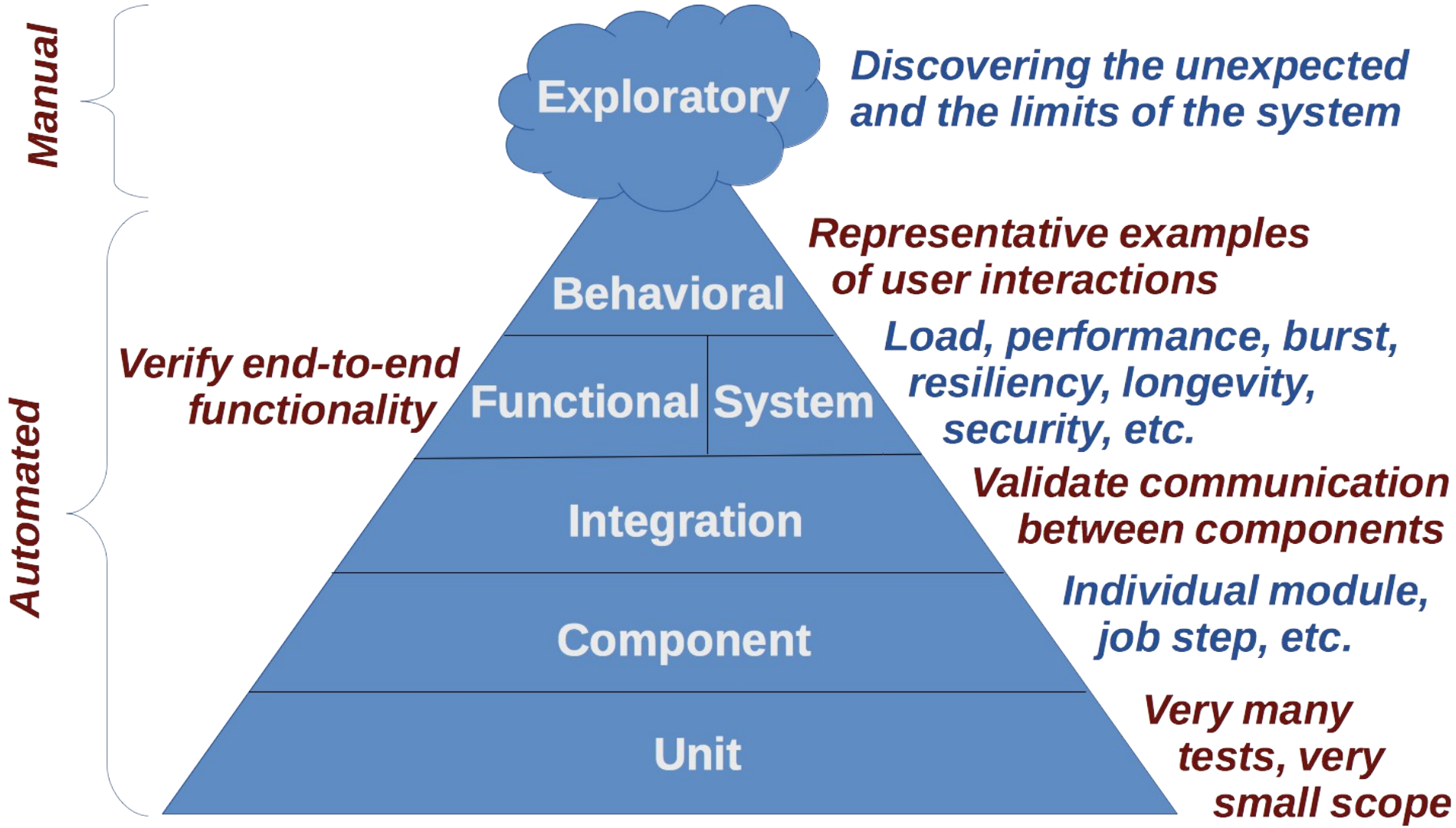
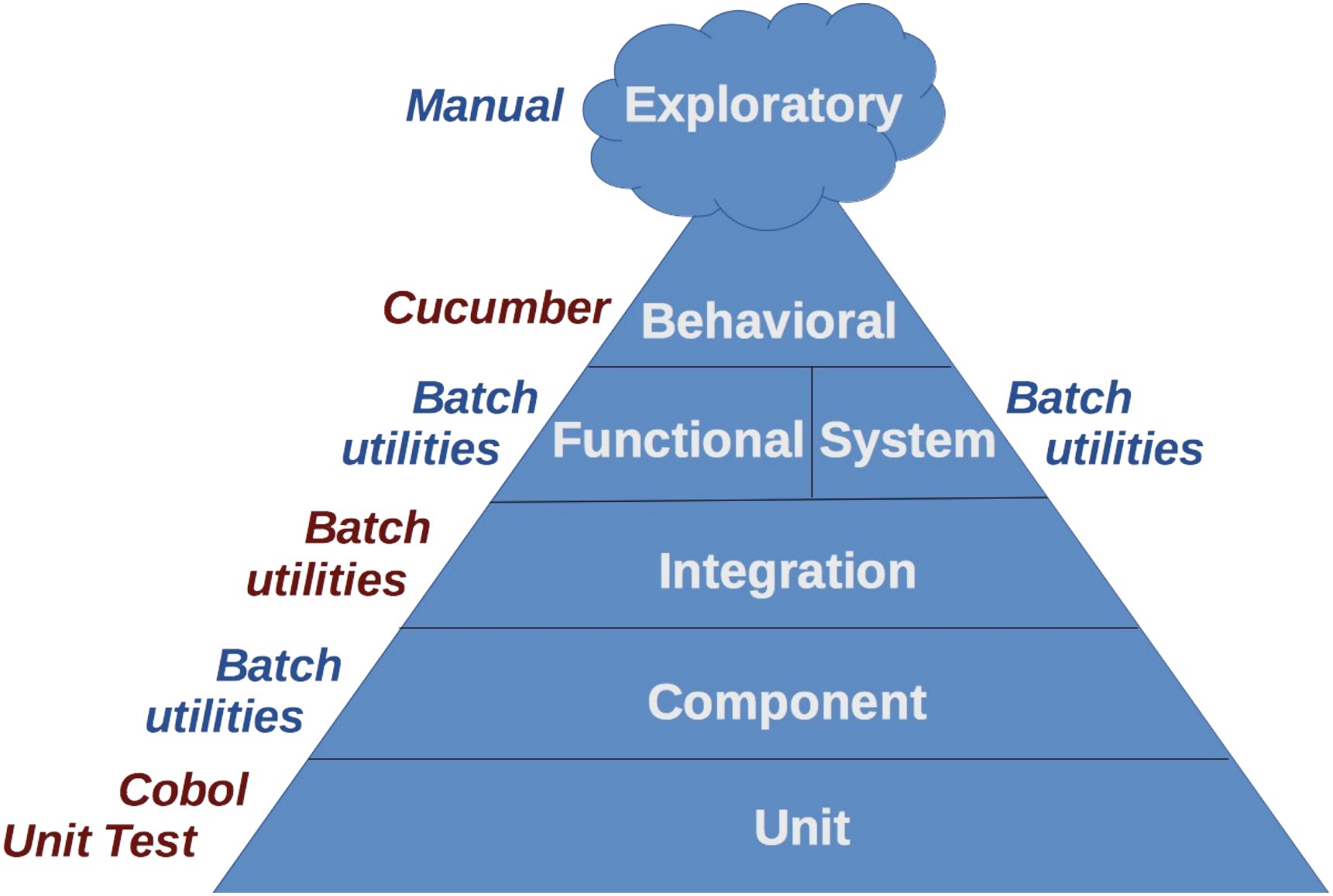
It’s possible to automate COBOL testing

...but it requires some additional work.

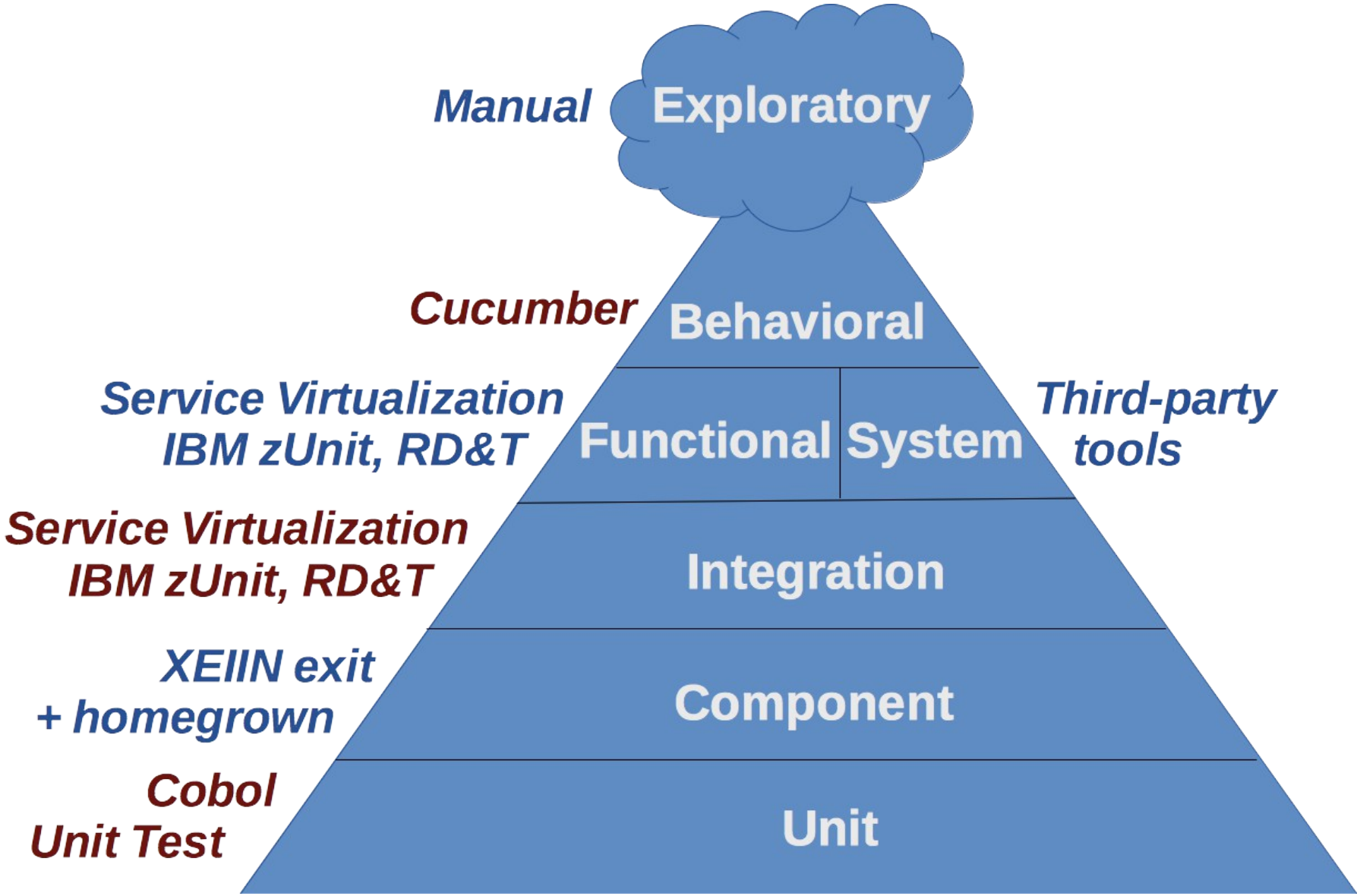
Testing Pyramid (General)



Testing Pyramid (Mainframe Batch)



Testing Pyramid (Mainframe CICS)



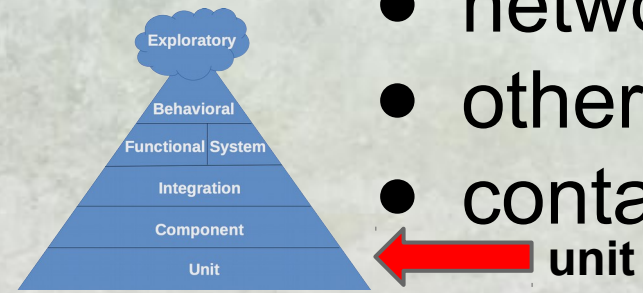
What's a "Unit Test?"

Single path through the smallest functional piece of code

- Java - *a method*
- C++ - *a function*
- Cobol - *a paragraph*

***All* external dependencies are fake**

- files
- databases
- network connections
- other applications
- containers & frameworks



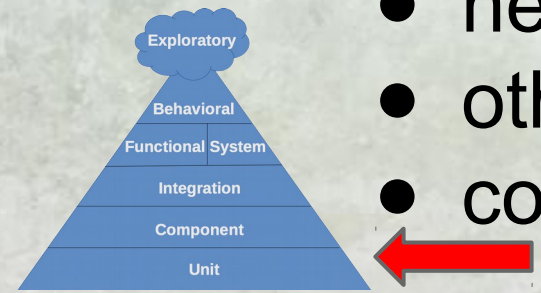
What's a "Unit Test?"

Single path through the smallest functional piece of code

- Java - *a method*
- C++ - *a function*
- Cobol - *a paragraph*

***All* external dependencies are fake**

- files
- databases
- network connections
- other applications
- containers & frameworks



unit

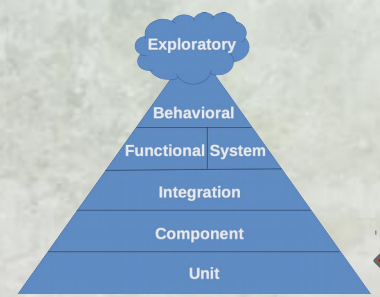
What's a "Unit Test?"

Single path through the smallest functional piece of code

- Java - *a method*
- C++ - *a function*
- Cobol - *a paragraph*

All **external dependencies are fake**

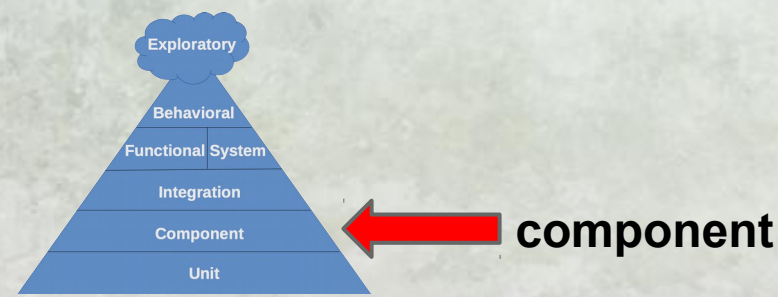
- files
- databases
- network connections
- other applications
- containers & frameworks



unit

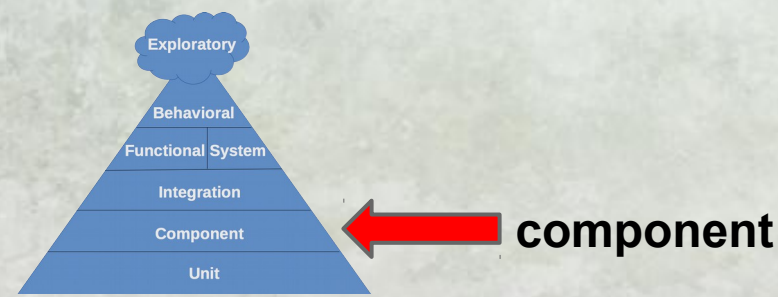
What's a "Component Test?"

- Verifies behavior of a software component
- Larger scope than a unit test
- Excludes dependencies outside the code under test



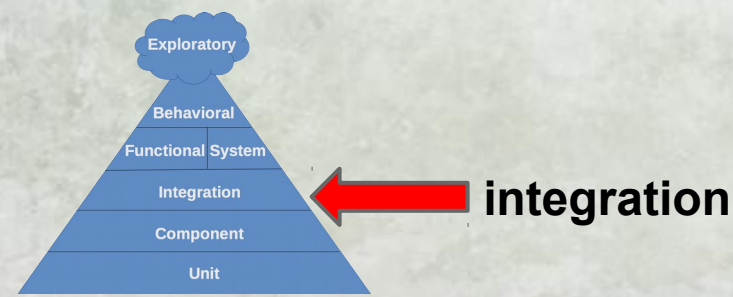
What's a "Component Test?"

- Batch - a single job step
- CICS - a single program or group of collaborating programs



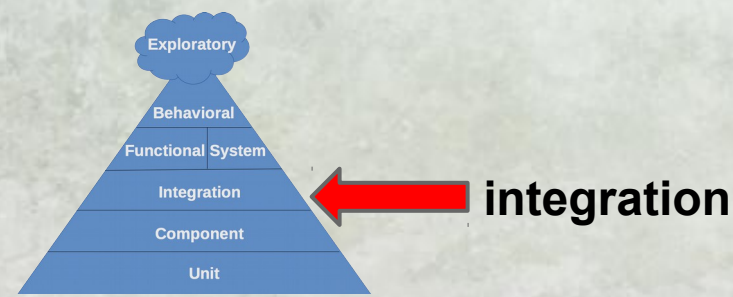
What's an "Integration Test?"

- Verifies that two components talk to each other as expected
- Normal interaction ("happy path")
- Abnormal interaction (timeouts, exceptions)
- Omits the details covered by unit tests - is concerned with *integration*, not *functionality*



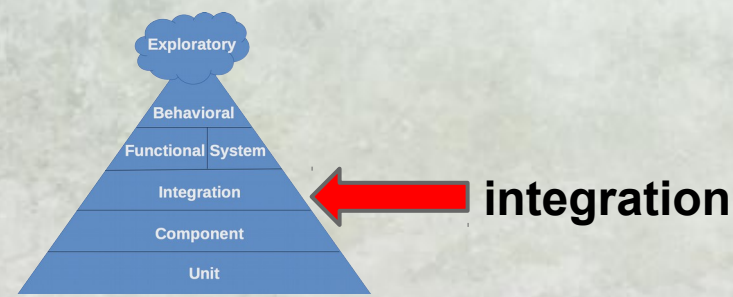
What's an "Integration Test?"

- Batch: Program handles responses from CALL statements appropriately
- Batch: Program handles responses from EXCI calls appropriately
- Batch: Program handles responses from MQ Series queues appropriately



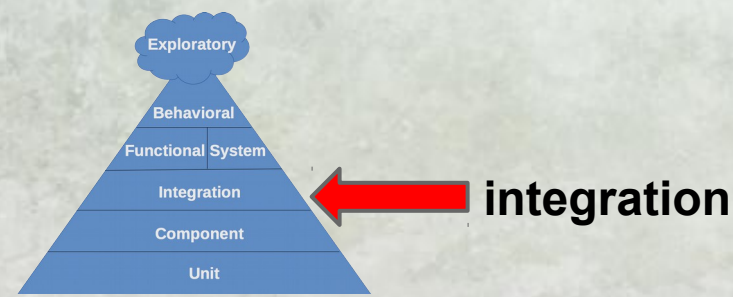
What's an "Integration Test?"

- CICS: Program handles responses from XPI calls appropriately
- CICS: Program handles responses from XMS calls appropriately
- CICS: Program handles responses from MQ Series queues appropriately



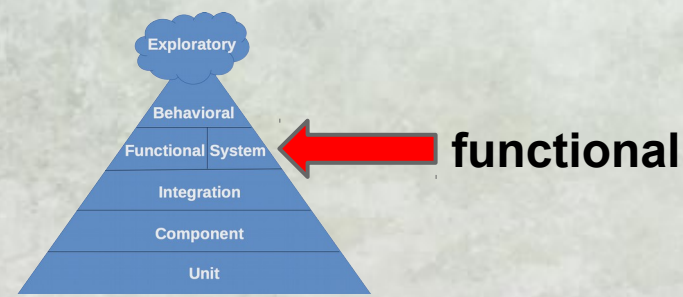
What's an "API Test?"

- Verifies client interaction with a programmatic API (usually a SOAP-based or RESTful API)
- A kind of integration test
- Probably not relevant to mainframe systems



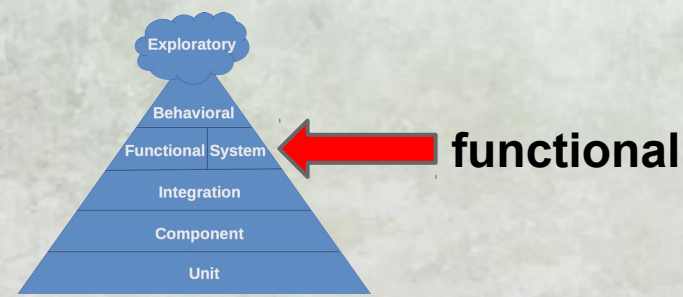
What's a "Functional Test?"

- A single path through the application for a single transaction
- End-to-end test with external interfaces live
- Usually omits the UI but touches all other architectural layers in the application



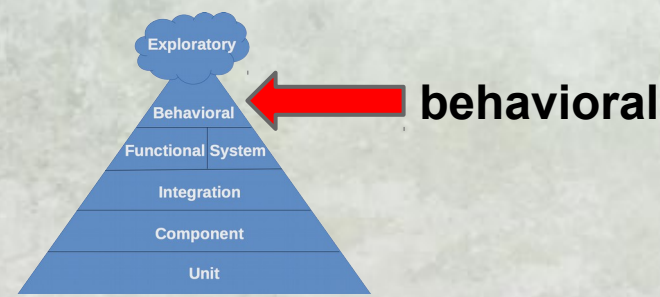
What's a "Functional Test?"

- Batch: A jobstream
- CICS: An end-to-end transaction, possibly skipping the UI and LINKing to the initial program



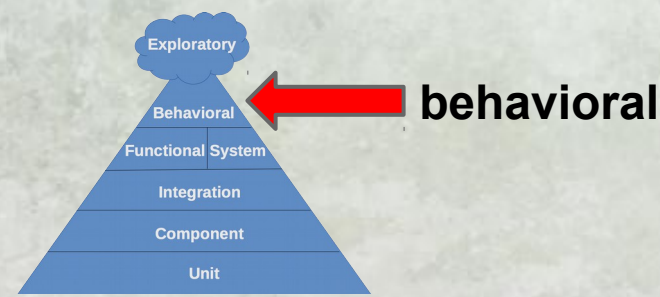
What's a "GUI Test?"

- Exercises an application through its user interface
- Also known as "behavioral test"
- Examples of representative interactions
- The basis of Acceptance Test Driven Development

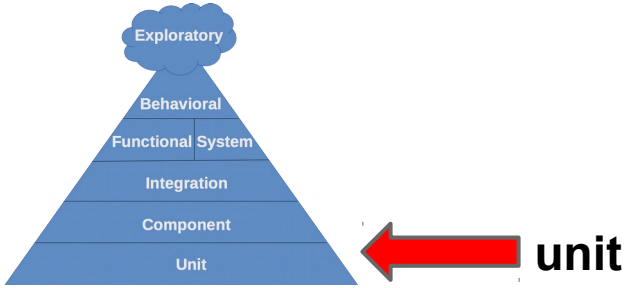


What's a "GUI Test?"

- Batch: N/A
- CICS: End-to-end test against the UI

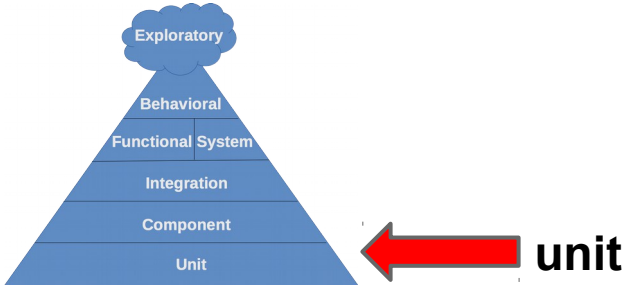


Unit


Exploratory
Behavioral
Functional
System
Integration
Component
Unit
unit

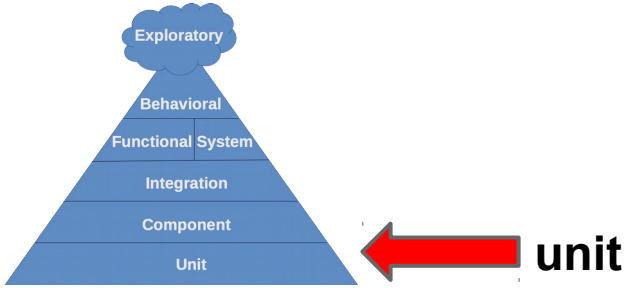
Mainframe Unit Testing On-Platform

- Java on USS
- Cobol Unit Test

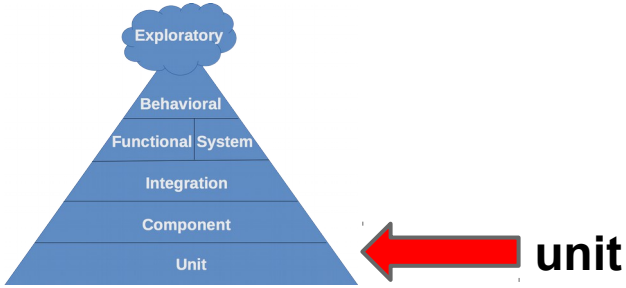


Mainframe Unit Testing Off-Platform

- IBM zUnit with RD&T and Greenhat
- Cobol Unit Test on Linux or Windows

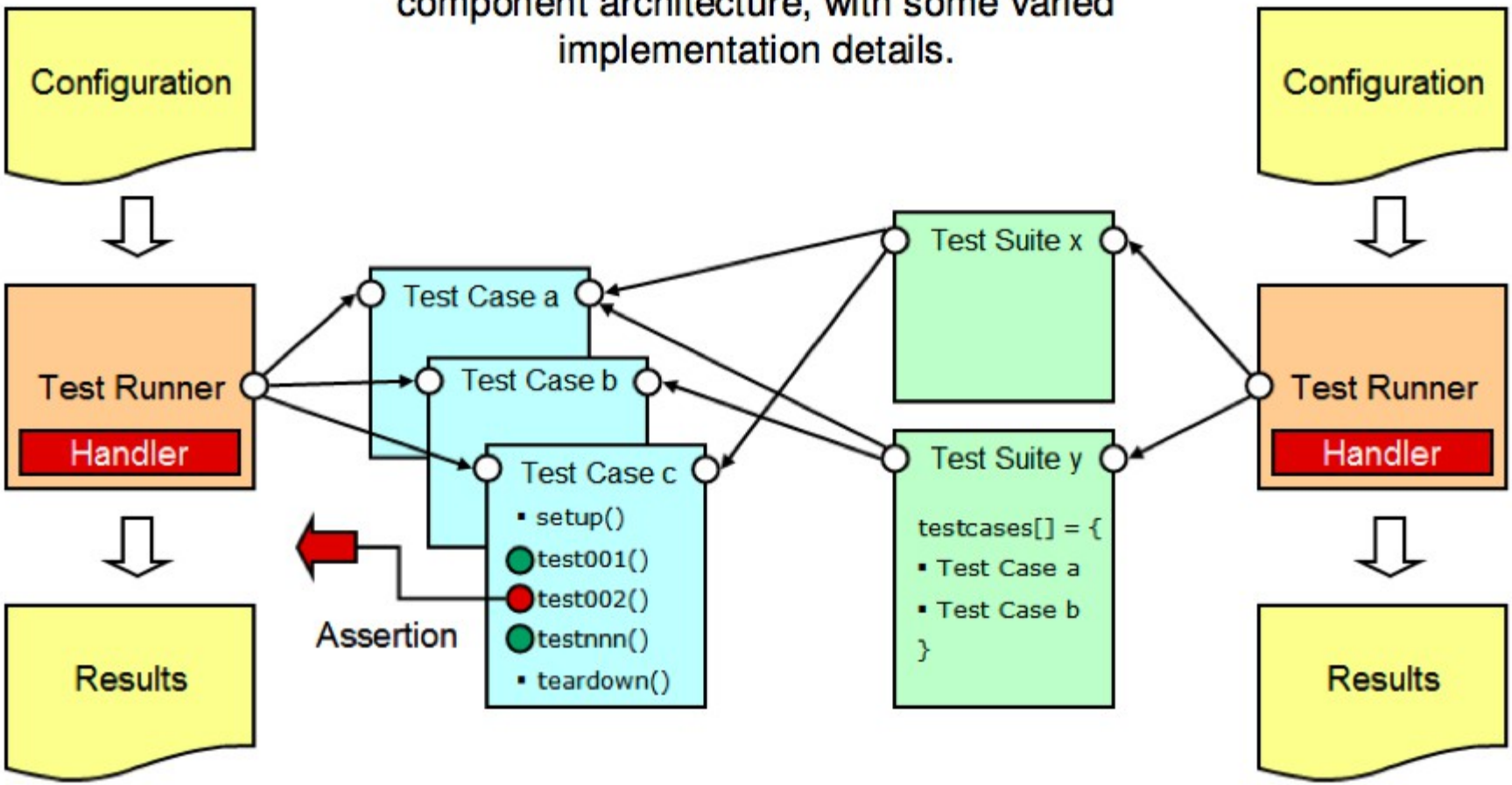


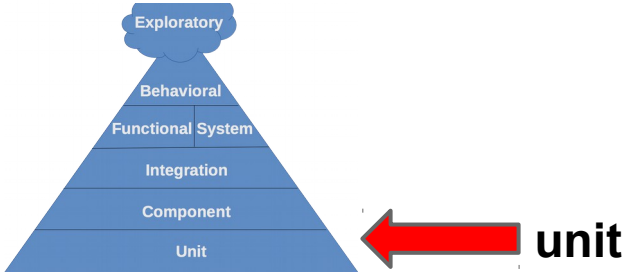
Ideally, unit testing requires no connectivity to external resources



Tooling: xUnit Architecture

All xUnit frameworks share the following basic component architecture, with some varied implementation details.

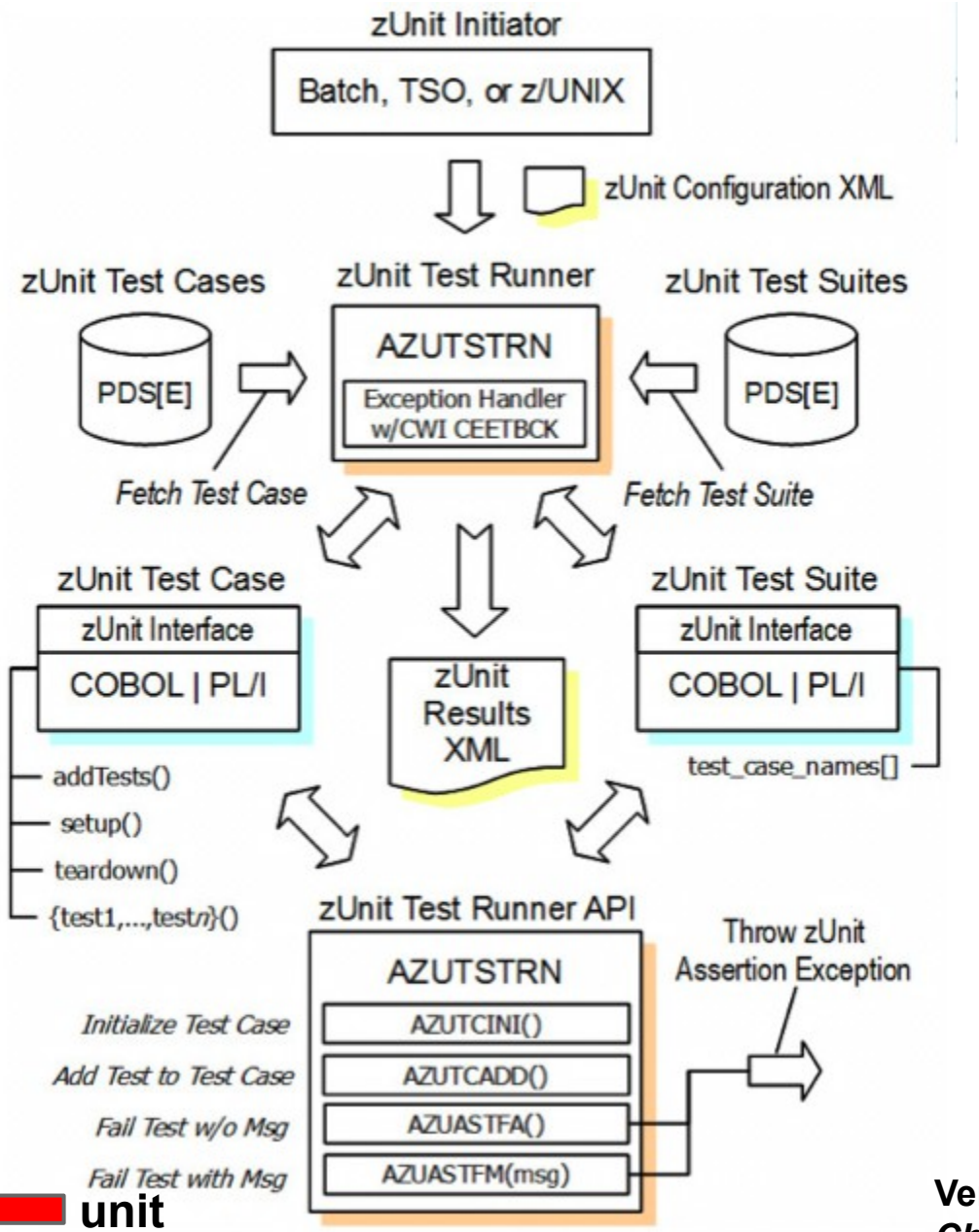


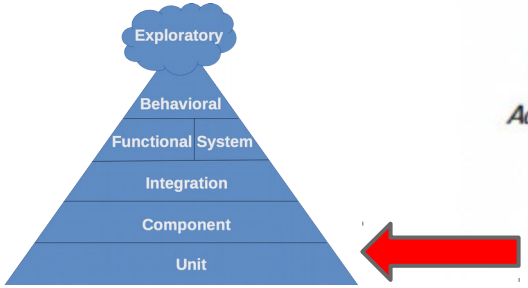


unit

Venkat Balabhadrapatruni
Chief Architect, RDz (IBM)

zUnit: A DSL for zOS unit testing





unit

Venkat Balabhadrapatruni
Chief Architect, RDz (IBM)

zUnit Implementation of xUnit

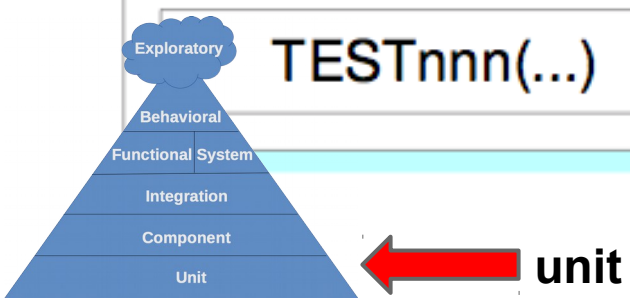
zUnit Test Case Module
USER.ZUNIT.LOAD(TESTCASE)

TESTCASE(...)
ADDTESTS(...)
SETUP(...)
TEARDOWN(...)
TEST001(...)
. . . . . .
TESTnnn(...)

zUnit Test Runner APIs
RDZ.SFEKSAMP(AZUTSTRX)

AZUTCINI(...)
AZUTCADD(...)
AZUASTFA(...)
AZUASTFM(...)

Invoking the assertion APIs, AZUASTFA or AZUASTFM, in the SETUP, TEARDOWN or active TEST entry will fail the current Test.



unit

Venkat Balabhadrapatruni
Chief Architect, RDz (IBM)

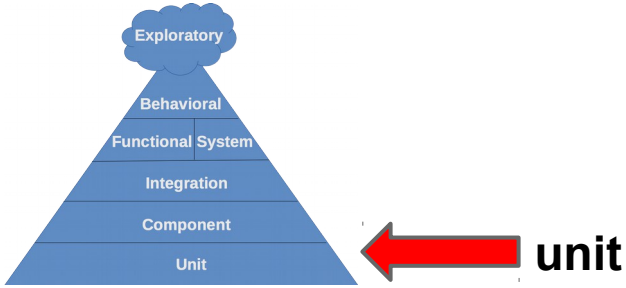
Sample zUnit Test Case for Cobol (1)

zUnit Test Case Module
USER.ZUNIT.LOAD(UNIT0001)

- UNIT0001(...)
- ADDTESTS(...)
- SETUP(...)
- TEARDOWN(...)
- TEST1(...)
- . . . . . .
- TESTn(...)

UNIT0001.cbl

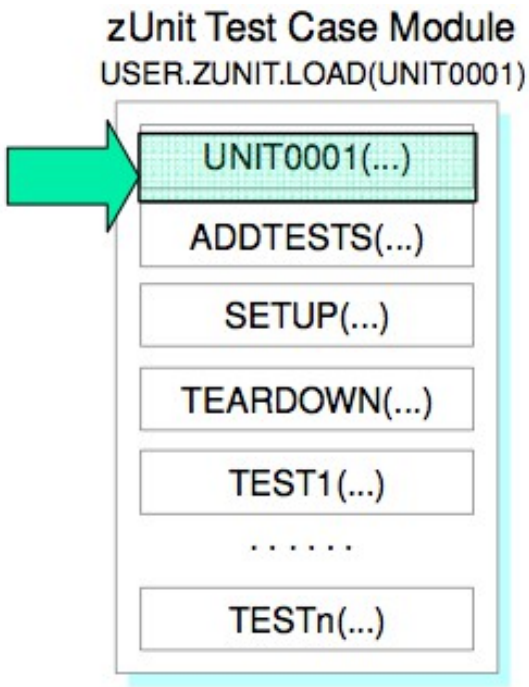
```
---+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+--
      *+-----------------------------------------------------------------+
      *| UNIT0001                                                        |
      *|     This program is immediately called by the IBM               |
      *|     z/OS Automated Unit Testing Framework (zUnit)               |
      *|     Test Runner to allow for initialization of the              |
      *|     Test Case. Upon return from this program, the               |
      *|     Test Runner will attempt to call the ADDTESTS               |
      *|     program.                                                    |
      *|                                                                 |
      *| @param TEST-CASE-PTR (input),                                   |
      *|     A pointer-by-value to an area maintained by the             |
      *|     zUnit Test Runner that identifies the Test Case             |
      *|     and associated resources.                                   |
      *|                                                                 |
      *| Note: this program does not require editing.                    |
      *+-----------------------------------------------------------------+
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'UNIT0001'.
       DATA DIVISION.
       PROCEDURE DIVISION USING BY VALUE TEST-CASE-PTR.
      * This program does not require editing.
           MOVE LENGTH OF CBLTESTC-ID TO CBLTESTC-ID-LEN
           MOVE LENGTH OF CBLTESTC-NAME TO CBLTESTC-NAME-LEN
           SET CBLTESTC-ADDTESTS TO ENTRY 'ADDTESTS'
           SET CBLTESTC-SETUP    TO ENTRY 'SETUP'
           SET CBLTESTC-TEARDOWN TO ENTRY 'TEARDOWN'
           CALL 'AZUTCINI' USING
               BY VALUE      TEST-CASE-PTR
               BY REFERENCE CBLTESTC-ID
               BY VALUE      CBLTESTC-ID-LEN
               BY REFERENCE CBLTESTC-NAME
               BY VALUE      CBLTESTC-NAME-LEN
               BY VALUE      CBLTESTC-ADDTESTS
               BY VALUE      CBLTESTC-SETUP
               BY VALUE      CBLTESTC-TEARDOWN
       END PROGRAM 'UNIT0001'.
```



unit

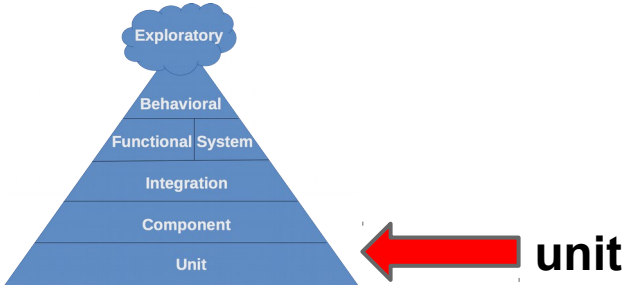
Venkat Balabhadrapatruni
Chief Architect, RDz (IBM)

Sample zUnit Test Case for Cobol (2)



UNIT0001.cbl

```
---+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+--
      *+-----------------------------------------------------------------+
      *| ADDTESTS                                                        |
      *|     This program is called by the zUnit Test Runner             |
      *|     to allow for adding Tests to the Test Case. Upon            |
      *|     return from this program, the Test Runner will              |
      *|     call the added Tests, surrounding each with calls           |
      *|     to the SETUP and TEARDOWN programs.                         |
      *|                                                                 |
      *| @param TEST-CASE-PTR (input),                                   |
      *|     A pointer-by-value to an area maintained by the             |
      *|     zUnit Test Runner that identifies the Test Case             |
      *|     and associated resources.                                   |
      *+-----------------------------------------------------------------+
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'ADDTESTS'.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       LINKAGE SECTION.
       PROCEDURE DIVISION USING BY VALUE TEST-CASE-PTR.
      * TODO: Add Tests to the Test Case.
           SET TEST-ENTRY TO ENTRY 'TEST1'
           MOVE 'TEST1' TO TEST-NAME
           MOVE 5 TO TEST-NAME-LEN
           CALL 'AZUTCADD' USING
               BY VALUE      TEST-CASE-PTR
               BY VALUE      TEST-ENTRY
               BY REFERENCE TEST-NAME
               BY VALUE      TEST-NAME-LEN
           SET TEST-ENTRY TO ENTRY 'TEST2'
           MOVE 'TEST2' TO TEST-NAME
           MOVE 5 TO TEST-NAME-LEN
           CALL 'AZUTCADD' USING
               BY VALUE      TEST-CASE-PTR
               BY VALUE      TEST-ENTRY
               BY REFERENCE TEST-NAME
               BY VALUE      TEST-NAME-LEN
           .
       END PROGRAM 'ADDTESTS'.
```



Venkat Balabhadrapatruni
Chief Architect, RDz (IBM)

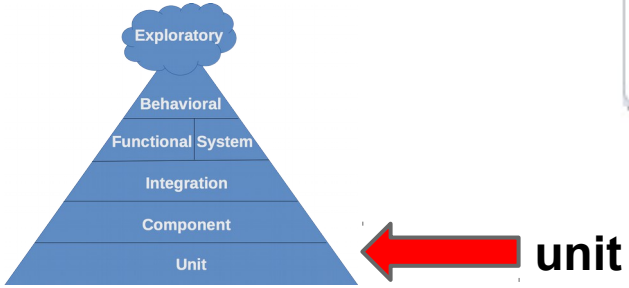
Sample zUnit Test Case for Cobol (3)

zUnit Test Case Module
USER.ZUNIT.LOAD(UNIT0001)

- UNIT0001(...)
- ADDTESTS(...)
- SETUP(...)
- TEARDOWN(...)
- TEST1(...)
- . . . . . .
- TESTn(...)

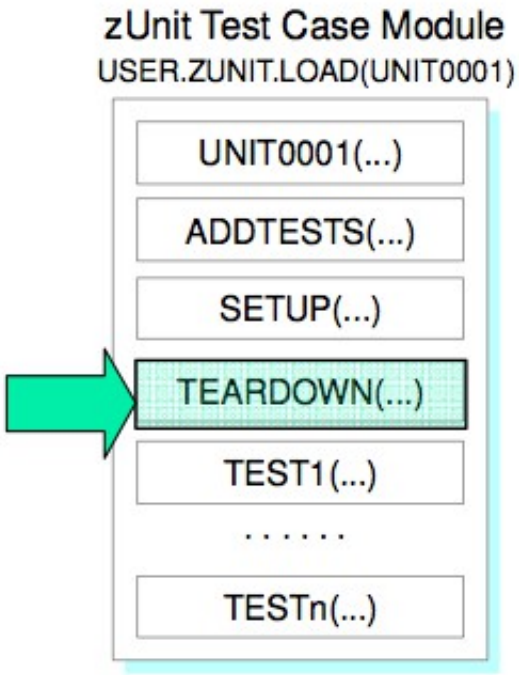
UNIT0001.cbl

```
---+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+--
      *+-----------------------------------------------------------------+
      *| SETUP                                                           |
      *|     This program is invoked by the zUnit Test Runner            |
      *|     prior to each Test to allow for allocation of               |
      *|     resources (e.g., memory, connections) that are              |
      *|     required to create the Test Fixture.                        |
      *|                                                                 |
      *| @param TEST-FIXTURE-PTR (output),                               |
      *|     A pointer-by-reference in which to store the address        |
      *|     of a user-defined structure that represents the Test        |
      *|     Fixture. References to all allocated resources should       |
      *|     be maintained in this structure so that they may be         |
      *|     accessed in the respective Test program, and released       |
      *|     in the TEARDOWN program.                                    |
      *|                                                                 |
      *| @param TEST-NAME-PTR (input),                                   |
      *|     A pointer-by-value to an area containing the name           |
      *|     of the Test for which a Test Fixture should be              |
      *|     allocated.                                                  |
      *+-----------------------------------------------------------------+
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'SETUP'.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       LINKAGE SECTION.
       PROCEDURE DIVISION USING BY VALUE TEST-CASE-PTR
                          BY REFERENCE TEST-FIXTURE-PTR
                          BY VALUE TEST-NAME-PTR
                          BY VALUE TEST-NAME-LEN.
           SET ADDRESS OF TEST-NAME TO TEST-NAME-PTR
           EVALUATE TEST-NAME(1:TEST-NAME-LEN)
              WHEN 'TEST1'
      * TODO: Allocate Test Fixture for Test 'TEST1'.
                DISPLAY 'SETUP (' TEST-NAME(1:TEST-NAME-LEN) ')'
              WHEN 'TEST2'
      * TODO: Allocate Test Fixture for Test 'TEST2'.
                DISPLAY 'SETUP (' TEST-NAME(1:TEST-NAME-LEN) ')'
           END-EVALUATE
           .
       END PROGRAM 'SETUP'.
```



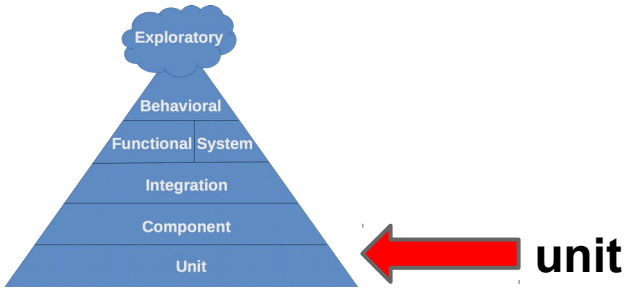
Venkat Balabhadrapatruni
Chief Architect, RDz (IBM)

Sample zUnit Test Case for Cobol (4)



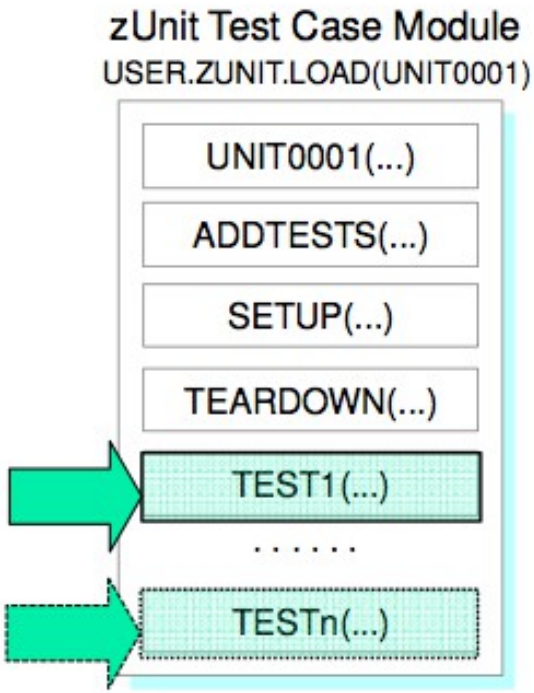
UNIT0001.cbl

```
*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+--
      *+-----------------------------------------------------------------+
      *| TEARDOWN                                                        |
      *|     This program is invoked by the zUnit Test Runner            |
      *|     after each Test to allow for releasing resources            |
      *|     (e.g., memory, connection) which were allocated             |
      *|     during creation of the Test Fixture in the SETUP            |
      *|     program.                                                    |
      *|                                                                 |
      *| @param TEST-FIXTURE-PTR (input),                                |
      *|     A pointer-by-value to a user-defined structure,             |
      *|     established previously in the SETUP program, that           |
      *|     represents the Test Fixture.                                |
      *|                                                                 |
      *| @param TEST-NAME-PTR (input),                                   |
      *|     A pointer-by-value to an area containing the name           |
      *|     of the Test for which a Test Fixture should be              |
      *|     allocated.                                                  |
      *+-----------------------------------------------------------------+
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'TEARDOWN'.
       DATA DIVISION.
       PROCEDURE DIVISION USING BY VALUE TEST-CASE-PTR
                          BY VALUE TEST-FIXTURE-PTR
                          BY VALUE TEST-NAME-PTR
                          BY VALUE TEST-NAME-LEN.
           SET ADDRESS OF TEST-NAME TO TEST-NAME-PTR
           EVALUATE TEST-NAME(1:TEST-NAME-LEN)
              WHEN 'TEST1'
       * TODO: Free Test Fixture for Test 'TEST1'.
                DISPLAY 'TEARDOWN (' TEST-NAME(1:TEST-NAME-LEN) ')'
              WHEN 'TEST2'
       * TODO: Free Test Fixture for Test 'TEST2'.
                DISPLAY 'TEARDOWN (' TEST-NAME(1:TEST-NAME-LEN) ')'
           END-EVALUATE
           .
       END PROGRAM 'TEARDOWN'.
```



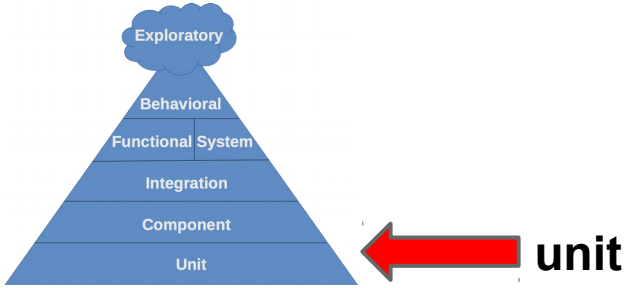
Venkat Balabhadrapatruni
Chief Architect, RDz (IBM)

Sample zUnit Test Case for Cobol (5)



UNIT0001.cbl

```
---+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+--
      *+-------------------------------------------------------------+
      *| TEST1                                                       |
      *|     A Test (supply more detail).                            |
      *+-------------------------------------------------------------+
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'TEST1'.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       DATA DIVISION.
       FILE SECTION.
         FD TEST-DATA-FILE RECORD CONTAINS 133 CHARACTERS
            DATA RECORD IS TEST-DATA-RECORD
            RECORDING MODE IS F.
         1 TEST-DATA-RECORD PIC X(133).
       WORKING-STORAGE SECTION.
       LINKAGE SECTION.
       PROCEDURE DIVISION USING BY VALUE TEST-CASE-PTR
                          BY VALUE TEST-FIXTURE-PTR
                          BY VALUE TEST-NAME-PTR
                          BY VALUE TEST-NAME-LEN.
      * 1. display the current test name
           SET ADDRESS OF TEST-NAME TO TEST-NAME-PTR
           DISPLAY TEST-NAME(1:TEST-NAME-LEN) ' STARTED'
      * 2. try to open the test data file, fail if unable to
           OPEN INPUT TEST-DATA-FILE
           IF TEST-DATA-STATUS NOT = '00'
             MOVE 1 TO FAIL-MESSAGE-LEN
             STRING 'FAILED TO OPEN DD:TEST-DATA, STATUS='
                    TEST-DATA-STATUS '.'
                 DELIMITED BY SIZE INTO FAIL-MESSAGE-TXT
                 WITH POINTER FAIL-MESSAGE-LEN
             END-STRING
             SUBTRACT 1 FROM FAIL-MESSAGE-LEN
             CALL 'AZUASTFM' USING BY VALUE TEST-CASE-PTR
                                   BY REFERENCE FAIL-MESSAGE-TXT
                                   BY VALUE FAIL-MESSAGE-LEN
           END-IF
      * 3. call program (functional unit) using test data
      * 4. compare output (actual vs expected)
      * 5. fail test if actual output NE expected output.
           .
       END PROGRAM 'TEST1'.
```



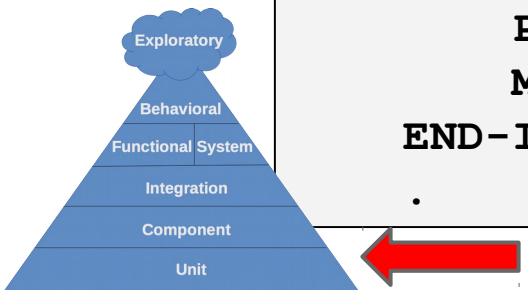
Venkat Balabhadrapatruni
Chief Architect, RDz (IBM)

What a clumsy DSL feels like

cobol-unit-test: A DSL for unit testing Cobol

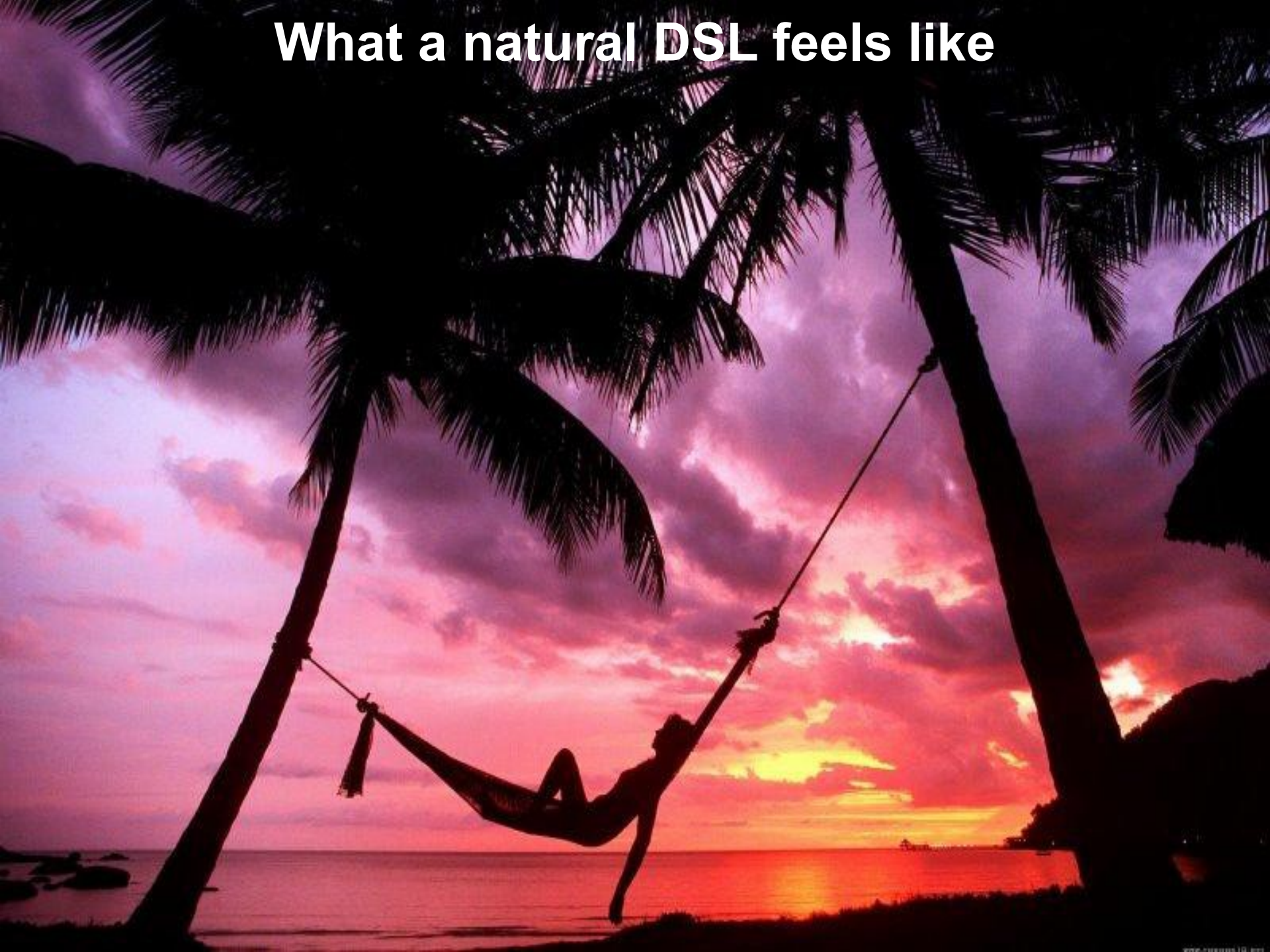
```
TESTSUITE 'Convert CSV file to fixed format'

TESTCASE 'It converts text field 1 to upper case'
    MOVE 'something' TO TEXT-VALUE-1
    PERFORM 2100-CONVERT-TEXT-FIELD-1
    EXPECT TEXT-OUT-1 TO BE 'SOMETHING'

TESTCASE 'It handles empty text field 1'
    MOVE LOW-VALUES TO TEXT-VALUE-1
    PERFORM 2100-CONVERT-TEXT-FIELD-1
    EXPECT TEXT-OUT-1 TO BE SPACES
```

```
2100-CONVERT-TEXT-FIELD-1.
    IF TEXT-VALUE-1 = LOW-VALUES
        MOVE SPACES TO TEXT-OUT-1
    ELSE
        MOVE TEXT-VALUE-1 TO TO-UPPER-CASE
        PERFORM 9000-TO-UPPER-CASE
        MOVE TO-UPPER-CASE TO TEXT-OUT-1
    END-IF
    .
```

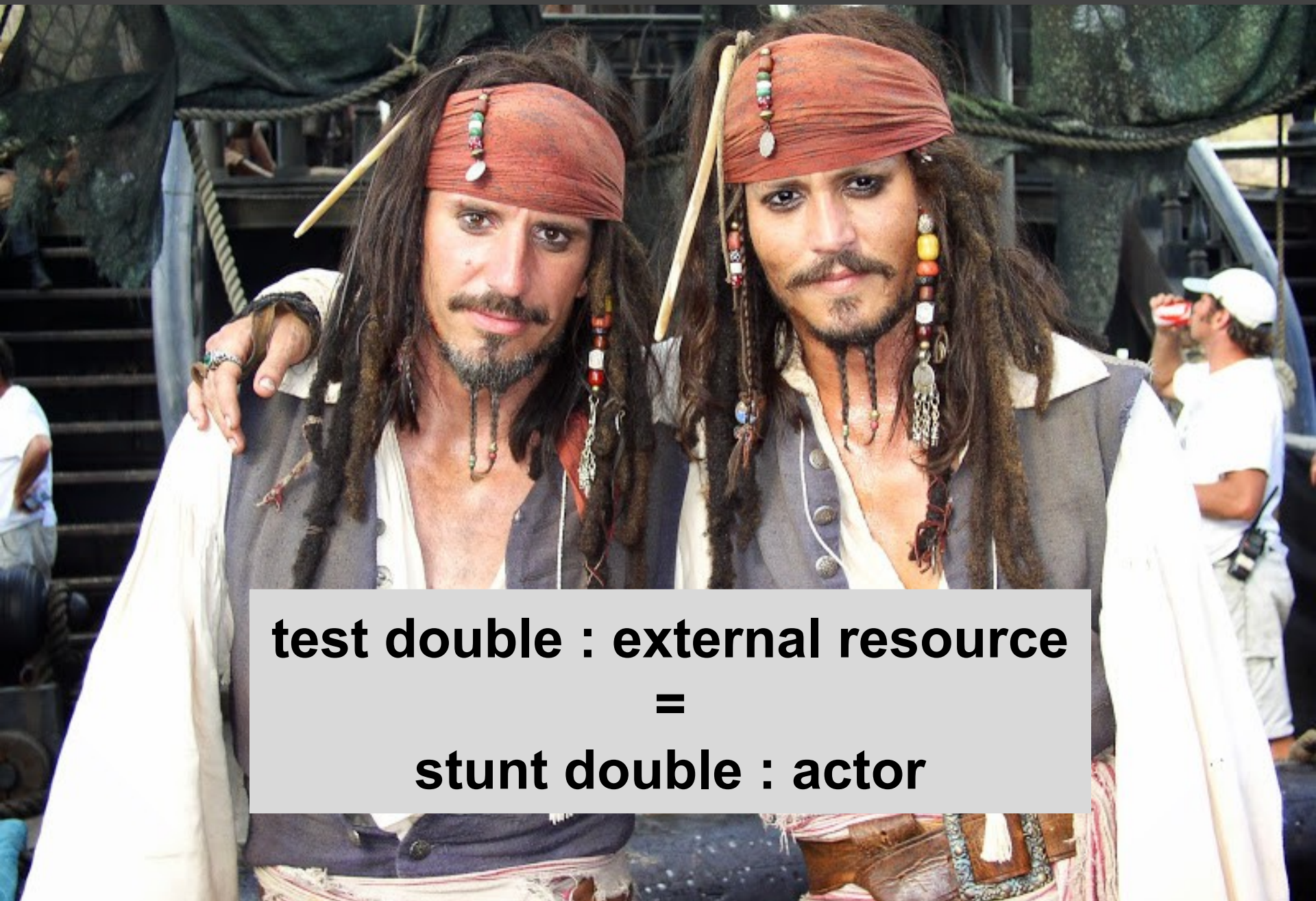


unit

What a natural DSL feels like



What are “test doubles?”

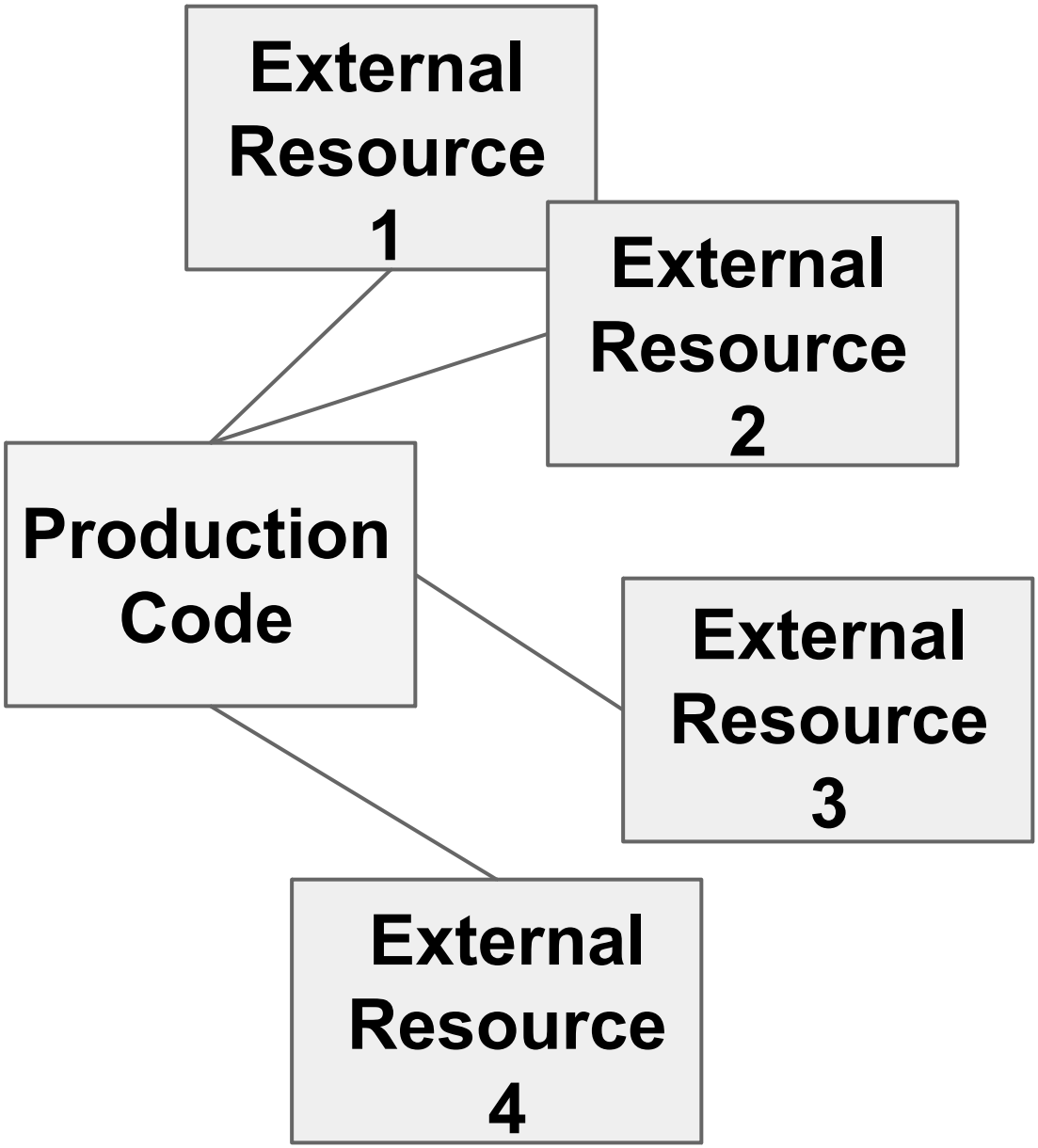


test double : external resource

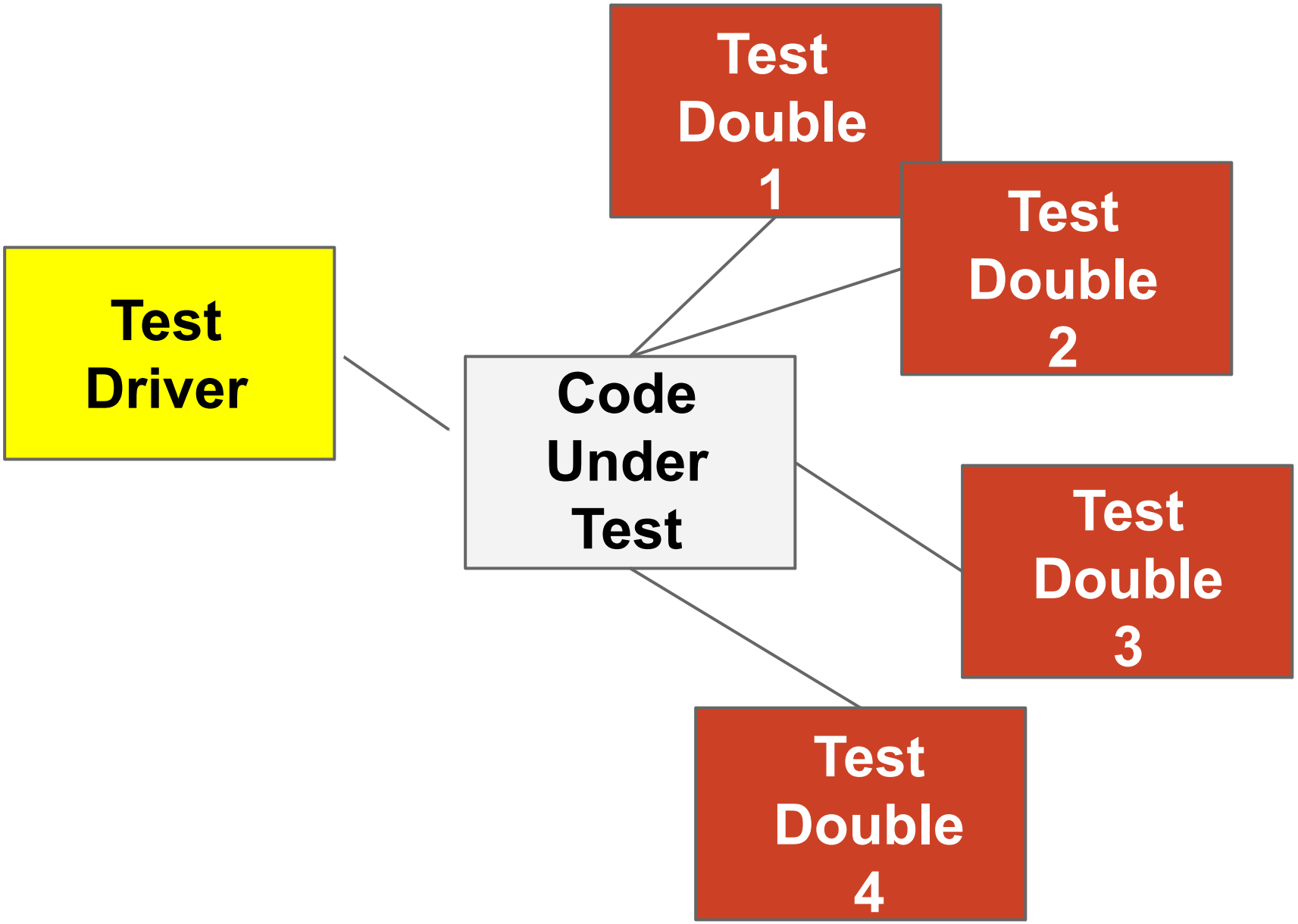
=

stunt double : actor

Everything is Real



Only the Code Under Test is Real



Test Doubles

Stub - returns lowest-common-denominator result based on the type of resource it represents

Mock - returns a value specified in the test case; keeps track of how many times it was accessed

Virtualized Service - mimics the behavior of a service interface

Simulator or **emulator** - a custom implementation of a resource intended to support testing or experimentation

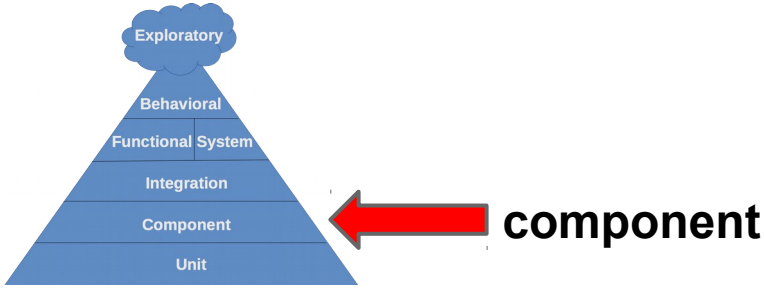
External Resources of COBOL Programs

- Files (batch)
- Paragraphs
- CALLs

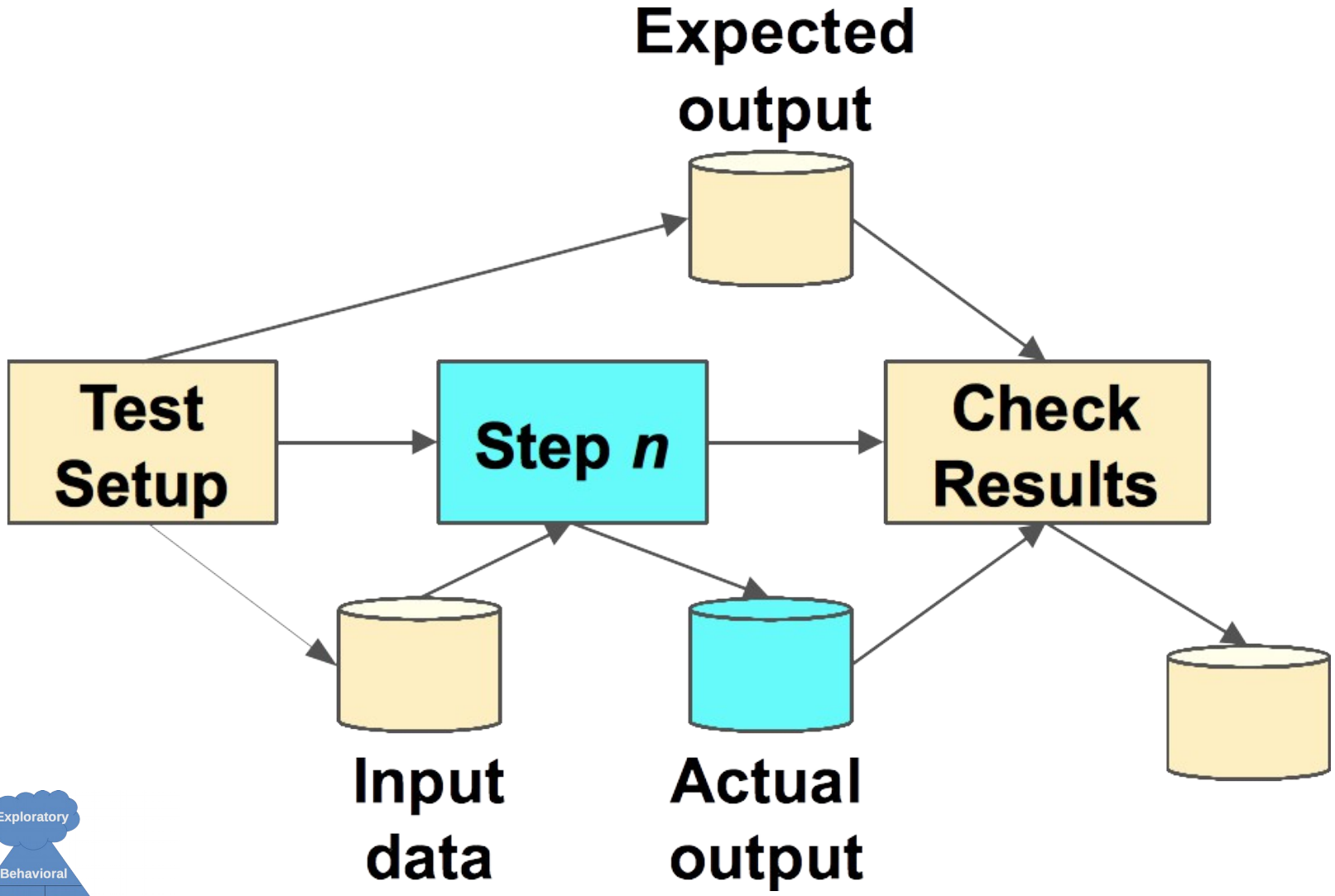
  this covers MQI and XMS

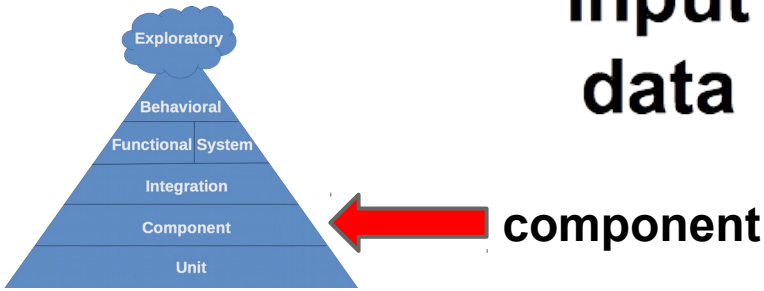
- EXEC CICS commands
- EXEC SQL commands

Component


Exploratory
Behavioral
Functional
System
Integration
Component
Unit
component

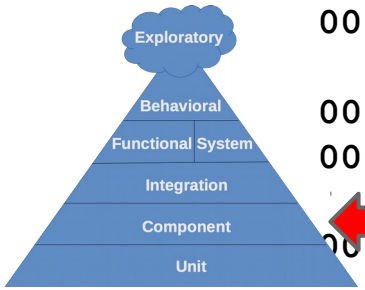
Component Test (Batch)





Batch Component Test: Setup

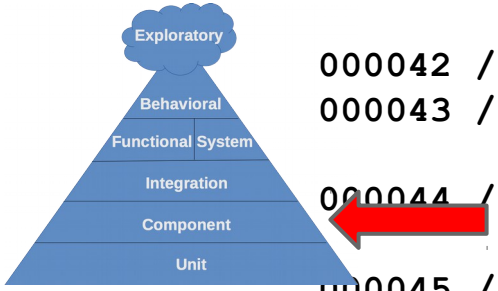
```
000001 //UMGP409A JOB (15901,TEST,000000),'MOB TDD JCL',
000002 //        CLASS=6,MSGCLASS=X,NOTIFY=&SYSUID
000003 //         SET HLQ=TPS
000004 //         SET ENV=T01
000005 //         SET FA=B1
000006 //         SET ODATE=141006
000007 //         SET JOBNAME=PPSTDDT1
000008 //         SET DB2ENV=TDB2
000009 //*****************************************
000010 //PROCLIB JCLLIB ORDER=(TPS.UMGP409.MOB.PRC,
000011 //        PRD1.BNVSM.PROCFIX,
000012 //        PRD1.BNVSM.PROCLIB)
000013 //*****************************************
000014 //JOBLIB   DD  DSN=TPS.UMGP409.UMM.P2.LOD,
000015 //                DISP=SHR
```



component

Batch Component Test: Test Case

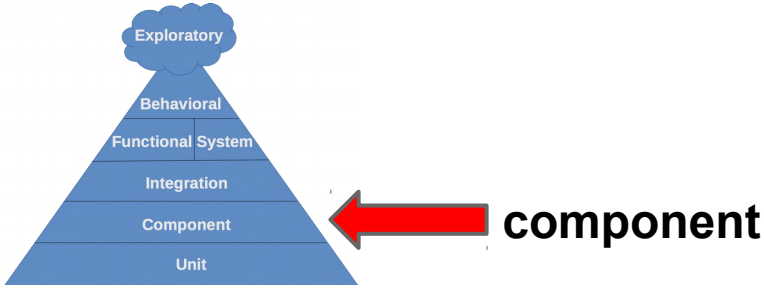
```
000026 //****************************************
000027 //*                  DELETE ERROR REPORT
000028 //*****************************************
000029 //CLRFRPT  EXEC PTSADELF,

000030 //         DSN=&HLQ..&ENV..&FA..D.&JOBNAME..FAIL.REPORT
000032 //*****************************************
000033 //*                   TEST CASE 01
000034 //*****************************************
000035 //JSDELDF  EXEC PTSADELF,

000036 //          DSN=&HLQ..&ENV..&FA..D.&JOBNAME..TC001.SYSOUT

000037 //*****************************************
000038 //TC001    EXEC PPSRCVRT,

000039 //         ODATE=&ODATE,

000040 //         ACHRECV=TPS.UMGP409.ACH.EMPTY,

000041 //         RESULT=&HLQ..&ENV..&FA..D.&JOBNAME..TC001.SYSOUT

000042 //****************************************
000043 //SUPTC001 EXEC SUPERCF,

000044 //         RESULT=&HLQ..&ENV..&FA..D.&JOBNAME..TC001.SYSOUT,

000045 //         REFDD=TPS.UMGP409A.TC001.SYSOUT.REF
```



component

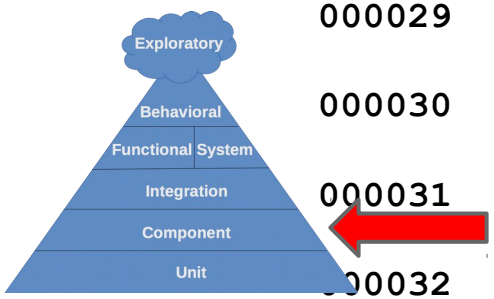
Batch Component Test: Wrapping Up

```
000152 //*************************************************
000153 //* AT LAST - WHEN ALL TEST CASES SUCCESSFULLY PASSED
000154 //*************************************************
000155 //PPSATLST EXEC PPSATLST

000156 //EMYRPT.SYSIN DD *

000157    LISTCAT ENTRIES('TPS.T01.B1.D.PPSTDDT1.FAIL.REPORT') ALL

000158 /*

000159 //IFDNEXT IF PPSATLST.EMYRPT.RC GT 0 THEN

000160 //PPSSUCCS EXEC PPSSUCCS,

000161 //        RPTDSN=TPS.T01.B1.D.PPSTDDT1.FAIL.REPORT

000162 //IFDNEXTE ENDIF

000163 //
```



Batch Component Test: Test Report (1)

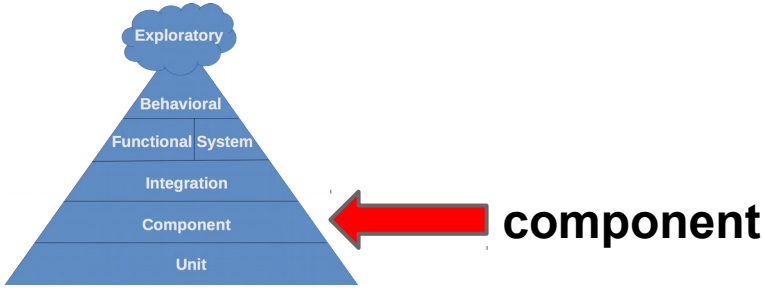
```
000001 /* REXX */

000020 PULL OUTDSN TTDTESTCASE

000021 IF SYSDSN(''"OUTDSN"'") = 'OK' THEN

000022 DO

000023      SAY 'OUTDSN IN OK' OUTDSN

000024     "ALLOC F(EXISTS) DS('"OUTDSN"') REUSE SHR"

000025     "EXECIO 0 DISKR EXISTS (OPEN"

000026     "EXECIO * DISKR EXISTS (STEM XX."

000027     "EXECIO 0 DISKR EXISTS (FINIS"

000028     "FREE F(EXISTS)"

000029      I = 1

000030      M = XX.0 + 1

000031      SAY 'XX.0' XX.0

000032     DO WHILE I <= XX.0
```



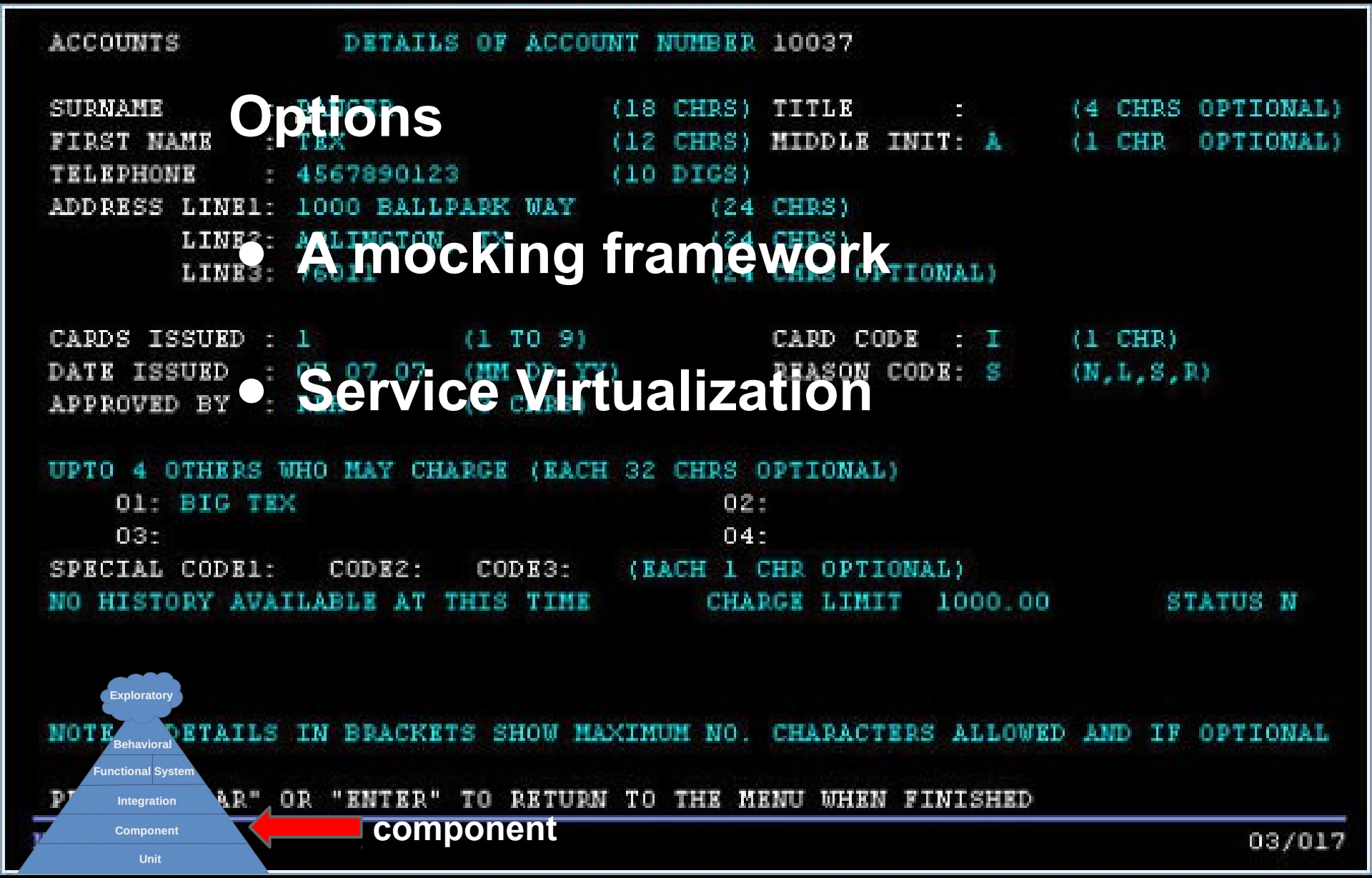
component

Batch Component Test: Test Report (2)

```
        ...
000041 ELSE

000042      SAY 'OUTDSN IN NOT OK' OUTDSN

000043     "ALLOC F(OUT) DS('"OUTDSN"') NEW MOD SPACE(50,20)

000044      DSORG(PS) RECFM(F,B) LRECL(80) BLKSIZE(0)"

000045      OUTVR.1 = TTDTESTCASE

000046     "EXECIO * DISKW OUT (STEM OUTVR. FINIS"

000047     "FREE FILE(OUT)"

000048 EXIT 00
```



CICS Component Testing



- Options
- A mocking framework
- Service Virtualization

component

CICS Component Testing

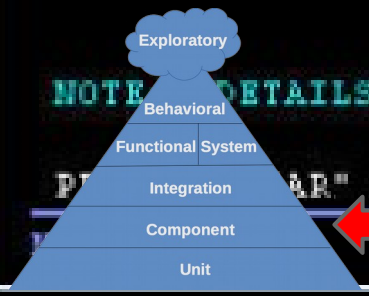
- Mocking framework

Pros

- Lightweight
- Fast
- Easy to use
- No additional resources needed

Cons

- No such beast exists - must write it



component

CICS Component Testing

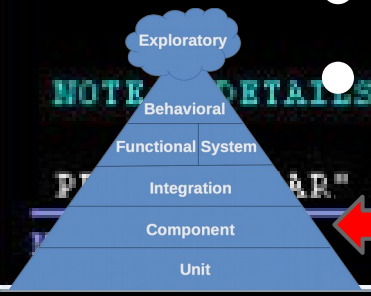
Service Virtualization

Pros

- Configurable
- Can learn existing interactions
- Production-like environment

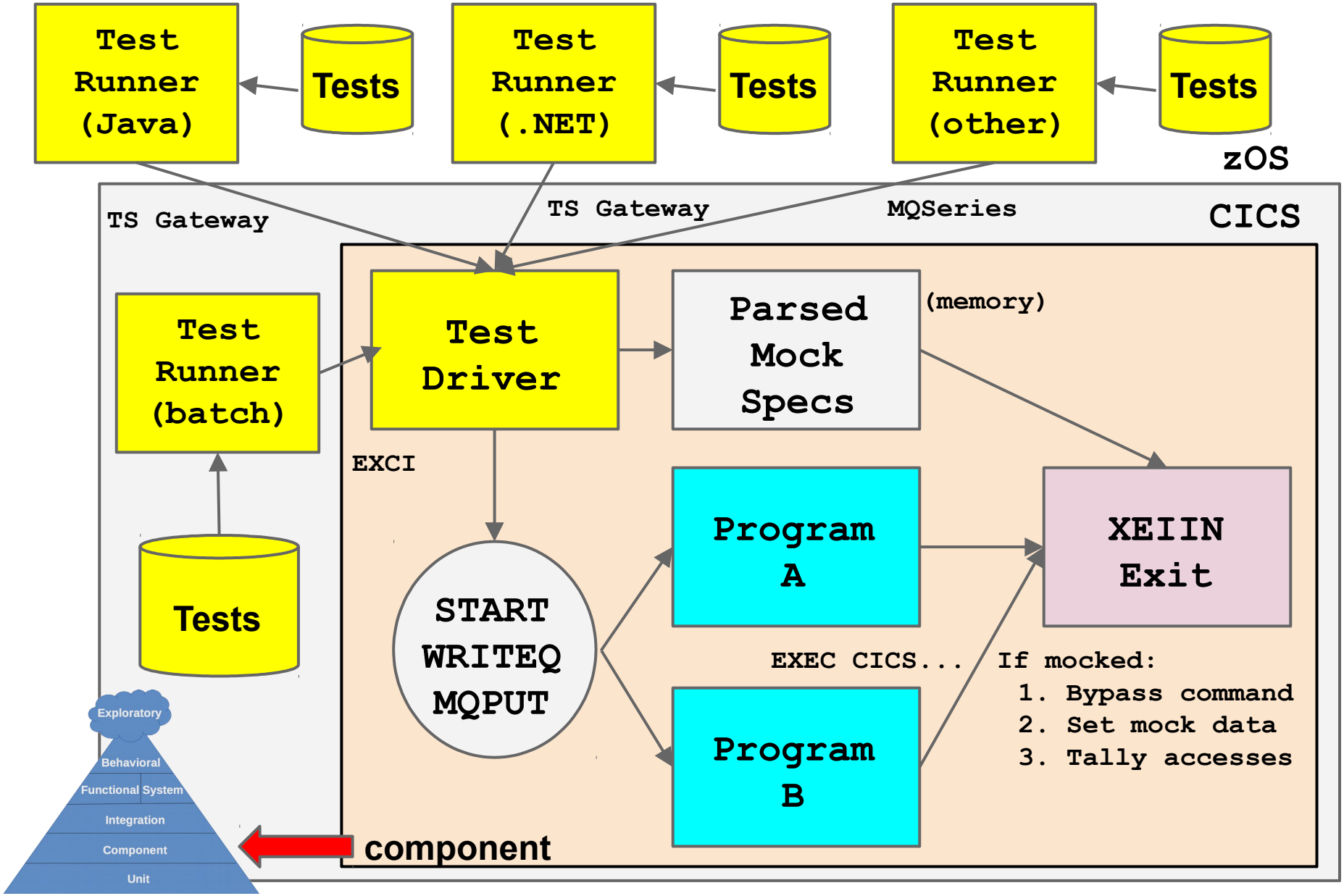
Cons

- Heavyweight - much setup needed
- Multiple servers needed
- Multiple points of failure
- Third-party products ($$$)
- More suited to functional testing

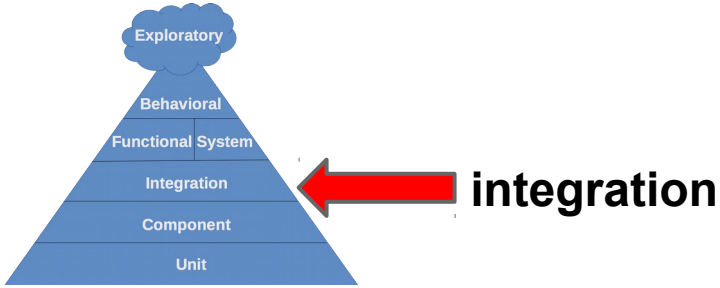


component

CICS Component Testing (Proposed)

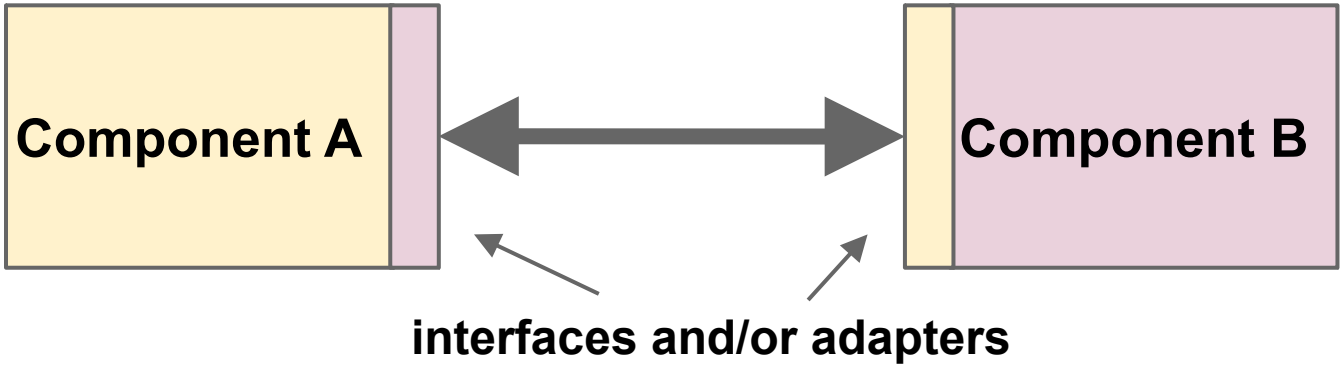


Integration

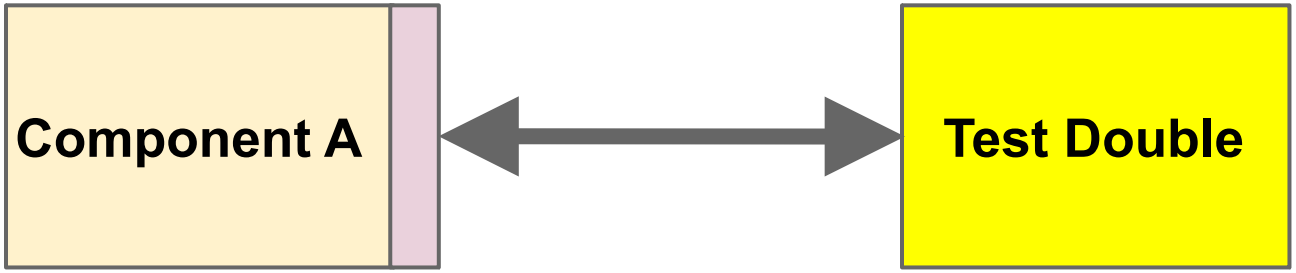


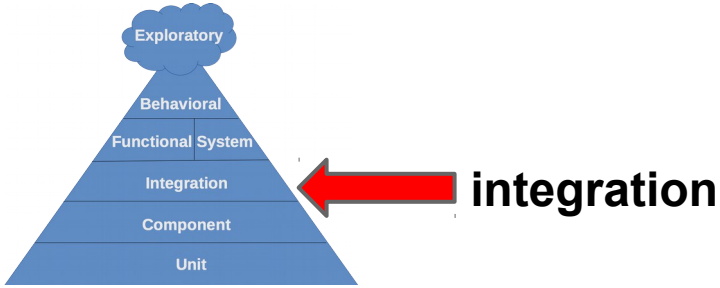
Integration Test (General)

Production

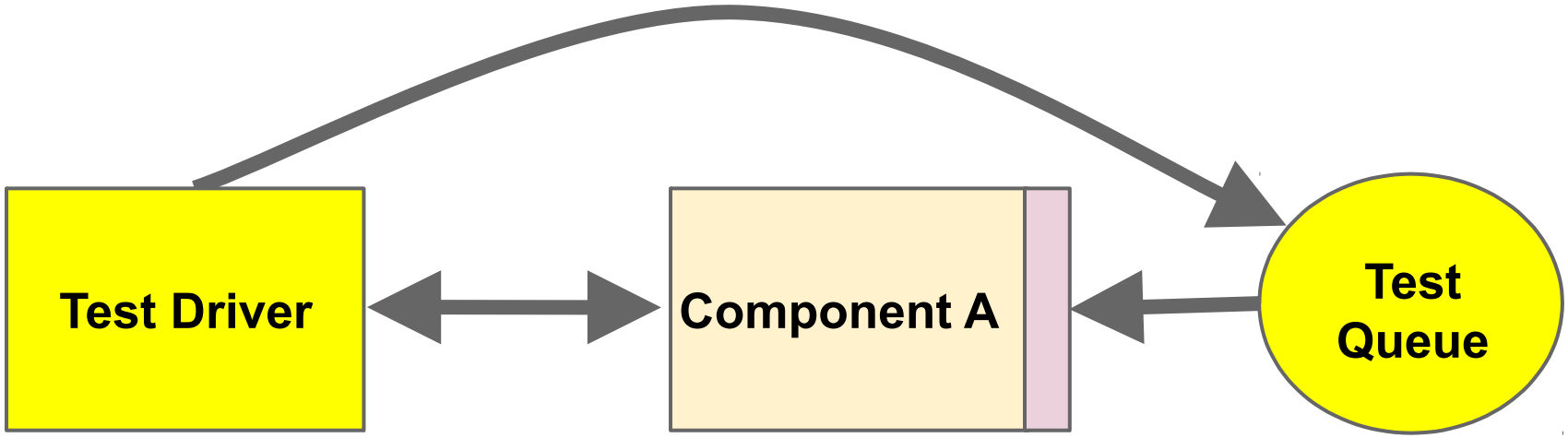


Does Component A handle everything defined in Component B's interface?

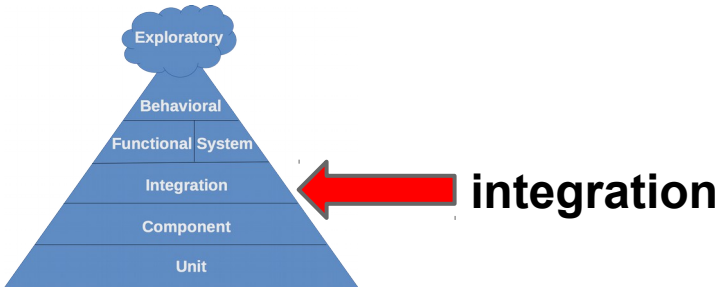




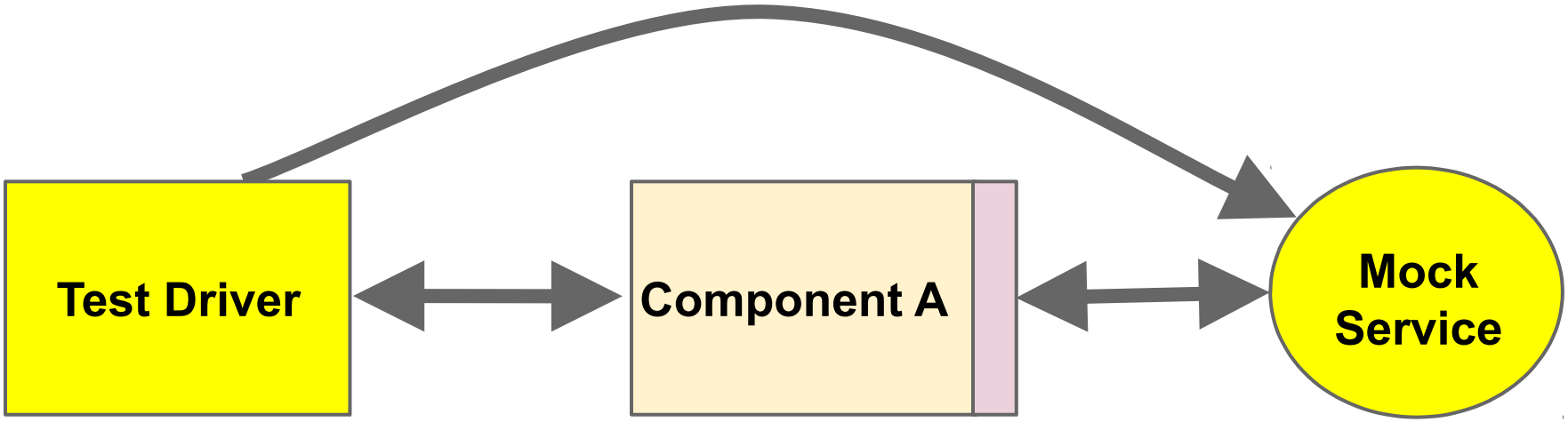
Integration Test (MQSeries)



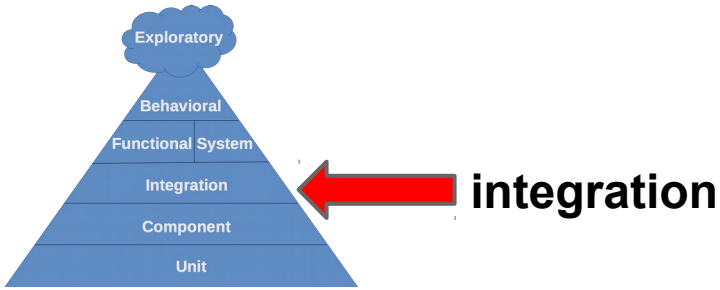
- Component A knows how to retrieve data from a queue
- Component A knows how to handle all the inputs that are defined for this interface



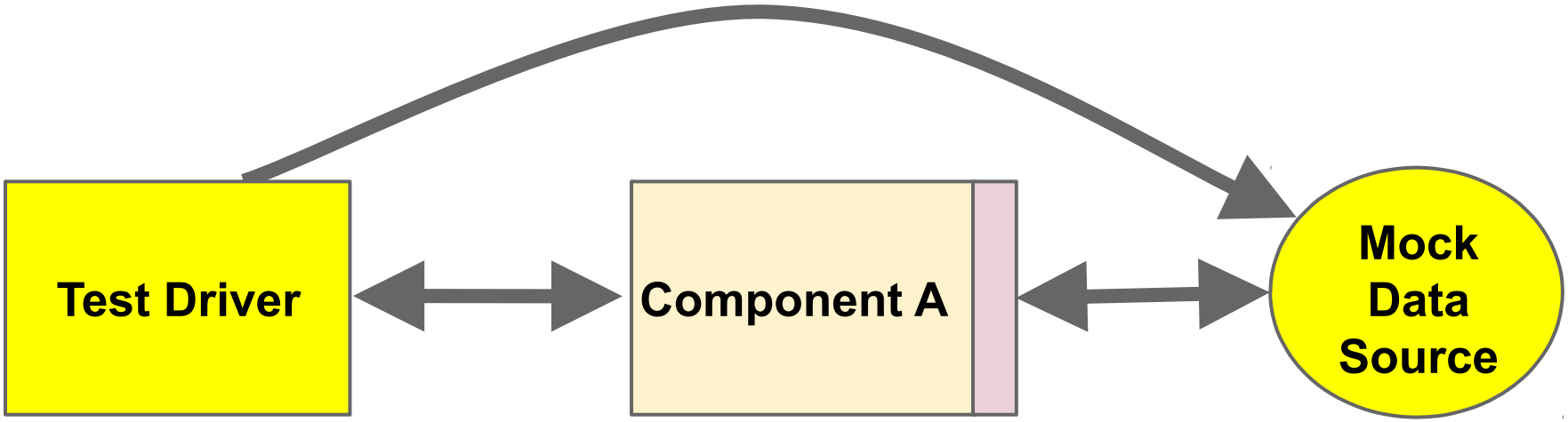
Integration Test (API)



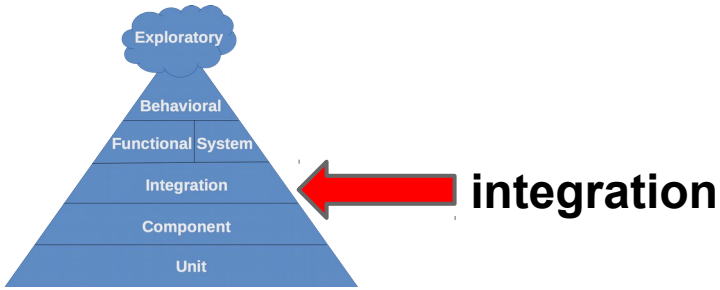
- Component A knows how to interact with the defined API
- Component A knows how to handle all the inputs that are defined for this interface



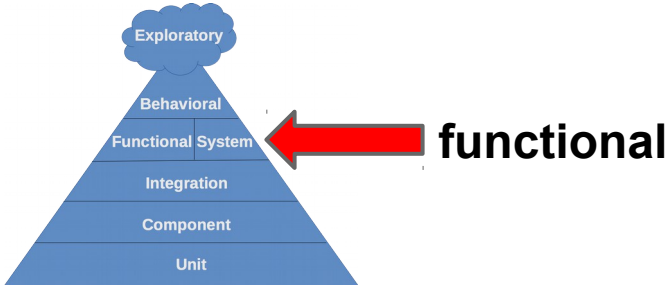
Integration Test (Third-party Supplier)



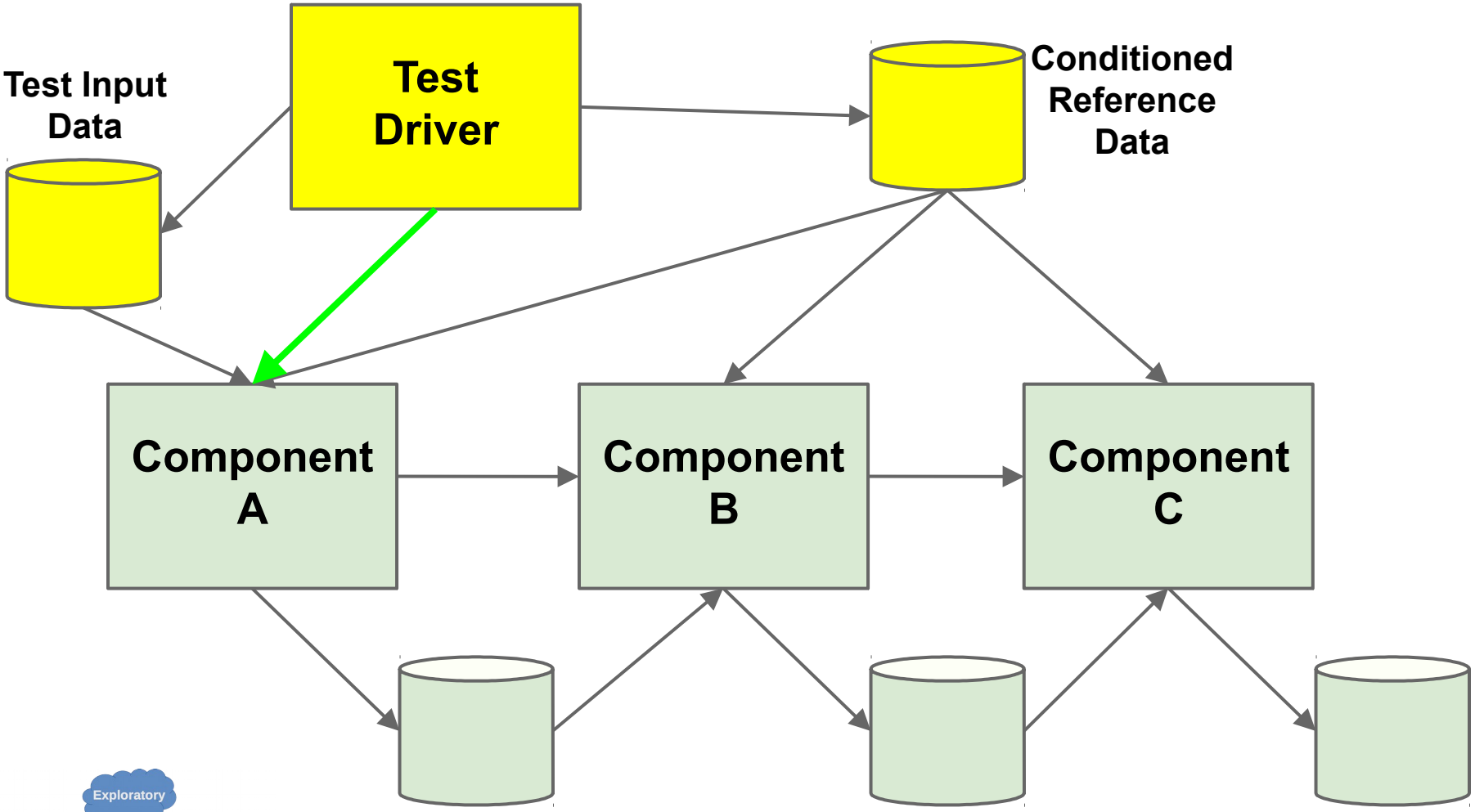
- Component A knows how to interact with the defined data feed
- Component A knows how to handle all the inputs that are defined for this interface

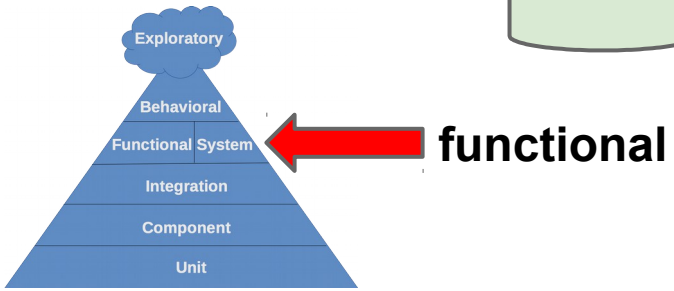


Functional

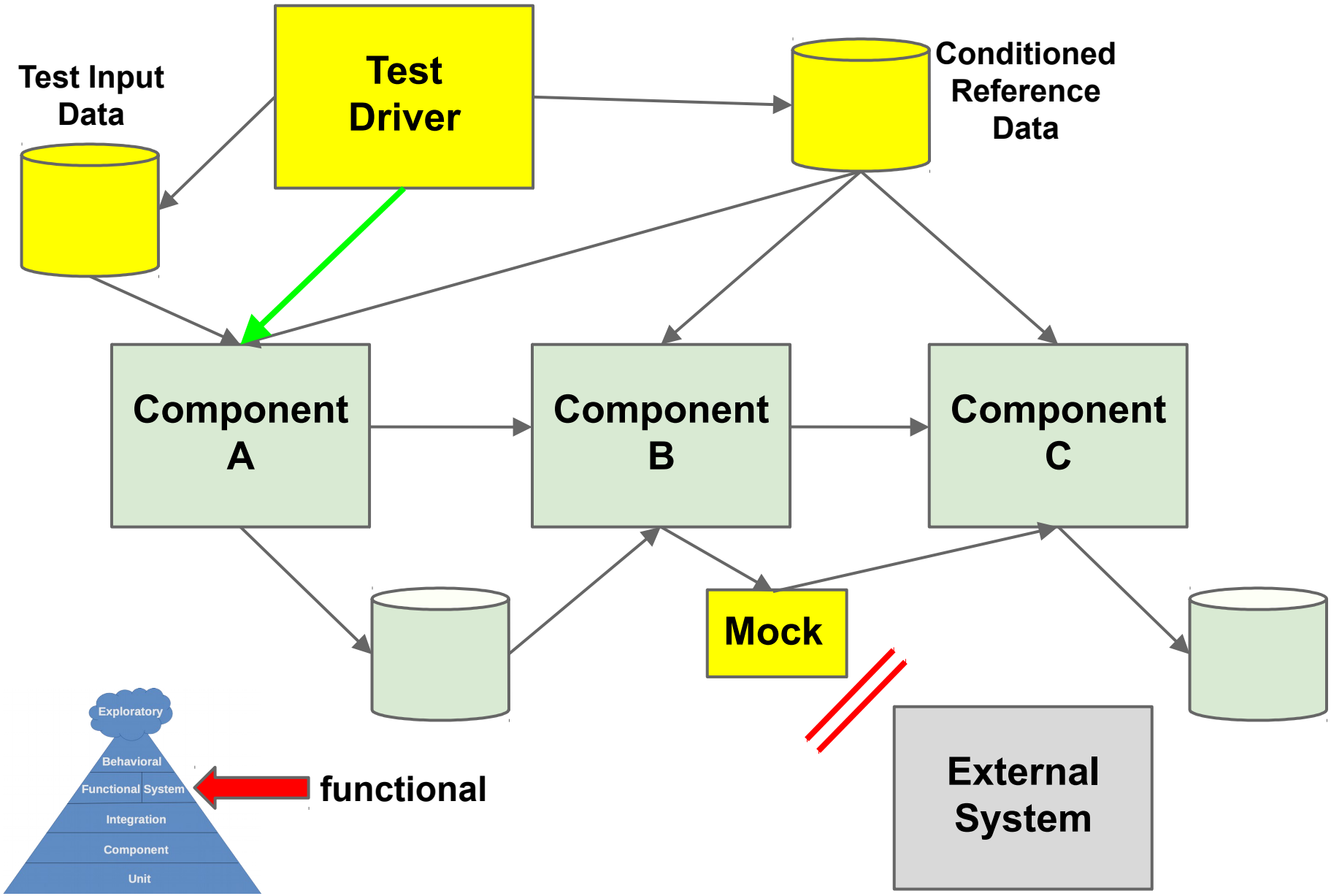

Exploratory
Behavioral
Functional
System
Integration
Component
Unit
functional

Functional Test (General)

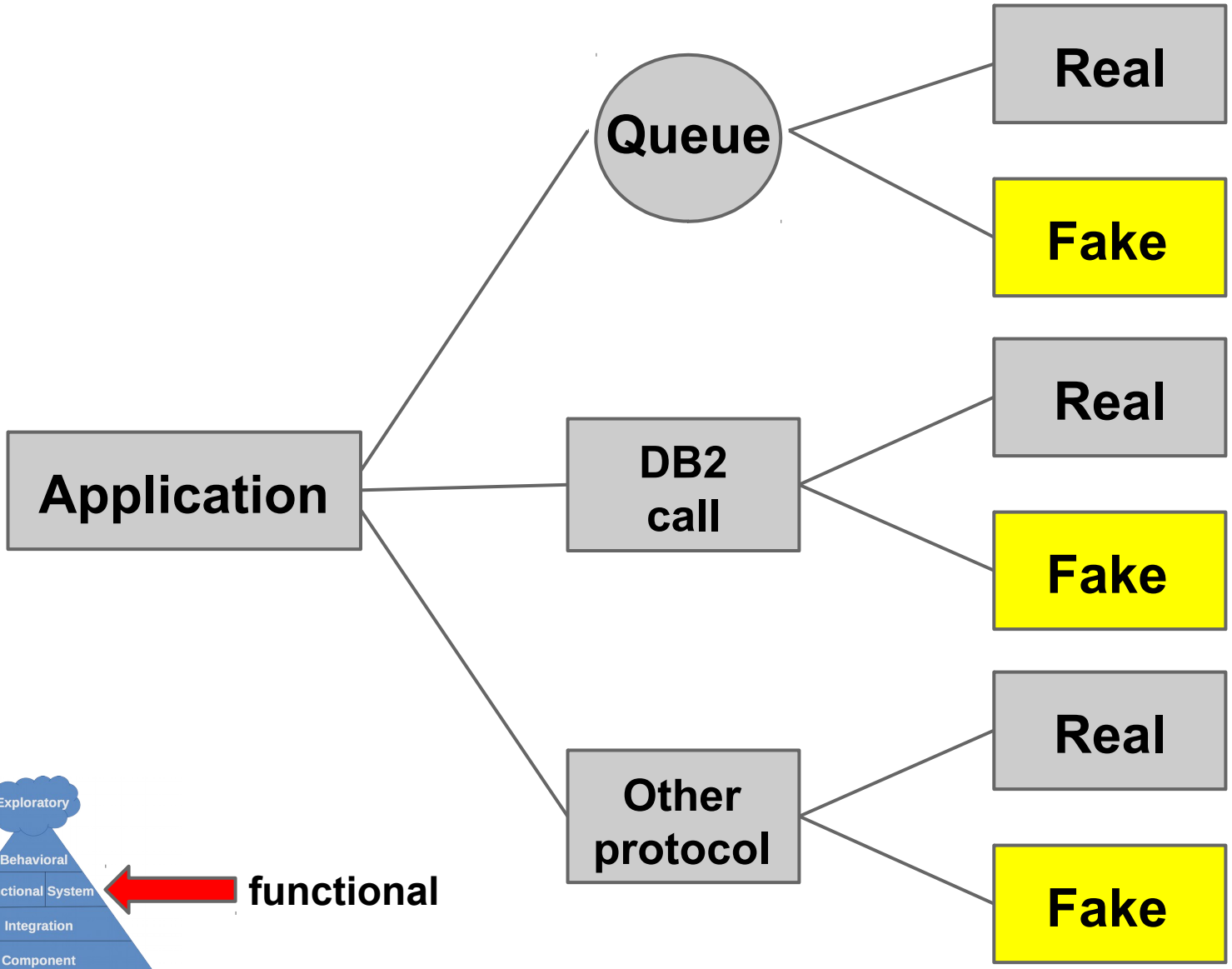


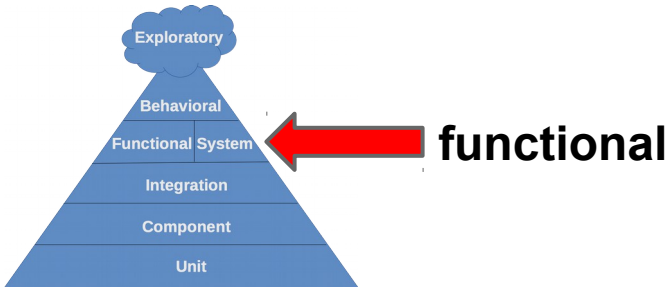


Functional Test with External Dependencies



Service Virtualization

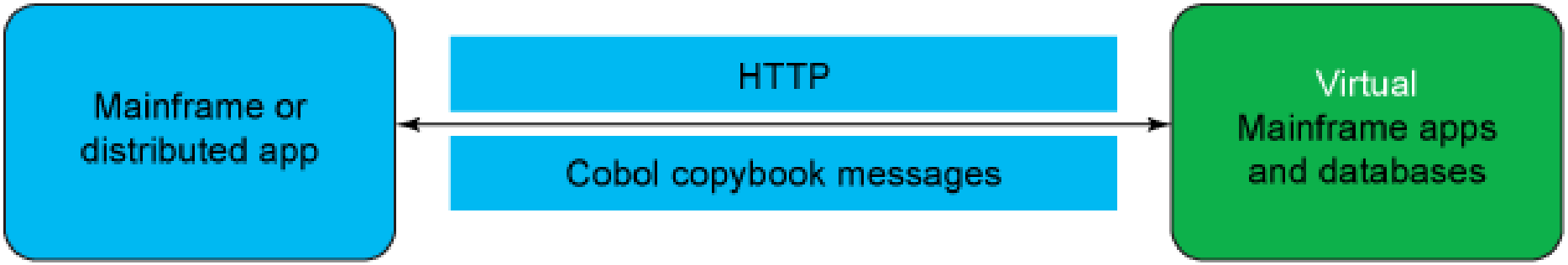




IBM Service Virtualization

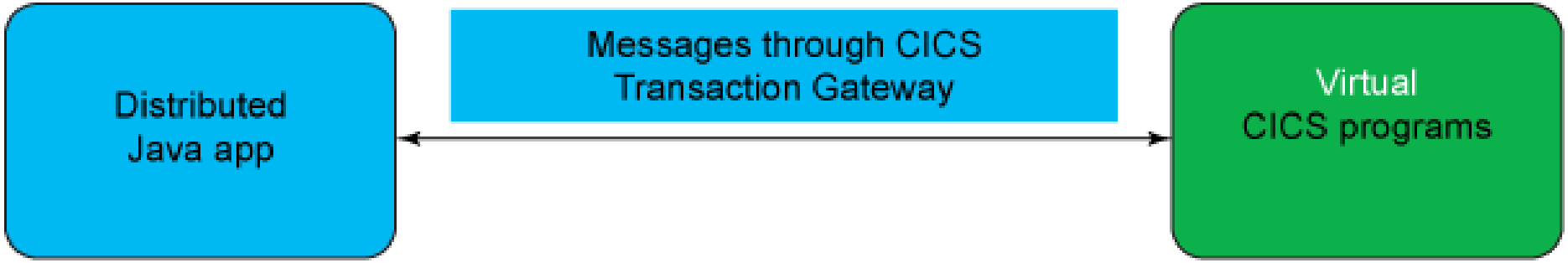
Scenario 1:

Mainframe or distributed app calls CICS program with HTTP



Scenario 2:

Distributed application calls CICS program through CICS Transaction Gateway

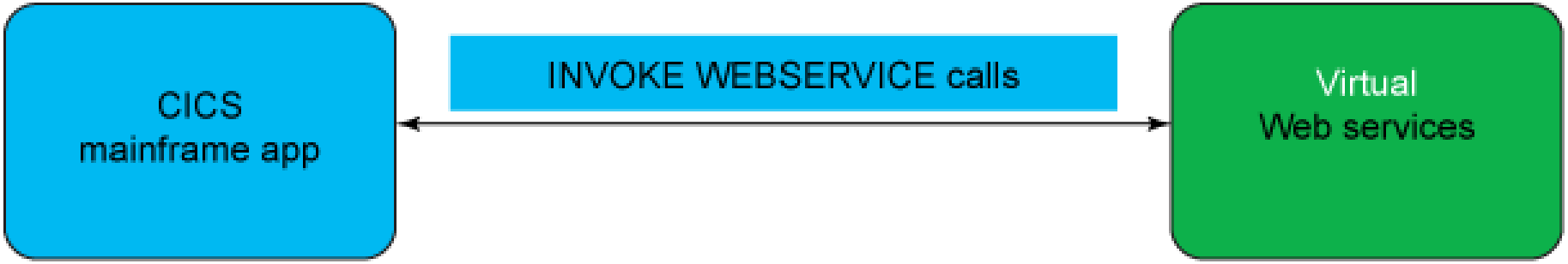


http://www.ibm.com/developerworks/rational/library/automated-mainframe-testing/

IBM Service Virtualization (2)

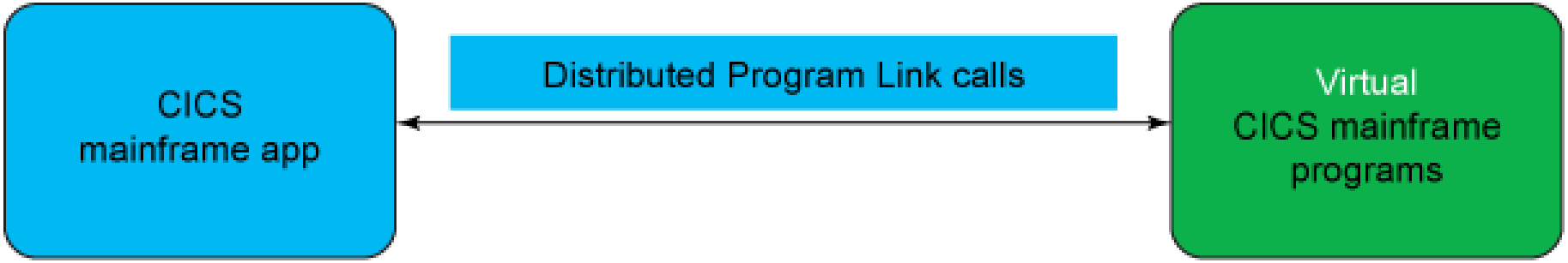
Scenario 3:

CICS program calls a web service with INVOKE WEBSERVICE calls



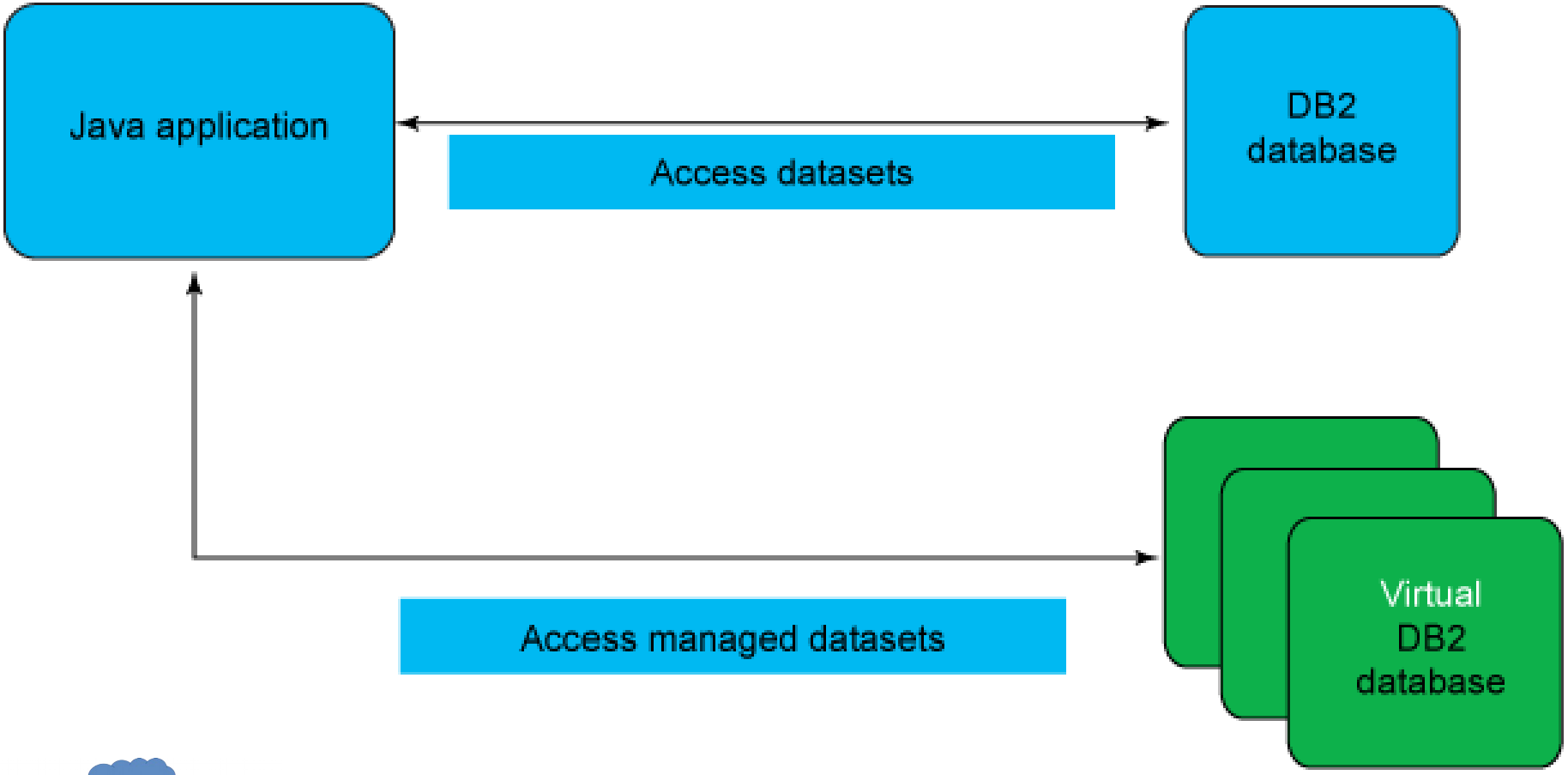
Scenario 4:

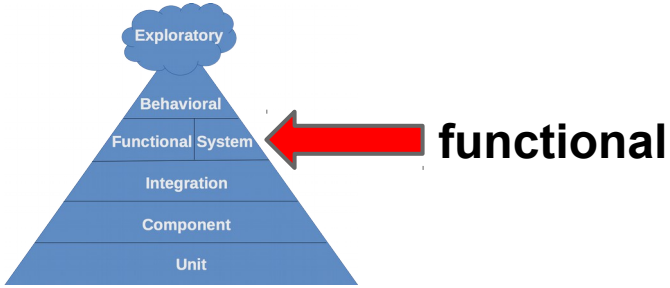
CICS program calls another CICS program with distributed program link



IBM Service Virtualization (3)

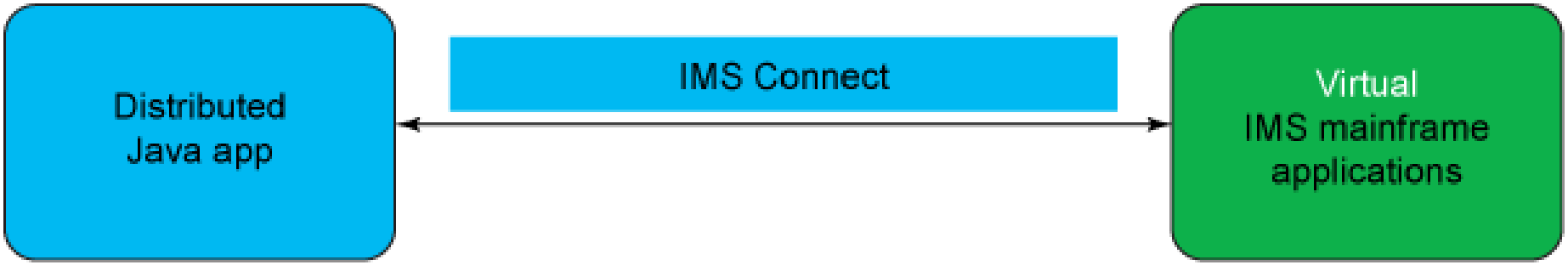
Scenario 5: DB2 database virtualization

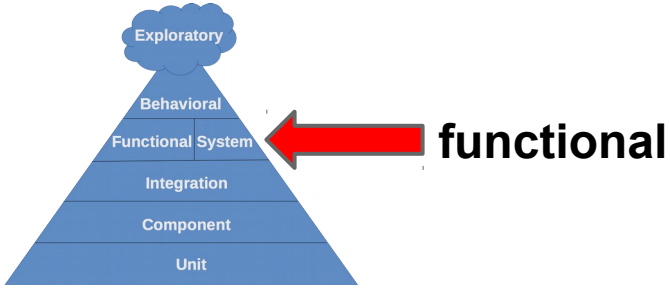




IBM Service Virtualization (4)

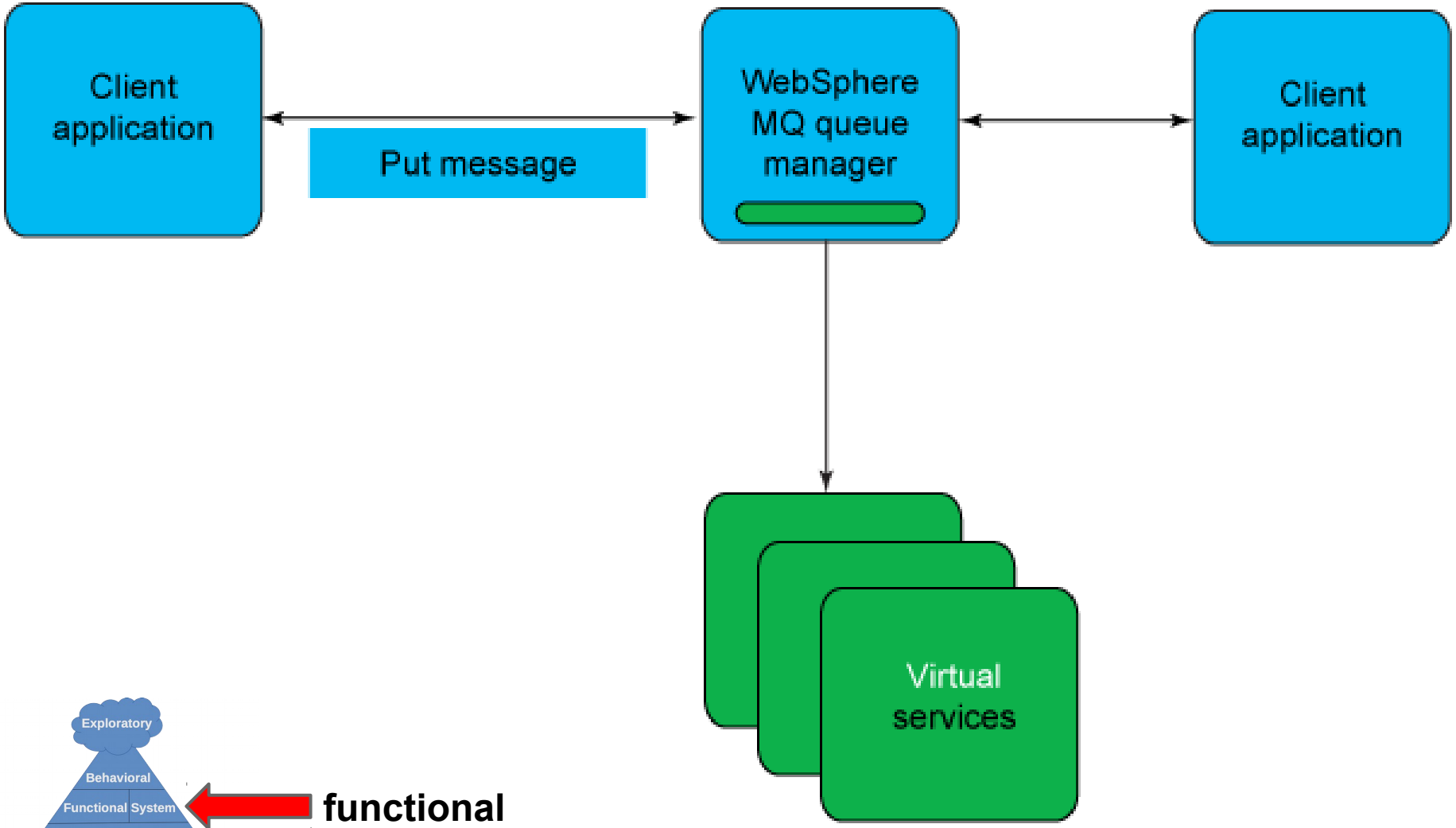
Scenario 6: IMS virtualization

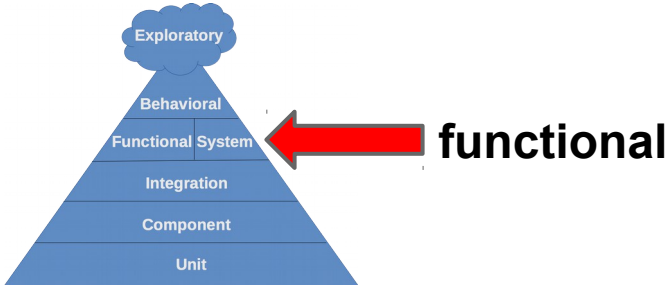




IBM Service Virtualization (5)

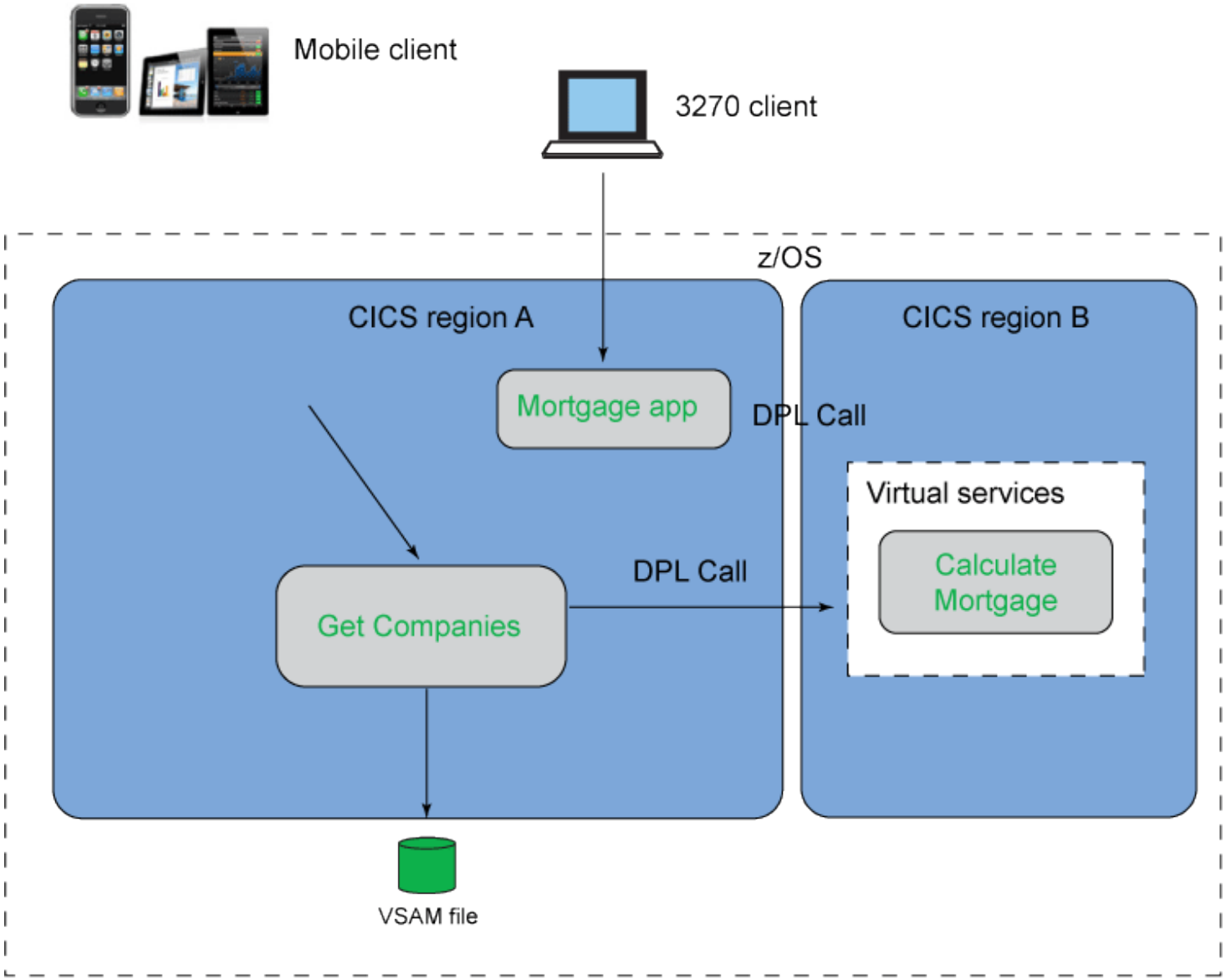
Scenario 7: WebSphere MQ virtualization



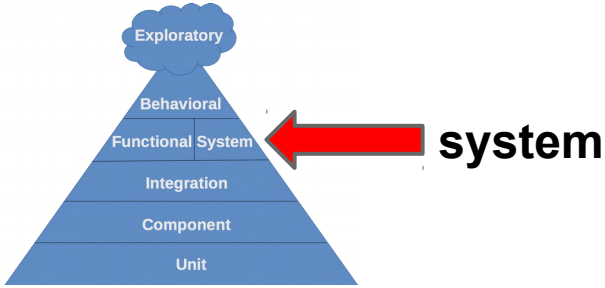


IBM Service Virtualization (6)

Scenario 8: CICS DPL virtualization



System


Exploratory
Behavioral
Functional
System
Integration
Component
Unit
system

System Testing

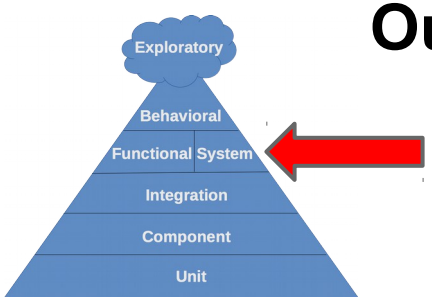
Validate:

- "system attributes," a.k.a.
- "system qualities," a.k.a.
- "non-functional requirements"

Examples:

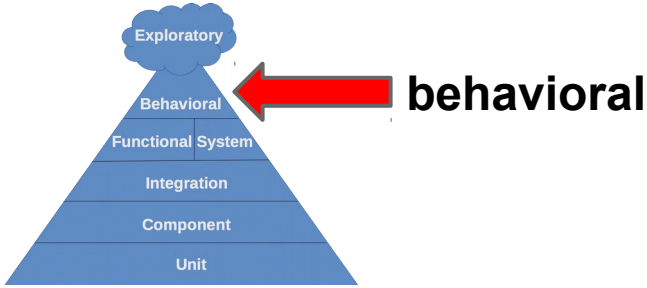
- Load testing
- Longevity testing
- Burst testing
- Security testing
- Performance testing

Out of scope for this session



system

Behavioral


Exploratory
Behavioral
Functional System
Integration
Component
Unit
behavioral

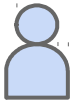
What is Acceptance Test Driven Development (ATDD)?

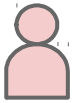
- Industry practice in which whole team collaborates on
  - acceptance criteria and the definition of done
- Automate acceptance tests while production code is being developed
  - automation completes before development
- Developer focuses on making acceptance tests pass

Acceptance Tests Before Code

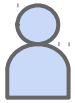
ATDD Workflow: Pre-Sprint

SPECIFICATION	ELABORATION	READY
Write initial specification and specific examples for a user story. **Feature** stories follow the format: "As a *<persona>* I want to *<to take action>* so that *<desired result or*	Elaborate Gherkin Evaluate Gherkin for: • Completeness • Additional scenarios • Edges or other	Stories elaborated Examples are specific and complete Gherkin is complete



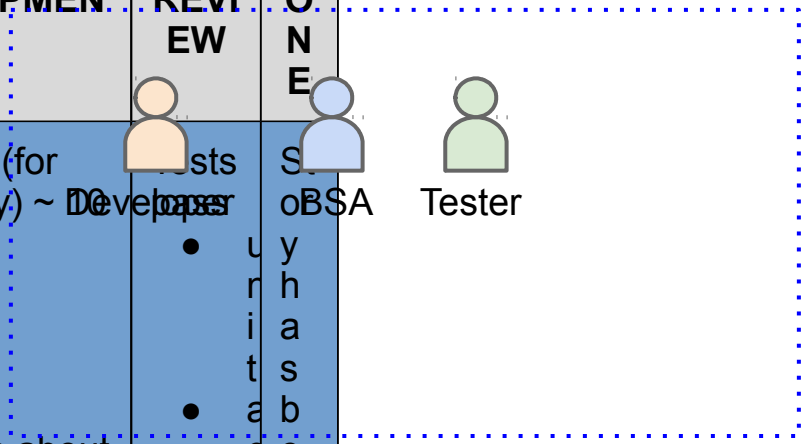


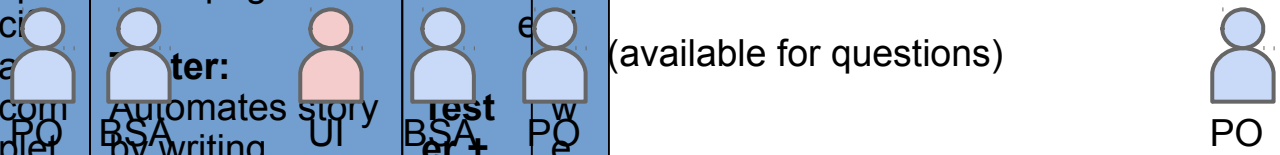
(only when needed)



ATDD Workflow: In-Sprint

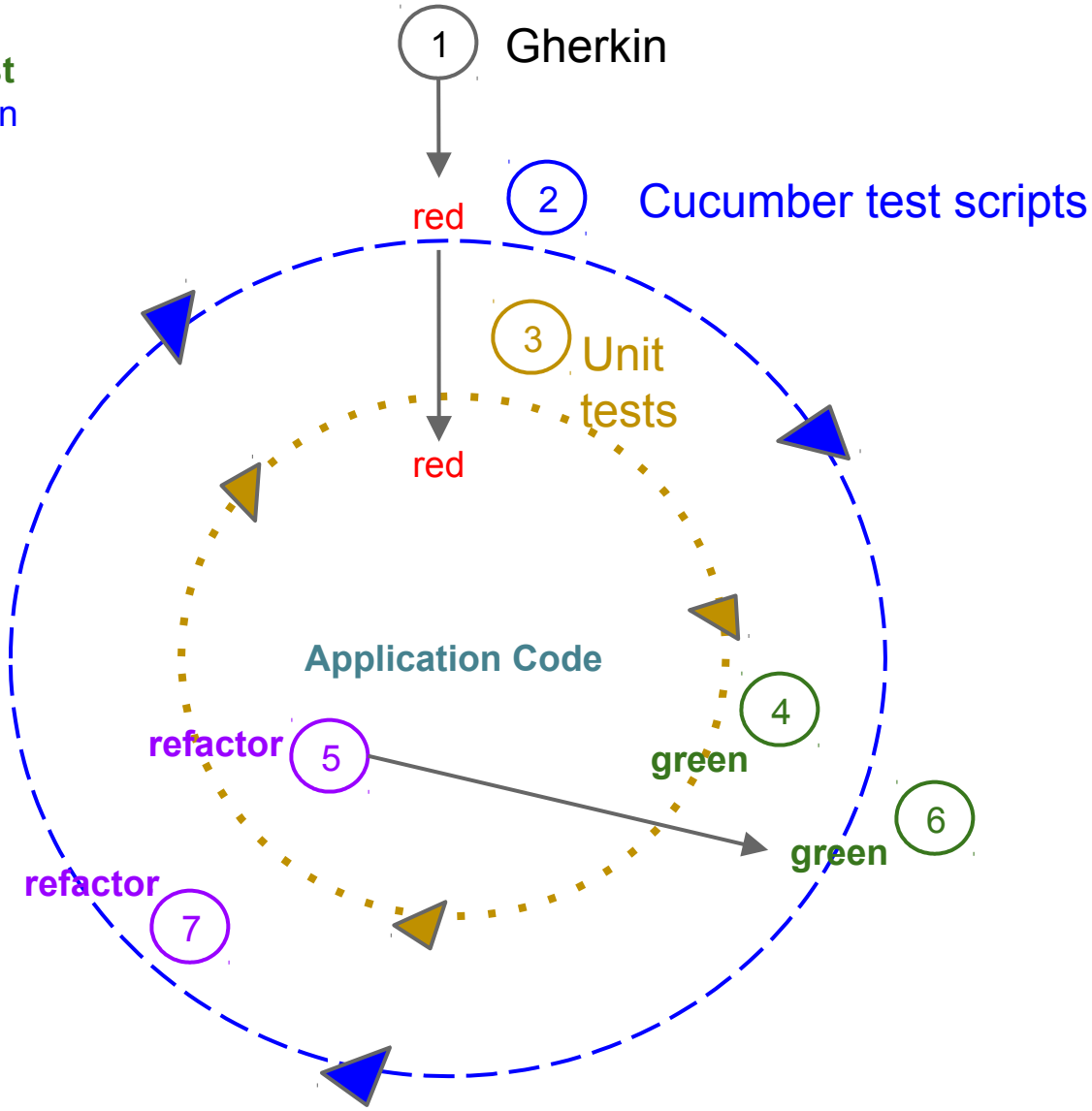
READY	DEVELOPMENT	REVIEW	DONE
Stories elaborated (Pre-sprint) Examples are specific and complete	3 Amigos (for each story) ~ 10 minutes Questions about story: • Doable? • Complete? • Are we on the same page? **Tester:** Automates story by writing Ruby/Cucumber scripts	Tests passes • unit and acceptance •	Story has been tested, reviewed and



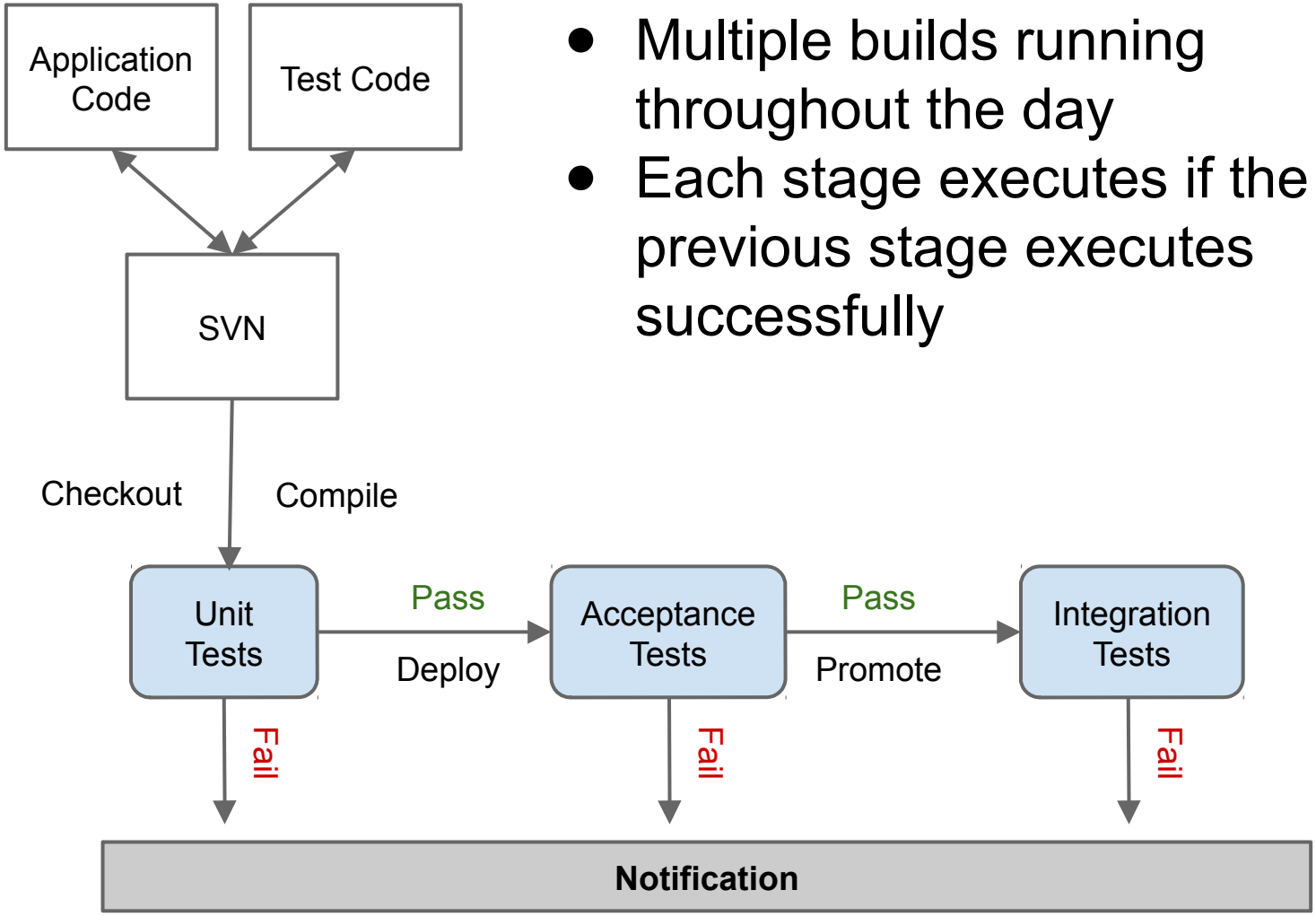


ATDD Workflow: In-Sprint

red: failing test
green: passing test
blue: test automation
gold: unit tests

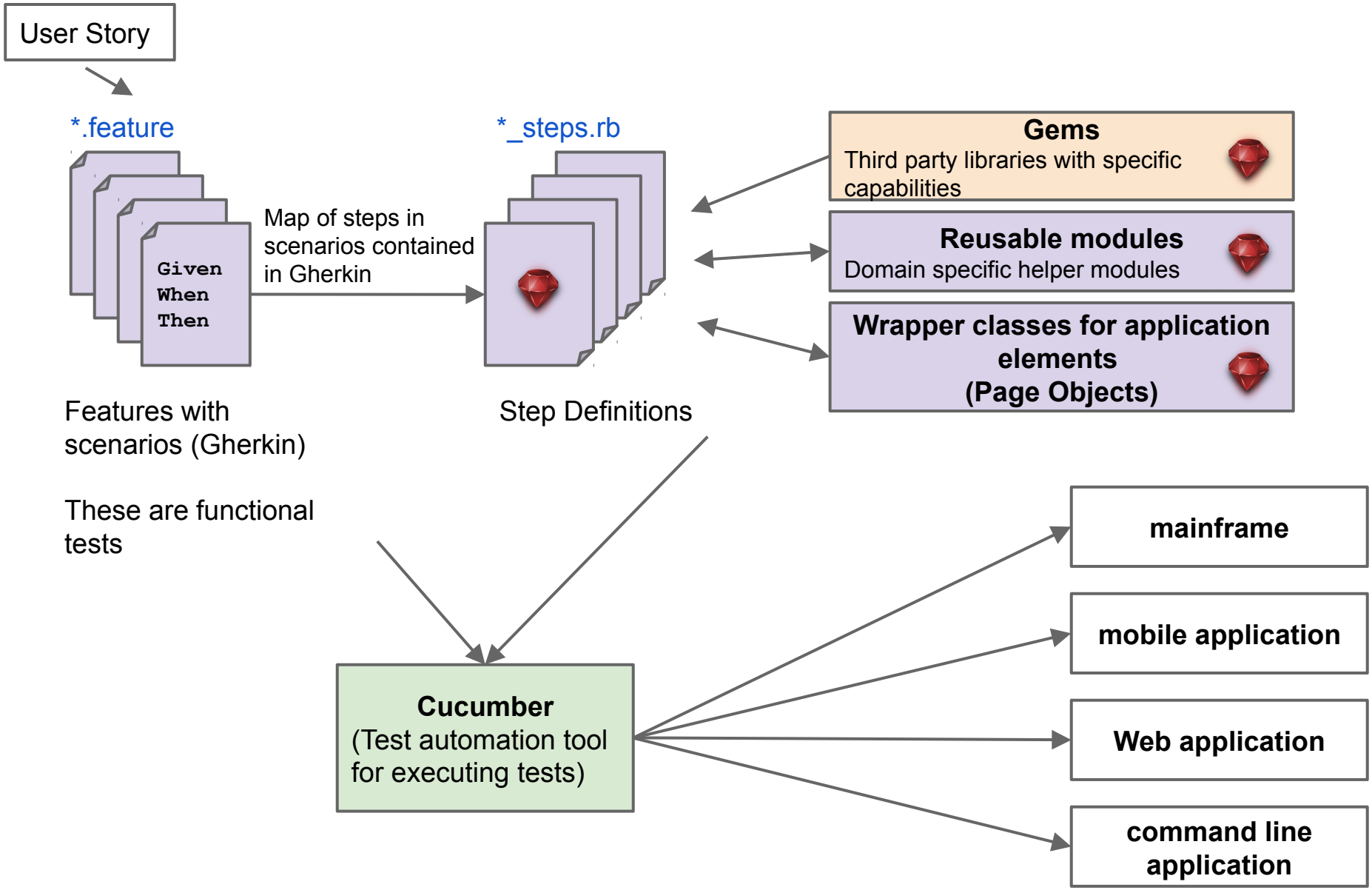


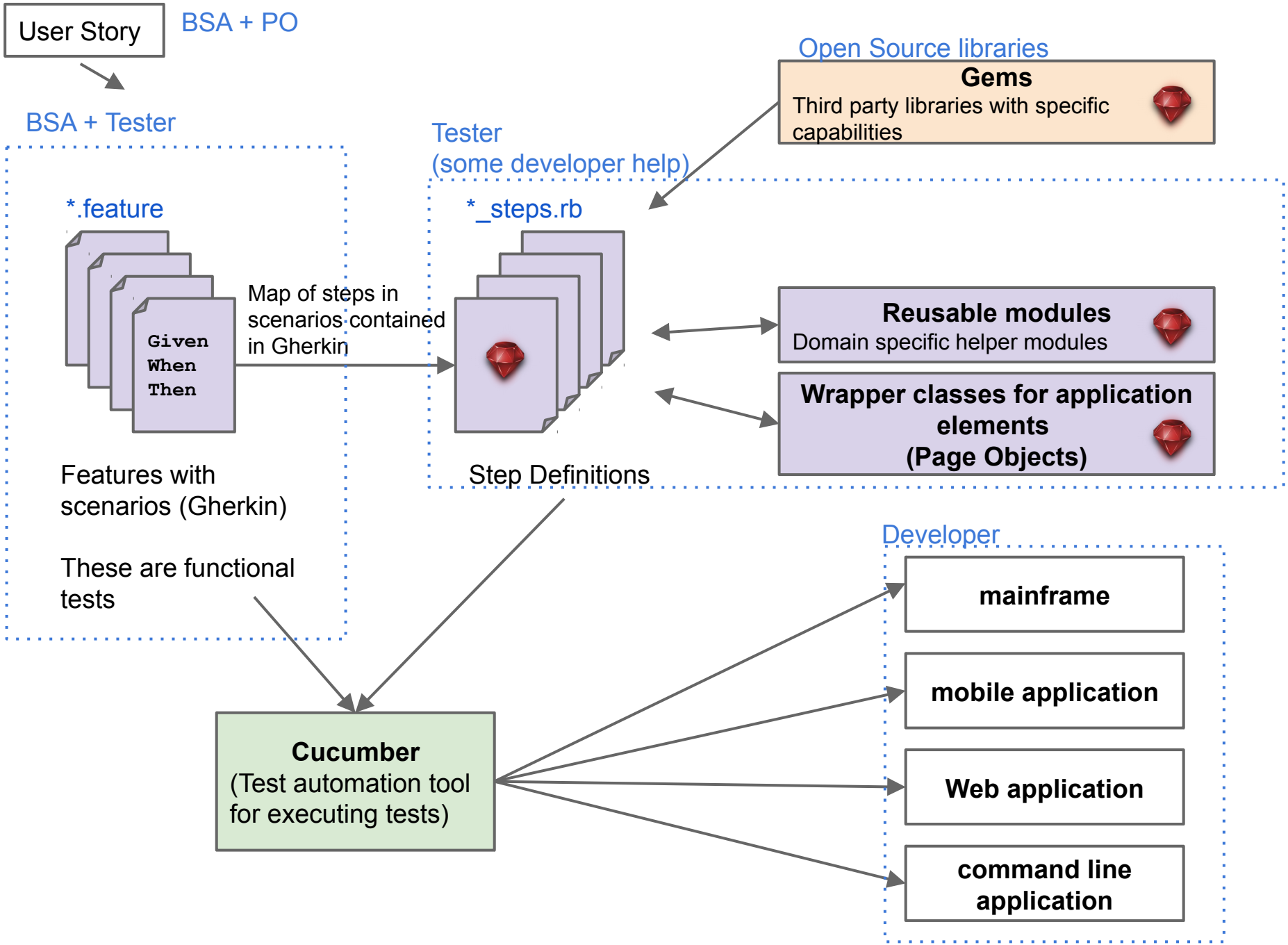
Build Pipeline



- Multiple builds running throughout the day
- Each stage executes if the previous stage executes successfully

Test automation using Cucumber




User Story
BSA + PO
BSA + Tester
*.feature
Given
When
Then
Map of steps in scenarios contained in Gherkin
Features with scenarios (Gherkin)
These are functional tests
Tester
(some developer help)
*_steps.rb
Step Definitions
Open Source libraries
Gems
Third party libraries with specific capabilities
Reusable modules
Domain specific helper modules
Wrapper classes for application elements
(Page Objects)
Cucumber
(Test automation tool for executing tests)
Developer
mainframe
mobile application
Web application
command line application

Cucumber Example: Feature File

```
Feature: Find coffee
  As a coffee lover
  I want to locate good coffee
  So I can start each day with a smile

  Scenario: The search ends well
    Given I am on Amazon
    When I search for "Ravens Brew"
    Then I find what I am looking for

  Scenario: The search ends badly
    Given I am on Amazon
    When I search for "Vlorch Zlongg"
    Then I find nothing
```

Cucumber Example: Steps File

```
Given /^I am on Amazon$/ do
  visit_page AmazonPage
end

When(/^I search for "(.*?)"$/) do |query_value|
  @current_page.query = query_value
  @current_page.submit
end

Then(/^I find nothing$/) do
  @current_page.no_results?
end

Then(/^I find what I am looking for$/) do
  @current_page.results?
end
```

Cucumber Example: Execution

```
neopragma@ubuntu:~/workspace/coffeequest$ HEADLESS=true rake
/home/neopragma/.rvm/rubies/ruby-2.0.0-p451/bin/ruby -S bundle exec cucumber
Using the default profile...
Feature: Find coffee
  As a coffee lover
  I want to locate good coffee
  So I can start each day with a smile

  Scenario: The search ends well
    Given I am on Amazon
    When I search for "Ravens Brew"
    Then I find what I am looking for

  Scenario: The search ends well
    Given I am on Amazon
    When I search for "Starbucks"
    Then I find what I am looking for

  Scenario: The search ends badly
    Given I am on Amazon
    When I search for "Vlorch Zlongg"
    Then I find nothing

3 scenarios (3 passed)
9 steps (9 passed)
0m44.478s
```