

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)

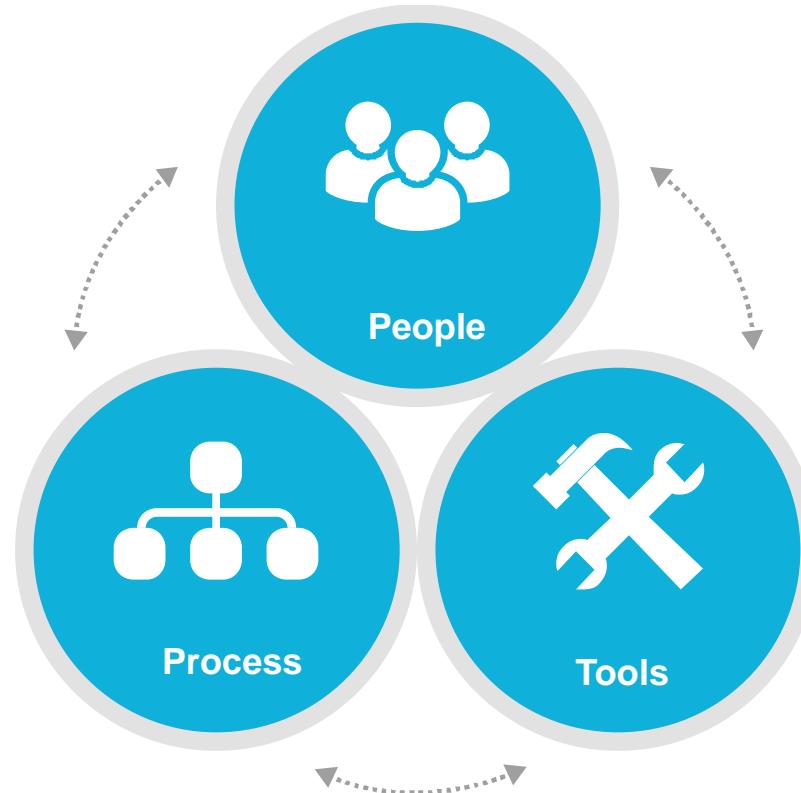
11:00 – 12:50 Choose one Optional lab: (Tim)

- 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)

DevOps is not one of
these things...

It's all of them!



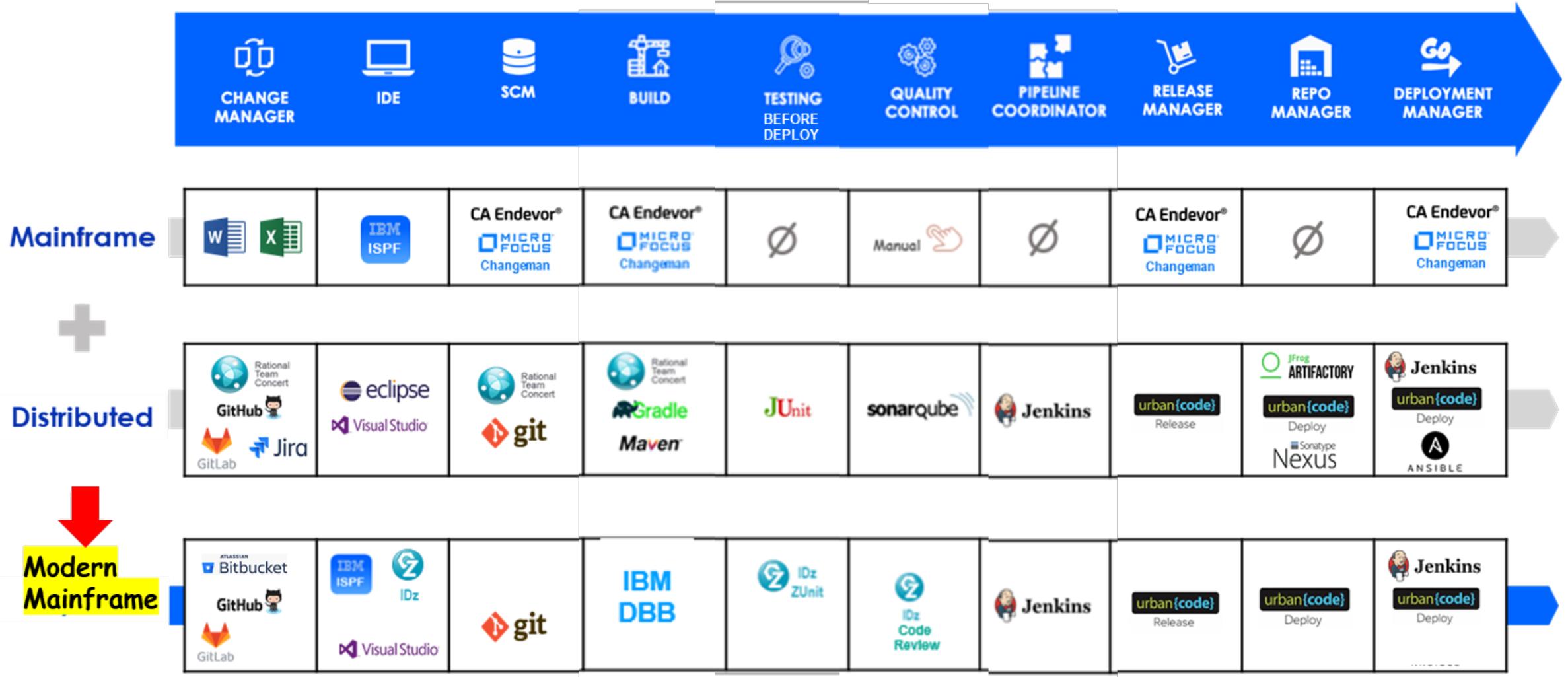
DevOps is a set of practices that combines software development (Dev) and IT operations (Ops).
It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.



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.

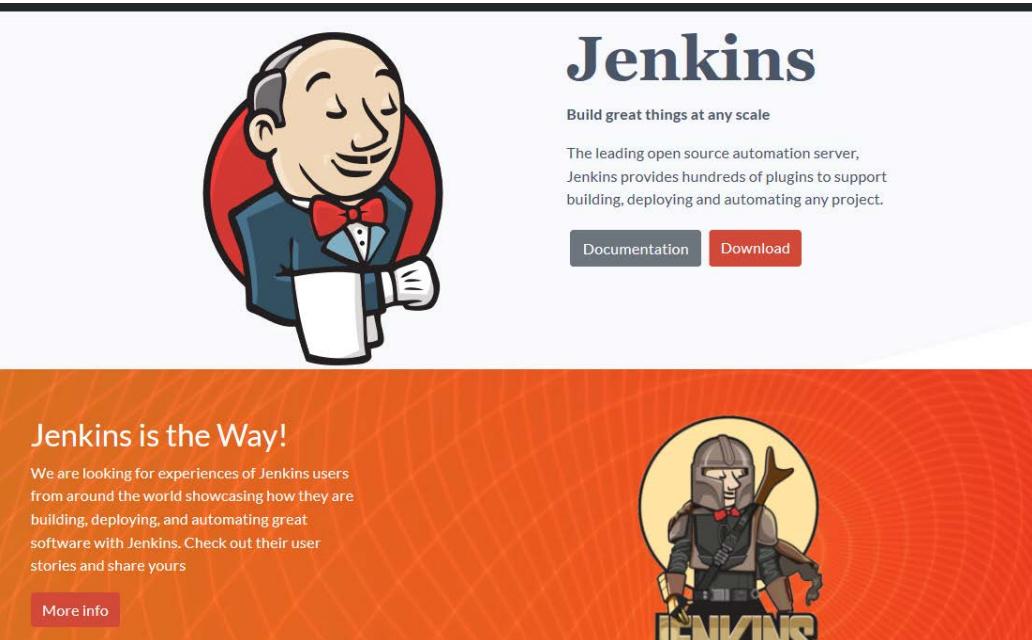


What is Jenkins?

“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.”

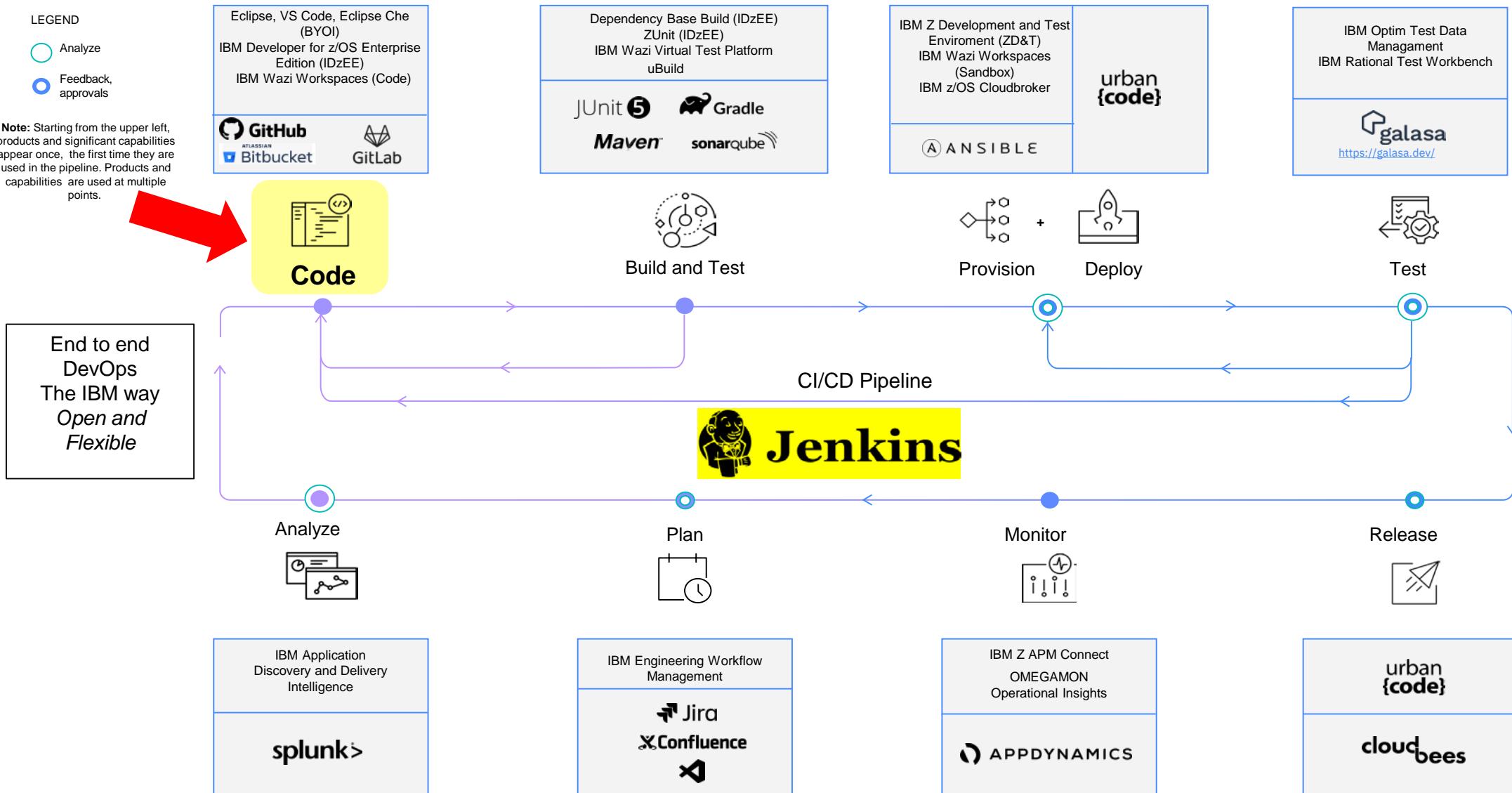
Ref: <https://www.jenkins.io/doc/>

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



The screenshot shows the Jenkins homepage. At the top left is a cartoon illustration of a man in a tuxedo and bow tie, holding a white scroll. To his right, the word "Jenkins" is written in a large, bold, blue sans-serif font. Below it, the tagline "Build great things at any scale" is displayed in a smaller, gray font. A brief description follows: "The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project." Two buttons are present: "Documentation" (gray) and "Download" (red). The bottom section has a red background with the slogan "Jenkins is the Way!" in white. It encourages users to share their experiences and features a cartoon illustration of a viking warrior. A "More info" button is located at the bottom left.

Z DevOps Pipeline

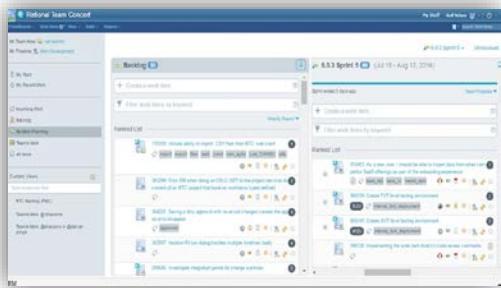


z/OS and subsystems

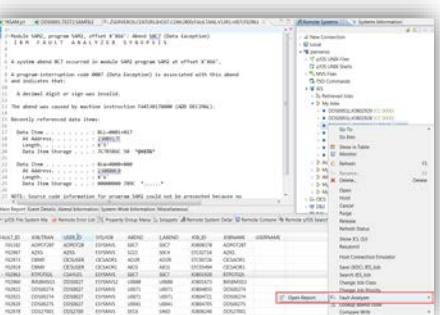


Build and SCM

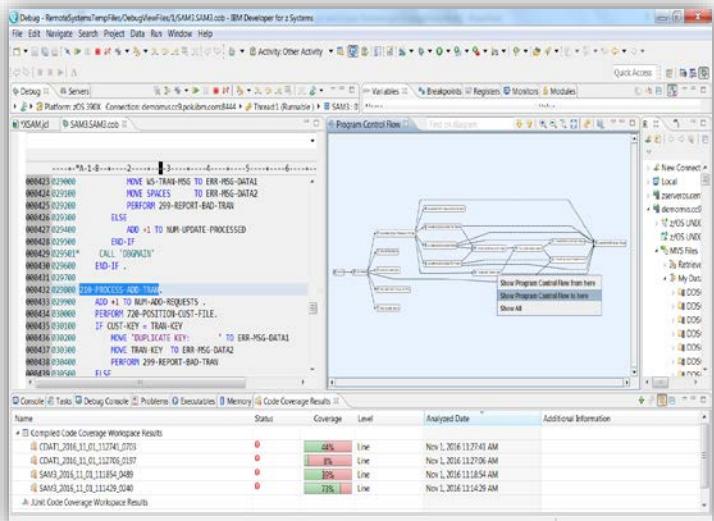
Integration with Git for Lifecycle and Source Management



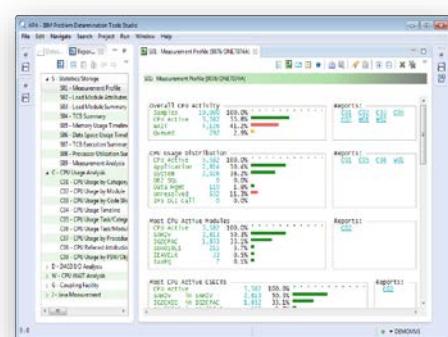
Abend analysis



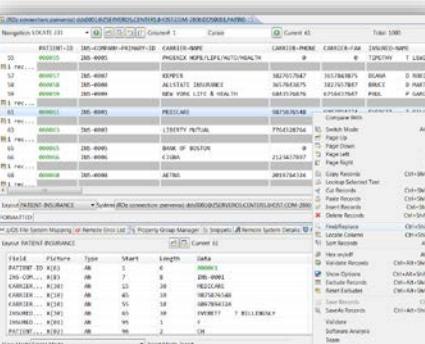
IBM Developer for z/OS Enterprise Edition- the developer's cockpit!



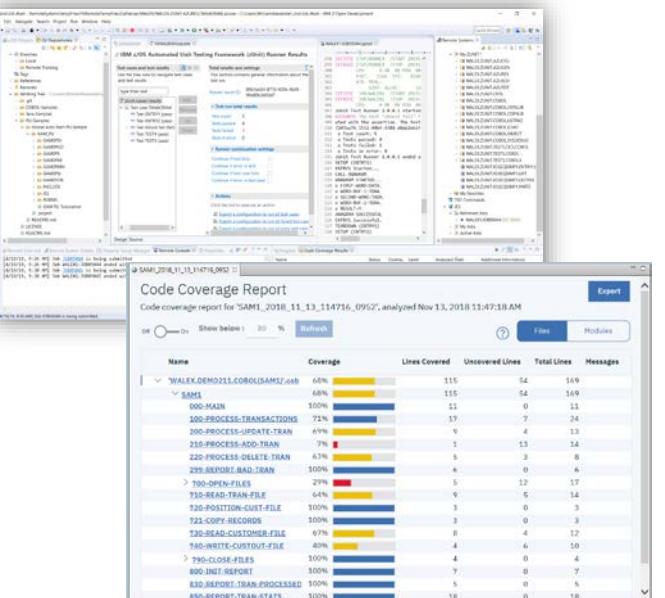
Performance analysis



File Management



Unit test and code coverage



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

The screenshot displays the Rational Application Developer (RAD) interface with several windows open, illustrating its analysis capabilities:

- Outline view:** Shows the hierarchical structure of the COBOL program, including sections like PROGRAM, IDENTIFICATION DIVISION, and DATA DIVISION.
- Program Control Flow:** A graphical view showing the flow of control between different program units (e.g., 1000-CREATE-DEALERSHIP, 2000-CREATE-MAKE-AND-MODEL).
- Data flow:** A graphical view showing the movement of data between different components or variables.
- Hover improvements:** A callout box highlights a feature where hovering over a paragraph or section in the code editor displays associated comments. This is demonstrated in the bottom right corner where a tooltip shows comments for the line "4000-CREATE-CENTRALINVENTORY".

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

The screenshot shows the SonarLint interface. On the left, a 'Properties' window displays various program metrics under 'Basic Metrics' and 'Complexity Metrics'. On the right, a code editor window shows COBOL code with several inspection results overlaid. A purple box highlights the 'Program statistics' section.

The screenshot shows the IBM Rational COBOL IDE. A context menu is open over some COBOL code, with 'Software Analyzer Configurations...' selected. To the right, a 'COBOL RULES' dialog box is open, showing a tree view of available rules categorized under 'Analysis Domains and Rules'. A purple box highlights the 'Code rules' section.

The screenshot shows the IBM Rational COBOL IDE with a COBOL program open. A blue box highlights a specific line of code labeled 'Unreachable code' with the text: '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

The screenshot displays the ADFz interface with two main windows. The top window is a 'File Manager Editor' showing a list of records for 'PATIENT-INSURANCE'. The bottom window is a 'File Editor' showing a detailed view of a single record's fields. A context menu is open over a selected row in the top window, with the 'File Manager' option highlighted and circled in orange. An arrow points from the 'File Manager' option to the bottom window's field editor. A blue box on the right contains promotional text about accessing VSAM files directly.

File Manager Editor (Top Window):

	PATIENT-ID	INS-COMPANY-PRIMARY-ID	CARRIER-NAME	CARRIER-PH
55	000055	INS-0005	PHOENIX HOME/LIFE/AUTO/HEALTH	0 type FULL
57	000057	INS-0007	KEMPER	3827657847
58	000058	INS-0008	ALLSTATE INSURANCE	3657843875
59	000059	INS-0009	NEW YORK LIFE & HEALTH	6843576876
61	000061	INS-0001	MEDICARE	9875876548
63	000063	INS-0003	LIBERTY MUTUAL	7764328764
65	000065	INS-0005	BANK OF BOSTON	0
66	000066	INS-0006	CIGNA	2123437897
68	000068	INS-0008	AETNA	2019784321

File Editor (Bottom Window):

Field	Picture	Type	Start	Length	Data
PATIENT-ID	X(6)	AN	1	6	000061
INS-COM...	X(8)	AN	7	8	INS-0001
CARRIER...	X(30)	AN	15	30	MEDICARE
CARRIER...	X(10)	AN	45	10	9875876548
CARRIER...	X(10)	AN	55	10	6097894324
INSURED...	X(30)	AN	65	30	EVERETT T B...
INSURED...	X(01)	AN	95	1	F
PATIENT...	X(02)	AN	96	2	CH

Context Menu (Top Window):

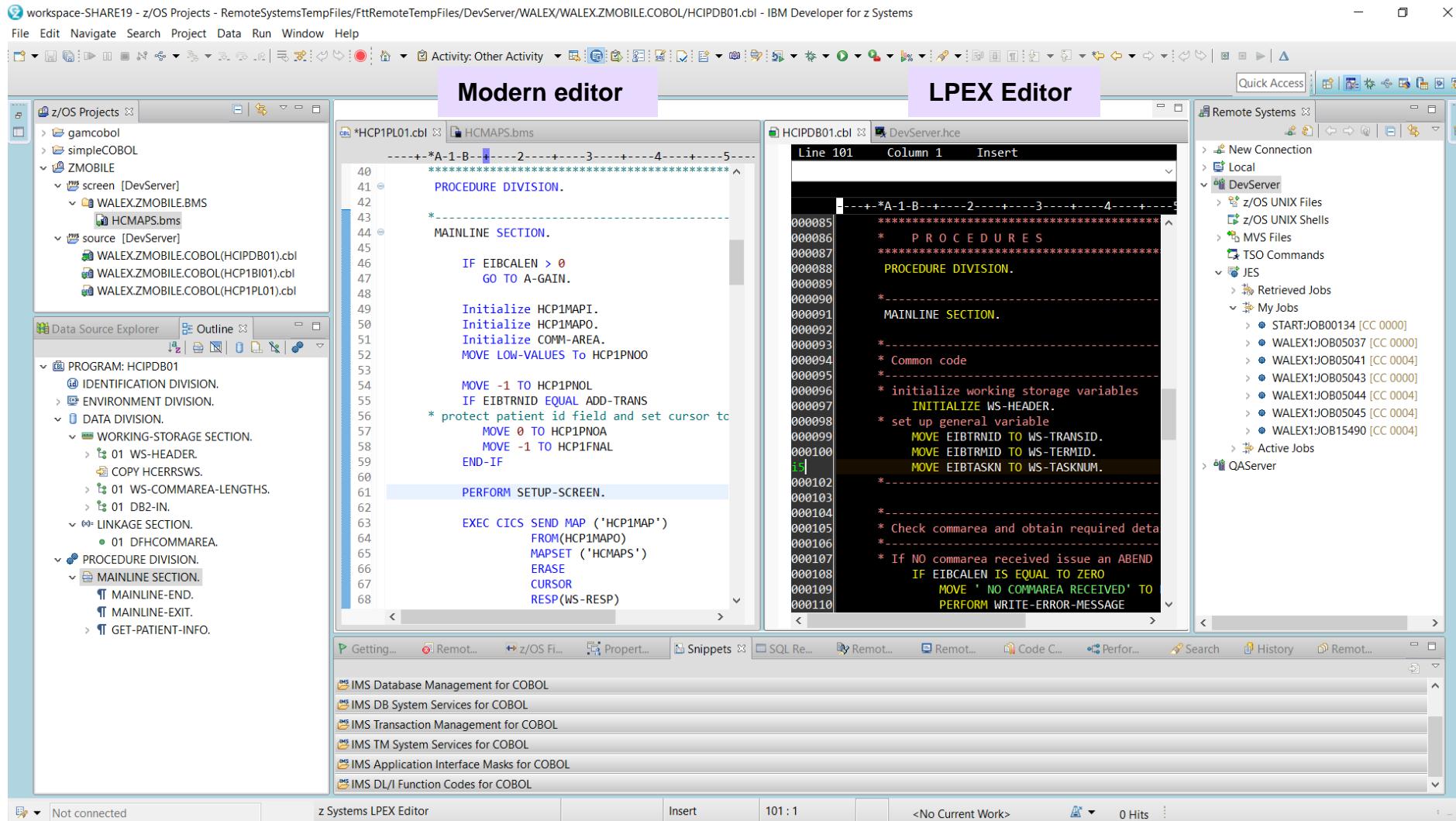
- INSURED...
- TIMOTHY
- DIANA
- BRUCE
- PAUL
- EVERETT
- LEROY
- File Manager Editor
- Template Editor
- Create Dynamic Template
- Copy Wizard
- Delete Records
- Find/Replace
- Locate Column
- Go To
- Refresh
- Rename...
- Delete...
- Profile As
- Debug As
- Run As
- Code Coverage As
- Host Connection Emulator
- Allocate Like...
- File Manager
- MVS Utilities
- TSO Commands
- Properties

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

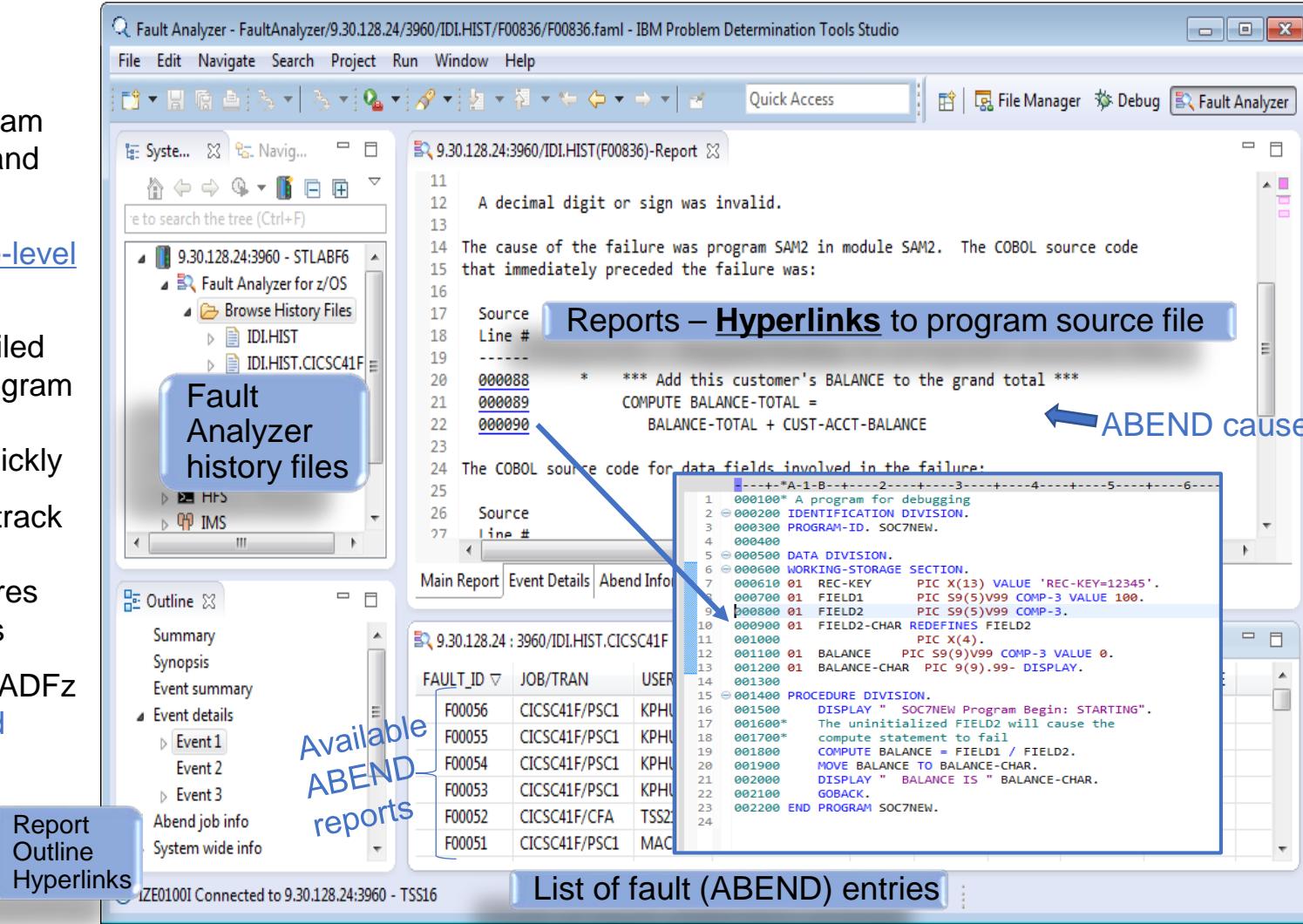
Traditional Eclipse style editors, plus ISPF-like LPEX editor with a command line and prefix area



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

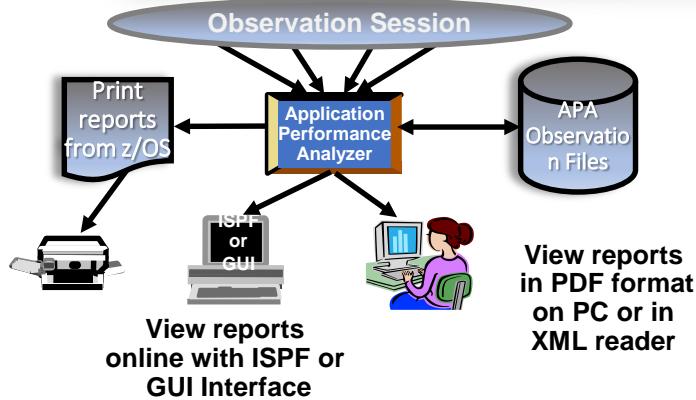
IBM Fault Analyzer improves developer productivity and decreases deployment costs by helping to analyze and correct application failures quickly (CICS,DB2,IMS,MQ,COBOL,PLI,ASM, C/C++,JAVA)

- 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



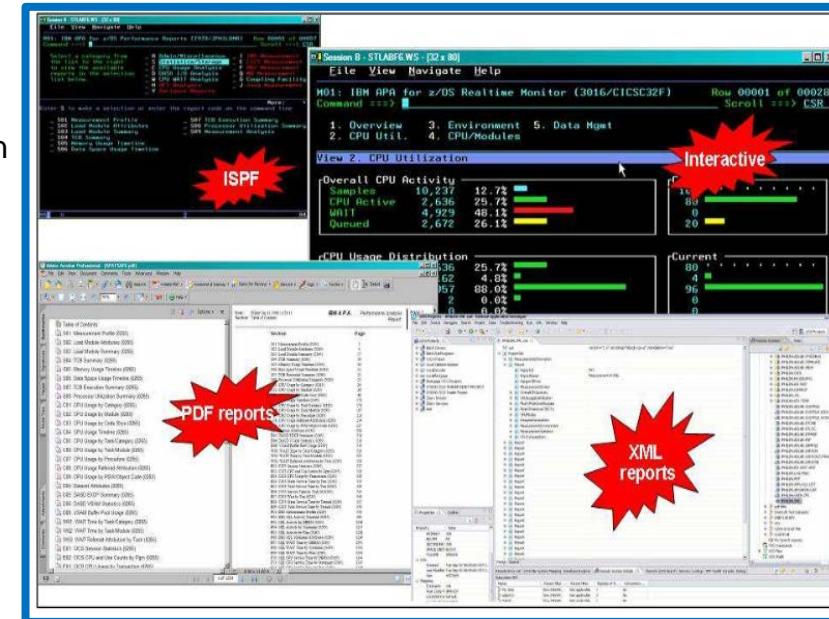
ADFz: Application Performance Analyzer (APA) for z/OS

Provides rapid pin-pointing of enterprise application bottlenecks



- 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

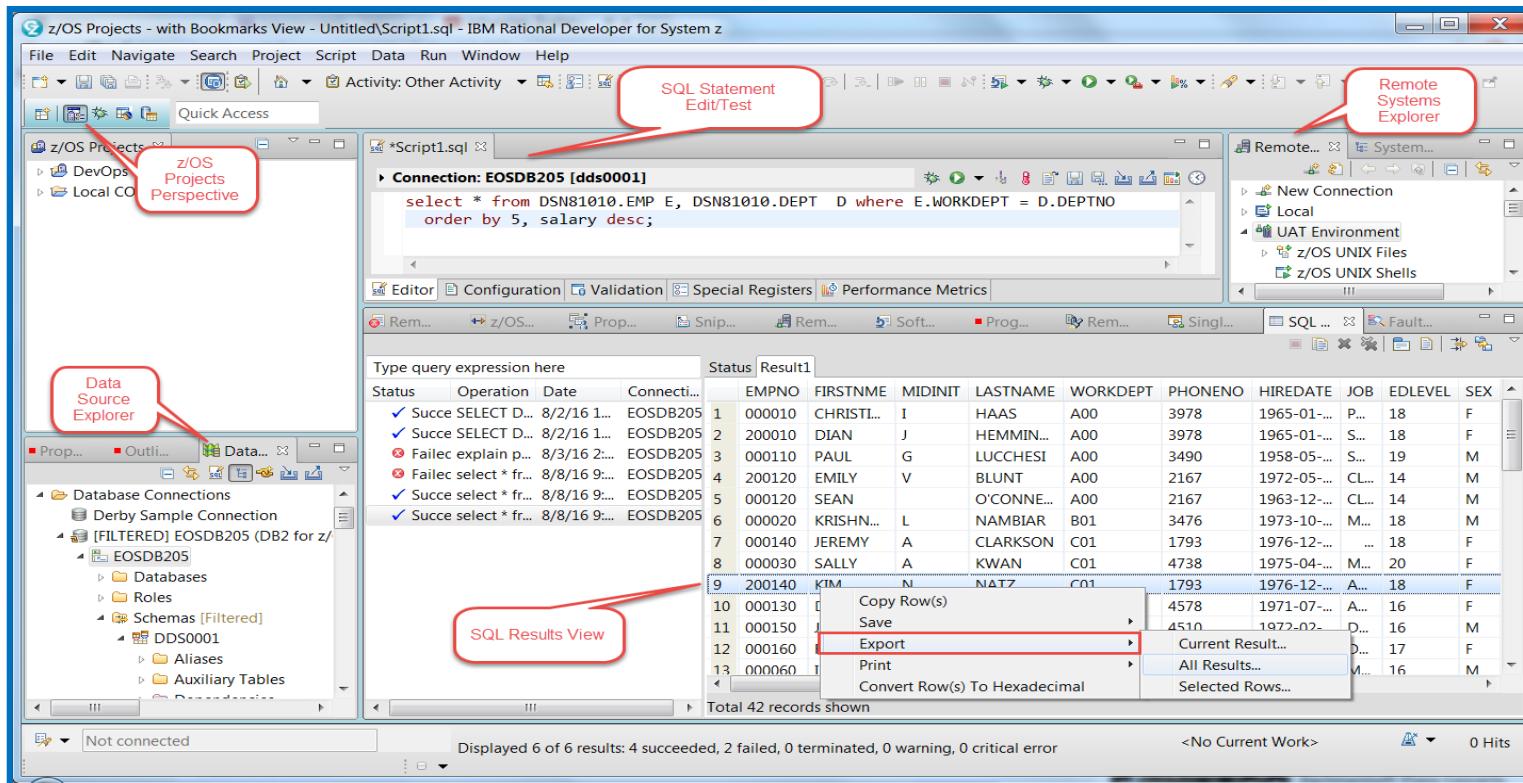


- Specify number of times APA monitors a job's performance when **job or program becomes active**
- **Invoke the monitoring capability** from other programs such as IBM Tivoli® OMEGAMON
- **Compare two observation reports** to see the relevant differences, to supplement threshold monitoring
- Assembler,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

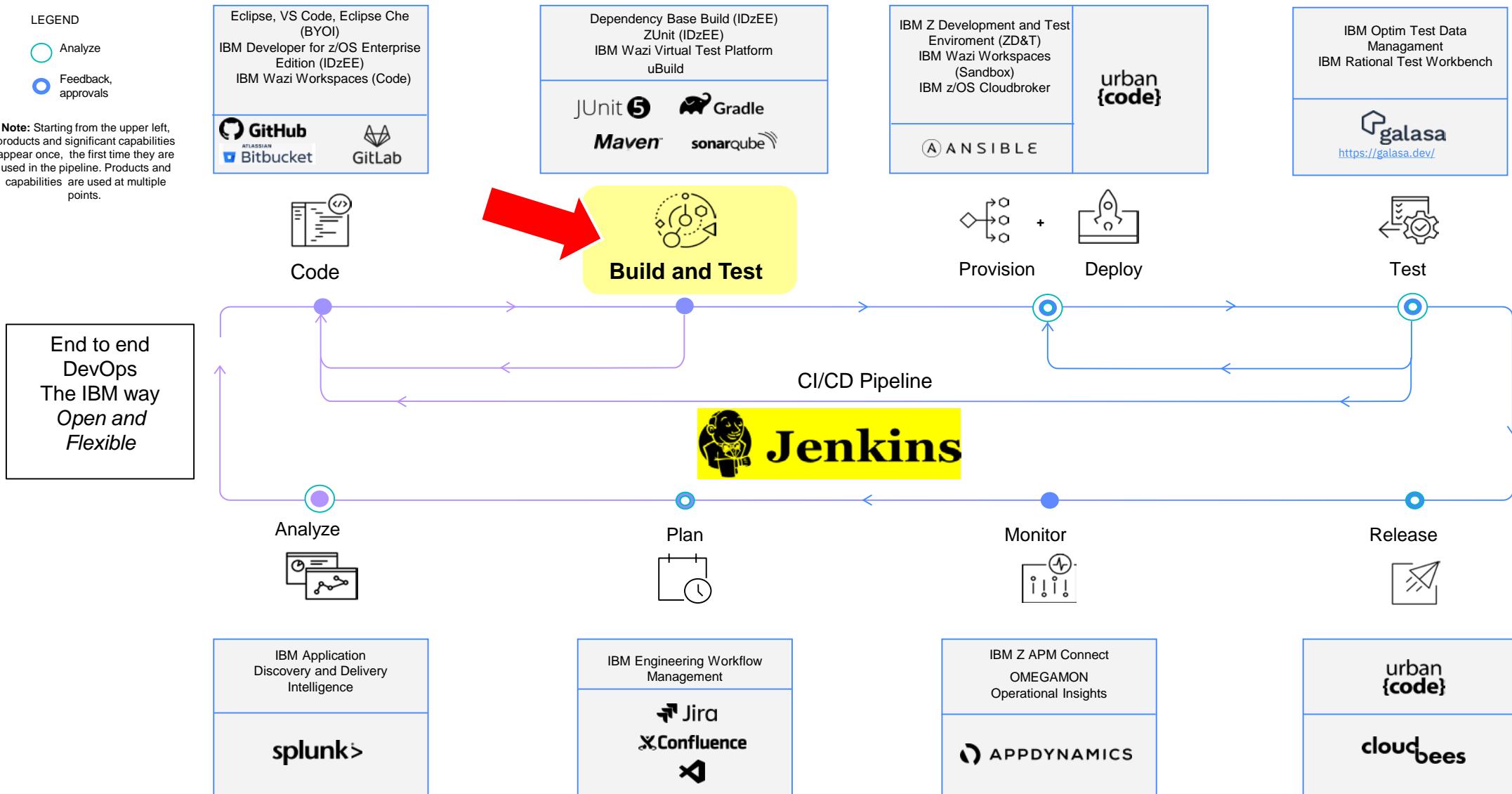


EGit included in IDz

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

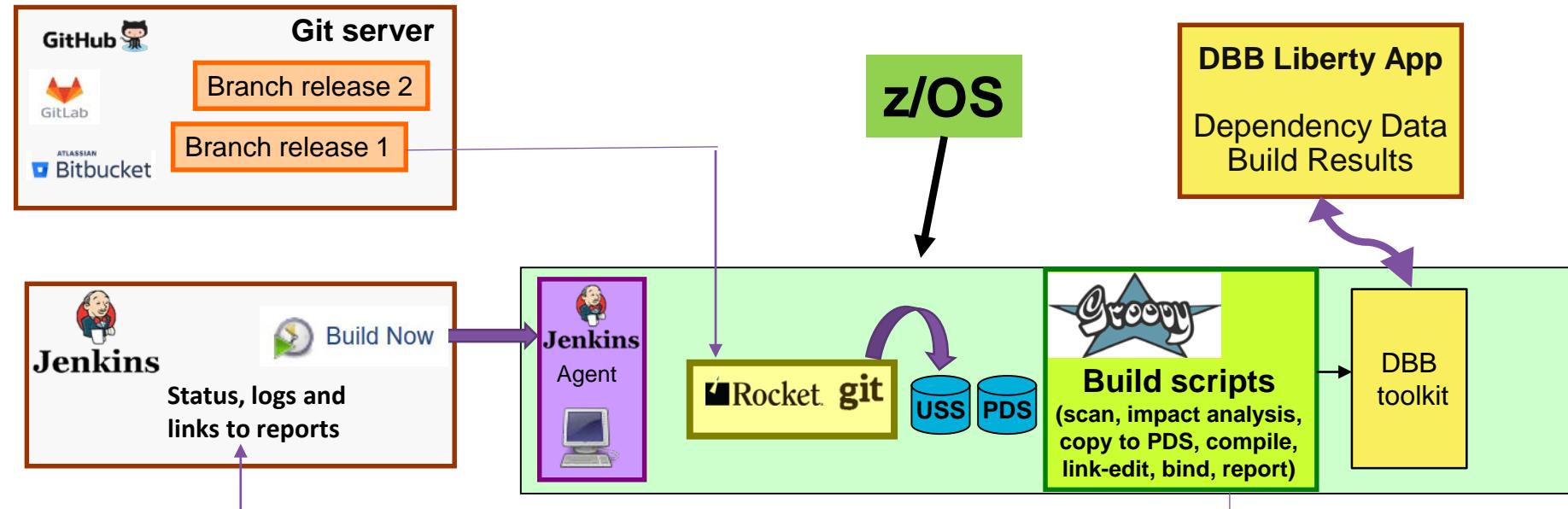
The screenshot illustrates the integration of EGit within the IBM Integration Designer (IDz) environment. On the left, the GitHub interface is visible, showing a repository named 'RegiBrazil / DemoHealthCare'. The main workspace shows a file tree for 'DemoHealthCare / DemoHealthCare / cobol_cics_db2' containing various COBOL source files like HCAPDB01.cbl, HCATDB01.cbl, etc. A red box highlights the repository name 'DemoHealthCare' in the GitHub header. On the right, the IDz interface displays a Git commit window for a commit titled 'Commit 59d1a2a816070038035ad43b56000d29bc5dc2b2'. The commit message is: 'diff --git a/DemoHealthCare/cobol_cics_db2/HCMADB02.cbl b/DemoHealthCare/cobol_cics_db2/HCMADB02.cbl index b3f3b88..1f6f463 100644 --- a/DemoHealthCare/cobol_cics_db2/HCMADB02.cbl +++ b/DemoHealthCare/cobol_cics_db2/HCMADB02.cbl @@ -1,1 +2,2 @@ - +'. Below the commit message, it says 'Pushed to DemoHealthCare - origin'. The commit details show a push from 'sandbox' to 'origin' at commit '59d1a2a..fb4f50c' on 2020-08-24 17:17:28, with a note about changing 'DemoHealthCare/cobol_cics_db2/HCMADB02.cbl'. The bottom section shows 'Message Details' with the repository path 'Repository git@github.com:RegiBrazil/DemoHealthCare.git'.

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)

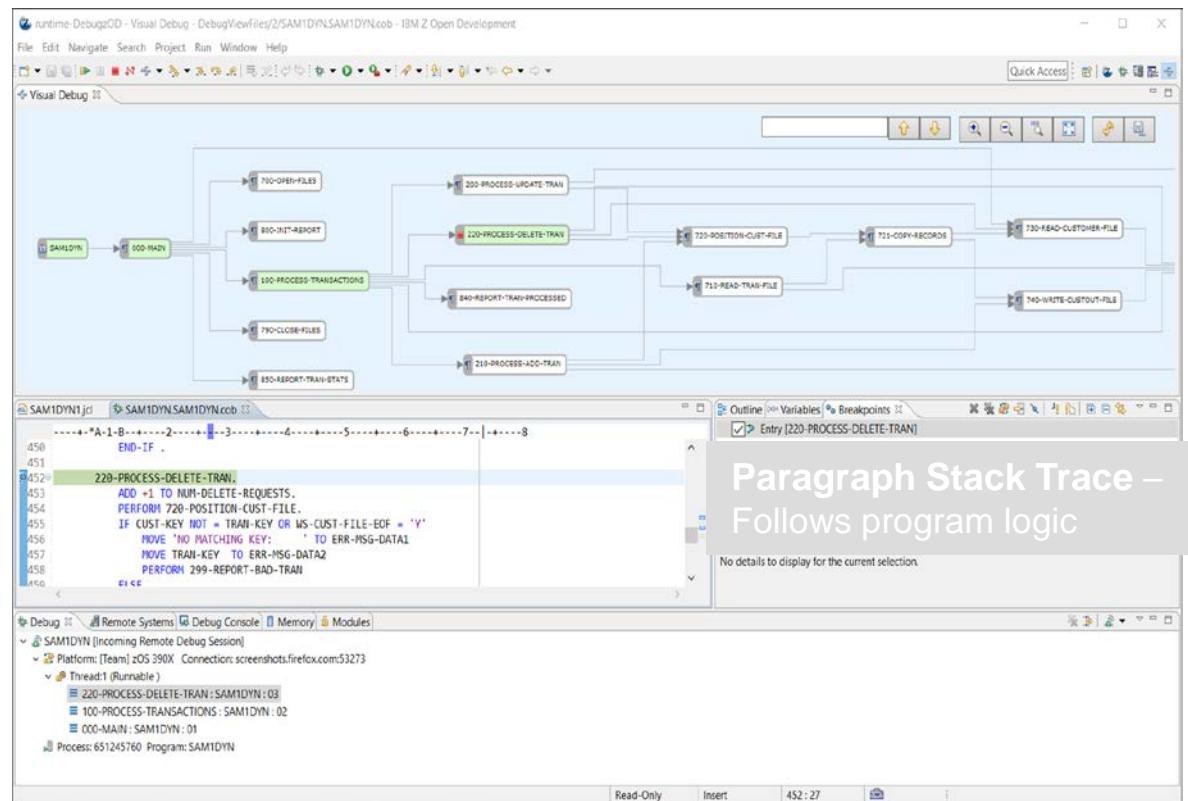
Program testing and analysis

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

New! Web UI look and feel (14.1.5)

With:

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



This screenshot shows a 3270 terminal window for a COBOL monitor session. The monitor is displaying input data for a birthdate field. The data shows a date of '1967-12-01'. The monitor also displays the source code for a verification routine, specifically lines 436 through 440. The terminal includes a command line at the bottom with PF keys and function keys for navigation.

```
COBOL LOCATION: CDAT1 :> 436.1
Command ==>
MONITOR <----1----2----3----4----5----6----7----8----9---- LINE: 1 OF 9
***** TOP OF MONITOR *****
-----1-----2-----3-----4-----
0001 1 03 W-COM-INPUT-BIRTHDATE
0002 04 W-COM-INPUT-DATE-CCYY
0003
0004 04 W-COM-INPUT-DATE-MM
0005 04 W-COM-INPUT-DATE-DD
0006 2 03 COM-INPUT-DATE
0007 04 COM-INPUT-DATE-CCYY
0008 04 COM-INPUT-DATE-MM
0009 04 COM-INPUT-DATE-DD
SOURCE CDAT1 <----1----2----3----4----5----6----7----8----9---- LINE: 434 OF 505
434
435 0500-VERIFY-INPUT-DATE.
436 IF W-COM-INPUT-BIRTHDATE NUMERIC
437 MOVE W-COM-INPUT-BIRTHDATE TO COM-INPUT-DATE, L-INPUT-
438 MOVE 'OK' TO W-DATE-VALID-SW
439 EVALUATE TRUE
440 WHEN W-COM-INPUT-DATE-CCYY < 1582
PF 1:?: 2:STEP 3:QUIT 4:LIST 5:FIND 6:AT/CLEAR
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:ZOOM LOG 12:RETRIEVE
MA a
PF 1:?: 2:STEP 3:QUIT 4:LIST 5:FIND 6:AT/CLEAR
PF 7:UP 8:DOWN 9:GO 10:ZOOM 11:ZOOM LOG 12:RETRIEVE
02/015
```

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

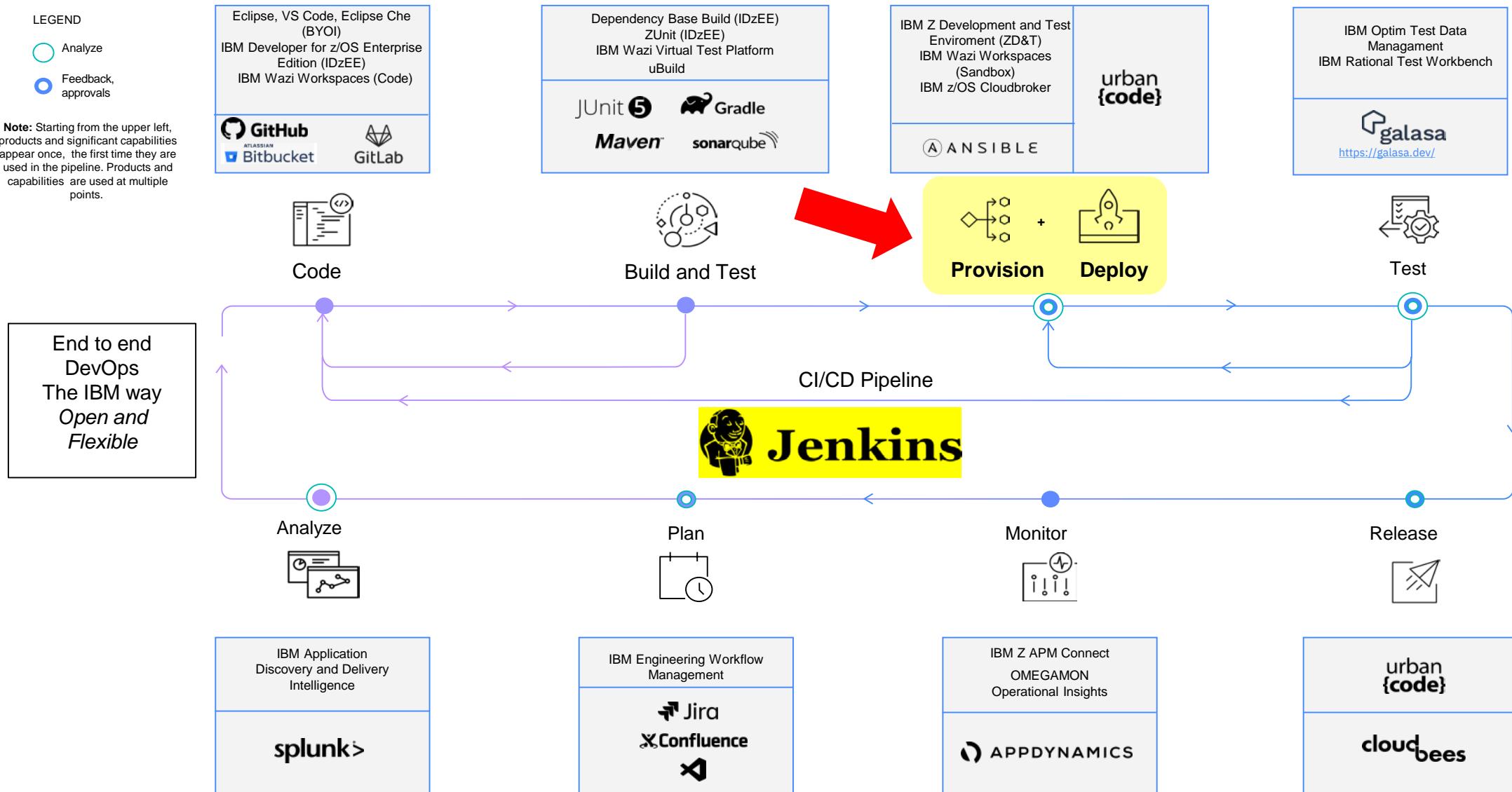
The screenshot displays the Rational Coverage tool interface, which includes:

- Code Coverage Results View:** A tree-based view showing coverage data for various workspaces and specific runs. A context menu is open over a run named "SAM1_2018_11_13_114716_0952". The "Compare" option is highlighted.
- Compare Results View:** A modal window titled "Compare Report (Dec 1, 2018 2:40:17 PM)" showing a "Code Coverage Comparison Report" for two runs: "SAM1_2018_12_01_144006_0842" and "SAM1_2018_11_13_114716_0952". It provides a detailed breakdown of coverage metrics for each program component.
- Coverage View:** A modal window showing the source code of a COBOL program ("WALEX.DEMO211.COBOL(SAM1).cob") with line numbers and coverage percentages indicated by colored bars.

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

Name	Coverage	Lines Covered	Uncovered Lines	Total Lines	Messages
'WALEX.DEMO211.COBOL(SAM1).cob'	75% (+7)	115 → 127	54 → 42	169	
SAM1	75% (+7)	115 → 127	54 → 42	169	
000-MAIN	100% (0)	11	0	11	
100-PROCESS-TRANSACTIONS	75% (+4)	17 → 18	7 → 6	24	
200-PROCESS-UPDATE-TRAN	77% (+8)	9 → 10	4 → 3	13	
210-PROCESS-ADD-TRAN	79% (+72)	1 → 11	13 → 3	14	
220-PROCESS-DELETE-TRAN	63% (0)	5	3	8	
299-REPORT-BAD-TRAN	100% (0)	6	0	6	
700-OPEN-FILES	29% (0)	5	12	17	
710-READ-TRAN-FILE	64% (0)	9	5	14	
720-POSITION-CUST-FILE	100% (0)	3	0	3	
721-COPY-RECORDS	100% (0)	3	0	3	
730-READ-CUSTOMER-FILE	67% (0)	8	4	12	
740-WRITE-CUSTOUT-FILE	40% (0)	4	6	10	
790-CLOSE-FILES	100% (0)	4	0	4	
800-INIT-REPORT	100% (0)	7	0	7	
830-REPORT-TRAN-PROCESSED	100% (0)	5	0	5	

Z DevOps Pipeline

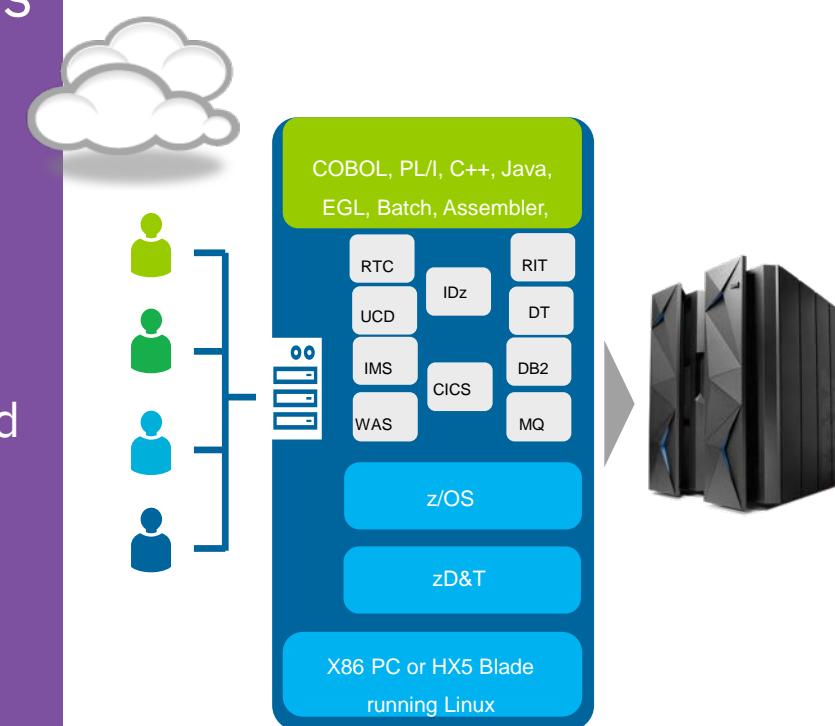


Provisioning

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

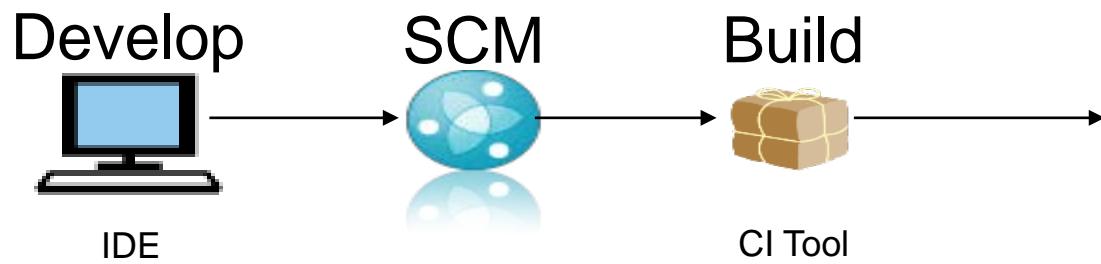
Develop and test IBM Z applications anywhere, anytime with z/OS optimized for x86 hardware

- **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)

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

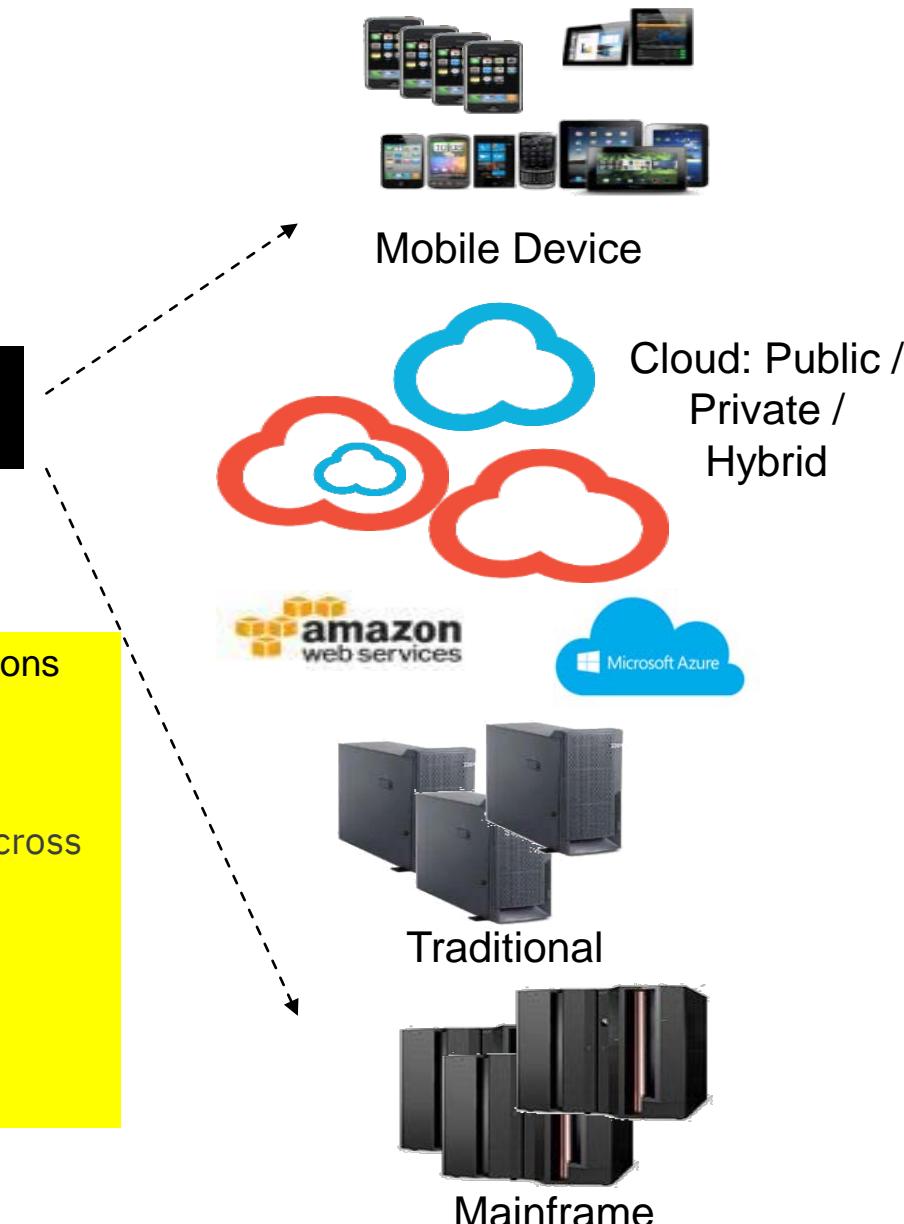


IBM UrbanCode Deploy automates the deployment of applications, databases and configurations into development, test and production environments, helping to drive down cost, speed time to market with reduced risk.

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

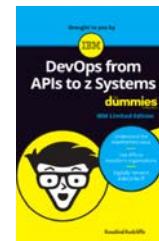
Integrates with Jenkins

Manages artifact versions via an in-built repository called codestation



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>



© Copyright IBM Corporation 2019

Questions / Comments

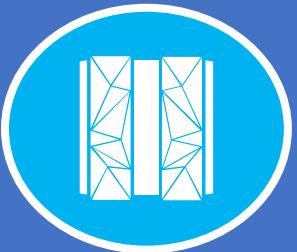


Agenda - Day 1

- 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)



DevOps Mainframe Tooling - Labs

Labs

Labs using VMware ...

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



Each time you see this symbol ► it means that you have to "do" something on your computer – not merely read the document.

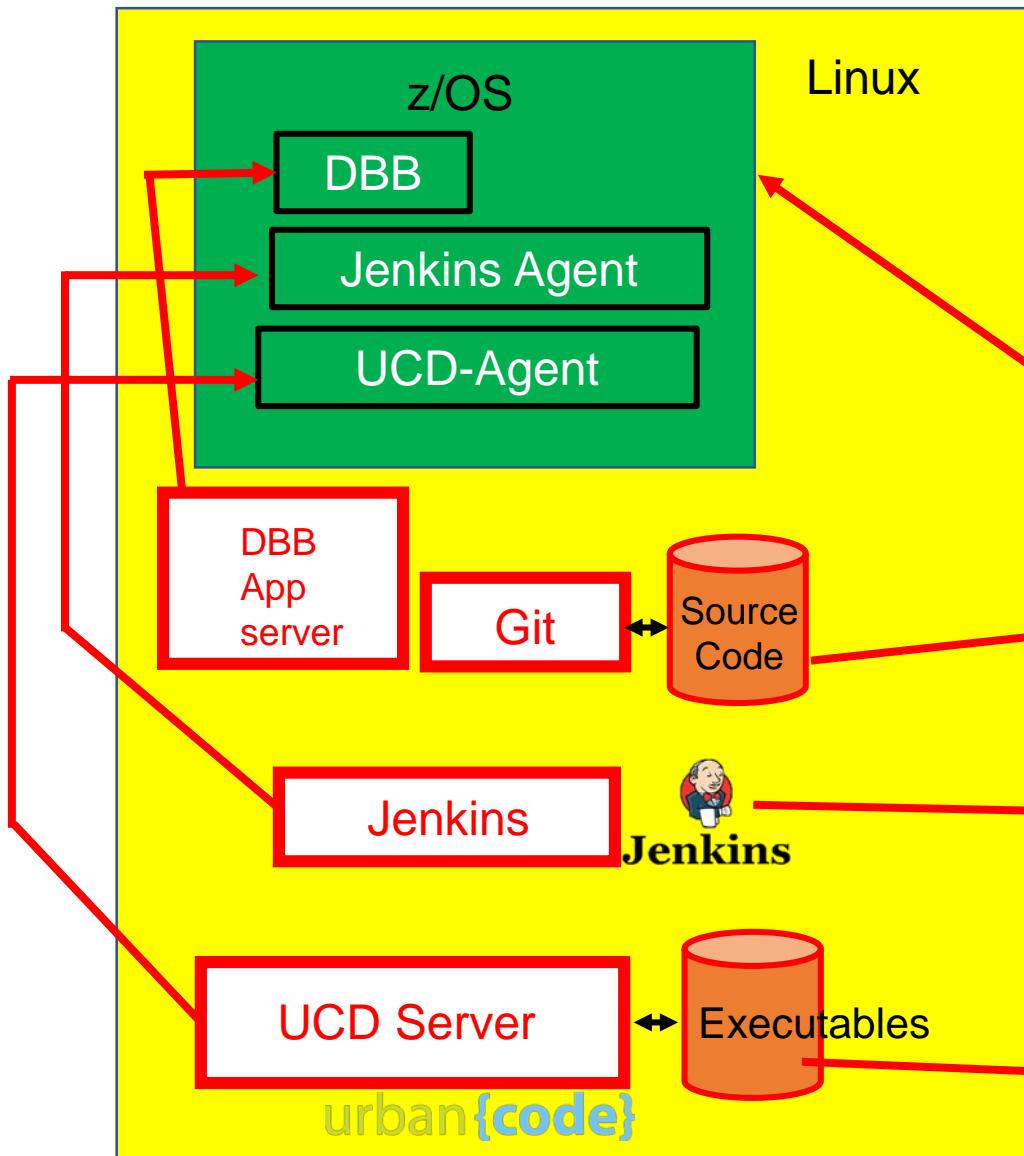
Tips!

- 1) If you want to have the lab in HTML format displayed in your browser, you can find it at the location: C:\RDZ8.0_POT\HTML
Then you can COPY/PASTE the names and the code, using the Browser.
- 2) Most of the labs will be performed under VMware. So we will run 2 'Windows' in the same machine. This will cause some overhead and sometimes performance will not be as good as if the program would be running in the native Windows... So, please be patient
- 3) If you lost the "VMware full screen" either type **Ctrl + Alt + Enter** or use the VMware icon (on top).

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

zD&T on Cloud environment

zD&T



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

zos.dev:2800/DID10.HIST(F00307)-Report S3

```
1@ 2 Module DB2REGI, program DB2REGI, source line # 365: Abend S0CB (Decimal-Divide Exception)
3 IBM FAULT ANALYZER SYNOPSIS
4
5
6 A system abend 0CB occurred in module DB2REGI program DB2REGI at offset X'FCE'.
7
8 A program-interruption code 000B (Decimal-Divide Exception) is associated with
9 this abend and indicates that:
10
11 The divisor was zero in a signed decimal division.
12
13 The cause of the failure was program DB2REGI in module DB2REGI. The COBOL
14 source code that immediately preceded the failure was:
15
16 Source
17 Line #
18 -----
19 000365 DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
20
21 The COBOL source code for data fields involved in the failure:
22
23 Source
24 Line #
25 -----
26 000200 03 RECEIVED-FROM-CALLED PIC 99.
27 000201 03 VALUE1 PIC 99.
28 000202 03 RESULT PIC 99.
```

520-LOGIC.
IF WHICH-LAB = 'LAB2'
* If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
MOVE "REG10B" TO PROGRAM-TO-CALL
CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
MOVE 66 TO VALUE1
DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
DISPLAY "The result is ... " RESULT
END-IF

000200 03 RECEIVED-FROM-CALLED PIC 99.
000201 03 VALUE1 PIC 99.
000202 03 RESULT PIC 99.

```
graph LR; DB2REGI[DB2REGI] --> 000SETUP[000-SETUP-ERROR-TRAP-RTN]; 000SETUP --> 000MAINLINE[000-MAINLINE-RTN]; 000MAINLINE --> 100DECLARE[100-DECLARE-CURSOR-RTN]; 100DECLARE --> 100EXIT[100-EXIT]; 000MAINLINE --> 150OPEN[150-OPEN-CURSOR-RTN]; 150OPEN --> 150EXIT[150-EXIT]; 000MAINLINE --> 200FETCH[200-FETCH-RTN]; 200FETCH --> 200EXIT[200-EXIT]; 200FETCH --> 250FETCH[250-FETCH-A-ROW]; 250FETCH --> 250EXIT[250-EXIT]; 000MAINLINE --> 300CLOSE[300-CLOSE-CURSOR-RTN]; 300CLOSE --> 300EXIT[300-EXIT]; 000MAINLINE --> 350TERMINATE[350-TERMINATE-RTN]; 350TERMINATE --> 350EXIT[350-EXIT]; 350TERMINATE --> 500SECOND[500-SECOND-PART]; 500SECOND --> 520LOGIC[520-LOGIC]; 520LOGIC --> 530SEYYA[530-SEYYA]; 530SEYYA --> 540GOODBYE[540-GOODBYE]; 540GOODBYE --> 999EXIT[999-EXIT];
```

Here calls REG10B that abends

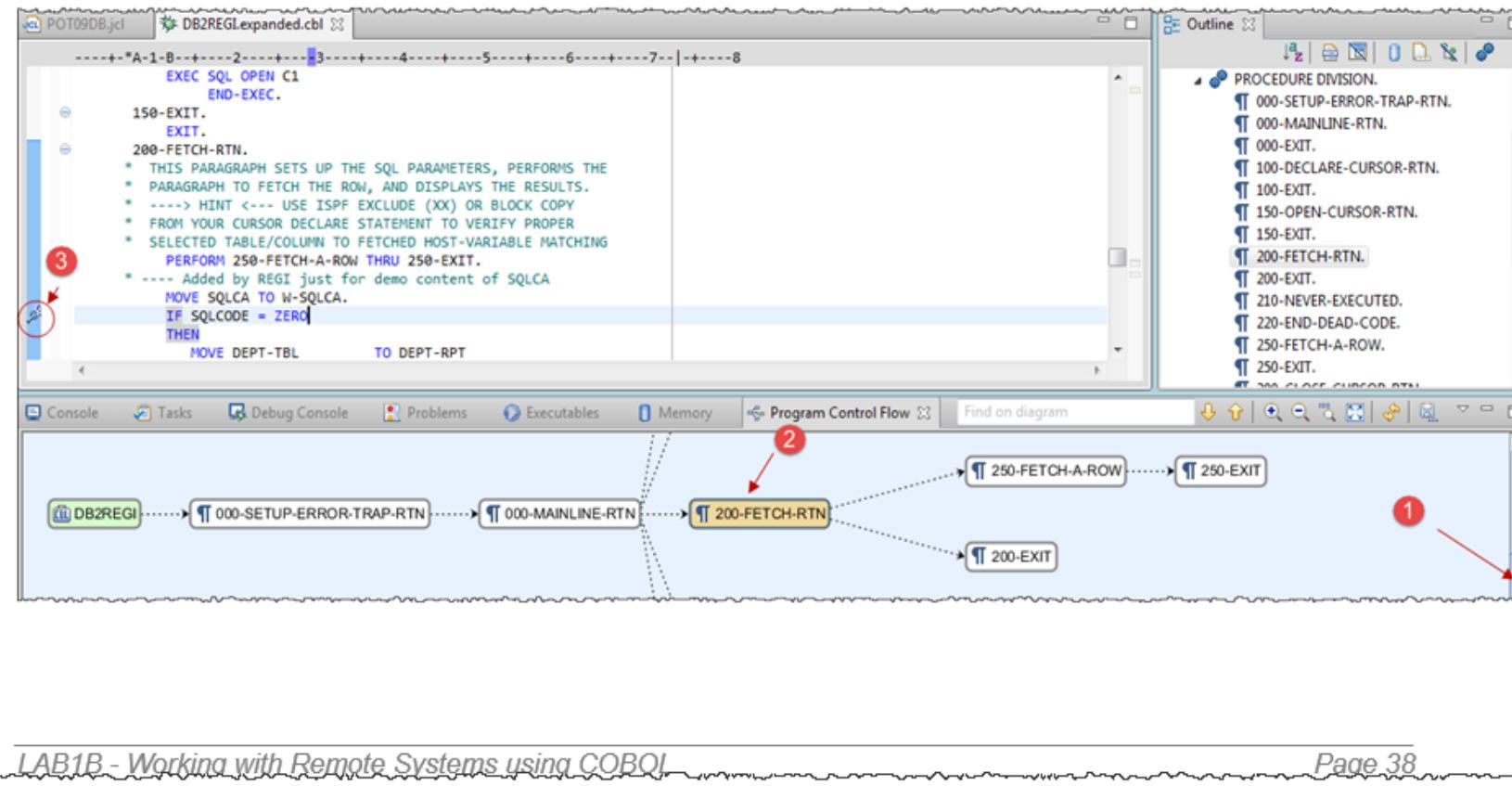
Lab 1: Don't Miss... The Debug... Page 22

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



The screenshot shows the Rational Application Developer interface. The top part is the COBOL editor with code for a program named DB2REGI. The code includes several paragraphs like 150-EXIT, 200-FETCH-RTN, and 250-FETCH-A-ROW, along with comments and an IF statement. A red circle labeled 3 points to the IF SQLCODE = ZERO line. The bottom part is the Program Control Flow diagram. It shows nodes for DB2REGI, 000-SETUP-ERROR-TRAP-RTN, 000-MAINLINE-RTN, 200-FETCH-RTN, 250-FETCH-A-ROW, and 250-EXIT. Arrows indicate the flow between these nodes. A red circle labeled 2 points to the 200-FETCH-RTN node. A red circle labeled 1 points to the vertical blue scrollbar on the left side of the editor window.

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**.

The screenshot shows a COBOL code editor window with the title bar "DB2REGI.expanded.cbl". The code is annotated with color-coded highlights:

- Red highlights:** Lines 350, 360, and the entire section from * 500-SECOND-PART to MOVE 2 TO BRANCHFLAG are circled in red. A yellow callout bubble labeled "not executed" points to the bottom of this section.
- Green highlights:** Lines 340, 350, and 360 are circled in green. A yellow callout bubble labeled "executed" points to the first line of the 360 block.

```
      +-+-----+-----+-----+-----+-----+-----+-----+-----+
      |A|1|B|2|3|4|5|6|7|8|
      +-+-----+-----+-----+-----+-----+-----+-----+-----+
      EXIT.
      350-TERMINATE-RTN.
      MOVE ROW-KTR TO ROW-STAT.
      DISPLAY ROW-MSG.
      360 EXIT.
      * EXIT.
      GO TO 500-SECOND-PART.
      999-ERROR-TRAP-RTN.
*****
*   ERROR TRAPPING ROUTINE FOR NEGATIVE SQLCODES *
*****
      DISPLAY '***** WE HAVE A SERIOUS PROBLEM HERE *****'.
      DISPLAY '999-ERROR-TRAP-RTN '.
      MULTIPLY SQLCODE BY -1 GIVING SQLCODE.
      DISPLAY 'SQLCODE ==> ' SQLCODE.
      DISPLAY SQLCA.
      DISPLAY SQLERRM.
      EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
      EXEC SQL ROLLBACK WORK END-EXEC.
      *ROLLBACK
      * 500-SECOND-PART.
      MOVE 2 TO BRANCHFLAG.
```

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 User id →  empot05 and password → empot05

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)

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)

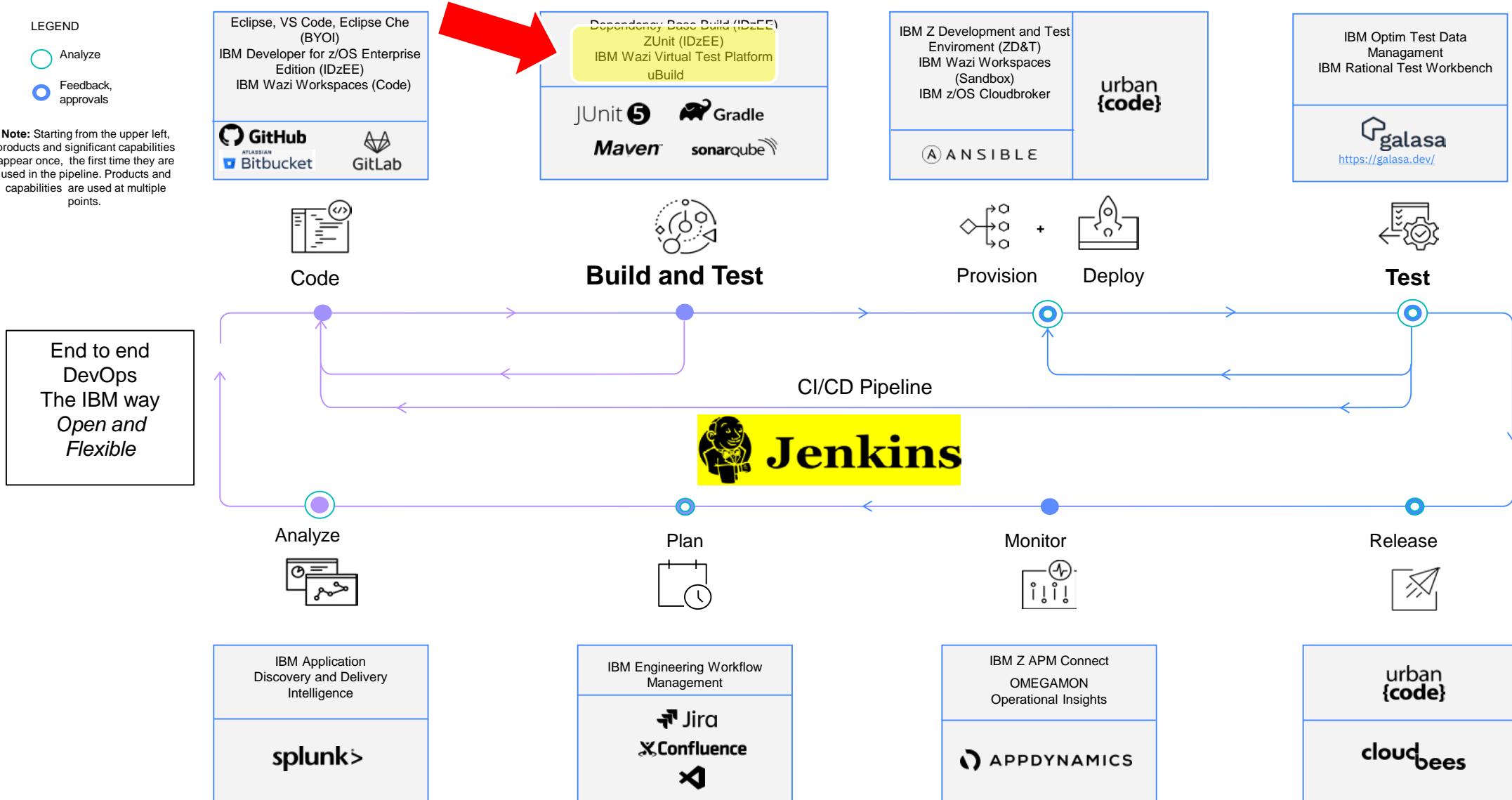
10:15 Demo: Scenario using Z DevOps solutions including **zUnit, Git and Jenkins** (Regi)

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)

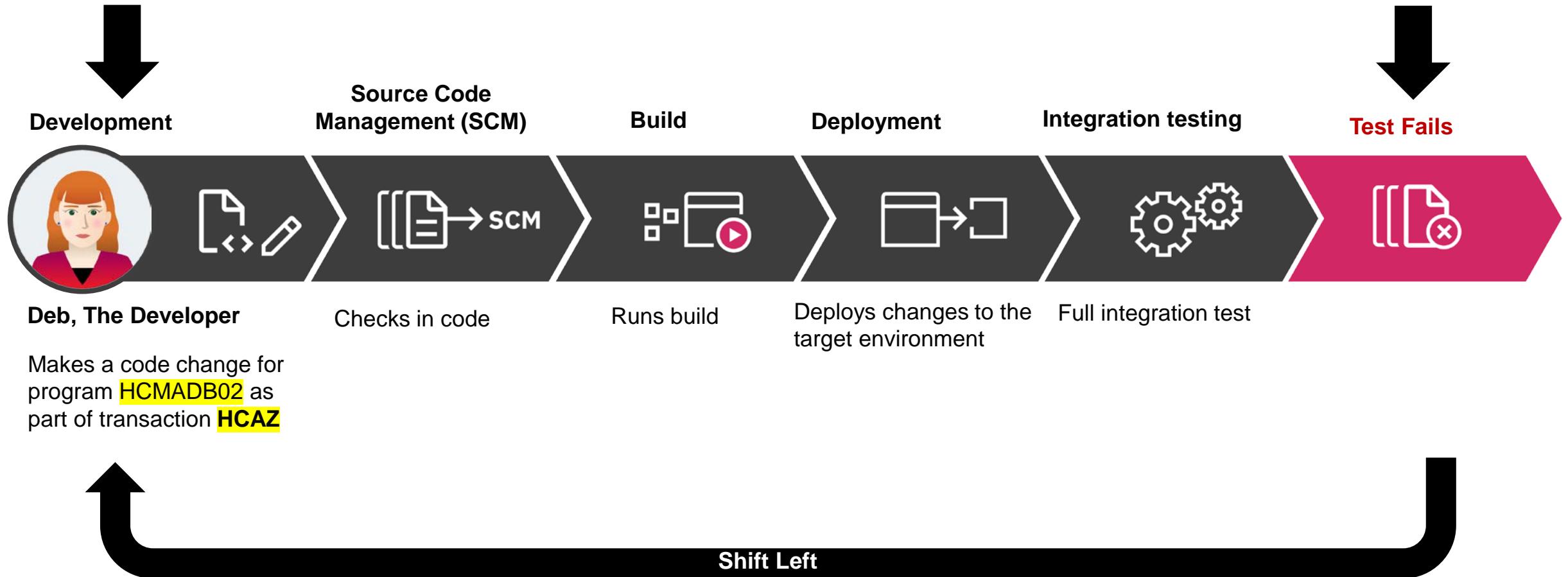
Z DevOps Pipeline



Shift Left testing

Start your testing here

Don't wait until here



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

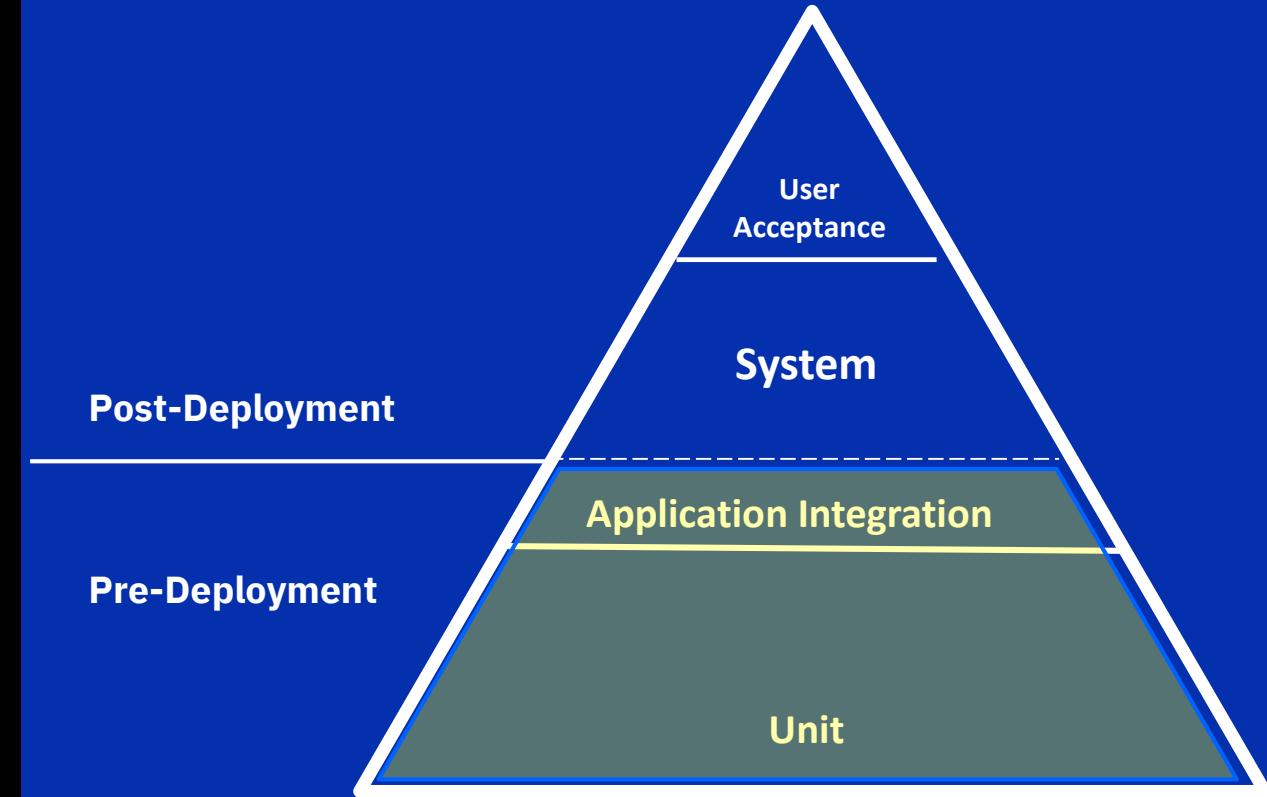
RTW (Rational Integration Tester/Rational functional tester/Test Virtualization Server)

Application Integration test

IBM Wazi Virtual Test Platform

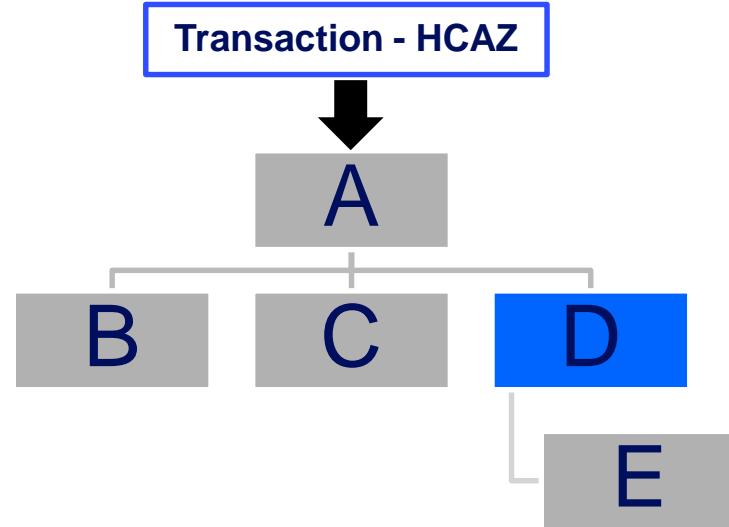
Unit test

ZUnit (IBM Developer for z/OS)



Unit Testing (zUnit)

Scope - Program



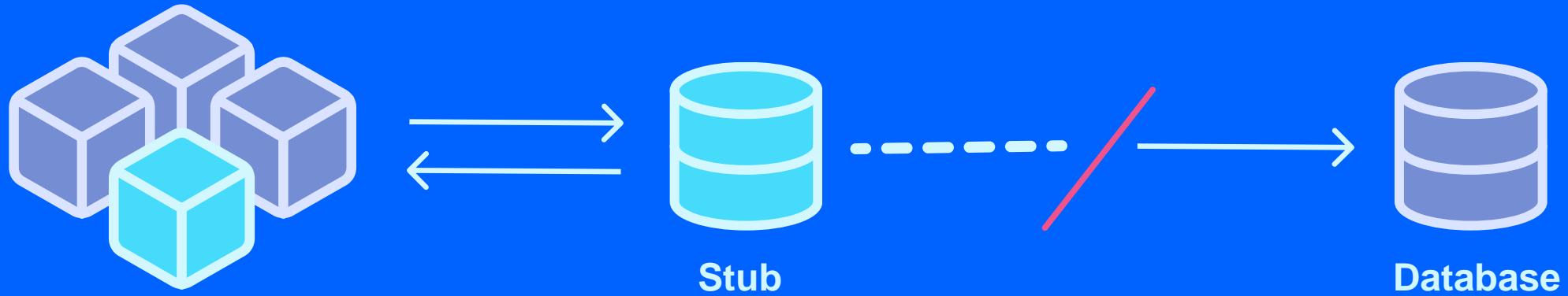
A unit is the smallest testable part of software – meaning testing an individual unit of source code, like a single source file which contains a list of methods, procedures.

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

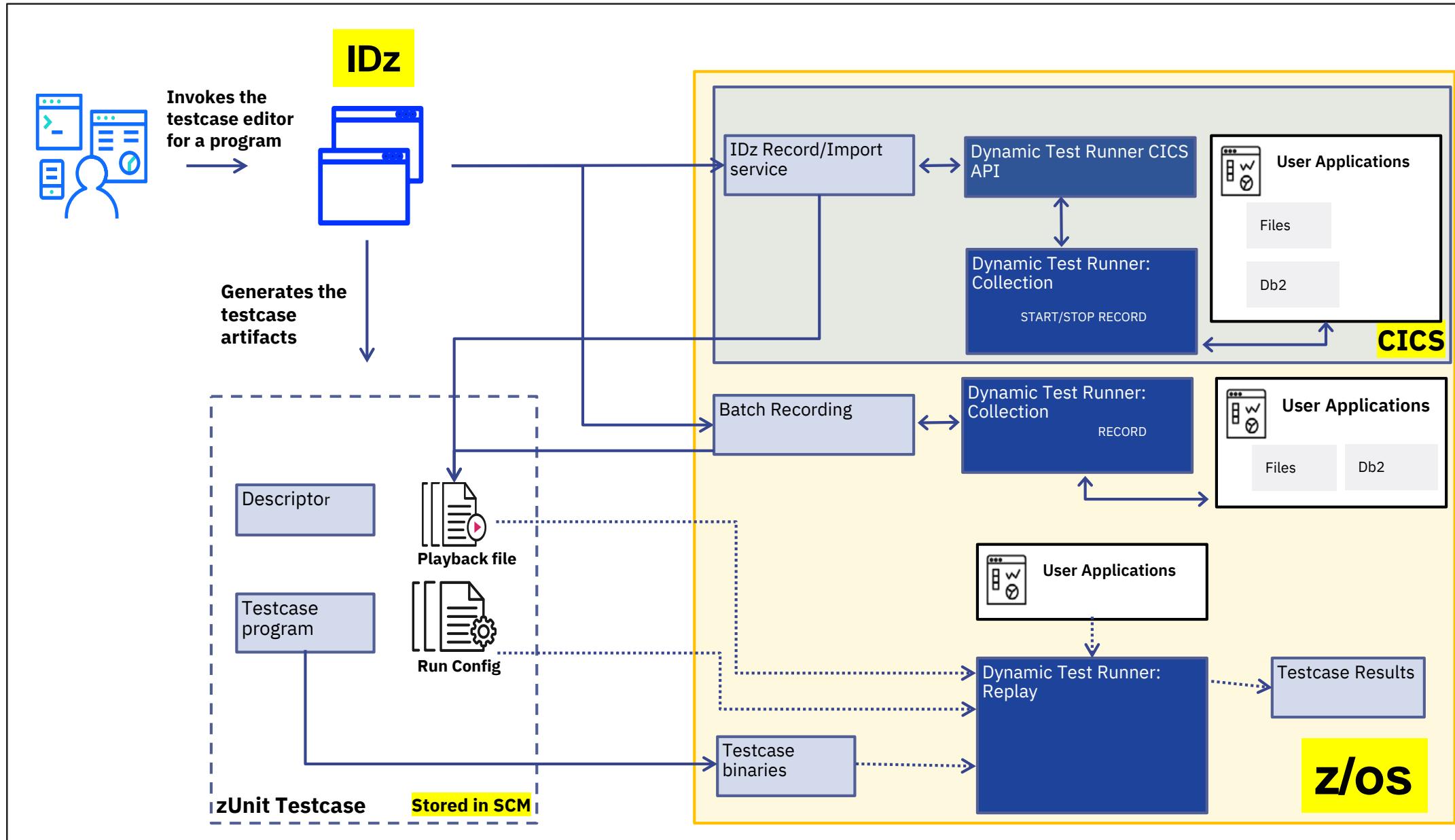
Shift Left – unit testing with ZUnit

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

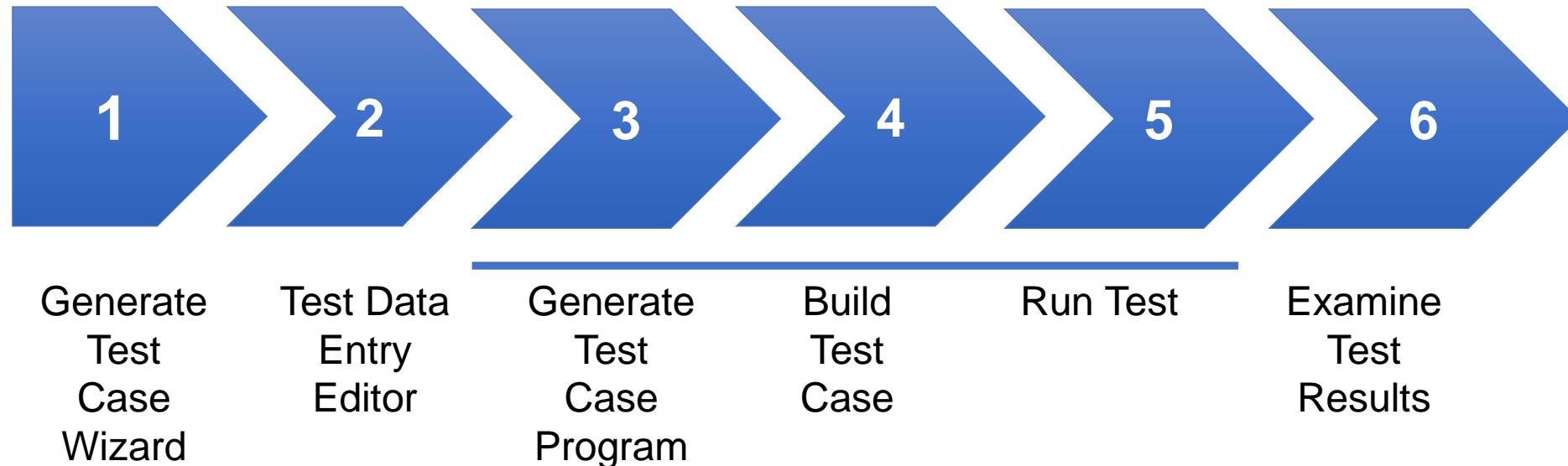
This is done at the **program level**



ZUnit High Level Architecture

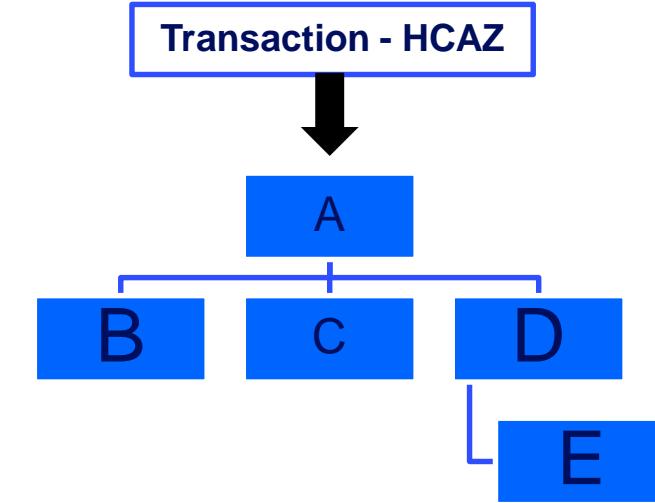


zUnit Basic Workflow



Application Integration test (Wazi VTP)

Scope - Transaction

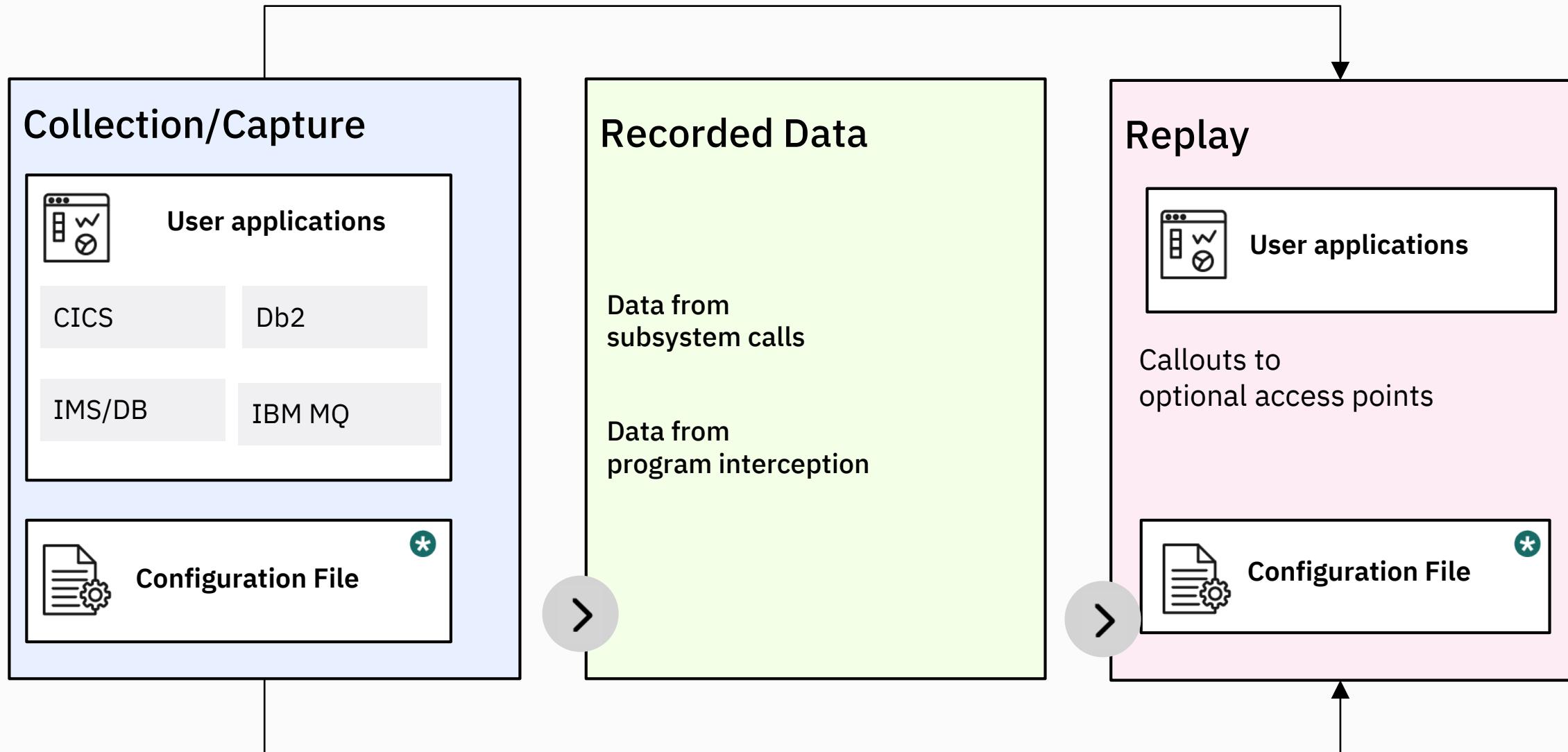


Application integration test is the next level of testing after Unit Testing. Its purpose is to confirm that the changes in a unit (programs) works with the interfaces to other units (programs), and ensure no unexpected impacts to other units

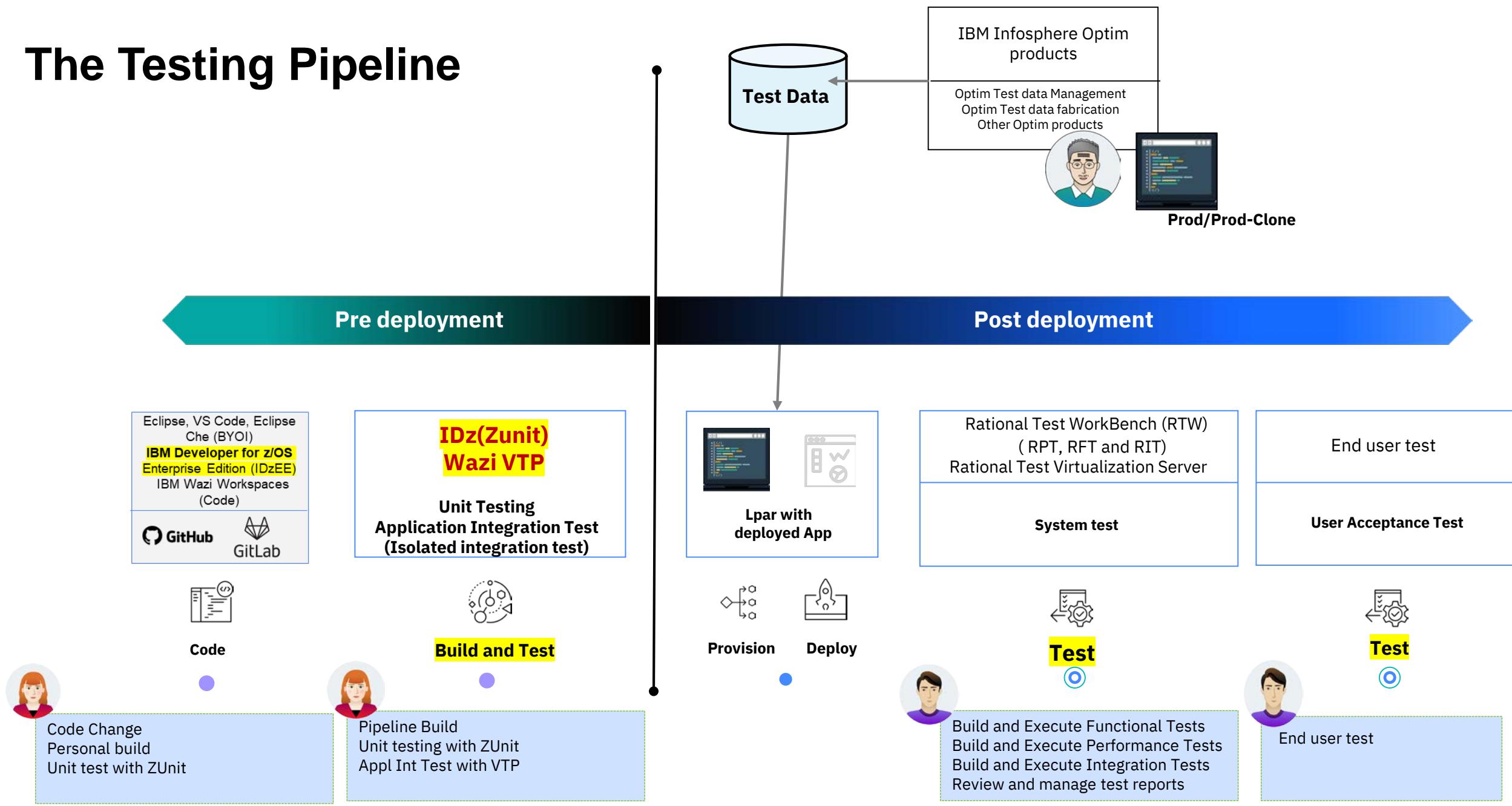
Transaction and batch jobs

Testing with Wazi Virtual Test Platform

Part 1 – Capture or Collection



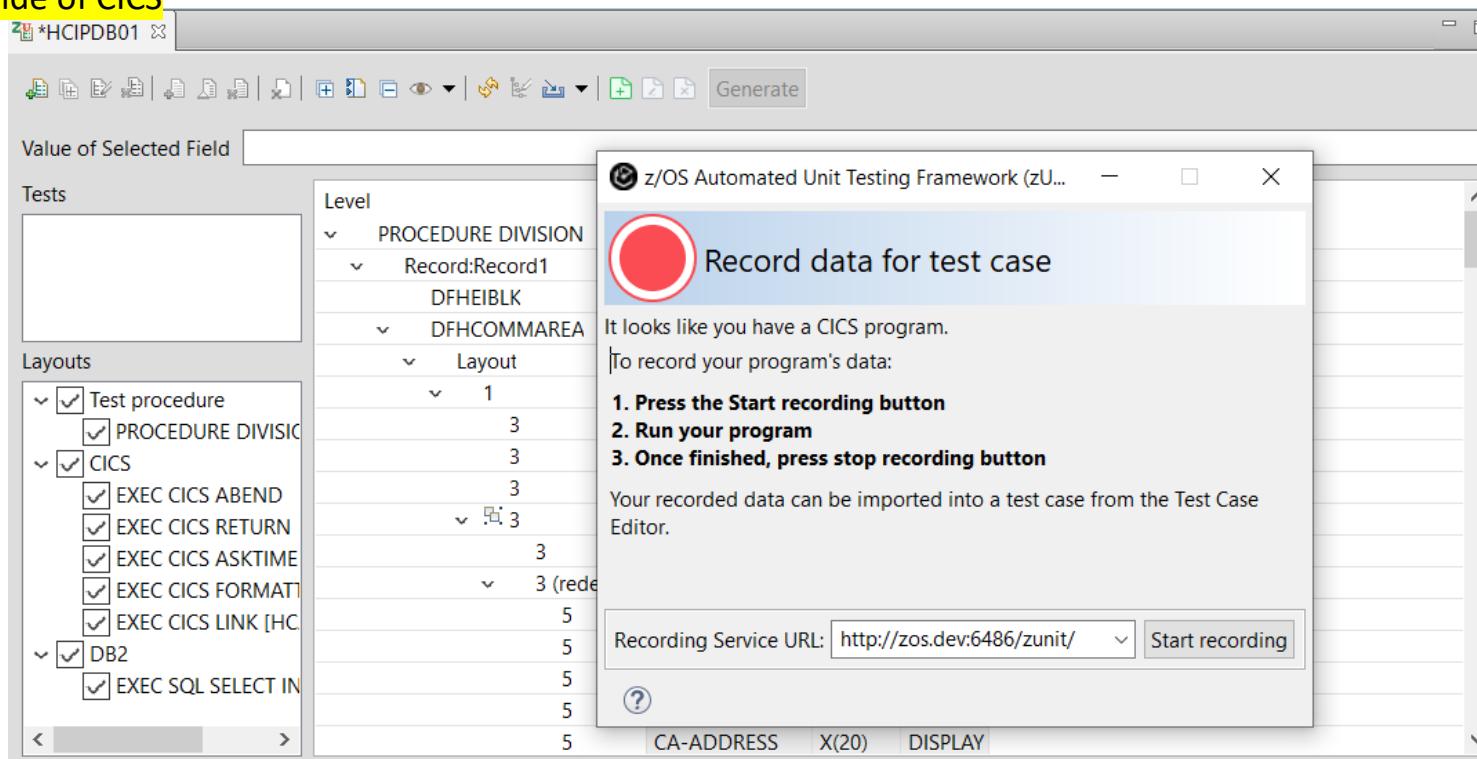
The Testing Pipeline



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)

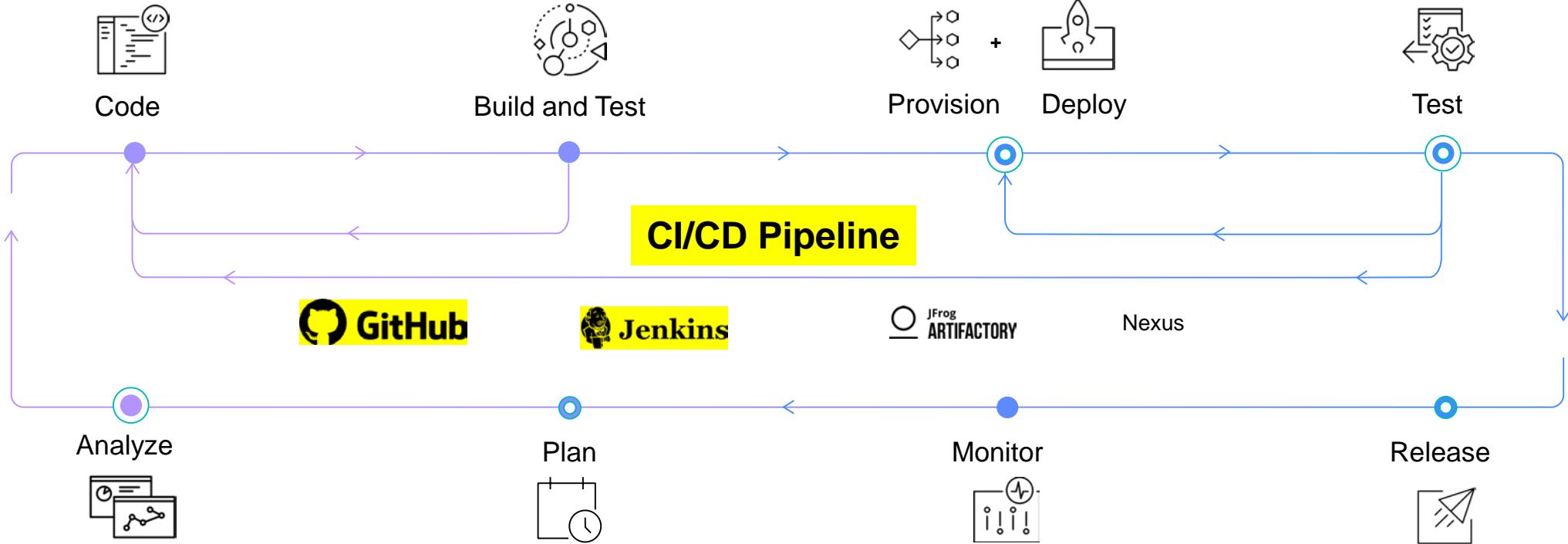
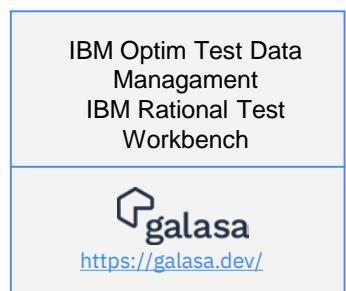
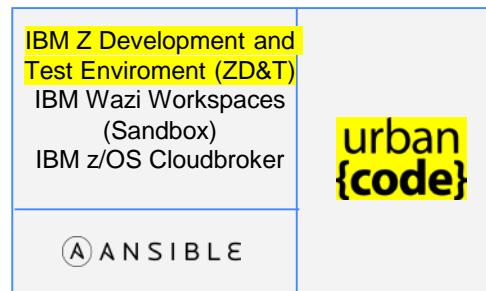
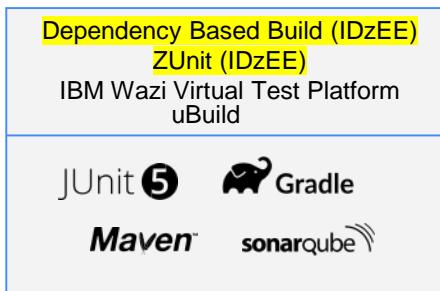
12:50 Checkpoint - Comments and Closing Day 2 (Tim -Ron/Wilbert/Regi)

LEGEND

 Analyze

 Feedback,
approvals

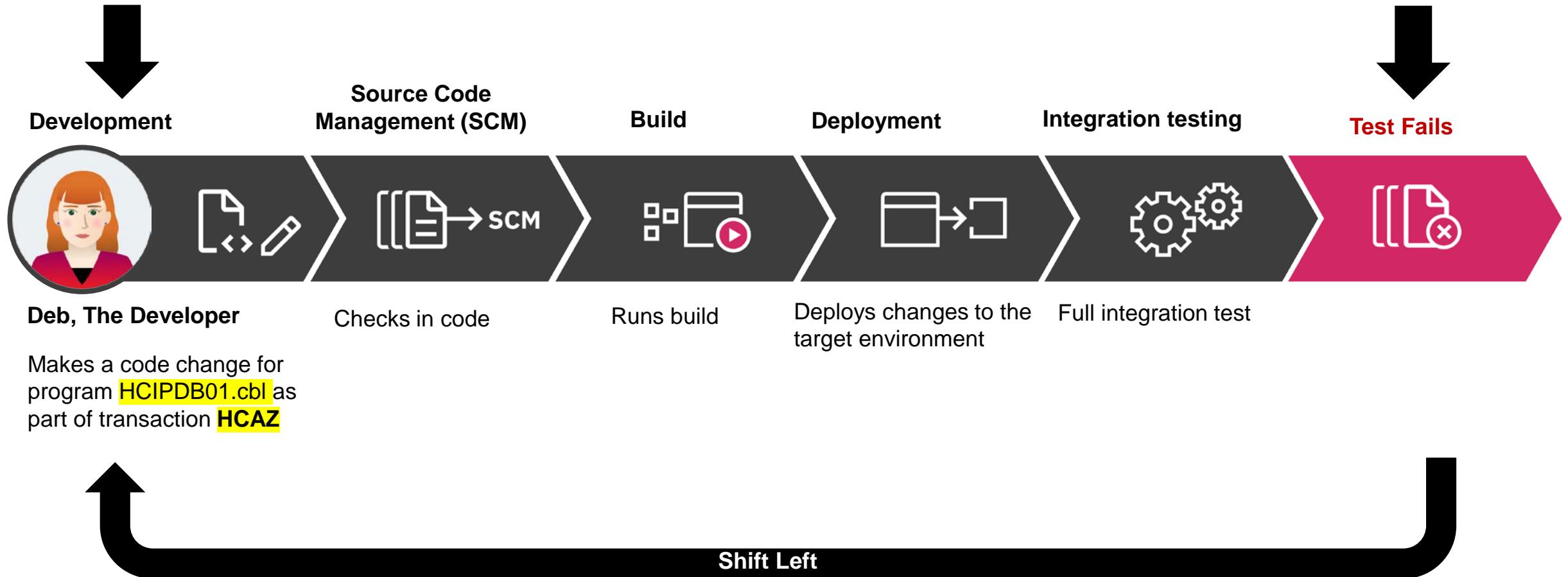
Note: Starting from the upper left, products and significant capabilities appear once, the first time they are used in the pipeline. Products and capabilities are used at multiple points.



Shift Left testing

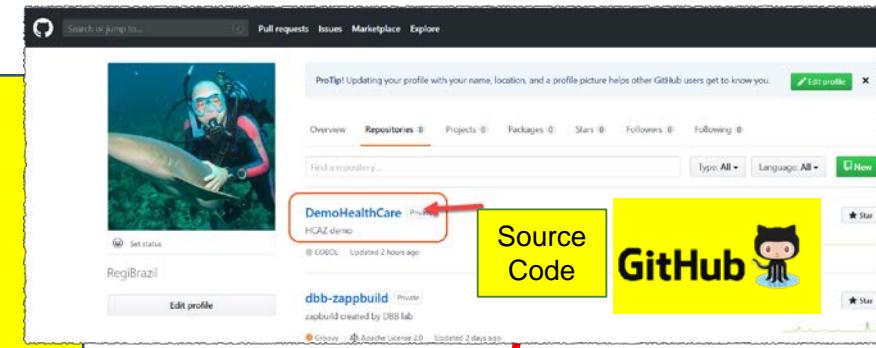
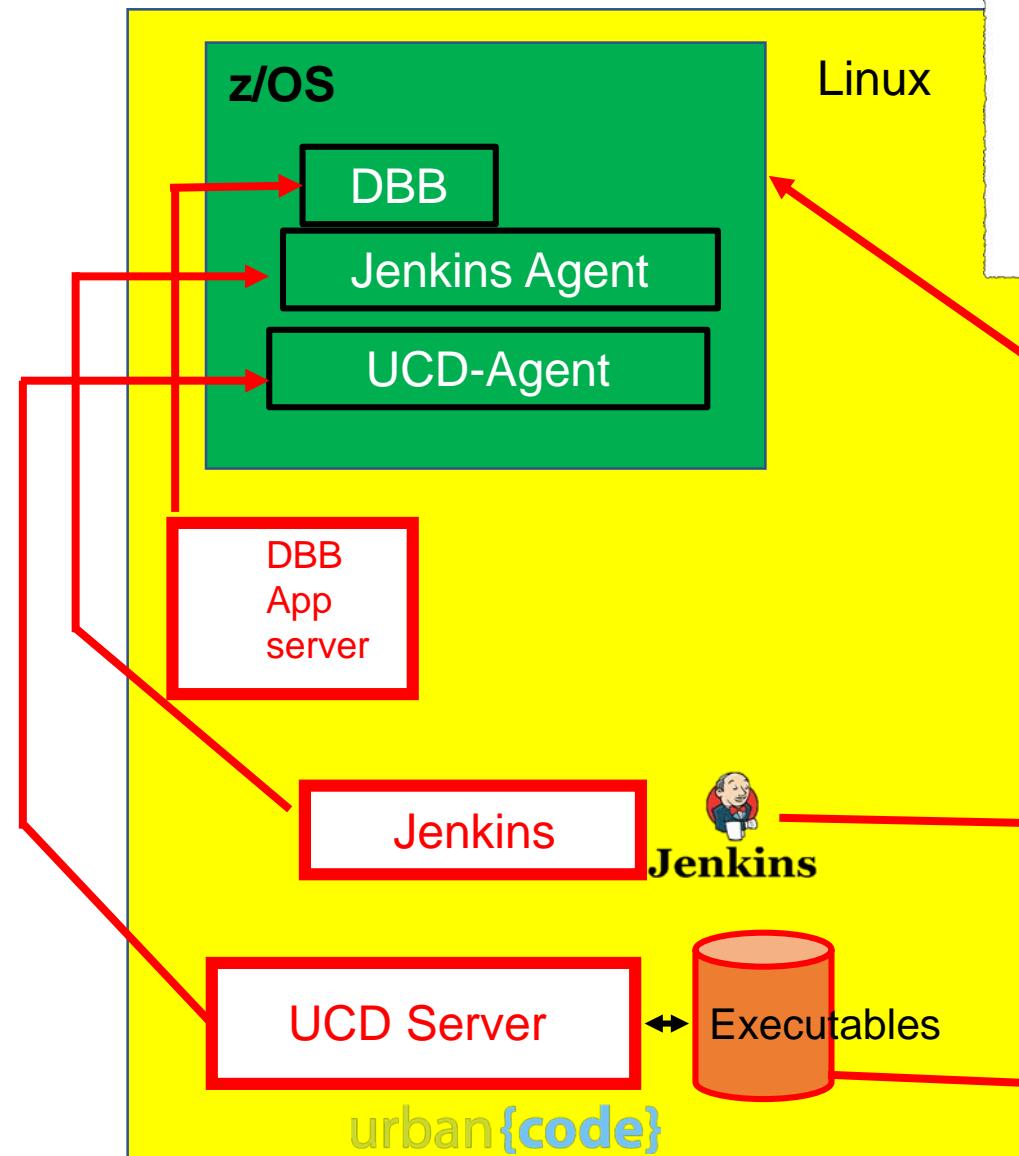
Start your testing here

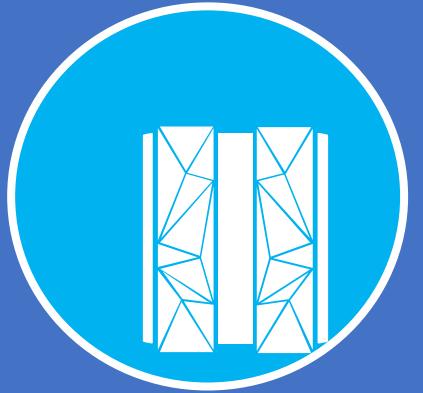
Don't wait until here



Topology used on this Demo

zD&T on VMWARE





Z DevOps Tooling

Demonstration COBOL/CICS/DB2 Program

PART 1 - Z Unit Test using IDz (zUnit)

Demonstration

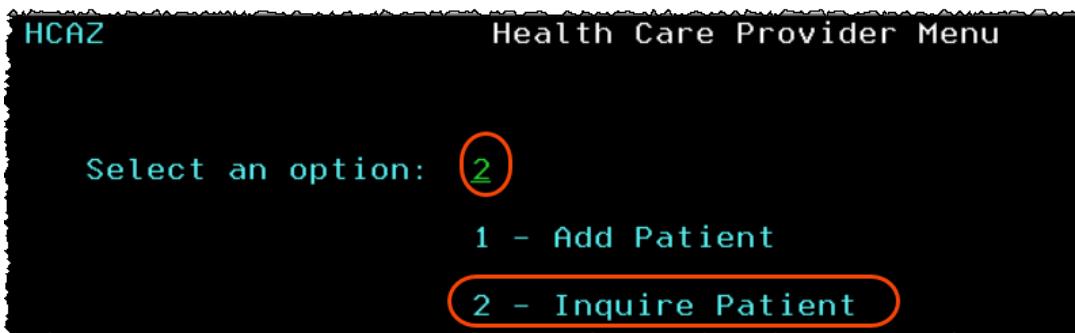
Scenario: COBOL CICS/DB2 Transaction

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



HCP1

Inquire Patient Info

Patient ID	0000000001
Name :First	Ralph
:Last	DALmeida
DOB	1980-07-11 (yyyy-mm-dd)
Address	34 Main Street
City	Toronto
Postcode	M5H 1T1
Phone: Mobile	077-123-9987
Email Addr	RalphD@ibm.com
Insurance Card	9627811234
User ID	ralphd

HCP1

Inquire Patient Information

Patient ID	0000000002
Name :First	John
:Last	Smith
DOB	1965-09-30 (yyyy-mm-dd)
Address	2 Smith Street
City	New York
Postcode	12345
Phone: Mobile	001-911-9113
Email Addr	JohnSmith@email.com
Insurance Card	1127811234
User ID	johns

A screenshot of two terminal windows. The left window is titled 'Inquire Patient Info' and shows patient details for 'Patient ID 0000000001'. The right window is titled 'Inquire Patient Information' and shows patient details for 'Patient ID 0000000002'. Both windows have their respective patient IDs highlighted with red ovals and red arrows pointing to them from the text above.

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

The screenshot illustrates the process of invoking the 'inquirepatient' API for two different patient numbers (1 and 2) using the z/OS Connect interface.

Step 1: The user right-clicks on the 'inquirepatient' API entry in the 'APIs' section of the z/OS Connect interface. A context menu appears with the option 'Open In Swagger UI' highlighted.

Step 2: In the 'Parameters' section of the Swagger UI, the 'patNum' parameter is set to '1'. A red arrow points from this step to the 'patNum' input field.

Step 3: The 'Try it out!' button is clicked, and the response body is displayed. The response shows patient information for 'John' (highlighted with a red oval). A red circle labeled '3' is placed over the 'CA_FIRST_NAME' field in the JSON response.

Step 4: The 'patNum' parameter is changed to '2' in the 'Parameters' section of the Swagger UI. A red arrow points from this step to the 'patNum' input field.

Step 5: The 'Try it out!' button is clicked again, and the response body is displayed. The response shows patient information for 'Ralph' (highlighted with a red oval). A red circle labeled '5' is placed over the 'CA_FIRST_NAME' field in the JSON response.

Response Body (JSON Format):

```
{
  "HCCMAREA": {
    "CA_REQUEST_ID": "01IPAT",
    "CA_PATIENT_REQUEST": {
      "CA_PHONE_MOBILE": "077-123-9987",
      "CA_ADDRESS": "34 Main Street",
      "CA_LAST_NAME": "DALmeida",
      "CA_CITY": "Toronto",
      "CA_POSTCODE": "M5H 1T1",
      "CA_INS_CARD_NUM": "9627811234",
      "CA_FIRST_NAME": "Ralph" 3
    },
    "CA_PATIENT_ID": 1,
    "CA_RETURN_CODE": 0
  }
}
```

Annotations:

- Red circle labeled '1': Points to the 'Open In Swagger UI' option in the context menu.
- Red circle labeled '2': Points to the 'patNum' input field in the 'Parameters' section.
- Red circle labeled '3': Points to the 'CA_FIRST_NAME' field in the JSON response for patNum 1.
- Red circle labeled '4': Points to the 'patNum' input field in the 'Parameters' section.
- Red circle labeled '5': Points to the 'CA_FIRST_NAME' field in the JSON response for patNum 2.

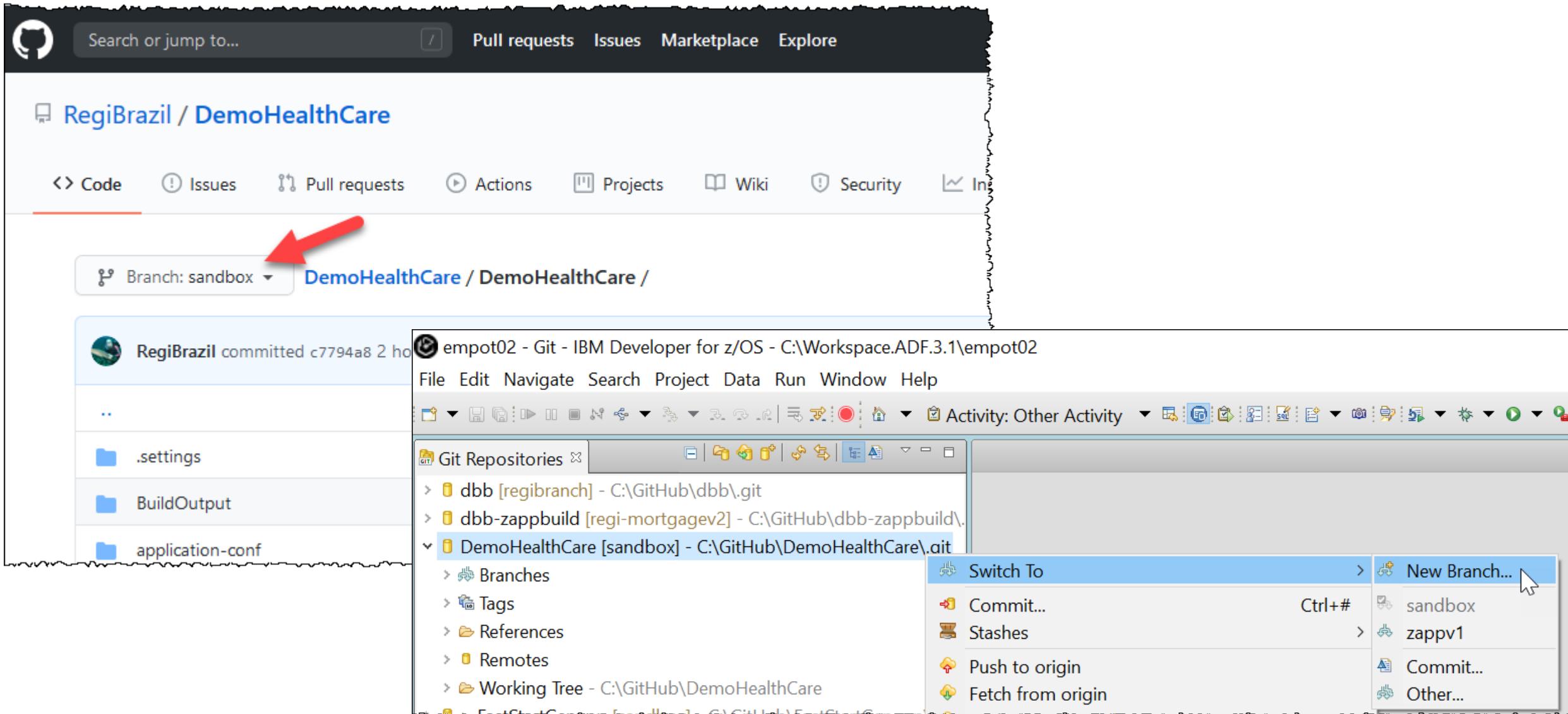
PART #1 – Unit test on COBOL/CICS program HCIPDB01

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>

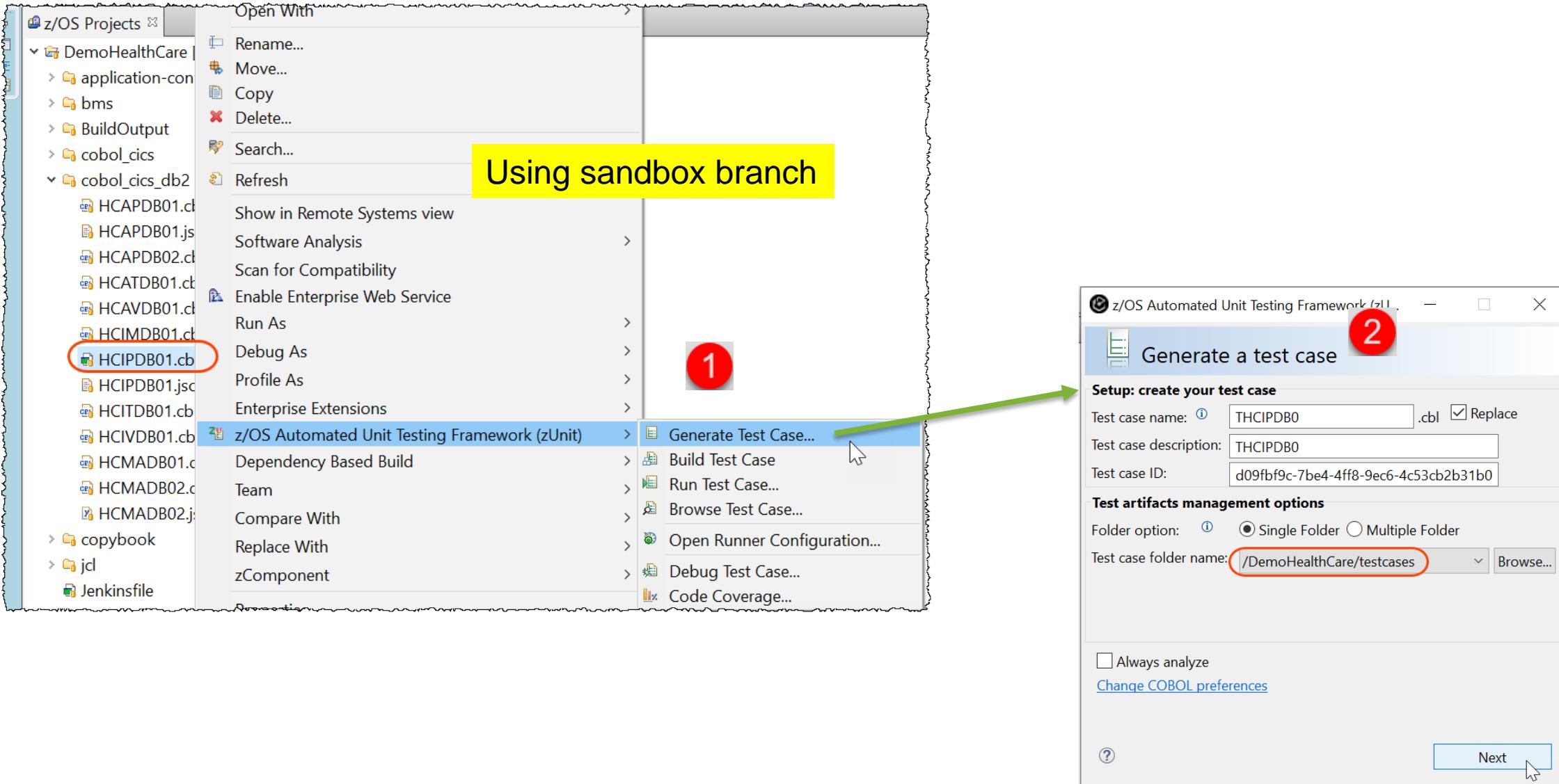


PART #1 – Unit test on COBOL/CICS program HCIPDB01

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

The screenshot illustrates the workflow for recording CICS program interactions using the z/OS Automated Unit Testing Framework and the IDz tool.

Top Left: A window titled "Record data for test case" shows instructions for recording data from a CICS program. It includes a "Start recording" button and a "Transactions to include" field containing an asterisk (*). A red arrow points from this window to the "Response body" of the first API call.

Top Right: A table titled "Value of Selected Field" displays recorded data for "TEST2". The table has columns for Level, Name, PIC, USAGE, TEST2 :Input, and TEST2 :Expected. A row for "CA-FIRST-NAME" is highlighted, showing "Ralph" in the input column and "DALmeida" in the expected column. A red circle labeled "1" is placed over the "TEST2" entry in the "Tests" column of the table.

Middle Left: An API call response body for "patNum": 1 is shown. It contains a JSON object with fields like "CA_REQUEST_ID", "CA_PATIENT_REQUEST", and "CA_FIRST_NAME" (highlighted in red). A red circle labeled "1" is placed over the "patNum" value in the "Parameters" section.

Middle Right: A table titled "Value of Selected Field" displays recorded data for "TEST3". The table has columns for Level, Name, PIC, USAGE, TEST3 :Input, and TEST3 :Expected. A row for "CA-FIRST-NAME" is highlighted, showing "John" in the input column and "Smith" in the expected column. A red circle labeled "2" is placed over the "TEST3" entry in the "Tests" column of the table.

Bottom Left: An API call response body for "patNum": 2 is shown. It contains a JSON object with fields like "CA_REQUEST_ID", "CA_PATIENT_REQUEST", and "CA_FIRST_NAME" (highlighted in red). A red circle labeled "2" is placed over the "patNum" value in the "Parameters" section.

Bottom Right: A table titled "Value of Selected Field" displays recorded data for "TEST2". The table has columns for Level, Name, PIC, USAGE, TEST2 :Input, and TEST2 :Expected. A row for "CA-FIRST-NAME" is highlighted, showing "John" in the input column and "Smith" in the expected column. A red circle labeled "2" is placed over the "TEST2" entry in the "Tests" column of the table.

2. Record CICS program interaction using IDz ...

The screenshot illustrates the workflow for recording CICS program interactions using the z/OS Automated Unit Testing tool.

Record data for test case: This window shows the steps to record data for a test case. Step 1 (red circle) points to the "Start recording" button. Step 2 (red circle) points to the "Stop recording" button. Step 3 (red circle) points to the "Playback file" input field where "JENKINS.ZUNIT.PB.HCIPDB01" is entered.

Filter Recorded Data: This window displays a task list of recorded data. Step 1 (green arrow) points to the "Task Numbr..." column header. Step 2 (red circle) points to the "Import Selections" button. Step 4 (red circle) points to the "Level" dropdown menu.

Export Playback File: This window shows the connection details and playback file selection. Step 3 (red circle) points to the "Update" button. Step 4 (red circle) points to the "Playback file" input field containing "JENKINS.ZUNIT.PB.HCIPDB01".

inquirepatient: This window shows the recorded data in a grid format. A red box highlights the data for patient ID 9627811234, which includes records for CA-THRESHOLD, CA-HR-THRSHLD, CA-BP-THRSHLD, CA-MS-THRSHLD, CA-ADDITIONAL, CA-VISIT-RECORD, CA-VISIT-DURATION, CA-VISIT-TIME, CA-HEART-RATE, CA-BLOOD-SUGAR, CA-MENTAL-STATUS, and CA-ADDITIONAL. The "EXEC CICS ABEND" section shows a record for DFHEIBLK with line number 81.

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

Developer A

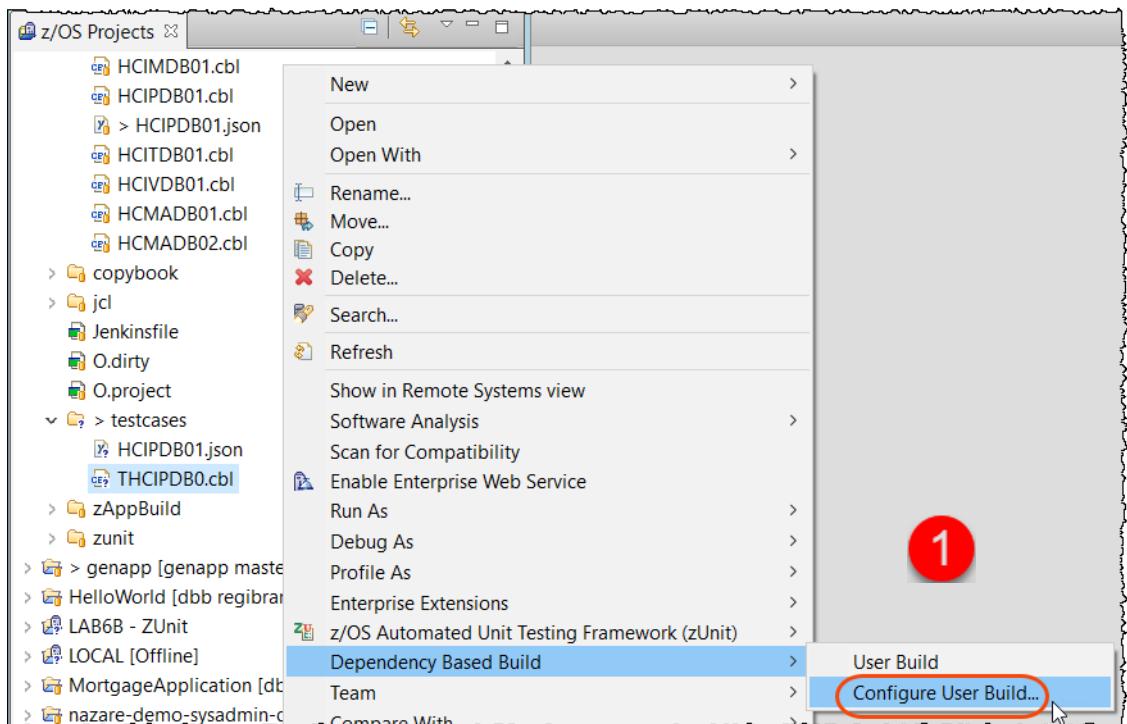
1. Load the COBOL code from **Git** (*GitHub*)
2. Before any change, developer records CICS program interactions using **IDz**
3. Generate, build and run the **zUnit** test
Compile and link-edit the generated unit test programs using **IBM DBB**, followed by running the unit test.
4. Modify the program 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..

The screenshot displays three windows related to generating test cases:

- Top Window (z/OS Automated Unit Testing):** Shows a table of tests (TEST2, TEST3) and their details. A red circle labeled "1" highlights the "Generate" button in the toolbar.
- Middle Window (z/OS Automated Unit Testing):** A dialog box titled "Generate test case programs". It lists "Programs to generate as test cases or stubs": "PROCEDURE DIVISION" (Type: Test Case). Under "Generation steps:", the radio button "Generate test case" is selected (red circle labeled "2"). Other options include "Generate and build test case" and "Generate, build, and run test case".
- Bottom Window (Code Editor):** Displays a COBOL source code file named "THCIPDB0.cbl". The code includes sections for IDENTIFICATION DIVISION, DATA DIVISION, and WORKING-STORAGE SECTION. A red circle labeled "3" highlights the "TEST_TEST3" entry in the IDENTIFICATION DIVISION. The code also contains several program definitions at the bottom.

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



DBB User Build

Configure User Build Operation

Specify information for user build operation.

Select the z/OS system to use:
zos.dev

Select the build script to use:
\\DemoHealthCare\\zAppBuild\\build.groovy

Enter the build sandbox folder:
/var/dbb/work_github

Enter the log file location:
/var/dbb/work_github/work

Enter the build destination HLQ:
JENKINS.HEALTH

Use this configuration as the project default

DBB User Build Information

DBB User Build is complete for THCIPDB0.CBL.

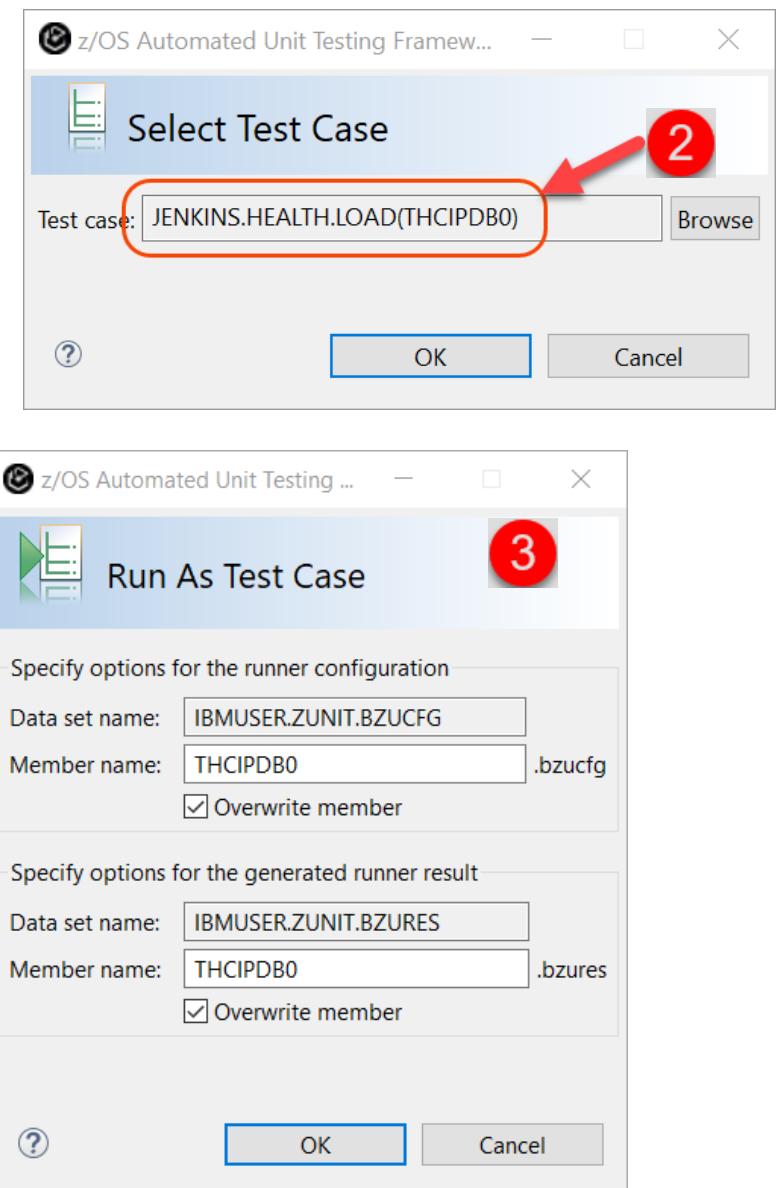
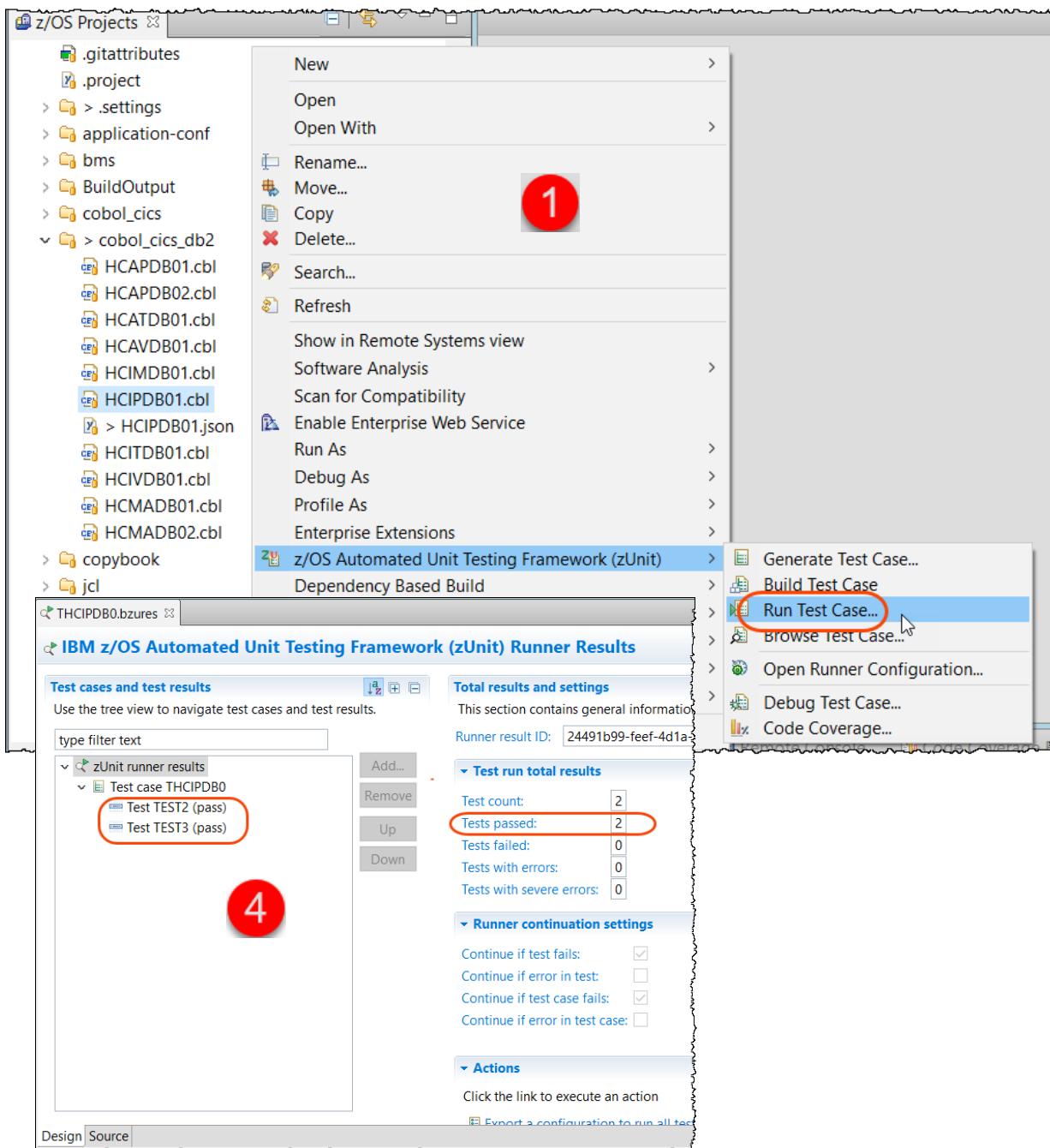
Click Details to see the complete build log.

OK << Details Save Log... Show Build Report

SAVE OPERATION SUMMARY:

MEMBER NAME	THCIPDB0
LOAD LIBRARY	JENKINS.HEALTH.LOAD
PROGRAM TYPE	PROGRAM OBJECT FORMAT 5 OS COMPAT LEVEL z/OS V1R8)
VOLUME SERIAL	C2SYS1
DISPOSITION	REPLACED

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...

The screenshot illustrates the workflow for modifying a COBOL program and running a unit test.

Top Panel: Shows the z/OS Projects view and the HCIPDB01.cbl source editor. A red circle labeled **1** points to the line of code: `* %bug -- the line below will introduce a BUG Jan 02 2020`. Below it, a section of code is highlighted with a red box:

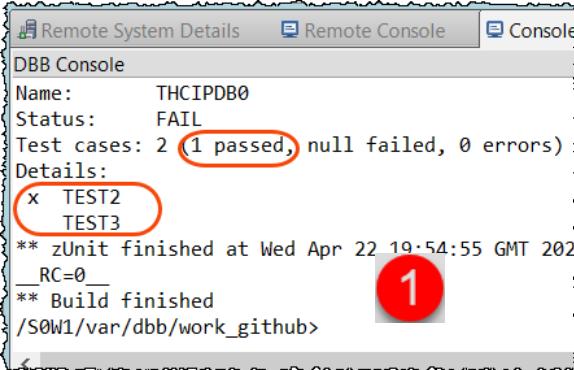
```
145 MOVE '01' TO CA-RETURN-CODE
146 When -913
147 MOVE '01' TO CA-RETURN-CODE
148 When Other
149 MOVE '90' TO CA-RETURN-CODE
150 PERFORM WRITE-ERROR-MESSAGE
151 EXEC CICS RETURN END-EXEC
152 END-Evaluate.
153 * %bug -- the line below will introduce a BUG Jan 02 2020
154 *
155 IF DB2-PATIENT-ID = 1
156 MOVE "BAD NAME" to CA-FIRST-NAME
157 END-IF
158 *
159 EXIT.
160 *
161 COPY HCERRSPD.
```

Middle Panel: Shows a context menu open over the project tree. A red circle labeled **2** points to the "User Build" option under the "Dependency Based Build" section.

Bottom Panel: Shows the "DBB User Build" configuration dialog. A red circle labeled **3** points to the "Select the build script to use:" field, which contains the value `\DemoHealthCare\zAppBuild\build_zUnit_DB01.groovy`.

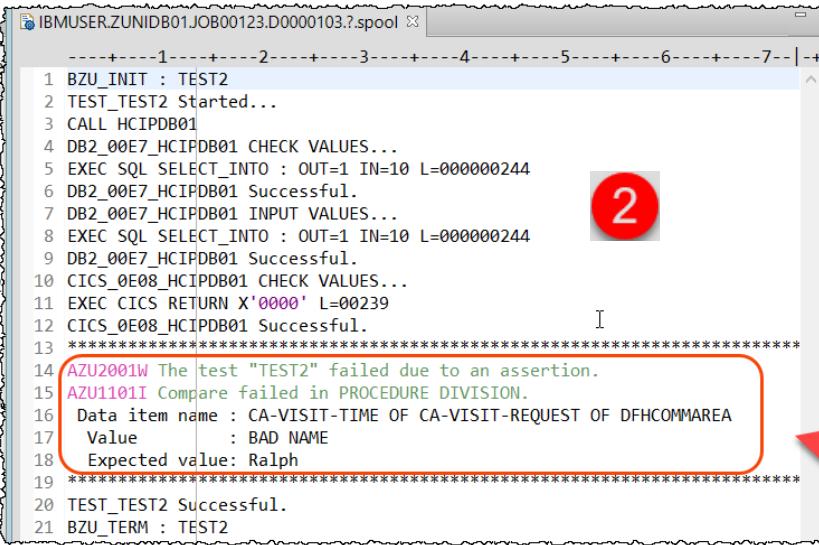
A yellow bar at the bottom right indicates the build command: `Using \DemoHealthCare\zAppBuild\build_zUnit_DB01.groovy`.

4. Modify the program and rerun the unit test



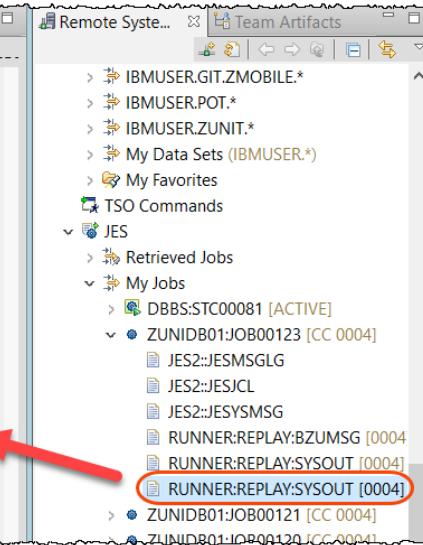
1

DBB Console
Name: THCHIPDB0
Status: FAIL
Test cases: 2 (1 passed, null failed, 0 errors)
Details:
x TEST2
TEST3
** zUnit finished at Wed Apr 22 19:54:55 GMT 2022
_RC=0
** Build finished
/S0W1/var/dbb/work_github>



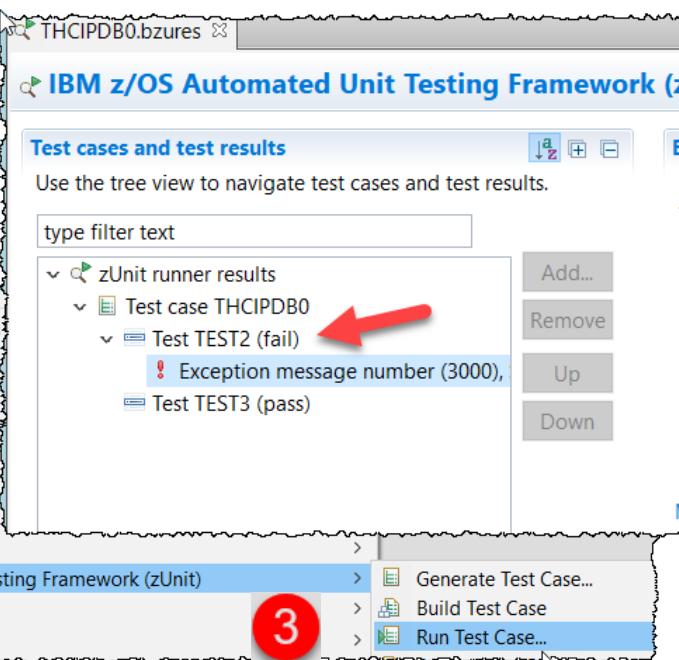
2

```
1 BZU_INIT : TEST2
2 TEST_TEST2 Started...
3 CALL HCIPDB01
4 DB2_00E7_HCIPDB01 CHECK VALUES...
5 EXEC SQL SELECT_INTO : OUT=1 IN=10 L=000000244
6 DB2_00E7_HCIPDB01 Successful.
7 DB2_00E7_HCIPDB01 INPUT VALUES...
8 EXEC SQL SELECT_INTO : OUT=1 IN=10 L=000000244
9 DB2_00E7_HCIPDB01 Successful.
10 CICS_0E08_HCIPDB01 CHECK VALUES...
11 EXEC CICS RETURN X'0000' L=00239
12 CICS_0E08_HCIPDB01 Successful.
13 ****
14 AZU2001W The test "TEST2" failed due to an assertion.
15 AZU1101I Compare failed in PROCEDURE DIVISION.
16 Data item name : CA-VISIT-TIME OF CA-VISIT-REQUEST OF DFHCOMMAREA
17 Value : BAD NAME
18 Expected value: Ralph
19 ****
20 TEST_TEST2 Successful.
21 BZU_TERM : TEST2
```



Remote System... Team Artifacts
IBMUSER.GIT.ZMOBILE.*
IBMUSER.POT.*
IBMUSER.ZUNIT.*
My Data Sets (IBMUSER.*)
My Favorites
TSO Commands
JES
Retrieved Jobs
My Jobs
DBBS:STC0081 [ACTIVE]
ZUNIDB01:JOB00123 [CC 0004]
JES2:JESMSG
JES2:JESJCL
JES2:JESYSMSG
RUNNER:REPLAY:BZUMSG [0004]
RUNNER:REPLAY:SYSOUT [0004]
RUNNER:REPLAY:SYSOUT [0004]
ZUNIDB01:JOB00121 [CC 0004]
ZUNIDB01:JOB00120 [CC 0004]

OR



3

THCHIPDB0.bzures
IBM z/OS Automated Unit Testing Framework (zUnit) Runner Results

Test cases and test results
Use the tree view to navigate test cases and test results.

type filter text

- zUnit runner results
 - Test case THCHIPDB0
 - Test TEST2 (fail)
 - Exception message number (3000),
 - Test TEST3 (pass)

Add... Remove Up Down

Exception details
If the result of the test is "fail" or "error", this section contains the details of the exception.

4

BZPU00W ASSERT=Compare failed in PROCEDU
BZUP220I TEST RUN TEST2 REGISTERED FOR SUBS
BZUP220I TEST RUN TEST3 REGISTERED FOR SUBS
BZUP400I STARTING TRANSACTION= USING PROC
BZPU00W ASSERT=Compare failed in PROCEDU
BZUPT001 ITEM NAME=CA-VISIT-TIME OF CA-VISIT
BZUPT001 VALUE=BAD NAME
BZUPT001 EXPECTED VALUE=Ralph
BZUP002I FINISHED EXECUTION RC=04

Message:

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

Developer A

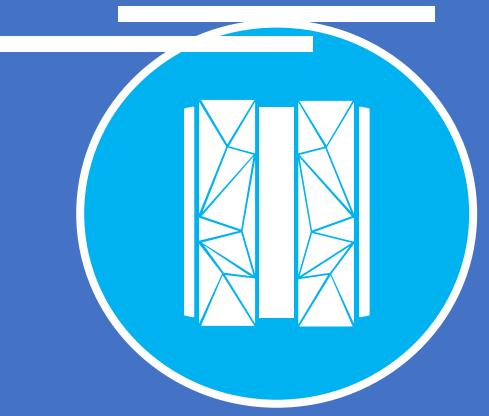
1. Load the COBOL code from **Git** (*GitHub*)
2. Before any change, developer records CICS program interactions using **IDz**.
3. Generate, build and run the **zUnit** test
Compile and link-edit the generated unit test programs using **IBM DBB**, followed by running the unit test.
4. Modify the program 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

IBM z/OS Automated Unit Testing Framework (zUnit) Runner Results

Test cases and test results

Use the tree view to navigate test cases and test results.

type filter text

zUnit runner results

- Test case THCPDB0
 - Test TEST2 (fail)
 - Exception message number (3000), S
 - Test TEST3 (pass)

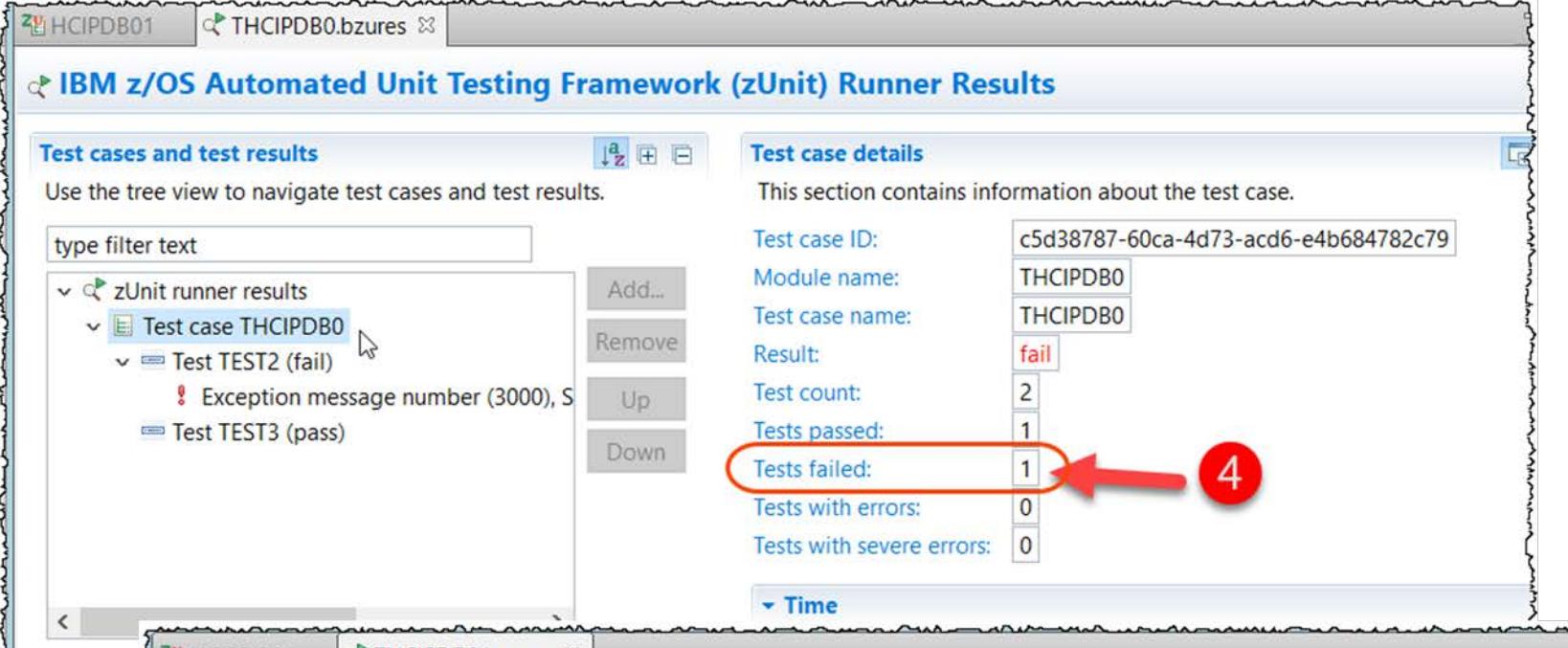
Add... Remove Up Down

Test case details

This section contains information about the test case.

Test case ID: c5d38787-60ca-4d73-acd6-e4b684782c79
Module name: THCPDB0
Test case name: THCPDB0
Result: fail
Test count: 2
Tests passed: 1
Tests failed: 1
Tests with errors: 0
Tests with severe errors: 0

Time



IBM z/OS Automated Unit Testing Framework (zUnit) Runner Results

Test cases and test results

Use the tree view to navigate test cases and test results.

type filter text

zUnit runner results

- Test case THCPDB0
 - Test TEST2 (fail)
 - Exception message number (3000), Severity (4), Component code (BZU)
 - Test TEST3 (pass)

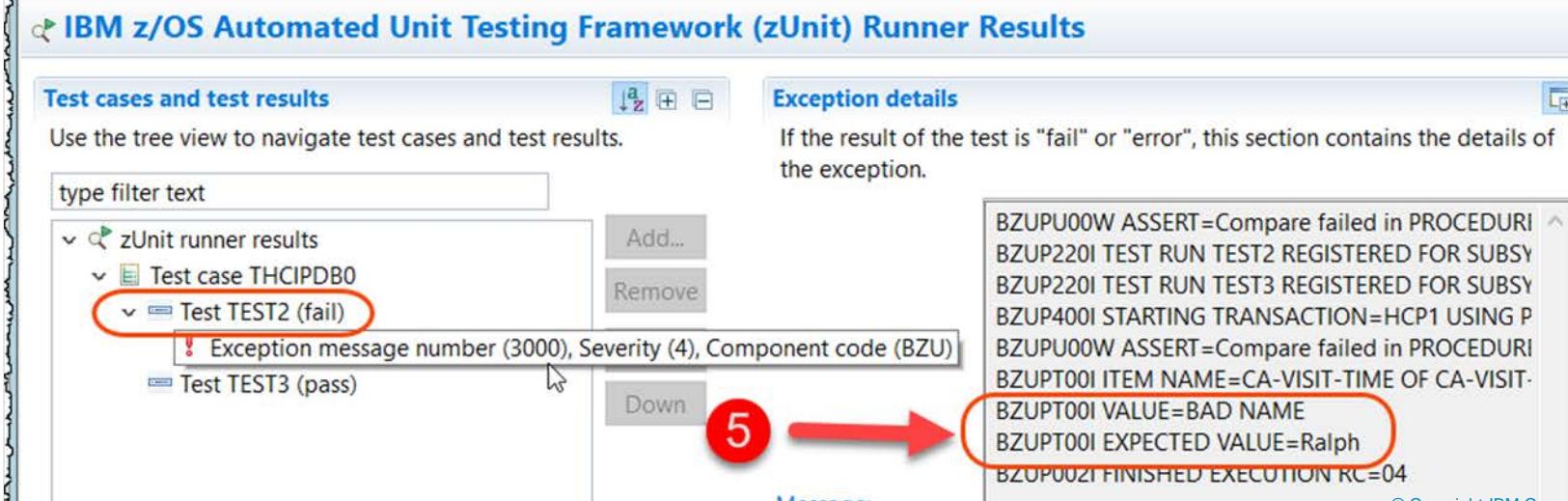
Add... Remove Down

Exception details

If the result of the test is "fail" or "error", this section contains the details of the exception.

BZUPU00W ASSERT=Compare failed in PROCEDURE
BZUP220I TEST RUN TEST2 REGISTERED FOR SUBSY
BZUP220I TEST RUN TEST3 REGISTERED FOR SUBSY
BZUP400I STARTING TRANSACTION=HCP1 USING P
BZUPU00W ASSERT=Compare failed in PROCEDURE
BZUPT00I ITEM NAME=CA-VISIT-TIME OF CA-VISIT-
BZUPT00I VALUE=BAD NAME
BZUPT00I EXPECTED VALUE=Ralph
BZUPU00I FINISHED EXECUTION RC=04

Message:

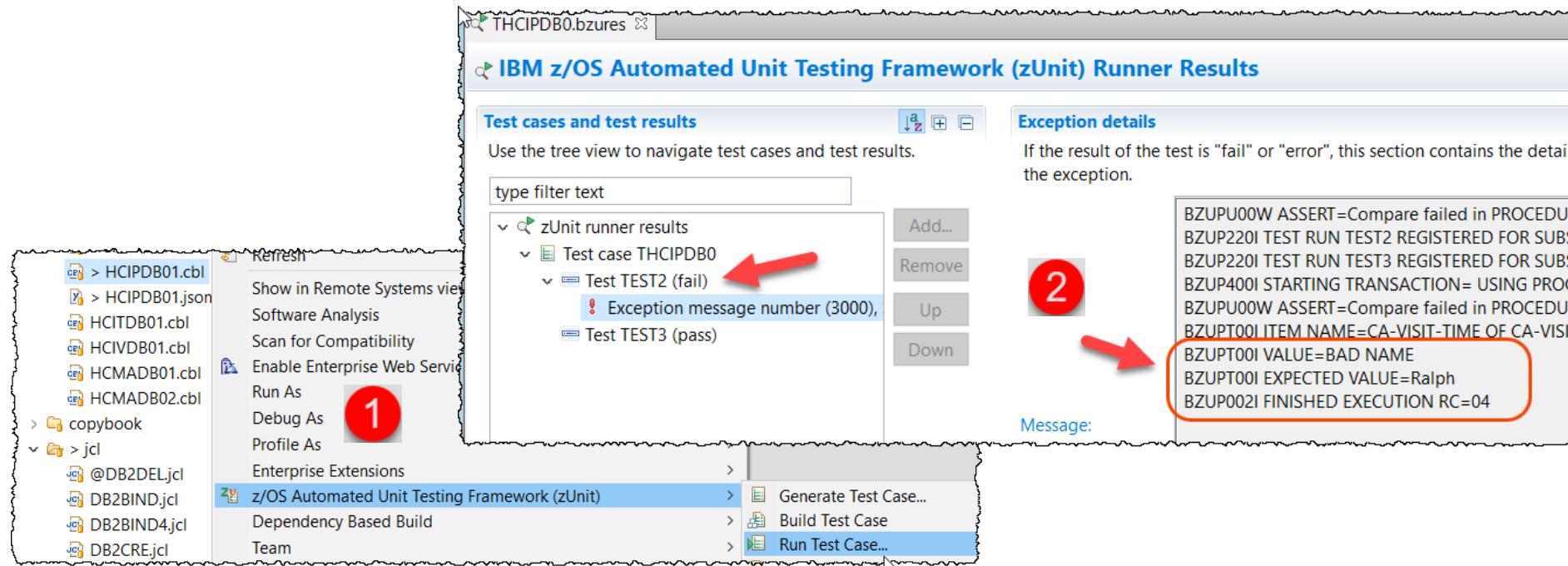


PART #2 – Fix COBOL/CICS HCIPDB01 broken program

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

The screenshot shows the 'z/OS Projects' interface. On the left, under 'z/OS Projects', there is a tree view with nodes: 'cobol_cics', 'copybook', and 'jcl'. Within the 'jcl' node, several JCL files are listed: '@DB2DEL.jcl', 'DB2BIND.jcl', 'DB2BIND4.jcl', 'DB2CRE.jcl', 'ZUNIDB01.jcl' (highlighted with a red circle), 'ZUNITCC.jcl', and 'ZUNITRUN.jcl'. In the center, the 'ZUNIDB01.jcl' file is open in a code editor. The code contains the following JCL:

```
1 //ZUNIDB01 JOB ,  
2 // MSGCLASS=H,MSGLEVEL=(1,1),REGION=0M,COND=(  
3 ///* Action: Run Test Case... for HCIPDB01  
4 ///* Source: IBMUSER.DBB.LOAD(THCIPDB0)  
5 //RUNNER EXEC PROC=BZUPPLAY,  
6 // BZUCFG=IBMUSER.ZUNIT.BZUCFG(THCIPDB0),  
7 // BZUCBK=IBMUSER.DBB.LOAD,  
8 // BZULOD=IBMUSER.DBB.LOAD,  
9 // PARM=('STOP=E,REPORT=XML')  
10 //BZUPLAY DD DUMMY  
11 //BZURPT DD DISP=SHR,  
12 // DSN=IBMUSER.ZUNIT.BZURES(THCIPDB0)
```

On the right, the 'IBMUSER.ZUNIDB01.JOB00127.D0000103.?spool' window shows the execution log. It includes:

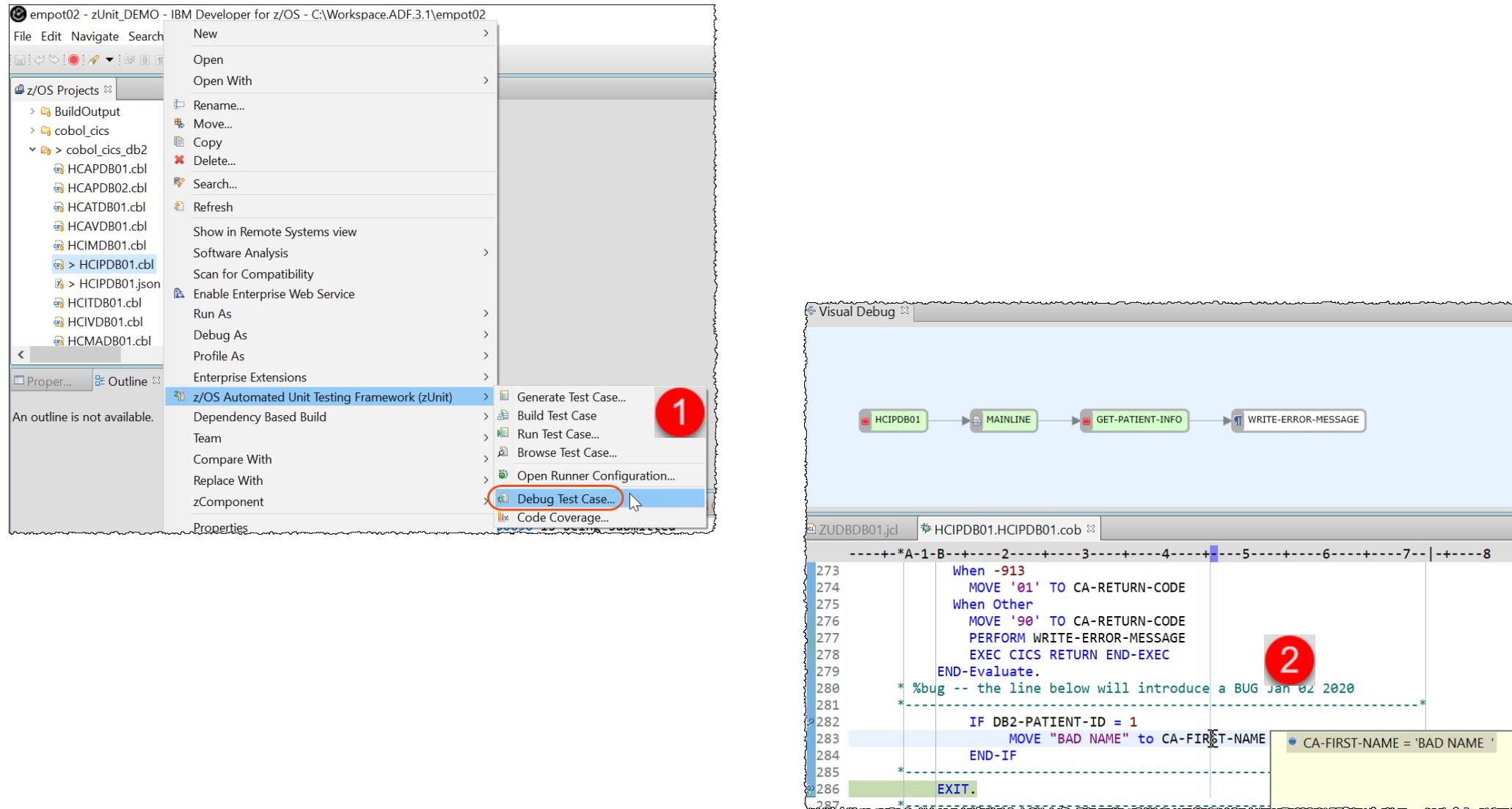
```
1 BZU_INIT : TEST2  
2 TEST_TEST2 Started...  
3 CALL HCIPDB01  
4 DB2_00E7_HCIPDB01 CHECK VALUES...  
5 EXEC SQL SELECT_INTO : OUT=1 IN=10 L=000000244  
6 DB2_00E7_HCIPDB01 Successful.  
7 DB2_00E7_HCIPDB01 INPUT VALUES...  
8 EXEC SQL SELECT_INTO : OUT=1 IN=10 L=000000244  
9 DB2_00E7_HCIPDB01 Successful.  
10 CICS_0E08_HCIPDB01 CHECK VALUES...  
11 EXEC CICS RETURN X'0000' L=00239  
12 CICS_0E08_HCIPDB01 Successful.  
13 *****  
14 AZU2001W The test "TEST2" failed due to an assertion.  
15 AZU1101I Compare failed in PROCEDURE DIVISION.  
16 Data item name : CA-VISIT-TIME OF CA-VISIT-REQUEST OF DFHCOMMAREA  
17 Value : BAD NAME  
18 Expected value: Ralph  
19 *****  
20 TEST_TEST2 Successful.  
21 BZU_TERM : TEST2
```

A red circle highlights the 'Value : BAD NAME' line in the log, and another red circle highlights the 'Expected value: Ralph' line.

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

The screenshot shows a COBOL program in an IDE. The code is as follows:

```
140      END-EXEC.  
141      Evaluate SQLCODE  
142      When 0  
143          MOVE '00' TO CA-RETURN-CODE  
144      When 100  
145          MOVE '01' TO CA-RETURN-CODE  
146      When -913  
147          MOVE '01' TO CA-RETURN-CODE  
148      When Other  
149          MOVE '90' TO CA-RETURN-CODE  
150          PERFORM WRITE-ERROR-MESSAGE  
151          EXEC CICS RETURN END-EXEC  
152      END-Evaluate.  
153      *%bug -- the line below will introduce a BUG Jan 02 2020  
154      *  
155      IF DB2-PATIENT-ID = 1  
156          MOVE "BAD NAME" to CA-FIRST-NAME  
157      END-IF  
158      *  
159      EXIT.  
160      *
```

A red arrow points to line 153, which contains the comment `*%bug -- the line below will introduce a BUG Jan 02 2020`. A red circle with the number 1 is in the top right corner.

The screenshot shows the same COBOL program after a local copy operation. The code is now:

```
143      MOVE '00' TO CA-RETURN-CODE  
144      When 100  
145          MOVE '01' TO CA-RETURN-CODE  
146      When -913  
147          MOVE '01' TO CA-RETURN-CODE  
148      When Other  
149          MOVE '90' TO CA-RETURN-CODE  
150          PERFORM WRITE-ERROR-MESSAGE  
151          EXEC CICS RETURN END-EXEC  
152      END-Evaluate.  
153      *%bug -- the line below will introduce a BUG  
154      *  
155      IF DB2-PATIENT-ID = 1  
156          MOVE "BAD NAME" to CA-FIRST-NAME  
157      END-IF  
158      *  
159      EXIT.  
160      *
```

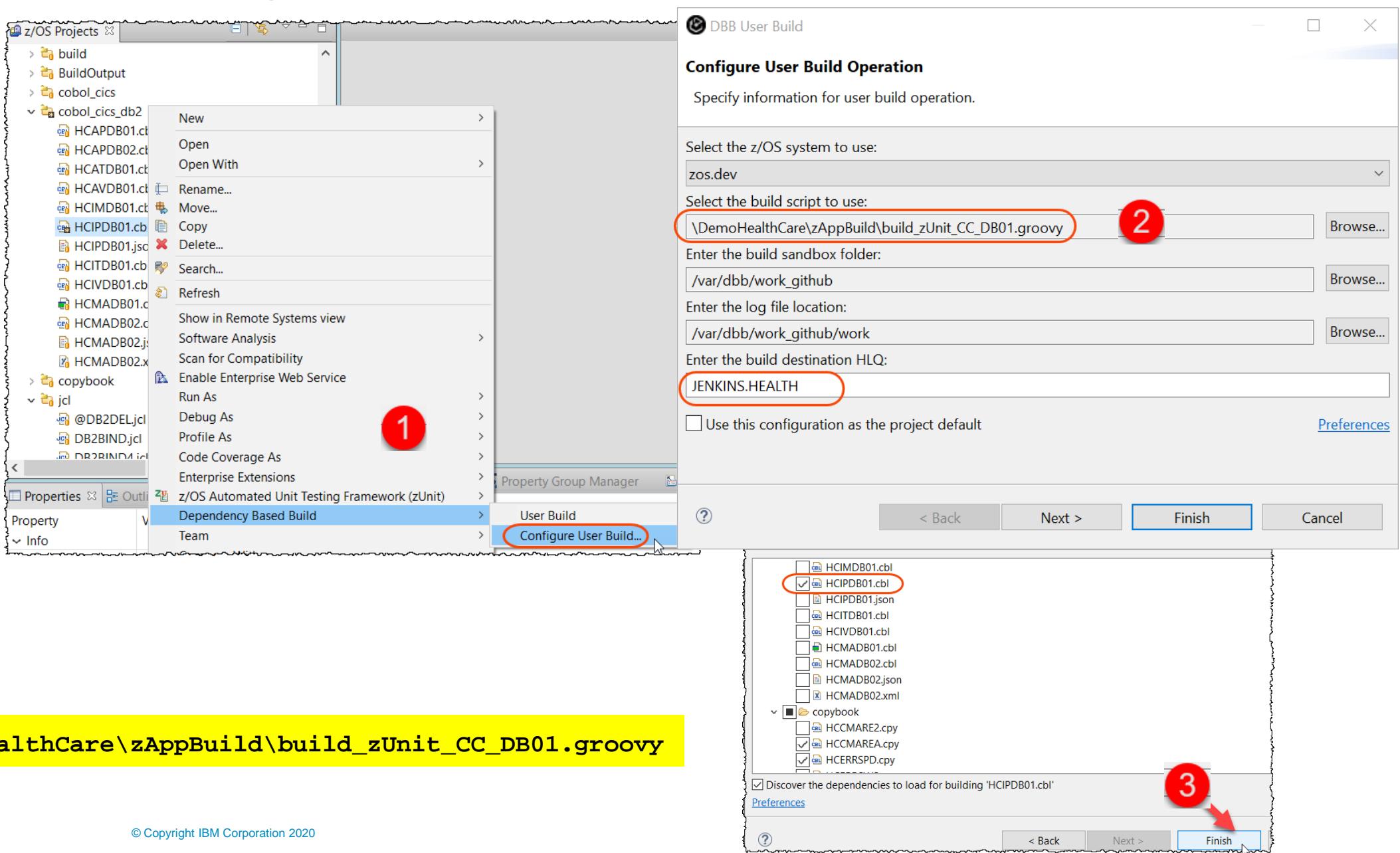
The code block from line 153 to 158 has been copied to a new location starting at line 143. The original code block is still present. A red arrow points to the copied code. A red circle with the number 2 is in the top right corner.

PART #2 – Fix COBOL/CICS HCIPDB01 broken program

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

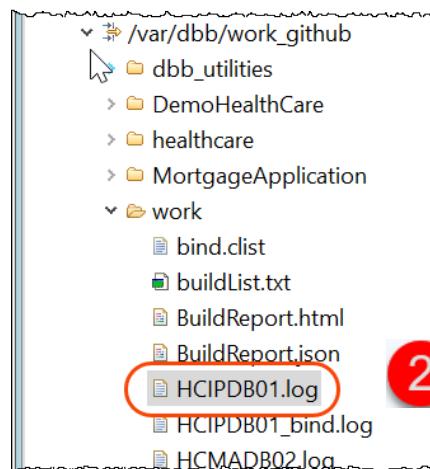
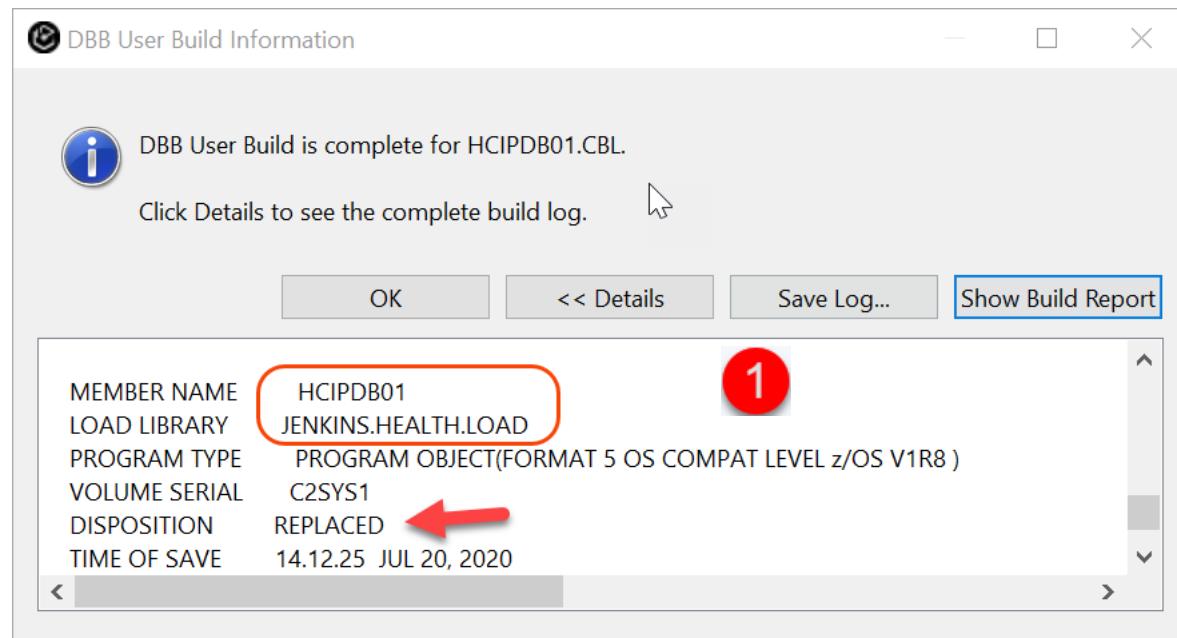
1. Developer **B** run the zUnit test case and verify the bug.
Optionally he could use the Debug (in BATCH) to see the “bug
2. Uses **IDz** to update the COBOL program
- 3. Uses **DBB** to perform a “*User Build*”, run the unit test (**zUnit**),
Code Coverage and verify that the abend is fixed
4. Optionally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



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



The screenshot shows the DBB Console output:

```
DBB Console
/S0W1/var/dbb/work
***** Module [THCIPDB0] *****
Name: THCIPDB0
Status: PASS
Test cases: 2 (2 passed, null failed, 0 errors)
Details:
  TEST2
  TEST3
** zUnit finished at Tue Apr 07 13:43:40 GMT 2020
__RC=0__
** Build finished
/S0W1/var/dbb/work
```

The "Status: PASS" line is highlighted with a red box and circled with a red number 3.

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

The screenshot shows the Rational Application Developer interface with three main windows:

- Code Coverage Report:** A tree view showing coverage for various modules. The **MAINLINE-END** module has 100% coverage, while **MAINLINE-EXIT**, **GET-PATIENT-INFO**, and **WRITE-ERROR-MESSAGE** have 0% coverage. Red arrows point from the report to the status bar of the code editor and the status bar of the results table.
- COBOL Source Editor:** Displays a portion of the **HCIPDB01.HCIPDB01.cob** file. Lines 259 through 266 are highlighted in red, indicating they were not covered by the test. A tooltip "Lines 259-266 not covered." is shown over the code. A red arrow points from the tooltip to the code editor.
- Code Coverage Results:** A table showing the overall coverage status for the workspace. It lists "Compiled Code Coverage Workspace Results" with a status of 0 and 23% coverage. Another row shows "THCIPDBX_2020_04_07_141440_0026" with a status of 0 and 23% coverage. A red arrow points from the status bar of the code editor to the status column of this table.

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*)

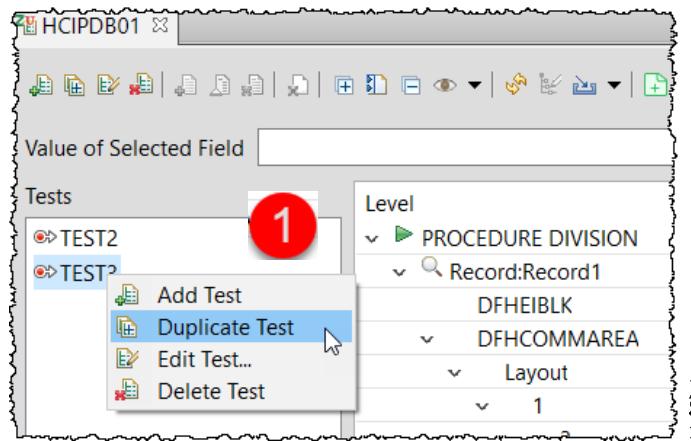
4. Optionally change zUnit test case for better Code Coverage...

```
END-EXEC.  
Evaluate SQLCODE  
When 0  
  MOVE '00' TO CA-RETURN-CODE  
When -913  
  MOVE '01' TO CA-RETURN-CODE  
When Other  
  MOVE '90' TO CA-RETURN-CODE  
  PERFORM WRITE-ERROR-MESSAGE  
  EXEC CICS RETURN END-EXEC
```

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



*HCIPDB01

Value of Selected Field -913

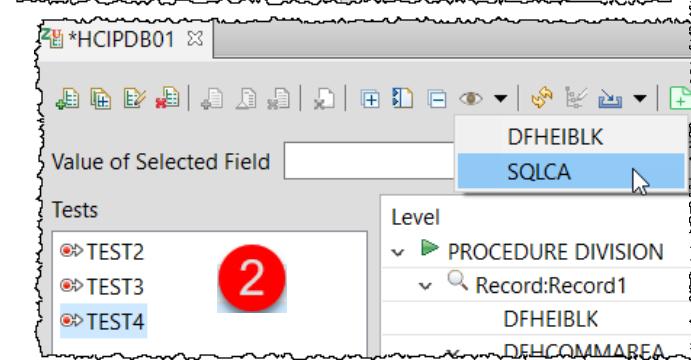
Tests

- TEST2
- TEST3
- TEST4

Layouts

- Test procedure
 - PROCEDURE DIVISION
 - CICS
 - DB2
 - EXEC SQL SELECT INTO [PATIENT]

Level	Name	PIC	USA...	TEST4 :Inp...	TEST4 :Exp...
3	DB2-PATIEN...	S9(9)	BINA...		2
SQLCA					
1	SQLCA				
5	SQLCAID	X(8)	DISP...	SQLCA	
5	SQLABC	S9(9)	BINA...	136	
5	SQLCODE	S9(9)	BINA...	-913	
5	SQLERRM				
5	SQLERRP	X(8)	DISP...	DSN	
5	SQLERRD O...				
5	SQLERRD (1)	S9(9)	BINA...	0	



*HCIPDB01

Value of Selected Field

Tests

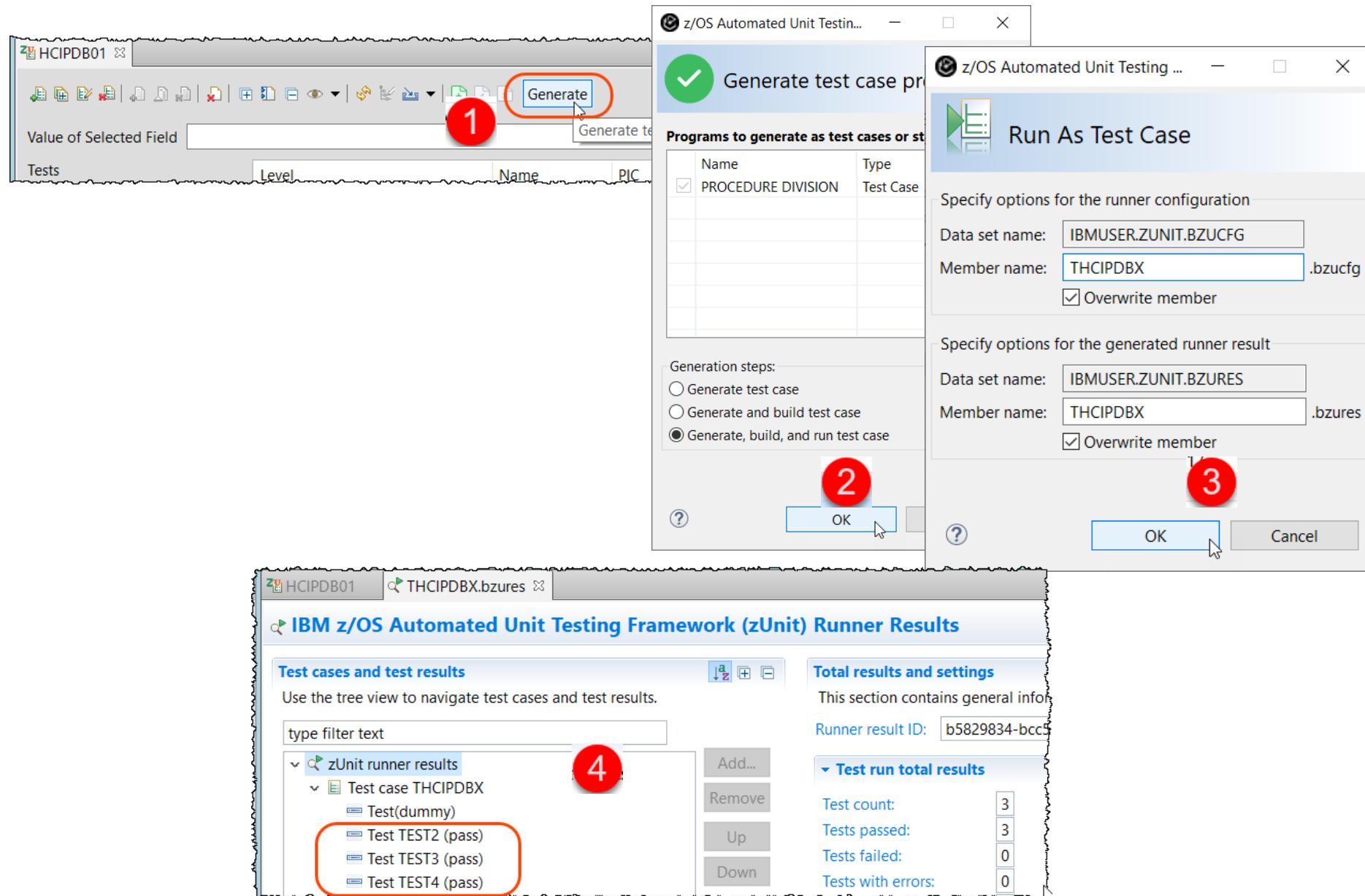
- TEST2
- TEST3
- TEST4

Layouts

- Test procedure
 - PROCEDURE DIVISION
 - CICS
 - DB2
 - EXEC SQL SELECT INTO [PATIENT]

Level	Name	PIC	USAGE	TEST4 :Inp...	TEST4 :Exp...
PROCEDURE DIVISION					
Record:Record1					
DFHEIBLK					
DFHCOMMAREA					
Layout					
1	DFHCOMMAREA				
3	CA-REQUEST-ID	X(6)	DISPLAY	01PAT	
3	CA-RETURN-CODE	9(2)	DISPLAY	0	
3	CA-PATIENT-ID	9(10)	DISPLAY	0000000002	
3	REDEFINES CA-REQ...				

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



4. Optionally change zUnit test case for better Code Coverage

The screenshot illustrates a workflow for improving code coverage through a zUnit test case. It consists of three main panels:

- Panel 1 (Top Left):** Shows the "z/OS Projects" view with several JCL files listed. A red circle labeled "1" highlights the "ZUNITCC.jcl" file.
- Panel 2 (Top Right):** Displays the contents of the "ZUNITCC.jcl" file. A red circle labeled "2" highlights the line: "/* Action: Code Coverage using Test Case..."
- Panel 3 (Bottom):** Shows three "Code Coverage Report" windows side-by-side, each corresponding to a different JCL run:
 - Left Window:** Analyzed on Apr 7, 2020, at 2:14:38 PM. The "WRITE-ERROR-MESSAGE" section shows 0% coverage.
 - Middle Window:** Analyzed on Apr 7, 2020, at 2:14:31 PM. The "WRITE-ERROR-MESSAGE" section shows 0% coverage.
 - Right Window:** Analyzed on Apr 7, 2020, at 2:14:48 PM. The "WRITE-ERROR-MESSAGE" section shows 83% coverage.A callout bubble from the right window points to the middle one with the text: "SQLCODE -913 must have 90 at CA-RETURN-CODE".

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)

The screenshot illustrates the workflow for committing and pushing corrected COBOL code to a GitHub repository.

Top Left: A "COBOL Structure Compare" window shows the "Local: HCIPDB01.cbl" and "Index: HCIPDB01.cbl (editable)" versions of the code. The code contains several lines of COBOL, including a section where a bug was identified and fixed.

Top Right: A "Push Results: DemoHealthCare - origin" dialog box displays the message "Pushed to DemoHealthCare - origin". It lists a single commit: "zappv1 → zappv1 [142d811..b60b0a8] (1)" with the commit message "Fixed Bug Bad Name (RegiBrazil on 2020-04-23 08:01:05)".

Bottom: The main interface shows the "Git Staging" tab selected. The "Commit Message" field contains the text "Fixed BUG Bad Name", which is highlighted with a red oval and a red arrow pointing to it. The "Author:" and "Committer:" fields both show the email "RegiBrazil <rbarosa@us.ibm.com>". A red arrow points from the "Commit and Push..." button at the bottom right towards the commit message field.

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

✓ GenAppNazarePipeline < 75

Branch: master 2m 12s Changes by budy.jy
Commit: edbdc19 6 days ago Push event to branch master

```
graph LR; Start((Start)) --> Init((Init)); Init --> GitClone((Git Clone/Refresh)); GitClone --> DBBBuild((DBB Build)); DBBBuild --> ZUnitTest((ZUnit Test)); ZUnitTest --> SonarQube((SonarQube)); SonarQube --> Deploy((Deploy)); Deploy --> MonitorPrep((Monitor Prep)); MonitorPrep --> IntegrationTests((Integration Tests)); IntegrationTests --> MonitorPost((Monitor Post)); MonitorPost --> End((End))
```

Monitor Post - 5s

Restart Monitor Post

✓ GenAppNazarePipeline < 71

Branch: master 1m 22s Changes by budy.jy
Commit: 3466d28 10 days ago Push event to branch master

```
graph LR; Start((Start)) --> Init((Init)); Init --> GitClone((Git Clone/Refresh)); GitClone --> DBBBuild((DBB Build)); DBBBuild --> ZUnitTest((ZUnit Test)); ZUnitTest --> SonarQube((SonarQube)); SonarQube --> Deploy((Deploy)); Deploy --> MonitorPrep((Monitor Prep)); MonitorPrep --> IntegrationTests((Integration Tests)); IntegrationTests --> MonitorPost((Monitor Post)); MonitorPost --> End((End))
```

Deploy - 12s

Restart Deploy

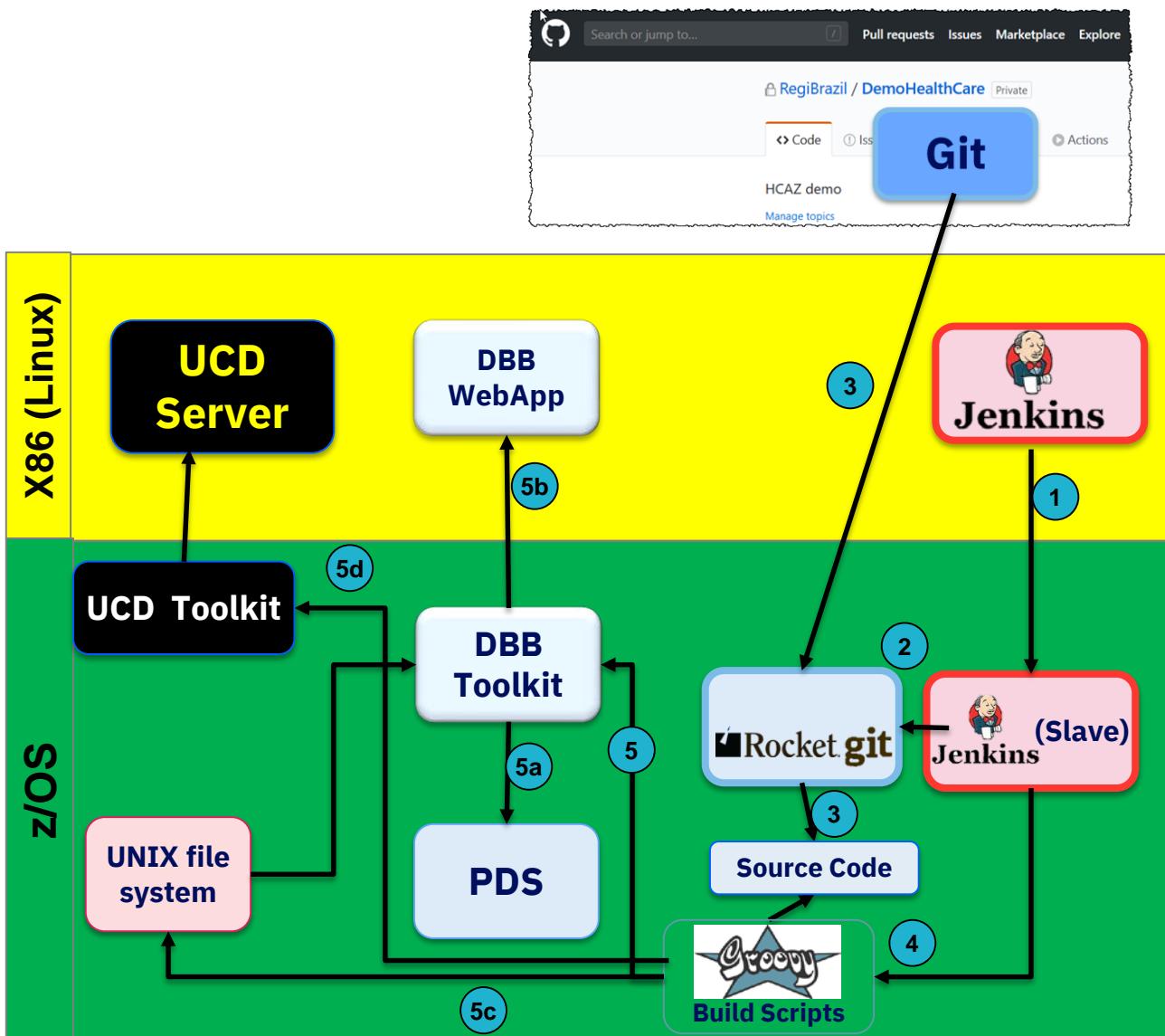
✓ > Shell Script 3s

✗ > Publish Artifacts to IBM UrbanCode Deploy 8s

```
1 Deploying component versions '{GenApp=[latest]}'
2 Starting deployment process 'DeployGenApp' of application 'GenApp-Deploy' in environment 'Z-QA'
3 Deployment request id is: '16a36188-8f8d-2046-b74d-d424ef7e167b'
4 Deployment is running. Waiting for UCD Server feedback.
5 Deployment has failed due to IOException Deployment process failed with result FAULTED
```

✓ > Send Slack Message <1s

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**.

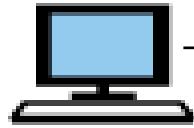
[5c] Copy logs from **PDS** to **zFS**, generate build reports (can be saved in build result).

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

URBancode DEPLOY FOR DEPLOYMENT AUTOMATION

DEPLOYING TO MAINFRAME

Develop



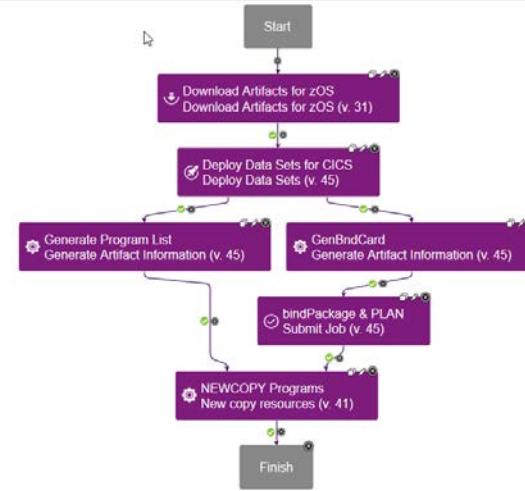
IDE
or
TSO/ISPF



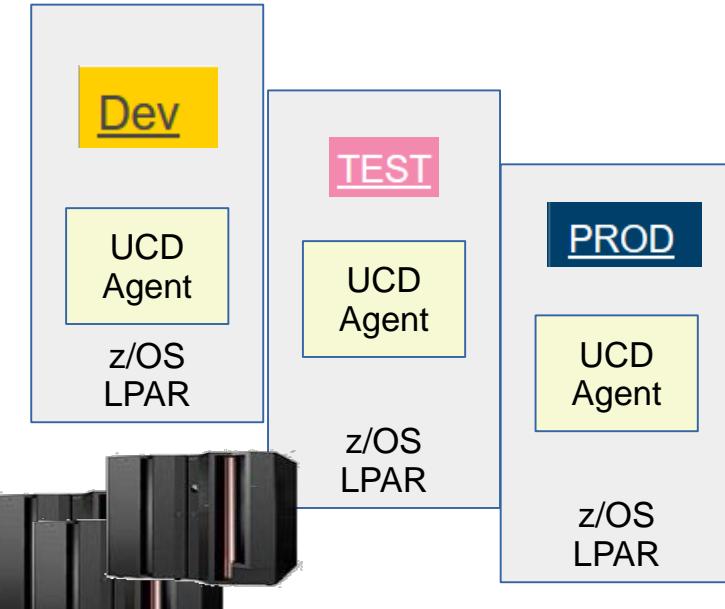
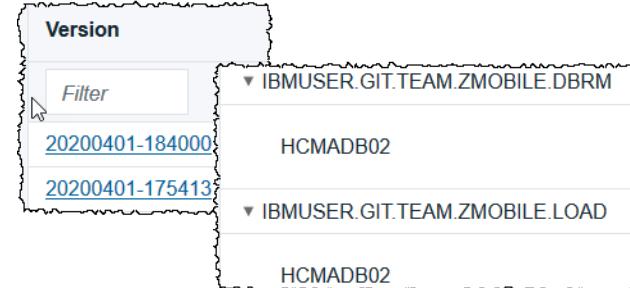
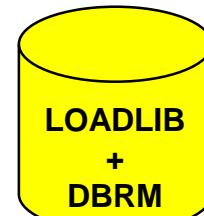
Build



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



IBM UrbanCode Deploy



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

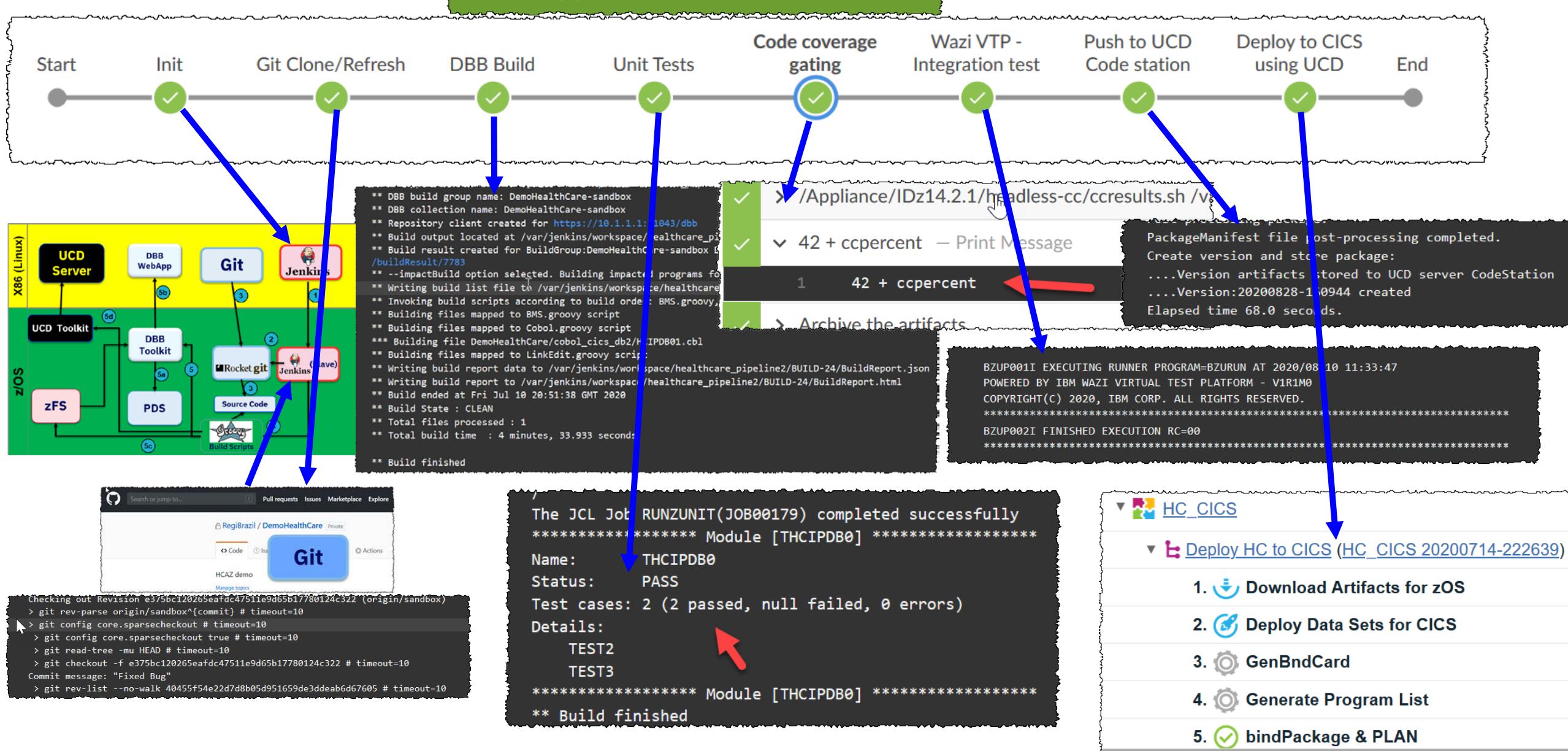
Rebecca – Release Engineer

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

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

healthcare_pipeline3_DemoHealthCare_Wazi < 32



* 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**

The screenshot shows two instances of the DBB browser interface. The left instance displays a list of collections, with 'DemoHealthCare-zappv1' expanded to show a list of build artifacts. Two specific builds are highlighted with orange circles: 'build.20200415.110739.007' at the bottom of the list and 'build.20200410.124708.047' near the top. The right instance shows the detailed build result for 'build.20200415.110739.007'. It includes sections for 'Build Report', 'Toolkit Version', 'Build Summary', and a table of build details.

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	IGYCRCTL	4	IBMUSER.GIT.TEAM.ZMOBILE.COBOLO(HCIPDB01)	IBMUSER.GIT.TEAM.ZMOBILE.DBRM(HCIPDB01)	DBRM
	<small>DemoHealthCare/copybook/HCERRSPD.cpy COPY DemoHealthCare/copybook/HCERRSWS.cpy COPY DemoHealthCare/copybook/HCCMAREA.cpy SQL INCLUDE DemoHealthCare/copybook/SQLC.A.cpy SQL INCLUDE</small>				IBMUSER.GIT.TEAM.ZMOBILE.LOAD(HCIPDB01)	LOAD
	<small>Hide Dependencies</small>	IEWBLINK	0			

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					
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	<div style="width: 100%;">1 / 1</div>	7:13:27 AM	0:04:57	Success
▼	HC_CICS	<div style="width: 100%;">1 / 1</div>	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		<div style="width: 100%;">1 / 1</div>	7:13:27 AM	0:04:57	Success

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

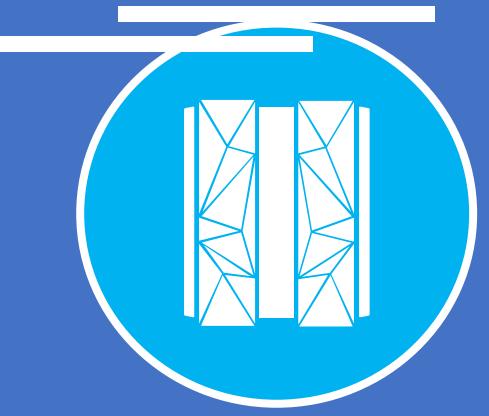
Rebecca – Release Engineer

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)



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 User id →  empot05 and password → empot05

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**

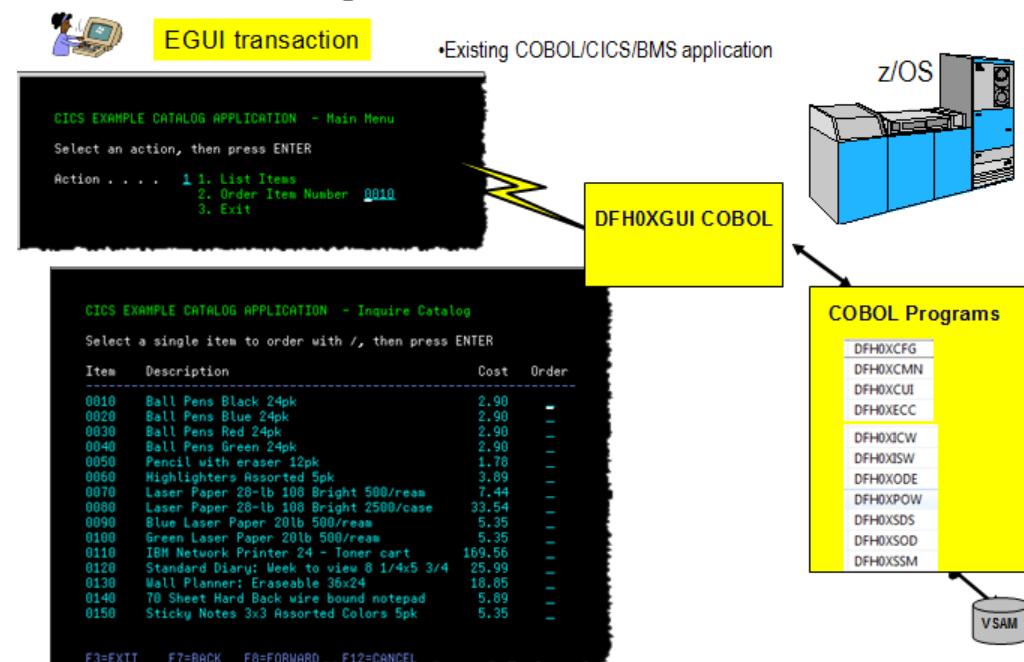
→ You will modify the program under test, rerun the unit test, and observe the failure of the test case.

- 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:..



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 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

© 2018 IBM Corporation

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

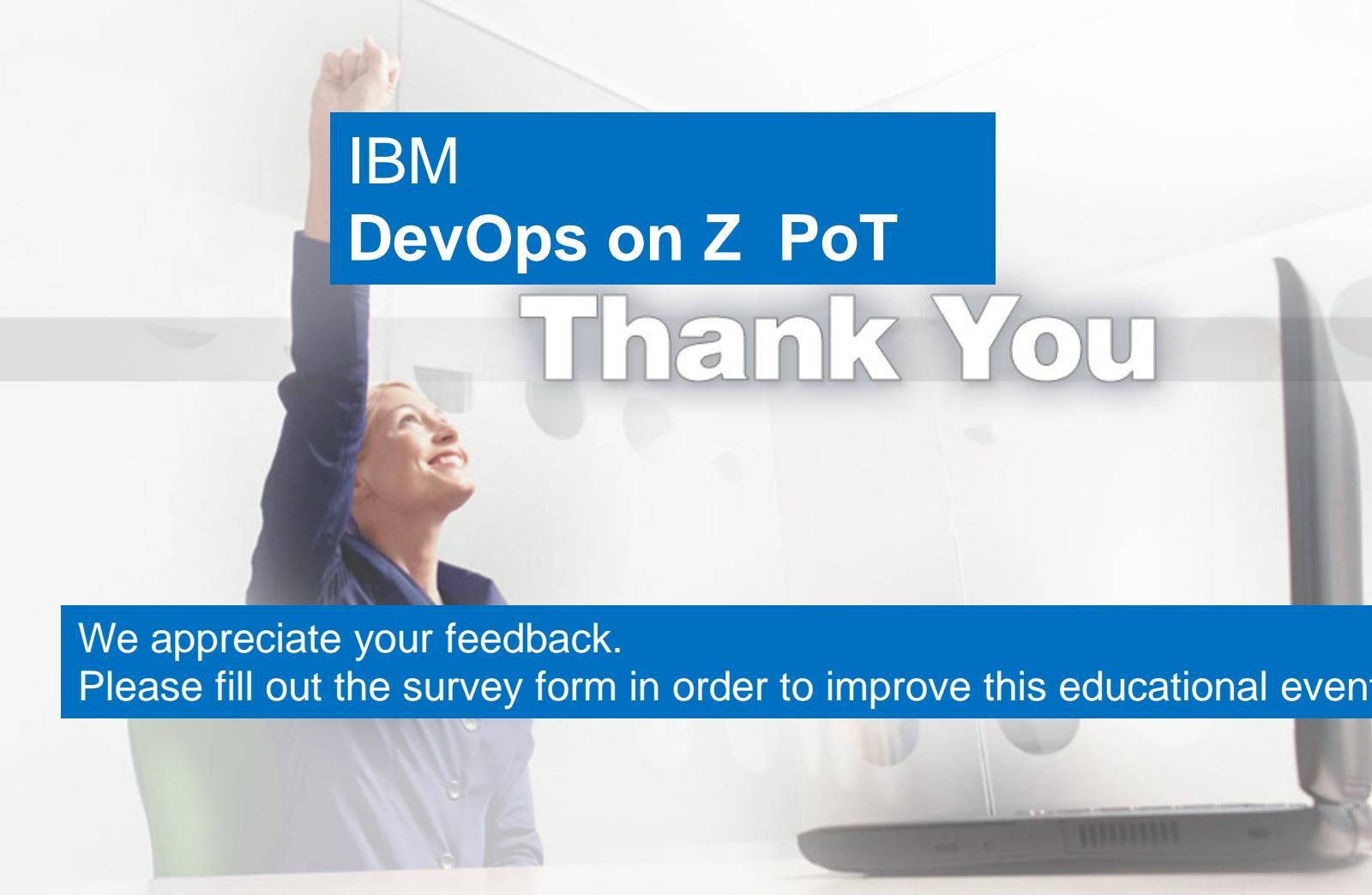
Training Module 9 - Debug and Code Coverage **Note September 14th date**

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

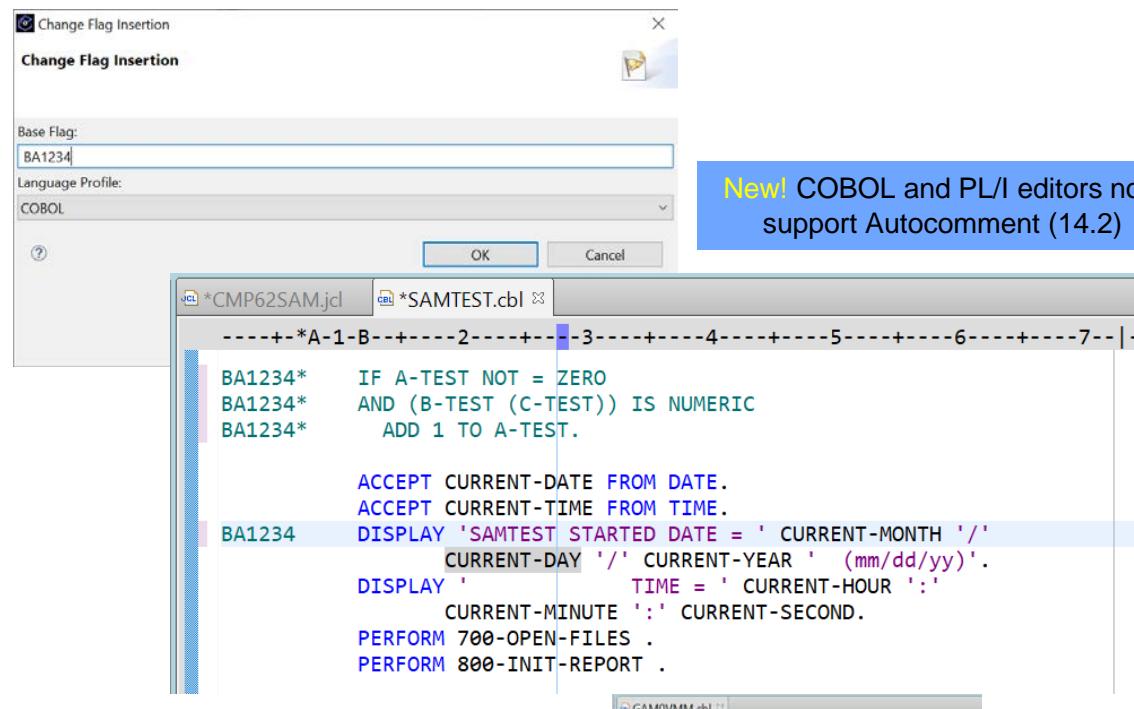
- Consistent tools across the enterprise.
 - Better collaboration.
 - Developer flexibility.
- Distributed teams have already figured out much of the process & culture issues.
- Easier on-boarding of new z developers.
- Better integration using open source.
- Elimination of administrative work.

Why Git

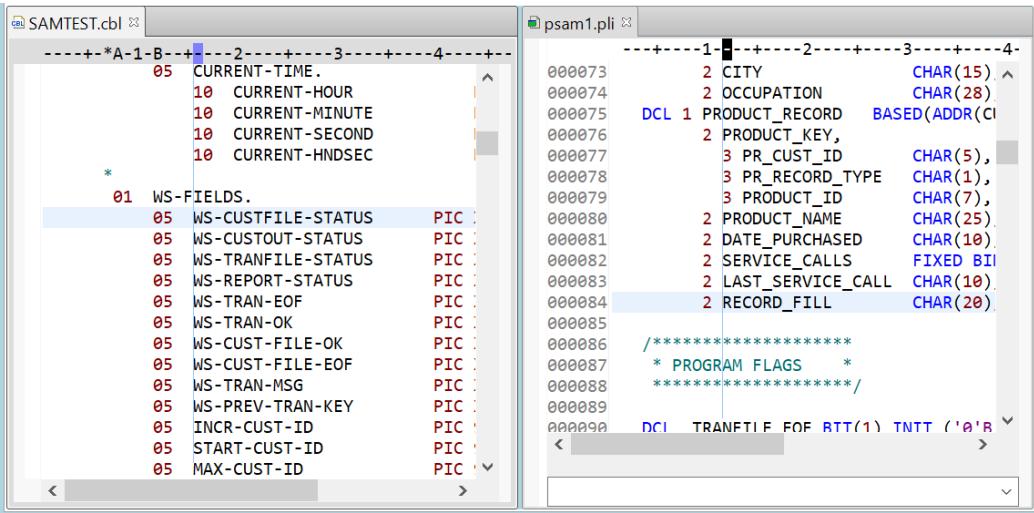
- Git is a fast and modern implementation of version control
 - Very lightweight
 - Extremely fast - The tool just gets out of the way
- Git is Distributed
 - Every developer has full copy, with history
 - No single point of failure
- Git facilitates collaborative changes to files
 - Enables branch-based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

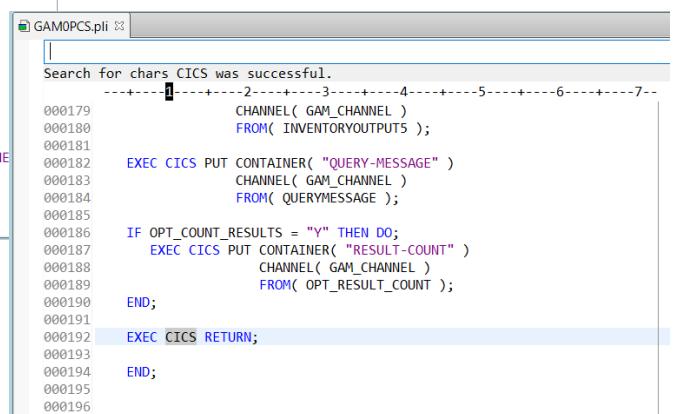
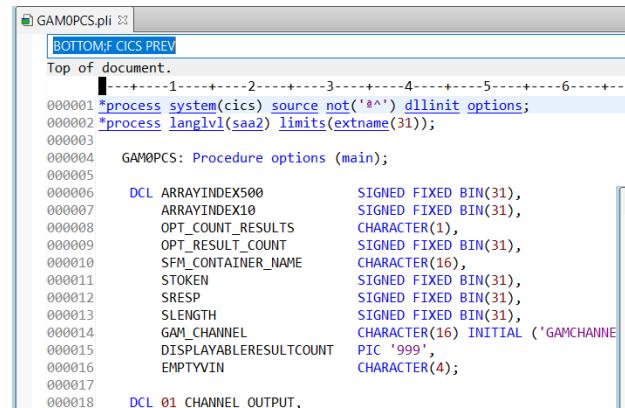
Analyze and Edit



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

Building vs. Compile/Link

