

DevOps on Z PoT

Proof of Technology

DevOps Solution for Z Development

David Hawreluk
dhawrel@us.ibm.com

Regi Barosa
rbarosa@us.ibm.com

Ronald Geraghty
Ronald.Geraghty@ibm.com

Wilbert Kho
wilbert@us.ibm.com

Sept-23-2021

Audience

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

Pre-requisites

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

Proof of Technology Objectives

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

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

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

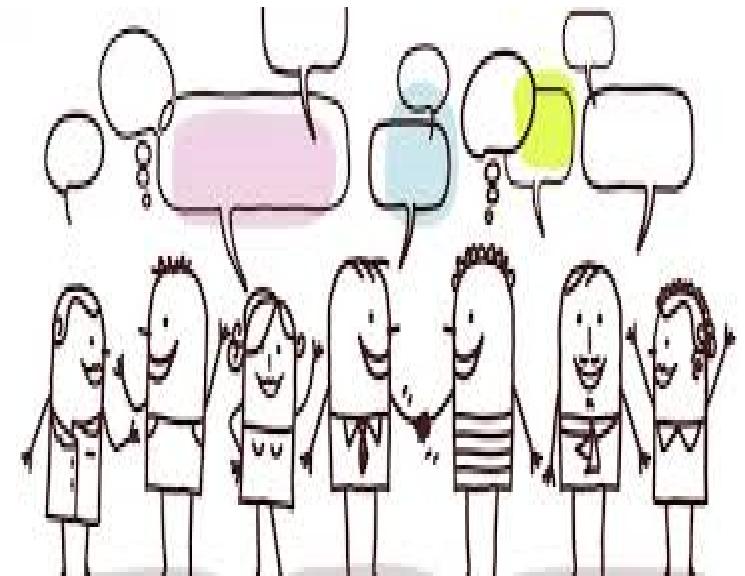
Areas covered:

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



Few introductions (15 min)...

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



1 minute Please....

Agenda - Day 1

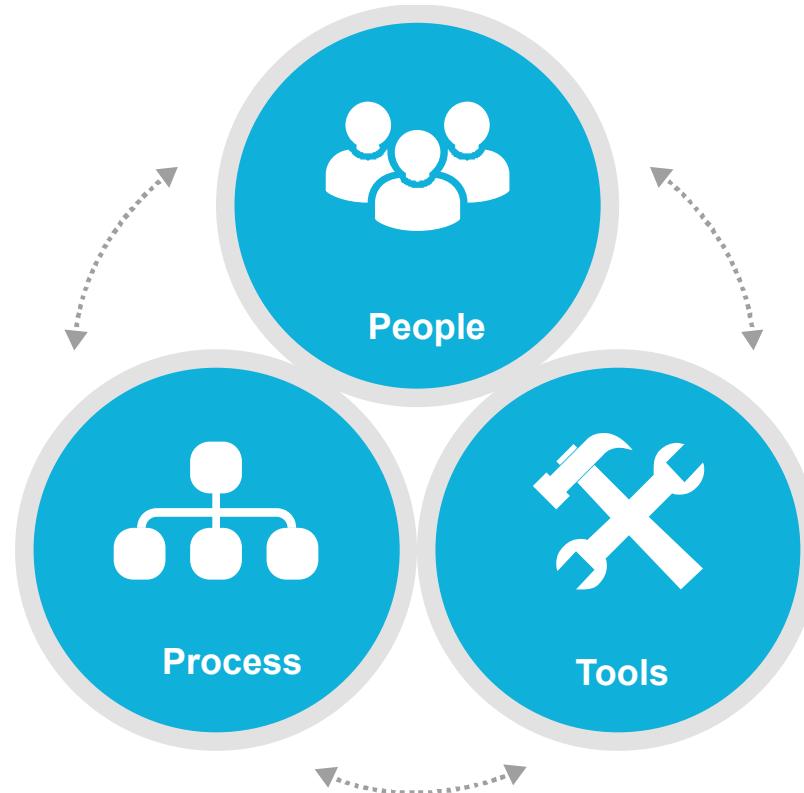
- 2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)
- 3:00 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z
- 5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

- 2:00 zUnit and VTP Overview (Wilbert)
- 2:15 Demo: Scenario using Z DevOps solutions including **ADDI**, **zUnit**, **Git** and **Jenkins** (David/Regi)
- 3:00 – 4:50 Choose one Optional lab:
- LAB 7 - IBM DBB with Git, Jenkins and UCD on Z
 - LAB 6B : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Remote Assets
 - LAB 6C : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Local Assets
 - LAB 6D : Using IBM zUnit to Unit Test a COBOL/DB2 batch program
 - LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
 - LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
 - LAB 8 - Using Application Performance Analyzer (APA)
 - LAB 9 - Using IBM Z VTP to test a COBOL /CICS / DB2 transaction
 - LAB 10 - Deploying COBOL/CICS/DB2 application using a GitLab CI Pipeline
 - LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
 - LAB 12 – Using ADDI to find CICS programs to be converted to API's
 - LAB 13 – Running Integration Tests using Galasa
- 5:00 Checkpoint - Comments and Closing Day 2 (David/Ron/Wilbert/Regi)

DevOps is not one of
these things...

It's all of them!



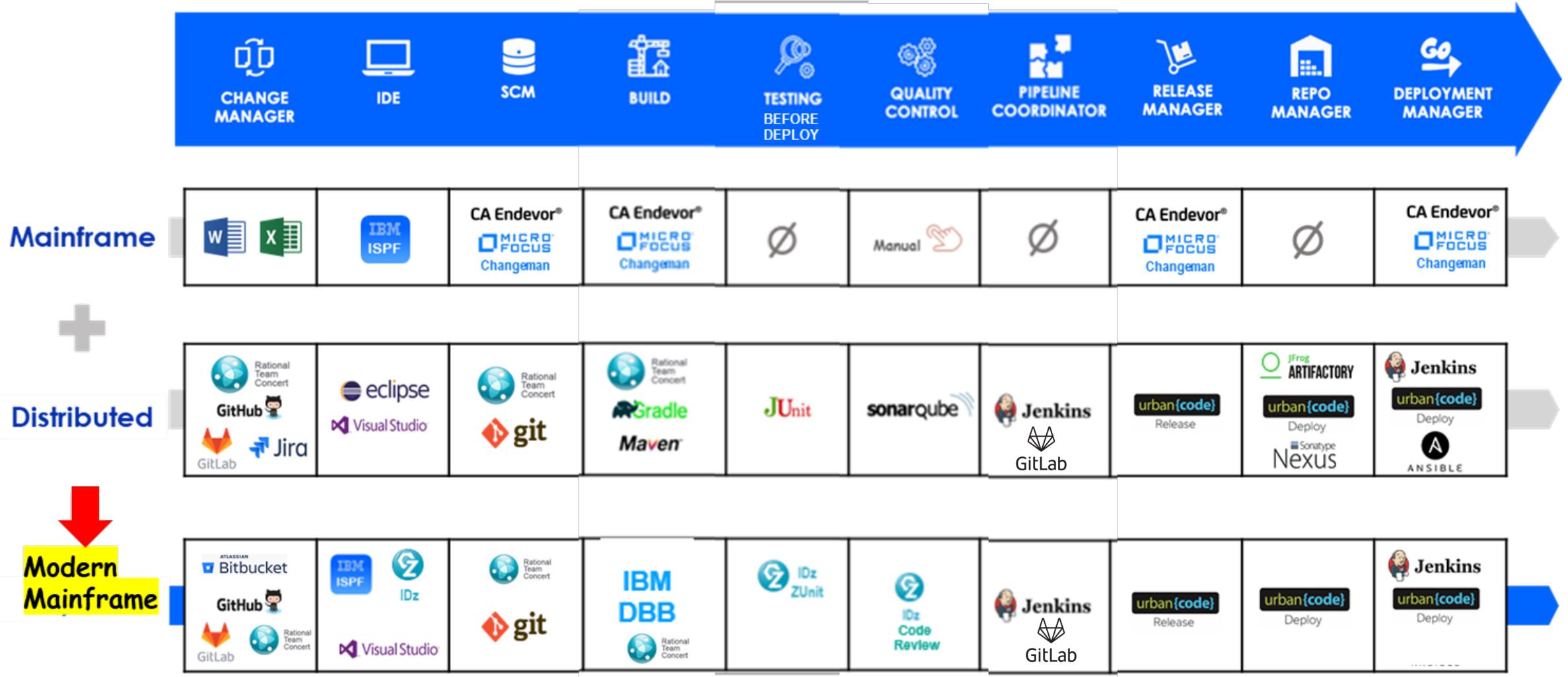
DevOps is a set of practices that combines software development (Dev) and IT operations (Ops).
It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.



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



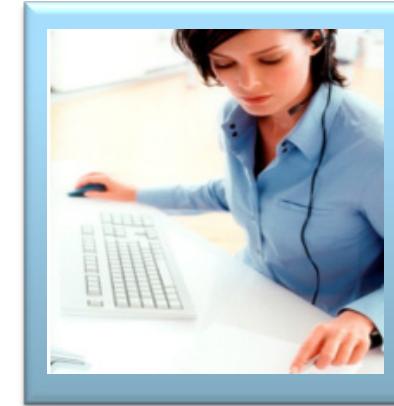
During the
BUILD phase

\$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 ?



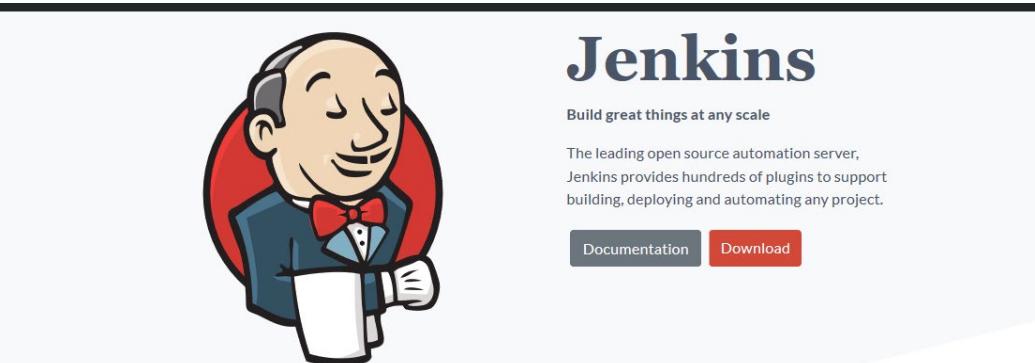
- A distributed, open source version control system
 - Runs on all platforms (z/OS, Linux, Windows, Mac)
 - z/OS uses Rocket® Open Source implementation on USS
 - Command line interface
 - Community supported
- The de-facto standard for software development
 - Integrated into many tools
 - Jenkins, Slack, 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 ...

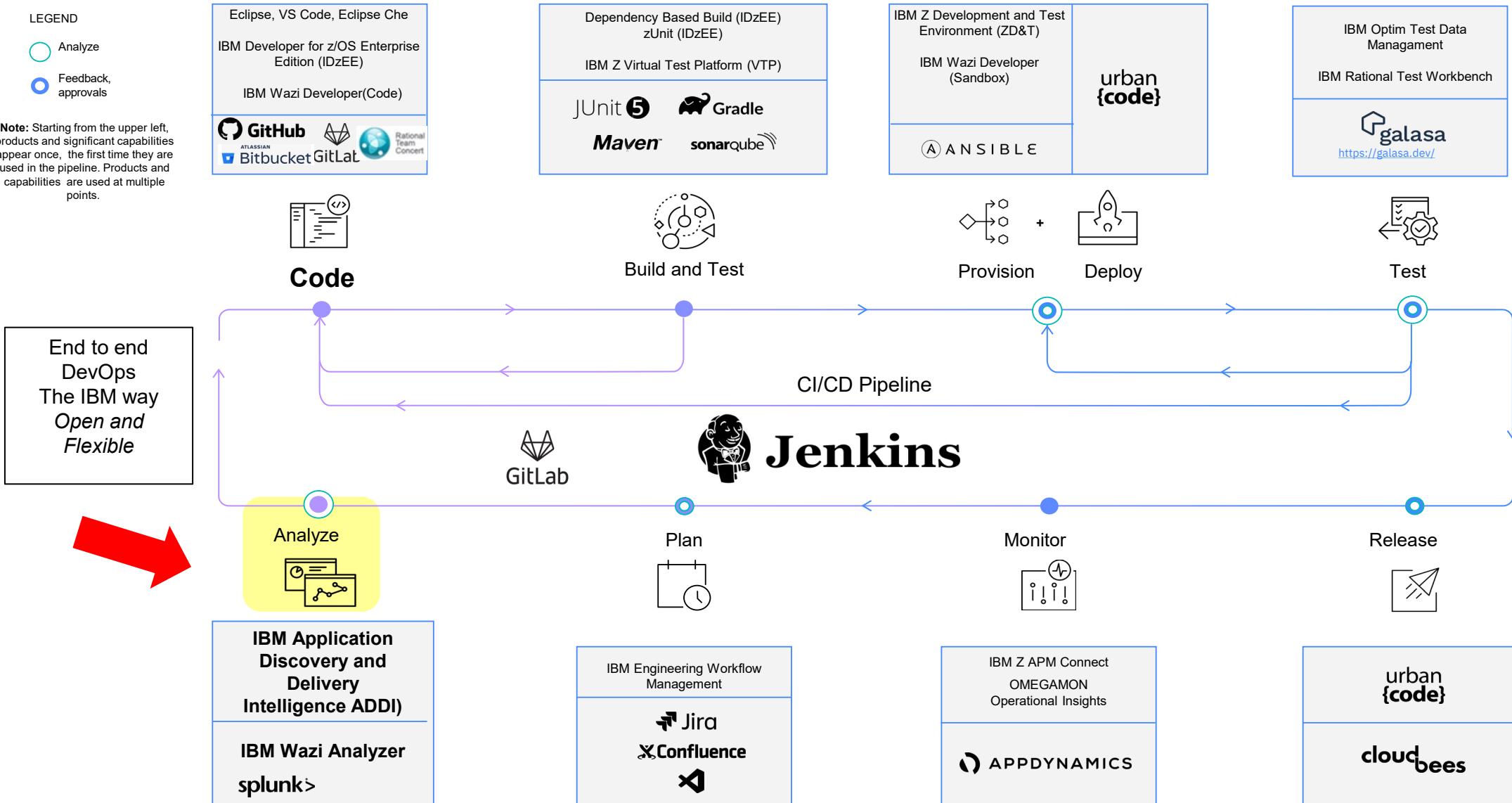


The screenshot shows the Jenkins homepage. On the left is a cartoon illustration of a man with a white beard, wearing a blue suit, a red bow tie, and a white shirt, holding a white scroll. To his right is the word "Jenkins" in a large, bold, blue sans-serif font. Below it is the tagline "Build great things at any scale". A short description follows: "The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project." At the bottom of this section are two buttons: "Documentation" (grey) and "Download" (red). The background of the main content area is white.



The screenshot shows the "User Stories" section of the Jenkins website. The background is orange with a grid pattern. At the top left, the text "Jenkins is the Way!" is displayed. Below it is a paragraph about user stories and a "More info" button. At the bottom center is a cartoon illustration of a knight in armor holding a sword, with the word "JENKINS" written below it. The background of this section is orange with concentric circles.

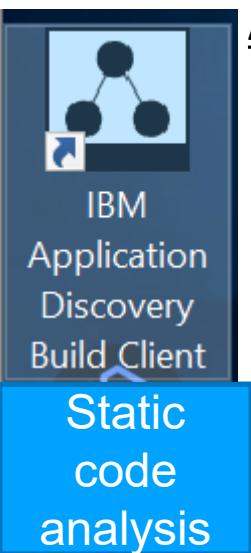
Z DevOps Pipeline



AD and ADDI Extensions

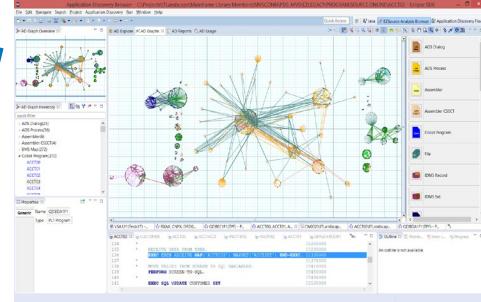
AD
“Application Discovery Browser”

Call graphs Resource Usage Reports Impact Analysis

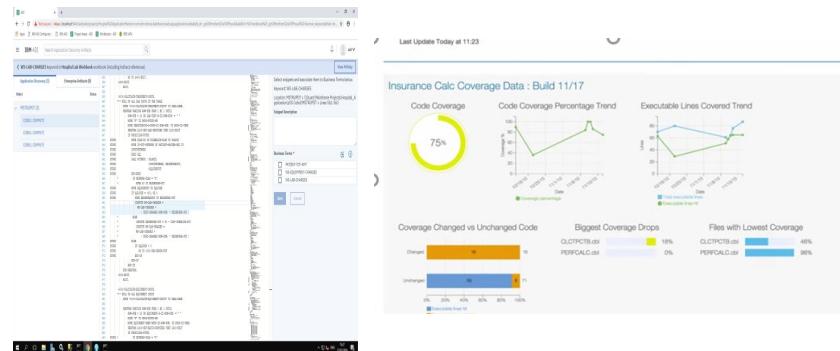


AD Discovers “Business Keywords”
used by/within business
Terms in COBOL

AD
Repository



ADDI Extensions
“WEB Browser”



ADDI Extensions Manages
Business Keywords, Terms, Rules



Source
Code

Incremental
update

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

AD = Application Discovery

ADDI = Application Discovery and Delivery Intelligence

If interested in doing the ADDI Lab tomorrow

- You MUST request an ADDI zTrial instance

1. Open IBM zTrial registration page

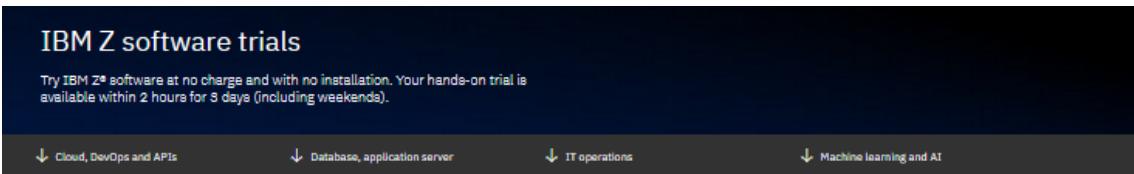
<https://www.ibm.com/it-infrastructure/z/resources/trial>

2. Click on Register for the trial

3. Login or Create a new IBMID

4. Select the Application Discovery and Delivery Intelligence zTrial

5. Look for emails for when your instance is ready



Cloud, DevOps and API integration software trials

Updated! IBM z/OS® Connect Enterprise Edition

Create efficient and scalable RESTful APIs for mobile and cloud applications. Learn how to:

- * Create REST APIs to: CICS® programs
IMS™ transactions
Db2® for z/OS data
IBM® MQ Queues.
- * Call a REST API from a COBOL application.

[Register for the trial](#)
[Explore the product first](#)

New! IBM Z Development and Test Environment

Use a web-based tool to learn how to self-provision a z/OS® environment on emulated Z hardware for agile development and test activities. Learn how to:

- * Explore how to use a z/OS system.
- * Self-provision a z/OS from the IBM pre-built software stack ADCD.
- * Extract CICS® and Db2® application artifacts from your source z/OS environment and deploy them to your z/OS target environment.

[Register for the trial](#)
[Explore the product first](#)

New! IBM Application Performance Analyzer for z/OS®

Quickly identify and act upon areas of low performance, high CPU consumption, low response time, and low throughput in your most critical z/OS applications. Learn how to:

- * Create, initiate, and analyze performance observations for a Java and COBOL batch program.
- * Exploit source-program mapping and then modify source code statements in a Java and COBOL program.
- * Create and review a Variance Report to compare the performance of original and modified program.

[Register for the trial](#)
[Explore the product first](#)

New! Bring Your Own (BYO) z/OS for Cloud Native Development

With this trial of IBM Wazi Developer and IBM Developer for z/OS® Enterprise Edition you can choose your preferred IDE (Eclipse® or Microsoft® VS Code™) integrated with familiar DevOps tools such as Git and Jenkins to develop a z/OS application. Learn how to:

- * Use GitHub to see your assigned tasks and access application source code stored in Git.
- * Code, debug and build the sample COBOL application using Eclipse or VS Code.
- * Commit and push modifications to Git and request a Jenkins pipeline.

[Register for the trial](#)
[Explore the product first](#)

Updated! IBM Application Discovery and Delivery Intelligence

Manage your software lifecycle and digital and cloud transformation. Learn how to:

- * Identify maintainability issues and update unreachable code.
- * Find CICS® programs able to be converted to APIs.
- * View business terms and rules inventory in ADDI.
- * Discover and manage terms and rules.

[Register for the trial](#)
[Explore the product first](#)

Updated! IBM Cloud Provisioning and Management for z/OS

Provision z/OS software subsystems with automated processes. Learn how to:

- * Provision and deprovision a CICS® region with a cloud provisioning marketplace.
- * Create and publish a CICS region service to facilitate self-service provisioning.
- * Provision, deprovision an MQ queue with a cloud provisioning marketplace.

[Register for the trial](#)
[Explore the product first](#)

IBM SDK for Node.js - z/OS

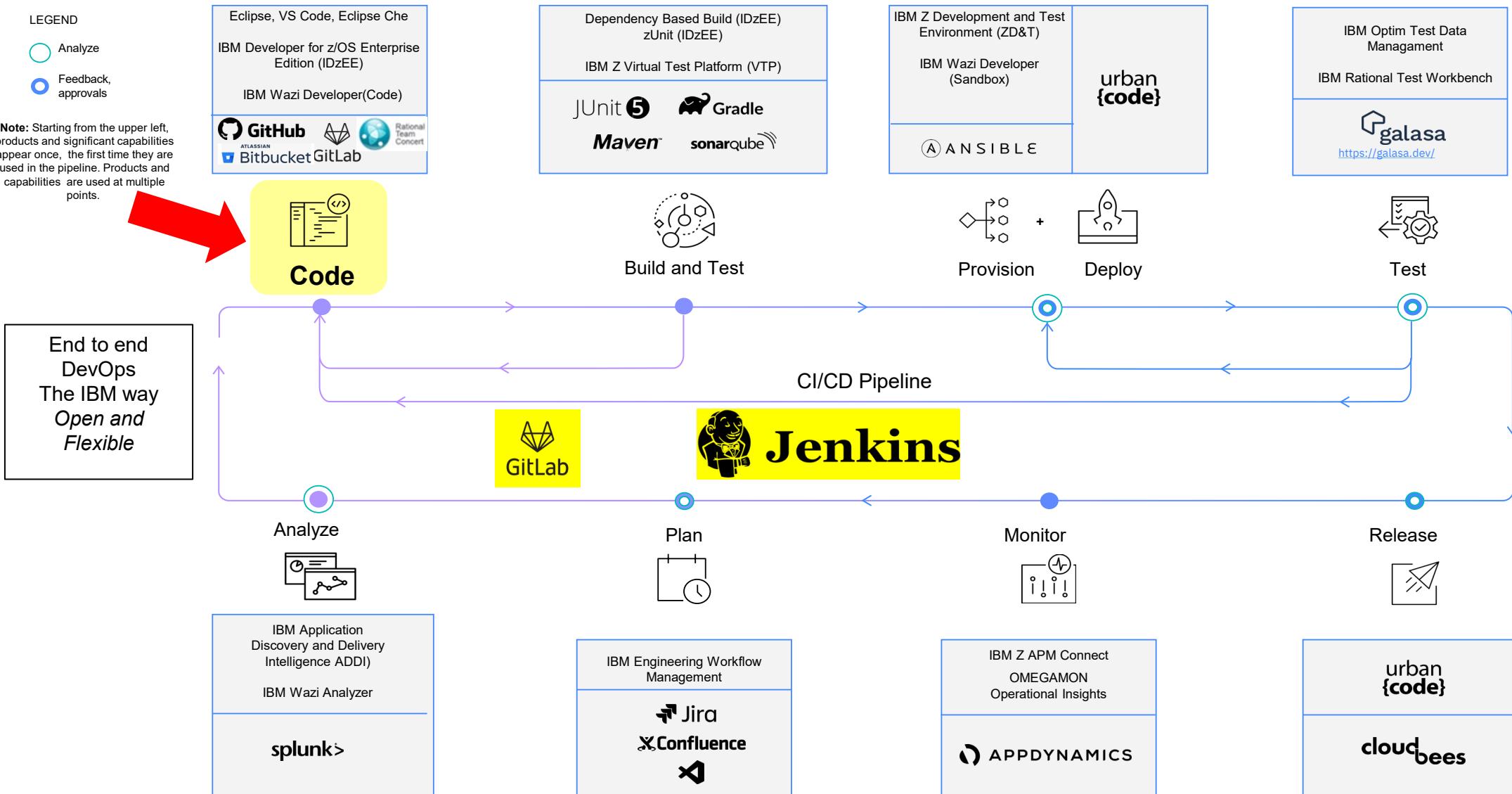
Modernize apps, orchestrate services with Node.js and connect to your z/OS assets. Learn how to:

Learn how to:

IBM Application Delivery Foundation for z/OS

Integrated tools that help teams design, build, test, and debug z/OS software. Learn how to:

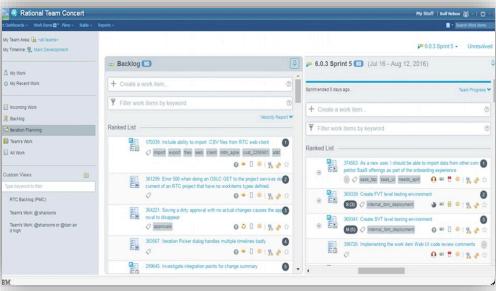
Z DevOps Pipeline



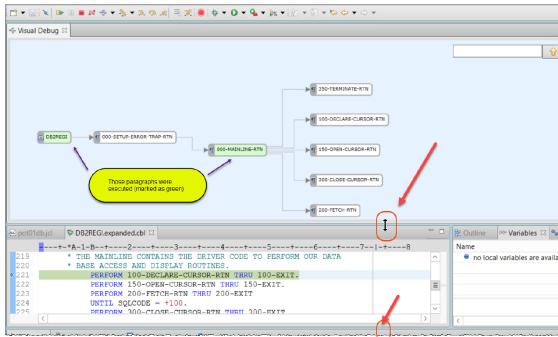
z/OS and subsystems

Build and SCM

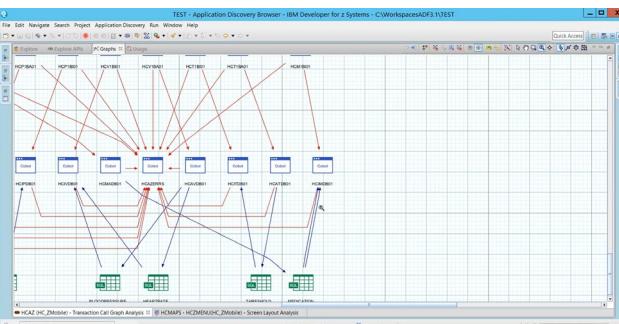
Integration with Git, RTC and Endeavor for Lifecycle and Source Management



Debug and code coverage

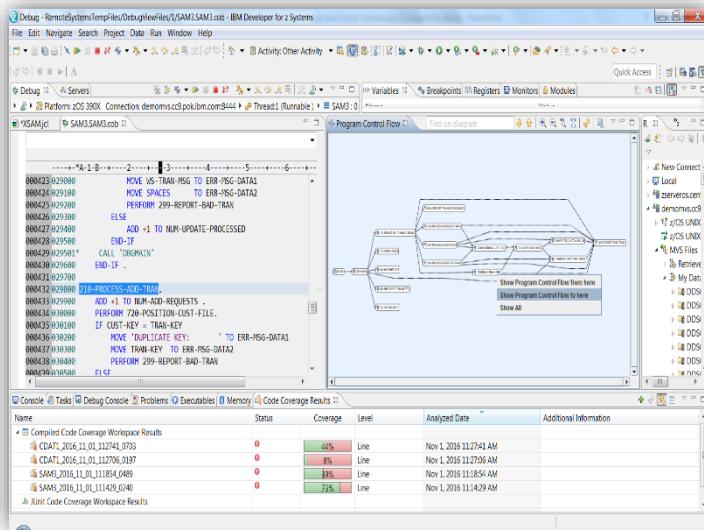


Application Discovery

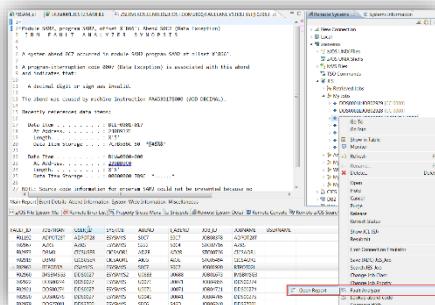


IBM Developer for z/OS Enterprise Edition - the developer's cockpit!

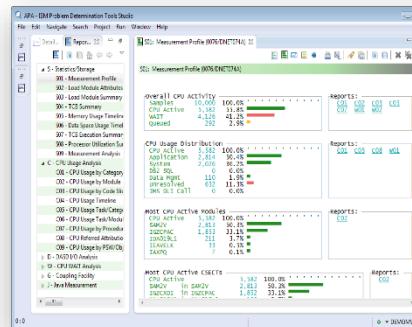
Unit test and code coverage



Abend analysis



Performance analysis



File Management

Analyze (program understanding)

Outline view

- Show high level abstraction of source

Program control flow

- Graphical view of program logic

Data flow diagram

- Graphical view of data movement throughout the program

Powerful local and remote search capabilities

The screenshot displays the Rational Application Developer interface with several windows open:

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

Analyze (code rules and statistics)

Code rules

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

Program level statistics

- Base metrics
- Complexity
- Halstead

SonarLint integration

Program statistics

Property	Value
Basic Metrics	
Lines of code	254
Number of comment lines	19
Number of copybooks	3
Number of data items	242
Number of lines	298
Complexity Metrics	
Cyclomatic complexity	15
Halstead Metrics	
Difficulty level	11.62
Effort to implement	21066.84
Number of delivered bugs	0.6
Number of operands	152
Number of operators	122
Number of unique operands	85
Number of unique operators	13
Program length	274
Program level	592.9
Program vocabulary size	98
Program volume	1812.43
Time to implement	1170.38

Code rules

Name: COBOL RULES

Scope Rules

Rule Sets:

Analysis Domains and Rules

- COBOL Code Review [22/25]
 - Naming Conventions [1/1]
 - Use PROGRAM-ID that matches the source member name
 - Performance [7/8]
 - Avoid INITIALIZE statements. Use elementary MOVE statements or
 - Avoid OCCURS DEPENDING ON clauses
 - Avoid subscripts in table access. Use indexes.
 - Use an odd number of digits in a COMP-3 PICTURE clause
 - Use binary subscripts
 - Use DFHRESP value to check response code
 - EVALUATE statement rather than nested IF statement
 - Use RESP option in EXEC command
 - Program Structures [14/16]
 - A GO TO statement can reference a paragraph only if it is an exit p-
 - Avoid ACCEPT FROM CONSOLE statements
 - Avoid ACCEPT statements
 - Avoid ALTER statements
 - Avoid CALL statement with program-name. Use dynamic calls.
 - Avoid COPY SUPPRESS statements
 - Avoid CORRESPONDING phrases
 - Avoid ENTRY statements
 - Avoid EXIT PROGRAM statements
 - Avoid GO TO statements
 - Avoid NEXT SENTENCE phrases
 - Avoid RESERVE clause in FILE-CONTROL paragraphs
 - Avoid STOP RUN and STOP literal statements
 - Avoid THREEL phrase in PERFORM statements

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

```
45     B0 TRAN-OK      REC-X      REC-X(X58).
46     B1 TRAN-HSG     REC-X      REC-X(X58).
47     B2 TRAN-HSG     REC-X      REC-X(X58).
48     B3 TRAN-HSG     REC-X      REC-X(X58).
49     B4 TRAN-HSG     REC-X      REC-X(X58).
50     B5 TRAN-HSG     REC-X      REC-X(X58).
51     B6 TRAN-HSG     REC-X      REC-X(X58).
52     B7 TRAN-HSG     REC-X      REC-X(X58).
53     B8 TRAN-HSG     REC-X      REC-X(X58).
54     B9 TRAN-HSG     REC-X      REC-X(X58).
55     B10 TRAN-HSG    REC-X      REC-X(X58).
56     B11 TRAN-HSG    REC-X      REC-X(X58).
57     B12 TRAN-HSG    REC-X      REC-X(X58).
58     B13 TRAN-HSG    REC-X      REC-X(X58).
59     B14 TRAN-HSG    REC-X      REC-X(X58).
60     B15 TRAN-HSG    REC-X      REC-X(X58).
61     B16 TRAN-HSG    REC-X      REC-X(X58).
62     B17 TRAN-HSG    REC-X      REC-X(X58).
63     B18 TRAN-HSG    REC-X      REC-X(X58).
64     B19 TRAN-HSG    REC-X      REC-X(X58).
65     B20 TRAN-HSG    REC-X      REC-X(X58).
66     B21 TRAN-HSG    REC-X      REC-X(X58).
67     B22 TRAN-HSG    REC-X      REC-X(X58).
68     B23 TRAN-HSG    REC-X      REC-X(X58).
69     B24 TRAN-HSG    REC-X      REC-X(X58).
70     B25 TRAN-HSG    REC-X      REC-X(X58).
71     B26 TRAN-HSG    REC-X      REC-X(X58).
72     B27 TRAN-HSG    REC-X      REC-X(X58).
73     B28 TRAN-HSG    REC-X      REC-X(X58).
74     B29 TRAN-HSG    REC-X      REC-X(X58).
75     B30 TRAN-HSG    REC-X      REC-X(X58).
76     B31 TRAN-HSG    REC-X      REC-X(X58).
77     B32 TRAN-HSG    REC-X      REC-X(X58).
78     B33 TRAN-HSG    REC-X      REC-X(X58).
79     B34 TRAN-HSG    REC-X      REC-X(X58).
80     B35 TRAN-HSG    REC-X      REC-X(X58).
81     B36 TRAN-HSG    REC-X      REC-X(X58).
82     B37 TRAN-HSG    REC-X      REC-X(X58).
83     B38 TRAN-HSG    REC-X      REC-X(X58).
84     B39 TRAN-HSG    REC-X      REC-X(X58).
85     B40 TRAN-HSG    REC-X      REC-X(X58).
86     B41 TRAN-HSG    REC-X      REC-X(X58).
87     B42 TRAN-HSG    REC-X      REC-X(X58).
88     B43 TRAN-HSG    REC-X      REC-X(X58).
89     B44 TRAN-HSG    REC-X      REC-X(X58).
90     B45 TRAN-HSG    REC-X      REC-X(X58).
91     B46 TRAN-HSG    REC-X      REC-X(X58).
92     B47 TRAN-HSG    REC-X      REC-X(X58).
93     B48 TRAN-HSG    REC-X      REC-X(X58).
94     B49 TRAN-HSG    REC-X      REC-X(X58).
95     B50 TRAN-HSG    REC-X      REC-X(X58).
96     B51 TRAN-HSG    REC-X      REC-X(X58).
97     B52 TRAN-HSG    REC-X      REC-X(X58).
98     B53 TRAN-HSG    REC-X      REC-X(X58).
99     B54 TRAN-HSG    REC-X      REC-X(X58).
100    B55 TRAN-HSG   REC-X      REC-X(X58).
101    B56 TRAN-HSG   REC-X      REC-X(X58).
102    B57 TRAN-HSG   REC-X      REC-X(X58).
103    B58 TRAN-HSG   REC-X      REC-X(X58).
104    B59 TRAN-HSG   REC-X      REC-X(X58).
105    B60 TRAN-HSG   REC-X      REC-X(X58).
106    B61 TRAN-HSG   REC-X      REC-X(X58).
107    B62 TRAN-HSG   REC-X      REC-X(X58).
108    B63 TRAN-HSG   REC-X      REC-X(X58).
109    B64 TRAN-HSG   REC-X      REC-X(X58).
110    B65 TRAN-HSG   REC-X      REC-X(X58).
111    B66 TRAN-HSG   REC-X      REC-X(X58).
112    B67 TRAN-HSG   REC-X      REC-X(X58).
113    B68 TRAN-HSG   REC-X      REC-X(X58).
114    B69 TRAN-HSG   REC-X      REC-X(X58).
115    B70 TRAN-HSG   REC-X      REC-X(X58).
116    B71 TRAN-HSG   REC-X      REC-X(X58).
117    B72 TRAN-HSG   REC-X      REC-X(X58).
118    B73 TRAN-HSG   REC-X      REC-X(X58).
119    B74 TRAN-HSG   REC-X      REC-X(X58).
120    B75 TRAN-HSG   REC-X      REC-X(X58).
121    B76 TRAN-HSG   REC-X      REC-X(X58).
122    B77 TRAN-HSG   REC-X      REC-X(X58).
123    B78 TRAN-HSG   REC-X      REC-X(X58).
124    B79 TRAN-HSG   REC-X      REC-X(X58).
125    B80 TRAN-HSG   REC-X      REC-X(X58).
126    B81 TRAN-HSG   REC-X      REC-X(X58).
127    B82 TRAN-HSG   REC-X      REC-X(X58).
128    B83 TRAN-HSG   REC-X      REC-X(X58).
129    B84 TRAN-HSG   REC-X      REC-X(X58).
130    B85 TRAN-HSG   REC-X      REC-X(X58).
131    B86 TRAN-HSG   REC-X      REC-X(X58).
132    B87 TRAN-HSG   REC-X      REC-X(X58).
133    B88 TRAN-HSG   REC-X      REC-X(X58).
134    B89 TRAN-HSG   REC-X      REC-X(X58).
135    B90 TRAN-HSG   REC-X      REC-X(X58).
136    B91 TRAN-HSG   REC-X      REC-X(X58).
137    B92 TRAN-HSG   REC-X      REC-X(X58).
138    B93 TRAN-HSG   REC-X      REC-X(X58).
139    B94 TRAN-HSG   REC-X      REC-X(X58).
140    B95 TRAN-HSG   REC-X      REC-X(X58).
141    B96 TRAN-HSG   REC-X      REC-X(X58).
142    B97 TRAN-HSG   REC-X      REC-X(X58).
143    B98 TRAN-HSG   REC-X      REC-X(X58).
144    B99 TRAN-HSG   REC-X      REC-X(X58).
145    B100 TRAN-HSG  REC-X      REC-X(X58).
146    B101 TRAN-HSG  REC-X      REC-X(X58).
147    B102 TRAN-HSG  REC-X      REC-X(X58).
148    B103 TRAN-HSG  REC-X      REC-X(X58).
149    B104 TRAN-HSG  REC-X      REC-X(X58).
150    B105 TRAN-HSG  REC-X      REC-X(X58).
151    B106 TRAN-HSG  REC-X      REC-X(X58).
152    B107 TRAN-HSG  REC-X      REC-X(X58).
153    B108 TRAN-HSG  REC-X      REC-X(X58).
154    B109 TRAN-HSG  REC-X      REC-X(X58).
155    B110 TRAN-HSG  REC-X      REC-X(X58).
156    B111 TRAN-HSG  REC-X      REC-X(X58).
157    B112 TRAN-HSG  REC-X      REC-X(X58).
158    B113 TRAN-HSG  REC-X      REC-X(X58).
159    B114 TRAN-HSG  REC-X      REC-X(X58).
160    B115 TRAN-HSG  REC-X      REC-X(X58).
161    B116 TRAN-HSG  REC-X      REC-X(X58).
162    B117 TRAN-HSG  REC-X      REC-X(X58).
163    B118 TRAN-HSG  REC-X      REC-X(X58).
164    B119 TRAN-HSG  REC-X      REC-X(X58).
165    B120 TRAN-HSG  REC-X      REC-X(X58).
166    B121 TRAN-HSG  REC-X      REC-X(X58).
167    B122 TRAN-HSG  REC-X      REC-X(X58).
168    B123 TRAN-HSG  REC-X      REC-X(X58).
169    B124 TRAN-HSG  REC-X      REC-X(X58).
```

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

Edit

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

The screenshot displays the IBM Developer for z Systems interface with several open windows:

- Modern editor:** Shows COBOL source code for HCP1PL01.cbl. It highlights the `MAINLINE SECTION.` and `PERFORM SETUP-SCREEN.` sections.
- LPEX Editor:** Shows COBOL source code for HCPDB01.cbl. It highlights the `PROCEDURE DIVISION.` section and the `MOVE -1 TO HCP1PNOL` statement.
- Smart Editor for COBOL, PL/I, JCL, Assembler, REXX, Java, XML, etc...:** A yellow box containing the text "Smart Editor for COBOL, PL/I, JCL, Assembler, [REXX](#), Java, XML, etc...".
- REXX Editor:** Shows a REXX script named CloudPocSecuritySetup.rexx. The script includes comments about symbolic names and a section for validating TSO addresses.
- Context menu:** A right-clicked menu is open over the REXX editor, listing options like Undo Typing, Revert File, Save, Open With, Show In, Cut, Copy, Paste, Quick Fix, Shift Right, Shift Left, and Add to Snapshot. The "Open With" option is highlighted, and a red arrow points to the "REXX Editor" entry in the submenu.

The interface also includes a z/OS Projects view, a Data Source Explorer, and a Remote Systems view showing job history and active jobs.

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 shows the ADFz interface with two main windows. On the left is a 'File Editor' window titled '(RDz connection: zserveros) dds0001@ZSERVEROS.CENTERS.IHOST.COM-2800:DDS0001.PATINS'. It displays a table with columns: PATIENT-ID, INS-COMPANY-PRIMARY-ID, CARRIER-NAME, and CARRIER-PH. The table contains several rows of data. Below this is a 'Field Editor' window titled 'Layout PATIENT-INSURANCE' which shows detailed field definitions for each column. On the right is a 'Remote Systems' browser window with a tree view of connections. A context menu is open over a connection named 'DDS0001.PATINS', with the 'File Manager' option highlighted. An orange arrow points from the 'File Manager' option in the context menu to the 'File Manager' section of the slide content.

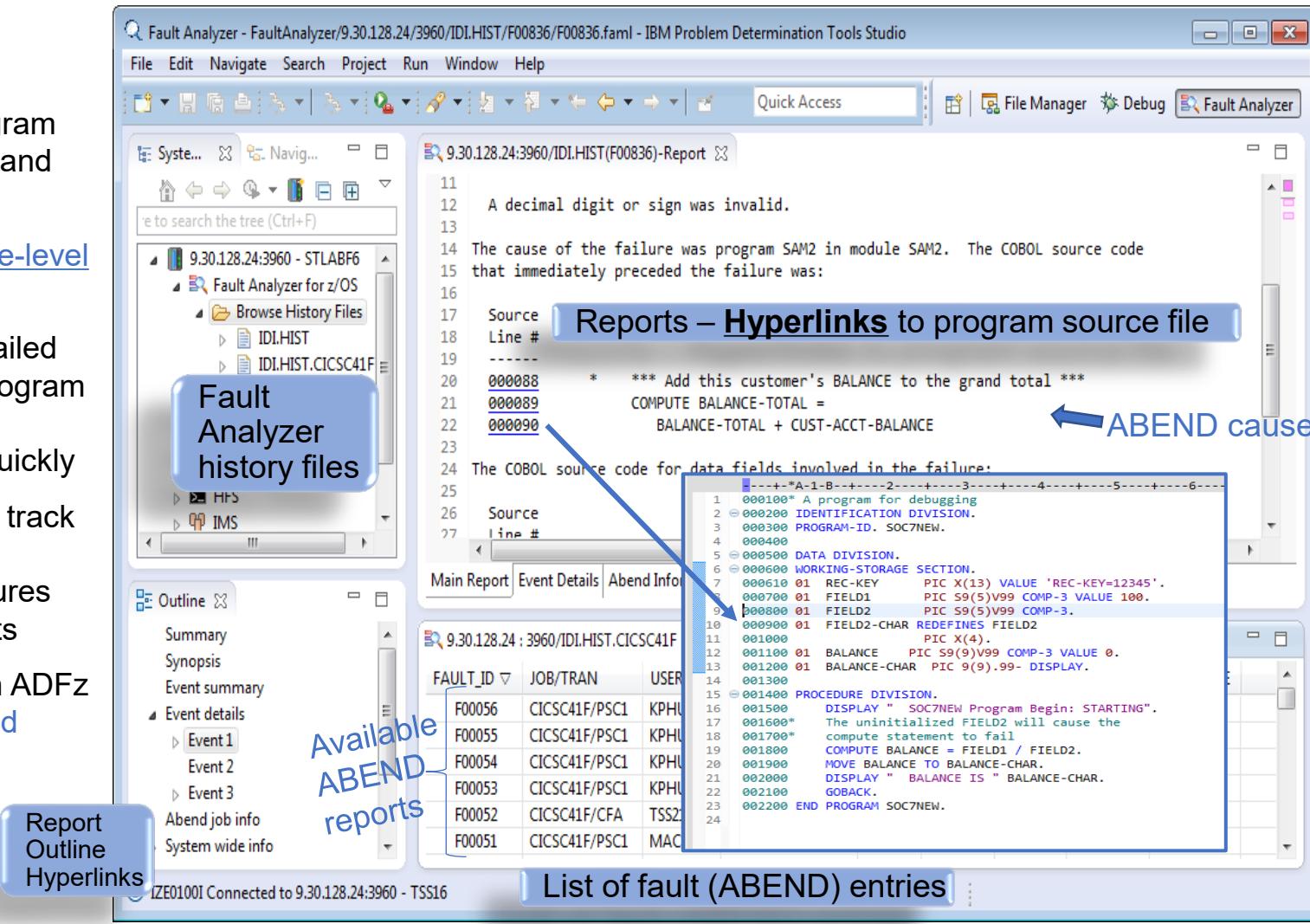
**Access VSAM Files directly from IDz Edit
(Remote Systems view)**

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

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

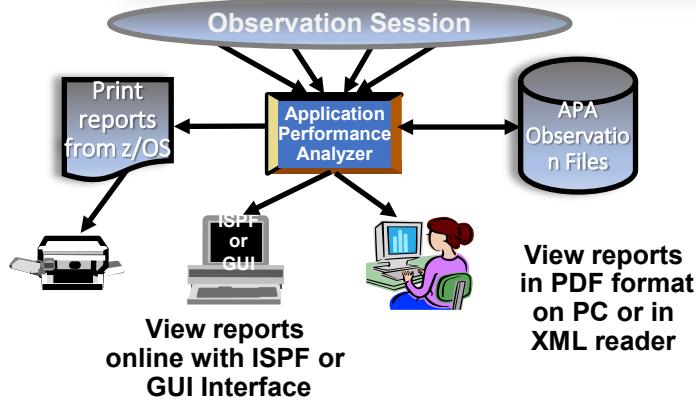
IBM Fault Analyzer improves developer productivity and decreases deployment costs by helping to analyze and correct application failures quickly (CICS,DB2,IMS,MQ,COBOL,PLI,ASM, C/C++,JAVA)

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



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

Provides rapid pin-pointing of enterprise application bottlenecks



- 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

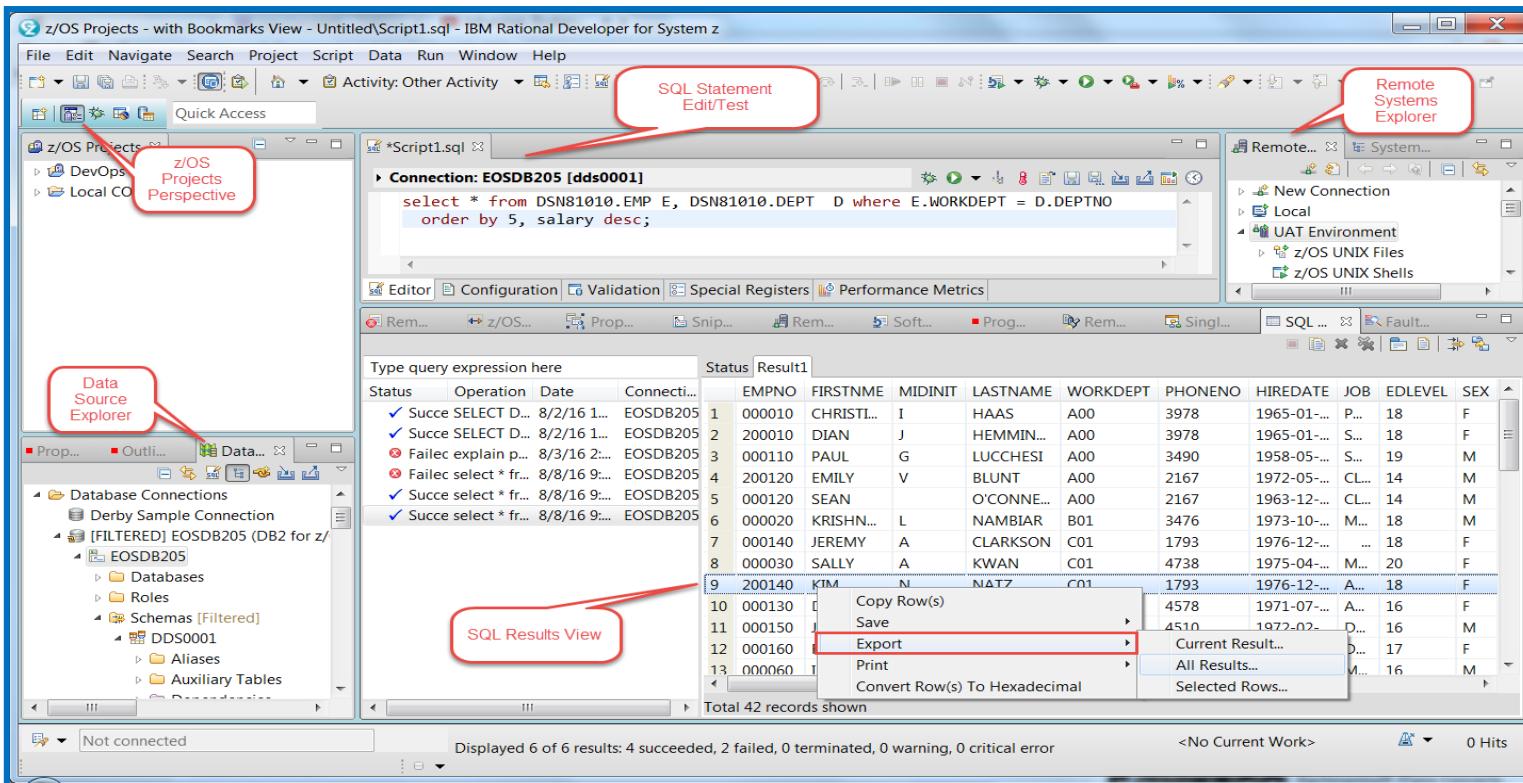


- Specify number of times APA monitors a job's performance when **job or program becomes active**
- **Invoke the monitoring capability** from other programs such as IBM Tivoli® OMEGAMON
- **Compare two observation reports** to see the relevant differences, to supplement threshold monitoring
- Assembler,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

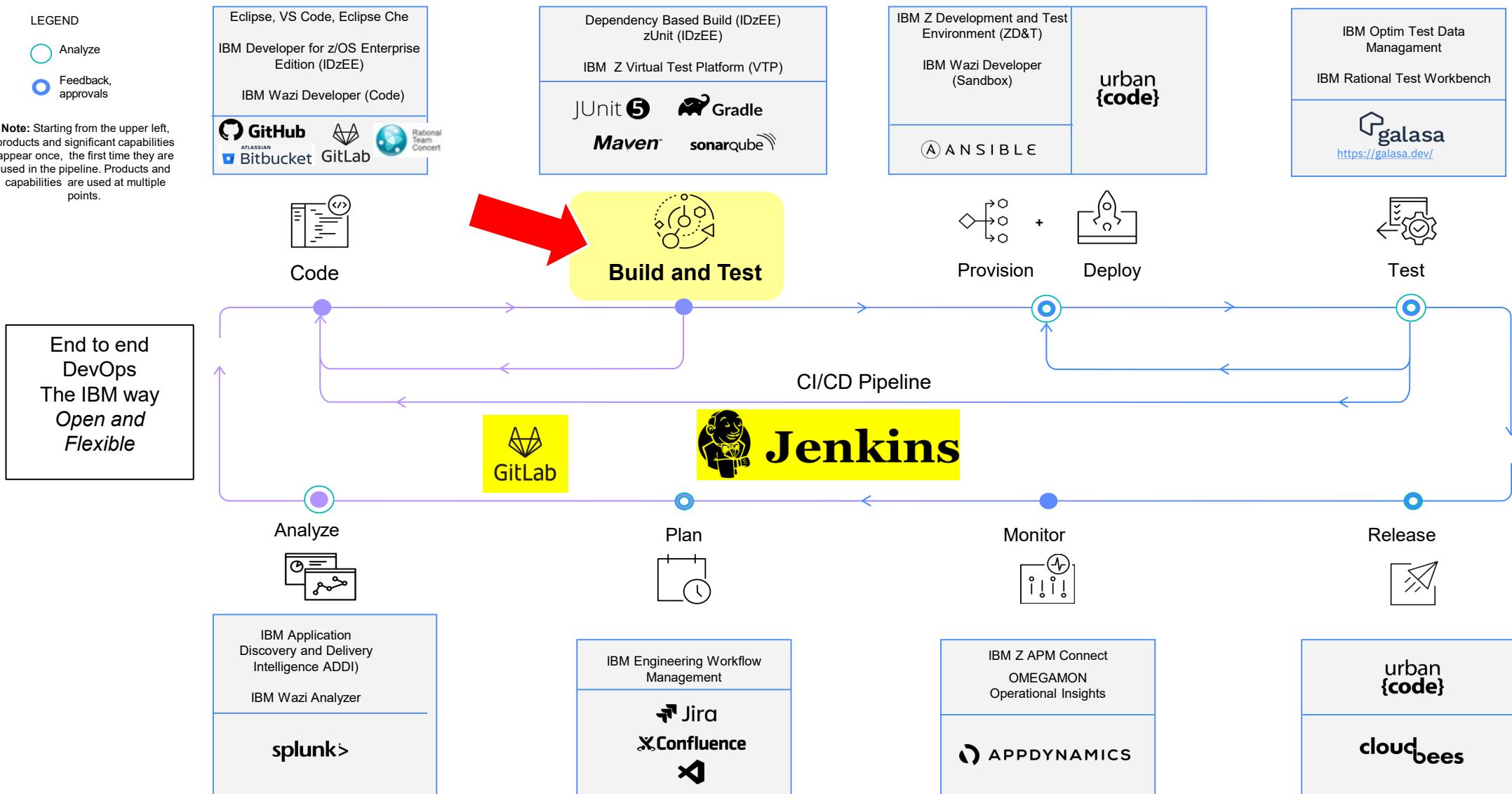


EGit included in IDz

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

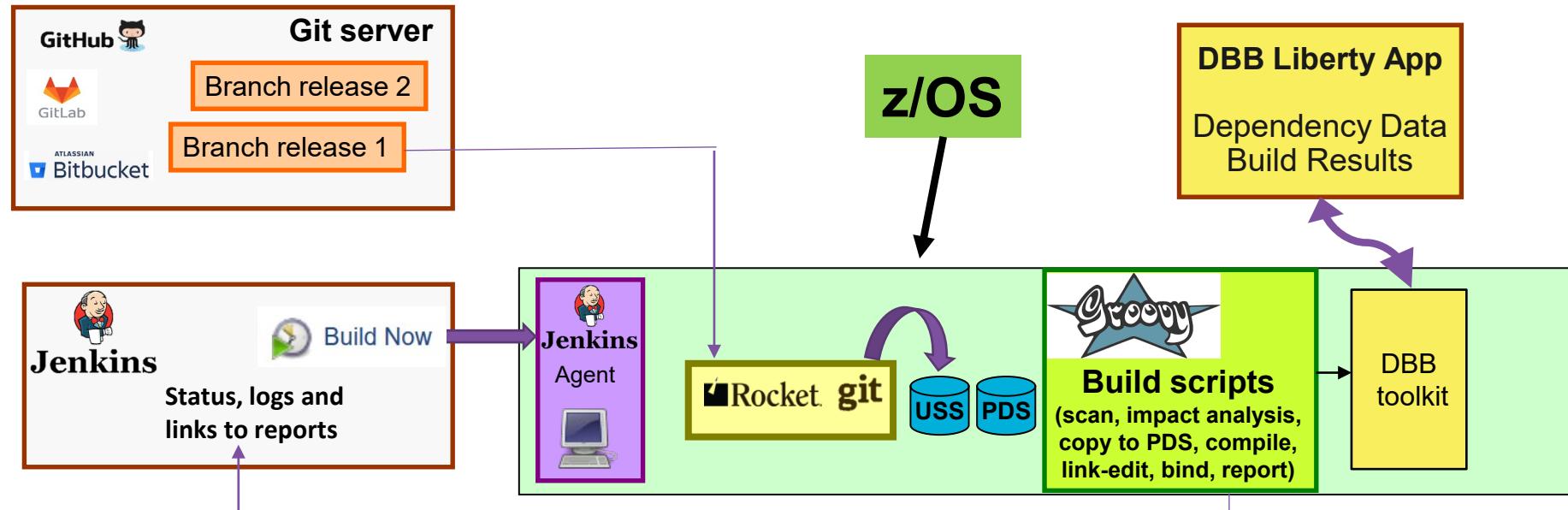
The screenshot illustrates the integration of EGit within the IBM Integration Designer (IDz) environment. On the left, the GitHub interface is visible, showing a repository named 'RegiBrazil / DemoHealthCare'. The main workspace shows a file tree for 'DemoHealthCare / DemoHealthCare / cobol_cics_db2' containing various COBOL source files like HCAPDB01.cbl, HCATDB01.cbl, etc. A red box highlights the GitHub tab. On the right, the IDz interface displays a Git Repositories view with a commit history for 'HCMADB02.cbl'. A specific commit, '59d1a2a [DemoHealthCare]', is selected, showing a diff view with changes made to the file. The commit message is 'diff --git a/DemoHealthCare/cobol_cics_db2/HCMADB02.cbl b/DemoHe'. Below the diff, it says 'Pushed to DemoHealthCare - origin'. The commit details show a push from 'sandbox' to 'origin' at 2020-08-24 17:17:28, with a note about a changed file 'DemoHealthCare/cobol_cics_db2/HCMADB02.cbl'.

Z DevOps Pipeline



What is IBM Dependency Based Build (DBB)?

- Provides a modern scripting languagebased automation capability that can be used on z/OS.
 - The DBB API can be called by Java based scripting languages such as **Groovy**, JRuby, Jython, Ant, Maven, Python, etc.
- Consists of a **build toolkit** (Java API, Groovy Installation) installed on **USS (z/OS)** and a **Liberty application server** that hosts build metadata (dependency data, build results) installed on **Linux**.



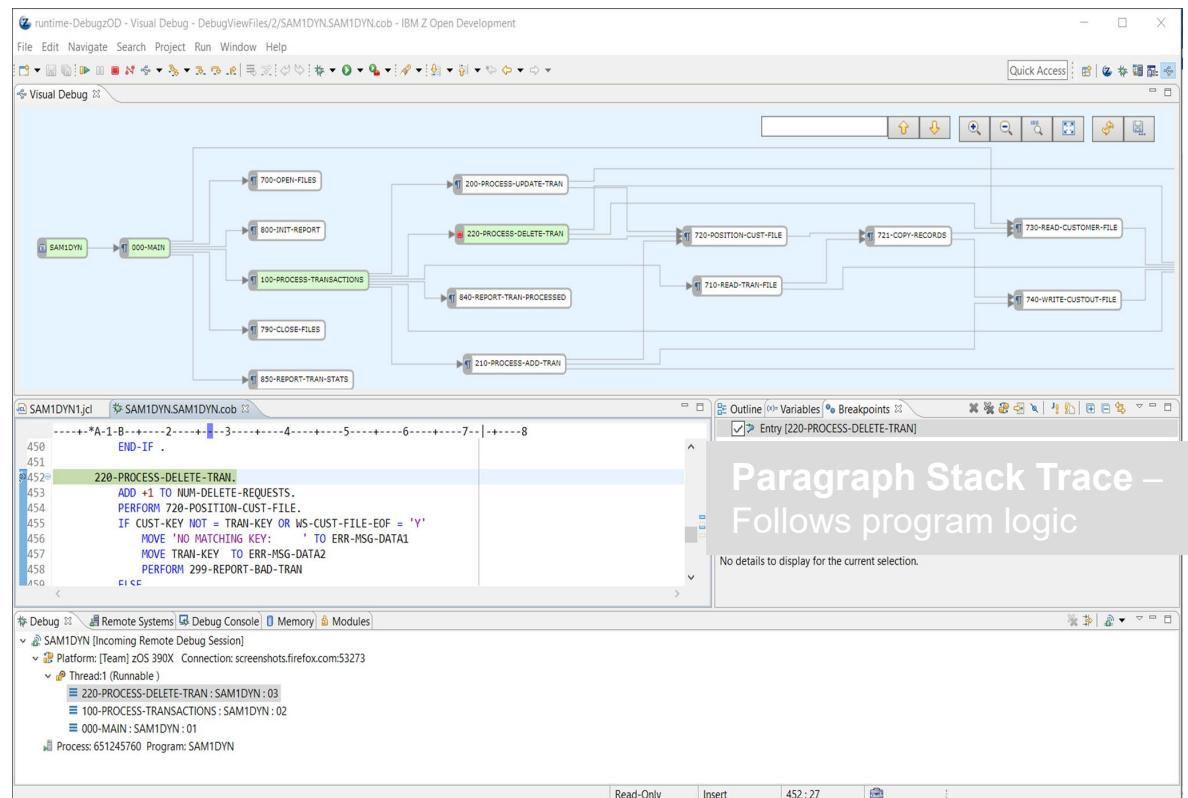
Diagnose (z/OS Debugger)

Program testing and analysis

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

With:

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



This screenshot shows a COBOL monitor session. The monitor command MONITOR is active, with the location set to CDAT1 and line 1 of 9. The screen displays several lines of COBOL code and their corresponding data. Lines 0001 through 0009 show the input of birthdate and date components. Line 436 contains the verification logic: 0500-VERIFY-INPUT-DATE. It checks if the birthdate is numeric and moves it to COM-INPUT-DATE. Lines 437 through 440 show the evaluation of the date validity. The bottom of the screen shows the PF key definitions and the date '02/015'.

```
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 02/015
```

Test (code coverage)

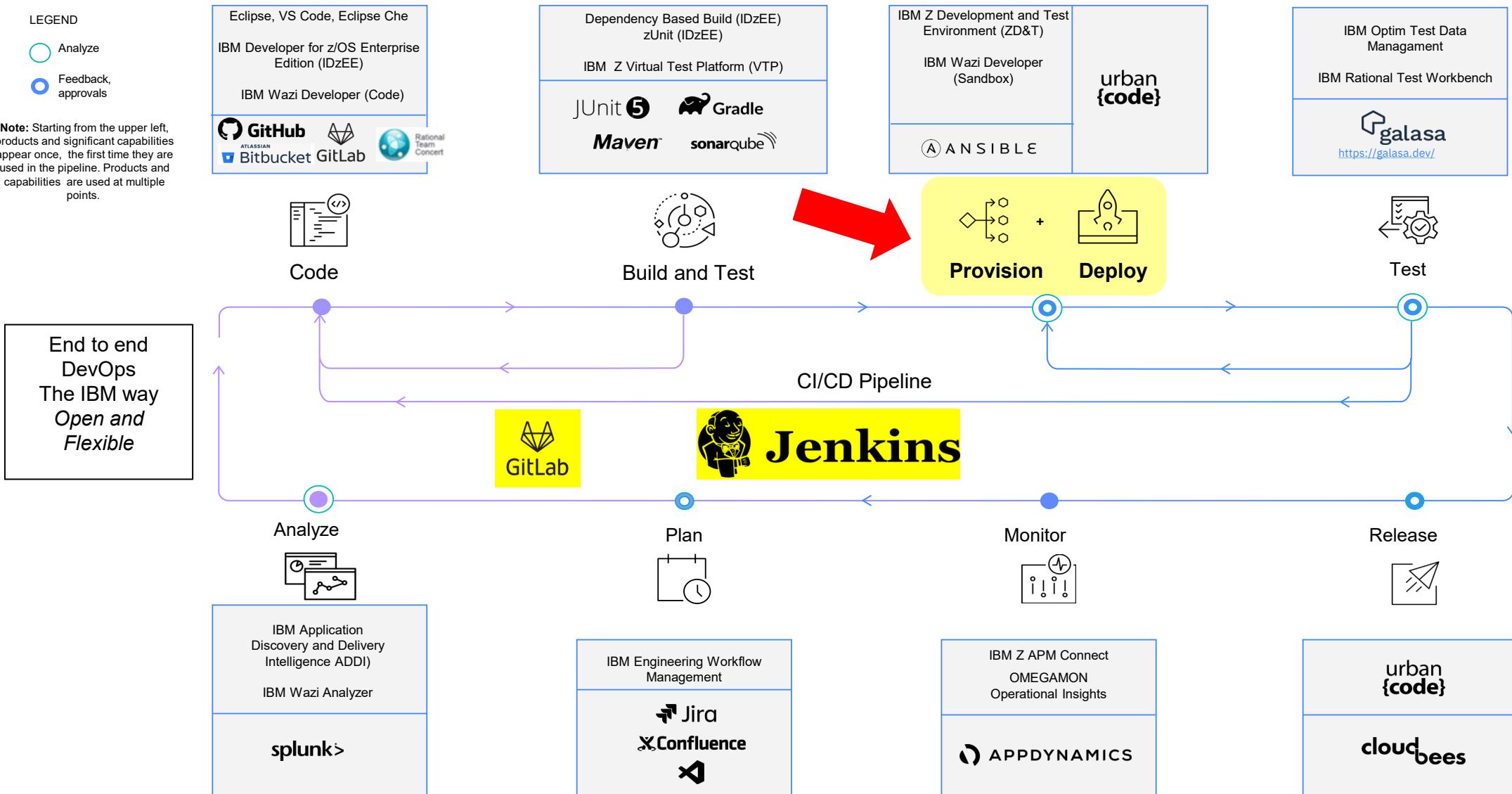
- Tracks lines of code that have been executed during test
- Improves application testing quality
- Focuses testing resource usage
- Numerous reports and options to assess testing and trends
 - Source view
 - Summary
 - Comparisons
 - Merging
- Supports: Batch, CICS and IMS
- Integration with ADDI and 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 in blue.
- Compare Results View:** A purple box labeled "Compare results" containing a sub-menu with options like Open, Filtered..., Compare, Add Result Location..., Show code coverage indicators, Delete..., Export..., and Properties... The "Compare" option is also present here.
- Coverage View:** A purple box labeled "Coverage" showing a snippet of COBOL source code with line numbers 221 through 260. The code handles file operations and transaction processing.
- Code Coverage Comparison Report:** A detailed report comparing two runs: "SAM1_2018_12_01_144006_0842" and "SAM1_2018_11_13_114716_0952". The report includes sections for Files and Modules, listing coverage percentages, lines covered, uncovered lines, total lines, and messages for each module.

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

Z DevOps Pipeline

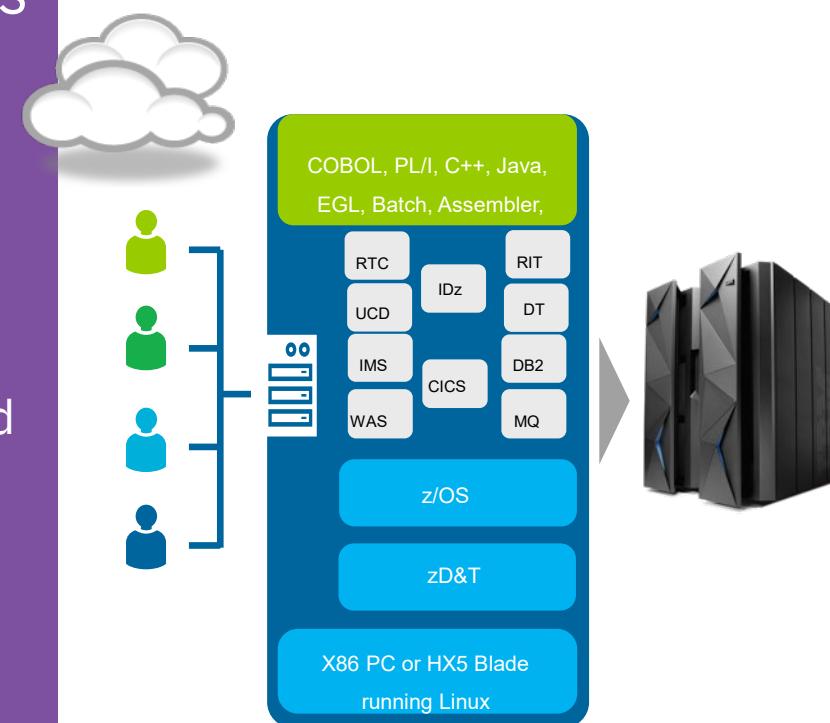


Provisioning

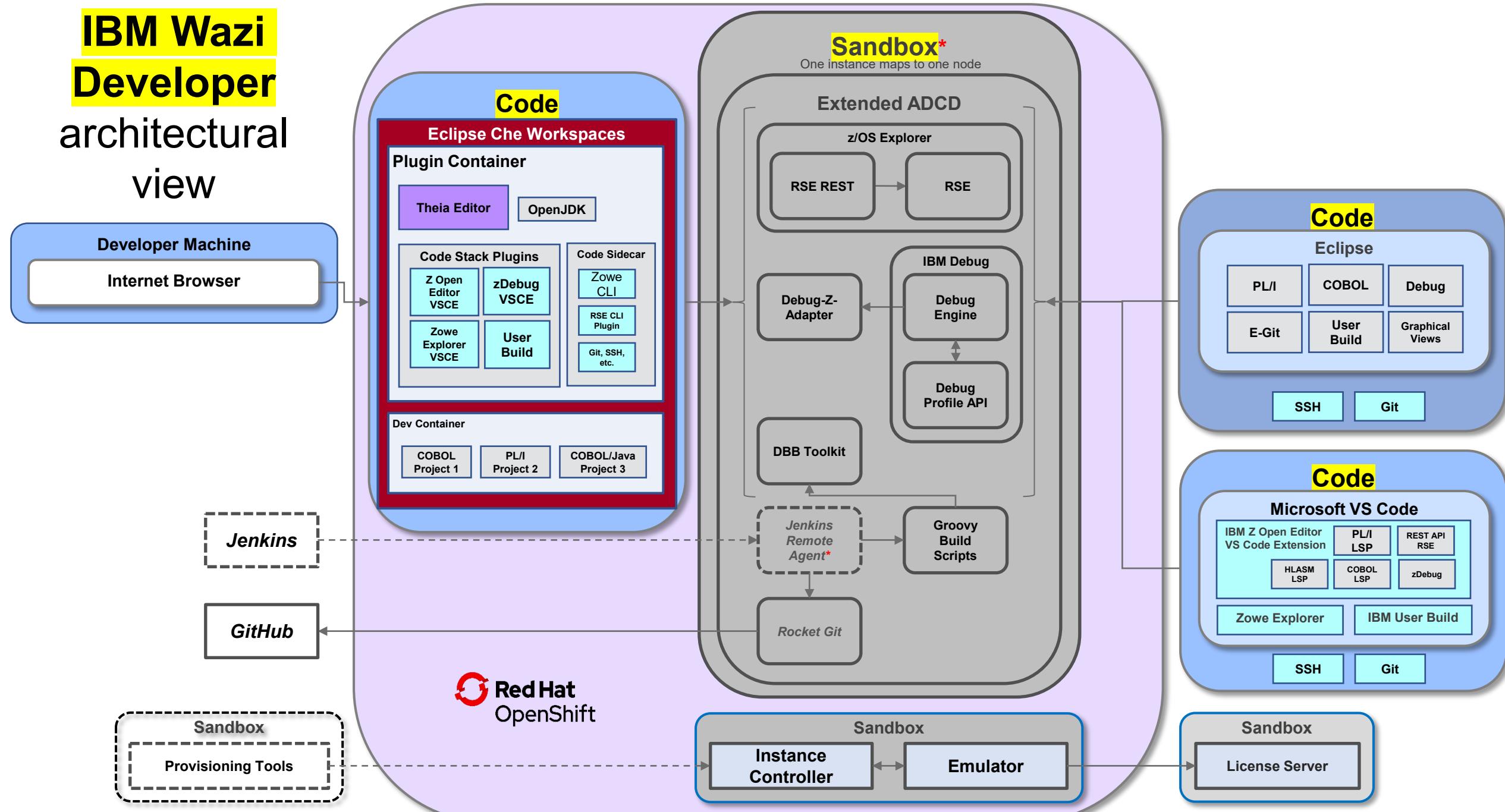
—Shift left with **Z Development and Test Environment**

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

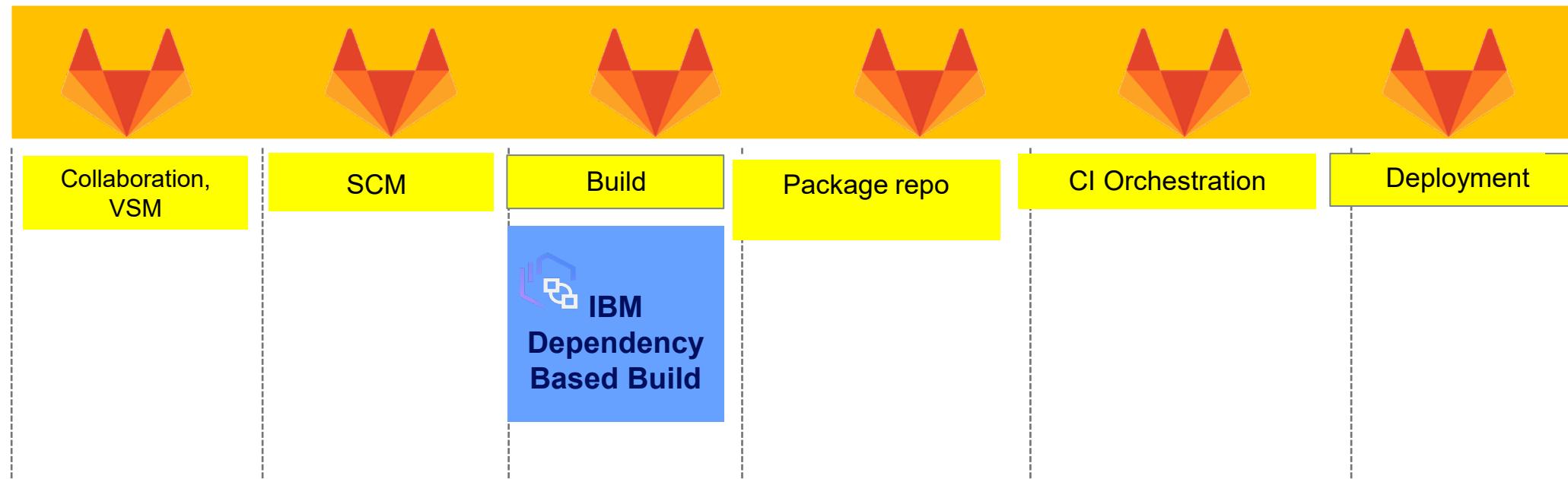


IBM Wazi Developer architectural view



DevOps mainframe pipeline with GitLab Ultimate for z/OS

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



Extend GitLab with Specialists



Jenkins



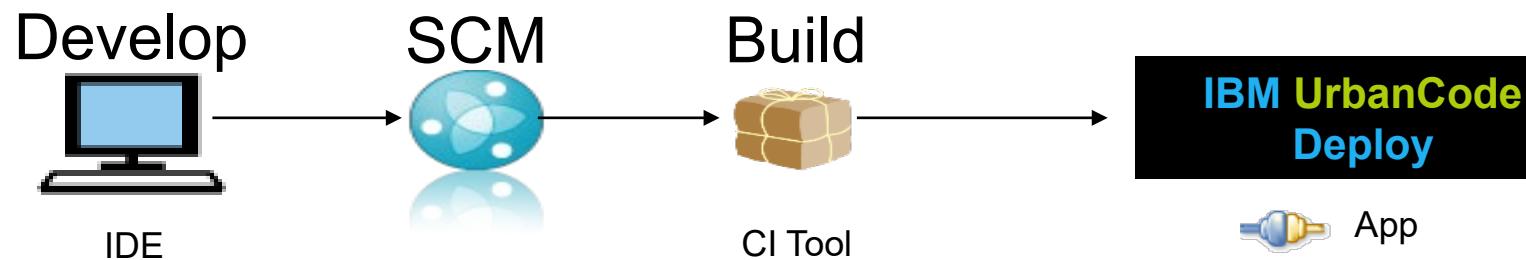
IBM Rational Test

urban{code}

Velocity (VSM, data,
pipeline orchestration)

UrbanCode for Deployment automation (UCD)

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

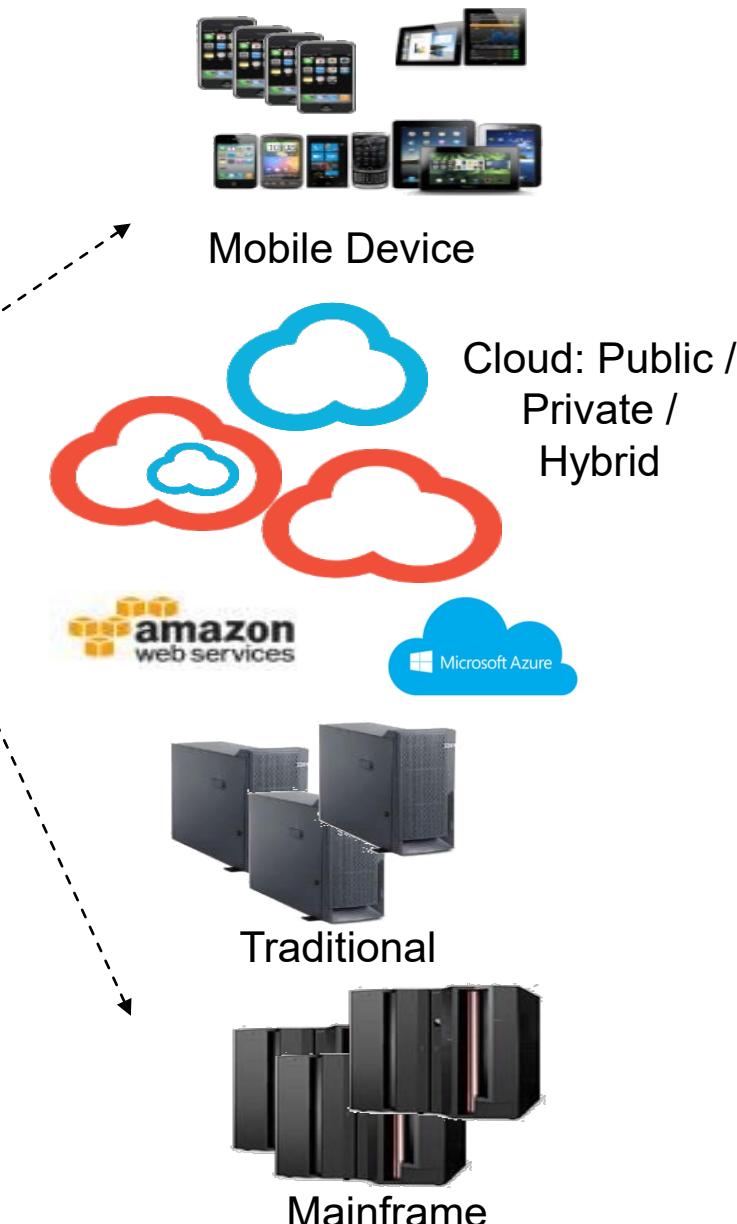


IBM UrbanCode Deploy automates the deployment of applications, databases and configurations into development, test and production environments, helping to drive down cost, speed time to market with reduced risk.

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

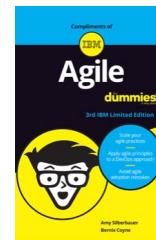
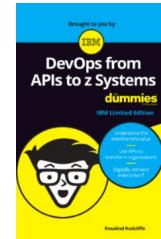
Integrates with Jenkins

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:
 - [Downloads - Mainframe DEV \(ibm.github.io\)](https://ibm.github.io/Mainframe-DEV/)
- IBM DevOps for Enterprise Systems:
 - <https://www.ibm.com/it-infrastructure/z/capabilities/enterprise-devops>
- For Dummies books:
 - <https://www.ibm.com/ibm/devops/us/en/resources/dummiesbooks/>
- IBM Z Trial:
 - <https://ibm.biz/z-trial>



QUESTIONS?



Agenda - Day 1

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

→ 3:00 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z

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

Agenda - Day 2

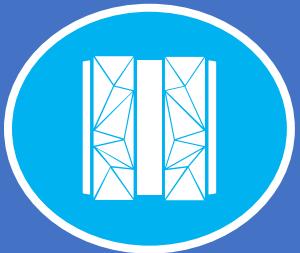
2:00 zUnit and VTP Overview (Wilbert)

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

3:00 – 4:50 Choose one Optional lab:

- LAB 7 - IBM DBB with Git, Jenkins and UCD on Z
- LAB 6B : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Remote Assets
- LAB 6C : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Local Assets
- LAB 6D : Using IBM zUnit to Unit Test a COBOL/DB2 batch program
- LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
- LAB 8 - Using Application Performance Analyzer (APA)
- LAB 9 - Using IBM Z VTP to test a COBOL /CICS / DB2 transaction
- LAB 10 - Deploying COBOL/CICS/DB2 application using a GitLab CI Pipeline
- LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- LAB 12 – Using ADDI to find CICS programs to be converted to API's
- LAB 13 – Running Integration Tests using Galasa

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



DevOps Mainframe Tooling - Labs

Labs

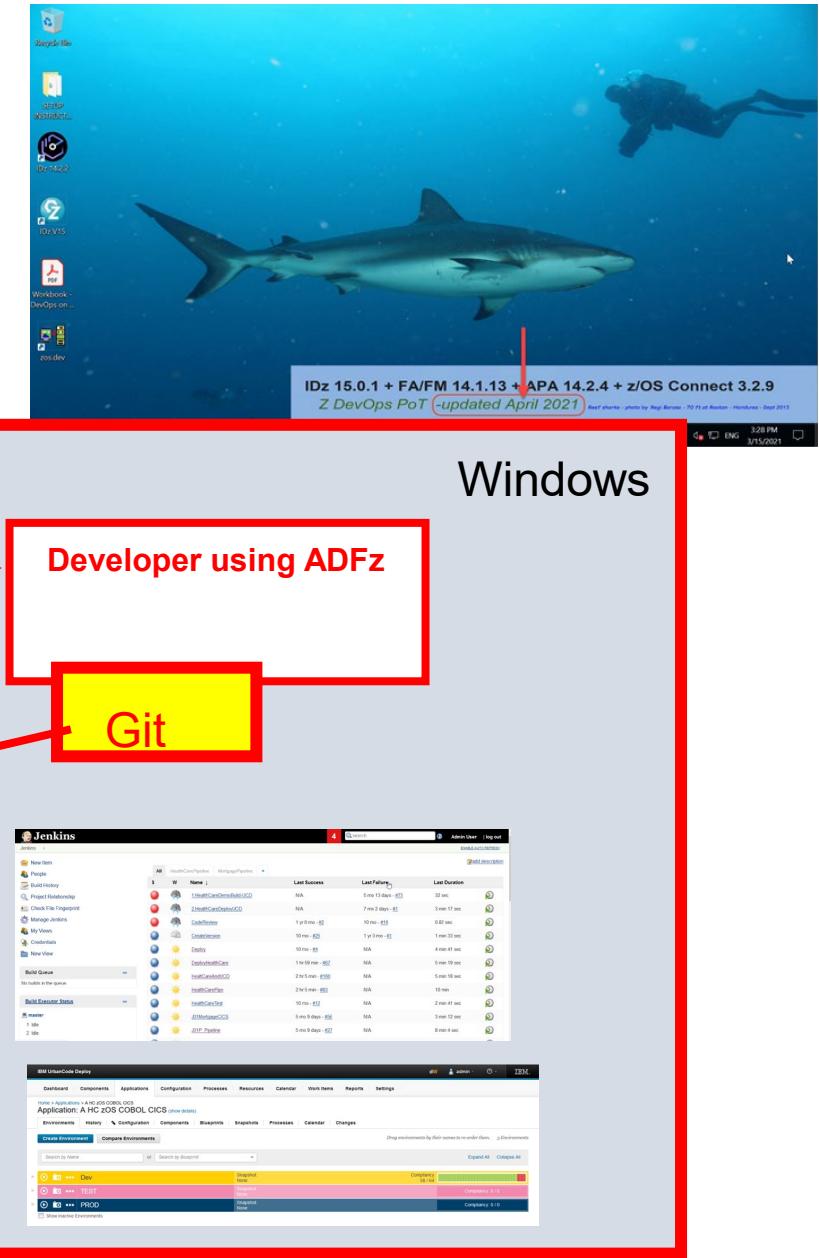
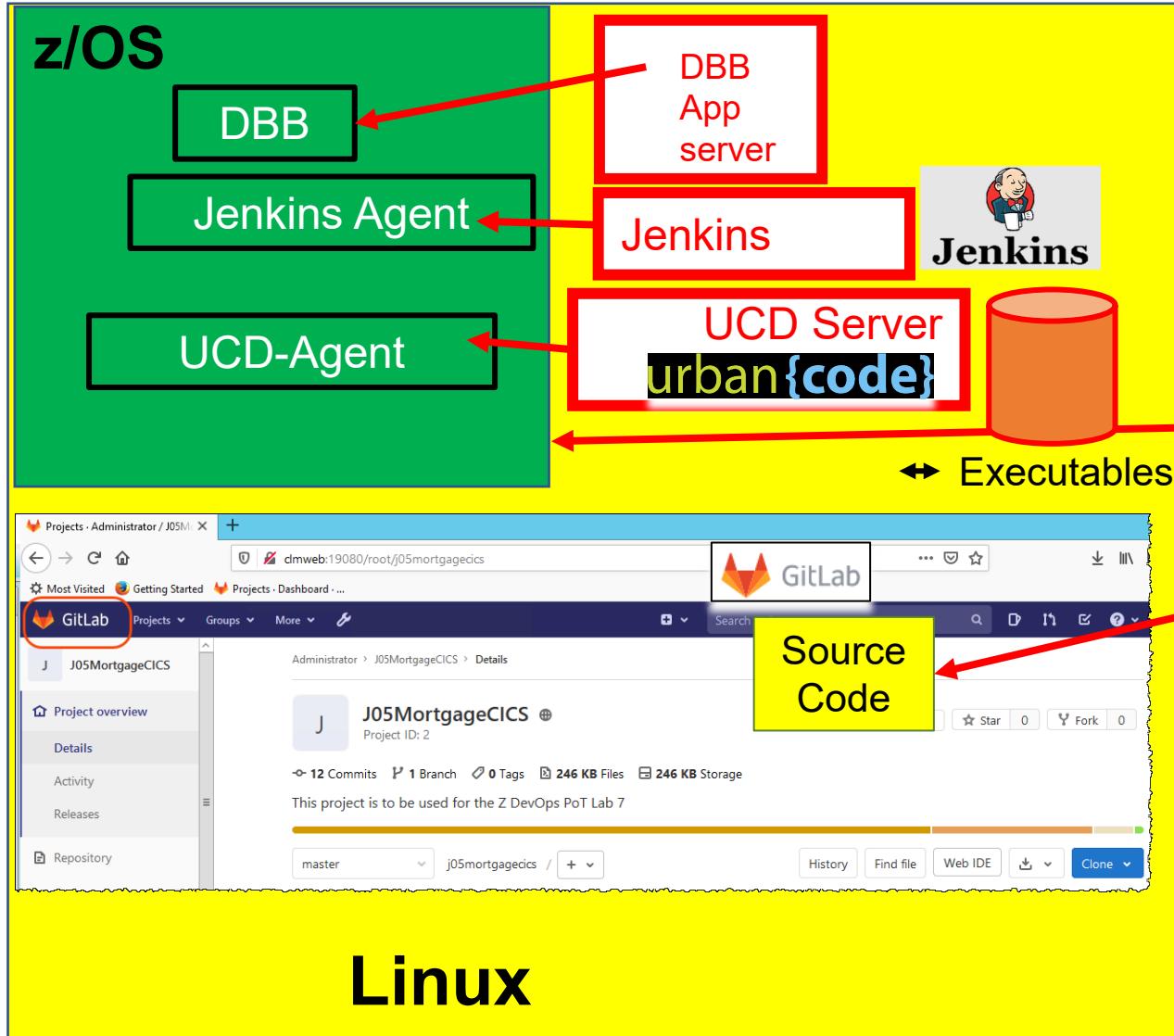
Depending on the LAB...

1. As IDE you will use either **IDz** version 14.2.4 or **IDz** V 15.0.1.
2. Use the correct z/OS user id: **EMPOT01** or **EMPOT05** or **IBMUSER** indicated on the exercise.
3. Each time that you see the symbol it means that you have to “do something” on your computer – not merely read the explanations.

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

Topology used on the labs

zD&T on Cloud



Lab 1: Working with mainframe using COBOL and DB2

Duration: 90 Minutes

■ Objective

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

Key Activities

1. [Connect to a z/OS System.](#)

Use your Workspace to connect to the z/OS system. Each student will have an unique z/OS userid.

2. [Execute the DB2/COBOL batch program and verify the ABEND.](#)

3. [Use Fault Analyzer to identify the cause of the ABEND](#)

4. [Use the IBM Debug for a temporary fix](#)

You will modify the field content to bypass the bug

5. [Modify the COBOL code to fix the bug.](#)

6. [Use Code Coverage](#)

7. [\(Optional\) Execute SQL statement when editing the program.](#)

While editing the COBOL/DB2 program you will be able to execute SQL statements and verify the results.

8. [\(Optional\) Using File Manager](#)

An example of using File Manager against a VSAM file

Lab 1: Working with mainframe using COBOL and DB2

zos.dev.2800/IDID10.HIST(F00307)-Report

```
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

DB2REGI → 000-SETUP-ERROR-TRAP-RTN → 000-MAINLINE-RTN → 200-FETCH-RTN → 520-LOGIC → 530-SEEEYA → 540-GOODBYE → 999-EXIT

100-DECLARE-CURSOR-RTN → 100-EXIT
150-OPEN-CURSOR-RTN → 150-EXIT
250-FETCH-A-ROW → 250-EXIT
300-CLOSE-CURSOR-RTN → 300-EXIT
350-TERMINATE-RTN → 350-EXIT → 500-SECOND-PART → 520-LOGIC

Here calls REG10B that abends

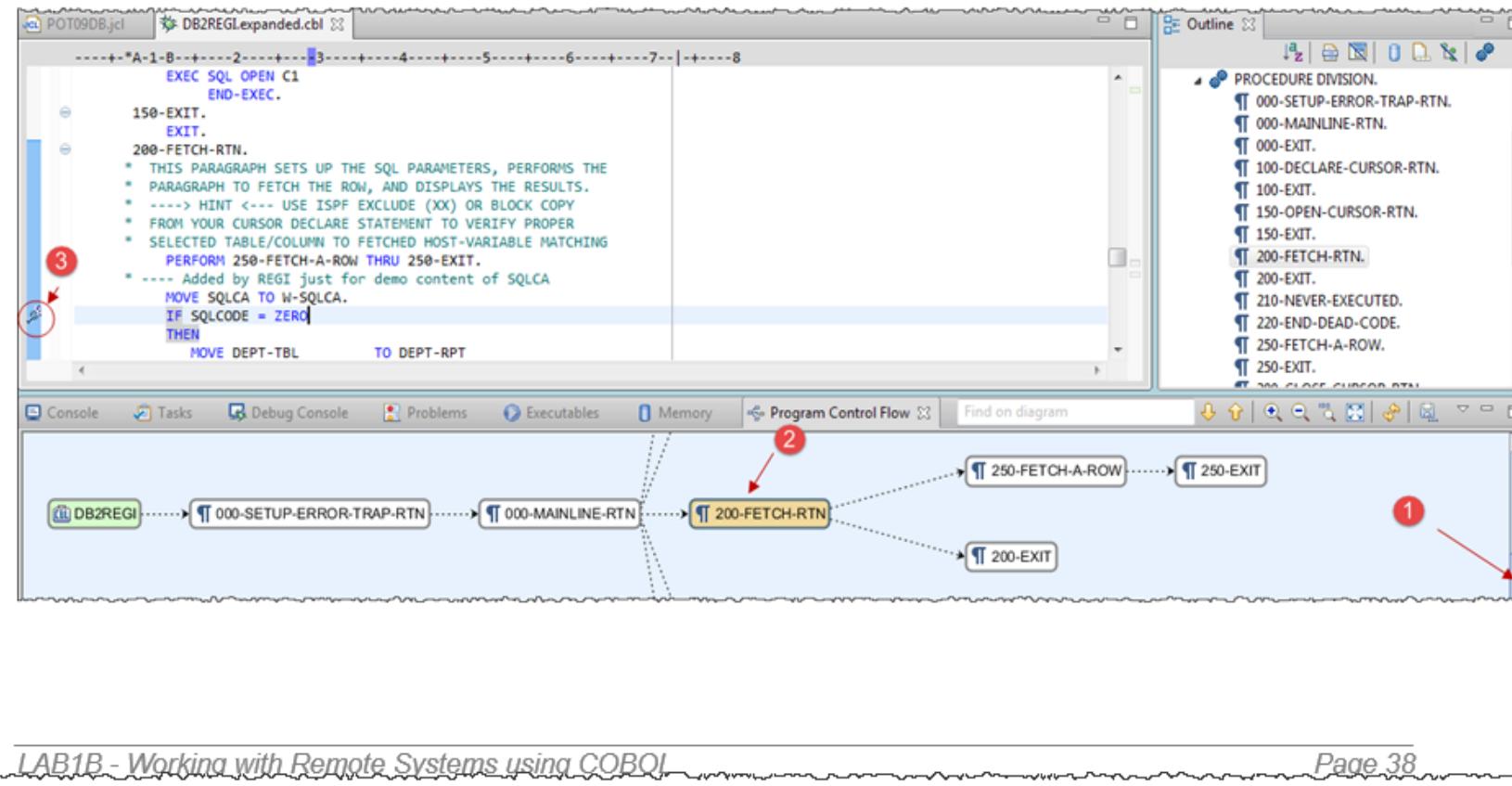
Lab 1: Don't Miss... The Debug... Page 22

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 sections like EXEC SQL OPEN C1, 150-EXIT, and 200-FETCH-RTN. A red circle labeled 3 points to the IF SQLCODE = ZERO line. The bottom part is the Program Control Flow diagram. It starts with a green rounded rectangle labeled DB2REGI, followed by several blue rounded rectangles representing procedures: 000-SETUP-ERROR-TRAP-RTN, 000-MAINLINE-RTN, 200-FETCH-RTN, 250-FETCH-A-ROW, 250-EXIT, and 200-EXIT. Arrows show the flow from DB2REGI to 000-SETUP-ERROR-TRAP-RTN, then to 000-MAINLINE-RTN, then to 200-FETCH-RTN. From 200-FETCH-RTN, arrows point to 250-FETCH-A-ROW and another 200-EXIT node. A red circle labeled 1 points to the vertical blue bar on the right side of the editor window. A red circle labeled 2 points to the 200-FETCH-RTN node in the Program Control Flow diagram.

Lab 1: Don't Miss... Code Coverage ...Page 48

8.2.4 Double click on DB2REGI.expanded.cbl

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

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

This lab will use [IBM Dependency Based Build](#) (DBB) along with [GitLab](#), [Jenkins](#) and [UrbanCode Deploy](#) (UCD) on z/OS.

You will modify an existing COBOL/CICS application stored on Git.

You will use ADFz to change the code and perform a personal test for later delivery and commit to Git and then use Jenkins for the final build and continuous delivery.

The updated code will be deployed to CICS using UrbanCode Deploy (UCD)

Overview of development tasks

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/link and perform personal tests.

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

5. Push and Commit the changed code to Git .

→ You will commit the changes to Git.

6. Use Jenkins with Git plugin to build the modified code

→ You will use Jenkins pipeline to build the new changed code and push the executables to be deployed using UCD.

7. Use Jenkins and UCD plugin to deploy results and test the code again

→ You will verify the results after the final deploy to CICS using UCD

8. (Optional) Understanding DBB Build Reports

→ You will understand the reports generated by DBB during the build

Agenda - Day 1

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

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

or

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

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

Agenda - Day 2

2:00 zUnit and VTP Overview (Wilbert)

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

3:00 – 4:50 Choose one Optional lab:

- LAB 7 - IBM DBB with Git, Jenkins and UCD on Z
- LAB 6B : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Remote Assets
- LAB 6C : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Local Assets
- LAB 6D : Using IBM zUnit to Unit Test a COBOL/DB2 batch program
- LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
- LAB 8 - Using Application Performance Analyzer (APA)
- LAB 9 - Using IBM Z VTP to test a COBOL /CICS / DB2 transaction
- LAB 10 - Deploying COBOL/CICS/DB2 application using a GitLab CI Pipeline
- LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- LAB 12 – Using ADDI to find CICS programs to be converted to API's
- LAB 13 – Running Integration Tests using Galasa

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

Agenda - Day 1

- 2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)
3:00 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z

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

Agenda - Day 2

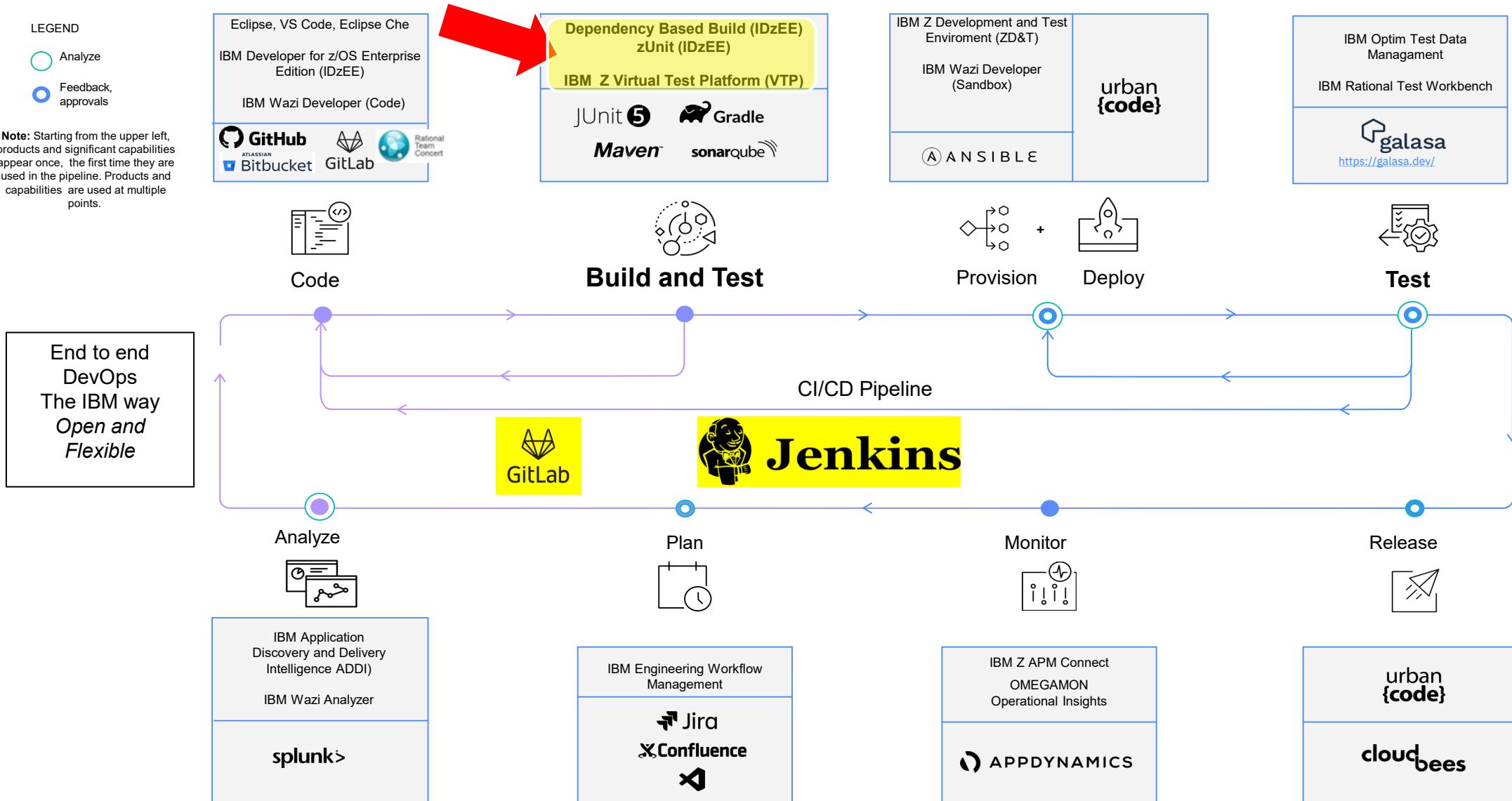
- 2:00 zUnit and VTP Overview (Wilbert)

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

3:00 – 4:50 Choose one Optional lab:
→ LAB 7 - IBM DBB with Git, Jenkins and UCD on Z
→ LAB 6B : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Remote Assets
→ LAB 6C : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Local Assets
→ LAB 6D : Using IBM zUnit to Unit Test a COBOL/DB2 batch program
→ LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
→ LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
→ LAB 8 - Using Application Performance Analyzer (APA)
→ LAB 9 - Using IBM Z VTP to test a COBOL /CICS / DB2 transaction
→ LAB 10 - Deploying COBOL/CICS/DB2 application using a GitLab CI Pipeline
→ LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
→ LAB 12 – Using ADDI to find CICS programs to be converted to API's
→ LAB 13 – Running Integration Tests using Galasa

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

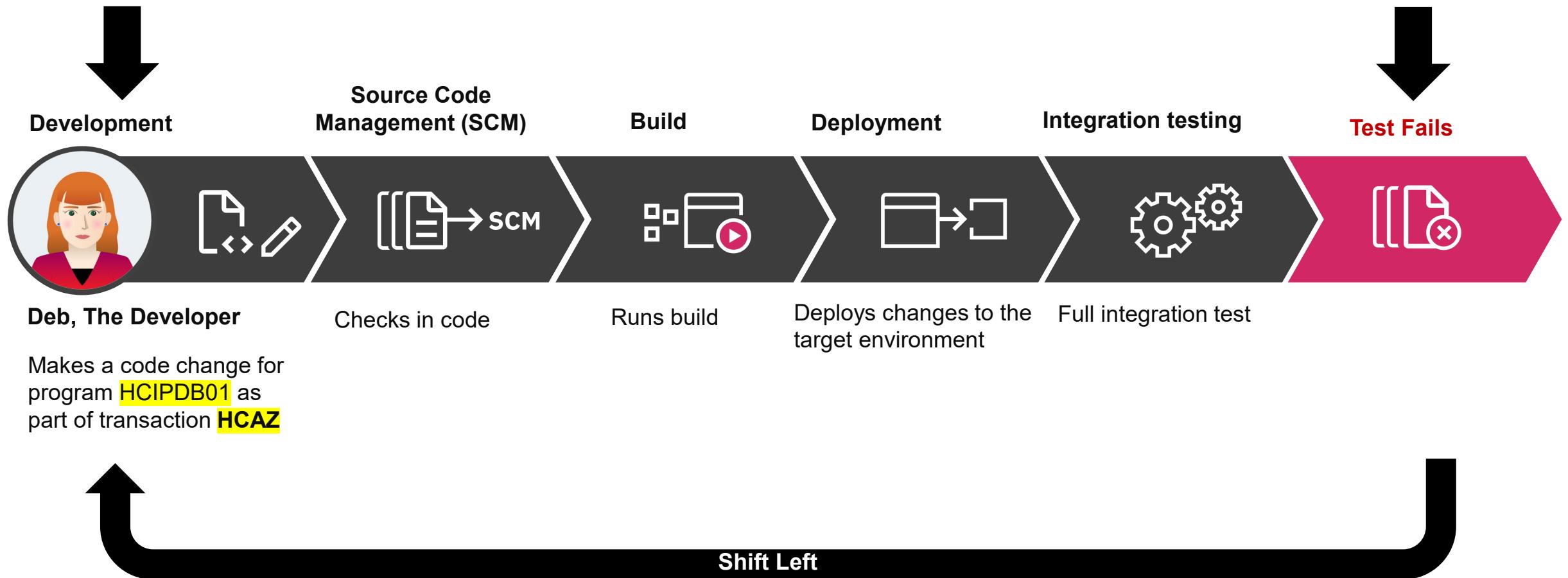
Z DevOps Pipeline



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

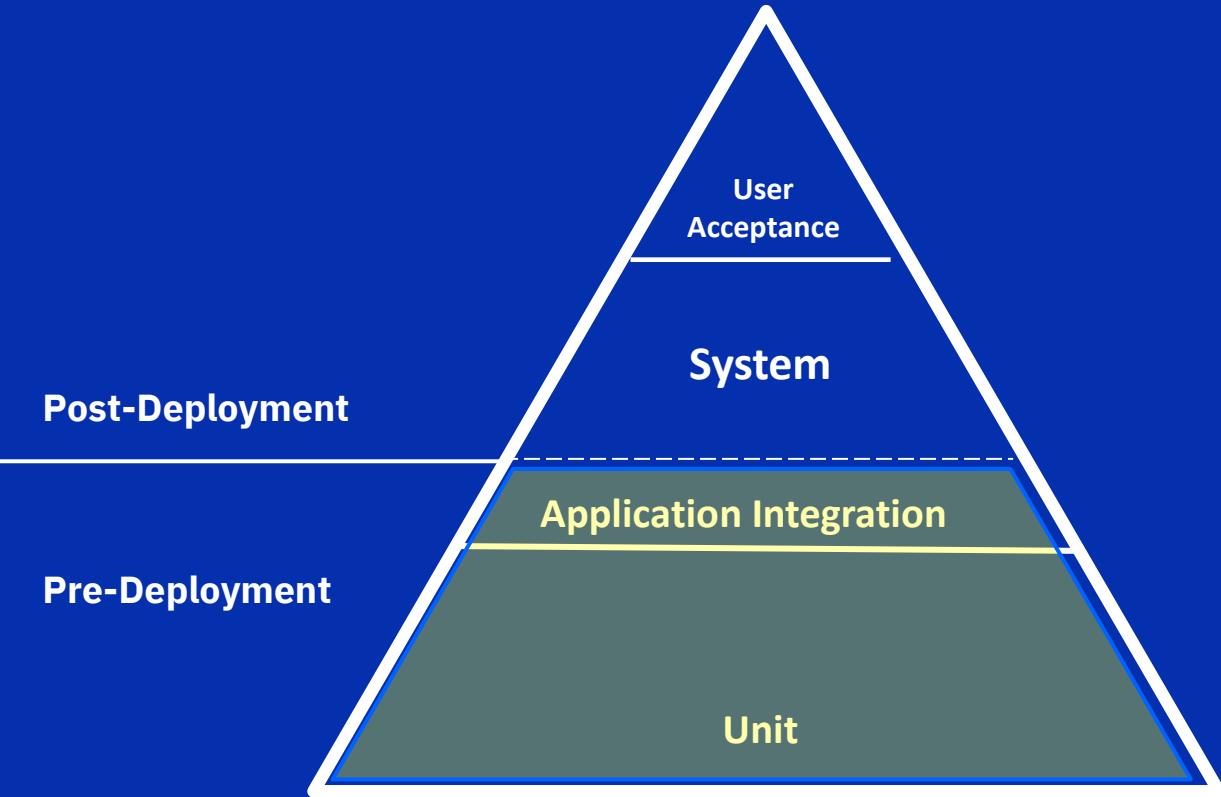
RTW (Rational Integration Tester/Rational functional tester/Test Virtualization Server)

Application Integration test

IBM Virtual Test Platform

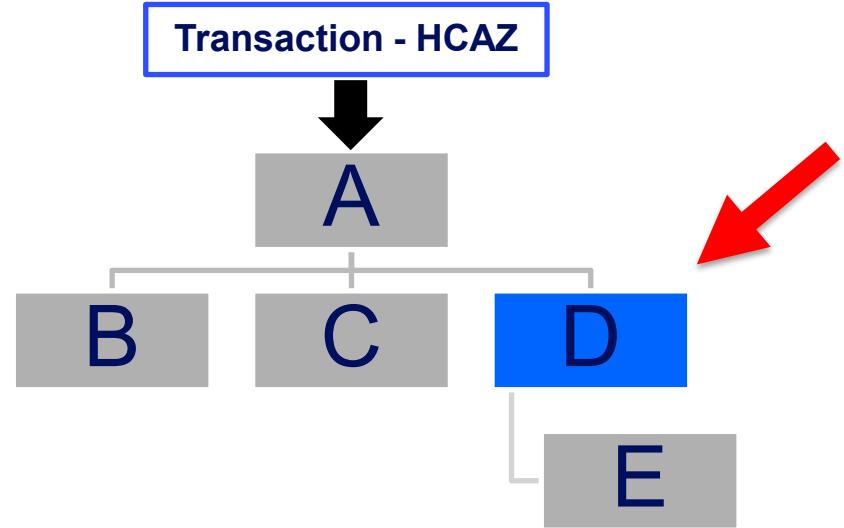
Unit test

zUnit (IBM Developer for z/OS)



Unit Testing (zUnit)

Scope - Program



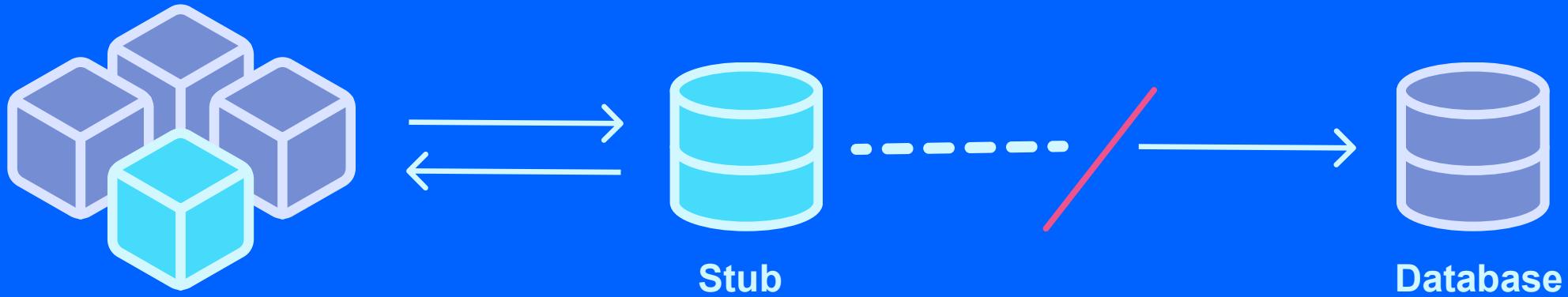
A unit is the smallest testable part of software – meaning testing an individual unit of source code, like a single source file which contains a list of methods, procedures.

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

Shift Left – unit testing with ZUnit

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

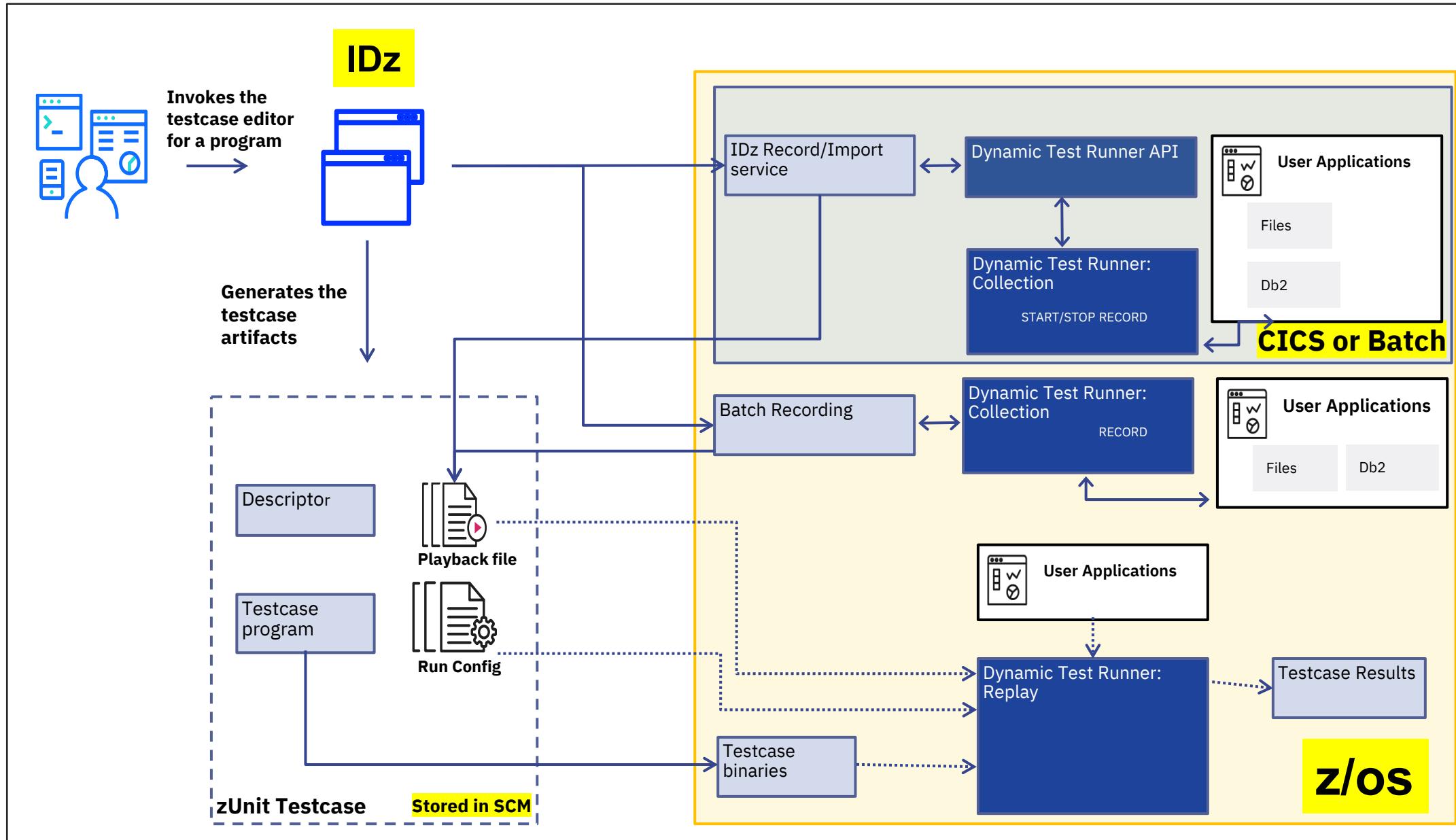
This is done at the **program level**



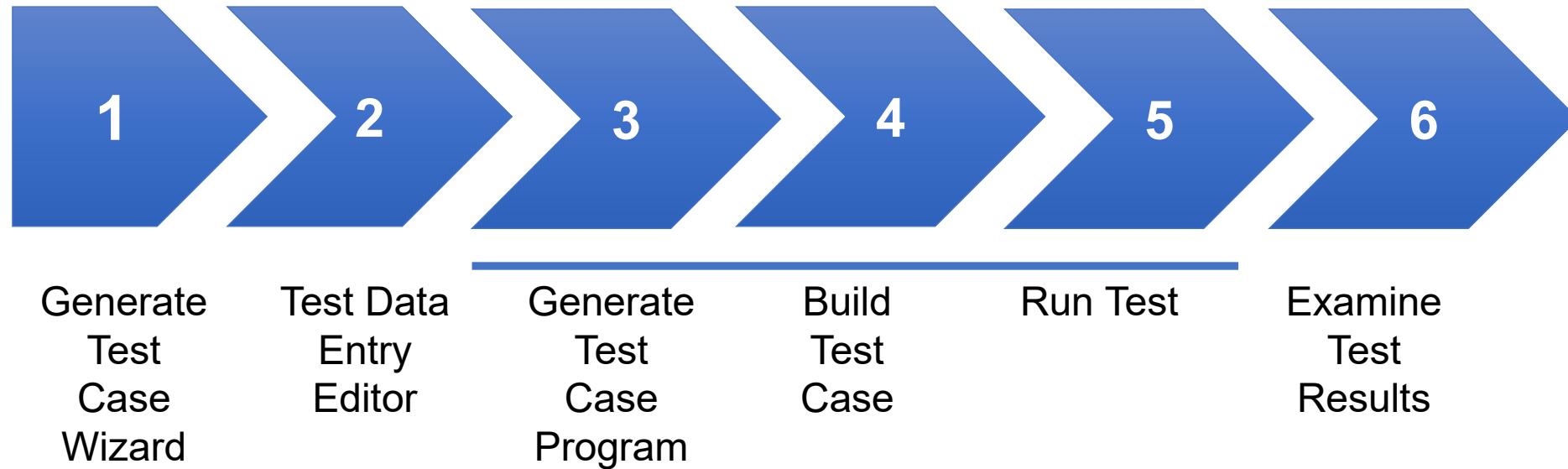
```
1 //IBMUSER1 JOB ,  
2 // MSGCLASS=H,MSGLEVEL=(1,1),REGION=0M,COND=(16,LT)  
3 /* Action: Run Test Case... NO CICS or DB2 required  
4 /* TEST CASE Module at: IBMUSER.GIT.ZMOBILE.LOAD(THCIPDB0)  
5 // SET FELJOB=ZUNITGO  
6 // RUNNER EXEC PROC=BZUPPLAY,  
7 // BZUCFG=IBMUSER.ZUNIT.BZUCFG(THCIPDB0),  
8 // BZUCBK=IBMUSER.GIT.ZMOBILE.LOAD,  
9 // BZULOD=IBMUSER.GIT.ZMOBILE.LOAD,  
10 // PRM='STOP=E,REPORT=XML'  
11 // REPLAY.BZUPPLAY DD DISP=SHR,  
12 // DSN=IBMUSER.ZUNIT.PB.HCIPDB01 *PLAYBACK File  
13 // REPLAY.BZURPT DD DISP=SHR,  
14 // DSN=TRMUSER.ZUNIT.RZURES(THCTPDB0),  
15 /* Below is where is the program to be TESTED  
16 // STEPLIB DD DISP=SHR,DSN=IBMUSER.GIT.ZMOBILE.LOAD
```

No CICS and DB2 required

ZUnit High Level Architecture

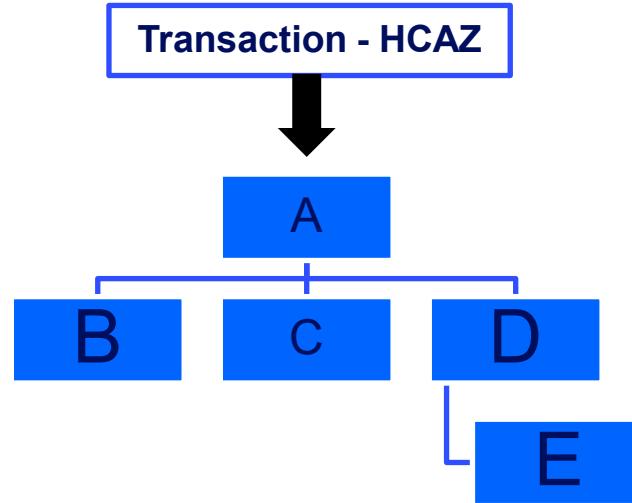


zUnit Basic Workflow



Application Integration test **(IBM Z VTP)**

Scope - Transaction



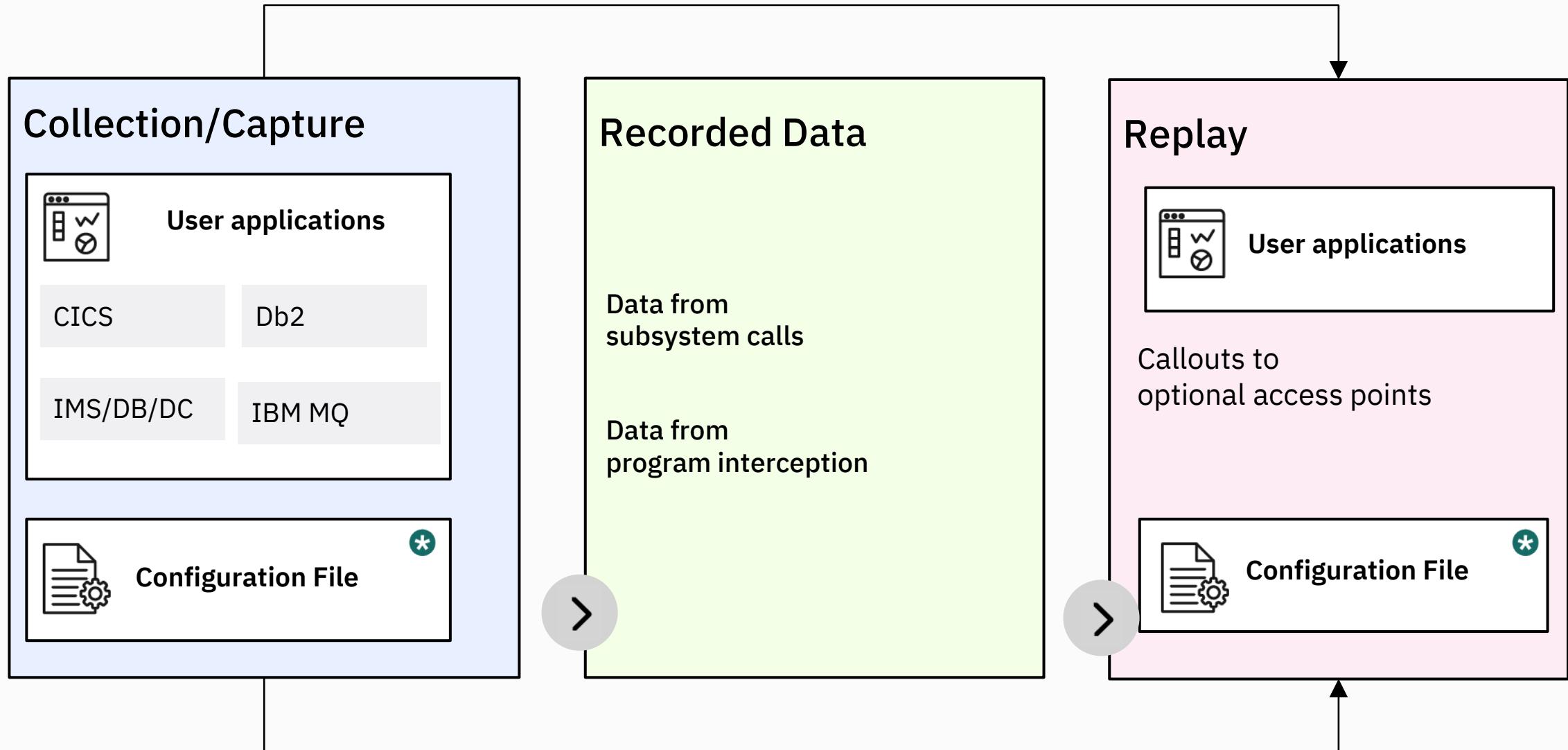
Application integration test is the next level of testing after Unit Testing. Its purpose is to confirm that the changes in a unit (programs) works with the interfaces to other units (programs), and ensure no unexpected impacts to other units

Transaction and batch jobs

```
1 //HCAZPLAY JOB ,NOTIFY=&SYSUID,REGION=0M
2 /* Run IBM Z VTP for HCAZ Application
3 //PLAYBK EXEC PGM=BZUPLAY,PARM='TRACE=N'
4 //STEPLIB DD DISP=SHR,DSN=BZU100.SBZULOAD
5 /* Below are the programs to be tested
6 // DD DISP=SHR,DSN=EMPOT.ZMOBILE.TEST.LOAD
7 /**
8 //BZUPLAY DD DISP=SHR,DSN=BZU100.ZUNIT.PLAYBACK.DEMO *playback file
9 //SYSOUT DD SYSOUT=*
10 //BZUMSG DD SYSOUT=*
```

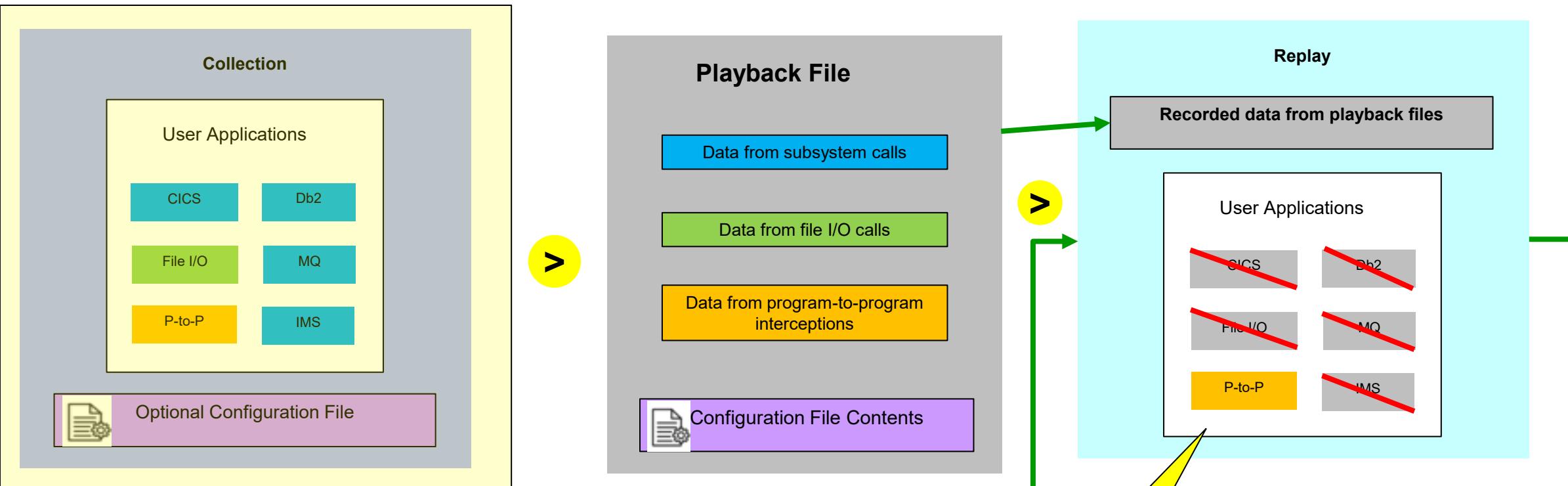
Testing with Z Virtual Test Platform

Part 1 – Capture or Collection



Z VTP common use case

Any LPAR or **zD&T**



Original LPAR

Recorded Data

Continue with
program changes

Middleware
not required
for replay

Galasa test automation framework

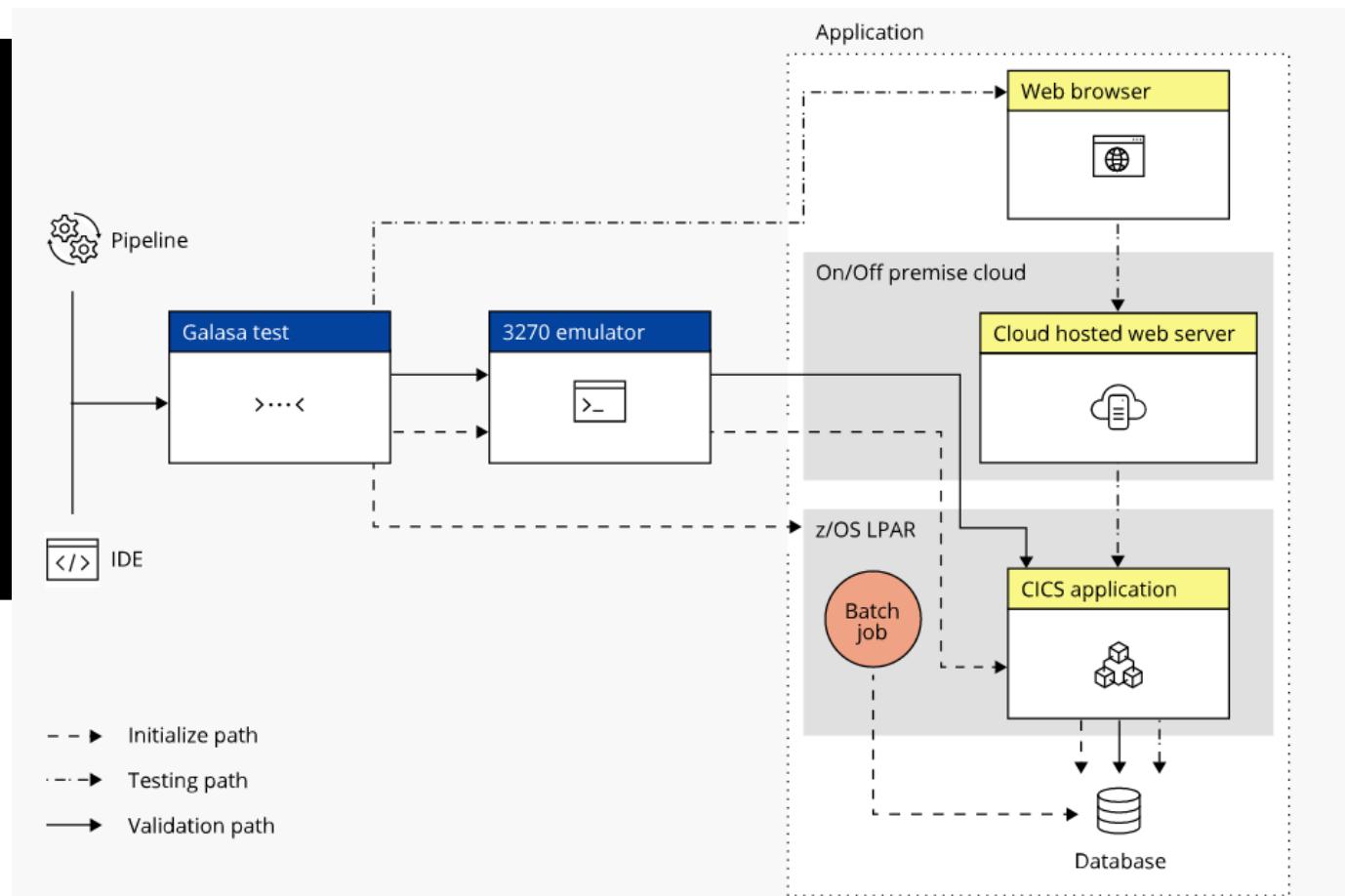
Run repeatable, reliable, automated tests and deliver software changes faster, with confidence



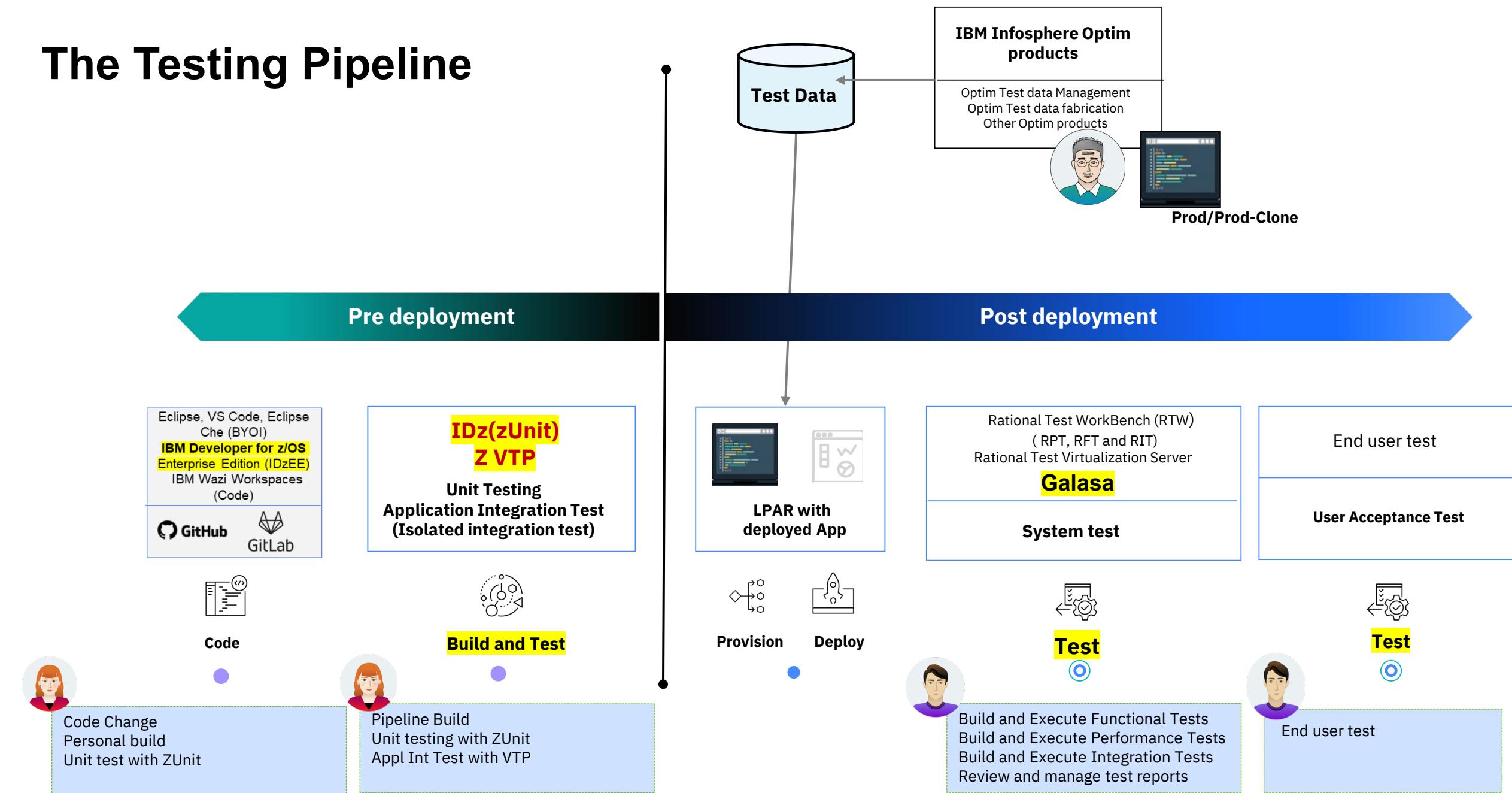
<https://galasa.dev>

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

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



The Testing Pipeline



QUESTIONS?



Agenda - Day 1

- 2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)
3:00 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z
5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

2:00 zUnit and VTP Overview (Wilbert)

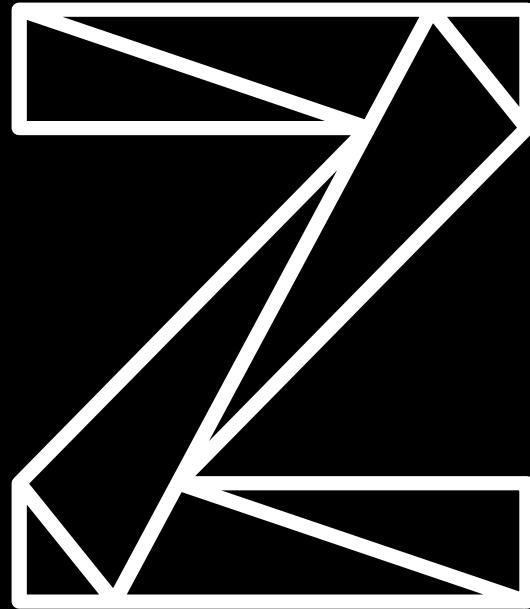
→ 2:15 Demo: Scenario using Z DevOps solutions including ADDI, zUnit, Git and Jenkins (David/Regi)

3:00 – 4:50 Choose one Optional lab:

- LAB 7 - IBM DBB with Git, Jenkins and UCD on Z
- LAB 6B : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Remote Assets
- LAB 6C : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Local Assets
- LAB 6D : Using IBM zUnit to Unit Test a COBOL/DB2 batch program
- LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
- LAB 8 - Using Application Performance Analyzer (APA)
- LAB 9 - Using IBM Z VTP to test a COBOL /CICS / DB2 transaction
- LAB 10 - Deploying COBOL/CICS/DB2 application using a GitLab CI Pipeline
- LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- LAB 12 – Using ADDI to find CICS programs to be converted to API's
- LAB 13 – Running Integration Tests using Galasa

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

IBM Z DevOps Tools in action



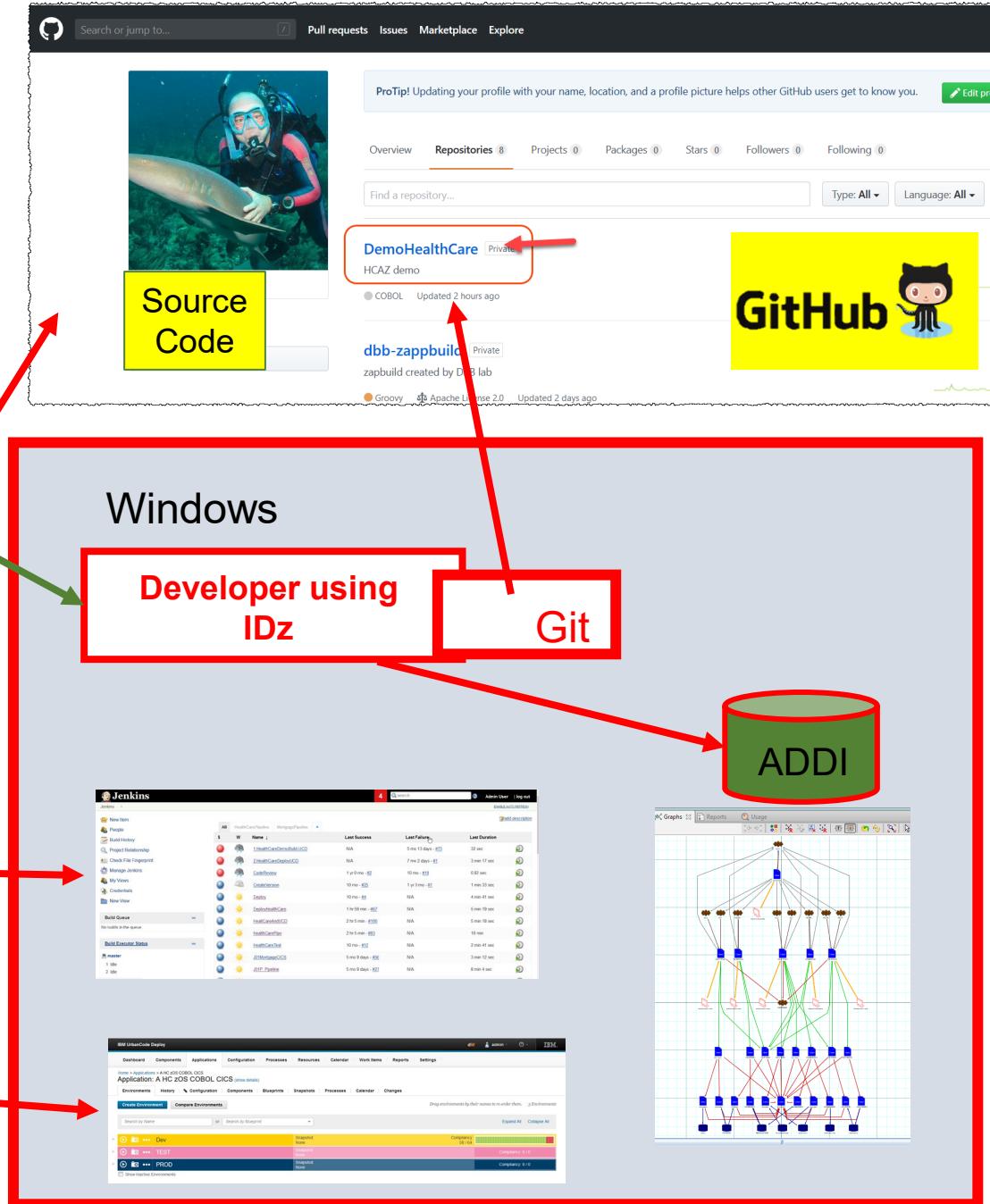
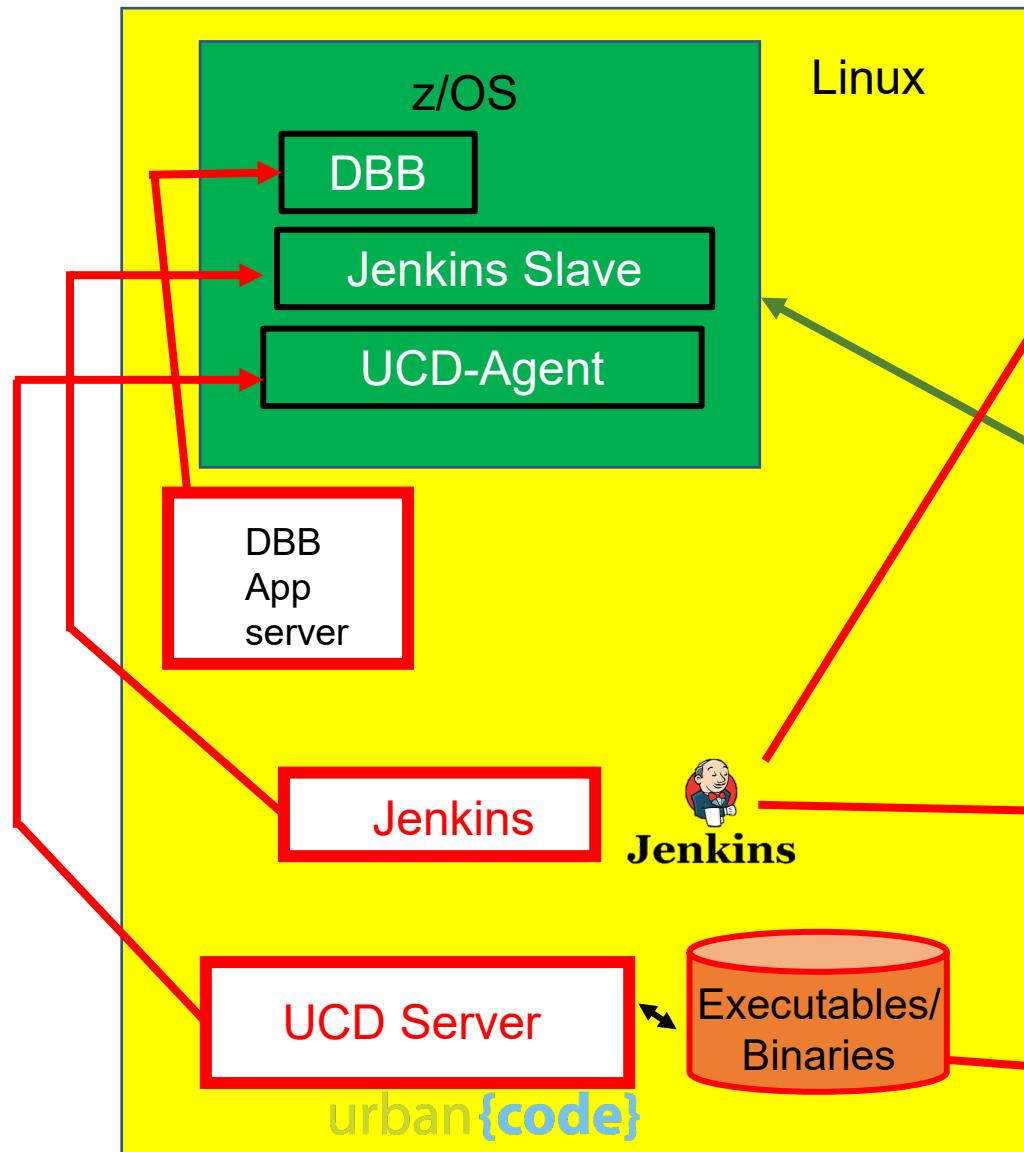
Demonstration

Session Objectives

- We will demonstrate how IBM DevOps and open source tools like [Git](#) and [Jenkins](#) help the continuous integration, continuous testing, continuous delivery on the z/OS space.
- Our scenario consists of a “broken” [COBOL/CICS/DB2](#) application. The developer will use IBM DevOps tools to understand the application (ADDI), locate and identify the faulty code, make the appropriate fixes and compile/link/bind using a new build automation tool ([DBB](#)) that integrates with [Git](#) and [Jenkins](#) pipeline on z/OS. Once the code is fixed, the developer will push the changes to a Git repository..
- In order to demonstrate continuous delivery capabilities, this session will showcase a [Jenkins](#) pipeline integrated with IBM tools for the build process, unit testing(using the new [zUnit](#) functionality), and deployment of code to a CICS environment (using [UrbanCode Deploy](#)).
- While our scenario uses a [COBOL/CICS/DB2](#) program as an example, the process is similar to other environments or languages (such as [Batch](#), [IMS](#), or [PL/1](#)

Topology used on this Demo

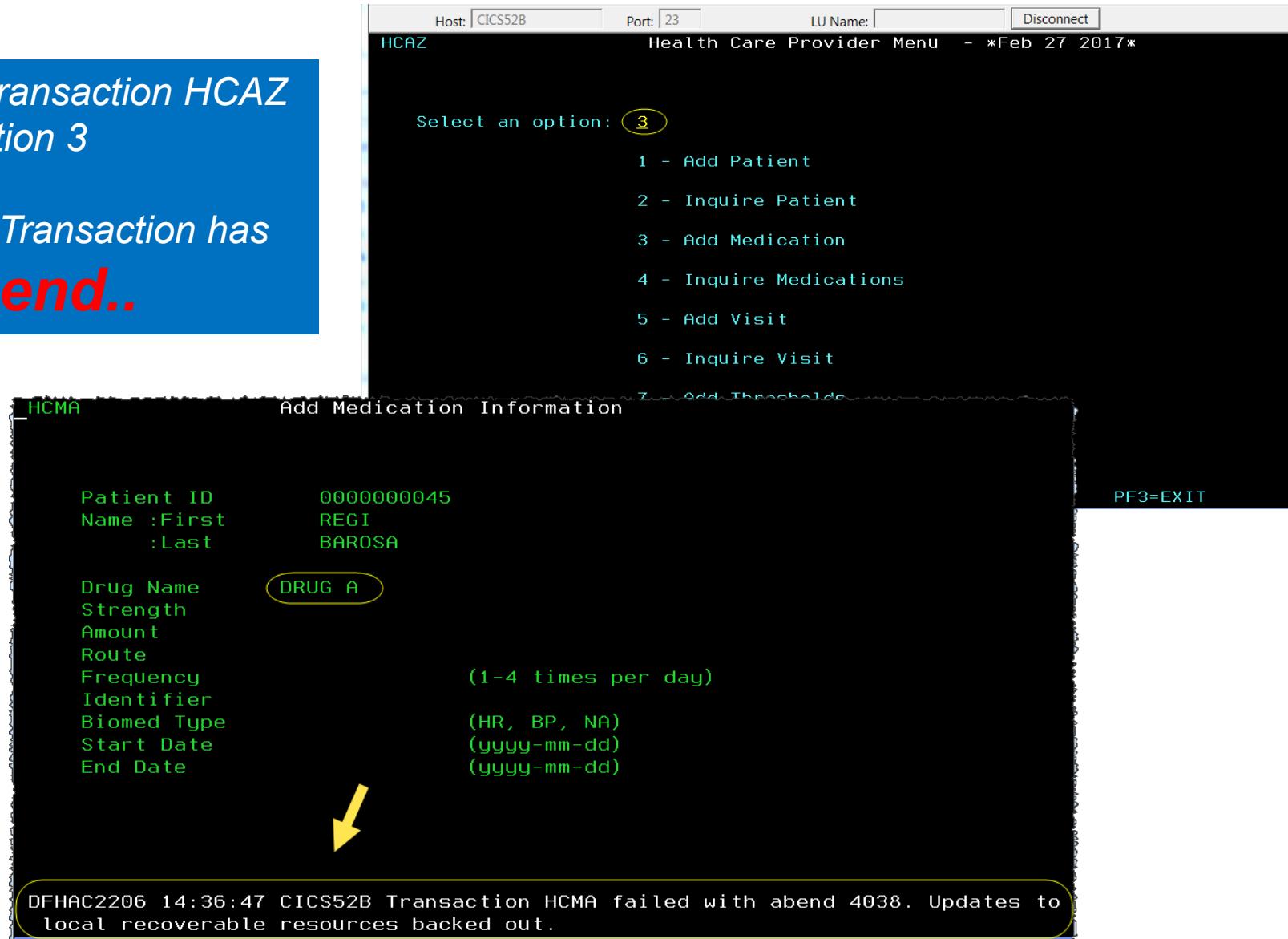
zD&T on VMWARE



Scenario: CICS Transaction ABEND

*Using transaction HCAZ
and option 3*

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



PART #1 - Understand the Application Components

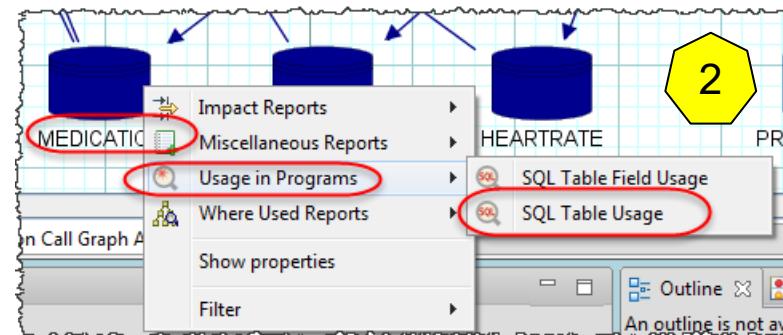
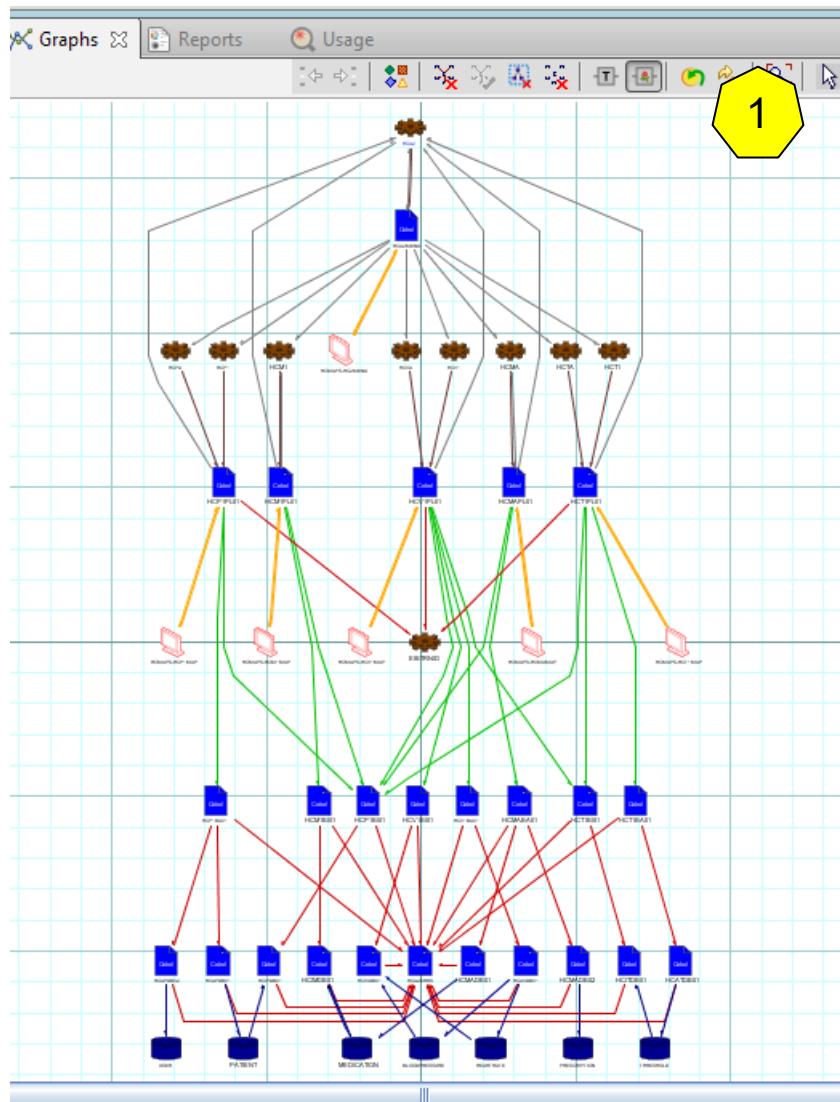
Objective: Fix the CICS Abend

Alex - Project Manager

Performs application analysis

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

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



The screenshot shows the HCAZ interface with a tree view under 'RESORTS' titled 'MEDICATION'. It lists several COBOL programs and their associated SQL statements. One program, 'HCMADB01', is expanded to show specific SQL statements like 'SQL: CLOSE', 'SQL: FETCH', etc. A yellow hexagon labeled '3' is positioned in the top right corner of the code editor area. Below the tree view, a code editor window titled 'HCMADB01.cbl' displays the COBOL source code, which includes comments indicating the execution of SQL statements.

```
--*-A-1-B--+-2-+3-+4-+5-+
158 *
159 MOVE ' INSERT MEDICATION' TO EM-SQLREQ
160 *
161 EXEC SQL
162     INSERT INTO MEDICATION
163     (
164         MEDICATIONID,
165         PATIENTID,
166         DRUGNAME,
167         STRENGTH,
168         AMOUNT,
169         ROUTE,
170         FREQUENCY,
171         IDENTIFIER,
172         TYPE )
173     VALUES (
174         DEFAULT,
175         :DB2-PATIENT-ID,
176         :CA-DRUG-NAME,
```

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

Bob – Mainframe Developer

Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

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

Bob starts CICS transaction HCAZ to verify the abend

====> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
====> Enter L followed by the APPLID
====> Examples: "L TSO", "L CICSTS52", "L IC...
l cicsts53

Type your userid and password, then press ENTER:

Userid ibmuser Groupid
Password -
Language -
New Password

hcaz_ Select an option: 3 ←

1 - Add Patient 2 - Inquire Patient 3 - Add Medication

Add Medication Information

Patient ID 5_

Patient ID Name :First :Last

Patient ID 0000000005

Name :First Regi
:Last Barosa

Drug Name DRUG A

Drug Name Strength
Amount Route
Frequency Identifier
Biomed Type Start Date
Start Date End Date

DRUG A

(1-4 times per day)
(HR, BP, NA)
(yyyy-mm-dd)
(yyyy-mm-dd)

Leave all fields empty

DFHAC2206 14:47:33 CICS52B Transaction HCMA failed with abend 4038. Updates to local recoverable resources backed out.

Enter medication information

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

Bob – Mainframe Developer

Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

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

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

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

```
10
20 Module HCMADB02, program HCMADB02, source line # 300: CICS abend 4038
3 IBM FAULT ANALYZER SYNOPSIS
4
5
6 A CICS abend 4038 occurred in module CEEPLPKA at offset X'CB5F8'.
7
8 The cause of the failure was program HCMADB02 in module HCMADB02. The COBOL
9 source code that immediately preceded the failure was:
10
11 Source
12 Line #
13 -----
14 000300 COMPUTE WS-INTEGER-START-DATE =
15          FUNCTION INTEGER-OF-DATE (WS-WORKING
16          000302
17
18 The COBOL source code for data fields involved in the failure:
19
20 Source
21 Line #
22 -----
23 000070      05 WS-WORKING-DATE      PIC 9(8).
24 000089      05 WS-INTEGER-START-DATE PIC 9(8).
25
26 Data field values at time of abend:
27
28 WS-INTEGER-START-DATE = X'D4C5C4C9C3C1E3C9'
29 WS-WORKING-DATE      = X'404040F040F040F0'
30
31 Important messages:

```

Main Report Event Details Abend Information System-Wide Information Miscellaneous

ZDT : 2800/IDID10.HIST

Lookup Markers

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

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

```
-----+*A-1-B-----2-----3-----4-----5-----6-----7---|-
291 * =%regi ===== Below must be comments to have the abend
292 * = when DEMO is day 1 must change to Compute = 20170901
293 * IF WS-WORKING-DATE < 16010101 or
294 *   WS-WORKING-DATE > 99991231
295 *   MOVE FUNCTION CURRENT-DATE (1:8) TO WS-WORKING-DATE
296 *   COMPUTE WS-WORKING-DATE = WS-WORKING-DATE - 1
297 * END-IF
298 * ++++++
299
300 | COMPUTE WS-INTEGER-START-DATE =
301 |           FUNCTION INTEGER-OF-DATE (WS-WORKING-DATE)
302 |
303 | MOVE WS-END-NUM-DATE TO WS-WORKING-DATE
304 * =%regi ===== Below must be comments to have the abend
305 * =added to fix abend #2 END DATE
306 * IF WS-WORKING-DATE < 16010101 or
307 *   WS-WORKING-DATE > 99991231
308 *   MOVE FUNCTION CURRENT-DATE (1:8) TO WS-WORKING-DATE
309 * END-IF
310 * ++++++
311
312 COMPUTE WS-INTEGER-END-DATE =
313           FUNCTION INTEGER-OF-DATE (WS-WORKING-DATE)
314
315 PERFORM UNTIL WS-INTEGER-START-DATE > WS-INTEGER-END-DATE
316           COMPUTE WS-WORKING-DATE =
317             FUNCTION DATE-OF-INTEGER (WS-INTEGER-START-DATE)
318           MOVE WS-WORKING-DATE TO WS-START-NUM-DATE
319           EVALUATE CA-FREQUENCY
```



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

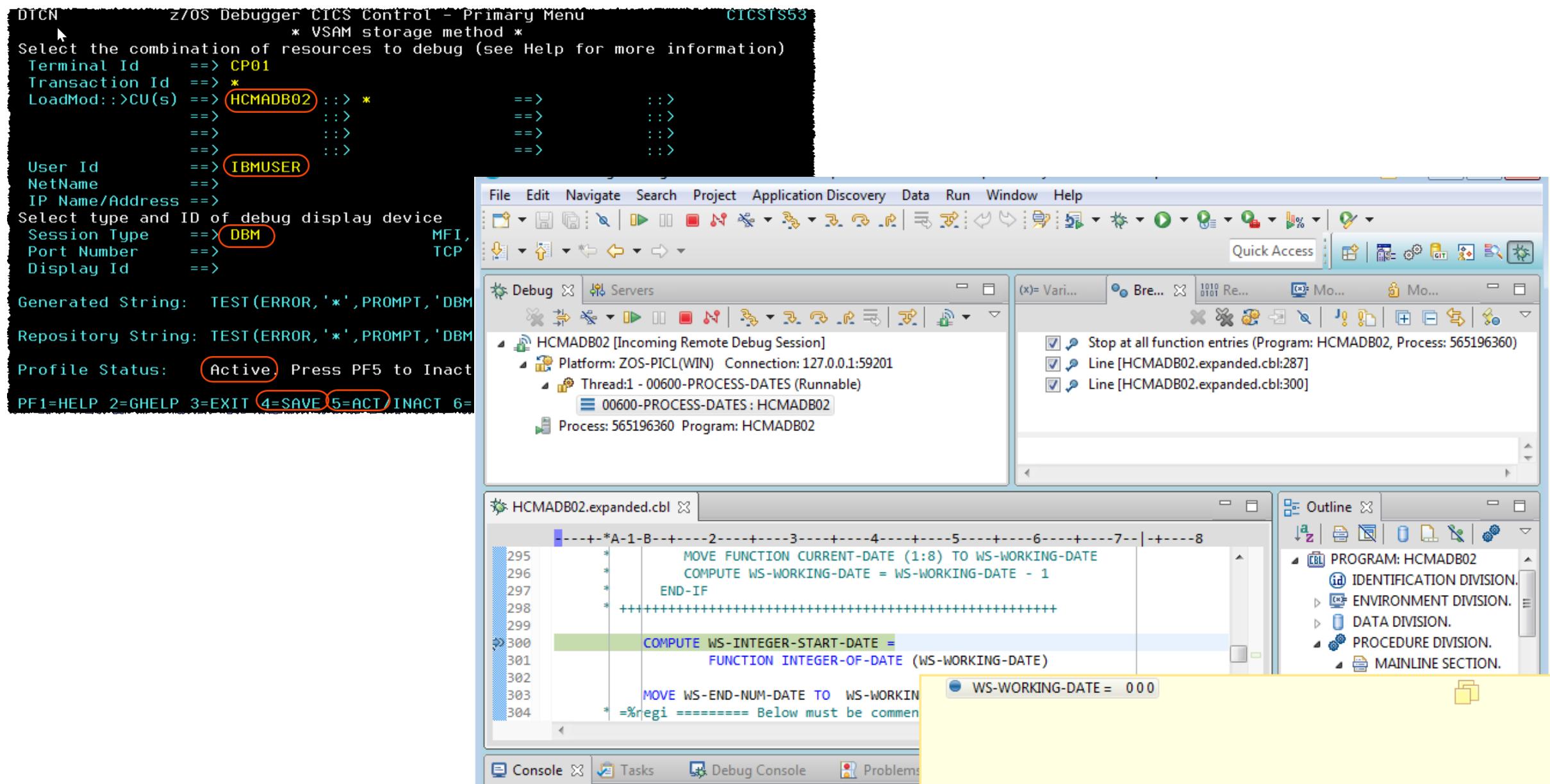
Bob – Mainframe Developer

Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

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

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



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

Bob – Mainframe Developer

Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

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

Uses IDz to update the COBOL program loaded from GIT

The screenshot illustrates the process of updating a COBOL program from GitHub using IBM Developer for z/OS.

GitHub Project Overview: The top half shows the GitHub repository `RegiBrazil / DemoHealthCare`. The `sandbox` branch is selected, highlighted with a red box. The repository has 4 branches and 0 tags. The `Code` tab is active, indicated by a red box around its button.

IBM Developer for z/OS Interface: The bottom half shows the IDE interface. A context menu is open over the `Branches` section of the `Git Repositories` tool window. The `New Branch...` option is highlighted with a red box. Other visible options in the menu include `Switch To`, `Paste Repository Path or URI`, and `Ctrl+V`.

Git Repositories Tool Window: The left pane lists local repositories: `dbb-zappbuild` and `DemoHealthCare`. The right pane shows the `Branches` section with local branches `master`, `regibranch`, and `zappv1`, along with remote tracking branches `zappv1`.

Uses IDz to update the COBOL program loaded from GIT

empot02 - DemoHealthCare/cobol_cics_db2/HCMADB02.cbl - C:\Workspace.IDz15\empot02 - IBM Developer for z/OS

File Edit Source Refactor Navigate Search Project Run Window Help

Git Repositories

DemoHealthCare [sandbox] - C:\GitHub\DemoHealthCare

- Branches
- Tags
- References
- Remotes
- Working Tree - C:\GitHub\DemoHealthCare
 - .git
 - DemoHealthCare
 - .settings
 - application-conf
 - bms
 - BuildOutput
 - cobol_cics
 - cobol_cics_db2
 - HCPDB01.cbl
 - HCPDB02.cbl
 - HCATDB01.cbl
 - HCAVDB01.cbl
 - HCAVDB01.json
 - HCIMDB01.cbl
 - hcipdb01.cbl
 - HCITDB01.cbl
 - HCIVDB01.cbl
 - HCMADB01.cbl
 - HCMADB02.cbl
 - copybook

HCMADB02.cbl

```
-----+*A-1-B-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+
158
159          MOVE CA-END-DATE (1:4) TO WS-END-NUM-YEAR
160          MOVE CA-END-DATE (6:2) TO WS-END-NUM-MONTH
161          MOVE CA-END-DATE (9:2) TO WS-END-NUM-DAY
162
163          MOVE WS-START-NUM-DATE TO WS-WORKING-DATE
164          * =%regi ===== Below must be comments to have the abend
165          * = when DEMO is day 1 must change to Compute = 20170901
166          * IF WS-WORKING-DATE < 16010101 or
167          *   WS-WORKING-DATE > 99991231
168          *   MOVE FUNCTION CURRENT-DATE (1:8) TO WS-WORKING-DATE
169          *   COMPUTE WS-WORKING-DATE = WS-WORKING-DATE - 1
170          *   COMPUTE WS-WORKING-DATE = 20170901
171          * END-IF
172          ++++++
173
174          COMPUTE WS-INTEGER-START-DATE =
175                  FUNCTION INTEGER-OF-DATE (WS-WORKING-DATE)
176
177
178          MOVE WS-END-NUM-DATE TO WS-WORKING-DATE
179          * =%regi ===== Below must be comments to have the abend
180          * = added to fix abend #2 FND_DATE
181          * IF WS-WORKING-DATE < 16010101 or
182          *   WS-WORKING-DATE > 99991231
183          *   MOVE FUNCTION CURRENT-DATE (1:8) TO WS-WORKING-DATE
184          * END-IF
185          ++++++
186
187          COMPUTE WS-INTEGER-END-DATE =
```

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

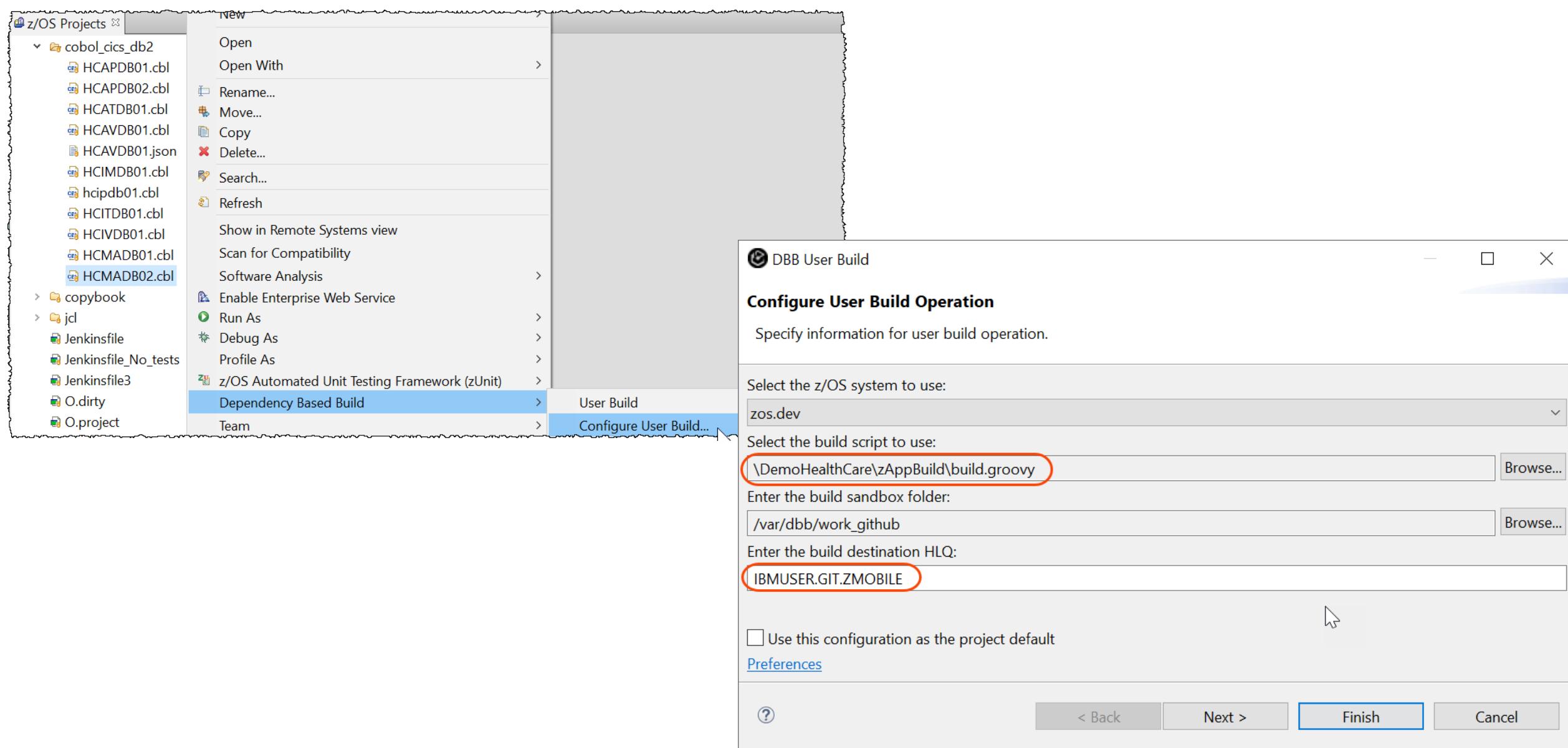
Bob – Mainframe Developer

Find the CICS issue, performs the COBOL updates, test the fix

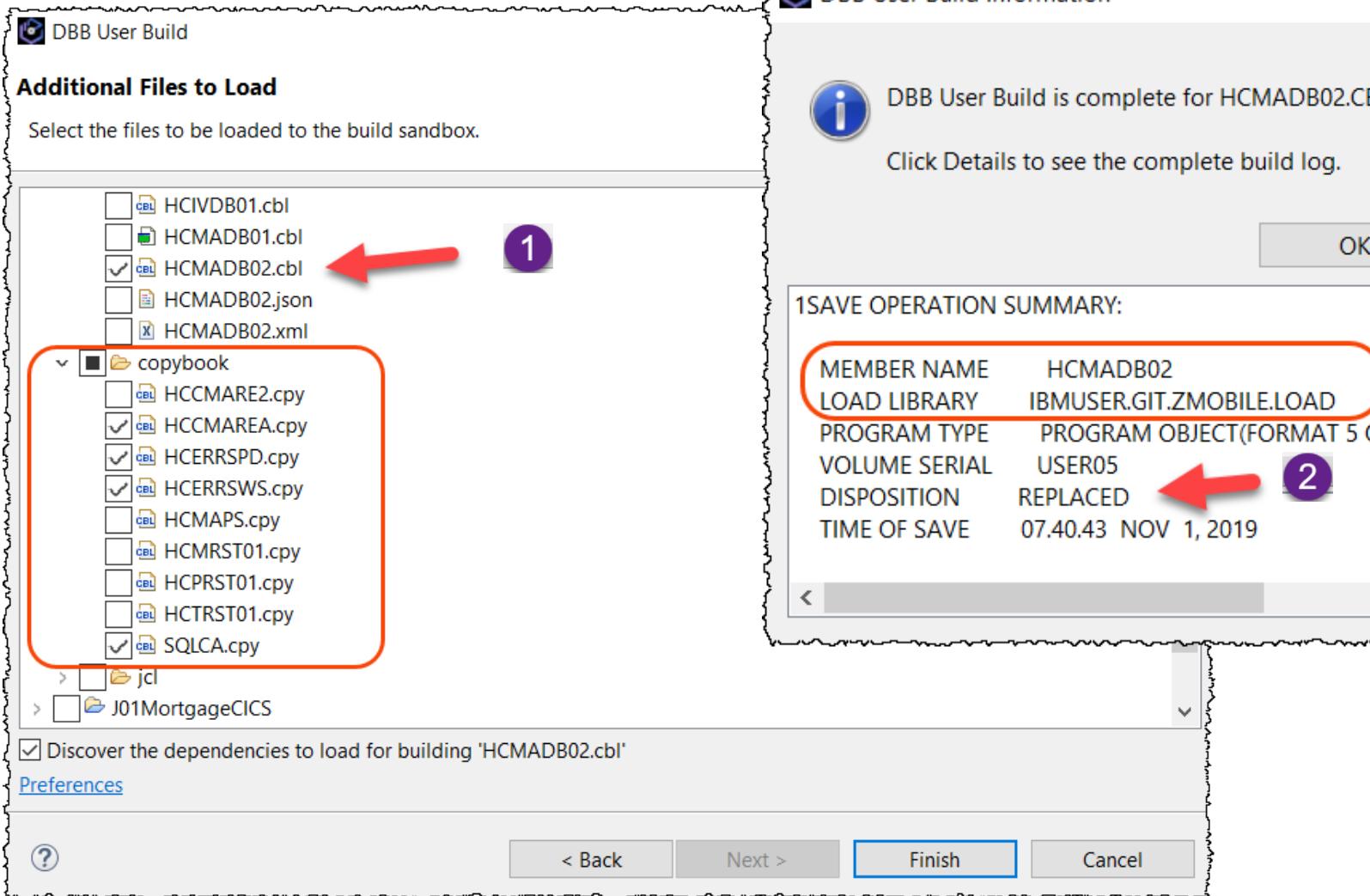
Objective: Fix the CICS Abend

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

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



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



The screenshot shows the "HCMADB02_bind.log" file. It contains several log entries. Two specific entries are highlighted with a red box and labeled "3":

```
DSNT255I -DBBG DSNTBCM2 BIND OPTIONS FOR  
PACKAGE = DALLASB.HCZMSA1.HCMADB02.()  
SQLERROR NOPACKAGE  
CURRENTDATA NO  
DEGREE 1  
DYNAMICRULES  
DEFER  
NOREOPT VARS  
KEEPDYNAMIC NO  
IMMEDWRITE INHERITFROMPLAN  
DBPROTOCOL DRDA  
OPTHINT  
ENCODING UNICODE(01208)  
PLANMGMT OFF  
PLANMGMTSCOPE STATIC  
CONCURRENTACCESSRESOLUTION  
EXTENDEDINDICATOR  
PATH  
DSNT275I -DBBG DSNTBCM2 BIND OPTIONS FOR  
PACKAGE = DALLASB.HCZMSA1.HCMADB02.()  
QUERYACCELERATION  
GETACCELARCHIVE  
DSNT232I -DBBG SUCCESSFUL BIND FOR  
PACKAGE = DALLASB.HCZMSA1.HCMADB02.()  
ISPF_RETURN_CODE = 0
```

A red arrow labeled "4" points to the "Error List" tab at the bottom of the screen.

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

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

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 THCMADBO
 - Test TEST2 (pass)
 - Test TEST3 (pass)

Add... Remove Up Down

Total results and settings
This section contains general information about the test run.

Runner result ID: c9f92ce3-966a-49b1-92c6-6edac879b7bb

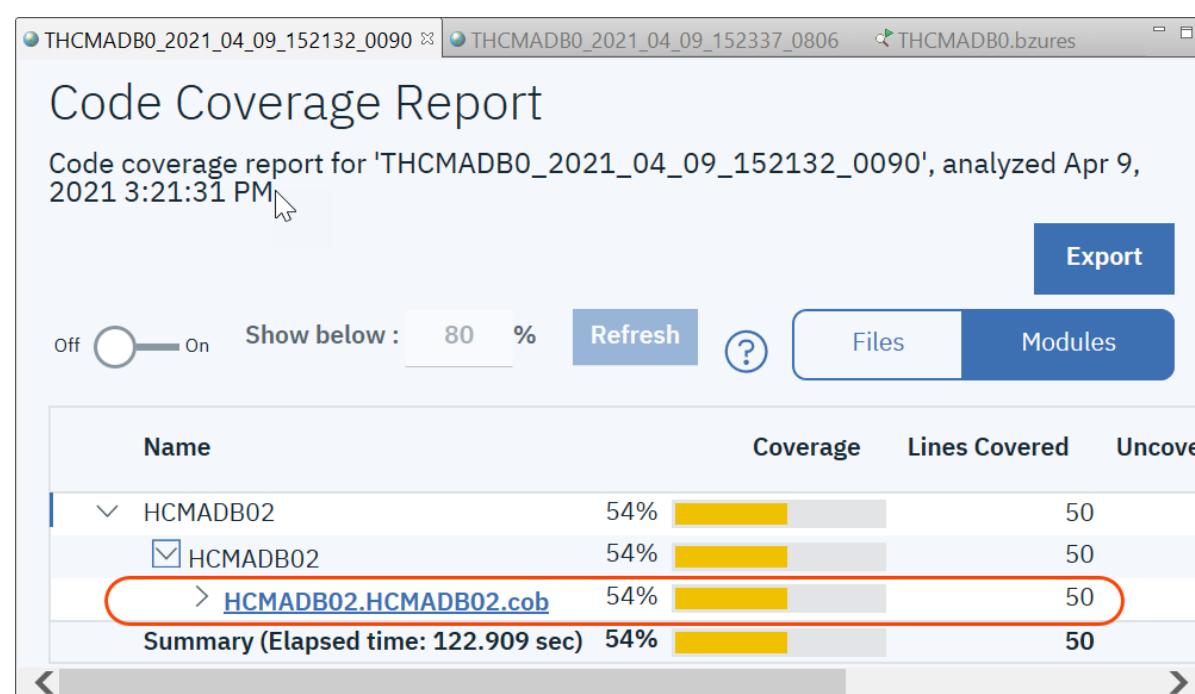
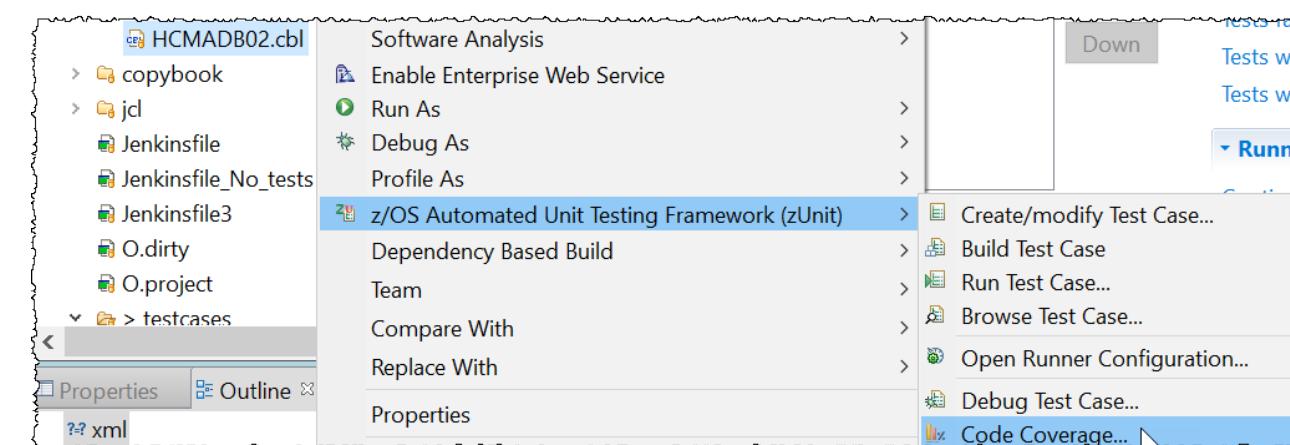
Test run total results

Test count:	2
Tests passed:	2
Tests failed:	0
Tests with errors:	0
Tests with severe errors:	0

Runner continuation settings
Continue if test fails:

[4/9/21, 3:16 PM] Job [JOB00098](#) is being submitted
[4/9/21, 3:17 PM] Job [IBMUSER1:JOB00098](#) ended with completion code CC 0000

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



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

The screenshot shows the DB2 command line processor with a COBOL program. Two specific lines are annotated with yellow speech bubbles:

- Line 338: "not executed" (with a red circle around it)
- Line 339: "were executed" (with a red circle around it)

```
MOVE 08000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 20000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
WHEN 3
MOVE 08000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 14000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 20000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
WHEN 4
MOVE 08000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 12000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 16000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 08000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 14000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 20000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
WHEN 4
MOVE 08000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 12000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 16000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
MOVE 20000000 TO WS-START-NUM-TIME
PERFORM INSERT-PRESCRIPTION
END-EVALUATE
```

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

CNX0211I Context: CICSTS53. Resource: PROGRAM. 29 (filtered,sorted) records collected at Mar 30, 2020, 10:08:33

Region	Name	Status	Use Count	Concurrent Us...	Language
CICSTS53	HCMADB02	ENARIED	3	0	COBOL
CICSTS53	HCMAPL01				COBOL
CICSTS53	HCMAPS				COBOL
CICSTS53	HCMRESTW				COBOL
CICSTS53	HCM1BI01				COBOL

Information - November 2019

HCZMSA1.MEDICATION

MEDICATIONID [INTEGER]	PATIENTID [INTEGER]	DRUGNAME [CHAR(50)]	STRENGTH [CHAR(20)]	AMOUNT [CHAR(10)]
1000190	3	RANIDITINE	20MG	0
1000191	3	RANIDITINE	30MG	0
1000192	1	DD	...	0
1000193	1	AA	...	0
1000211	1	EE	...	0
1000212	1	AA	...	0
1000226	1	AA	...	0
1000227	1	AA	...	0
1000248	1	BEGE	...	0
1000249	1	SS	...	0
1000250	1	SS	...	0
1000251	1	TILENOL	...	0

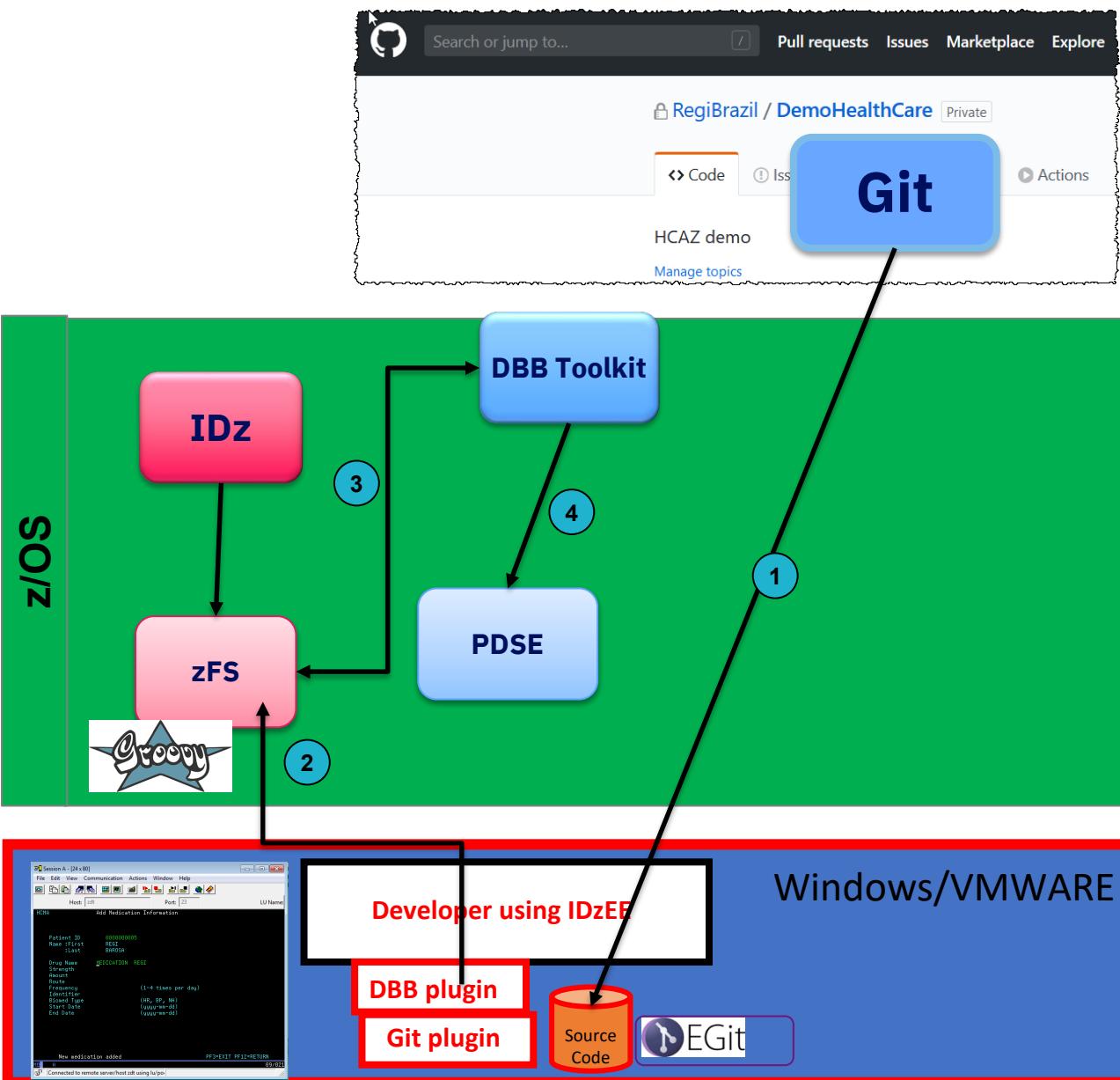
Patient ID 0000000001
Name :First RALPH
:Last DALMEIDA

Drug Name TILENOL
Strength
Amount
Route
Frequency (1-4 times per day)
Identifier
Biomed Type (HR, BP, NA)
Start Date (yyyy-mm-dd)
End Date (yyyy-mm-dd)

New medication added

PF3=EXIT PF12=RETURN

Using DBB User Build (Personal test)



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

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

[3] **DBB Toolkit** provides Java APIs to:

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

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

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

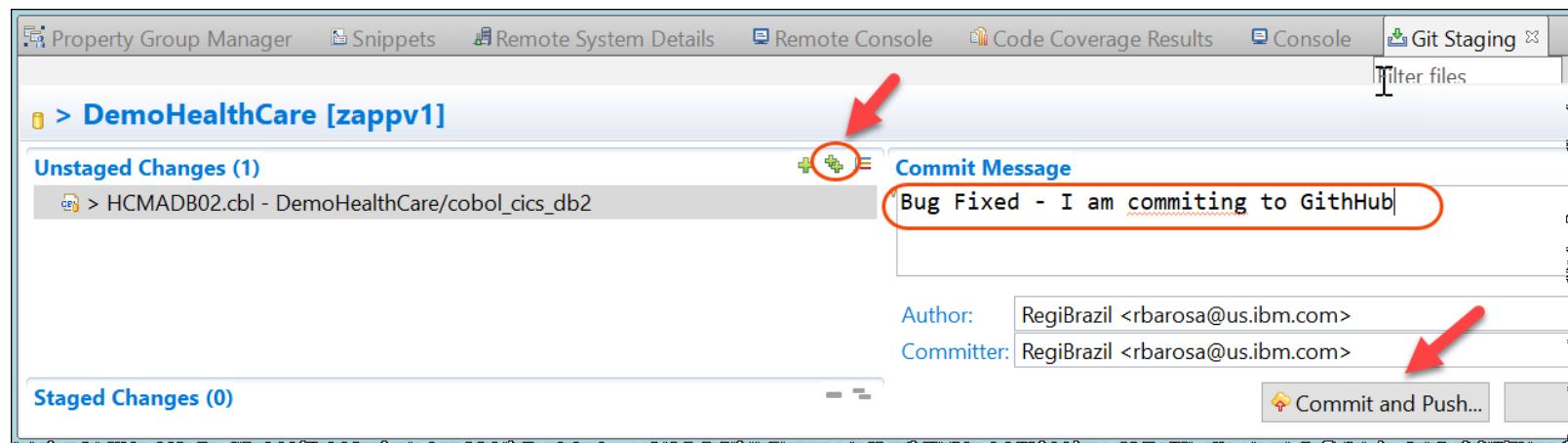
Bob – Mainframe Developer

Find the CICS issue, performs the COBOL updates, test the fix

Objective: Fix the CICS Abend

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

Uses GIT to Commit and Push the corrected code to Repository



Push Results: DemoHealthCare - origin

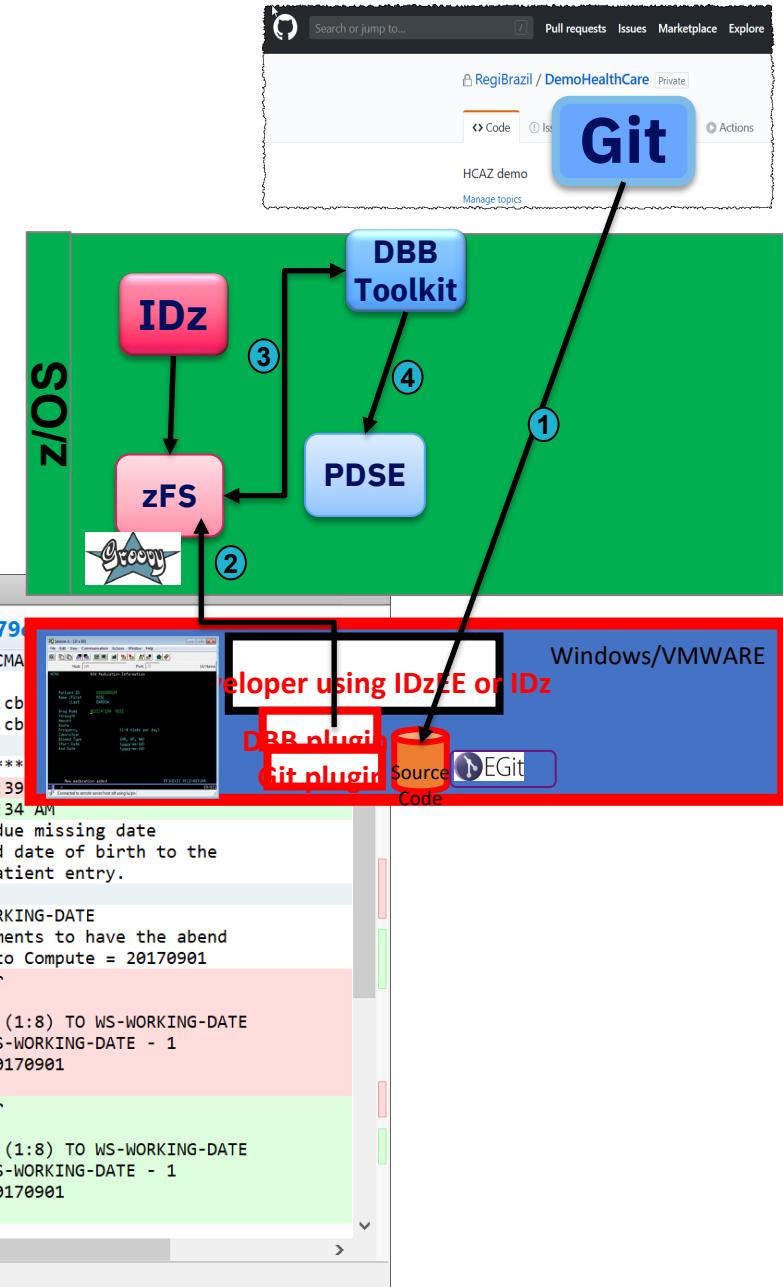
Pushed to DemoHealthCare - origin

zappv1 → zappv1 [6c0af85..ab45f3d] (1)

6c0af855: Bug Fixed - I am committing to GitHub (RegiBrazil on 2020-04-15 06:59:22)

Repository <https://github.com/RegiBrazil/DemoHealthCare.git>

Configure... Close



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

Rebecca – Release Engineer

Executes the Team Building using UCD and Jenkins

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

Example Pipeline Results

✓ GenAppNazarePipeline < 75

Branch: master 2m 12s Changes by baudy.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 baudy.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

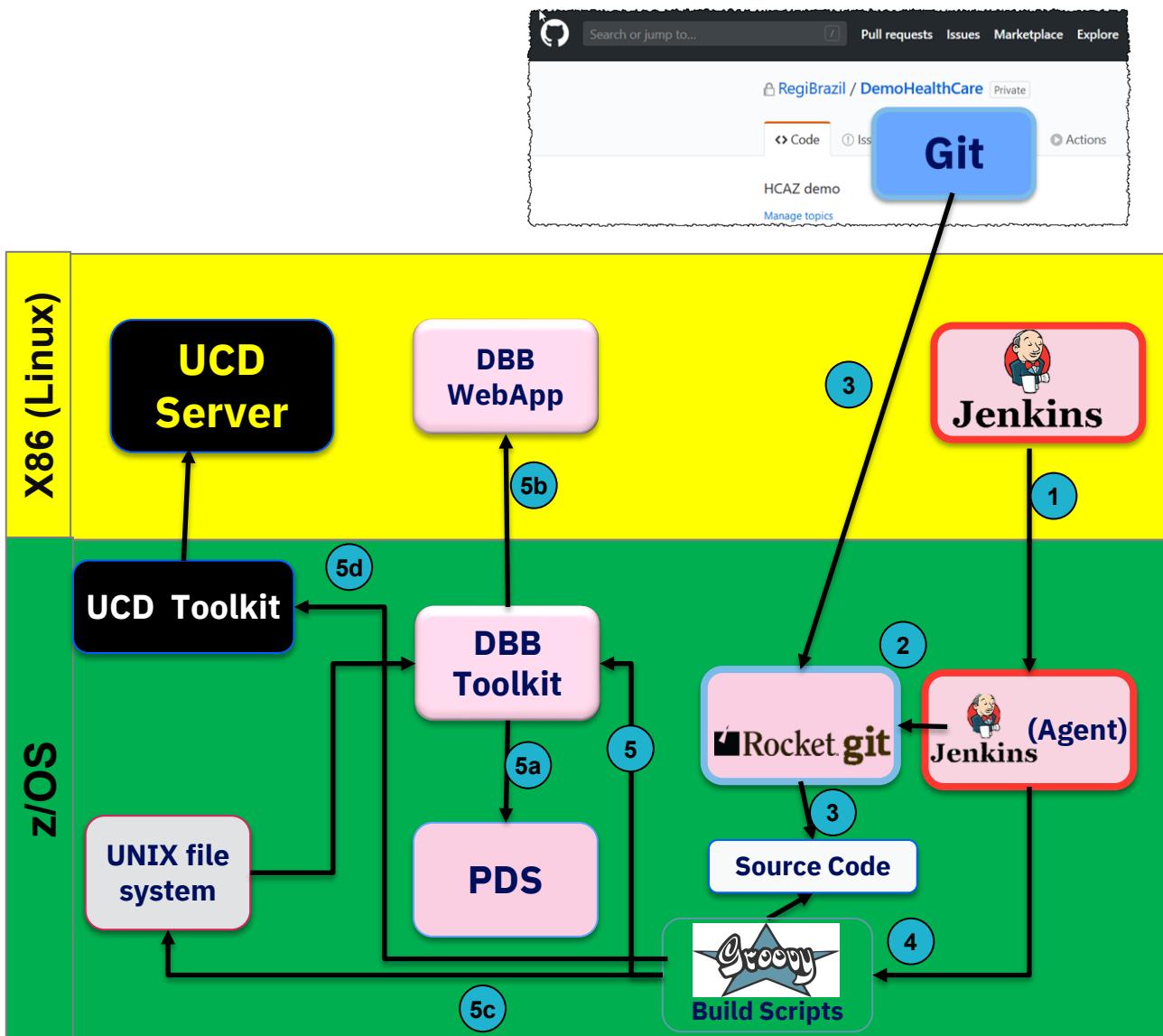
✓ > Shell Script 3s

✗ > 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**.

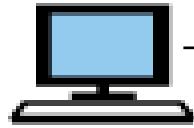
[5c] Copy logs from **PDS** to **zFS**, generate build reports (can be saved in build result).

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

URBancode DEPLOY FOR DEPLOYMENT AUTOMATION

DEPLOYING TO MAINFRAME

Develop



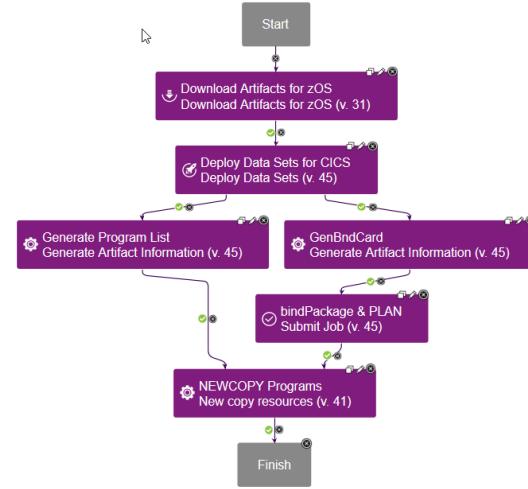
IDE
or
TSO/ISPF



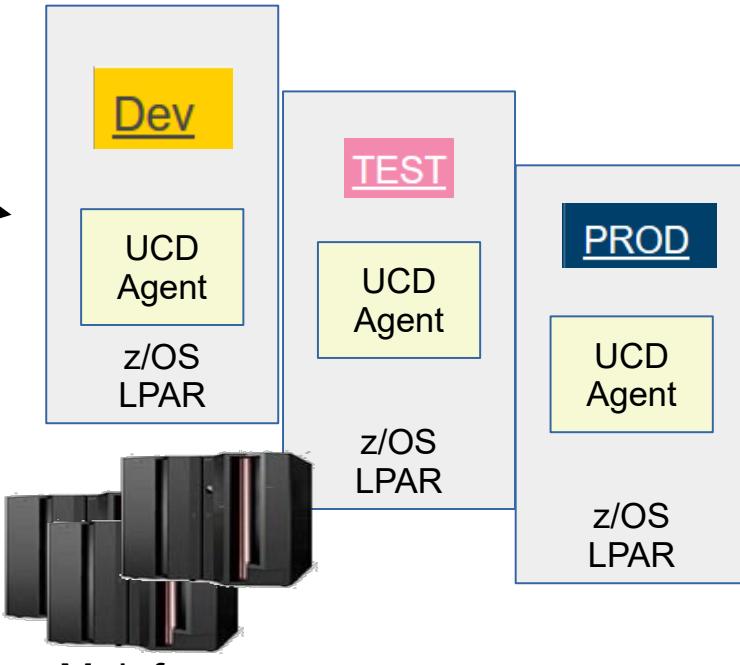
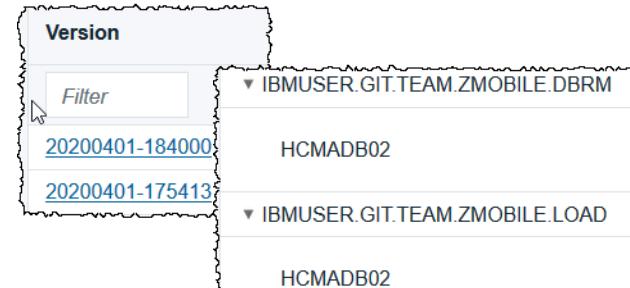
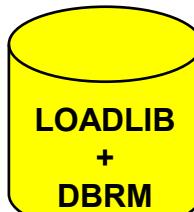
Build



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

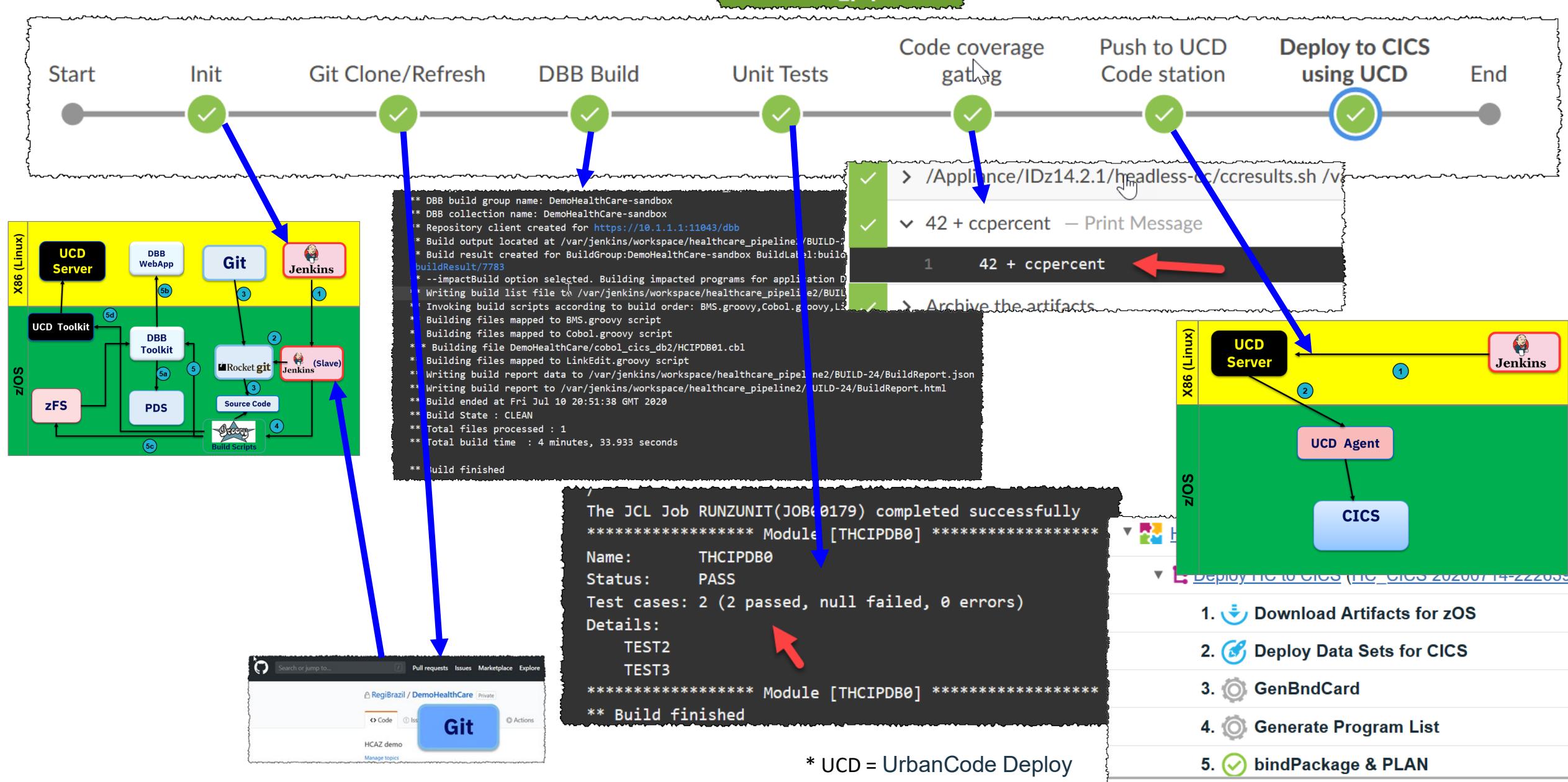


IBM UrbanCode Deploy

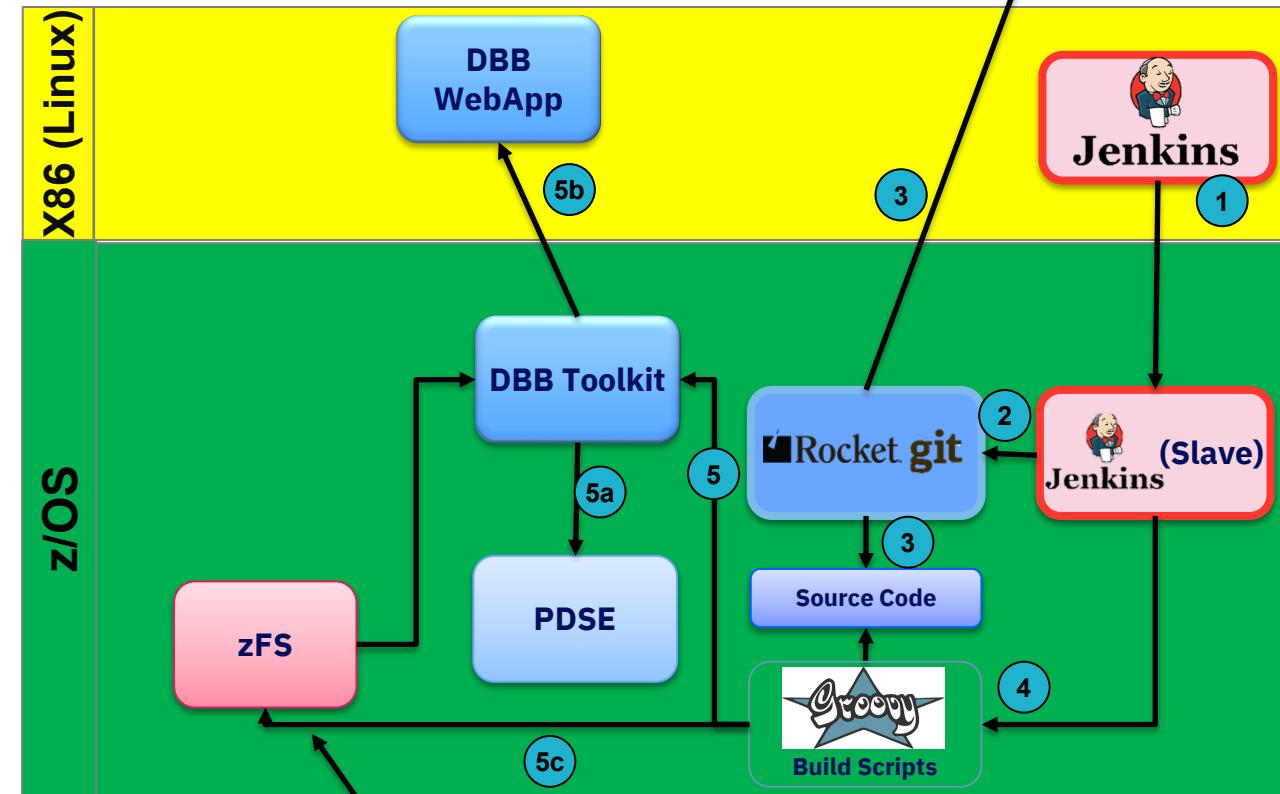


Demo High Level Diagram using Jenkins-Git-UCD

healthcare_pipeline2



Solution High Level Diagram using Jenkins-Git-DBB



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

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

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

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

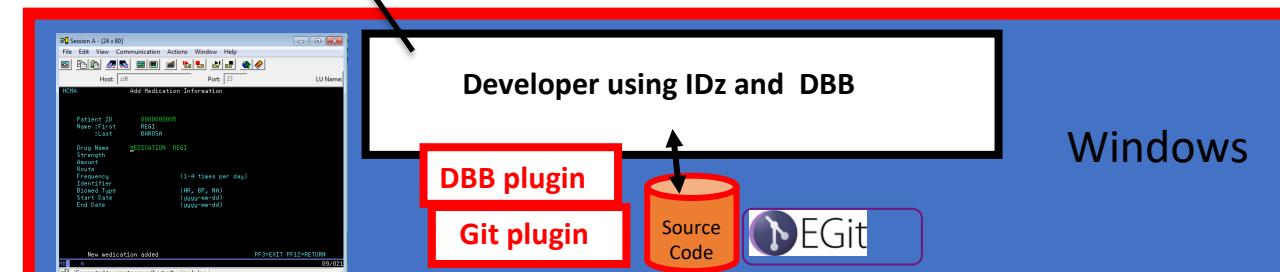
Example **scan changed DBB WebApp to find dependencies (5b)**

[5] **DBB Toolkit** provides Java APIs to:

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

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

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



Agenda - Day 1

- 2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)
3:00 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z

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

Agenda - Day 2

- 2:00 zUnit and VTP Overview (Wilbert)

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

3:00 – 4:50 Choose one Optional lab:
→ LAB 7 - IBM DBB with Git, Jenkins and UCD on Z
→ LAB 6B : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Remote Assets
→ LAB 6C : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Local Assets
→ LAB 6D : Using IBM zUnit to Unit Test a COBOL/DB2 batch program
→ LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
→ LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
→ LAB 8 - Using Application Performance Analyzer (APA)
→ LAB 9 - Using IBM Z VTP to test a COBOL /CICS / DB2 transaction
→ LAB 10 - Deploying COBOL/CICS/DB2 application using a GitLab CI Pipeline
→ LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
→ LAB 12 – Using ADDI to find CICS programs to be converted to API's
→ LAB 13 – Running Integration Tests using Galasa

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

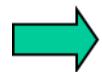
Agenda - Day 1

- 2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)
3:00 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z
5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

Agenda - Day 2

(Wilbert)

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



- 3:00 – 4:50 Choose one Optional lab:

- LAB 7 - IBM DBB with Git, Jenkins and UCD on Z
- LAB 6B : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Remote Assets
- LAB 6C : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Local Assets
- LAB 6D : Using IBM zUnit to Unit Test a COBOL/DB2 batch program
- LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
- LAB 8 - Using Application Performance Analyzer (APA)
- LAB 9 - Using IBM Z VTP to test a COBOL /CICS / DB2 transaction
- LAB 10 - Deploying COBOL/CICS/DB2 application using a GitLab CI Pipeline
- LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- LAB 12 – Using ADDI to find CICS programs to be converted to API's
- LAB 13 – Running Integration Tests using Galasa

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

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 **J05P** to become familiar with the Application that you intend to modify.

2. Load the source code from Git to the local IDz workspace

→ You will load the COBOL code that is stored on Linux to your windows client to be modified.

3. Modify the COBOL code using IDz.

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

4. Use IDz DBB User Build to compile/link and perform personal tests.

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

5. Push and Commit the changed code to Git .

→ You will commit the changes to Git.

6. Use Jenkins with Git plugin to build the modified code

→ You will use Jenkins pipeline to build the new changed code and push the executables to be deployed using UCD.

7. Use Jenkins and UCD plugin to deploy results and test the code again

→ You will verify the results after the final deploy to CICS using UCD

8. (Optional) Understanding DBB Build Reports

→ You will understand the reports generated by DBB during the build

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

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

Overview of development tasks

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

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

1. Get familiar with the application using the 3270 terminal

→ You will start a 3270 emulation and execute a transaction named **HCAZ** to become familiar with the Application that you will be recording.

2. Import a z/OS project

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

3. Record interaction with the application.

→ You will record an interaction with the COBOL CICS program.

4. Generate, build and run the unit test

→ You will compile and link-edit the generated unit test program, followed by running the unit test.

5. Introduce a bug in the program and rerun the unit test

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

6. Run the unit test from a batch JCL .

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

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

In this lab you will record interaction with a COBOL CICS/DB2 program (program under test) and import the recorded data into the unit test case generation process. You will build and run the unit test, make a change to the CICS COBOL program, and rerun the unit test.

Overview of development tasks

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

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

1. Get familiar with the application using the 3270 terminal

→ You will start a 3270 emulation and execute a transaction named **HCAZ** to become familiar with the Application that you will be recording.

2. Record interaction with the application.

→ You will record an interaction with the COBOL CICS/DB2 program.

3. Generate, build and run the unit test

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

4. Introduce a bug in the program and rerun the unit test

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

5. Run the unit test from a batch JCL .

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

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

In this lab you will record interaction with a COBOL/DB2 program via JCL execution and import the recorded data into the unit test case generation process. You will build and run the unit test, make a change to the COBOL/DB2 program, and rerun the unit test.

Overview of development tasks

Using IBM DBB to compile/link/bind

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

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

- 1. Run the COBOL/DB2 batch program using JCL**
→ You will submit a JCL to execute a COBOL/DB2 program that calls a COBOL subprogram to print a small report.
- 2. Use zUnit to record the batch execution.**
→ You will record the batch execution using the COBOL/DB2 program and subprograms.
- 3. Generate, build, and run the unit test generated program**
→ You will compile and link-edit the generated unit test program, followed by running the unit test.
- 4. Modify the COBOL/DB2 program (introduce a bug) and rerun the unit test**
→ You will modify the COBOL/DB2 program introduce a bug, rerun the unit test, and observe the failure of the test case

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

- 5. Run the batch program using the provided JCL and verify the bug .**
→ You will run the Batch JCL and observe the bug on the printed report..
- 6. Use IDz to fix the bug, recompile the COBOL/DB2 program**
→ The bug is fixed using IDz, program fixed is rebuilt
- 7. Rerun the zUnit and verify that the bug is eliminated**
→ When the zUnit test case is executed, you can verify that the program is fixed.

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

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

Overview of development tasks

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

Using IBM DBB to compile/link

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

1. Use VTP to record the CICS/DB2 application in action

→ You will record the CICS application in action. This application starts multiple CICS transactions using COBOL programs with and without DB2 access.

2. Run a JCL to execute the recorded replay sequence

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

3. Modify one COBOL/DB2 program (introduce a bug) and rerun the VTP JCL

→ You will modify the COBOL/DB2 program to introduce a bug, rerun the replay JCL, and observe the failure due the introduced bug.

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

4. Run the CICS transaction and verify the bug

→ You will run CICS application and observe the bug introduced.

5. Use IDz to fix the bug, recompile the COBOL/DB2 program

→ The bug is fixed using IDz, program fixed is rebuilt.

6. Rerun the VTP JCL and verify that the bug is eliminated

→ When running the JCL recorded replay you can verify that the program is fixed.

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

This lab will take you through the steps of using GitLab CI along with DBB, ZUnit and UrbanCode Deploy (UCD) on z/OS. On this lab you will fix a bug of an existing COBOL/CICS application stored in GitLab.

You will use IDz to change the code and perform a personal test for later delivery and commit to GitLab and then use GitLab CI for the final build and continuous delivery. The updated code will be deployed to CICS using UrbanCode Deploy (UCD)

Overview of development tasks

1. Review the GitLab issue and verify the bug using the 3270 terminal

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

2. Load the source code from GitLab to the local IDz workspace

→ using IDz you will clone the GitLab loading all COBOL source code to your windows client.

3. Use IDz to run the zUnit test case and verify the error

Running zUnit you will verify that the DOB is incorrect,

4. Modify the COBOL/CICS/DB2 program that has the bug using IDz.

→ Using IDz you will modify the COBOL program LGICDB01 to fix the bug.

5. Use IDz DBB User Build to compile/link and run the zUnit

→ You will compile and link the modified code using the *DBB User Build Function*.

When complete you will run zUnit generated test case and verify that the bug is fixed.

6. Push and Commit the changed code to GitLab .

→ You will commit the changes to *Git*. This will initiate the GitLab CI pipeline

7. Verify the GitLab CI Build execution.

→ You will use Gitlab to build the new changed code, run zUnit and push the executables to be deployed using *UCD* ..

8. Verify the GitLab CI Package and Deploy to UCD and test the CICS transaction again using 3270

→ You will verify the results after the final deploy to CICS using *UCD*

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

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

To facilitate the move to Git as an SCM for z/OS development, you can use a DBB migration tool that copies source code from partitioned data sets (PDS) to a local Git repository on the z/OS File System (zFS) to be committed and pushed to a distributed Git server.

More details at <https://www.ibm.com/docs/en/adfz/dbb/1.1.0?topic=migrating-data-sets-git>

Overview of development tasks

- 1. Logon to IDz and prepare the necessary directories and script migration file on zFS.**
→ You will start IDz and prepare some folders and the script migration file required for the migration on z/OS UNIX Files.
- 2. Migrating using DBB migration tool to a local Rocket Git client repository on zFS**
→ You will use the DBB migration tool, and the Rocket Git client installed on USS system and create a client git.
- 3. Push the migrated files to Git server (Gitlab)**
→ When all source files have been copied to the local Rocket Git repository, you are ready to push the files to the Git server as follows,
- 4. Load the source code from Git to the local IDz workspace and start working with the migrated assets**
→ Once the migration is complete on Git server you may clone using IDz and start working with the migrated assets.

Lab 13 – Running Integration Tests using Galasa

This lab will take you through the steps needed to execute some sample Integration tests using **Galasa** against a sample mainframe application using IDz as IDE.

In this lab we will use the **SimBank** example that is provided through the Galasa eclipse plugin. This example provides a **sample mainframe style application** as well as a set of tests that can be executed to test the application.

Overview of development tasks

PART #1 –

Explore the *SimBank* application and execute the Installation Verification Test (IVT)

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

PART #2 –

Execute a web service and 3270 integration test

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

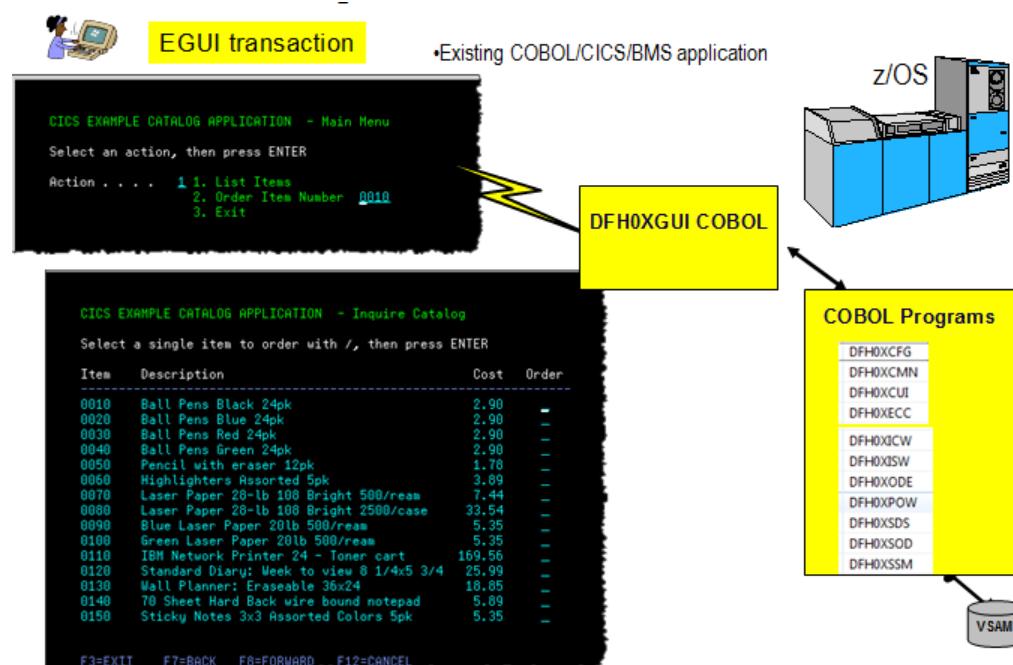
PART #3 –

Execute a Batch test

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

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

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



Lab 5: UrbanCode Deploy → Focus on Deploy

Abstract:

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

Part 1 – Create the UrbanCode application infrastructure.

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

Part 2 - Create the UrbanCode deployment processes.

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

Part 3 – Deploy the application to z/OS CICS

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

Lab 8: Using Application Performance Analyzer (APA)

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

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

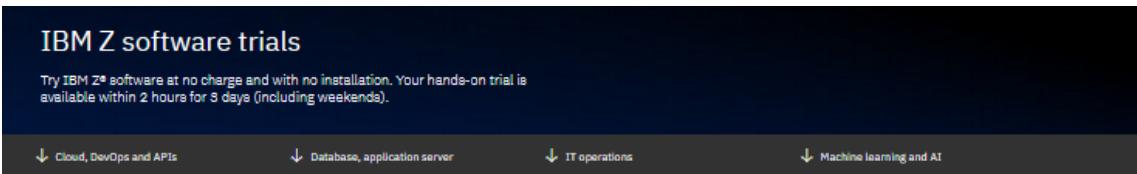
Overview of development tasks

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

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

If interested in doing the ADDI Lab (LAB 12)

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



Cloud, DevOps and API integration software trials

Updated! IBM z/OS® Connect Enterprise Edition

Create efficient and scalable RESTful APIs for mobile and cloud applications. Learn how to:

- * Create REST APIs to:
CICS® programs
IMS™ transactions
Db2® for z/OS data
IBM® MQ Queues.
- * Call a REST API from a COBOL application.

[→ Register for the trial](#)
[→ Explore the product first](#)

New! IBM Z Development and Test Environment

Use a web-based tool to learn how to self-provision a z/OS® environment on emulated Z hardware for agile development and test activities. Learn how to:

- * Explore how to use a z/OS system.
- * Self-provision a z/OS from the IBM pre-built software stack ADCD.
- * Extract CICS® and Db2® application artifacts from your source z/OS environment and deploy them to your z/OS target environment.

[→ Register for the trial](#)
[→ Explore the product first](#)

New! IBM Application Performance Analyzer for z/OS®

Quickly identify and act upon areas of low performance, high CPU consumption, low response time, and low throughput in your most critical z/OS applications. Learn how to:

- * Create, initiate, and analyze performance observations for a Java and COBOL batch program.
- * Exploit source-program mapping and then modify source code statements in a Java and COBOL program.
- * Create and review a Variance Report to compare the performance of original and modified program.

[→ Register for the trial](#)
[→ Explore the product first](#)

New! Bring Your Own (BYO) z/OS for Cloud Native Development

With this trial of IBM Wazi Developer and IBM Developer for z/OS® Enterprise Edition you can choose your preferred IDE (Eclipse® or Microsoft® VS Code™) integrated with familiar DevOps tools such as Git and Jenkins to develop a z/OS application. Learn how to:

- * Use GitHub to see your assigned tasks and access application source code stored in Git.
- * Code, debug and build the sample COBOL application using Eclipse or VS Code.
- * Commit and push modifications to Git and request a Jenkins pipeline.

[→ Register for the trial](#)
[→ Explore the product first](#)

Updated! IBM Application Discovery and Delivery Intelligence

Manage your software lifecycle and digital and cloud transformation. Learn how to:

- * Identify maintainability issues and update unreachable code.
- * Find CICS® programs able to be converted to APIs.
- * View business terms and rules inventory in ADDI.
- * Discover and manage terms and rules.

[→ Register for the trial](#)
[→ Explore the product first](#)

Updated! IBM Cloud Provisioning and Management for z/OS

Provision z/OS software subsystems with automated processes. Learn how to:

- * Provision and deprovision a CICS® region with a cloud provisioning marketplace.
- * Create and publish a CICS region service to facilitate self-service provisioning.
- * Provision, deprovision an MQ queue with a cloud provisioning marketplace.

[→ Register for the trial](#)
[→ Explore the product first](#)

IBM SDK for Node.js - z/OS

Modernize apps, orchestrate services with Node.js and connect to your z/OS assets. Learn how to:

Learn how to:

IBM Application Delivery Foundation for z/OS

Integrated tools that help teams design, build, test, and debug z/OS software. Learn how to:

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

Follow workbook instructions..

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

Manage your software lifecycle and digital and cloud transformation.

Learn how to:

IBM Application Discovery and Delivery Intelligence

Welcome to your IBM Z Trial environment. Get started by exploring the scenarios below.

SCENARIO 30 MINS Assess and improve the maintainability of a COBOL application Explore scenario	SCENARIO 30 MINS Discover candidate APIs in your code base Explore scenario	SCENARIO 30 MINS View business terms and business rule packages inventory Explore scenario
SCENARIO 30 MINS Discover business terms and business logics Explore scenario		

Exposing and leveraging APIs has become a critical component of the modern IT business model.

The first step in this process is discovering what potential API candidates exist in your codebase. This scenario guides you through the discovery process in roughly 30 minutes. By the end of the session, you will know how to perform the following tasks:

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

IBM Z Trial Program



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



No charge environment



No set up, no install



Hands-on tutorials

Available now:

- IBM z/OS Connect Enterprise Edition
- IBM Db2 Utilities Solution for z/OS
- IBM Cloud Provisioning and Management for z/OS
- IBM Information Management System (IMS)
- IBM Machine Learning for z/OS
- IBM Dependency Based Build
- IBM Application Discovery and Delivery Intelligence (ADDI)
- IBM Application Delivery Foundation for z Systems
- IBM Z Development and Test Environment

- IBM Db2 Administration Solution for z/OS
- IBM Operational Decision Manager for z/OS
- IBM Open Data Analytics for z/OS
- IBM OMEGAMON for JVM on z/OS
- IBM CICS Transaction Server
- IBM SDK for Node.js – z/OS

Coming soon:

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

Register now at ibm.biz/z-trial

© 2018 IBM Corporation

No-Charge IDz/ADFz Training “Free education”

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

Overview

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

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

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

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



No-Charge IDz/ADFz Training offerings

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

Home > IDz/RDz Remote Training

IDz/RDz Remote Training

Entry-Level Training - Module Content Details

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



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

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

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

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

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

Consider adding a link to this page to your Workspace. As more blogs, PDFs and video recordings are updated to include them.

IDz and Eclipse - Source Editing, Options and Techniques

- Overview - Terms and Concepts
 - Overview of the ADFz and IDz Solution - Feature/Function PDF
 - IDz Workbench Overview - A short product intro-video
 - Introduction to IDz - Click for Click Workshops - "Labs.PDF"
 - Workspace Design - Part of the IDz Product Rollout Framework
- + Application Development and IDz Navigation Tools and Techniques
- + [ISPF Techniques for Editing and Browsing Source Files](#)
- + Courses for PL/I, Assembler, C, Java and Preprocessing 4GL Code for use with IDz

IBM Developer for z/OS (IDz)

Self paced Learning

https://ibm.github.io/mainframe-downloads/DevOps_Acceleration_Program/idz-self-paced-learning.html



Questions / Comments



Agenda - Day 1

- 2:00 Introduction to DevOps on IBM Z - focus on open-source tools (Regi/Ron)
3:00 LAB 1 - Working with z/OS using COBOL and DB2
or
LAB7 - IBM DBB with Git, Jenkins and UCD on Z
5:00 Final Checkpoint - Comments and Closing Day 1 (David/Ron/Wilbert/Regi)

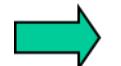
Agenda - Day 2

Review (Wilbert)

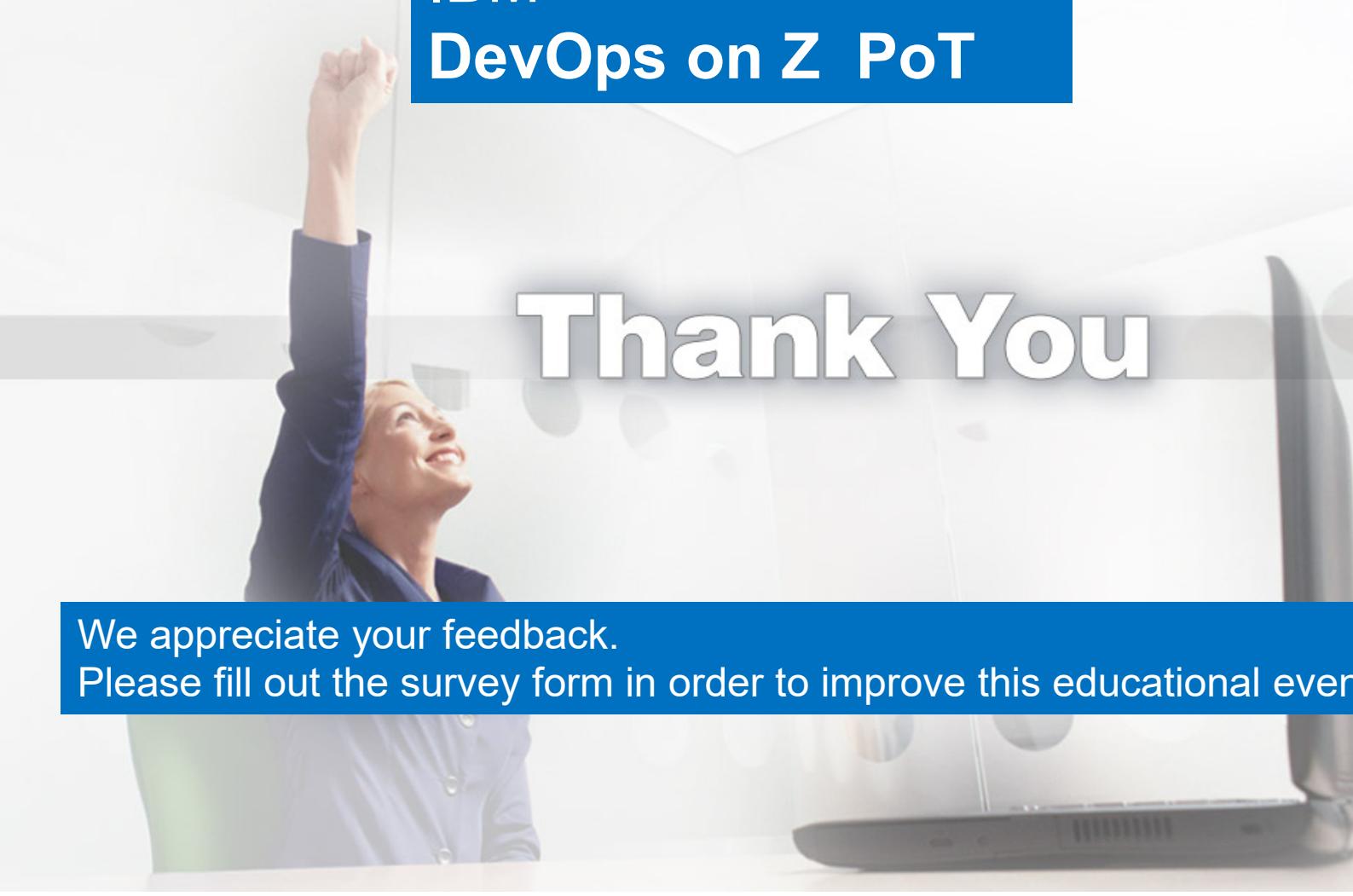
2:15 Demo: Scenario using Z DevOps solutions including **ADDI**, **zUnit**, **Git** and **Jenkins** (David/Regi)

3:00 – 4:50 Choose one Optional lab:

- LAB 7 - IBM DBB with Git, Jenkins and UCD on Z
- LAB 6B : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Remote Assets
- LAB 6C : Using IBM zUnit to Unit Test a COBOL/CICS/DB2 program using Local Assets
- LAB 6D : Using IBM zUnit to Unit Test a COBOL/DB2 batch program
- LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App
- LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS
- LAB 8 - Using Application Performance Analyzer (APA)
- LAB 9 - Using IBM Z VTP to test a COBOL /CICS / DB2 transaction
- LAB 10 - Deploying COBOL/CICS/DB2 application using a GitLab CI Pipeline
- LAB11 – Migrating partitioned data sets (PDS) to a distributed Git repository
- LAB 12 – Using ADDI to find CICS programs to be converted to API's
- LAB 13 – Running Integration Tests using Galasa



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



IBM
DevOps on Z PoT

Thank You

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

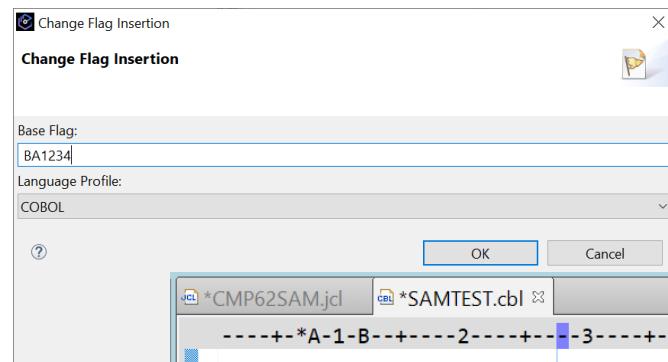
Enterprise DevOps – The Benefits

- 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

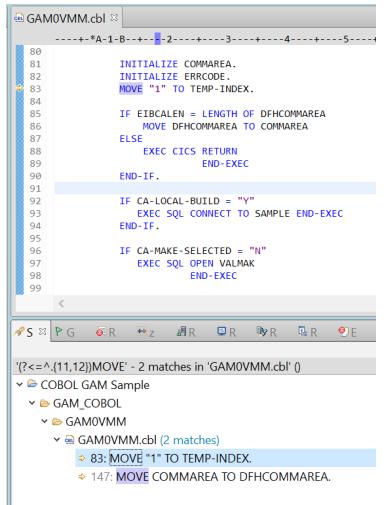
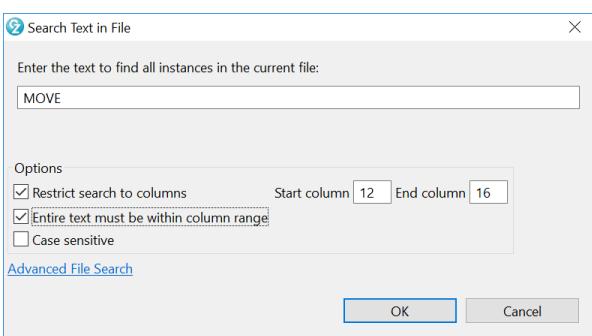


```
JCL *CMP62SAM.jcl CBL *SAMTEST.cbl
-----+*A-1-B-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+
BA1234* IF A-TEST NOT = ZERO
BA1234* AND (B-TEST (C-TEST)) IS NUMERIC
BA1234* ADD 1 TO A-TEST.

ACCEPT CURRENT-DATE FROM DATE.
ACCEPT CURRENT-TIME FROM TIME.

BA1234 DISPLAY 'SAMTEST STARTED DATE = ' CURRENT-MONTH '/'
               CURRENT-DAY '/' CURRENT-YEAR '(mm/dd/yy)'.
DISPLAY '          TIME = ' CURRENT-HOUR ':'
               CURRENT-MINUTE ':' CURRENT-SECOND.

PERFORM 700-OPEN-FILES .
PERFORM 800-INIT-REPORT .
```



```
SAMTEST.cbl
-----+*A-1-B-----+-----2-----+-----3-----+-----4-----+
05 CURRENT-TIME.
10 CURRENT-HOUR
10 CURRENT-MINUTE
10 CURRENT-SECOND
10 CURRENT-HNDSEC

* WS-FIELDS.
05 WS-CUSTFILE-STATUS PIC .
05 WS-CUSTOUT-STATUS PIC .
05 WS-TRANFILE-STATUS PIC .
05 WS-REPORT-STATUS PIC .
05 WS-TRAN-EOF PIC .
05 WS-TRAN-OK PIC .
05 WS-CUST-FILE-OK PIC .
05 WS-CUST-FILE-EOF PIC .
05 WS-TRAN-MSG PIC .
05 WS-PREV-TRAN-KEY PIC .
05 INCR-CUST-ID PIC .
05 START-CUST-ID PIC .
05 MAX-CUST-ID PIC .

DCL 1 PRODUCT_RECORD BASED(ADDR(CI))
2 PRODUCT_KEY,
3 PR_CUST_ID CHAR(5),
3 PR_RECORD_TYPE CHAR(1),
3 PRODUCT_ID CHAR(7),
2 PRODUCT_NAME CHAR(25),
2 DATE_PURCHASED CHAR(10),
2 SERVICE_CALLS FIXED BI,
2 LAST_SERVICE_CALL CHAR(10),
2 RECORD_FILL CHAR(20)

***** PROGRAM FLAGS *****
***** ***** *****
DCI TRANFILE FOF RTT(1) TNTT ('0'R')
```

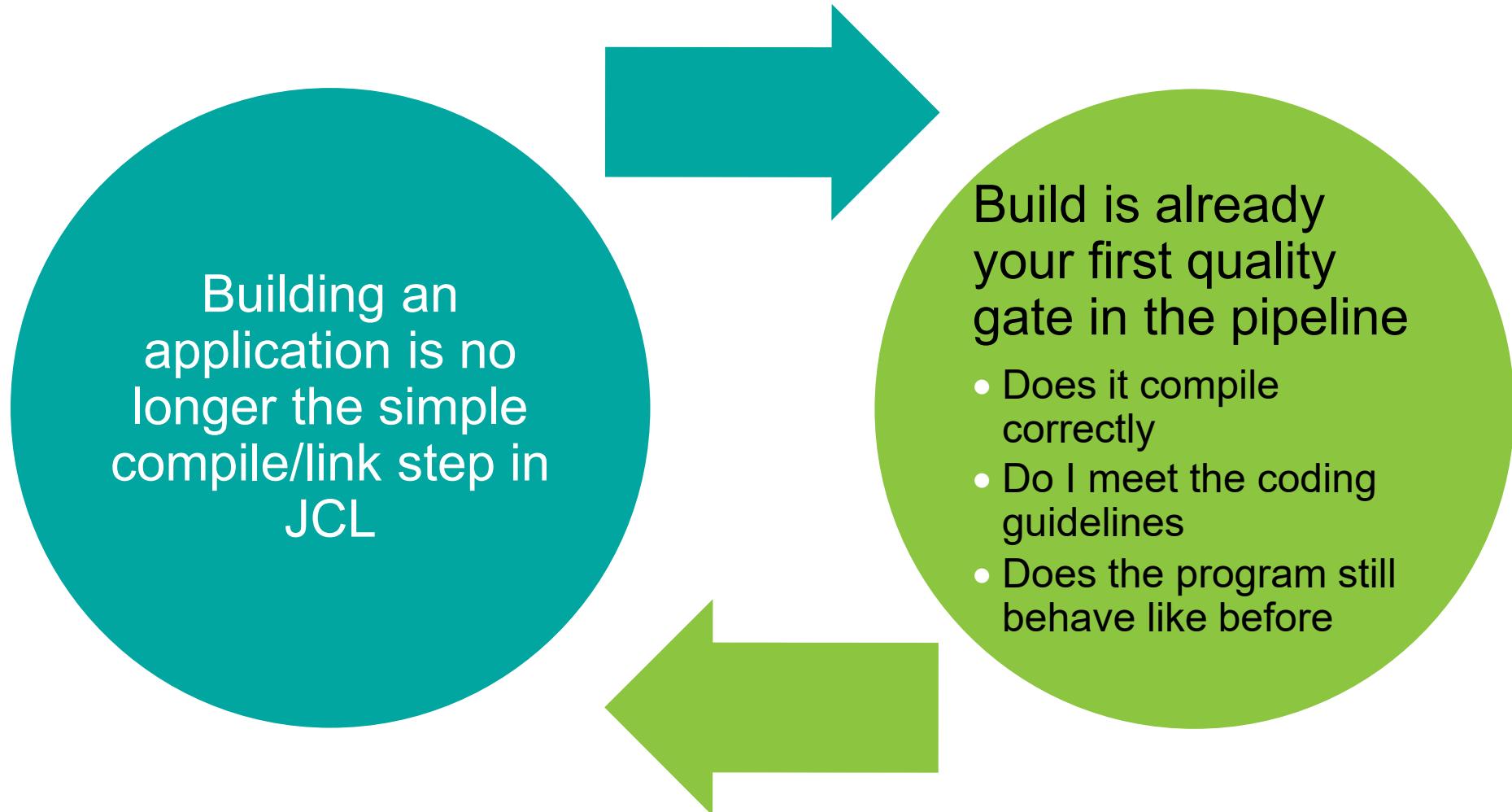
New! COBOL and PL/I editors offer a shortcut to toggle a vertical bar on/off in the editor (14.2)

```
GAMOPCS.pli
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+
000001 *process system(cics) source not('**') dlinit options;
000002 *process langlvl(saa2) limits(extname(31));
000003
000004 GAMOPCS: Procedure options (main);
000005
000006 DCL ARRAYINDEX500           SIGNED FIXED BIN(31),
000007           ARRAYINDEX10           SIGNED FIXED BIN(31),
000008           OPT_COUNT_RESULTS     CHARACTER(1),
000009           OPT_RESULT_COUNT      SIGNED FIXED BIN(31),
000010           SFM_CONTAINER_NAME     CHARACTER(16),
000011           STOKEN                SIGNED FIXED BIN(31),
000012           SRESP                 SIGNED FIXED BIN(31),
000013           LENGTH                SIGNED FIXED BIN(31),
000014           GAM_CHANNEL          CHARACTER(16) INITIAL ('GAMCHANNE
000015           DISPLAYABLERESULTCOUNT PIC '999',
000016           EMPTYVIN              CHARACTER(4);
000017
000018 DCL 01 CHANNEL_OUTPUT,
```

```
GAMOPCS.pli
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+
000179   CHANNEL( GAM_CHANNEL )
000180
000181
000182   EXEC CICS PUT CONTAINER( "QUERY-MESSAGE" )
000183   CHANNEL( GAM_CHANNEL )
000184   FROM( QUERYMESSAGE );
000185
000186   IF OPT_COUNT_RESULTS = "Y" THEN DO;
000187   EXEC CICS PUT CONTAINER( "RESULT-COUNT" )
000188   CHANNEL( GAM_CHANNEL )
000189   FROM( OPT_RESULT_COUNT );
000190
000191
000192
000193
000194
000195
000196
```

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

Building vs. Compile/Link



Topology used on LAB 12

zD&T on Cloud

