

z/OS Unit Testing

Powered by **Test4z**

Venkat Balabhadrapatruni

Distinguished Engineer, Broadcom MSD

venkat.balabhadrapatruni@broadcom.com

When done wrong (or not done)



Code changes are...



...too *slow*



...too *costly*



...too *risky*

Unit Testing Basics

Complex & cumbersome tools have led to test neglect & quality voids

Does the change perform *as expected?*



Where tests exist, they're far too broad; difficult to understand & maintain

Where 'Unit' = smallest testable part of a program

'Shift-left' benefits unrealized

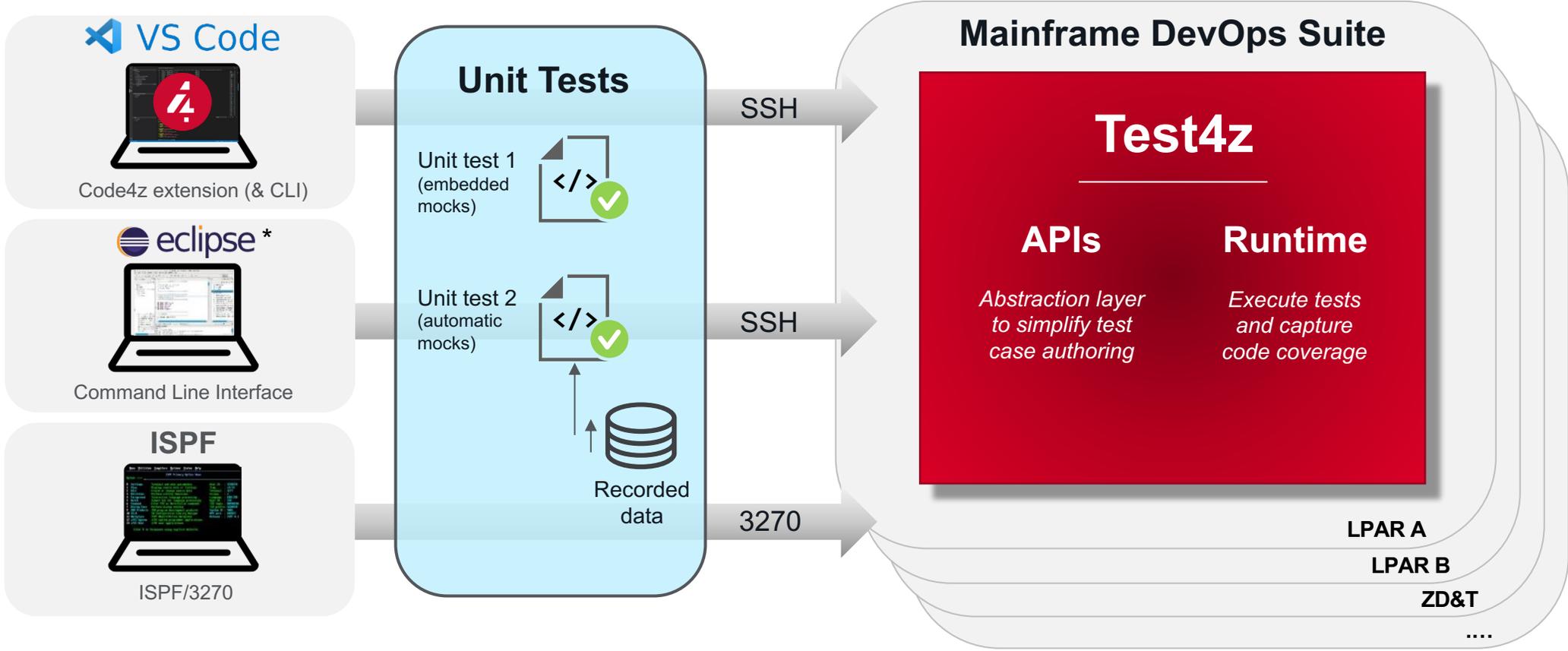
Introducing Test4z: Unit Testing Done Right

 KEEP TESTS SMALL		Fine-grained 'microtests' at paragraph level, subroutines in load modules
 ISOLATION		Advanced mocking of code (COBOL paragraphs, CALLS), data (Db2, file I/O, IMS), subsystems (CICS)
 MAINTAIN TESTS		Tests are code, co-located with application source; edit tests (e.g., add logic); programmatic access to variables
 CONTINUOUS INTEGRATION		Automated test execution whenever changes are pushed
 KEEP TESTS INDEPENDENT		Native record/replay w/ automatic data capture eliminates sharing of data & state across tests
 CODE COVERAGE		Native visibility facilitates coverage optimization by devs & managers

- 6 CORE PRINCIPLES -

Test4z Developer Experience

Simple for devs to understand and use; no tooling prerequisites



* Test4z CLI can be used with any IDE



Live Demo

Scenario 1 - Unit testing of a simple COBOL program

- Steps involved
 - Auto-create a Unit test template
 - Add data and logic to do the unit test
 - Run the unit test
 - View the code coverage

Scenario 2 – Record to capture test data

- Steps involved
 - Run / execute the program with real data
 - Capture the data
 - Leverage the data in unit test

Marquee Features

Test4z

- Easy to set up
(non APF auth load library)
- No prerequisite on Debugger
- zD&T compliant
- No instrumentation of user code

Unit Testing

- Load module | CSECT | paragraph
(level of testing)
- Programmatic unit testing
- Early development testing
(no integration environment needed, TDD)
- Architecture build to support multiple languages
(COBOL test API, Python*)
- Code/dependency mocking
- Data stubbing
- Code coverage

Record & Replay

- Source code language independent
(can be COBOL, PL/I*, C*, HLASM*)
- No source code needed
(DWARF required for source statement coverage)
- Recordings in human readable format
(JSON)
- Sub-system virtualization
- Unhappy path recording

Client

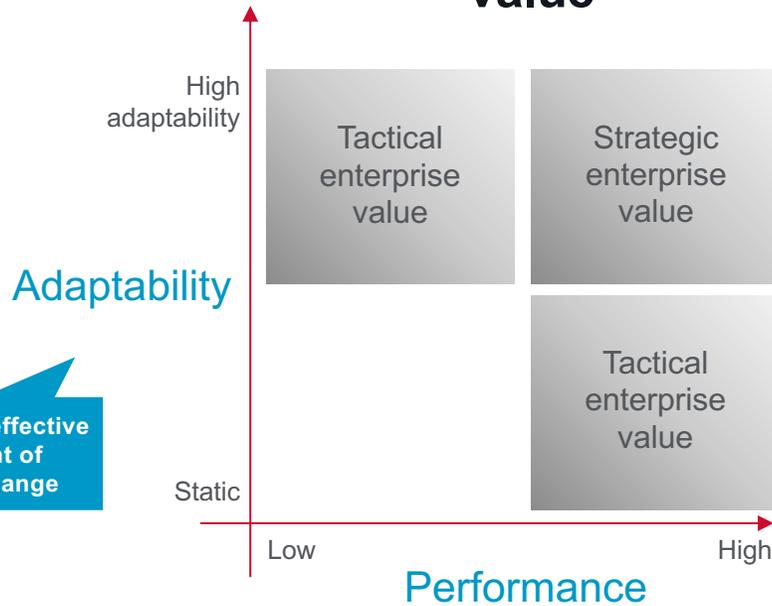
- Developers' choice of tooling
(Layered approach: 3270, Command Line, VS Code, ...)
- Integrates with VS Code
(one of the most used IDE)
- Command Line (CLI) interface with any IDE
- Seamless integration into automated CI/CD pipelines
- Combined test coverage

Why Unit test?

Improve the value of mainframe applications

Greater business value

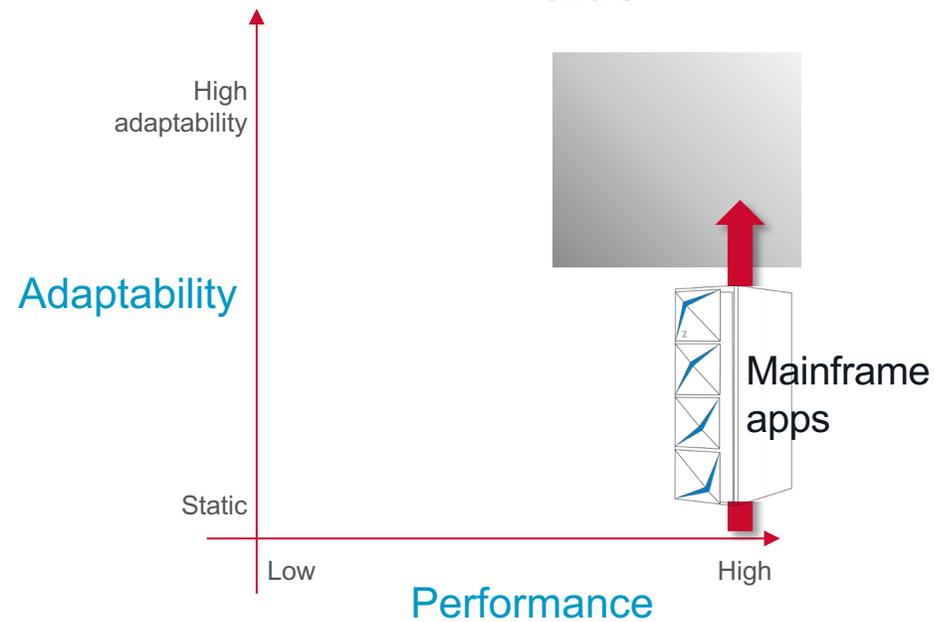
Application value



timely & cost effective enablement of business change

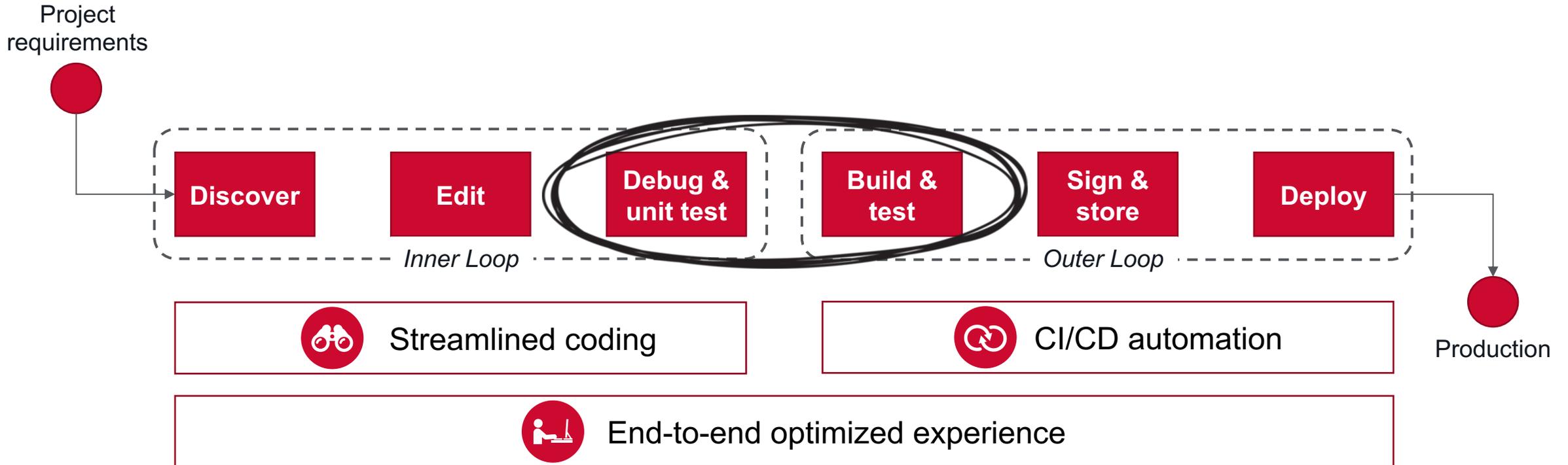
reliability, cost effectiveness, security, etc. as designed

Application value



Project Sustainability

Unit tests become automation building blocks; tribal knowledge retained



Let's make mainframe a platform of YES...



...by doing unit testing right



Test4z

Summary

- Traditional mainframe application code can be unit tested!
- Tools, technology and learnings are available to address the need
- Partner with us to learn and help evolve the technology

Next Steps

On-site engagement

Time	2-3 days
Goal	Validate Test4z capabilities in your environment on your application
Setup	COBOL batch application (Db2, File I/O, IMS DLI)

Alternatives

- *Virtual engagement using workshop setup*
- *Technical deep dive follow up*

Prerequisites

Business/sample application

- Batch COBOL program (ideally that is used in real-world production)
 - Simple Example
 - More Complex Example
- CICS COBOL programs not supported yet - work in progress
- Db2, File I/O, IMS DLI (IMS batch is supported, IMS/TM not supported)
- Only IBM supported compilers (COBOL 5.1 and below unsupported)

Mainframe

- LPAR or zD&T
- Ability to XMIT data sets - load library, sample library and JCL data sets will be created
- No APF authorization required
- No specific security profile required
- USS - user home directory and ability to SSH into USS using ssh-key
- (optional) z/OSMF or FTP or RSE configured for Zowe Explorer to connect to

Client

- Windows or macOS
- [Node.js](#)
- (optional) [Java 11](#) - needed for Zelda language
- (optional) [VSCode](#) - needed for Code Coverage visualization
- (optional) [Zowe Explorer](#) - optional for easier data set access from VSCode. Requires either z/OSMF or FTP or RSE to connect to the mainframe.

Broadcom artifacts to download

- Access to <https://validate.broadcom.com/> validation portal
- Alternatively we can provide the artifacts through a private repository on <https://github.com/>

Thank you