Test Driven Development

Best practices applied to IBM i with the assistance of tooling

Presentation by

Barbara Morris – RPG compiler lead

Edmund Reinhardt – RDi lead

The Vision

- IBM i developers are able to confidently change their code
  - Because their code is tested 100%
  - They can achieve this through Test Driven Development
  - They can prove this coverage level
  - They can refactor the code confidently to make it readable, well-structured and taking advantage of the latest technology.

  - As a result the platform is not known for old, hard to modify code,
    - Rather as the most reliable, cost-effective platform for business

Definition of TDD (Test Driven Development)

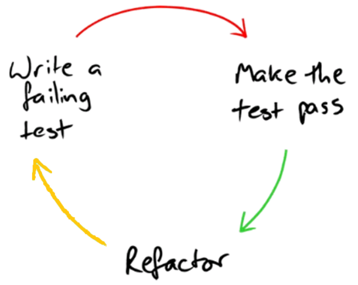
TDD is a way of doing development. The "test" aspect is Unit Test, not Acceptance Test:

Development is done using this loop:

Loop

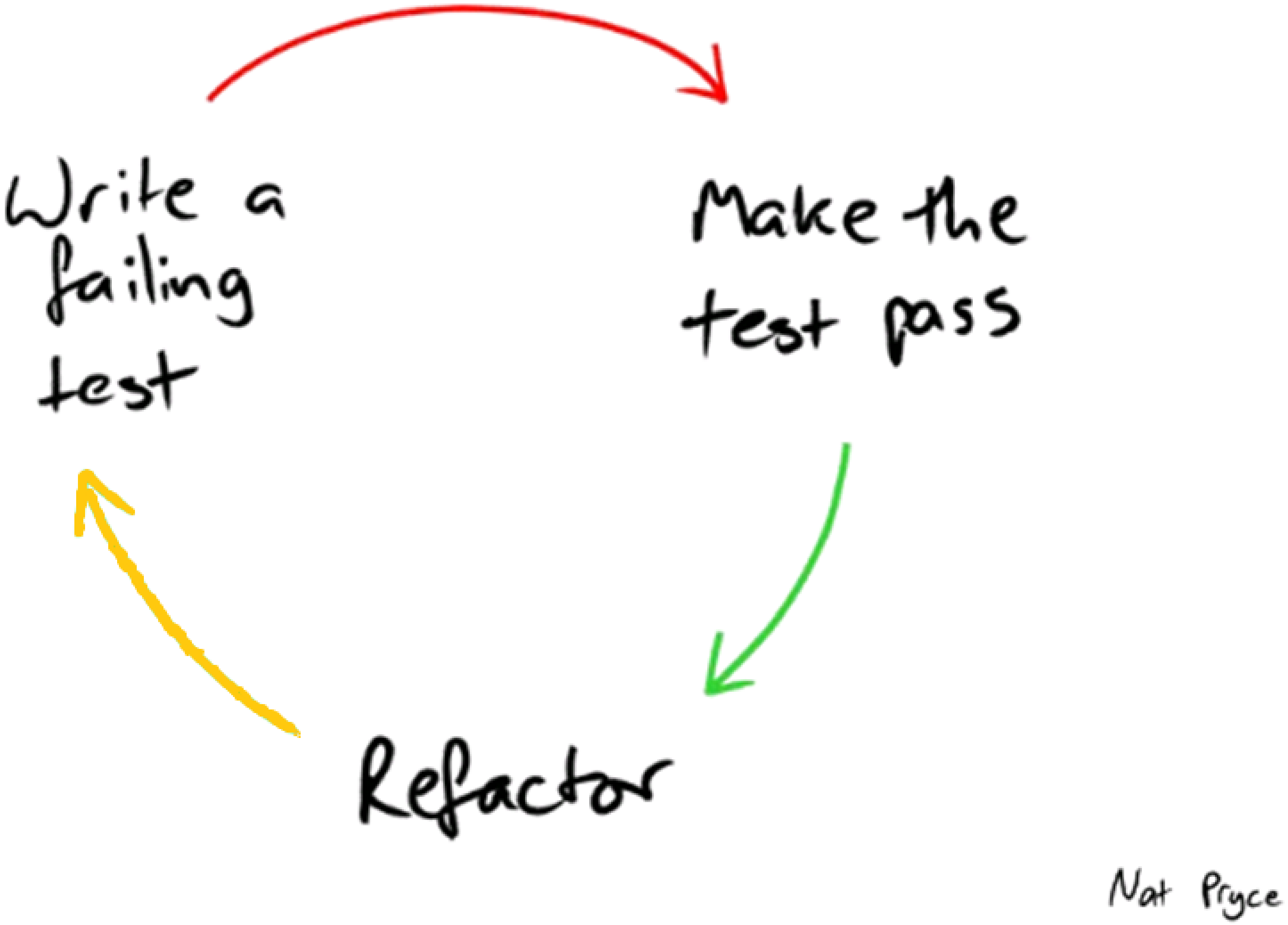
- Write a failing test
- Write the minimum amount of code to make the test pass
  - If you want to add a bit of extra code … **don't do it!**
    - Just add a test to your test list
- Run all existing tests until they all pass
- Refactor if necessary

End-loop

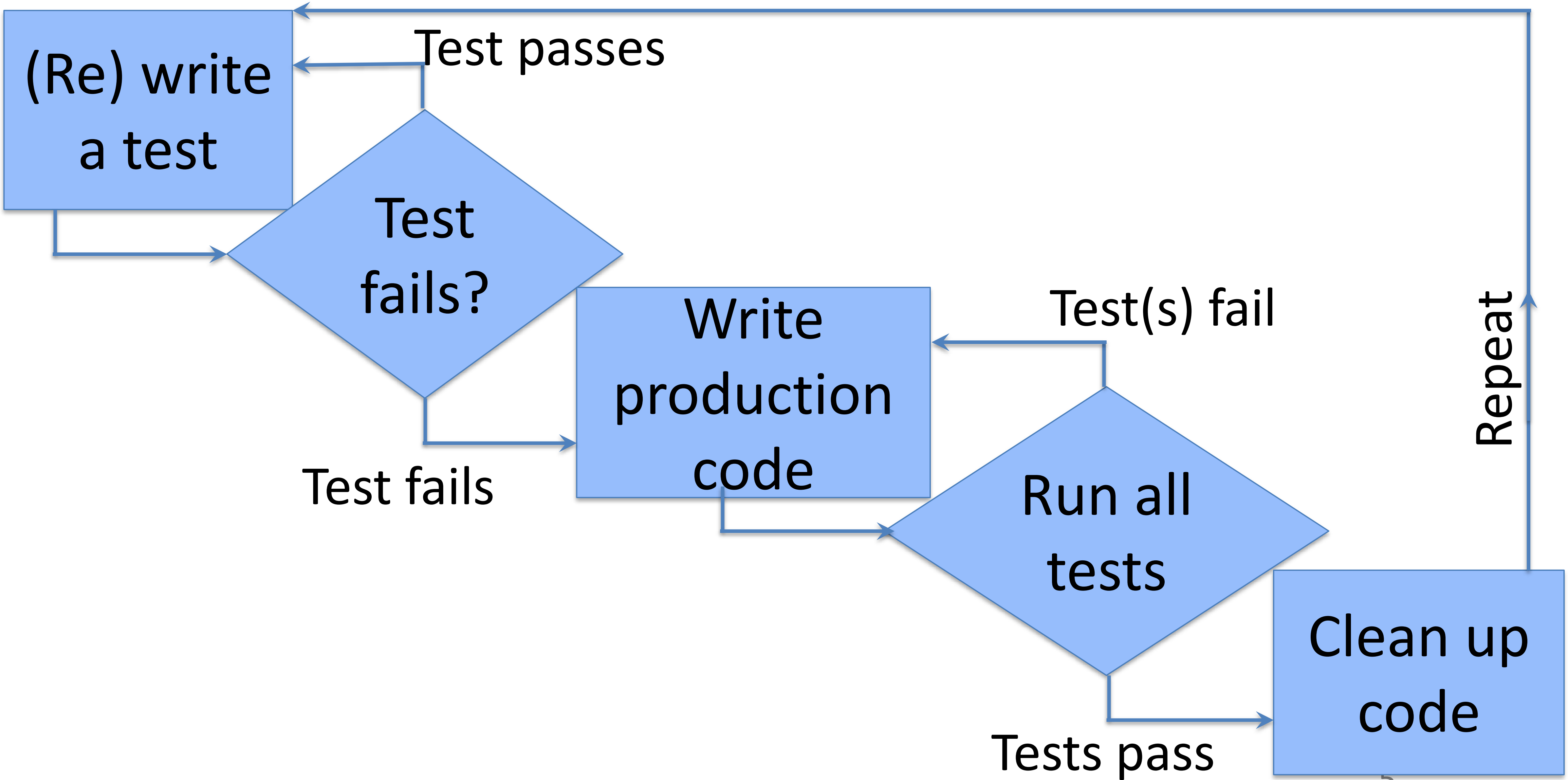


Nat Pryce

TDD at a glance



TDD process flowchart



Rules for TDD

- Maintain a test list with all the test scenarios you know you will need. You will continually add to this list as you develop.
- **Never** write a bit of extra code because you know it will be needed
  - If you think you might forget to write the code, add an item to your test list

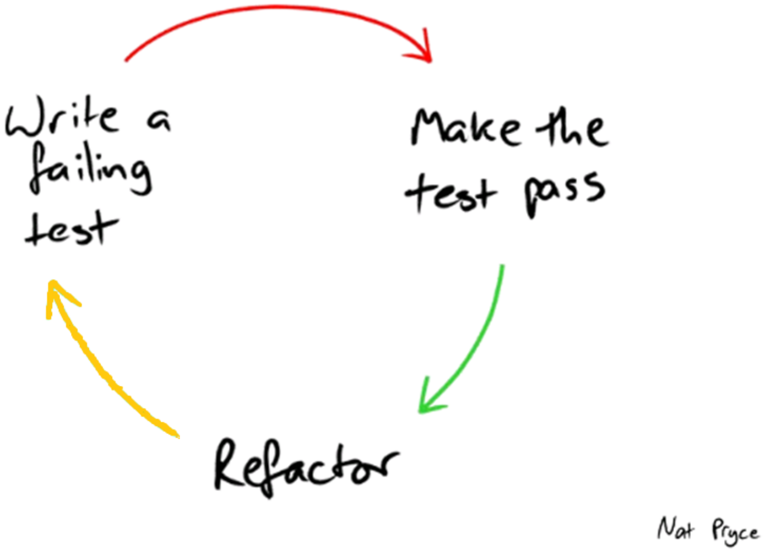
What TDD is not

TDD **does not** mean creating all the testcases for a feature before you start developing the feature.

(Well, some people misuse the term to mean that …)

> But it doesn't matter whether some people use the term to mean writing acceptance tests before development starts.

If you are doing TDD for development, you are using the loop we saw at the beginning:



TDD is hard

TDD requires a lot of self-discipline

It is especially hard to start it in the middle of developing a new feature

But it is possible, and it is worth the effort

What is so great about TDD?

If you use TDD properly

- If you write test scenarios that advance your code in tiny increments
- If you never write extra code
- If you make your unit tests permanent

Then you never have any code that is not tested

But it takes so long, isn't it a waste of time?

It's true, development using TDD can seem very slow

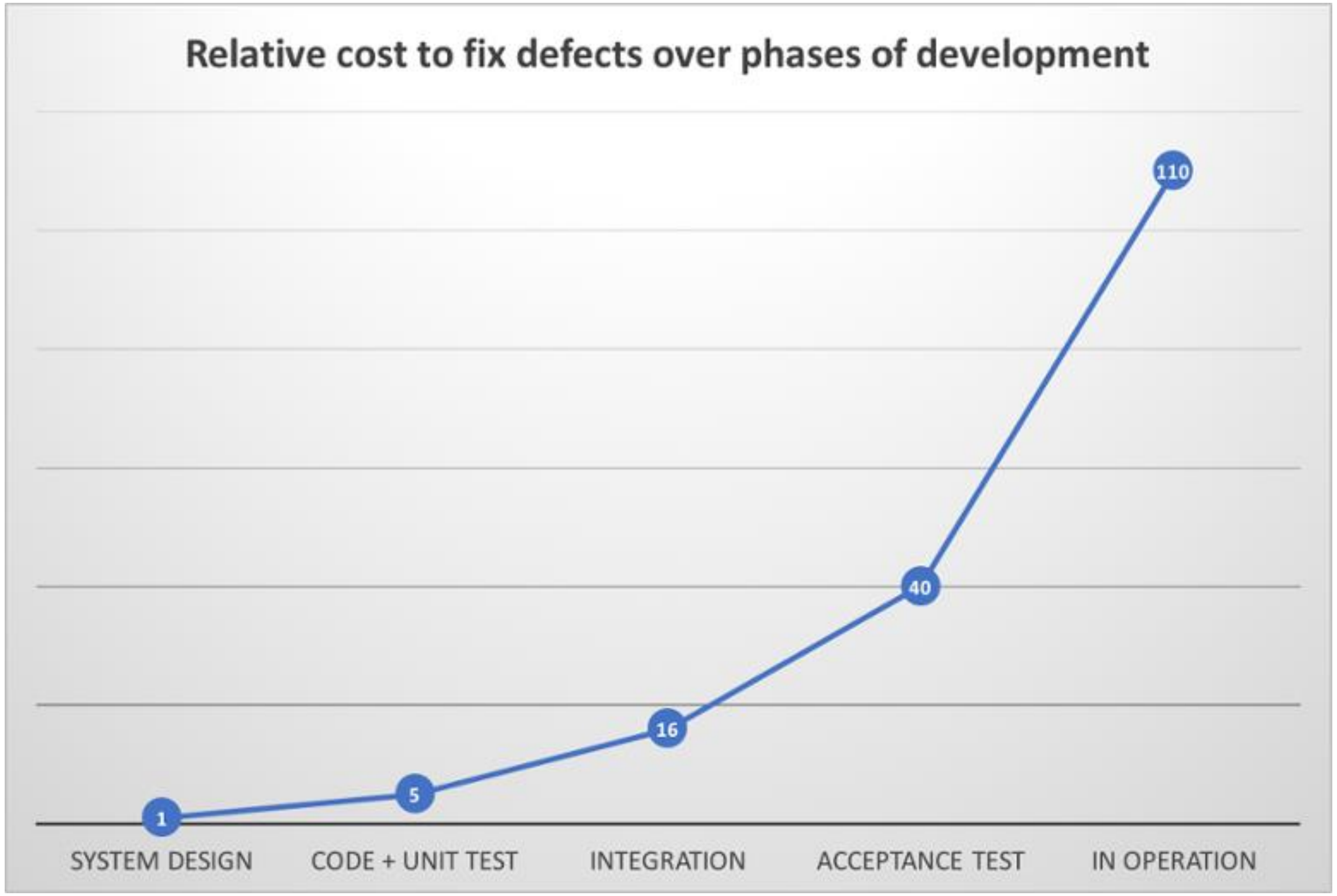
But consider the "Shift Left" principle. From Wikipedia

> *"Shift left testing is an approach to software testing and system testing in which testing is performed earlier in the lifecycle (i.e., moved left on the project timeline)."*

The earlier a defect is found, the cheaper it is to fix.

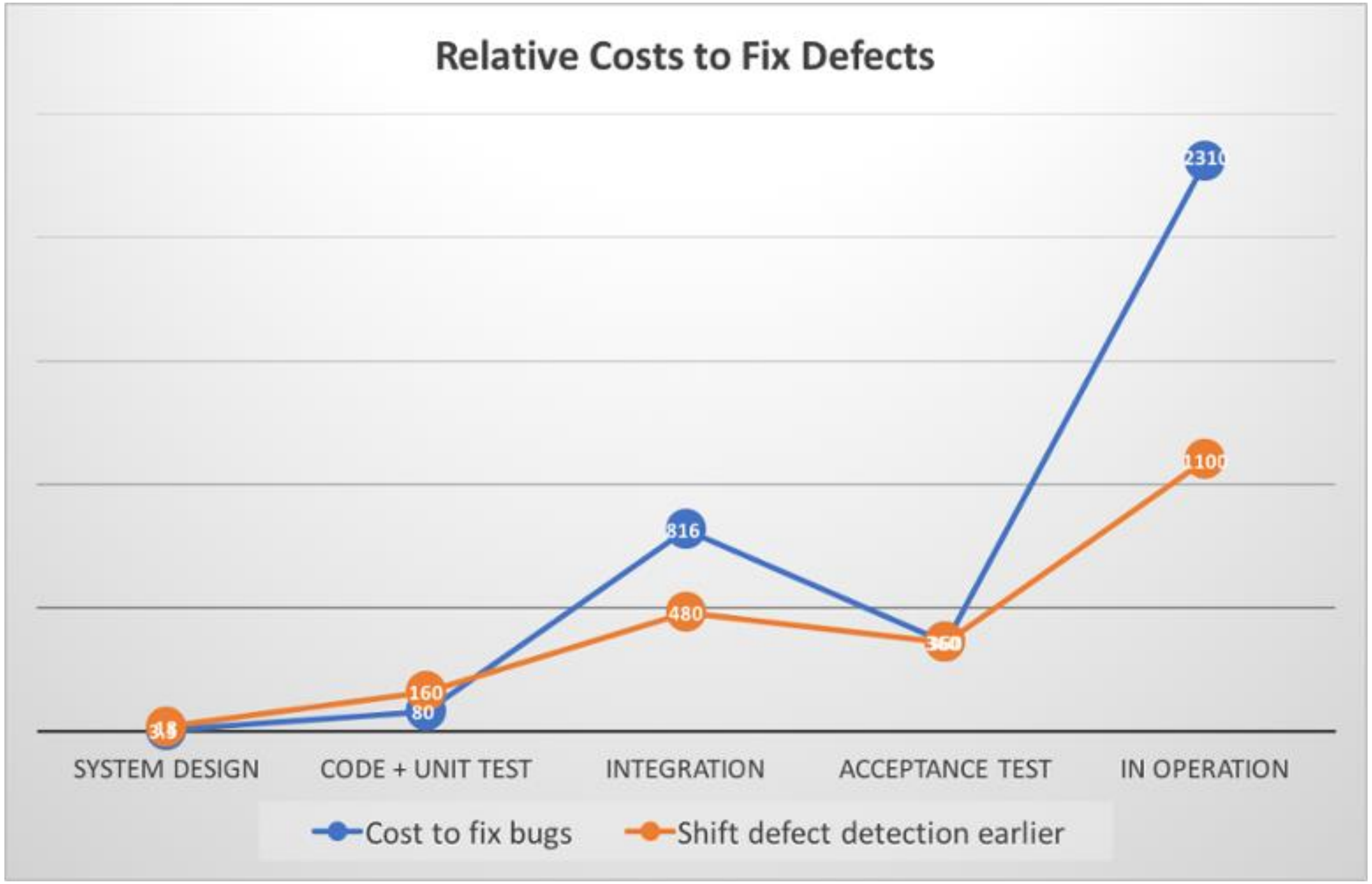
- The cheapest defect is one that is found by the developer while coding.
- The most expensive defect (real $$$) is one that is found by an end user.

Exponential growth in cost, the later a bug is caught



https://dzone.com/articles/shift-left-your-safety-critical-software-testing-w

40% reduction in overall cost by fixing same # of defects earlier



https://dzone.com/articles/shift-left-your-safety-critical-software-testing-w

Example: Write a procedure to center a string

First, we need a failing test:

```
// Centering an empty string returns an empty string
check (center('') : '');
```

This fails compile. We don't even have a procedure yet.

The tests drive the development

We don't write any code until we have a failing testcase!

Example: Write a procedure to center a string

Now we can write some code

The first version of our procedure:

```
dcl-proc center;
  dcl-pi *n varchar(1000);
     string varchar(1000) const;
  end-pi;
  return '';                   // WHAT????
end-proc;
```

That is ridiculous to just return an empty string

Why code such a silly version of the procedure?

Because we only write enough code to make the test pass.

Our first test expects a result of '', so returning '' is the simplest way to make the test pass.

This is what makes TDD hard.

It seems ridiculous at first.

But as you work on "real" procedures, you will see that it means you never have any production code that is not tested.

Instead of writing that little bit of extra code, just add something to your test list.

Example: Write a procedure to center a string

While writing the first version of the procedure, we have lots of ideas for the code we want to write.

We only write the minimum code to make the test pass

We don't write any extra code.

Instead, we write our ideas in our "test list".

Example

The test-list file after writing the first version of the procedure:

```
- Pass a value with no blanks, no centering
  needed
- Pass a value with trailing blanks
- Pass a value with leading blanks
- Pass a value with leading and trailing blanks
- Maybe add an optional parameter to return a
  different length?
```

Example: Write a procedure to center a string

Compile the code and run the test again. It passes!

Add another test:

```
// An empty string returns an empty string
check (center('') : '');
// A string with no leading/trailing blanks
// returns the same string
check (center('abc') : 'abc');
```

Compile the tests and run again. The new test fails! **GOOD!**

Example: Write a procedure to center a string

Second version of procedure: Change the RETURN statement a bit

```
dcl-proc center;
    dcl-pi *n varchar(1000);
       string varchar(1000) const;
    end-pi;
    return string;
  end-proc;
```

Compile the code and run the tests again. They both pass! **GOOD!**

Example

The test-list file after writing the second version of the procedure:

- ~~Pass a value with no blanks, no centering needed~~
- Pass a value with trailing blanks
- Pass a value with leading blanks
- Pass a value with leading and trailing blanks
- Maybe add an optional parameter to return a different length?

Example: Write a procedure to center a string

Add another test:

```
// An empty string returns an empty string

check (center('') : '');

// A string with no leading/trailing blanks returns same string

check (center('abc') : 'abc');

// String with trailing blanks returns balanced blanks

check (center('abc  ') : ' abc ');
```

Compile the tests and run again. The new test fails!

That is GOOD! We want our tests to fail at first.

Example: Write a procedure to center a string

The third version of the procedure will have to actually have some "real" code ... Here's some simple code to make the third test work too

```
if %len(%trim(string)) <> %len(string); // some blanks
  return ' ' + %trim(string) + ' ';
else;
  return string;
endif;
```

Compile the code and run the tests again. They all pass! **GREAT!**

A motivating article by "Uncle Bob"

http://blog.cleancoder.com/uncle-bob/2017/12/18/Excuses.html

The article compares the disciple of developing using TDD to the discipline of doing double-entry bookkeeping.

"Both are disciplines used by experts who carefully manipulate complex documents full of arcane symbols that must, under pain of terrible consequences, be absolutely correct in both type and position."

...

Do accountants have deadlines? Do managers put pressure on them to finish their accounting by a certain date? Of course! There’s no difference in the pressure. Accountants feel the pressure just the same as programmers do.

...

Can you imagine a group of accountants saying to each other: “Guys, we’re really under the gun. We’ve got to finish these accounts by Friday. So, do all the Assets and Equities. We’ll come back and do the Liabilities later.”

Read the whole article!

Do I need something like JUnit to do TDD?

Often, the term "TDD" gets tied in with frameworks such as JUnit (for testing Java), PyUnit (for testing Python).

Using a framework is nice, but it's not necessary.

You just need to have a way of running self-checking testcases.

Two extremes:

- Edmund can use Junit to test the Java code he writes for RDi.
- Barbara writes RPG statements that target the function within the compiler that she is working on.

Is there a framework like JUnit for RPG?

Yes! RPGUNIT.

The next section of this presentation will walk through a more meaningful example using RPGUNIT.

Develop a procedure to get an account balance from a file using RPGUNIT

Some initial setup

- The RPG unit framework expects the testcases to be in a service program

```
ctl-opt nomain;
```

- The framework has some procedures to check whether the test passed

```
/copy rpgunit/rpgunit1,testcase
```

- To simplify creating the test service program, we'll create a binding directory with the module we are developing

```
ctl-opt bnddir('TDD_DEMO/UNITTEST');
```

- The prototype for the module we are developing is in TDD_DEMO/SOURCE

```
/copy TDD_DEMO/source,getbal_h
```

The first version of the test list

Our (silly) design says that if the account is not found, the balance is zero.

```
- Account not found, return zero balance
- Account found, return the balance
- Maybe add a way to say whether the account
  exists?
```

The first version of the test module

```
**free
ctl-opt option(*srcstmt);
ctl-opt nomain;
ctl-opt bnddir('TDD_DEMO/UNITTEST');
/copy rpgunit/rpgunit1,testcase
/copy TDD_DEMO/source,getbal_h

dcl-proc Test_...
```

Test procedures must start with "test".

They must be exported.

The first testcase

- Give a meaningful name to the procedure

```
   dcl-proc Test_returnZeroForIdNotFound export;
```

- Use EXTPROC(*DCLCASE) to have the "real" name be mixed case

```
     dcl-pi *n extproc(*dclcase) end-pi;
```

- The code to actually test the procedure call

```
     dcl-s balance packed(11:2);
     balance = getBalance ('abc');
```

- The code to check the result

```
     assert (balance = 0 : 'Balance not 0 for id not found');
```

The first testcase

```
dcl-proc Test_returnZeroForIdNotFound export;
   dcl-pi *n extproc(*dclcase) end-pi;
   dcl-s balance packed(11:2);
   balance = getBalance ('abc');
   assert (balance = 0 : 'Balance not 0 for id not found');
end-proc;
```

Create the test service program (first attempt)

```
RUCRTTST TSTPGM(TDD_DEMO/TEST_BAL)
         SRCFILE(TDD_DEMO/UNITTEST) SRCMBR(TEST_BAL)
         COPTION(*SRCSTMT *EVENTF) DBGVIEW(*ALL)
```

Joblog:

```
File SOURCE in library TDD_DEMO with member GETBAL_H not found.
Compilation stopped. Severity 40 errors found in program.
Compilation failed. Module TEST_BAL not created in library TDD_DEMO.
```

We have to write some code now.

The first version of our copy file

Copy file GETBAL_H

```
**free
dcl-s balance_t packed(11:2) template;
dcl-s id_t char(10) template;
dcl-pr getBalance like(balance_t) extproc(*dclcase);
   id like(id_t) const;
end-pr;
```

The first version of our code

Member GETBAL

```
**free
ctl-opt nomain;
/copy TDD_DEMO/source,getbal_h

dcl-proc getBalance export;
   dcl-pi *n like(balance_t);
      id like(id_t) const;
   end-pi;
   return 0;
end-proc;
```

There is enough code here for the module to compile.

We'll return zero because that's the simplest thing to do to make the test pass.

We don't actually need to define the file yet.

Create the GETBAL module and create the UNITTEST bnddir

One time setup

```
CRTBNDDIR TDD_DEMO/UNITTEST
ADDBNDDIRE BNDDIR(TDD_DEMO/UNITTEST)
           OBJ((TDD_DEMO/GETBAL *MODULE))
```

Do this every time you change the module under development or its test

```
CRTRPGMOD TDD_DEMO/GETBAL SRCFILE(TDD_DEMO/SOURCE) DBGVIEW(*ALL)
RUCRTTST TSTPGM(TDD_DEMO/TEST_BAL)
         SRCFILE(TDD_DEMO/UNITTEST) SRCMBR(TEST_BAL)
         COPTION(*SRCSTMT *EVENTF) DBGVIEW(*ALL)
```

Joblog:

```
   Service program TEST_BAL created in library TDD_DEMO.
```

Run the test

```
RUCALLTST TDD_DEMO/TEST_BAL DETAIL(*ALL)
```

Spool file:

```
 *** Tests from TEST_BAL *** (iRPGUnit v2.1.0)
     Date : 2018-05-08 / 17:42:40
     Job  : 138881/BMORRIS/QPADEV0001
     User : BMORRIS
     iRPGUnit : BMORRIS
     Test_returnZeroForIdNotFound - Success
       1 assertions
     -----------------------
     Success. 1 test case, 1 assertion, 0 failure, 0 error.
```

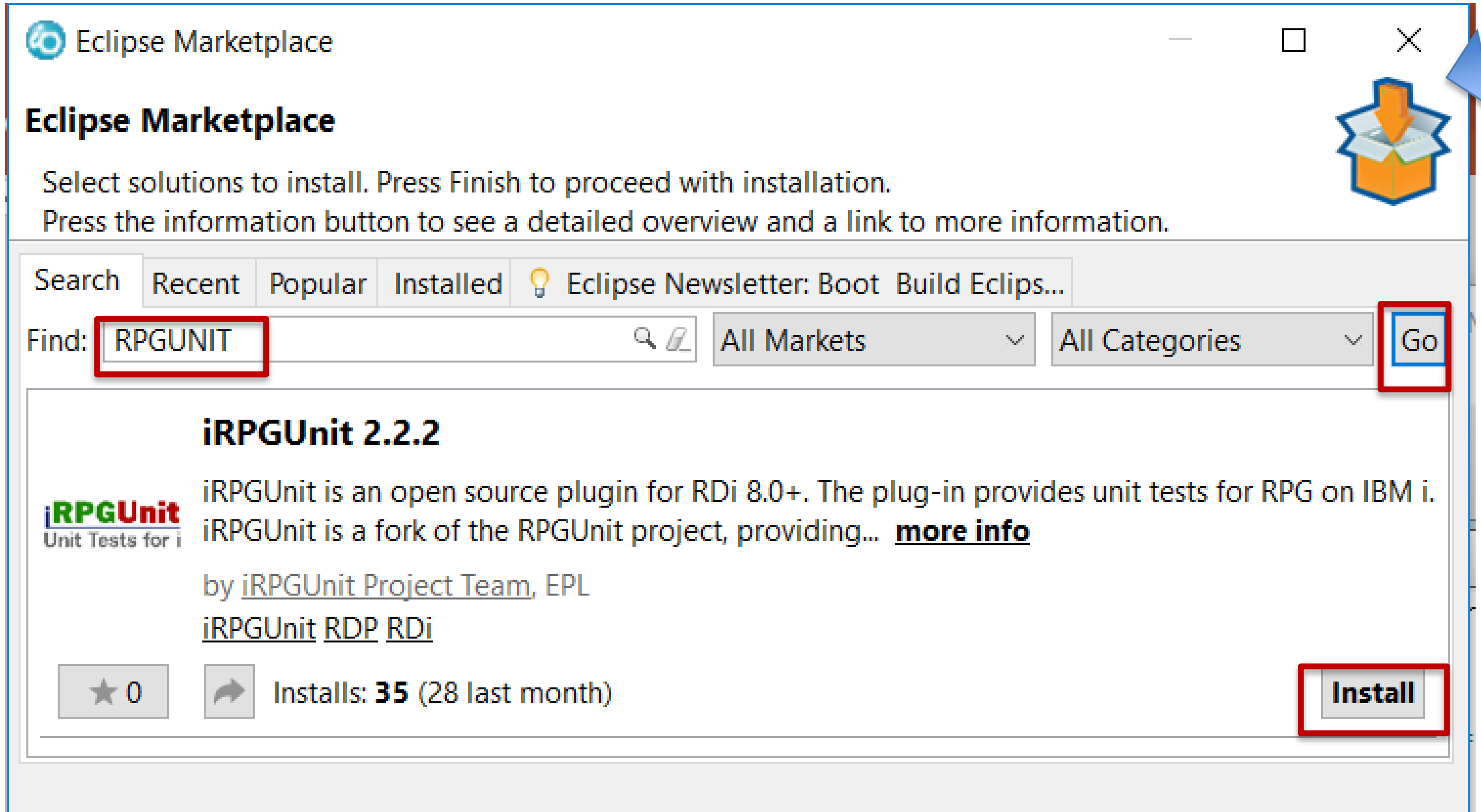
Using the RPGUNIT plugin for RDi

Adding the RPGUNIT plugin to RDi is easy:

In RDi, do ***Help > Eclipse Marketplace***

Type in ***RPGUNIT*** and press ***Go*** to find the plugin

Press the ***Install*** button and it will be installed



Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Find: RPGUNIT

All Markets

All Categories

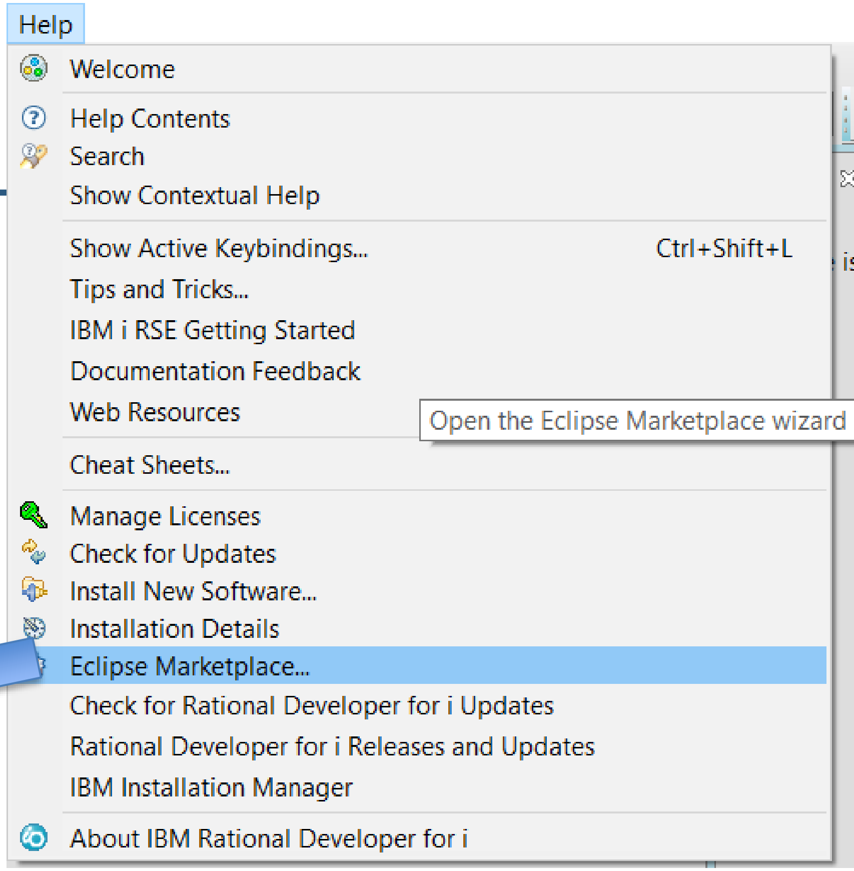
iRPGUnit 2.2.2

iRPGUnit is an open source plugin for RDi 8.0+. The plug-in provides unit tests for RPG on IBM i. iRPGUnit is a fork of the RPGUnit project, providing... **more info**

by iRPGUnit Project Team, EPL

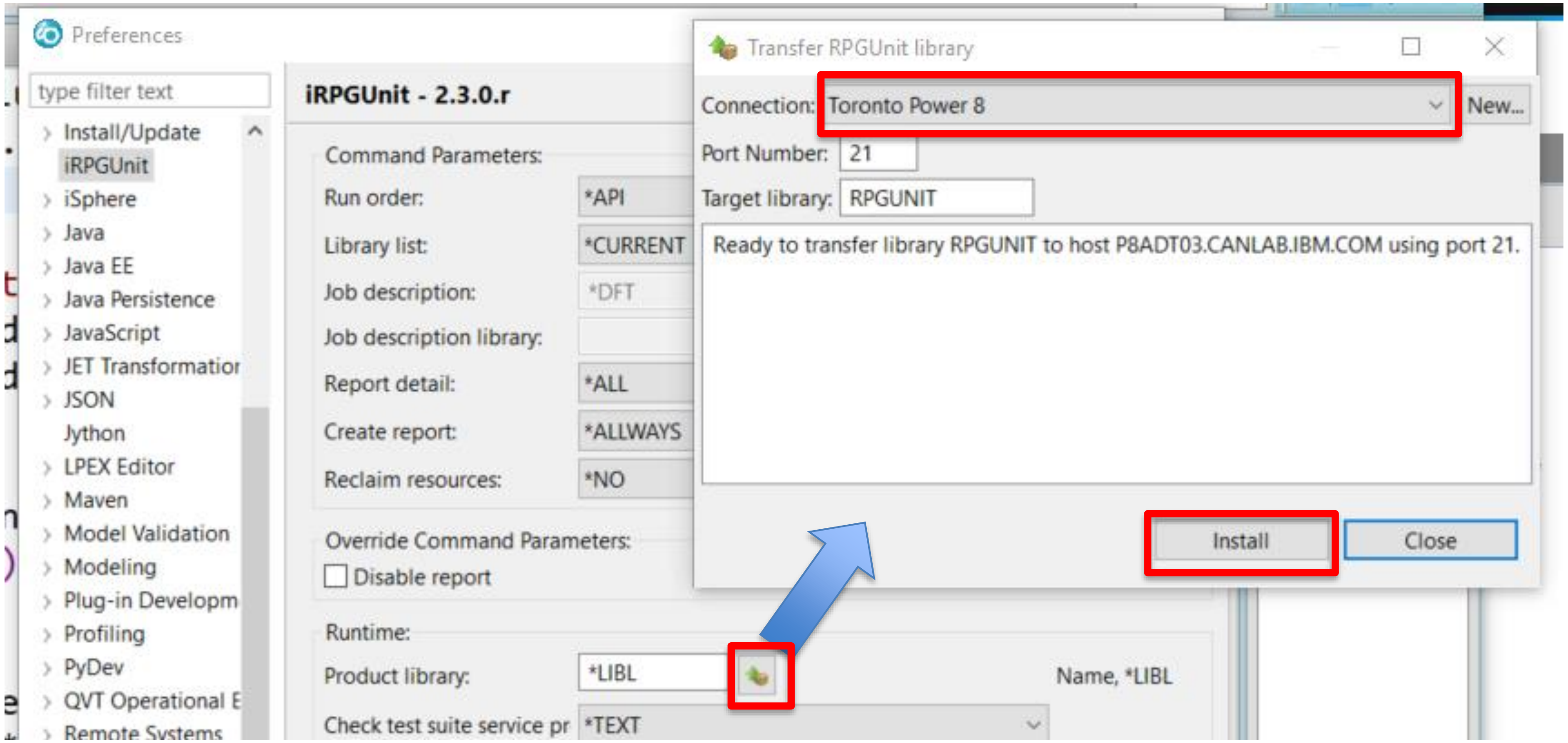
iRPGUnit RDP RDi

Installs: **35** (28 last month)



Upload the RPGUNIT library

- Simply click on the upload button in the iRPGUnit preference page and it will be done for you



iRPGUnit - 2.3.0.r

Command Parameters:

Run order: *API

Library list: *CURRENT

Job description: *DFT

Job description library:

Report detail: *ALL

Create report: *ALLWAYS

Reclaim resources: *NO

Override Command Parameters:

□ Disable report

Runtime:

Product library: *LIBL

Name, *LIBL

Check test suite service pr *TEXT

Transfer RPGUnit library

Connection: Toronto Power 8

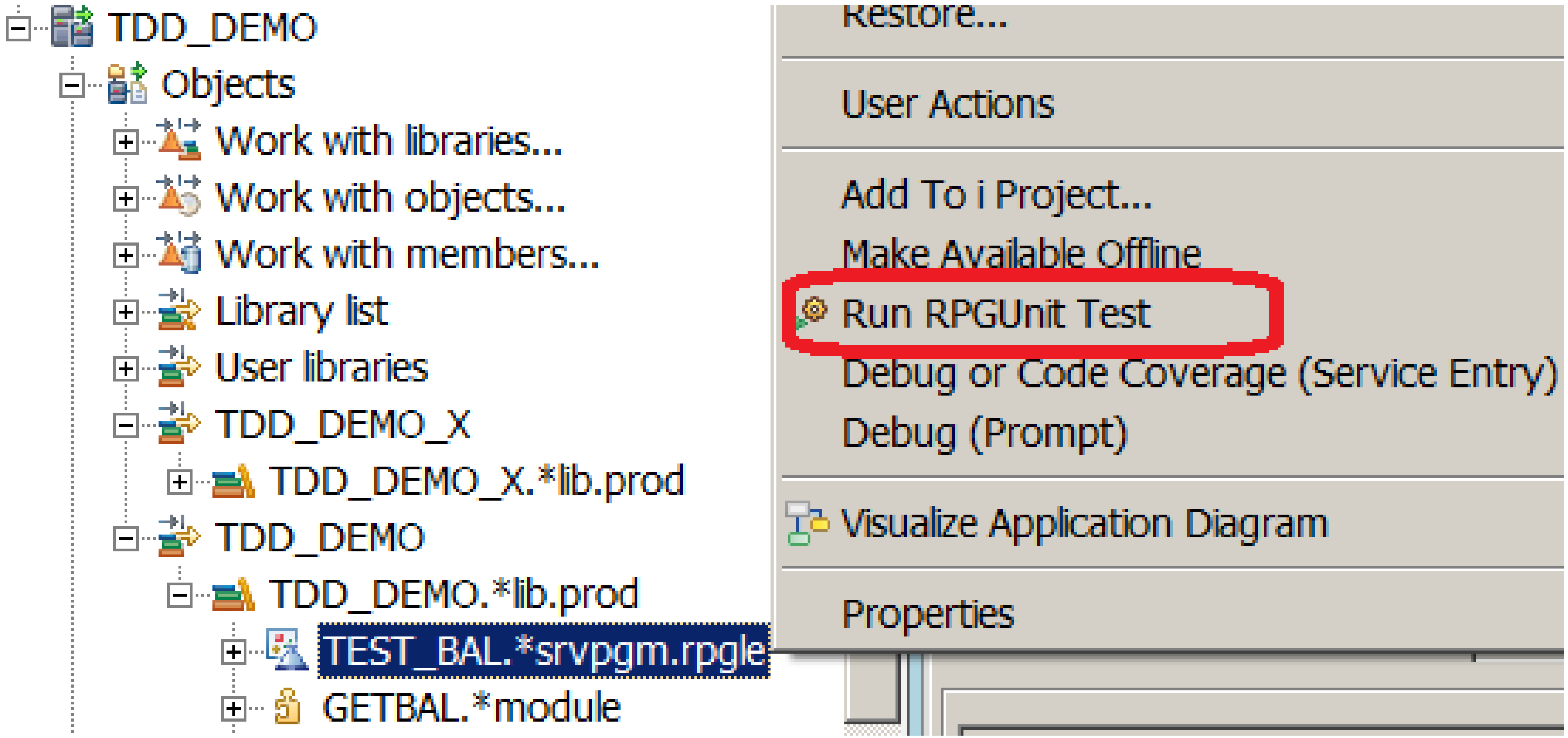
Port Number: 21

Target library: RPGUNIT

Ready to transfer library RPGUNIT to host P8ADT03.CANLAB.IBM.COM using port 21.

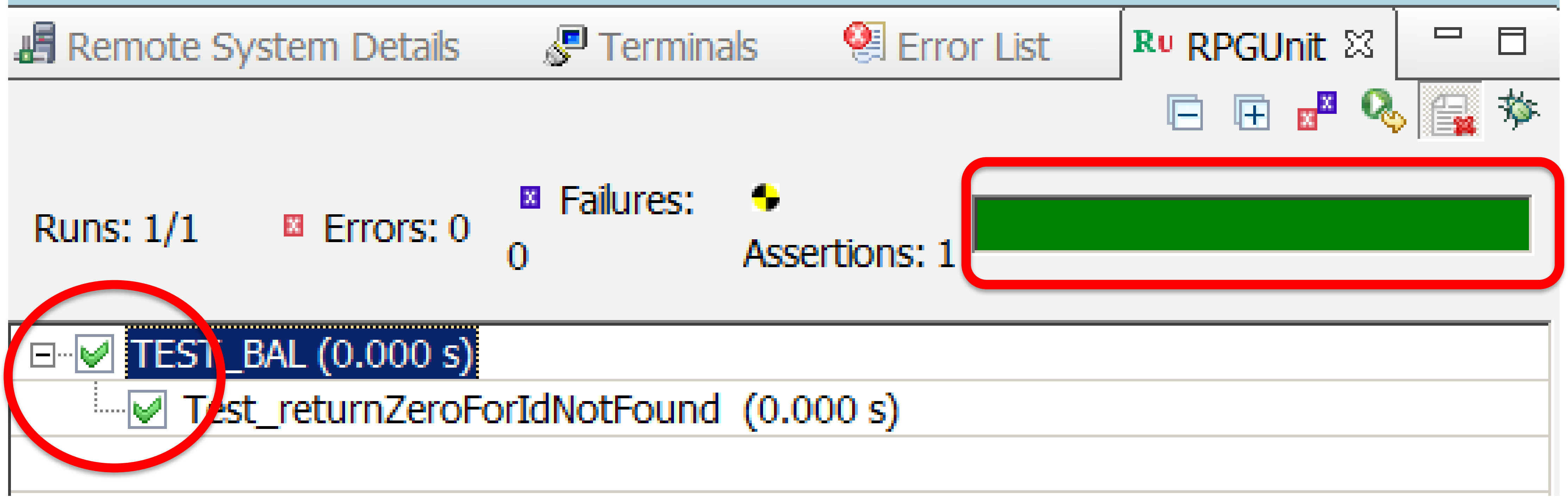
Using the RPGUNIT plugin for RDi

Now that the plugin is installed, a new action appears for service programs:



Using the RPGUNIT plugin for RDi

It's easy to see that the test passed.



Runs: 1/1 Errors: 0 Failures: 0 Assertions: 1

TEST_BAL (0.000 s)
Test_returnZeroForIdNotFound (0.000 s)

Don't forget to add library RPGUNIT to the library list!

The second testcase - look at our test list

The next test scenario is to actually return the balance for a real account

- ~~Account not found, return zero balance~~
- Account found, return the balance
- Maybe add a way to say whether the account exists?

The second testcase

```
dcl-proc Test_returnBalanceForValidId export;
  dcl-pi *n extproc(*dclcase) end-pi;
  dcl-s balance packed(11:2);

  add_record ('123' : 123.45);
  balance = getBalance ('123');
  assert (123.45 = balance : 'Balance <> 123.45');
end-proc;
```

Notice the call to add_record. This is a new procedure in our test source.

The second testcase

It's time to actually work with the file.

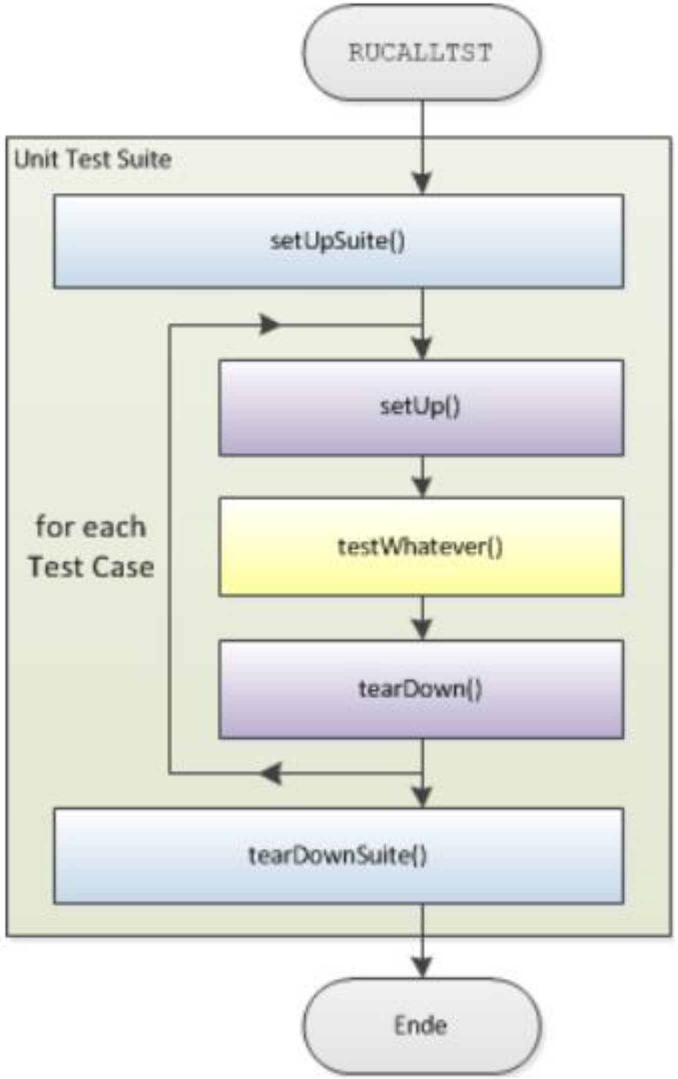
- Create a test file for the procedure to work with
- Override to the file so getBalance won't pick up the production file
- Add a record

Notice in the testcase that there is only a call to add_record.

Where is the code to create the file and do the override?

Set up and tear down

- Often it is required to set up data or allocate resource before a test suite is executed to ensure a reproducible test environment for valid test results. Also people often want to clean up test data or release resources at the end of a unit test.
- Sometimes the same wish applies to a single test case.
- Both requirements are supported by RPGUnit
- If the procedure is present and exported, it will be called at the right time



Setup procedure

RPGUNIT will call the "setup" procedure before running each testcase.

```
dcl-proc setup export;
  callp(e) runCmd ('DLTF QTEMP/BALANCEF');
  create_empty_file ();
end-proc;

dcl-proc create_empty_file;
   runCmd ('CRTPF QTEMP/BALANCEF SRCFILE(TDD_DEMO/SOURCE) '
           + 'SRCMBR(BALANCEF)');
   runCmd ('OVRDBF BALANCEF QTEMP/BALANCEF');
end-proc;
```

NOTE: runCmd() is provided by RPGUNIT and just calls QCMDEXEC

Teardown procedure

RPGUNIT will call the "teardown" procedure after running each testcase.

```
dcl-proc teardown export;
  callp(e) runCmd ('DLTF QTEMP/BALANCEF');
end-proc;
```

Some assumptions

Our new procedure is working with an existing file.

So we can assume

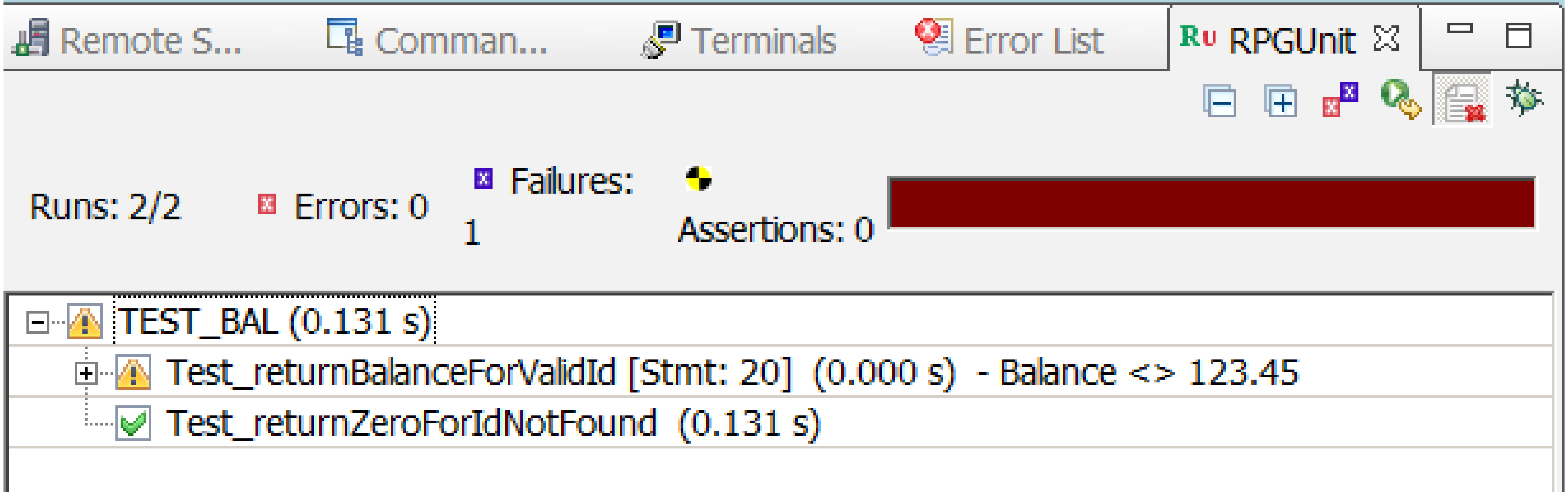
- The compile-time file TDD_DEMO/BALANCEF_D exists
- The source for the file exists in TDD_DEMO/SOURCE, member BALANCEF

DEMO
TIME
(tv2 *)

Run the new version of the test

(Don't forget to recreate the test by calling RUCRTTST)

The new test failed! (As expected.)



```
Runs: 2/2    Errors: 0    Failures: 1    Assertions: 0

TEST_BAL (0.131 s)
    Test_returnBalanceForValidId [Stmt: 20]  (0.000 s)  - Balance <> 123.45
    Test_returnZeroForIdNotFound  (0.131 s)
```

The second version of our getBalance procedure

```
dcl-proc getBalance export;
   ...
   dcl-f balancef keyed
      extdesc('TDD_DEMO/BALANCEF_D')
      extfile('BALANCEF');
   dcl-ds ds likerec(balancer) inz;
   chain id balancer ds;
   return ds.balance;
end-proc;
```

Added:

- The file definition
- The data structure definition
- The CHAIN opcode

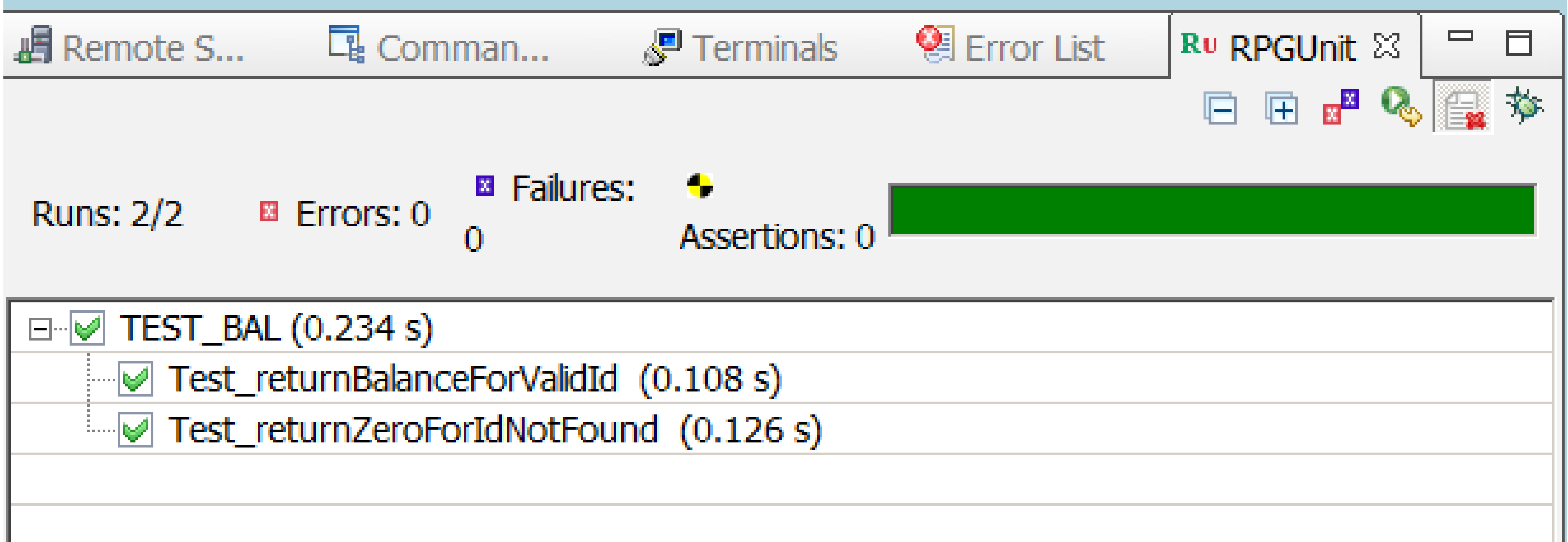
DEMO TIME

dv2

Run the new version of the test

Recreate the GETBAL module and recreate the test so it picks up the new GETBAL module.

The new test passed! And the first test still passed too!



Runs: 2/2 Errors: 0 Failures: 0 Assertions: 0

TEST_BAL (0.234 s)
- Test_returnBalanceForValidId (0.108 s)
- Test_returnZeroForIdNotFound (0.126 s)

The third testcase - look at our test list

The next test scenario is to add a way to say whether the account exists

- ~~Account not found, return zero balance~~
- ~~Account found, return the balance~~
- Maybe add a way to say whether the account exists?

The third testcase

```
dcl-proc Test_setFoundParmIfIdNotFound export;
  dcl-pi *n extproc(*dclcase) end-pi;
  dcl-s balance packed(11:2);
  dcl-s found ind inz(*on);

  balance = getBalance ('123' : found);
  assert (balance = 0 : 'Balance should be 0');
  assert (found = *off : 'Found should be *off');
end-proc;
```

Add a "found" parameter to the getBalance call

Compiling this will fail since our prototype doesn't have two parameters.

.

Add the "found" parameter to the prototype in the copy file

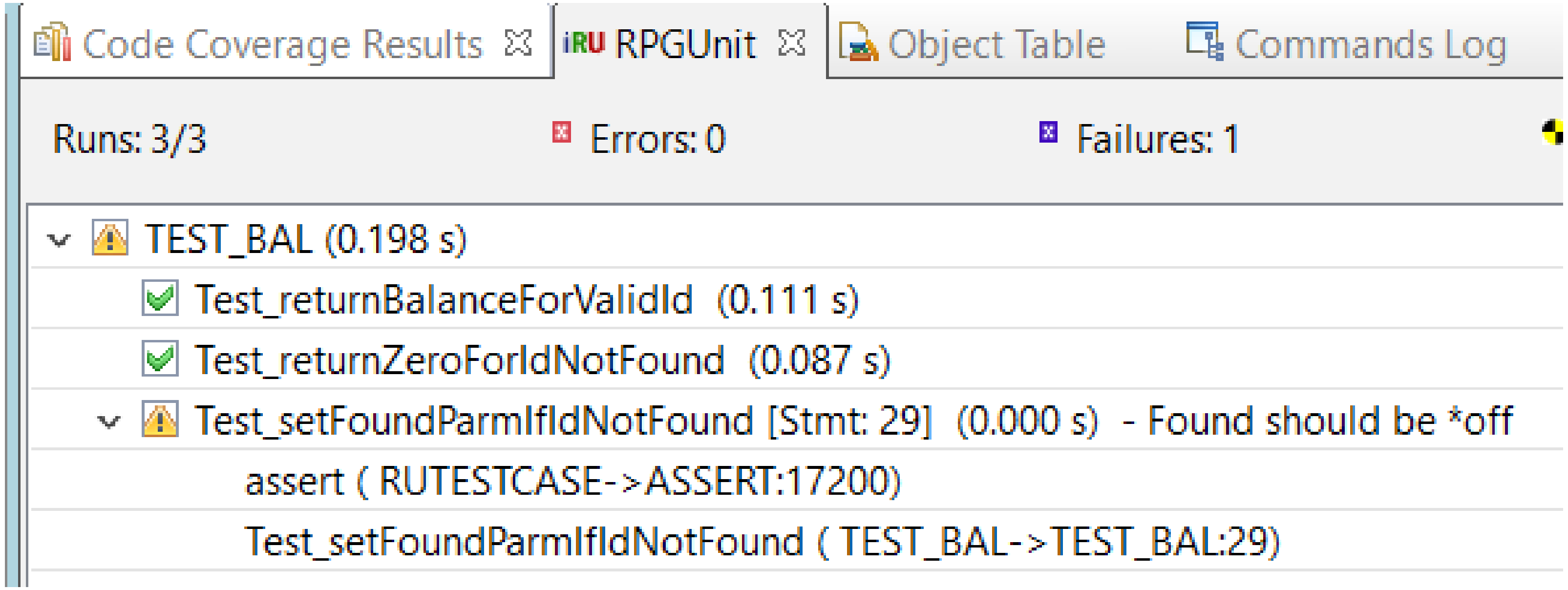
```
**free
dcl-s balance_t packed(11:2) template;
dcl-s id_t char(10) template;
dcl-pr getBalance like(balance_t) extproc(*dclcase);
   id like(id_t) const;
   found ind options(*nopass);
end-pr;
```

Make it optional, in case the caller doesn't care.

DEMO
TIME
dv3_h

Run the third test

- It should fail



```
Runs: 3/3          Errors: 0          Failures: 1

TEST_BAL (0.198 s)
    Test_returnBalanceForValidId  (0.111 s)
    Test_returnZeroForIdNotFound  (0.087 s)
    Test_setFoundParmIfIdNotFound [Stmt: 29]  (0.000 s)  - Found should be *off
        assert ( RUTESTCASE->ASSERT:17200)
        Test_setFoundParmIfIdNotFound ( TEST_BAL->TEST_BAL:29)
```

The third version of our getBalance procedure

```
dcl-proc getBalance export;
  ...
  chain id balancer ds;
  if %found;
    return ds.balance;
  else;
    if %parms >= %parmnum(found);
      found = *off;
    endif;
    return 0;
  endif;
end-proc;
```

Wouldn't it be simpler to just set the "found" parameter.

Isn't the "IF" statement too much code for how TDD works?

No, we need the IF, because it's very dangerous to access an unpassed parameter. Our earlier testcase doesn't pass the parameter.

The third version of our getBalance procedure

```
chain id balancer ds;
   if %found;
      if %parms >= %parmnum(found);
         found = *off;
      endif;
      return ds.balance;
   else;
      if %parms >= %parmnum(found);
         found = *off;
      endif;
      return 0;
   endif;
end-proc;
```

We might as well add the new code to the other branch of the IF too.

DEMO
TIME
dv3b

The updated test list

- ~~Account not found, return zero balance~~
- ~~Account found, return the balance~~
- ~~Check the found parameter when the record is not found~~
- Check the found parameter when the record is found

Run the new version of the test

(Don't forget to recreate the GETBAL module and recreate the test so it picks up the new GETBAL module.)

The new test passed! And the other tests passed too!

…

Now, let's imagine that it's the end of the working day.

And the next day, there's an emergency problem to work on.

We forget that we're not done yet.

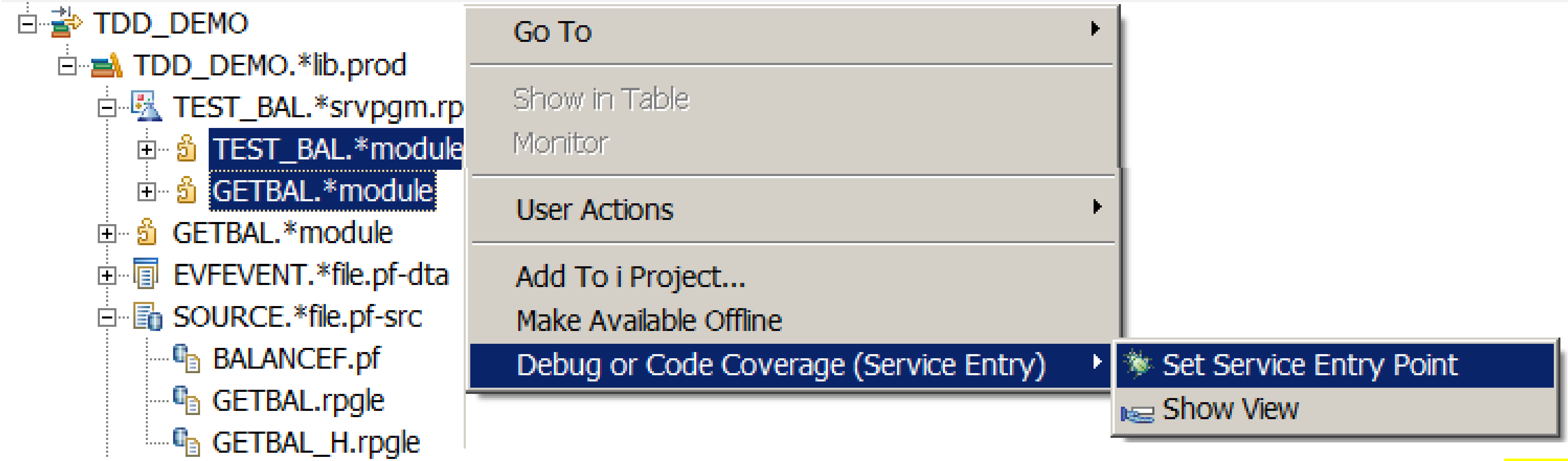
Make sure all the code in our procedure is tested

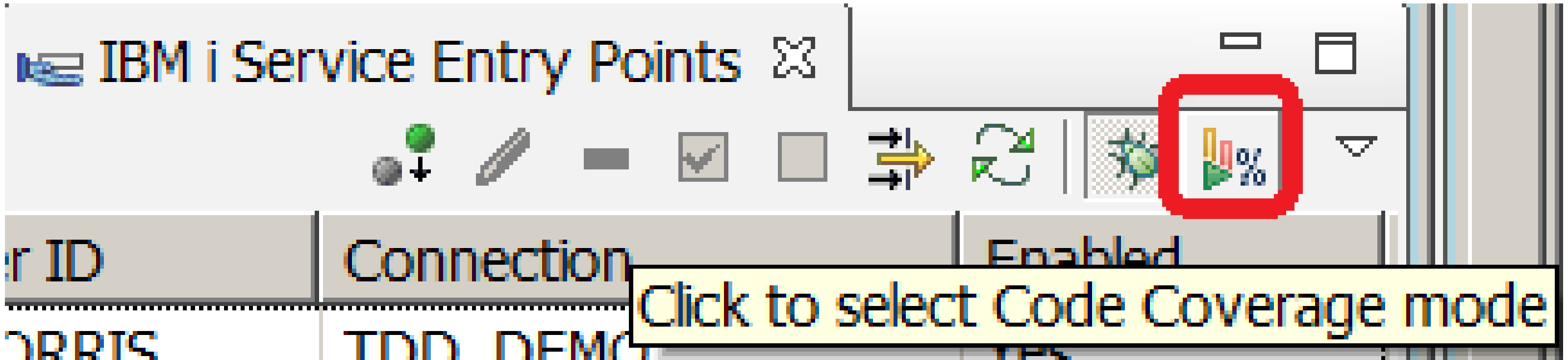
(Remember that we forgot we aren't actually finished our test list yet.)

Let's use Code Coverage to make sure we have tested all our code. (Even though we're using TDD, we're also human …)

Make sure all the code in our procedure is tested

Set a service breakpoint on our service program modules





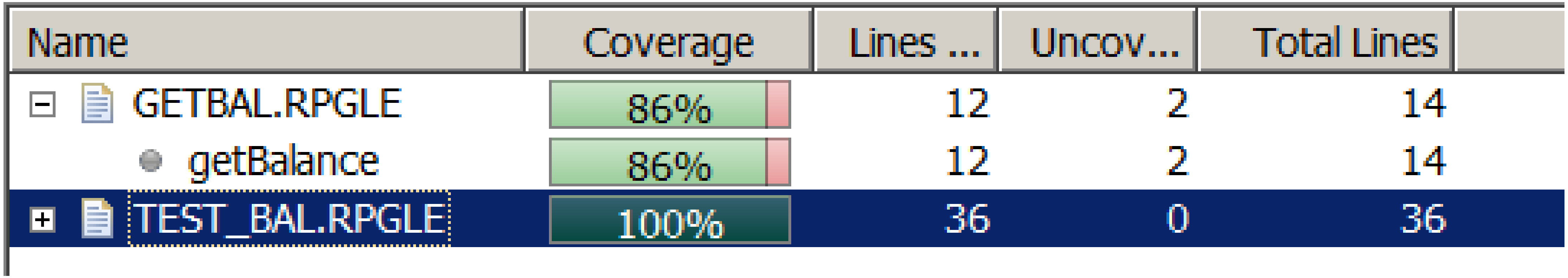
Click to select Code Coverage mode

Set "Code Coverage" mode

DEMO TIME (codecov *)

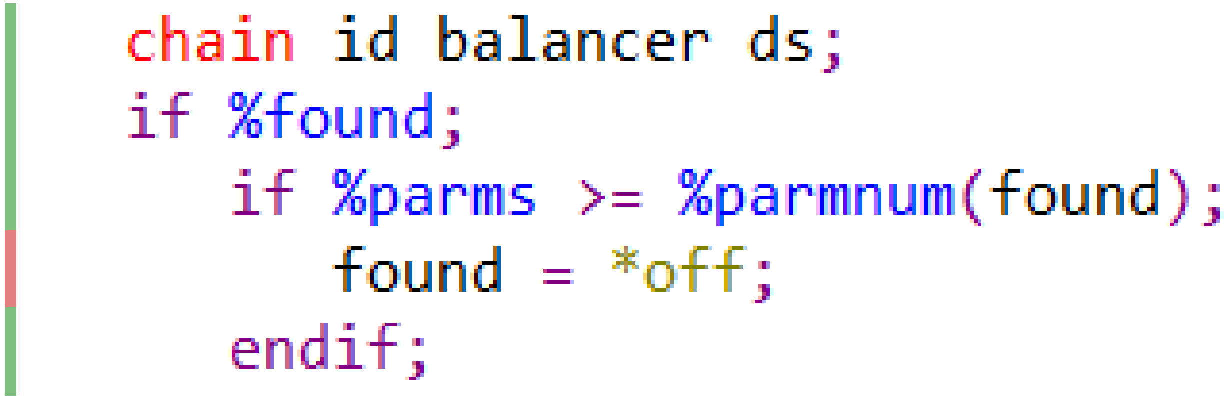
Oh oh … our tests don't test all our code

Only 86% of our code is tested!



Name	Coverage	Lines ...	Uncov...	Total Lines
⊟ GETBAL.RPGLE	86%	12	2	14
● getBalance	86%	12	2	14
⊞ TEST_BAL.RPGLE	100%	36	0	36

Drill down into the code. We haven't tested the "found = *off" line



```
chain id balancer ds;
if %found;
   if %parms >= %parmnum(found);
      found = *off;
   endif;
```

DEMO TIME (debug *)

We didn't follow TDD guidelines!

```
if %found;
   if %parms >= %parmnum(found);
      found = *off;
   endif;
   return ds.balance;
else;
   if %parms >= %parmnum(found);
      found = *off;
   endif;
   return 0;
endif;
```

Remember that we said this?

> "*We might as well add the new code to the other branch of the IF too.*"

We should not have done that!

It violates one of the principles of TDD.

The fourth testcase

Add the missing testcase from our test list

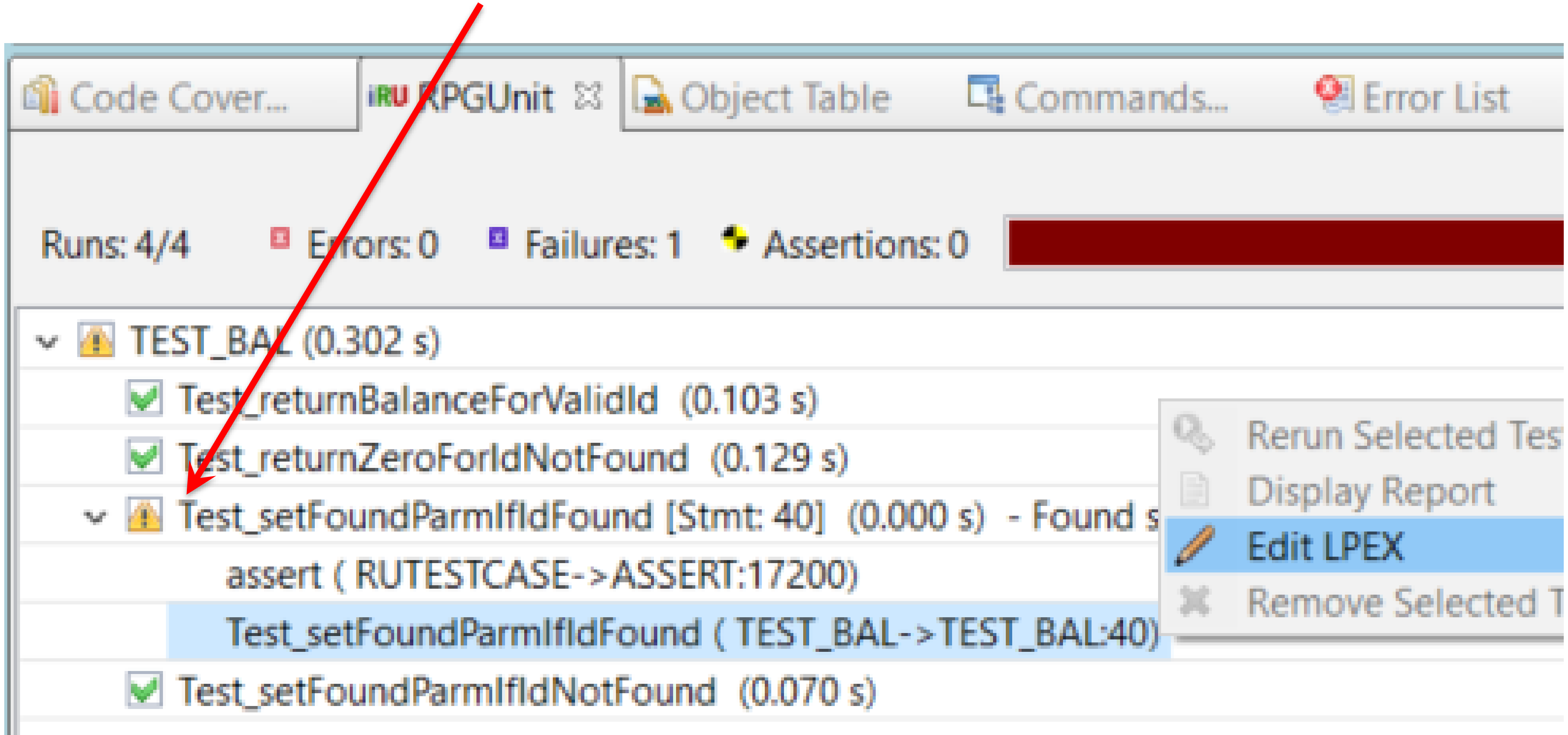
```
dcl-proc Test_setFoundParmIfIdFound export;
  dcl-pi *n extproc(*dclcase) end-pi;
  dcl-s balance packed(11:2);
  dcl-s found ind inz(*off);
  add_record ('123' : 123);
  balance = getBalance ('123' : found);
  assert (balance = 123 : 'Balance should be 123');
  assert (found = *on : 'Found should be *on');
end-proc;
```

DEMO TIME
(tv4)

Run the new version of the test

(Don't forget to recreate the GETBAL module and recreate the test so it picks up the new GETBAL module.)

The new test failed!



Runs: 4/4 Errors: 0 Failures: 1 Assertions: 0

```
TEST_BAL (0.302 s)
    Test_returnBalanceForValidId  (0.103 s)
    Test_returnZeroForIdNotFound  (0.129 s)
    Test_setFoundParmIfIdFound [Stmt: 40]  (0.000 s)  - Found s
        assert ( RUTESTCASE->ASSERT:17200)
        Test_setFoundParmIfIdFound ( TEST_BAL->TEST_BAL:40)
    Test_setFoundParmIfIdNotFound  (0.070 s)
```

Here's our getBalance procedure

```
chain id balancer ds;
   if %found;
      if %parms >= %parmnum(found);
         found = *off;
      endif;
      return ds.balance;
   else;
      if %parms >= %parmnum(found);
         found = *off;
      endif;
      return 0;
   endif;
end-proc;
```

Oops!

It should be "found = *on;

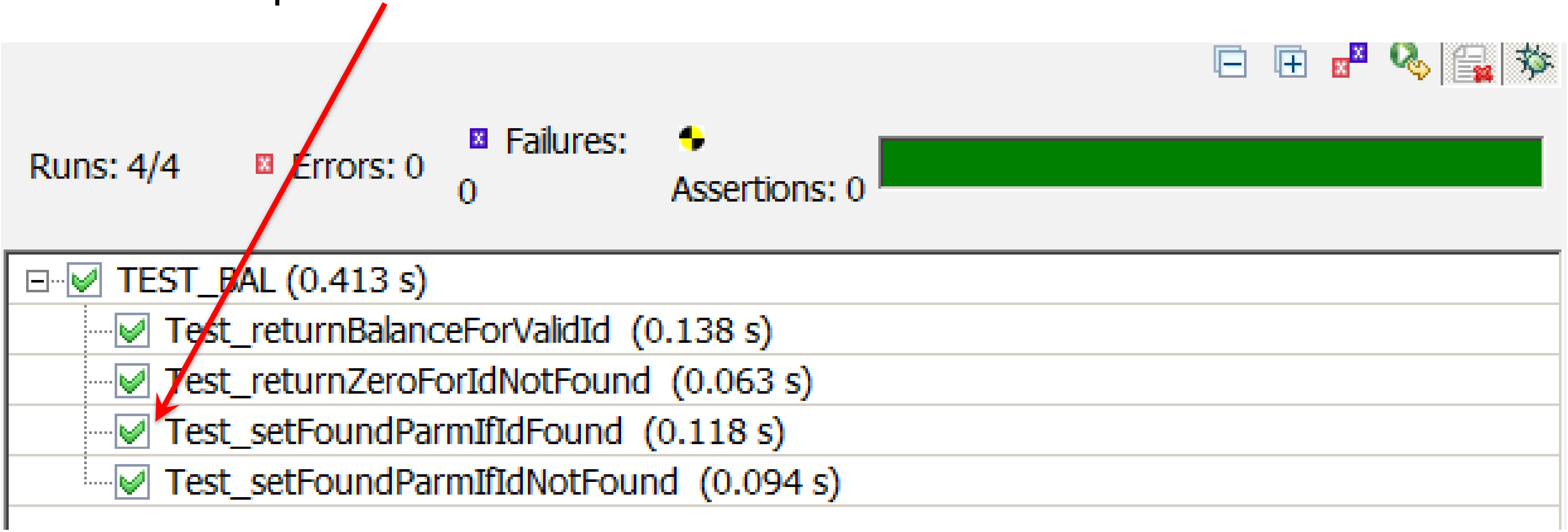
This is why we never add extra code when making our changes.

DEMO TIME

After fixing the code

(Don't forget to recreate the GETBAL module and recreate the test so it picks up the new GETBAL module.)

The new test passed!

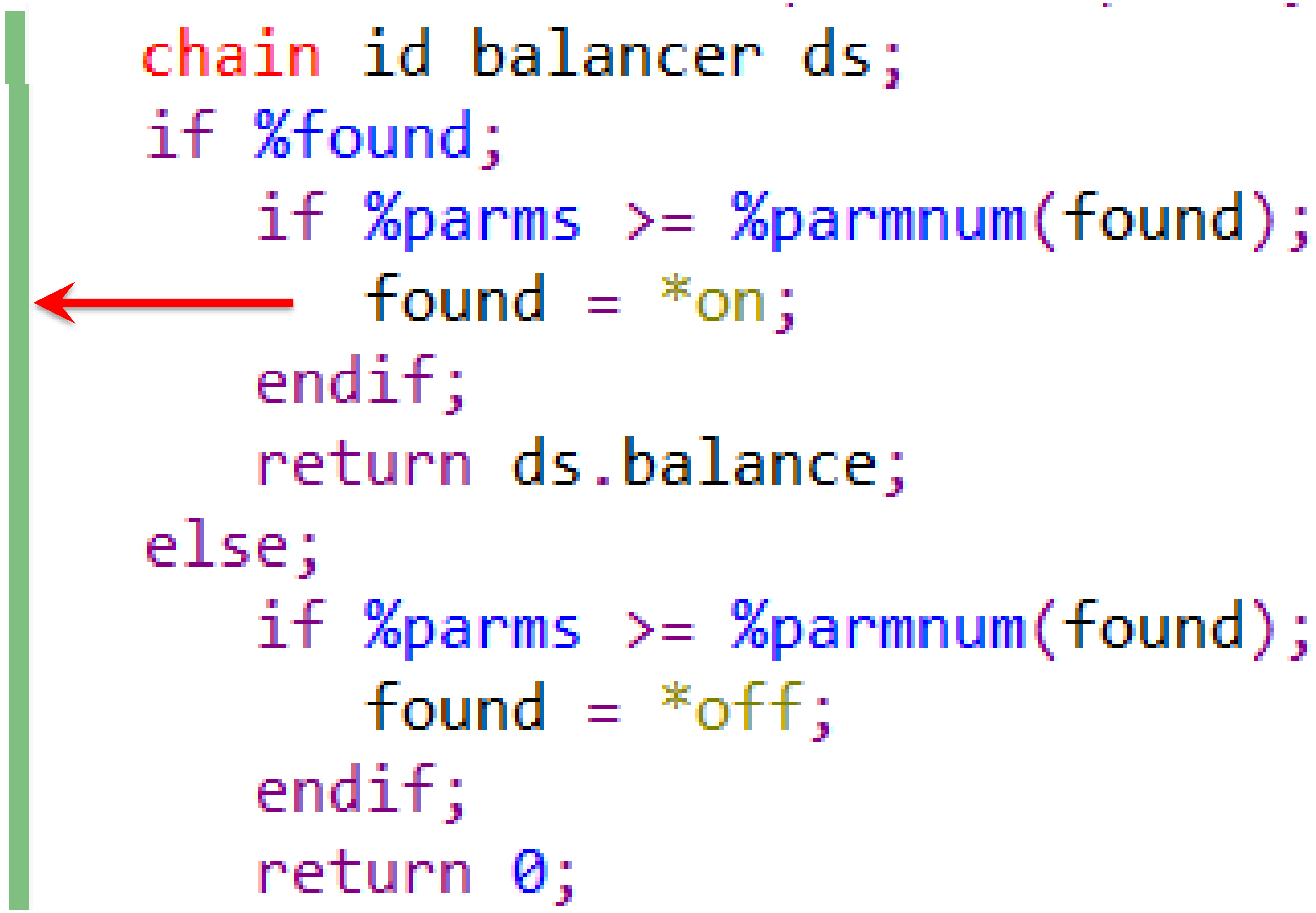


Runs: 4/4 Errors: 0 Failures: 0 Assertions: 0

```
⊟ ✓ TEST_BAL (0.413 s)
    ✓ Test_returnBalanceForValidId  (0.138 s)
    ✓ Test_returnZeroForIdNotFound  (0.063 s)
    ✓ Test_setFoundParmIfIdFound  (0.118 s)
    ✓ Test_setFoundParmIfIdNotFound  (0.094 s)
```

Run Code Coverage again

- Now, the "found = *on" line is tested!



```
chain id balancer ds;
if %found;
   if %parms >= %parmnum(found);
      found = *on;
   endif;
   return ds.balance;
else;
   if %parms >= %parmnum(found);
      found = *off;
   endif;
   return 0;
```

The updated test list

~~- Account not found, return zero balance~~

~~- Account found, return the balance~~

~~- Check the found parameter when the record is not found~~

~~- Check the found parameter when the record is found~~

We are done!

Almost …

Here's our getBalance procedure

```
chain id balancer ds;
if %found;
   if %parms >= %parmnum(found);
      found = *on;
   endif;
   return ds.balance;
else;
   if %parms >= %parmnum(found);
      found = *off;
   endif;
   return 0;
endif;
```

This code is very similar.

Let's see if we can only code the %parms check once.

"Refactoring"

Changing code without changing what it does is called "refactoring".

Considering whether we should refactor our code is part of the TDD loop.

Thinking about refactoring is done after each time we get all our tests to pass.

Our refactored procedure

```
chain id balancer ds;
if %found;
  if %parms >= %parmnum(found);
    found = *on;
  endif;
  return ds.balance;
else;
  if %parms >= %parmnum(found);
    found = *off;
  endif;
  return 0;
endif;
```

Before

```
chain id balancer ds;
if %parms >= %parmnum(found);
  found = %found;
endif;
return ds.balance;
```

After

Now, we're done!

We're done, but let's think about the mistake we made

We stopped work before completing our test list

(That's normal to have to take breaks … the work might last days or weeks)

But we forgot to finish our testing!

And there was a bug in our code

If no other production code calls our procedure with the "found" parameter when the record was found, we will never find out about the bug

Until … someone adds the "found" parameter to their code, but they don't test their own code properly and the bug gets discovered by a customer

How could we avoid this?

New rule

A new rule:

Always stop work with a failing testcase

Even if you only have time to add a testcase like this:

```
dcl-proc Test_finish_testing;
  assert (*off : 'Finish this testing!');
end-proc;
```

But make sure you also take the few seconds to recreate the test, to make sure it fails when you run it as part of your general testing.

Updated "TDD loop"

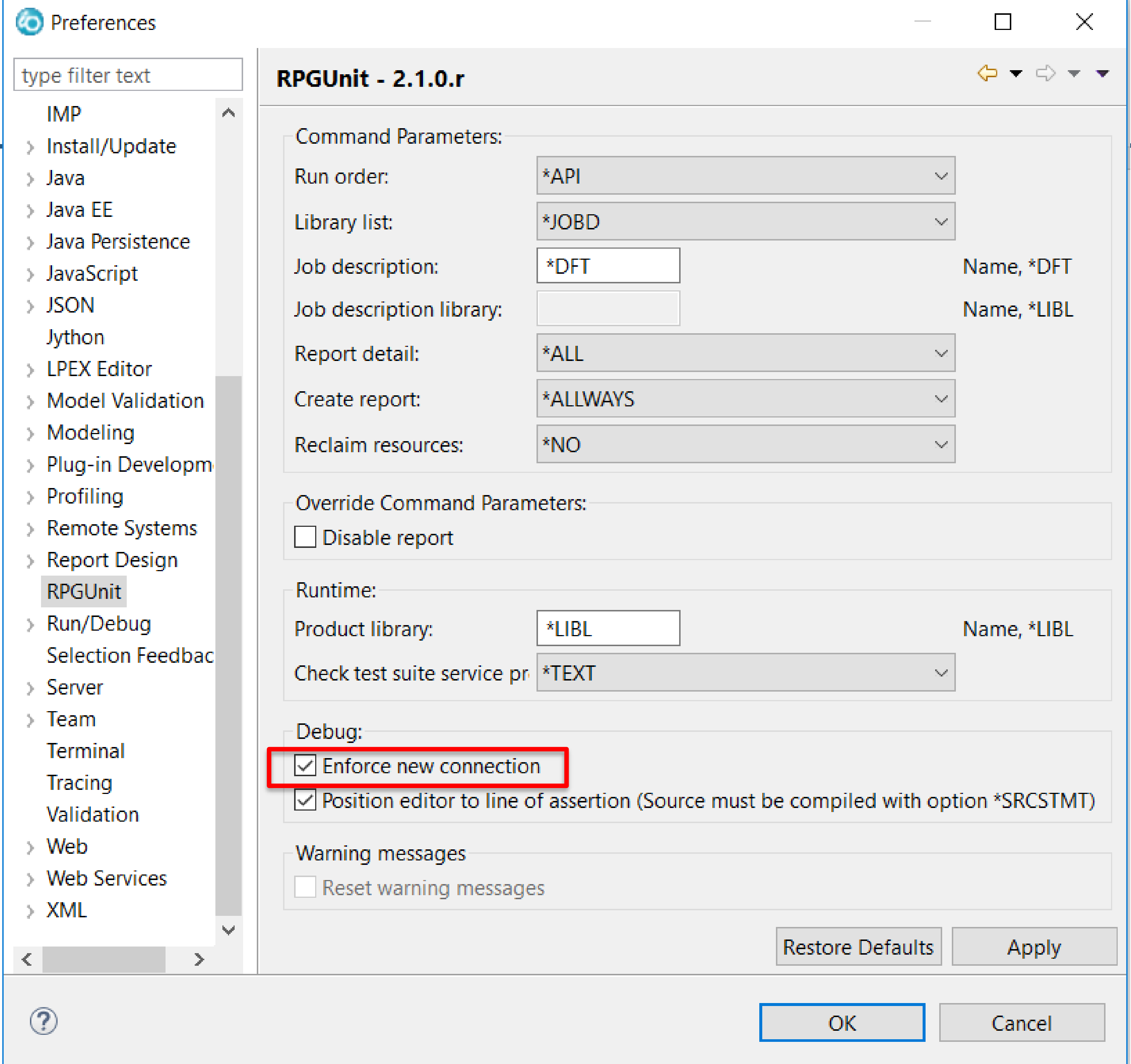
```
Loop
    Write a failing test
    Write the minimum amount of code to make the test pass
        If you want to add a bit of extra code … don't do it!
            Just add a test to your test list
    Run all existing tests until they all pass
    Refactor if necessary
    If stopping work, make sure you have a failing testcase   (NEW)
End-loop
```

Some "gotchas" we ran into using RPGUNIT

- Library RPGUNIT must be in the library list
  - The preferences by default point to a job description named RPGUNIT in the library where the tests are. If this is missing, an error occurs.
  - Using *CURRENT for the *LIBL avoids this requirement
- The test procedure ends as soon as the first assert fails
- For calls to assert(), avoid using CALLP(E) or putting them in a monitor, since that will prevent them from being logged

Before debugging

- To debug or
- Analyze code coverage
  - Set this preference
  - Otherwise will hang RDi
- For details, see link



RPGUnit - 2.1.0.r

Command Parameters:

Run order: *API

Library list: *JOBD

Job description: *DFT — Name, *DFT

Job description library: — Name, *LIBL

Report detail: *ALL

Create report: *ALLWAYS

Reclaim resources: *NO

Override Command Parameters:

☐ Disable report

Runtime:

Product library: *LIBL — Name, *LIBL

Check test suite service pr *TEXT

Debug:

☑ Enforce new connection

☑ Position editor to line of assertion (Source must be compiled with option *SRCSTMT)

Warning messages

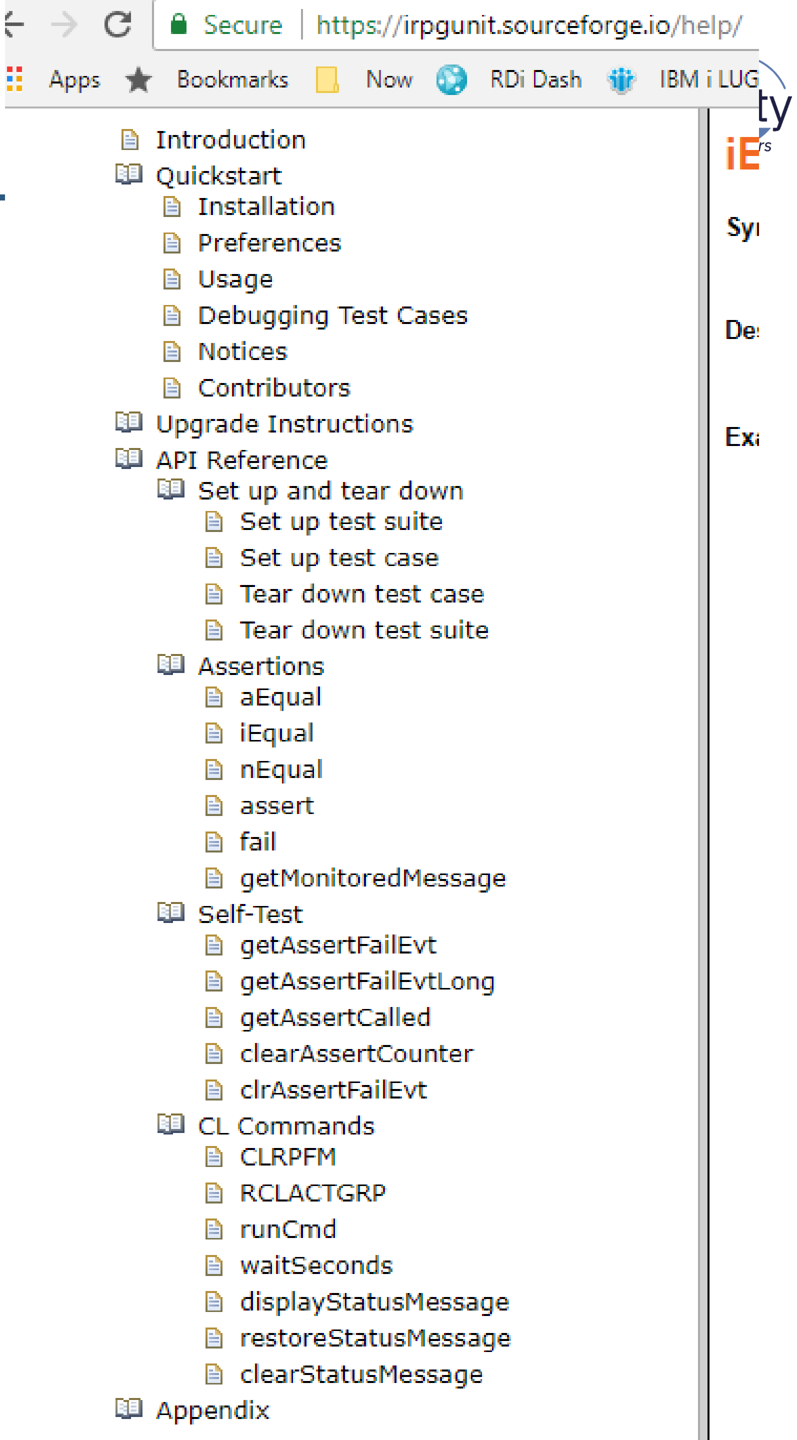
☐ Reset warning messages

RPGUNIT documentation and tutorials

- https://irpgunit.sourceforge.io/
- https://irpgunit.sourceforge.io/help/
- http://rpgunit.sourceforge.net/tutorial.html

- Comments, suggestions to rpgunit@tools400.de
- Discussion of RPGUnit RDi plugin at WDSCI-L mailing list
- For bug reports open a ticket at iRPGUnit bug tracker.

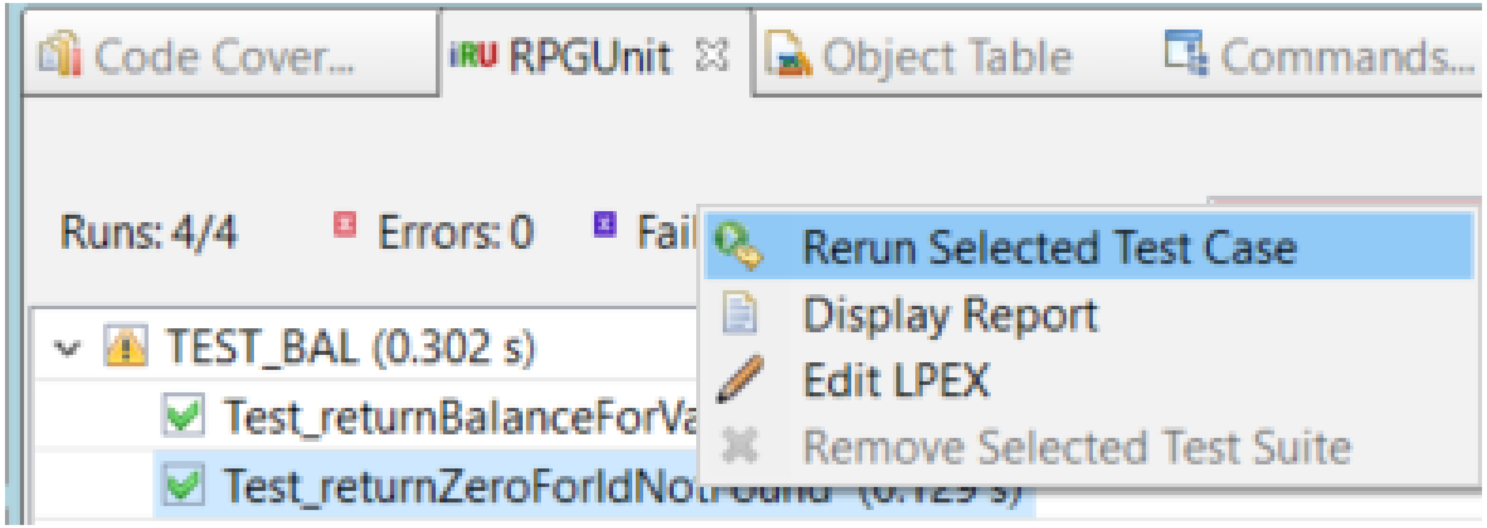
- Included APIs
  - runCmd() – execute command via QCMDEXC
  - waitSeconds() – suspends job for specified number of seconds
  - getAssertFailEvtLong() - Extract the testcase status including the stacktrace
  - etc.



- Introduction
- Quickstart
  - Installation
  - Preferences
  - Usage
  - Debugging Test Cases
  - Notices
  - Contributors
- Upgrade Instructions
- API Reference
  - Set up and tear down
    - Set up test suite
    - Set up test case
    - Tear down test case
    - Tear down test suite
  - Assertions
    - aEqual
    - iEqual
    - nEqual
    - assert
    - fail
    - getMonitoredMessage
  - Self-Test
    - getAssertFailEvt
    - getAssertFailEvtLong
    - getAssertCalled
    - clearAssertCounter
    - clrAssertFailEvt
  - CL Commands
    - CLRPFM
    - RCLACTGRP
    - runCmd
    - waitSeconds
    - displayStatusMessage
    - restoreStatusMessage
    - clearStatusMessage
- Appendix

More RPGUNIT features

- From context menu can do:



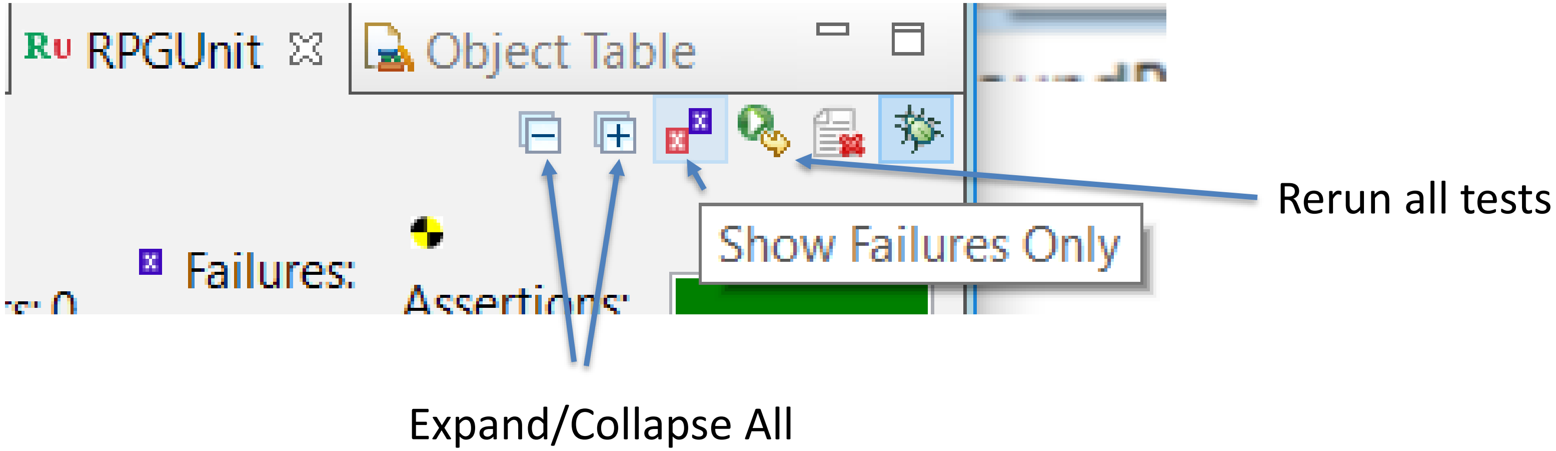
Runs: 4/4 Errors: 0 Fail

TEST_BAL (0.302 s)

Test_returnBalanceForVa

Test_returnZeroForIdNotFound (0.129 s)

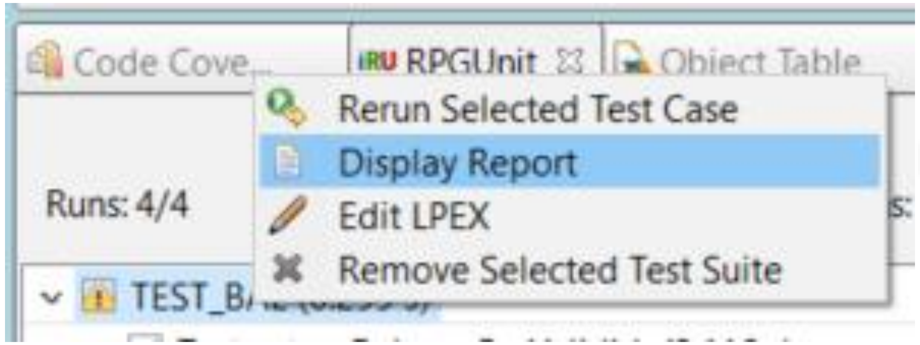
- Can filter only failing tests etc.

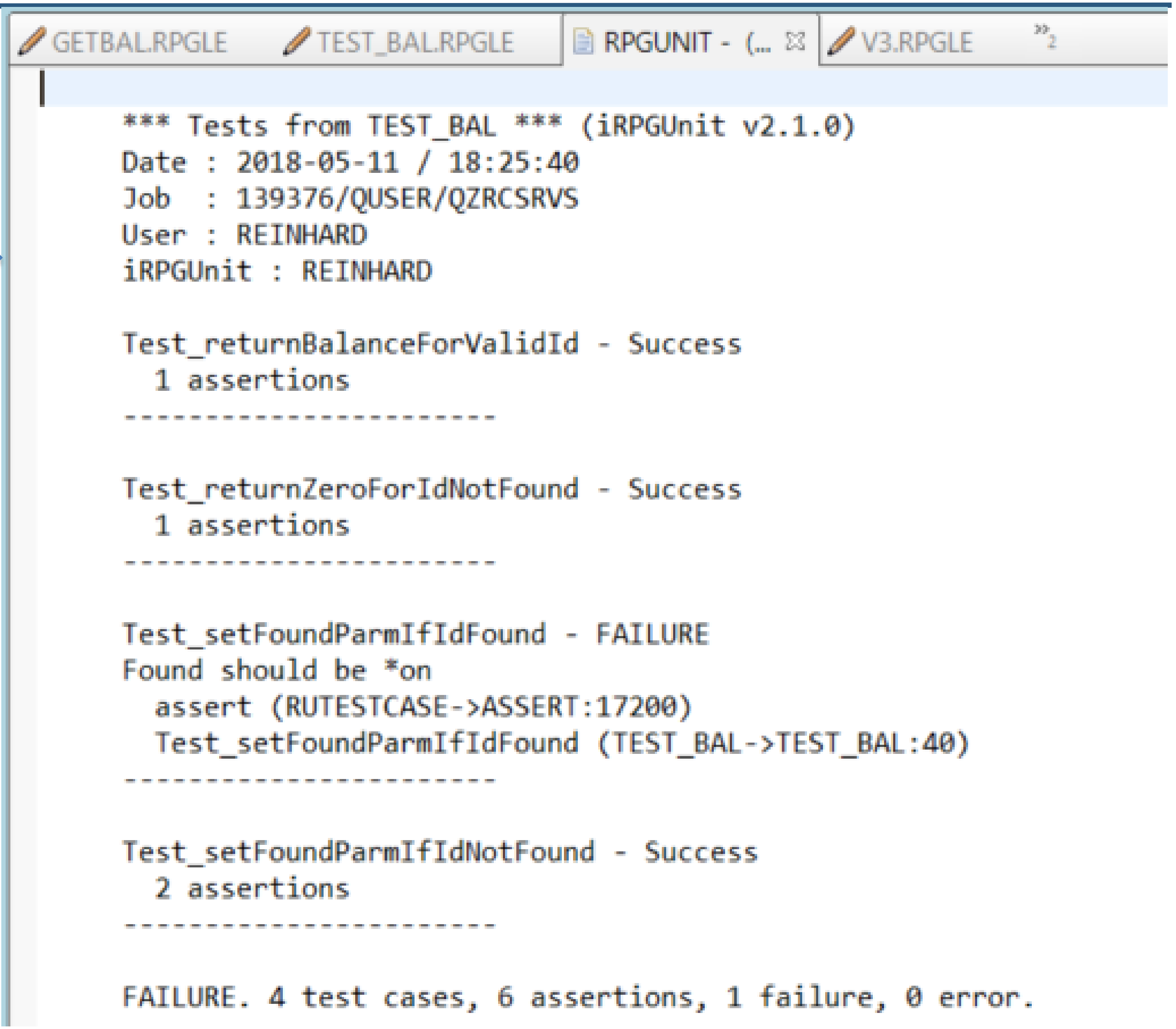


Failures:

Assertions:

Show RPGUNIT reports





```
*** Tests from TEST_BAL *** (iRPGUnit v2.1.0)
Date : 2018-05-11 / 18:25:40
Job  : 139376/QUSER/QZRCSRVS
User : REINHARD
iRPGUnit : REINHARD

Test_returnBalanceForValidId - Success
  1 assertions
-----------------------

Test_returnZeroForIdNotFound - Success
  1 assertions
-----------------------

Test_setFoundParmIfIdFound - FAILURE
Found should be *on
  assert (RUTESTCASE->ASSERT:17200)
  Test_setFoundParmIfIdFound (TEST_BAL->TEST_BAL:40)
-----------------------

Test_setFoundParmIfIdNotFound - Success
  2 assertions
-----------------------

FAILURE. 4 test cases, 6 assertions, 1 failure, 0 error.
```

Take Aways

- Software Best Practices will save you time and money
- Professional IBM i developers are professional software engineers
- Test Driven Development gives you confidence to move your software forward
- IBM i tooling exists to
  - Run automated unit tests – RPGUNIT
  - Do code coverage analysis – CODECOV
  - Refactor ILE RPG – RDi
- Michael Feathers – green tufts of grass in a muddy field