

# Z Open Development PoT

## Proof of Technology

DevOps Solution for Z Development

Mark Boyd  
[mboyd@us.ibm.com](mailto:mboyd@us.ibm.com)

Regi Barosa  
[rbarosa@us.ibm.com](mailto:rbarosa@us.ibm.com)

Timothy Han  
[timothy.han@ibm.com](mailto:timothy.han@ibm.com)

Thuy-Mi Le  
[Thuy-mi.le@ibm.com](mailto:Thuy-mi.le@ibm.com)

# Audience

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

# Pre-requisites

- Familiarity with z/OS (MVS) concepts
- Awareness (not necessarily proficiency) of basic z/OS System Programming tasks
- Basic familiarity with JCL concepts (JCL skills NOT required)
- No Java skills are required
- COBOL, PL/I and 4GL programmers are welcome

# Proof of Technology Objectives

This Proof of Technology (PoT) will demonstrate some of the IBM DevOps Solution for z Systems Development:

- ❑ Introduces DevOps Tools for z/OS:  
Application Discovery (AD)  
Application Delivery Foundation for z Systems (ADFz – IDz + PDTools)  
UrbanCode Deploy (UCD)  
z Systems Development and Test Environment z (zD&T – formerly RD&T)
- ❑ Provides basic skills and introductory hands-on exposure to AD, ADFz and UCD
- ❑ No prerequisite programming skills are required
- ❑ The lab exercises are structured to provide a guided walk-through of tools capabilities

# Agenda



9:00 Introductions

9:20 DevOps on z Systems Introduction

9:45 Application Delivery Foundation (ADFz):

- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break

10:30 LAB 1 - Working with z/OS using COBOL and DB2

or

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

11:45 Lunch

12:30 z Open Unit Test (Zunit) : Overview & Demonstration

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

2:00 – 4:30 Choose one Optional lab:

→ LAB 0 - Unit Testing for COBOL CICS (1 hour)

→ LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)

→ LAB 2C Application Discovery : Find a candidate API from Catalog Manager Application

→ LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App

→ LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS

→ LAB 8 - Using Application Performance Analyzer (APA)

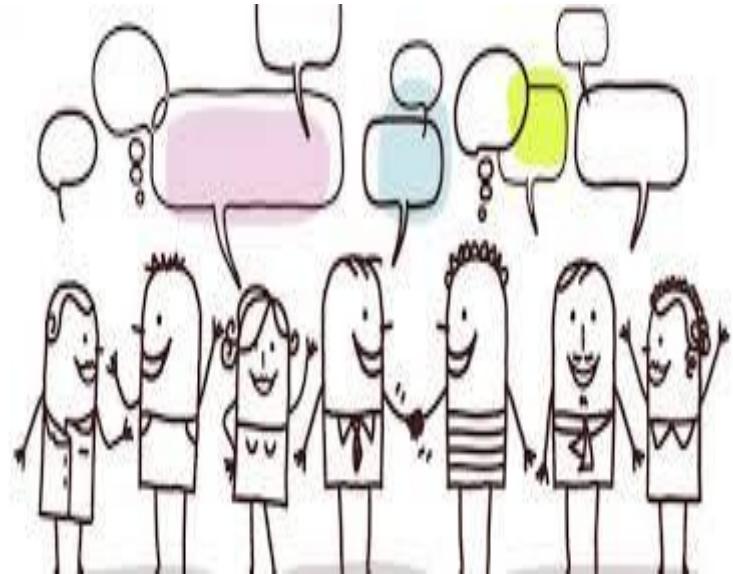
■ Lecture

■ Labs

■ Breaks

# Introductions

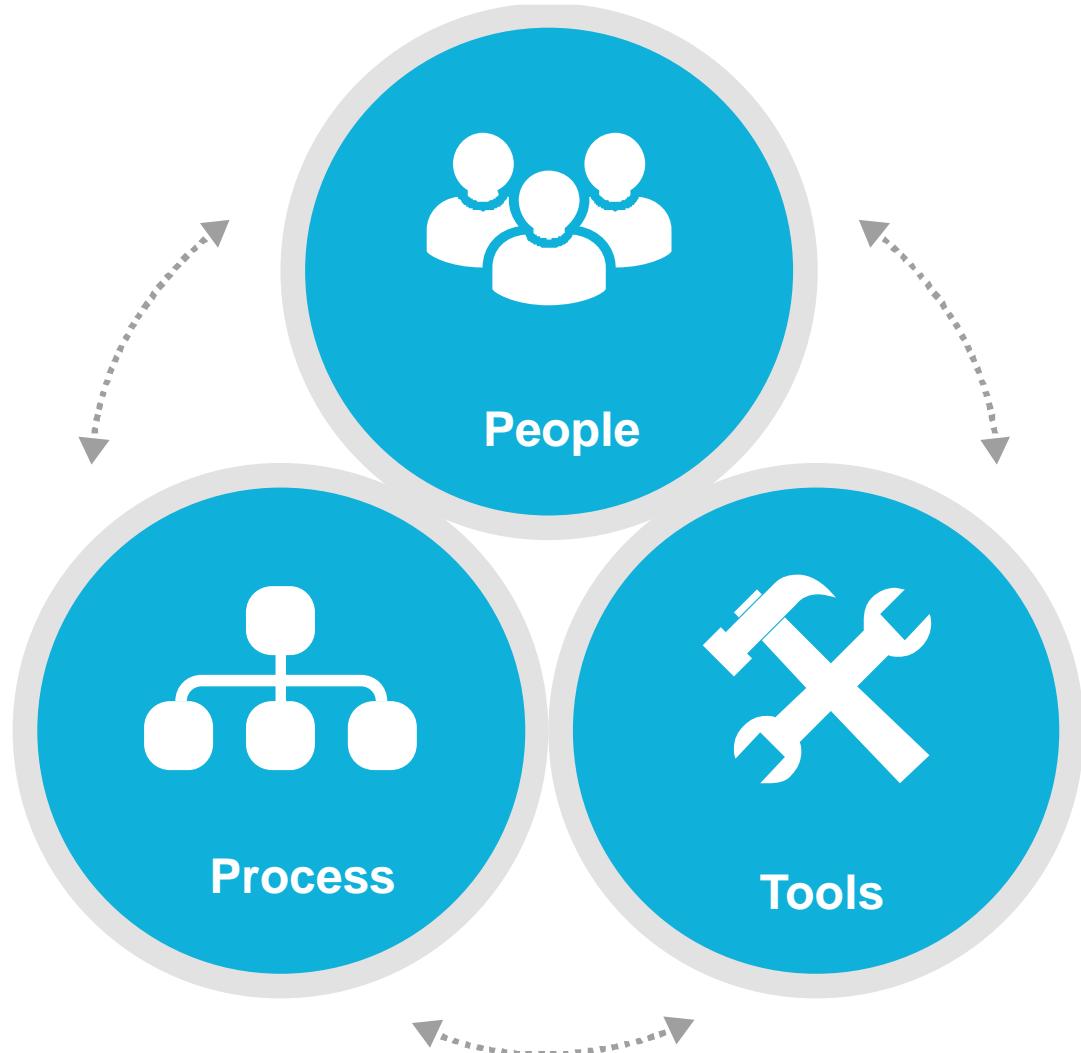
- Introduce yourself
- Name and organization
- Current integration technologies/tools in use
- Experience with Eclipse base products?



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



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



# Applications and teams move at variable speeds...



Rapid iterations



Systems of Engagement



Alignment

- Continuous synchronization and planning
  - Continuous testing

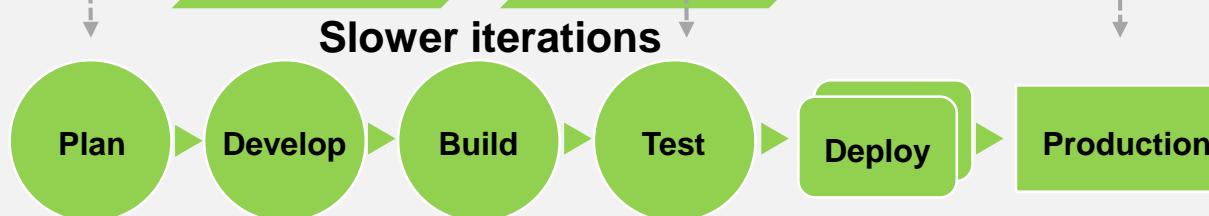
- Continuous deployment and monitoring



Systems of Record (Mainframe)

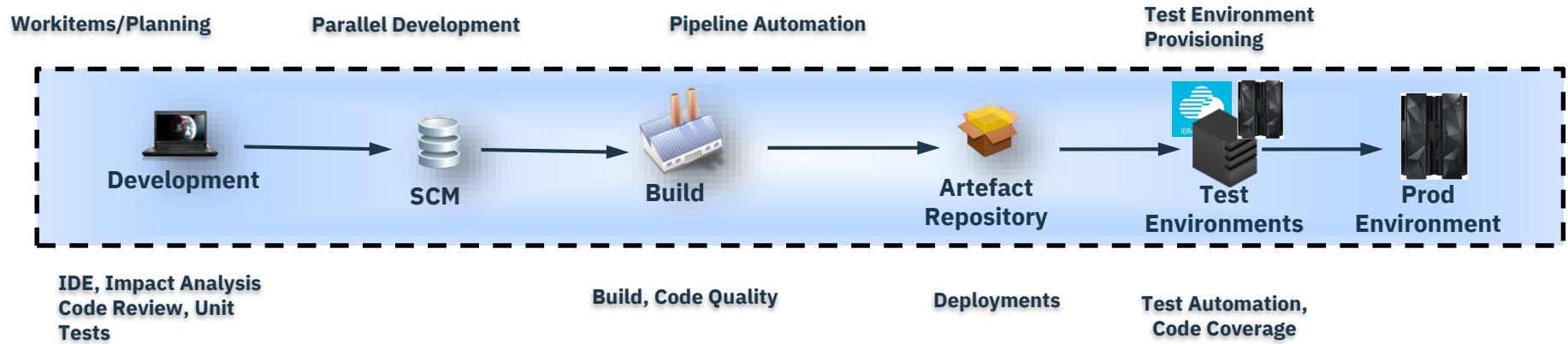


Slower iterations

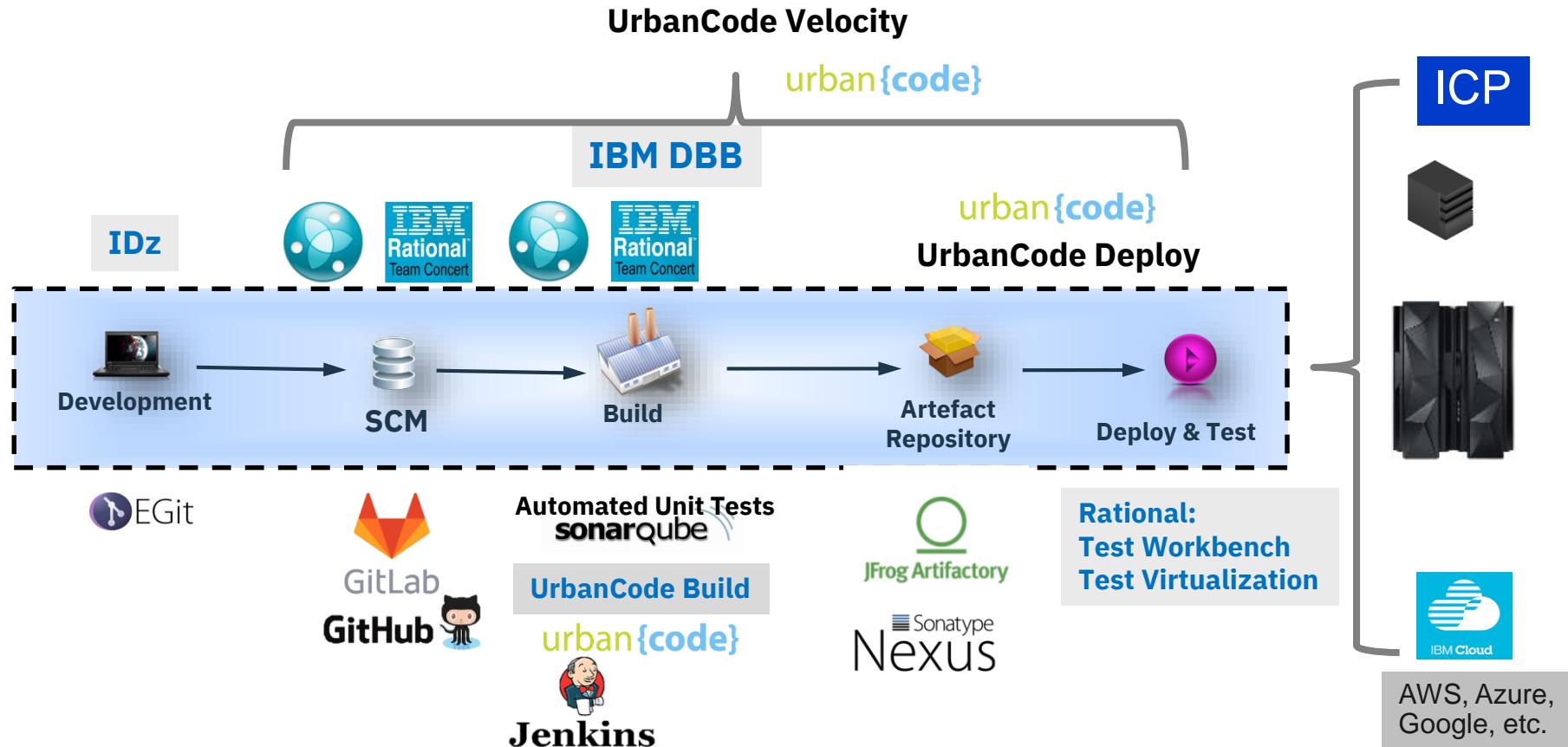


# End to End Strategy

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



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



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

.....

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

- Pipeline Coordination
- Release Mgmt
- Deployment Mgmt
- Artifact Repo
- Requirements/Tasks
- Quality Control
- Testing
- Source Control
- Build
- Development

Mainframe

Distributed

Jenkins

UrbanCode Release

Jenkins or UrbanCode or Ansible

Artifactory, Nexus or UrbanCode

Issues or Rational Team Concert or Jira

SonarQube

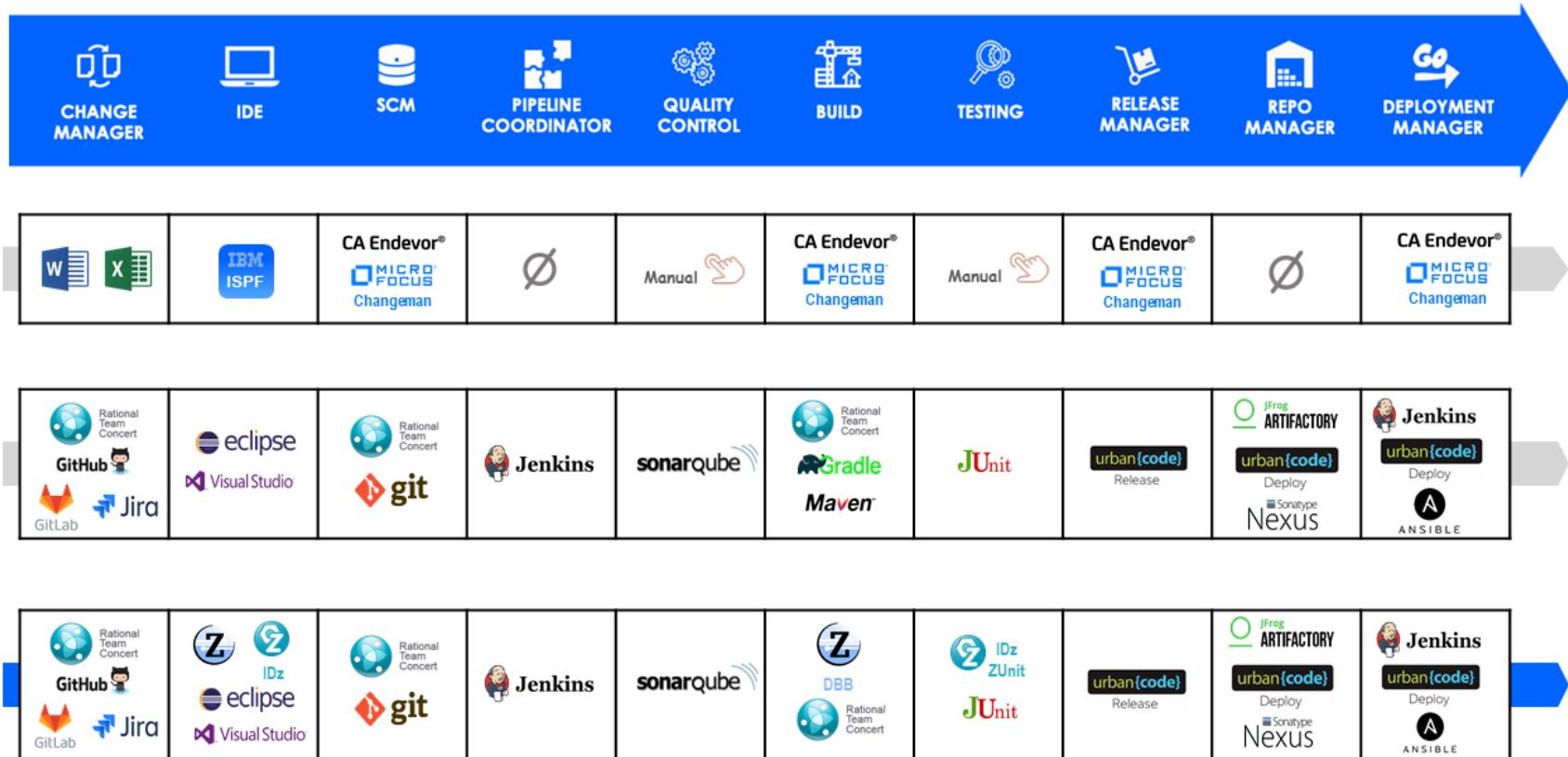
Multiple, JUnit, ZUnit

Rational Team Concert or Git

Maven, Gradle, Make, IBM DBB

Eclipse (including IDz) , Visual Studio

# Imagine a unified pipeline on Z



# The value of early and extensive testing

*“80% of development costs are spent identifying and correcting defects” \*\**



During the  
Coding or  
Unit Testing  
phases

**\$80/defect**



During the  
**BUILD** phase

**\$240/defect**



During  
Quality Assurance  
or the System Test  
phases

**\$960/defect**



Once released  
into production

**\$7,600/defect**  
+  
**Law suits, loss  
of customer trust,  
damage to brand**

\*\*National Institute of Standards & Technology

Source: GBS Industry standard study

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

# DevOps for Enterprise Systems

## Key Takeaways

1. DevOps is about transforming application development and delivery in order to accelerate digital innovation.

*So DevOps is a topic for both business and IT roles in the organization.*

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

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

*Key concepts: automation, feedback loops.*

# Questions / Comments



# Agenda

9:00 Introductions

9:20 DevOps on z Systems Introduction



## 9:45 Application Delivery Foundation (ADFz):

- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break

10:30 LAB 1 - Working with z/OS using COBOL and DB2

or

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

11:45 Lunch

12:30 z Open Unit Test: Overview & Demonstration

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

2:00 – 4:30 Choose one Optional lab:

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

Lecture

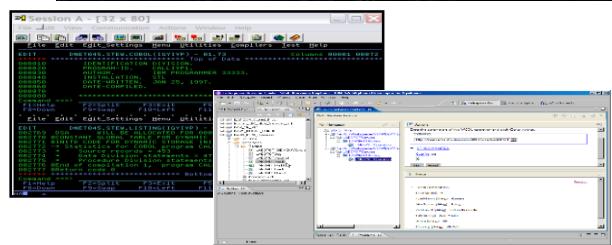
Labs

Breaks

# Hardware has been improved.. How about our Software?

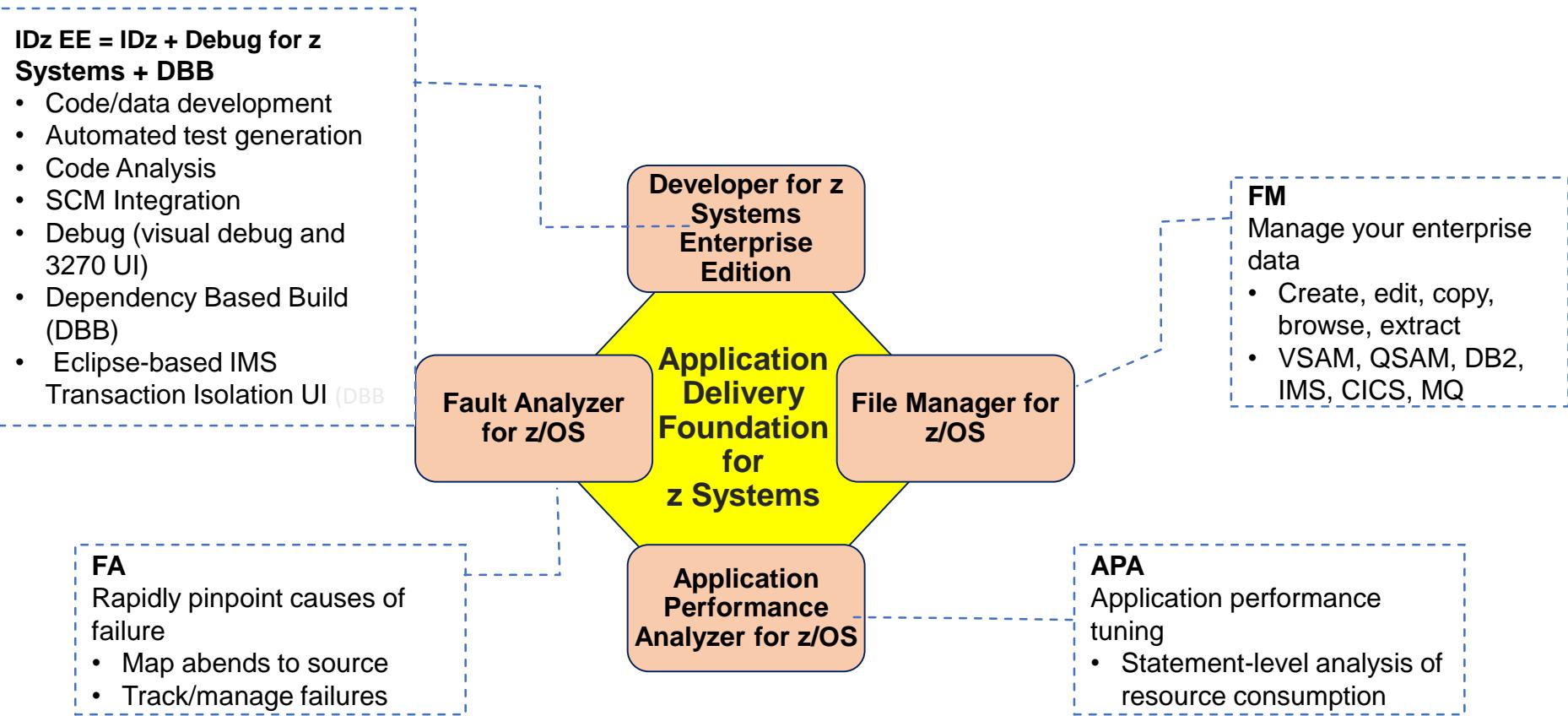
In September 1956, IBM launched the 305 RAMAC, the first computer with a hard disk drive (HDD). That HDD weighed over a ton and stored 5MB of data.

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

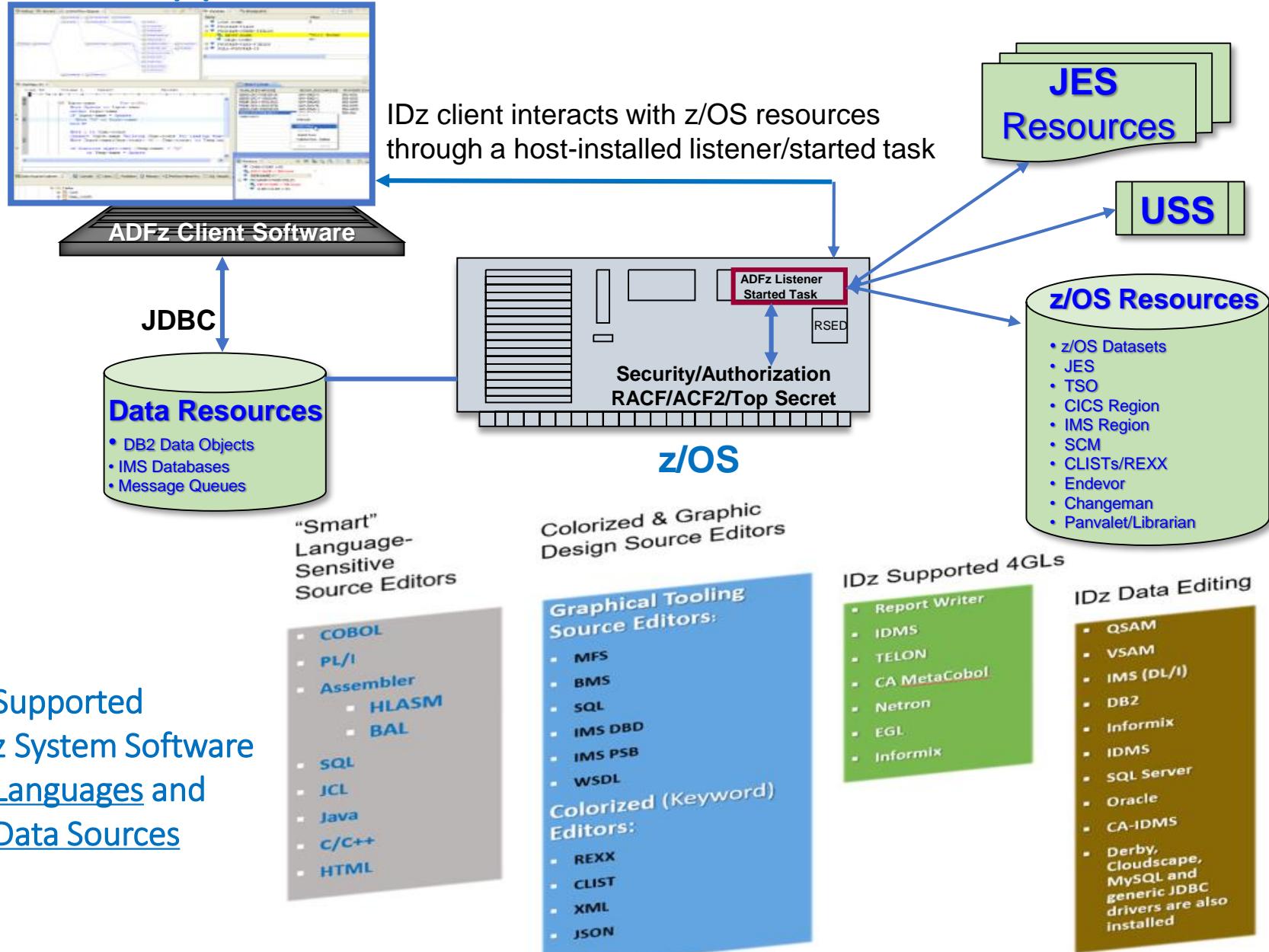


→ Software also needs Improvements....

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



# IDz: Supported Environments and Architecture



# Using IDz for Development & Testing

IDz is the ideal DevOps Entry-Point !!!

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

The screenshot displays the IDz IDE interface with several key features highlighted:

- ISPF-Style & Java/Eclipse-Style Editors**: A list of supported languages and frameworks, including JCL, ASM, IMS DBD/PSB, BMS/MFS, PLI, Java, C/C++, REXX, SQL, and others.
- Edit a program**: A code editor window showing COBOL source code for BNCHS601.cbl, specifically Line 376, Column 31, with an "Insert" cursor.
- File Compare**: A feature allowing comparison between two files, with options like Validate, Team, Compare With, Replace With, RAA Integration, Start Flagging Change, Syntax Check, Browse Copy Member, Open Copy Member, and Refresh Cached Copy.
- Submit-Compile/Link**: A feature for managing jobs and datasets, including a table of job entries and their status.
- Dataset Statistics**: A table showing dataset statistics for DDS0001, including Data Library, DSORG, Extents, LRECL, Primary, Recfm, Secondary, and Space Units.
- Access Jobs (JES Outlist facility)**: A table showing access to JES Outlist facility, listing resources, TSU numbers, and corresponding job details.
- File Search**: A search interface for finding files within the project structure.
- Access Datasets + Dataset Management**: A feature for managing datasets, shown in a tree view of the project structure.
- Remote Systems Details**: A panel showing details about the remote system, including job entry, return code, return info, user return, return status, spool tracks, and priority.

# Program Analytics

The image displays two side-by-side screenshots of z/OS Program Flow Analysis tools.

**Left Screenshot:** Shows a code editor window for file `EBUD01.cbl` with assembly-like code. A specific section of the code is highlighted:

```
-----+*A-1-B-----2-----3-----4-----5-----6-----7-----8-----+
000064          EXIT.
000065
000066
000067      A100-VERIFY-INPUT-DATE SECTION.
000068      IF L-INPUT-DATE NUMERIC
000069      MOVE L-INPUT-DATE TO W-INPUT-DATE
000070      DISPLAY 'WORKING DATE:'           W-INPUT-DATE
000071      MOVE W-CCYY TO RETURN-CODE
000072      ELSE
000073      DISPLAY 'INPUT DATE NOT NUMERIC'  INPUT-DATE
000074
000075
000076
```

Below the code is a data flow diagram titled "Data Flow Analysis". It shows various data items (W-RET-MM, W-MM, W-DD, W-CCYY, etc.) connected by arrows representing data flow between them.

**Right Screenshot:** Shows a code editor window for file `DF5IVNA34.CBL` with assembly-like code. A specific section of the code is highlighted:

```
-----+*A-1-B-----2-----3-----4-----5-----6-----7-----+
04980000          DLET-DB-END.
04990000          EXIT.
05000000
05010000
05020000
05030000
05040000
05050000
05060000
05070000
```

Below the code is a complex control flow graph titled "Program Flow Analysis". The graph consists of numerous nodes (e.g., TO-ADD, TO-END, TO-UPD, etc.) and directed edges representing the flow of control between them. A callout box points to a node labeled "MAIN RTN".

# SQL Coding within IDz editor

The screenshot shows the IDz editor interface with several windows open:

- Code Editor:** The main window displays a PL/I or COBOL source file (\*CURSRAVG.cbl) containing embedded SQL statements. A portion of the code is highlighted, showing a cursor declaration and a SELECT statement.
- Context Menu:** A context menu is open over the highlighted SQL code, with the "Tune SQL" and "Run SQL" options highlighted with a red box.
- Toolbars:** Standard toolbar icons for Paste, Shift+Insert, Select, Selected, Deselect, Filter view, Show all, Source, Preprocessor Statements, Refactor, Tune SQL, Run SQL, Refresh SQL in Outline View, Occurrences in Compilation Unit, Open Declaration, Open Perform Hierarchy, View, Show In, Open With, Profile As, Debug As, Run As, Code Coverage As, Validate, Software Analysis, Team, Replace With, Formatted Editor, Search selection in Lookup, Compare With, Start Flagging Changed Lines, Preferences..., Save and Syntax Check.
- Remote Systems:** A sidebar titled "Remote Systems" shows a tree structure of remote systems and their resources, including z/OS UNIX Files, z/OS UNIX Shells, MVS Files, Retrievied Data Sets, My Data Sets, Test Libraries, CX Files, P\* Files, Copybook and DCLGEN Libraries, and Datasets.
- Search Results:** A search results window titled "remote z/OS Search" shows results for DDS0001.PATINS, including "SQL Results" and "SQL Results X".
- Data Grid:** A data grid window titled "SQL Statement Results" displays a table of results with columns labeled 2, 3, 4, 5, 6, and 7. The data is as follows:

	2	3	4	5	6	7
1	2	2	2	15.99	26.75	21.370000
1	4	3		8.89	32.00	22.546666
1	1	1		212.01	212.01	212.0100
NULL	NULL	NULL		6.11	32.41	17.250000
				67.82	67.82	67.820000

SQL Statement Results

Can also create/test new  
embedded SQL  
functionality within PL/I  
COBOL Edit – including  
host variables

Source SELECT D 3/2/17 1 E0SDR205

# COBOL Program Refactoring

The screenshot shows a COBOL program named MSTRUPTD.cbl in an IDE. The code includes several sections like COMPUTE-OUT-OF-NETWORK, EVALUATE EMP-STATE, and COMPUTE PATIENT-TOT-AMT. A context menu is open over the code, with the 'Create Program...' and 'Create Copybook...' options highlighted in red. A blue arrow points from the top right towards the menu. The 'Create Program...' dialog is also visible, showing fields for Name (ONNATES), Data Set (USER03.BNCHMR.COBOL), and Communication Type (CALL).

```
598      7000-COMPUTE-OUT-OF-NETWORK.  
599      *** OUT OF NETWORK RATES FOR PATIENTS  
600      MOVE 72 TO REIMBURSE-PCT IN CALC-COSTS-REC.  
601      MOVE ZERO TO STATE-FACTOR.  
602  
603      EVALUATE EMP-STATE  
604      WHEN "NC" MOVE 82 TO STATE-FACTOR  
605      WHEN "ND" MOVE 54 TO STATE-FACTOR  
606      WHEN "NY" MOVE 19 TO STATE-FACTOR  
607      WHEN "ND" MOVE 79 TO STATE-FACTOR  
608      WHEN "AZ" MOVE 48 TO STATE-FACTOR  
609      WHEN "AR" MOVE 68 TO STATE-FACTOR  
610      WHEN "ID" MOVE 17 TO STATE-FACTOR  
611      WHEN "DE" MOVE 90 TO STATE-FACTOR  
612      WHEN "WA" MOVE 85 TO STATE-FACTOR  
613      WHEN "TX" MOVE 58 TO STATE-FACTOR  
614      WHEN "PA" MOVE 58 TO STATE-FACTOR  
615      WHEN "HI" MOVE 92 TO STATE-FACTOR  
616      WHEN "OR" MOVE 60 TO STATE-FACTOR  
617  
618      END-EVALUATE  
619  
620      COMPUTE PATIENT-TOT-AMT =  
621      ( WS-LAB-CHARGES + WS-EQUIP-CHARGES )  
622      * ( REIMBURSE-PCT / 100 ) + ( STATE-FACTOR / 100 )  
623  
624      MOVE STATE-FACTOR TO COPAY IN PATIENT-MASTER-REC.  
625  
626      7000-EXIT.  
627      EXIT.  
628  
629      MOVE 72 TO REIMBURSE-PCT IN CALC-COSTS-REC.  
630      MOVE ZERO TO STATE-FACTOR.  
631  
632      EVALUATE EMP-STATE  
633      WHEN "NC" MOVE 82 TO STATE-FACTOR  
634      WHEN "ND" MOVE 54 TO STATE-FACTOR  
635      WHEN "NY" MOVE 19 TO STATE-FACTOR  
636      WHEN "ND" MOVE 79 TO STATE-FACTOR  
637      WHEN "AZ" MOVE 48 TO STATE-FACTOR  
638      WHEN "AR" MOVE 68 TO STATE-FACTOR  
639      WHEN "ID" MOVE 17 TO STATE-FACTOR  
640      WHEN "DE" MOVE 90 TO STATE-FACTOR  
641      WHEN "WA" MOVE 85 TO STATE-FACTOR  
642      WHEN "TX" MOVE 58 TO STATE-FACTOR  
643      WHEN "PA" MOVE 58 TO STATE-FACTOR  
644      WHEN "HI" MOVE 92 TO STATE-FACTOR  
645      WHEN "OR" MOVE 60 TO STATE-FACTOR  
646  
647      END-EVALUATE  
648  
649      COMPUTE PATIENT-TOT-AMT =  
650      ( WS-LAB-CHARGES + WS-EQUIP-CHARGES )  
651      * ( REIMBURSE-PCT / 100 ) + ( STATE-FACTOR / 100 )  
652  
653      MOVE STATE-FACTOR TO COPAY IN PATIENT-MASTER-REC.  
654  
655      7000-EXIT.  
656      EXIT.
```

## • Modularization

- Increase application testability
- Easier to create automated unit test cases
- Accelerate development and deployment
- Separate presentation logic from business logic and data access logic
- Allows for concurrent development

## • Maintainability

- Improve readability of existing applications making them easier to maintain
- Decrease complexity of application logic

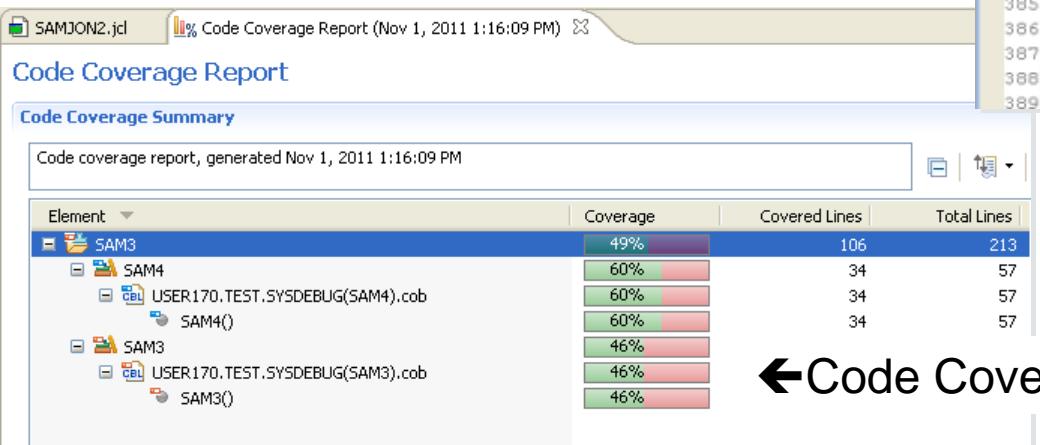
# Code Coverage

## Extension to Testing & Debugging:

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

```
000      024000  PERFORM 710-READ-TRAN-FILE.
001      024500
002      024600  IF WS-TRAN-EOF NOT = 'Y'
003      024700  COMPUTE NUM-TRAN-RECS = NUM-TRAN-RECS + 1
004      024800  MOVE 'Y' TO WS-TRAN-OK
005      024900  IF TRAN-KEY < WS-PREV-TRAN-KEY
006      025000    MOVE 'TRANSACTION OUT OF SEQUENCE' TO ERR-MSG-DATA1
007      025100    MOVE SPACES TO ERR-MSG-DATA2
008      025200    PERFORM 299-REPORT-BAD-TRAN
009      025300  ELSE
010      025400    EVALUATE TRAN-CODE
011      025500      WHEN 'UPDATE'
012      025600        PERFORM 200-PROCESS-UPDATE-TRAN
013      025700
014      025800  END-EVALUATE
015      025900
016      026000  PERFORM 220-PROCESS-DELETE-TRAN
017      026100  WHEN OTHER
018      026200    IF TRAN-COMMENT NOT = '**'
019      026300      MOVE 'INVALID TRAN CODE:' TO ERR-MSG-DATA1
020      026400      MOVE TRAN-CODE TO ERR-MSG-DATA2
021      026500      PERFORM 299-REPORT-BAD-TRAN
022      026600
023      026700
024      026800
025      026900
026      027000
027      027100
028      027200
029      027300
030      027400
031      027500
032      027600
033      027700
034      027800
035      027900
036      028000
037      028100
038      028200
039      028300
040      028400
041      028500
042      028600
043      028700
044      028800
045      028900
046      029000
047      029100
048      029200
049      029300
050      029400
051      029500
052      029600
053      029700
054      029800
055      029900
056      029950  END-IF
057      029970
058      029990  MOVE TRAN-KEY TO WS-PREV-TRAN-KEY
059      029995  IF WS-TRAN-OK = 'Y'
```

## ← Coverage Details



Code Coverage central to DevOps  
Continuous Testing

## ← Code Coverage Report

# ADFz: “File Manager (FM)” Edit file types

- Copy/Field records
- Different record types in dataset
- Sample
- Sort
- Fixed, Variable

- HFS files
- Sequential (PDS and PDSE)
- VSAM (including IAM)
- Websphere MQ messages on a queue
- CICS: TS or TD queues

The screenshot displays the ADFz interface with several windows open:

- Main Window:** Shows a table of data with columns: PATIENT-ID, INS-COMPANY-PRIMARY-ID, CARRIER-NAME, and CARRIER-PH. An orange arrow points from the 'File Manager Editor' window below to the CARRIER-PH column.
- File Manager Editor:** A detailed view of the CARRIER-PH column, showing individual records for each carrier. The right side of this window has a context menu with options like 'File Manager Editor', 'File Manager', and 'Delete Records'.
- Remote Systems:** A list of remote systems connected to DDS0001.PATINS. The 'File Manager' option in the context menu is highlighted with an orange circle.
- Bottom Left Window:** Shows a layout named 'PATIENT-INSURANCE' with a table of fields and their definitions.
- Bottom Right Text Box:** A callout box with a blue border contains the text: "Access VSAM Files directly from IDz Edit (Remote Systems view)" and a list of features:
  - 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

The screenshot shows the IBM Problem Determination Tools Studio interface with the following components:

- Report Outline Hyperlinks:** A callout points to the left pane where a list of ABEND reports is displayed.
- Fault Analyzer history files:** A callout points to the tree view under "System..." showing "Fault Analyzer for z/OS" and "Browse History Files".
- Available ABEND reports:** A callout points to the "Event Details" tab of the main report window.
- List of fault (ABEND) entries:** A callout points to the "Abend Info" tab of the main report window, which displays a table of ABEND entries.
- Reports – Hyperlinks to program source file:** A callout points to the right pane where COBOL source code is shown with hyperlinks.
- ABEND cause:** A blue arrow points from the "Reports – Hyperlinks to program source file" area to the COBOL source code.

**Main Report Content (Abend Info Tab):**

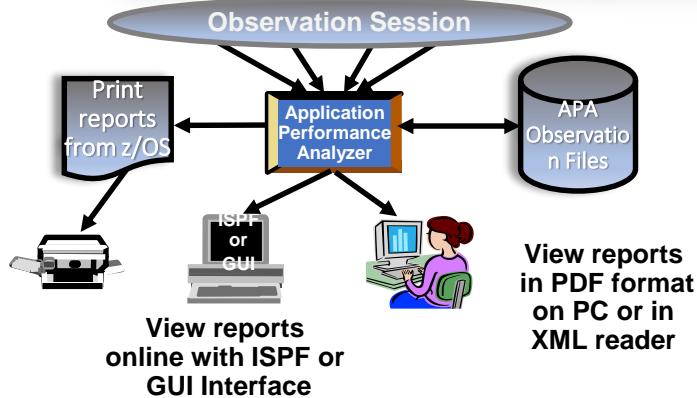
FAULT_ID	JOB/TRAN	USER
F00056	CICSC41F/PSC1	KPHU
F00055	CICSC41F/PSC1	KPHU
F00054	CICSC41F/PSC1	KPHU
F00053	CICSC41F/PSC1	KPHU
F00052	CICSC41F/CFA	TSS2
F00051	CICSC41F/PSC1	MAC

**COBOL Source Code (Right Pane):**

```
000100* A program for debugging
000200 IDENTIFICATION DIVISION.
000300 PROGRAM-ID. SOC7NEW.
000400
000500 DATA DIVISION.
000600 WORKING-STORAGE SECTION.
000610 01 REC-KEY    PIC X(13) VALUE 'REC-KEY=12345'.
000700 01 FIELD1    PIC S9(5)V99 COMP-3 VALUE 100.
000800 01 FIELD2    PIC S9(5)V99 COMP-3.
000900 01 FIELD2-CHAR REDEFINES FIELD2
001000          PIC X(4).
001100 01 BALANCE   PIC S9(9)V99 COMP-3 VALUE 0.
001200 01 BALANCE-CHAR PIC 9(9).99- DISPLAY.
001300
001400 PROCEDURE DIVISION.
001500          DISPLAY " SOC7NEW Program Begin: STARTING".
001600*      The uninitialized FIELD2 will cause the
001700*      compute statement to fail
001800          COMPUTE BALANCE = FIELD1 / FIELD2.
001900          MOVE BALANCE TO BALANCE-CHAR.
002000          DISPLAY " BALANCE IS " BALANCE-CHAR.
002100          GOBACK.
002200 END PROGRAM SOC7NEW.
```

# 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

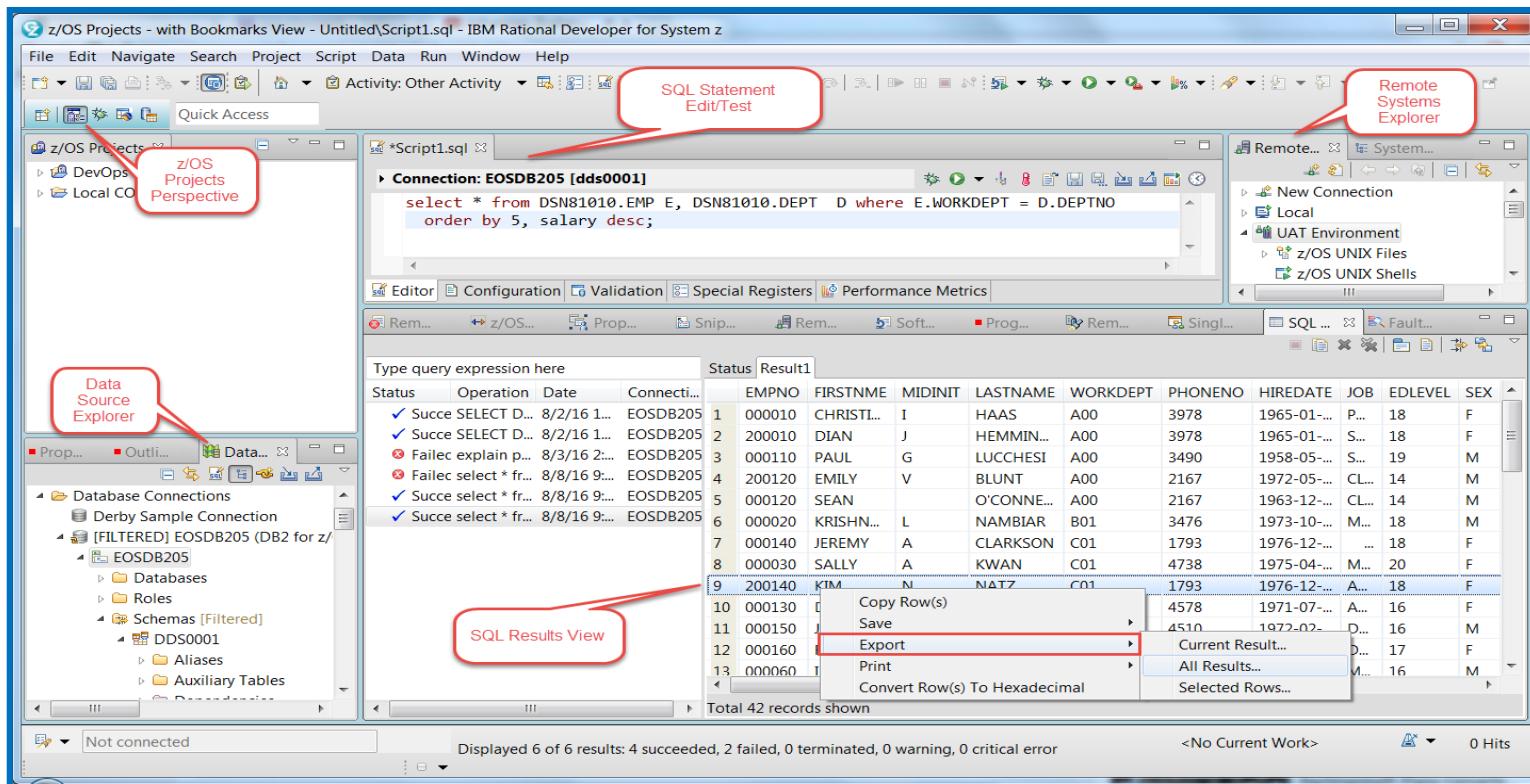


- Specify number of times APA monitors a job's performance when job or program becomes active
- Invoke the **monitoring capability** from other programs such as IBM Tivoli® OMEGAMON
- Compare two observation reports to see the relevant differences, to supplement threshold monitoring
- Assembler, COBOL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

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

A flexible graphical interface to DB2 and SQL

- DB2 Table/View/Index Analysis
- DB2 Data Model and for Test Data Management and manipulation:
  - CRUD
  - Simple Table row/column sub-setting (sampling)
  - Test/Run/Tune SQL directly from COBOL or PL/I programs
  - Run existing SPUFI files (DDL, DML)
- SQL code and Test
  - Code embedded SQL directly within COBOL, PL/I and Assembler programs
  - Statement content assist from the DB2 Catalog
- Code SQL with graphical tooling
- Interactively Code/Test/Tune SQL statements



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

## Using “Data” perspective

The screenshot shows the IBM Developer for z Systems interface in the "Data" perspective. Key components include:

- Data Project Explorer:** Shows project areas like Repository Connections, My Repository Workspaces, and Team Artifacts.
- Application Discovery Analysis:** Displays a complex network diagram of system relationships.
- Data Source Explorer:** Lists tables, temporary tables, user-defined functions, and types. A context menu is open over a table entry, with the option "Add to Overview Diagram" circled in blue.
- SQL Outline view:** A panel showing an SQL query: `SELECT DEPT, MIN(PERF), MAX(PERF), AVG(PERF), MIN(HOURS), MAX(HOURS) FROM VACT`. A callout points to the "Open Visual Explain" option in the context menu.
- Visual Explain diagram:** A detailed diagram showing the execution plan for the query, including stages like SORT, NESTED LOOP JOIN, and TABLE SCANS. It includes cost estimates and optimizer notes.
- Properties and Results panels:** Show detailed statistics for the query phases, such as Outer Input Cardinality, Inner Input Cardinality, Join Predicates, and Cumulative CPU Cost.

**From SQL Outline view:**

- Open Visual Explain on and embedded SQL statement

**View/Analyze results**

**Save results to discuss with DB2/DBA operations team**

**Different Query stages in the Visual Explain diagram**

Pop-up DB2 Optimizer analyzes access path and estimates query step costs

Detailed Cost estimates for the query and each query phase

**DB2 Visual Explain**

# What is IBM Dependency Based Build?

- **IBM Dependency Based Build (DBB)** is a tool to build traditional z/OS applications such as COBOL and PL/I as part of a continuous integration (CI) pipeline.
- Provides a modern scripting language based automation capability that can be used on z/OS.
  - The DBB API is written in Java and can be called by Java applications as well as Java based scripting languages such as [Groovy](#), [JRuby](#), [Jython](#), [Ant](#), [Maven](#), etc.
- **Not tied to a specific source code manager** (most common → Git) or **pipeline automation tool** (most common → Jenkins) .
- Consists of a [build toolkit](#) (Java API, Groovy Installation) [installed on USS \(UNIX Systems Services \)](#) and a [Liberty application server](#) that hosts [build metadata](#) (dependency data, build results) installed on Linux.



# DBB Sample Groovy compile step versus JCL

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

println("Copying source from zFS to PDS . . .")
def copy = new CopyToPDS()
    .file(new File("/u/usr1/build/helloworld.cbl"))
    .dataset("USR1.BUILD.COBOL").member("HELLO")
copy.execute()

println("Compiling . . .")
def compile = new MVSExec().pgm("IGYCRCTL").parm("LIB")
compile.dd(new DDStatement().name("TASKLIB")
    .dsn("IGY.SIGYCOMP").options("shr"))
compile.dd(new DDStatement().name("SYSIN")
    .dsn("USR1.BUILD.COBOL(HELLO)").options("shr"))
compile.dd(new DDStatement().name("SYSLIN")
    .dsn("USR1.BUILD.OBJ(HELLO)").options("shr"))
compile.dd(new DDStatement().name("SYSPRINT")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT1")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT2")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT3")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT4")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT5")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT6")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT7")
    .options("tracks space(5,5) unit(vio) new"))
compile.copy(new CopyToHFS().ddName("SYSPRINT")
    .file(new File("/u/usr1/build/helloworld.log")))
def rc = compile.execute()

if (rc > 4)
    println("Compile failed!  RC=$rc")
else
    println("Compile successful!  RC=$rc")
```

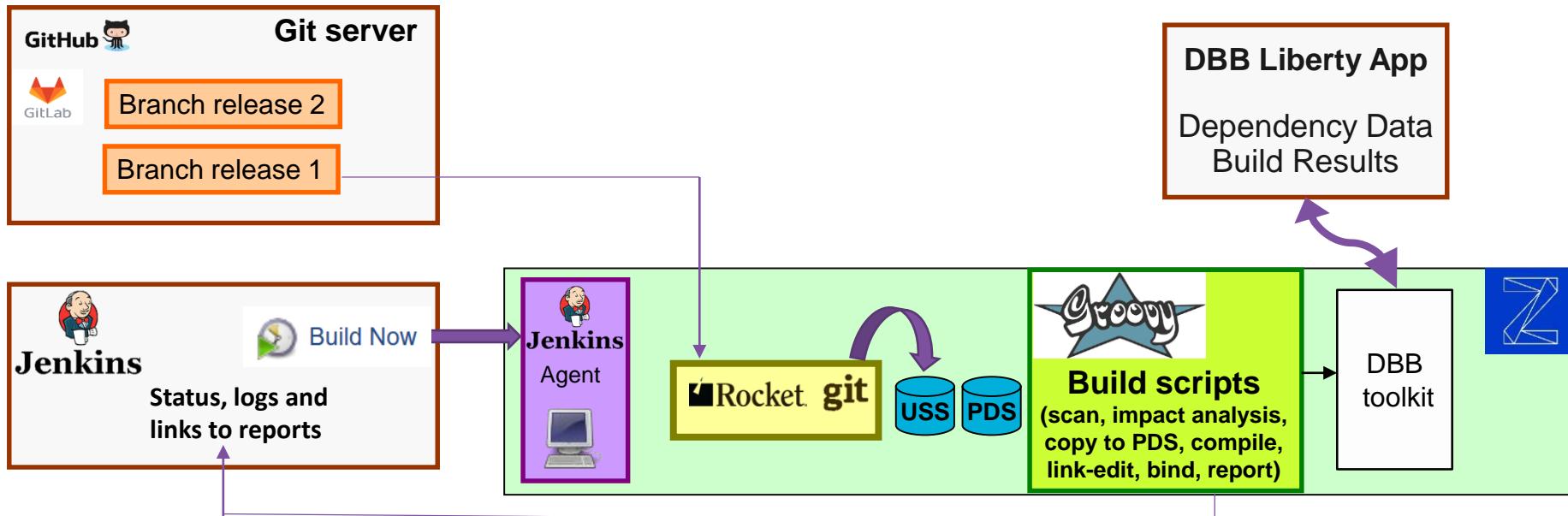


```
//COBOL EXEC PGM=IGYCRCTL,REGION=0M,
//          PARM='LIB'
///*
//STEPLIB  DD DISP=SHR,DSN=IGY.SIGYCOMP
///*
//SYSIN    DD DISP=SHR,DSN=USER1.BUILD.COBOL(HELLO)
//SYSLIN   DD DISP=SHR,DSN=USER1.BUILD.OBJ(HELLO)
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT2   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT3   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT4   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT5   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT6   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT7   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
```



# IBM Dependency Based Build (DBB)

- Intelligent build environment for applications
  - **Groovy** driven build environment for **compiling, linking and processing** z/OS programs and applications
  - Dependency identification and understanding
    - Automatically identify program dependencies , build intelligently based on dependencies
  - Integration with any CI tooling (e.g. *Jenkins*)



# Questions / Comments



# Agenda

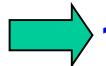
9:00 Introductions

9:20 DevOps on z Systems Introduction

9:45 Application Delivery Foundation (ADFz):

- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break



**10:30 LAB 1 - Working with z/OS using COBOL and DB2**

or

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

11:45 Lunch

12:30 z Open Unit Test: Overview & Demonstration

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

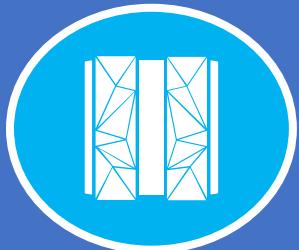
2:00 – 4:30 Choose one Optional lab:

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

■ Lecture

■ Labs

■ Breaks



Labs

# DevOps Mainframe Tooling - Labs

## Labs using VMware ...

1. User id → **empot01** and password → **empot01**

2. Follow exactly what is described in the lab

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

**Tips!**

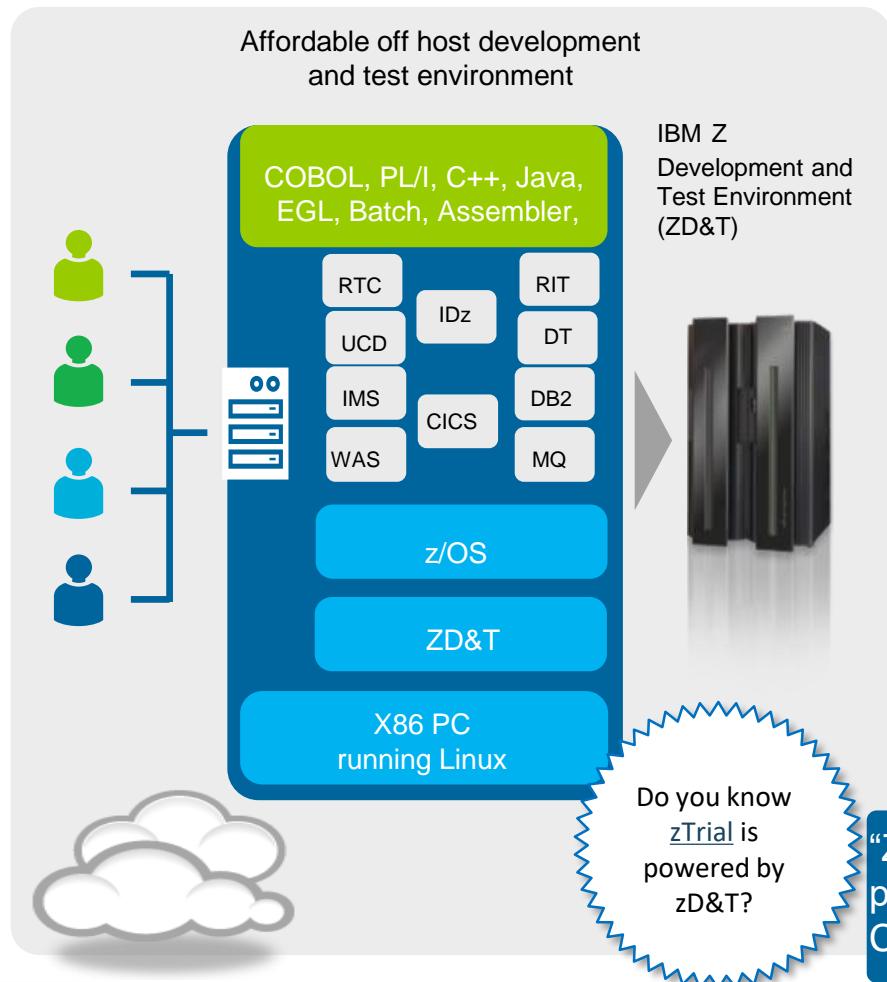
1) If you want to have the lab in HTML format displayed in your browser, you can find it at the location: C:\RDZ8.0\_POTVHTML  
Then you can COPY/PASTE the names and the code, using the Browser.

2) Most of the labs will be performed under VMware. So we will run 2 'Windows' in the same machine. This will cause some overhead and sometimes performance will not be as good as if the program would be running in the native Windows... So, please be patient.

3) If you lost the "VMware full screen" either type **Ctrl + Alt + Enter** or use the VMware icon  (on top).

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

# ZD&T → Dev/Test Environment



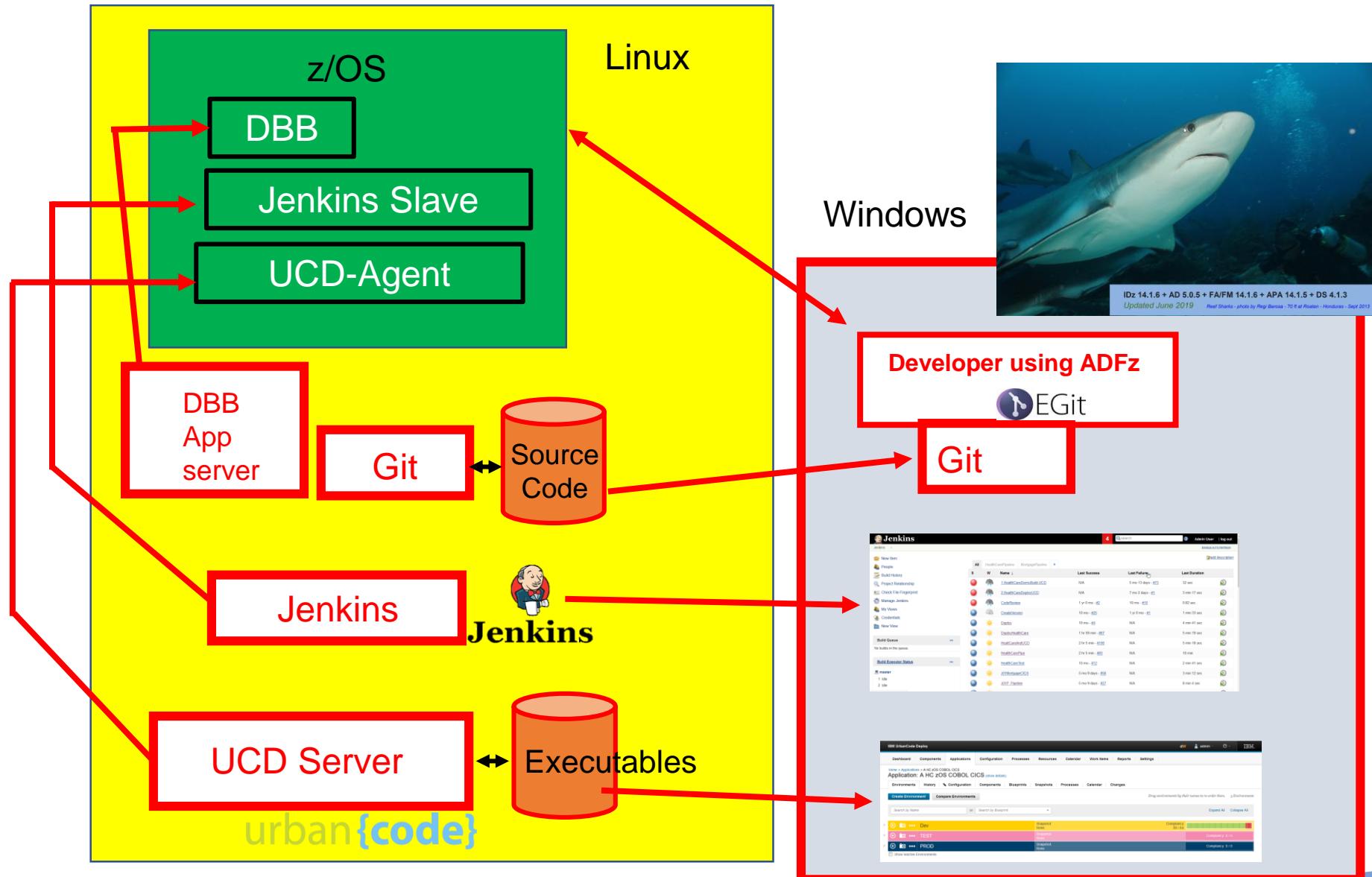
- Develop and test z/OS applications anywhere, anytime
  - Free up mainframe development MIPS for production workload
  - Eliminate costly delays by reducing burden on existing IT operations staff
  - Reduce time to value and minimize ongoing administration and capital expense with zD&T Cloud Managed DevOps
  - Exploit the z14 hardware capability, **including z14 pervasive encryption**
  - Comprehensive z/OS 2.3 software distribution:
    - z/OS plus major subsystems
    - Underpinned by the z/OS components of DevOps for the Enterprise development, test, and deployment tooling
    - **CICS subsystem pre-provisioned**
- New in V12: Cloud Integration & Deployment Automation**

“ZD&T improved our development and testing timeline and provided stability and quality” Developer, Large Enterprise Computer Services Company

<https://www.techvalidate.com/tvid/C99-3E2-1ED>

# zD&T on Cloud environment

zD&T



# Lab 1: Working with mainframe using COBOL and DB2

Duration: 90 Minutes

## ■ Objective

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

## Key Activities

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

# Lab 1: Working with mainframe using COBOL and DB2

zos.dev:2800/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 0008 (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      01 RESULT                  ZPF 99.
```

520-LOGIC.

IF WHICH-LAB = 'LAB2'  
\* If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO  
MOVE "REGI0B" 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

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

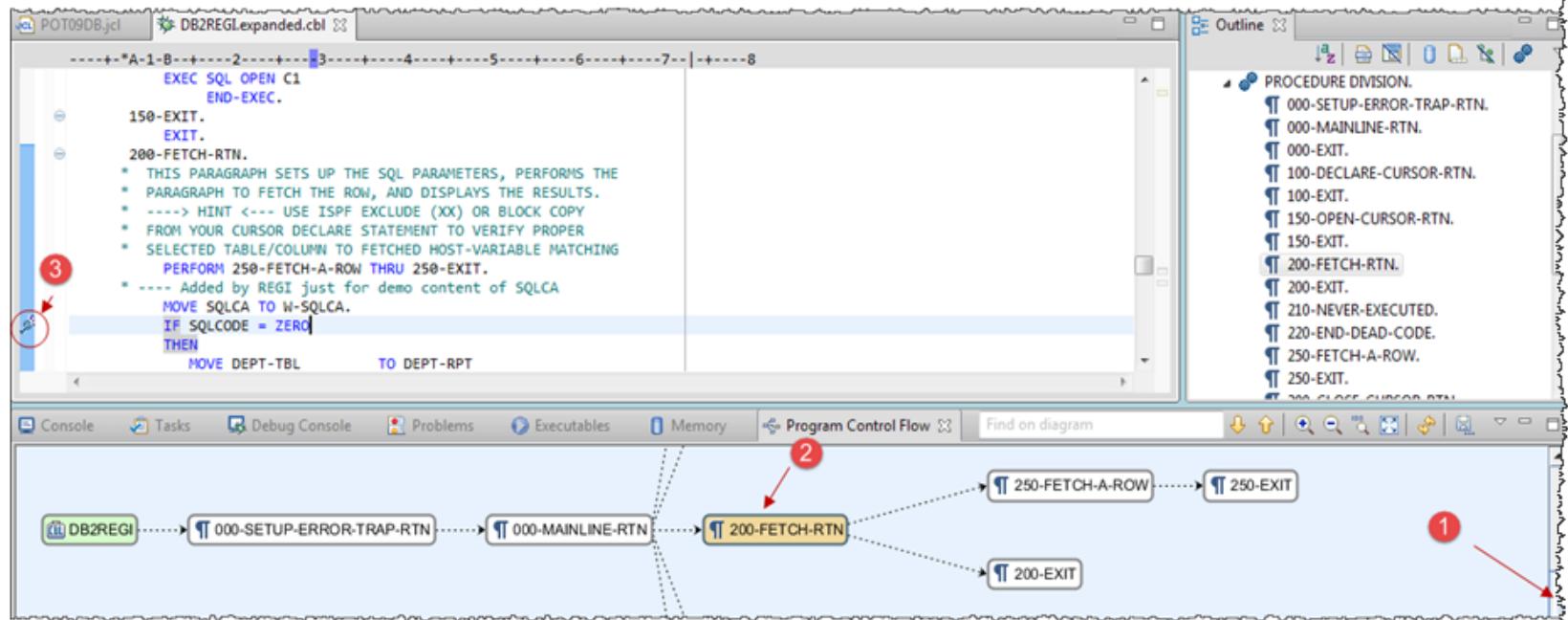
Here calls REGI0B that abends

# Lab 1: Don't Miss... The Debug... Page 20

## 6.2.5 The Program Control Flow diagram opens

- ▶ On Program Control Flow view, ① scroll down and click on ② 200-FETCH-RTN
- ▶ On the COBOL editor move the mouse to the blue column on left and ③ double click on the line **IF SQLCODE = ZERO** to create a breakpoint.

The icon  is shown



# Lab 1: Don't Miss... Code Coverage ...Page 47

## 8.2.4 ► Double click on DB2REGI.expanded.cbl

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

```
JCL P0T09CC.jcl @ DB2REGI_2017_01_06_220851_0268 CBL DB2REGI.expanded.cbl
-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+
      *A:1-B:2-C:3-D:4-E:5-F:6-G:7-H:8
      |
      | EXIT.
      | 350-TERMINATE-RTN.
      |   MOVE ROW-KTR TO ROW-STAT.
      |   DISPLAY ROW-MSG.
      | 350-EXIT
      *   EXIT.
      *   GO TO 500-SECOND-PART.
      999-ERROR-TRAP-RTN.
*****
*     ERROR TRAPPING ROUTINE FOR NEGATIVE SQLCODES *
*****
      DISPLAY '***** WE HAVE A SERIOUS PROBLEM HERE *****'.
      DISPLAY '999-ERROR-TRAP-RTN '.
      MULTIPLY SQLCODE BY -1 GIVING SQLCODE.
      DISPLAY 'SQLCODE ==> ' SQLCODE.
      DISPLAY SQLCA.
      DISPLAY SQLERRM.
      EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
      EXEC SQL ROLLBACK WORK END-EXEC.
      ROLLBACK
      *-----+
      500-SECOND-PART.
      MOVE 2 TO BRANCHFLAG.
```

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

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

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

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

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

Overview of development tasks    User id →  empot05 and password → empot05

## 1. Get familiar with the application using the 3270 terminal

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

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

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

## 3. Modify the COBOL code using IDz.

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

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

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

## 5. Push and Commit the changed code to Git .

→ You will commit the changes to Git.

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

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

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

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

## 8. (Optional) Understanding DBB Build Reports

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

# Agenda

9:00 Introductions

9:20 DevOps on z Systems Introduction

9:45 Application Delivery Foundation (ADFz):

- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break

10:30 LAB 1 - Working with z/OS using COBOL and DB2

or

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

11:45 Lunch

## → 12:30 z Open Unit Test: Overview & Demonstration

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

2:00 – 4:30 Choose one Optional lab:

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

■ Lecture

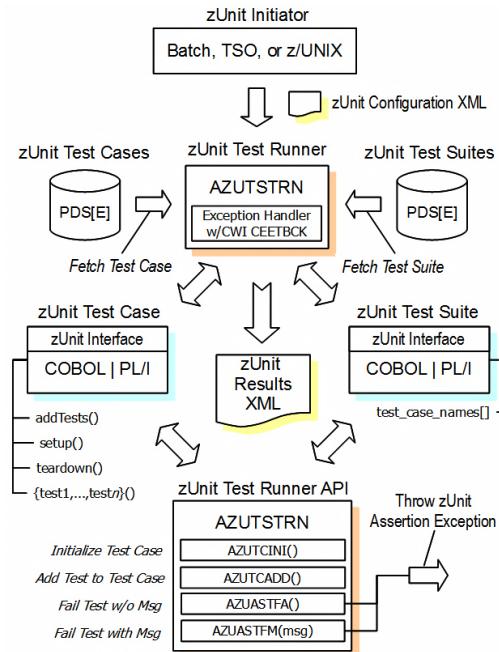
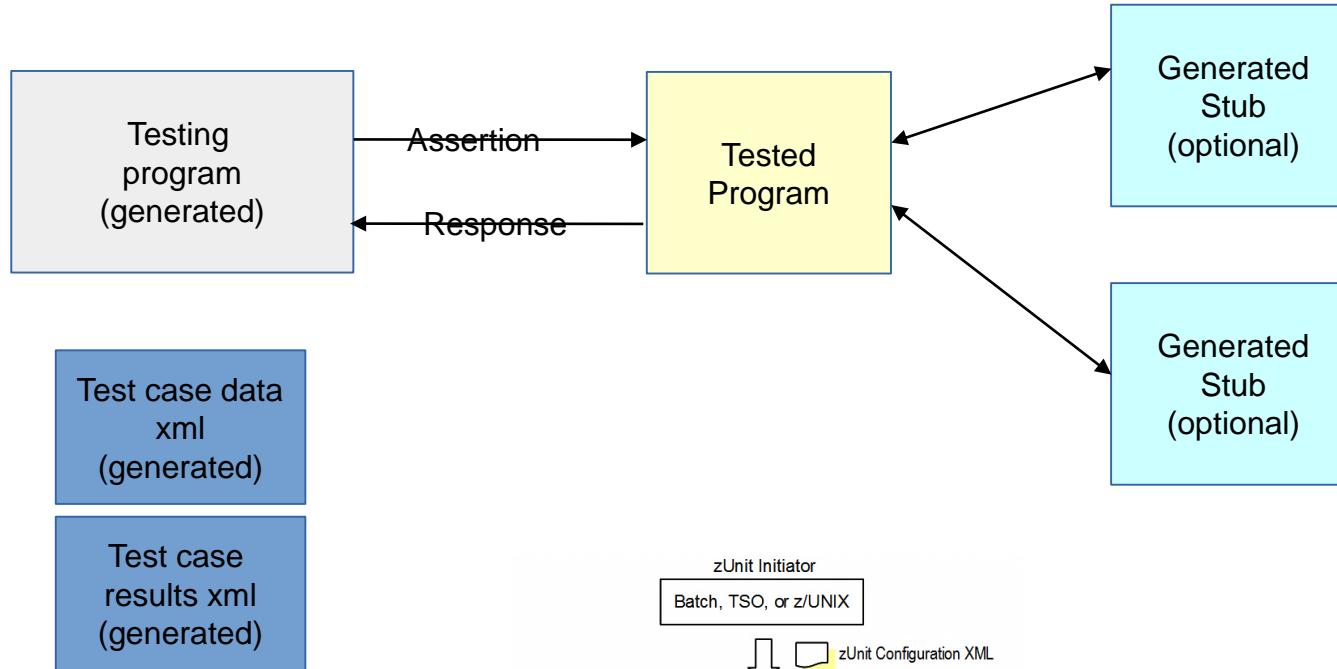
■ Labs

■ Breaks

# What is Unit Testing?

- A Unit is the smallest practical piece of software to be tested
  - Object oriented languages may treat a Class or Method as a unit
  - Procedural languages may treat a program as a unit
- Unit Testing is a method of testing software units to determine if they are fit for use
  - Unit Tests are usually created by developers, using development tools and unit testing frameworks
  - Unit Tests are expected to run quickly, so they can be run repeatedly
  - Some developers prefer that unit tests isolate the unit to be tested from other units and resources

# IDz zUnit Basics



# zUnit Basic Workflow



1      2      3      4      5      6

Generate  
Test  
Case  
Wizard

Test Data  
Entry  
Editor

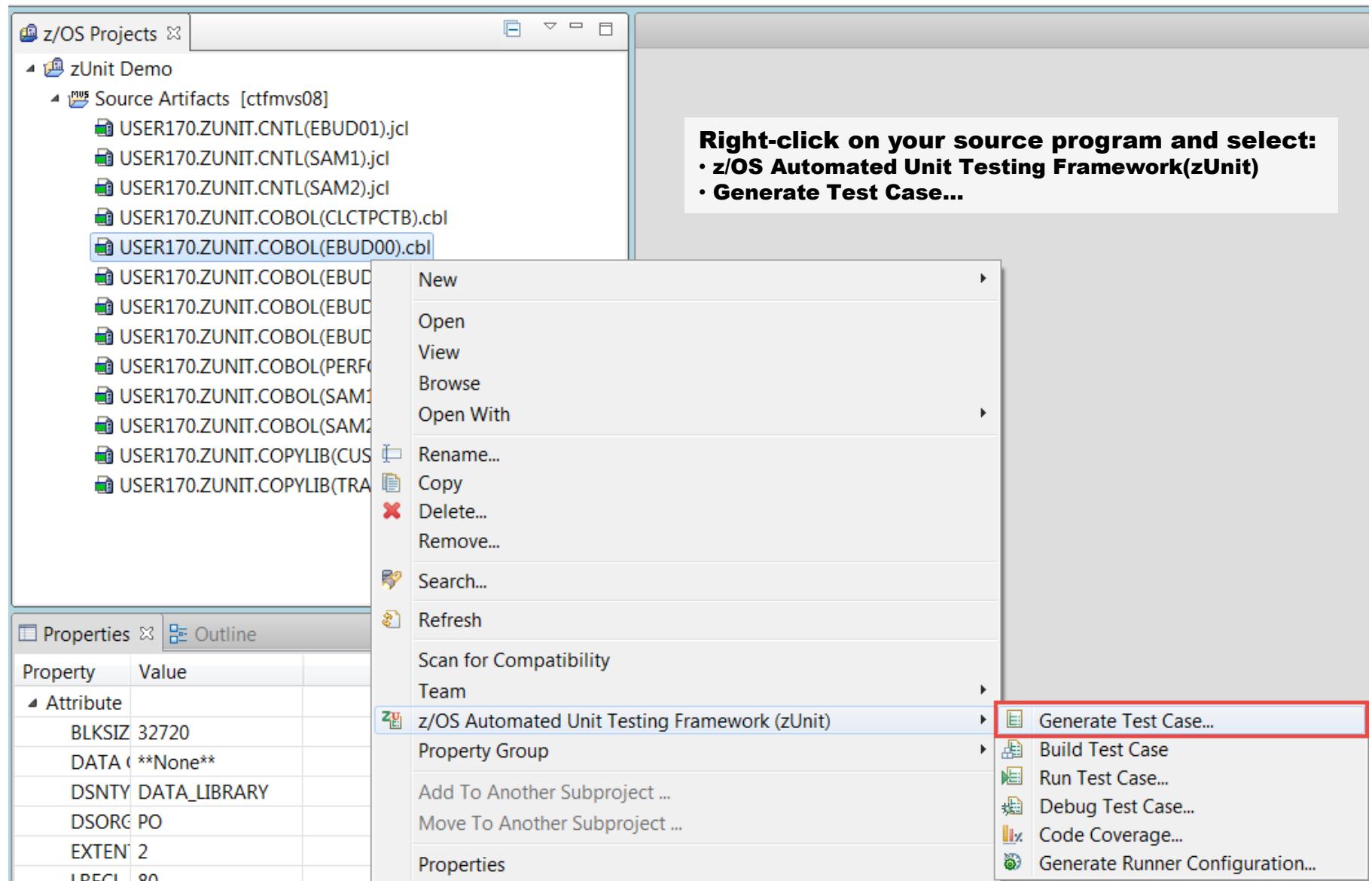
Generate  
Test  
Case  
Program

Build  
Test  
Case

Run Test

Examine  
Test  
Results

# Generating the Test Case - Using the IDz Wizard



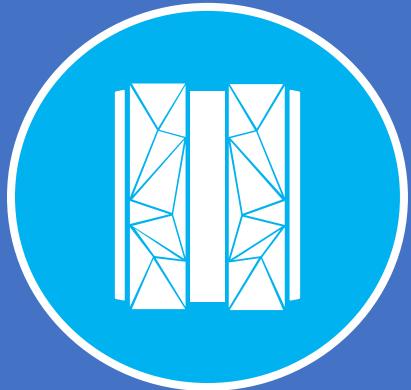
# IBM zUnit for CICS programs

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

Currently  
COBOL only

The screenshot shows a z/OS terminal window titled '\*EPSCMORT' containing a configuration menu with sections like 'Test procedure', 'Subprograms', and 'CICS'. A modal dialog box titled 'Record data for test case' is open in the foreground. It contains instructions: 'It looks like you have a CICS program. To record your program's data:' followed by three steps: 1. Press the Start recording button, 2. Run your program, 3. Once finished, press stop recording button. Below the steps, it says 'Your recorded data can be imported into a test case from the Test Case Editor.' A 'Recording Service URL' input field contains '4.svl.ibm.com:41048/zunit/' and a 'Start recording' button is visible. In the bottom right corner of the main window, there is a small table:

Z	EIBRPN	X(Z)	DISPLAY
2	EIBRCODE	X(6)	DISPLAY
2	EIBDS	X(8)	DISPLAY
2	EIBREQID	X(8)	DISPLAY
2	EIBRSRCE	X(8)	DISPLAY



DevOps Mainframe Tooling

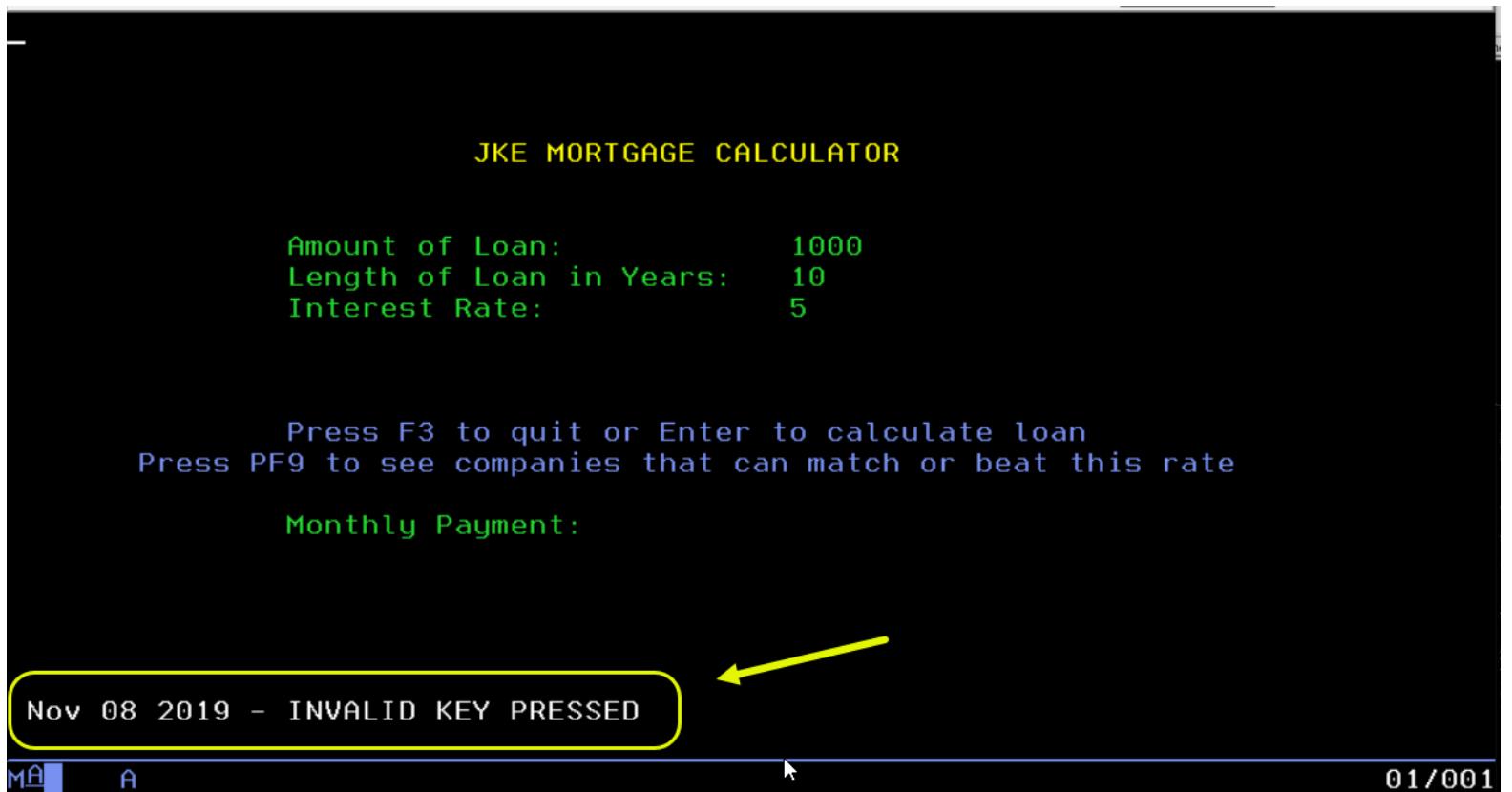
# Demonstration

Z Unit Test using IDz (Zunit)

Demonstration

# Scenario: COBOL CICS Transaction

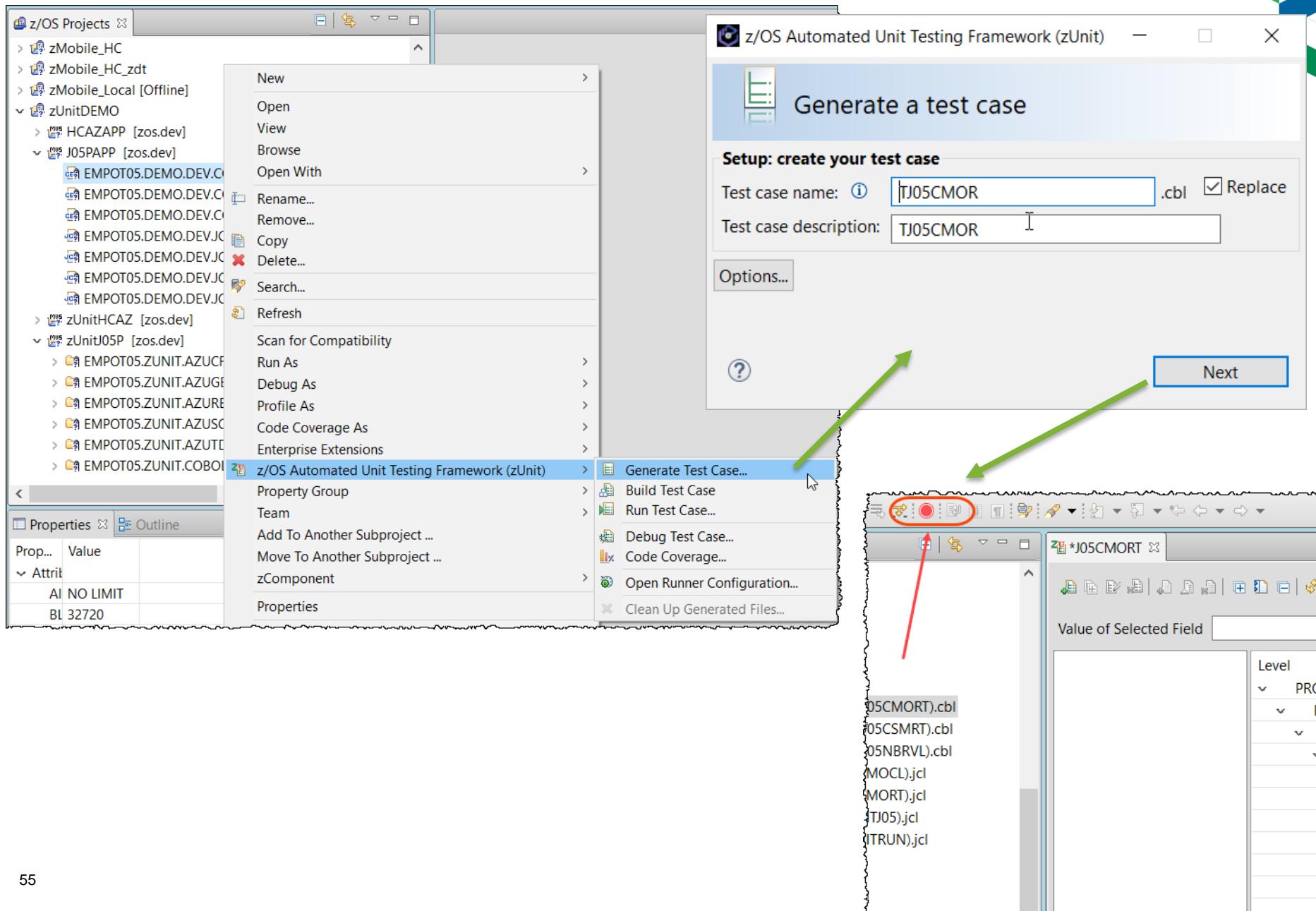
*Using transaction J05P  
and pressing F1  
message is displayed*



# Unit test on COBOL/CICS program J05CMORT

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

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





# 1. Record CICS program interaction using IDz

z/OS Automated Unit Testing Framework (zU...)

## Record data for test case

It looks like you have a CICS program.  
To record your program's data:

1. Press the Start recording button
2. Run your program
3. Once finished, press stop recording button

Your recorded data can be imported into a test case from the Test Case Editor.

Recording Service URL: <http://zos.dev:6486/zunit/> Start recording

Value of Selected Field Nov 08 2019 - INVALID KEY PRESSED

TEST2	Level	Name	PIC	USAGE	TEST2 :Input	TEST2 :Expected
	v 2	REDEFINES MS...				
	2	MSGERRF	X	DISPLAY		
	v 2 (redefines)	FILLER				
	3	MSGERRA	X	DISPLAY		
	2	FILLER	X(2)	DISPLAY		
	2	MSGERRI	X(80)	DISPLAY	Nov 08 2019 - I...	Nov 08 2019 - I...
	v Layout					
	v 1	JKEMENUO				

The screenshot shows the z/OS Automated Unit Testing Framework interface. The main window is titled 'Record data for test case'. It contains instructions for recording CICS program data, a recording service URL input field, and a 'Start recording' button. A red arrow points to the 'Start recording' button. Below this, a CICS terminal window is displayed with the title '\*J05CMORT'. The terminal shows a table with data rows. A red box highlights the value 'INVALID KEY PRESSED' in the 'Value of Selected Field' field. Another red box highlights the row for 'MSGERRI' in the CICS table, which has columns for TEST2, Level, Name, PIC, USAGE, TEST2 :Input, and TEST2 :Expected. The 'TEST2 :Input' column for the MSGERRI row contains the value 'Nov 08 2019 - I...', which is also highlighted by a red box.

# Unit test on COBOL/CICS program J05CMORT

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

## 2. Generate, build and run the unit test

The screenshot illustrates the workflow for generating, building, and running a unit test case using the z/OS Automated Unit Testing Framework.

**Step 1: Generating Test Cases**

In the main window, the "Generate" button is highlighted with a red circle and a red arrow pointing to it. A modal dialog titled "Generate test case program" is displayed, showing the "Programs to generate as test cases or stubs" section. It lists two items: "PROCEDURE DIVISION" (Test Case) and "J05NBRVL" (Subprogram). The "Generate, build, and run test case" radio button is selected, indicated by a red arrow.

**Step 2: Running the Test Case**

A second modal dialog titled "Run As Test Case" is shown. It specifies the runner configuration: Data set name: "EMPOT05.ZUNIT.AZUCFG" and Member name: "TJ05CMOR". The "Overwrite member" checkbox is checked. In the "Specify options for the generated runner result" section, the Data set name is "EMPOT05.ZUNIT.AZURES" and the Member name is "TJ05CMOR". The "Overwrite member" checkbox is also checked. The "OK" button is highlighted with a red circle and a red arrow.

**Step 3: Viewing Test Results**

The final window shows the "IBM z/OS Automated Unit Testing Framework (zUnit) Runner Results". The left pane displays the "Test cases and test results" tree view, which includes "zUnit runner results" and "Test case TJ05CMOR" (Test TEST2 (pass)). The right pane shows "Test case details" with the following information:

Test case ID:	f453459f-1eed-4b82-bd33-22191b752e60
Module name:	TJ05CMOR
Test case name:	TJ05CMOR
Result:	pass
Test count:	1
Tests passed:	1
Tests failed:	0
Tests in error:	0

A red arrow points to the "Tests passed" value of 1.

# Unit test on COBOL/CICS program J05CMORT

1. Record CICS program interaction using **IDz** .
2. Generate, build and run the unit test

Compile and link-edit the generated unit test program, followed by running the unit test.
3. Modify the program and rerun the unit test

Modify the program under test, rerun the unit test, and observe the failure of the test case.
4. Uses Jenkins to perform the Build, ZUnit and Deploy

The Z unit test will run from a Z/OS shell script and deploy only if does not fail.

### 3. Modify the program and rerun the unit test...

The screenshot displays the Rational Application Developer (RAD) interface, specifically the z/OS Projects perspective.

**Top Editor:** Shows a COBOL source file named `J05CMORT.cbl`. The code includes a `WHEN OTHER` section that moves low-values to `JKEMENUO` and handles invalid keys. A specific line of code, `MOVE 'Nov 15 2019 - INVALID KEY PRESSED' TO MSGERRO`, is highlighted with a red oval and a red arrow points to it from the left.

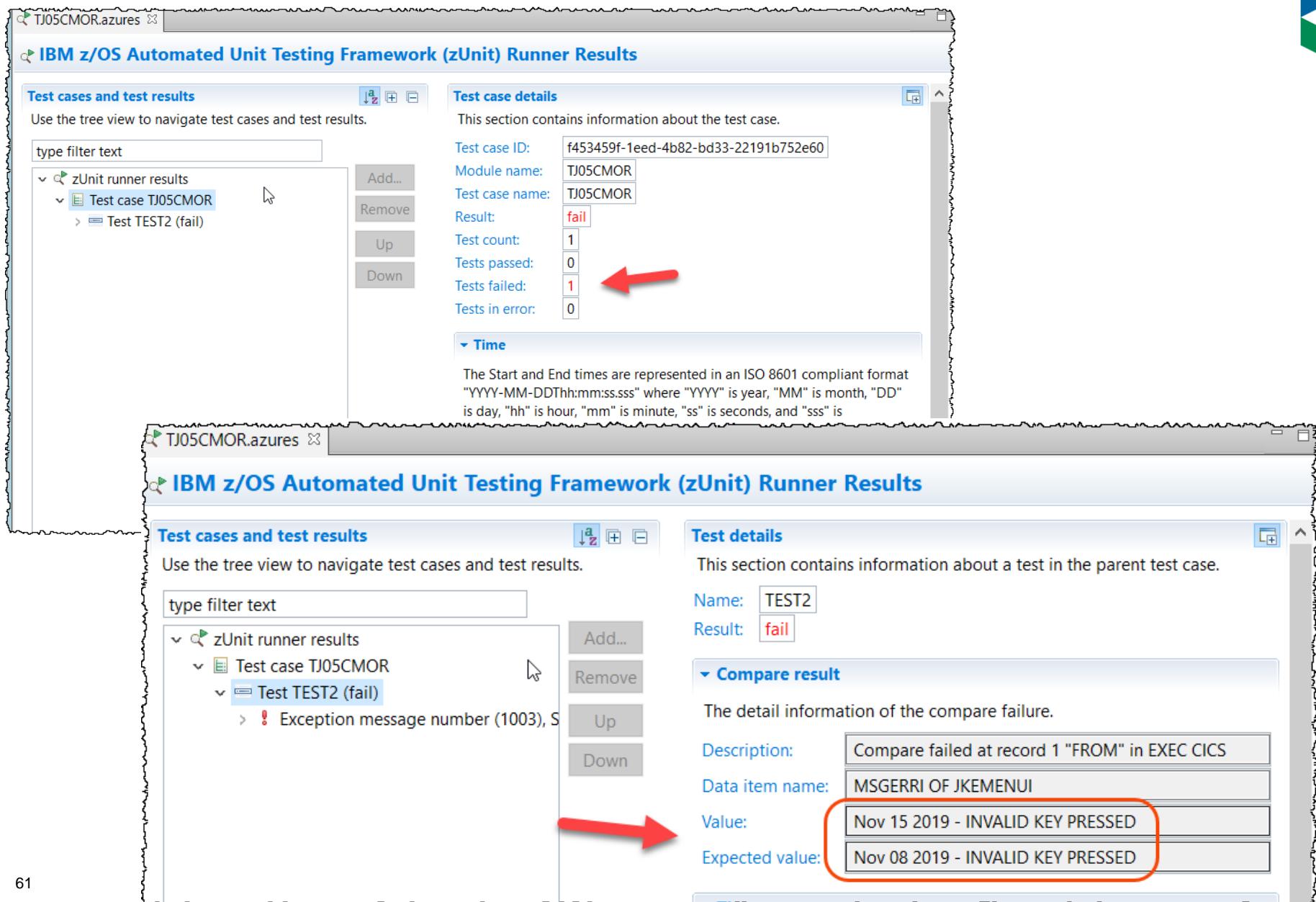
**Bottom Editor:** Shows a JCL source file named `J05CMORT.jcl`. It contains a job definition for `J05CMOCL` with various parameters like `MSGCLASS=H`, `TIME=1440`, and `REGION=0M`. A comment line, `// compile/link program J05CMORT and lik-edit the test case`, is highlighted with a red oval and a red arrow points to it from the left.

**File Browser:** On the left, a tree view shows the project structure. Under the `J05PAPP` sub-project, the `EMPOT05.DEMO.DEV.JCL(J05CMOCL.jcl)` file is selected and highlighted with a red oval.

**Context Menu:** A context menu is open over the JCL code, with the following options visible:

- z/OS Automated Unit Testing Framework (zUnit)
  - Generate Test Case...
  - Build Test Case
  - Run Test Case...** (highlighted with a blue oval and a blue arrow points to it from the left)
  - Debug Test Case...
  - Code Coverage...
  - Open Runner Configuration...
  - Clean Up Generated Files

### 3. Modify the program and rerun the unit test



The screenshot shows two instances of the IBM z/OS Automated Unit Testing Framework (zUnit) Runner Results interface. Both windows have the title "IBM z/OS Automated Unit Testing Framework (zUnit) Runner Results".

**Top Window (Test case details):**

- Test cases and test results:** A tree view showing "zUnit runner results" expanded, with "Test case TJ05CMOR" selected. Under it, "Test TEST2 (fail)" is listed.
- Test case details:** Information about the selected test case:
  - Test case ID: f453459f-1eed-4b82-bd33-22191b752e60
  - Module name: TJ05CMOR
  - Test case name: TJ05CMOR
  - Result: fail
  - Test count: 1
  - Tests passed: 0
  - Tests failed: 1
  - Tests in error: 0
- Time:** A section explaining ISO 8601 time format.

A red arrow points to the "Tests failed: 1" field.

**Bottom Window (Test details):**

- Test cases and test results:** Similar tree view showing "zUnit runner results" expanded, with "Test case TJ05CMOR" selected. Under it, "Test TEST2 (fail)" is listed, which has an exclamation mark icon next to it.
- Test details:** Information about the test in the parent test case:
  - Name: TEST2
  - Result: fail
- Compare result:** Details of the compare failure:
  - Description: Compare failed at record 1 "FROM" in EXEC CICS
  - Data item name: MSGERR1 OF JKEMENU1
  - Value: Nov 15 2019 - INVALID KEY PRESSED
  - Expected value: Nov 08 2019 - INVALID KEY PRESSED

A red arrow points to the "Value" field.

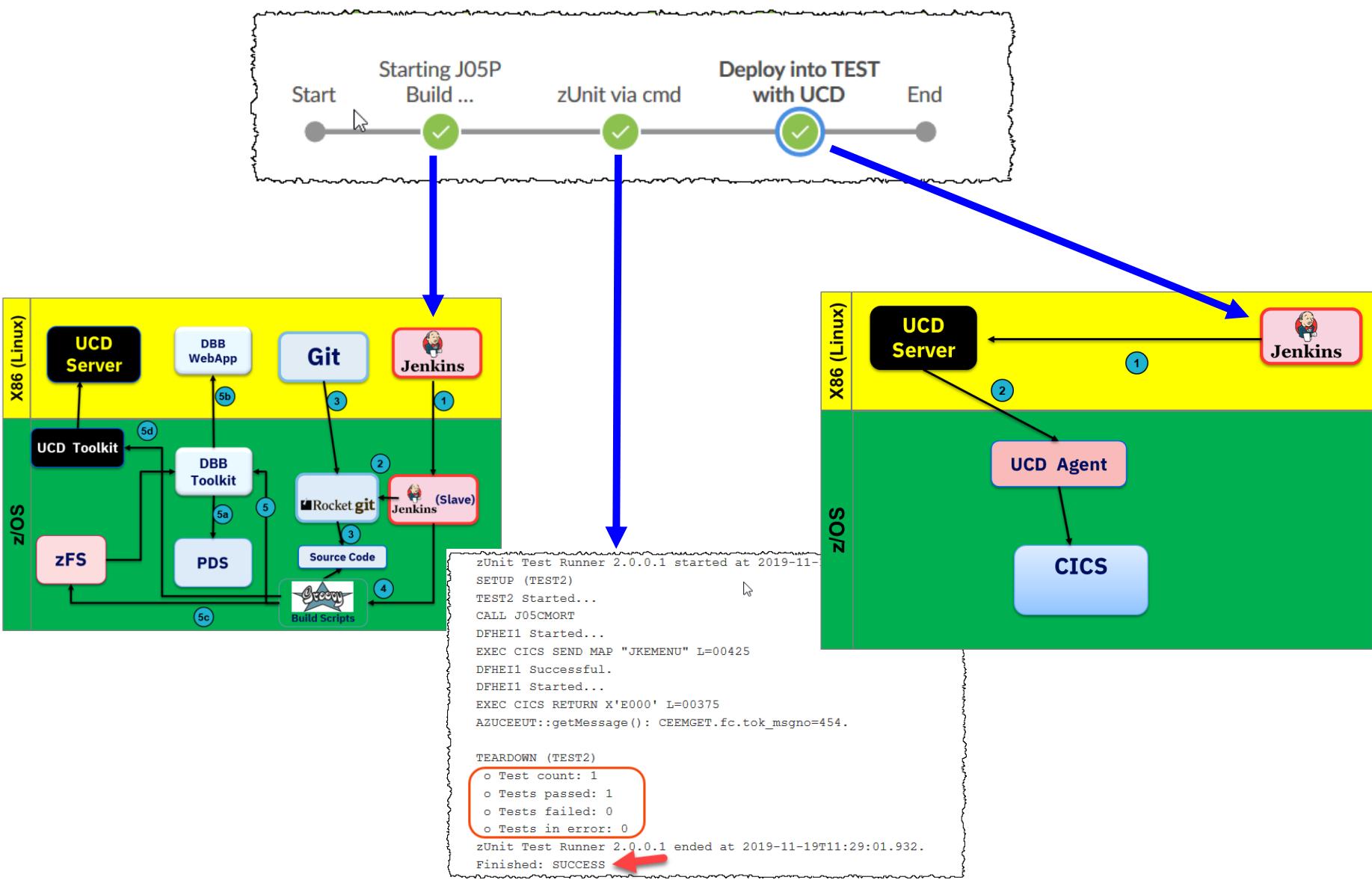
# Unit test on COBOL/CICS program J05CMORT

1. Record CICS program interaction using **IDz** .
2. Generate, build and run the unit test

Compile and link-edit the generated unit test program, followed by running the unit test.
3. Modify the program and rerun the unit test

Modify the program under test, rerun the unit test, and observe the failure of the test case.
4. **→** Uses Jenkins to perform the Build, ZUnit and Deploy
  - Developer push changes to Git
  - The Z unit test will run from a Z/OS shell script and deploy only if it does not fail.

# High Level Diagram using Jenkins/Git/Zunit/UCD



# Agenda

9:00 Introductions

9:20 DevOps on z Systems Introduction

9:45 Application Delivery Foundation (ADFz):

- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break

10:30 LAB 1 - Working with z/OS using COBOL and DB2

or

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

11:45 Lunch

12:30 z Open Unit Test: Overview & Demonstration

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

2:00 – 4:30 Choose one Optional lab:

→ LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)

→ LAB 2C Application Discovery : Find a candidate API from Catalog Manager Application

→ LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App

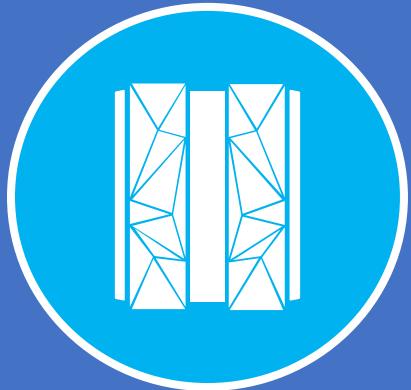
→ LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS

→ LAB 8 - Using Application Performance Analyzer (APA)

■ Lecture

■ Labs

■ Breaks



DevOps Mainframe Tooling

# Demonstration

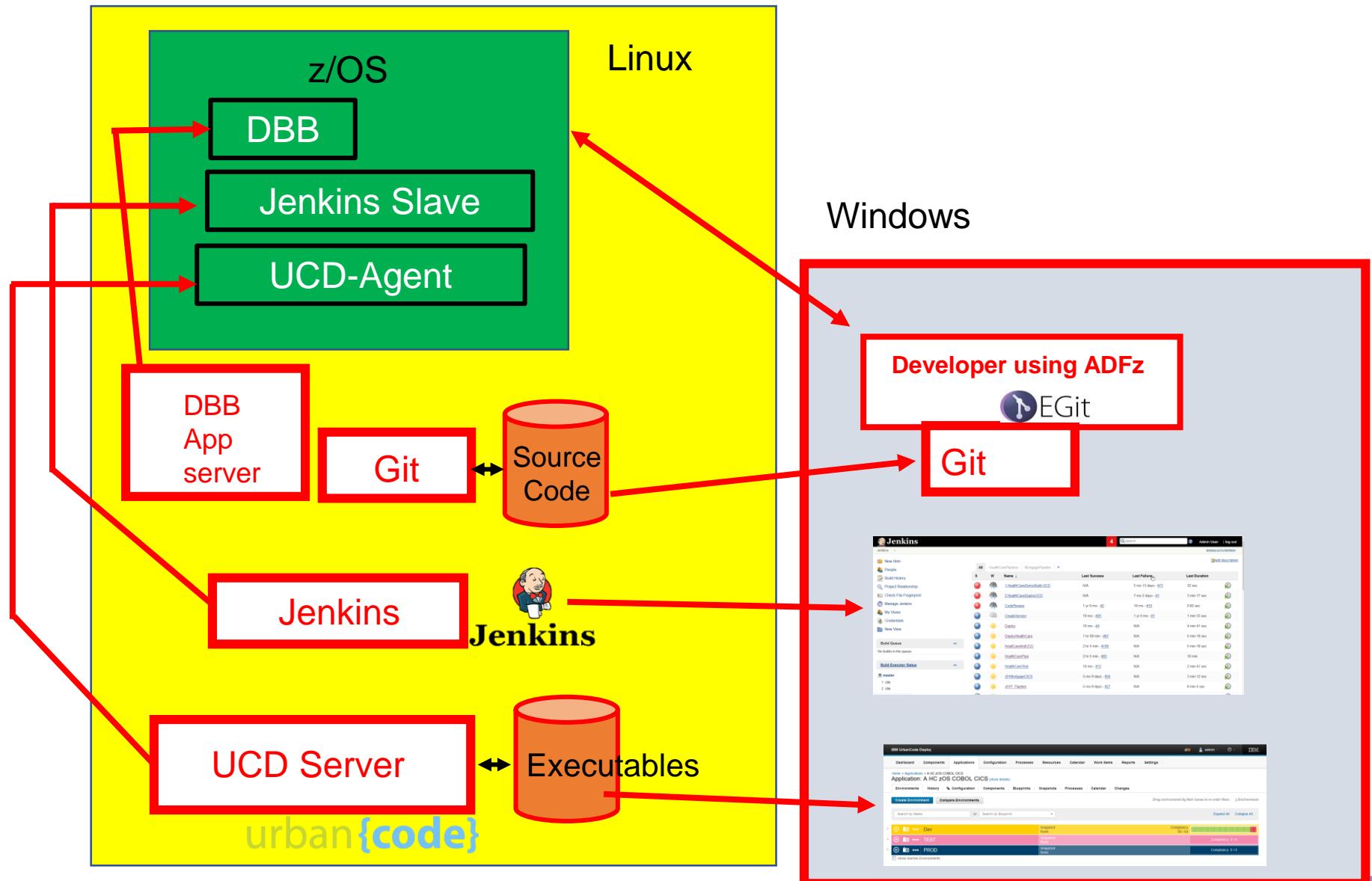
zD&T: AD - ADFz – DBB – UCD

Using Git and Jenkins

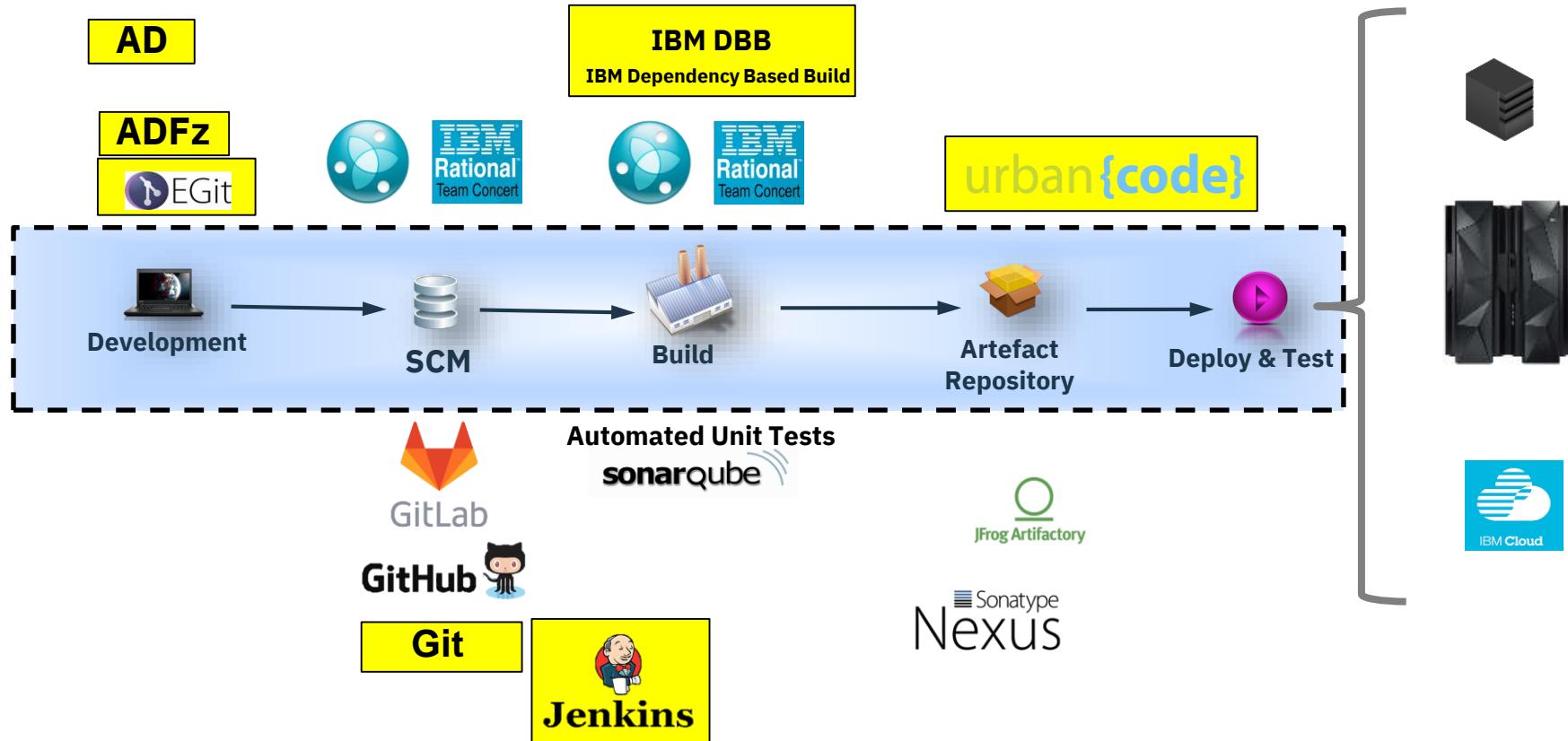
Demonstration

# zD&T Topology for ADFz/Git/Jenkins/UCD

## zD&T on VMWARE



# Open Pipe Line across the Enterprise



AD = Application Discovery

ADFz = Application Delivery Foundation for z Systems

IBM DBB = IBM Dependency Based Build

urban{code} = IBM UrbanCode Deploy (UCD)

# Agenda

1

- What is  
*IBM Dependency Based Build (DBB)*?

2

- General architecture of the solution

4

- Git/DBB/Jenkins/UCD in action (demo)

# What is IBM Dependency Based Build?

- **IBM Dependency Based Build (DBB)** is a tool to build traditional z/OS applications such as COBOL and PL/I as part of a continuous integration pipeline.
- Provides a modern scripting language based automation capability that can be used on z/OS.
  - The DBB API is written in Java and can be called by Java applications as well as Java based scripting languages such as **Groovy**, **JRuby**, **Jython**, **Ant**, **Maven**, etc.
- **Not tied to a specific source code manager or pipeline automation tool.** (We will use *Git* and *Jenkins* on the demo)
- Consists of a **build toolkit** (Java API, Groovy Installation) installed on **USS (z/OS)** and a **Liberty application server** that hosts build metadata (dependency data, build results) installed on **Linux**.



# DBB Sample Groovy compile step versus JCL

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

println("Copying source from zFS to PDS . . .")
def copy = new CopyToPDS()
    .file(new File("/u/usr1/build/helloworld.cbl"))
    .dataset("USR1.BUILD.COBOL").member("HELLO")
copy.execute()

println("Compiling . . .")
def compile = new MVSExec().pgm("IGYCRCTL").parm("LIB")
compile.dd(new DDStatement().name("TASKLIB")
    .dsn("IGY.SIGYCOMP").options("shr"))
compile.dd(new DDStatement().name("SYSIN")
    .dsn("USR1.BUILD.COBOL(HELLO)").options("shr"))
compile.dd(new DDStatement().name("SYSLIN")
    .dsn("USR1.BUILD.OBJ(HELLO)").options("shr"))
compile.dd(new DDStatement().name("SYSPRINT")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT1")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT2")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT3")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT4")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT5")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT6")
    .options("tracks space(5,5) unit(vio) new"))
compile.dd(new DDStatement().name("SYSUT7")
    .options("tracks space(5,5) unit(vio) new"))
compile.copy(new CopyToHFS().ddName("SYSPRINT")
    .file(new File("/u/usr1/build/helloworld.log")))
def rc = compile.execute()

if (rc > 4)
    println("Compile failed!  RC=$rc")
else
    println("Compile successful!  RC=$rc")
```



```
//COBOL EXEC PGM=IGYCRCTL,REGION=0M,
//          PARM='LIB'
///*
//STEPLIB  DD DISP=SHR,DSN=IGY.SIGYCOMP
///*
//SYSIN    DD DISP=SHR,DSN=USER1.BUILD.COBOL(HELLO)
//SYSLIN   DD DISP=SHR,DSN=USER1.BUILD.OBJ(HELLO)
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT2   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT3   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT4   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT5   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT6   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
//SYSUT7   DD
DISP=(NEW,PASS),SPACE=(TRK,(5,5)),UNIT=VIO,
//          BLKSIZE=80,LRECL=80,RECFM=FB
```



# Agenda

1

- What is  
*IBM Dependency Based Build (DBB)*?

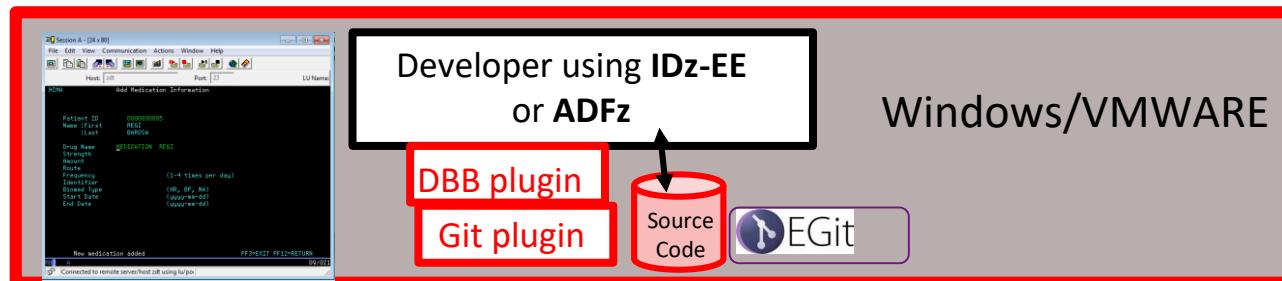
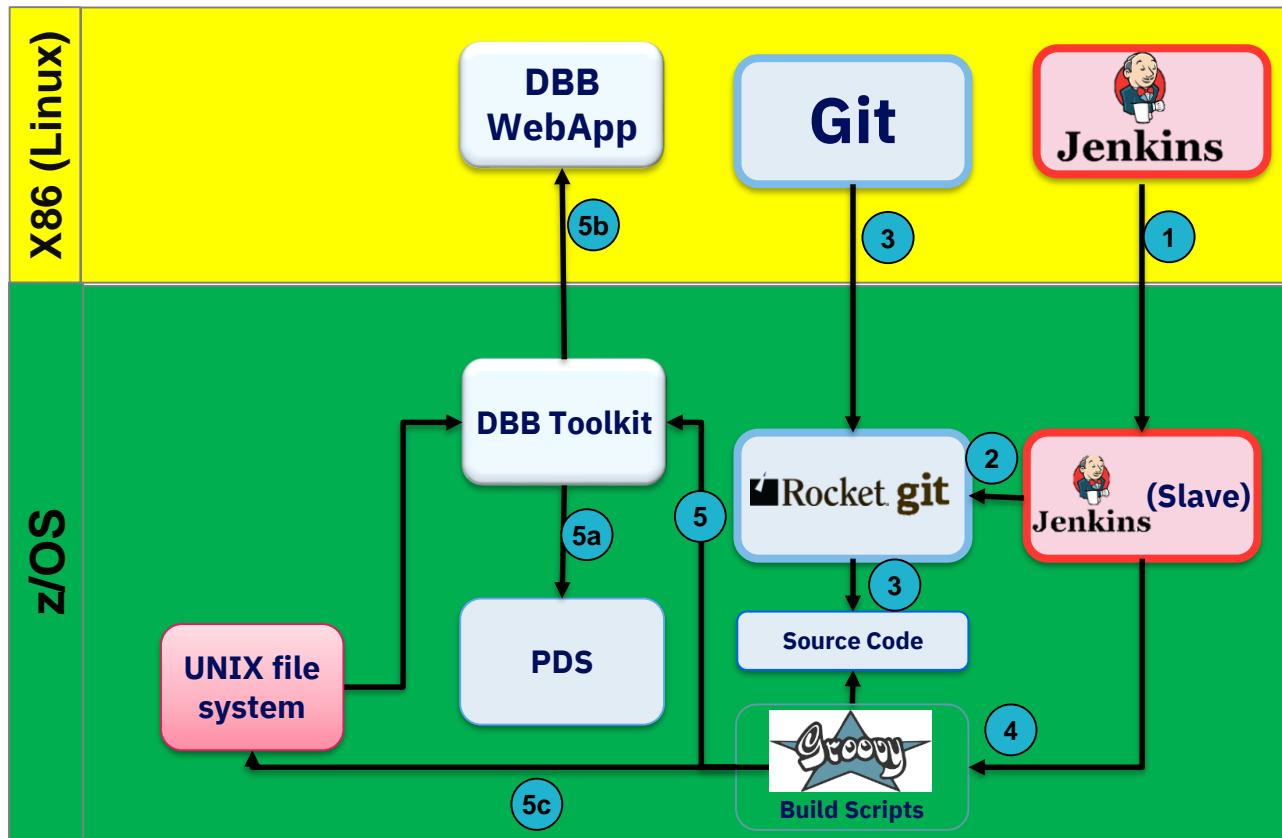
2

- General architecture of the solution

4

- Git/DBB/Jenkins/UCD in action (demo)

# Solution High Level Diagram



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

[2] Jenkins agent issues Git pull command to update local Git repository.

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

[4] Jenkins agent invokes build scripts containing DBB APIs in local zOS Git repository.

[5] DBB Toolkit provides Java APIs to:

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

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

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

# Agenda

1

- What is  
*IBM Dependency Based Build (DBB)*?

2

- General architecture of the solution

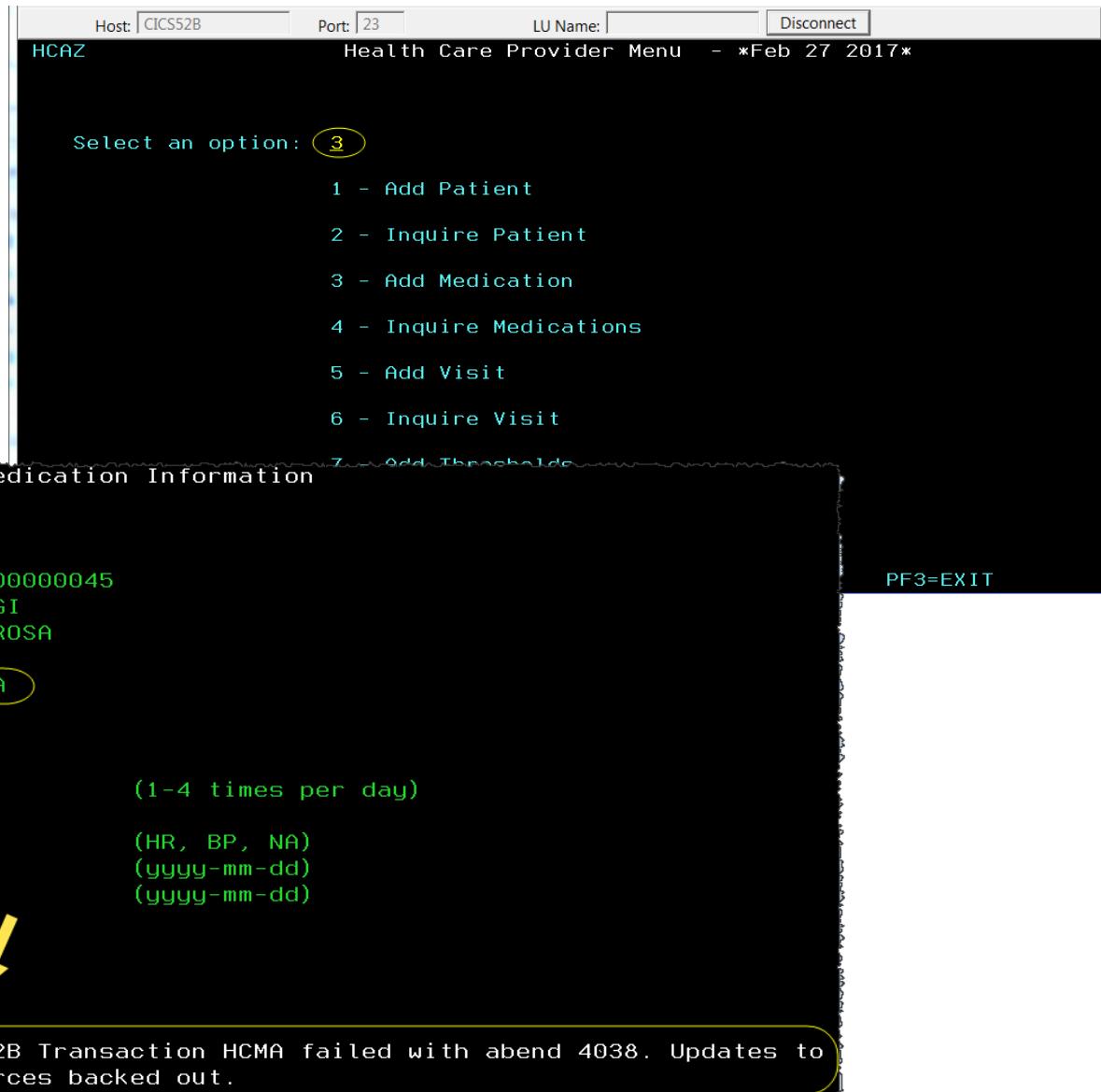
4

- Git/DBB/Jenkins/UCD in action (demo)

# Scenario: CICS Transaction ABEND

*Using transaction HCAZ  
and option 3*

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



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

### Objective: Fix the CICS Abend

Alex - Project Manager

Performs application analysis

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

# Alex starts CICS transaction HCAZ to verify the abend

====> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or  
====> Enter L followed by the APPLID  
====> Examples: "L TSO", "L CICSTS52", "L CICSTS53", "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  
Name :First :Last

Patient ID 0000000005  
Name :First Regi :Last Barosa  
Drug Name DRUG A  
Strength  
Amount  
Route  
Frequency  
Identifier  
Biomed Type  
Start Date  
End Date

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

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

### Objective: Fix the CICS Abend

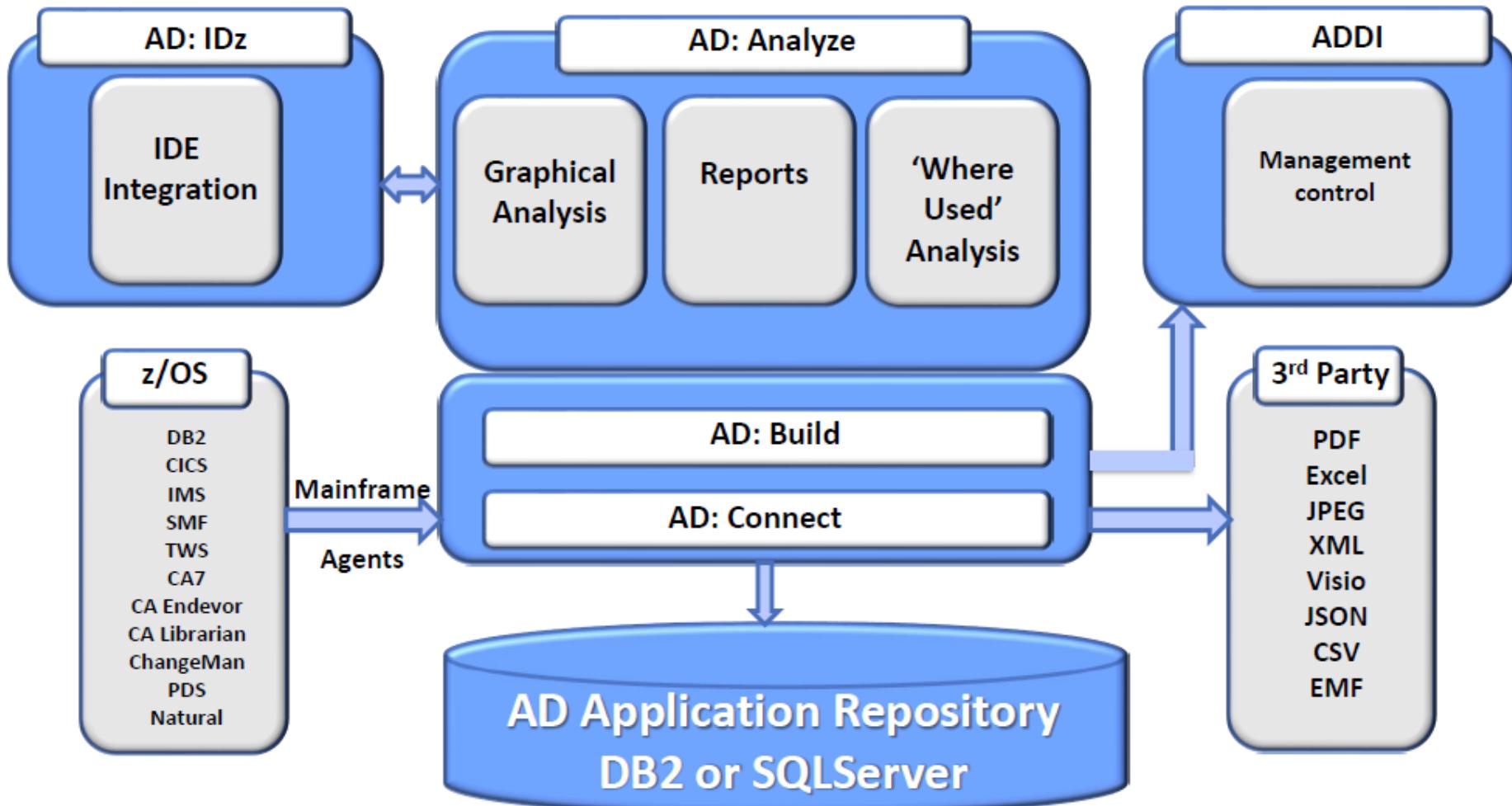
Alex - Project Manager

Performs application analysis

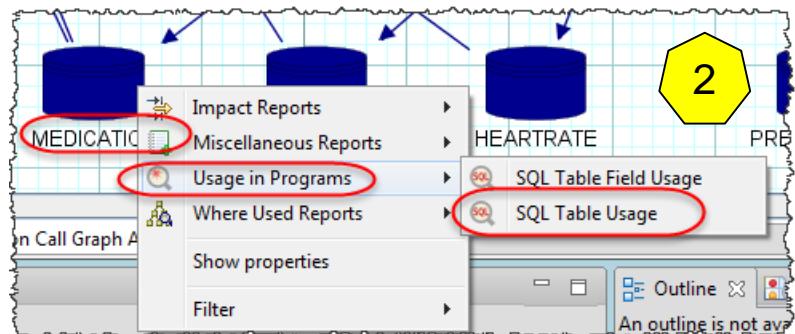
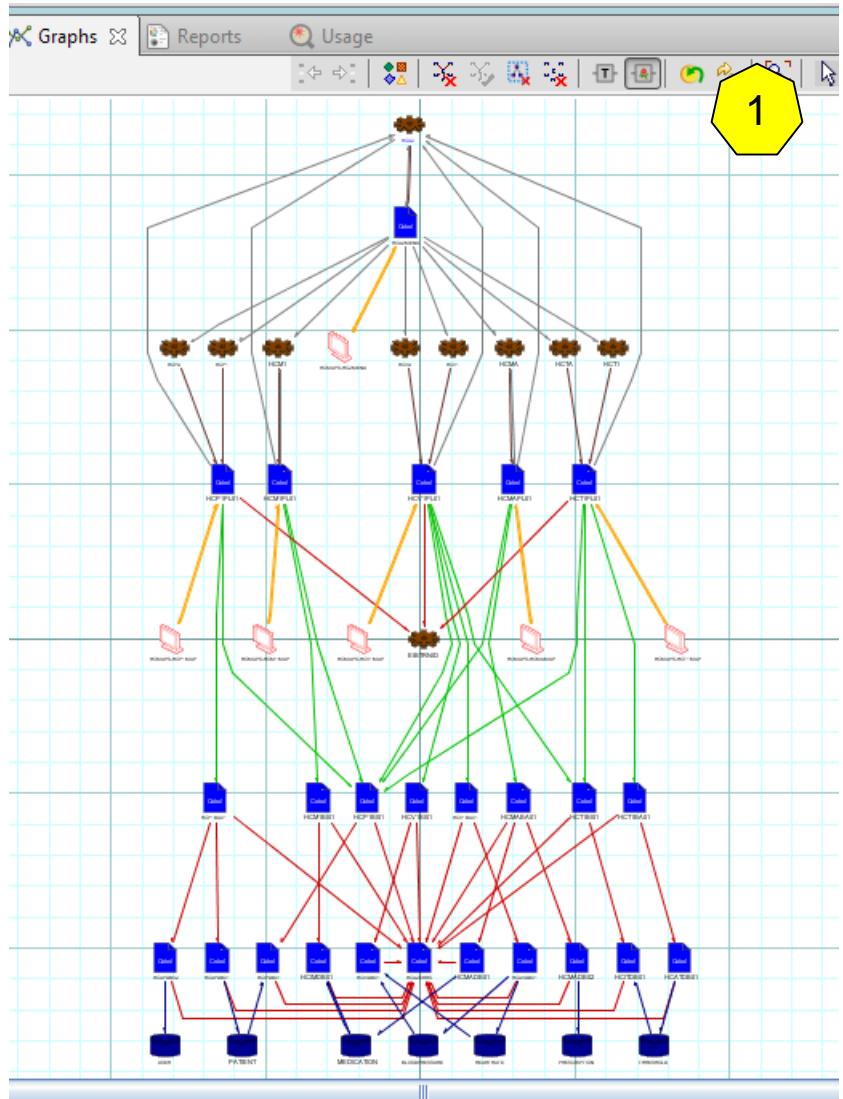
1. Executes CICS transaction **HCAZ** to verify the abend.
2. Uses **Application Discovery (AD)** to understand the application.



# IBM Application Discovery – Big Picture



# Uses Application Discovery (AD) to understand the application



A screenshot of the Application Discovery interface. On the left, a 'Results' tree view shows a hierarchy: MEDICATION > COBOL > HCIMDB01 > various SQL statements like CLOSE, FETCH, OPEN, SELECT, and INSERT. A yellow hexagon labeled '3' is in the top right corner. On the right, a code editor window titled 'HCMADB01.cbl' displays Cobol code. The code includes a section for inserting data into the Medication table based on patient ID, moving data to an SQL request, and executing an SQL insert statement.

```
-----+-----+-----+-----+-----+
156
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
-----+-----+-----+-----+-----+
      MOVE ' INSERT MEDICATION' TO EM-SQLREQ
      EXEC SQL
      INSERT INTO MEDICATION
      ( MEDICATIONID,
        PATIENTID,
        DRUGNAME,
        STRENGTH,
        AMOUNT,
        ROUTE,
        FREQUENCY,
        IDENTIFIER,
        TYPE )
      VALUES ( DEFAULT,
              :DB2-PATIENT-ID,
              :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 **ADFz/Fault Analyzer** to identify what is causing the abend
3. Uses **ADFz/Debug** to debug the program and verify the error
4. Uses **ADFz** to update the COBOL program loaded from **GIT**
5. Uses **DBB** to perform a User Dependency Build and verify that the abend is fixed
6. Uses **GIT** to **Commit and Push** the corrected code to Repository

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

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

```
1①
2② 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 000301      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

cbl 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.

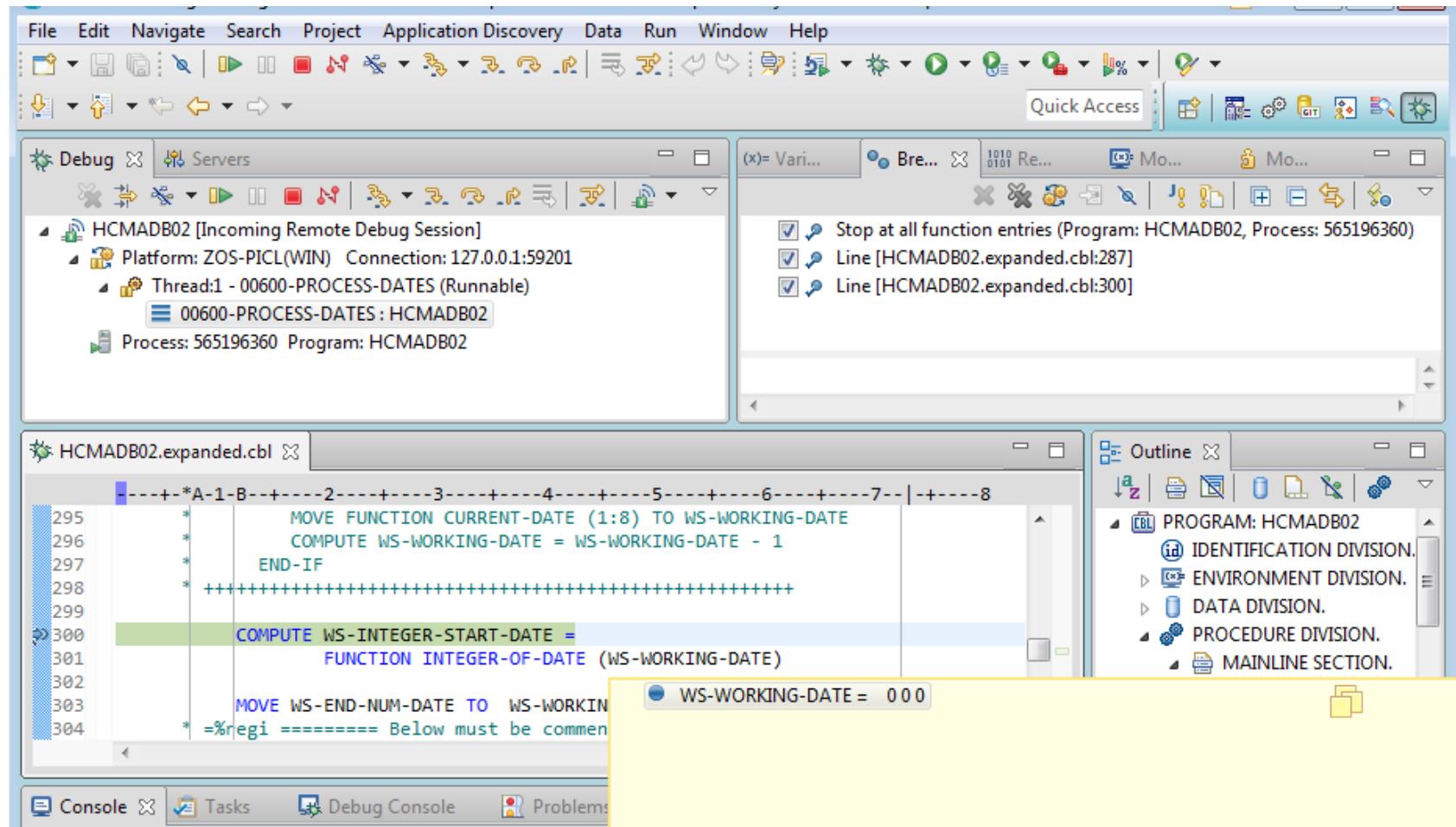
### Bob – Mainframe Developer

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

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



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

### Bob – Mainframe Developer

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

#### Objective: Fix the CICS Abend

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

# Uses ADFz to update the COBOL program loaded from GIT

The screenshot shows the ADFz integrated development environment. On the left, a code editor displays a COBOL program named HCMADB02.cbl. The code includes several comments and logic for handling dates. In the center, a 'Remote Systems' browser lists various connection nodes, including Local, zos.dev, zdt, zserveros.centers.ihost.com, demomvs.cc9.pok.ibm.com, 10.149.60.141, 192.168.1.106, and zdtv8. The right side of the interface shows expanded details for the zdtv8 connection, listing z/OS UNIX Files and MVS Files. At the bottom, a navigation bar shows the current application stack: HCMADB02, MAINLINE, 00100-INITIAL-PARAGRAPH, 00200-SETUP, 00300-INITIALIZE-DB2, 00400-CHECK-COMMAREA, 00500-COMMAREA-INIT, and 00.

```
--+--A-1-B---2---+---3---+---4---+---5---+---6---+---7---+---8
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      * END-IF
171      *+++++
172
173      COMPUTE WS-INTEGER-START-DATE =
174          FUNCTION INTEGER-OF-DATE (WS-WORKING-DATE)
175
176      MOVE WS-END-NUM-DATE TO WS-WORKING-DATE
177      *=%regi ===== Below must be comments to have the abend
178      *=added to fix abend #2 END DATE
179      * IF WS-WORKING-DATE < 16010101 or
180      *     WS-WORKING-DATE > 99991231
181      *     MOVE FUNCTION CURRENT-DATE (1:8) TO WS-WORKING-DATE
182      * END-IF
183      *+++++
184
185      COMPUTE WS-INTEGER-END-DATE =
186          FUNCTION INTEGER-OF-DATE (WS-WORKING-DATE)
187
188      PERFORM UNTIL WS-INTEGER-START-DATE > WS-INTEGER-END-DATE
189          COMPUTE WS-WORKING-DATE -
```

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

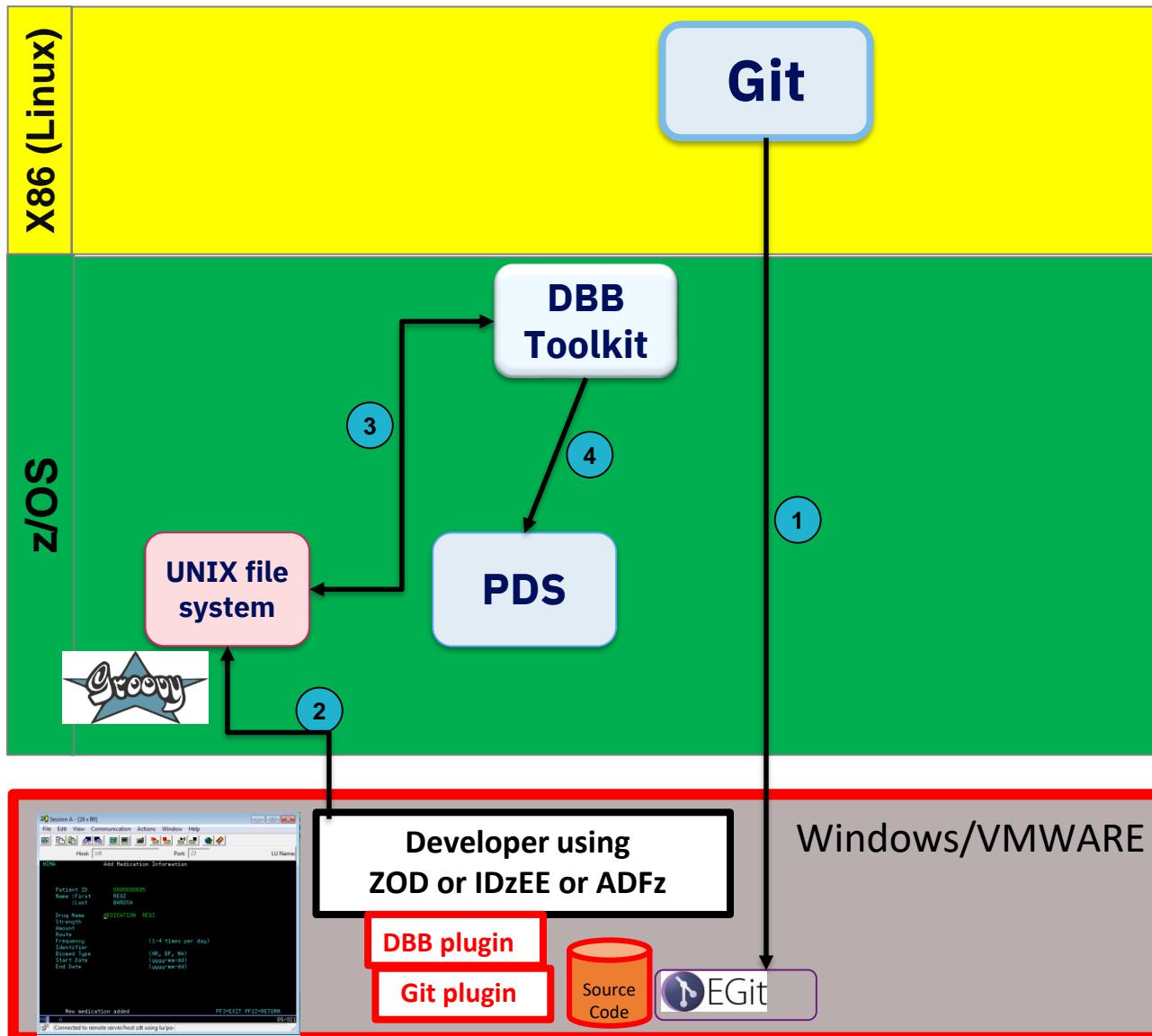
### Bob – Mainframe Developer

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

# Using DBB User Build (Personal test)



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

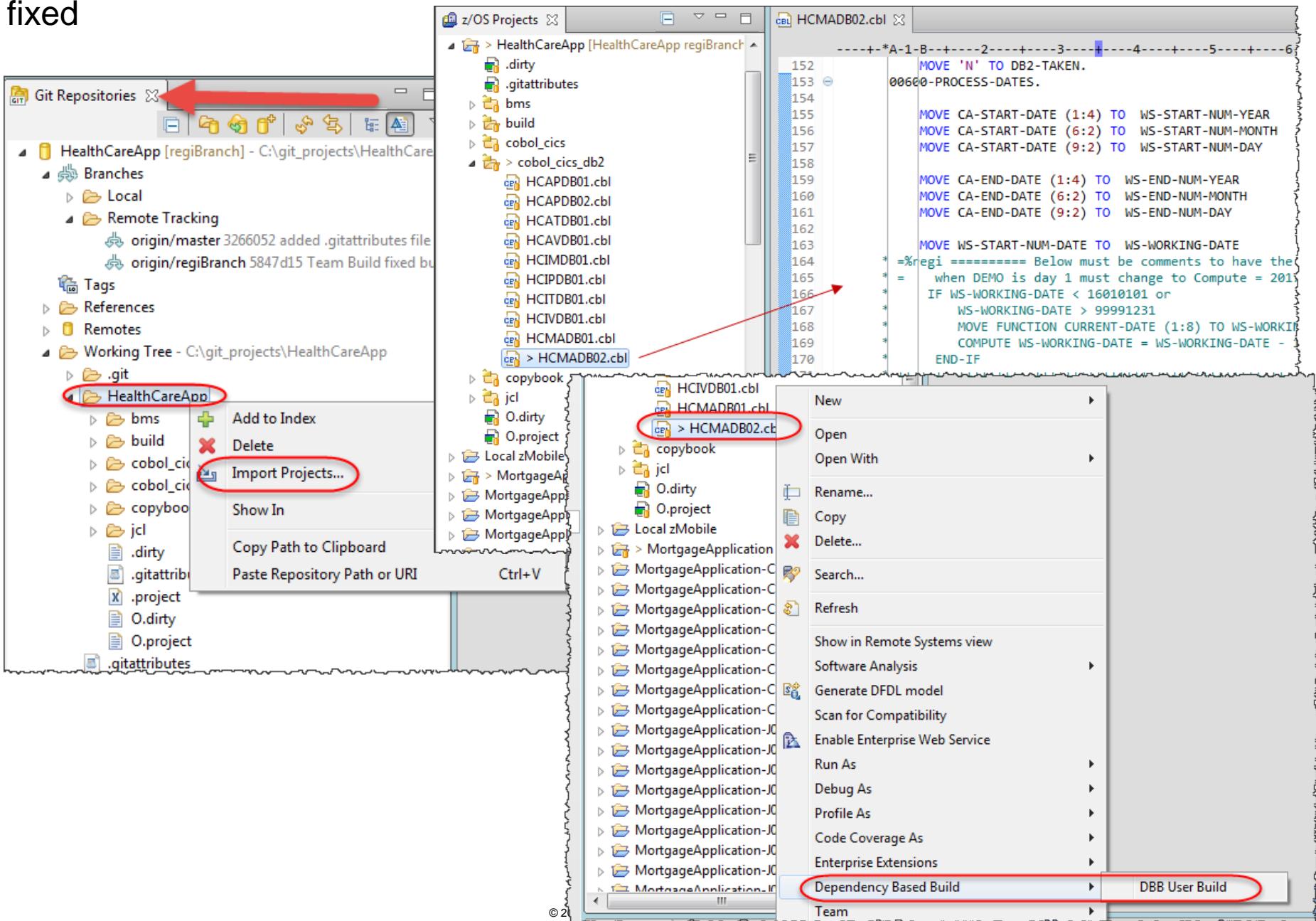
[2] **DBB User Build** moves COBOL source/copybooks from local IDz workspace to **zFS** files, invokes Groovy build scripts containing DBB APIs.

[3] DBB Toolkit provides Java APIs to:

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

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

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



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

## DBB User Build

### Configure User Build operation

Specify information for User Build operation.

Select the z/OS system to use:

zdtv8

Select the build script to use:

\HealthCareApp\build\build.groovy

Enter the build sandbox folder:

/var/dbb/work

Enter the build destination HLQ:

IBMUSER.GIT.ZMOBILE

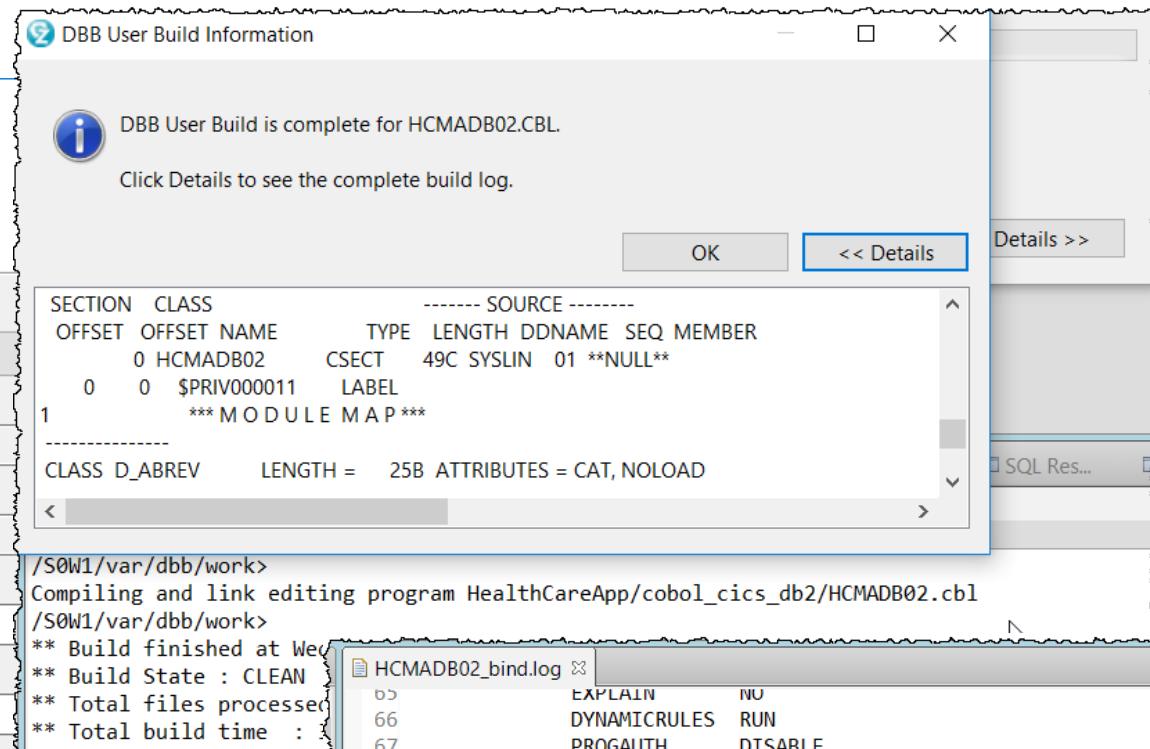
Enter the Team build HLQ:

[Preferences](#)

< Back

Next >

Finish



The screenshot shows the 'HCMADB02\_bind.log' window. It displays a series of bind log entries numbered from 55 to 86. The log entries show various parameters and their values, such as EXPLAIN, DYNAMICRULES, PROGAUTH, and DSNT253I. The log concludes with a successful bind command and an ISPF\_RETURN\_CODE of 0.

Line Number	Parameter	Value
55	EXPLAIN	NO
66	DYNAMICRULES	RUN
67	PROGAUTH	DISABLE
68	DSNT253I	-DBBG DSNTBCM1 BIND OPTIONS FOR PLAN PLAN
69		NODEFER
70		CACHESIZE 3072
71		QUALIFIER HCZMSA1
72		CURRENTSERVER
73		CURRENTDATA NO
74		DEGREE 1
75		SQLRULES DB2
76		DISCONNECT EXPLICIT
77		REOPT NONE
78		KEEPDYNAMIC NO
79		IMMEDWRITE NO
80		DBPROTOCOL DRDA
81		OPTHINT
82		ENCODING UNICODE(01208)
83		CONCURRENTACCESSRESOLUTION
84		PATH
85	DSNT200I	-DBBG BIND FOR PLAN PLAN SUCCESSFUL
86		ISPF_RETURN_CODE = 0

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

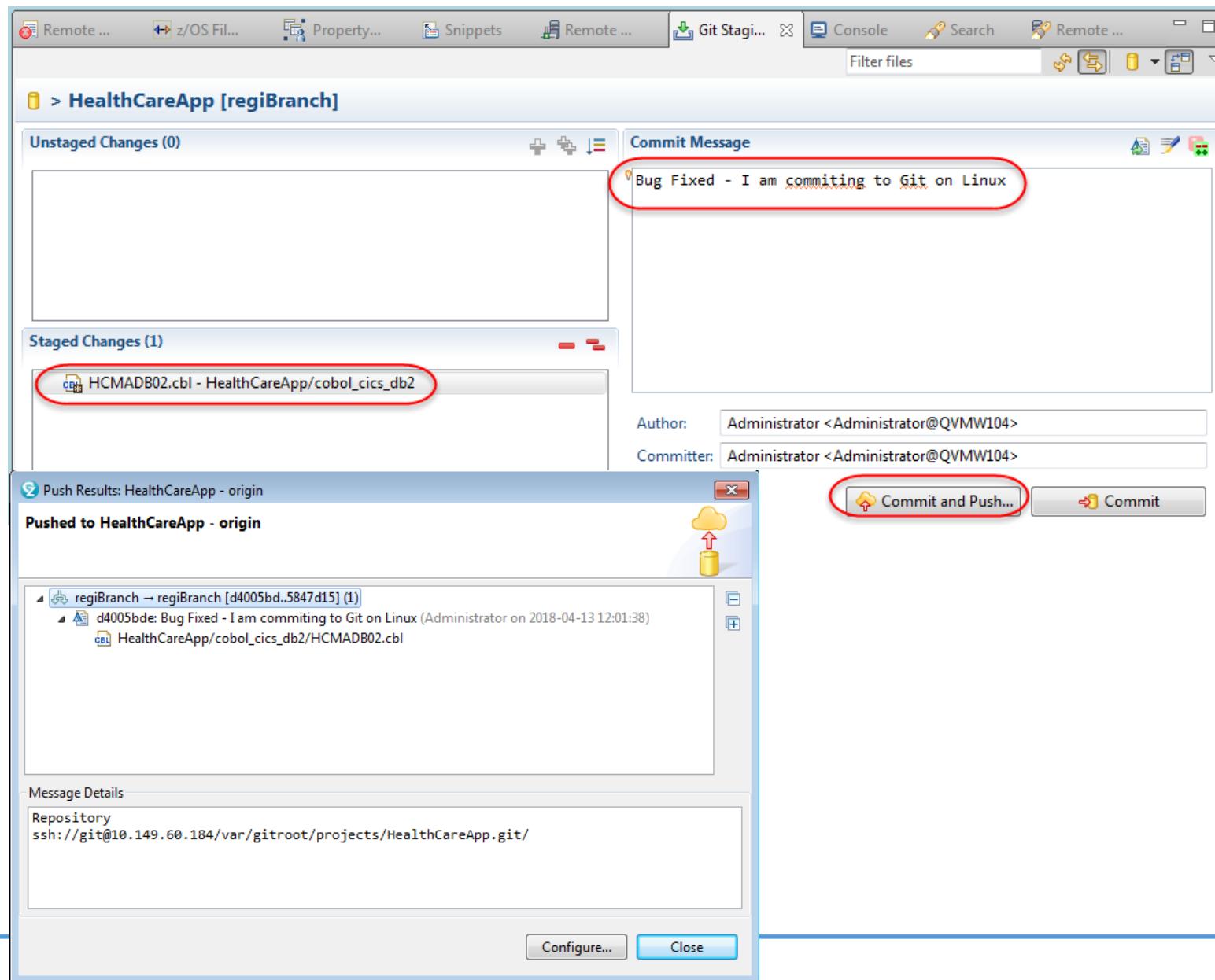
### Bob – Mainframe Developer

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

#### Objective: Fix the CICS Abend

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

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



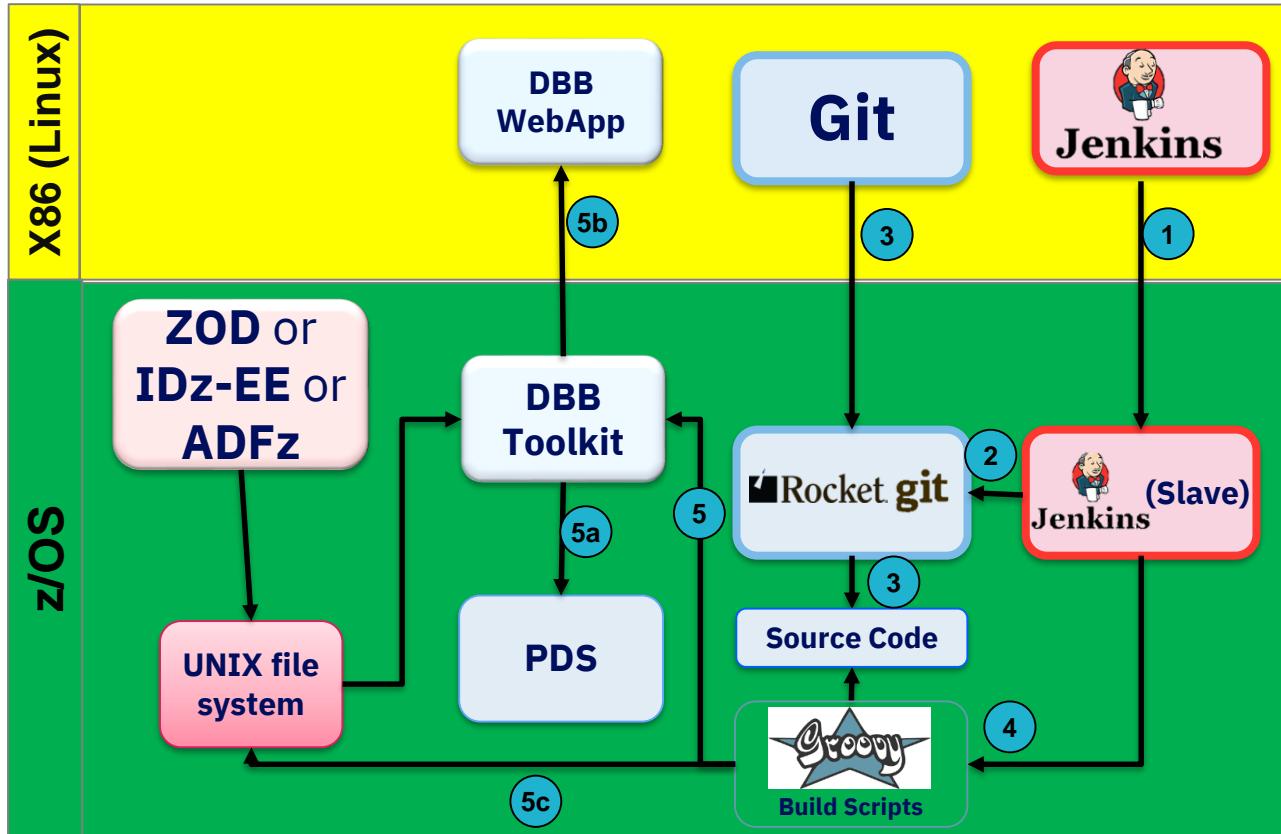
## PART #3 - Release Engineer Deploy to z/OS and Mobile.

### Rebecca – Release Engineer

Executes the Team Building using Jenkins

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

# Solution High Level Diagram



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

[2] Jenkins agent issues Git pull command to update local Git repository.

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

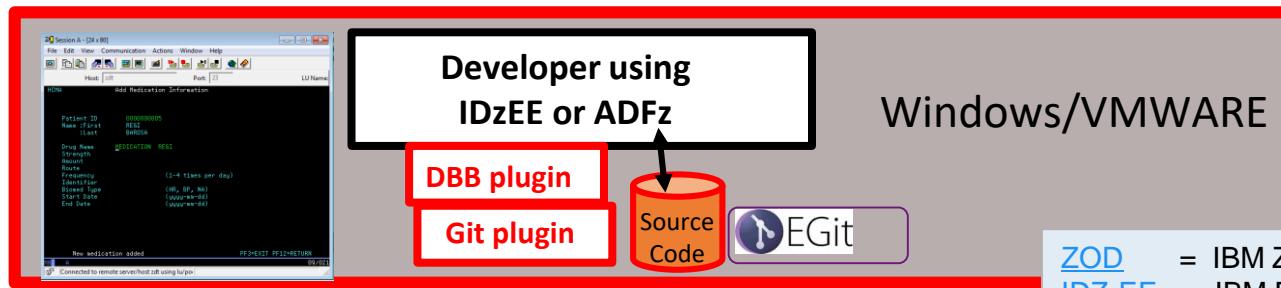
[4] Jenkins agent invokes build scripts containing DBB APIs in local Git repository.

[5] DBB Toolkit provides Java APIs to:

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

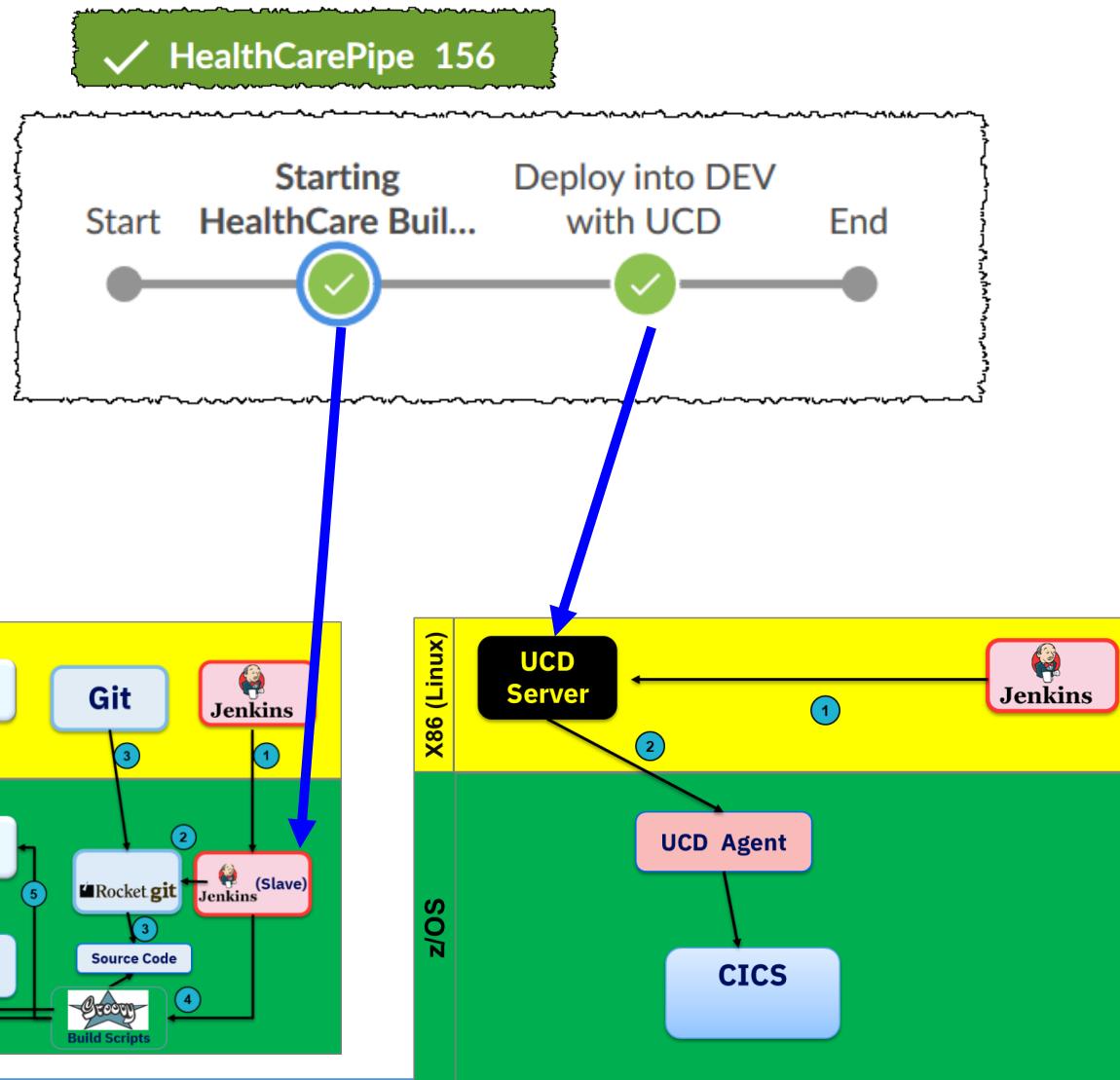
[5b] Scan and store dependency data from source files, perform dependency and impact analysis, store build results.

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

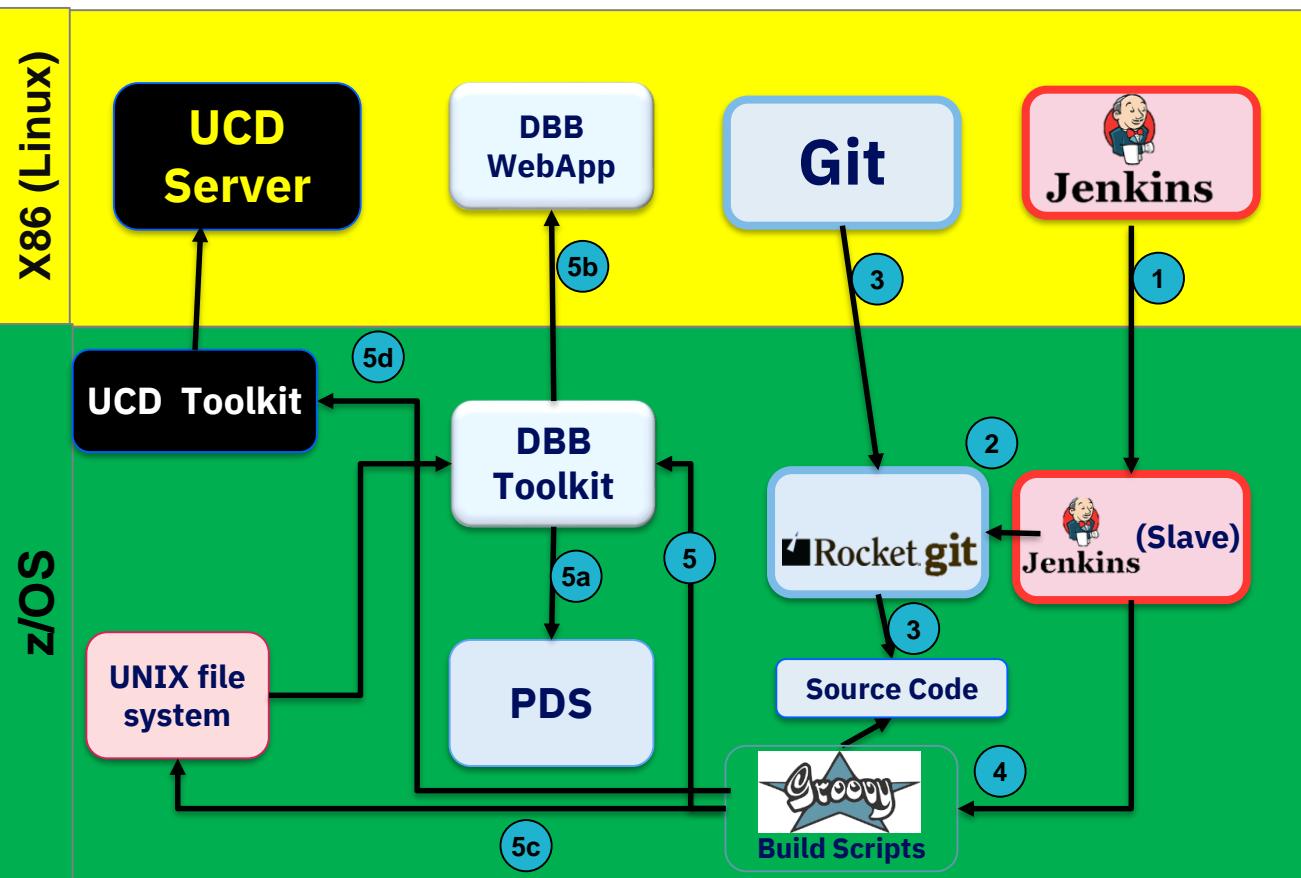


<u>ZOD</u>	= IBM Z Open Development
<u>IDz-EE</u>	= IBM Developer for z Systems Enterprise Edition
<u>ADFz</u>	= Application Delivery Foundation for z Systems
<u>DBB</u>	= IBM Dependency Based Build

# Demo High Level Diagram using Jenkins/Git/UCD



# High Level Demo Diagram with Jenkins and UCD (UrbanCode Deploy)



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

[2] Jenkins agent issues Git pull command to update local Git repository.

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

[4] Jenkins agent invokes build scripts containing DBB APIs in local Git repository.

[5] DBB Toolkit provides Java APIs to:

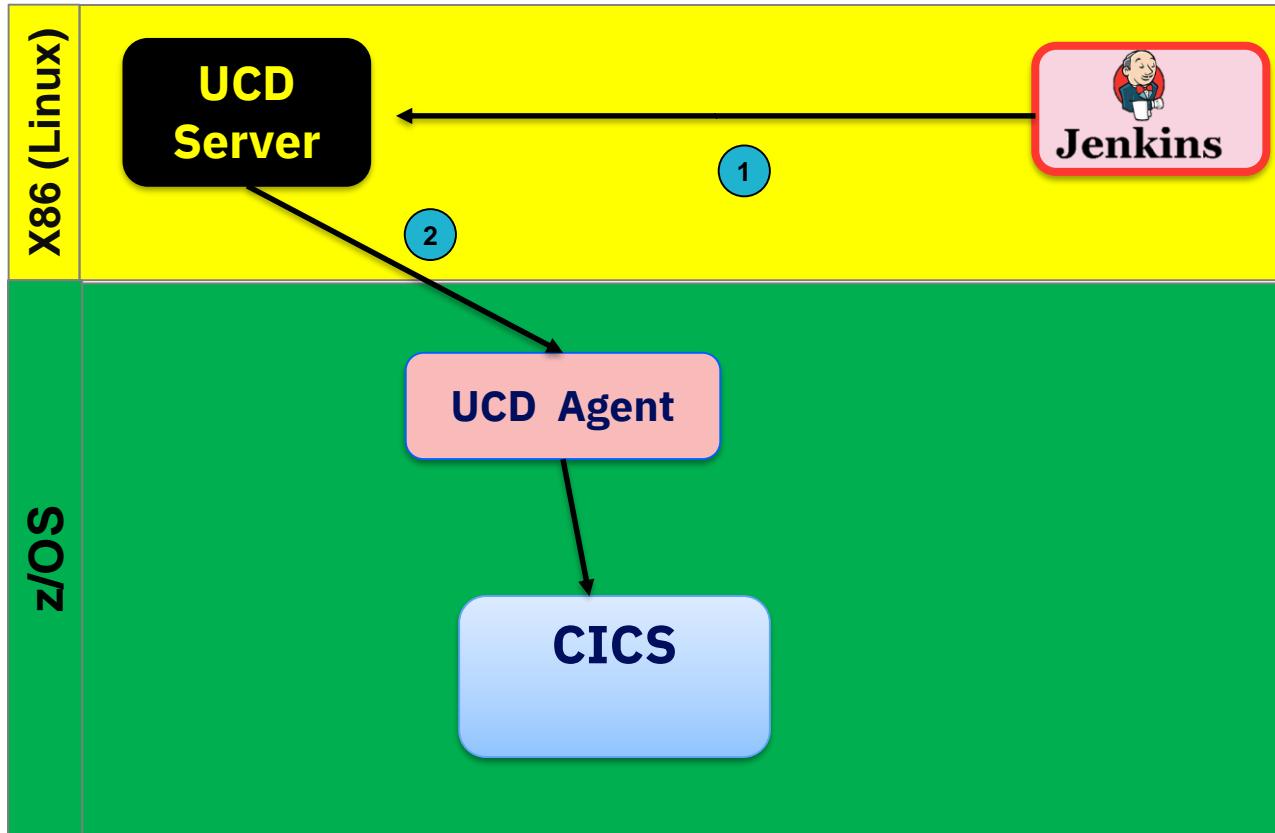
[5a] Create datasets, copy source from **zFS** to **PDS**, invoke zOS Compiler/Linkage Editor/ REXX for DB2 Bind.

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

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

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

# High Level Demo Diagram with Jenkins and UCD (2)



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

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

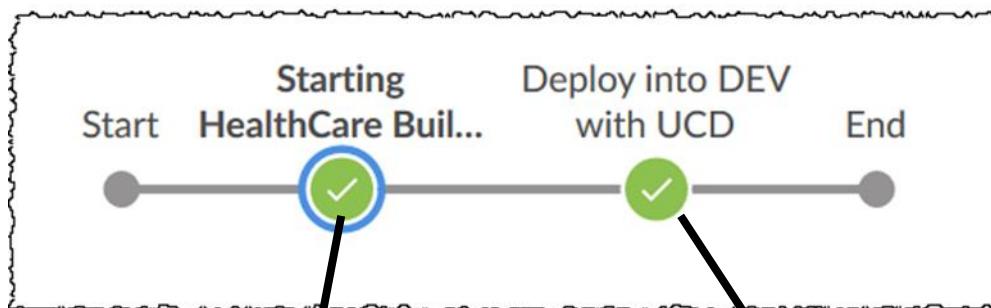
## PART #3 - Release Engineer Deploy to z/OS and Mobile.

### Rebecca – Release Engineer

Executes the Team Building using Jenkins



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



```
** Invoking build scripts according to build order: Compile, LinkEdit, CobolCompile, JCL
* Building HealthCareApp/cobol_cics_db2/HCMADB02.cbl using CobolCompile.groovy script
Copying /var/jenkins/workspace/HealtCareAndUCD/HealthCareApp/cobol_cics_db2/HCMADB02.cbl to
IBMUSER.GIT.TEAM.ZMOBILE.COBOLO(HCMADB02)
Resolving dependencies for file HealthCareApp/cobol_cics_db2/HCMADB02.cbl and copying to
IBMUSER.GIT.TEAM.ZMOBILE.COPYBOOK
Compiling and link editing program HealthCareApp/cobol_cics_db2/HCMADB02.cbl
** Build finished at Fri Apr 13 15:34:35 GMT 2018
** Build State : CLEAN
** Total files processed : 1
** Total build time: 1 minutes 17.527 seconds
Create version and store package:
....Version artifacts stored to UCD server CodeStation
....Version:20180413-153539 created
Elapsed time 59.0 seconds.

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

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

# What is Git?



- **Git** is a free and open source distributed version control system.
  - Version Control System is a system that records changes to a file or set of files over time so that you can recall specific versions later
  - Distributed means that there is no main server and all of the full history of the project is available once you cloned the project.
- **GitHub**, **GitLab** and **Bitbucket** are code collaboration and version control tools offering repository management based on Git.
  - They each have their share of fans, though **GitHub** is by far the most-used of the three.
  - Of the three, only **GitLab** is open source, though all three support open source projects.



# Why Git?

- Distributed
  - Full local repo, with history
  - Work offline
- Lightweight branching, users collaborate better and spend less time managing version control and more time developing code.
- De-facto standard



# Git vs. traditional SCM

- True parallel development
  - Branches on demand
  - Freedom from pre-defined streams
- Continuous Improvement to build processes
  - Use generic artifact repositories
  - Easy integration with CI/CD Pipelines (Jenkins)



# What is Jenkins?

- Jenkins is an automation engine which supports a number of automation patterns
- Jenkins can invoke scripts for builds and tests continuously and monitors the execution and status of remote executions.



# Why Jenkins

- Used everywhere
- Continuous Integration leads to Continuous Deployment allowing us to deliver software more rapidly.



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

### Rebecca – Release Engineer

Executes the Team Building using Jenkins

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

# Verify the Build results using DBB browser

Collections	Build Results
<h2>DBB Build Result</h2>	
Field	Value
id	427
group	HealthCareApp
label	build.20180413.033317.033
Build Report	<a href="#">view</a>
buildReportData	<a href="#">view</a>
state	2 (COMPLETE)
status	0 (CLEAN)

Toolkit Version:						
Version:	0.9.1	Build:	96	Date:	16-Feb-2018 21:49:21	
<b>Build Summary</b>						
Number of files being built: 1						
File	Commands	RC	Data Sets	Outputs	Logs	
1 HealthCareApp/cobol_cics_db2 /HCMADB02.cbl <a href="#">Show Dependencies</a>	IGYCRCTL IEWLINK exec 'IBMUSER.GIT.ZMOBILE.REXX(BINDHC2)'	4 0 0	IBMUSER.GIT.TEAM.ZMOBILE.COBOL(HCMADB02) IBMUSER.GIT.TEAM.ZMOBILE.DBRM(HCMADB02)	IBMUSER.GIT.TEAM.ZMOBILE.LOAD(HCMADB02)	HCMADB02.log HCMADB02_bind.log	

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

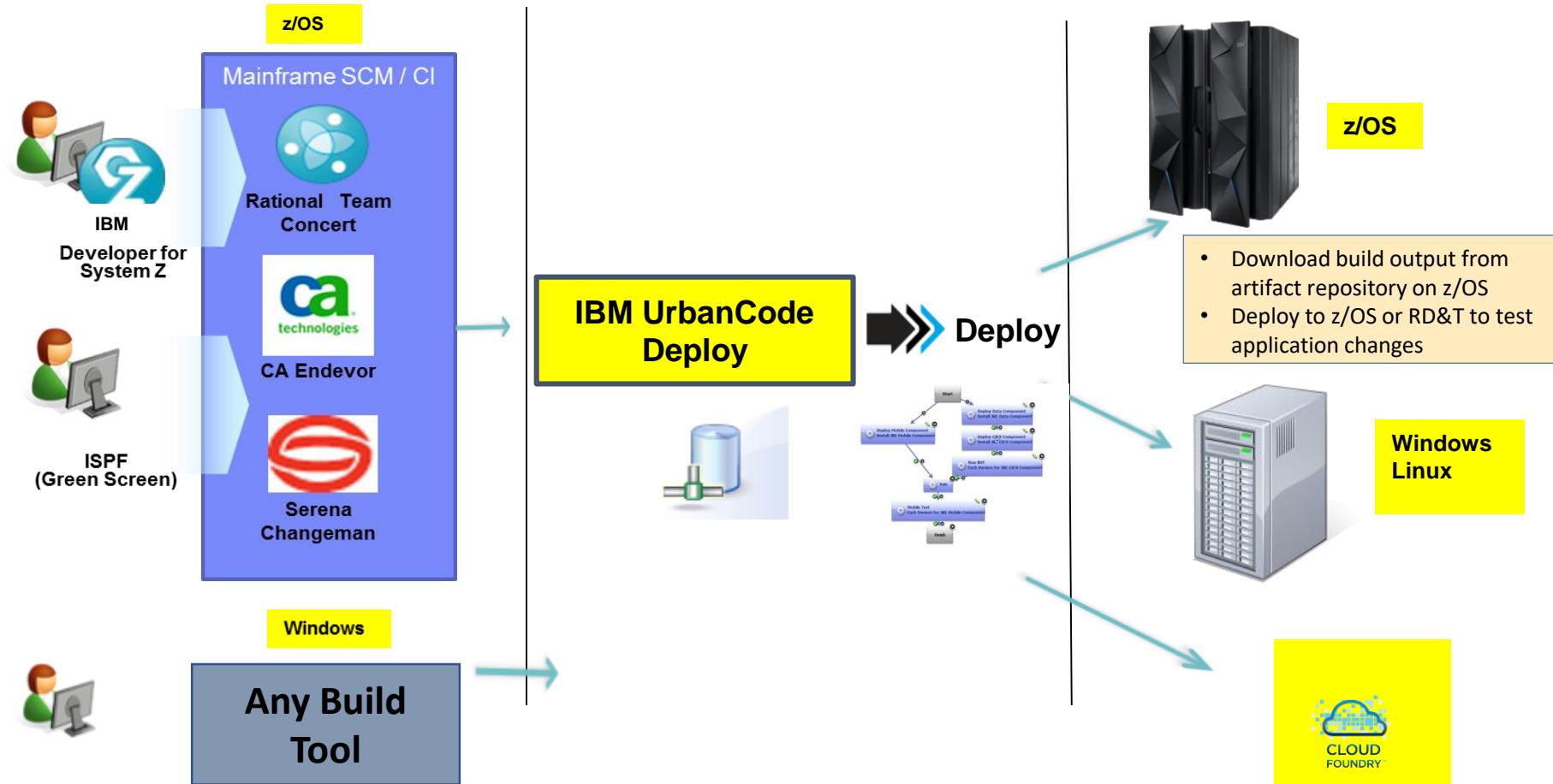
### Rebecca – Release Engineer

Executes the Team Building using Jenkins

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

# Continuous Delivery for the Mainframe and Others

Capabilities to speed delivery of interdependent, multi-platform applications



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

# UrbanCode Deploy: What makes it different?

Composite Applications



The “What”

Components

Re-usable Workflows



The “How”

Deployment Automation

The “Where”

Environment Management

SIT



PROD



# Verify the Deploy results using UCD browser

The screenshot shows a browser window for IBM UrbanCode Deploy. The URL is <https://zdt:18443/#applicationProcessRequest/1ad41013-f10c-48ff-950c-4ee6a20641bf>. The page title is "IBM UrbanCode Deploy". The navigation bar includes links for Dashboard, Components, Applications, Configuration, Processes, Resources, Calendar, and Work Items. The main content area shows the "Applications" section with a breadcrumb trail: Home > Applications > A HC zOS COBOL CICS > Process Request. The title is "Application Process Request: A HC zOS COBOL CICS" with a "(show details)" link. Below the title are tabs for Log, Properties, Manifest, Configuration Changes, and Inventory Changes. The "Log" tab is selected. Under the "Execution" section, there is a table listing the steps of the process:

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

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

### Rebecca – Release Engineer

Executes the Team Building using Jenkins

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



# Example Pipeline Results

✓ GenAppNazarePipeline < 75

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

Pipeline Changes Tests Artifacts ⌂ ⚙️ ⌂ Logout X

```
graph LR; Start((Start)) --> Init((Init)); Init --> Git((Git Clone/Refresh)); Git --> DBB((DBB Build)); DBB --> ZUnit((ZUnit Test)); ZUnit --> 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 ⌂ ⌂ X

✓ > Perform monitor job  
✓ > monitor/...  
✓ > monitor/...  
✓ > Publish H...  
✓ > Send Slack message

✗ GenAppNazarePipeline < 71

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

Pipeline Changes Tests Artifacts ⌂ ⚙️ ⌂ Logout X

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

Deploy - 12s

↻ Restart Deploy ⌂ ⌂ X

✓ > Shell Script  
✗ > Publish Artifacts to IBM UrbanCode Deploy  
1 Deploying component versions '{GenApp=[latest]}'  
2 Starting deployment process 'DeployGenApp' of application 'GenApp-Deploy' in environment 'Z-QA'  
3 Deployment request id is: '16a36188-8ff8d-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 ⌂ ⌂ X

# Agenda

9:00 Introductions

9:20 DevOps on z Systems Introduction

9:45 Application Delivery Foundation (ADFz):

- IBM Developer for z Systems (IDz)
- Problem Determination Tool (PD Tools)

10:15 Break

10:30 LAB 1 - Working with z/OS using COBOL and DB2

or

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

11:45 Lunch

12:30 z Open Unit Test (Zunit) : Overview & Demonstration

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

→ 2:00 – 4:30 Choose one Optional lab:

→ **LAB 0 - Unit Testing for COBOL CICS (1 hour)**

→ **LAB 7 - IBM DBB with Git, Jenkins and UCD on Z (1 hour)**

→ LAB 2C Application Discovery : Find a candidate API from Catalog Manager Application

→ LAB 2D: z/OS Connect EE toolkit – Create/deploy Service and API for Catalog Manager App

→ LAB 5: UCD – Create UrbanCode Deploy infrastructure and deploy to z/OS

→ LAB 8 - Using Application Performance Analyzer (APA)

■ Lecture

■ Labs

■ Breaks

# Lab 0: Unit Testing for COBOL CICS

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

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

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

## Overview of development tasks

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

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

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

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

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

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

Overview of development tasks    User id →  empot05 and password → empot05

## 1. Get familiar with the application using the 3270 terminal

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

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

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

## 3. Modify the COBOL code using IDz.

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

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

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

## 5. Push and Commit the changed code to Git .

→ You will commit the changes to Git.

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

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

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

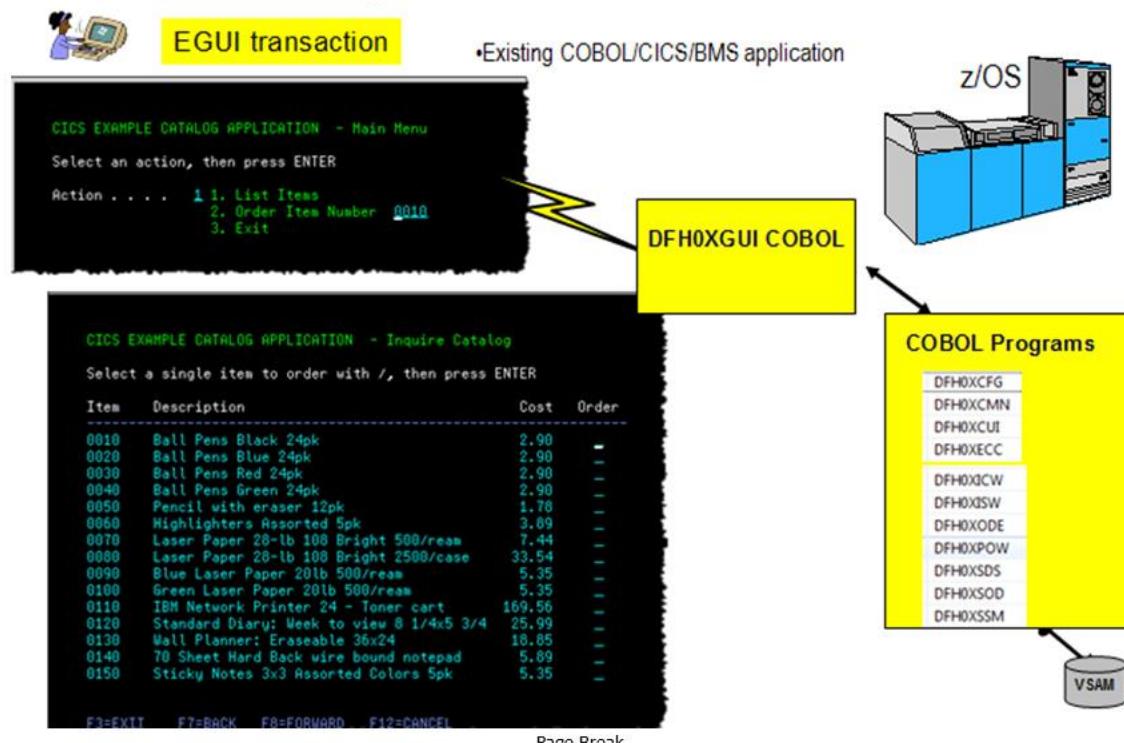
→ You will verify the results after the final deploy to CICS using UCD

## 8. (Optional) Understanding DBB Build Reports

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

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

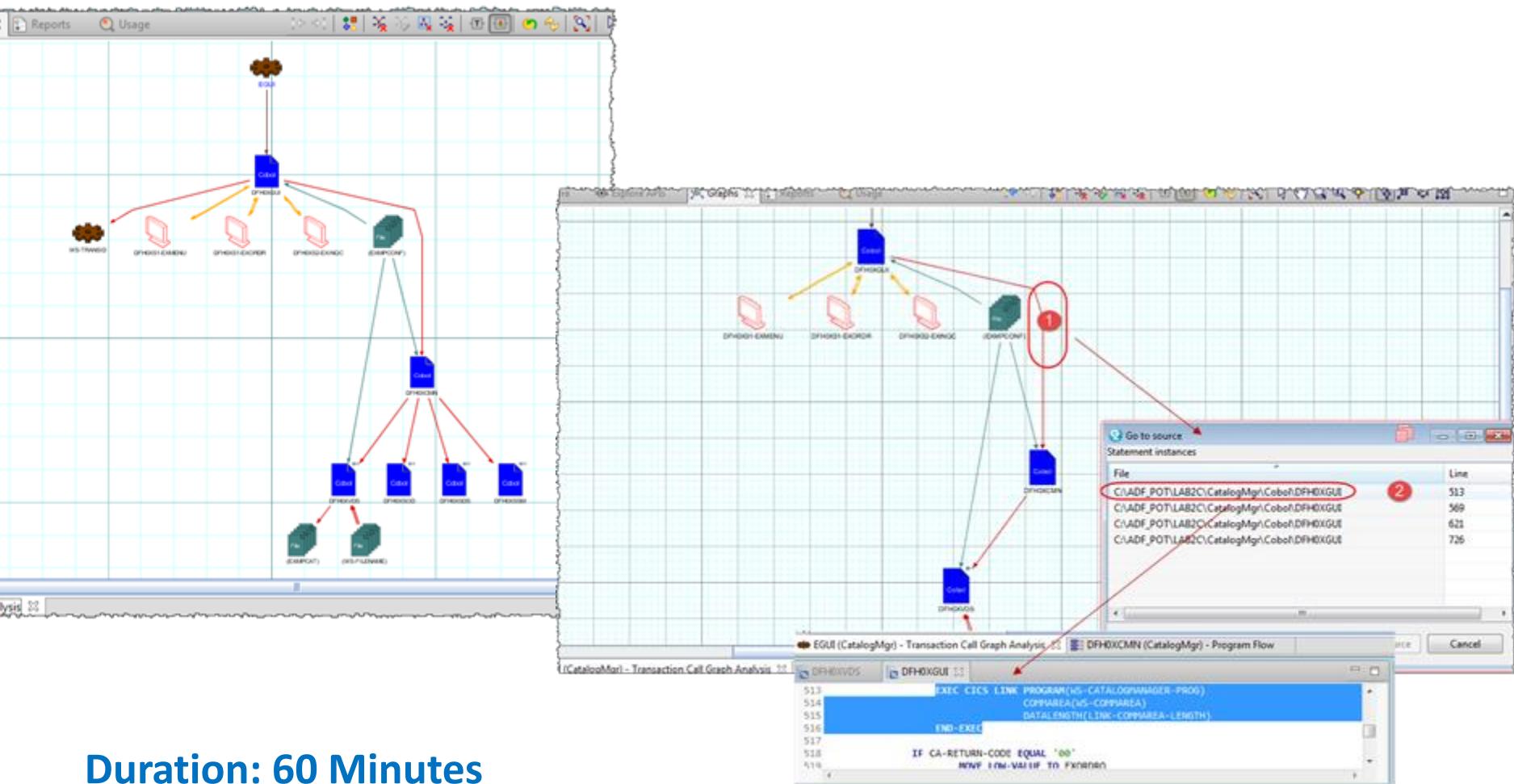
**Objective:** This lab you will go through the process of discovering a possible API from existing the Catalog Manager Application (EGUI transaction).



**Duration: 60 Minutes**

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

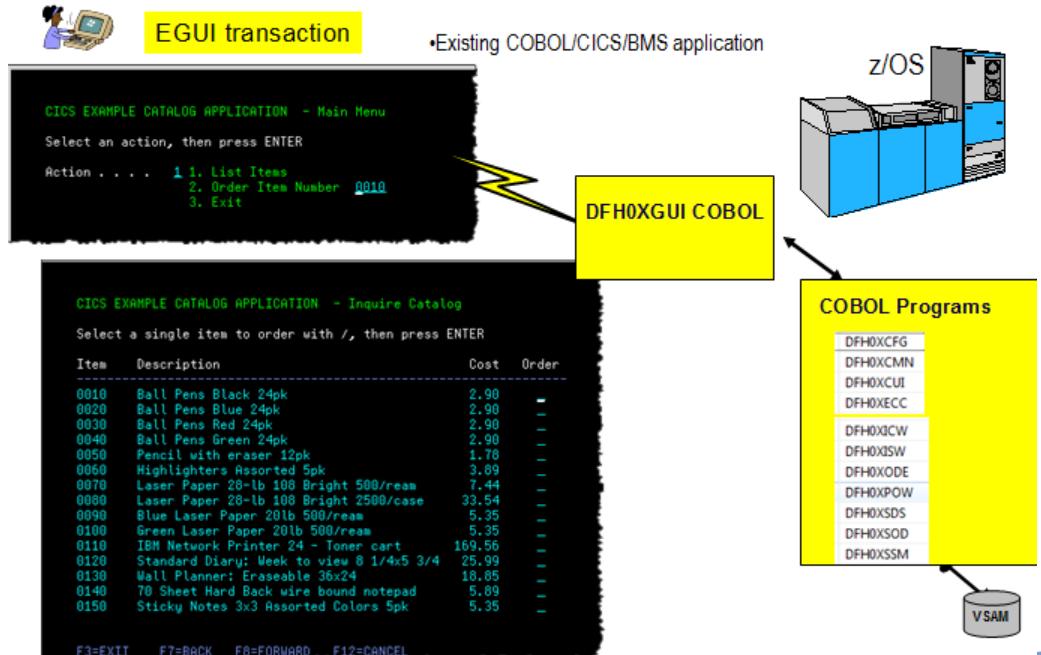
**Objective:** This lab you will go through the process of discovering a possible API from existing the Catalog Manager Application (EGUI transaction).



**Duration: 60 Minutes**

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

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



Duration: 60 Minutes

# Lab 5: UrbanCode Deploy → Focus on Deploy

## **Abstract:**

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

### **Part 1 – Create the UrbanCode application infrastructure.**

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

### **Part 2 - Create the UrbanCode deployment processes.**

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

### **Part 3 – Deploy the application to z/OS CICS**

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

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

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

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

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

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

Overview of development tasks    User id →  empot05 and password → empot05

## 1. Get familiar with the application using the 3270 terminal

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

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

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

## 3. Modify the COBOL code using IDz.

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

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

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

## 5. Push and Commit the changed code to Git .

→ You will commit the changes to Git.

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

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

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

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

## (Optional) Understanding DBB Build Reports

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

# Lab 8: Using Application Performance Analyzer (APA)

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

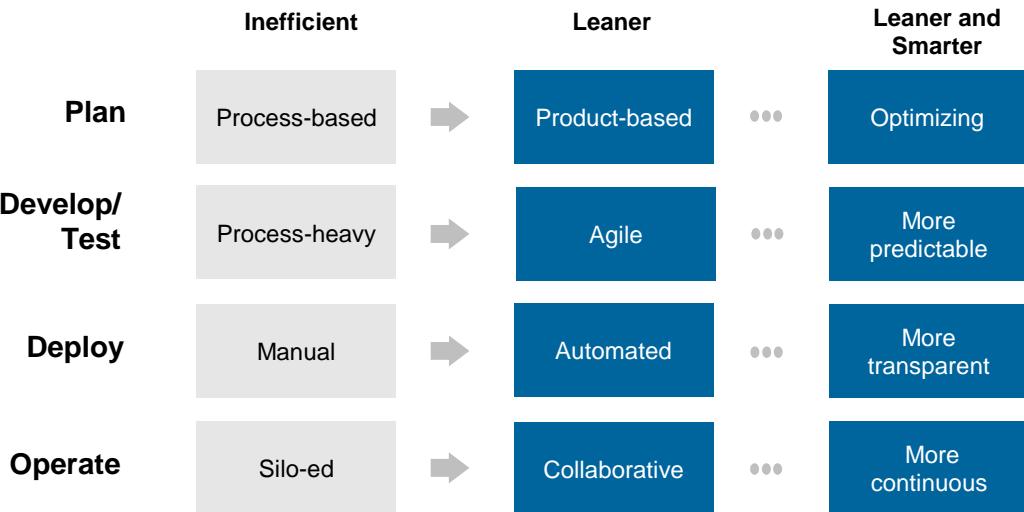
On this lab you will measure an existing COBOL/DB2 program running in batch.

## **Overview of development tasks**

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

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

# Start your transformation today with an IBM DevOps Workshop



Contact us about a no-charge  
DevOps assessment workshop

## Workshop Objectives

- Define business drivers for DevOps,
- Identify existing or planned DevOps initiatives
- Determine the top inhibitors within current software lifecycle
- Create an adoption roadmap for DevOps practices

## Overview

- Led by IBM DevOps Solution Architects
- Audience:
  - LOB Executives
  - Senior IT
  - Managers in Application Development and IT Operations

# IBM Z Trial Program



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



No charge environment



No set up, no install



Hands-on tutorials

## Available now:

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

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

## Coming soon:

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

Register now at [ibm.biz/z-trial](http://ibm.biz/z-trial)

## Additional Resources

— Mainframe CI/CD with an open toolchain:

- <https://www.linkedin.com/pulse/mainframe-cicd-open-toolchain-its-real-spectacular-minaz-merali/>

— Mainframe Dev Center:

- <https://developer.ibm.com/mainframe/>

— IBM DevOps for Enterprise Systems:

- <https://www.ibm.com/it-infrastructure/z/capabilities/enterprise-devops>

— IBM Z Trial:

- <https://ibm.biz/z-trial>

# No-Charge IDz/ADFz Training offerings for the fall

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

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

Date & Time	Topic
September 11 - 9:00 AM EDT	<b>IDz Entry-Level Training Module 6</b> This module instructs on the dataset utility features of IDz that are utilized for ISPF 3.x tasks: File allocation, File rename, File delete, Library compress, VSAM File create, GDG Model create. Batch Job management and interactive CLIST/REXX support is also part of Module 6.
September 18 - 9:00 AM EDT	<b>IDz Entry-Level Training Module 7 - MVS Subprojects</b> Learn how to use this convenient & powerful IDz feature to organize and manage individual PDS members and individual QSAM files on a per-project bases. MVS Subprojects are also utilized with SCM access and Embedded SQL code/test/tune.
September 25 - 9:00 AM EDT	<b>IDz Entry-Level Training Module 8 - DB2/SQL</b> IDz's world-class features for understanding your logical and physical data model (entities & their relationships), modifying DB2 test table values and coding/testing SQL: Using SPUFI files, Interactive SQL, and coding Embedded SQL statements directly into COBOL and PL/I programs, then running and using Visual Explain on the statements.
September 30 - 9:00 AM EDT	<b>** Next Iteration of IDz Entry Level Training Starts - Monday Sessions **</b> <b>Module 1 - IDz/Eclipse Introduction for ISPF Developers</b> Class overview, Workspace installation & customization, IDz Terms & Concepts & Eclipse Navigation for ISPF Developers, DevOps Tooling overview, Split-Screen, Use of multi-window'd views, Program Control Flow, Bookmarks for ISPF-Labels and more.



# No-Charge IDz/ADFz Training offerings for the fall

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

September 30 - 9:00 AM EDT

## **\*\* Next Iteration of IDz Entry Level Training Starts - Monday Sessions \*\* Module 1 - IDz/Eclipse Introduction for ISPF Developers**

Class overview, Workspace installation & customization, IDz Terms & Concepts & Eclipse Navigation for ISPF Developers, DevOps Tooling overview, Split-Screen, Use of multi-window'd views, Program Control Flow, Bookmarks for ISPF-Labels and more.

October 1 - 9:00 AM EDT

## **IDz Experienced Training: File Manager**

Learn how to use the File Manager "GUI" plug-in with IDz to browse and edit QSAM (Sequential) and VSAM file records - both in unformatted and in formatted (Copybook overlay) mode. Also learn how to connect to File Manager and use the tools such as Queries and Filtering Sorting & Excluding records. Time-permitting DB2 will be covered.

October 2 - 9:00 AM EDT

## **IDz Entry-Level Training Module 9 - Debug & Code Coverage**

Batch & Online debugging features and techniques, including JCL to launch Debug, program animation features, break points, variable monitors/changing variable values dynamically, Record & Playback and an introduction to Code Coverage.

October 7 - 9:00 AM EDT

## **IDz Entry-Level Training Module 2 - IDz Editing and Code Development Tools & Techniques**

Overview of IDz code editors & productivity techniques, ISPF emulation: Command line commands/Prefix area commands, Hot Key definitions, Editing source with embedded Hex characters, JCL editing, IDz's source download and program model, and Content Assist ("look-ahead typing")

October 14, 9:00 AM - 10:00 AM EDT

## **IDz Entry Level Training - Module 3 - Program Analysis Tools**

ISPF Search techniques (review), IDz graphical tools for: Logic Flow and program execution analysis, Graphical & declarative tools for Data Flow and impact analysis (within a single program).



# No-Charge IDz/ADFz Training offerings for the fall

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

October 30, 9:00 AM EDT

**\*\* Next Iteration of IDz Entry Level Training Starts - Wednesday Sessions \*\* IDz Module 1 - IDz/Eclipse Introduction for ISPF Developers**

Class overview, Workspace installation & customization, IDz Terms & Concepts & Eclipse Navigation for ISPF Developers, DevOps Tooling overview, Split-Screen, Use of multi-window'd views, Program Control Flow, Bookmarks for ISPF-Labels and more.

November 4, 9:00 AM EST

**IDz Entry-Level Training Module 6 - the ISPF 3.x Data Set Utilities**

This module instructs on the data set utility features of IDz that are available from the ISPF 3.x dialogs: **File allocation, File rename, File delete, Library compress, VSAM File create, GDG Model and GDG Data Set create**. Batch Job management and interactive CLIST/REXX support is also part of Module 6. An introduction to Menu Manager is presented

November 6, 9:00 AM EST

**IDz Entry-Level Training Module 2 - IDz Editing and Code Development Tools & Techniques**

Overview of IDz code editors & productivity techniques, ISPF emulation: Command line commands/Prefix area commands, Hot Key definitions, Editing source with embedded Hex characters, JCL editing, IDz's source download and program model, and Content Assist ("look-ahead typing")

November 11, 9:00 AM EDT

**IDz Entry-Level Training Module 7 - MVS Subprojects**

Learn how to use this convenient & powerful IDz feature to organize and manage individual PDS members and individual QSAM files on a per-project bases. MVS Subprojects are also utilized with SCM access and Embedded SQL code/test/tune.

November 12, 9:00 AM EDT

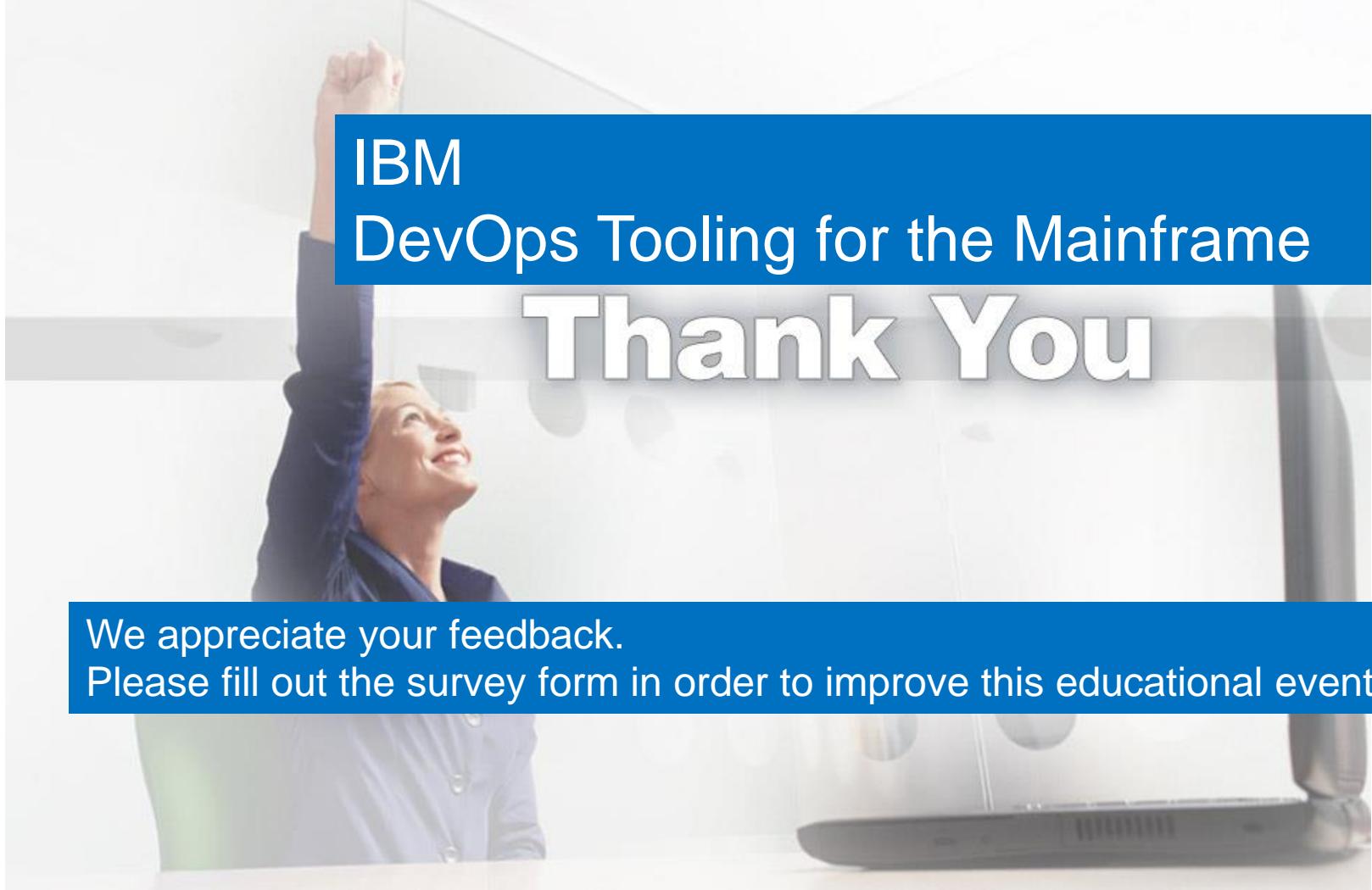
**IDz Experienced Training: Code Coverage**

Learn how to create Code Coverage statistical reports that analyze program lines executed during a batch or online (CICS) test run. Topics include: The ENVAR statement, Including/Excluding Programs, Running in batch



# Questions / Comments





IBM  
DevOps Tooling for the Mainframe

**Thank You**

We appreciate your feedback.

Please fill out the survey form in order to improve this educational event.

## Z Open Development (ZOD)

## IDz Enterprise Edition (IDz-EE)

## IBM Developer for z Systems (IDz)

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