

PLM and Innovation
Excellence

Learning Campus

Your partner for
Business Learning

Siemens
Core
Learning
Program

Test Environment

Authors:
Peter Zimmerer, CT | Sylvia Jell, CT

Critical success factor

The test environment is arguably the most important tool of a tester, it is one of the critical success factors of testing.

Sogeti



Test environment

Learning objectives

- Understand needed requirements and drive quality of test environments
- Learn how to design and implement appropriate test environments
- Get to know supporting technologies

Test Environment

Agenda

Basics

Test Architecture

Specification and Design

Supporting Technologies

Summary

Test environment

Reference to other sessions

Remember ?

SIEMENS
Ingenuity for life

Topic has already been touched within other sessions:

Role of the Test Architect

- Test Architect has the role of a Software / System Architect for the test system

Processes

- Test environment may change / evolve between iterations which has to be considered in planning

Test Strategies

- Test environment (test lab) is a test strategy driver
- Test environment readiness is a test-related risk
- Preventive testing can entail pretesting test environments

RE

- Test environment requirements need stakeholders to be involved (to address their real needs)

Estimation

- Test Architects are involved in effort, duration and cost estimations regarding the test environment

Test Automation

- Effort and cost of the test environment are test automation factors to be considered
- New directions in test automation: virtualization to rapidly define and provision test environments

Test environment – Test infrastructure

ISTQB Glossary (<http://www.istqb.org/>)

Test infrastructure – The organizational artifacts needed to perform testing, consisting of test environments, test tools, office environment and procedures.

Test environment – An environment containing hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test.
Synonyms: **test bed**, **test rig**

ISO/IEC/IEEE 29119-1

Test environment – Facilities, hardware, software, firmware, procedures, and documentation intended for or used to perform testing of software.

Test environment requirements – Description of the necessary properties of the test environment.

TMap® (Test Management Approach by Sogeti)

Test environment – A composition of parts, such as hardware and software, connections, environment data, maintenance tools and management processes in which a test is carried out.



Test environment planning, design, and management

Test environments by no means are self-evolving

They have to be **actively and professionally planned, designed, and managed**, requiring to take a **multitude of parameters** into consideration



Test environment areas

A **test environment** is a composition of parts, such as hardware and software, connections, environment data, maintenance tools and management processes in which a test is carried out.

Sogeti

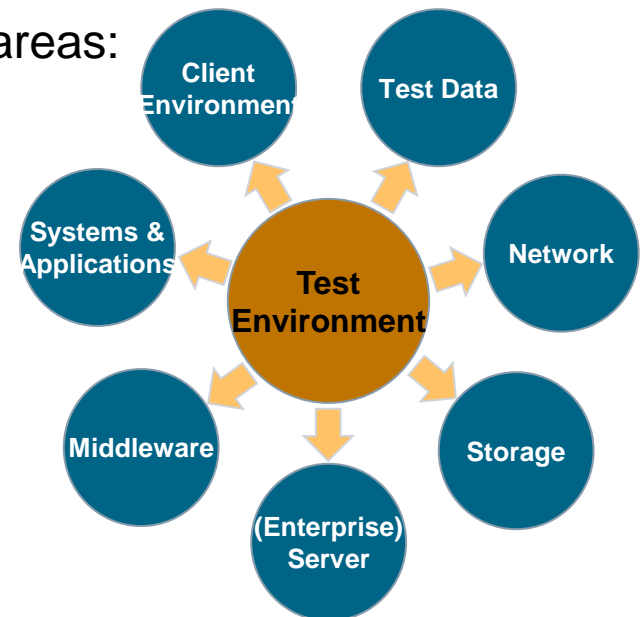
The **test environment** consists of a number of distinct areas:

- Systems and applications
- Client environment, incl. all kinds of devices
- Test data
- Network
- Storage
- (Enterprise) Server
- Middleware

with tooling as an overarching, binding influence

Organizational and procedural tasks in addition to the technological ones

→ Setting up and maintaining a test environment is a multi-disciplinary effort !



Test environment

Responsibilities of the Test Architect

Architecture & Design

- Test infrastructure requirements
- Design prototypes for the test infrastructure
- Technical risks of the test environment
- Integration plan for the test infrastructure
- Integration progress for the test infrastructure
- Reuse strategy for test infrastructure components

Maintenance & Evolvment

- Continuous architecture management for the test environment
- Documentation of the test infrastructure's architecture
- Internal quality of the test environment
- Removal of defects found in the test environment

Test environment – Basic facts

Setup and composition of a test environment is dependent on the aim of a test

**The success of a test environment depends on the degree to which
it can be determined
to what extent
the test object meets the requirement**

Different test types have different aims → different test environments necessary

- E.g. a unit test requires a completely different configuration of the test environment than a production acceptance test

**Test environments can differ considerably in size and complexity:
limited vs. huge collection of HW, SW, interfaces, procedures**

Koomen, Tim; van der Aalst, Leo; Broekman, Bart; Vroon, Michiel: *TMap NEXT, for result-driven testing*, Sogeti, 2014

Test environment types

Different test types require different test environments

- **Development test environment**
 - For the execution of technical (white-box) unit tests
 - Usually responsible: developer of the application

- **System test environment**
 - For technical (white-box) and functional (black-box) system tests
 - Must be logically separated from the development environment

- **Acceptance test environment**
 - For acceptance tests by users and (functional and operation) managers
 - Must be logically and physically separated from the other environments
 - Must reflect the final production system

Pinkster/van de Burgt/Janssen/van Veenendaal, *Successful Test Management: An Integral Approach*, Springer, 2004

Setting up test environments

A number of steps are involved in the setting up of a test environment

- **Definition of the requirements of the relevant test environment**
 - Product/project risks due to differences between the test infrastructure and the final production infrastructure
 - Measures to be undertaken considering the risks
 - Timetable of required activities and contingencies of the environment set-up
- **Specification of the test environment**
 - Detailed plan of the required test environment
 - Agreement with suppliers and specialist
- **Realization of the test environment**
 - Setup according to plan
- **Acceptance of the test environment**
 - Check of completeness
 - Adjusted planning according to noted shortcomings

Pinkster/van de Burgt/Janssen/van Veenendaal, *Successful Test Management: An Integral Approach*, Springer, 2004

Designing near-production test environments (1)

(Test-) activity	(Test-) objective
Evaluation	Free testing.
Development	Coding.
Component tests	Early identification of functional, robustness and performance errors.
Component integration tests	Test of interfaces and cross-component functionality.
Functional system tests	Tests from the user's perspective (regarding functionality, robustness, system interfaces).
Performance tests	System test looking at performance and resource efficiency.
Usability tests	Considers usability and accessibility.
Functional acceptance tests	Validation of acceptance criteria.
Technical acceptance tests	Test to establish the fulfillment of operational acceptance criteria.
System integration tests	End-to-end test of important business processes.
Production	Productive use of the actual system.
Training	Training of follow-up, as well as current versions of test objects.
Failure analysis and re-test	Reproduction of complex production-flaws. Proof of their redemption.

Christian Brandes, *Designing near-production test environments*, professionaltester.com, 2016

Designing near-production test environments (2)

R = required

ACTIVITY	HARDWARE					SOFTWARE/MIDDLEWARE					DATA				OTHER (SECURITY, ADMINISTRATION...)						
	Server	Client	Storage	Peripherals	Network	OS Server	OS Client	Load balancer	DB/Cluster	Test object/config.	Base data	pre-cond. data	amount	up-to-date-ness	Deployment	partner systems	accessibility tools	downtime	date simulation	SSO	Auth. Server
Evaluation	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Development	○	○	○	○	○	○	R	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Component tests	○	○	○	○	○	○	R	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Integration tests	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	R	○	○	○	○	○
Functional system tests	○	R	○	R	○	R	R	○	○	○	R	R	○	○	○	R	○	○	○	○	○
Performance tests	R	R	R	○	R	R	R	R	R	○	R	○	R	R	○	○	○	○	○	R	R
Usability tests	○	R	○	○	○	○	○	○	○	○	○	○	○	○	?	○	○	○	○	○	○
Accessibility tests	○	○	○	○	○	○	○	○	○	R	○	○	○	○	○	○	R	○	○	○	○
Functional acceptance tests	○	R	○	R	○	R	R	○	○	R	○	○	○	R	R	R	○	R	R	R	R
Technical acceptance tests	R	R	○	R	○	R	R	○	R	R	○	○	○	○	R	R	○	R	R	R	R
System integration tests	○	○	○	○	○	○	○	○	R	○	○	○	○	○	○	R	R	○	○	○	○
Production	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Training	○	○	○	R	○	○	R	○	○	○	○	○	○	○	○	R	R	○	○	○	?
Failure analysis & re-tests	R	R	○	R	○	○	○	R	R	R	R	R	○	R	R	R	○	R	○	○	○

Christian Brandes, *Designing near-production test environments*, professionaltester.com, 2016

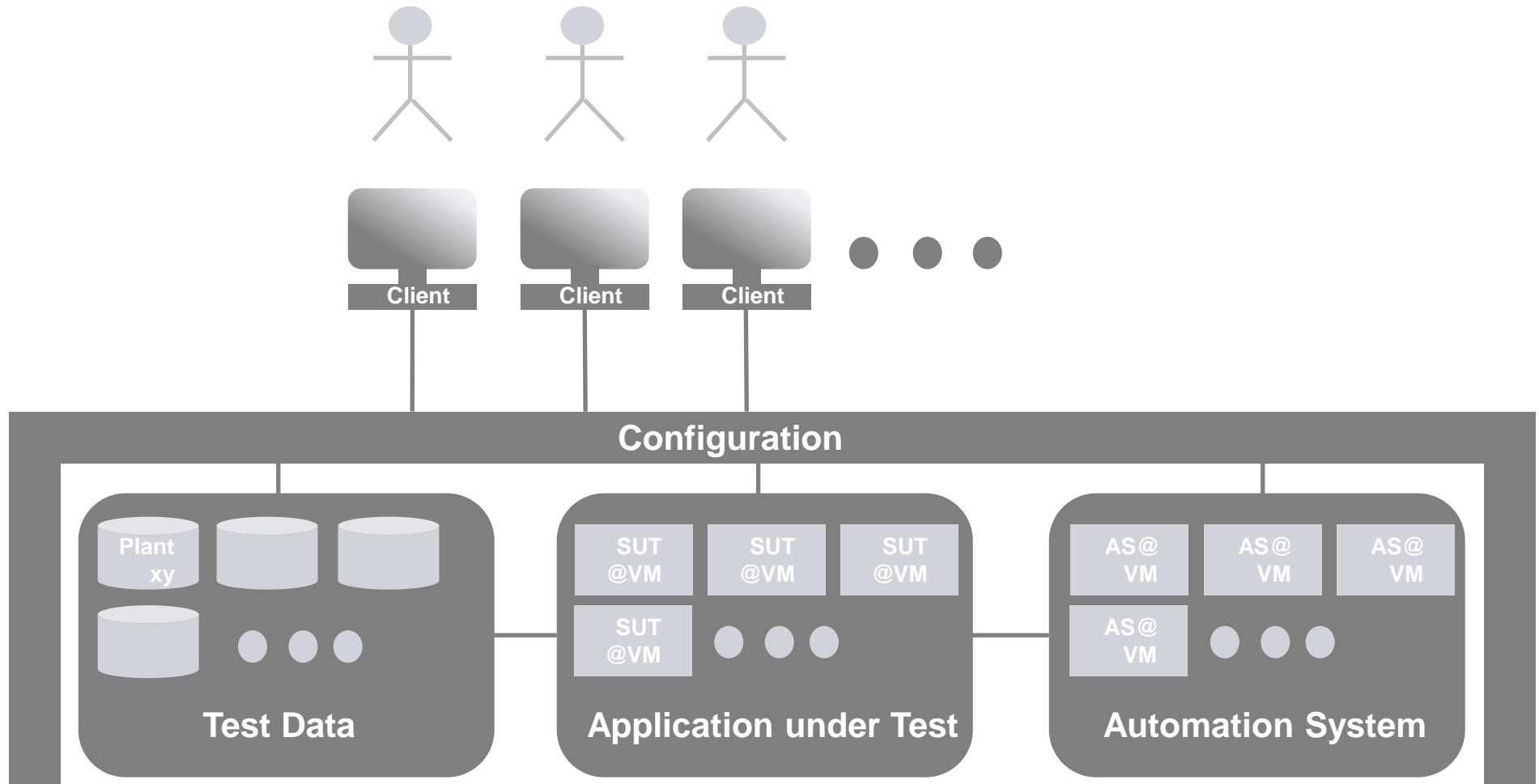
Test environment – efficiency

Flexible, configurable test environments necessary for effective and efficient test execution

Can be achieved using technologies such as

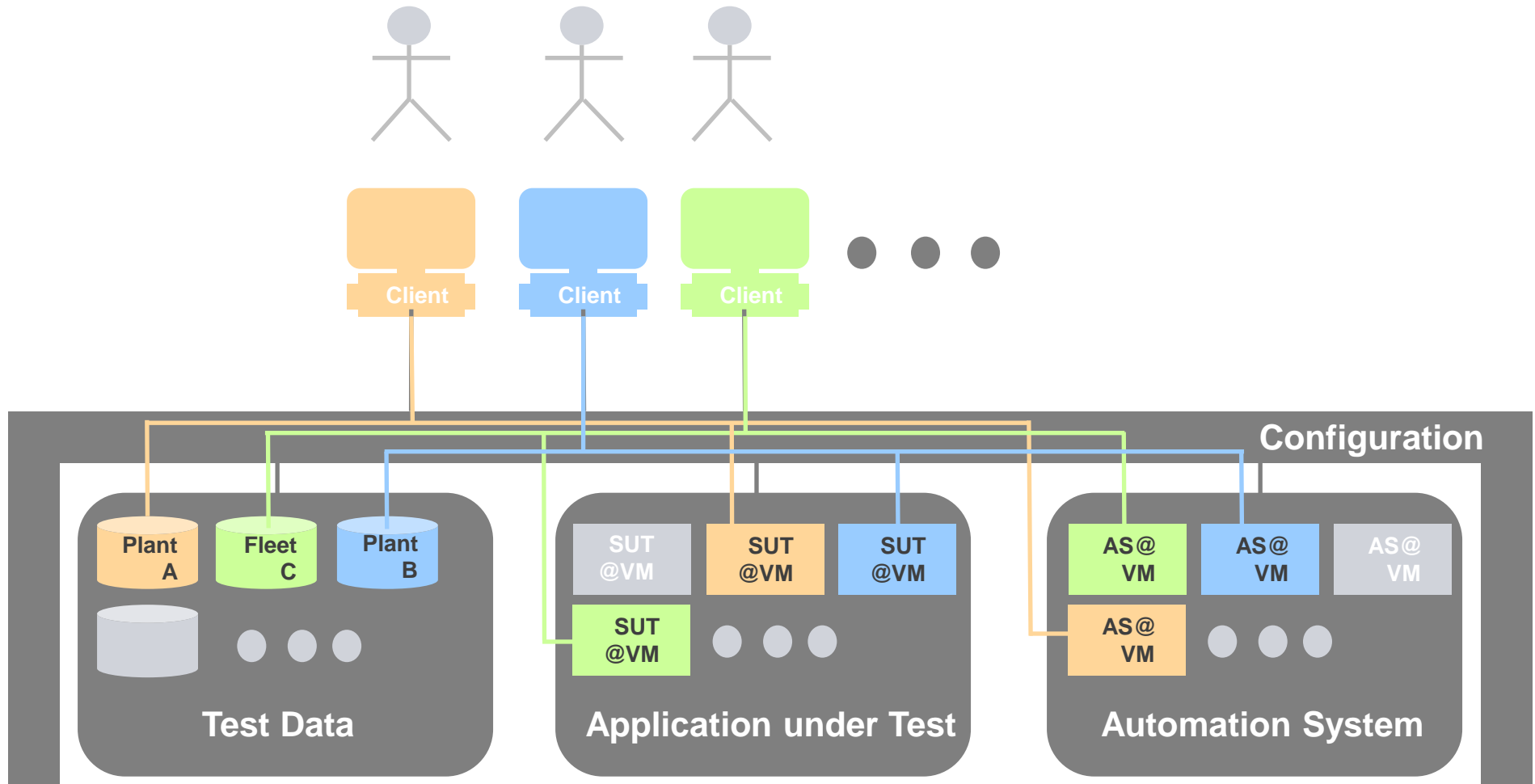
- **Virtual machines for the SUT application**
- **Virtual machines for automation system including field simulation**
- **Selectable test data (engineering data, users etc.)**
- **Cloud technologies**

Configurable test infrastructure



Example from E F IE

Configuration example



Example from E F IE

Exercise

Test environment setup in your organization

- What are the tools / best practices you are using?
- Sketch the tool / process landscape from one of your projects
- Discuss advantages and improvement options



Test Environment

Agenda

Basics

Test Architecture

Specification and Design

Supporting Technologies

Summary

Test architecture

Test architecture refers to the high level structures of a test system that is used/required to test a system, the discipline of creating such structures, and the documentation of these structures. These structures are needed to reason about the test system. Each structure comprises test elements, relations among them, and properties of both elements and relations ... (*adapted from Wikipedia*)



TestArchitecture
@InSTA2015

Test architecture is the **architecture** of the **test system**

<http://www.aster.or.jp/workshops/insta2015/>

Two kinds of scope: **test system architecture** and **test suite architecture**

- **Test system architecture** is for test system, system/software to be tested (SUT), platform where SUT is executed, generator of test cases etc (see UTP)
- **Test suite architecture** is for test suite inside, which mainly consists of test levels, test types, test conditions, test cycles, groups of test cases, etc.

A good **test architecture** and design for testability is a precondition/enabler/driver for effective and cost-efficient test automation that is easy to maintain and evolve

A **test architecture** is needed to design and manage the quality attributes (NFRs) of a test system

Test architecture in ISTQB Advanced Level Specialist Syllabus for Test Automation Engineer

ISTQB Glossary (<http://www.istqb.org/>)

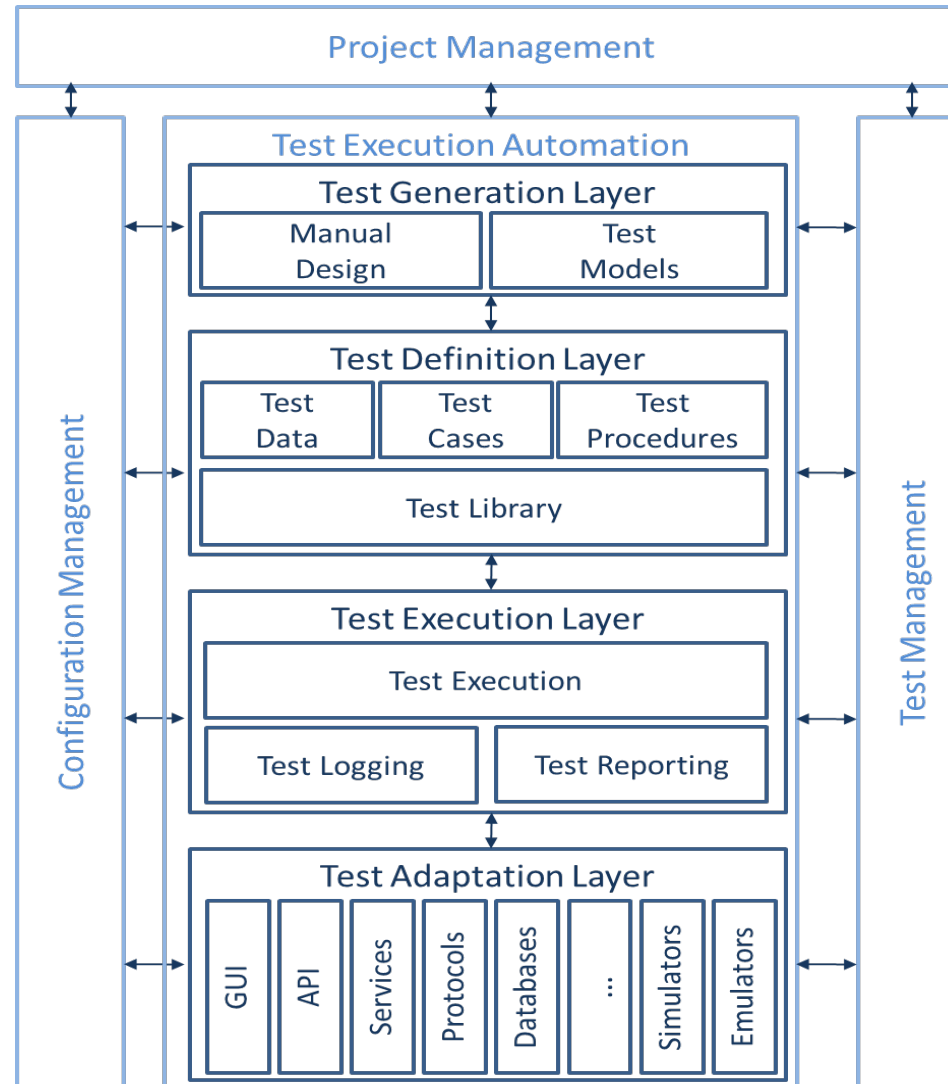
Generic test automation architecture – Representation of the layers, components, and interfaces of a test automation architecture, allowing for a structured and modular approach to implement test automation.

Test automation architecture – An instantiation of the generic test automation architecture to define the architecture of a test automation solution, i.e., its layers, components, services and interfaces.

Test automation engineer – A person who is responsible for the design, implementation and maintenance of a test automation architecture as well as the technical evolution of the resulting test automation solution.

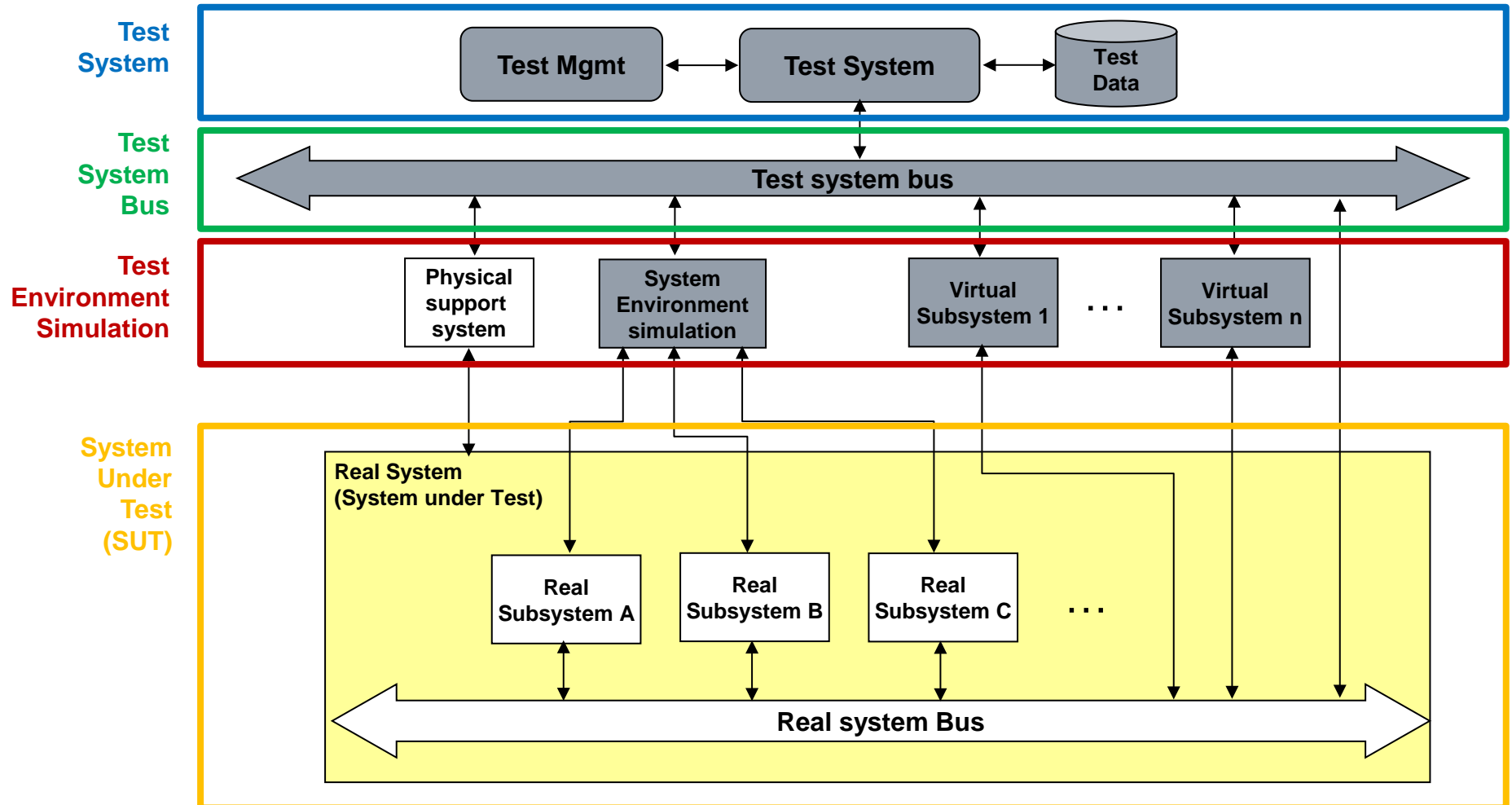
Test automation solution – A realization/implementation of a test automation architecture, i.e., a combination of components implementing a specific test automation assignment. The components may include commercial off-the-shelf test tools, test automation frameworks, as well as test hardware.

ISTQB Generic test automation architecture (gTAA)

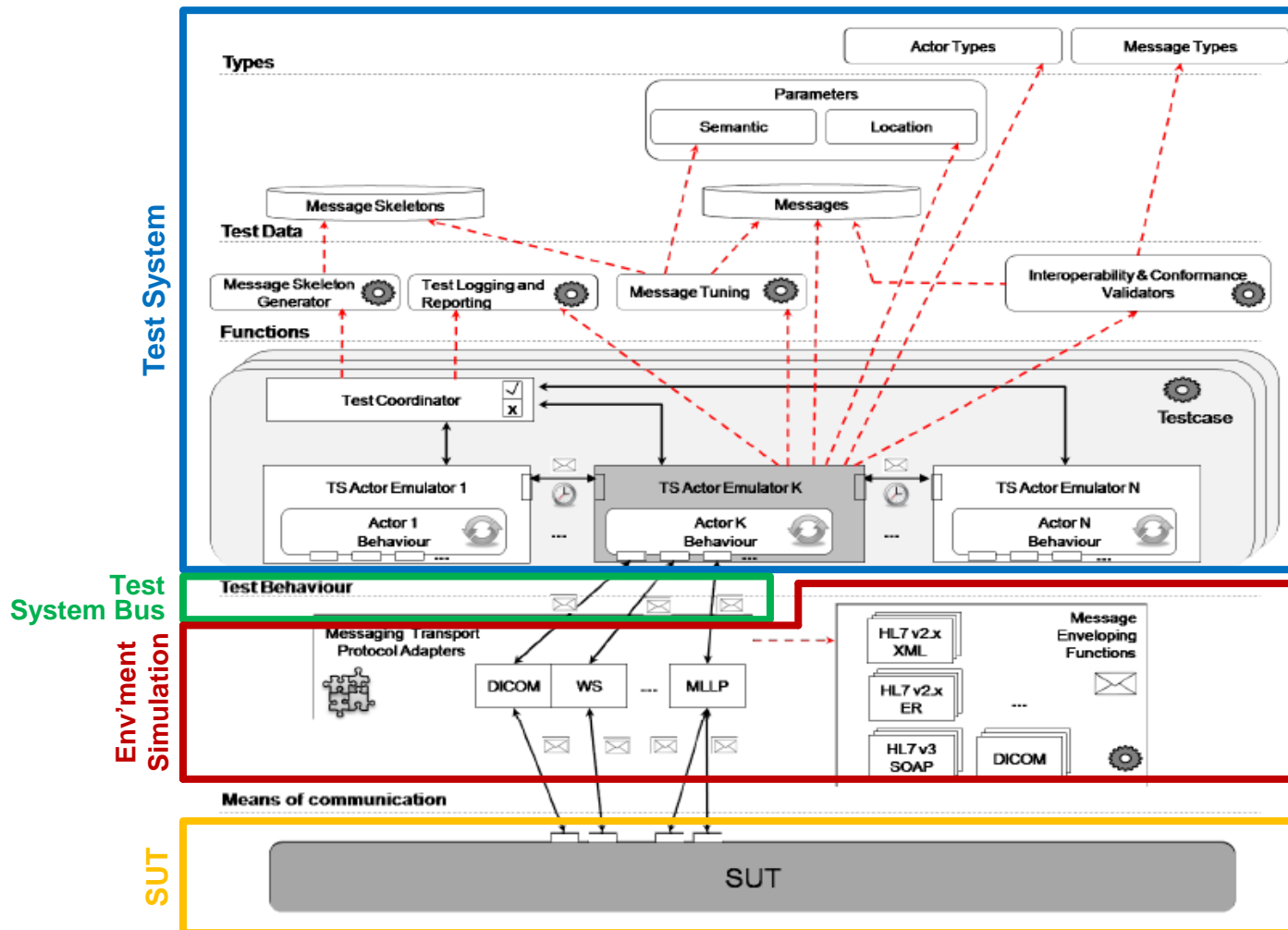


<http://www.istqb.org/>

Test architecture for system integration testing

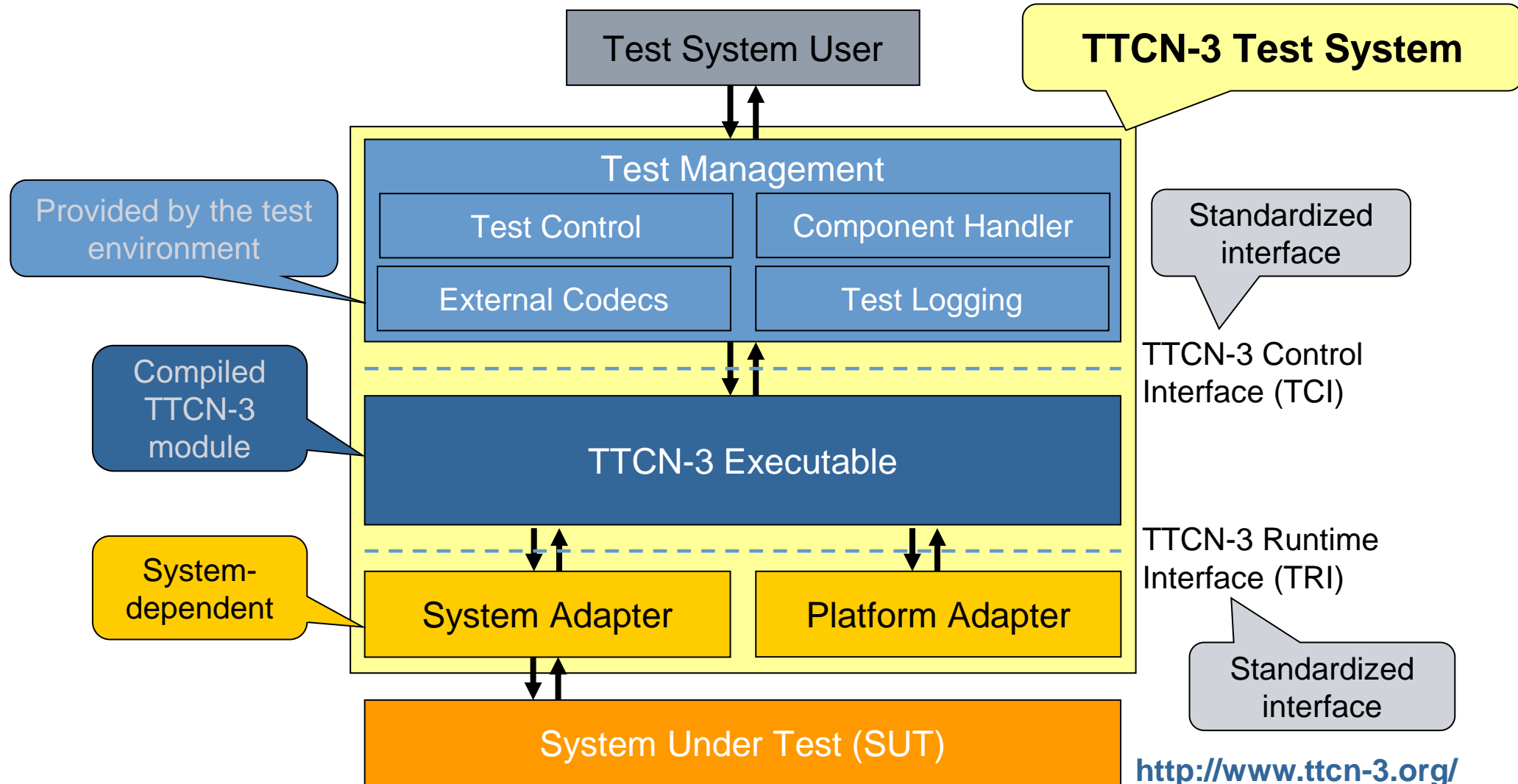


Complex example: Healthcare domain

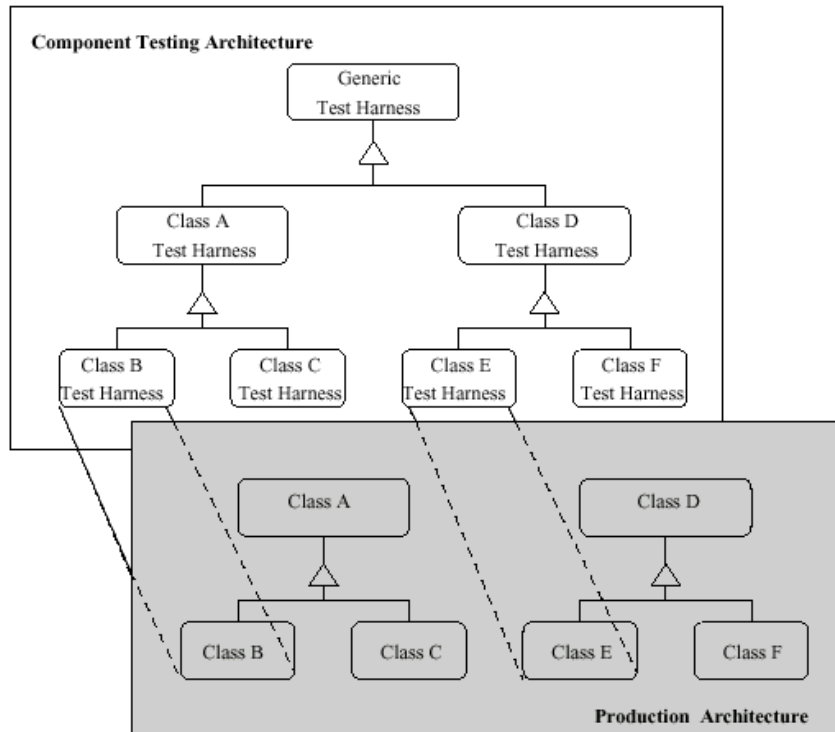


See also:
Diana Vega: A Methodology for Automated Interoperability Testing of Healthcare Information Systems based on an Actor Emulation Approach, TU Berlin, Fakultät Elektrotechnik und Informatik, submitted PhD.

TTCN-3 test architecture

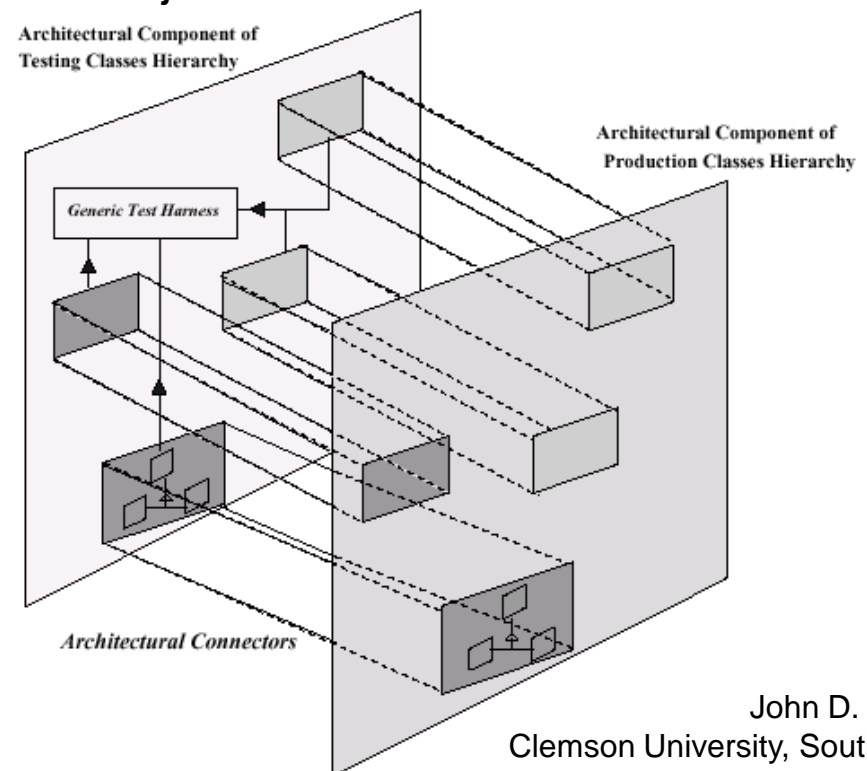


Parallel Architecture for Component Testing (PACT)



Intent

Implement a driver as a class hierarchy that is symmetric to the hierarchy of the class under test. Each driver class sends a test message to an object of the class under test

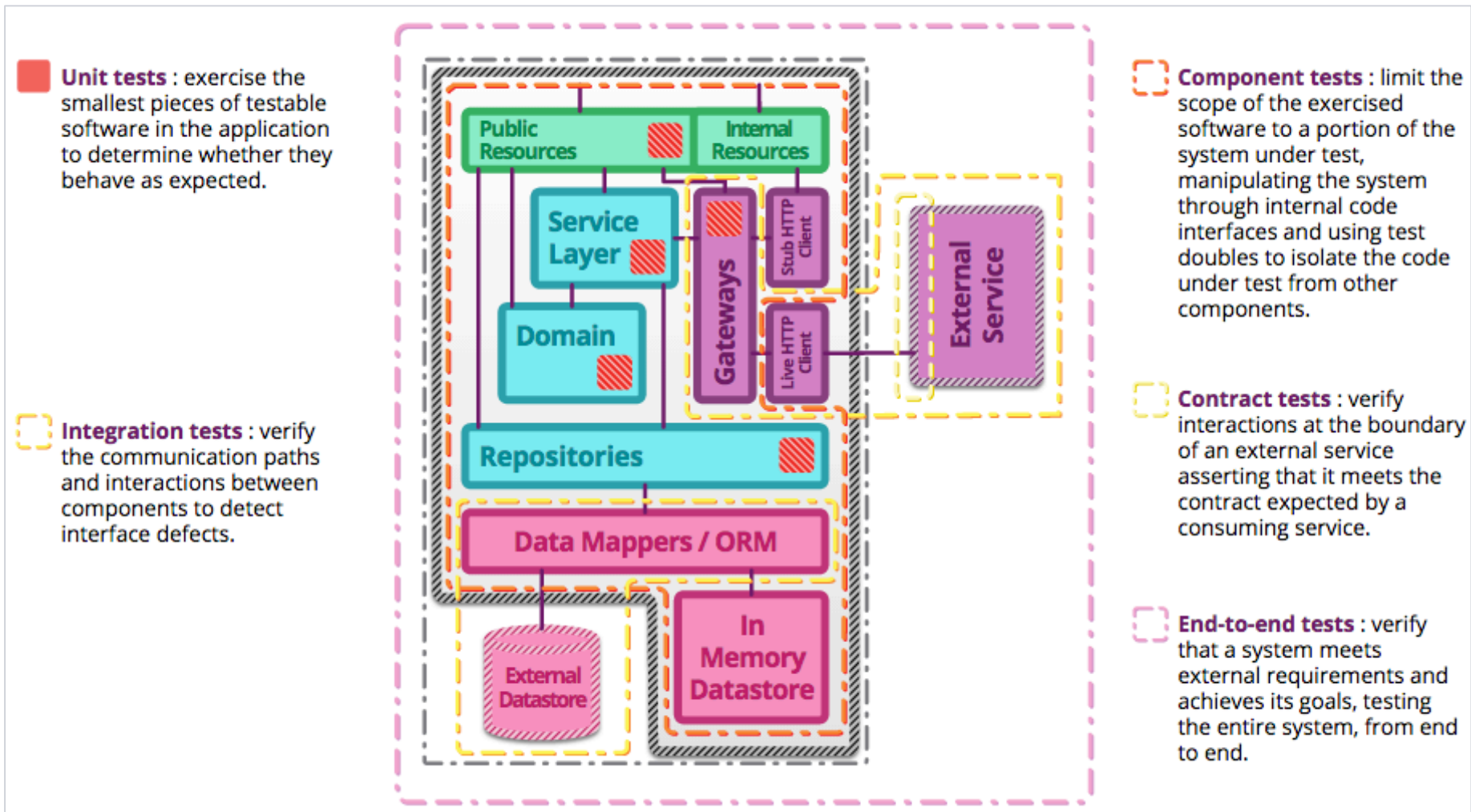


Applicability

This approach is useful at class and cluster scope for unit and integration testing of an application class hierarchy

John D. McGregor
Clemson University, South Carolina

Testing strategies in a microservice architecture



Toby Clemson, ThoughtWorks: Testing Strategies in a Microservice Architecture
<https://martinfowler.com/articles/microservice-testing/>



TIA Portal – Multi user feature – Non-deterministic testing

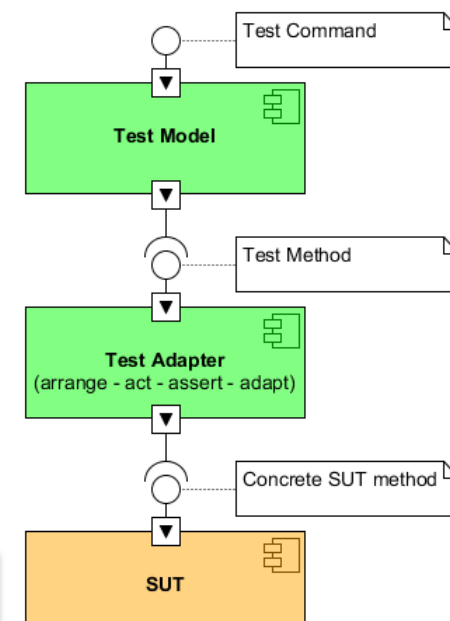
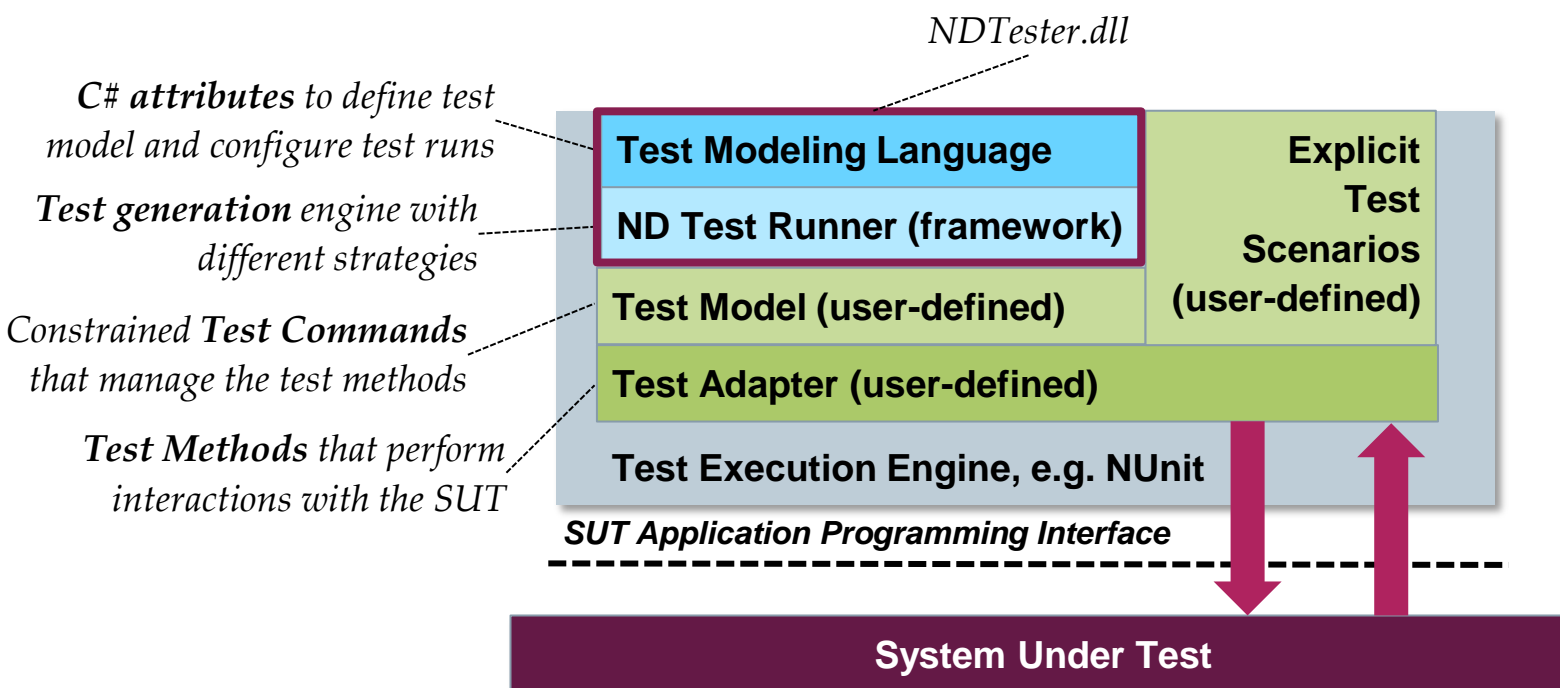
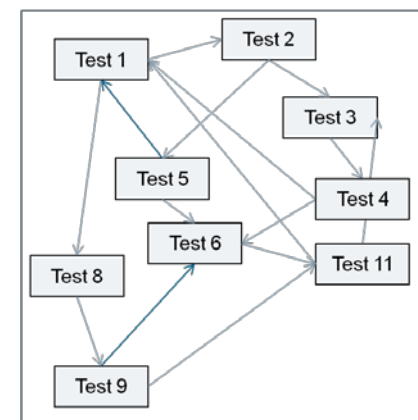
Test architecture for testing of multi user feature

ND tester (Non-Deterministic) for endurance testing

- Reducing the risk of failing system integration
- Supporting API testing in .NET through test generation



NDTester



Test Environment

Agenda

Basics

Test Architecture

Specification and Design

Supporting Technologies

Summary

UML Testing Profile (UTP) – Overview

Based on UML (Unified Modeling Language) 2.0

- Meta model provided in the UML superstructure volume
- Principle of UML profiling defined in the UML infrastructure volume



Domain-independent test modeling language for designing, visualizing, specifying, analyzing, constructing, and documenting the artifacts of test systems

UTP

V1.0 2005

V1.1 2012

V1.2 2013

V2.0 2017

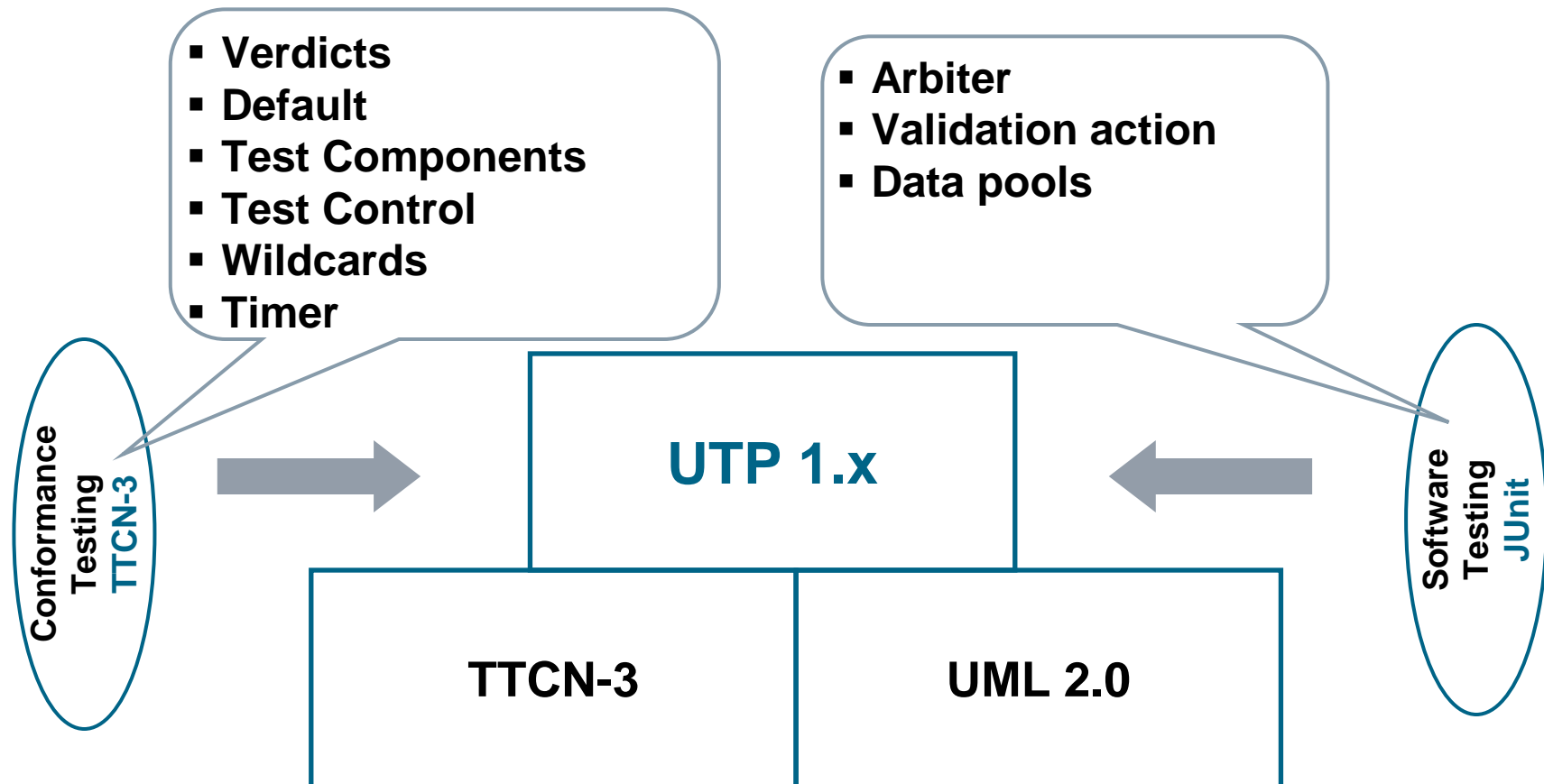
Test modeling language that extends UML with specific concepts needed for testing

- Test architecture – structural aspects, architectural foundation
- Test behavior – dynamic aspects, test-specific actions for test case behavior
- Test data – concepts to define data partitions, data pools, and wildcards
- Timer concepts – imperative timer mechanism
- Test management – concepts for model-based test management

<http://www.omg.org/spec/UTP/>
<http://utp.omg.org/>
<http://utp.zen-tools.com/>

Mapping to test execution frameworks (included: JUnit, TTCN-3), integrated in Eclipse TPTP (<https://projects.eclipse.org/projects/tptp.platform>)

UTP 1.x Roots





UTP 2 – New version 2.0 in 2017

Conceptual and terminological fundamentals of UTP 2

BS 7925-1 & 2		JUnit		
IEEE:829		TTCN-3		
ISO 29119	ISTQB	UTP 1	ES 202 951	UML
UTP 2				
Test Planning	Test Analysis & Design	Test Implementation & Execution	Test Evaluation	
Test Context	Test Case	Test Execution Schedule	Test Log	
Test Set	Test Design Strategy	Arbitration Specification	Test Log Structure	
Test Objective	Test Design Model	Test Component	Verdict	
Test Requirement	Test Configuration	Test Item		
Test Type	Test Data Specification			
Test Level	Data Values			
	Data Pools			

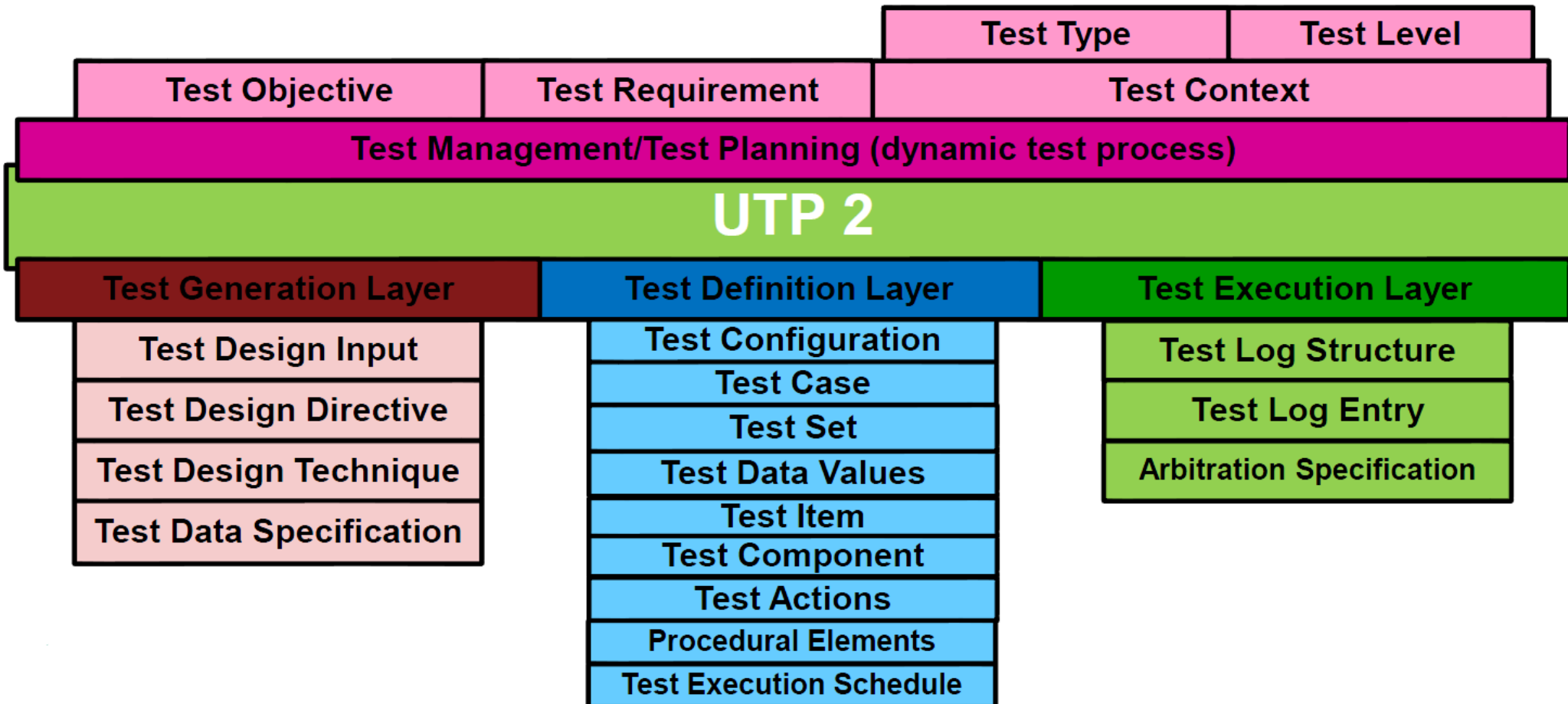
Terminological origins

Concepts

 Terminological origins
 Concepts

UTP 2 – New version 2.0 in 2017

Conceptual overview of UTP 2



UTP Test architecture

TestContext

- Test case grouping, test configuration definition, and test control specification

TestComponent

- Intended to stimulate the SUT and to evaluate its (expected) outcome

SUT

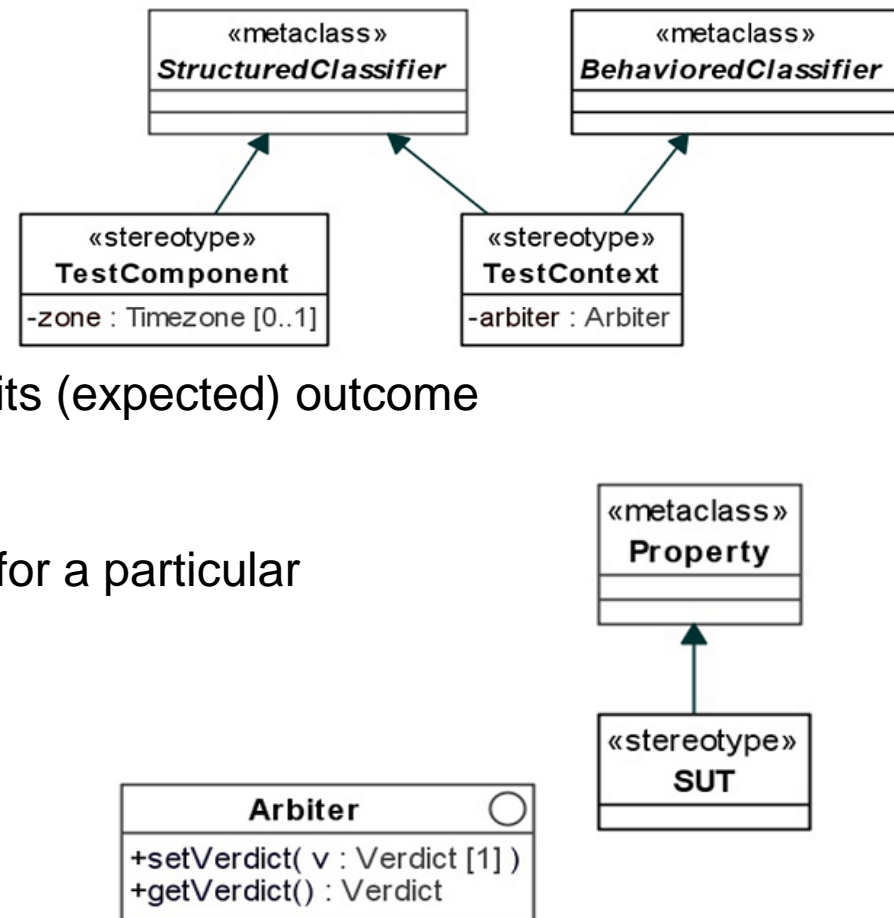
- System Under Test, represents the test object for a particular test context

Arbiter

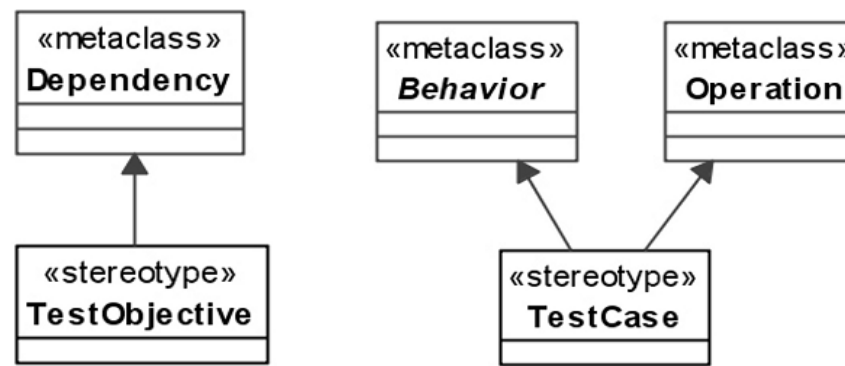
- Predefined interface for verdict calculation and assignment

Scheduler

- A property of a test context to control the execution of different test components



UTP Test behavior



TestObjective

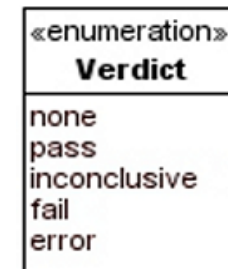
- Describes the purpose for realization and execution of a test case

TestCase

- A specification of operations or behaviors to test the SUT; comprises test steps that represent the interaction between test components and SUT

Verdict

- Represents the final conclusion of a test case:
none < pass < inconclusive < fail < error

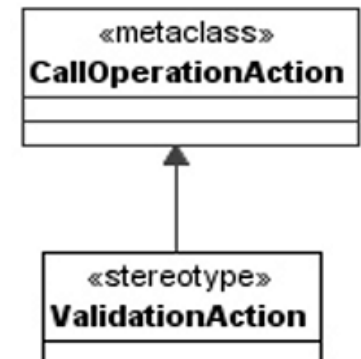


TestActions

- ValidationAction, LogAction, FinishAction, DetermAlt

Defaults and Logs

- Default behavior, TestLog



UTP Test data

DataPartition

- Definition of logical partitions of test-relevant data types

DataPool

- Specification of data tables that can be used for repeated test case execution

CodingRule

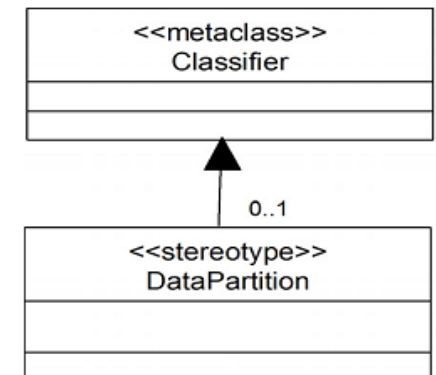
- Defines how values of messages shall be decoded / encoded for communication between the SUT and test components

LiteralAny / LiteralAnyOrNull

- Express wildcards (?, *) for concrete test data instances

DataSelector

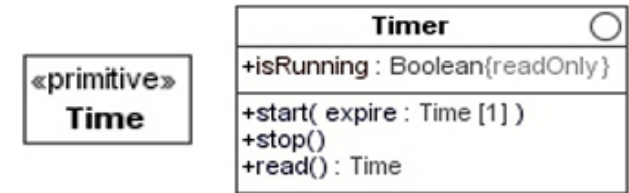
- Defines how data values are selected from data pool



UTP Timer concepts

Time

- Primitive type for an abstract description of time units within test case executions



Timer

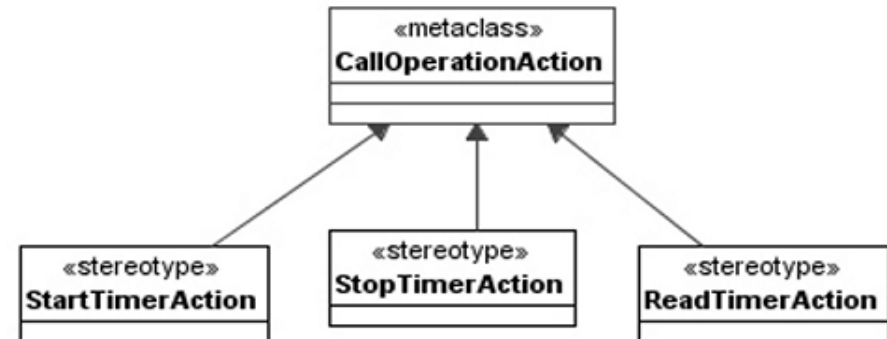
- Predefined interface that represents the timer handling interface of the test execution environment; responsible to manage the creation and expiration of timers used by test components within a test case

Start- / Stop- / ReadTimerActions

- Used in the test case execution to call the operations of the timer interface

TimeZone

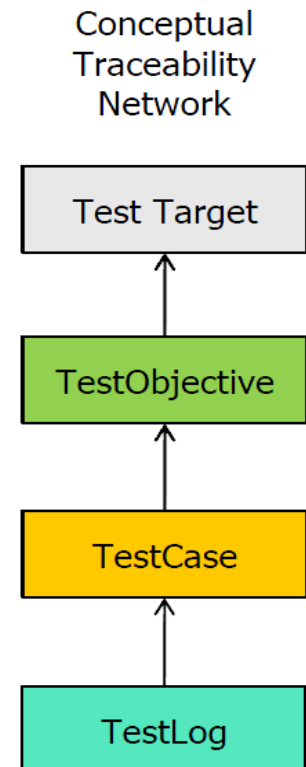
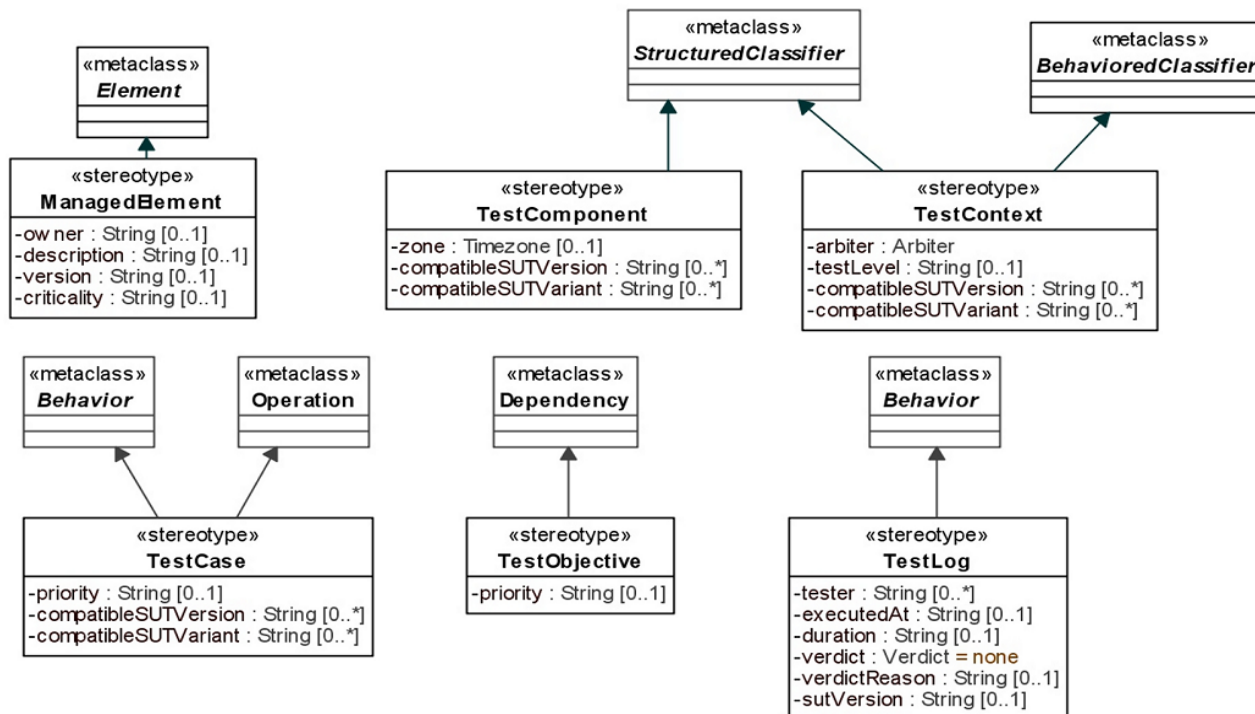
- Grouping mechanism for test components; each test component belong to a certain time zone



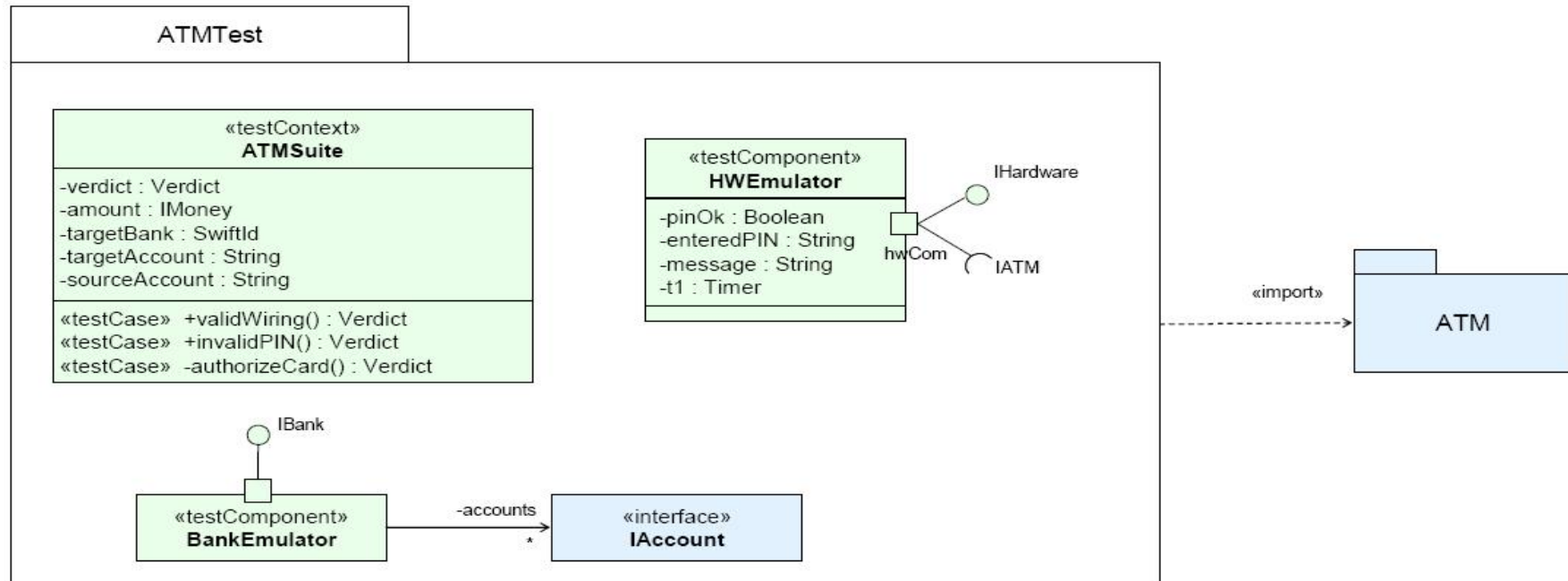
UTP Test management

Foster better integration of UTP models and test management

- Stereotype **ManagedElement**
- Add test-specific attributes into existing concepts and enable them for test management purposes



UTP Example – ATMTTest package



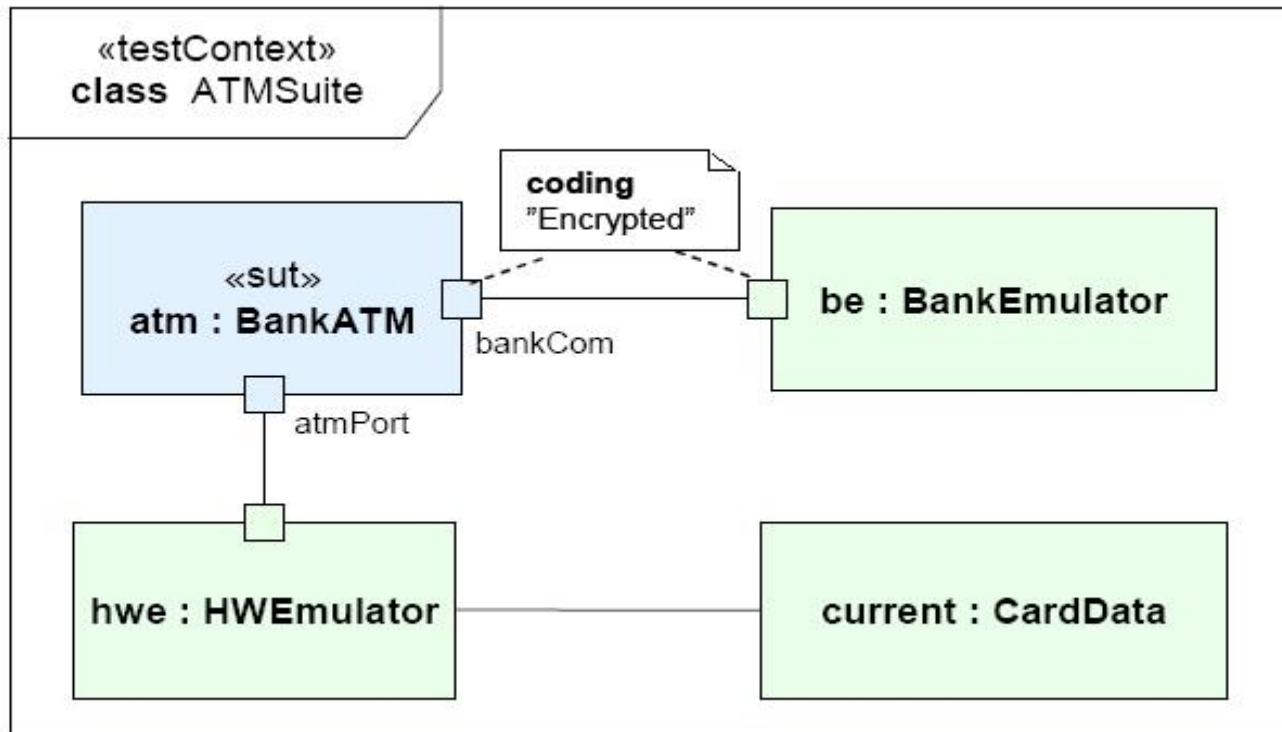
ATMTTest imports the ATM package to get access to the elements to be tested.

ATMTTest consists of one test context, ATMSuite, and two test components, BankEmulator and HWEEmulator.

ATMSuite has three test cases: validWiring(), invalidPIN() and authorizeCard(). Two have public visibility and one private.

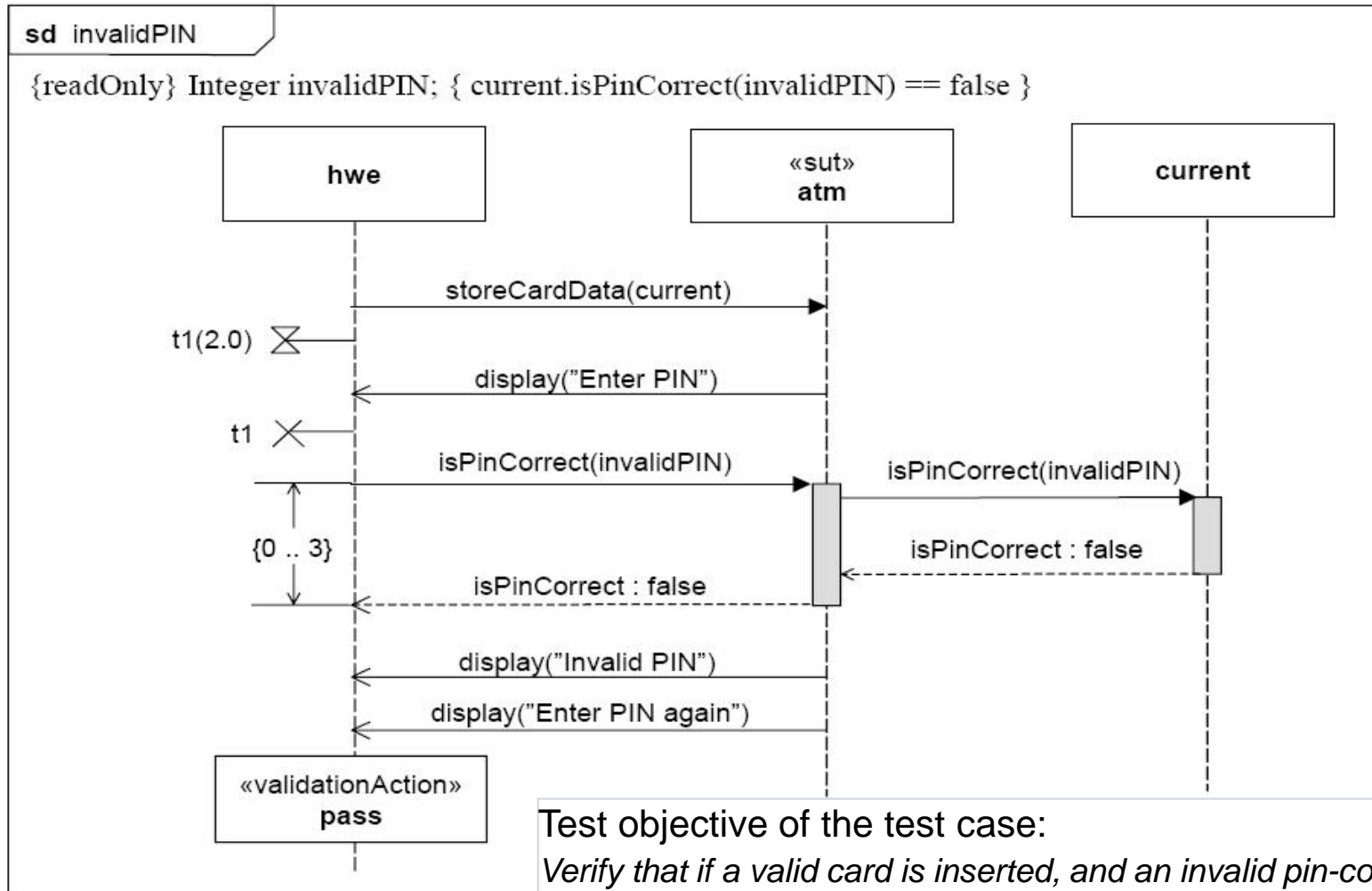
The test components implements the interfaces of the HWEEmulator and BankNetwork packages and will serve as emulators for these packages.

ATMSuite test context

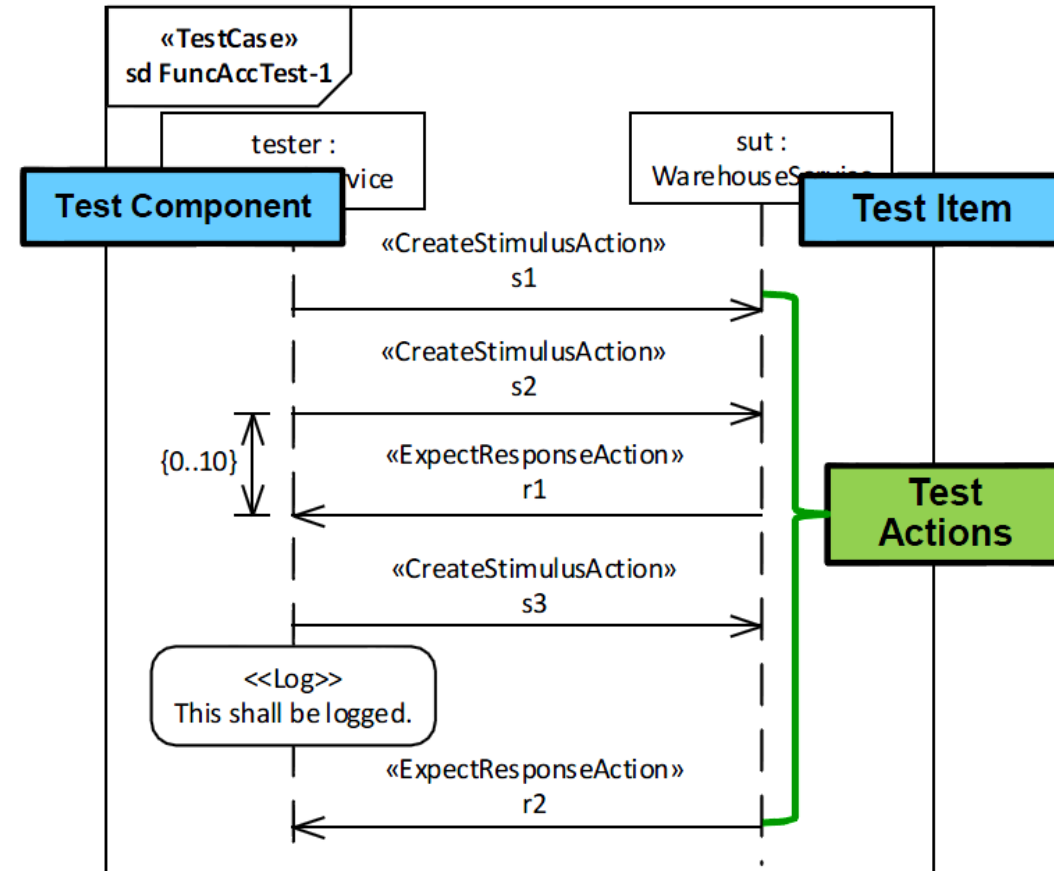
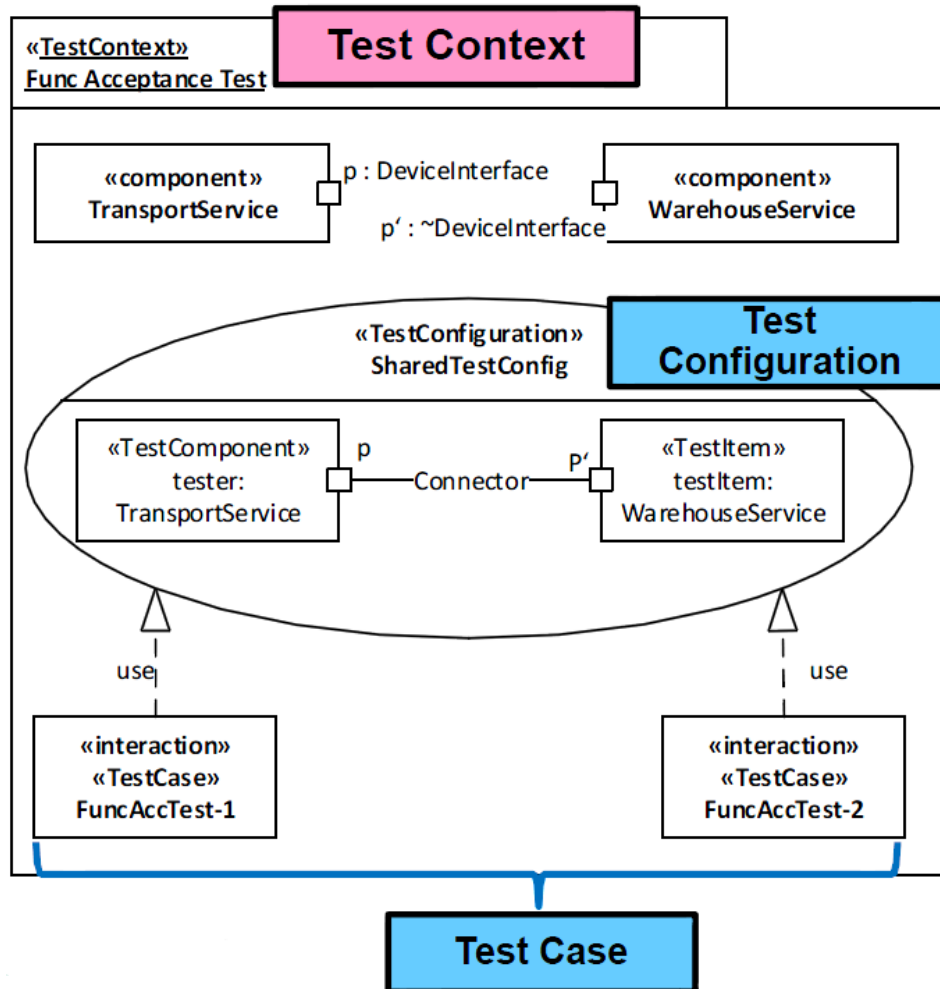


The test configuration (i.e., the composite structure of the test context) specifies how the SUT, a number of test components and one utility part are used in a particular test context. Ports and connectors are used to specify possible ways of communication. Each test configuration must consist of at least one SUT.

Behavior of the *invalidPIN* test case



Example UTP 2.0



Test Environment

Agenda

Basics

Test Architecture

Specification and Design

Supporting Technologies

Summary

Supporting technologies

Different types of substitutes and placeholders can be used to effectively and efficiently set up and create test environments

Virtualization – hardware, OS, services (application behavior)

- Creating a virtual (rather than actual) version of something
 - **Virtual machines**
Hardware/platform acting like a real computer with an operating system
 - **Service virtualization**
Emulation of the behavior of software components
 - **Virtual assets**
Asset simulating the behavior of a real component (service virtualization asset)
 - **Virtual integration**
Replacement of physical components with information for integration purposes

Simulation

- Mimic behavior of a real component

X-in-the-Loop

- Simulation at different abstraction levels in combination with real tests

Test Double

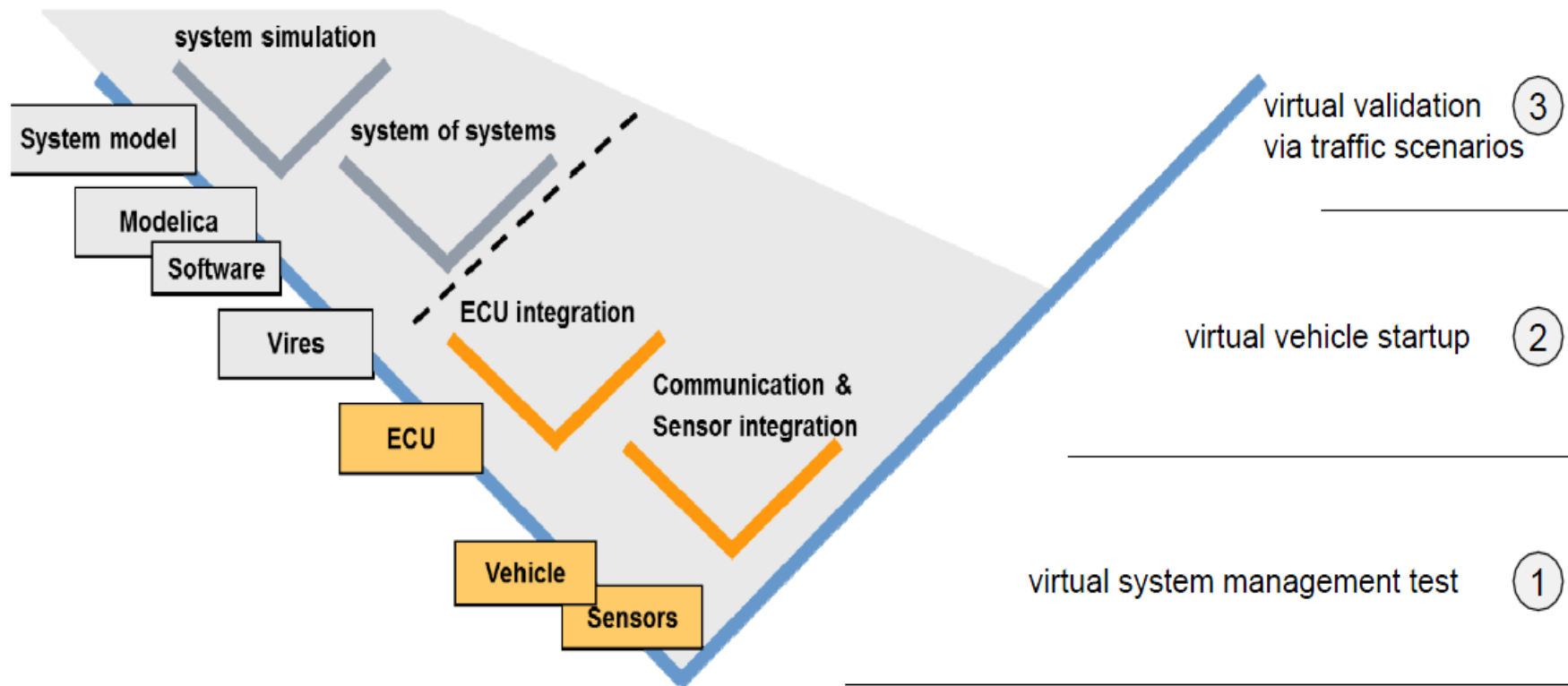
- Substitute object for dependent resources (especially for unit and integration testing)

Cloud technologies

Example: eCar

Virtual system integration and test

Close and early interaction between electronics, mechanics and software



Example: TrainSIM

Lab test environment for end-to-end tests of the ICx platform

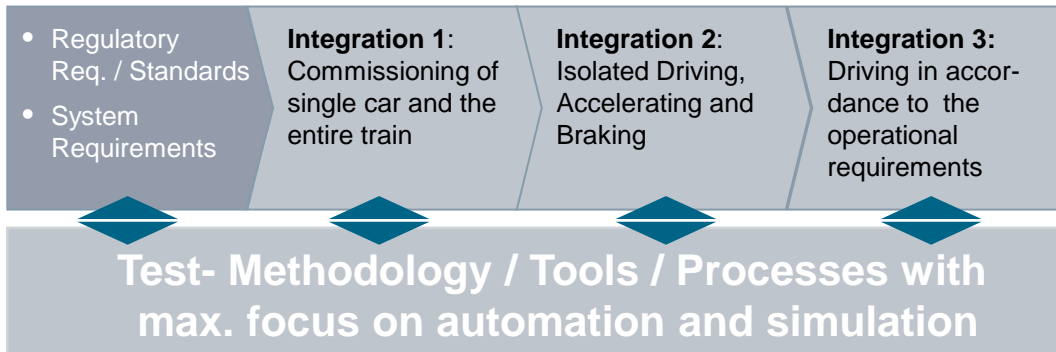


The design involves a modular environment for simulation & test

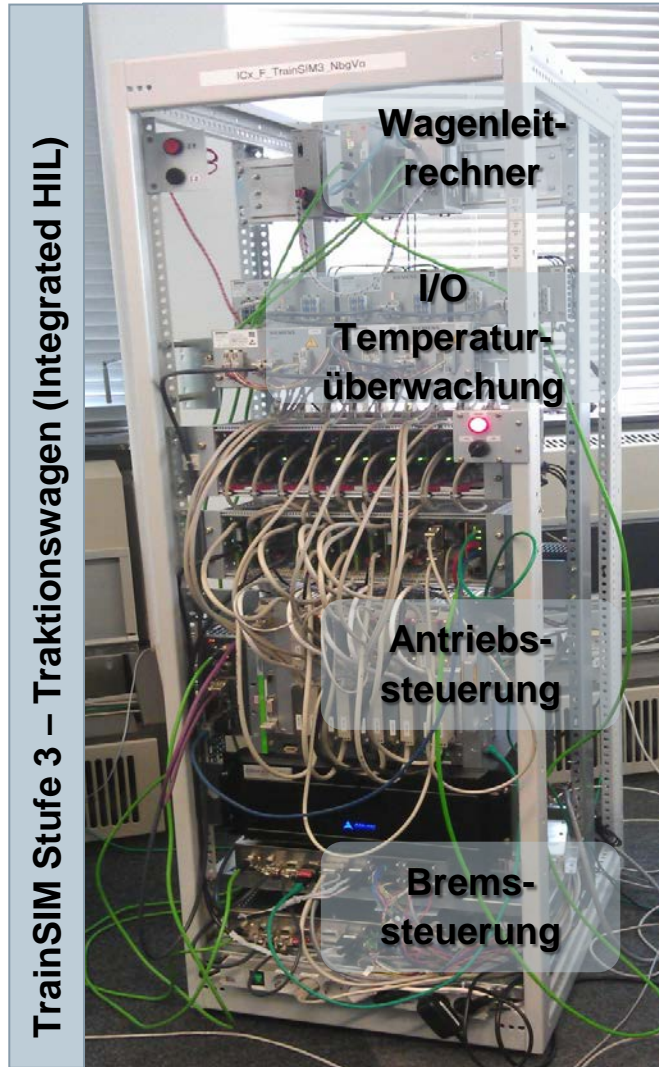
- A 12-car train
- Use of real control systems with genuine software and circuitry
- Provision of all relevant communication infrastructure
- Must have: Model-based specification and Test Automation
- Staged integration approach (from single car to a whole train integration)

The major goals are

- Test of all parameters and functions in the test lab
- Verification of the functional E-2-E view
- Examining the precise reaction of the system in the case of fault-injections
- Run-time behavior



TrainSIM – Lab



TrainSIM Stufe 4 (Level “Zug”)

16 Racks

2 Driver desks

Simulation :

>3.000 signals

(Brakes: 12x160 +

Acceleration Unit: 6x180)

Driver desk

- all user elements integrated
- used for manual as well as automated testing

Wiring Simulation

consists of :

- 155 standard components
 - 350 complex components
- Up to 58.000 components in real system and TrainSIM

Increasing reality using “in the loop simulation”

“in the loop simulations” are used in the context of control systems development, where a control SW (“controller”) interacts with a mechatronic system (“plant”)

Model in the Loop (MiL)

- using a model of the control to work with a model of the plant
- extremely fast development; possible to make small changes to the control model and immediately test the system.

Software in the Loop (SiL)

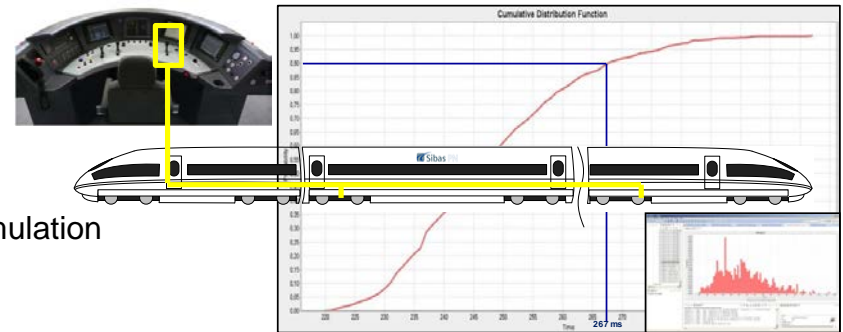
- coded control SW in simulation environment connected to plant simulation
- coding failures or issues start to become evident
- design iteration slows down slightly from MiL

Processor in the Loop (PiL)

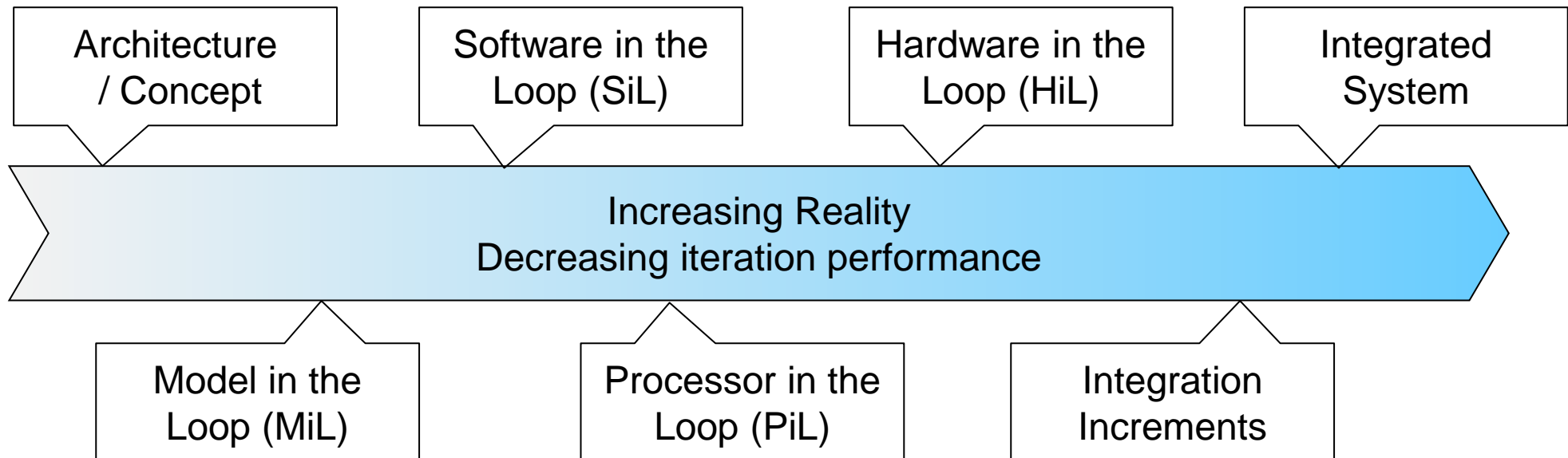
- controller SW deployed to a representative microprocessor, coupled via a high speed bus with plant simulation
- execution issues on the embedded processor are easy to find and fix; e.g. sufficient performance of the embedded processor
- design iteration slows noticeably; changes must be coded and deployed to the control system;

Hardware in the Loop (HiL)

- real control system connected with plant model, running on a real-time computer, through proper controller IO
- often used only for software validation rather than development as the design iteration is very slow at this point; however, this test is closest to the final application and therefore exposes most of the problems that will be seen.
- fidelity of the plant model and the test vectors determine the only difference between the final application and the HiL environment



Increasing reality using “in the loop simulation”



Test environment simulation tools – Examples

Parasoft Service Virtualization

- Simulation of complete, realistic test environments
- Mimic of special scenarios (security, performance, reliability, failover)
- Modeling, monitoring and analyzing of system behavior

WindRiver Simics

- Simulation of a full target system (from a single processor to large, complex, connected systems)
- Allows fault injection and change of network reachability

Fraunhofer Fokus V2X test system

- Hybrid approach of real devices and simulated environments
- Simulation, impairment and monitoring on all levels
- TTCN3 as test description language
- Test control with TTworkbench

Test doubles

Gerard Meszaros has coined the term ***test double***

Double

- Any kind of pretend object used in place of a real object for testing purposes

Dummy

- Object that is passed around but never actually used; usually just to fill parameter lists

Stub

- Object that provides canned answers to calls made during the test

Spy

- Stub used to maintain state

Fake

- Object that has working implementations, but usually takes some shortcut which makes it not suitable for production

Mock

- Object pre-programmed with expectations which form a specification of the calls it is expected to receive

Gerard Meszaros: xUnit Test Patterns: Refactoring Test Code, Addison-Wesley, 2007
<http://xunitpatterns.com/>



When to use mocks

Classical way: Use real objects if possible

- Mocking if using real objects is awkward or if real objects cannot be used
- Useful to verify that a method with a void return is called
- Focus on the result of the behavior
- Cost/benefit balance

Mockist way: Always use a mock for any object

- Focus on how unit/component is implemented to write expectations
- Problem: strong coupling to the implementation
- Too powerful and overkill in many situations

Test Environment

Agenda

Basics

Test Architecture

Specification and Design

Supporting Technologies

Summary

What we have learned

No test can be executed without a test environment; a robust, stable and reliable test environment is essential.

A test environment consists of distinct areas and therefore requires a multi-disciplinary effort.

The test environment can be as complex as the SUT.

Development needs also apply to the test environment: requirements, architecture, effort estimation, quality, ...


Tools and techniques available to speed up test setup.



Further readings

Use the SSA Wiki :
<https://wiki.ct.siemens.de/x/fReTBQ>

and check the “Reading recommendations”:
<https://wiki.ct.siemens.de/x/-pRgBg>

- 
- **Architect's Resources:**
 - Competence related content
 - Technology related content
 - Design Essays
 - Collection of How-To articles
 - Tools and Templates
 - Reading recommendations
 - Job Profiles for architects
 - External Trainings
 - ... more resources