Effective Testing & Test Automation for Developers

J-Fall Pre-conference

Sebastian Daschner

- Lead Java Developer Advocate @ IBM
- Conference speaker, trainer, book author
- JAX-RS, JSON-P, Config Expert Group member
- Java Champion
- JavaOne Rockstar / Code One Star

Agenda

- Testing introduction
- Test scopes
- Maintainable tests & code quality
- Running tests locally
- Continuous Delivery & Automation

Testing & Automation

Tests

- Necessary to rely on a certain functionality
- Natural part of the process
- Simulate production behavior
- Failing tests are *positive*

Necessity of Tests

- Verifying behavior
- Required to be able to “move fast”
- Prerequisite of Continuous Delivery

Test requirements

- Predicability
- Isolation
- Reliability
- Fast execution
- Automation
- Maintainability

Predictability

- Same conditions must produce the same outcome
- Not influenced by circumstances
- No alternating behavior

Typical Test circumstances

- Current time
- Time zones, locales
- Randomly generated data
- Concurrent test execution
- External systems

Isolation

- Tests run self-sufficiently
- Not affecting other tests or scenarios

Reliability

- Reliably test all functionality*
- Passed tests == ready for production
- No human interaction required

Fast execution

- Necessity for fast feedback
- Important with growing number of tests
- Think in seconds, not minutes

Automation

- Execution & verification
- No human interaction required

Maintainability

- Shows once code changes
- Usually involves a lot of effort
- Possibility to change or extend test cases

QA Departments & Testers?

QA Departments & Testers?

- Exploratory testing
- Thinking about reasonable test cases
- Crafting test scenarios

Test execution?

Done by computers instead. Especially, as things grow.

What to test

- Business logic
- Non-functional requirements
- Code-level vs. deployment-level
- “Object under test”

Test Scopes

Test Scopes

- Unit tests
- Code-level integration tests
- Database integration tests
- System tests
- End-to-end tests

Unit Tests

- Verifying behavior of individual unit
- Simple & fast test technology (JUnit)
- Instantiating beans, mocking dependencies

Parameterized Tests & Dynamic Tests

Assertion Matchers

AssertJ vs. Hamcrest Matchers

Code-Level Integration Tests

- Verifying behavior of coherent components
- Framework integration (dependency injection)
- Easy setup, using embedded containers
- E.g. Arquillian, Spring Tests

Shortcomings of Integration Tests

Shortcomings

- Slow execution
- Re-execution of framework wiring
- Insufficient test coverage

Code-Level Integration Tests: Alternatives?

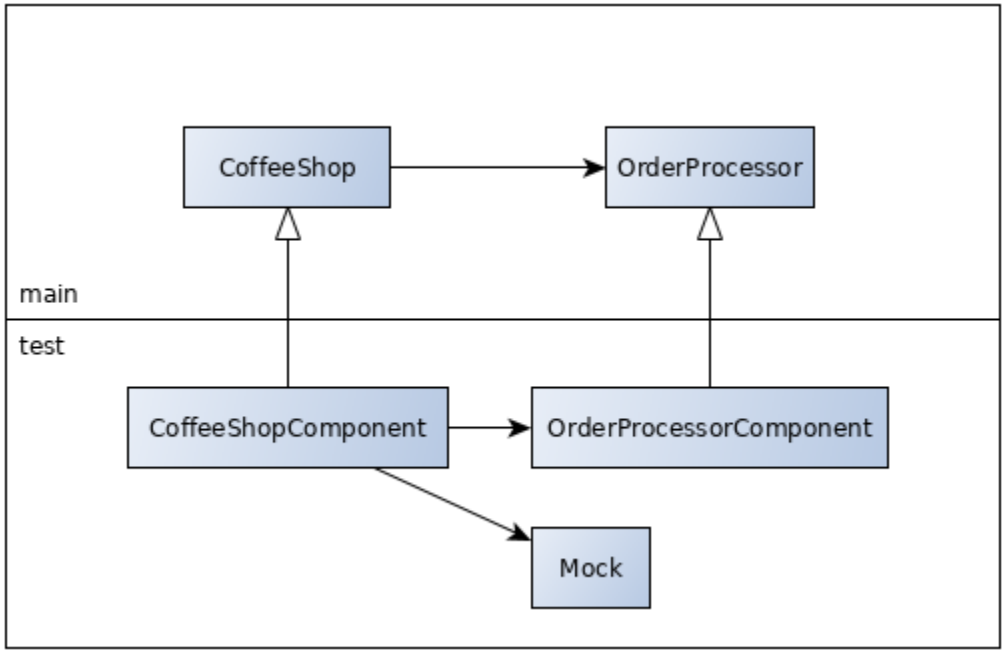
Alternative: Use Case Tests

aka Code-Level Integration Tests without containers

Use Case Tests

- aka component tests / service tests
- Code-level tests
- Includes all code-level components involved
- Includes use case boundary except external boundaries

Use Case Tests



Alternative: System Tests?

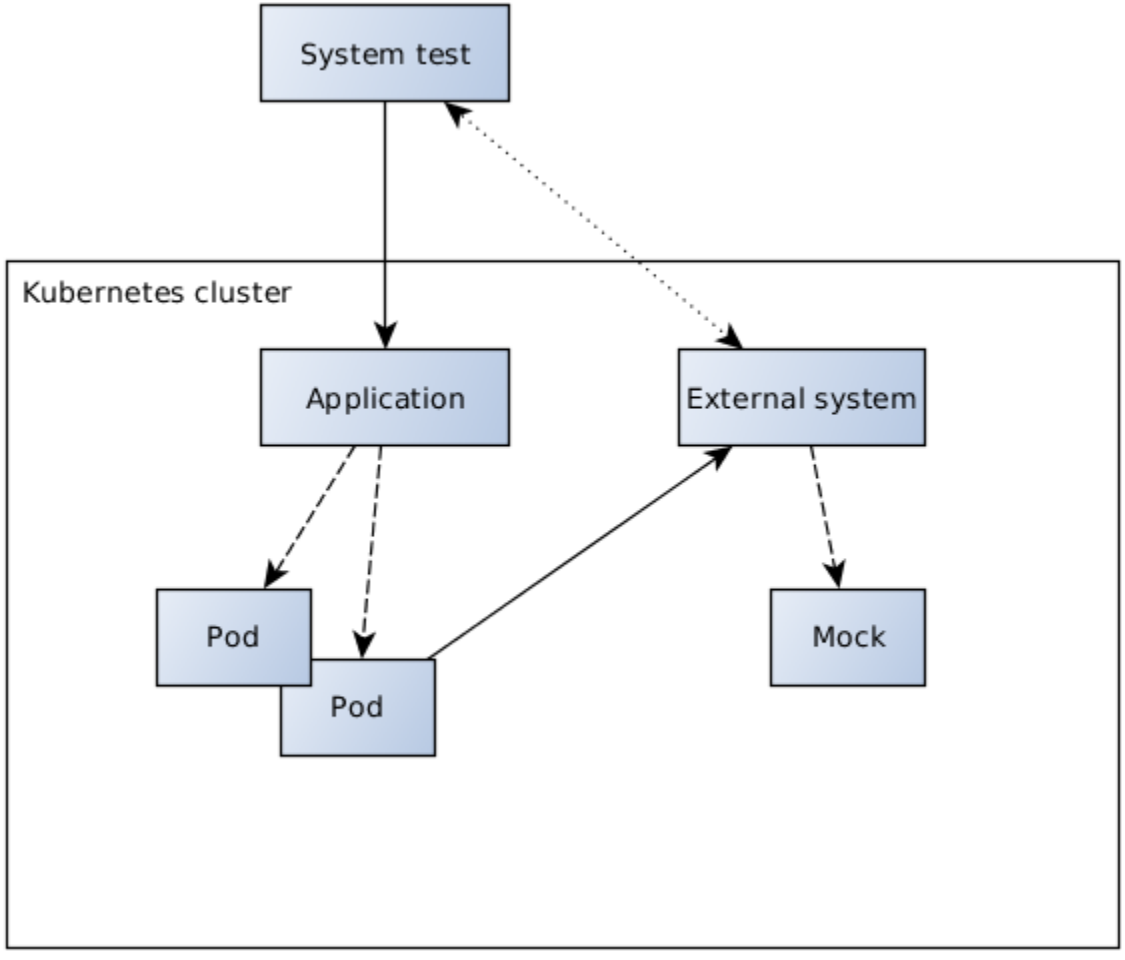
System Tests: Motivation

- Testing software in *exactly* same way as production
- Same runtime, setup, configuration
- Minimizing overall test execution time & effort

System Tests: Approach

- Application-under-test runs without modifications
- Application accessed from outside
- External systems mocked away & controlled
- Complex test scenarios, verifying use cases

System Test on Kubernetes



System Test Project

- Access using external boundaries
- Separate transfer objects & class definitions
- Controlling mock servers

Shortcomings of System Tests

- Slower feedback (comparably)
- Maintaining complex test data and scenarios
- Mocking of external system required

Test Code Quality to The Rescue!

Test Code Quality

- Abstraction layers
- Single responsibility principles
- Don’t repeat yourself

Test Scenarios First, Implementation Second

- On paper, comments, concepts
- Implementation afterwards
- Walking down abstraction layers

System Test Components

- Crafting “test component APIs”
- Beware of leaky abstractions

Proper Test Code Quality Is Not Forbidden

It’s actually crucial.

Signs of Lack in Test Code Quality

- Copy-paste programming
- Amount of changes required on code change
- Mixing concerns of test classes

Contract Testing

Database Tests

- Verifying database mappings
- Embedded database, fast feedback
- First barrier, no deployment required

Running Tests Locally

Running Tests Locally

- Required for fast development
- Avoiding to needlessly disturb team
- Doable using container technology

Local Container Tests

- Setting up (simplified) environment, similar to system tests
- Using same test scenarios

Local Tests: Technology

- Docker, Docker Compose
- Testcontainers
- Minikube, Minishift, etc.

Additional Test Frameworks

- Spock
- Scala Test
- FitNesse

Test code quality > Test technology

Continuous Delivery & Automation

Testing in Continuous Delivery

- Tests executed as part of pipeline
- Verification of each step before continuing
- Include multiple test scopes
- No human interaction required

Testing in Continuous Delivery

- Tests are key to enable full Continuous Delivery
- Verification for production usage

To take away

- Consider test requirements
- Test code quality matters
- Abstraction layers & seperation of concerns
- Craft reusable test components
- Consider fast feedback vs. completeness
- Test code quality > test frameworks

Resources

- https://github.com/sdaschner/coffee-testing/tree/nljug-workshop

Thank you for your attention!

- sebastian-daschner.com
- @DaschnerS