



NatUnit

a unit test framework for
Natural

STEFAN MACKE
ALTE OLDENBURGER
Krankenversicherung AG

NatUnit – a unit test framework for Natural

© Stefan Macke
ALTE OLDENBURGER AG
Berlin, 08.05.2012

contact information:

- stefan@macke.it
- <http://blog.macke.it>
- @StefanMacke



NatUnit

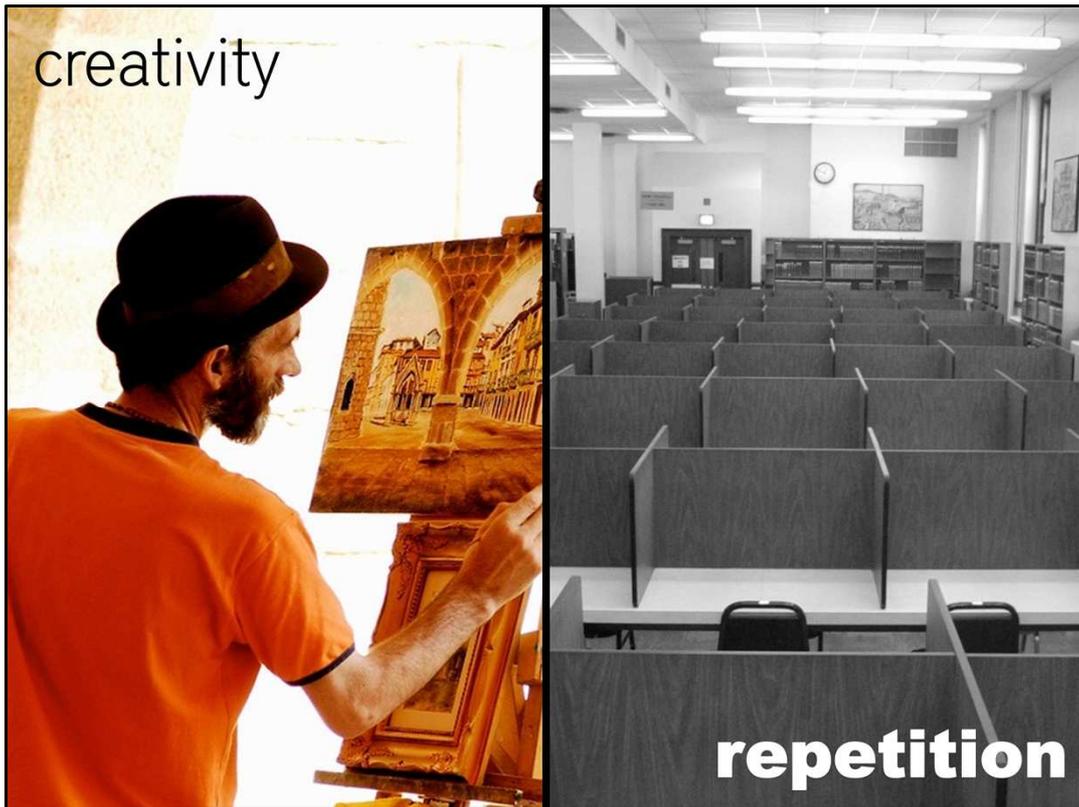
a unit test framework for
Natural

STEFAN MACKE
ALTE OLDENBURGER
Krankenversicherung AG

Most developers, when thinking of testing their applications, think of a very boring task.



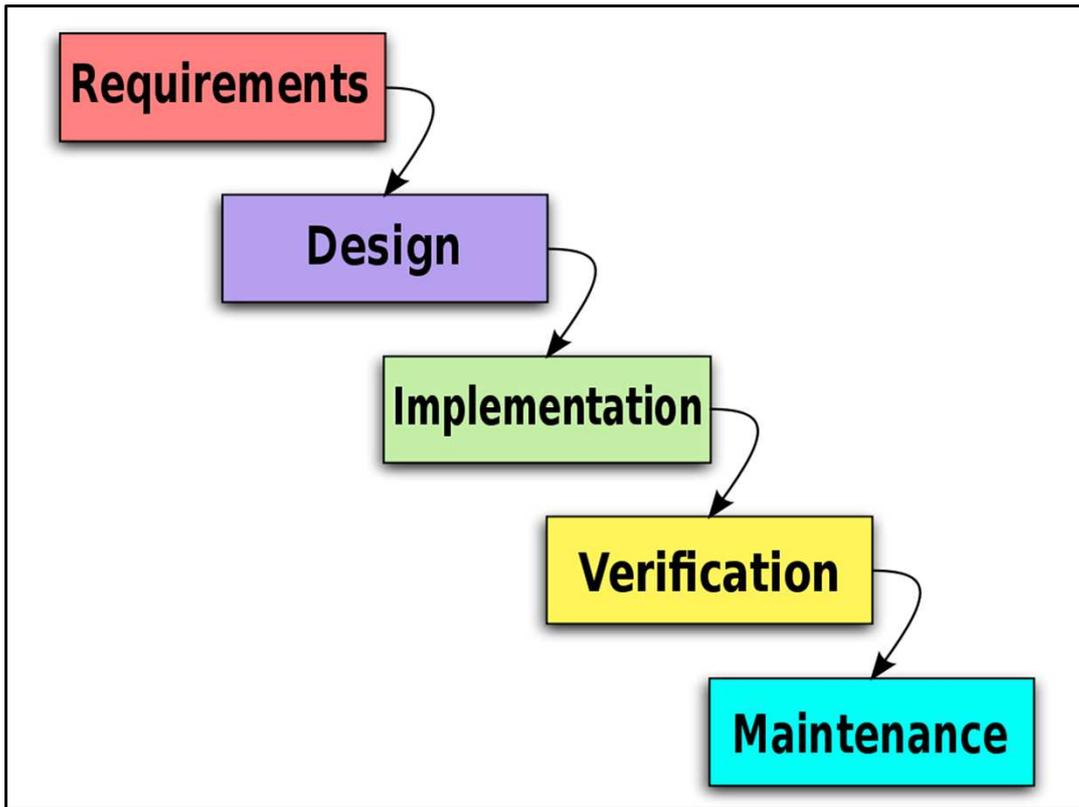
This also held true for myself and most of my colleagues: we didn't like testing either.



I always thought of programming as some kind of creative task compared to the repetitive testing process.



When creating a program I could create something new and be constructive. However, when testing I did quite the opposite: searching for bugs in the program, trying to make it crash or behave unexpectedly, finding even the smallest flaws in the programmer's work.



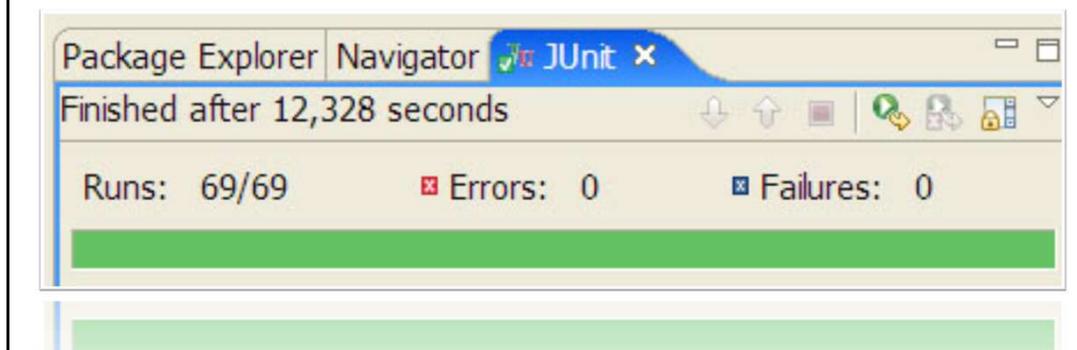
Additionally, testing often gets deferred to the latest possible time in a project. And whatever task is scheduled to be done last is prone to not getting done at all.



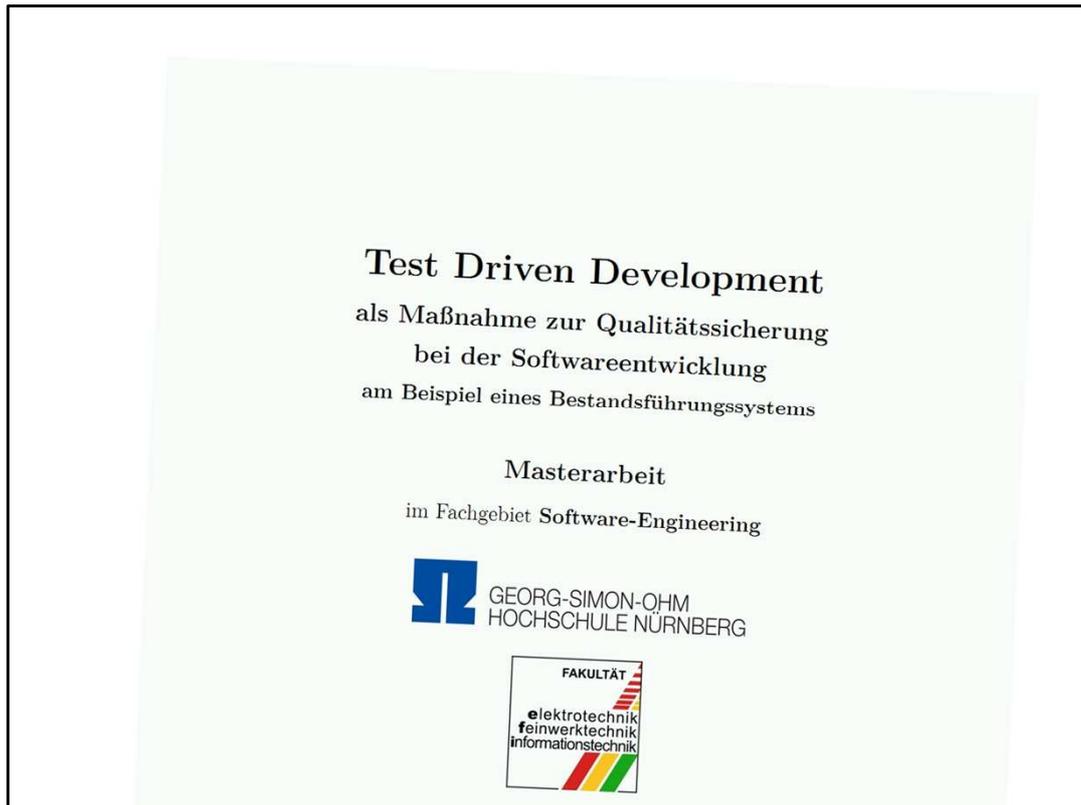
Sadly, this approach leads to what you may call a „banana product“: it (hopefully) ripens at the customer’s office (which means that the customer finds the bugs and complains about them to you).

So how do we make sure that our programs get tested thoroughly or even get tested at all?

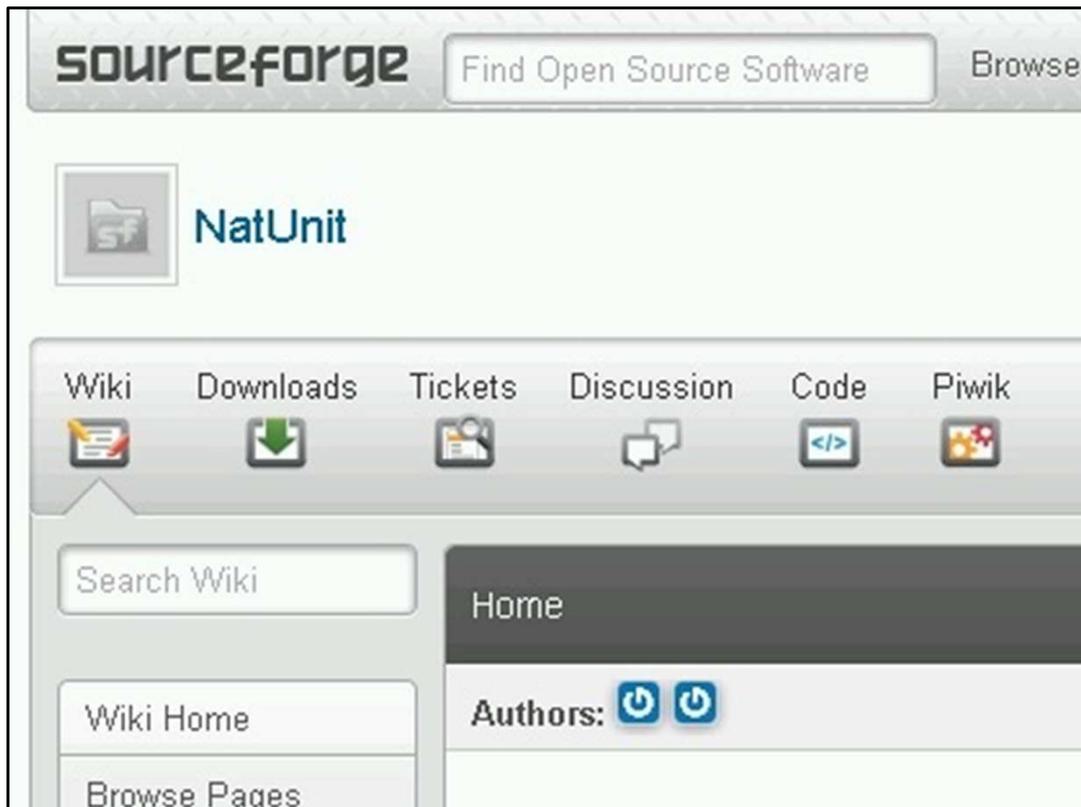
unit **tests**



The answer is automated tests for your application: unit tests. Recently, unit testing became a de facto standard in software development, especially in agile development processes. It all started with JUnit, a unit test framework for Java, developed by Kent Beck and Erich Gamma.



But there was no framework for Natural. So a little over two years ago, I developed NatUnit, a unit testing framework for Natural as part of my Master's thesis in software engineering.



Recently, we published the current version of the framework on SourceForge, a web platform for hosting Open Source software.

<http://sourceforge.net/p/natunit/>



NatUnit is licensed under LGPL, which means that you can download it for free, use it in your production code and develop it further if you like, as long as you mention us as the original authors of the framework.



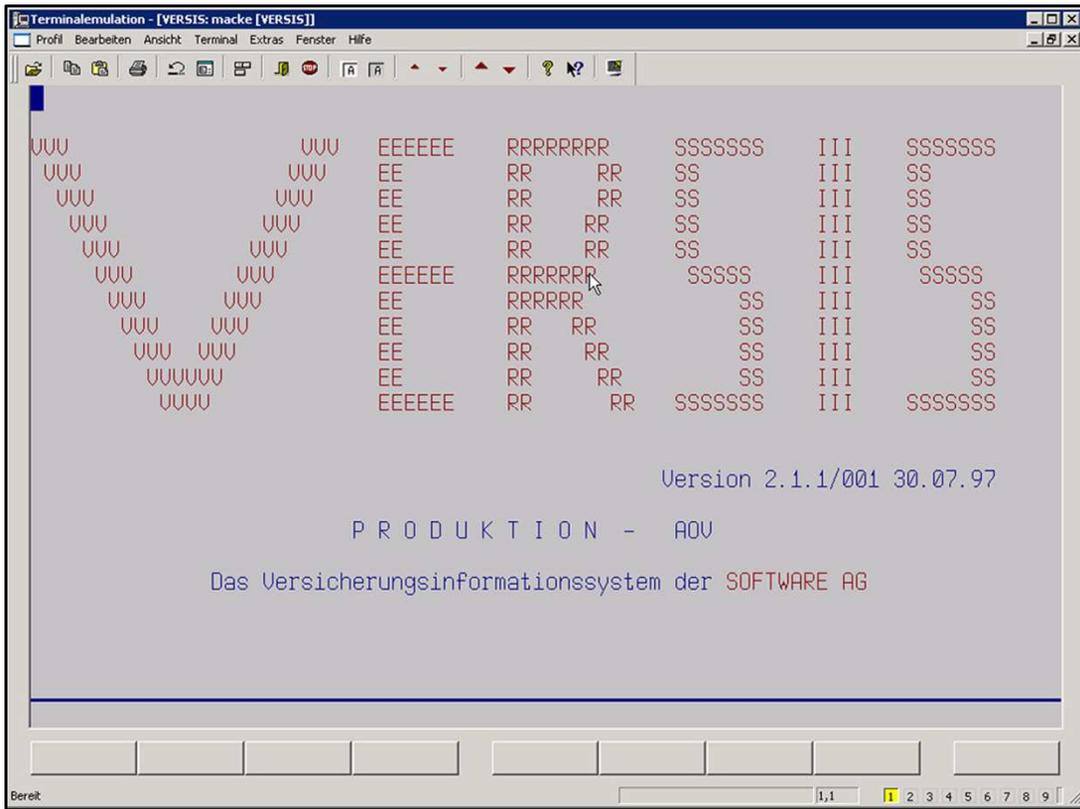
These are my contact information. Feel free to contact me.



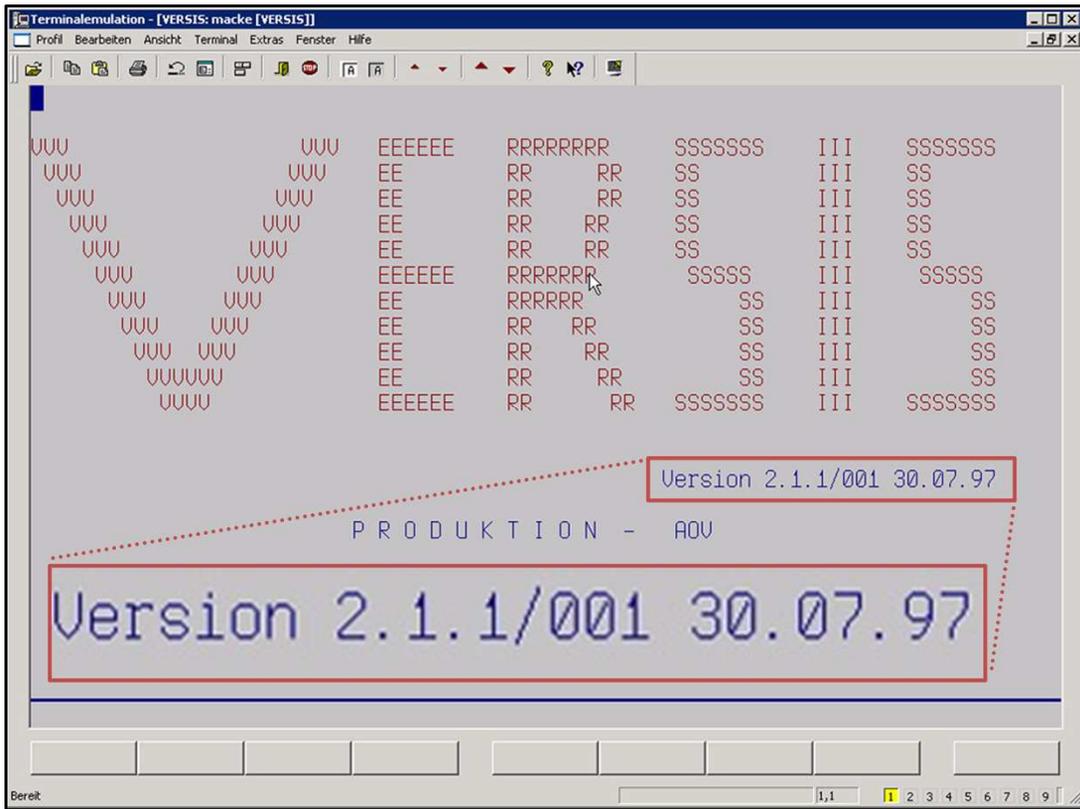
I work at ALTE OLDENBURGER AG (AO), a small German private health insurance company in northern Germany. We have almost 200 employees...



...about 20 of which are developers.



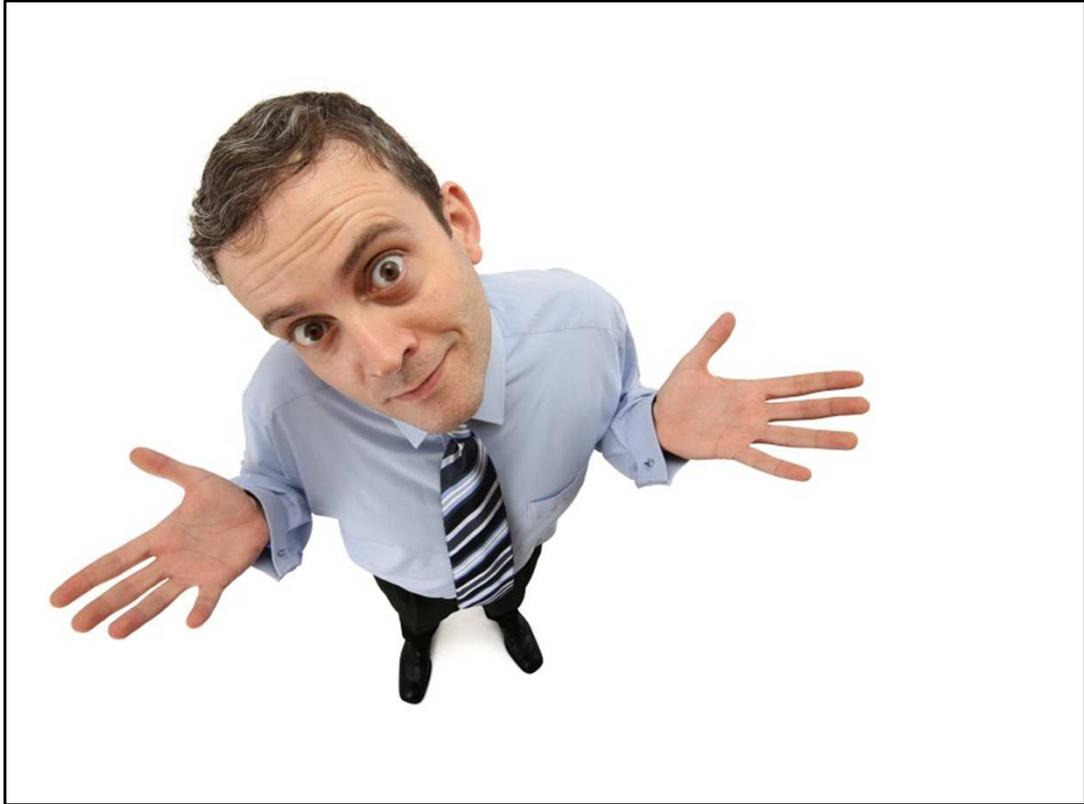
Our main system is VERGIS, which is short for „insurance information system“.



It was developed in the late 1990s in a cooperation between SAG and AO using Natural/Adabas.



in 2008 we celebrated VERSIS' 10th birthday. And all of this was possible without writing unit tests! Senior developers at our company didn't write any tests at all.



So why bother about unit testing, when everything works quite well?



I think the benefits of unit tests speak for themselves:

- You can safely refactor your code mercilessly.
- No more boring repetitive slow and costly testing.
- Tests force you to write clean code.
- Developers get instant feedback.

The **number one goal** is to write a framework within which we have some **glimmer of hope** that developers will actually write tests.



Erich **Gamma**

But how do I get a Senior Developer to actually write some tests? Answer: Provide him with an easy to use framework!



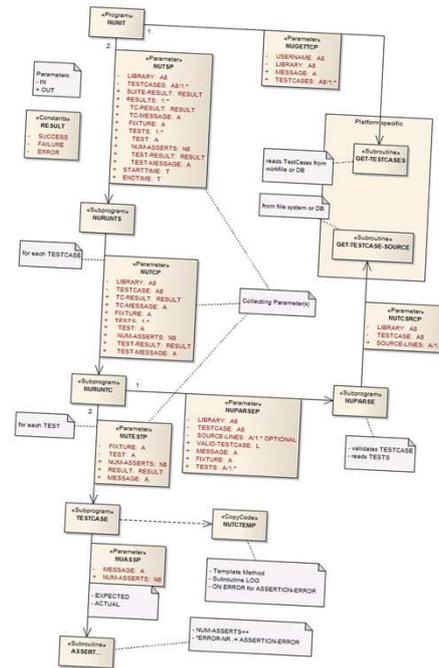
The most important aspect in my opinion is: the framework has to use the same language as the one you write your code in.



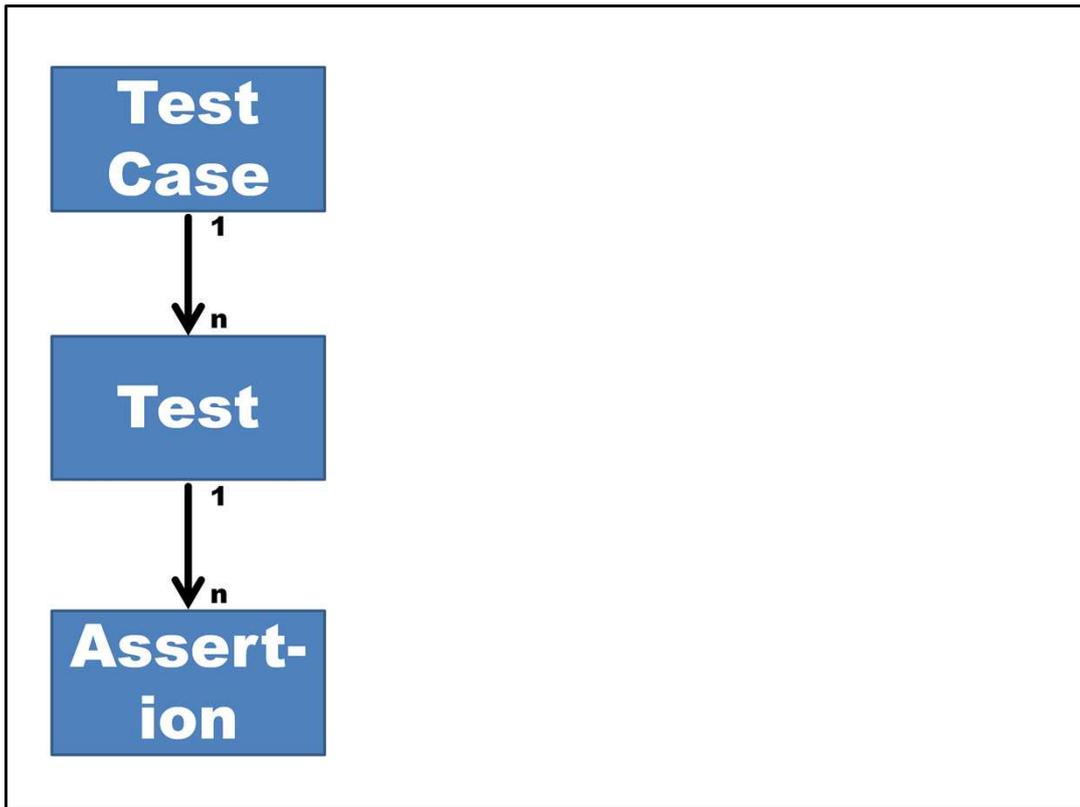
For Natural this of course means, that tests and production code should both be written in Natural.

NatUnit

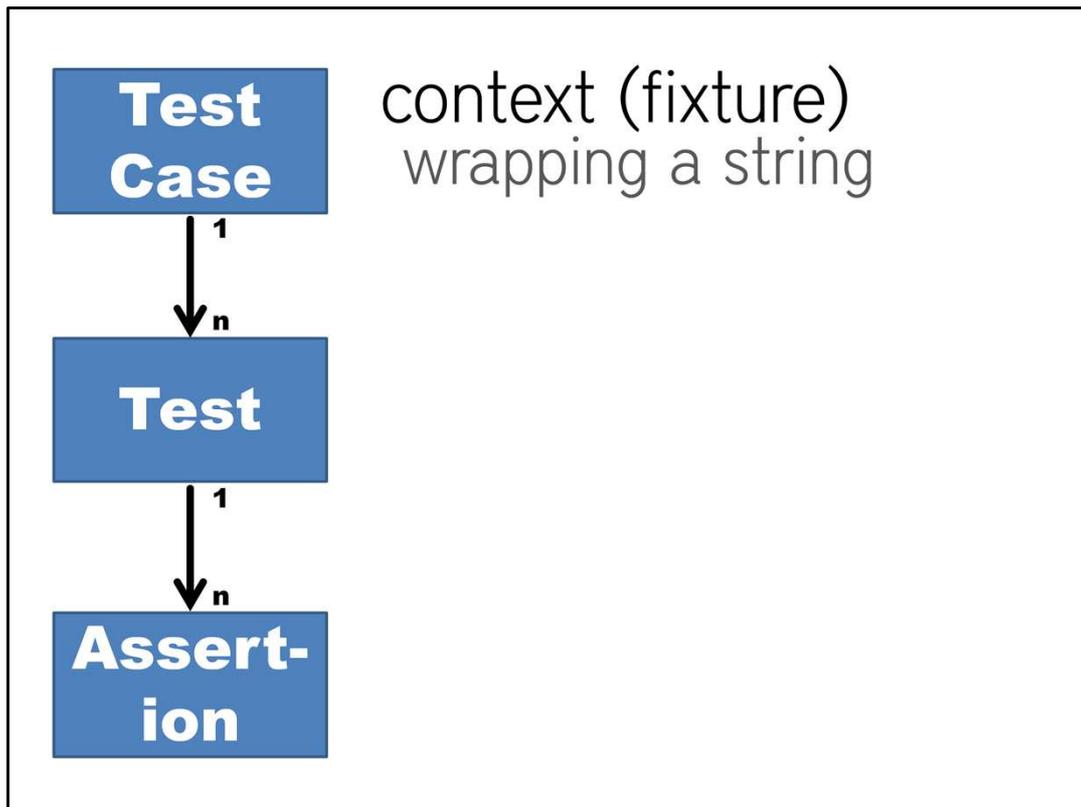
the framework



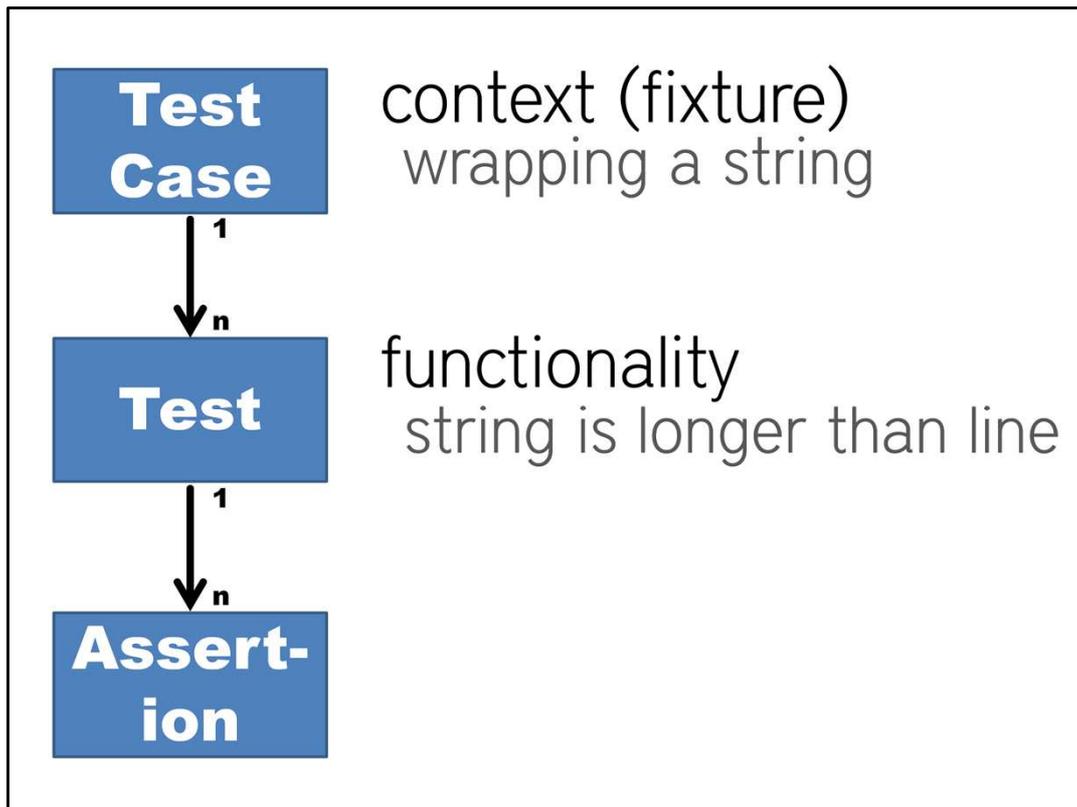
Let's take a closer look at NatUnit.



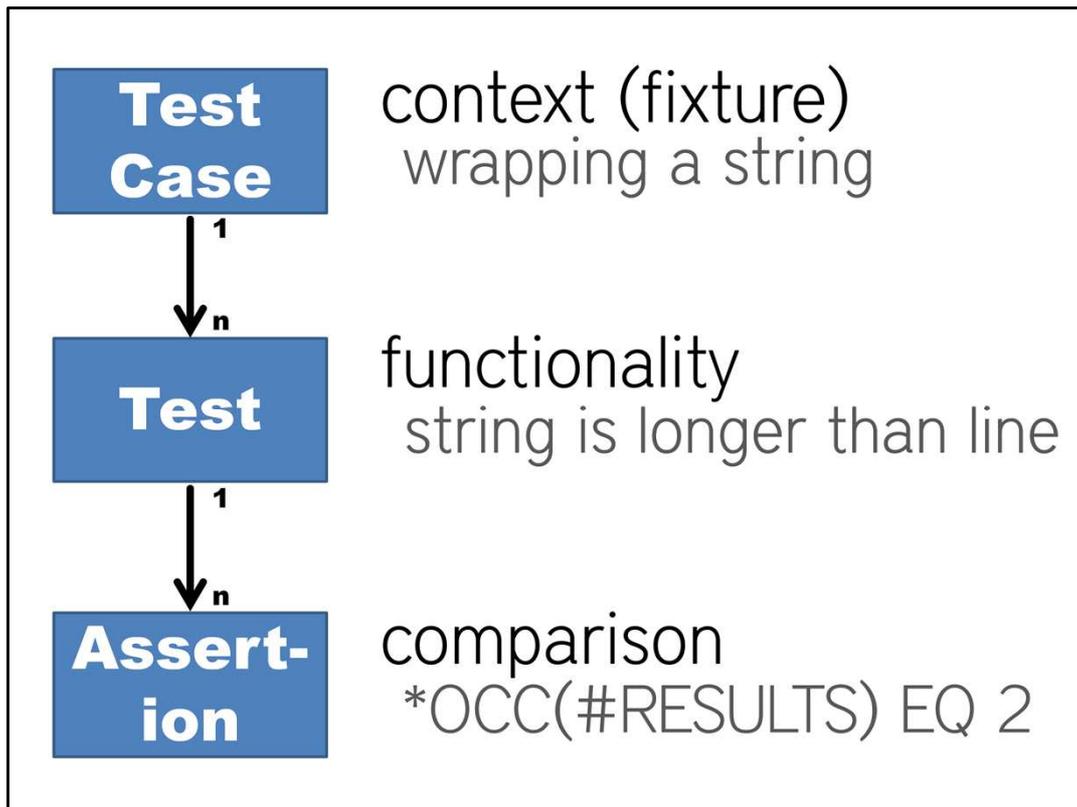
The framework was inspired by JUnit and therefore uses the same organisational structure for its tests.



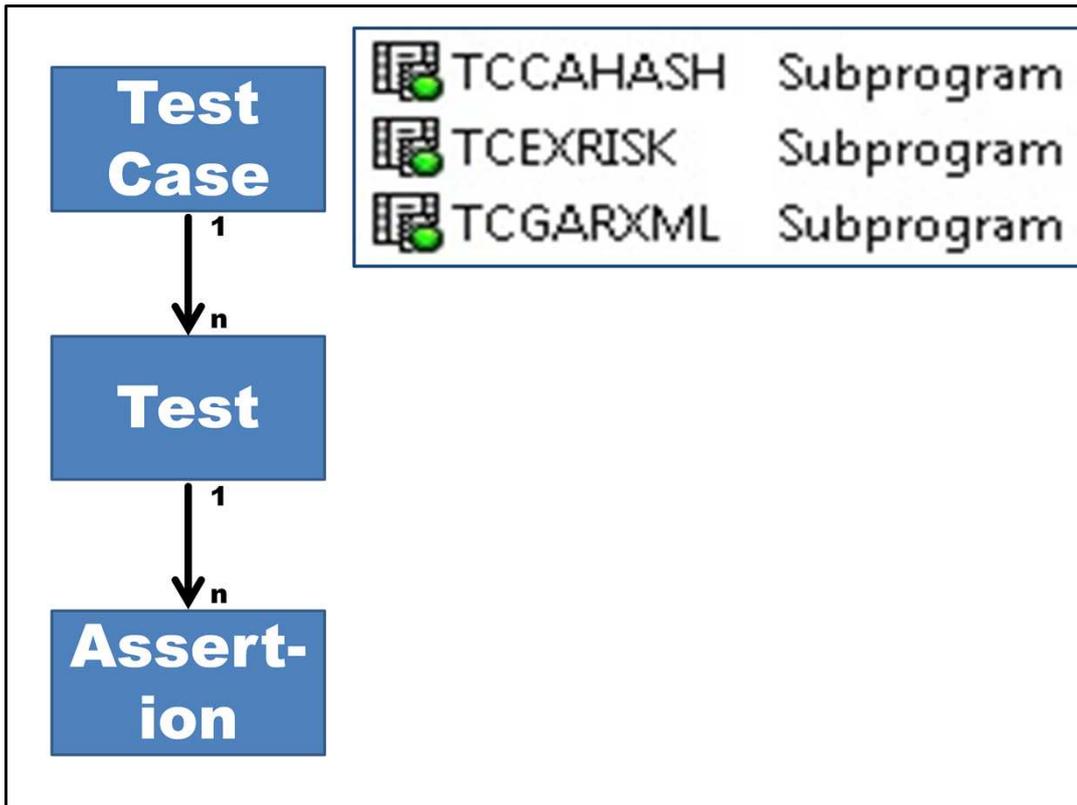
A TestCase describes the context that we are testing, e.g. the wrapping of a string at a given line length.



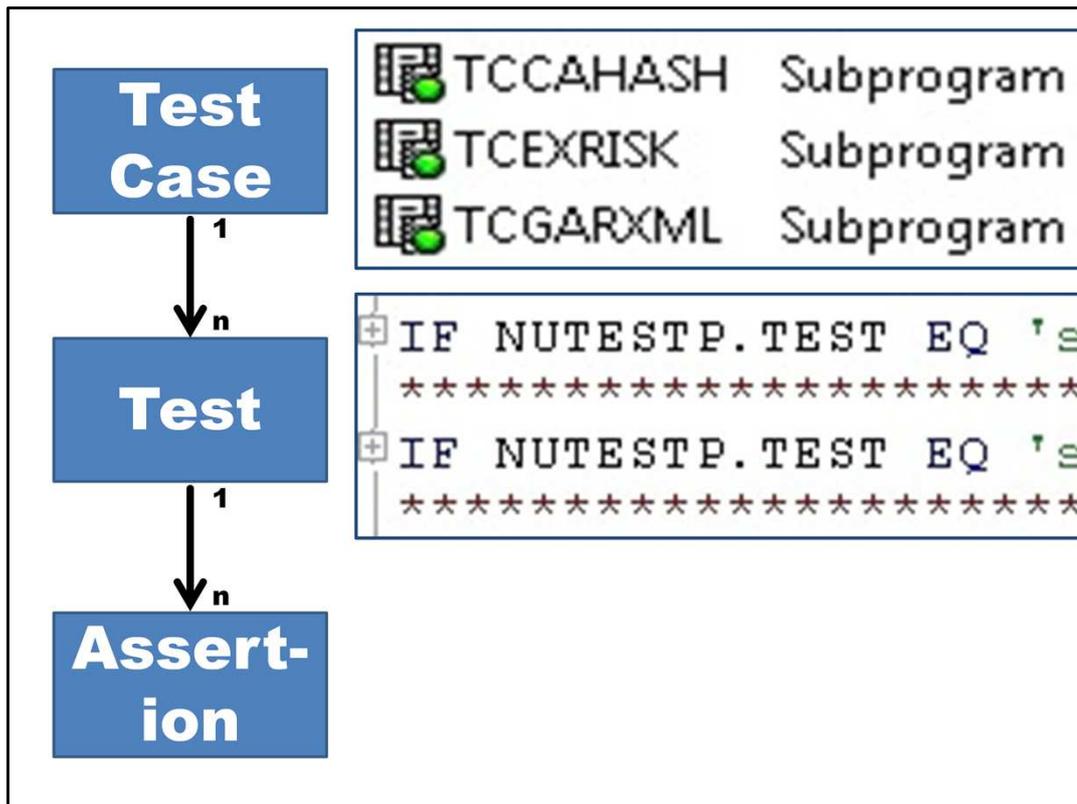
A Test is a single functionality in the context, e.g. the string to wrap is longer than the line length.



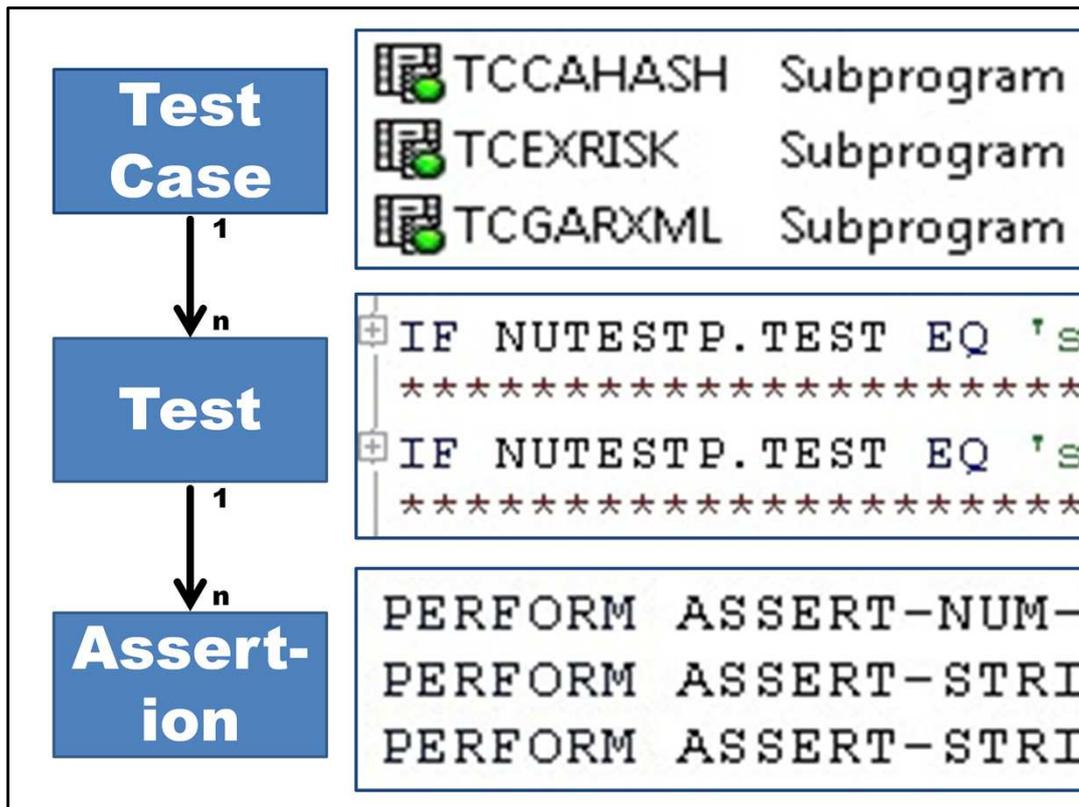
An Assertion is a simple comparison that is needed to verify the test, e.g. the resulting array containing the wrapped string should have a length of 2.



In NatUnit TestCases are written as subprograms. The convention suggests using TC as a prefix for the modules' names.



A Test in NatUnit is written as a special IF statement. This may seem a bit weird at first, but the reason for this is that you can't call (inline) subroutines dynamically (i.e. by providing its name in a variable/parameter) in Natural.



An Assertion is simply a call to an external subroutine that does the comparison. There are already quite a few useful Assertions included with NatUnit but you can also easily write your own.



```
NUTESTP.FIXTURE :=
  'Wrap a string at a given line length'
*
*****
IF NUTESTP.TEST EQ
  'string longer than line length should be wrapped'
*****
  TEXT := 'Test 123'
  LINE-LENGTH := 6
*
  PERFORM WRAP-STRING TEXT LINE-LENGTH TEXT-ARRAY(*)
*
  #NR-OF-LINES := *OCC(TEXT-ARRAY)
  PERFORM ASSERT-NUM-EQUALS 2 #NR-OF-LINES
  PERFORM ASSERT-STRING-SAME 'Test 1' TEXT-ARRAY(1)
  PERFORM ASSERT-STRING-SAME '23' TEXT-ARRAY(2)
END-IF
```

This is a sample test for the wrapping of a string.

```

NUTESTP.FIXTURE :=
  'Wrap a string at a given line length'
NUTESTP.FIXTURE :=
  'Wrap a string at a given line length'
*
*****
IF NUTESTP.TEST EQ
  'string longer than line length should be wrapped'
*****
  TEXT := 'Test 123'
  LINE-LENGTH := 6
*
  PERFORM WRAP-STRING TEXT LINE-LENGTH TEXT-ARRAY(*)
*
  #NR-OF-LINES := *OCC(TEXT-ARRAY)
  PERFORM ASSERT-NUM-EQUALS 2 #NR-OF-LINES
  PERFORM ASSERT-STRING-SAME 'Test 1' TEXT-ARRAY(1)
  PERFORM ASSERT-STRING-SAME '23' TEXT-ARRAY(2)
END-IF

```

It starts with the description of the TestCase, the fixture. This is free text so it is easily readable and understandable by developers.

```

IF NUTESTP.TEST EQ
  'string longer than line leng
NUTESTP.FIXTURE :=
  'Wrap a string at a given line length'
*
*****
IF NUTESTP.TEST EQ
  'string longer than line length should be wrapped'
*****
  TEXT := 'Test 123'
  LINE-LENGTH := 6
*
  PERFORM WRAP-STRING TEXT LINE-LENGTH TEXT-ARRAY(*)
*
  #NR-OF-LINES := *OCC(TEXT-ARRAY)
  PERFORM ASSERT-NUM-EQUALS 2 #NR-OF-LINES
  PERFORM ASSERT-STRING-SAME 'Test 1' TEXT-ARRAY(1)
  PERFORM ASSERT-STRING-SAME '23' TEXT-ARRAY(2)
END-IF

```

The same holds true for the Tests: they are also defined as free text. A Test should describe a single bit of functionality that gets tested including the expected outcome.

```

TEXT := 'Test 123'
LINE-LENGTH := 6
NUTESTP.FIXTURE :=
  'Wrap a string at a given line length'
*
*****
IF NUTESTP.TEST EQ
  'string longer than line length should be wrapped'
  *****
TEXT := 'Test 123'
LINE-LENGTH := 6
*
  PERFORM WRAP-STRING TEXT LINE-LENGTH TEXT-ARRAY(*)
*
  #NR-OF-LINES := *OCC(TEXT-ARRAY)
  PERFORM ASSERT-NUM-EQUALS 2 #NR-OF-LINES
  PERFORM ASSERT-STRING-SAME 'Test 1' TEXT-ARRAY(1)
  PERFORM ASSERT-STRING-SAME '23' TEXT-ARRAY(2)
END-IF

```

The implementation of the Test follows the AAA method:

1) **arrange** the prerequisites needed to run the test (in this example: provide a text that is longer than the line length)...

PERFORM WRAP-STRING TEXT LINE-I

```
NUTESTP.FIXTURE :=
  'Wrap a string at a given line length'
*
*****
IF NUTESTP.TEST EQ
  'string longer than line length should be wrapped'
*****
  TEXT := 'Test 123'
  LINE-LENGTH := 6
*
PERFORM WRAP-STRING TEXT LINE-LENGTH TEXT-ARRAY(*)
*
#NR-OF-LINES := *OCC(TEXT-ARRAY)
PERFORM ASSERT-NUM-EQUALS 2 #NR-OF-LINES
PERFORM ASSERT-STRING-SAME 'Test 1' TEXT-ARRAY(1)
PERFORM ASSERT-STRING-SAME '23' TEXT-ARRAY(2)
END-IF
```

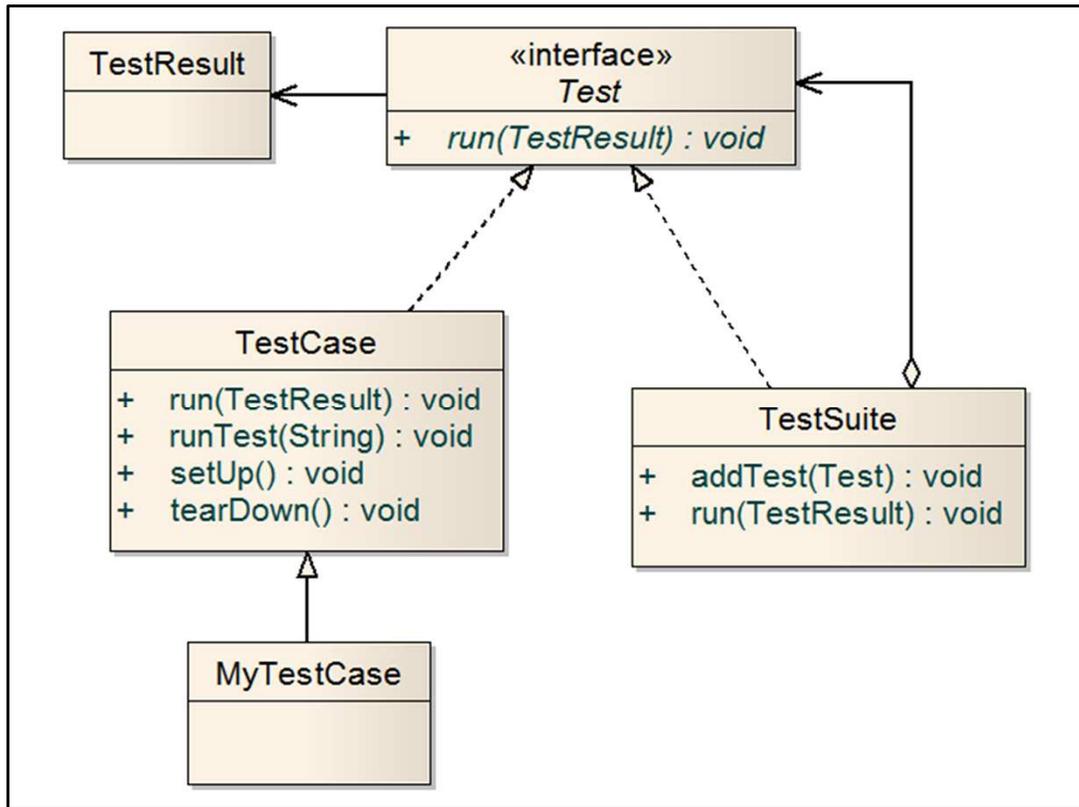
2) **act**: call the module under test (in the example the subroutine WRAP-STRING)...

```

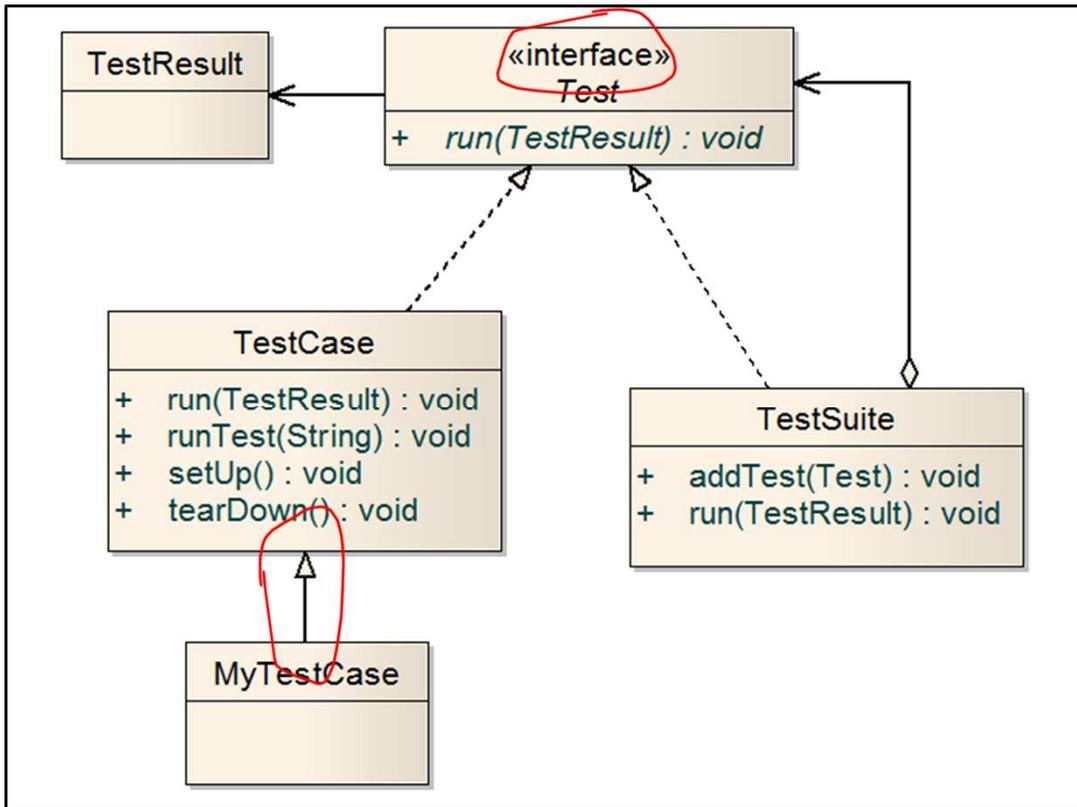
ASSERT-NUM-EQUALS 2 #NR-OF-LINE
ASSERT-STRING-SAME 'Test 1' TEXT
NUTESTP.FIXTURE :=
  'Wrap a string at a given line length'
*
*****
IF NUTESTP.TEST EQ
  'string longer than line length should be wrapped'
*****
  TEXT := 'Test 123'
  LINE-LENGTH := 6
*
  PERFORM WRAP-STRING TEXT LINE-LENGTH TEXT-ARRAY(*)
*
  #NR-OF-LINES := *OCC(TEXT-ARRAY)
  PERFORM ASSERT-NUM-EQUALS 2 #NR-OF-LINES
  PERFORM ASSERT-STRING-SAME 'Test 1' TEXT-ARRAY(1)
  PERFORM ASSERT-STRING-SAME '23' TEXT-ARRAY(2)
END-IF

```

3) **assert**: compare the actual results to the expected results (in the example: the string should be split into two parts according to the line length).



Now how does the framework run these TestCases?



JUnit relies heavily on inheritance and the use of interfaces to make TestCases runnable, which is not possible in Natural.

```

DEFINE DATA
PARAMETER USING NUTESTP
LOCAL USING NUCONST
LOCAL USING NUASSP
END-DEFINE
*
NUTESTP.FIXTURE := 'Example TestCase 1'
*
INCLUDE NUTCTEMP ...
INCLUDE NUTCSTUB ...
*
DEFINE SUBROUTINE TEST
*
IF NUTESTP.TEST EQ ...
*
END-SUBROUTINE

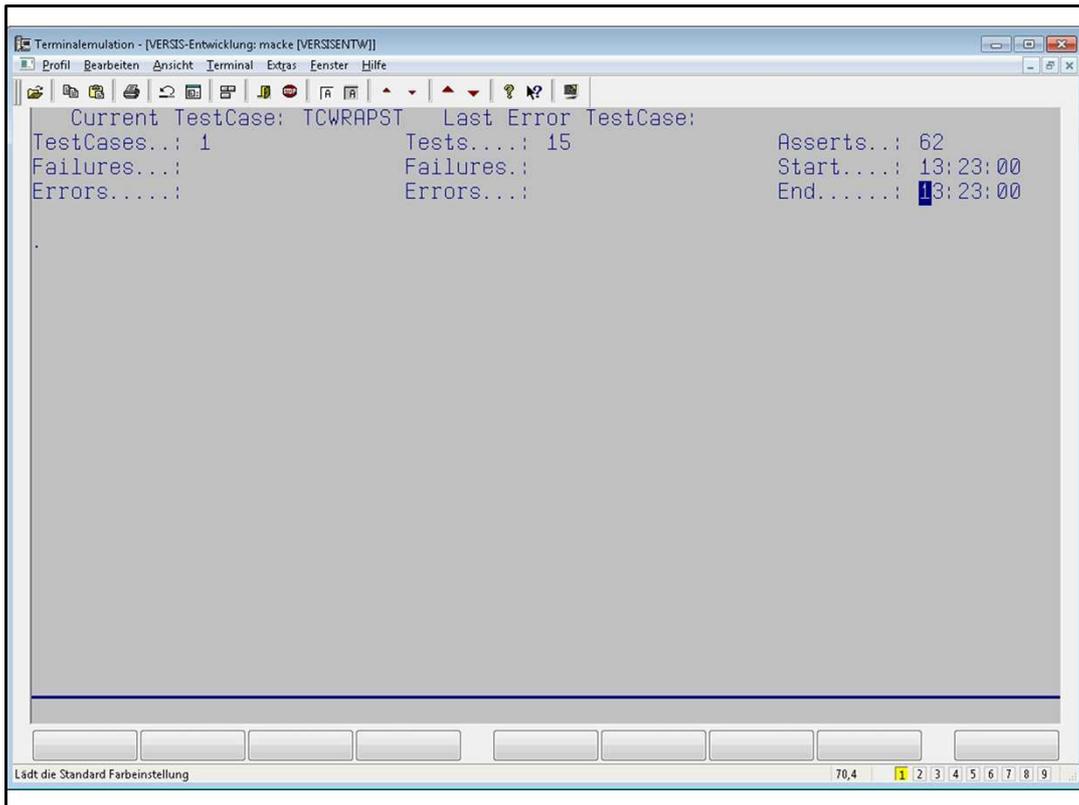
```

Therefore, TestCases in NatUnit have to contain some additional code (PDAs, LDAs, INCLUDEs) to be recognized by the framework.

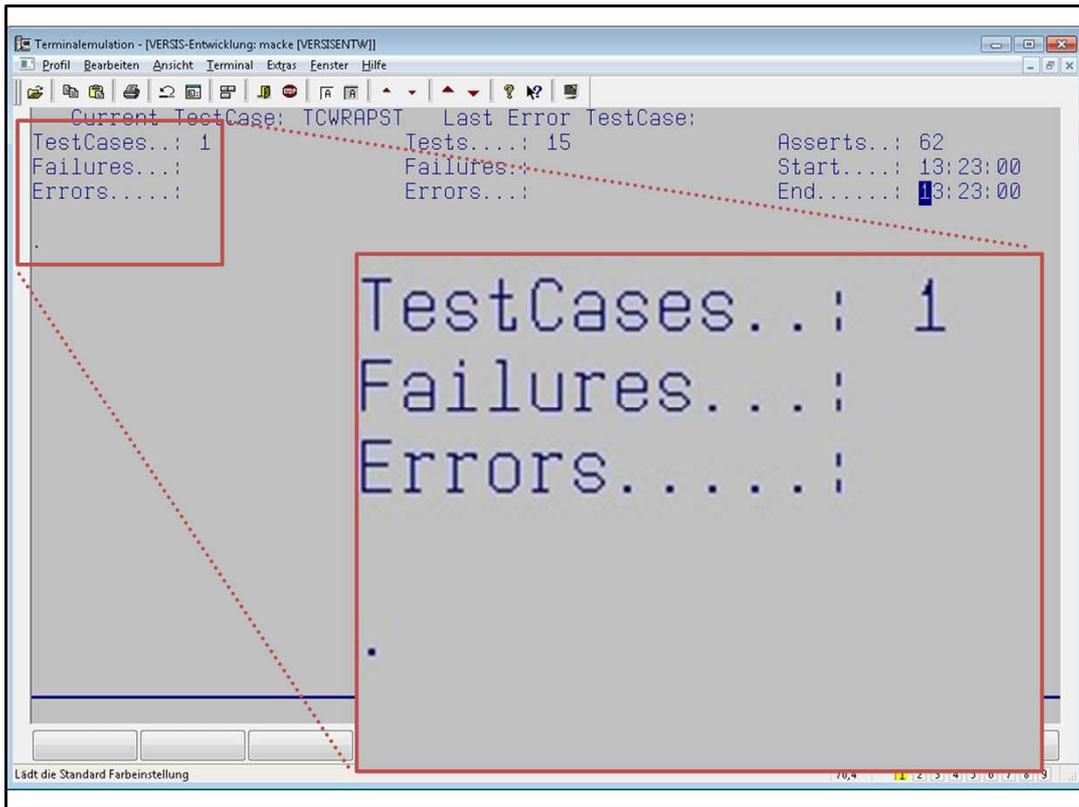
```
Terminalemulation - [VERSIS-Entwicklung: macke [VERSISENTW]]
Profil Bearbeiten Ansicht Terminal Extras Fenster Hilfe

**** U E R S
Current Library.....: UTILITY
Library.....: U-SYSTEM
TestCase.....: TCWRAPST
Use L4N.....:
Use L4N-Level (1-6)..: 1
Close L4N.....:
XML-Resultfile.....:
```

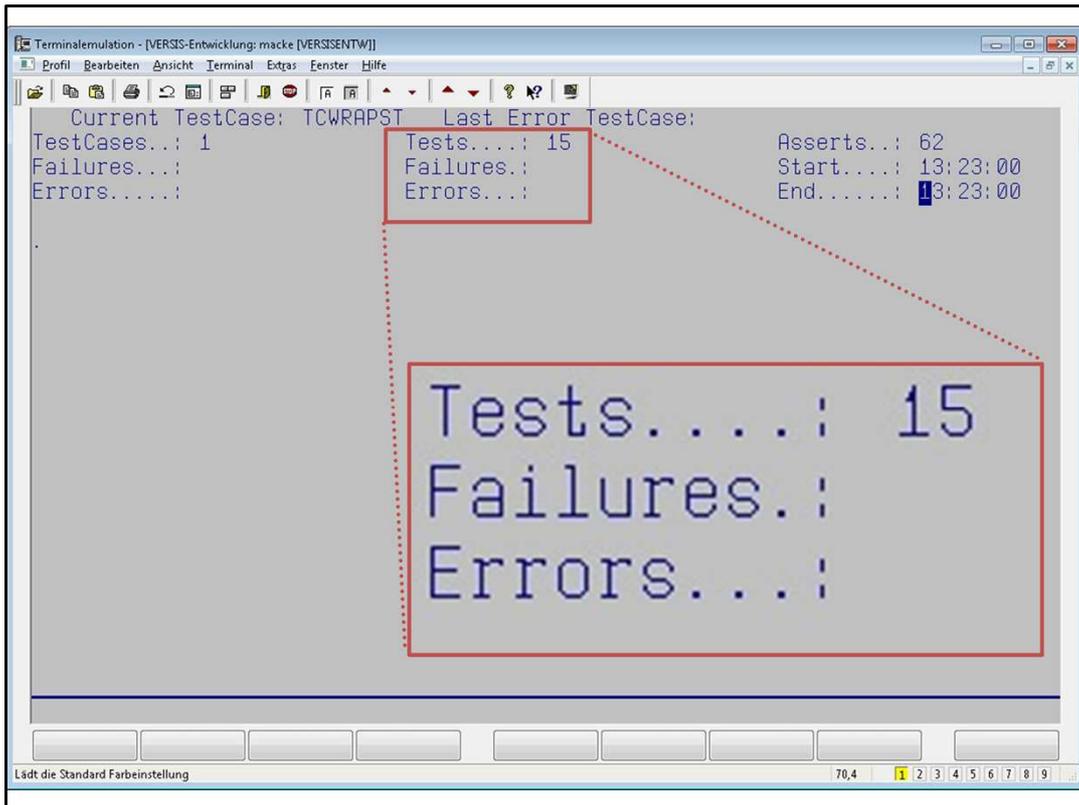
Now the TestCase can be executed by NatUnit, e.g. with NUSINGLE which runs a single TestCase.

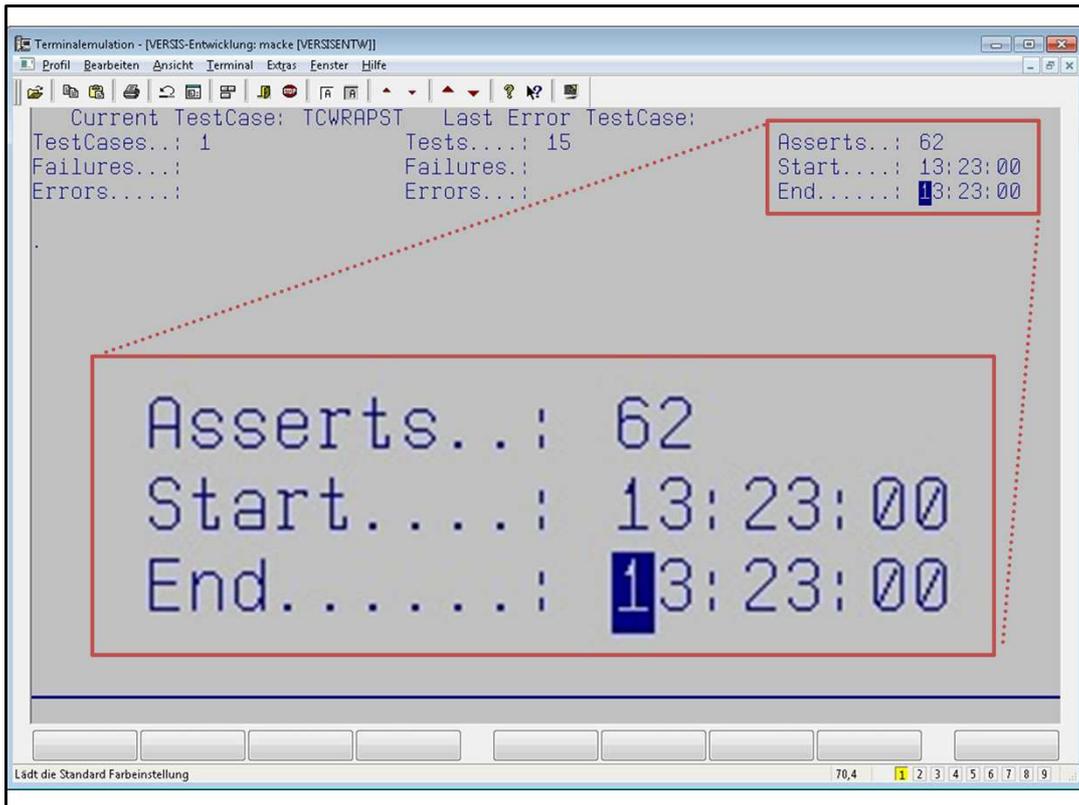


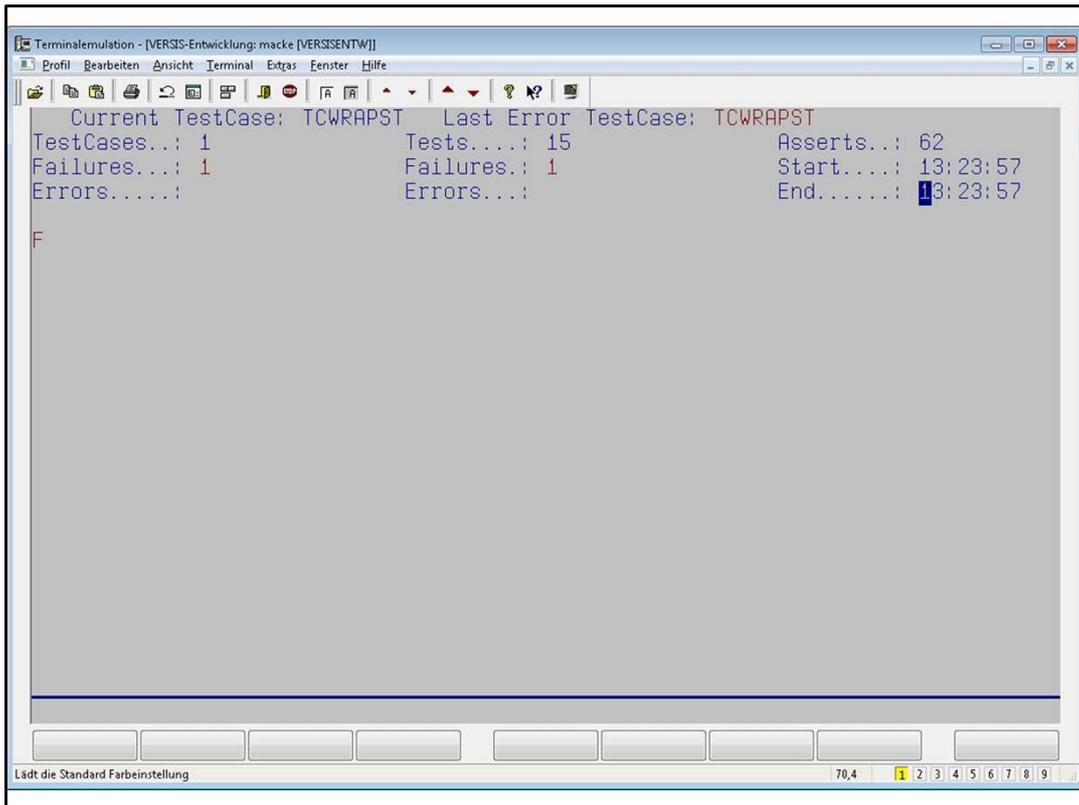
These are the results of the test run.



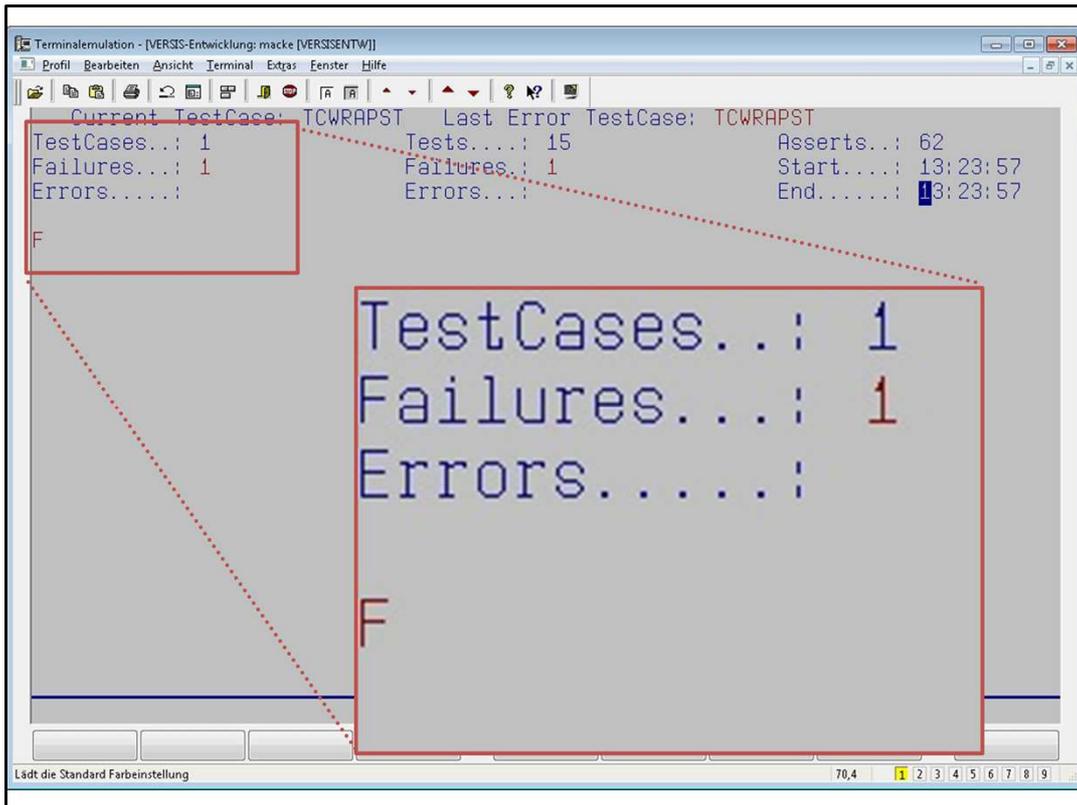
Each dot at the bottom of the output represents a successful TestCase.

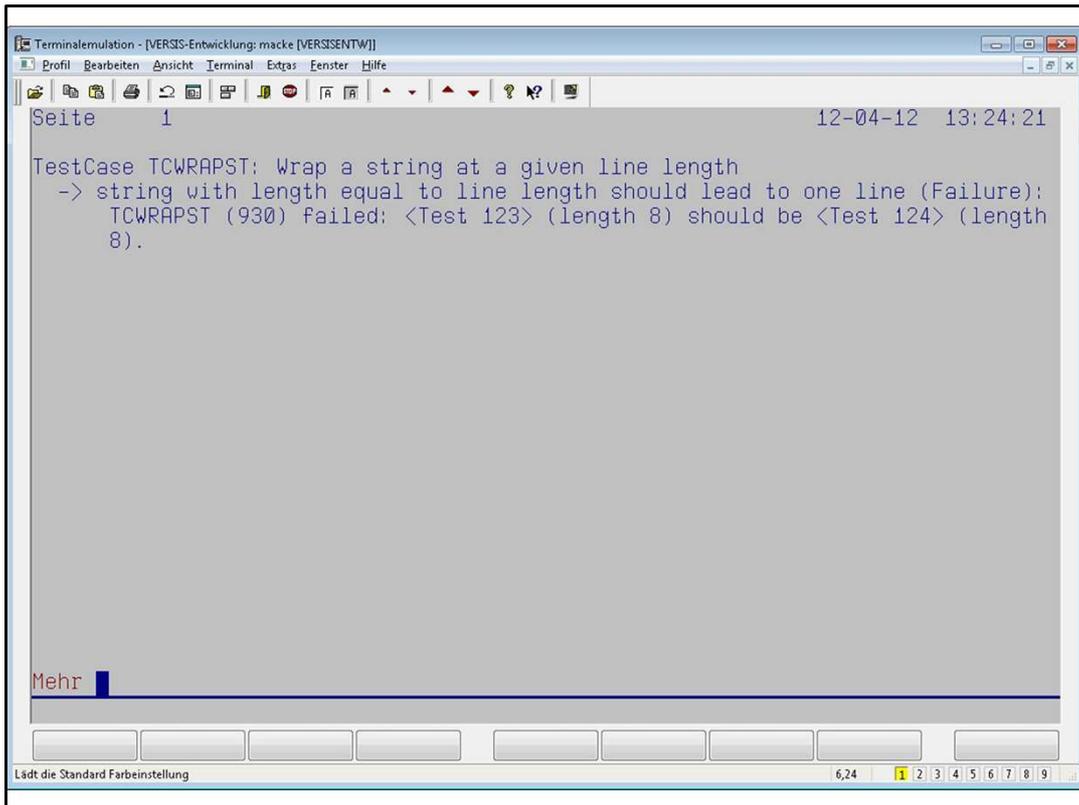




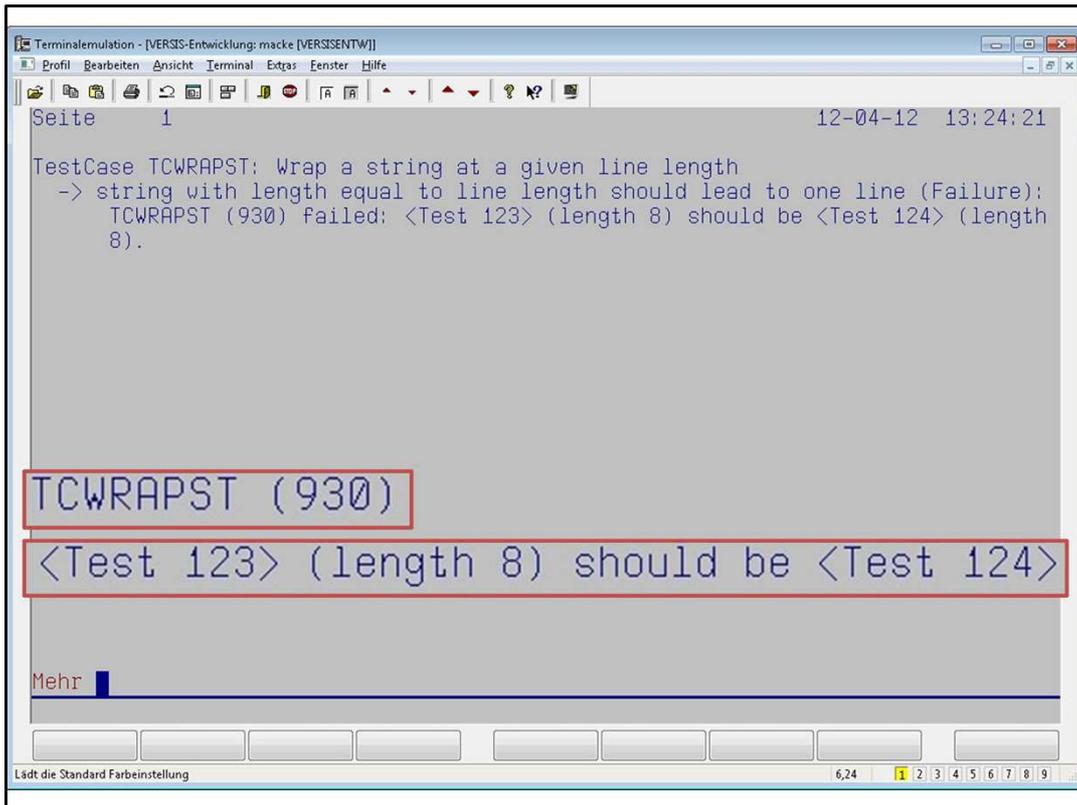


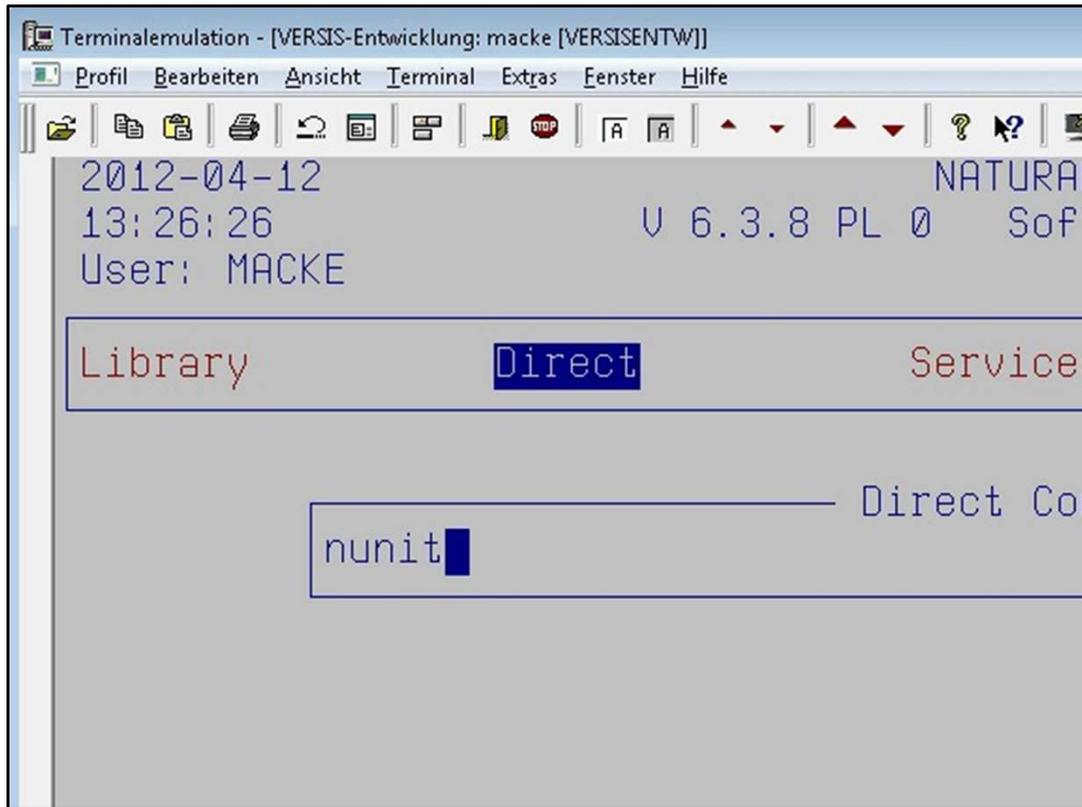
If the TestCase had failed, it would be represented by an F, an error (e.g. a Natural error) would be an E.



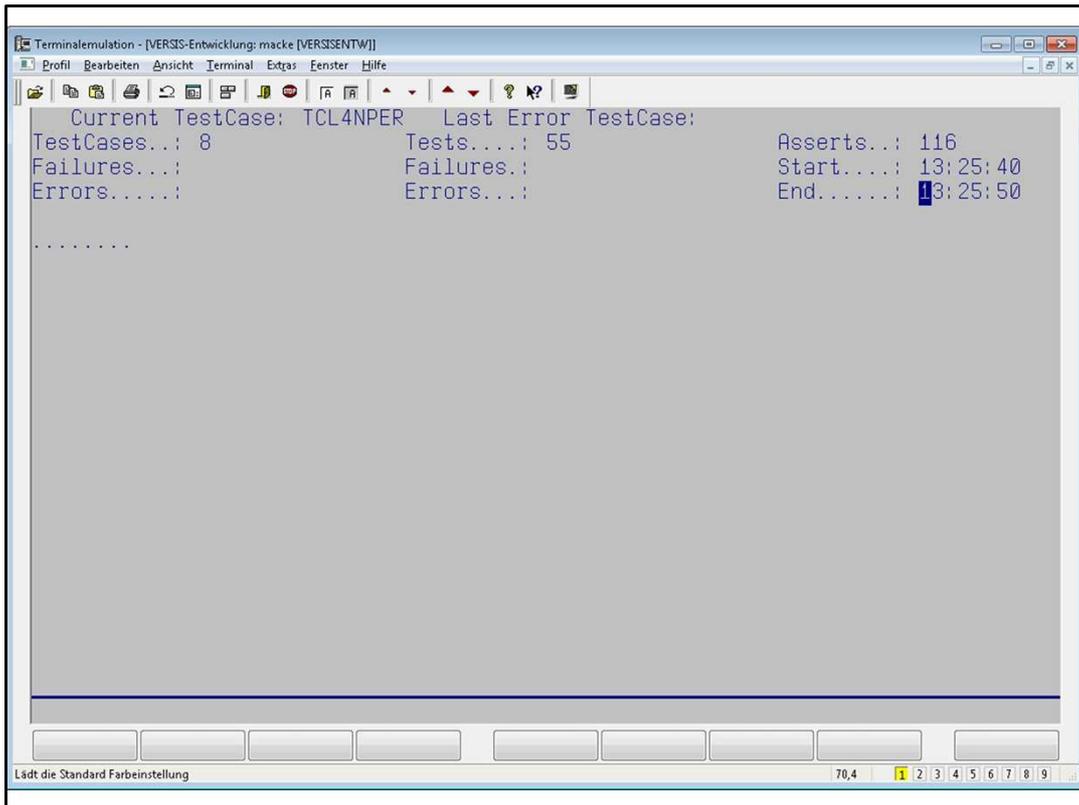


If a TestCase fails, the detailed information about what went wrong will be displayed on a separate screen.

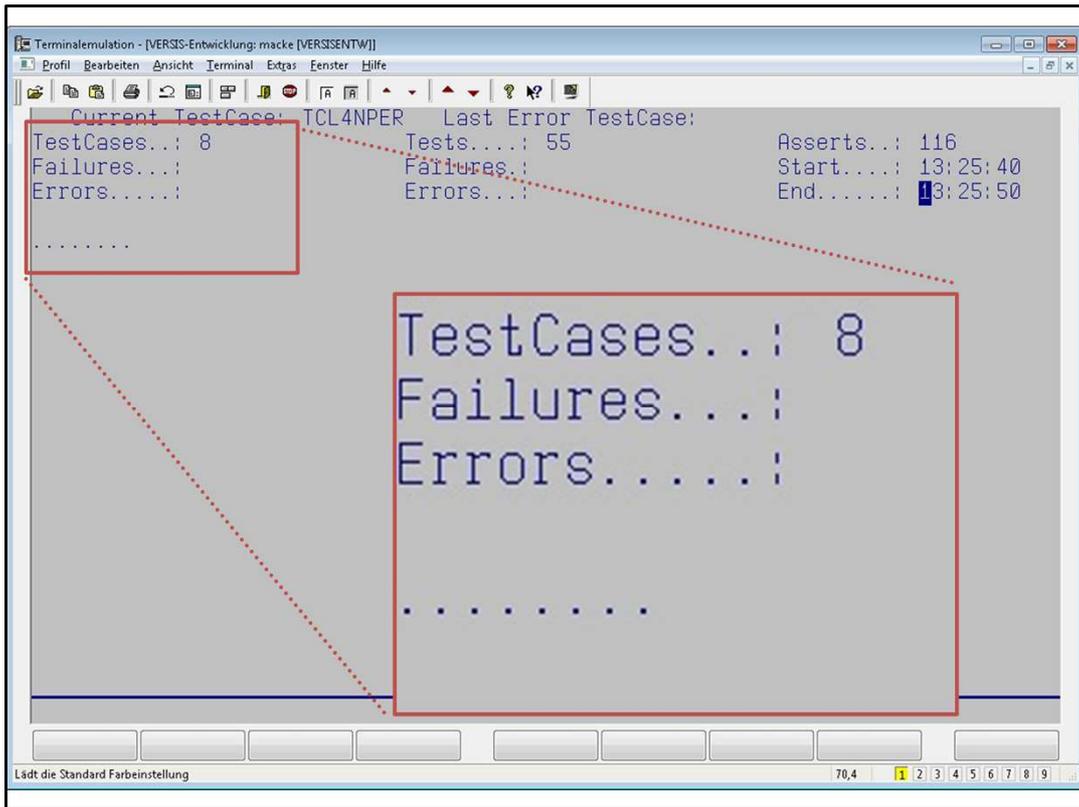




Of course you can run multiple TestCases at once using NUNIT.



As you can see, each TestCase results in a dot.





The results of the NatUnit test runs can be exported (as an XML file) to a continuous integration server like Hudson.

<http://hudson-ci.org/>

Testergebnis : (root)

Fehlschläge (±0)

Tests (±0)
Dauer: 1,5 Sekunden

Alle Tests

Klasse	Dauer	Fehlgeschlagen	(Diff.)	Übersprungen	(Diff.)	Summe	(Diff.)
<u>CAR</u>	1 ms	0		0		6	
<u>SUP</u>	69 ms	0		0		81	
<u>UTILITY</u>	0,38 Sekunden	0		0		368	
<u>V-SYSTEM</u>	1 Sekunde	0		0		3854	
<u>V-USER</u>	30 ms	0		0		676	

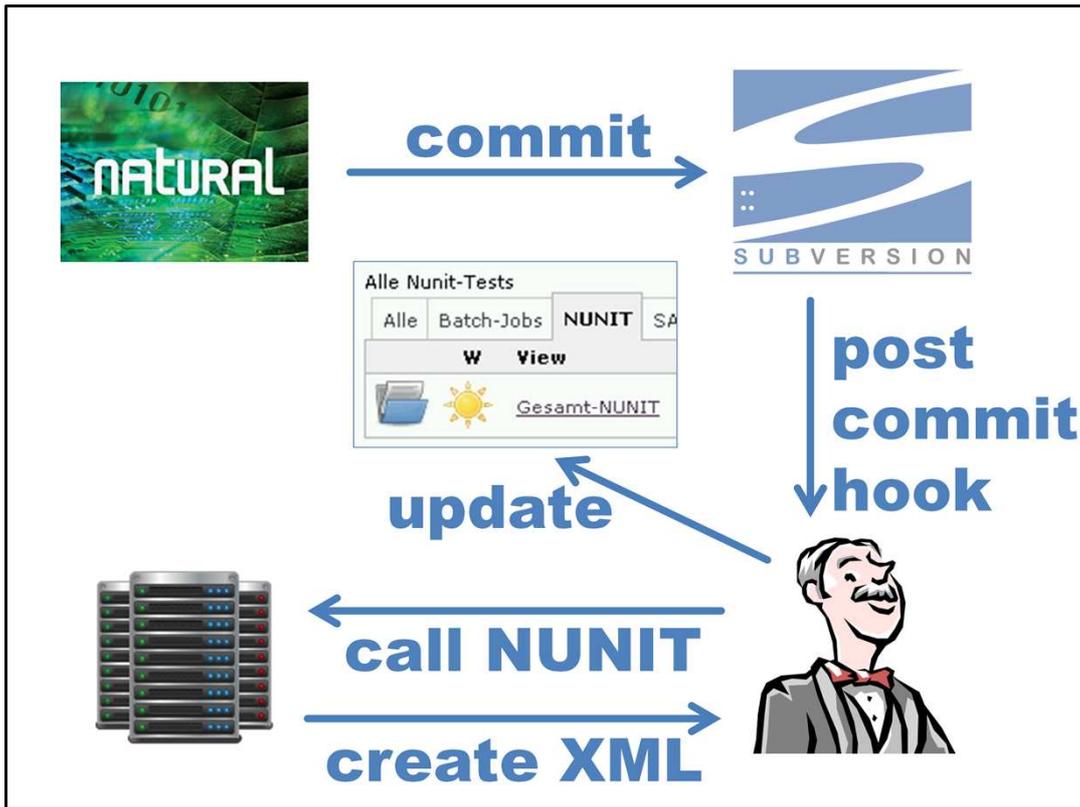
As you can see, the tests run pretty fast (which is needed for them to be run as often as possible).

Tests (± 0)
Dauer: 1,5 Sekunden

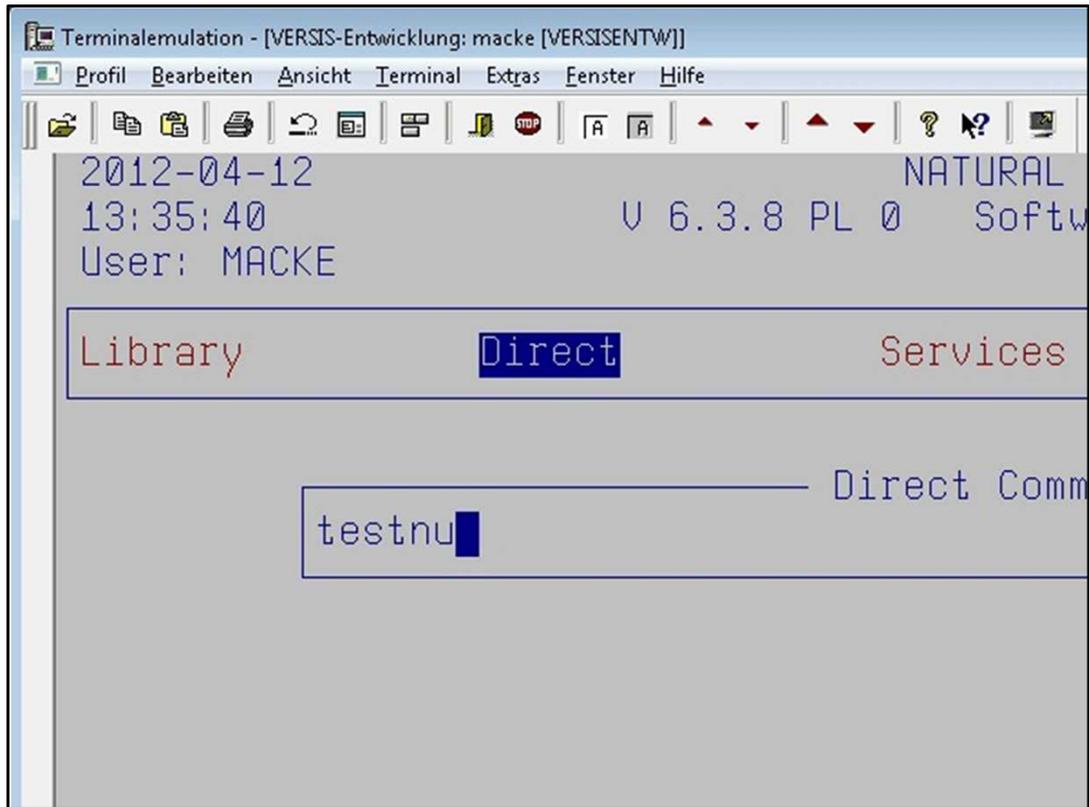
Summe	(Diff.)
6	
81	
368	
3854	
676	

Tests (± 0)
Dauer: 1,5 Sekunden

(Diff.)	Übersprungen	(Diff.)	Summe	(Diff.)
	0		6	
	0		81	
	0		368	
	0		3854	
	0		676	



This flow diagram shows the interaction between Natural/SVN/Hudson.



NatUnit itself was developed using TDD. You can run the internal TestSuite by calling TESTNU.

The image shows a terminal window titled "Terminal emulation - [VERSI-Entwicklung: macke [VERSI-ENTW]]". The window contains the following text:

```
Seite 1 12-04-12 13:35:58

NUTCASS Assertion tests: OK
NUTCSEQ Template method: OK
NUTCSRC Get Natural source for TestCase: OK
NUTCPARS Parse TestCase: OK
NUTCCOLL Collect results from TestCases: OK
NUTCGTTC Get TestCases from string array: OK
NUTCGTTS Get TestCases for TestSuite: OK

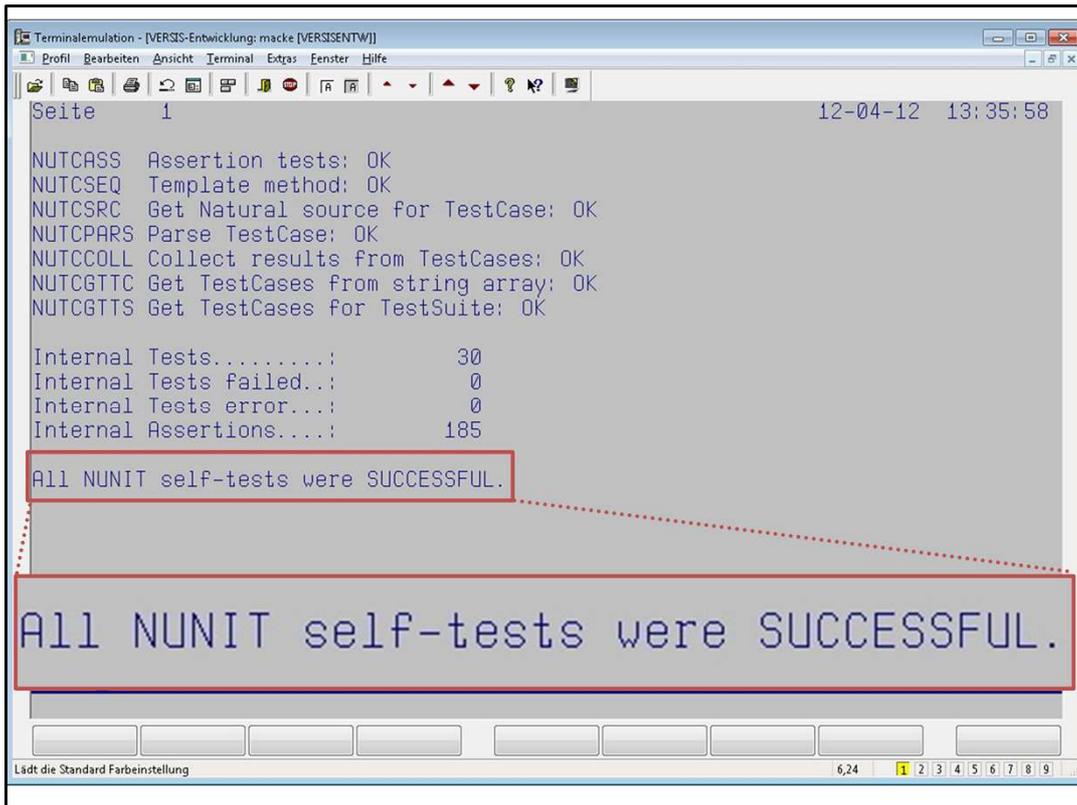
Internal Tests.....:      30
Internal Tests failed..:    0
Internal Tests error...:    0
Internal Assertions....:   185

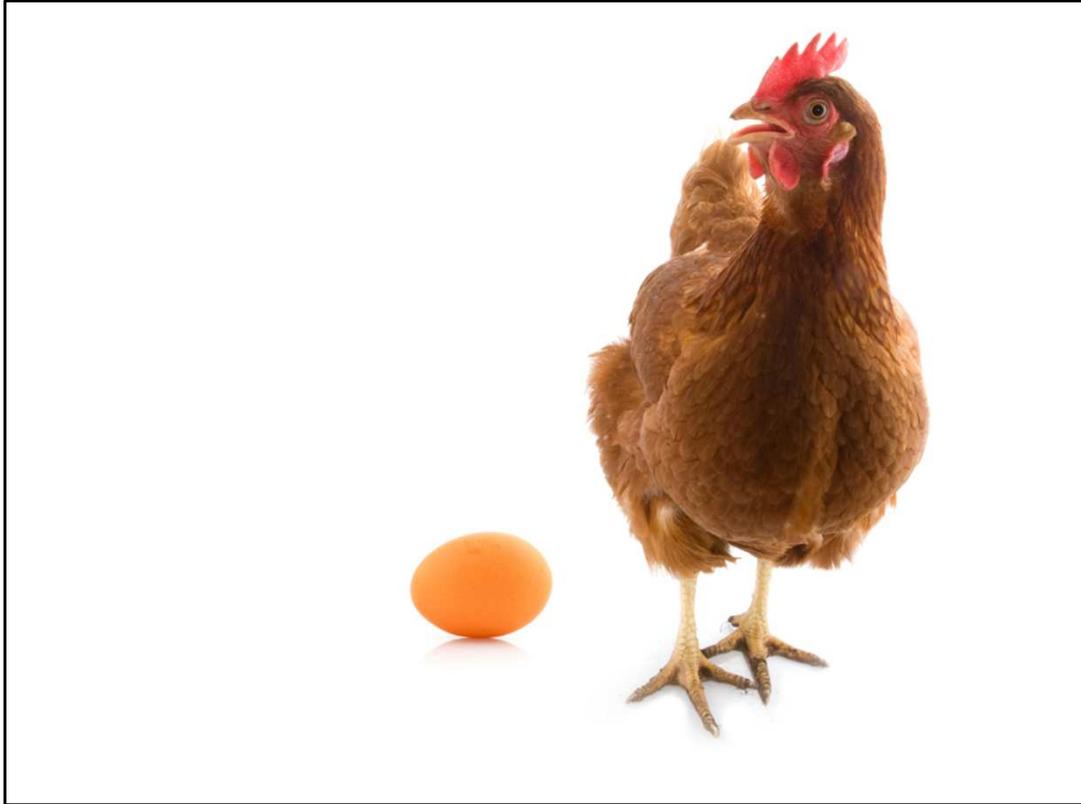
All NUNIT self-tests were SUCCESSFUL.

Mehr
```

At the bottom of the terminal window, there is a status bar with the text "Lädt die Standard Farbeinstellung" on the left, a small number "6,24" in the middle, and a row of numbers "1 2 3 4 5 6 7 8 9" on the right.

Here's an overview of the internal tests' results.





TDD of a test framework seems a bit like a chicken-and-egg problem, if you would like to know how I did it, check out my Master's thesis.

<http://blog.stefan-macke.com/2009/12/29/nunit-a-unit-test-framework-for-natural/>



If you want to get started writing unit tests, there are a few rules for writing good tests. I can't show all of them, but the most important ones are...

...are

fast

and

easily

runnable



Tests need to be fast (i.e. milliseconds instead of seconds) to provide instant feedback to the developer and they have to be easy to run (e.g. by starting a single program like NUNIT) so the developer can run them as often as possible.



To make the Tests fast and reliable (i.e. producing the same results everytime they run), they are not allowed to cross boundaries, i.e. infrastructure components like databases, the network, or harddrives should not be accessed in the Tests or in the modules under test.



That approach seems reasonable (even if you do not write tests at all) and could easily be implemented in green field projects.



However, most developers do not work on green field projects but on brown field projects with existing code bases (also called Legacy Code).

```
READ IMPORTANT-DDM BY SUPERDESCRIPTOR
*
  IF FIELD EQ 'value'
*   insert business logic here
  ELSE
    ESCAPE TOP
  END-IF
*
  INPUT USING MAP 'OUTPUT'
*
END-READ
```

Here's an example...

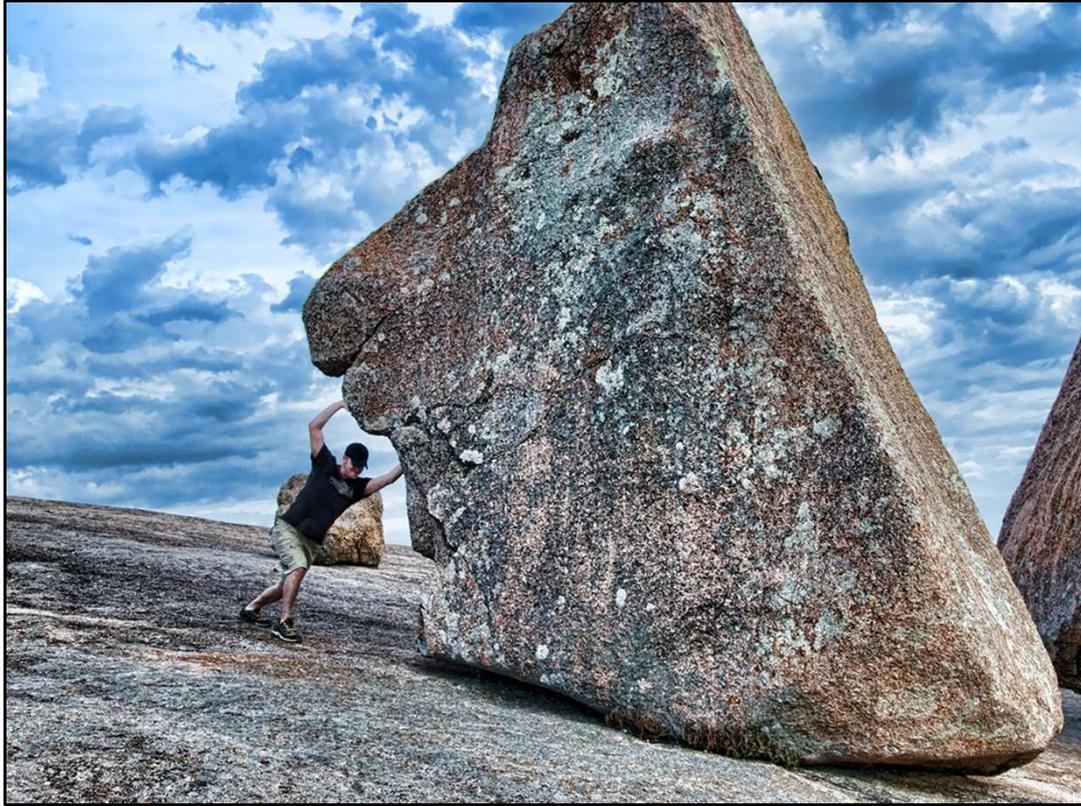
```
READ IMPORTANT-DDM BY SUPERDESCRIPTOR
*
  IF FIELD EQ 'value'
*   insert business logic here
  ELSE
    ESCAPE TOP
  END-IF
*
  INPUT USING MAP 'OUTPUT'
*
END-READ
```

...this program is hard to test, because it does three things at once: access the database, process the data, and display the data.

By the way: this is a program so you cannot test it at all because it returns no parameters which could be asserted against.

```
DEFINE DATA  
PARAMETER USING BUSIPARA  
END-DEFINE  
*  
IF FIELD EQ 'value'  
*   insert business logic here  
ELSE  
    RETURNCODE := C-RC-ERROR  
END-IF  
*  
END
```

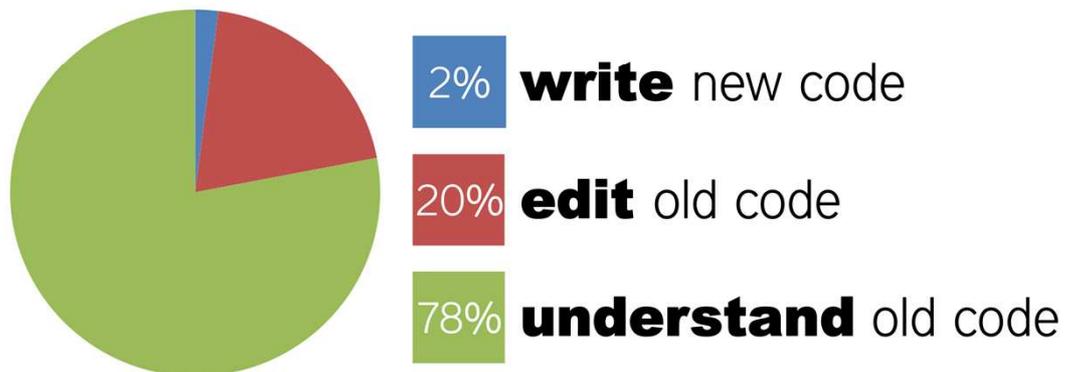
So, you have to break down the different tasks into different modules, e.g. a subprogram/subroutine that only processes the data (business logic) given to it via PDAs and also returns its results. This way, the business logic can be tested separately from the DB and the view.



However, this means hard work at first, especially if you work on a large existing code base. Many existing programs will be hard to change and modularize.

daily work of a software developer

(according to Peter Hallam and Jeff Atwood)



The modularization combined with Tests that speak to the developer (they provide examples/documentation of the code's behaviour) help the developers to understand (even their own) code and do a better job changing and extending it.

<http://blogs.msdn.com/peterhal/archive/2006/01/04/509302.aspx>

<http://www.codinghorror.com/blog/archives/000684.html>



Once you start writing unit tests and see the advantages for your daily work you will never go back to not writing tests. You will get test infected (as Kent Beck calls it).

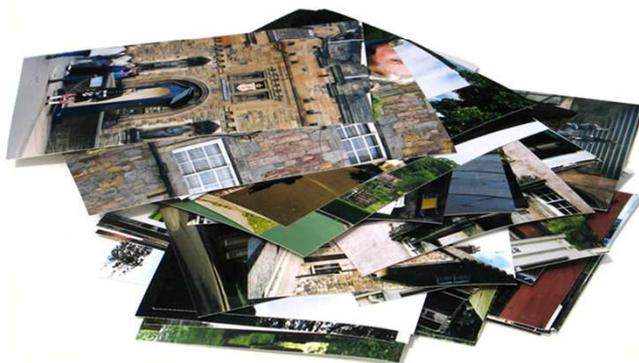
<http://junit.sourceforge.net/doc/testinfected/testing.htm>



To get started simply visit NatUnit on SourceForge.



picture **credits**





johnmarchan

www.flickr.com/photos/johnmarchan/562116408/



ferrantraite

www.istockphoto.com/stock-photo-6080208-bored-businessman.php



yuan2003

www.flickr.com/photos/yuan2003/130559143/



Pedro Simoes

www.flickr.com/photos/pedrosimoes/7/190673196/



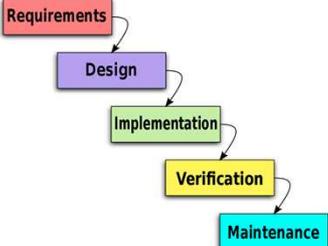
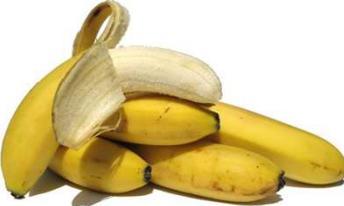
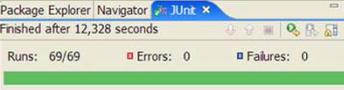
416style

www.flickr.com/photos/sookie/36356334/



Christy C

www.flickr.com/photos/christy_chen/3603006756/

 <pre>graph TD; A[Requirements] --> B[Design]; B --> C[Implementation]; C --> D[Verification]; D --> E[Maintenance];</pre>	<p>Paul Smith en.wikipedia.org/wiki/File:Waterfall_model_%281%29.svg</p>
	<p>sanja gjenero www.sxc.hu/photo/1186300</p>
	<p>Ben Hermann www.flickr.com/photos/theredroom/121963809/</p>



Murat Giray Kaya

www.istockphoto.com/stock-photo-7529059-desperate-businessman.php



longhairthai.com

www.flickr.com/photos/longhairthai/3280485878/



sanja gjenero

www.sxc.hu/photo/1064586



Ove Topfer

www.sxc.hu/photo/998524



Arnett Gill

www.flickr.com/photos/gagillphoto/336353424/



harald walker

www.flickr.com/photos/sonicwalker/322373355/



Tamlyn Rhodes

www.sxc.hu/photo/499571



IvonneW

www.istockphoto.com/stock-photo-8566536-isolated-chicken-with-egg.php



sanja gjenero

www.sxc.hu/photo/731544



JasonGulledge

www.flickr.com/photos/ramdac/373881476/



Nispa

www.sxc.hu/photo/678805



Igor Dugonjic

www.sxc.hu/photo/859291



Thomas Lobker

www.sxc.hu/photo/175983



jan flaska

www.sxc.hu/photo/1113494



degem

www.flickr.com/photos/degem/92304899/



Kristin Bradley

www.flickr.com/photos/krikrit/2880756271/



Bliz

www.istockphoto.com/stock-photo-4400982-good-news.php



Michal Zacharzewski

www.sxc.hu/photo/431263



Michele

www.flickr.com/photos/damaradeael/2822846819/



sanja gjenero

www.sxc.hu/photo/779191

NatUnit

f.macke.it/NatUnit

STEFAN MACKE

ALTE OLDENBURGER
Krankenversicherung AG

