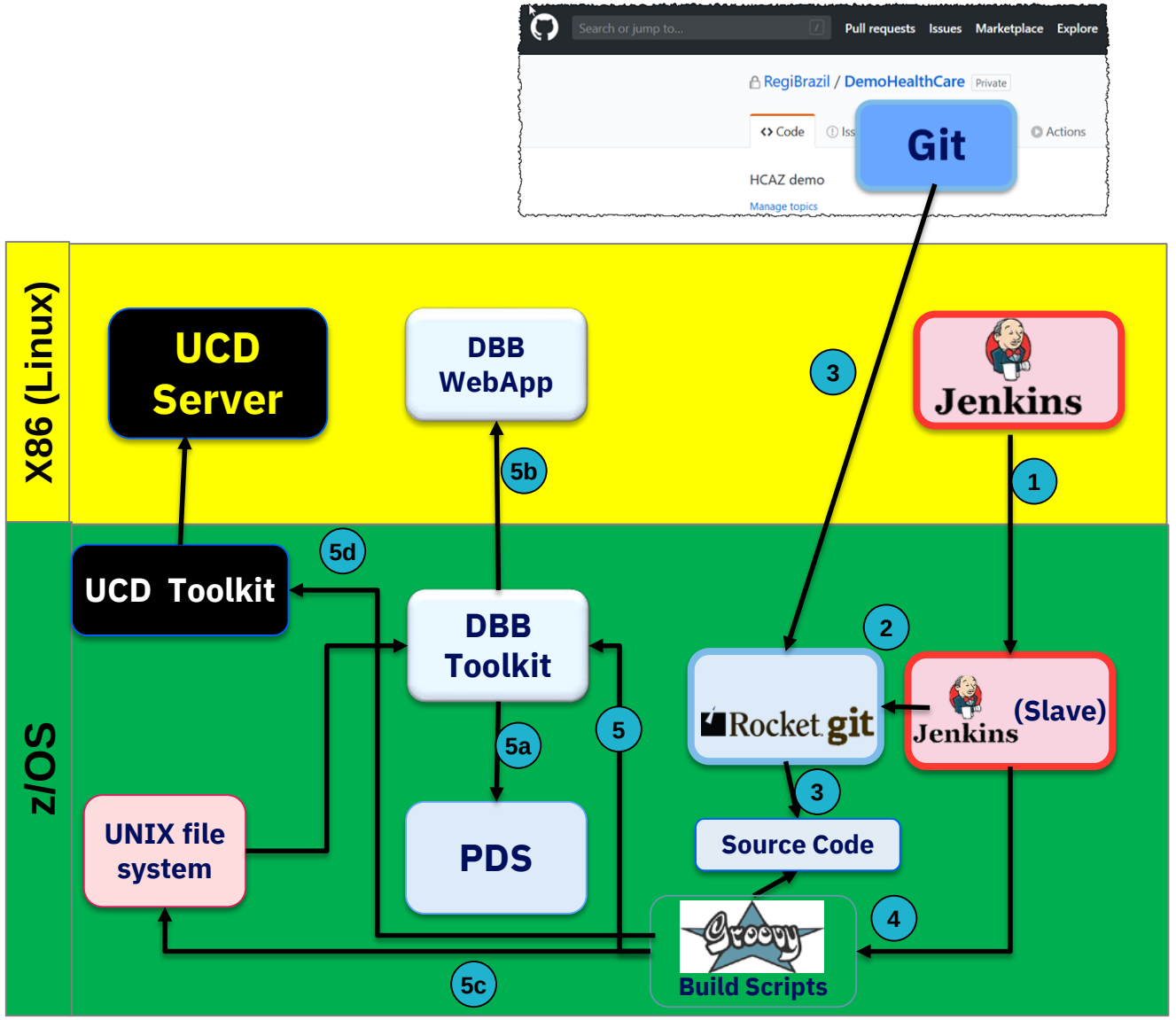
Rebecca – Release Engineer
Executes the Team Building using Jenkins

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

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.

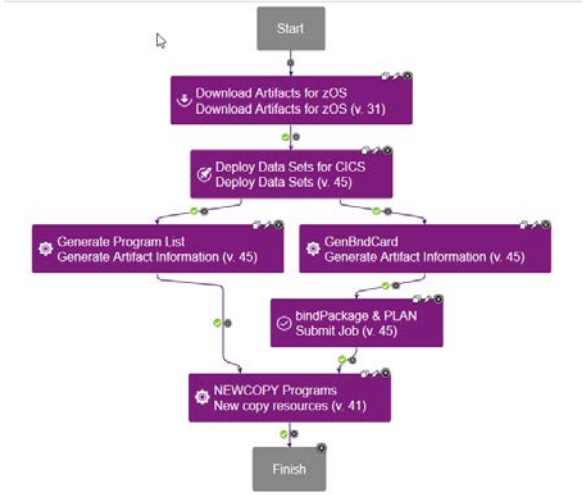
[5b] **DBB** scan and store dependency data from source files, perform dependency and impact analysis, store build results on DBB WebApp .

[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



Develop

SCM

Build

IDE
or
TSO/ISPF

CA Endevor

Serena Changeman

git

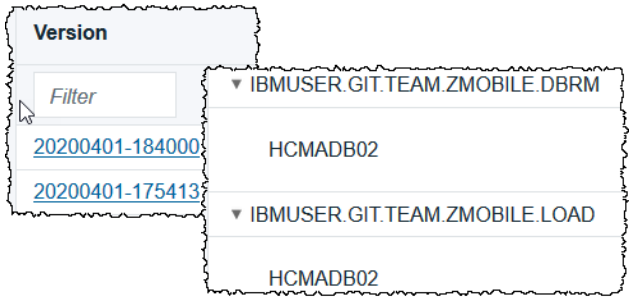
RTC

EE

From Source Code,
Using JCL:
Compile, Link and **Bind**

IBM UrbanCode Deploy

LOADLIB + DBRM



Dev

UCD Agent

z/OS LPAR

TEST

UCD Agent

z/OS LPAR

PROD

UCD Agent

z/OS LPAR

Mainframe

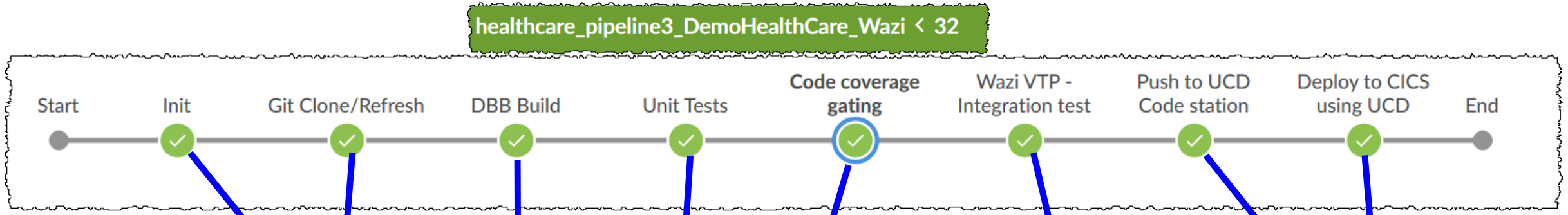
* RTC → IBM® Engineering Workflow Management

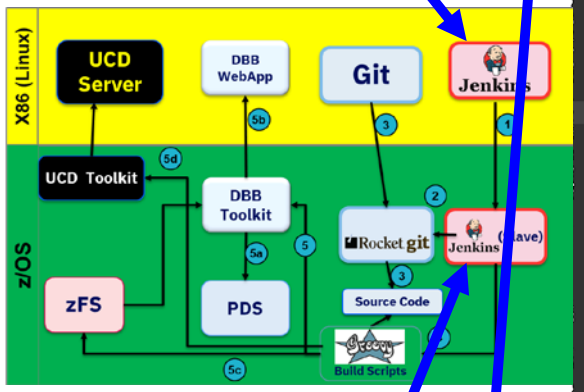
PART #3 - Release Engineer Deploy to z/OS and Mobile.

Rebecca – Release Engineer
Executes the Team Building using Jenkins

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

Demo High Level Diagram using Jenkins-Git-DBB-UCD





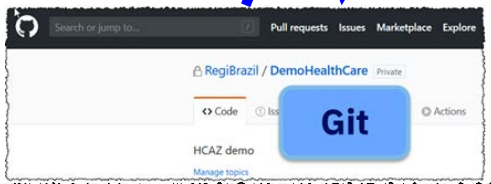

```
** DBB build group name: DemoHealthCare-sandbox
** DBB collection name: DemoHealthCare-sandbox
** Repository client created for https://10.1.1.1:1043/dbb
** Build output located at /var/jenkins/workspace/healthcare_pi
** Build result created for BuildGroup:DemoHealthCare-sandbox
/buildResult/7783
** --impactBuild option selected. Building impacted programs fo
** Writing build list file to /var/jenkins/workspace/healthcare
** Invoking build scripts according to build order : BMS.groovy,
** Building files mapped to BMS.groovy script
** Building files mapped to Cobol.groovy script
*** Building file DemoHealthCare/cobol_cics_db2/HCIPDB01.cbl
** Building files mapped to LinkEdit.groovy script
** Writing build report data to /var/jenkins/workspace/healthcare_pipeline2/BUILD-24/BuildReport.json
** Writing build report to /var/jenkins/workspace/healthcare_pipeline2/BUILD-24/BuildReport.html
** Build ended at Fri Jul 10 20:51:38 GMT 2020
** Build State : CLEAN
** Total files processed : 1
** Total build time  : 4 minutes, 33.933 seconds

** Build finished
```

```
/Appliance/IDz14.2.1/headless-cc/ccresults.sh /v
42 + ccpercent — Print Message
1  42 + ccpercent
Archive the artifacts
```

```
PackageManifest file post-processing completed.
Create version and store package:
....Version artifacts stored to UCD server CodeStation
....Version:20200828-150944 created
Elapsed time 68.0 seconds.
```

```
BZUP001I EXECUTING RUNNER PROGRAM=BZURUN AT 2020/08/10 11:33:47
POWERED BY IBM WAZI VIRTUAL TEST PLATFORM - V1R1M0
COPYRIGHT(C) 2020, IBM CORP. ALL RIGHTS RESERVED.
********************************************************************************
BZUP002I FINISHED EXECUTION RC=00
********************************************************************************
```




```
Checking out Revision e375bc120265eafdc47511e9d65b17780124c322 (origin/sandbox)
 > git rev-parse origin/sandbox^{commit} # timeout=10
 > git config core.sparsecheckout # timeout=10
 > git config core.sparsecheckout true # timeout=10
 > git read-tree -mu HEAD # timeout=10
 > git checkout -f e375bc120265eafdc47511e9d65b17780124c322 # timeout=10
Commit message: "Fixed Bug"
 > git rev-list --no-walk 40455f54e22d7d8b05d951659de3ddeab6d67605 # timeout=10
```

```
The JCL Job RUNZUNIT(JOB00179) completed successfully
********** Module [THCIPDB0] **********
Name:      THCIPDB0
Status:    PASS
Test cases: 2 (2 passed, null failed, 0 errors)
Details:
    TEST2
    TEST3
********** Module [THCIPDB0] **********
** Build finished
```

HC_CICS

Deploy HC to CICS (HC_CICS 20200714-222639)

1. Download Artifacts for zOS
2. Deploy Data Sets for CICS
3. GenBndCard
4. Generate Program List
5. bindPackage & PLAN

* UCD = UrbanCode Deploy

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

Rebecca – Release Engineer
Executes the Team Building using Jenkins

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

Verify the Build results using **DBB browser**

IBM

Collections **Build Results**

- DemoHealthCare
- DemoHealthCare-zappv1
  - build.20200410.124708.047
  - build.20200410.010913.009
  - build.20200410.011634.016
  - build.20200410.012517.025
  - build.20200410.013510.035
  - build.20200410.015112.051
  - build.20200410.022038.020
  - build.20200410.023546.035
  - build.20200413.013548.035
  - build.20200413.024708.047
  - build.20200413.035658.056
  - build.20200413.063559.035
  - build.20200413.065610.056
  - build.20200413.091435.014
  - build.20200414.025702.057
  - build.20200414.032928.029
  - build.20200414.041341.013
  - build.20200414.042941.029
  - build.20200414.075518.055
  - build.20200414.080313.003
  - build.20200414.043421.034
  - build.20200414.093859.038
  - build.20200415.110739.007

IBM

Collections **Build Results**

DBB Build Result

Field	Value
id	4438
group	DemoHealthCare-zappv1
label	
Build Report	
buildReportData	
state	
status	
owner	
permission	
created	
createdBy	
lastUpdated	

Build Report

Toolkit Version:

Version:	1.0.6
Build:	11
Date:	19-Aug-2019 17:09:57

Build Summary

Number of files being built: 1

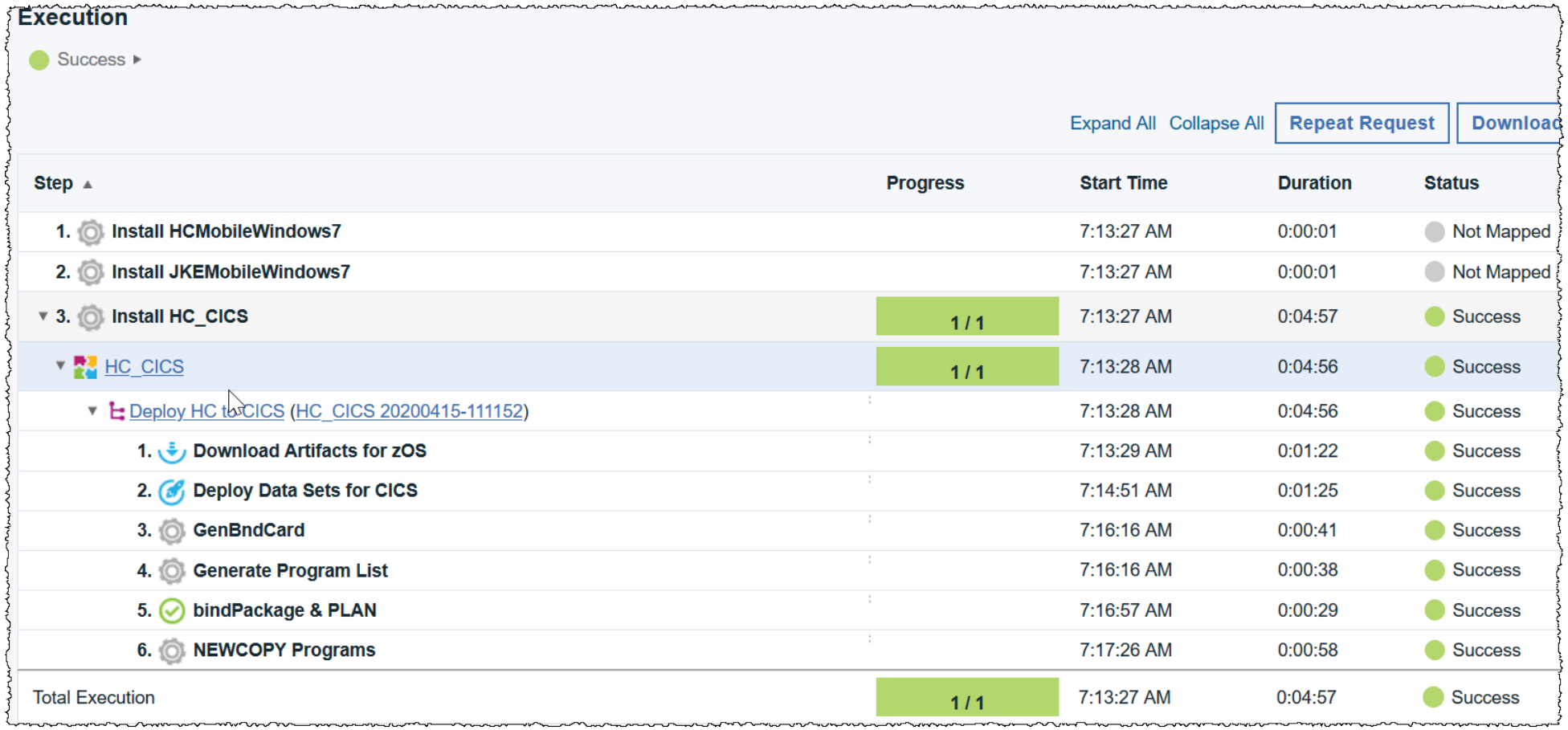
	File	Commands	RC	Data Sets	Outputs	Deploy Type
1	DemoHealthCare/cobol_cics_db2/HCIPDB01.cbl • DemoHealthCare/copybook/HCERRSPD.cpy COPY • DemoHealthCare/copybook/HCERRSWS.cpy COPY • DemoHealthCare/copybook/HCCMAREA.cpy SQL INCLUDE • DemoHealthCare/copybook/SQLCA.cpy SQL INCLUDE Hide Dependencies	IGYCRCTL	4	IBMUSER.GIT.TEAM.ZMOBILE.COBOL(HCIPDB01)	IBMUSER.GIT.TEAM.ZMOBILE.OBJ(HCIPDB01) IBMUSER.GIT.TEAM.ZMOBILE.DBRM(HCIPDB01)	DBRM
		IEWBLINK	0		IBMUSER.GIT.TEAM.ZMOBILE.LOAD(HCIPDB01)	LOAD

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

Rebecca – Release Engineer
Executes the Team Building using Jenkins

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

Verify the Deploy results using **UCD** browser



Execution

Success ▸

Expand All Collapse All **Repeat Request** **Download**

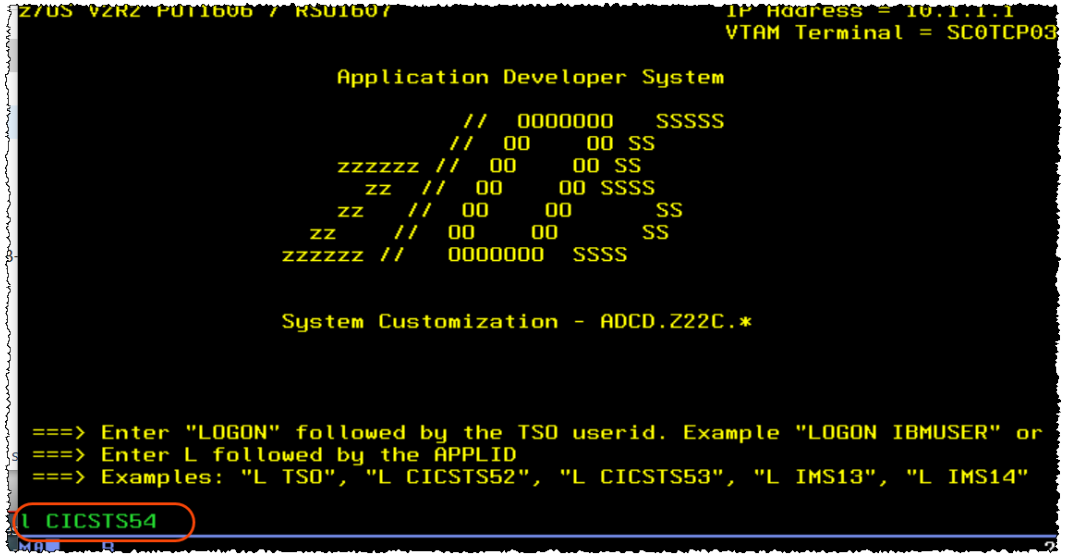
Step ▲	Progress	Start Time	Duration	Status
1. Install HCMobileWindows7		7:13:27 AM	0:00:01	Not Mapped
2. Install JKEMobileWindows7		7:13:27 AM	0:00:01	Not Mapped
▾ 3. Install HC_CICS	1 / 1	7:13:27 AM	0:04:57	Success
▾ HC_CICS	1 / 1	7:13:28 AM	0:04:56	Success
▾ Deploy HC to CICS (HC_CICS 20200415-111152)		7:13:28 AM	0:04:56	Success
1. Download Artifacts for zOS		7:13:29 AM	0:01:22	Success
2. Deploy Data Sets for CICS		7:14:51 AM	0:01:25	Success
3. GenBndCard		7:16:16 AM	0:00:41	Success
4. Generate Program List		7:16:16 AM	0:00:38	Success
5. bindPackage & PLAN		7:16:57 AM	0:00:29	Success
6. NEWCOPY Programs		7:17:26 AM	0:00:58	Success
Total Execution	1 / 1	7:13:27 AM	0:04:57	Success

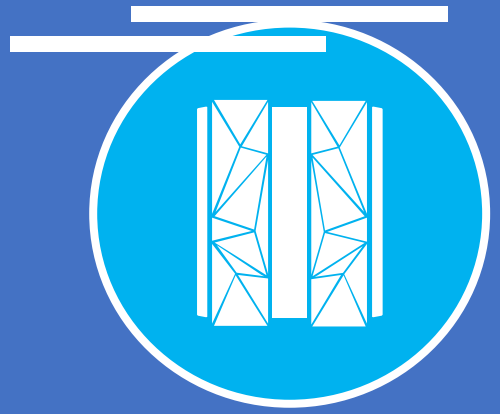
PART #3 - Release Engineer Deploy to z/OS and Mobile.

Rebecca – Release Engineer
Executes the Team Building using Jenkins

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

Run the new deployed CICS application





DevOps Mainframe Tooling

Demonstration

CICS/DB2 COBOL Program

Part 3 - Deploy using /Jenkins/UCD

Demonstration

Agenda - Day 1

10:00 Introduction to DevOps on IBM Z - focus on open source tools (Ron)

10:40 LAB 1 - Working with z/OS using COBOL and DB2 (Tim)

or

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

12:50 Checkpoint - Comments and Closing Day 1 (Tim -Ron/Wilbert/Regi)

Agenda - Day 2

10:00 zUnit and Wazi VTP Overview (Wilbert)

10:15 Demo: Scenario using Z DevOps solutions including **zUnit**, **Git** and **Jenkins** (Regi)

11:00 – 12:50 Choose one Optional lab: (Tim)

- → LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)
- → LAB 6B : Using IBM zUnit to Unit Test a COBOL CICS application (1 hour)
- → 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)

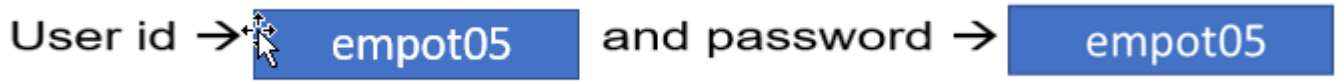
12:50 Checkpoint - Comments and Closing Day 2 (Tim -Ron/Wilbert/Regi)

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 **JxxP** to become familiar with the Application that you intend to modify.

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

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

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

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

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

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

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

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

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

.

Overview of development tasks

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

1. **Get familiar with the application using the 3270 terminal**
   → You will start a 3270 emulation and execute a transaction named **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.

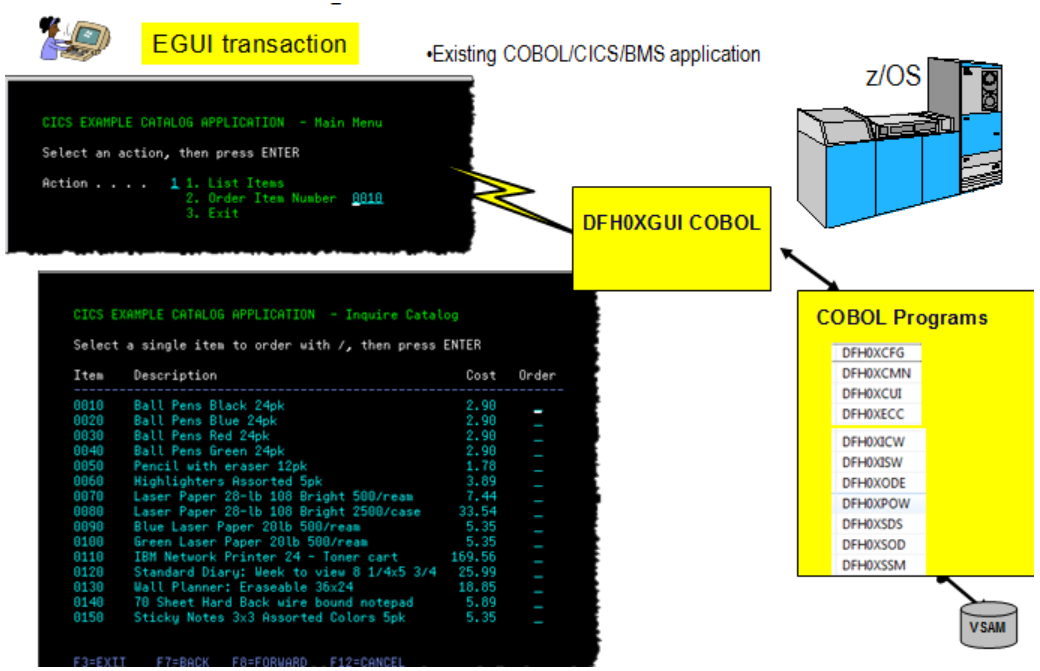
4. **Generate, build and run the unit test**
   → You will compile and link-edit the generated unit test program, followed by running the unit test.

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

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 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.

IBM Z Trial Program

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

No charge environment

No set up, no install

Hands-on tutorials

Available now:

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

Coming soon:

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

Register now at ibm.biz/z-trial

No-Charge IDz/ADFz Training offerings

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

Home › IDz/RDz Remote Training

IDz/RDz Remote Training

Entry-Level Training - Module Content Details

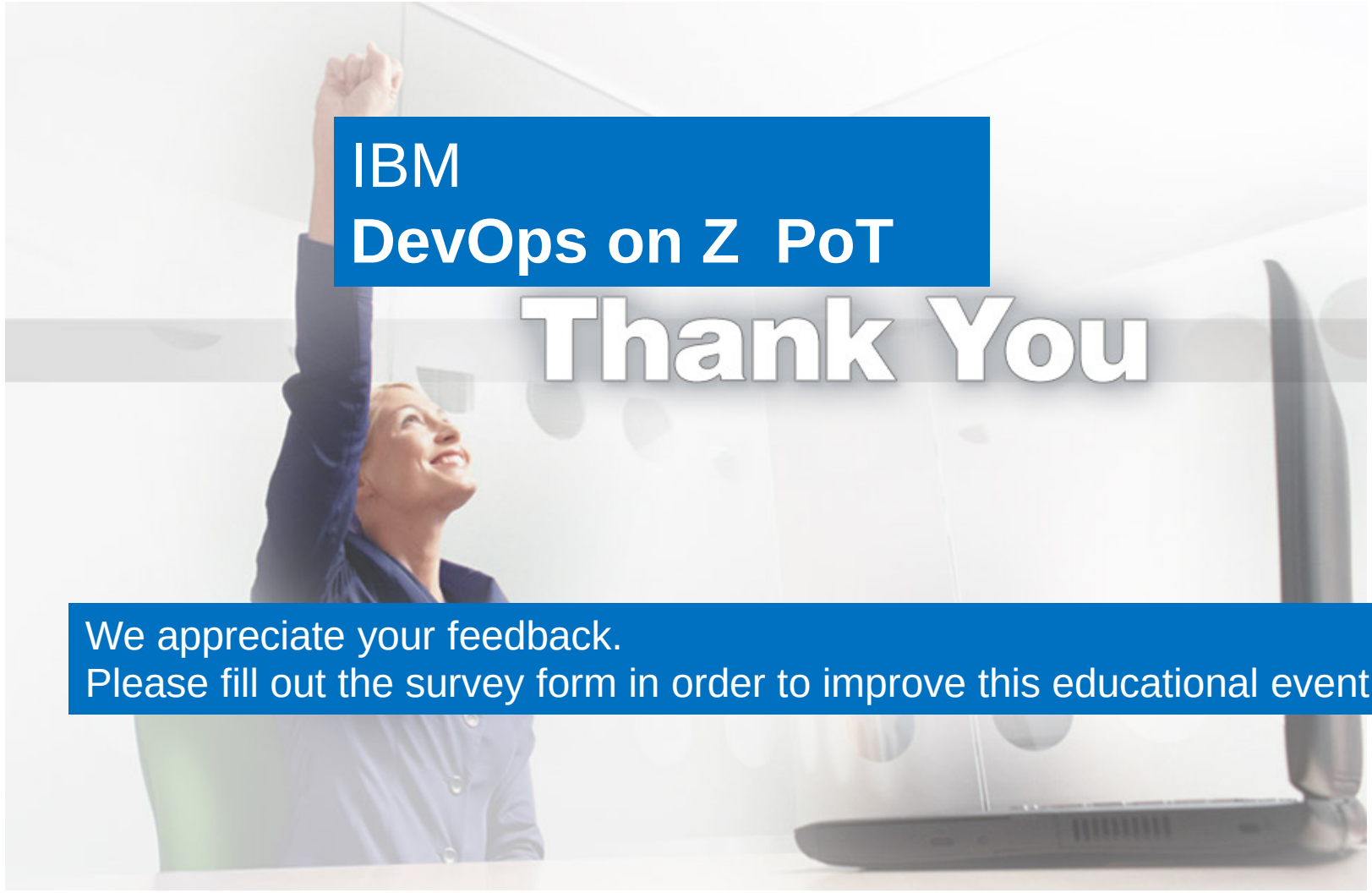
+ Module 1: The RDz Workbench and introduction to Eclipse for ISPF Developers
+ Module 2: Editing Program Source
+ Module 3: Analyzing COBOL Programs
+ Module 4: Introduction to RDz Remote Systems' Features
+ Module 5: Dataset Access and Organization
+ Module 6: ISPF 3.x Options, Batch Job Submission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

No-Charge IDz/ADFz Training offerings for September

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

September 14th - 9:00 AM - 10:00 AM Eastern	**Training Module 9 - Debug and Code Coverage **Note September 14th date**** Batch & Online debugging features and techniques, including JCL to launch Debug, program animation features, break points, variable monitors/changing variable values dynamically, Record & Playback. An introduction to IDz Code Coverage - both for batch and online (CICS) test-quality analysis.

Questions / Comments


IBM
DevOps on Z PoT
Thank You
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

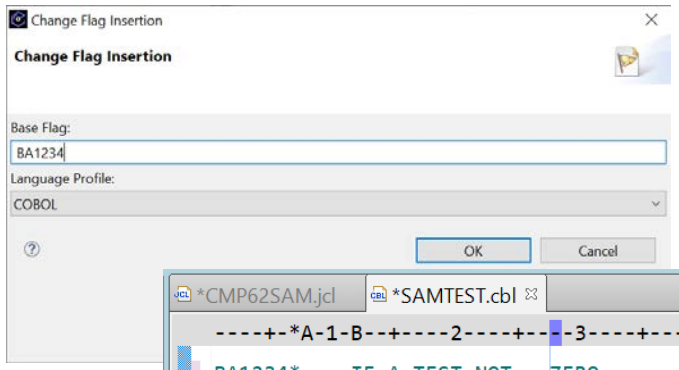
- Git is a fast and modern implementation of version control
  - Very lightweight
  - Extremely fast - The tool just gets out of the way
- Git is Distributed
  - Every developer has full copy, with history
  - No single point of failure
- Git facilitates collaborative changes to files
  - Enables branch-based 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.

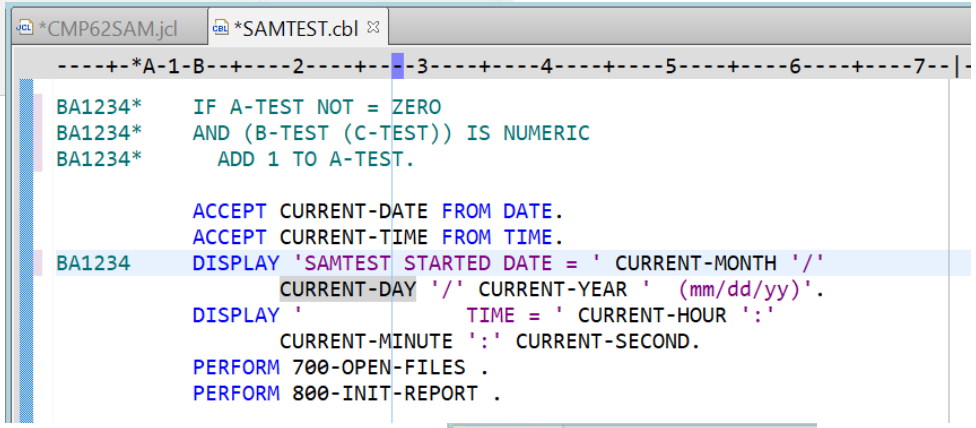
- Git is easy to use for any type of knowledge worker
  - Native environment is a Command Line terminal
  - There are UI extensions that can provide a pleasant user experience for version control.
  - Ex:
    - IBM Developer for z/OS (IDz) - eGit Perspective
    - Visual Studio Code (VSCode)

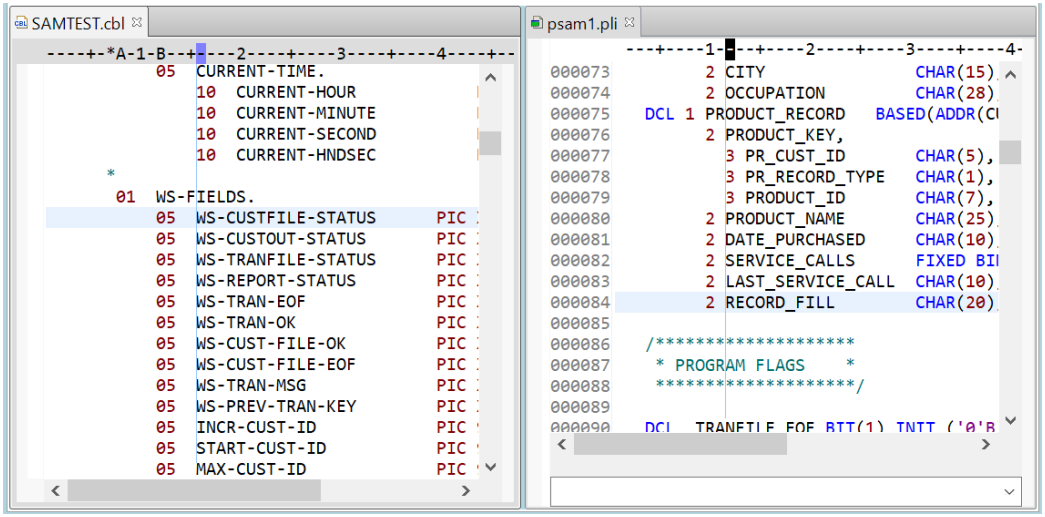
New editing capabilities

Analyze and Edit

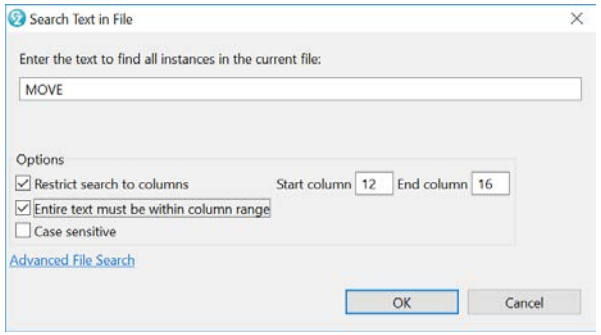


New! COBOL and PL/I editors now support Autocomment (14.2)

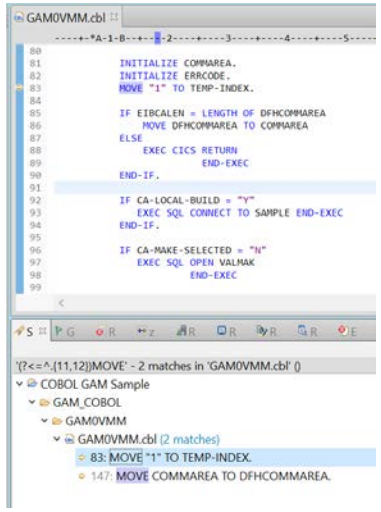


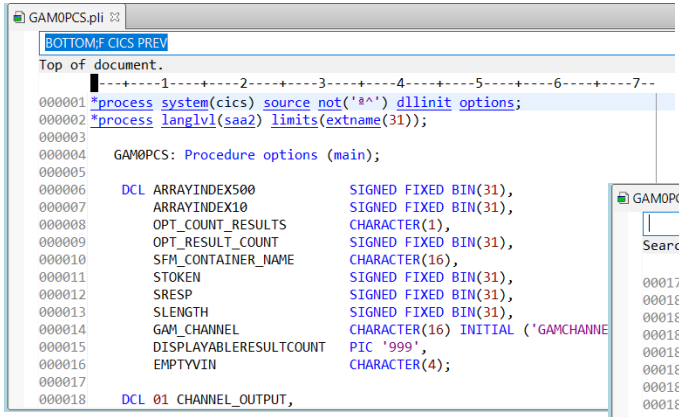


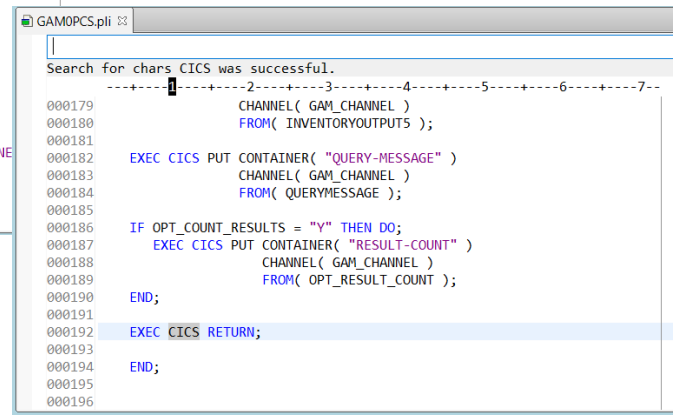
New! COBOL and PL/I editors offer a shortcut to toggle a vertical bar on/off in the editor (14.2)



New! Search file Text- specify columns







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

Building vs. Compile/Link