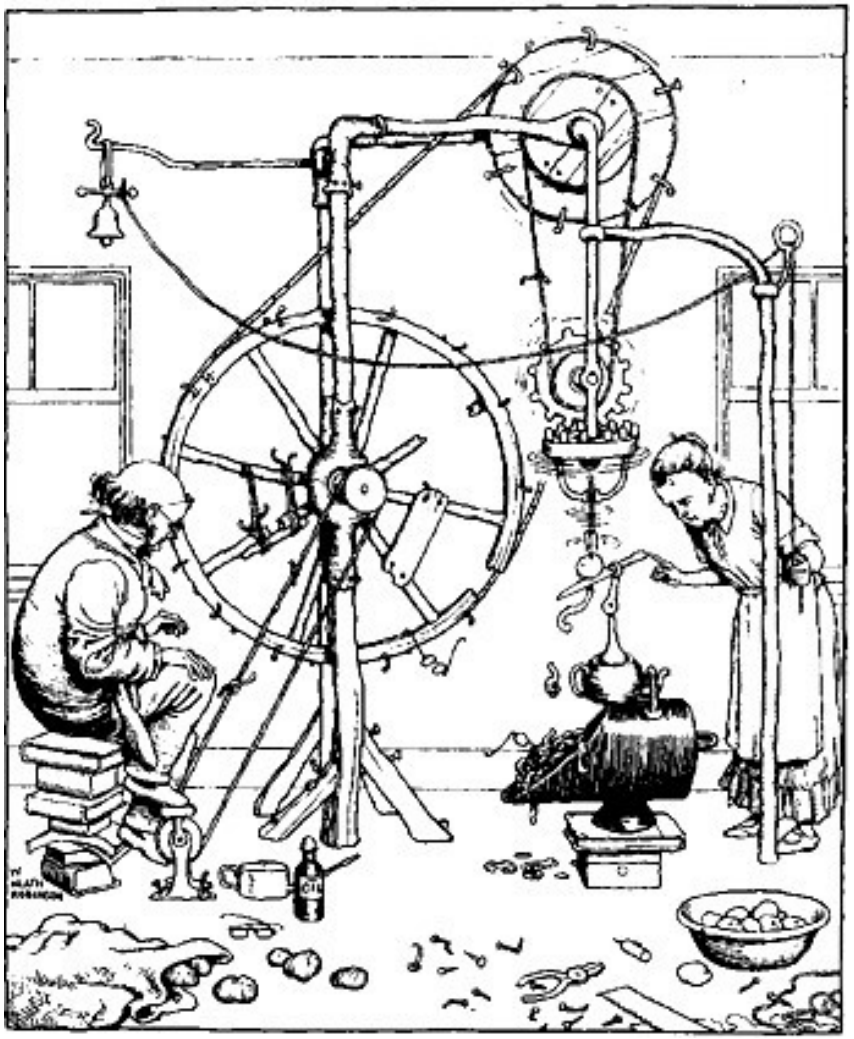
Conformance testing and standards



How do you know it works if you don't know what it's supposed to do?

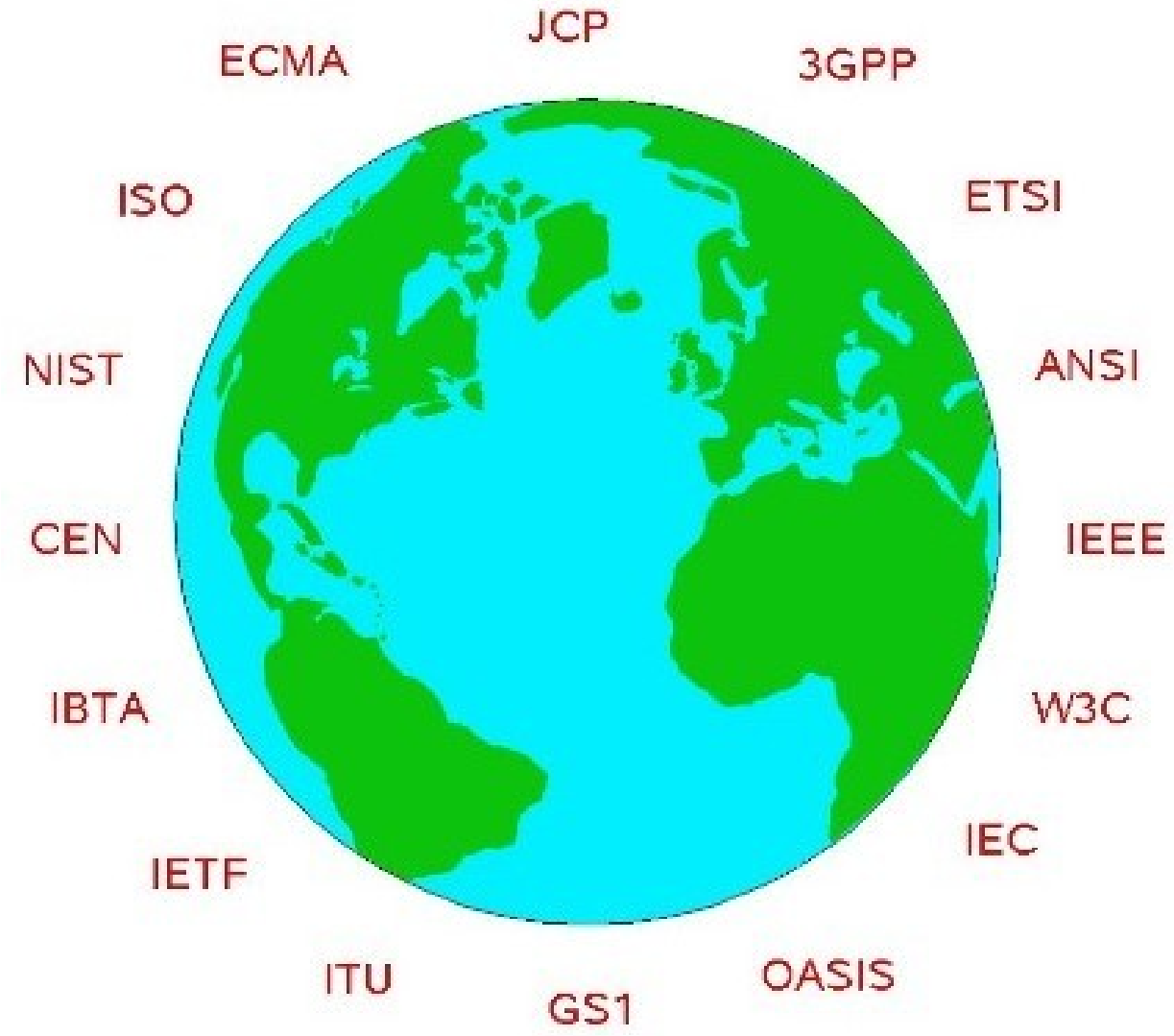
Patrick Curran
JCP Chair
patrick@jcp.org

http://jcp.org

ORACLE

Java Community Process

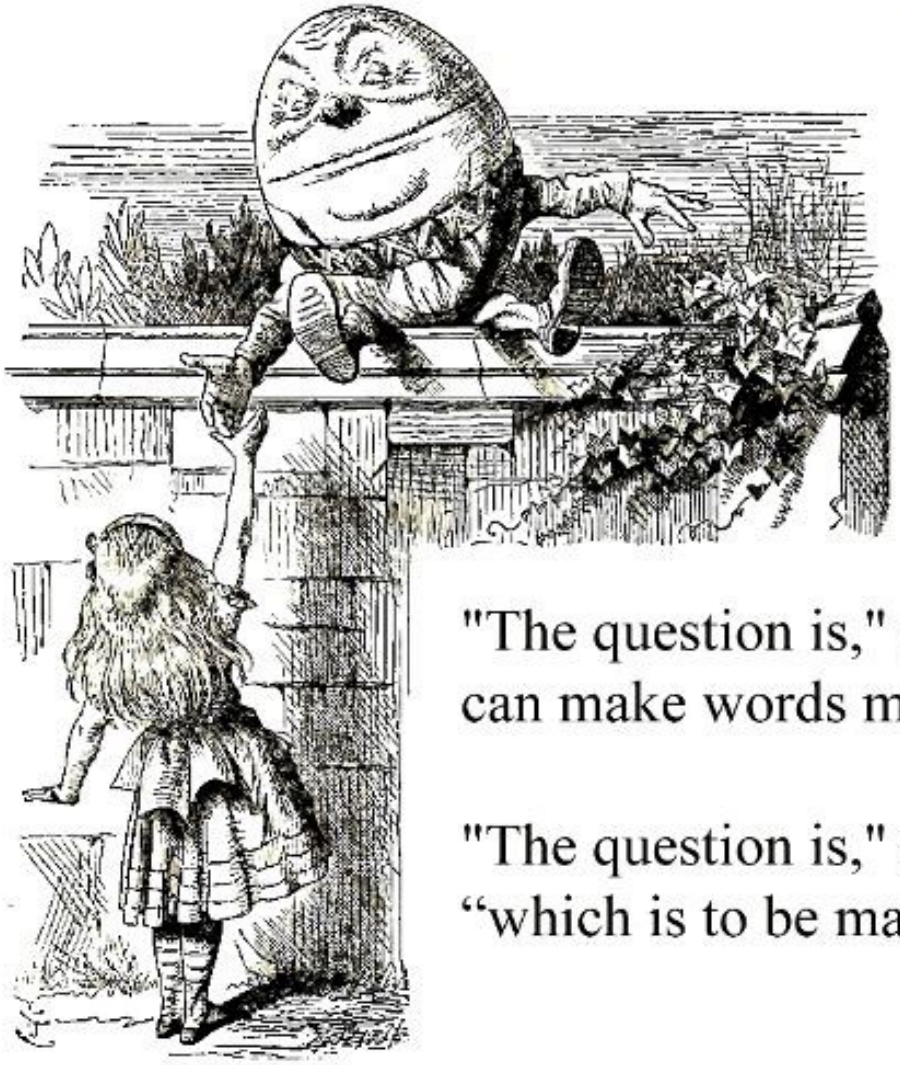
Standards make the world go round



http://jcp.org

Java Community Process

Language

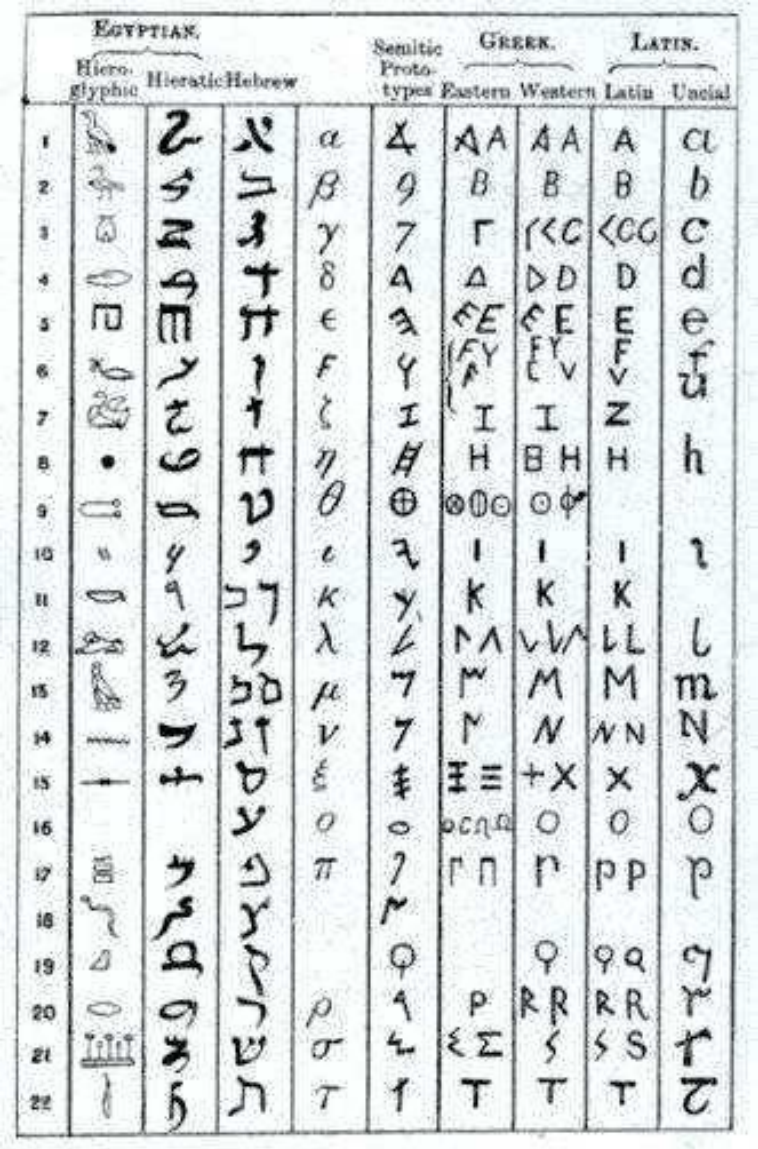


"When I use a word,"
Humpty Dumpty said,
in rather a scornful tone,
"it means just what
I choose it to mean -
neither more nor less."

"The question is," said Alice, "whether you can make words mean so many different things."

"The question is," said Humpty Dumpty, "which is to be master - that's all."

Writing



Java Community Process

Number systems

Currency

Time...

http://jcp.org

Java Community Process

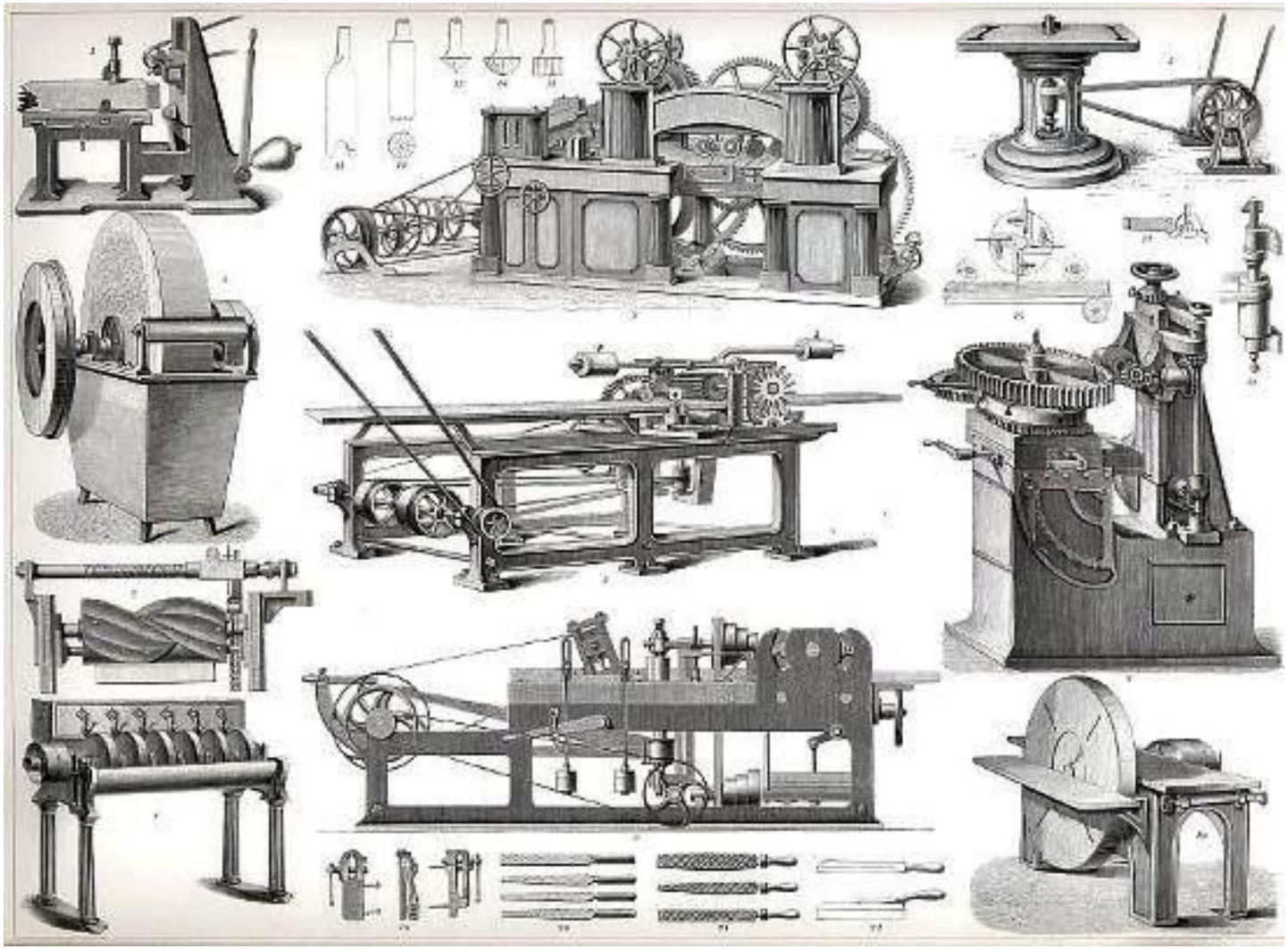
And space

http://jcp.org

Java Community Process

Weights and measures

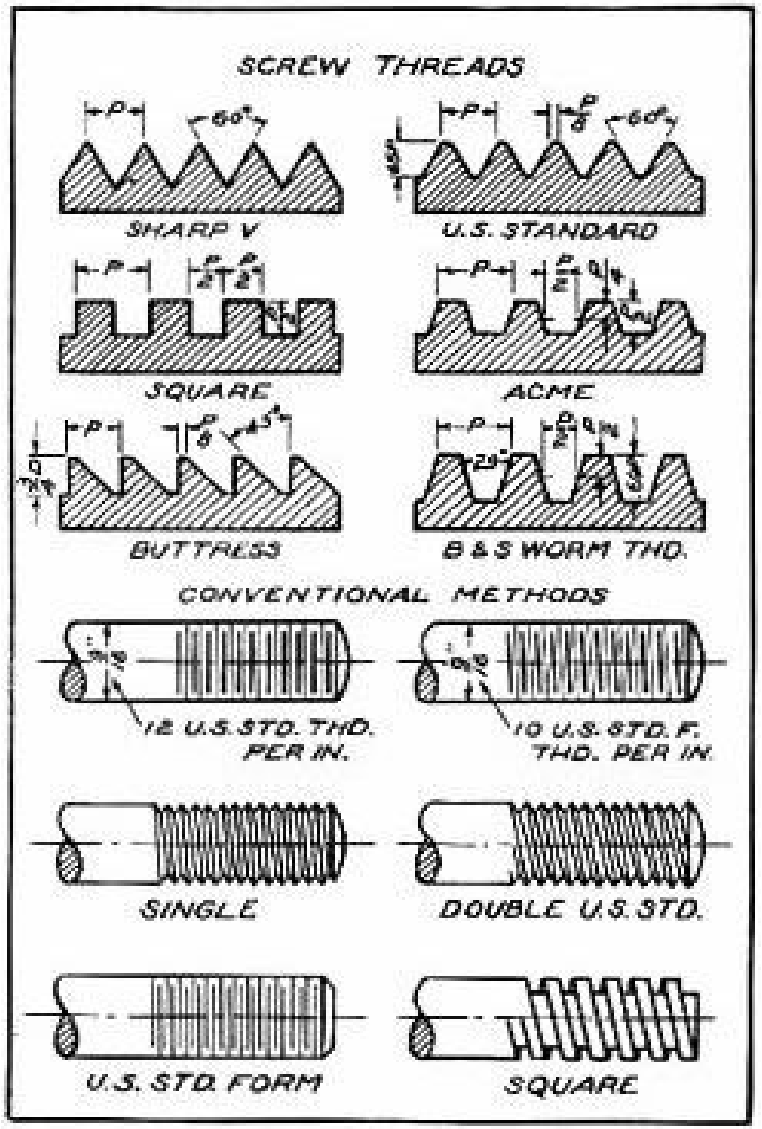
Machine tools



http://jcp.org

Java Community Process

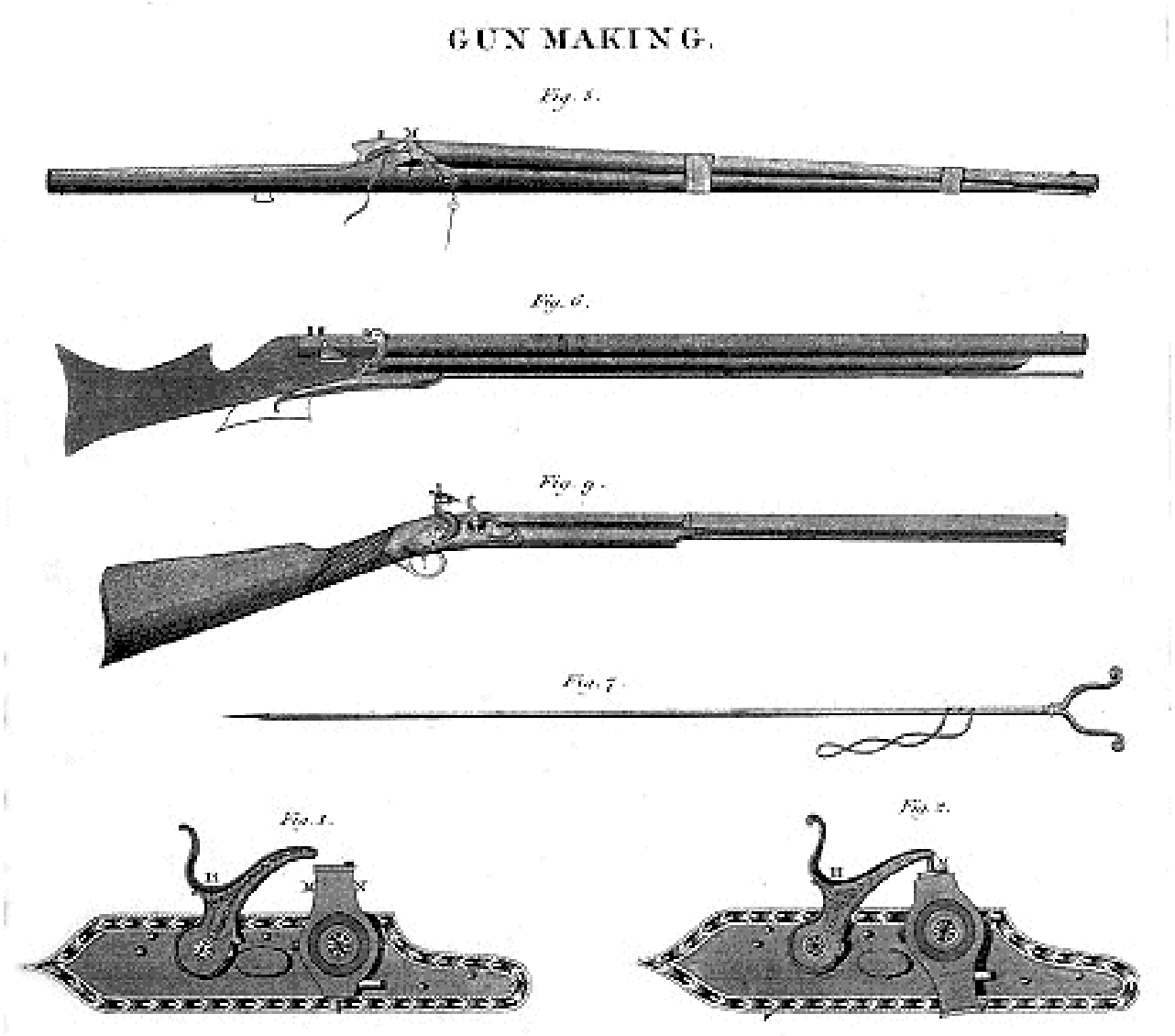
Screws and threads



http://jcp.org

Java Community Process

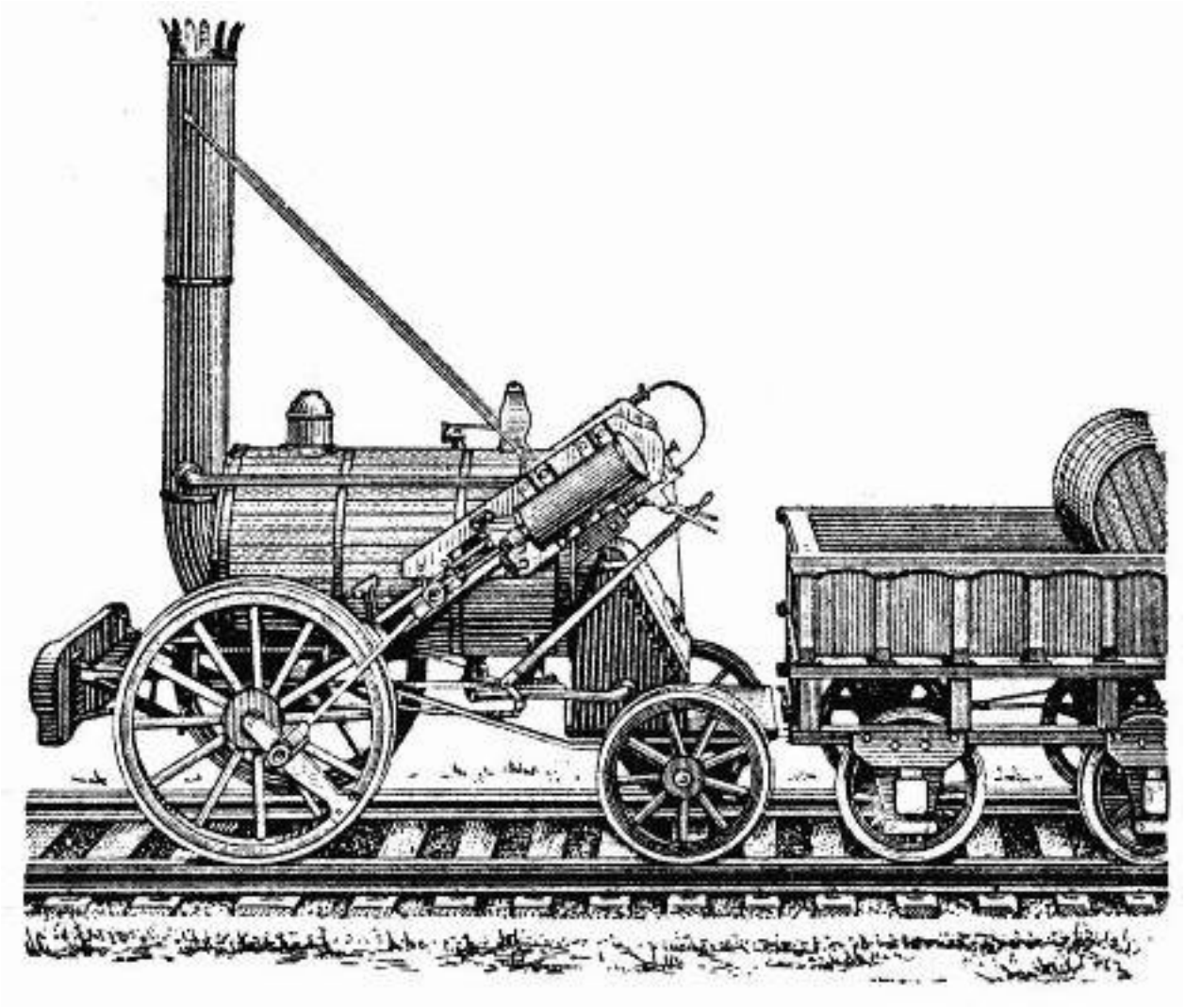
Interchangeable parts



http://jcp.org

Java Community Process

Railways



Mass production

Shipping

Traffic

Beer

http://jcp.org

Java Community Process

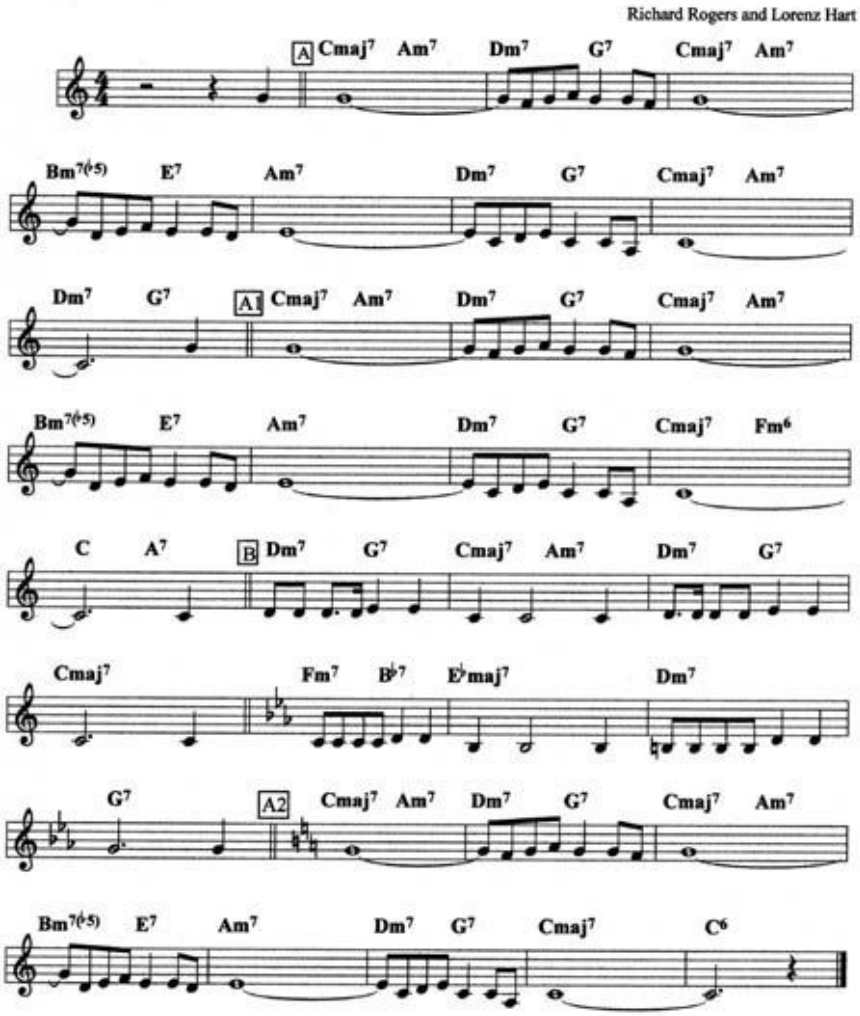
Chocolate

WHO/FAO: Codex Alimentarius Official Standard for Chocolate

http://jcp.org

Java Community Process

Music



ISO 16:1975 Acoustics -- Standard tuning frequency
(Standard musical pitch)

http://jcp.org

Java Community Process

Color

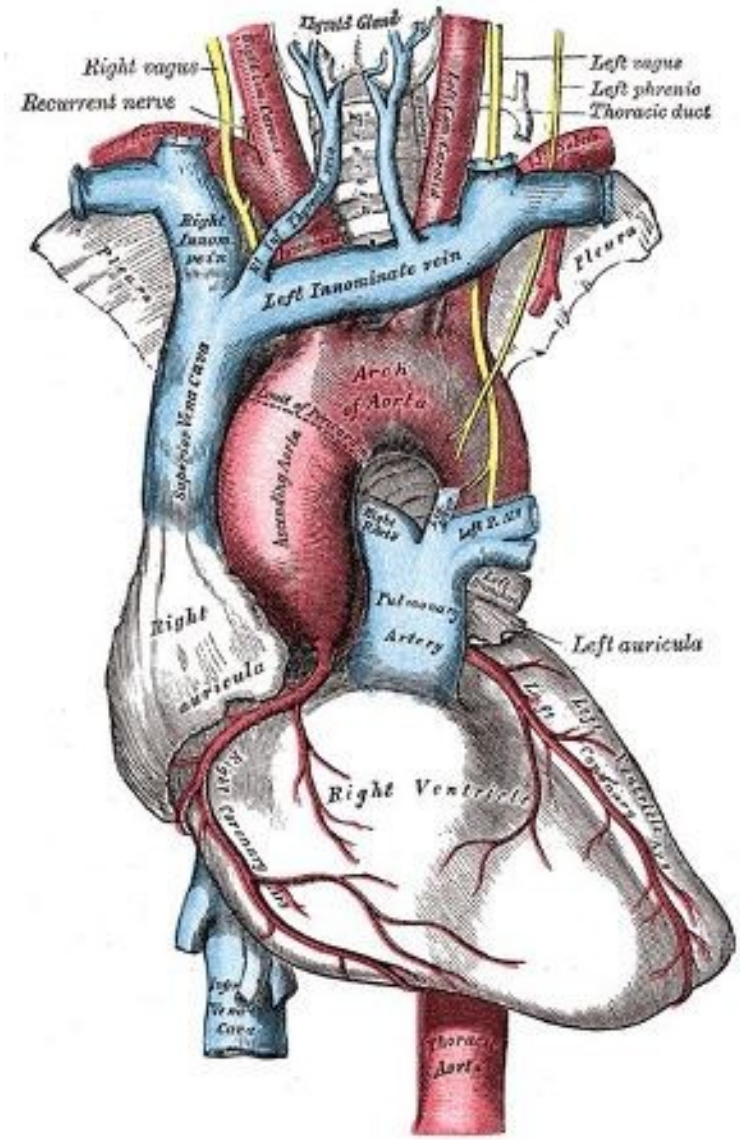
http://www.color.org

Sport

http://www.lords.org/laws-and-spirit/laws-of-cricket/laws/

http://jcp.org

Java Community Process



Chronic rheumatic heart diseases

I05: Rheumatic mitral valve diseases

Includes:
conditions classifiable to 105.0 and 105.2-105.9, whether specified as rheumatic or not
Excludes:
when specified as nonrheumatic
I05.0: Mitral stenosis
Mitral (valve) obstruction (rheumatic)
I05.1: Rheumatic mitral insufficiency
Rheumatic mitral
• Incompetence
•Regurgitation
I05.2: Mitral stenosis with insufficiency
Mitral stenosis with incompetence or regurgitation
I05.8:Other mitral valve diseases
Mitral (valve) failure
I05.9: Mitral valve disease, unspecified
Mitral (valve) disorder (chronic) NOS

From the World Health Organization International Classification of Diseases

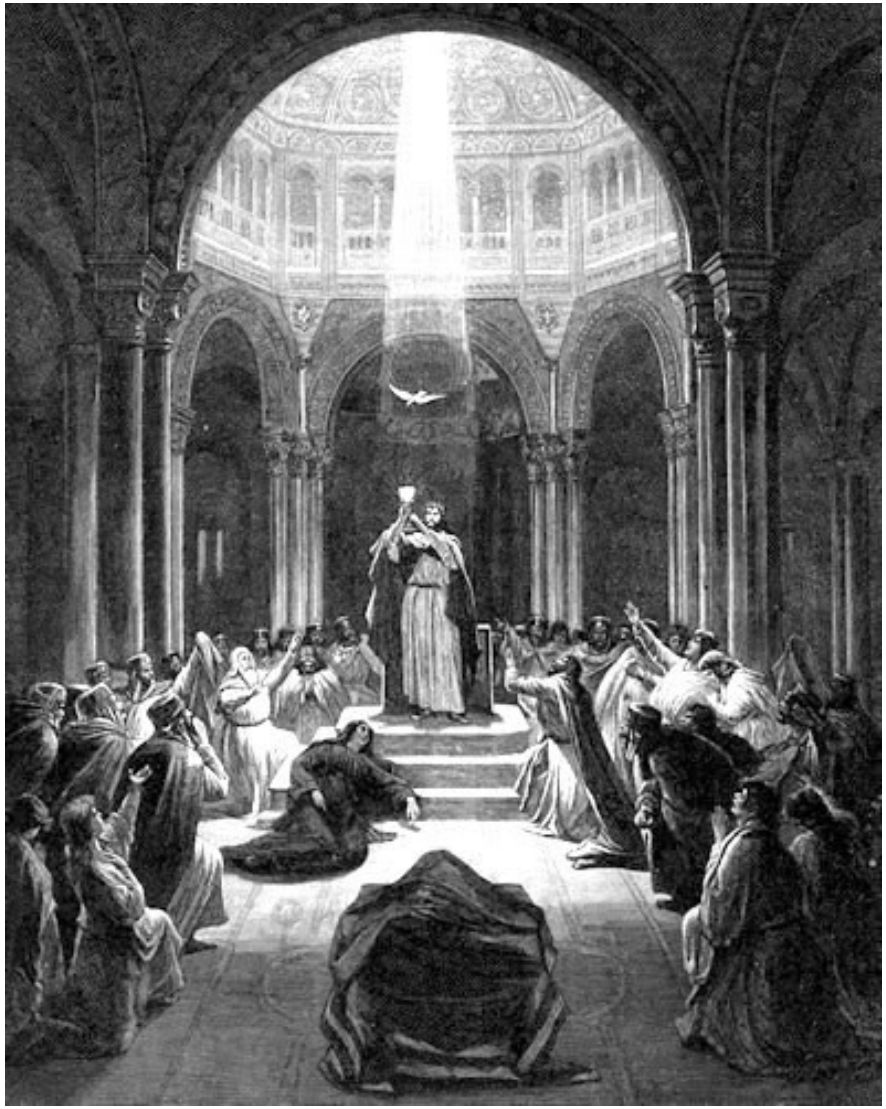
Shopping

http://jcp.org

Java Community Process

Home entertainment

The Holy Grail



- Components, systems, and processes that are:
  - Reproducible
  - Predictable
  - Reusable
  - Interoperable
  - Interchangeable

http://jcp.org

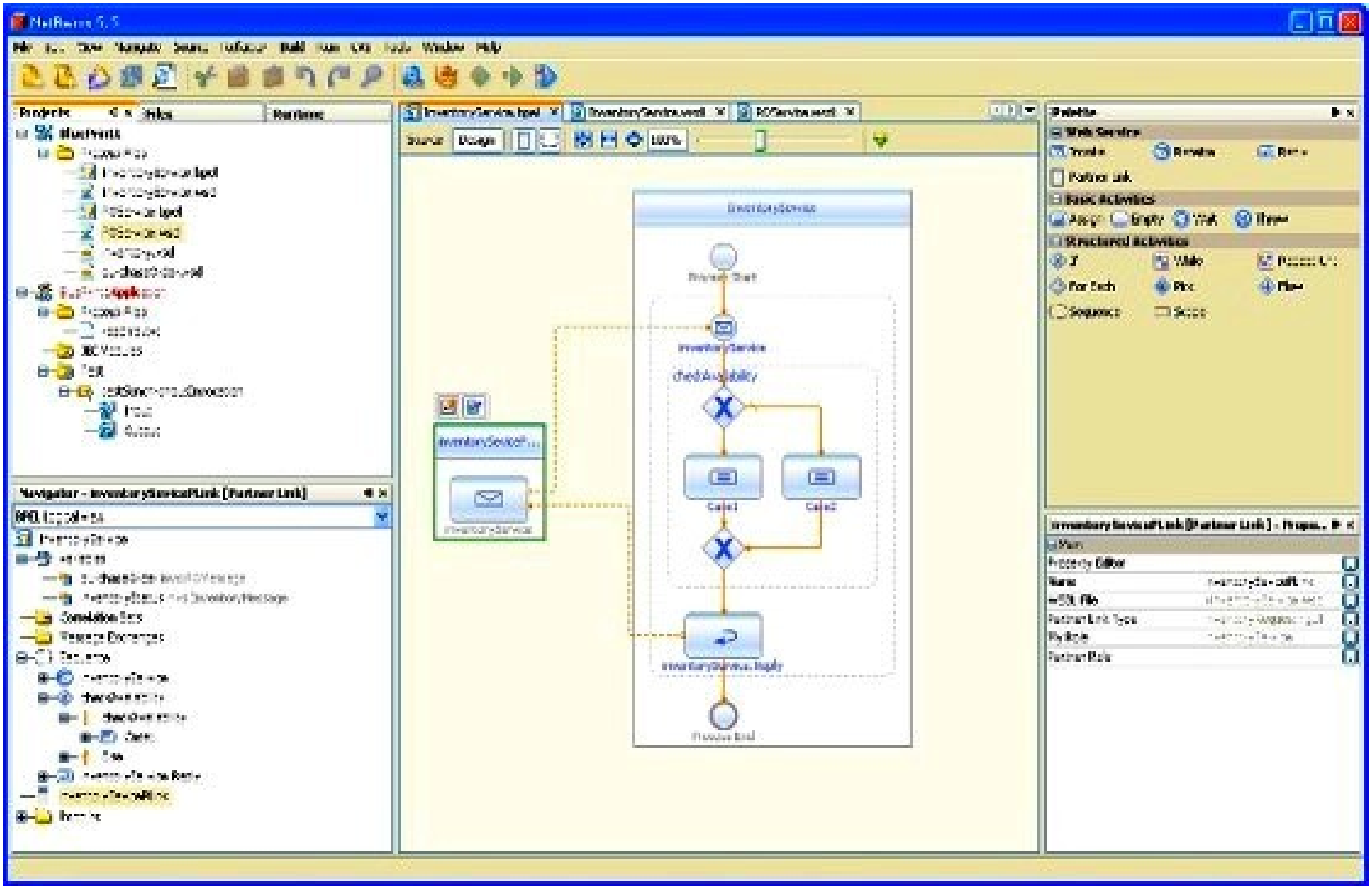
Java Community Process

Then

Now

- We are no longer willing to buy all of our hardware and software from a single supplier.
- We want the freedom to chose and the option to switch.
  - All systems are heterogeneous.
- This requires standards.
  - For interfaces, so we can mix and match components.
  - For protocols, so systems can talk to each other.

Interfaces



Protocols

http://jcp.org

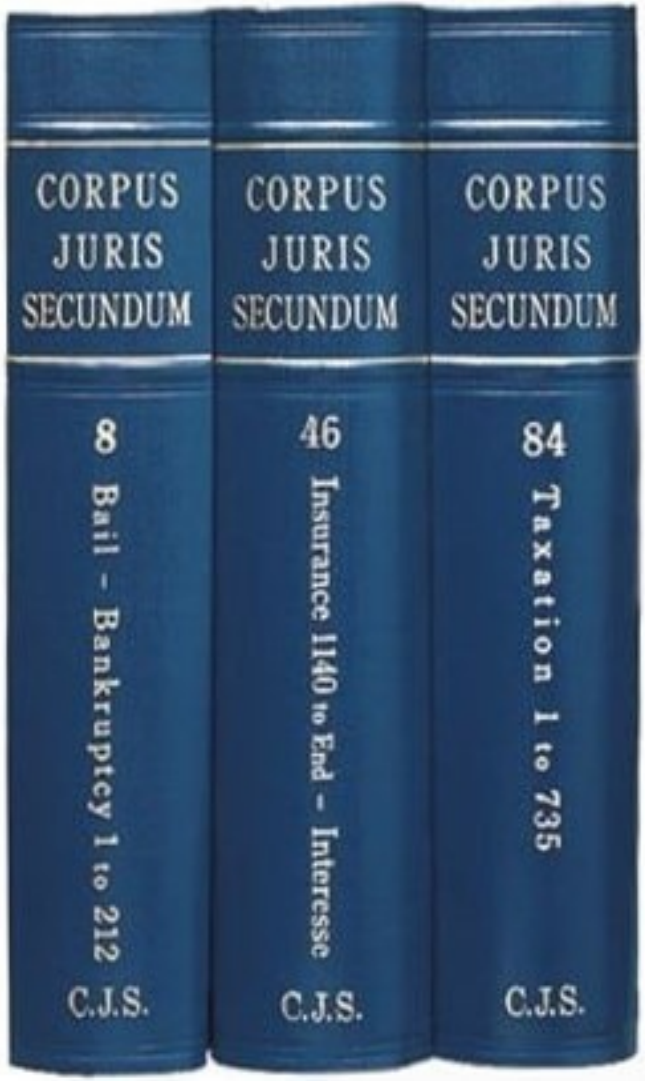
Java Community Process

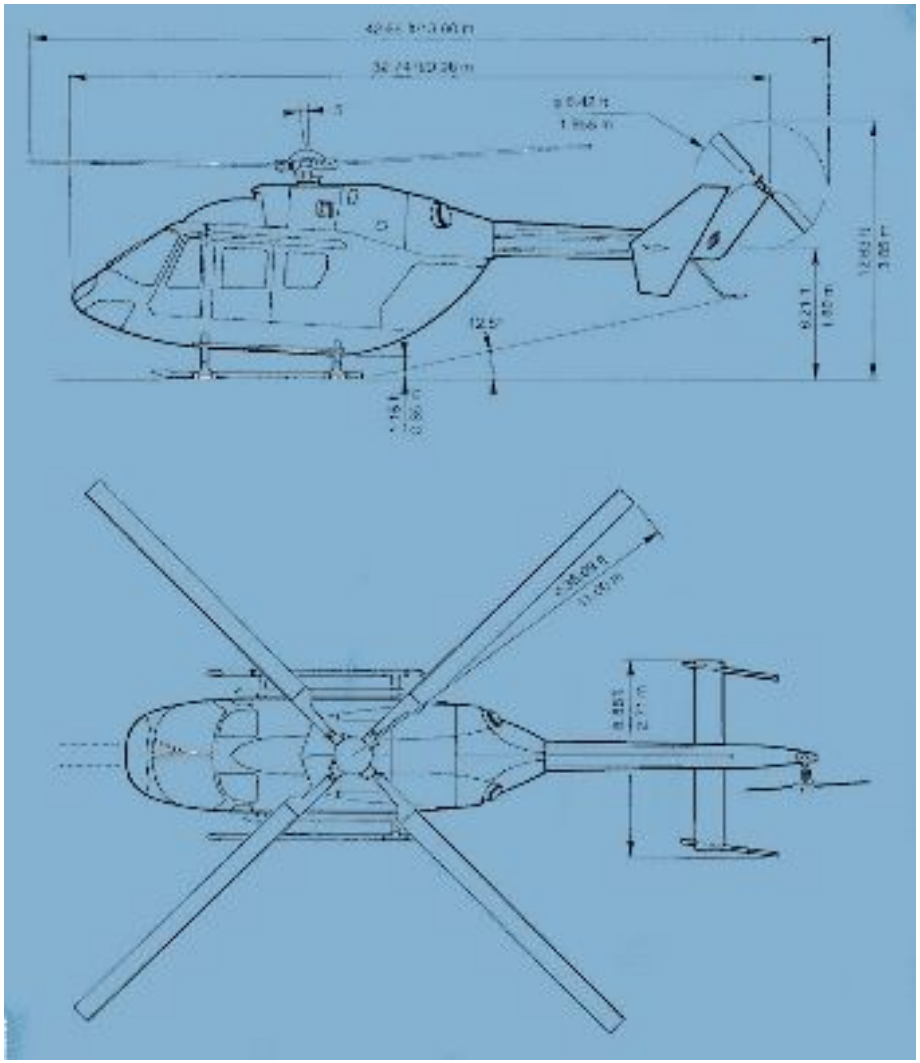
No vendor lock-in

http://jcp.org

Java Community Process

Specifications...





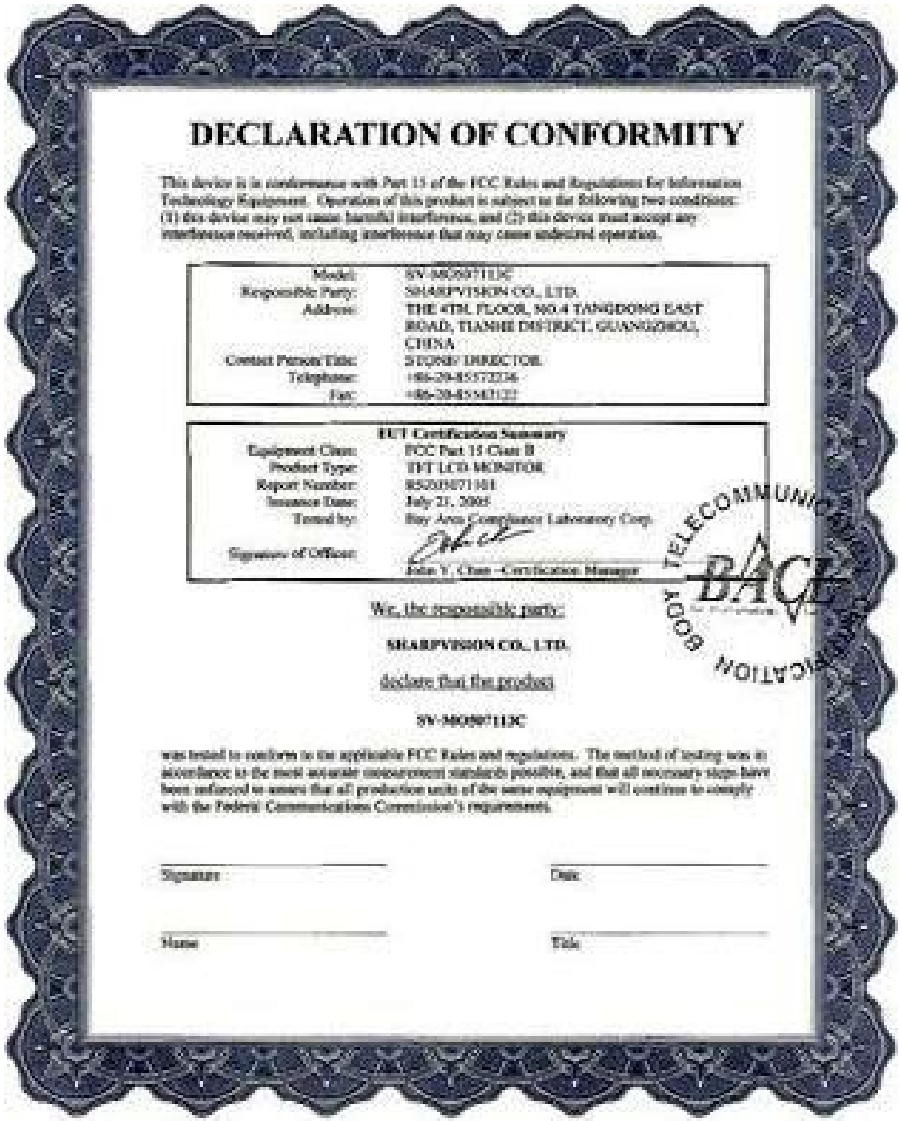
http://jcp.org

Java Community Process

Are not enough...

- Interoperability and interchangeability are harmed by:
- Poor-quality specs.
  - Imprecise language.
    - Implementors interpret the spec differently.
  - Too much optionality.
    - Unspecified behavior cannot be tested at all.
    - Specified but optional behavior can be tested, but hampers interoperability (developers don't know what they can rely on.)
- Poor-quality implementations.
  - Specified requirements are not met.
  - Specified requirements are implemented incorrectly.

We also need testing



DECLARATION OF CONFORMITY

This device is in conformance with Part 15 of the FCC Rules and Regulations for Information Technology Equipment. Operation of this product is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

Model:	SV-M0597113C
Responsible Party:	SHARPVISION CO., LTD.
Address:	THE 4TH. FLOOR, NO.4 TANGDONG EAST ROAD, TIANHE DISTRICT, GUANGZHOU, CHINA
Contact Person/Title:	SIDONG DIRECTOR
Telephone:	+86-20-85572236
Fax:	+86-20-85587122

EUT Certification Summary	
Equipment Class:	FCC Part 15 Class B
Product Type:	TFT LCD MONITOR
Report Number:	85203071101
Issuance Date:	July 21, 2005
Tested by:	Bay Area Compliance Laboratory Corp.
Signature of Officer:	John Y. Chan - Certification Manager

We, the responsible party:

SHARPVISION CO., LTD.

declare that the product

SV-M0597113C

was tested to conform to the applicable FCC Rules and regulations. The method of testing was in accordance to the most accurate measurement standards possible, and that all necessary steps have been enforced to assure that all production units of the same equipment will continue to comply with the Federal Communications Commission's requirements.

Signature: Date:

Name: Title:

http://jcp.org

Java Community Process

Conformance testing

- The process of verifying that implementations of a technology conform to the specification.
- Focuses precisely (solely) on testing functionality that is normatively required in the specification.
  - Quality, robustness, performance, usability, and other desirable attributes of software can not and must not be tested (unless explicitly specified.)
- Can make no assumptions about the internals of the implementation (black-box testing.)
- Improves the quality of specifications:
  - by identifying ambiguities, contradictions, omissions,
- And of implementations:
  - by identifying failures to conform to the spec.

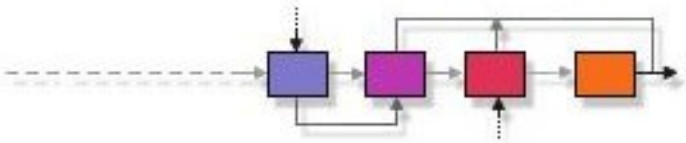
What makes a good spec?

- Specify.
  - Unspecified or implementation-specific behaviour can't be tested.
- Require.
  - In clear, unambiguous language (see RFC 2119)
  - We like “must,” “shall,” “must not”...
  - We don't like “may,” “it's obvious,” “it's up to you”...
- Beware optional functionality.
  - Can be tested, but doesn't promote interoperability or application portability.
    - Developers won't know what they can depend on.
  - If you must, clearly define optionality with Profiles.

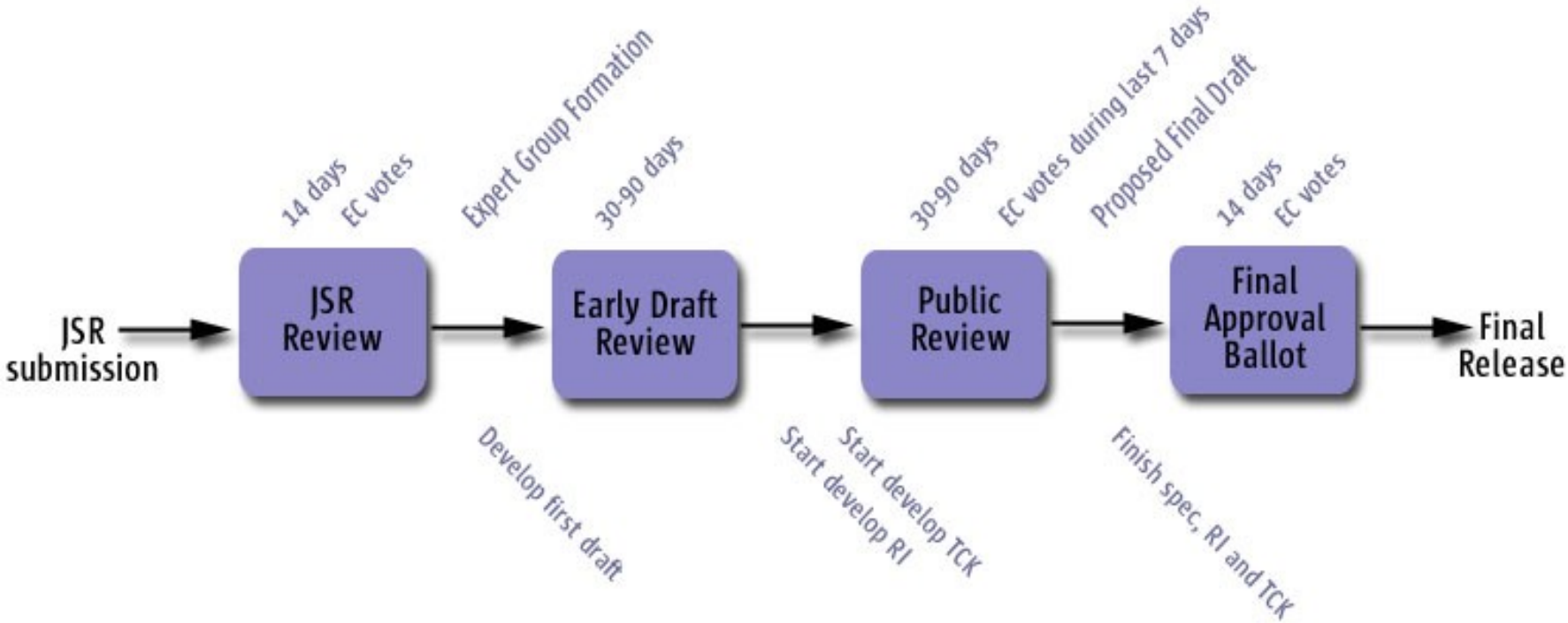
Java: what?

Open, standards-based technologies enabling the development and deployment of secure, portable, reliable, and scalable applications on hardware platforms from cellphones to high-end servers.

Java: how?



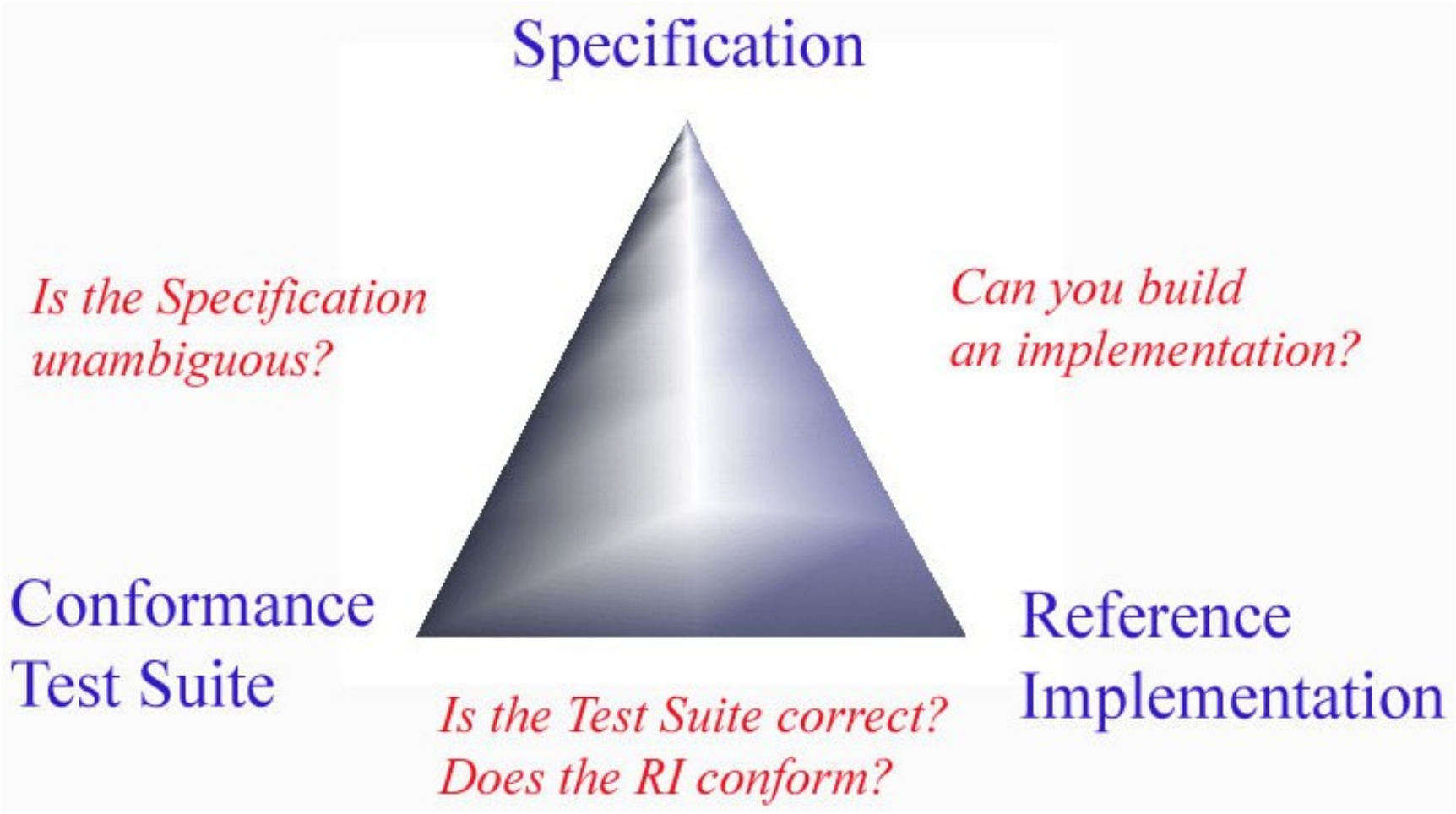
Community Development of
Java Technology Specifications



http://jcp.org

Java Community Process

Deliverables



Deliverables

http://jcp.org

Java Community Process

Java conformance testing

http://jcp.org

Java Community Process

Why conformance-test?

- To promote the compatibility and interoperability of Java technology implementations by ensuring that the technologies are well specified and that implementations conform to the specifications.
- Results:
  - Implementors know how to develop compatible products.
    - Multiple compatible implementations are available.
  - Developers know what to expect from an implementation.
    - Applications behave identically on compatible implementations.

Java conformance (1)

- Compatibility is a contractual obligation.
  - Shipping incompatible products is prohibited.
- Compatible products can use the Java name and display the *Java Compatible* logo.
- Compatibility is binary.
  - You can't be “almost compatible” or “a little bit incompatible.”
  - If the TCK contains 100,000 test cases and you pass 99,999, you are not compatible.
  - If you pass 100,000 tests and you don't meet all other requirements, you are not compatible.

Java conformance (2)

- Implementors self-certifiy that they are compatible.
- They assert that they are capable of passing all tests in all possible configurations.
  - For any significant technology it is impossible in practice to run all combinations.
- They must also assert that other *Compatibility Requirements* have been met.
- The Spec Lead has the option to audit results and to investigate complaints.
- Level playing-field: everyone including (especially!) Oracle must comply.

Testing is not enough...

- Coverage will never be complete.
- Compatibility Requirements (*The Rules*) define additional conditions that must be met.
  - Typically documented in the *TCK User's Guide*.
- Specify what it means to pass the test suite.
  - All required tests must pass.
  - Tests must not be modified.
- Specify additional (possibly untested, or even untestable) criteria that must be met.
  - All specified classes and interfaces must be provided and must function as specified.
  - Syntax and semantics of the Java language must be preserved.

Fundamental *Rules*

- "The Product must contain the full set of public and protected classes and interfaces for all the Libraries. Those classes and interfaces must contain exactly the set of public and protected methods, constructors, and fields defined in the Specifications for those Libraries. No subsetting, supersetting, or modifications of the public and protected API of the Libraries are allowed except only as specifically exempted by these Rules."
- "The functional programmatic behavior of any binary class or interface must be that defined by the Specifications."
- Translation: ***Implement the API – no more and no less – as specified. Even if we don't test it!***

No matter how hard you try...

- The test suite will contain bugs
- The Spec Lead must define a test appeals process allowing implementors to assert that they should not be required to pass a test because it is invalid.
- Possible justifications:
  - A bug in the test (eg, a logic error, or an incorrect interpretation of the spec.)
  - A bug in the spec (eg, contradictory requirements.)
  - The test is biased to a particular implementation (typically, the RI, against which the test suite itself was tested.)
- Invalid tests are added to the *exclude list* and need not be executed.

Alternate tests

- The Spec Lead may, but is not obliged to, provide alternate tests to replace those that are excluded.
- Alternate tests should be provided when:
  - Exclusions would significantly weaken the TCK, or
  - The consequence of incompatible implementations would be significant.

Planning and building a high-quality TCK

A TCK is not just a collection of tests

- It should also include:
  - A test harness to automate execution.
  - Documentation explaining
    - How to run the test suite.
    - How to interpret test results.
    - Compatibility Requirements (*The Rules.)*
    - The test appeals process.
- It must be portable.
  - Unlike most other software, a TCK must be capable of running on systems that don't yet exist.
    - You can't test it on the platforms where it will be run!
- The Spec Lead must commit to ongoing maintenance.
  - Fix bugs, expand coverage.

A good test is...

- Mappable to the specification.
  - You must know what portion of the specification it tests.
- Atomic.
  - Tests a single feature rather than multiple features.
- Self-documenting.
  - Explains what it is testing and what output it expects.
- Focused on the technology under test rather than on ancillary technologies.
- Useful.
  - Likely to catch real-world problems.
- Correct, efficient, portable, and maintainable.

Specification markup

- Identify normative requirements (test assertions) within the spec.
- Provide feedback to the authors where the spec is ambiguous, contradictory, incomplete, or untestable.
- Publish an assertion list.
  - Ask the spec authors to review and approve it.
- This process significantly improves spec quality.

Test assertions

- A specific statement of functionality or behavior derived from a 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."
  - “During preparation of a class or interface C, the Java virtual machine also imposes loading constraints (§5.3.4). Let L1 be the defining loader of C. For each method *m* declared in C that overrides a method declared in a superclass or superinterface, the Java virtual machine imposes the following loading constraints: Let T0 be the name of the type returned by m, and let T1, ..., Tn be the names of the argument types of *m*. Then $Ti^{L1}=Ti^{L2}$ for $i = 0$ to n (§5.3.4).”

How many tests are enough?

- There is no simple answer to this question.
  - It depends on your goals and on the available resources.
- What is most important is that you get the optimal coverage with the resources you can apply.
- You cannot do this unless you set explicit goals, and measure or estimate test coverage.

Calculating coverage

Coverage analysis

- Partition the spec.
  - By feature, APIs, language elements, testable assertions, logical sections, or even pages or paragraphs.
- Estimate or measure the extent of coverage in each area
- Breadth coverage (relatively simple)
  - What percentage of spec elements are covered by at least one test?
- Depth coverage (more subjective)
  - On average, what percentage of the tests that would be required to ***completely*** test each element have actually been written?
    - (How thoroughly is each element tested?)

Test development strategy

- Define coverage goals.
  - Where should resources be focused?
  - How extensively should each area be tested?
- Start with breadth (test everything minimally.)
- Drill down (increase depth coverage) in selected areas.
- Publish a test-coverage report.
  - Minimally, map tests to areas of the spec.
  - Ideally, provide counts and averages of the number of tests in each area.
- This helps users to understand the strengths and weaknesses of the test suite.
- It will also help you to improve the next version.

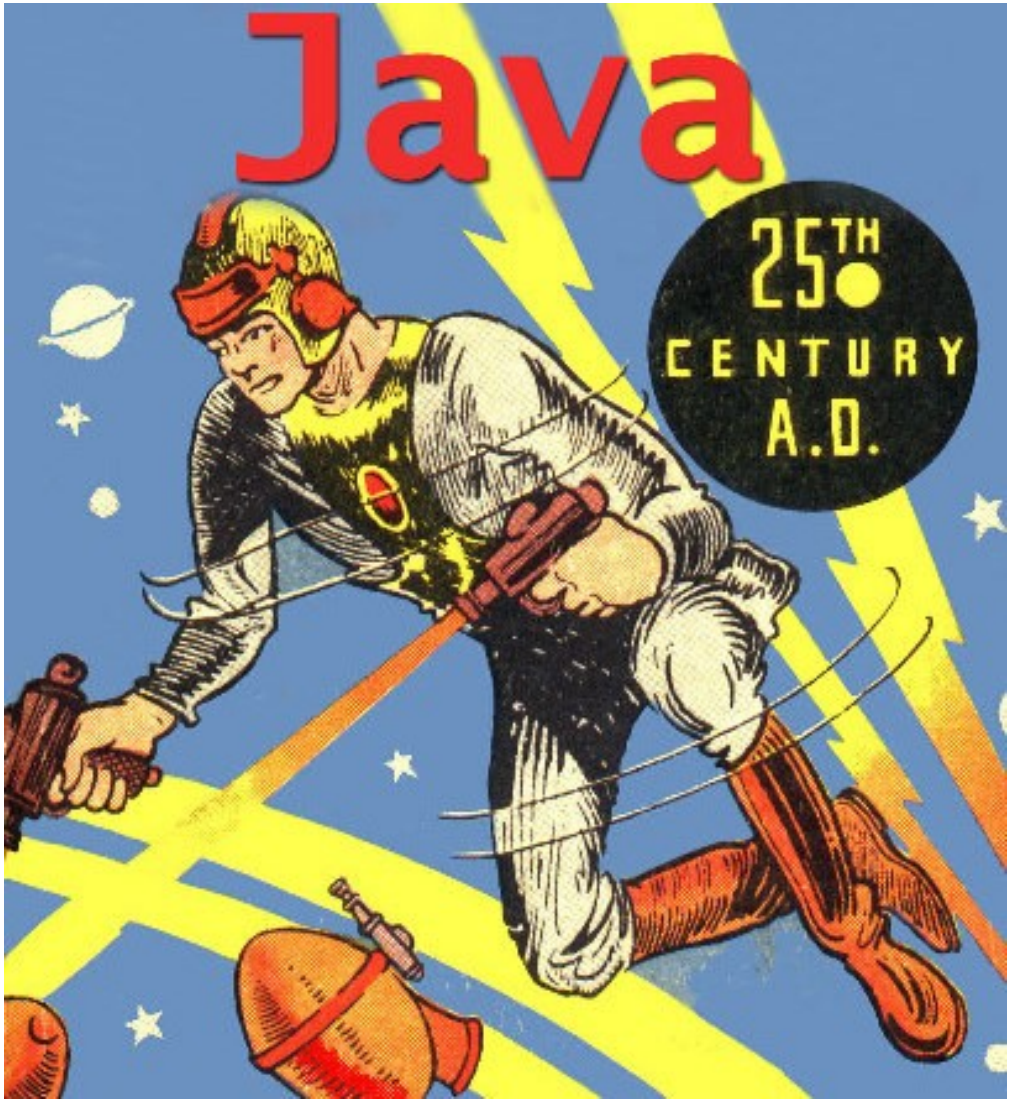
What to test and what not to test?

- "Full coverage" for the majority of real-world specs is impossible (impractical.)
- Don't just test what's easiest.
- Focus on areas where:
  - The consequences of non-conformance are greatest.
    - Eg, breaking interoperability or jeopardizing security.
  - Implementations are more likely to be non-conformant because:
    - Implementation presents technical difficulties.
    - The specification is ambiguous.
    - Implementers are less likely to discover problems.
    - Implementers have an incentive to cheat (eg, to increase performance.)

70% of what?

- If...
  - the spec is complete and unambiguous...
  - and we correctly identify all the normative requirements...
  - and every requirement is testable...
  - and we develop tests that cover an appropriate number of test cases for each requirement...
  - and our tests are correct (they do actually test the requirement and we have no bugs)...
  - and we don't exclude any tests...
- Then...
  - we have 100% “Coverage.”

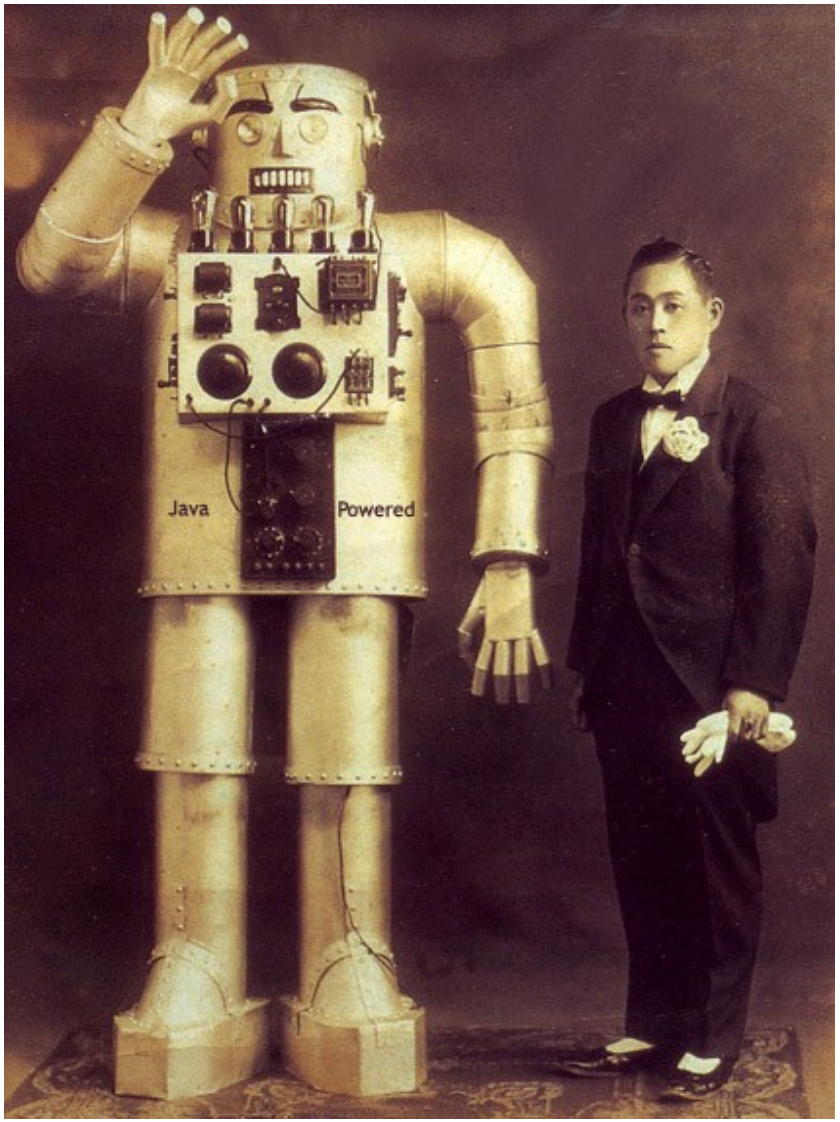
Spec + RI + TCK =



http://jcp.org

Java Community Process

Thank you!



http://jcp.org

Java Community Process