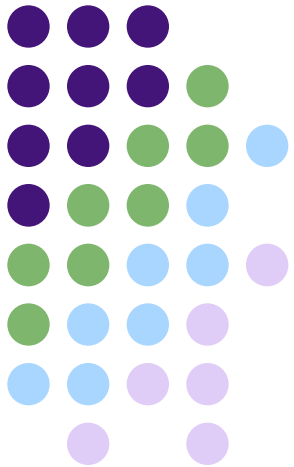
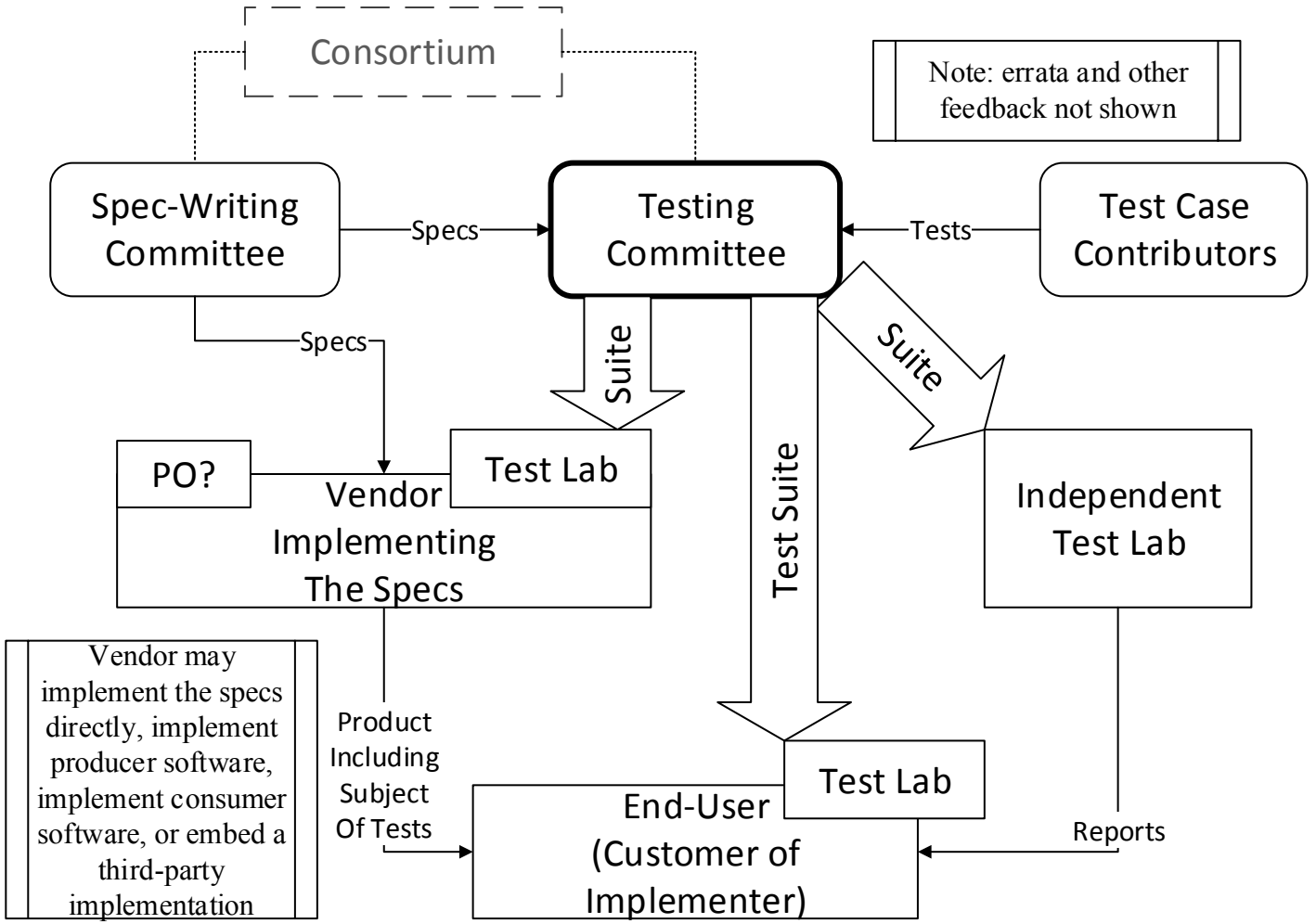
Test Case Management Systems and Metadata

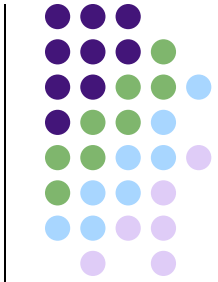
David Marston



Stakeholders

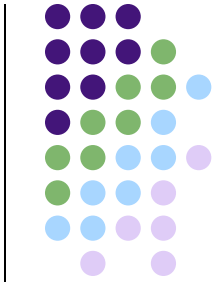


Scope and Scale



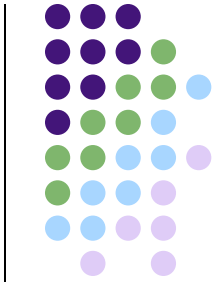
- There are written requirements that can be cited ("the specs")
- 5000 to 100,000 test cases
- There is some form of database or catalog record for every test case, which is where we put the metadata
- Atomic tests, exercising highly specific bits of behavior
- May need inputs when running tests

Functions of Metadata



- Tracking test cases during the development and review process
- Filtering test cases according to a variety of criteria (for example, whether or not they are applicable for a particular profile or optional feature)
- Identifying the area of the specification that is tested by each test case
- Parts are extracted when constructing a script to automatically execute the tests
- Formatting test results so that they are easily understood
- Providing data for some fields of a bug report

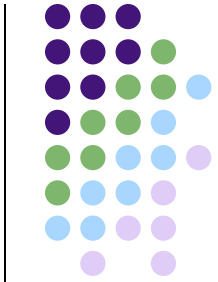
Example Metadata in XML



```
<test-case name="axes001-1">
  <description>Try child::* where several children exist</description>
  <file-path>axes</file-path>
  <spec-citation type="section" place="3.2.1.1" version="1.0"
   spec="xquery"/>
  <spec-citation type="anchor" place="axes" version="1.0"
   spec="xquery"/>
  <scenario operation="standard">
    <input-file role="query">axes001.xq</input-file>
    <input-file role="principal-data">TreeMany.xml</input-file>
    <output-file role="principal" compare="XML">axes001-1.xml</output-
   file>
  </scenario>
</test-case>
```

If we have a standard for this data, it would encourage vendors of test automation to use it, making their product more flexible.

Metadata Technology at work



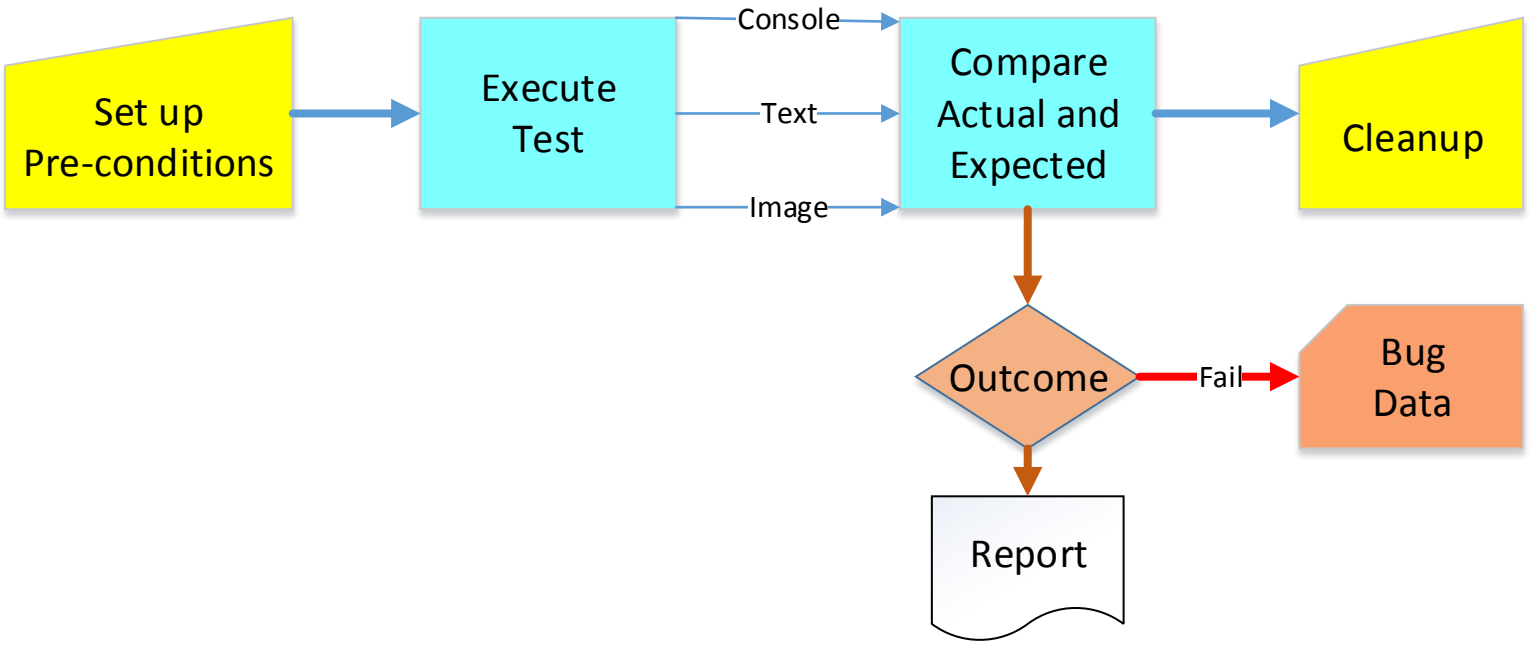
- We use XML so it can be transformed for many purposes
- Distinguishes each test case from all the others, which aids in bug isolation
- Testing Scenarios (setup, running test, comparing results, cleanup)
- Distinguishes prerequisites, pre-conditions, and input artifacts
- Could even be the only place where all the necessary artifacts are referenced at once
- All Dimensions of Variability accommodated
- Versions filtered by VersionAdd and VersionDrop
- Can even filter for errata on external specifications, if they are systematic
- Can embed status of each test case
- Directory of materials in a filesystem tree
- Contains pieces of text that can be assembled into scripts

Even the simplest metadata requires thoughtful design

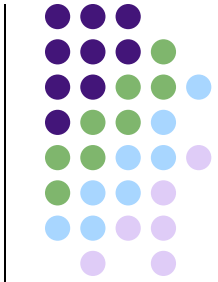
- **Identifier:** short, unique key string
- **Title:** unique, understandable by humans
- **Purpose:** one-liner (ideally, unique)
- **Description:** Detailed Explanation

Above are from W3C's 2005 *QA Working Group Note on Test Metadata*

Operational Scenarios

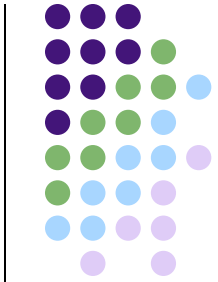


“Result” vs. “Outcome”



- Result is produced by the item under test
- Outcome is obtained by comparing the actual and reference results with the correct comparitor (could be separate open-source tools or tool-vendor opportunity)
- Outcomes are from an enumerated list: pass, fail, NA, couldn’t run, etc.

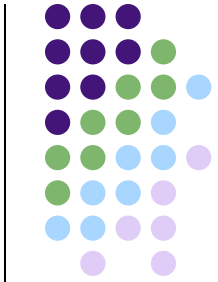
What About Prerequisites?



Should test case metadata identify other test cases as prerequisites?

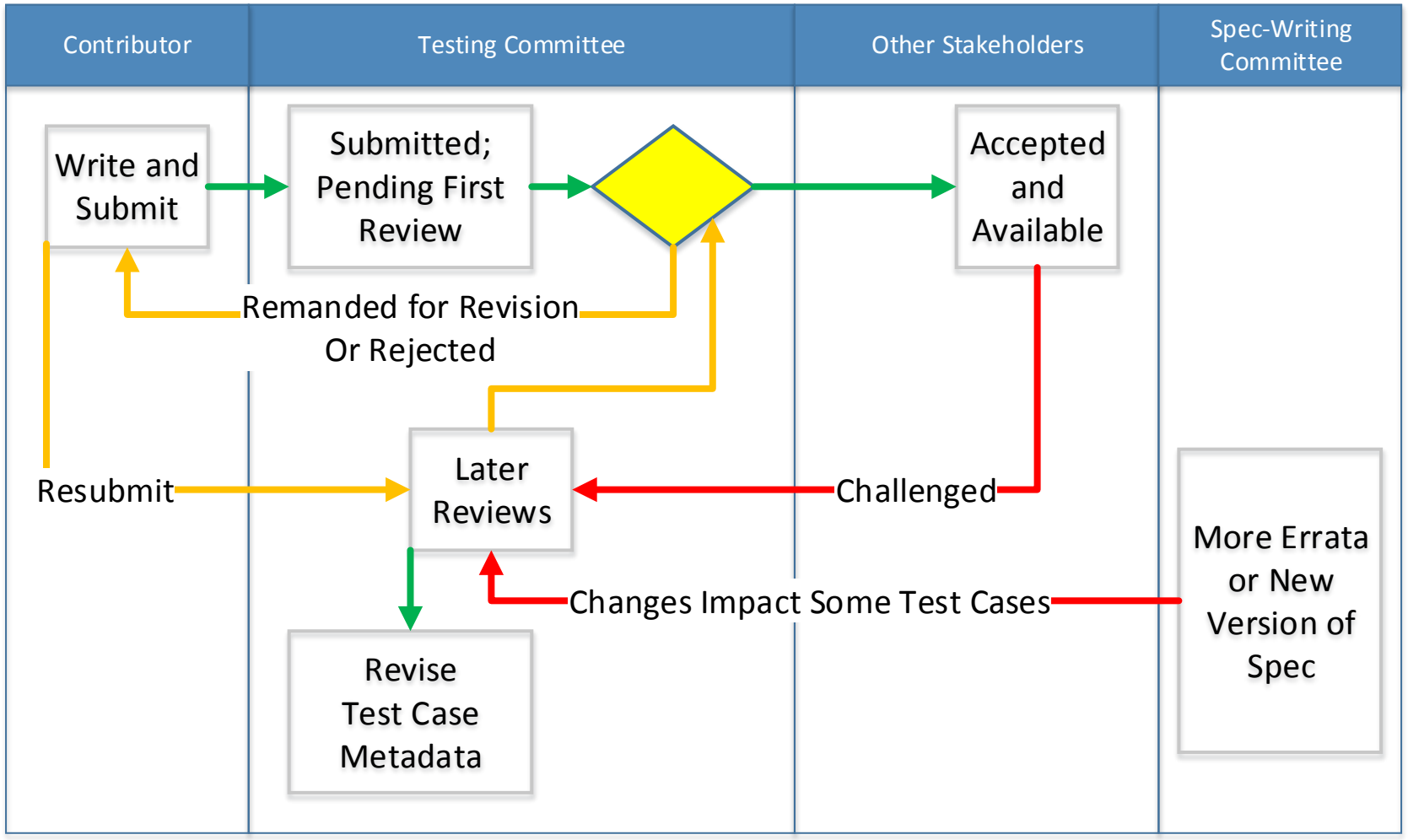
- **One extreme:** every test contains everything it needs to run in isolation
- **Other extreme:** test case Y literally depends on test case X to run, and to leave something behind
- **Middle:** metadata or test assertions tell that if X fails, there is no point in even attempting to run Y

Specification Cycle



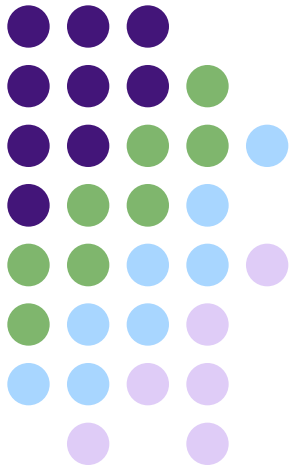
- Raw
- Internal draft (exposure determined by Consortium policy)
- Published drafts
- Formal RFC stage(s)
- Passed
- Errata – raw
- Errata draft
- Errata - passed

Lifecycle of a Test Case

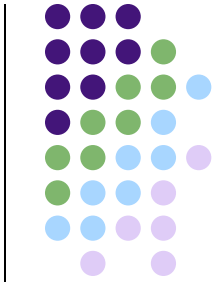


Specific Data Items

(A selected set, plus your requested topics)

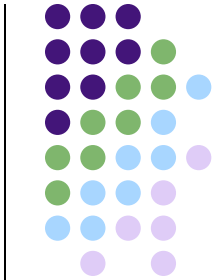


Do you rank Test Cases?



- “Priority” or “Importance” is a way to represent how early or how often you run a particular test case
- Could be indirect, by using tests in various suites (smoke test suite is early/high)
- How fine-grained is your scale?

Versions of the software

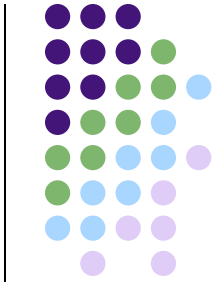


- Typically pseudo-numeric
- Want to avoid a whole new set of test materials for each version, despite new specs
- VersionAdd: lowest-numbered version to which the case applies
- VersionDrop: lowest-numbered version to which the case does not apply (if not specified, applies to the latest version)
- If no VersionAdd, applies from 1.0.0 onward
- Test cases can also have versions (or the input assets could)
- When there is concern for backward compatibility, tests usually apply over a range of versions
- Some test materials apply specifically to backward compatibility, forward compatibility, or deprecation

Dimensions of Variability - Overview

- There are 7 ways that a spec may permit variances in implementations
- Not about versioning! Every version can have DoV of its own
- Versions can be accounted for in the test case metadata by VersionAdd/VersionDrop
- Some DoV constrain others; they are not all orthogonal
- Well-written specs have explicit recognition of every DoV used

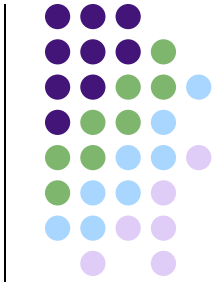
Dimensions of Variability Enumerated



- Class of Product
- Profile
- Module
- Level
- Discretionary Item
- Deprecation
- Extensibility

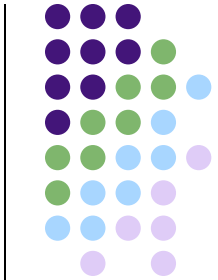
(Last slide has a link for the whole document)

Metadata Using DoV



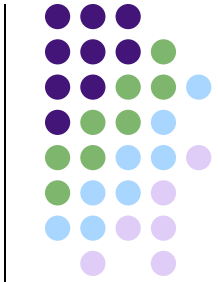
- A single test case may have metadata that says it only applies:
  - To a certain level or higher
  - When a particular module is present
  - When a particular module is absent
  - By capturing the specific choice that the implementer made on some parameter, then parameterizing the test case with that value
- The test case metadata should specify the comparitor needed for each case, and other details of the scenario, which could be influenced by profile or level (for degree of exactitude)
- XQuery metadata had to specify the initial state of the document and the values of any variables, yet implementations could accomplish the setup in different ways – the test case itself would not be portable if it tried to do the setup!

Deprecation



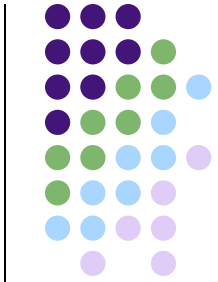
- Deprecated features work (contrast to obsolete)
- Ideally, there is a better way to do it, rather than it being a capability you will lose in the future
- Products that are producers of the material-under-test should change to the replacement technology as soon as they can
- Warnings may be appropriate
- Test Cases can be deprecated separately from the technology
  - Test Case that is current tests that deprecated feature still works (and issues warning, if specified)
  - Test Case is deprecated because there is now a better way to test a current/ongoing feature
  - Test Case uses the deprecated feature in its testing of some other feature

Test Assertions



- More-or-less atomic statements of behavior, possibly with qualifiers
- Several past attempts; OASIS defined predicate-style TAs whose truth value can be measured
- Good way to check that the spec is suitably precise, and may also help in coverage analysis
- May help when distributing the burden of writing test cases
- Facilitates test-driven development
- Through dependencies and contingencies, TAs could be cascaded or used in resolving root causes of problems
- Not just for conformance!

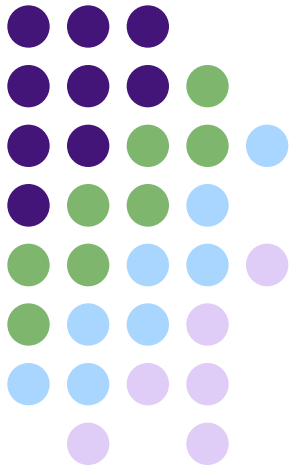
Test Suite Contents



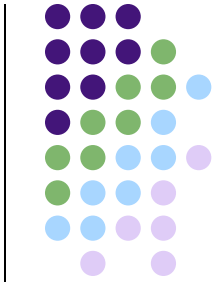
- The test case metadata defines the test cases
- Keys to the metadata: allowable values for scenario, profile/module names, comparitor names, etc.
- Definitions of testing scenarios
- Inputs for the cases
- Reference outputs (correct results)
- Specifications (at least) for running tests and comparing results – harness requirements
- Protocol tests may need a server
- An executable test suite is built by the Test Lab to account for platform differences
- The Test Lab should be able to check out later versions of the materials from the issuing body

Bonus Material

David_Marston@pegasystems.com

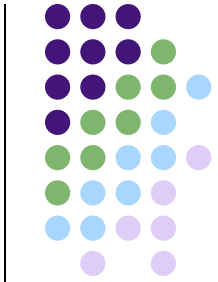


Conformance Principles



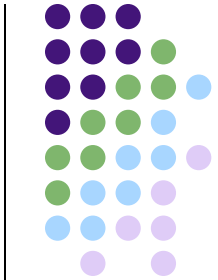
- Passing all the provided conformance tests (that are relevant) does not prove conformance
- Failing any relevant test proves non-conformance
- The "correct" result provided must be compared to actual results with a proper comparitor
- "Result" vs. "Outcome" (Pass, Fail, NA, etc.)
- Each specification should have a conformance clause
- Conformance testing can be platform-neutral and reproducible
- Conformance claims can be analyzed against a standard for well-formed conformance claims
  - Who did the testing and when
  - Versioning info on the product under test
  - Versioning info on the test materials
  - Open and verifiable test harness
- Test Lab disclaims official status unless the SDO has stated that they support certification and the lab has been authorized by the SDO to issue certifications

Compound Constraints



- A specification can have a compound sentence that, as a single sentence, spawns a "cross product" of its provisions. Here's an example from Section 3.7 of XPath 1: `"If the character following an NCName (possibly after intervening ExprWhitespace) is (, then the token must be recognized as a NodeType or a FunctionName."`
- This yields four explicit cases:
  - NCName which is a NodeType - no space - (
  - NCName which is a NodeType - white space - (
  - NCName which is a function - no space - (
  - NCName which is a function - white space - (
- and implies two negative cases:
  - NCName which is neither - no space - (
  - NCName which is neither - white space - (
- Both of the negative cases should raise an error like "unknown function.“
- But it gets worse, because NodeType actually allows 4 words.

Errata

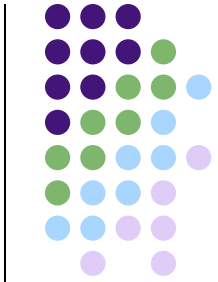


- Formal specification documents, such as standards, have errata
  - Formally published
  - Have their own lifecycle
  - Sequentially numbered, citable
  - Newer one can override older one
- Test cases can have errors, but usually we just check in an improved version
- Test case metadata should indicate recognition of relevant specification errata

Cross-version Compatibility

- Backward Compatibility: software is a higher version than the data/scripts
- Forward Compatibility: software is a lower version, attempting to do the best it can with data/scripts intended for a higher version

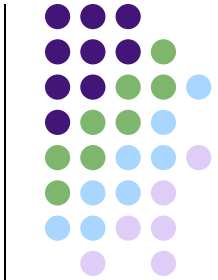
Construct a DOS/Windows BATch file from XML Metadata



```
<xsl:template match="/">
  <xsl:text>@ECHO OFF &#10;</xsl:text>
  <!-- Resulting batch file takes an argument to specify where output goes
   -->
  <xsl:text>set OUTDIR=\results\xalan\%1 &#10;</xsl:text>
  <!-- Whatever else goes at the top of the batch file -->
  <xsl:apply-templates select="test-suite/test-catalog/test-case"/>
  <xsl:text>ECHO Done! &#10;</xsl:text>
  <!-- Whatever else goes at the bottom of the batch file -->
</xsl:template>

<!-- Below this point is an xsl:template for test-case -->
<!-- There could be different templates for each testing
     scenario, like test-case[scenario/@operation="negative"] -->
```

To Go Deeper



- W3C discussion: http://www.w3.org/wiki/TestCaseMetadata
- W3C QAWG Note on Test Metadata:
  http://www.w3.org/TR/2005/NOTE-test-metadata-20050914/
- Variability in Specifications:
  http://www.w3.org/TR/2005/NOTE-spec-variability-20050831/
- XML Namespaces:
  - http://www.ibm.com/developerworks/library/x-nmspace.html
  - http://www.ibm.com/developerworks/library/x-nmspace2.html
- XQuery Test Suite (includes guides to use and contribute):
  http://dev.w3.org/2011/QT3-test-suite/
- OASIS Test Assertions TC
  https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tag