



andrena

OBJECTS

# Testgetriebene Entwicklung mit ABAP

XP Days 2007  
Karlsruhe

Achim Bangert  
Achim.Bangert@andrena.de  
www.andrena.de

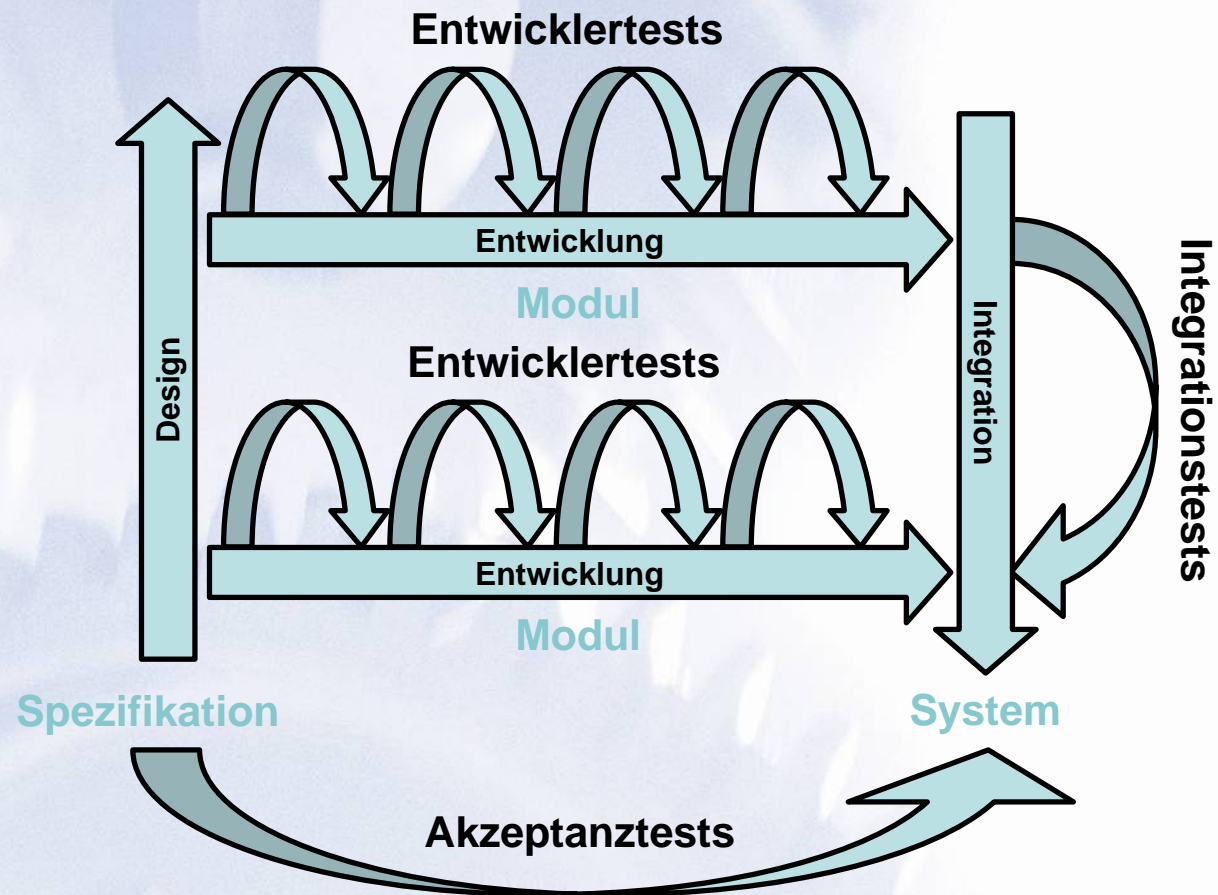
# Agenda

- **Testgetriebene Entwicklung**
- **ABAP Unit**
- **Testausführung über Code Inspector**
- **Dummy- und Mock-Objekte**
- **Weiterführende Themen (Datenbankzugriff, ...)**

## Testen ist integraler Bestandteil jedes Software-Entwicklungsprozesses

- Entwicklertests
- Integrationstests
- Akzeptanztests

# Testgranularitäten



# Entwicklertests (1)

## Der „herkömmliche“ Entwicklertest

- ABAP Workbench › Test
- Debugging
- Testreport

# Entwicklertests (2)

The screenshot shows the SAP Class Builder interface for class ZCL\_TDD\_ACCOUNT. The 'Test' menu is open, showing options: 'Test' and 'Unit Test'. The class interface is displayed with the 'Methods' tab selected. The methods table is as follows:

Method	Level	Visi...	M...	Description
CONSTRUCTOR	Insta...	Pub...	☑	CONSTRUCTOR
GET_CUSTOMER	Insta...	Pub...		Returns the customer
				Returns the account's balance
				Makes a deposit (only positive amounts are allowed)

### Kennzeichen „herkömmlicher“ Entwicklertests

- Ad hoc
- Manuell

### Nachteile „herkömmlicher“ Entwicklertests

- Zeitaufwendig
- Unsystematisch
- Nicht einfach reproduzierbar
- Macht keinen Spaß

# Testgetriebene Entwicklung

**Aber: Testen ist unerlässlich für die Entwicklung qualitativ hochwertiger Softwaresysteme!**

**Grundidee der testgetriebenen Software-Entwicklung:**

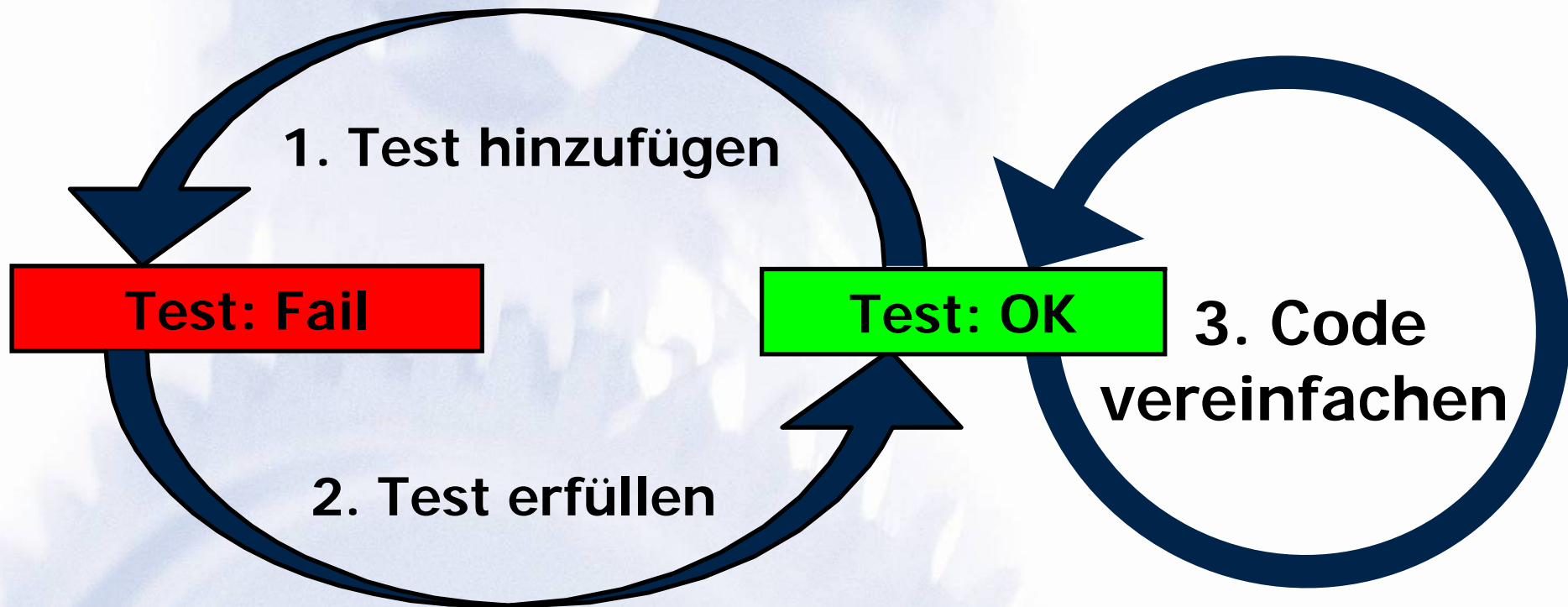
**Wenn Testen so wichtig ist, warum (automatisierte) Tests nicht an den Anfang statt ans Ende der Entwicklungsaktivitäten setzen?**

# Testgetriebene Entwicklung auf Entwicklerebene

## Systematisches Unit-Testing statt „zufälliger“ Entwicklertests

- **Test-First:** Jede verhaltensändernde Codemodifikation wird mit einem automatisierten Test motiviert
- **Refactoring:** Die Struktur des geschriebenen Codes wird kontinuierlich verbessert

# Test/Code/Refactor-Zyklus (1)



## Test/Code/Refactor-Zyklus (2)

- **Grün-Rot**: Schreibe einen Test, der zunächst fehlschlagen sollte; schreibe gerade soviel Implementierungscode, dass der Test aktiviert werden kann
- **Rot-Grün**: Schreibe gerade soviel Implementierungscode, dass alle Tests laufen
- **Grün-Grün**: Verbessere die Struktur des Codes, eliminiere Duplikation und andere Code-Smells

# Vorteile von Unit-Tests

## Unit-Tests

- geben uns konkretes, quantifizierbares Feedback
- erzwingen eine testbare und damit wartbare Codebasis
- ermöglichen sichere Änderungen
- sichern den Erhalt der vorhandenen Funktionalität

# Eigenschaften guter Unit-Tests

## Gute Unit-Tests

- sind schnell
- sind knapp und übersichtlich gehalten
- sind frei von Seiteneffekten (beliebig oft in beliebiger Reihenfolge ausführbar)
- spezifizieren das Verhalten einer überschaubaren Codestrecke
- testen die öffentliche Schnittstelle eines Objekts
- lokalisieren im Fehlerfall zielgenau das Problem

# Testgetriebene Entwicklung auf Projektebene

**Unit-Tests können funktionale Tests auf Systemebene nicht ersetzen!**

**Aber: Es gibt Ansätze, die Vorteile des testgetriebenen Ansatzes auch auf Akzeptanztests auszudehnen**

- Spezifikation in Form potentiell ausführbarer Testfälle
- FIT (derzeit nur Java, aber...)
- eCATT (?)

# ABAP Unit – Testframework für ABAP

- Framework zum Schreiben und Ausführen automatisierter Unit-Tests in ABAP
- Inspiriert von der xUnit-Familie von Testframeworks
- Teil des ABAP-Stacks seit Web AS 6.40
- Komplett integriert in die ABAP Workbench
- Dynamische Weiterentwicklung, neue Features in nahezu jedem Release seither

# Generierung der Testklasse (1)

The screenshot shows the SAP Class Builder interface. The 'Class Builder' menu is open, and the 'Test Class Generation' option is highlighted. The interface includes a menu bar (Class, Edit, Goto, Utilities, Environment, System, Help), a toolbar, and a main workspace. The workspace displays the class hierarchy for 'ZTDD\_ACCOUNT' and 'ZCL\_TDD\_ACCOUNT'. The 'Methods' tab is selected, showing a table of methods.

Level	Visi...	M...	Description
Insta...	Pub...	☒	CONSTRUCTOR
Insta...	Pub...		Returns the customer
Insta...	Pub...		Returns the account's balar
Insta...	Pub...		Makes a deposit (only posit

## Generierung der Testklasse (2)

Method selection for test class generation

Local Test Class

Class Name

Options

Generate Method Calls

Generate 'Assert Equal'

Validity Period Category

Risk Level

Method
<input checked="" type="checkbox"/> DEPOSIT
<input type="checkbox"/> GET_BALANCE
<input type="checkbox"/> GET_CUSTOMER

# Anatomie einer Testklasse (1)

```
CLASS Abap_Unit_Testclass DEFINITION FOR TESTING "#AU Duration Medium
  "#AU Risk_Level Harmless
.

PRIVATE SECTION.
  DATA: m_ref TYPE REF TO Zcl_Tdd_Account.           "#EC NOTEXT

  METHODS: Setup.
  METHODS: Teardown.
  METHODS: Deposit FOR TESTING.
  METHODS: Get_Balance FOR TESTING.
  METHODS: Get_Customer FOR TESTING.
ENDCLASS.           "Abap_Unit_Testclass

CLASS Abap_Unit_Testclass IMPLEMENTATION.

METHOD Setup.
  DATA Iv_Customer TYPE String.

  CREATE OBJECT m_ref
  EXPORTING
    IV_CUSTOMER = Iv_Customer.
ENDMETHOD.           "Setup

...
```

## Zusatz FOR TESTING

Mit dem Zusatz `FOR TESTING` wird eine Klasse für ABAP Unit als Testklasse ausgezeichnet

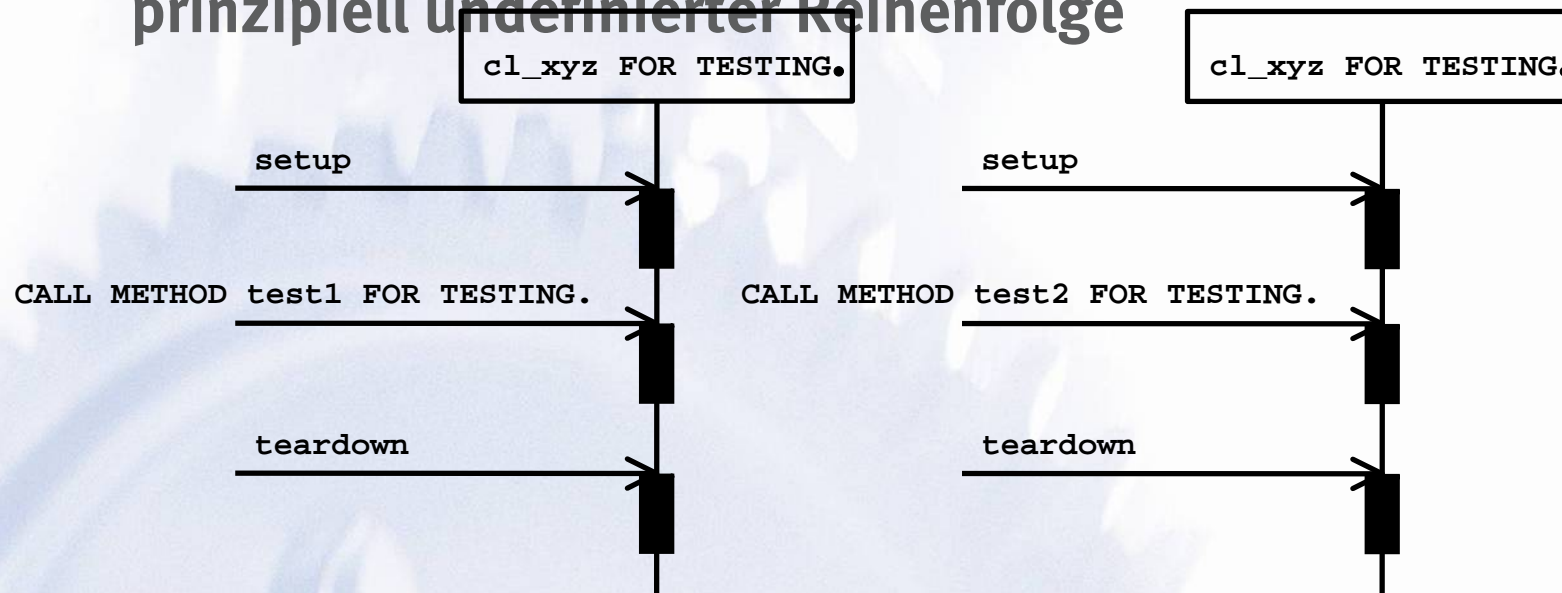
- Kann Testmethoden `FOR TESTING` enthalten, die im Testlauf aufgerufen werden
- Kann die Instanzmethoden `setup` und `teardown` enthalten
- Kann die statischen Methoden `class_setup` und `class_teardown` enthalten

**Testkonfiguration, die eindeutiges Testverhalten garantiert**

- Kann in `setup` vor jedem Test aufgebaut werden
- Kann in `teardown` nach jedem Test abgebaut werden
- Kann in `class_setup` einmalig vor allen Tests aufgebaut werden
- Kann in `class_teardown` einmalig nach allen Tests abgebaut werden
- Klassenattribute halten Testobjekte, die von mehreren Testfällen benötigt werden

# Lebenszyklus eines Testfalls

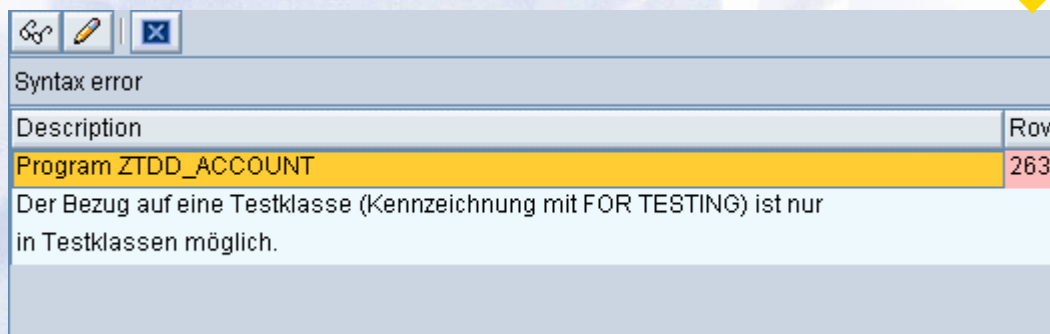
- Pro Testfall wird eine eigene Instanz der Testklasse erzeugt
- Die Testfälle sind voneinander unabhängig, laufen in prinzipiell undefinierter Reihenfolge



# Test- vs. Produktivcode

Eine Testklasse gehört nicht zum produktiven Coding und wird in Produktiv-Systemen nicht generiert

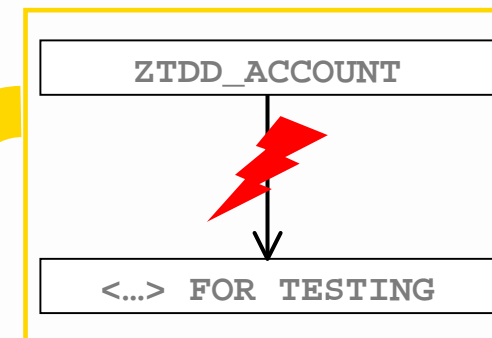
- Produktives Coding darf nicht von Testcode abhängen
- Wird vom System überprüft



Syntax error

Description	Row
Program ZTDD_ACCOUNT	263

Der Bezug auf eine Testklasse (Kennzeichnung mit FOR TESTING) ist nur in Testklassen möglich.



## Anatomie einer Testklasse (2)

```
CLASS Abap_Unit_Testclass DEFINITION FOR TESTING "#AU Duration Medium
  #AU Risk_Level Harmless
.

PRIVATE SECTION.
  DATA: m_ref TYPE REF TO Zcl_Tdd_Account.           "#EC NOTEXT

  METHODS: Setup.
  METHODS: Teardown.
  METHODS: Deposit FOR TESTING.
  METHODS: Get_Balance FOR TESTING.
  METHODS: Get_Customer FOR TESTING.
ENDCLASS.           "Abap_Unit_Testclass

CLASS Abap_Unit_Testclass IMPLEMENTATION.

METHOD Setup.
  DATA Iv_Customer TYPE String.

  CREATE OBJECT m_ref
  EXPORTING
    IV_CUSTOMER = Iv_Customer.
ENDMETHOD.           "Setup

...
```


# Testeigenschaften

**Seit Release 7.00 können Testeigenschaften über Pseudokommentare festgelegt werden**

- `"AU Risk_Level Critical|Dangerous|Harmless`
- `"AU Duration Short|Medium|Long`
- **Werden an abgeleitete Testklassen vererbt**
- **Steuern Systemverhalten bei der Testausführung**
- **Transaktion SAUNIT\_CLIENT\_SETUP**

# Transaktion SAUNIT\_CLIENT\_SETUP

## Grenzwerte für die Testausführung

Basic Settings	
<b>Client</b>	
<input type="checkbox"/> Prohibit Unit Tests	
Limit of Risk Level:	Critical 
Last Changed By	BCUSER
<b>Application Server</b>	
Long Duration	720
Medium Duration	120
Short Duration	12
Last Changed By	BCUSER

## Anatomie einer Testklasse (3)

```
METHOD deposit.  
  DATA lv_balance TYPE zcl_tdd_account=>t_amount.  
  DATA lv_amount TYPE zcl_tdd_account=>t_amount.  
  
  lv_amount = '100.0'.  
  m_ref->deposit( lv_amount ).  
  lv_balance = m_ref->get_balance( ).  
  cl_aunit_assert=>assert_equals(  
    exp = lv_amount  
    act = lv_balance ).  
  lv_amount = '50.0'.  
  m_ref->deposit( lv_amount ).  
  lv_balance = m_ref->get_balance( ).  
  lv_amount = '150.0'.  
  cl_aunit_assert=>assert_equals(  
    exp = lv_amount  
    act = lv_balance ).  
ENDMETHOD.
```

## Systemklasse `CL_AUNIT_ASSERT`

- Stellt statische Methoden bereit, um die erwarteten mit den tatsächlich gelieferten Resultaten zu vergleichen
- `ASSERT_EQUALS`
- `ASSERT_DIFFERS`
- `ASSERT_INITIAL`
- `ASSERT_BOUND`
- `ASSERT_NOT_INITIAL`
- `ASSERT_NOT_BOUND`
- `FAIL`

# Testausführung (1)

The screenshot shows the SAP ABAP IDE interface. On the left, the Repository Browser displays the package structure: ZTDD\_ACCOUNT > Class Library > Classes > ZCL\_TDD\_ACCOUNT\_TEST (highlighted). A context menu is open over this class, with the 'Test' option selected. A sub-menu is also open, showing 'Test' and 'Unit Test' (highlighted). On the right, the Class Interface for ZCL\_TDD\_ACCOUNT\_TEST is visible, showing a table of methods.

Method	Level	Visi...	M...	Description
CONSTRUCTOR	Insta...	Pub...	☰	CONSTRUCTOR
GET_CUSTOMER	Insta...	Pub...		Returns the custo
GET_BALANCE	Insta...	Pub...		Returns the accou
DEPOSIT	Insta...	Pub...		Makes a deposit (

## Testausführung (2)

 Unit tests processed successfully; 1 programs, 1 classes, 1 methods

## Neue Anforderung: Auszahlung

- Saldo wird um Auszahlungsbetrag vermindert
- Nur positive Auszahlungsbeträge sind zulässig
- Nur vom Saldo gedeckte Beträge dürfen abgehoben werden

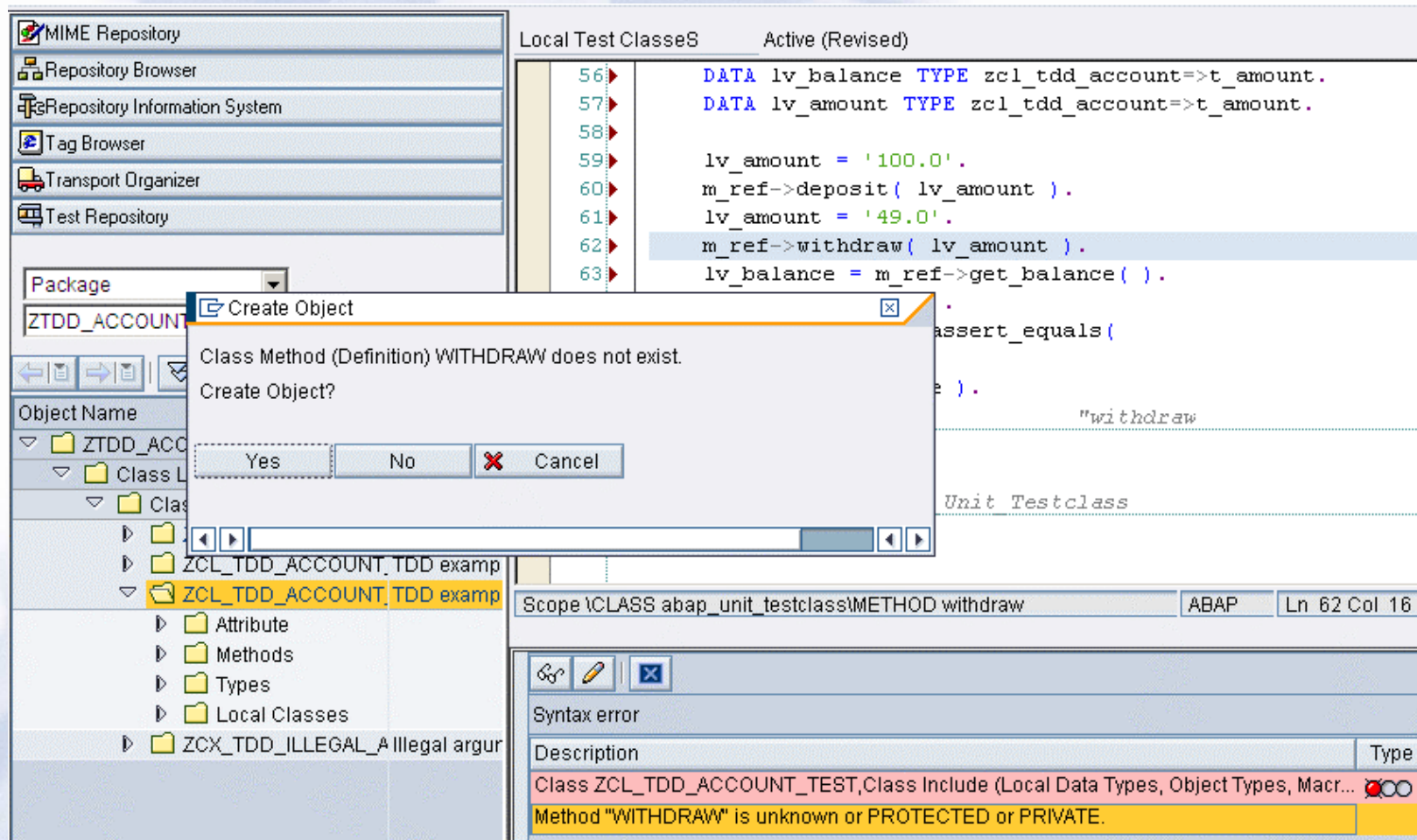
# TDD am Beispiel: Erster Testfall

```
CLASS abap_unit_testclass DEFINITION FOR TESTING "#AU Duration Medium
  "#AU Risk_Level Harmless
.
...
  METHODS withdraw FOR TESTING
    RAISING cx_static_check.
ENDCLASS.
...

METHOD withdraw.
  DATA lv_balance TYPE zcl_tdd_account=>t_amount.
  DATA lv_amount TYPE zcl_tdd_account=>t_amount.

  lv_amount = '100.0'.
  m_ref->deposit( lv_amount ).
  lv_amount = '49.0'.
  m_ref->withdraw( lv_amount ).
  lv_balance = m_ref->get_balance( ).
  lv_amount = '51.0'.
  cl_aunit_assert=>assert_equals(
    exp = lv_amount
    act = lv_balance ).
ENDMETHOD.
```

# TDD am Beispiel: Methode anlegen



The screenshot shows the SAP IDE interface. On the left is the Repository Browser with a tree view showing the package structure: ZTDD\_ACCOUNT, Class L, Class, ZCL\_TDD\_ACCOUNT\_TDD examp, ZCL\_TDD\_ACCOUNT\_TDD examp, Attribute, Methods, Types, Local Classes, and ZCX\_TDD\_ILLEGAL\_A Illegal argur. The main editor displays the code for 'Local Test ClassS Active (Revised)'. The code includes data declarations for 'lv\_balance' and 'lv\_amount', and method calls for 'deposit', 'withdraw', and 'get\_balance'. A dialog box titled 'Create Object' is open, asking 'Class Method (Definition) WITHDRAW does not exist. Create Object?' with 'Yes', 'No', and 'Cancel' buttons. Below the code, a status bar shows 'Scope \CLASS abap\_unit\_testclass\METHOD withdraw' and a 'Syntax error' message: 'Method "WITHDRAW" is unknown or PROTECTED or PRIVATE.'

```
56 DATA lv_balance TYPE zcl_tdd_account=>t_amount.  
57 DATA lv_amount TYPE zcl_tdd_account=>t_amount.  
58  
59 lv_amount = '100.0'.  
60 m_ref->deposit( lv_amount ).  
61 lv_amount = '49.0'.  
62 m_ref->withdraw( lv_amount ).  
63 lv_balance = m_ref->get_balance( ).
```

Class Method (Definition) WITHDRAW does not exist.  
Create Object?  
Yes No Cancel

Scope \CLASS abap\_unit\_testclass\METHOD withdraw ABAP Ln 62 Col 16

Syntax error

Description	Type
Class ZCL_TDD_ACCOUNT_TEST,Class Include (Local Data Types, Object Types, Macr...	000
Method "WITHDRAW" is unknown or PROTECTED or PRIVATE.	

# TDD am Beispiel: Methode deklarieren

Create Method

Class

Method

Description

Attributes Parameters Exceptions

Visibility

Public

Protected

Private

Method

Static

Instance

Abstract

Final

Event handler for

Class/interface

Event

Modeled

# TDD am Beispiel: Testausführung

**ABAP Unit: Result Display**

Task/Program/Class/Method	tatu	Fatal	C
TASK_BCUSER_20071121_2058	❌	0	
ZCL_TDD_ACCOUNT_TEST=	❌	0	
ABAP_UNIT_TESTCLASS	❌	0	
DEPOSIT	✅	0	
WITHDRAW	❌	0	

Type	Message
❗	Critical Assertion Error: 'ASSERT_EQUALS'

Info

Different Values:

Expected	[	51.00	]
Actual	[	100.00	]

Test 'ABAP\_UNIT\_TESTCLASS->WITHDRAW' in Main Program

Stack

In ZCL\_TDD\_ACCOUNT\_TEST=====CCAU (Line: 65)

# TDD am Beispiel: Methode implementieren

Ty.	Parameter	Type spec.	Description
▸	IV_AMOUNT	TYPE T_AMOUNT	Amount type

Method	WITHDRAW	Active
1	<b>METHOD</b> withdraw.	
2	mv_balance = mv_balance - iv_amount.	
3	<b>ENDMETHOD.</b>	

# TDD am Beispiel: Testausführung

 Unit tests processed successfully; 1 programs, 1 classes, 2 methods

**Manuelle Testausführung ist für größere Testsuiten offensichtlich nicht sinnvoll**

- **Ausweg: Code Inspector (Transaktion SCI)**
- **Erlaubt das Festlegen der zu prüfenden Objektmenge**
- **Erlaubt die Definition einer Prüfvariante für Unit-Tests**
- **Erlaubt „wiederkehrende“ oder „ad hoc“ Inspektionen**

# Code Inspector : Ad hoc Inspektion

Object Selection

Object Set  Vers.

Request/Task

Single

Check Variant

Predefined

Temporary Definition

Selection	D...	Att...	Tests
▼ <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	List of Checks
▶ <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	General Checks
▶ <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Performance Checks
▶ <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Security Checks
▶ <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Syntax Check/Generation
▶ <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Programming Conventions
▼ <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Dynamic Tests
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ABAP Unit
▶ <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	User Interfaces
▶ <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Search Funct.
▶ <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Application Checks
▶ <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Internal Performance Tests
▶ <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Intern. Tests

# Code Inspector : Ergebnisanzeige

## Code Inspector: Results

Person Responsible: BCUSER    Inspection:    Version: 1

Messages

	D...	...	Ex...	Tests	Error	Warnin...	Inform...
▼				List of Checks	1	0	2
▼				Dynamic Tests	1	0	2
▼				ABAP Unit	1	0	2
▼				Errors	1	0	0
▼				Message Code Critical	1	0	0
				Program ZTDD_TIC_TAC_TOE Include ZTDD_TIC_T...	1	0	0
				Critical Assertion Error: 'exception expected'			
				==> Critical Assertion Error: 'exception expected'			
▶				Information	0	0	2

# Dummy- und Mock-Objekte: Motivation

Ein „Unit-Test“ soll eine Klasse oder eine Gruppe von Klassen in Isolation testen

Probleme:

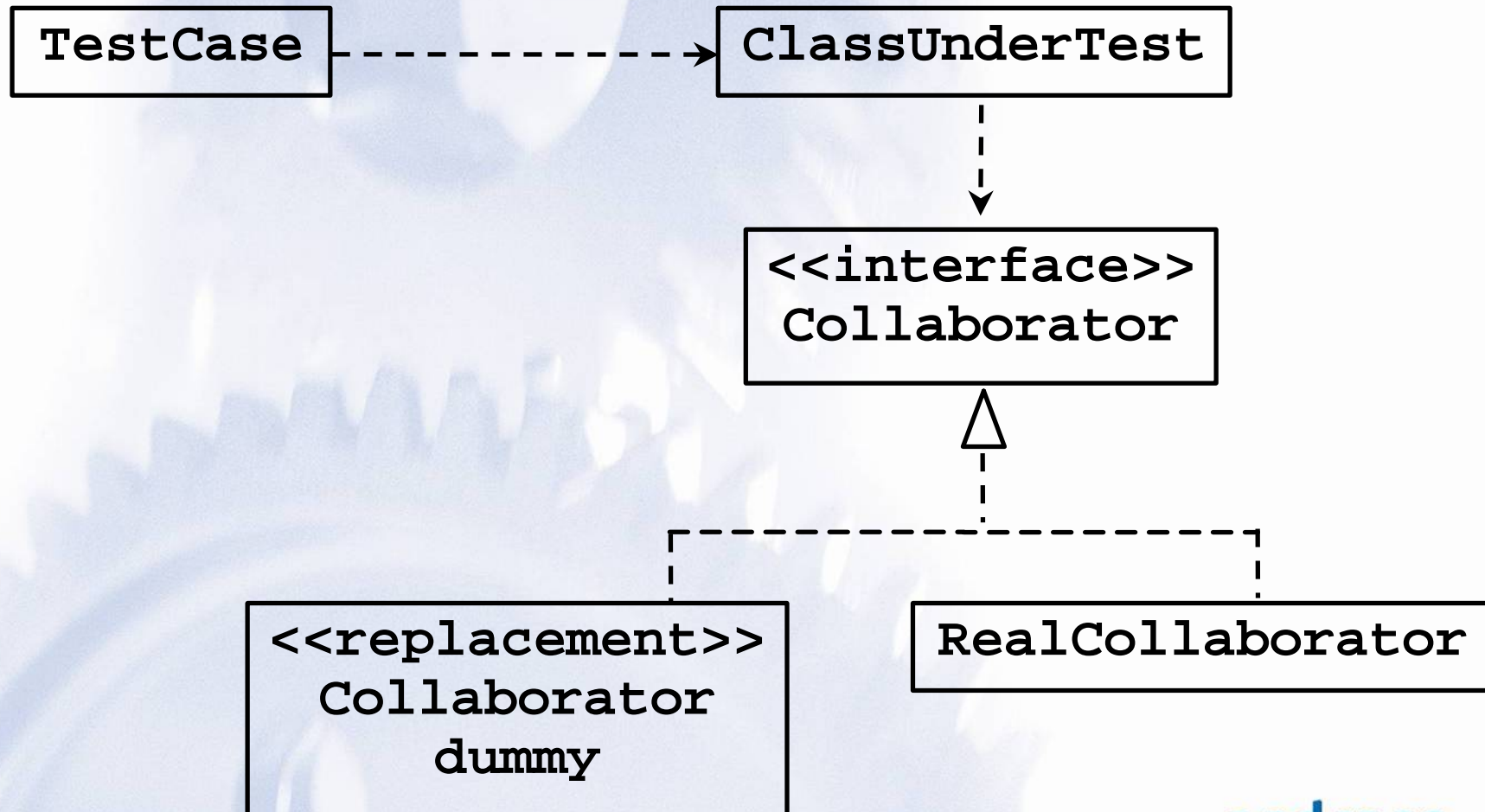
- Objekte arbeiten nicht isoliert
- Aufbau der Testumgebung ist oft aufwändig
- Testen von Ausnahmesituationen ist schwierig
- langsame Tests bei Überschreiten der Systemgrenzen

Für die Dauer der Tests werden Abhängigkeiten des „Object Under Test“ (OUT) zu kollaborierenden Objekten durch die Einführung einfacher „Attrappen“ ersetzt

## Vorteile:

- Tests laufen schnell
- Auftretende Fehler sind leicht zu lokalisieren
- Notwendige Testkombinatorik ist kleiner als beim Testen mehrerer Objekte

# Attrappen im Test



## Dummy-Objekte (Stubs)

Dummy-Objekte sind die einfachsten „Attrapen“

- Ersetzen die vom OUT verwendeten Klassen oder Schnittstellen während des Tests
- Stellen eine einfache Implementierung zur Verfügung, die feste Werte zurückliefert
- Der (korrekte) Aufruf der Methoden wird nicht geprüft

**Mock-Objekte sind Dummy-Objekte mit integrierter Testfunktionalität**

- **Erlauben die dynamische Spezifikation erwarteter Ein- und zugehöriger Ausgabeparameter**
- **Erlauben die Verifikation der Methodenaufrufe durch Vergleich der erwarteten mit den tatsächlichen Parametern**

# Mock-Framework für ABAP

Leider existiert derzeit für ABAP noch kein Framework zur Erzeugung von Mock-Objekten „on the fly“

- Keine prinzipiellen Schwierigkeiten dank guter Unterstützung von dynamischer Programmierung und Code-Generierung in ABAP
- Erste Anläufe wurden bereits unternommen
- Derzeit müssen die benötigten Mock-Objekte noch von Hand erstellt werden, dabei kann ein Mini-Framework hilfreich sein

# Minimales Mock-Framework: Basis

## Basis für ein minimales Mock-Framework bildet eine generische Container-Klasse

```
CLASS zcl_mo_value_fifo DEFINITION.  
  PUBLIC SECTION.  
    METHODS is_empty  
      RETURNING  
        value(rv_empty) TYPE abap_bool.  
    METHODS put  
      IMPORTING  
        iv_value TYPE any.  
    METHODS get  
      EXPORTING  
        ev_value TYPE any  
        er_data TYPE REF TO data  
      RAISING  
        zcx_mo_container_empty.  
    ...  
ENDCLASS.
```

# Hilfsklasse für Eingabeparameter

Instanzen der Container-Klasse werden zur Speicherung erwarteter und tatsächlicher Werte eines Eingabeparameters verwendet

```
CLASS zcl_mo_call_values DEFINITION.  
  
PUBLIC SECTION.  
METHODS add_exp  
IMPORTING  
    iv_value TYPE any.  
METHODS add_act  
IMPORTING  
    iv_value TYPE any.  
METHODS verify.  
...  
ENDCLASS.
```

The diagram shows two green callout boxes. The first callout points from the `add_exp` method to the code `mr_exp_values->put( iv_value )`. The second callout points from the `add_act` method to the code `mr_act_values->put( iv_value )`.

# Hilfsklasse für Ausgabeparameter

Eine Instanz der Container-Klasse wird auch zur Speicherung der Rückgabewerte für einen Ausgabeparameters verwendet

```
CLASS zcl_mo_return_values DEFINITION.  
  PUBLIC SECTION.  
    METHODS get_next  
      EXPORTING  
        ev_value TYPE any.  
    METHODS add  
      IMPORTING  
        iv_value TYPE any.  
    METHODS set_default  
      IMPORTING  
        iv_value TYPE any.  
    METHODS verify.  
  ...  
ENDCLASS.
```

mr\_ret\_values->get ( importing  
ev\_value = ev\_value )

mr\_ret\_values->put( iv\_value )

## Pro Methode und pro Parameter werden im Mock-Objekt öffentliche Attribute vom Typ dieser Hilfsklassen deklariert

```
CLASS zcl_tdd_account_mock DEFINITION
  INHERITING FROM zcl_tdd_account.

  PUBLIC SECTION.
    DATA mr_get_bal_ret TYPE REF TO zcl_mo_return_values.
    DATA mr_deposit_amnt_cal TYPE REF TO zcl_mo_call_values.

    METHODS verify.

    METHODS deposit REDEFINITION.
    METHODS get_balance REDEFINITION.
    ...
ENDCLASS.
```

mr\_deposit\_amnt\_cal->add\_act( iv\_amount )

mr\_get\_bal\_ret->get\_next( importing ev\_value = rv\_balance )

# Mock-Objekt im Test/Test des Mocks

## Exemplarischer Testcode, der ausnahmsweise den Mock selbst testet

```
METHOD deposit.  
  DATA lv_amnt TYPE zcl_tdd_account_mock=>t_amount VALUE '12.34'.  
  DATA lv_bal TYPE zcl_tdd_account_mock=>t_amount.  
  
  m_ref->mr_deposit_amnt_cal->add_exp( lv_amnt ).  
  m_ref->mr_get_bal_ret->add( lv_amnt ).  
  m_ref->deposit( lv_amnt ).  
  lv_bal = m_ref->get_balance( ).  
  cl_aunit_assert=>assert_equals(  
    exp = lv_amnt  
    act = lv_bal ).  
  m_ref->verify( ).  
ENDMETHOD.
```

## Weiterführende Themen: Datenbankzugriffe

Datenbankinteraktionen sind aus TDD-Sicht problematisch

- Schreibzugriffe ändern den Systemzustand, gefährden Unabhängigkeit und Wiederholbarkeit der Tests
- Testdaten in der Datenbank sind global (fragile Tests/Synchronisationsprobleme bei konkurrierender Testausführung)
- Lange Ausführungszeiten z. B. wg. aufwändigem Aufbau des Testfixtures

- **Problem minimieren (Schichtenarchitektur, dedizierte Zugriffsschicht mit Interfaces, Mocks)**
- **Echte Datenbanktests auf das notwendige Minimum beschränken (CRUD)**
- **Mehrere Testmandanten, evtl. pro Entwickler**
- **Jeder Testfall (auch fehlschlagende!) hinterlässt die Datenbank unverändert/in einen definierten Zustand**

**Vielen Dank für Ihre Aufmerksamkeit**

**DISKUSSION**