Automated Integration Testing in Agile Environments

Slobodanka Sersik, InfoDesign OSD GmbH
Dr. Gerald Schröder, InfoDesign OSD GmbH

Speakers

- senior software developers and consultants at InfoDesign OSD GmbH
  infodesigner.biz
- architects and developers of the Open Source testing framework iValidator
  ivalidator.org

Contact

- Slobodanka.Sersik [at] infodesigner.biz
- Gerald.Schroeder [at] infodesigner.biz

Contents of this talk

→ Why integration testing?

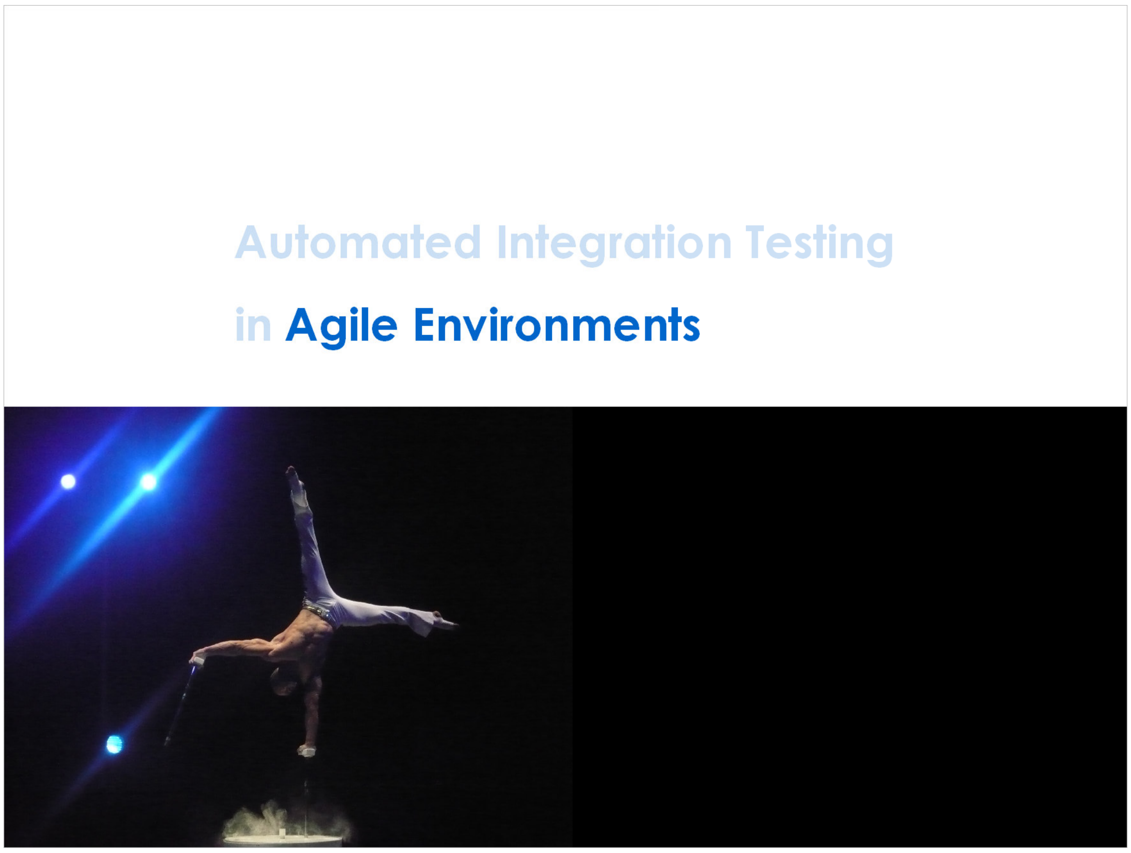
Automated Integration Testing in Agile Environments

Why integration testing?

- Test Driven Development (TDD) of each component – yes, of course, it's the base.
  - We are finding basic programming errors in isolated components.
- Then we integrate the components … but they do not work together.
  - Integration errors!
- For example:
  - Use case scenarios that are breaking in between.
  - Component APIs not used correctly (wrong parameters).
  - Calling sequences that do not adher to protocols.
  - Timing problems.
- Note: We do not speak here about acceptance testing …

  It must be done anyway at the end of each iteration by the user.
- How do we find these integration errors?
  - By integration testing!

→ What is special about integration testing in agile environments?



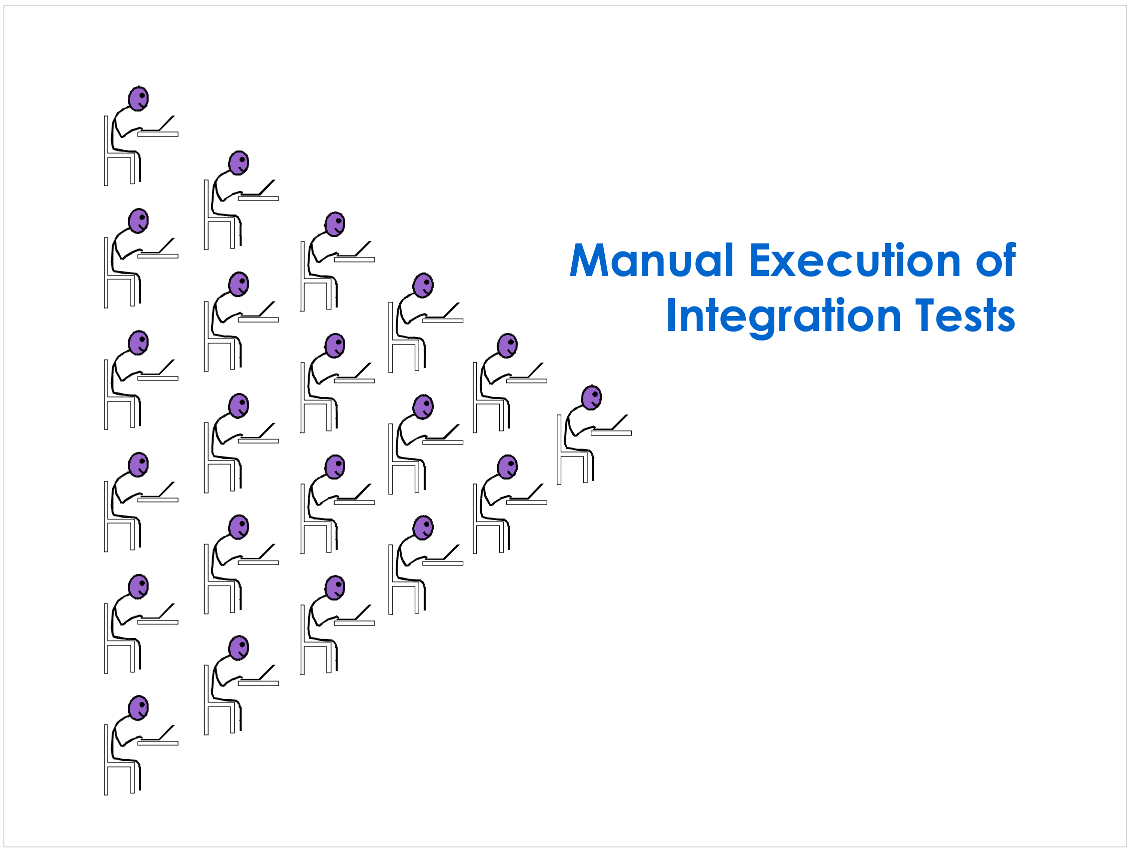
Automated Integration Testing in Agile Environments

What is special about integration testing in agile environments?

Agile projects

- embrace change – of API, behaviour, user interface:
    - Integrations tests have to change also!
- release often:
    - Integrations tests release in the same velocity – and are repeated often.
- deliver user stories – not functionality developed against a specification:
    - Integration tests are built against user stories!
- do not contain a testing phase:
    - Integration tests run while developing – repeatedly.

→ How can we execute integration tests?



Manual Execution of Integration Tests

How do we execute integration tests?

Manually

- Adaptable to the changes and new features - even though not always easy.
- At the beginning the system is small and we need only few testers.

  But the number of tests increase each iteration.
  In each iteration all tests have to be repeated.
  So we need many testers and lots of time …
  … and we are still not sure if it will work at the end of the iteration.

- Manual testing takes time – the development does not stop.
- Manual testers take shortcuts, do not prove all exceptional cases, make errors.

→ It seems there is not enough time ...

“Time is a precious resource, and is the only commodity that, from day to day, people are equally endowed with.”

by Barrie Pearson

There is not enough time …

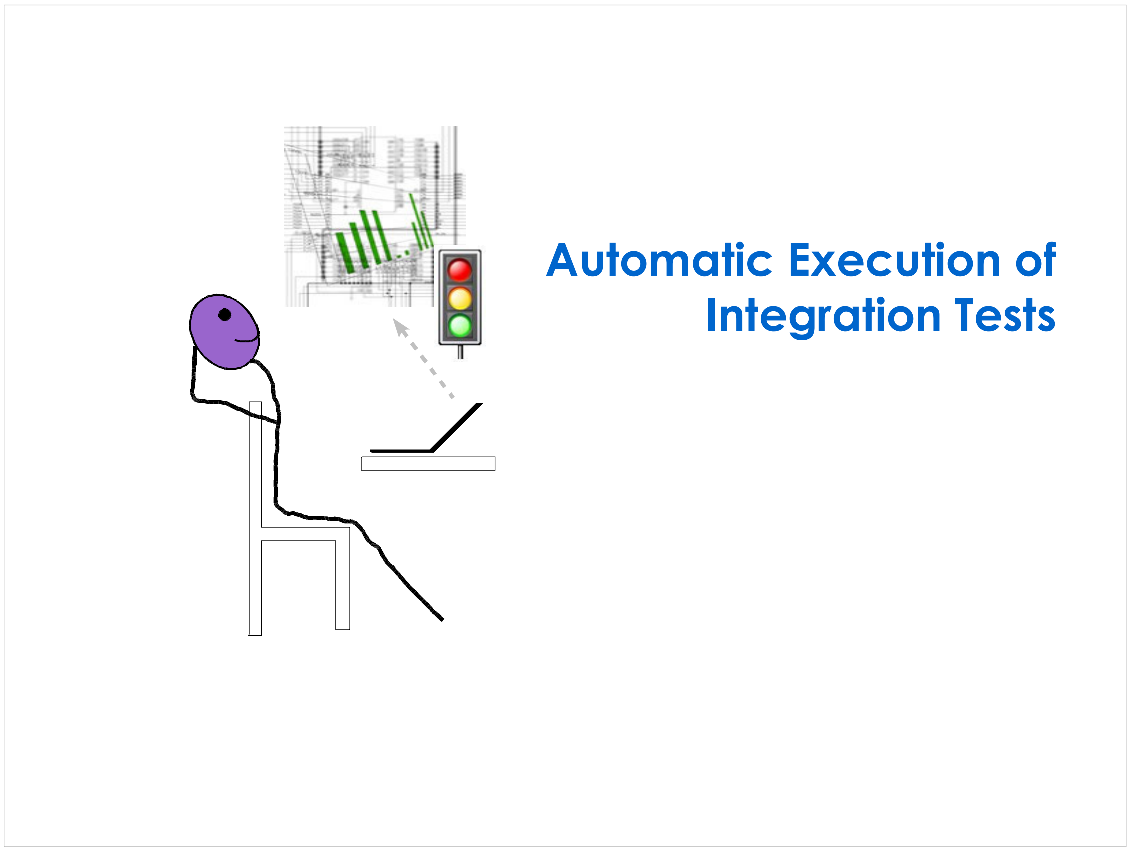
How often have you heard:

- “It is impossible to go through all the test cases in this iteration!”
- “It is impossible to adapt to all changes and new features!”

However, there are agile projects where beside the time pressure and the ever-changing circumstances, integration testing is done!

How? When everybody has the same time?

→ Automate!



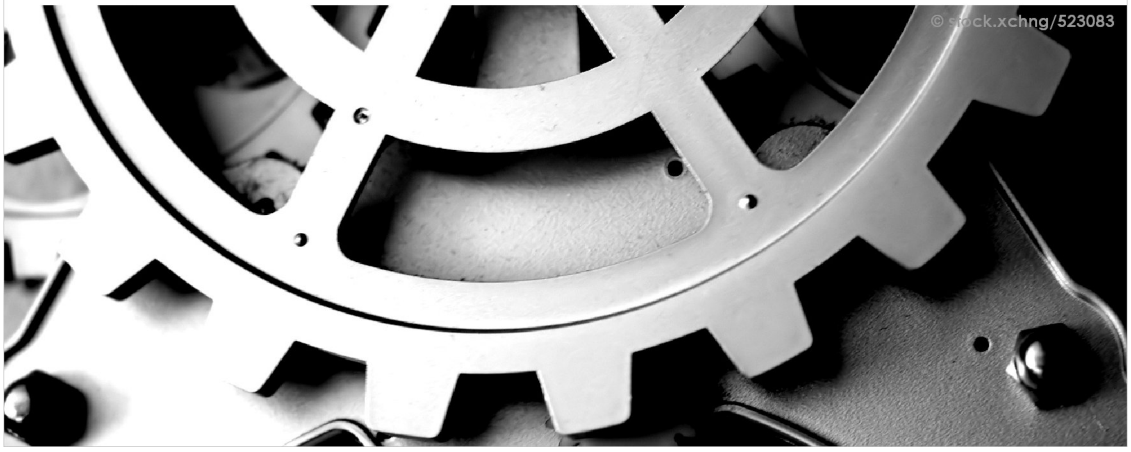
Automatic Execution of Integration Tests

Automate integration tests!

- Running with a night-build.
  - Not the incremental build – the tests are too slow.
- Showing each morning the release status.
  - They should not break the build – they show the status of the release.

→ How do we automate integration tests?

Automated Integration Testing in Agile Environments



How do we automate integration tests?

- Often integration test automation relies on GUI scripting simulating users.
- **But**: How do we automate integration tests for systems ...
  - that are highly automated themselves?
  - that are interfacing to other systems or machines without user interaction?
- **But**: How do we implement integration tests in parallel with development – or even ahead of it!
- **But**: How do we prepare for change – and minimize the effort to adapt the integration tests?

- During this talk we will show you an integration testing model we designed to specifically address these issues.

→ We developed this model during several projects.

Reference project

System: Container Terminal - Automatic Control and Container Management

Effort: > 200 person years

Technologies: Java, JMS, EJB, RMI, external systems, legacy systems

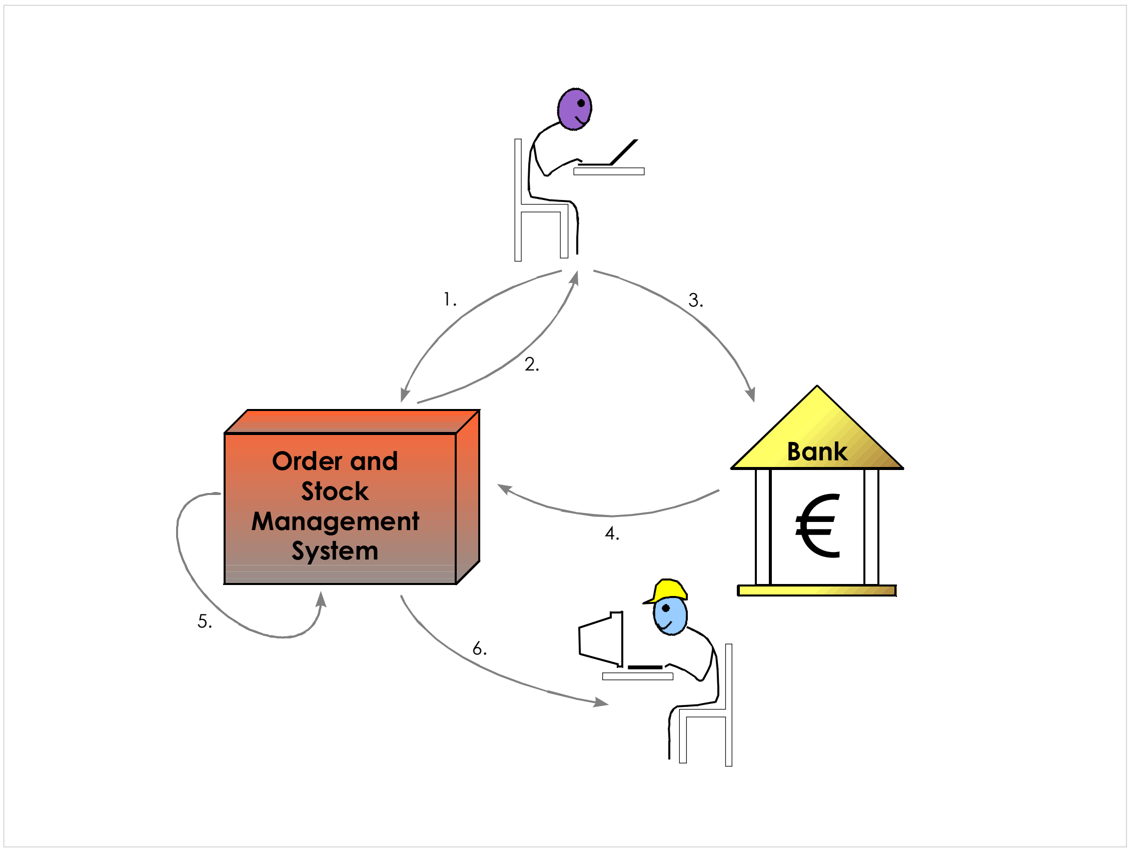
Container handling devices: cranes and yard vehicles

Numbers:

- Test scenarios: approx. 200
- Test steps per scenario: 5-30
- Reusable test steps: approx. 200
- Adapters: approx. 50
- Execution duration: 3 – 24 hours

This project is too big for an example.

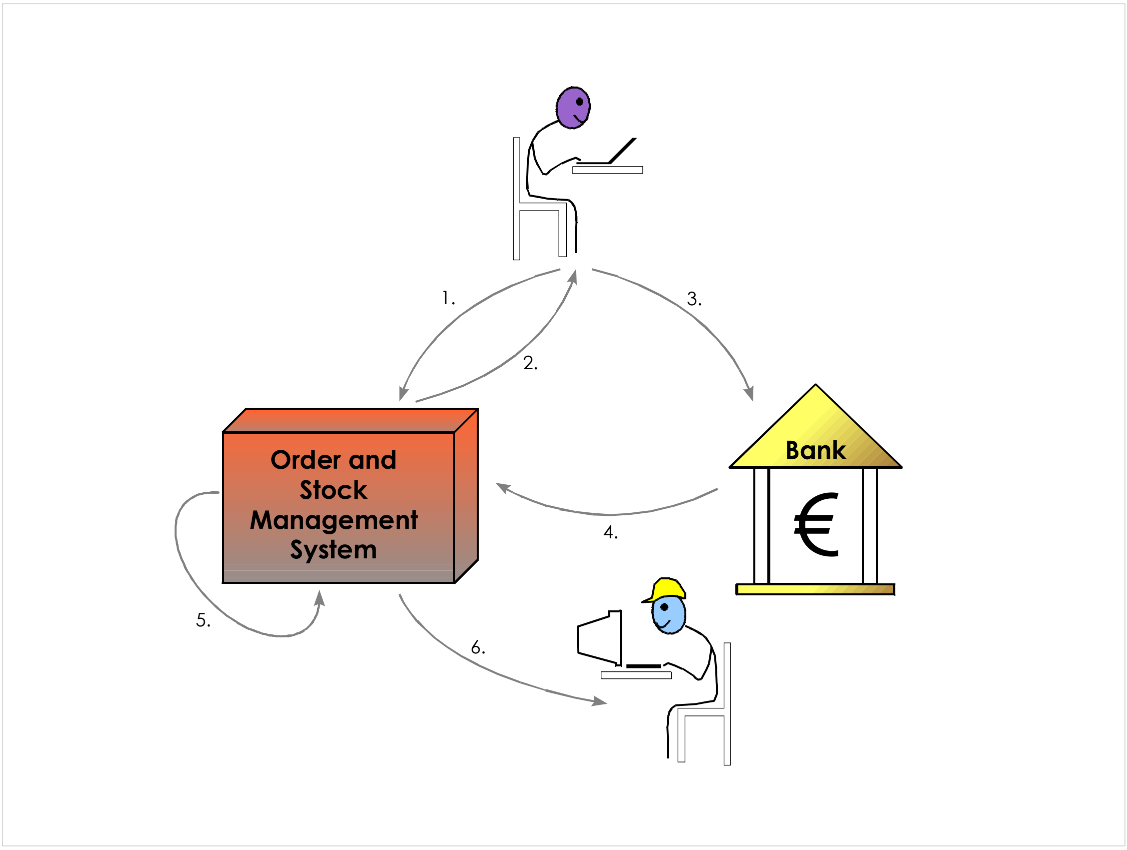
→ We have chosen a smaller example.



Example story:

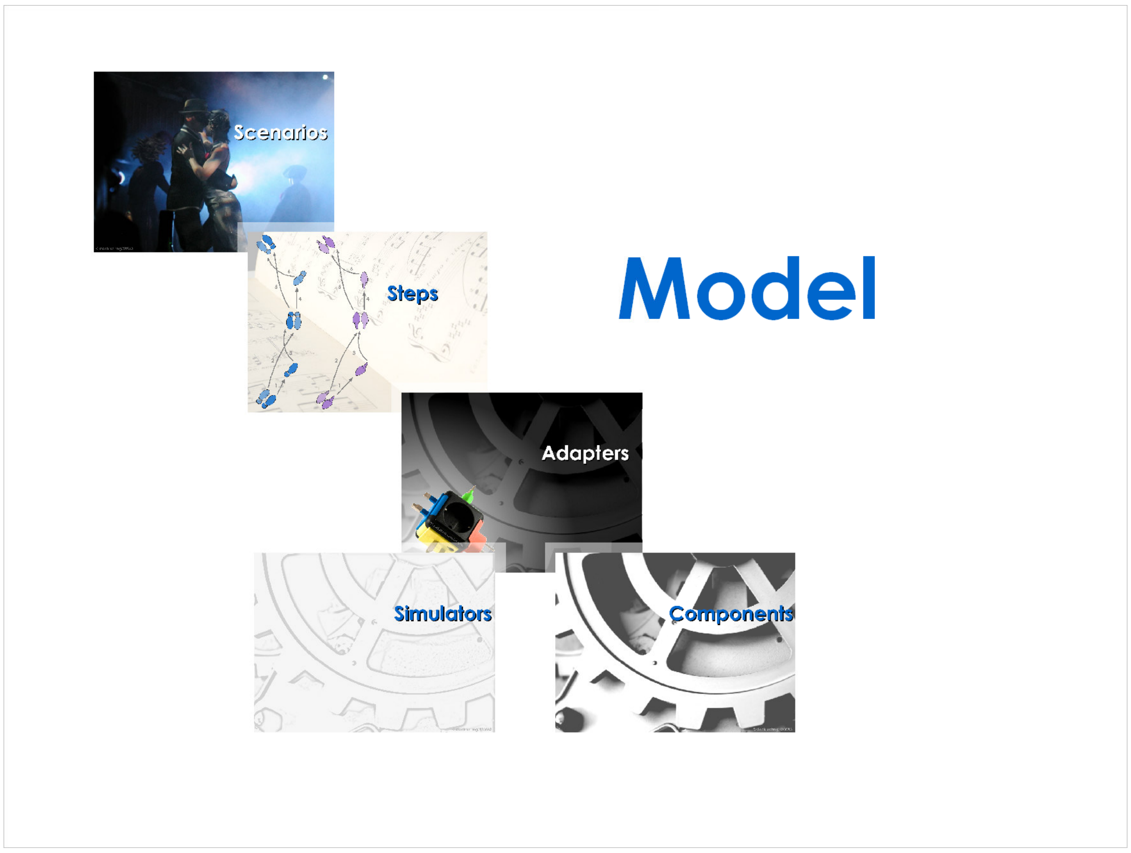
1) Customer places order containing goods and quantities via web front-end.
2) Order management system generates order reference number presented to customer.
3) Customer initiates bank transfer giving order reference number.
4) Bank sends bank transfers to order management system.
5) Order management system matches unpaid orders against bank transfers using order reference numbers.
6) For each paid order, a shipment order is being presented to the stock manager in his desktop application.

→ Where are the challenges when automating this scenario?
→ What do we have to implement when automating this scenario?



Where are the challenges when automating this scenario?

- Complex mocks of external systems that have to be controlled by a test execution engine (e.g. the bank)
- Integration tests have to be robust against technical changes (e.g. changing the protocol client-server)
- Integration tests do need parametrized reusability (e.g. same test steps with different test data)
- Integration tests made up of smaller steps, in order to reuse them in other constellations (e.g. step 1 and 2, than customer doesn't pay, and after some time receives a reminder from the system)
- Integration tests spread across multiple front-ends (e.g. (1) order system and (2) stock management system) - In case of a GUI test with a Capture&Replay tool, it might be problematic to test multiple front-ends.



Integration testing model

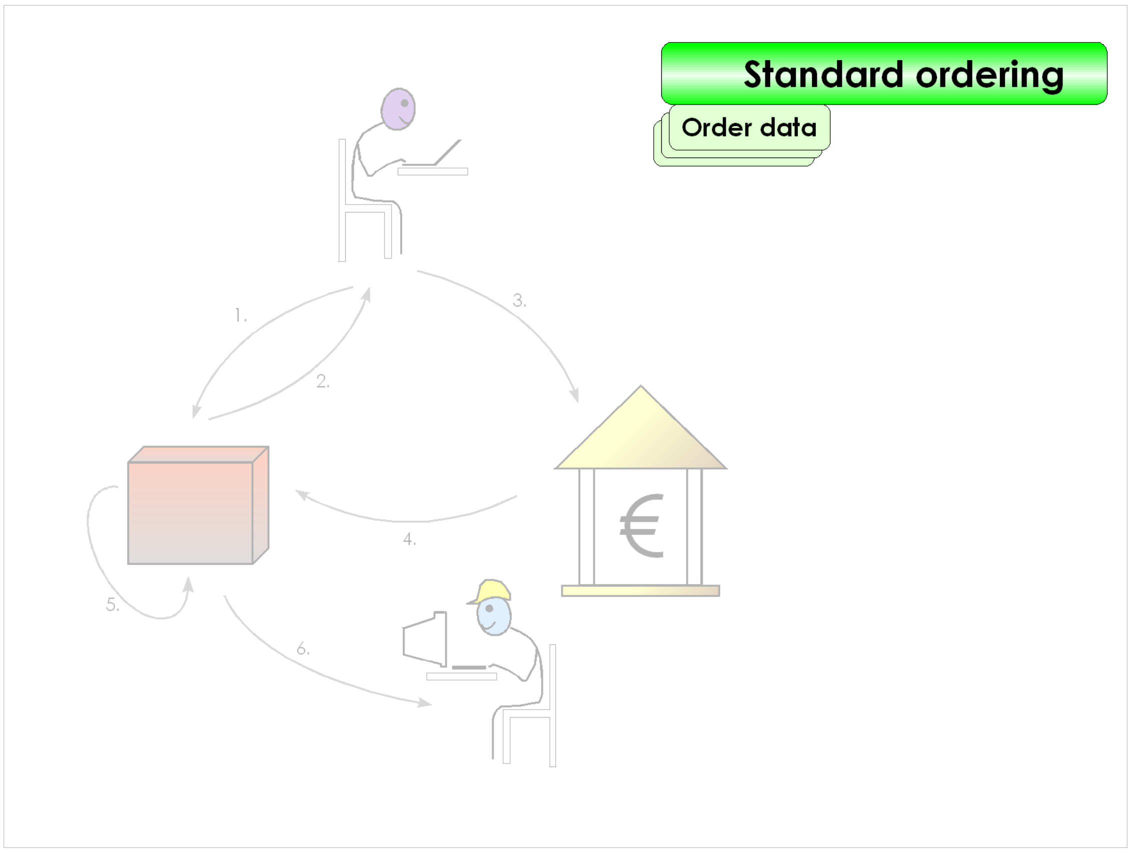
- Test scenarios
  - spanning different system components
  - each composed of several test steps
- Test steps
  - reusable
  - parameterized
  - using adapters to trigger actions in the system under test
- Adapters
  - uniform interfacing to different components - or simulators
- Components
  - parts of the System under Test (SuT)
- Simulators
  - simulating components or machines that cannot be integrated in the automatic integration test (yet)
- Focus in this talk: testing – scenarios, steps, adapters, simulators.

→Scenarios

Integration test scenarios

- Based on user stories.
- Span different system components and/or simulators.
- Contain test data.
- Built from test steps.
  - Parametrizing the test steps.
- Contain checks.
- Exist in various variants, e.g., for exceptional behaviour.
- Different scenarios reuse test steps.
- May be nested, i.e., a test scenario contains another test scenario.
- Development is mostly business-driven.
- Test scenarios can be developed before the system under test exists.
- **Changes in business processes are reflected here**.

→ Example.



Example test scenario

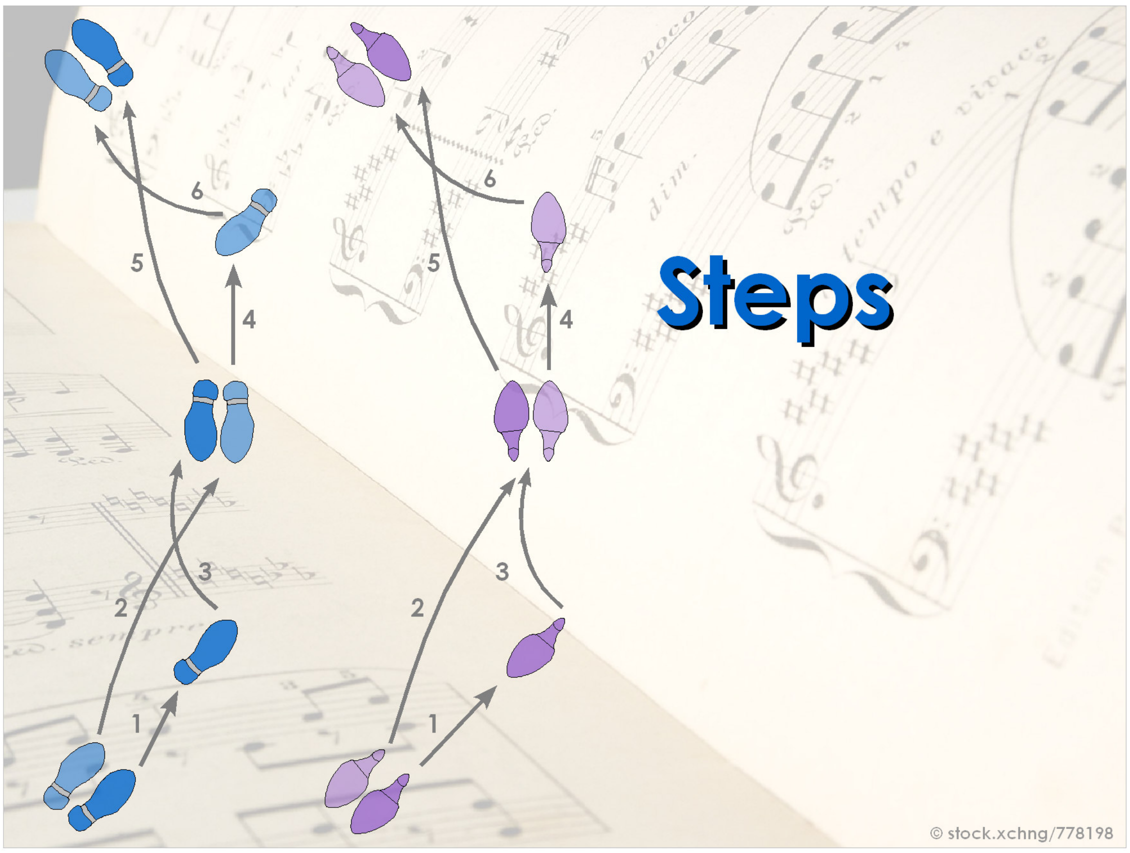
Various test scenarios:

- Same test flow
- Different test data

Test data:

- User logins (customer and stock manager)
- Order data
- Bank transfers
- Order reference number (generated dynamically)

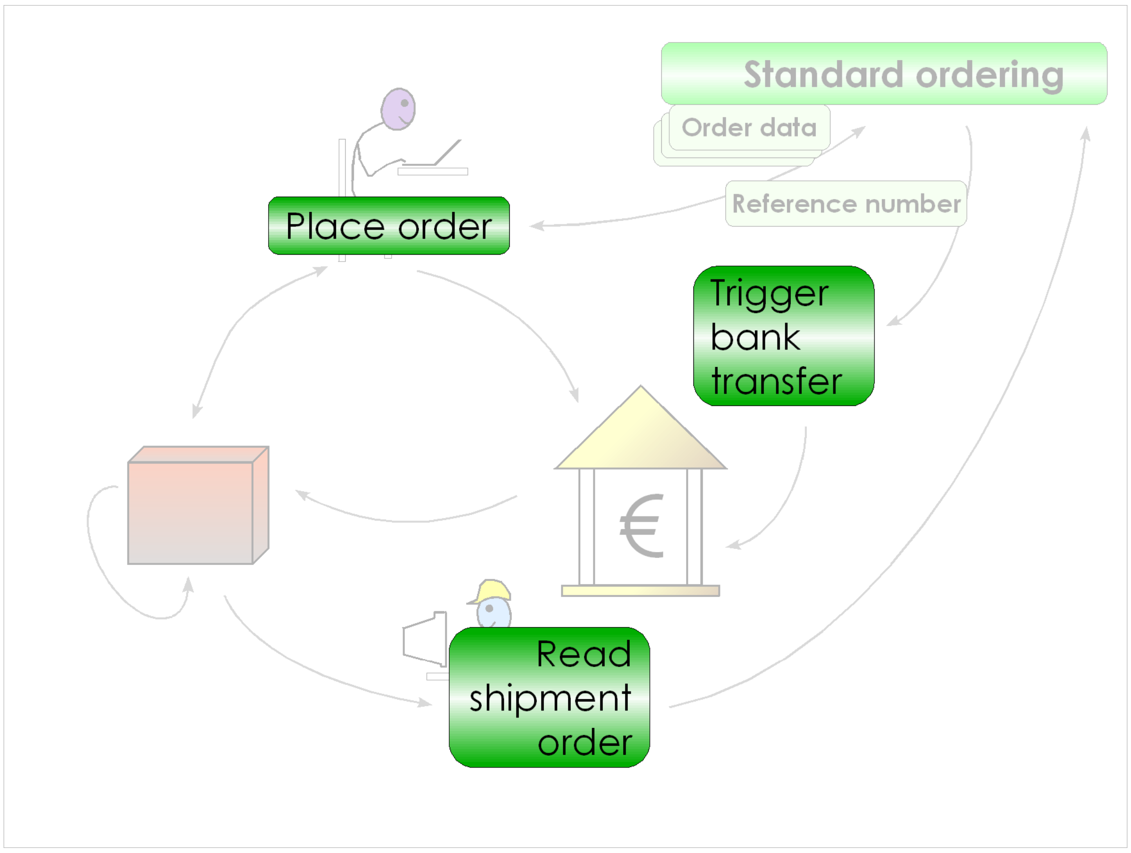
→ Test steps.



Test steps

- Reusable in different test scenarios.
- Parametrized by test data.
- A step accesses the system components or simulators through adapters.
  - Adapters abstract from the technical API of a component.
  - Test steps use the adapter API only.
- A step may use different adapters.
- Test steps may be implemented before the components exists that are to be used.
- **A change in the basic (business-driven) usage of a component is reflected here**.

→ Example.



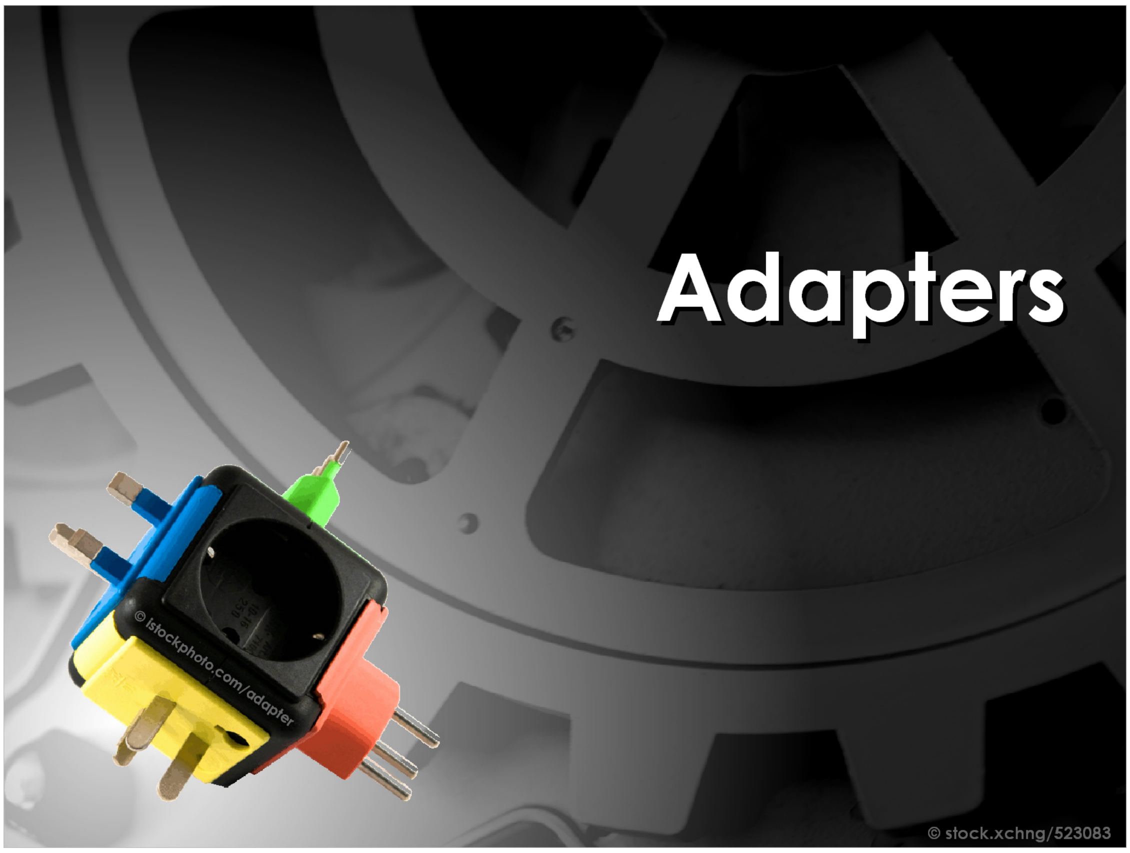
Test scenario consists of test steps:

- Place order
- Trigger bank transfer
- Read shipment order

Example test step: Place order

- Test data: goods + quantities, shipment address, ...
- Returns: order reference number
- Implementation
  - select a good from list
  - enter quantity
  - add to shopping basket
  - select next good
  - …

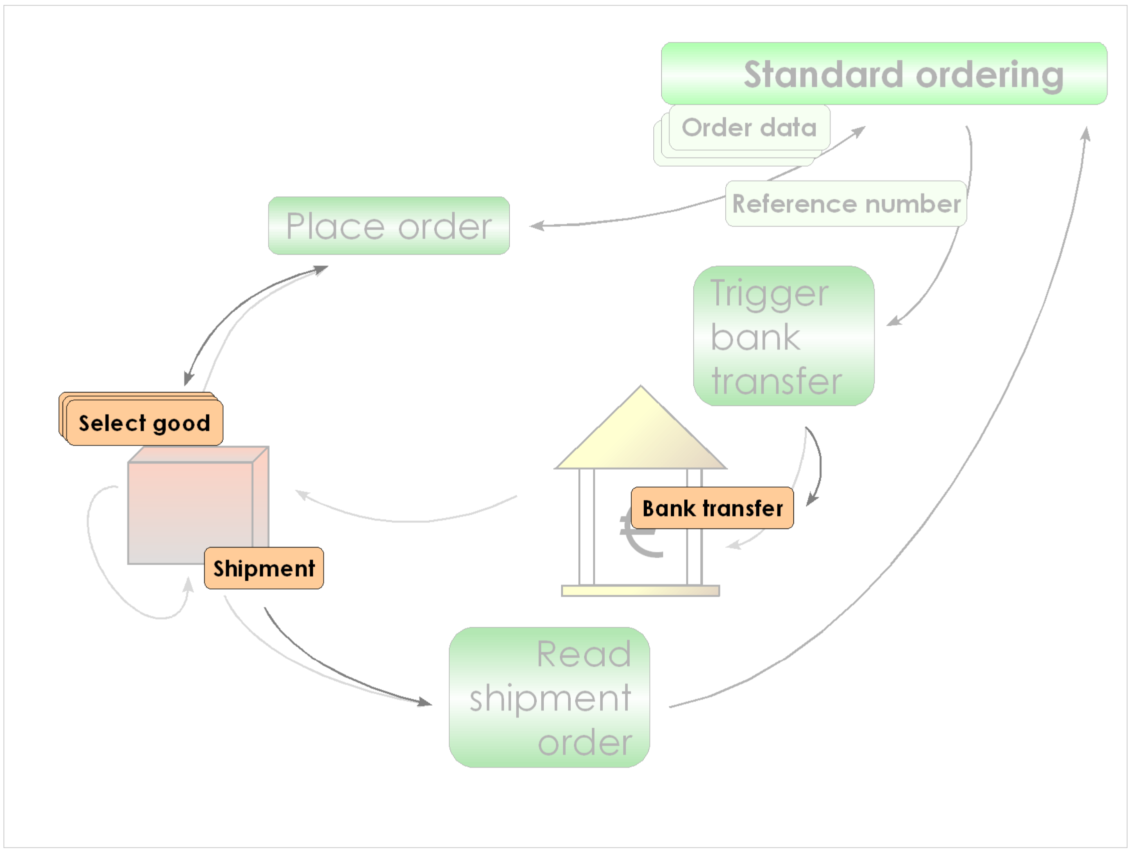
→ Adapters.



Adapters

- Encapsulate the technical implementation of the interfaces to system components or simulators.
- Use environment parameters as IP addresses, database instances, …
- May provide default values for easier use of interface.
- Provide a uniform interface to components and simulators for test steps.
- Abstract from different technical APIs or transport protocols.
- Enable exchange of simulator against system component.
- **API changes – or transport protocol changes – are reflected here.**

→ Example.

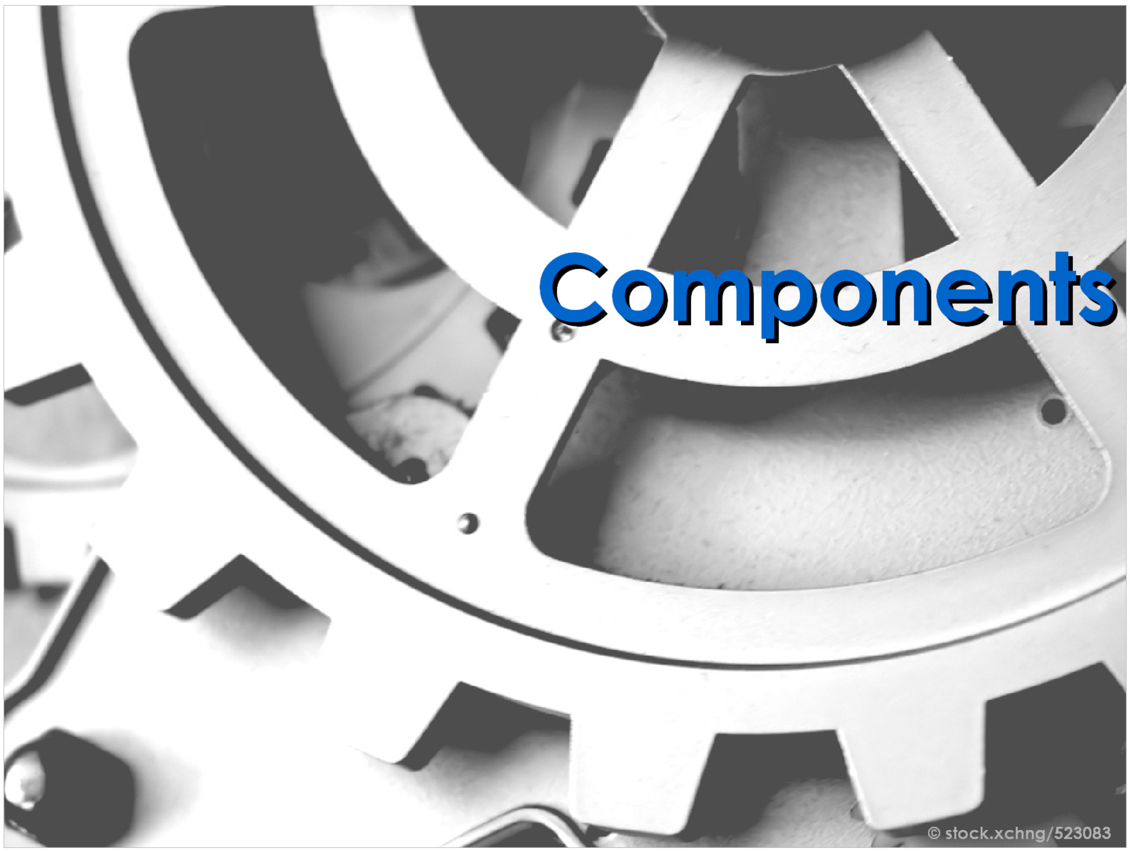


Adapters

Adapter for selecting goods in order management system

- Select a good
- Select several goods
- Select good with quantity (shortcut)
- Select several goods with quantities (shortcut)
- ...

→ Components & Simulators.

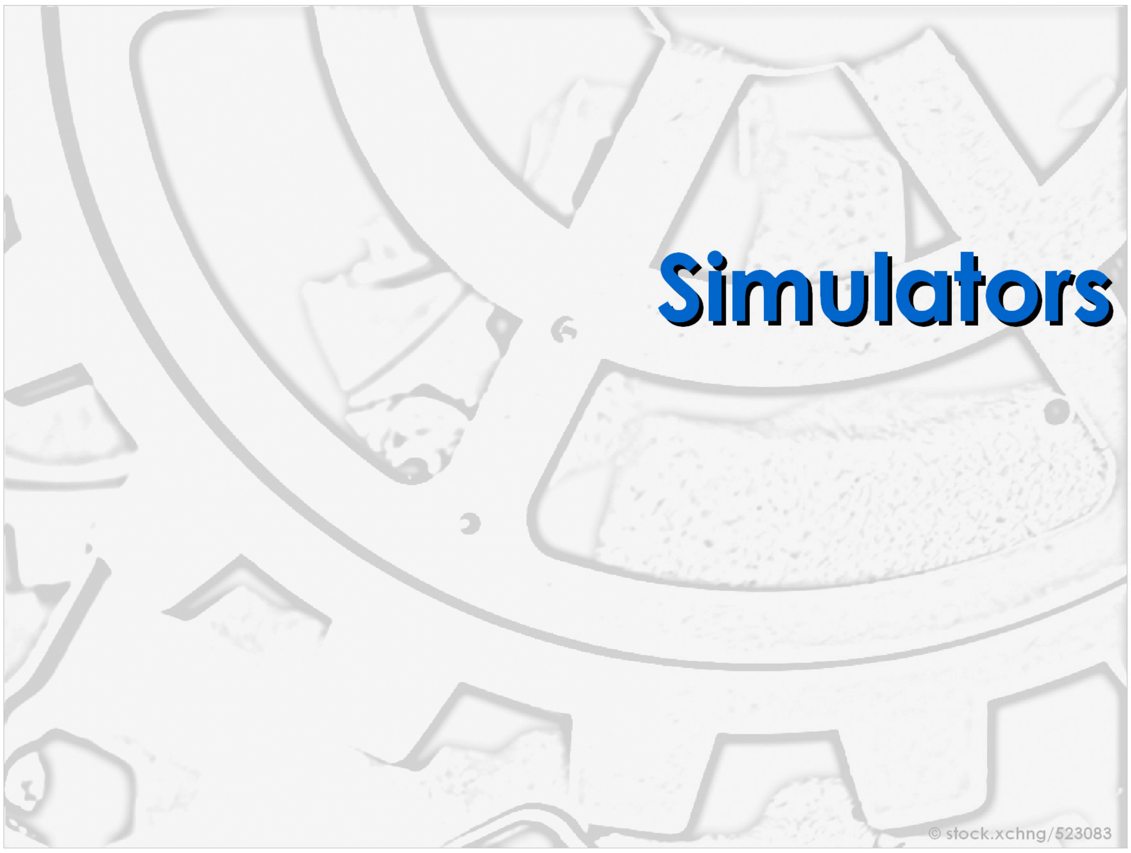


Components

- Parts of the system under test.
- Component APIs are hidden in adapters.
- While integration testing some components may be mocked.

- Other systems around the system under test.
- APIs of these systems are also hidden in adapters.
- Often these systems may not be included in the integration test.

-→ Simulators.


Simulators

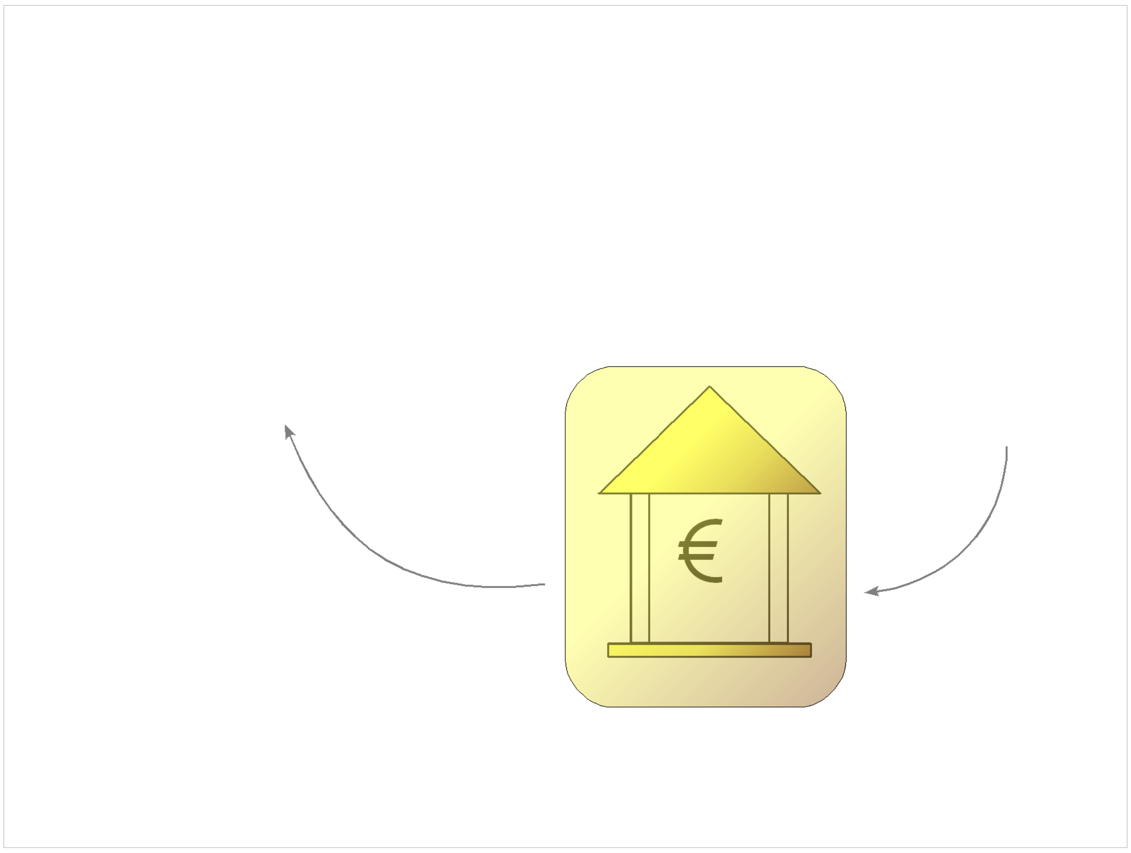
© stock.xchng/523083

Simulators

- Simulate a system component that is not implemented yet – or does not provide the expected behaviour.
- Simulate a system or machine that may not be integrated in an integration test, e.g., devices.
- Provide normally more behaviour than a simple mock, e.g., state, sessions.

- The same test scenario may be executed against the simulator or the system component.
  - Configuration of integration test runs!

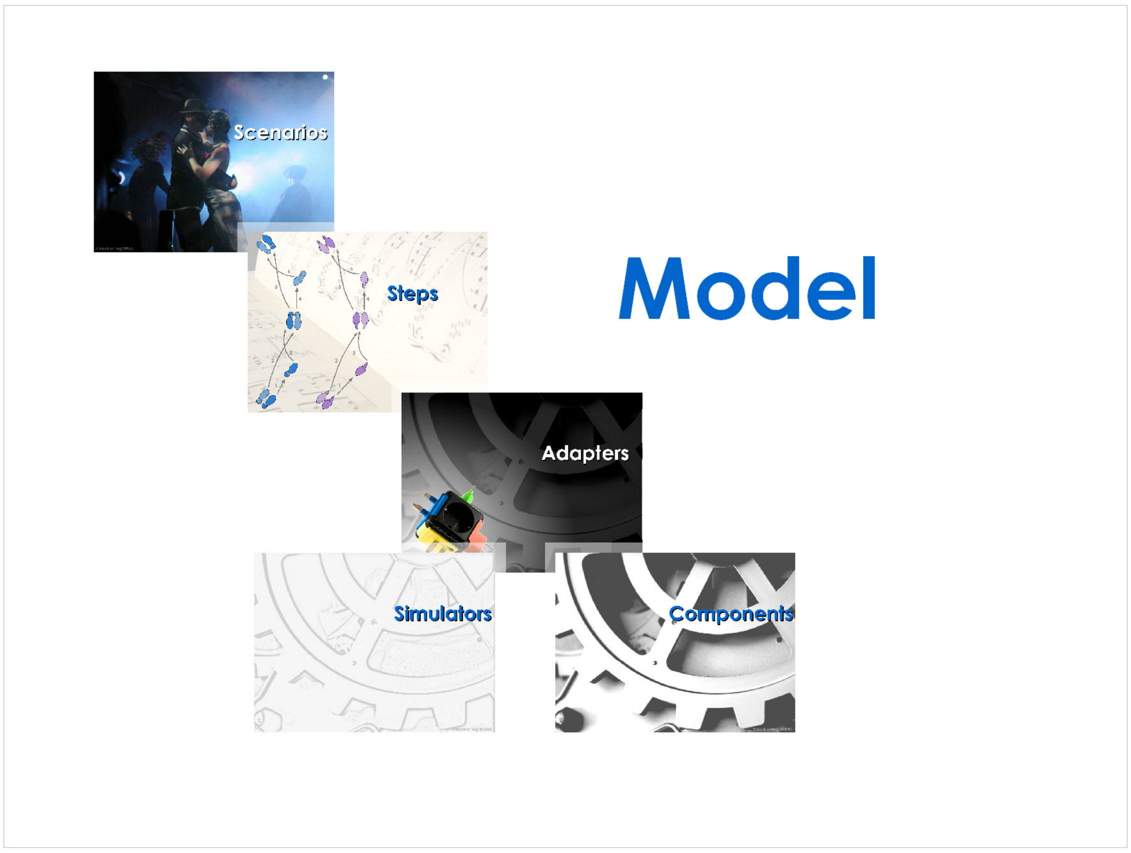
→ Example.



Simulator example

- A bank transfer is triggered.
- The bank simulator waits for a predefined time.
- Then it sends a report containing this bank transfer together with some others.

→ Model summary.



Model

Consider

- Scenarios – executable user stories.
- Steps – reusable, parametrized behaviour.
- Adapters – encapsulating technical parameters.
- Simulators – filling gaps.

→ From model to execution

Test Execution Engine

From model to execution: test execution engine

- The test execution engine controls the test execution.
    - Interpreting test scenarios.
    - Calling executable test steps.
    - Instantiating and destroying adapters.

→ Implementation examples

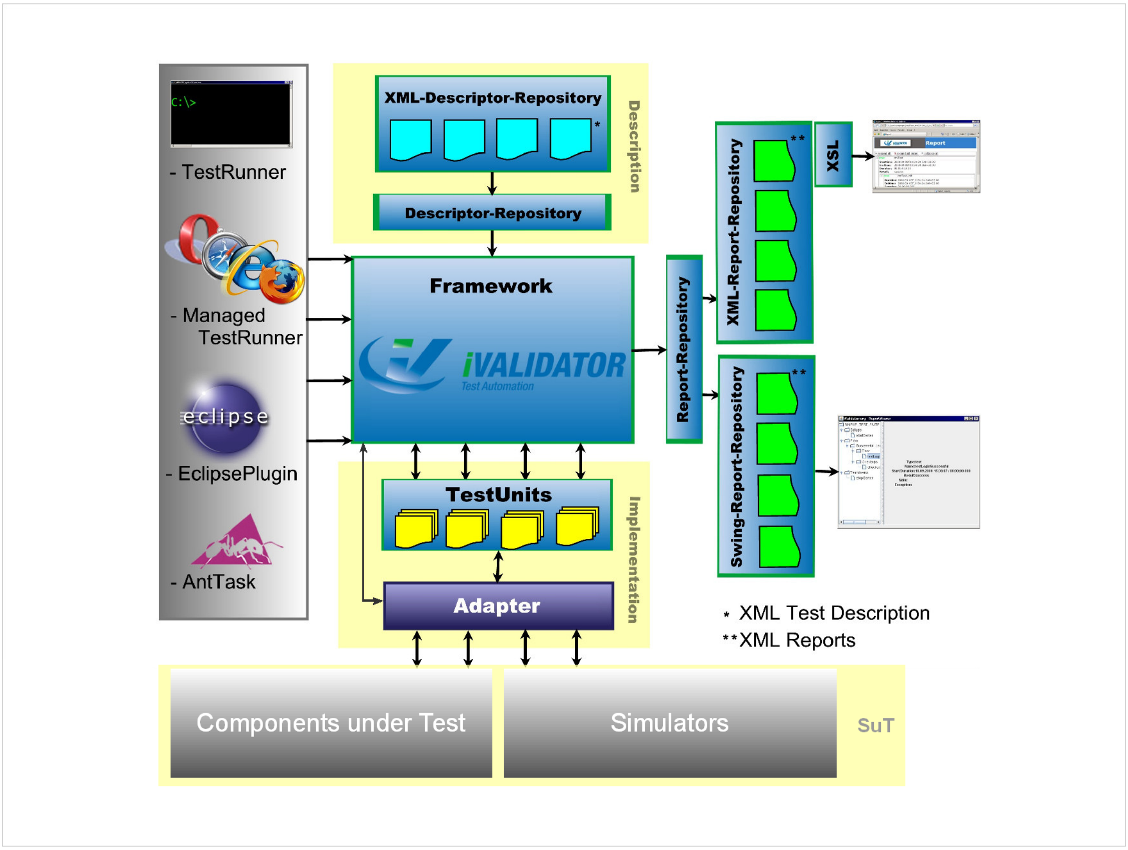
iVALIDATOR

Test Automation

Implementation examples

- The open source framework iValidator implements such an engine
  based on Java and XML..
- There are many others around, e.g., Fit/Fitnesse.
  - they only have to be used in the right way

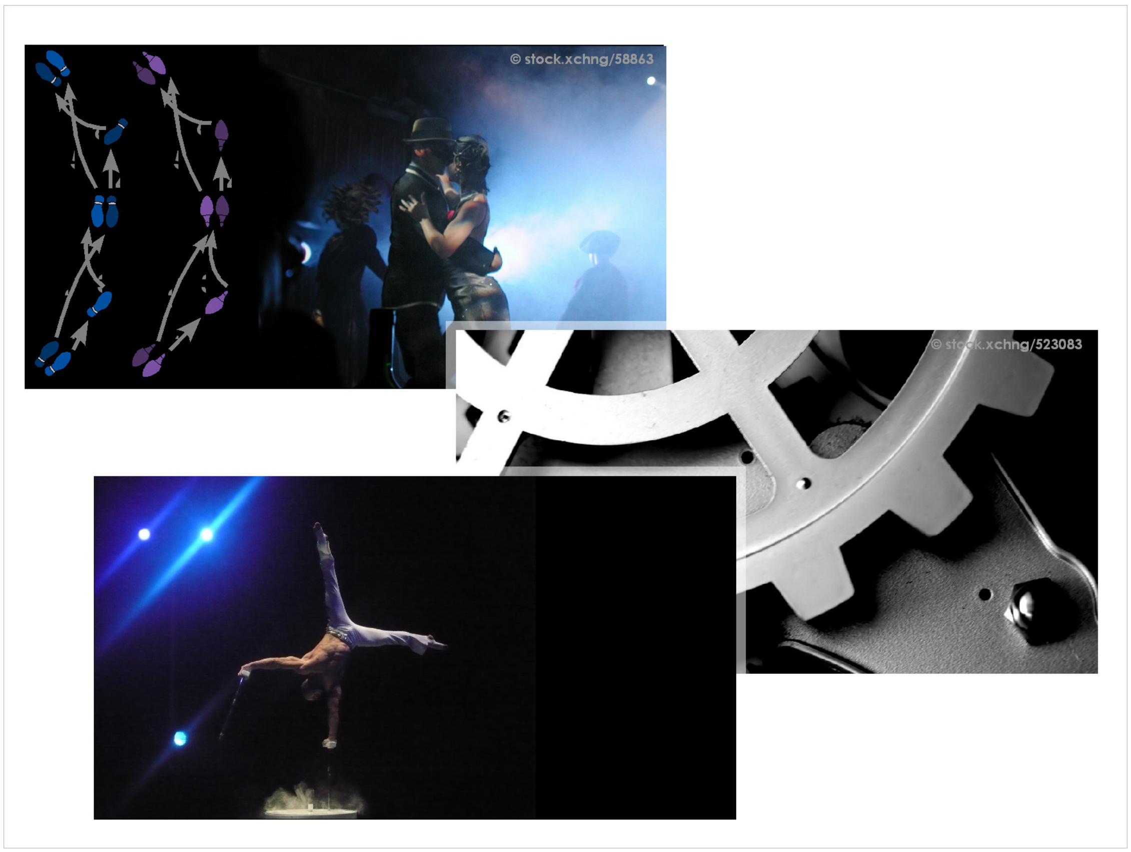
→ iValidator



iValidator

- The application of the integration testing model evolved into an automated testing framework: the open source project iValidator (ivalidator.org).
- It directly supports three of the development areas:
  - Description of test scenarios
  - Implementation of test steps
  - Implementation of test adapters

→ Summary.



Summary

- Using this testing model it is possible to automate integration tests:
  - For systems that are highly automated themselves
  - For systems that are interfacing to other systems or machines without user interaction
  - In parallel with development, or even ahead of it
  - While meeting the agile requirements - always prepared for change

Our advice:

Do integration testing - always!

Automate integration tests – whenever possible and necessary!

Be agile - prepare for change!