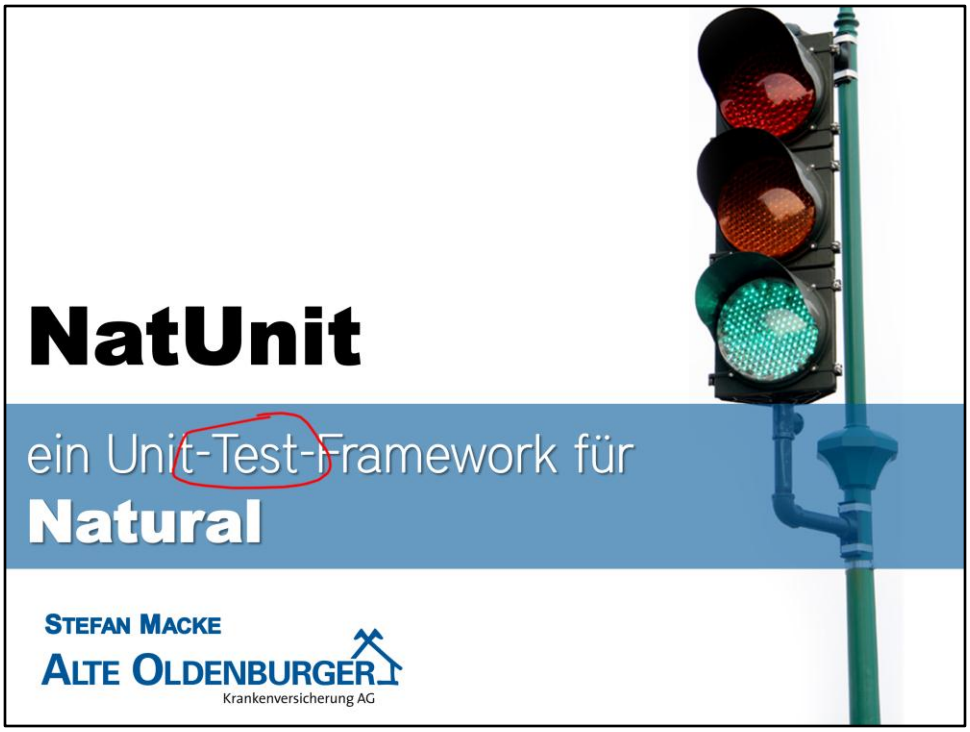
NatUnit

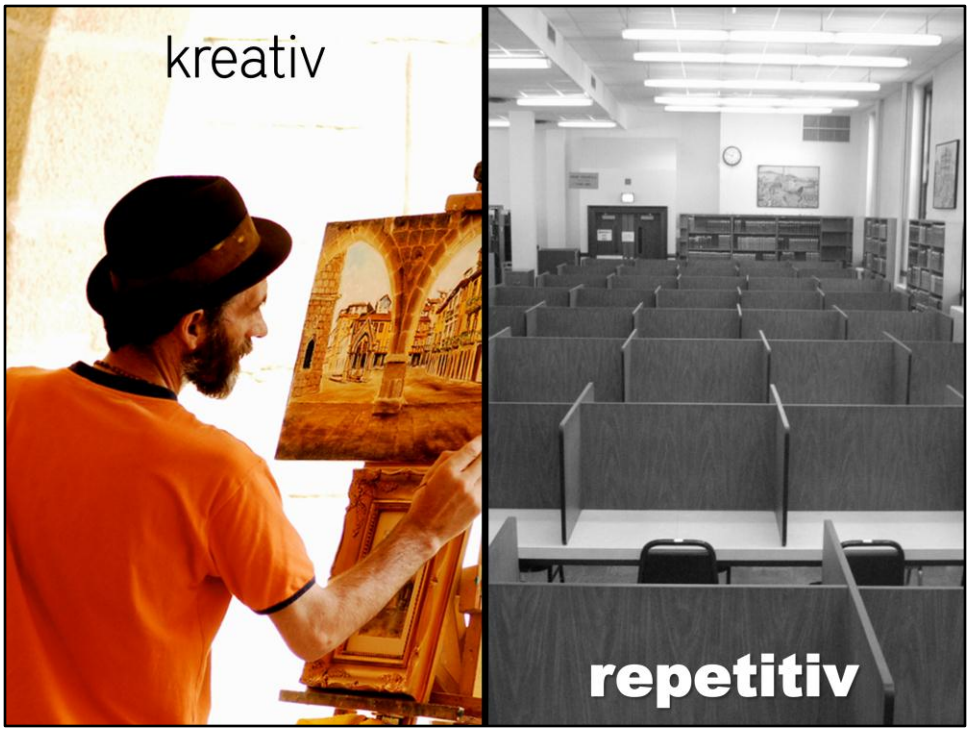
ein Unit-Test-Framework für **Natural**

STEFAN MACKE
ALTE OLDENBURGER
Krankenversicherung AG



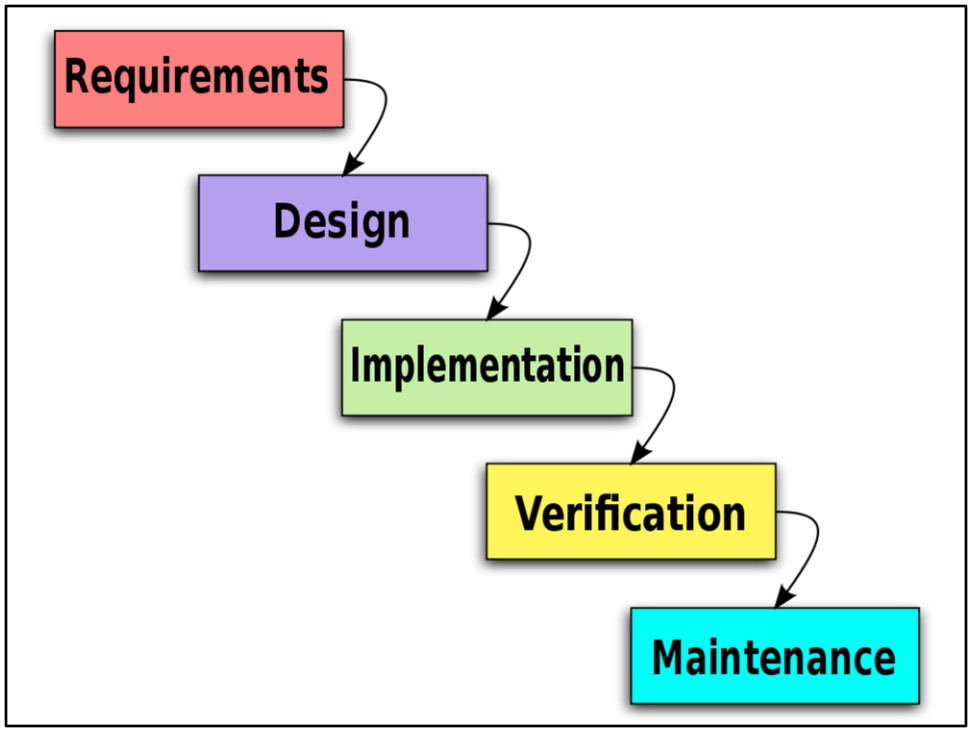
Viele Entwickler finden Testen langweilig.

Das galt auch für mich und meine Kollegen.



Programmierung → kreativ
Testen → wiederholend, langweilig, eintönig

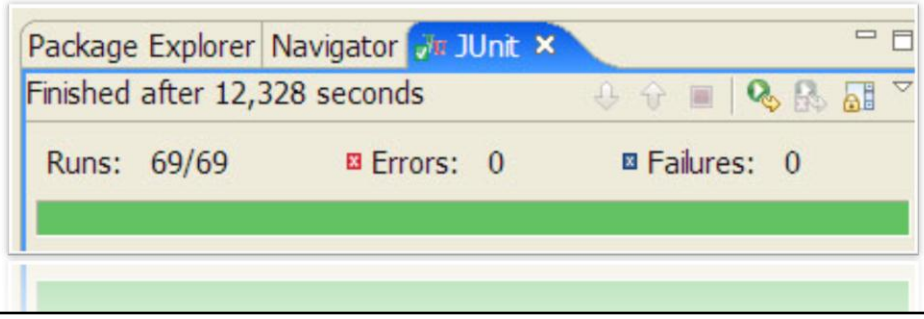
Programmierung → konstruktiv, neuen Nutzen schaffen
Testen → destruktiv, Bestehendes kaputtmachen



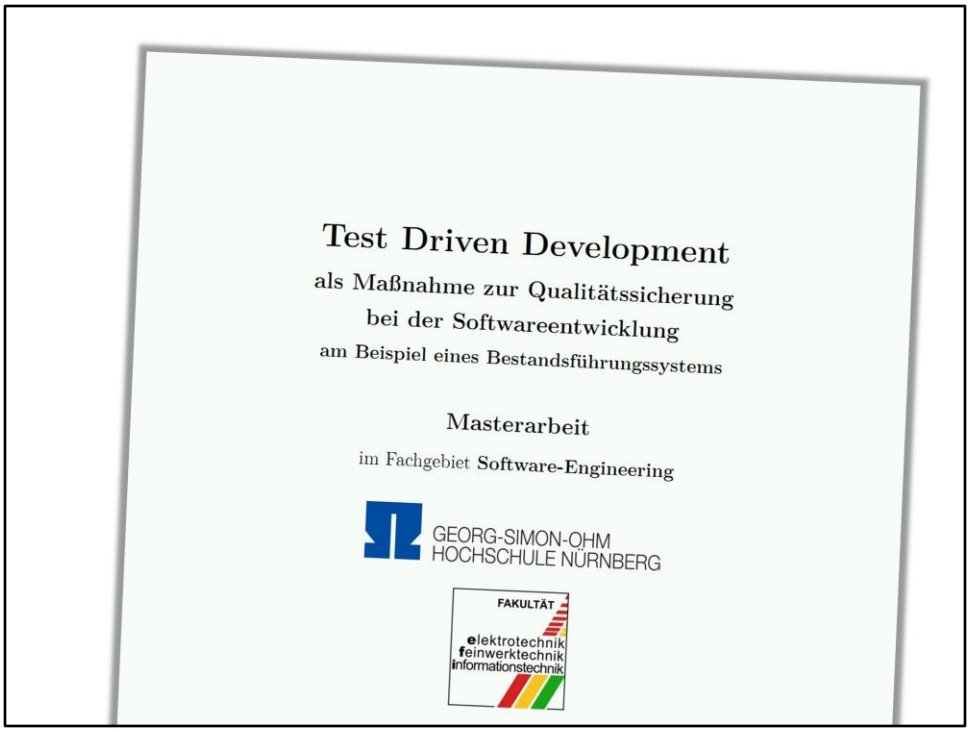
Klassische Prozesse → Testen kommt am Schluss, wird häufig weggelassen

Folge → Bananenprodukte, die beim Kunden reifen.

Unit-**Tests**

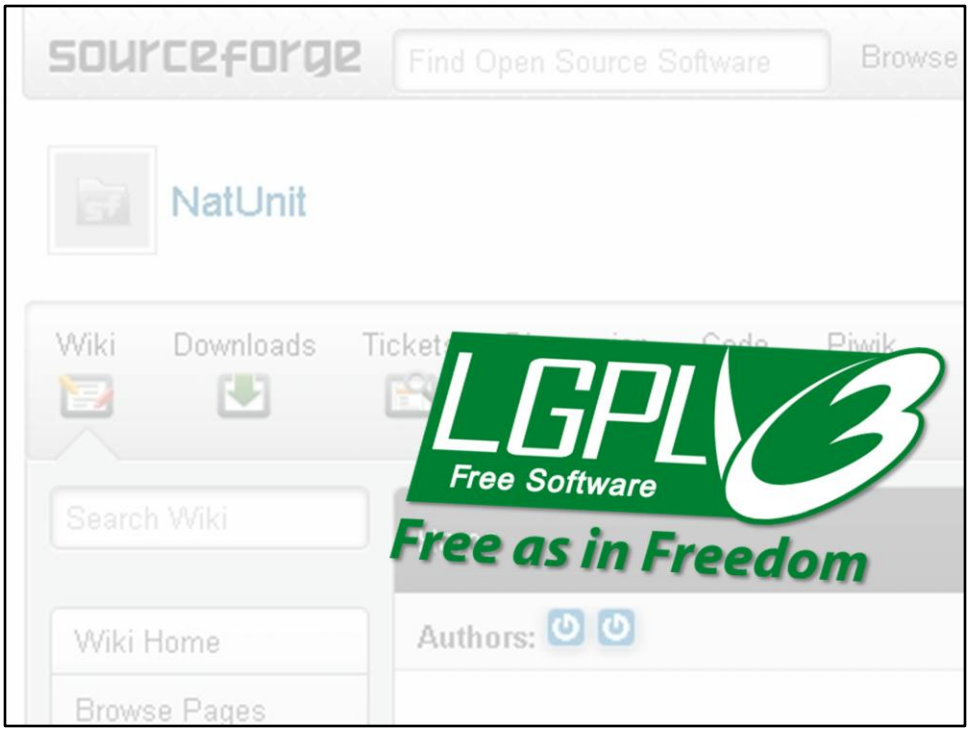


Mögliche Lösung: automatische Tests.
Unit-Tests sind heutzutage Standard, gerade in agilen Prozessen.

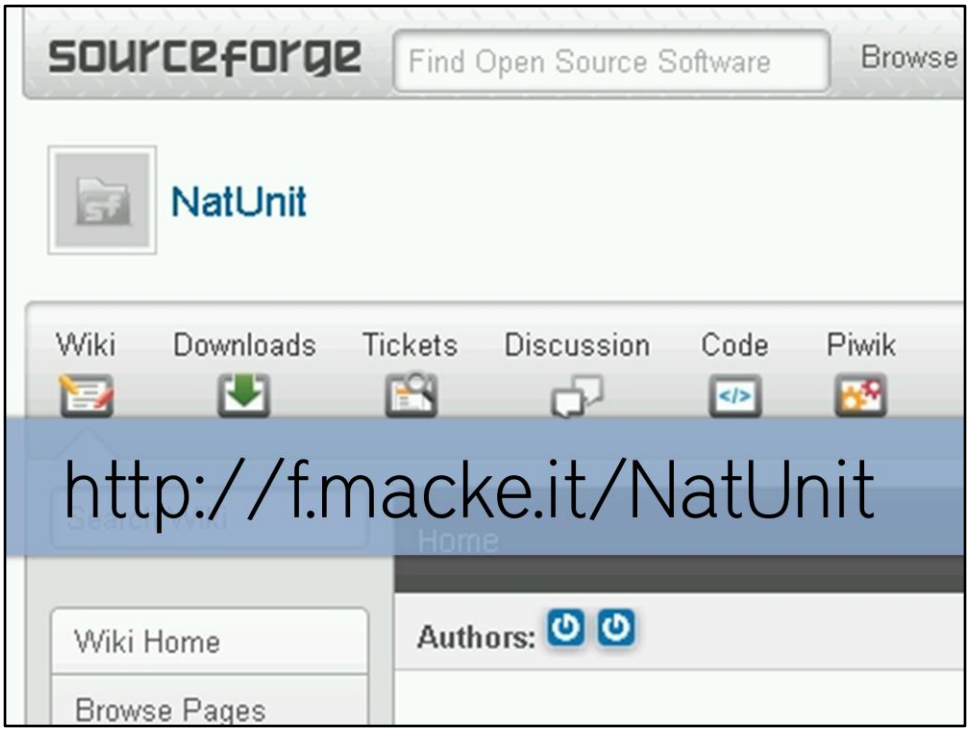


Für Natural gab es nichts → NatUnit als Teil meiner Masterarbeit.

Die aktuelle Version von NatUnit gibt es seit ein paar Monaten auf SourceForge.



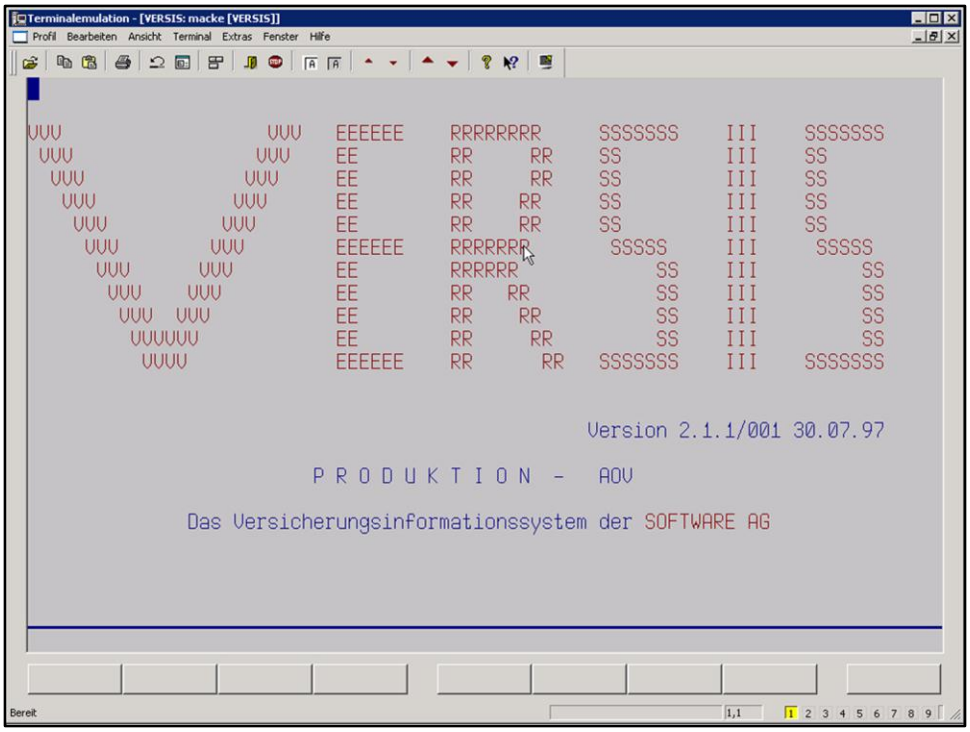
NatUnit kann kostenlos genutzt und weiterentwickelt werden, solange wir als Autor genannt werden.


sourceforge
Find Open Source Software
Browse
NatUnit
Wiki
Downloads
Tickets
Discussion
Code
Piwik
http://f.macke.it/NatUnit
Home
Wiki Home
Browse Pages
Authors:

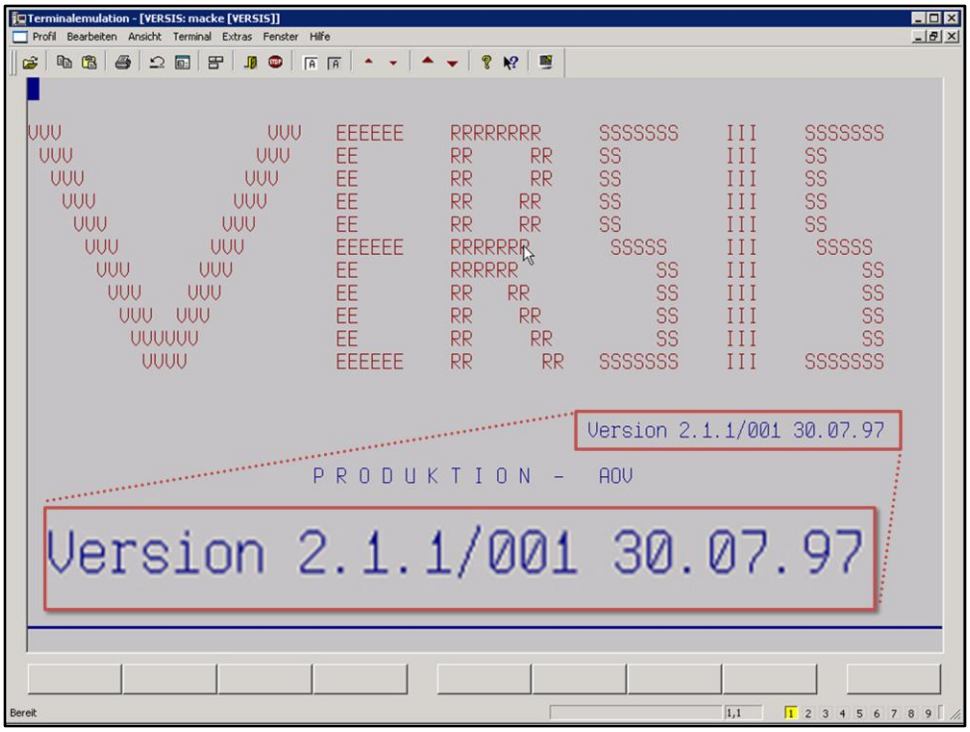
Stefan Macke
stefan@macke.it
blog.macke.it
@StefanMacke

AO → Norddeutschland, ca. 250 Mitarbeiter

Ca. 20 EDV-Mitarbeiter, 15 Entwickler



VERSIS → zentrales Bestandsführungssystem



In den 90er Jahren entwickelt durch SAG und AO

2008 wurde VERSIS 10.
All das war möglich ohne einen einzigen Unit-Test.

Warum sollte man sich also um Unit-Tests kümmern?

Die Vorteile:

- Sicheres Refactoring
- Keine langweiligen und teuren manuellen Tests mehr
- Tests forcieren sauberen Code
- Entwickler bekommen regelmäßiges Feedback

> Das **oberste Ziel** ist, ein Framework zu schreiben, das uns einen **Hoffnungsschimmer** gibt, dass die Entwickler wirklich Tests schreiben.

Erich **Gamma**

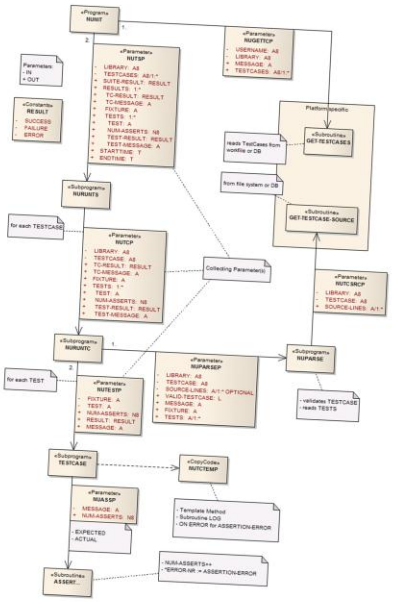
Wie bekommen wir Entwickler dazu, Tests zu schreiben? → einfaches Framework

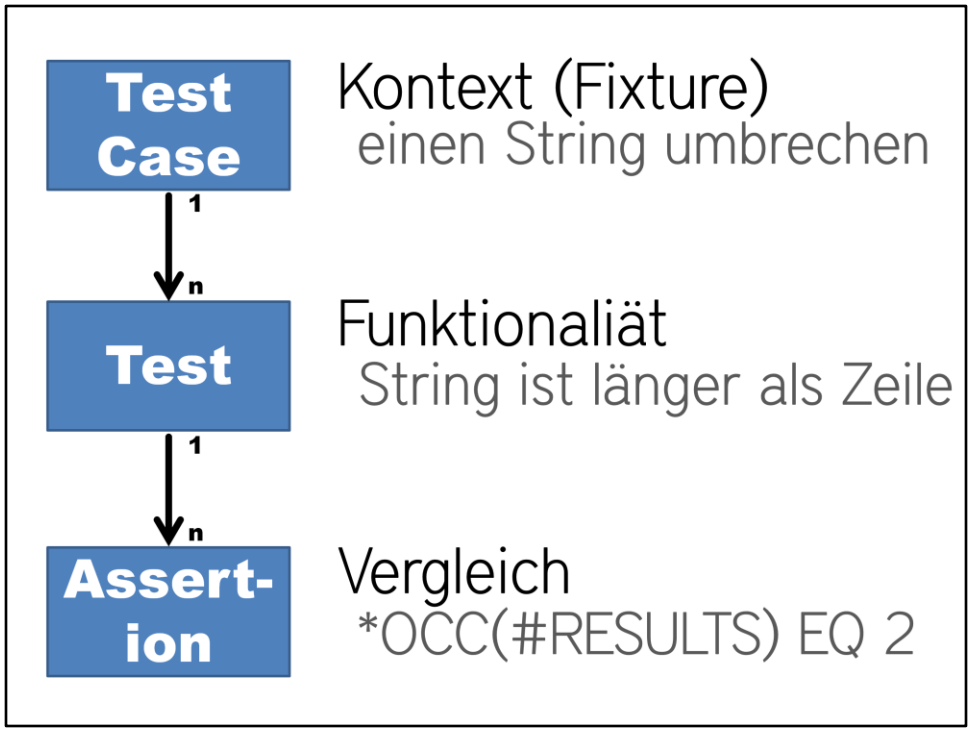
Wichtigster Punkt meiner Meinung nach: gleiche Programmiersprache wie Produktivcode.

→ Natural als Sprache für Unit-Tests

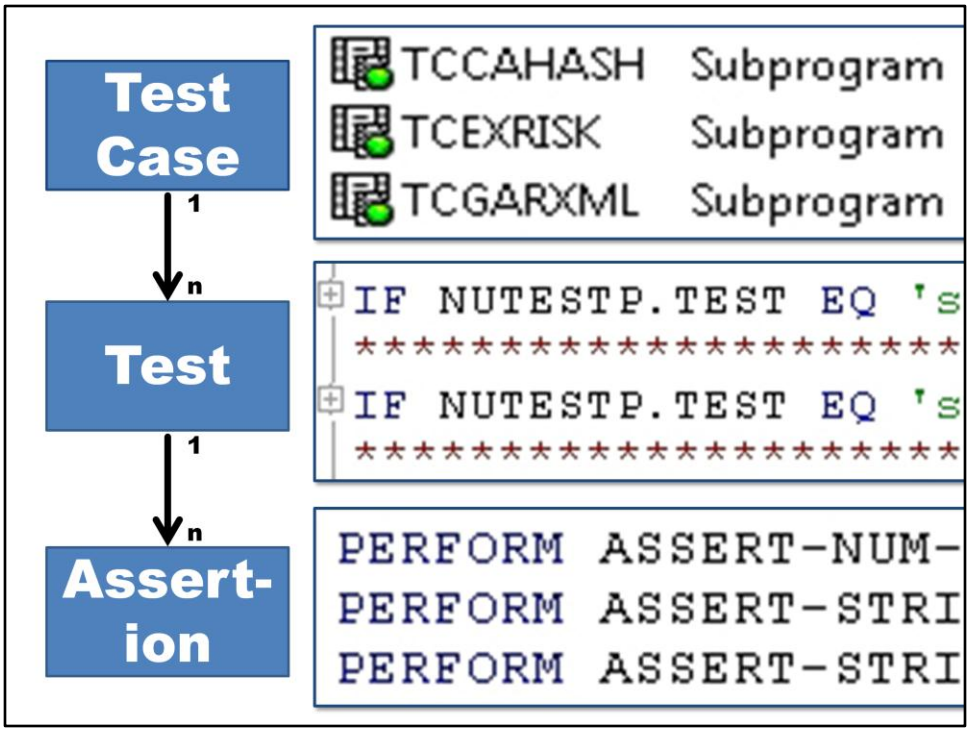
NatUnit

Das Framework





NatUnit wurde inspiriert durch Junit → bekannte Struktur

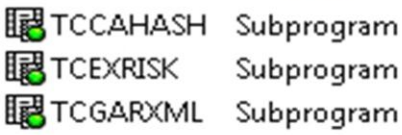


TestCase → Subprogram
Konvention TC*

Test → IF-Statements
sieht zuerst seltsam aus, Vorteile: Freitext

Assertion → externe Subroutine
leicht um eigene Assertions erweiterbar

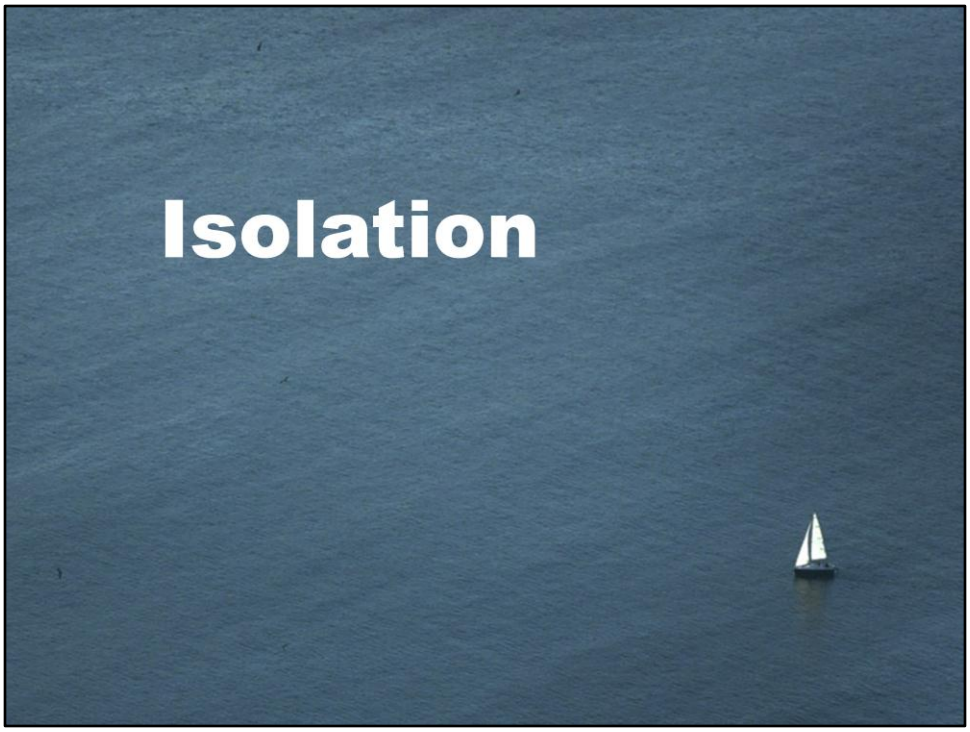
foreach TC in TestCases



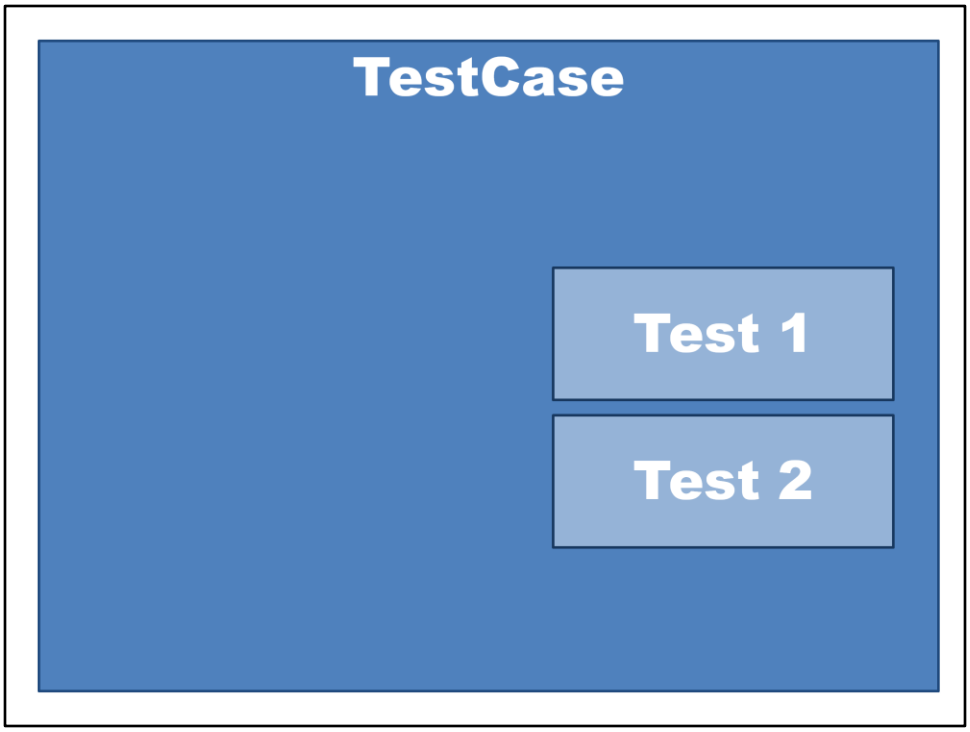
foreach T in TC.Tests

```
IF NUTESTP.TEST EQ 'string shorter than
**************************************
IF NUTESTP.TEST EQ 'string with length e
**************************************
```

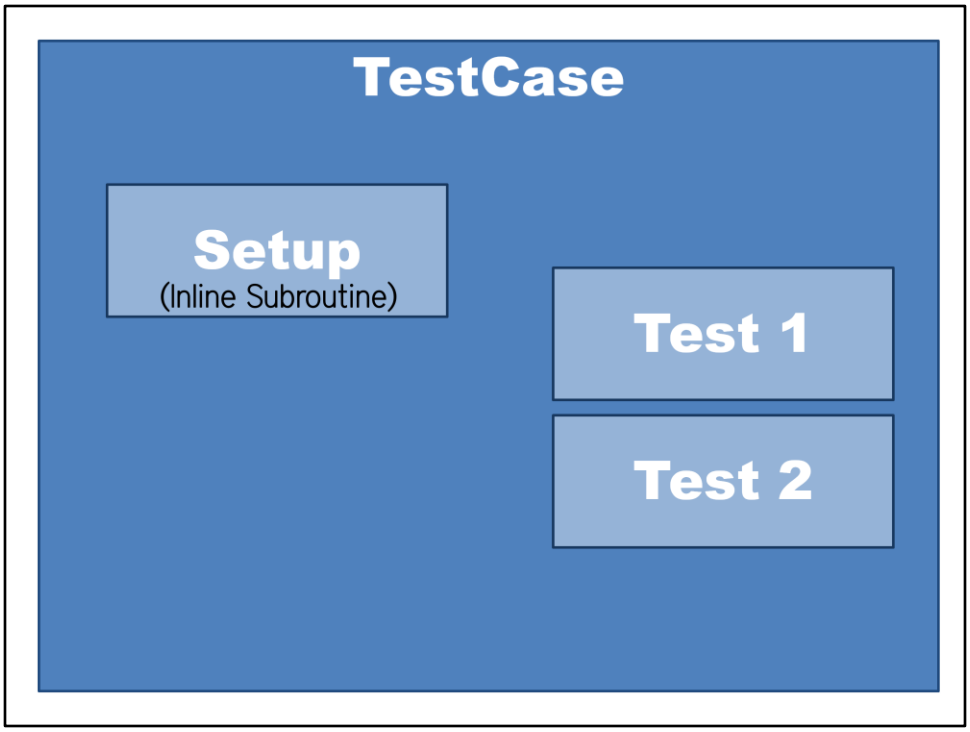
CALLNAT TC(T)



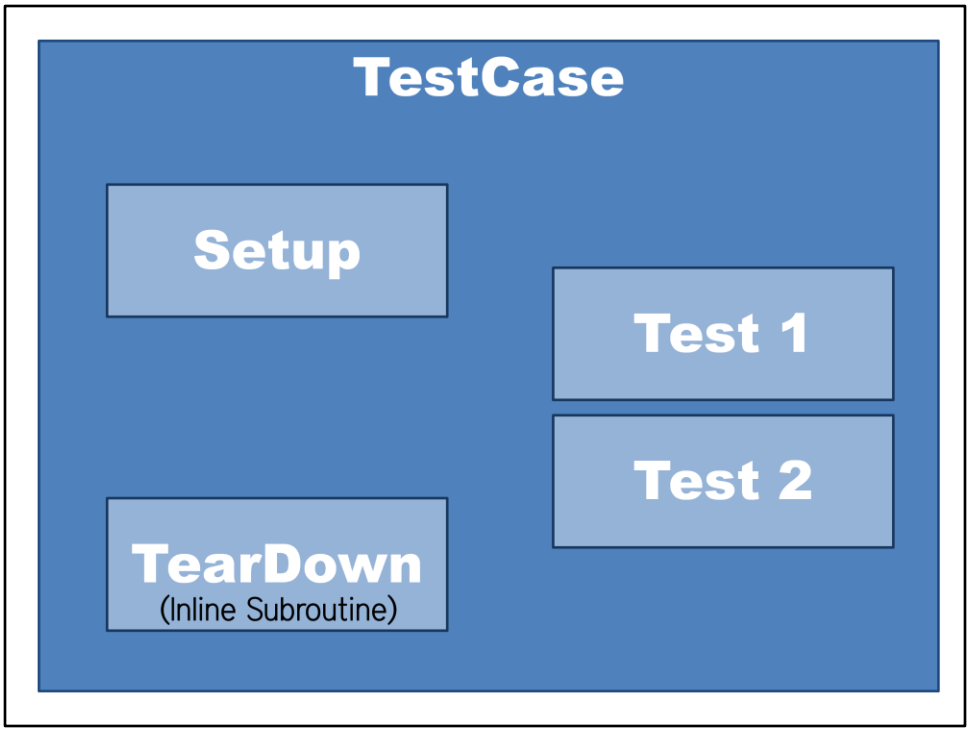
Alle Tests müssen isoliert voneinander laufen. Keine Abhängigkeiten untereinander!



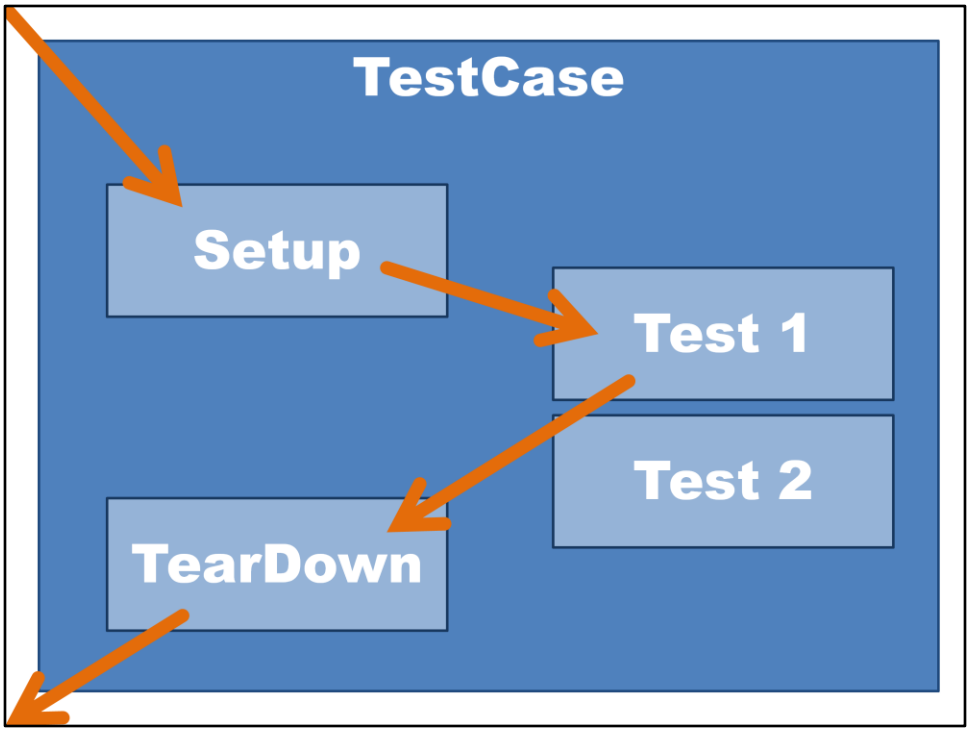
Das bedeutet, …



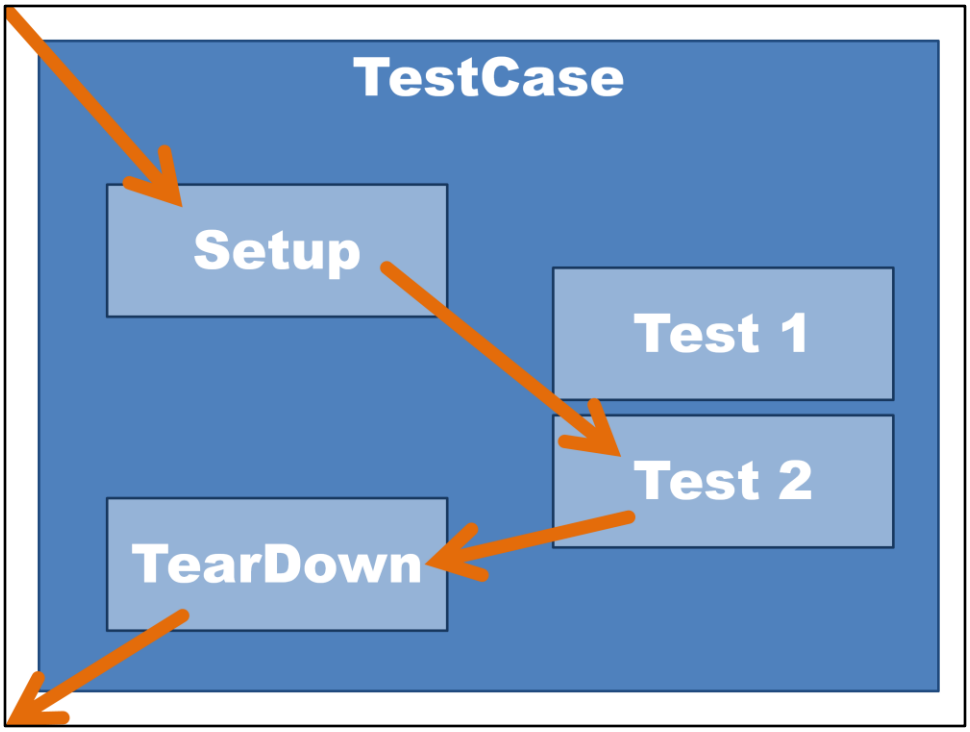
dass alle Voraussetzungen für jeden Test neu geschaffen…

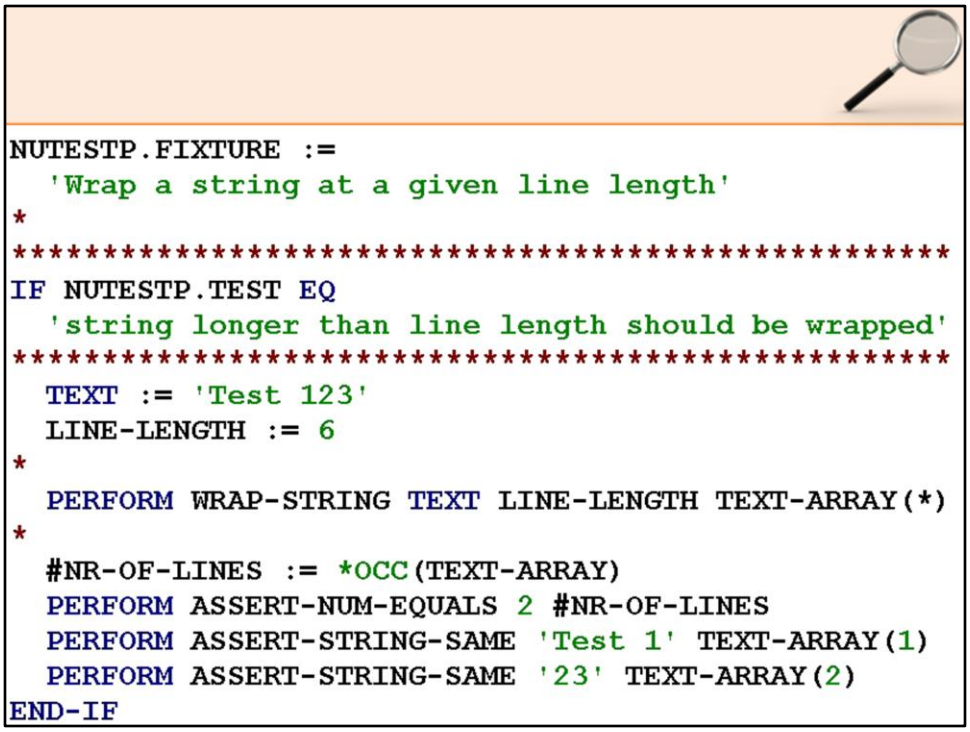


und nachher wieder entfernt werden müssen.



Bei jedem Aufruf eines Tests sorgt NatUnit für die nötige Ausführungsreihenfolge.


TestCase
Setup
Test 1
Test 2
TearDown



```
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
```

Beispieltest

```
NUTESTP.FIXTURE :=
   'Wrap a string at a given lin
```

```
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
```

Fixture → Beschreibung des Testfalls, Freitext

```
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
```

Test ist auch Freitext

```
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
```

Testimplementierung

1. **A**rrange
2. **A**ct
3. **A**ssert

Aufbau der Tests:
- Vorbereitung
- Durchführung
- Vergleich

```
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
```

Vorbereitung

PERFORM WRAP-STRING TEXT LINE-I

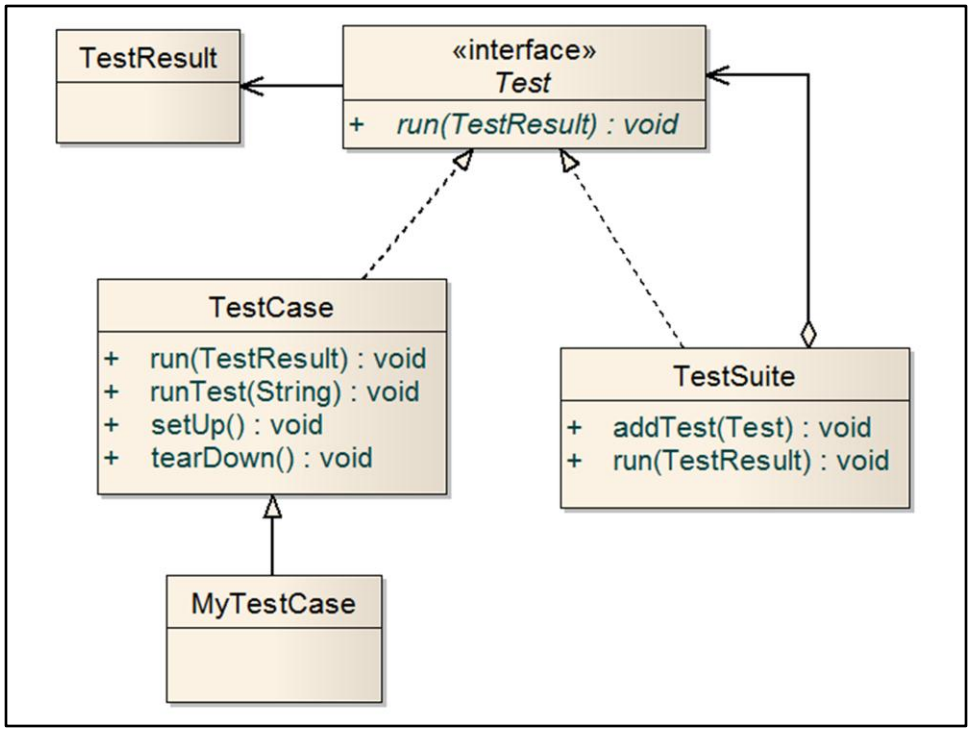
```
NUTESTP.FIXTURE :=
  'Wrap a string at a given line length'
*
*****************************************************
IF NUTESTP.TEST EQ
  'string longer than line length should be wrapped'
*****************************************************
  TEXT := 'Test 123'
  LINE-LENGTH := 6
*
  PERFORM WRAP-STRING 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
```

Aufruf

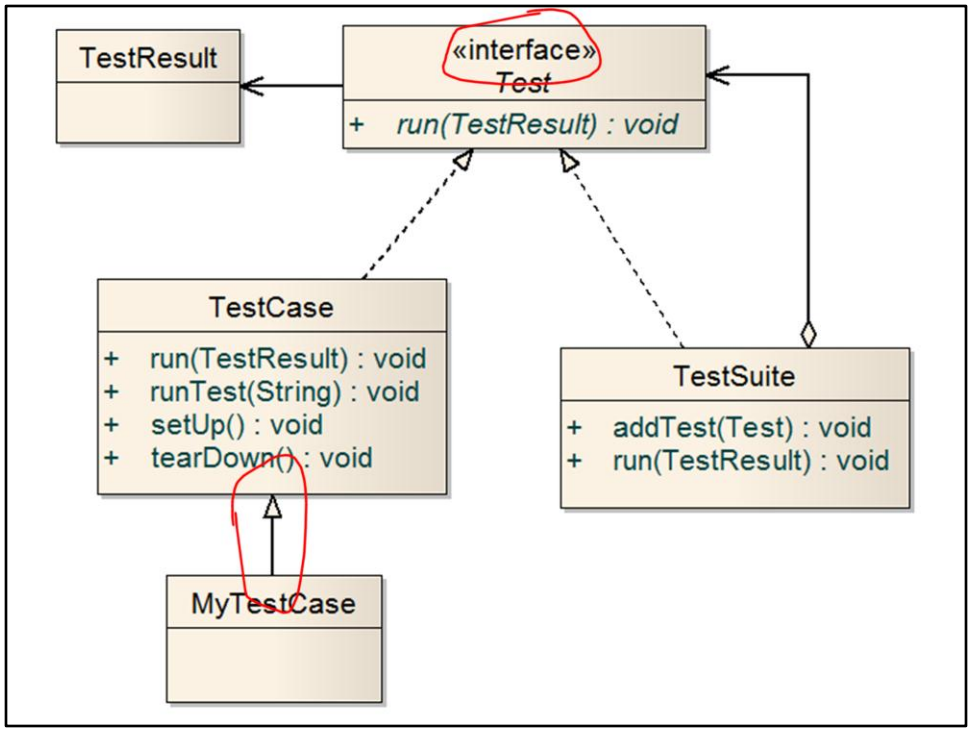
```
ASSERT-NUM-EQUALS 2 #NR-OF-LINE
ASSERT-STRING-SAME 'Test 1' TEX
NUTESTP.FIXTURE :=
  'Wrap a string at a given line length'
*
*****************************************************
IF NUTESTP.TEST EQ
  'string longer than line length should be wrapped'
*****************************************************
  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
```

Vergleich(e)

Aber wie kann diese ganze Aufruferei denn eigentlich funktionieren?



Die Vorlage war JUnit.



In JUnit werden hierfür OO-Konzepte wie Interfaces und Vererbung eingesetzt, was in Natural nicht möglich ist.

```
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
```

Damit NatUnit einen TestCase ausführen kann, muss er bestimmte Code-Bestandteile enthalten. Das wird durch einen kleinen Code-Parser geprüft.

```
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
```

Parameter und lokale Variablen

```
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
```

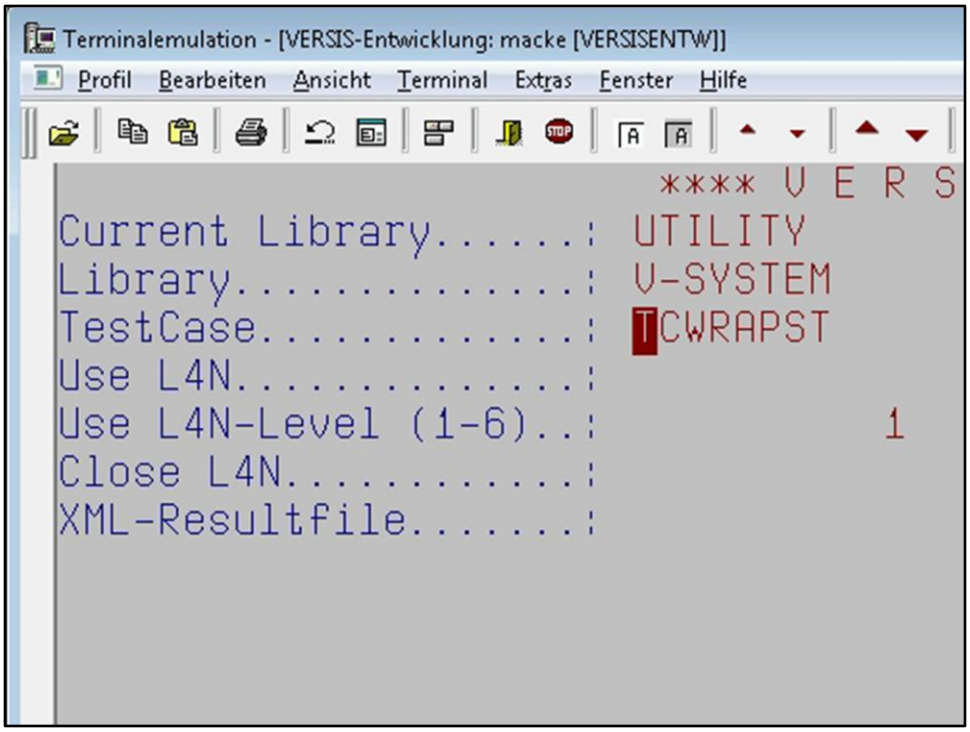
Ein Fixture

```
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
```

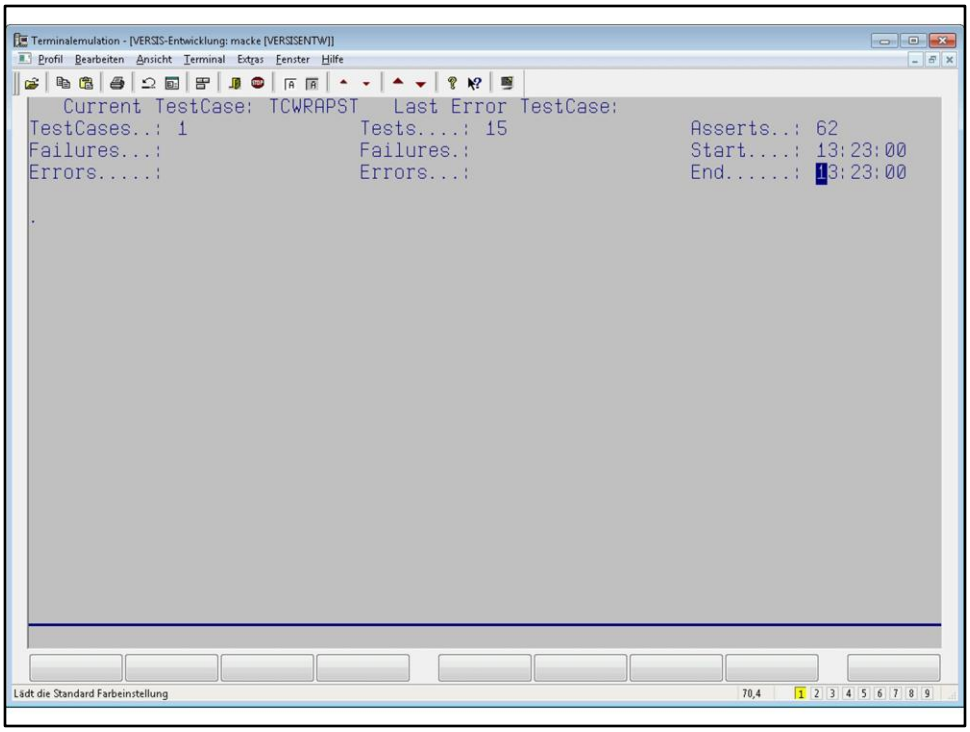
Die „Template-Methode“ und Setup/Teardown

```
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
```

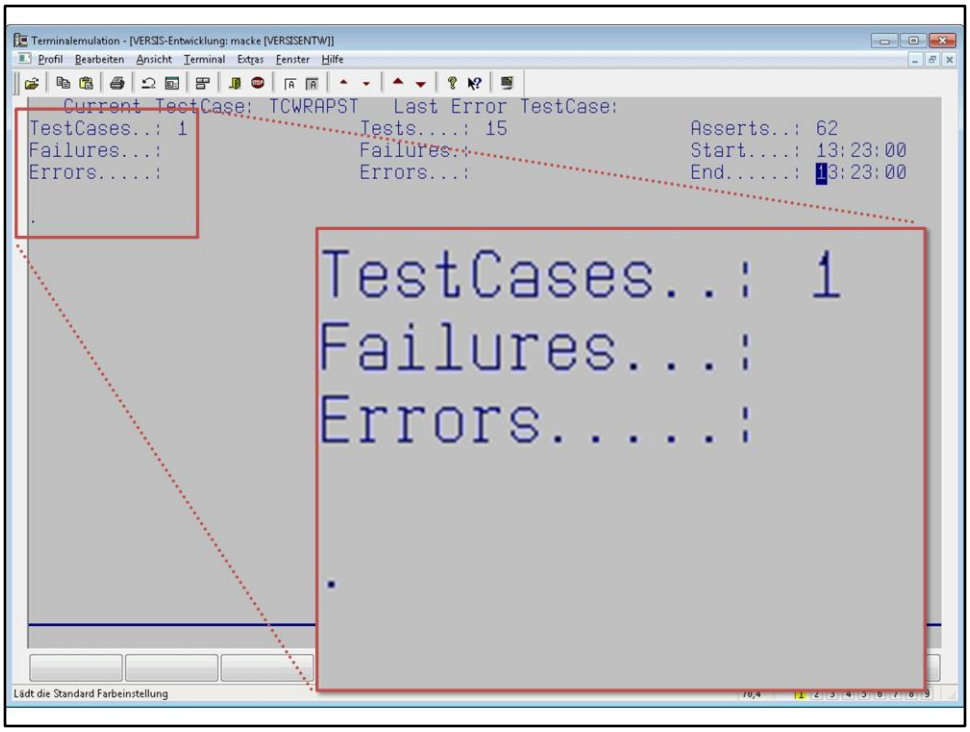
Und mindestens einen Test



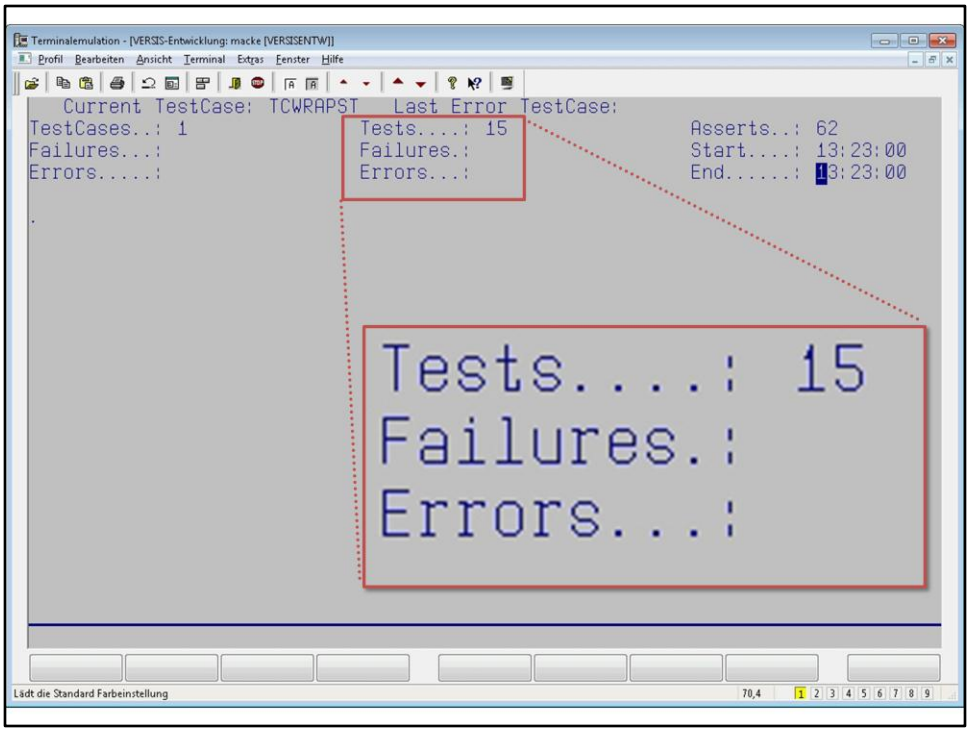
NUSINGLE führt einen einzelnen Test aus.

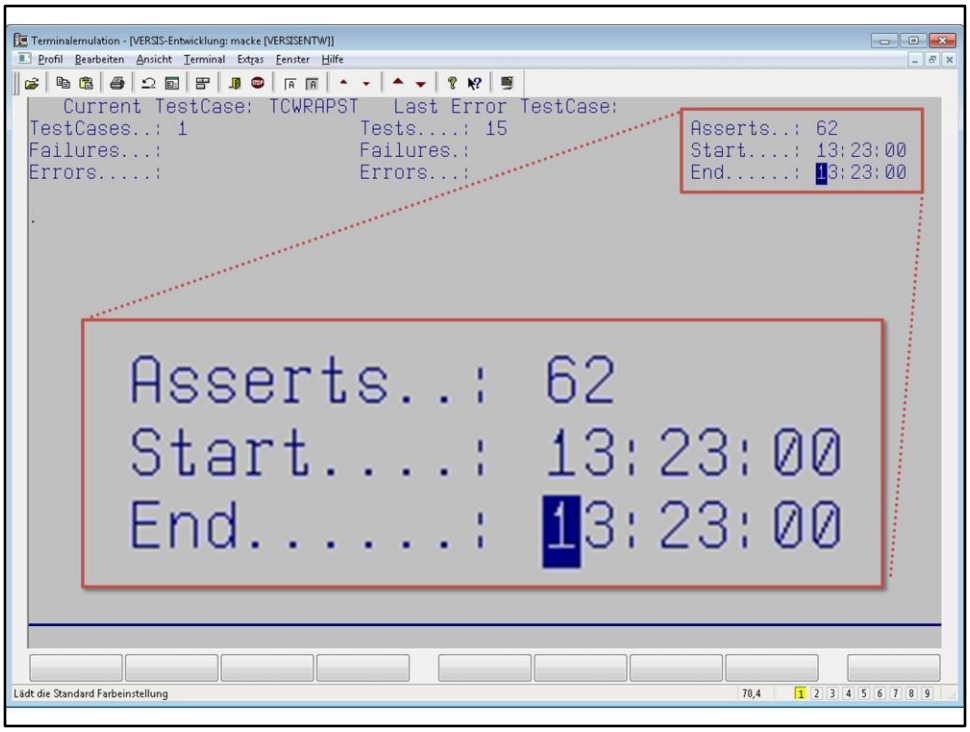


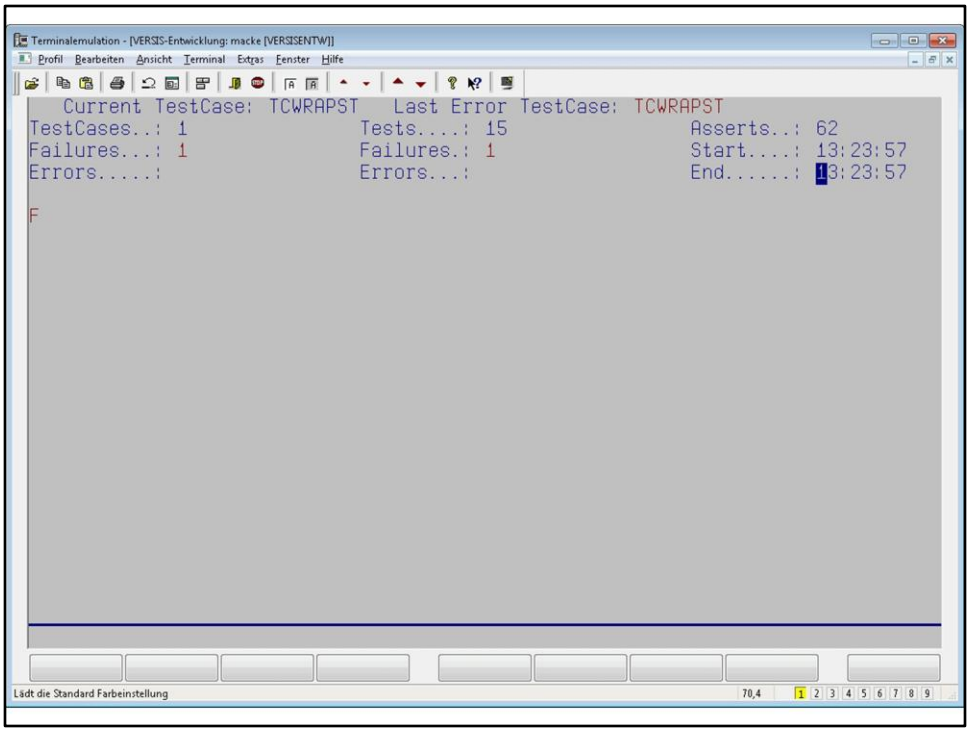
Testergebnis



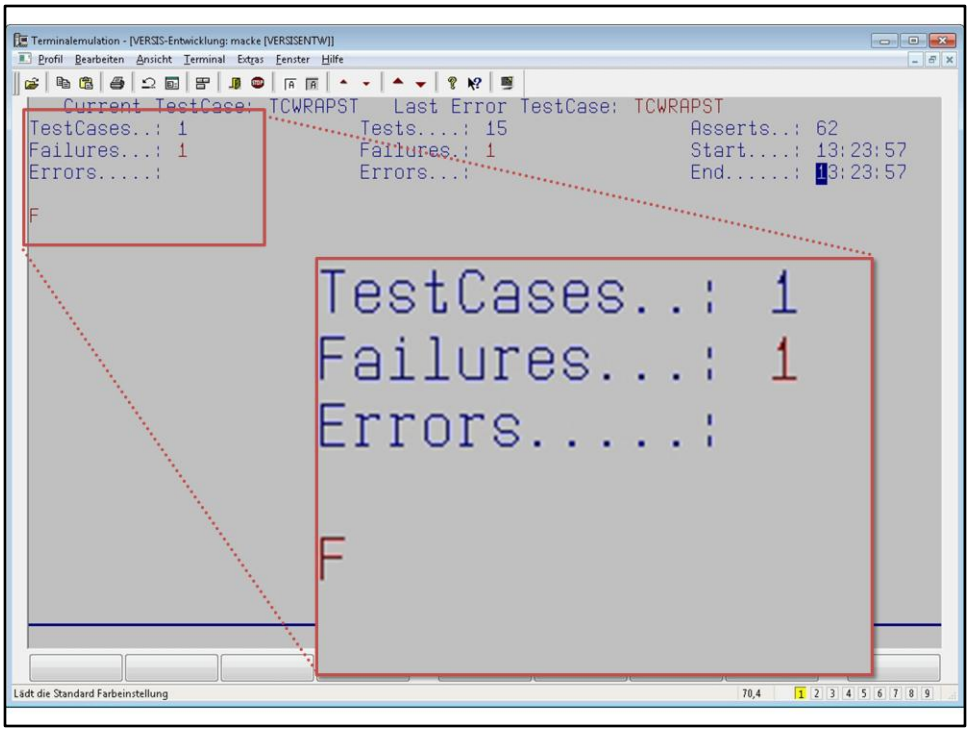
Jeder Punkt ist ein erfolgreicher TestCase

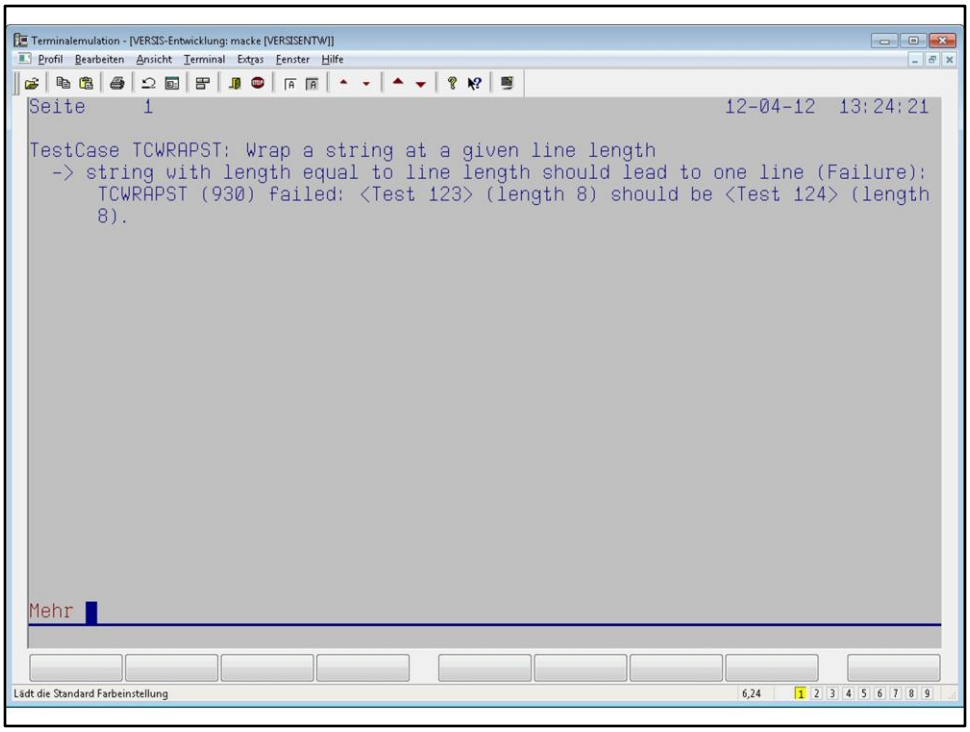

Terminalemulation - [VERSIS-Entwicklung: macke [VERSISENTW]]
Profil Bearbeiten Ansicht Terminal Extras Fenster Hilfe
Current TestCase: TCWRAPST Last Error TestCase:
TestCases..: 1
Failures...:
Errors.....:
Tests....: 15
Failures.:
Errors...:
Asserts..: 62
Start....: 13:23:00
End.....: 13:23:00
Tests....: 15
Failures.:
Errors...:
Lädt die Standard Farbeinstellung
70,4


Terminalemulation - [VERSIS-Entwicklung: macke [VERSISENTW]]
Profil Bearbeiten Ansicht Terminal Extras Fenster Hilfe
Current TestCase: TCWRAPST Last Error TestCase:
TestCases..: 1 Tests....: 15
Failures...: Failures.:
Errors.....: Errors...:
Asserts..: 62
Start....: 13:23:00
End.....: 13:23:00
Asserts..: 62
Start....: 13:23:00
End.....: 13:23:00
Lädt die Standard Farbeinstellung
70,4

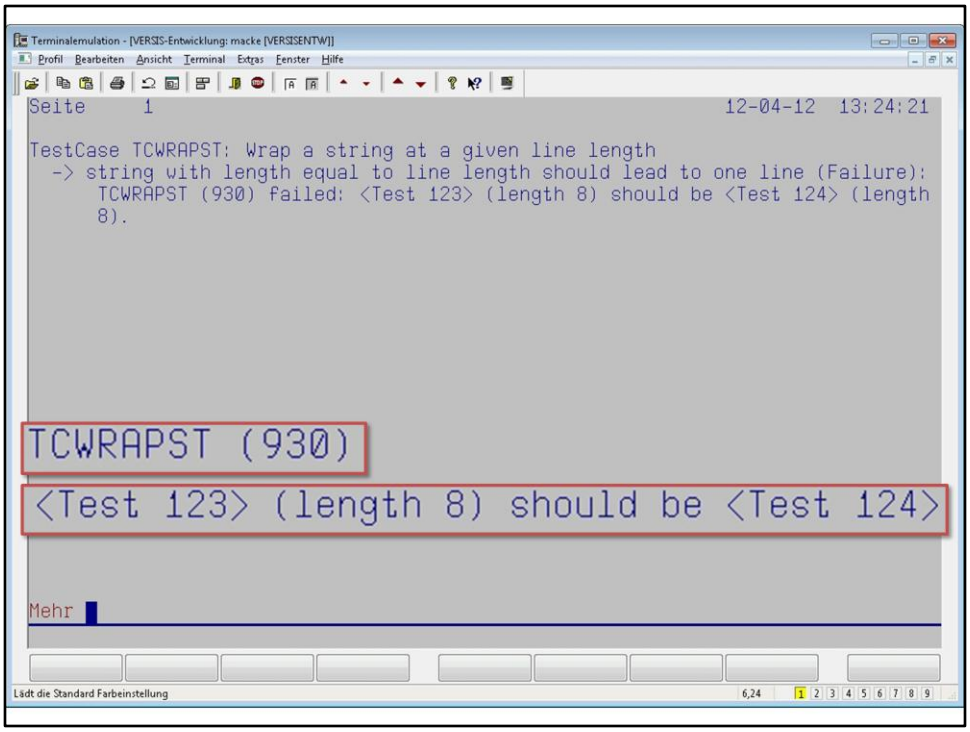


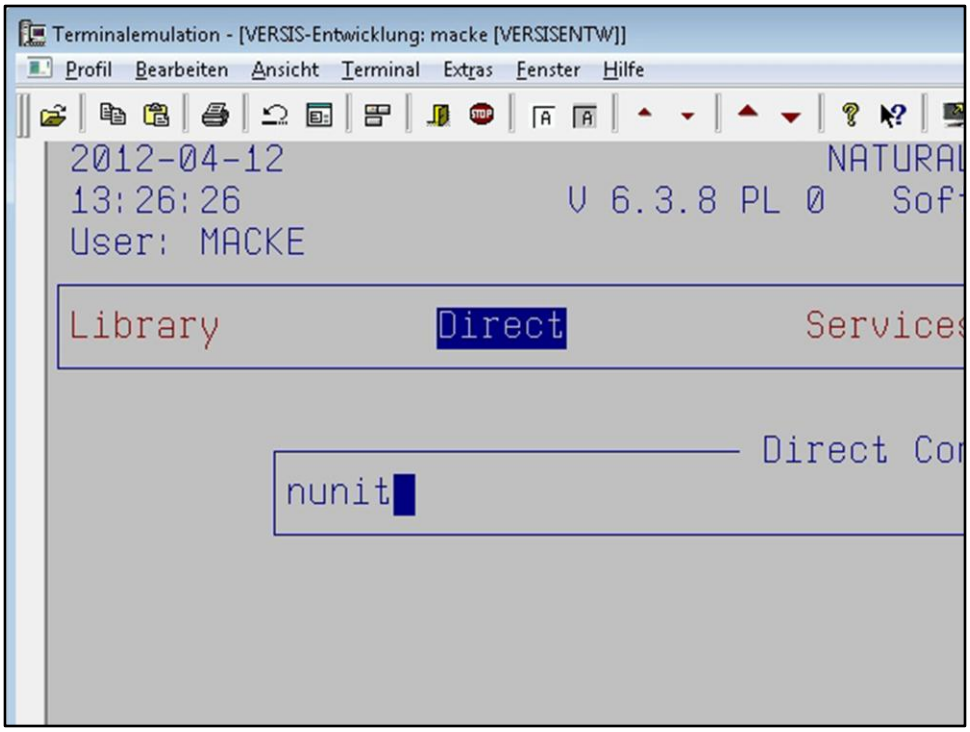
Ein fehlgeschlagener Test wird als F dargestellt, ein Natural-Fehler als E.


Terminalemulation - [VERSIS-Entwicklung: macke [VERSISENTW]]
Profil Bearbeiten Ansicht Terminal Extras Fenster Hilfe
Current TestCase: TCWRAPST Last Error TestCase: TCWRAPST
TestCases..: 1 Tests....: 15 Asserts..: 62
Failures...: 1 Failures..: 1 Start....: 13:23:57
Errors.....: Errors...: End.....: 13:23:57
F
TestCases..: 1
Failures...: 1
Errors.....:
F
Lädt die Standard Farbeinstellung
70,4

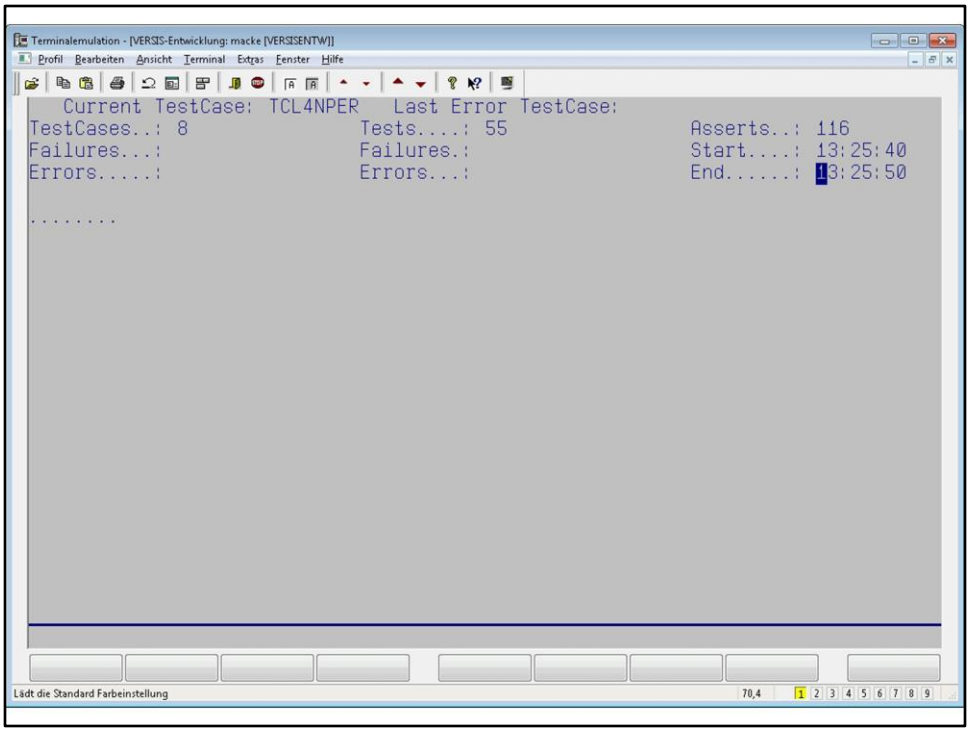


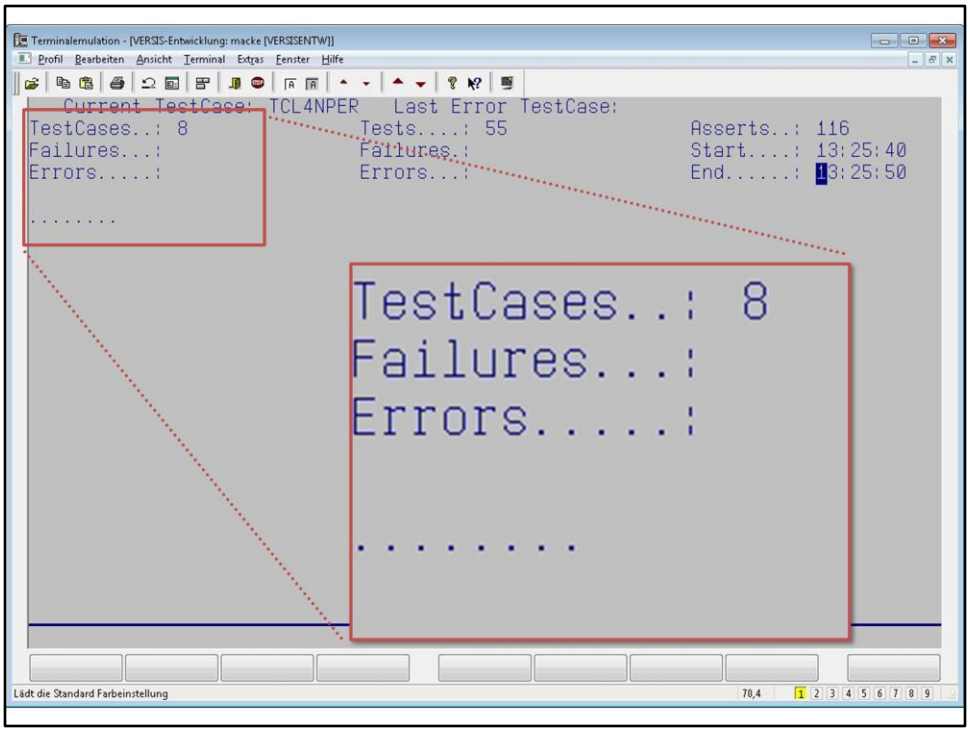
Im Fehlerfall wird eine detaillierte Meldung ausgegeben.


Terminalemulation - [VERSIS-Entwicklung: macke [VERSISENTW]]
Profil Bearbeiten Ansicht Terminal Extras Fenster Hilfe
Seite 1 12-04-12 13:24:21
TestCase TCWRAPST: Wrap a string at a given line length
-> string with length equal to line length should lead to one line (Failure):
TCWRAPST (930) failed: <Test 123> (length 8) should be <Test 124> (length
8).
TCWRAPST (930)
<Test 123> (length 8) should be <Test 124>
Mehr
Lädt die Standard Farbeinstellung
6,24

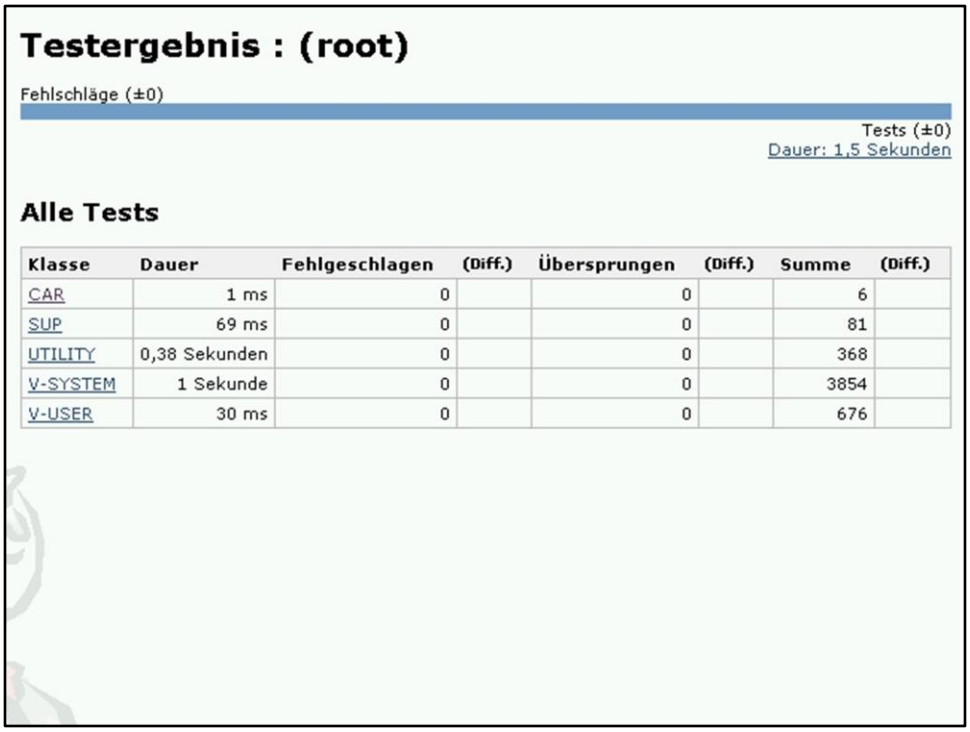


Mit NUNIT können mehrere TestCases hintereinander gestartet werden.


Terminalemulation - [VERSIS-Entwicklung: macke [VERSISENTW]]
Profil Bearbeiten Ansicht Terminal Extras Fenster Hilfe
Current TestCase: TCL4NPER Last Error TestCase:
TestCases..: 8 Tests....: 55 Asserts..: 116
Failures...: Failures.: Start....: 13:25:40
Errors.....: Errors...: End.....: 13:25:50
.....
Lädt die Standard Farbeinstellung
70,4
1 2 3 4 5 6 7 8 9


Terminalemulation - [VERSIS-Entwicklung: macke [VERSISENTW]]
Profil Bearbeiten Ansicht Terminal Extras Fenster Hilfe
Current TestCase: TCL4NPER Last Error TestCase:
TestCases..: 8
Failures...:
Errors.....:
.....
Tests....: 55
Failures..:
Errors...:
Asserts..: 116
Start....: 13:25:40
End.....: 13:25:50
TestCases..: 8
Failures...:
Errors.....:
.....
Lädt die Standard Farbeinstellung
70,4
1 2 3 4 5 6 7 8 9

Die NatUnit-Ergebnisse können als XML exportiert und in Hudson/Jenkins angezeigt werden.



Testergebnis : (root)

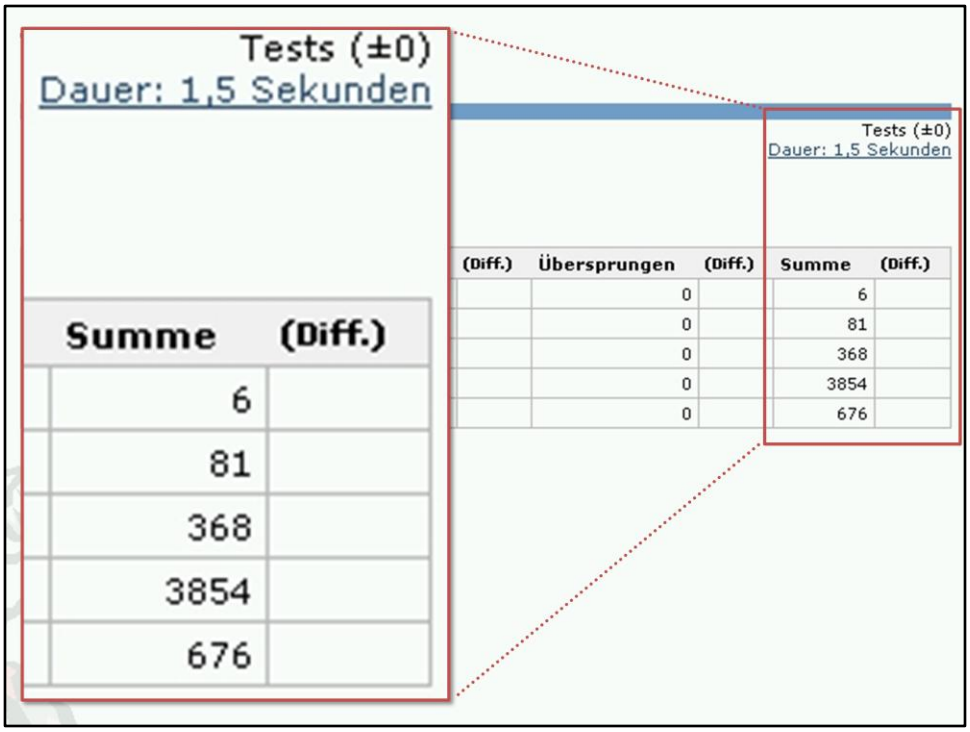
Fehlschläge (±0)

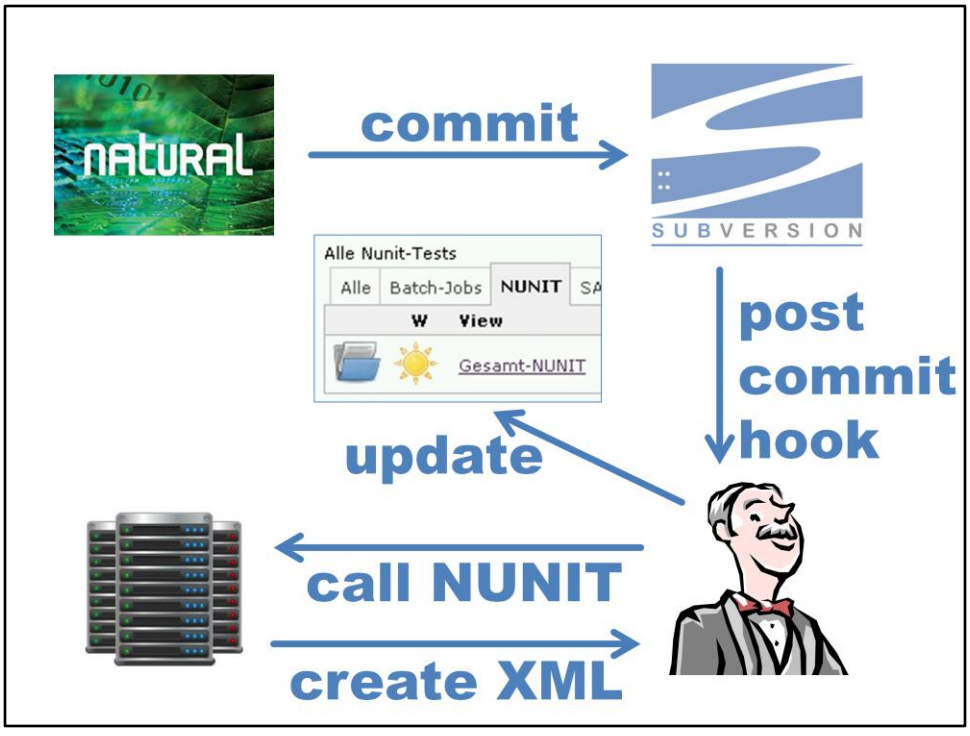
Tests (±0)
Dauer: 1,5 Sekunden

Alle Tests

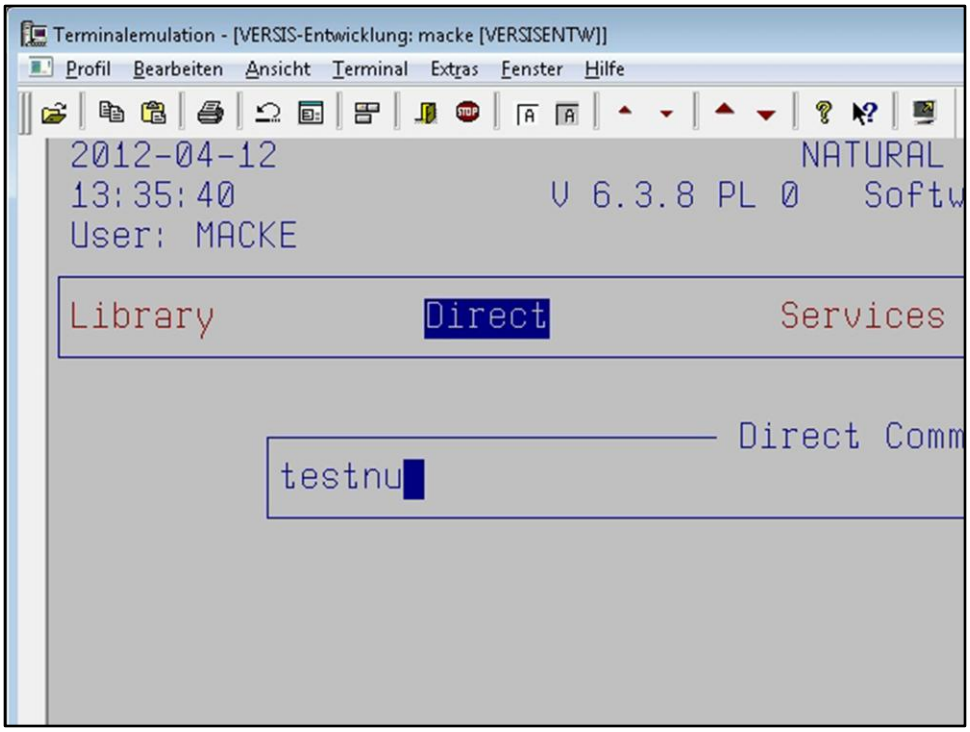
Klasse	Dauer	Fehlgeschlagen	(Diff.)	Übersprungen	(Diff.)	Summe	(Diff.)
CAR	1 ms	0		0		6	
SUP	69 ms	0		0		81	
UTILITY	0,38 Sekunden	0		0		368	
V-SYSTEM	1 Sekunde	0		0		3854	
V-USER	30 ms	0		0		676	

Wie man sieht, sind die Tests ziemlich schnell, was auch notwendig ist, um sie so oft wie möglich laufen zu lassen.

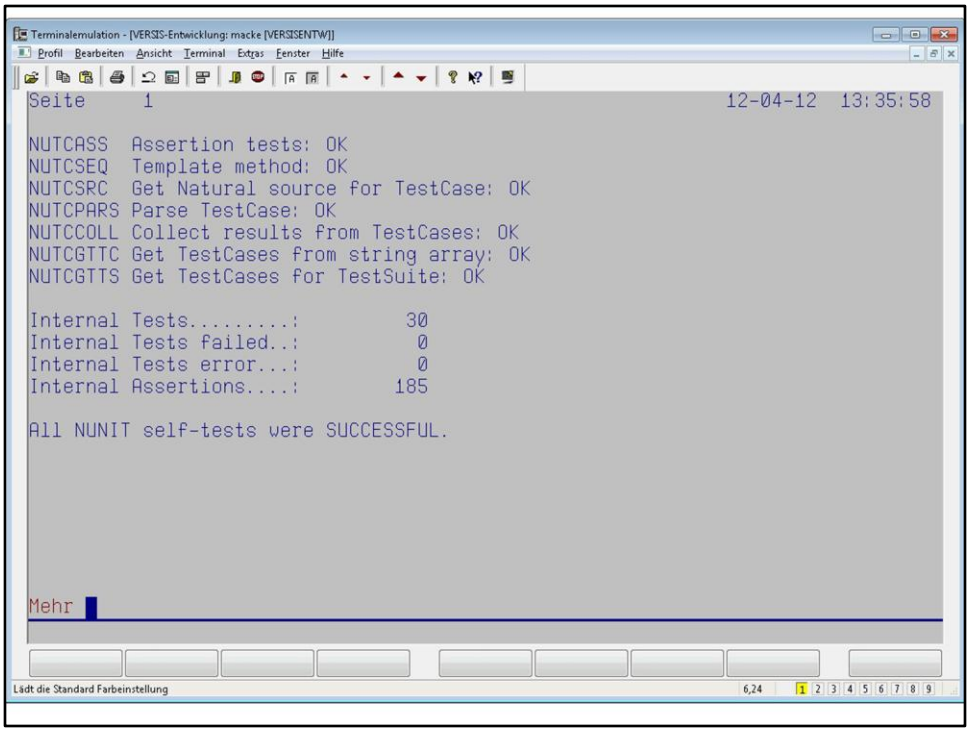

Tests (±0)
Dauer: 1,5 Sekunden
Summe (Diff.)
6
81
368
3854
676
Tests (±0)
Dauer: 1,5 Sekunden
(Diff.) Übersprungen (Diff.) Summe (Diff.)
0 6
0 81
0 368
0 3854
0 676

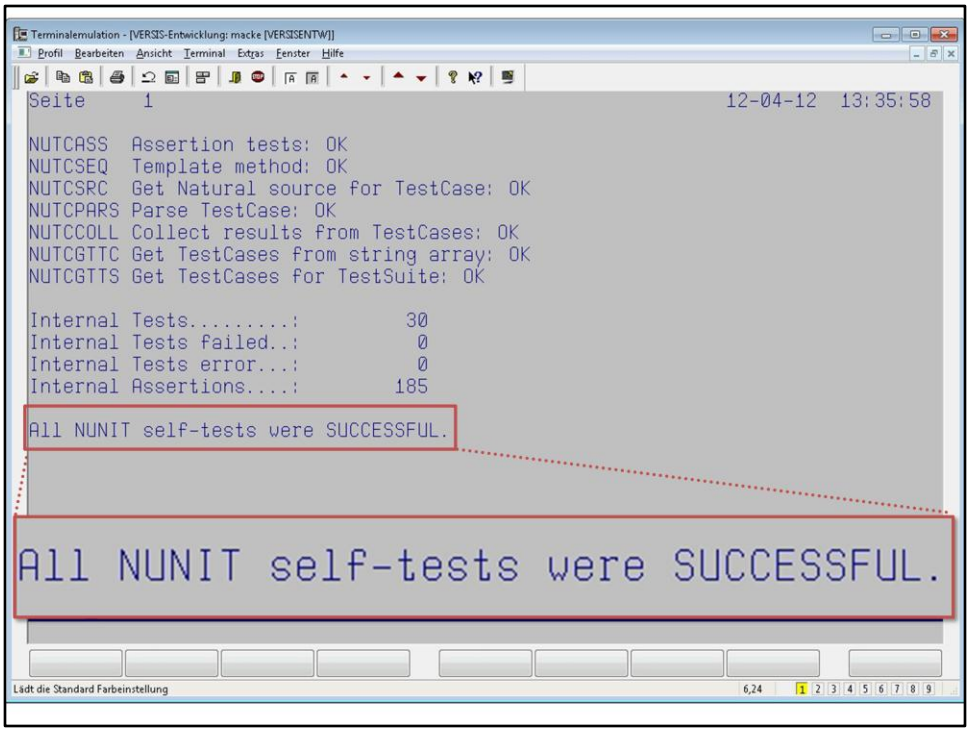


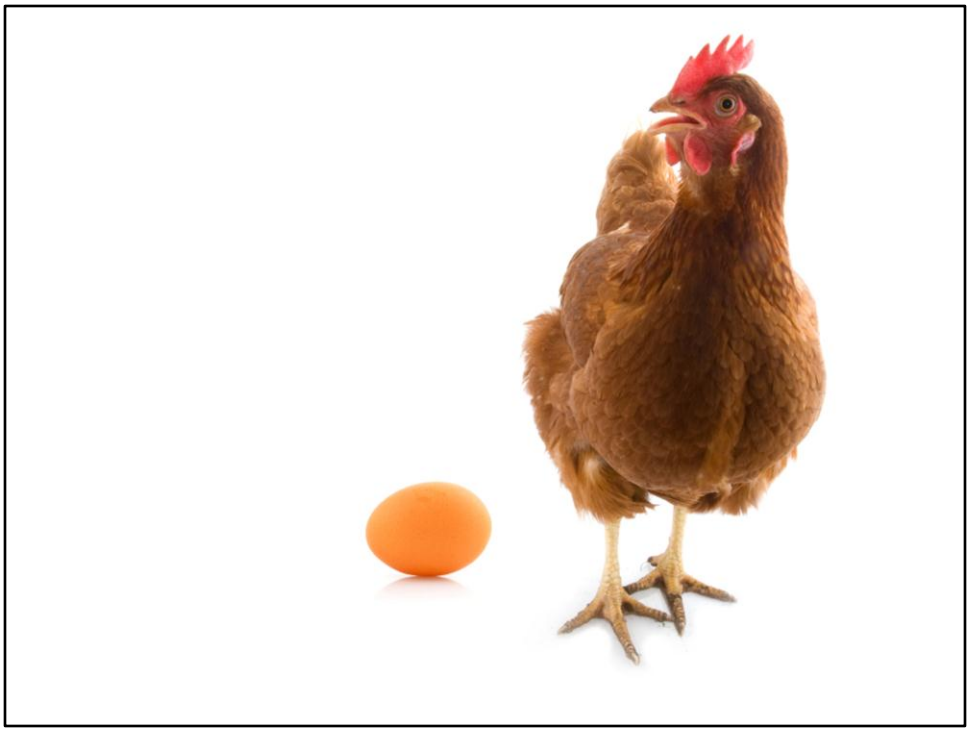
Integration zwischen NatUnit/SVN/Hudson



NatUnit wurde mittels TDD entwickelt und bringt daher eine Reihe interner Testfälle mit.


Terminalemulation - [VERSIS-Entwicklung: macke [VERSISENTW]]
Profil Bearbeiten Ansicht Terminal Extras Fenster Hilfe
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
Lädt die Standard Farbeinstellung
6,24
1 2 3 4 5 6 7 8 9


Terminalemulation - [VERSIS-Entwicklung: macke [VERSISENTW]]
Profil Bearbeiten Ansicht Terminal Extras Fenster Hilfe
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.
All NUNIT self-tests were SUCCESSFUL.
Lädt die Standard Farbeinstellung
6,24



Wie kann man ein Testframework testgetrieben entwickeln, sodass es sich selbst testet?

Die Antwort gibt es in meiner Masterarbeit. Damals hieß NatUnit noch NUNIT.

Jetzt noch ein paar Tipps für sinnvolle Unit-Tests.

...sind

schnell

und

einfach

ausführbar

Schnell → Millisekunden
Nach jeder Änderung ausführen

Infrastruktur darf nicht in Tests benutzt werden, da diese sonst fehleranfällig und langsam werden.

Das hört sich gut an, wenn man mit einem neuen Projekt startet.

Aber die meisten Entwickler arbeiten wohl eher in 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
```

Beispiel

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

Dieses Programm kann nicht getestet werden, da DB-Zugriff, Logik und Ausgabe hart verdrahtet sind.
Außerdem muss es ein Subprogram oder eine Subroutine sein und Parameter haben, damit es überhaupt aus einem Test heraus aufgerufen werden kann.

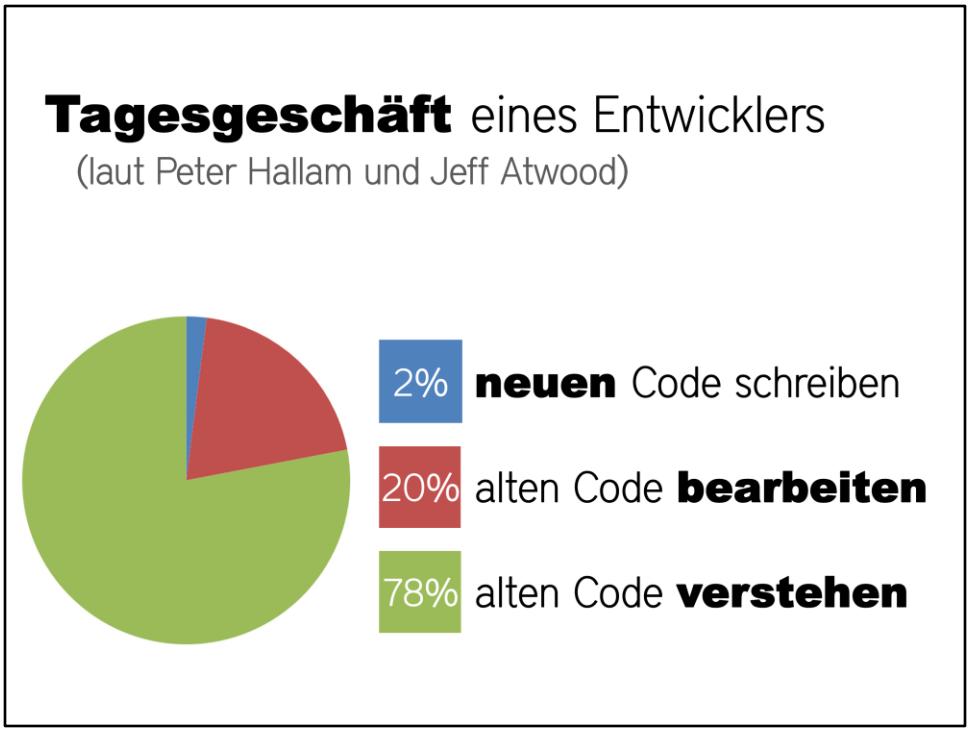
Single
Responsibility
Principle

→ Single Responsibility Principle wird verletzt

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

→ Geschäftslogik extrahieren und aufrufbar machen

Das bedeutet harte Arbeit und ist in vielen Fällen sogar fast unmöglich.

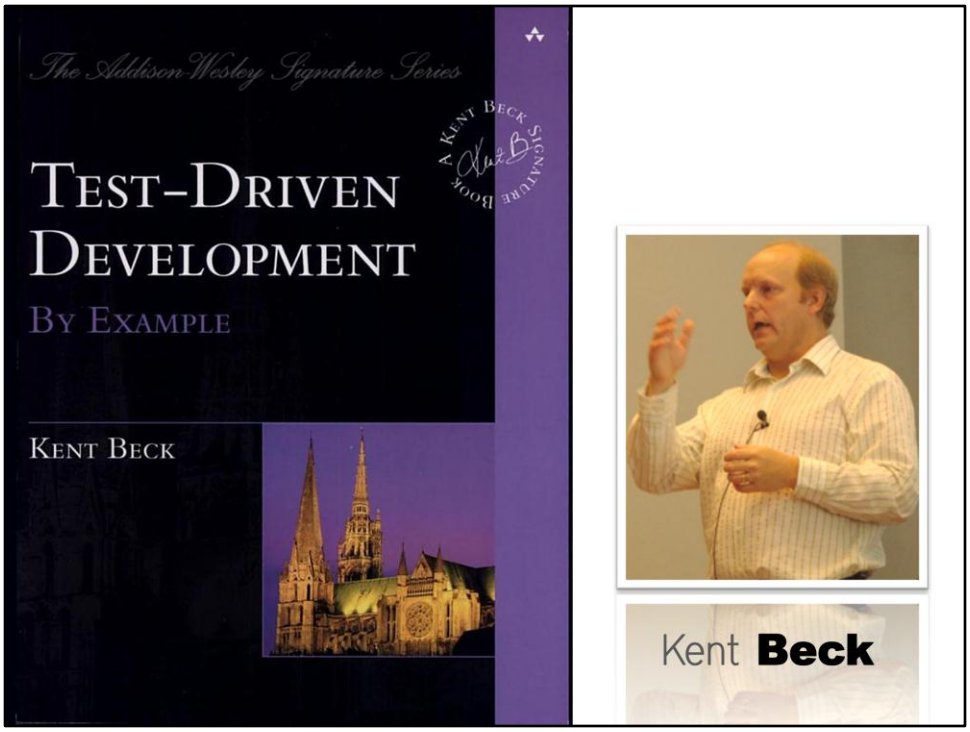


Die Arbeit zahlt sich allerdings aus, da der modulare Code in Verbindung mit seinen Tests sich quasi selbst dokumentiert. Und das ist wichtig, wenn man sich anschaut, was Entwickler eigentlich so machen.

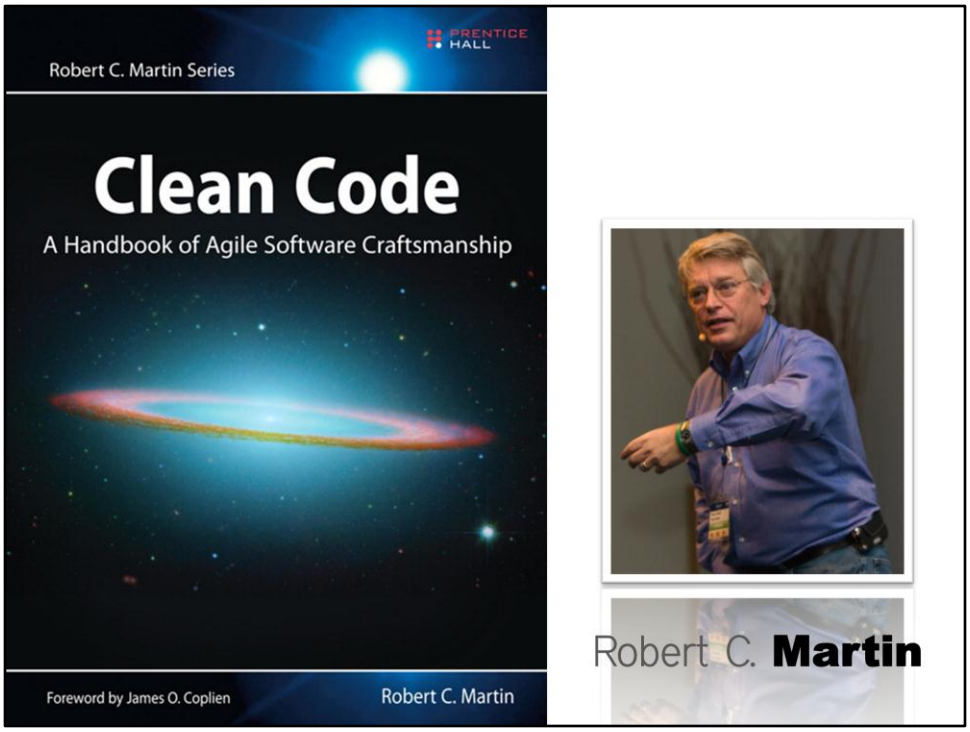
http://blogs.msdn.com/peterhal/archive/2006/01/04/509302.aspx
http://www.codinghorror.com/blog/archives/000684.html

Sobald man einmal mit Unit-Tests begonnen hat, will man nicht mehr zurück zur alten Vorgehensweise.

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



http://www.amazon.de/gp/product/0321146530/ref=as_li_ss_tl?ie=UTF8&camp=1638&creative=19454&creativeASIN=0321146530&linkCode=as2&tag=blovonstemac-21



http://www.amazon.de/gp/product/0132350882/ref=as_li_ss_tl?ie=UTF8&camp=1638&creative=19454&creativeASIN=0132350882&linkCode=as2&tag=blovonstemac-21


f.macke.it/NatUnit


THE
END

Bild**nachweise**

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

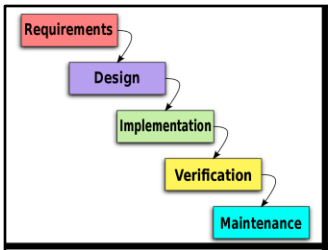
www.flickr.com/photos/pedrosimoes7/190673196/

416style

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

Christy C

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

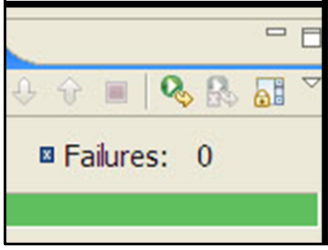


Paul Smith

en.wikipedia.org/wiki/File:Waterfall_model_%281%29.svg

sanja gjenero

www.sxc.hu/photo/1186300

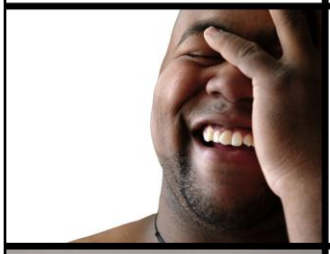


Ben Hermann

www.flickr.com/photos/theredroom/121963809/

	**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/3336353424/

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

Tamlyn Rhodes
www.sxc.hu/photo/499571

Adriano Goncalves
www.sxc.hu/photo/1389679/

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/krikit/2880756271/

Bliz

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

Improve IT
www.flickr.com/photos/improveit/1573962517/

Chris Hedgate
www.flickr.com/photos/chrishedgate/3048835850/

Michal Zacharzewski
www.sxc.hu/photo/431263

Michele

www.flickr.com/photos/damaradeaella/2822846819/

sanja gjenero

www.sxc.hu/photo/779191

NatUnit

f.macke.it/NatUnit

STEFAN MACKE
ALTE OLDENBURGER
Krankenversicherung AG