Automated regression and component test

Will Yates
Sept 2018

IBM

Disclaimer

- IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice and at IBM's sole discretion.
- Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
- The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.
- The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.
- Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here

24 hours

3234 Test Classes

Average 8 minutes

26,835 mins elapsed time

447 hours of test time

> 18 ½ parallel tests constantly executing

0 mins of human intervention

Where we started

And why we needed to change

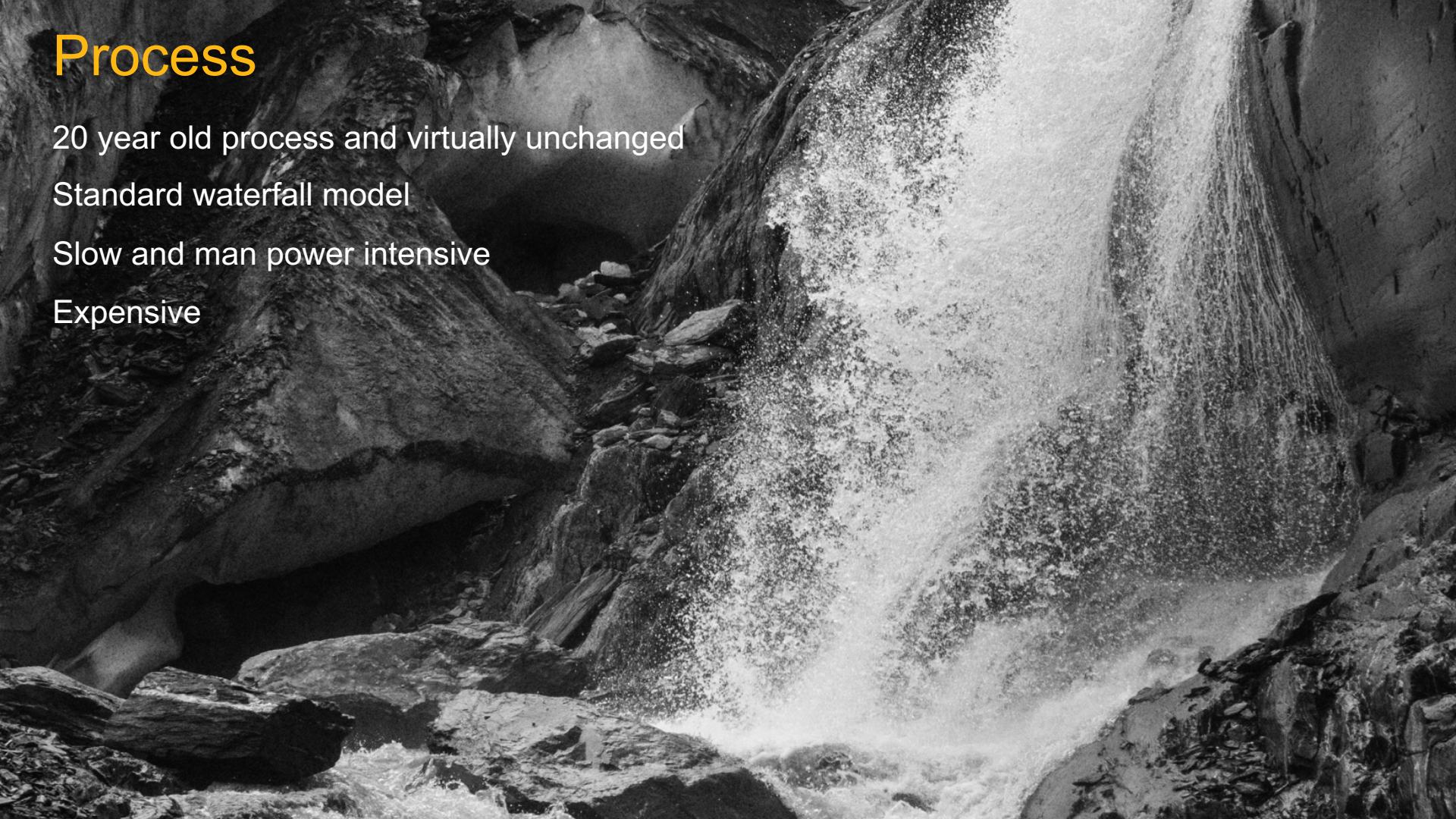
Process

20 year old process and virtually unchanged

Standard waterfall model

Slow and man power intensive

Expensive



Tools

In house built VM Based IDE & SCM

Lack of skills

Couldn’t support future development such as Java

Tests

Home grown test tooling across multiple frameworks

Unreliable tests – Lots of manpower to maintain

Difficult to extend

Not a scalable test runner architecture = long running tests

New Beginnings

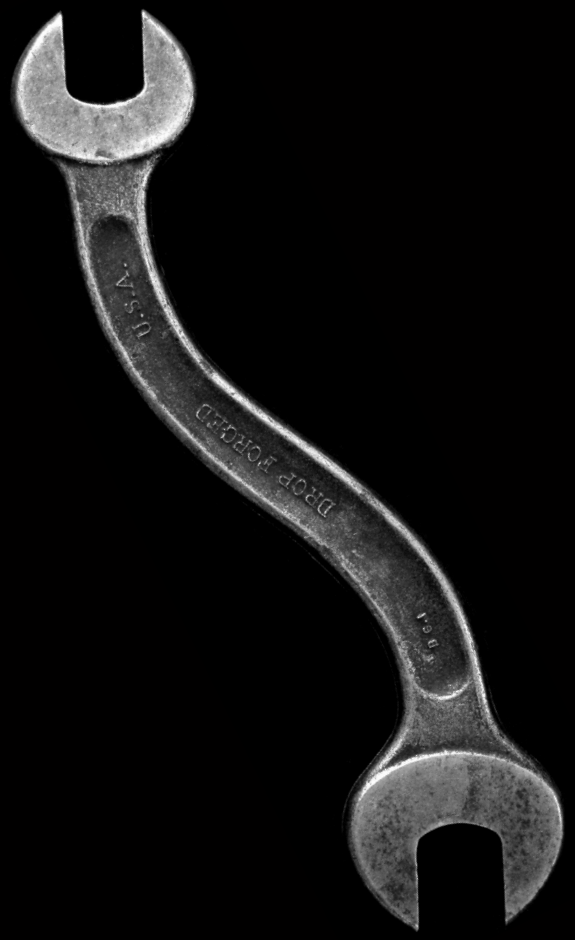
2005 - 2009

Process Changes

- Adoption of IBM Rational Unified Process
- Adoption of iterative development
  - CICS (4 months -> 2 months -> 1 month)
  - CICS Explorer (-> 2 weeks)
- Adoption of multi-disciplinary teams (Dev, Test, Doc, Support, Performance)
- Adoption of Agile Artefacts (Epic, Story, tasks, burndown, prioritised backlog)

Tools Changes

- Rational Team Concert
  - Initially used for project & task management
- ANTz (ANT for z/OS)
  - Used for product build (75% reduction in build time)
- Rational Team Concert
  - Used to hold source code
- Engineers adopted IDz / RDz / Eclipse

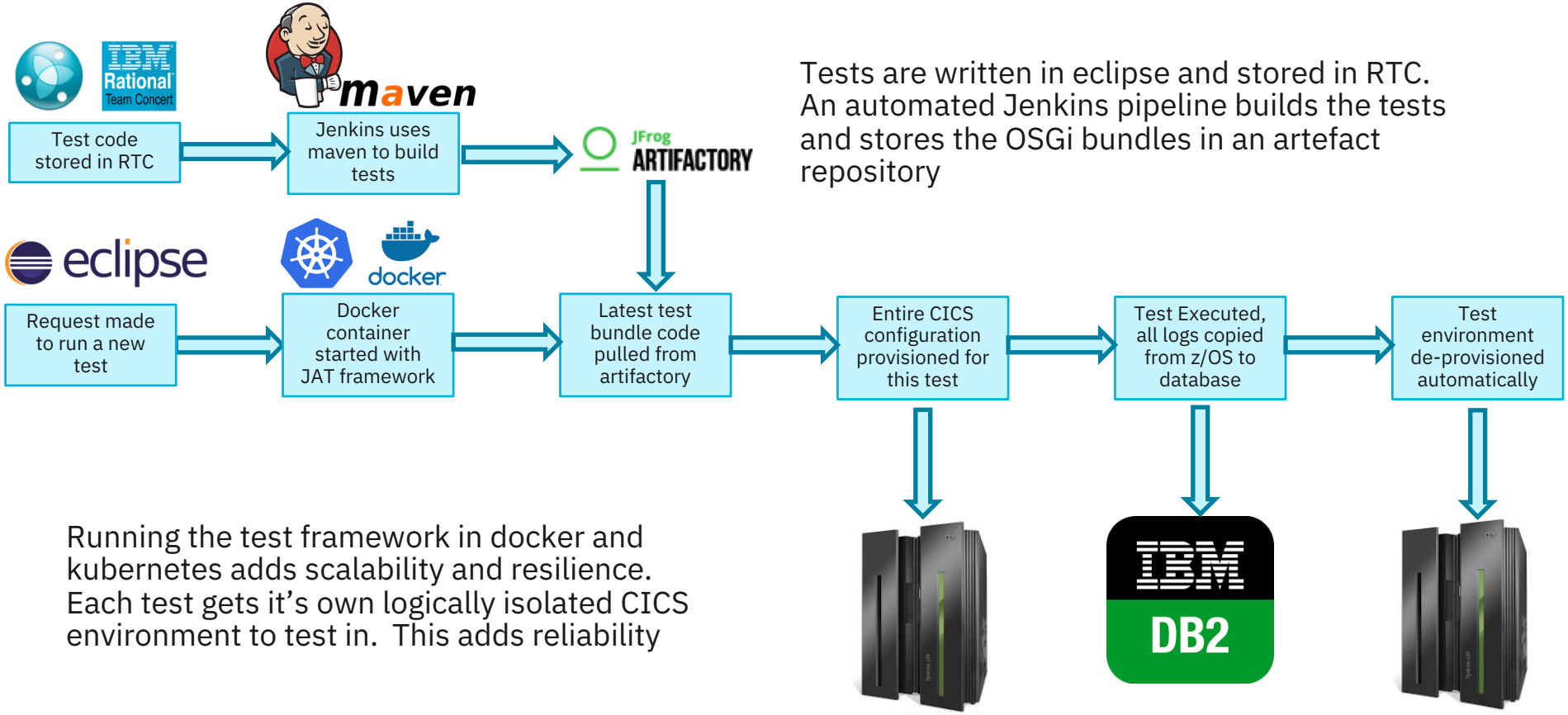


The problem with test tooling – one size does not fit all

- We are fairly unique:
  - Thousands of test suites
  - Using varied test technologies (terminal interaction, batch jobs, web services, selenium web browser interaction) etc
  - Running against z/OS and integrating with various subsystems
  - 5 in service releases
  - 3000 test suite invocations per 24 hours
  - 600 hours of testing per 24 hour period

  - No existing tool on the market satisfied all our needs, although we could learn from a lot of them:
  - Junit had an ideal extensible test framework
  - OSGi / Maven provided a java architecture
  - Docker provided good test isolation
  - z/OSMF provided a way to talk to z/OS
  - We just had to fill in the blanks …

Running a test in JAT



Tests are written in eclipse and stored in RTC. An automated Jenkins pipeline builds the tests and stores the OSGi bundles in an artefact repository

Running the test framework in docker and kubernetes adds scalability and resilience. Each test gets it's own logically isolated CICS environment to test in. This adds reliability

Writing a test in JAT

```
//Let JAT know this is a CICS test
@CICSTest

//Provision a CICS environment
@Topology(”SingleRegion”)

//Provide some local input to the provisioned CSD
@CSDInput(file = "csdinput/LE370CSD", tag = "A")

//Compile a CICS enabled program
@CICSProgram(lang = Language.COBOL, name = "programs/oocob014.cobol")

//Add a load library to DFHRPL
@LoadLibrary(LLQ = "FV.PROGRAMS.LOAD", addHLQ = false)

//What area does this test exercise
@AreasTested(primaryArea=TestingArea.Core_Logger)

public class BasicTest{

@CICSTerminal(tag = "A")
public ITerminal terminal;  //JAT inserts an object ref to a simulated terminal connected to CICS

@CICS(cicsTag = "A")
public ICICS cicsRegion; //JAT inserts an object ref allowing the test to interact with CICS

@Test
public void basicTest() throws TerminalException {
      terminal.sendTextWithEnter(“CEMT”);
}
```

Venus Integrated

Adapaters_MQ
Mike Fish

AppDeployment_Bundles
Chris Atkinson

AppServices

AppServices_API

AppServices_AsyncServices

AppServices_Debug
Jeremy Weaving

Appservice_EP
Jackie Scott

Cloud
Mat Comer

Comms

Comms_FunctionShipping
Will Yates

Comms_IPIC
Xue Yong Zhang

Core_Initialisation
Roger Brooks

Core_Logger
Roger Brooks

Core_Recovery
Roger Brooks

Core_Security
Chris Atkinson

CPSM
Mat Comer

Operations_Policy
Jackie Scott

Runtimes_JVMServer
Alex Brown

Runtimes_Node
Chris Atkinson

Web_TCPIPServices
Jeremy Weaving

Web_Webservices
Gill Curwen

Web_WebServices_JSON
Jeremy Weaving

RDO
Gill Curwen

Web_WebServices_PHP
Qin Li

Web_Webservices_zOSConnect

Core Regression Operations_Policy (Global)

Testcase Name ↑	Environment	Results (Most recent on left)	Run ID	Owner	Tags	End Time
AsyncRunTransidRequestPolicy	Venus:Integrated	✓✓✓✓✓✓✓✓✓✓	J719510	Hill3_Provisioning	DTS core_regression phoenix	05/09/2018, 05:07:42
CLOUDV5_core	Venus:Integrated	✓✗✓✓✓✓✗✓✗✓	J718839	Hill3_Provisioning	DTS core_regression phoenix	05/09/2018, 01:30:51
CPUTimePolicyandElapsedTimePolicy	Venus:Integrated	✓✓✓✓✓✓✓✓✓✓	J719516	Foundation	DTS core_regression phoenix	05/09/2018, 05:20:52
DB2Policy	Venus:Integrated	✓✓✓✓✓✓✓✓✓✓	J719402	Hill3_Provisioning	DTS core_regression phoenix	05/09/2018, 04:42:50
DLIPolicy	Venus:Integrated	✓✓✓✓✓✓✓✓✓✓	J719515	Hill3_Provisioning	DTS core_regression phoenix	05/09/2018, 05:22:24
ExecCicsRequestPolicy	Venus:Integrated	✓✓✓✓✓✓✓✓✓✓	J719380	Hill3_Provisioning	DTS core_regression phoenix	05/09/2018, 04:23:05
FileRequestPolicyandSyncpointRequestPolicy	Venus:Integrated	✓✓✓✓✓✓✓✓✓✓	J719065	Foundation	DTS core_regression phoenix	05/09/2018, 02:38:41
NameCounterRequestPolicy	Venus:Integrated	✓✓✓✓✓✓✓✓✓✓	J719540	Hill3_Provisioning	DTS core_regression phoenix	05/09/2018, 05:22:56

Overview | Definition

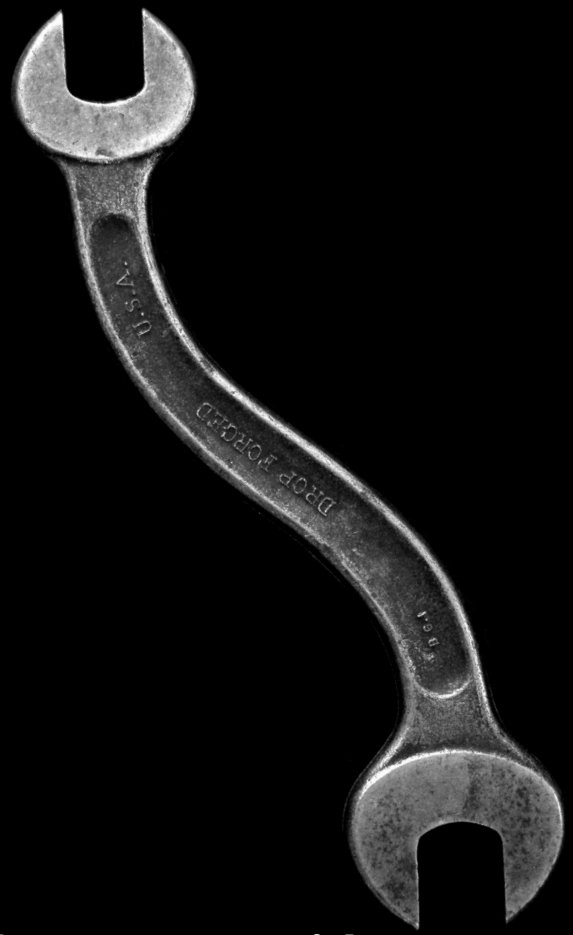
Results

Result Counts

Result	Count	Perc
Passed	30	100.0%
Total	30	

What JAT brings us

- Error resistant way of writing tests
- Configurations specified through annotation
- Scalable, logically isolated, repeatable test environments
- All diagnostic information held in one place
- Reporting of test results
- Reliable, performant, maintainable tests

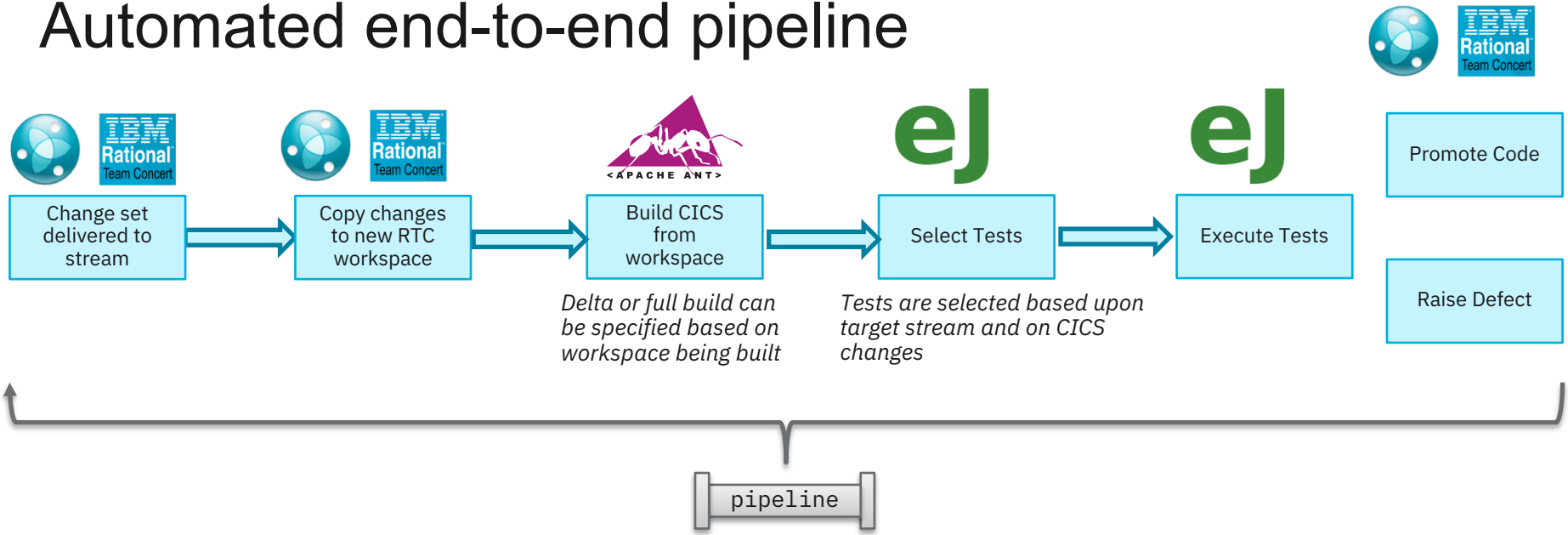


No other test framework can provide what JAT provides

Target

Include a change set in a quality beta build within 24 hours!

Automated end-to-end pipeline



Jenkins coordinates the entire pipeline

Where possible off the shelf plugins have been used to integrate with our chosen tools, if plugins are not available they have been written

Pipeline is a parallel process so multiple instances can be executed

Pipeline can be triggered for any set of changes making it re-usable

Automated end-to-end pipeline

Integrated Pipeline

- Every 30 mins
- Pipeline triggered for smallest set of delivered changes
- BVT set of tests executed along with a more intense selection based on CICS modules changed
- If pipeline succeeds
    - Changes promoted to Integrated stream
- Stream built each night for a daily build

Integrated Build

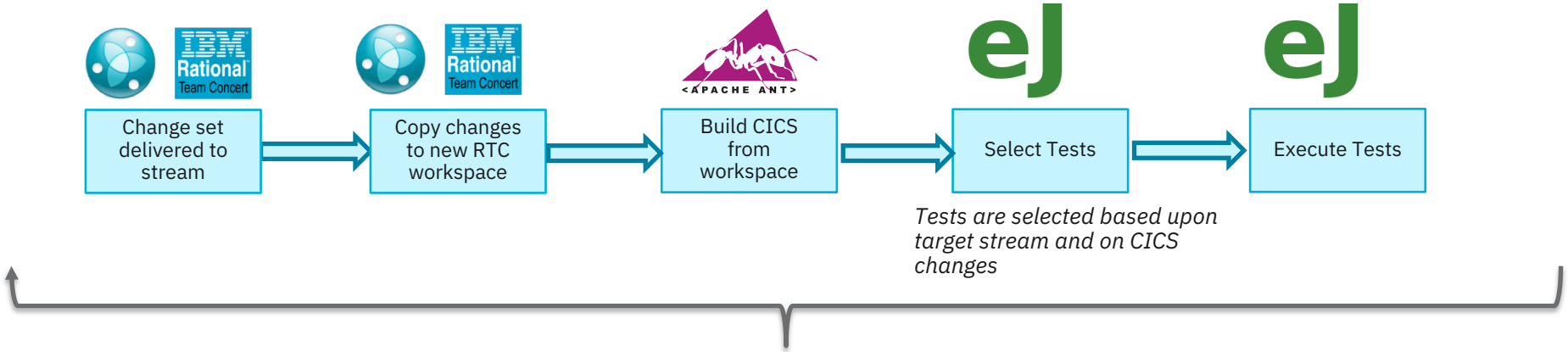
- Every 8 hours
- Pipeline asked to run a delta build
- Build is SMPE packaged and installed
- SMPE package is installed
- Core Regression Tests executed

Beta Build

- Every 8 hours (+4 hrs from Integrated)
- Entire workspace is built by Ant/z (FULL)
- Build is SMPE packaged
- SMPE package is installed (DFHISTAR and non ISTAR)
- Entire test corpus is run

Bad changes never make it to a build, neither do they hold up good code

Shift Left



Since the pipeline can be executed against any set of changes we can even run it against a developers own RTC workspace before code is delivered

JATs scalable nature means tests are run in parallel – minimising developer wait time

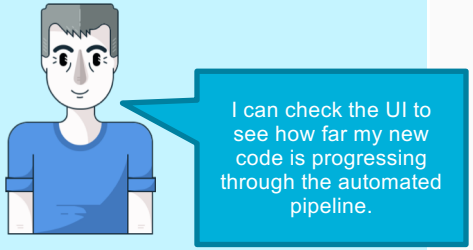
Highly reliable tests reduce false positives and builds trust

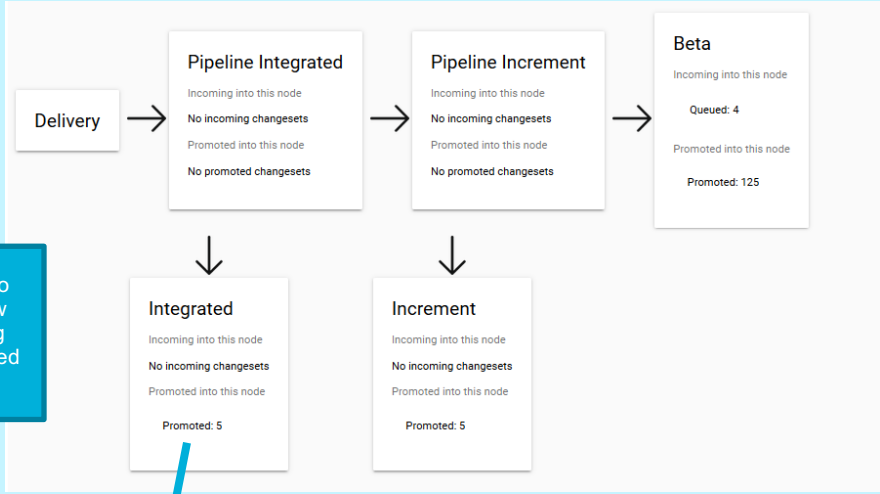
All diagnostic data is available in the eclipse client for a developer

Supports TDD

Only once the developer has run shift left can they deliver their code.

The CICS Automated Build, Deploy and Test Pipeline





13:09 22/05/2017	Fix xsdfor policy events. Add {USERTAG} variable for user to replace.	promoted	139435: System rule XSD files don't match the expected event in DSI
13:09 22/05/2017	Flow to 690 plus additional chnage for R2	promoted	138823: Defect for fixing APAR PI79851 in CICS TS V5.2 138817: Defect for fixing APAR PI79851 in CICS TS V5.3
13:09 22/05/2017	Flow to 700	promoted	138817: Defect for fixing APAR PI79851 in CICS TS V5.3
13:09 22/05/2017	Guard trace call	promoted	139648: WLPLink.c has unguarded logging message
13:09 22/05/2017	INCOMPLETE state correct when not UNDEPLOY	promoted	139566: QTYPE: DFHDPLOY: UNDEPLOY APPLICATION can transition from disable to discard state too early

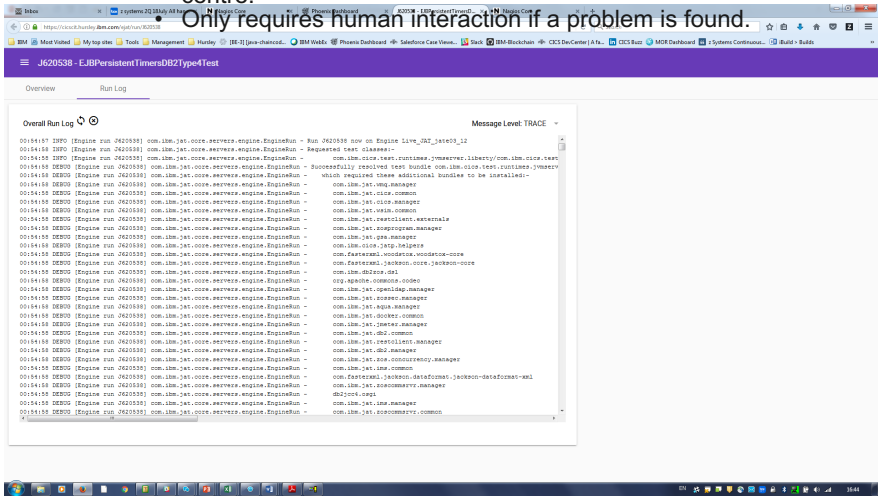
Automated pipeline testing

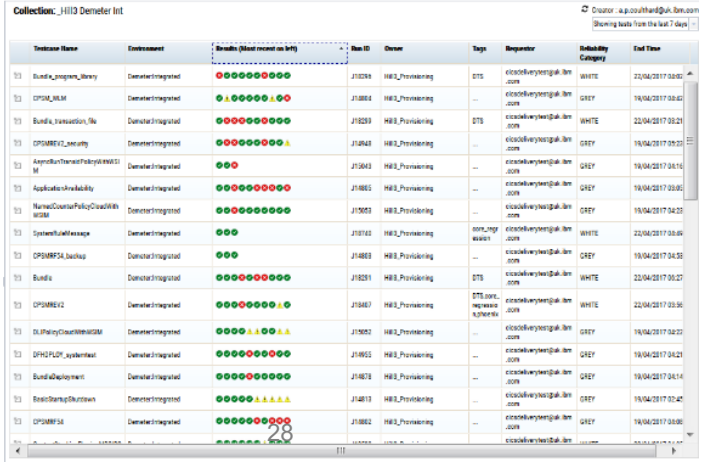
In parallel to the automated pipeline, extensive overnight automated regression testing gives daily awareness of the quality of the CICS product

- More in-depth testing than the core pipeline tests. Over 1000 multi-language, multi-product test suites compared to over 500 test suites executed in the pipeline
- JAT design enables highly parallel execution
- Stable, high quality test cases. Expectation is a 100% pass rate.
- Tests are CICS level agnostic allowing them to be run on all releases of CICS for which that test is relevant
- Test results are comprehensively displayed in a UI.
  - Displays the results of the last 10 runs upfront, with details of the last 100 runs also available.
  - Allows teams to create collections of tests. Teams can create “collections” of tests to view test results in anyway they wish e.g. by Hill, by Functional Area etc.
  - Enables a collection of tests to be created for each Hill

Implemented an Automated Pipeline for regression testing of PTF fixes.

- One click of a button to initiate building and packaging of a PTF, deploying CICS, running automated tests, automated submit to the PTF distribution centre.
  - Only requires human interaction if a problem is found.





Build notifications in Slack

Service Tests APP 1:08 PM
Service » 710-Apply-Test-Accept-Ship - #131 Success after 47 min (Open)
Test Status:
Passed: 513, Failed: 0, Skipped: 0

Service Tests APP 3:08 PM
Service » 700-Apply-Test-Accept-Ship - #128 Success after 52 min (Open)
Test Status:
Passed: 513, Failed: 0, Skipped: 0

Service Tests APP 4:54 PM
Service » 670-Apply-Test-Accept-Ship - #34 Success after 41 min (Open)
Test Status:
Passed: 492, Failed: 0, Skipped: 0

Service Tests APP 7:05 PM
Service » 710-Apply-Test-Accept-Ship - #132 Success after 44 min (Open)
Test Status:
Passed: 513, Failed: 0, Skipped: 0

Service Tests APP 9:03 PM
Service » 700-Apply-Test-Accept-Ship - #129 Success after 47 min (Open)
Test Status:
Passed: 513, Failed: 0, Skipped: 0

So why tell you all this

Why am I telling you this

- To demonstrate the art of the possible
- To show the amount of testing we perform for CICS
- To understand how mature your testing infrastructure is
- To understand the appetite if we were to open source JAT and make it freely available
  - Yes we are seriously considering this

Summary

Summary

- Testing CICS ≈ Testing CICS applications
- In CICS we have a 1st Class Test framework we believe is the best in the world
- We really want to share this with you
- Interested?

wyates@uk.ibm.com