

Software techniques on legacy systems.

**Contribution : GSE Academic Award for Excellence 2011
31.07.2011 – S. Vaupel**

**University of Marburg
Department of Computer Science and Mathematics
svaupel@mathematik.uni-marburg.de**

Software techniques on legacy systems.

Content

- Definition
 - Problem
 - Motivation
 - Project aim
 - Project infrastructure
 - Testing programs
 - Retrospective testing
 - Introspective testing
- Testing user interfaces
 - Definition test
 - Functional test
 - Demonstration
 - Conclusion

Software techniques on legacy systems.

Definition

- Software techniques includes
 - Requirements engineering
 - Architecture considerations
 - Design proposals and practices
 - Software development processes
 - Development (CASE-Tools)
 - Testing
 - Maintenance
 - Configuration and build management
 - Software quality
 - ...

Software techniques on legacy systems.

Problem

- Legacy systems provides poor computer aided software engineering (CASE) tools, especially testing.
- Testing is done
 - manually,
 - one-time,
 - at the end of the development process.
- Refactoring is critical, because there is no guarantee for same functionality afterwards.
- Faults can not detect reliable or repeatable (Bug known - cause mysterious) .

Software techniques on legacy systems.

Motivation

- Automated testing is a requirement to act more agile.
- Automated testing allows further actions on legacy systems:
 - Cheaper and reliable maintenance
 - Refactoring and modernization
 - Build management with integrity and consistency guarantees after changes.

Software techniques on legacy systems.

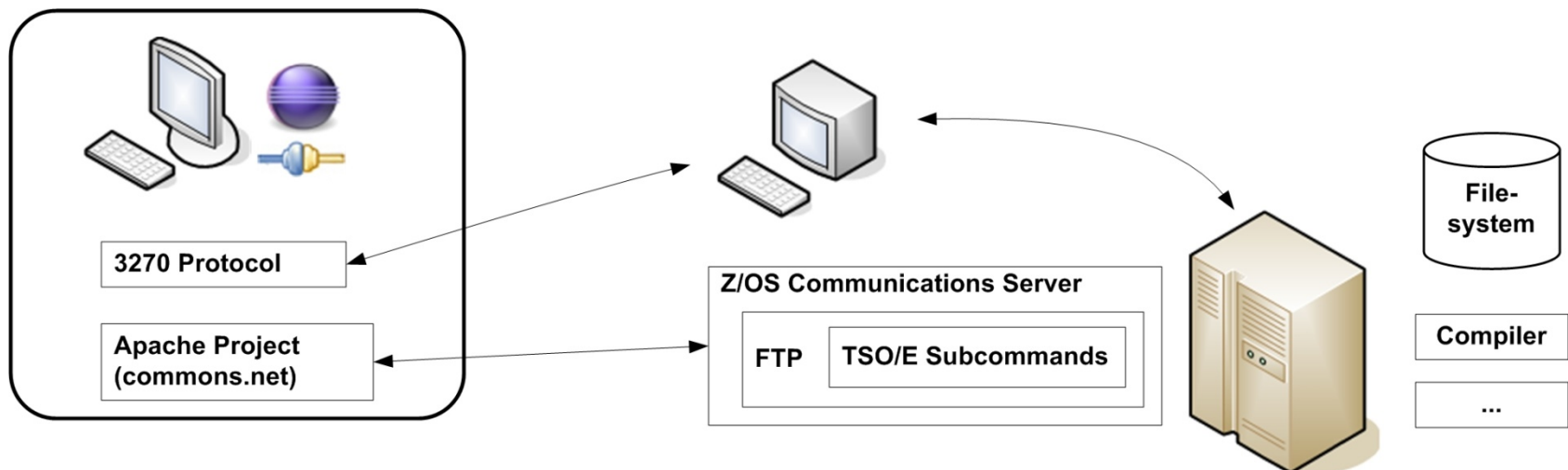
Project aim

- The aim of the project was
 - to develop a concept for testing legacy code and user interfaces
 - without changing the code (non-invasive testing)
 - without expanding the mainframe environment directly (i.e. system software)
 - and to proof the concept with a runnable implementation.

Software techniques on legacy systems.

Project infrastructure

- The tool works remote on a z/OS system.
 - The 3270 protocol was used to access the user interfaces.
 - The z/OS Communications Server was used for file access and program execution (TSO/E).
- The implementation uses Eclipse as local platform.



Software techniques on legacy systems.

Testing programs

- The tested modules are callable COBOL programs.
- The test paradigm is the XUnit paradigm. This means:
 - a test setup (fixture)
 - a test referencedefine a test case.

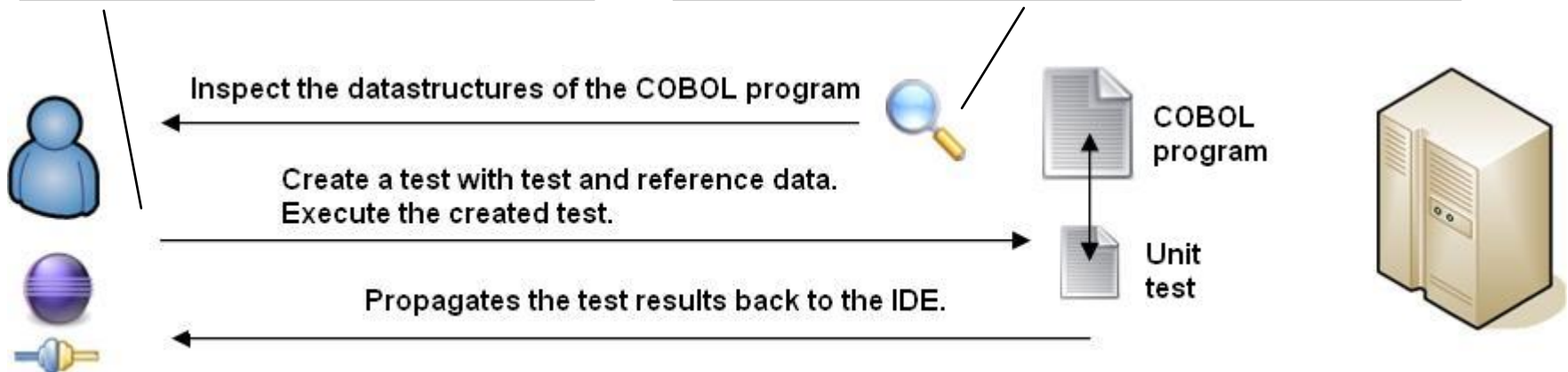
While a test execution the DUT (device under test) is calling with the test setup. Afterwards the program output is comparing with the expected output (reference).

Software techniques on legacy systems.

Testing programs (Retrospective)

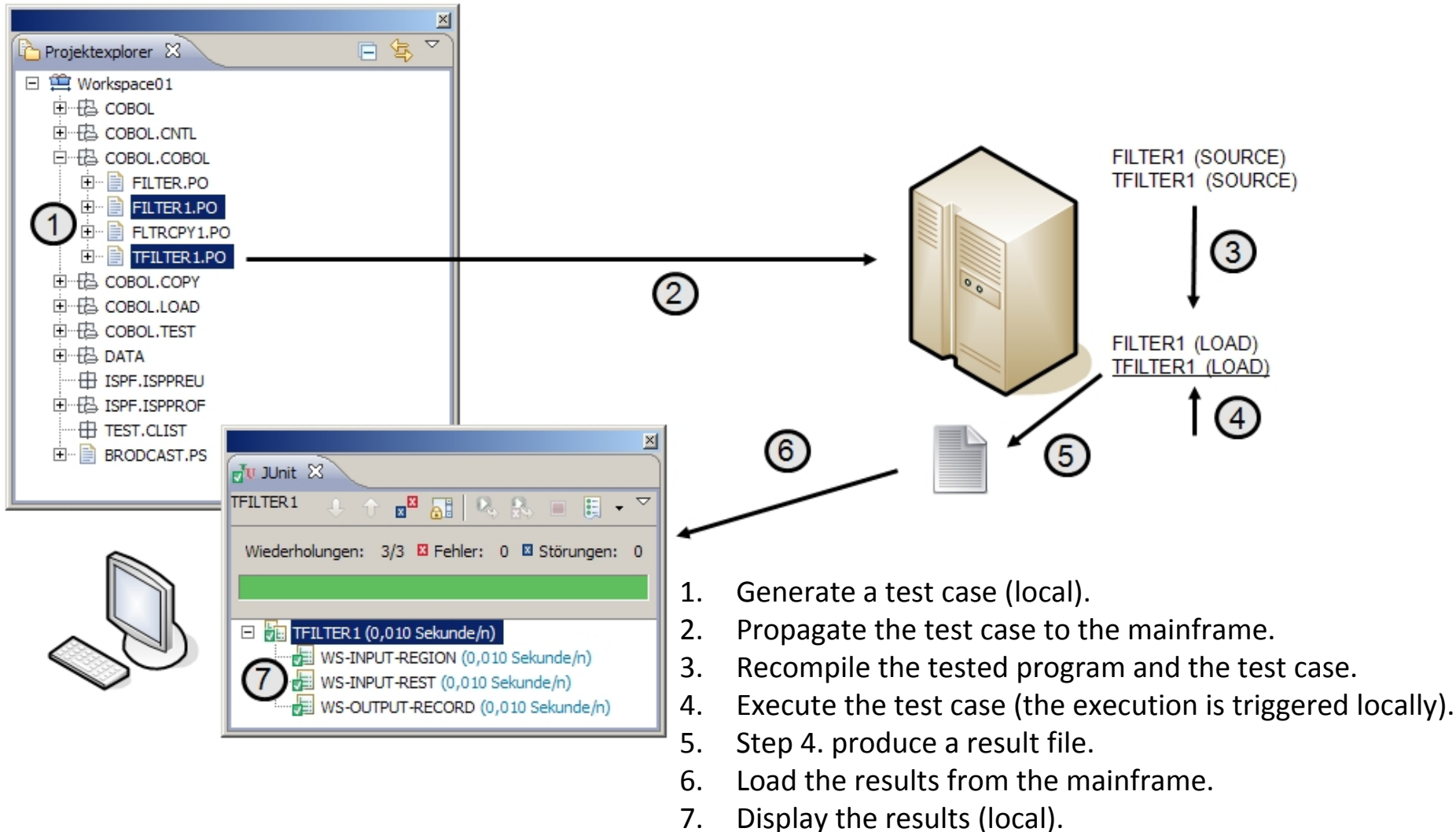
```
Test data (fixture)
01 WS-INPUT-RECORD   = 91234
01 WS-OUTPUT-RECORD = 00000
Reference data
01 WS-INPUT-RECORD   = 91234
01 WS-OUTPUT-RECORD = 47110
```

```
LINKAGE SECTION.
01 WS-INPUT-RECORD.
   05 WS-INPUT-REGION   PIC X.
   05 WS-INPUT-REST     PIC X(4).
01 WS-OUTPUT-RECORD    PIC X(5).
```



Software techniques on legacy systems.

Testing programs (Retrospective)



Software techniques on legacy systems.

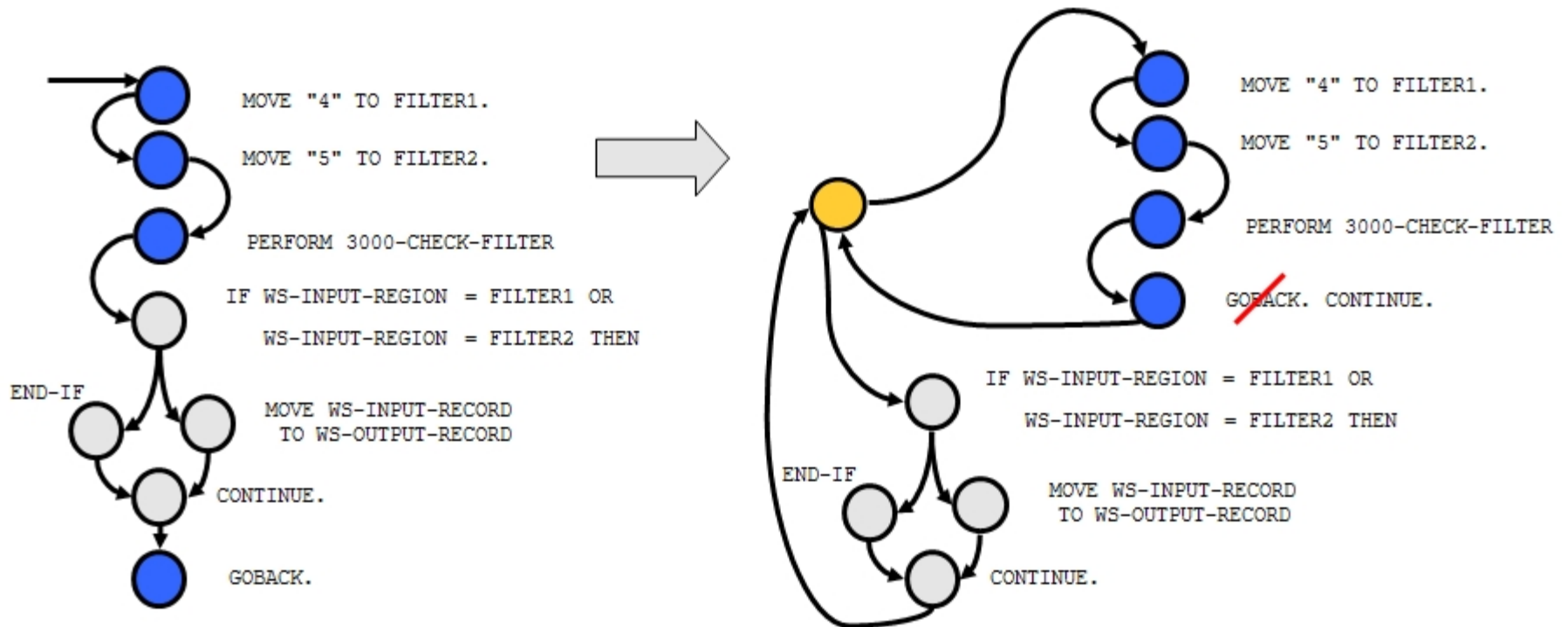
Testing programs (Retrospective)

- Note: The tested program was not changed in his structure. The given data determines which lines of code, subsections etc. were called within a test execution.
- Problem: Some code can never reach and test – independent to the chosen test data!
This kind of test is a “**retrospective**” test, because the called subsections are determine by the given test data.
- Goal: Testing every part of a program, surely without merging test and program code.

Software techniques on legacy systems.

Testing programs (Introspective)

- Solution: Temporally program transformation.



Software techniques on legacy systems.

Testing programs (Introspective)

- Every subsection is now a separately test case.
- For all subsections a test setup and reference data is require.
- This kind of test is a “**introspective**” test, because all subsections are called. The structure of the original program is not longer important.
- While a test execution the test data and the origin program code joined temporally.

Software techniques on legacy systems.

Testing programs (Introspective)

Subsection A

```
05 WS-INPUT-REGION    PIC X.  
05 WS-INPUT-REST     PIC X(4).
```

Subsection B

```
05 WS-INPUT-REST     PIC X(4).
```

Subsection C

...

Test data (fixture)

Test data (fixture)

Test data (fixture)

01 WS-INPUT-RECORD = 91234

01 WS-OUTPUT-RECORD = 00000

Reference data

01 WS-INPUT-RECORD = 91234

01 WS-OUTPUT-RECORD = 47110

4

0

4

0

4

0

4

0



Inspect the paragraphs and data structures



Create a transformation program based on the testsopes, the testdata and the reference data.



Propagates the test results back to the IDE.



COBOL program



Transformation program

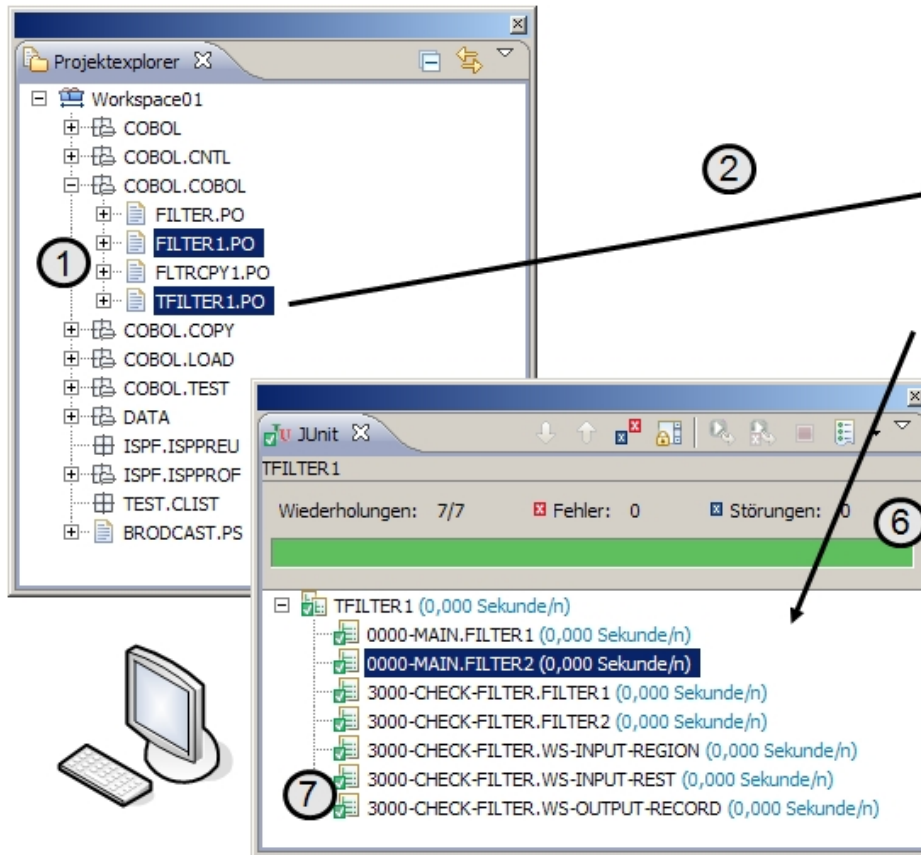


Unit test



Software techniques on legacy systems.

Testing programs (Introspective)



1. Generate a test case (local).
2. Propagate the test case to the mainframe.
3. Recompile the tested program and the test case.
4. Execute the test case (the execution is triggered locally).
 - a.) Transform the origin program
 - b.) Compile the transformed program
 - c.) Execute the transformed program
5. Step 4. c. produce a result file.
6. Load the results from the mainframe.
7. Display the results (local).

Software techniques on legacy systems.

Testing user interfaces (Definition test)

- The easiest way to test an existing user interface is the comparison with a definition.
- The comparison includes the data itself and the metadata (mask structure) too.
- The definition can be taken from an existing mask (post-definition) or defined in a editor (pre-definition).



Definition with data

=



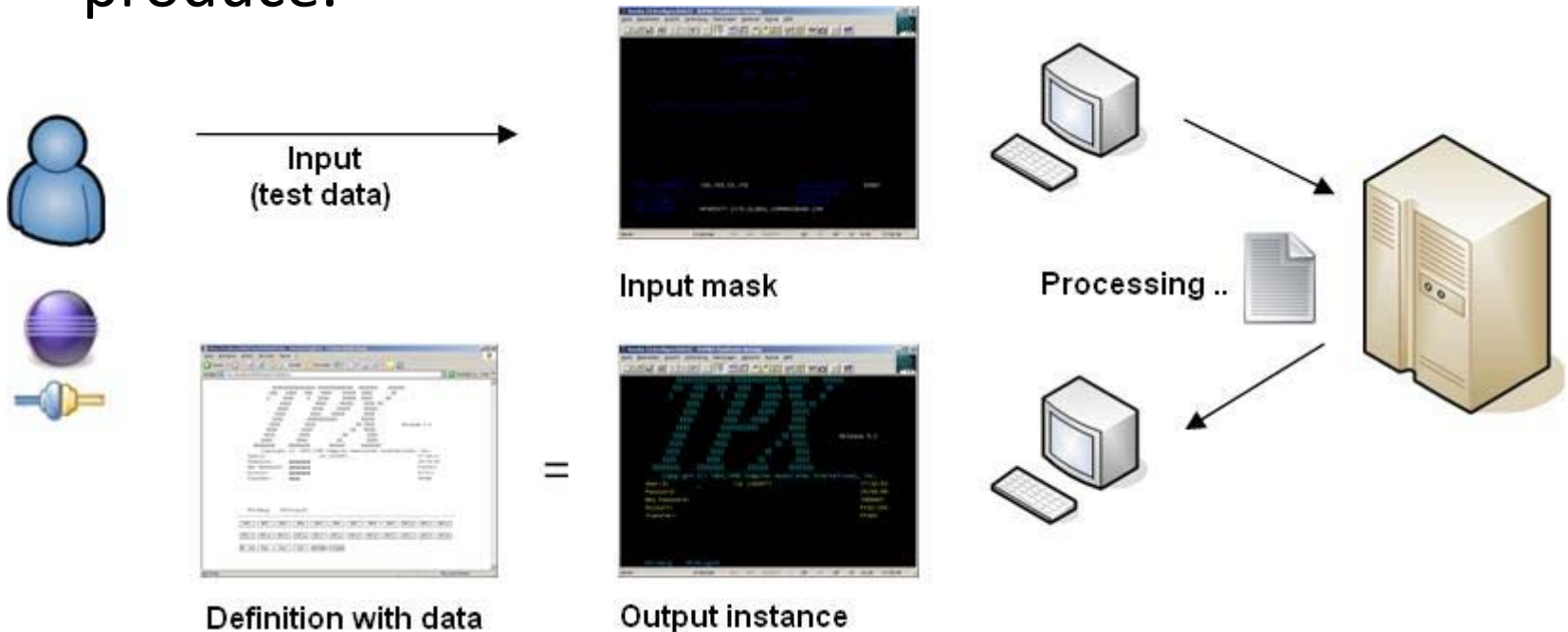
Instance with data



Software techniques on legacy systems.

Testing user interfaces (Functional test)

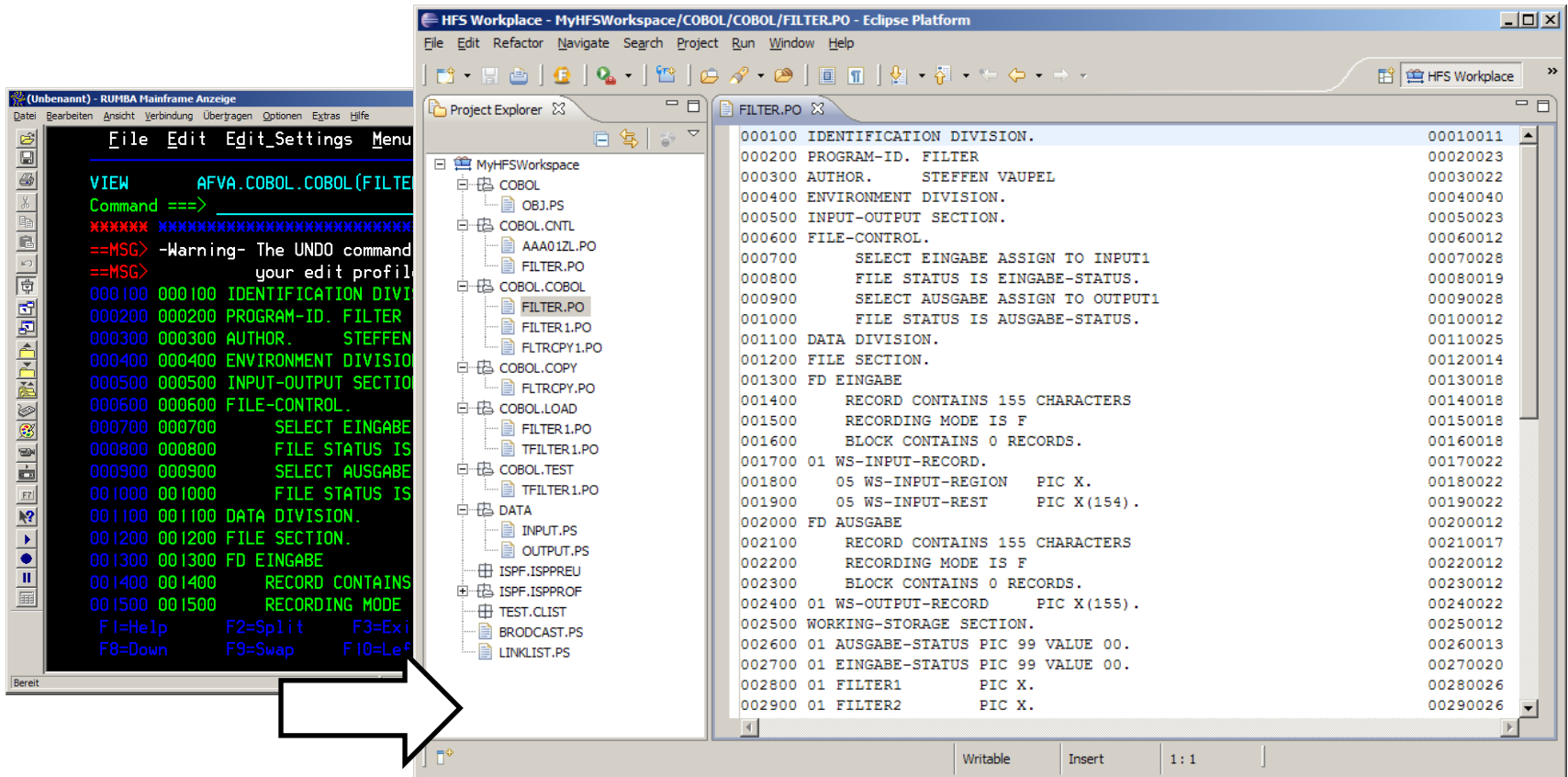
- The functional test extends a definition test.
- The test data will be insert in a mask, process by a program (not only COBOL) and an output mask will be produce.



Software techniques on legacy systems.

Demonstration (Prerequisites)

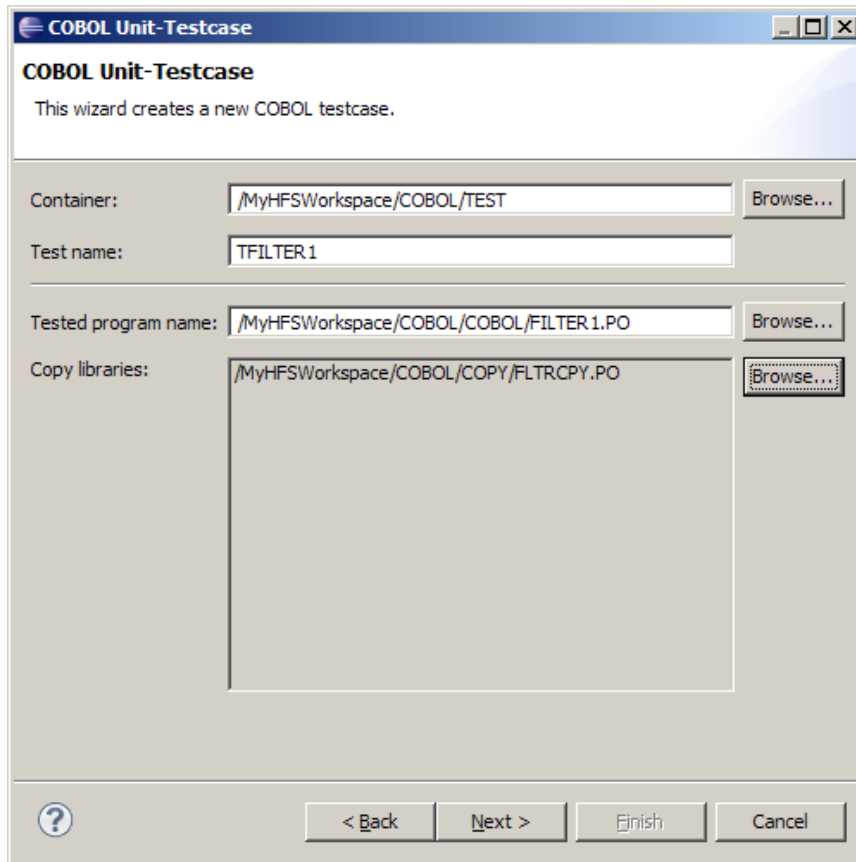
1. A “HFS Project” in Eclipse maps the mainframe members into a local workspace. Changes were propagated automatically back to the mainframe.



Software techniques on legacy systems.

Demonstration (New retrospective test)

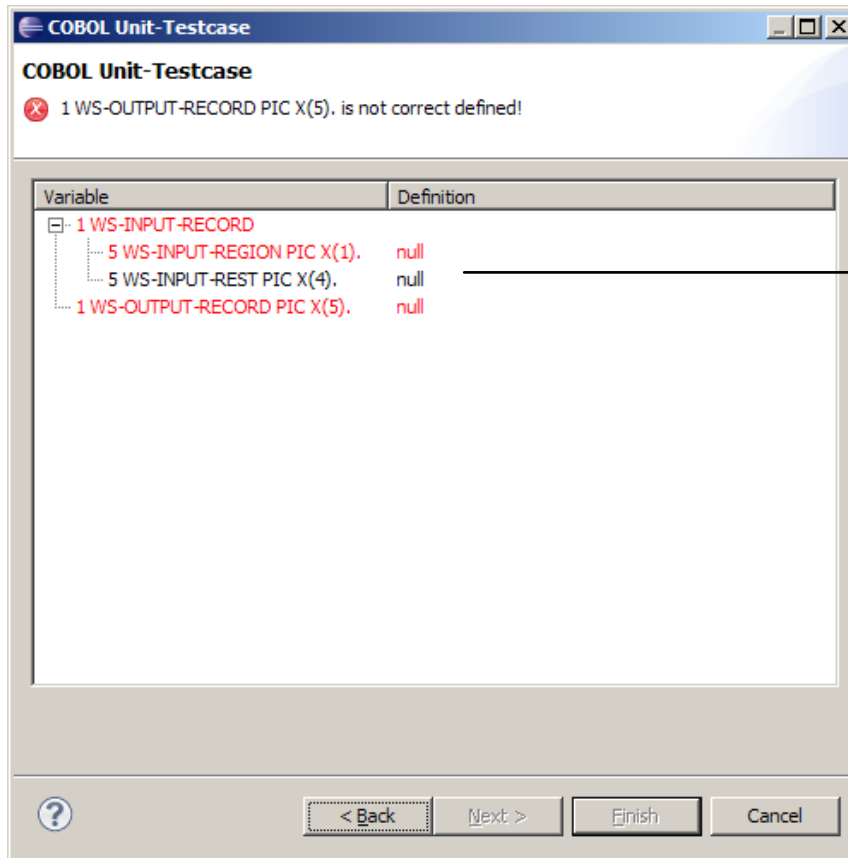
2. The wizard “New COBOL test case (Retrospective)” define a new test case.



Software techniques on legacy systems.

Demonstration (New retrospective test)

3. The test setup has to be define ...

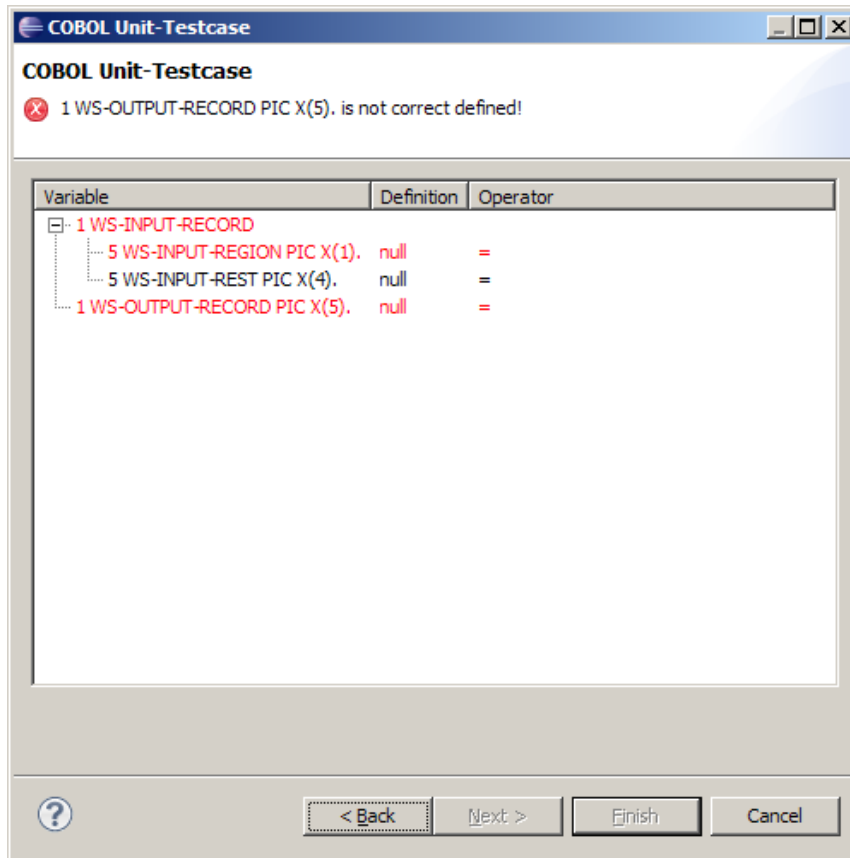


Variables from the LINKAGE SECTION.

Software techniques on legacy systems.

Demonstration (New retrospective test)

4. ... and the reference data with the comparison operators, too.



Software techniques on legacy systems.

Demonstration (New retrospective test)

5. The generated test case.

```
...
PROCEDURE DIVISION.
0000-MAIN.
  MOVE "4" TO WS-INPUT-REGION
  MOVE "9999" TO WS-INPUT-REST
  MOVE "00000" TO WS-OUTPUT-RECORD

  MOVE FUNCTION CURRENT-DATE TO WS-CUR-DATE-FIELDS
  CALL „PFILTER1" USING
  BY REFERENCE WS-INPUT-RECORD
  BY REFERENCE WS-OUTPUT-RECORD

  DISPLAY "<?xml version=""1.0"" encoding=""UTF-8""?>".
  ...

  IF WS-OUTPUT-RECORD = "49999" THEN
  DISPLAY "<testcase name=""WS-OUTPUT-RECORD"" classname=""""""
  DISPLAY "time="""" WS-CUR-SECONd1 ." WS-CUR-HS1 """/>"
  ELSE
  DISPLAY "<testcase name=""WS-OUTPUT-RECORD"" classname=""""""
  DISPLAY "time="""" WS-CUR-SECONd1 ." WS-CUR-HS1 """/>"
  DISPLAY "<error> Expected was 49999, result is "
    WS-OUTPUT-RECORD ".</error>"
  DISPLAY "</testcase>"
  END-IF
  ...
  DISPLAY "</testrun>".
GOBACK.
```

Test data (fixture)

01 WS-INPUT-RECORD = 49999

01 WS-OUTPUT-RECORD = 00000

Reference data

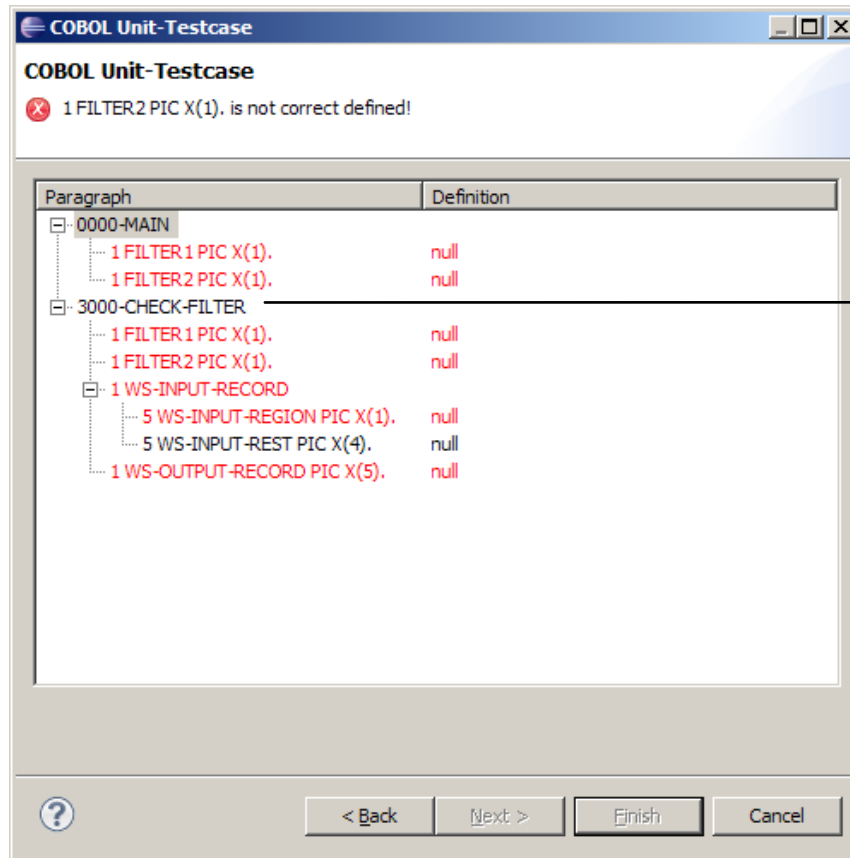
01 WS-INPUT-RECORD = 49999

01 WS-OUTPUT-RECORD = 49999

Software techniques on legacy systems.

Demonstration (New introspective test)

For introspective test cases the steps 3. and 4. looks like this:

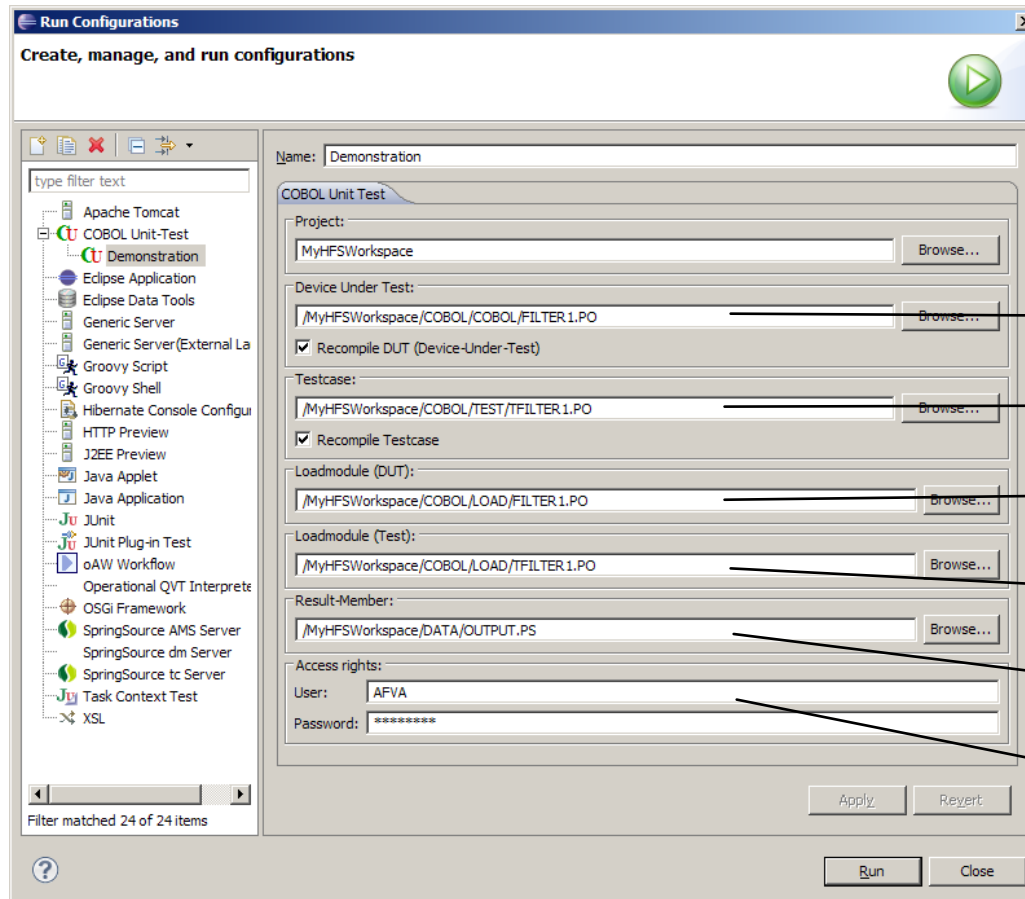


Subsection followed by the scoped variables.

Software techniques on legacy systems.

Demonstration (Test execution)

6. For both kinds of tests the execution configuration is the same:



Tested module + recompile ?

Test case + recompile ?

Load module (tested module)

Load module (test case)

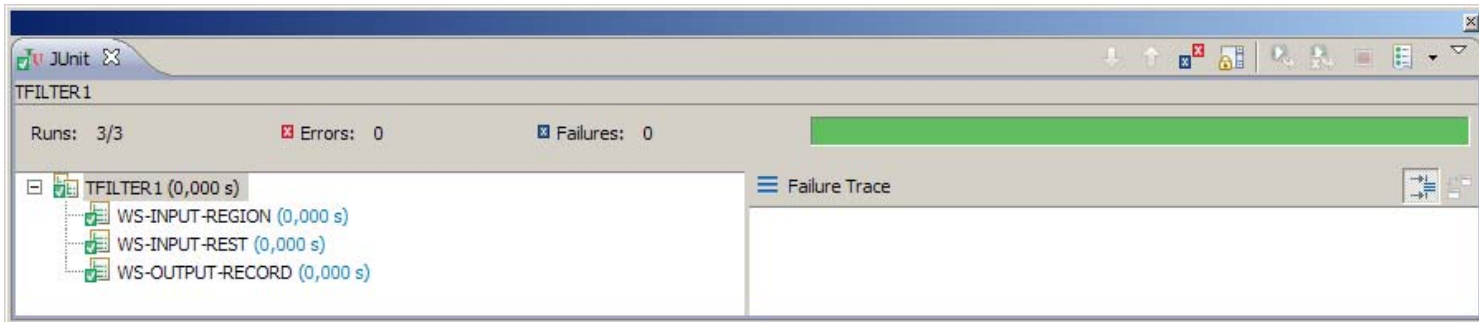
Result member

User authentication

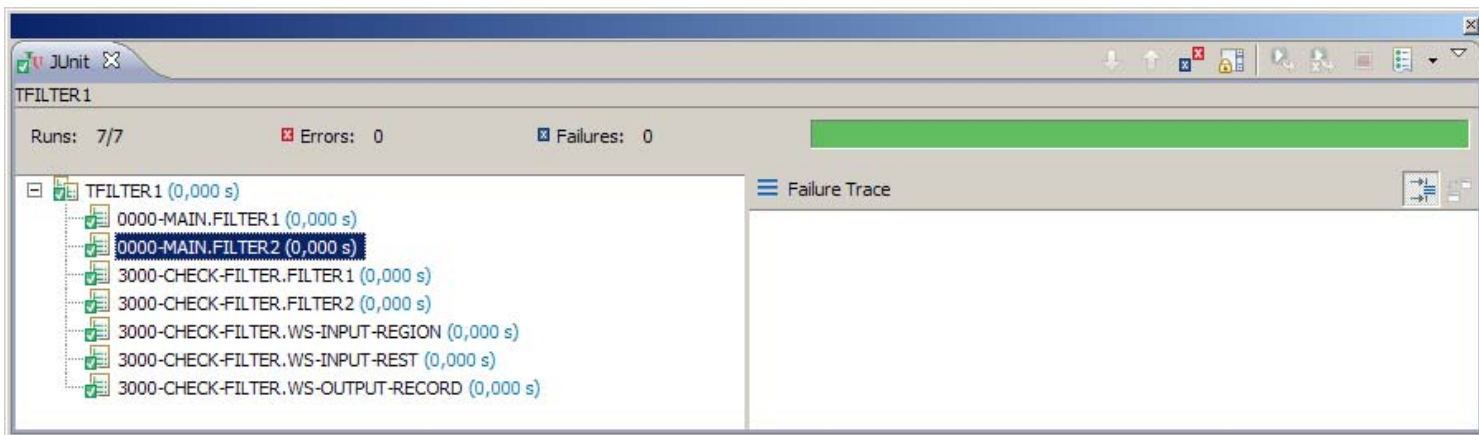
Software techniques on legacy systems.

Demonstration (Test execution)

7. The results of the test cases looks like this:



retrospective results

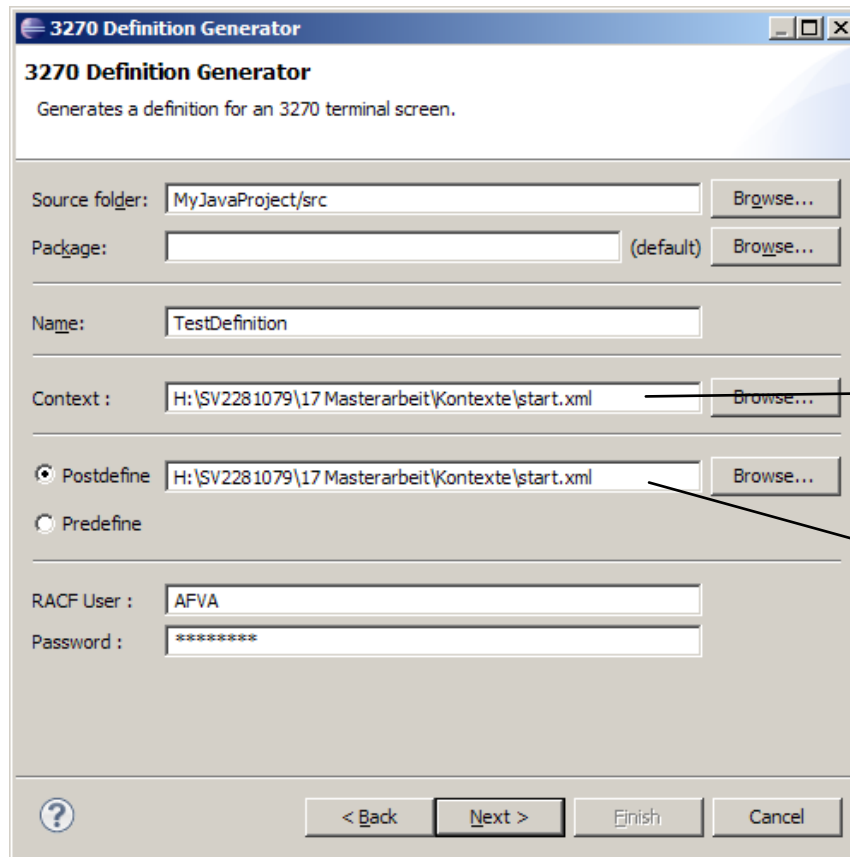


introspective results

Software techniques on legacy systems.

Demonstration (Definition test)

1. The wizard “New TUI test case (Definition)” define a new test case.



The screenshot shows the "3270 Definition Generator" wizard window. The title bar reads "3270 Definition Generator". Below the title bar, the text "3270 Definition Generator" and "Generates a definition for an 3270 terminal screen." are displayed. The window contains several input fields and buttons:

- Source folder:** MyJavaProject/src (with a "Browse..." button)
- Package:** (empty) (default) (with a "Browse..." button)
- Name:** TestDefinition
- Context :** H:\SV2281079\17 Masterarbeit\Kontexte\start.xml (with a "Browse..." button)
- Postdefine:** H:\SV2281079\17 Masterarbeit\Kontexte\start.xml (with a "Browse..." button)
- Predefine:**
- RACF User :** AFVA
- Password :** *****

At the bottom of the window, there are four buttons: "< Back", "Next >", "Finish", and "Cancel". A help icon (?) is located in the bottom left corner.

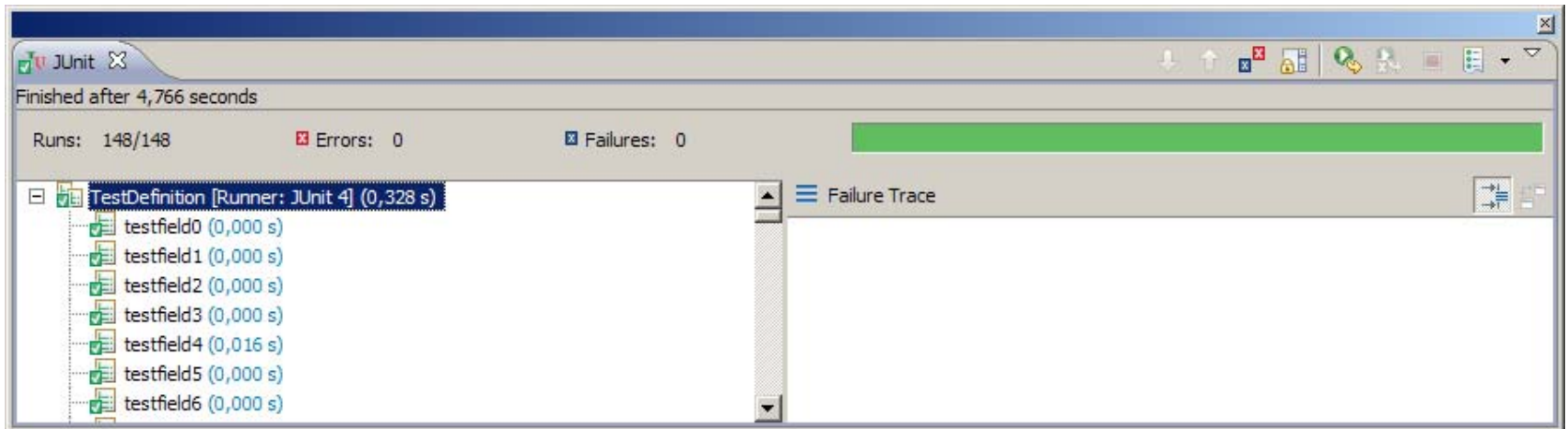
The context is a kind of path to the mask.

The same context is used for a post definition.

Software techniques on legacy systems.

Demonstration (Definition test)

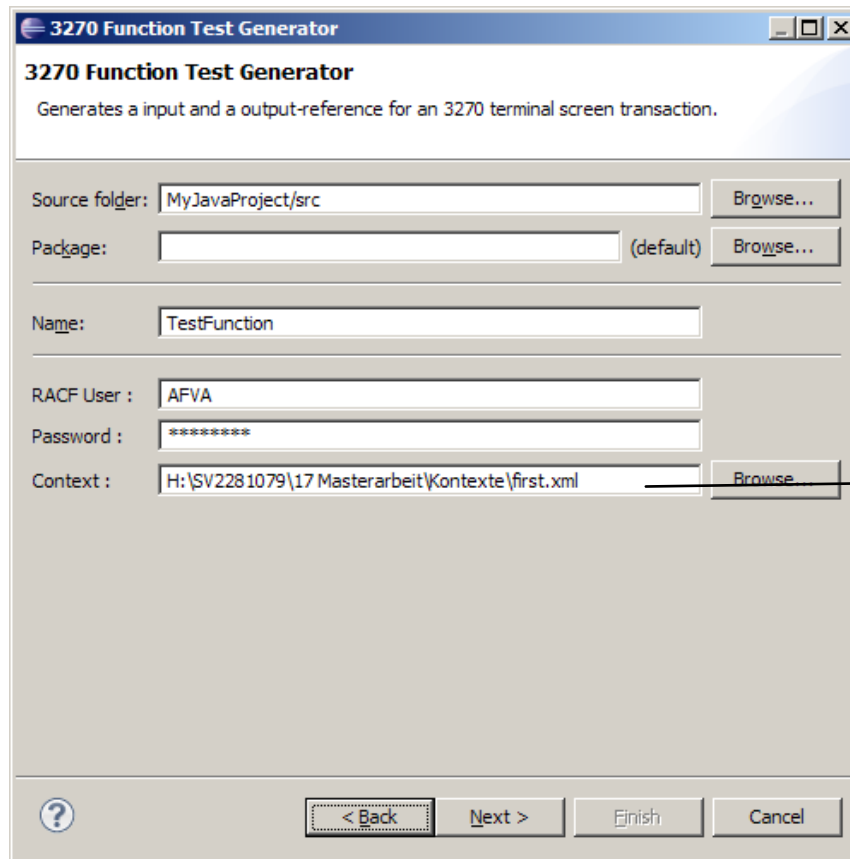
2. The generated JUnit test can execute directly.
In this case the metadata and the data is identical with the definition for every field. The test pass.



Software techniques on legacy systems.

Demonstration (Functional test)

1. The wizard “New TUI test case (Function)” define a new test case.



The screenshot shows a Windows-style dialog box titled "3270 Function Test Generator". The window has a blue header bar with the title and standard window controls. Below the header, the text "3270 Function Test Generator" is displayed, followed by a subtitle: "Generates a input and a output-reference for an 3270 terminal screen transaction." The main area contains several input fields and buttons:

- Source folder:** A text box containing "MyJavaProject/src" and a "Browse..." button to its right.
- Package:** A text box containing "(default)" and a "Browse..." button to its right.
- Name:** A text box containing "TestFunction".
- RACF User :** A text box containing "AFVA".
- Password :** A text box containing "*****".
- Context :** A text box containing "H:\SV2281079\17 Masterarbeit\Kontexte\first.xml" and a "Browse..." button to its right.

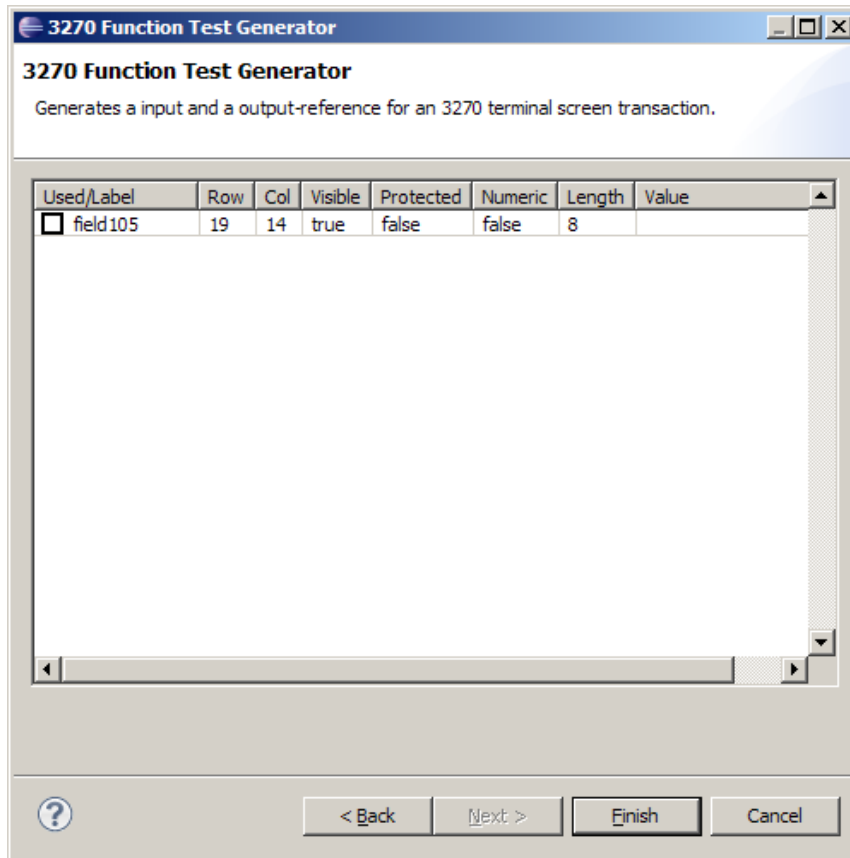
At the bottom of the window, there is a navigation bar with a question mark icon on the left, and four buttons: "< Back", "Next >", "Finish", and "Cancel".

The context is a kind of path to the mask.

Software techniques on legacy systems.

Demonstration (Functional test)

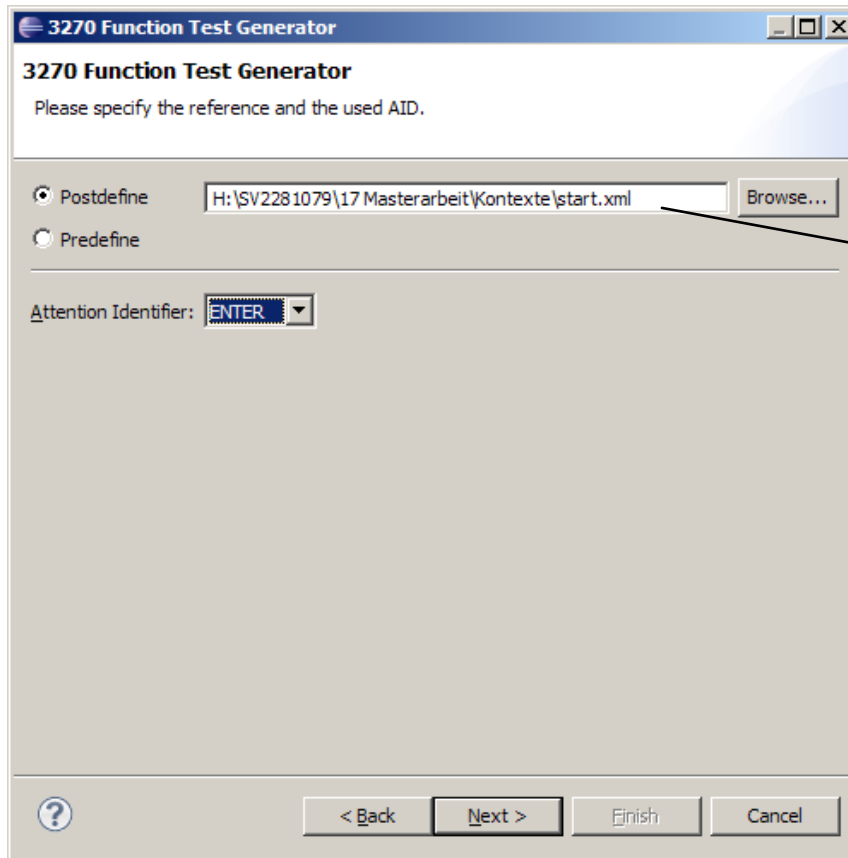
2. All input fields of the first mask are display. They can fill with input data.



Software techniques on legacy systems.

Demonstration (Functional test)

3. The definition for the response mask is require.



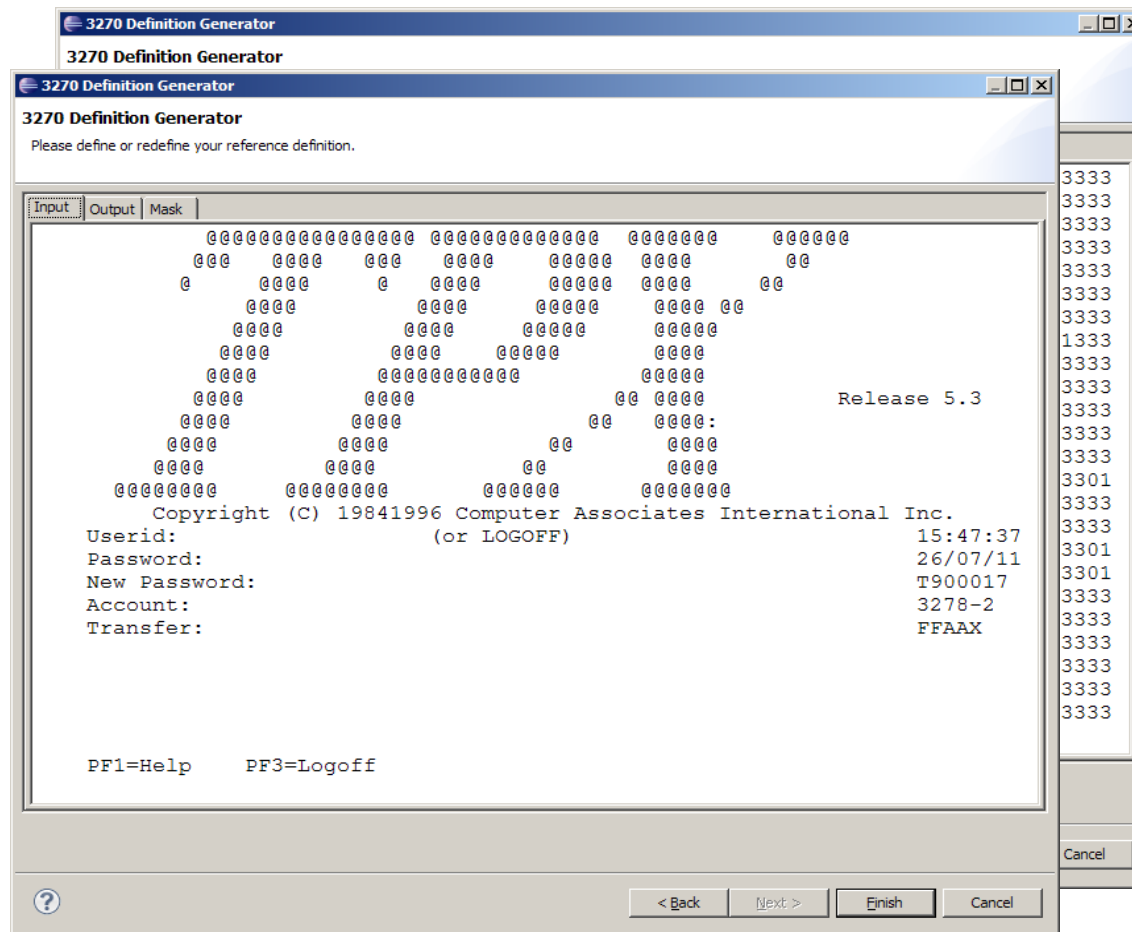
The screenshot shows a dialog box titled "3270 Function Test Generator". The main text reads "Please specify the reference and the used AID." There are two radio buttons: "Postdefine" (which is selected) and "Predefine". Next to the "Postdefine" radio button is a text input field containing the path "H:\SV2281079\17 Masterarbeit\Kontexte\start.xml" and a "Browse..." button. Below this, there is a label "Attention Identifier:" followed by a dropdown menu showing "ENTER". At the bottom of the dialog, there are four buttons: a help button (question mark), "< Back", "Next >", "Finish", and "Cancel".

Post definition.

Software techniques on legacy systems.

Demonstration (Functional test)

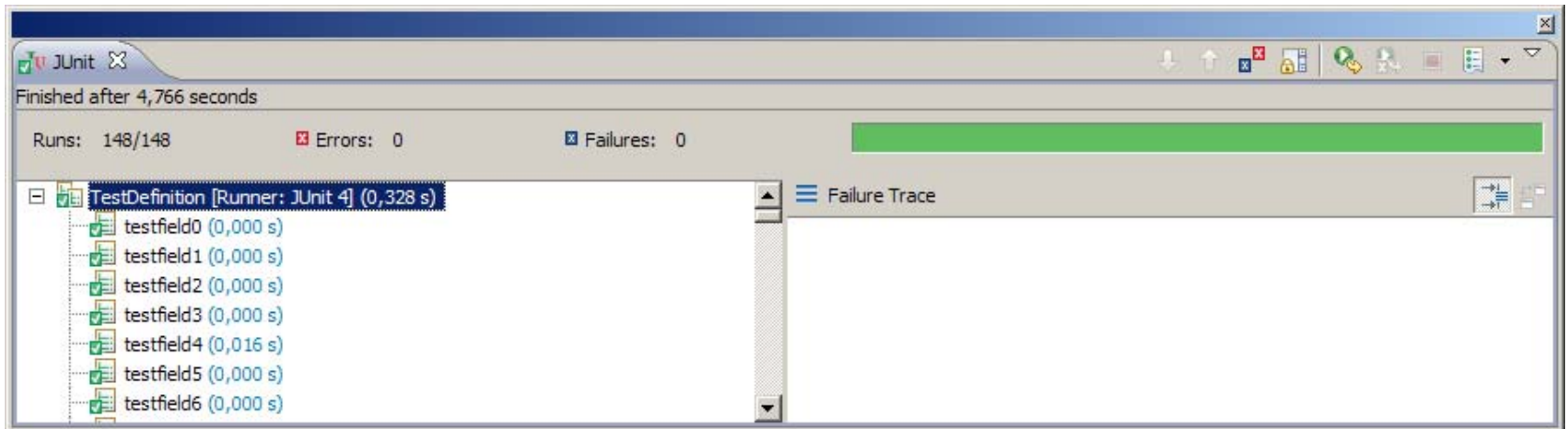
3. ... or pre-definition.



Software techniques on legacy systems.

Demonstration (Functional test)

4. The generated JUnit test can be executed directly.
In this case the metadata and the data is for every field identical with the definition. The test passes.



Software techniques on legacy systems.

Conclusion

- The developed tool can use to create easily test cases for COBOL programs.
- The test scopes are not restrict to the naturally program flow.
- The program code and the test code is separate (non-invasive testing)
- The system environment of z/OS is untouched.
- The mainframe is totally encapsulate. The test designer needs no green screen terminal.
- All generated test cases could be modified / customized if necessary.