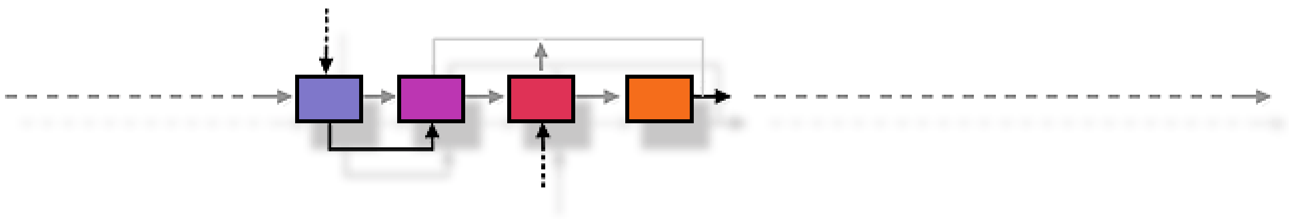
Java
Community
Process



TCK development

Patrick Curran, JCP Chair

December 2012

Agenda

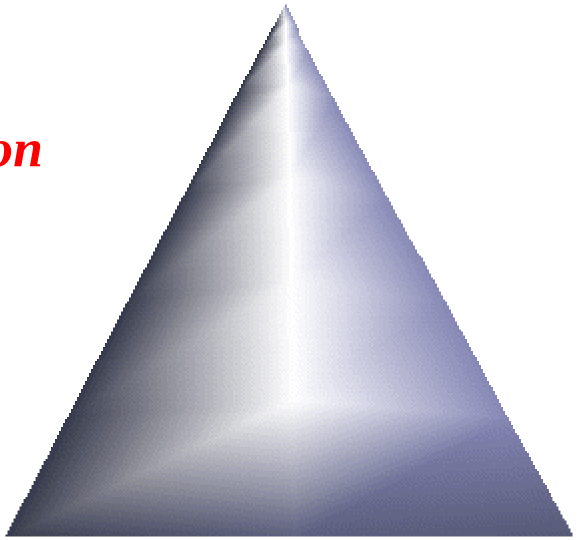
- JSR deliverables and JCP obligations.
- What is conformance testing?
- What makes a good spec?
- What's in the TCK?
- Compatibility Requirements.
- What makes a good TCK?
- How to decide what to test.
- Measuring coverage.
- Testing the TCK.
- The test challenge process.

JSR deliverables

Specification

Is the specification unambiguous?

Can you build an implementation?



Technology Compatibility Kit

Reference Implementation

Is the TCK correct?
Does the RI conform?

Process Document obligations

- Include documentation explaining how to run the TCK and interpret the test results.
- Include *Compatibility Requirements* explaining what is necessary, in addition to passing the tests, to be compatible.
- Provide a mechanism for running the tests automatically.
- Define a test-appeals process.
- Ensure that all of the APIs required by the Specification are completely and correctly implemented and that no un-Specified APIs are included in the JSR's namespace.
- Provide a *TCK Coverage Document* explaining the criteria used to measure TCK test coverage, and justifying the level of coverage provided and the overall quality of the TCK.

What is conformance testing?

- Test what is specified and only what is specified.
  - Implementation-specific features are off-limits.
- Typically we “don’t care” about performance, efficiency, usability, security, robustness, or even quality!
  - Unless these are specified (usually they aren’t.)
- Make no assumptions about the environment in which the implementation is run.
- Assume no knowledge of implementation internals.
  - Black-box testing only.

What makes a good spec?

- Developing TCK tests improves the quality of the spec.
  - If you start early enough, and provide feedback to the Expert Group.
- Specify.
  - Unspecified or implementation-specific behavior can't be tested.
- Require.
  - In clear, unambiguous language (see RFC 2119.)
  - We like “must”, “shall”, “shall not”...
  - We don't like “may”, “it’s up to you”, “it's obvious”...
- Avoid optional functionality.
  - Can be tested, but doesn't promote WORA (developers won't know what they can depend on.)

What is (should be) in the TCK?

- The tests (in source as well as binary form.)
- Documentation.
- Automation harness or script.
- Compatibility Requirements.
- Definition of *Test Appeals* process.
- Exclude list (list of tests that need not be run.)

Compatibility Requirements

- Specify what implementations must do – in addition to passing the tests – to be compatible.
- At a minimum must require implementations to:
  - fully implement the Spec(s) including all required interfaces and functionality.
    - whether or not the TCK tests these features!
  - do not modify, subset, superset, or otherwise extend the implementation name space.
- Typically contains many other requirements, eg:
  - Must be able to pass all the tests in all configurations.
  - Don't modify the tests or the test harness.

What makes a good TCK?

- Not just “coverage”
- Are the tests focused where it's most important?
- Are the tests correct?
- Are they efficient?
- Are they portable?
- Are they documented?
- Are they maintainable?

How to decide what to test

- It is impossible to “completely test” any non-trivial Spec.
- You must consciously decide what not to test.
- Focus on areas:
  - Where the risk of incompatibility is greatest.
    - Where implementation is difficult.
    - Where implementers may have an incentive to “tweak” the code (or even to cheat.)
  - Where the consequences of incompatibility would be greatest.
    - Security breaches.
    - Application breakage.
  - Where you will get the biggest bang for the buck.
    - Fundamental classes, mainline code, etc.

Measuring coverage

- Identify normative requirements (*Test Assertions*) within the specification:
  - java.lang.Integer.toString(int i, int radix)
    - "If the radix is smaller than Character.MIN_RADIX or larger than Character.MAX_RADIX, then the radix 10 is used instead."
  - java.lang.Integer.parseInt(String s)
    - "Throws: NumberFormatException if the String does not contain a parsable int."
- For each area of the spec (class, functional area, chapter…) measure the *breadth* of your assertion coverage (the percentage of assertions that are tested) and estimate the *depth* of that coverage (how thoroughly the assertions are tested.)

Specification mark-up and coverage reporting

- Mark-up the specification (identifying the assertions.)
- Provide feedback to the Expert Group where the spec is ambiguous, incomplete, or untestable.
- The Spec Lead should review and approve your assertion list.
- This process can significantly improve spec quality.
- Provide a *coverage report* (as required by the JCP.)
- This will help implementers to understand where the TCK is strong and where it is weak (and hence where they should do extra testing.)
- It will also help you to further develop the TCK in subsequent releases.

Testing the TCK

- Your job is to test the TCK, not the RI.
- If all the TCK tests pass on the RI this may be good for the RI but it tells you very little about the quality of your TCK.
  - How do you know that the TCK will detect bugs in implementations?
  - You must exercise the negative code-paths.
  - How can you make the TCK fail?
  - How do you know that the TCK will provide useful results on other implementations?
- Testing a TCK is hard!
  - You must verify that it will run correctly on implementations that do not yet exist.

The test challenge process

- You must define a test-appeals process enabling implementers to assert that a test is invalid:
  - because of a bug in the test (for example, a logic error, or incorrect interpretation of spec)
  - because of a bug in the spec (for example, its requirements are contradictory)
  - because the test is biased towards a particular implementation.
- You must investigate such claims, and provide remediation.
- Invalid tests can be added to an *exclude list* and need not be run.
- You may provide alternate tests or an updated test suite
- This should be done when excluding tests would significantly weaken the TCK.

See also

- The *Developing TCKs* section of the Spec Lead Guide.

Thank you!

http://jcp.org