

# Software Engineering I: Software Technology

WS 2008/09

## ***Model Based Testing***

Bernd Bruegge  
*Chair for Applied Software Engineering  
Technische Universitaet Muenchen*

# Outline of the Lectures on Testing

- ✓ Terminology
- ✓ Types of errors
- ✓ Approaches for dealing with errors
- ✓ Testing activities
- ✓ Unit testing
- ✓ Integration testing
- ✓ System testing
- Model-based Testing

# Generating Tests from a System Model

- There are many different ways to "derive" tests from a system model
  - Manual generation
  - Automatic generation
- Because testing is usually experimental and based on heuristics, there is no one best way to do this.
  - It is common to consolidate all test related decisions into a package that is often known as "test requirements" or "test package".
- This test package can contain e.g. information about the part of the system model that should be the focus for testing, or about the conditions where it is correct to stop testing (test stopping criteria).

# Model Driven Architecture (MDA)

- Recall: MDA focuses on forward engineering, i.e. producing code from a system model
  - Note: The term “architecture” in MDA does not mean the architecture of the system being modeled, but refers the architecture of the various standards and models that serve as the technology basis for MDA
    - UML: Unified Modeling Language
    - MOF: Meta-Object Facility
    - XMI: Metadata Interchange
    - EDOC: Enterprise Distributed Object Computing
    - SPEM: Software Process Engineering Metamodel
    - CWM: Common Warehouse Metamodel

An important objective of MDA is **executable UML**.

# Executable UML

- **Executable UML** means that given a UML system model the following steps can be performed automatically:
  - Code generation
  - Simulation Validation
  - Test generation (“Model based Testing”)

# Model Based Testing

- **Definition Model Based Testing:** All testing is based on a system model which is used to (manually) generate test cases and oracles
- Advantage:
  - Increased effectiveness of testing
  - Decreased costs
  - Reuse of artifacts such as analysis and design models
- Requirement:
  - The system model is formalized (specification model) and described in UML 2.0

# Model Driven Testing

- **Model Driven Testing** is a special case of model based testing:
  - All the test cases are **automatically** generated from the system model
- Advantage:
  - Automatic generation of tests suites
  - Better test thoroughness
  - Reduces the cost of test execution.

# Observations on Model Based Testing

- A system model is an abstract representation of the system's desired behavior
  - The test cases derived from this model are functional tests on the same level of abstraction as the model
  - These test cases are known as the **abstract test suite**
- Because test suites are derived from models - not from source code - model-based testing is a form of **black-box testing**
- The abstract test suite cannot be executed
  - An **executable test suite** must be derived from the abstract test suite which communicates with the system under test (SUT)
  - This is done by mapping the abstract test suite to concrete test cases suitable for execution.
    - Mapping is not part of this lecture.



# U2TP: A UML Profile for Model Based Testing:

- In 2001, the OMG released a Request for Proposals (RFP) for a Testing Profile for UML to support model based testing
- Several companies and institutions (Ericsson, Fraunhofer/FOKUS, IBM, Motorola, Rational, Softeam, Telelogic, University of Lübeck) responded to this RFP, and after some discussion, decided to work together as a consortium to produce U2TP
- U2TP: UML 2 based Test Profile
- Current version: 1.0

# U2TP

- UML-2 Testing Profile (U2TP):
  - A UML extension that makes UML applicable to software testing.
  - U2TP allows the collection of all information required for the model-based testing process
    - Deriving a test model from the system model.
- Test Model:
  - A set of test cases derived from the system model.

# Overview of Concepts used in U2TP

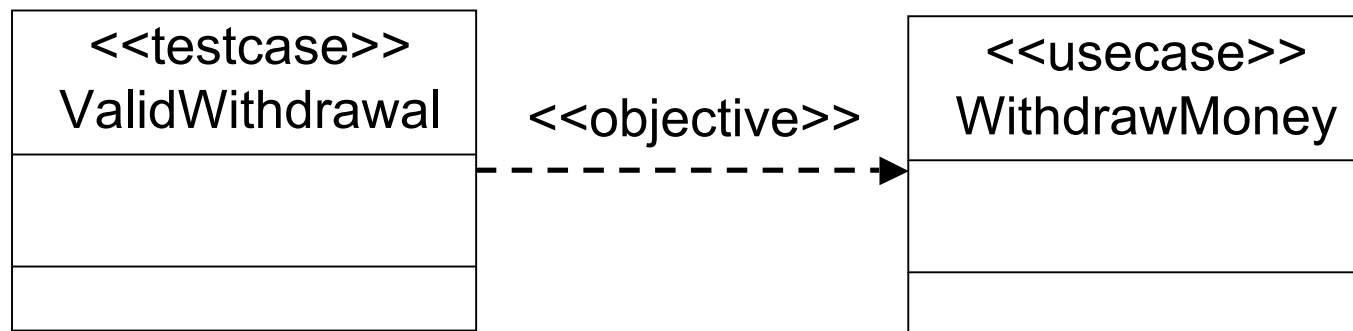
- **Test behavior**
  - Test objective, test case
  - Default behavior, verdict
- **Test architecture**
  - Test component, test configuration, test context
  - SUT (System under Test)
  - Test control, arbiter, scheduler
- **Test Data**
  - Wildcards, data pools, data partition, data selector, coding rules
- **Time Concepts**
  - Timers, time zone.

# General Definitions used in U2TP

- **Test:** An attempt to create a difference between the observed behavior and the specified behavior of a system a planned way
- **Test data:** The structures and meaning of values to be processed in a test
- **System under test (SUT)**
  - A system, subsystem or component
  - The SUT is treated as black-box. This means:
    - The SUT can be accessed only via a public interface
    - No information on the internal structure of the SUT is available for use in the specification of test cases.
  - **Test case:** Describes a specified behavior, that is, a behavior specified in the system model (normative behavior) of the SUT.

# Test Objective

- **Test objective**
  - Find any difference between the observed behavior and the specified behavior, that is, the behavior specified in the system model.
- **UML Example**
  - The test objective in the diagram below specifies that the objective of the ValidWithdrawal test case is to validate the WithdrawMoney use case.



# Test Architecture Definitions

- **Test component:** An object within a test system
- **Test architecture:** The test components and their relationships in a test system
- **Testing configuration:** The combination of a test system with a SUT
- **Test control:** A ordering of the execution of the tests performed on the SUT
- **Test context:** A description of
  - A set of test cases
  - A test configuration
  - A test control.

# Mandatory Elements in a test based on U2TP

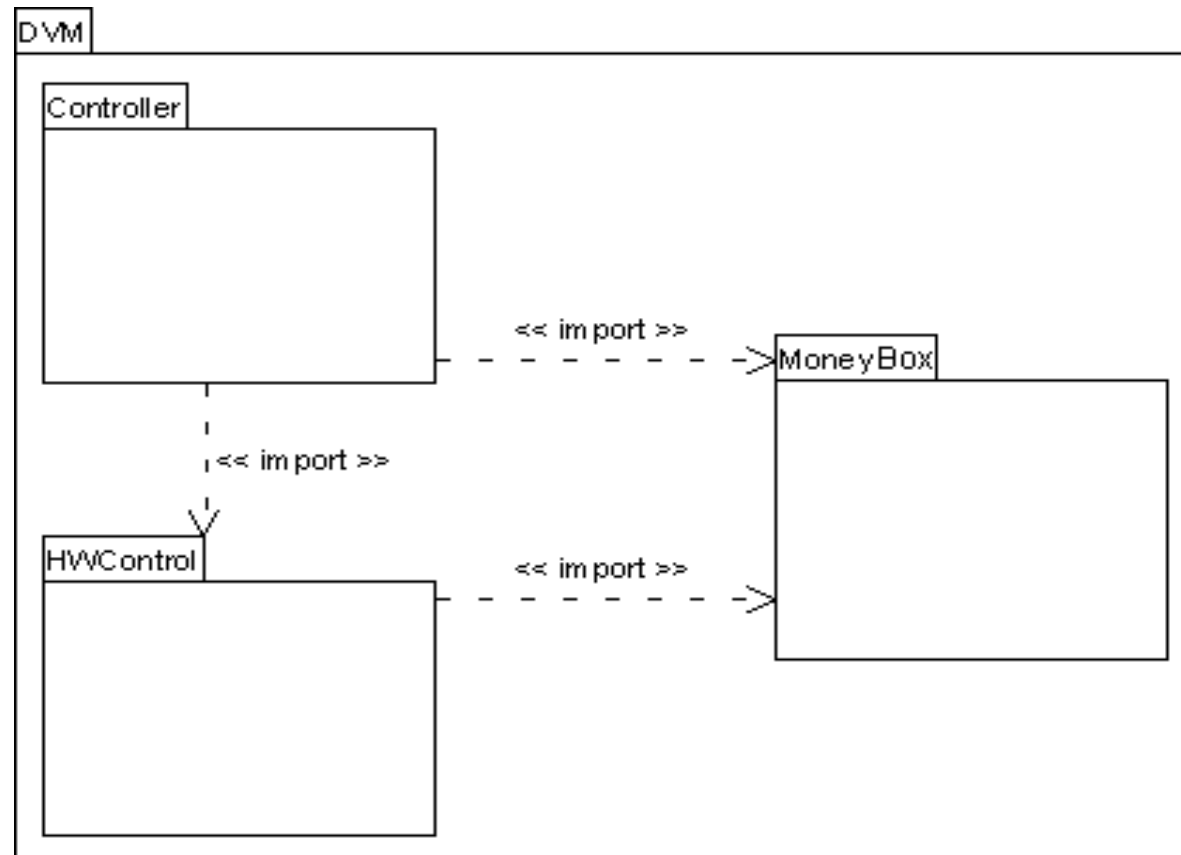
- A test system following the U2TP testing profile must consists at least of the following parts:
  - Test architecture: SUT, Test Context
  - Test behavior: TestCase, Verdict.

Architecture	Behavior	Data	Time
SUT	Objective		Timer
TestContext	TestCase		StartTimerAction
TestConfiguration	Observation		StopTimerAction
TestComponent	Stimulus		TimeOut
Arbiter	Verdict		

Mandatory Elements

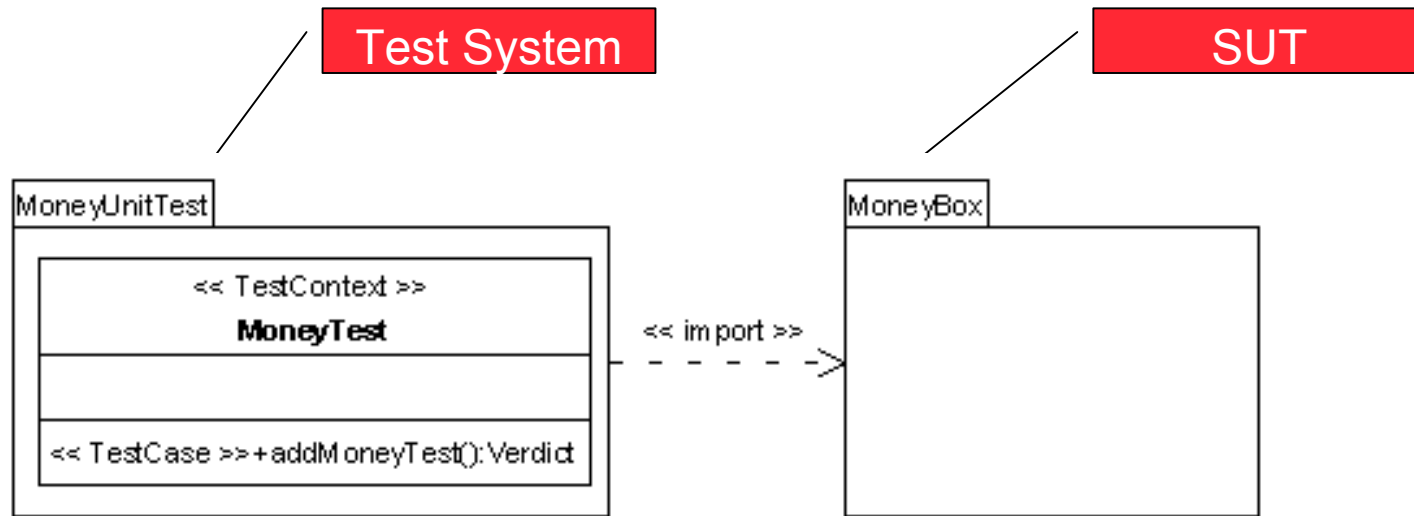
# Example: Test System for a Vending Machine

- Drink Vending Machine (DVM) with 3 use cases:
  - The drinking machine accepts cash payment
  - If the inserted money is sufficient, a drink is returned
  - If not, the money is returned.





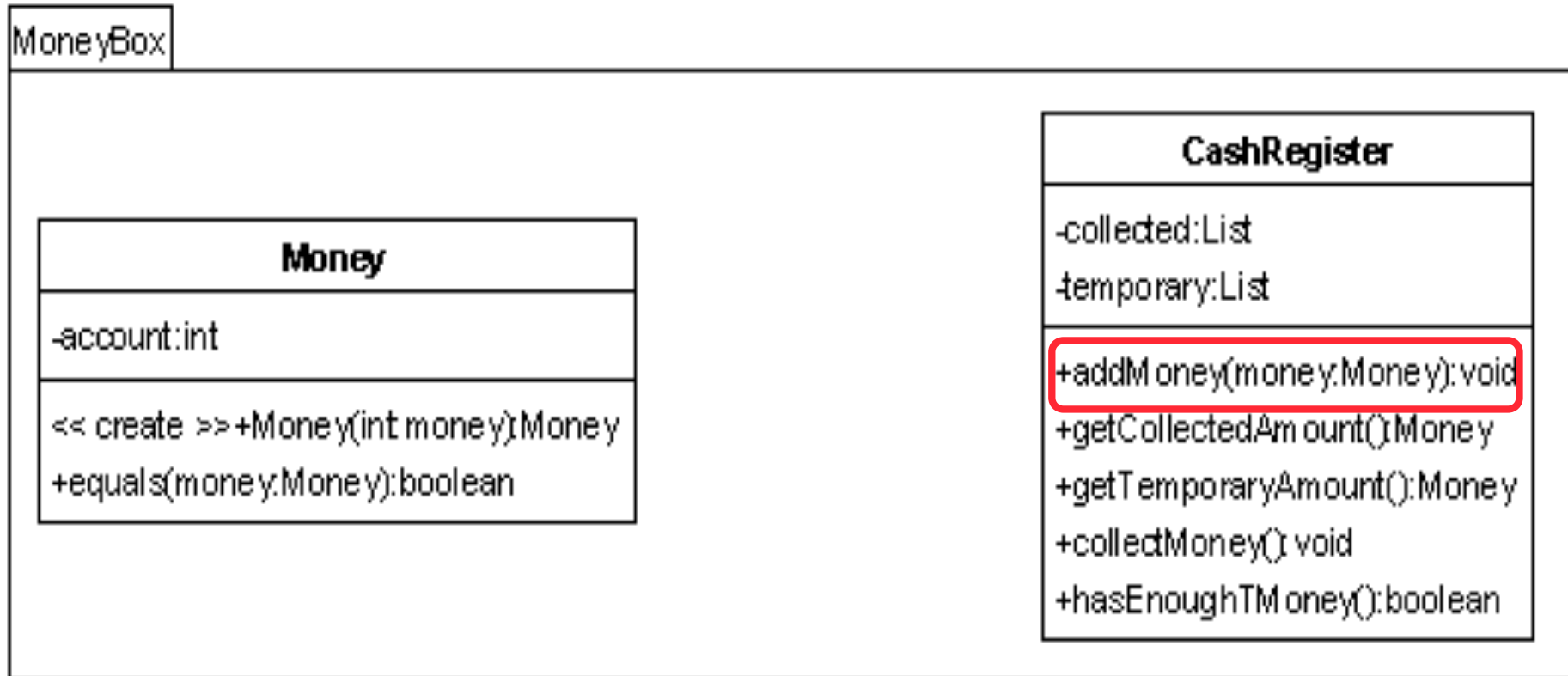
# Model-based Testing of Moneybox



The **test system** `MoneyUnitTest` is developed with the purpose of unit testing the class `Moneybox`, which is part of the **system under test**

`Moneybox` must be imported by `MoneyUnitTest` to be accessible.

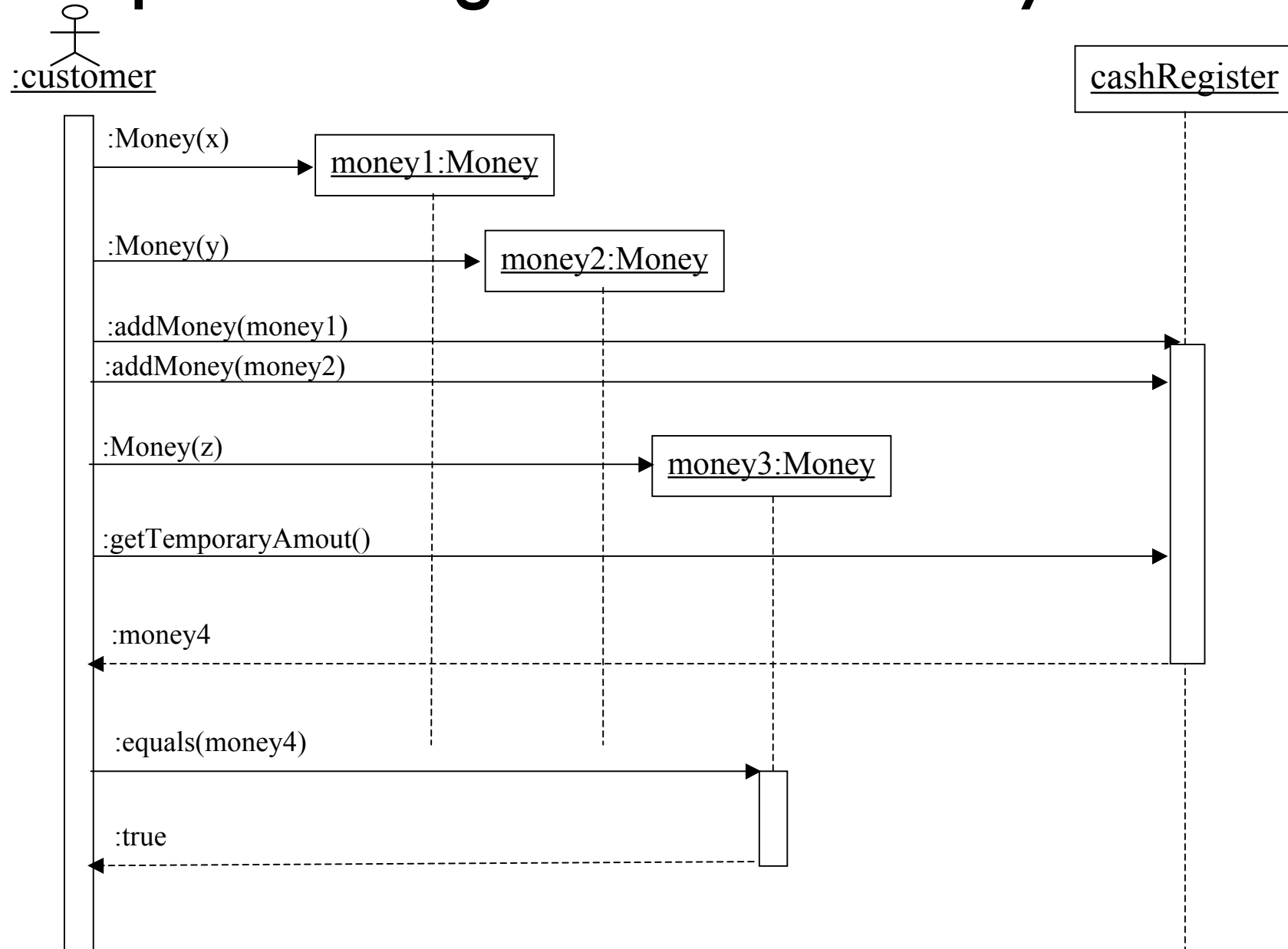
# UML System Model of the Moneybox



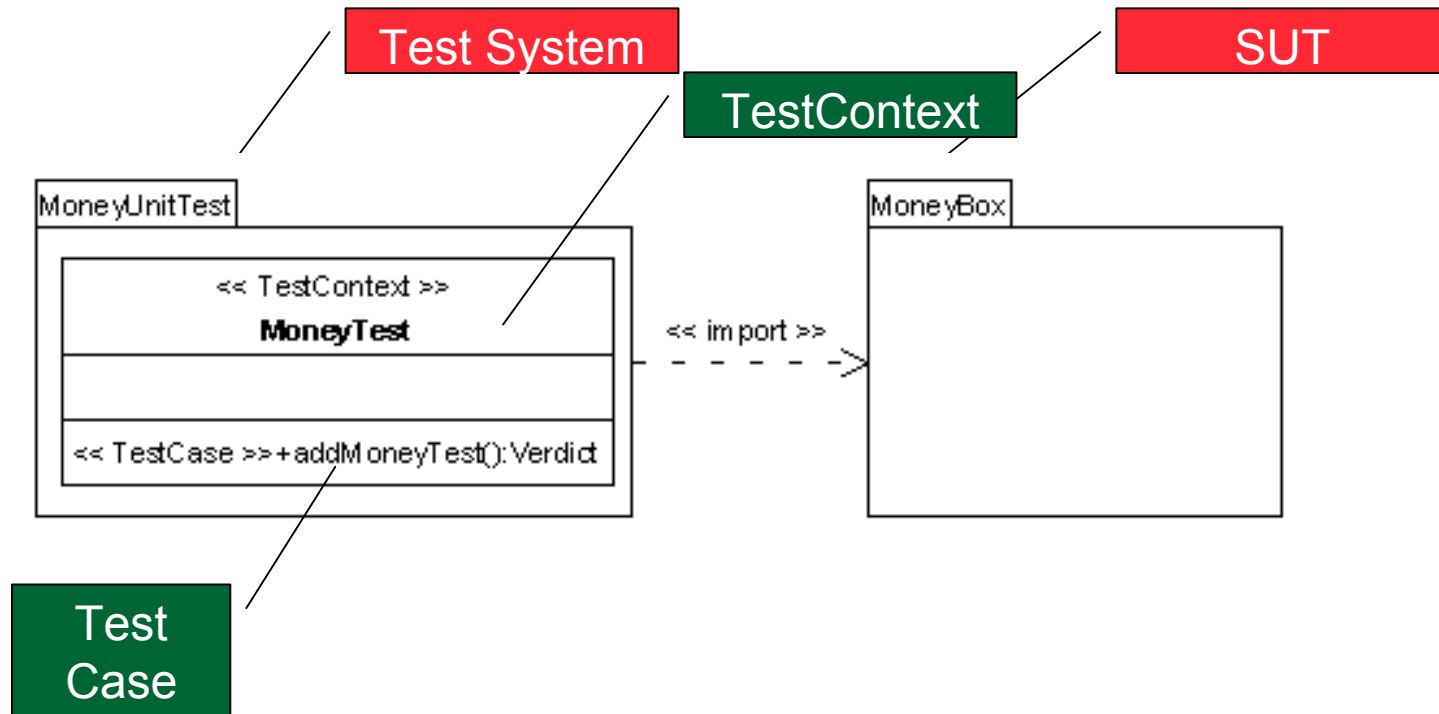
# Testing the addMoney Operation (Unit Test)

- Test that the machine correctly counts the money inserted by the user into the Moneybox.

# Sequence Diagram for addMoney



# Model-based Testing of Moneybox



MoneyTest is the **test context**  
MoneyUnitTest is the **test case**

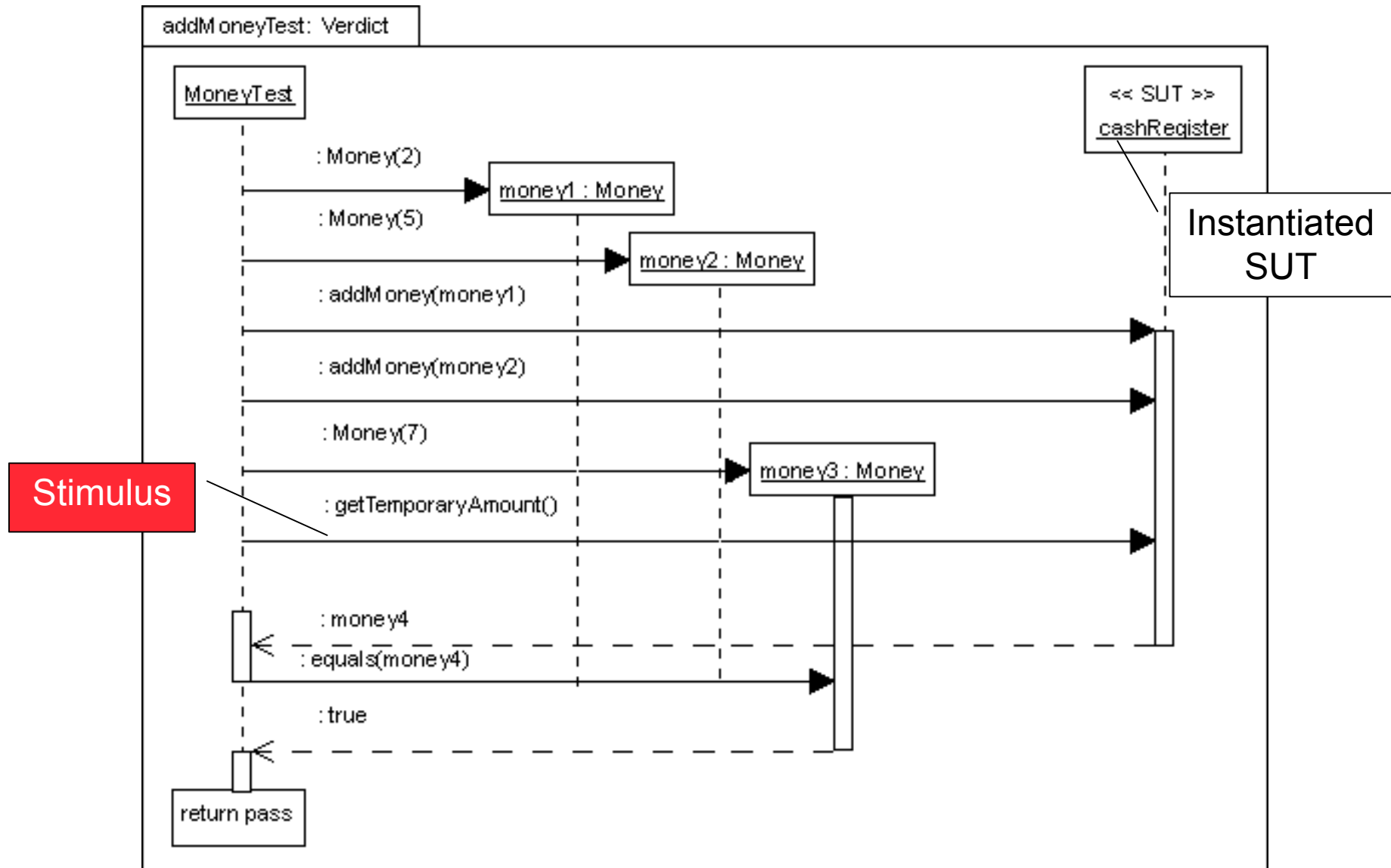
# Definition Test Case

- **Test case:** A test case implements a test objective. It describes a specified behavior, that is, a behavior specified in the system model (normative behavior) of the SUT
- A test case is defined in terms of sequences, alternatives, loops, and defaults of stimuli to and observations from the SUT.
- A test case consists of
  - **Flow of events** (use case, sequence diagram)
    - Stimulus to and Observation from the SUT
  - **Verdict:** Assessment of the correctness of the SUT
  - **Arbiter:** Evaluates the outcome of the test.

# Test Case

- A **Test case** specifies how a set of test components interact with the system under test realize a test objective
- A test case is owned by a test context
- Components of a test case
  - Event flow: Sequence of steps to execute the SUT
    - May include timing information
  - Input: Stimulus
  - Output: Observation
  - Expected result: Oracle
  - Verdict: Observation equals Oracle
    - Comparison done by an arbiter
- A test case may invoke other test cases.

# Sequence Diagram for addMoneyTest



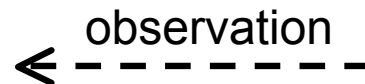


# Stimulus and Observation

- **Stimulus**: Data sent to the SUT

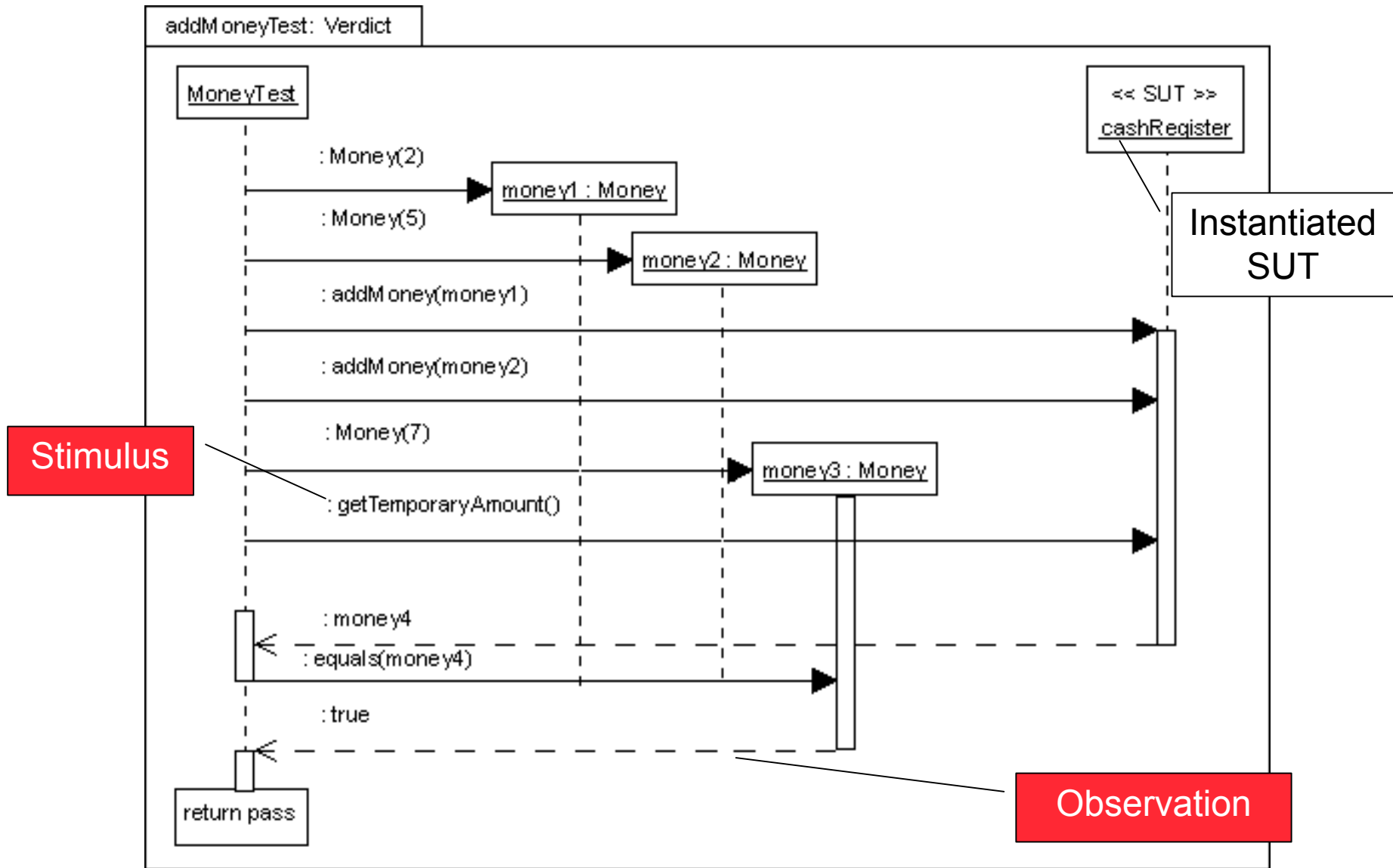


- **Observation**: Data returned from the SUT as a result of a stimulus.



- Both, stimuli and observations are described as **Test data**.

# Stimulus and Observation in addMoneyTest



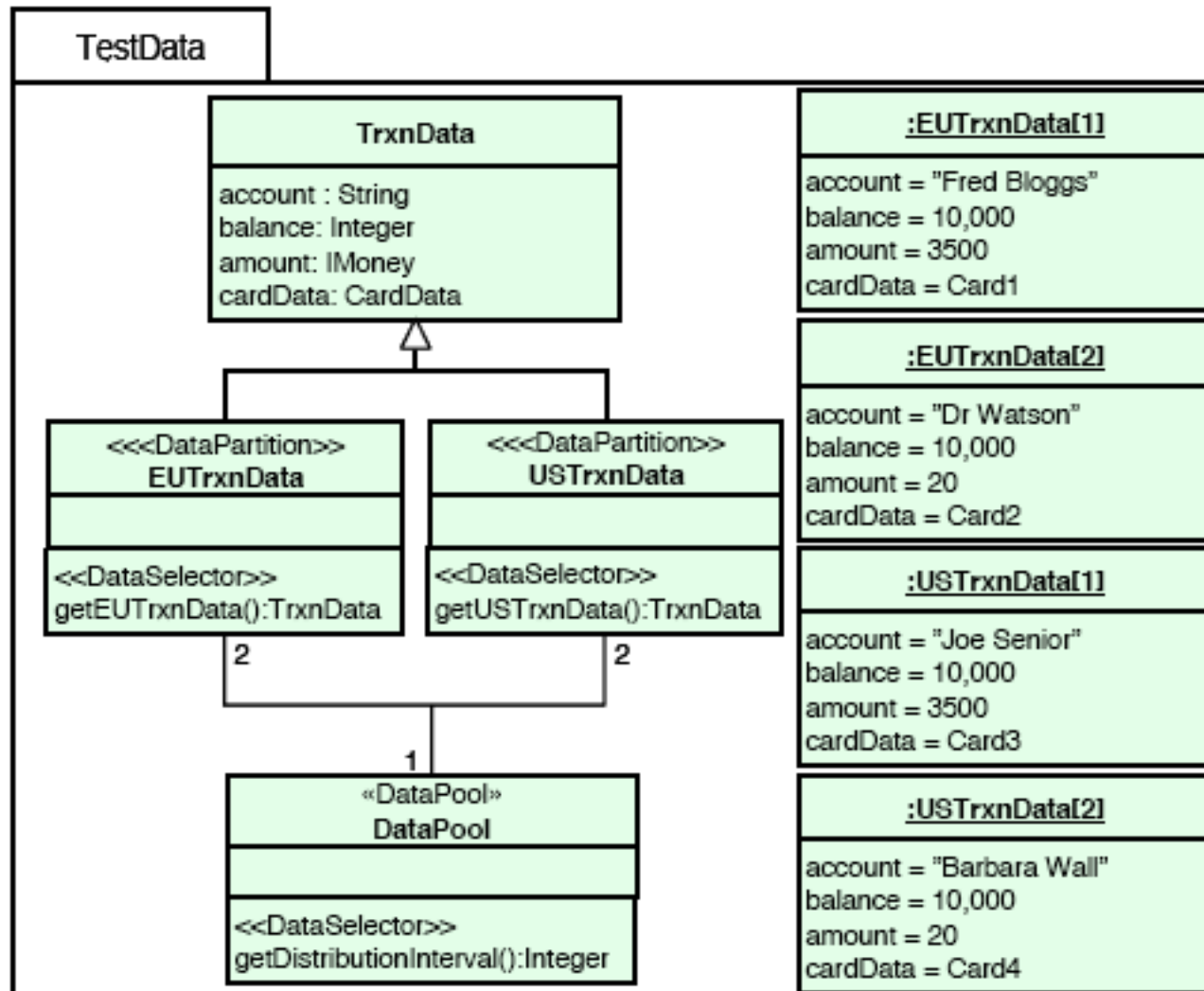
# Test Data

- **Test Data:** A generalization of stimulus and observation. There are three types of test data:
  - **Data pool:** Test data which are unspecific in their properties
  - **Data partition:** Test data which are unspecific in their values
  - **Wild card:** Test data which are unspecific in certain elements
  - **Data selector:** An operation that defines how data values or equivalence classes are selected from a data pool or data partition.
- **Coding rules:** Describe the encoding/decoding of test data to be sent to the interfaces of the SUT.

# Data Pool

- Data pools are a collection of data partitions or explicit values that are used by a test context, or test components, during the evaluation of test contexts and test cases. They can be used for repeated tests.
- Example
  - see figure on next slide

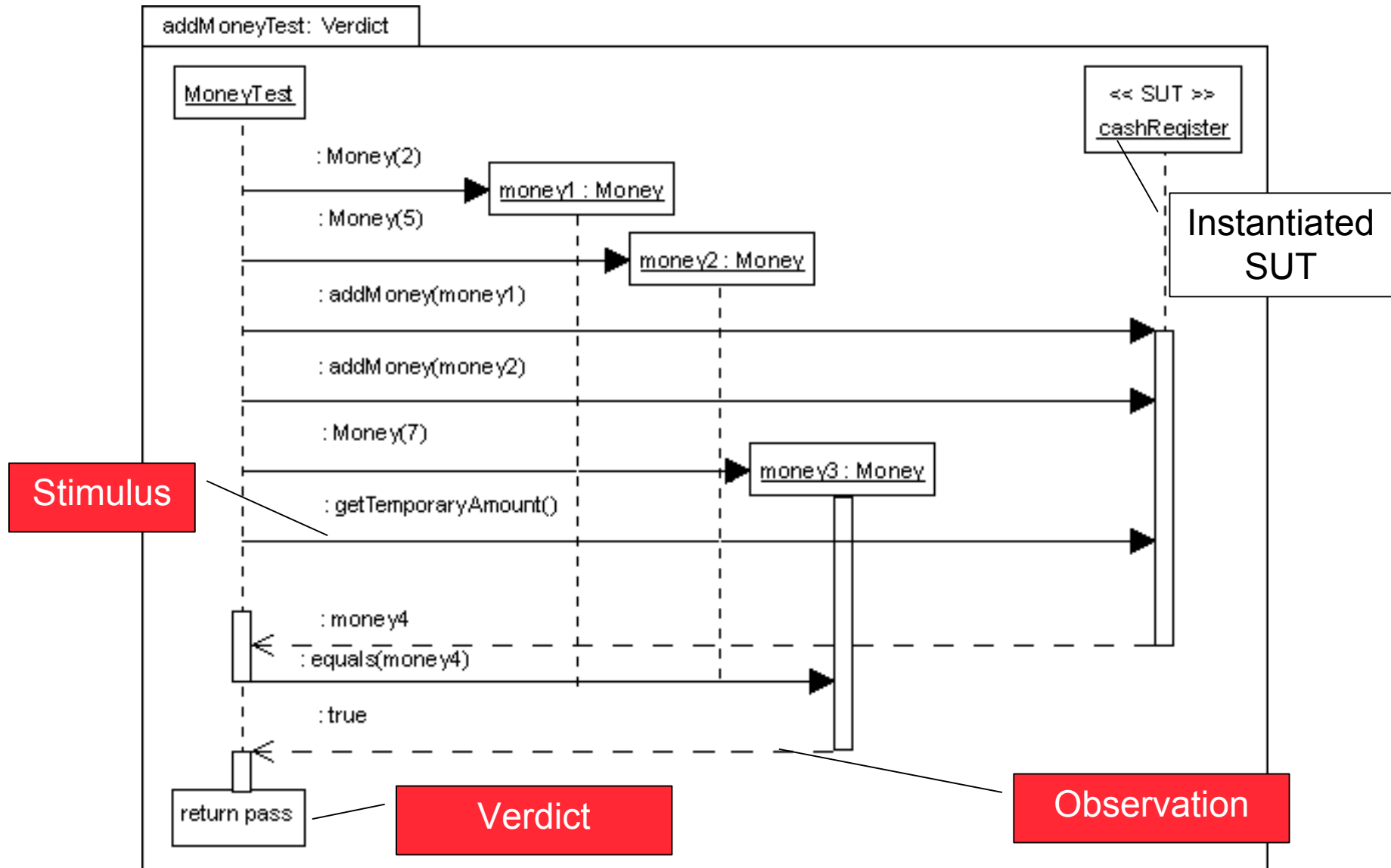
# Data Pool Example



# Verdict

- A test **verdict describes the result of a test** case execution:
  - **Pass** indicates that the specified behavior equals the observed behavior of the SUT for that specific test case
  - **Fail** describes that the specified behavior is not equal to the observed behavior of the SUT for that test case
  - **Inconclusive** is used if neither a Pass nor a Fail verdict can be given
  - **Error** is used to indicate a failure in the test case itself.

# Verdict in addMoneyTest



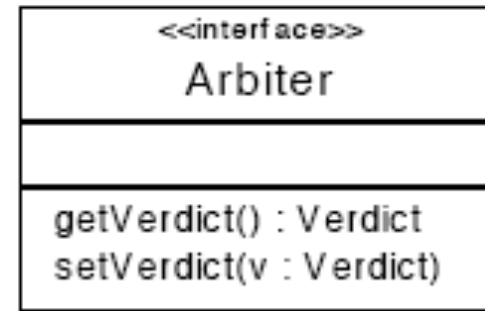
# Scheduler:

- Predefined interface in U2TP that controls the execution of the test components that take part in a test case
  - Keeps control over the creation and destruction of test components
  - Collaborates with the arbiter to inform it when it is time to issue a verdict.
- **Operations**
  - **Scheduler()** : Constructor of Scheduler. Starts SUT and Arbiter.
  - **startTestCase()** : Starts test case by notifying all involved test components.
  - **finishTestCase(t:TestComponent)** : Records that test component t has finished its execution. Notifies the arbiter, when all test components in the current test case are finished
  - **createTestComponent(t:TestComponent)** : Records that the test component t has been created by some other test component.



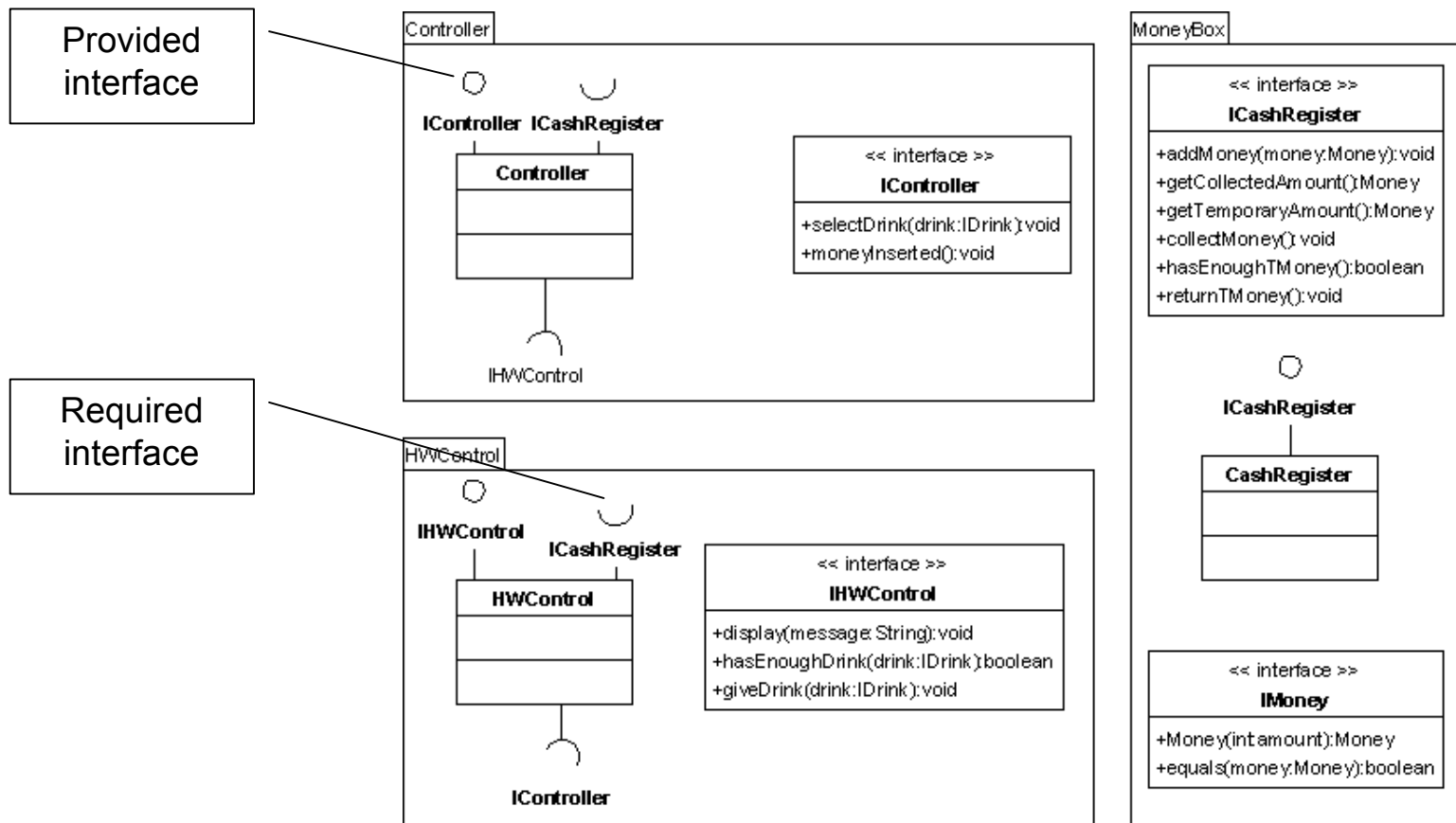
# Arbiter

- **Arbiter:** Called by the scheduler. Evaluates the execution of the test case and assigns a verdict to it
  - Arbiter is an interface provided by the UML Testing Profile.
  - **Operations**
    - `getVerdict`: Returns the current verdict
    - `setVerdict`: Sets a new verdict value
  - **Semantic of Arbiter:**
    - If a verdict is **pass**, it can be changed to inconclusive, fail, or error.
    - If a verdict is **inconclusive**, it can be changed to fail or error.
    - If a verdict is **fail**, it can be changed to error only.
    - If a verdict is **error**, it cannot be changed.



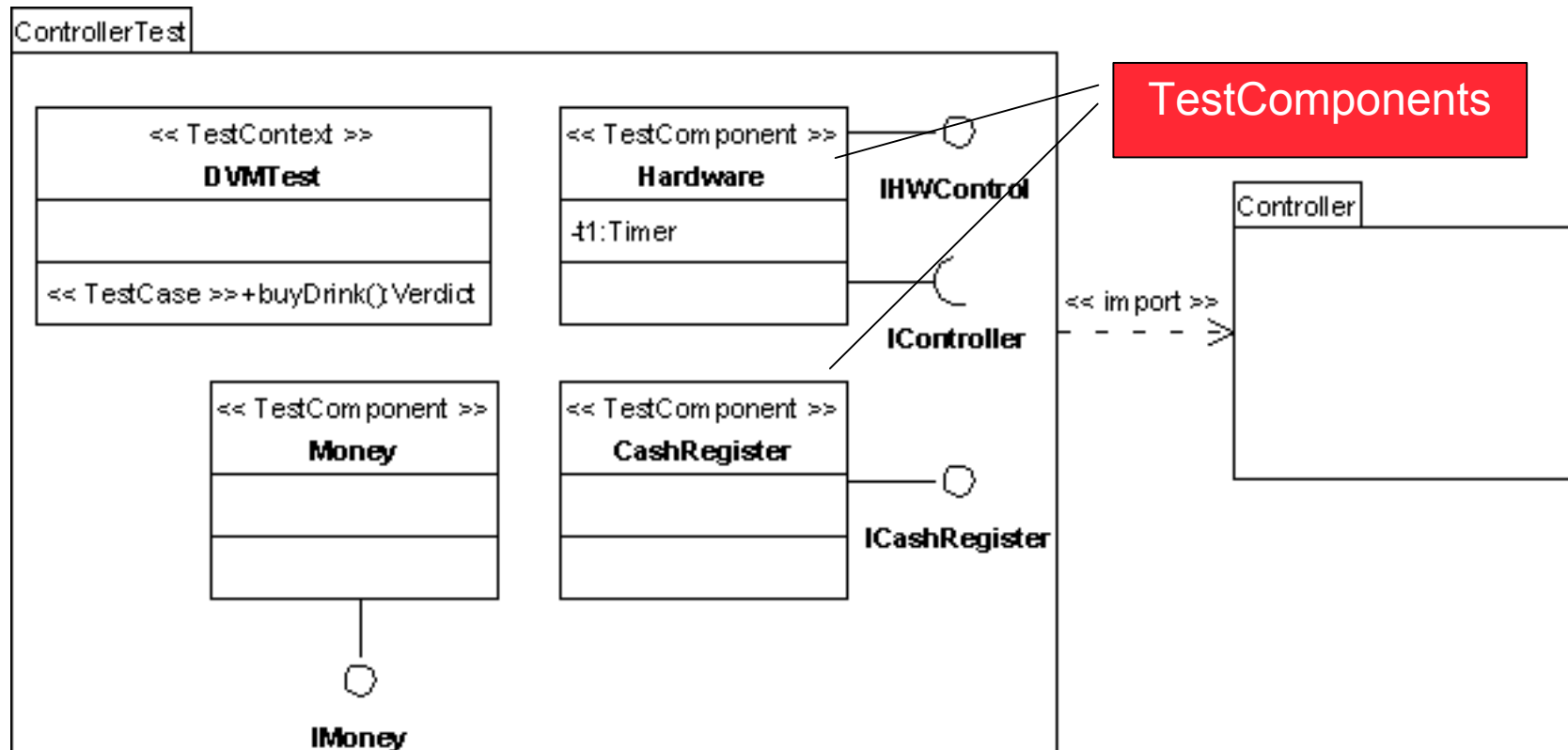
# Testing the DVM (Integration Test)

- Test Objective
  - Check the operation of the DVM when a user buys a drink
  - System is not yet complete:
    - **HWControl** and **CashRegister** are not yet developed.



# Develop the Test Components

- Test components implement the interfaces of emulated components
- The test components also interact with SUT

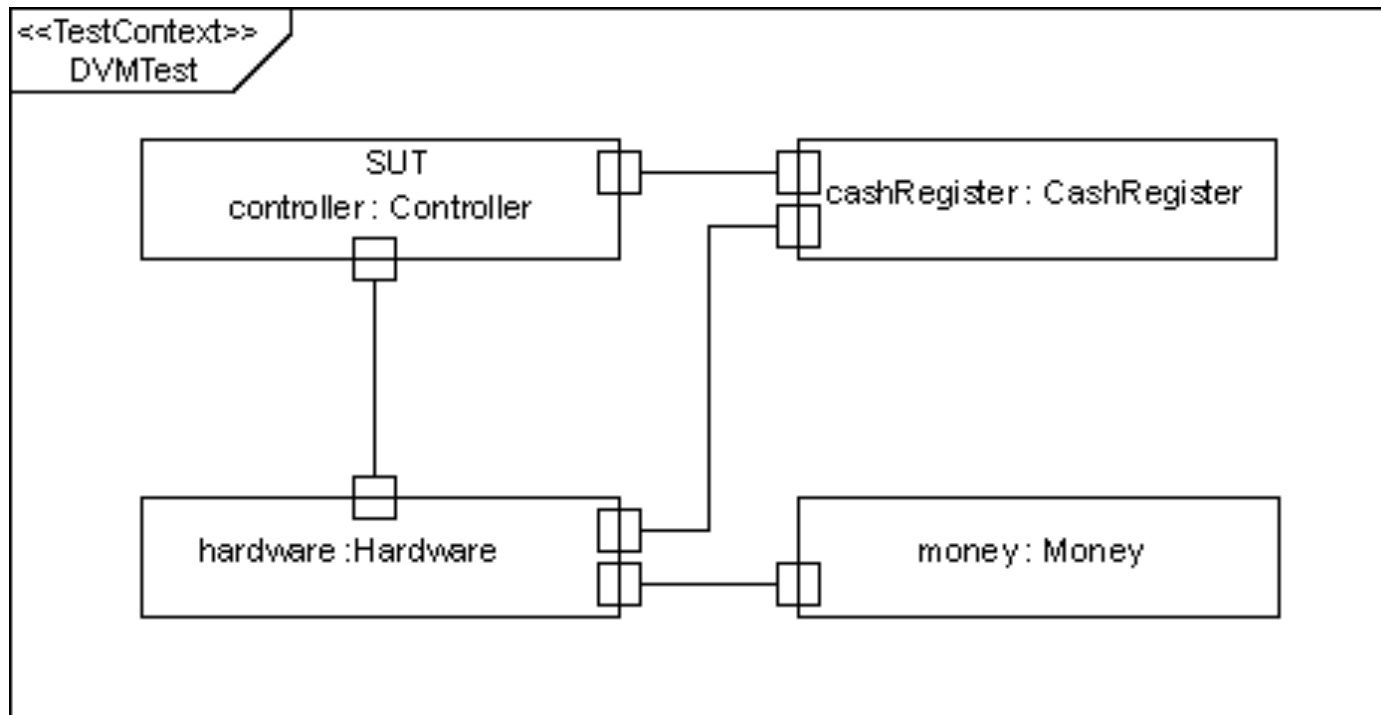


# Test Configuration

- **Test Configuration:** The test configuration defines instances of the test component (test objects and the connections between them)
- **Initial test configuration:** State of the test objects at the beginning of a test
- **UML Example:**
  - Testing Configuration for the Vending machine
  - Next slide.

# Test Configuration for the Vending Machine

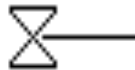
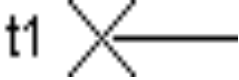
- Collection of test components and ports between these components and the SUT
- Defines the connections when the test case is launched.
  - the maximum number of ports and test components during the execution of the test case.



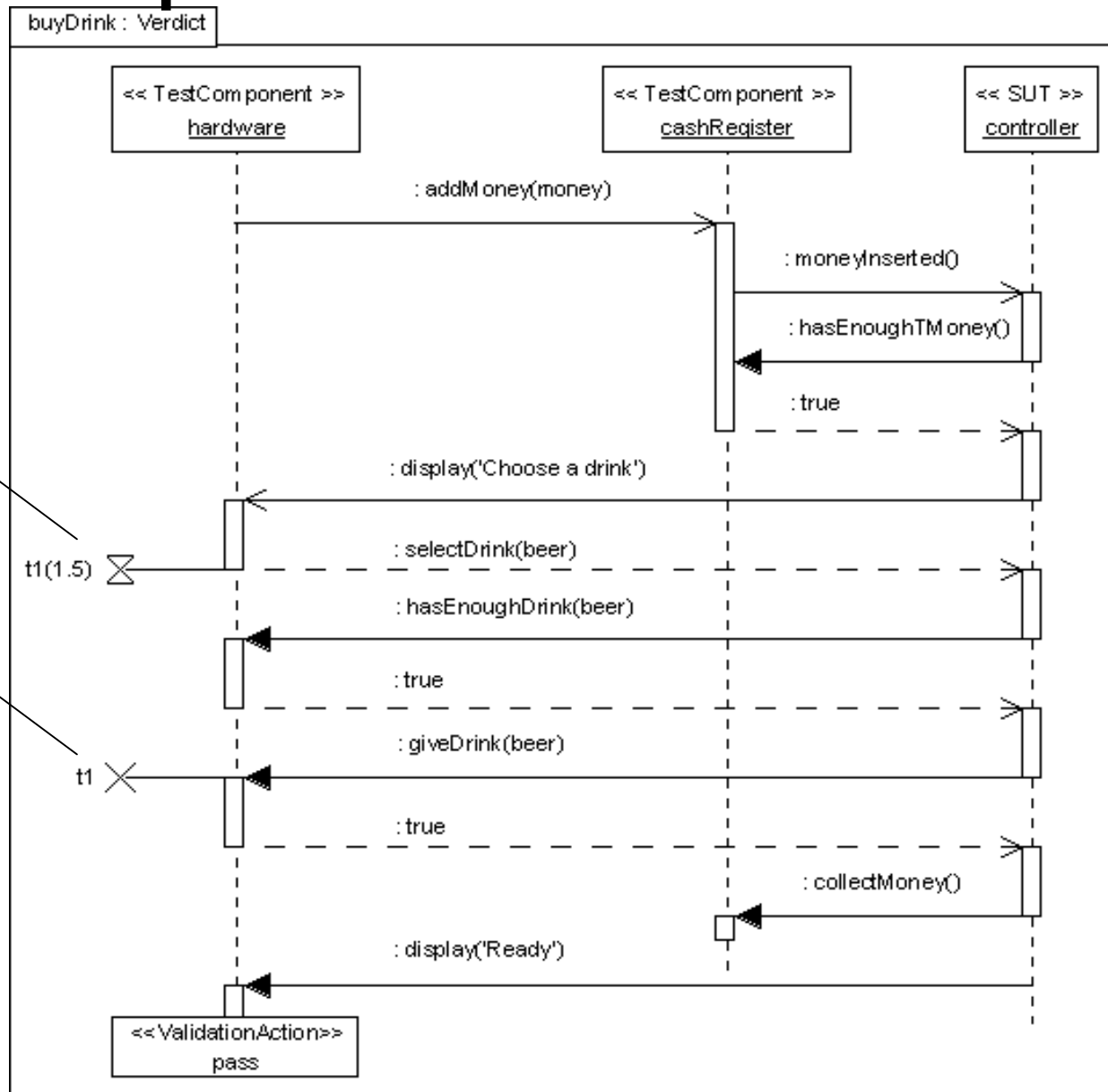
# Timer

- Timer is a mechanism to generate a timeout event occurrence
- This may occur when a pre-specified time interval has expired
- Timers are defined as properties of test components
  - A timeout indicates the timer expiration
  - A timer can be stopped.
  - The expiration time of a running timer and its current status (e.g., active/inactive) can be checked.

# Timer (2)

- StartTimerAction
  - Semantics: Start Timer t1 with the value 1.5 sec
  - Parameter: duration
  - Notation: t1(1.5) 
- StopTimerAction
  - Semantics: Stop Timer
  - Notation: t1 
- TimeOut
  - The time specified as duration in StartTimer has expired
- Example:
  - Test that the hardware provides a drink ("giveDrink") within 1.5 seconds after a drink has been selected.

# Timer Example



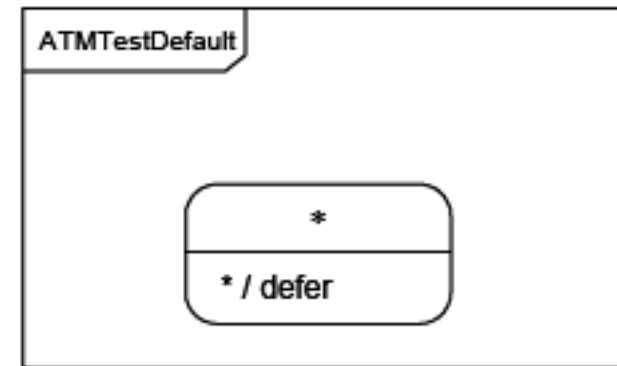
Start the  
Timer t1

Stop the  
Timer t1



# Wildcards

- Wildcards are special symbols representing values or ranges of values
- They specify whether the value is present or not, and/or whether it is of any value
- Wildcard types:
  - Wildcard for any value
    - Example: "?" (question mark)
  - Wildcard for any value or no value at all (i.e. an omitted value)
    - Example: "\*" (star)
  - Wildcard for an omitted value



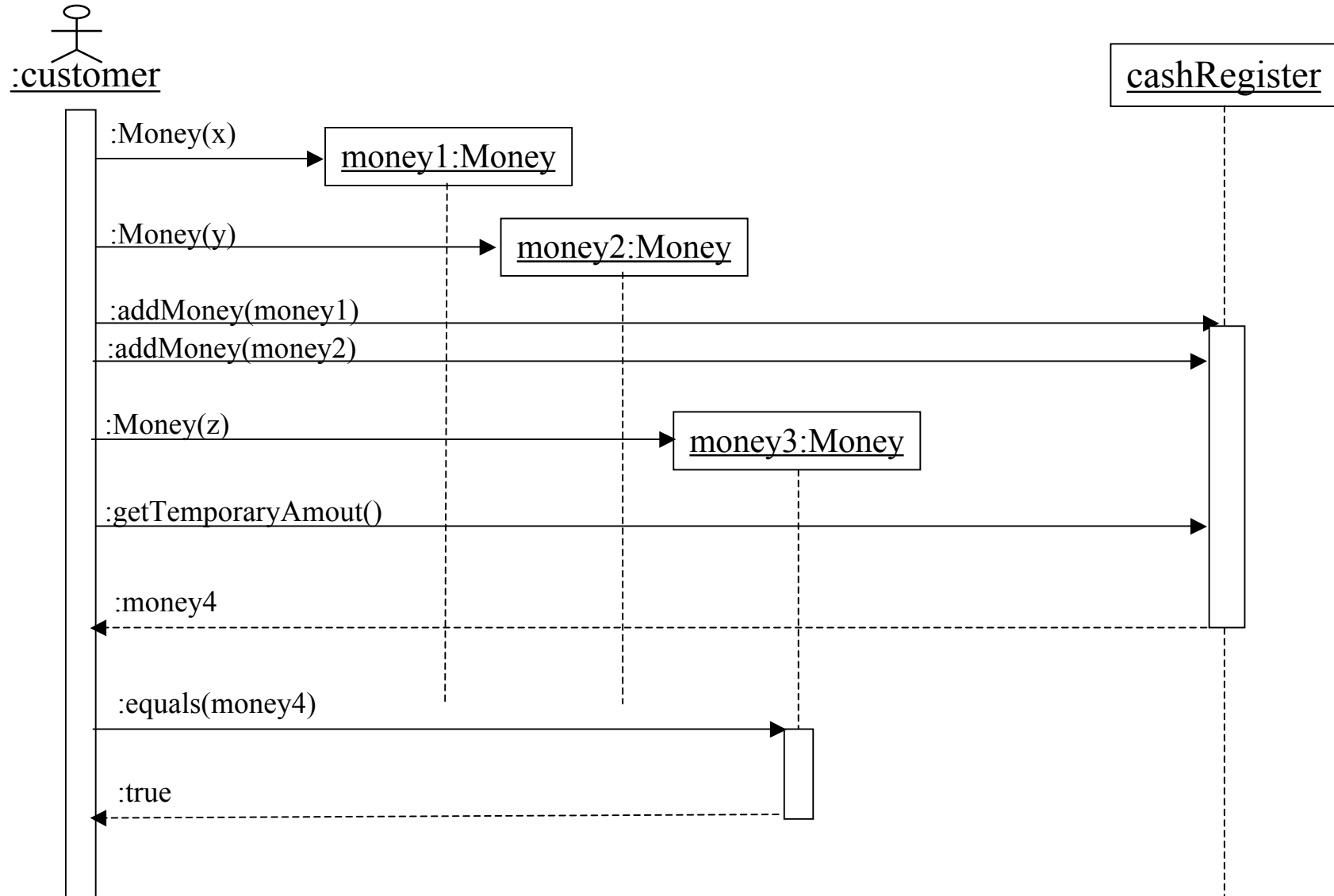
# Additional Readings

- XMI
  - <http://en.wikipedia.org/wiki/XMI>
- MDA
  - [http://en.wikipedia.org/wiki/Model-driven\\_architecture](http://en.wikipedia.org/wiki/Model-driven_architecture)
- The UML Testing Profile
  - Ina Schieferdecker, Øystein Haugen, 2004
  - <http://folk.uio.no/oystein/h/Schieferdecker-Haugen-ECOOP2004-U2TP.pdf>

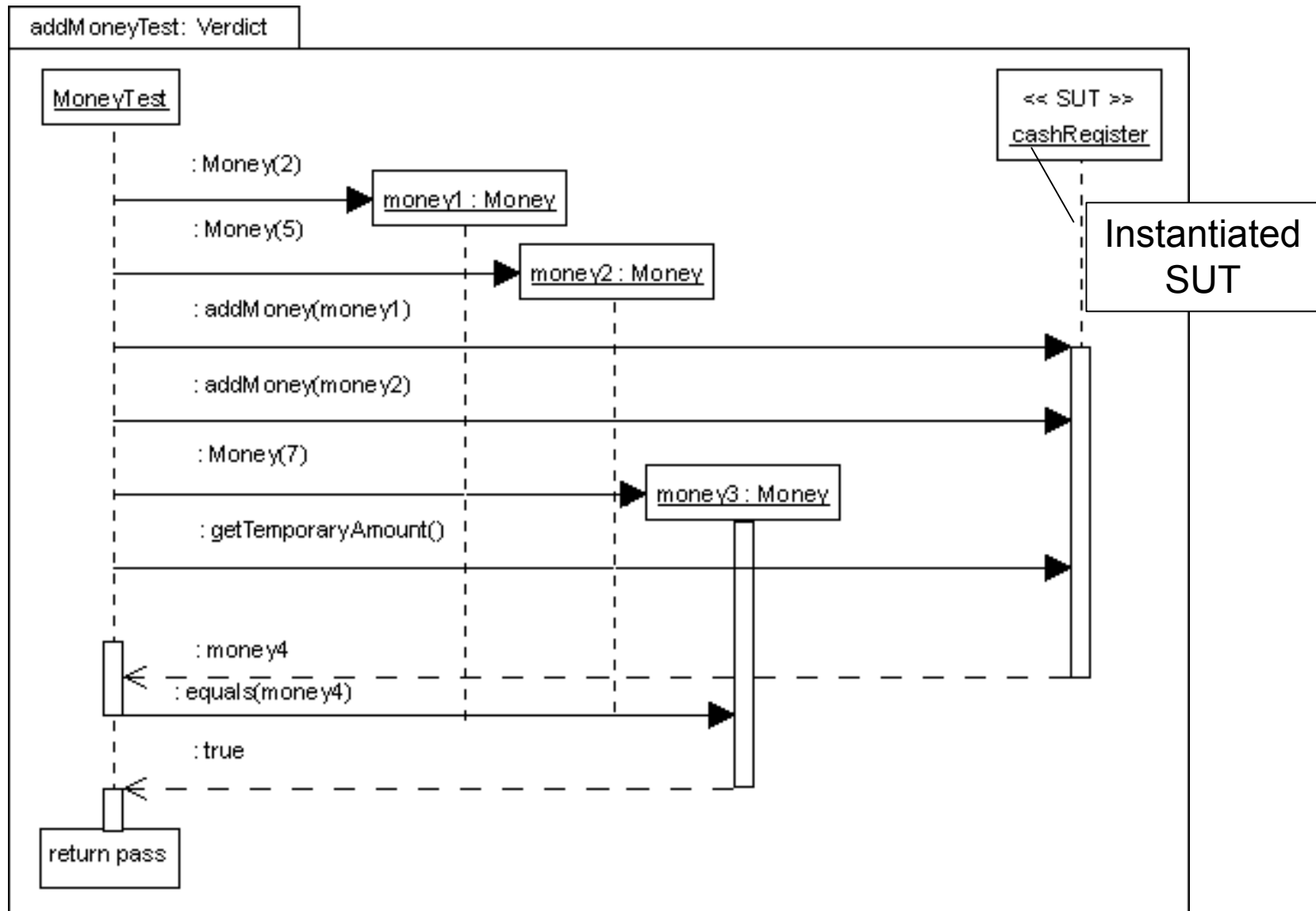
# Summary

- Review of all the definitions introduced in this part of the lecture.
- Writing test cases for a use case
  - Write the use case
  - Determine SUT
  - Add Stimulus, Observation and Verdict to use case, turning it into a test case.

# Sequence Diagram for addMoney



# Stimulus and Observation in addMoneyTest

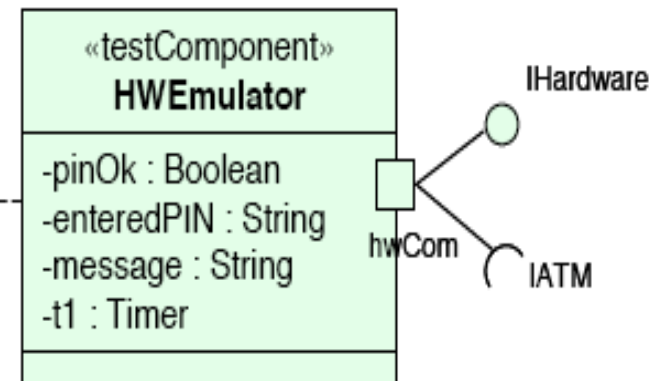
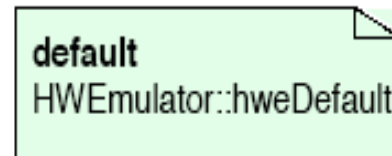
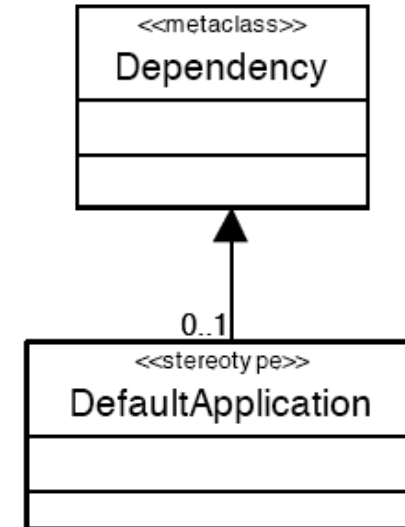
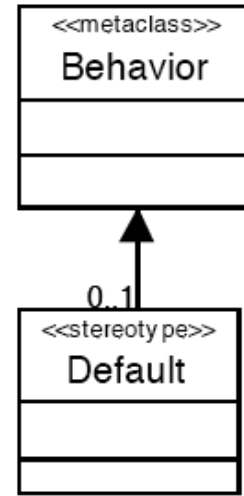
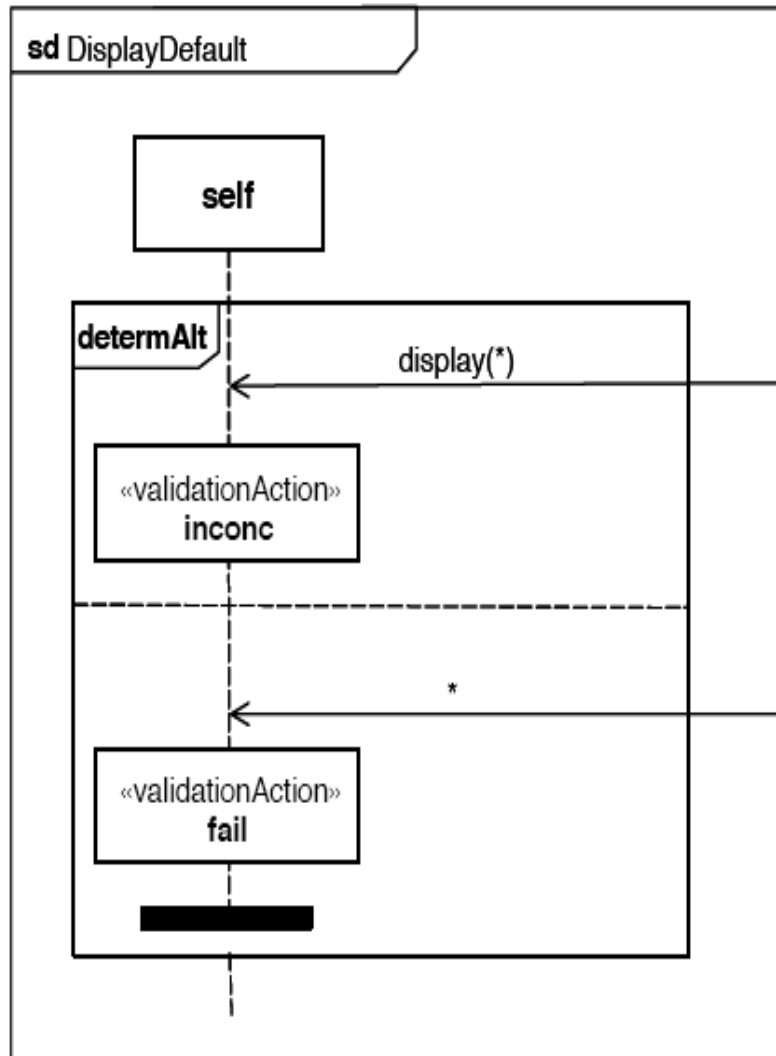


# Defaults

A default is a behavior triggered by a test observation that is not handled by the behavior of the test case per se. A default is triggered whenever a test observation is not handled by the behavior of a test case. There is a hierarchy of defaults:

- within the behavior of a test component object associated to a state (i.e. in a state machine) or to a combined fragment (in an interaction diagram);
- for the complete behavior of a test component object associated to a part (typed with a test component) in the internal structure of a test context (i.e. in a test configuration);
- for all test component objects of a test component associated to a test component in a test architecture.

# Default Examples







# Validation Action

A validation action is an action performed by a test component to assess a test observations and/or additional characteristics/parameters. A notation to model validation actions is still to be defined.

- Example

```
<<validationAction>>  
pass
```

```
<<validationAction>>  
x > 10 ? pass : fail
```