ESE 2015

Spring Testing

Best practices

Mario Kaufmann

b UNIVERSITÄT BERN

Use the tools available: Dependencies

Most importantly:

- JUnit: classical unit testing framework for Java
- Mockito: mock framework, can also be used for Spring
- Spring test: provides test contexts and utilities for testing

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>4.2.2.RELEASE</version>
    <scope>test</scope>
</dependency>
```

Use the tools available: Assertions

The spring framework provides its own assertions, e.g. *ModelAndViewAssert*

assertViewName(ModelAndView mav, String expectedName)

Check whether the correct view was returned from the controller (for example the result page or an error page)

assertModelAttributeValue(ModelAndView mav, Object modelName, Object expectedValue)

Check whether a given attribute was added to the model and furthermore if it is equal to the expected value

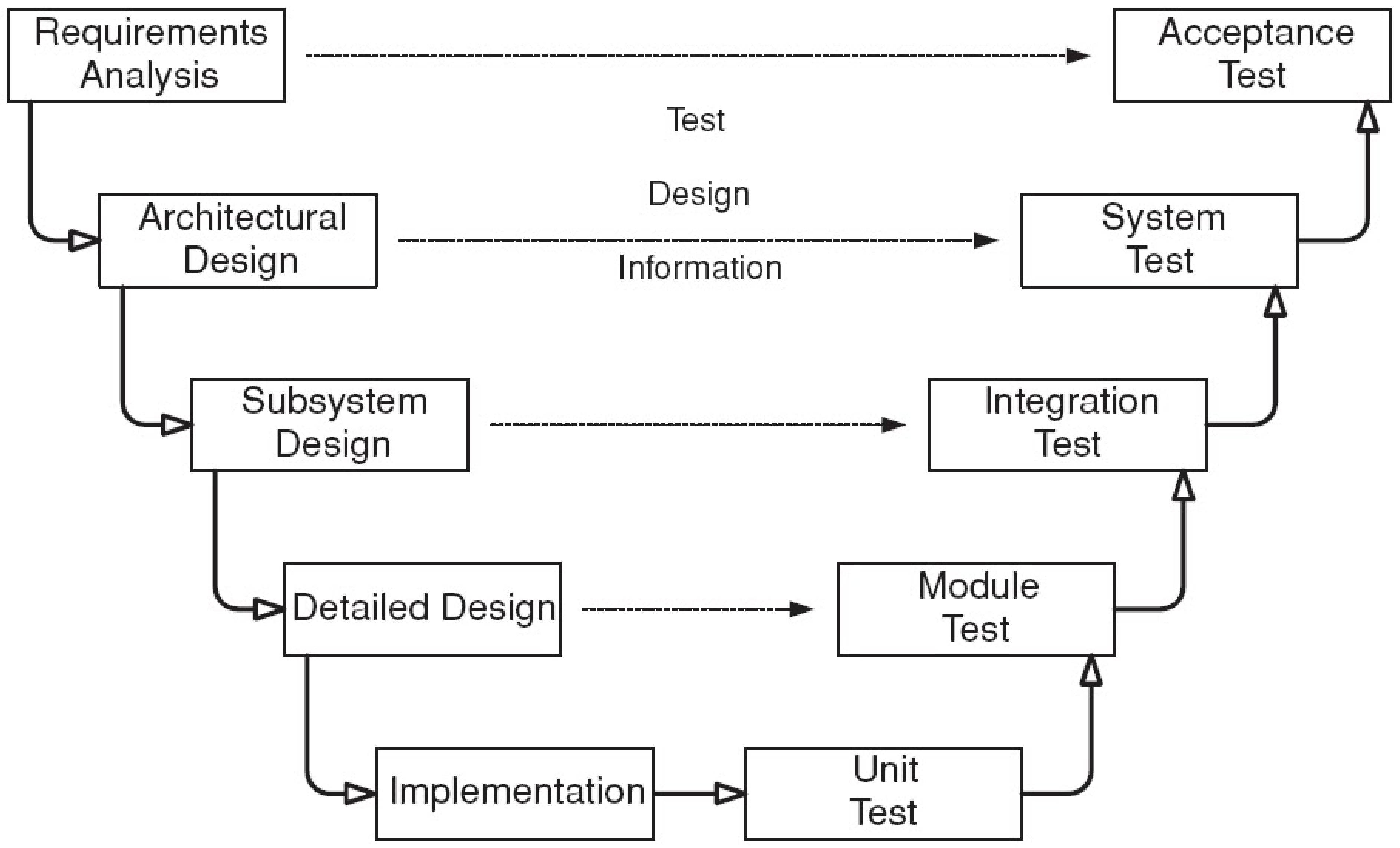
Use the tools available: Mocks

```
MockMvc mockMvc = webAppContextSetup(wac).build();

this.mockMvc.perform(post("/create")
     .param("email", “john@doe.com")
     .param("firstName", “John")
     .param("lastName", “Doe"))
     .andExpect(status().isOk())
     .andExpect(model().attributeExists("page_error"));
```

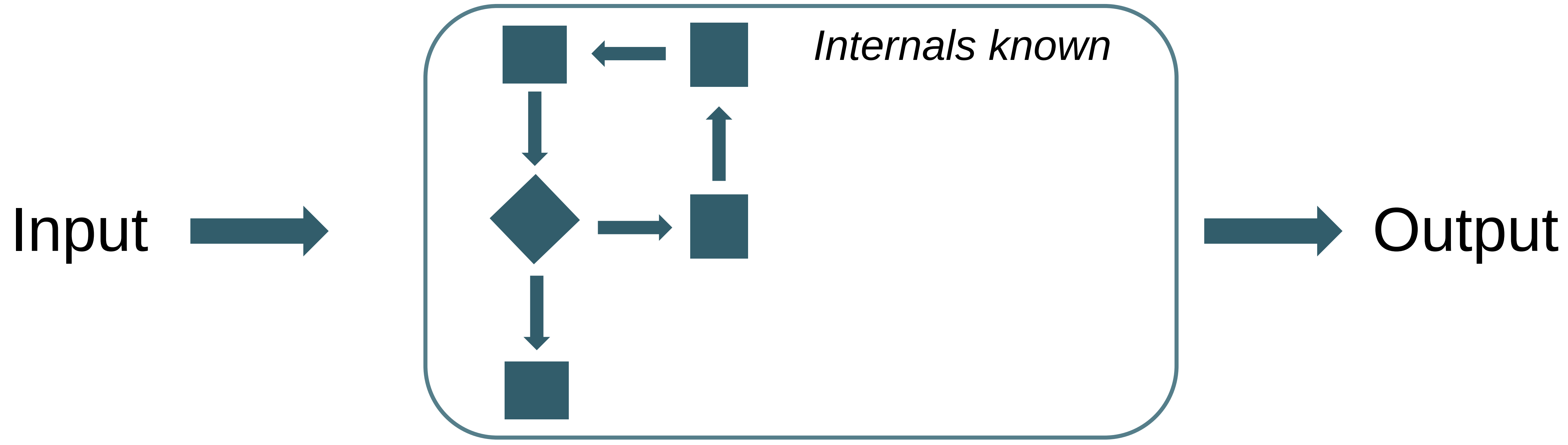
Create mock POST request and perform assertions on results

Unit testing vs. Integration testing



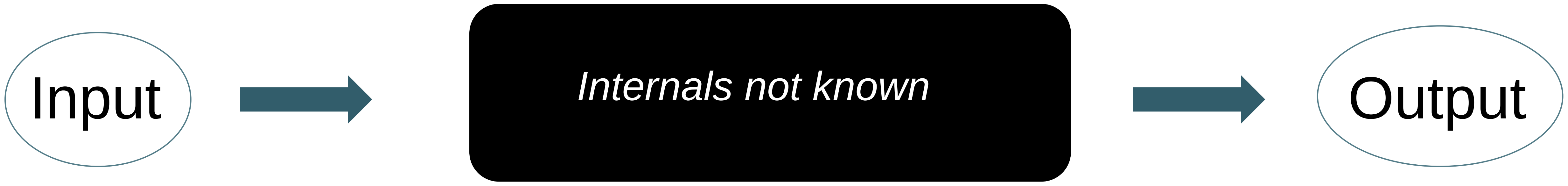
White-box testing

- Test internal structures
- Testing with knowledge of code in mind
  - Throwing of specific exceptions
  - All code paths (coverage)

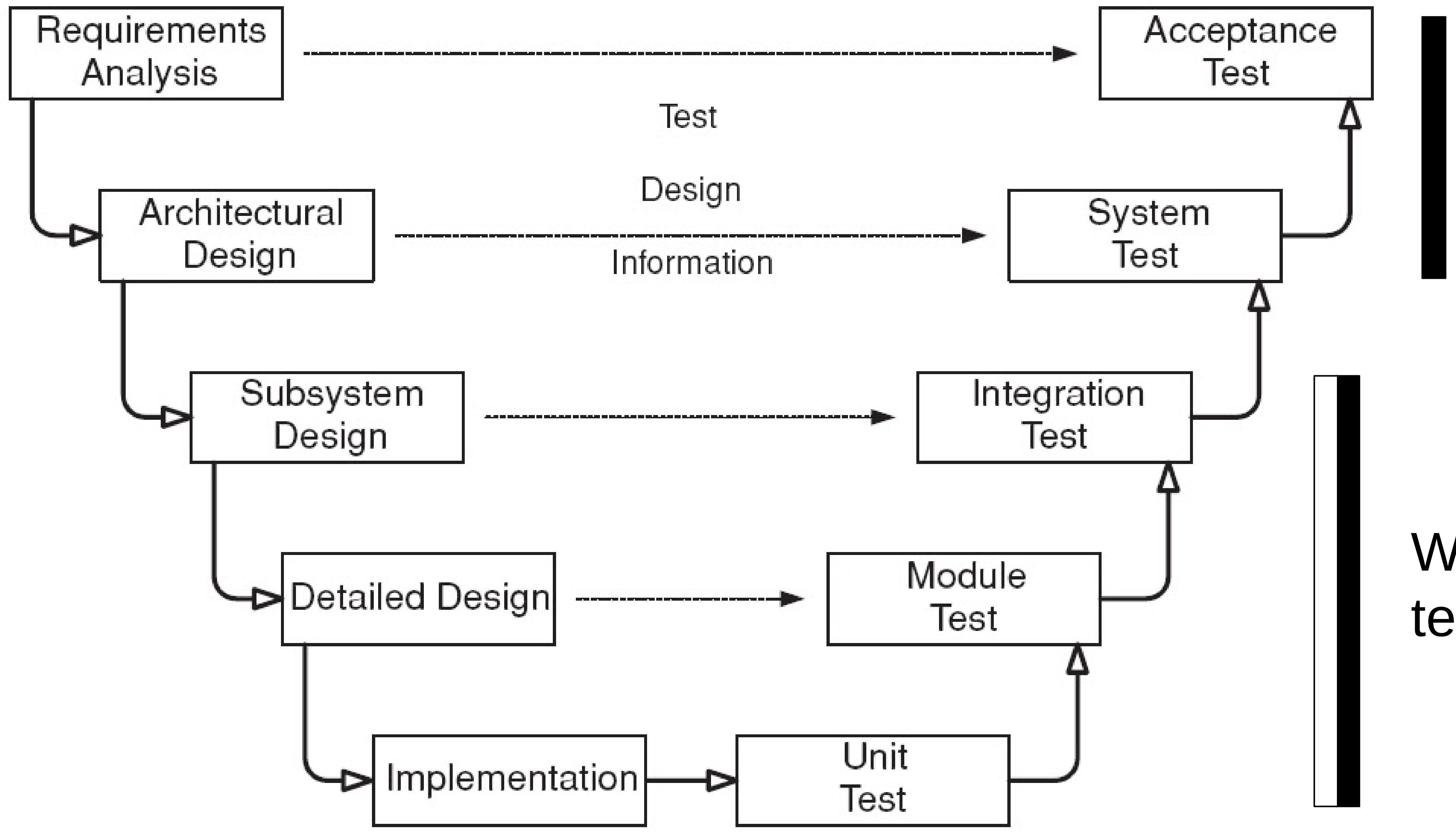


Black-box testing

- Only test using input and output
- Internal structures not known or not relevant
- *What* the software does instead of *how* it does it
- Test cases derived from requirements or external specification



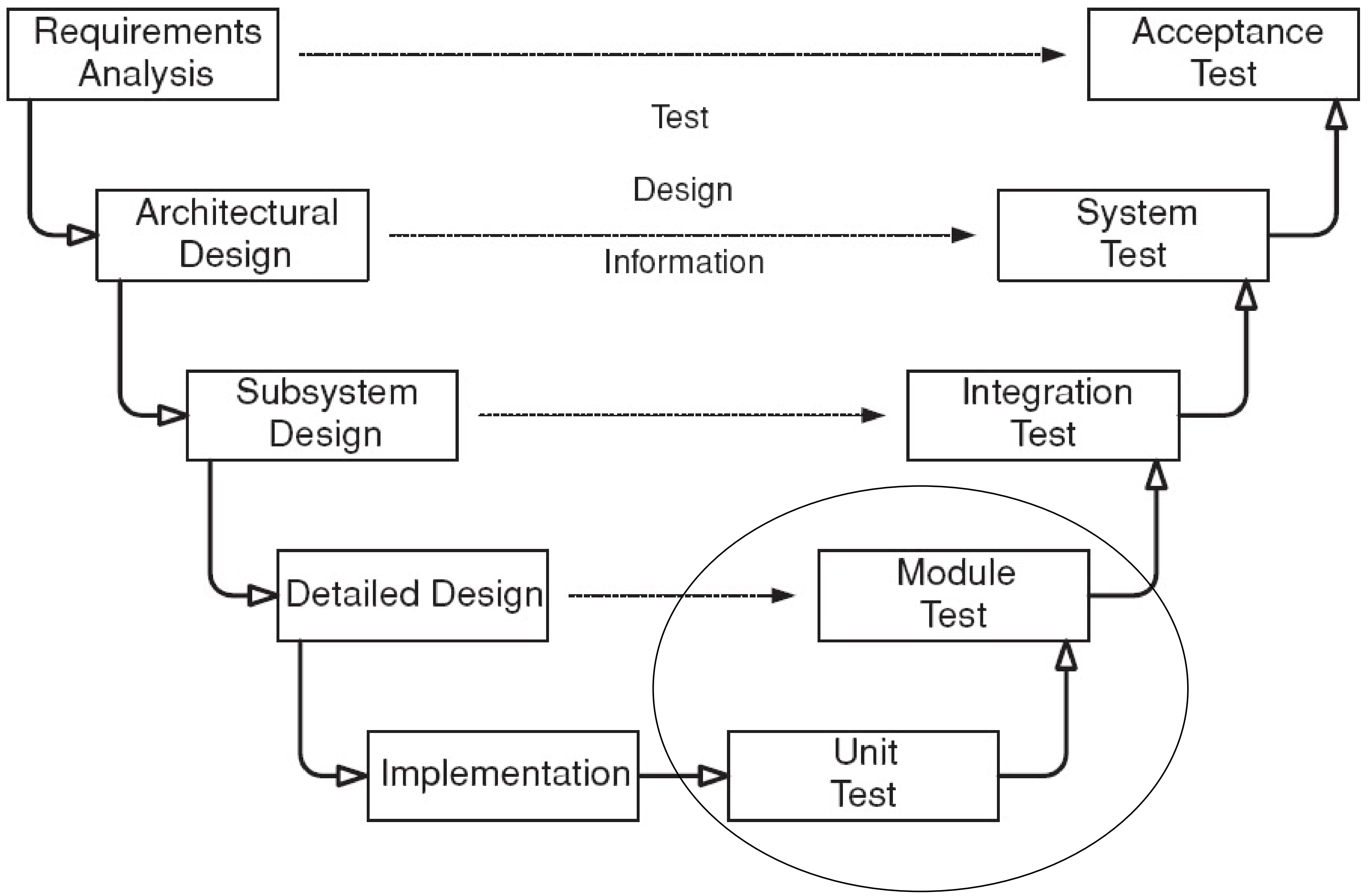
Unit testing vs. Integration testing



Black-box testing

White- and black-box testing

Unit testing vs. Integration testing



Unit testing vs. Integration testing

- True unit tests require more work to set up in Spring
  - Configuration needs to be provided
  - Autowired dependencies need to be mocked

```
@Autowired
private OrderService orderService;
```

- Also possible: integration testing
  - Including parts or the entire original application context

Testing services

- Test only service layer
- Mock out the data access objects (e.g. with Mockito)
- Independent of database, can be run on any system

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
public class AccountServiceTest {

   @Autowired
   private AccountService accountService;

   private AccountDao accountRepository;

   // …
}
```

```
@Before
public void setup() {
   Account account =
       new Account("john", "john@doe.com");

   Mockito.when(
      accountRepository.findByUsername("john"))
        .thenReturn(account);
}
```

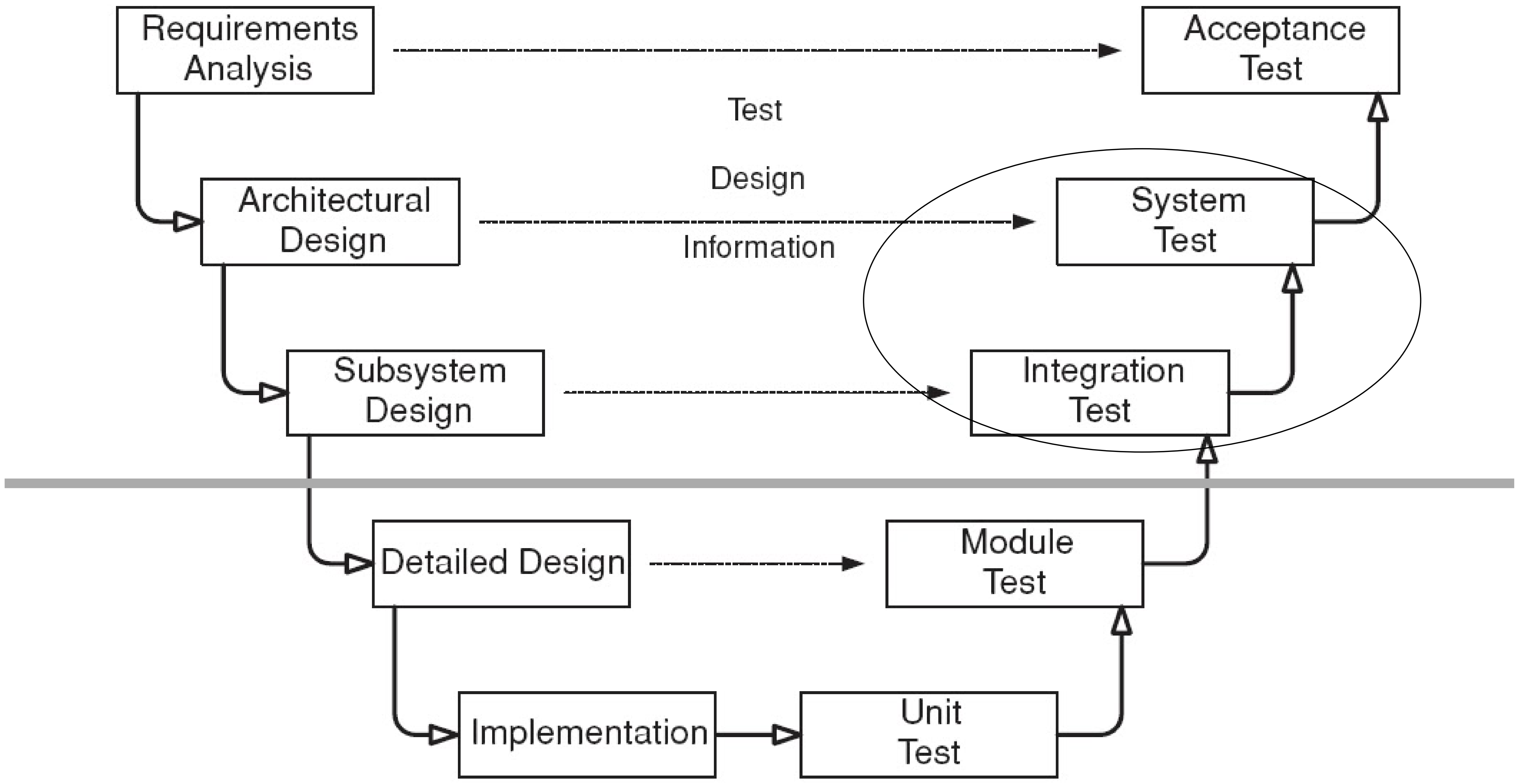
Testing controllers

- Test only controller, mock out services
- Test whether correct view name (i.e. JSP page) is returned
- Test whether correct exception thrown
- Test whether JSON data is returned and has right form

For example:

```
mockMvc.perform(get("/"))
            .andExpect(status().isOk())
            .andExpect(view().name("todo/list"))
            .andExpect(model().attribute("todos", hasSize(2)));
```

Testing controllers and services



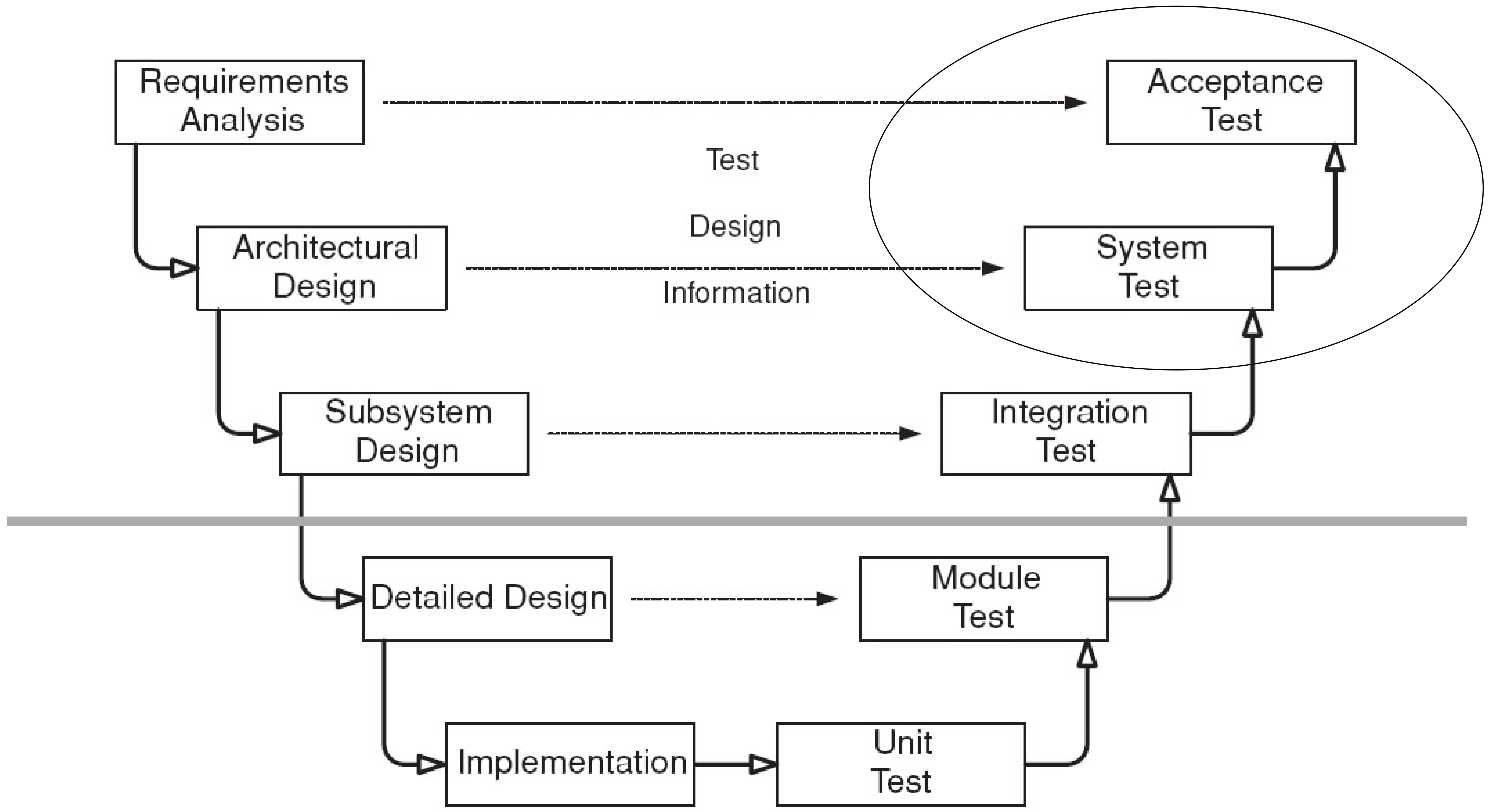
Testing controllers and services

- Test controllers and services together
- Integration test
- Several stages possible:
  - Test controllers and services with mocked data access layer
  - Test controllers and services completely without mocks (run on real database)

```
mockMvc.perform(get("/"))
        .andExpect(status().isOk())
        .andExpect(view().name("todo/list"))
        .andExpect(model().attribute("todos", hasSize(4)));
```

Data now from database

Testing controllers and services



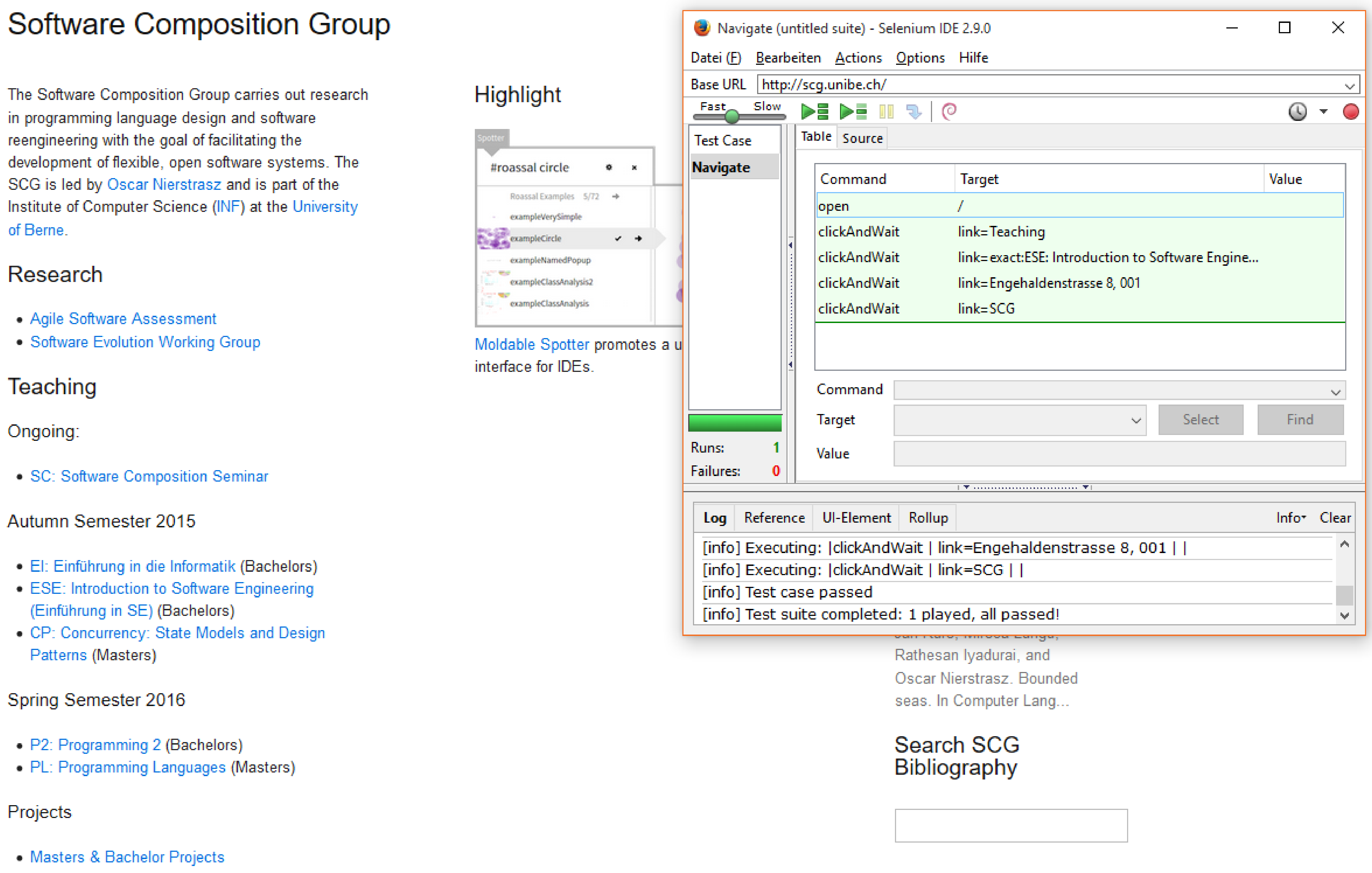
Testing JSP pages

- Short answer: should not really be necessary!
- JSP pages are only views, should not contain code that needs to be tested
- Such code generally belongs in the controller
- Controller tests: only test for correct view name
- However: It can make sense to test view, e.g. with Selenium

Selenium

- Implemented as Firefox Add-On
- Automated tests from the user perspective
- Recorded directly in the browser (XPath expressions possible)
- Many possible actions: follow links, fill out forms etc.
- Actions can be replayed in the browser
- Can be used together with Jenkins (continuous integration server)

Selenium



Software Composition Group

The Software Composition Group carries out research in programming language design and software reengineering with the goal of facilitating the development of flexible, open software systems. The SCG is led by Oscar Nierstrasz and is part of the Institute of Computer Science (INF) at the University of Berne.

Research

- Agile Software Assessment
- Software Evolution Working Group

Teaching

Ongoing:

- SC: Software Composition Seminar

Autumn Semester 2015

- EI: Einführung in die Informatik (Bachelors)
- ESE: Introduction to Software Engineering (Einführung in SE) (Bachelors)
- CP: Concurrency: State Models and Design Patterns (Masters)

Spring Semester 2016

- P2: Programming 2 (Bachelors)
- PL: Programming Languages (Masters)

Projects

- Masters & Bachelor Projects

Highlight

Moldable Spotter promotes a u interface for IDEs.

Rathesan Iyadurai, and Oscar Nierstrasz. Bounded seas. In Computer Lang...

Search SCG Bibliography

Navigate (untitled suite) - Selenium IDE 2.9.0
Datei (F) Bearbeiten Actions Options Hilfe
Fast Slow
Test Case
Navigate
Table Source
Select Find
Log Reference UI-Element Rollup Info Clear

Base URL: http://scg.unibe.ch/

Command	Target	Value
open	/	
clickAndWait	link=Teaching	
clickAndWait	link=exact:ESE: Introduction to Software Engine...	
clickAndWait	link=Engehaldenstrasse 8, 001	
clickAndWait	link=SCG	

Runs: 1
Failures: 0

```
[info] Executing: |clickAndWait | link=Engehaldenstrasse 8, 001 | |
[info] Executing: |clickAndWait | link=SCG | |
[info] Test case passed
[info] Test suite completed: 1 played, all passed!
```

Useful Links

Black-box testing / white-box testing
https://en.wikipedia.org/wiki/Black-box_testing
https://en.wikipedia.org/wiki/White-box_testing

Spring unit testing
http://docs.spring.io/spring/docs/current/spring-framework-reference/html/unit-testing.html

Spring integration testing
http://docs.spring.io/spring/docs/current/spring-framework-reference/html/integration-testing.html

ModelAndViewAssert documentation
http://docs.spring.io/spring-
framework/docs/2.5.x/api/org/springframework/test/web/ModelAndViewAssert.html

MockMVC documentation
https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/test/web/servlet/MockMvc.html

Selenium
http://www.seleniumhq.org/projects/ide/