```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0">
    <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2" xmlns:sqf="http://www.schematron-quickfix.com/validator/process">
        <x:description xmlns:x="http://www.jenitennison.com/xslt/xspec" schematron="jats.sch">
            <html xmlns="http://www.w3.org/1999/xhtml">
                <grammar xmlns="http://relaxng.org/ns/structure/1.0">
                    <article article-type="case-report" xml:lang="en"  xmlns:xlink="http://www.w3.org/1999/xlink">
```

# Self-Generating Quality Control

## A Case Study

Declarative Amsterdam, October 8 - 9, 2020

# Tomos Hillman

eXpertML Ltd.

Web: http://expertml.com/

Email: tom@eXpertML.com

Tom is the founder of eXpertML, and provides technical services and consultation to publishers and other businesses interested in the applications of XML.

# Vincent Lizzi

Taylor & Francis Group

Web: https://www.tandfonline.com/

Email: vincent.lizzi@taylorandfrancis.com

Vincent is Head of Information Standards and technical lead for the JATS 1.2 upgrade project. Taylor & Francis publishes over 2,700 peer-reviewed research journals.

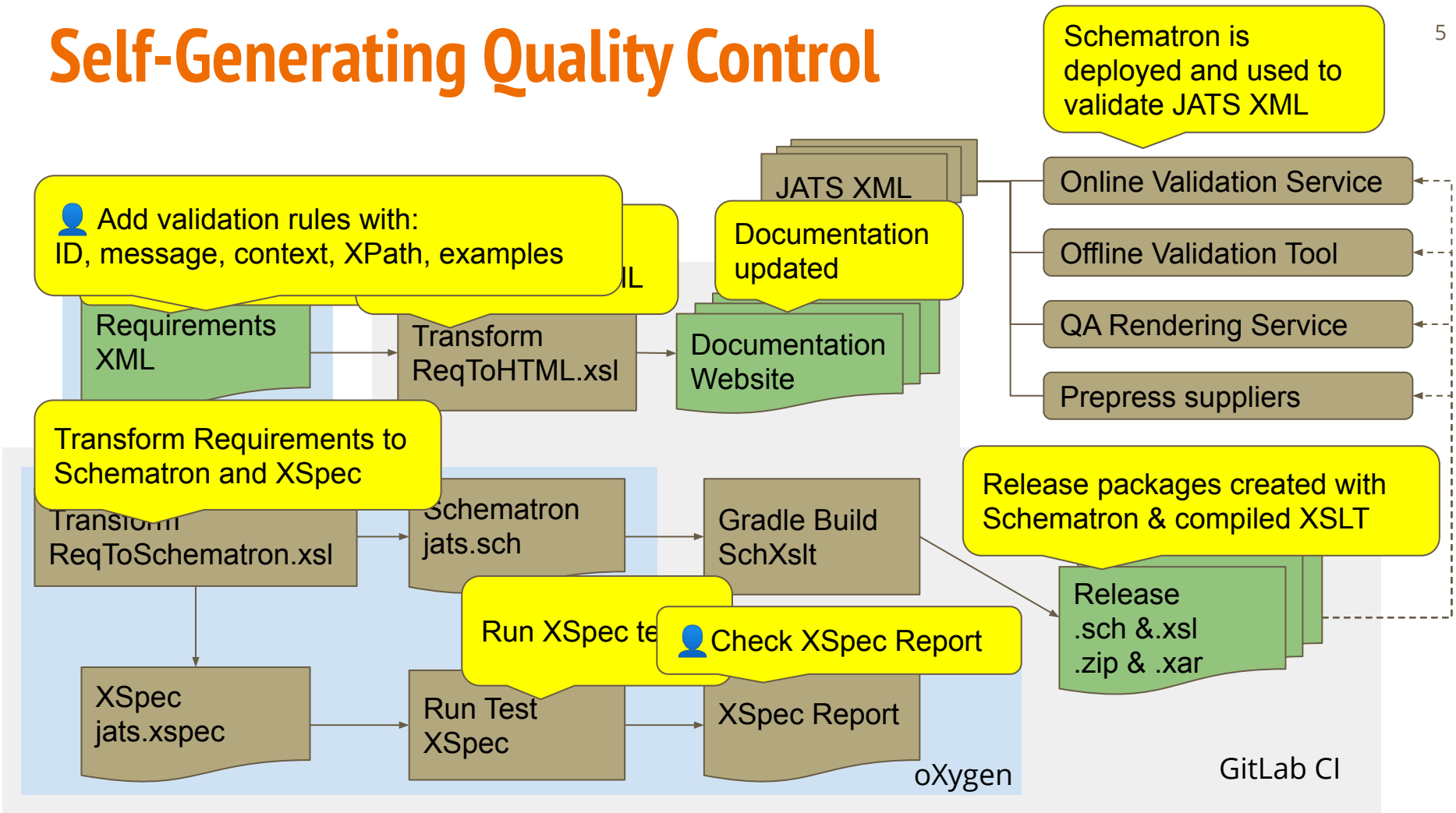
# The project

Scope:

1. Implement 200+ validation rules for JATS 1.2 in Schematron.
2. Corresponding XSpec scenarios to ensure that the Scematron functions correctly according to expectations.

Part of a larger project to upgrade Taylor & Francis journal production DTD from JATS 1.0 to JATS 1.2.

# Self-Generating Quality Control



# Gathering validation requirements

During the analysis phase a team of production staff wrote tagging guidelines documentation and validation rules. Created ~40 documents in MS Word.

MS Word template designed for the goals of creating documentation and Schematron validation rules.

- Description - documentation
- Examples - correct and incorrect
- Message - text for a validation message
- Phase - current_content, scanned_content, converted_content, rendering_alerts (or all)
- Level - error, warning, or information
- Context - element or attribute, if known
- XPath test - if known

| | |
|---|---|
| **Message** | If there is more than one <abstract> element the additional <abstract>'s should be identified with abstract-type attribute or specific-use attribute. |
| **Phase** | All |
| **Level** | Error |
| **Context** | <abstract> |
| **XPath Test** | count($abstracts) = 1 or ( every $a in ($abstracts except $abstracts[not(@abstract-type \| @specific-use)][1]) satisfies boolean($a[@abstract-type \| @specific-use]) ) |

# Requirements Word Documents

- Word Requirements thorough, to a human reader, BUT
  - Informal, described information
  - Missing fields or Invalid XML syntax (e.g. closing tags, etc)
  - Inconsistencies in content and/or formatting
  - Not an automatable data source.
- Needs semantic mark-up!

# Requirements XML

- XML Early workflow!
- Collect data into a single source
- Check for errors
- Correct errors and eliminate discrepancies
- Analyse and identify globally:
  - Rule Phases
  - Contexts
  - Severities

# Requirements XML - Example.

```
<?xml version="1.0" encoding="UTF-8"?>
<schematron>
        <topic id="JATS-0003">
                <title>Appendices</title>
                <rule id="JATS-0003-001">
                        <Message xml:space="preserve">Appendix &lt;app/&gt; must contain a label &lt;label/&gt;</Message>
                        <Phases>
                                <phase>converted content</phase>
                        </Phases>
                        <Level>Error</Level>
                        <contexts>
                                <context>app-group/app</context>
                        </contexts>
                        <xpath>label</xpath>
                        <xspec>
                                <expect-not-assert eg="e001" location="/app-group/app"/>
                                <expect-assert eg="ce001" location="/app-group/app"/>
                        </xspec>
                </rule>
                <examples>
                        <example xml:space="preserve" id="JATS-0003-e001">
<app-group/>
                        </example>
                </examples>
        </topic>
</schematron>
```

Corresponds to a word document

Error message

Rules

Applicable phases

Applicable contexts

Rule logic

Links rule to specific nodes in examples and counter-examples.

Examples (which demonstrate a correct sample) and counter-examples (which contain the relevant error)

# Requirements XML - Validation and fix-up

- Relax NG + Schematron
  - Consistency
  - Identification of errors
  - Schematron phases reflect workflow
- Schematron Quick Fixes
  - Missing data, standardising formats, etc
- JATS examples easily extracted for validation

# Generating from the requirements document

- Automation was always intended
  - but as a one-off!
  - Turns out to be easier to manage
  - Unexpected advantages
- Generation of outputs, including:
  - Schematron
  - XSpec
  - Project tasks/issues/tickets
  - Release documentation

# Generating Schematron:

- Rules from the requirements document are transformed into abstract rules in a single pattern.
- Patterns are generated corresponding to each combination of phases present
- Phases include relevant patterns
- Generated patterns include rules for each context, which extend the relevant abstract rule.

# Generating Schematron: Shadowed Contexts



- Specific rule context, e.g. `pub-history/event/date`
- General rule context, e.g. `date` may be ‘shadowed’
- Rules such as ISO date format may be missed from event dates⚠️

# Generating Schematron: Shadowed Contexts Fix



- Ensure abstract rules which apply to any simplified context also apply to the more specific context
- Parse the context statements, and generate a list of more general, simpler match statements
  - Using ReX Parser Generator
  - Plus some combination maths and recursion for location steps and predicates
- The best is the enemy of the good: not foolproof.

# Generating XSpec

- Examples and counter-examples exported to separate files
- Scenarios created for each file, with the file as the context of the tests
- Expectation tests created from data linking rules to examples in the requirements document.

# Generating documentation

https://tfjats.gitlab.io/jats1.2/jats-schematron/

- Hardest part was the CSS
- Indices by rule ID and by contexts
- Including simplified contexts!
- Documentation will never be out of date

# Beyond Schematron, within Schematron

**Motivation:**

- **Schematron portability ensures the same tests run in all environments**
- **Rules not enforced by DTD**
- **Save time by preventing problems from occurring in JATS XML files**

**Techniques:**

- **Access XML that is not yet parsed**
- **① unparsed-text(base-uri()) to read the XML file as a string**
- **② Alternatively, host program uses a parameter to provide in-memory XML as a string**
- **③ Verify the unparsed XML string begins with a left angle bracket <**
- **④ Parse the XML using an XML parser in an XSLT function**
- **RelaxNG to constrain content models**

② <sch:let name="unparsedXml" value="""/>
① <sch:let name="unparsedXmlString" value="if (string-length($unparsedXml) gt 0) then $unparsedXml else unparsed-text(base-uri())"/>
③ <sch:let name="unparsedXmlAvailable" value="matches($unparsedXmlString, '^\s*<')"/>
④ <sch:let name="parsed" value="xmlstart:parse-document($unparsedXmlString)"/>

# Parsing XML in XSLT



**Motivation:**

- **Use XPath to test parts of the XML document that are usually thrown away by the parser before Schematron is run**
- **Considered using regular expressions as an alternative to a grammar. Would need a series of regular expressions.**

① <?xml version="1.0" encoding="UTF-8"?>
② <!DOCTYPE article PUBLIC "-//NLM//DTD JATS (Z39.96) Journal Archiving and Interchange DTD with OASIS Tables with MathML3 v1.2 20190208//EN" "https://jats.nlm.nih.gov/archiving/1.2/JATS-archive-oasis-article1-mathml3.dtd">
③ <article article-type="" xml:lang="en" xmlns:mml="http://www.w3.org/1998/Math/MathML" xmlns:oasis="http://www.niso.org/standards/z39-96/ns/oasis-exchange/table" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:ali="http://www.niso.org/schemas/ali/1.0/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>

**Techniques:**

- **XML 1.0 grammar reduced to the beginning of an XML file:**
  - **① XML Declaration**
  - **② DOCTYPE declaration**
  - **comments, processing instructions**
  - **③ root element and its attributes**
  - **Ignore everything after the end of the root element opening tag.**
  - **Internal DTD subset not allowed**
- **XML parser in XSLT generated using REx Parser Generator https://bottlecaps.de/rex/**
- **The unparsed XML string is parsed to provide a tree model accessible by XPath**
- **Rule context is document node "/"**

# Testing the DOCTYPE declaration

**Motivation:**

- **Incorrect DOCTYPE declarations cause failures when parsing XML**
- **JATS DTD informs the XML parser about mixed content models where space characters should not be ignored and default attribute values**
- **Ensure correct public identifier and http URI in the DOCTYPE**

**Techniques:**

- **The parsed XML string provides a tree model of the DOCTYPE declaration that is accessible by XPath**
- **① Assert that the DOCTYPE declaration contains one of the 2 expected options for a root element with a Public Identifier and a System URI (JATS DTD or Issue DTD)**

① $parsed//doctypedecl
  Name = "'article"
  ExternalID/PubidLiteral = "-//NLM//DTD JATS (Z39.96) Journal Archiving and Interchange DTD with OASIS Tables with MathML3 v1.2 20190208//EN"
  ExternalID/SystemLiteral = "https://jats.nlm.nih.gov/archiving/1.2/JATS-archive-oasis-article1-mathml3.dtd"
Or
  Name = "issue-xml"
  ExternalID/PubidLiteral = "-//Atypon//DTD Atypon JATS Journal Archiving and Interchange Issue XML DTD v1.1 20160222//EN"
  ExternalID/SystemLiteral = "http://cats.informa.com/tfjats/1.2/dtd/atypon-jats-v1.1-issue.dtd"

# Testing presence of default attributes - xml:lang

**Motivation:**

- **JATS DTD provides default value “en” article/@xml:lang. The document’s primary language should be declared not assumed to be English. (WCAG 2.1 sec 3.1.1)**

**Techniques:**

- **The parsed XML string provides a tree model of the root elements’ attributes that does not have defaults from the DTD**
- **① Assert that xml:lang attribute is present**
- **Another rule checks the value of all xml:lang attributes using a list of ISO language codes**

① $parsed/element/Attribute[Name = 'xml:lang']

# Testing presence of default attributes - namespaces



**Motivation:**

- **JATS DTD provides #FIXED attribute defaults for namespace declarations. XML parsing fails if namespace declarations are neither contained in the XML document nor filled in from the DTD by the XML parser.**

**Techniques:**

- **The parsed XML string provides a tree model of the root elements' attributes that does not have defaults from the DTD**
- **① Assert namespace definition for xmlns:xlink is present (xlink is always used in JATS XML)**
- **② Assert namespace definition is present for mml, oasis, ali, and xsi if the namespace is used in the document**

① $parsed/element/Attribute[Name = 'xmlns:xlink' and AttValue = '"http://www.w3.org/1999/xlink"']

② (not(.//mml:*) or $parsed/element/Attribute[Name = 'xmlns:mml' and AttValue = '"http://www.w3.org/1998/Math/MathML"'])
(not(.//oasis:*) or $parsed/element/Attribute[Name = 'xmlns:oasis' and AttValue='"http://www.niso.org/standards/z39-96/ns/oasis-exchange/table"'])
(not(.//ali:*) or $parsed/element/Attribute[Name = 'xmlns:ali' and AttValue = '"http://www.niso.org/schemas/ali/1.0/"'])
(not(.//@xsi:*) or $parsed/element/Attribute[Name = 'xmlns:xsi' and AttValue = '"http://www.w3.org/2001/XMLSchema-instance"']))"

# Testing character encoding

**Motivation:**

- **Character encoding problems can cause cause XML document to fail processing or to publish with characters that appear “broken”.**
- **Require characters to be encoded as entities, e.g. combining characters, characters with less font support**
- **Disallow problematic characters e.g. private use areas, control characters, Windows-1252 smart quotes.**

**Techniques:**

- **The parsed XML string provides a tree model of the XML declaration that is accessible by XPath**
- **① Assert that XML declaration specifies encoding UTF-8, US-ASCII, or ISO646-US**
- **② Assert characters not in the allowed list are encoded as entities using a regular expression on the unparsed XML string**
- **③ Assert characters on denied list are not present in any element using a regular expression, includes character entities after expansion**

① $parsed/prolog/XMLDecl/EncodingDecl/EncName = ('UTF-8', 'US-ASCII', 'ISO646-US')

② replace($unparsedXml,
concat('([ 	
*<>?=\-\./:#;+"&,/\\\[\]^_`\|{}\(\)@~%!$0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz', codepoints-to-string(39), ']+)+'), "")

③ context=* test string-join(for $t in (@*, node() except *) return analyze-string($t, "'([&#x7F;-Ÿ]|[-]| ... )'")//*:group)

# Adding grammatical constraints

## Motivation:

- JATS Archiving DTD is permissive, content models give many options
- Restrict content models to create consistency, e.g. author metadata
- Not customizing the DTD to reduce maintenance, must use Schematron

## Techniques:

- ①RelaxNG XML syntax to express content model
- ②RelaxNG transformed to Regular Expression
- ③RelaxNG transformed to DTD-like syntax for the validation message
- ④Create string representation of element content found in the XML document
- ⑤Regular Expression match tests the element content with the content model

```
<rule id="JATS-0045-002">
   <Message>contrib-group</Message>
   <Phases><phase>converted content</phase><phase>current
content</phase><phase>scanned content</phase></Phases>
   <Level>Error</Level>
   <contexts><context>contrib-group</context></contexts>
①<model>
      <oneOrMore><element name="contrib"/></oneOrMore>
      <zeroOrMore><choice><element name="aff"/><element
name="aff-alternatives"/><element name="bio"/><element
name="etal"/></choice></zeroOrMore>
   </model>
   <xspec><expect-not-assert eg="e001"/><expect-assert eg="ce002"/></xspec>
</rule>
```

② $regex =  ^(?:contrib,?)+,?(?:(?:aff|aff-alternatives|bio|etal),?)*$

③ $text = (contrib)+, ((aff | aff-alternatives | bio | etal))*

④ $sequence = string-join(for $n in node() return if ($n instance of element()) then local-name($n) else if ($n instance of text() and normalize-space($n)) then '#PCDATA' else (), ',')
e.g. contrib,contrib,aff

⑤ matches($sequence, $regex)

# Continuous Integration

All XSpec tests must pass, then documentation is updated and release packages are built automatically.

- Generate Schematron
- Generate XSpec and run XSpec
- Generate documentation web site
- Build release packages
  - Includes the generated Schematron and compiled XSLT for each phase
  - Zip format
  - XAR format including an XQuery wrapper
- Protected “master” branch requires a merge request for each change
- Version tag for each release

GitLab configuration was relatively easy.

# Conclusion

By first declaring WHAT we know in a clear format, rather than how it should be used, we discovered a range of benefits:

- Documentation and tests represent the validation rules exactly
- Time saved in developing Schematron rules
- Future maintenance of the Schematron will use this infrastructure

# Thank you Declarative Amsterdam!

```
                </article>
            </grammar>
        </html>
    </x:description>
</sch:schema>
</xsl:stylesheet>
```

# Notes

Time slot: 30 minutes
Allow 5-10 for questions
Presentation length: 20 minutes

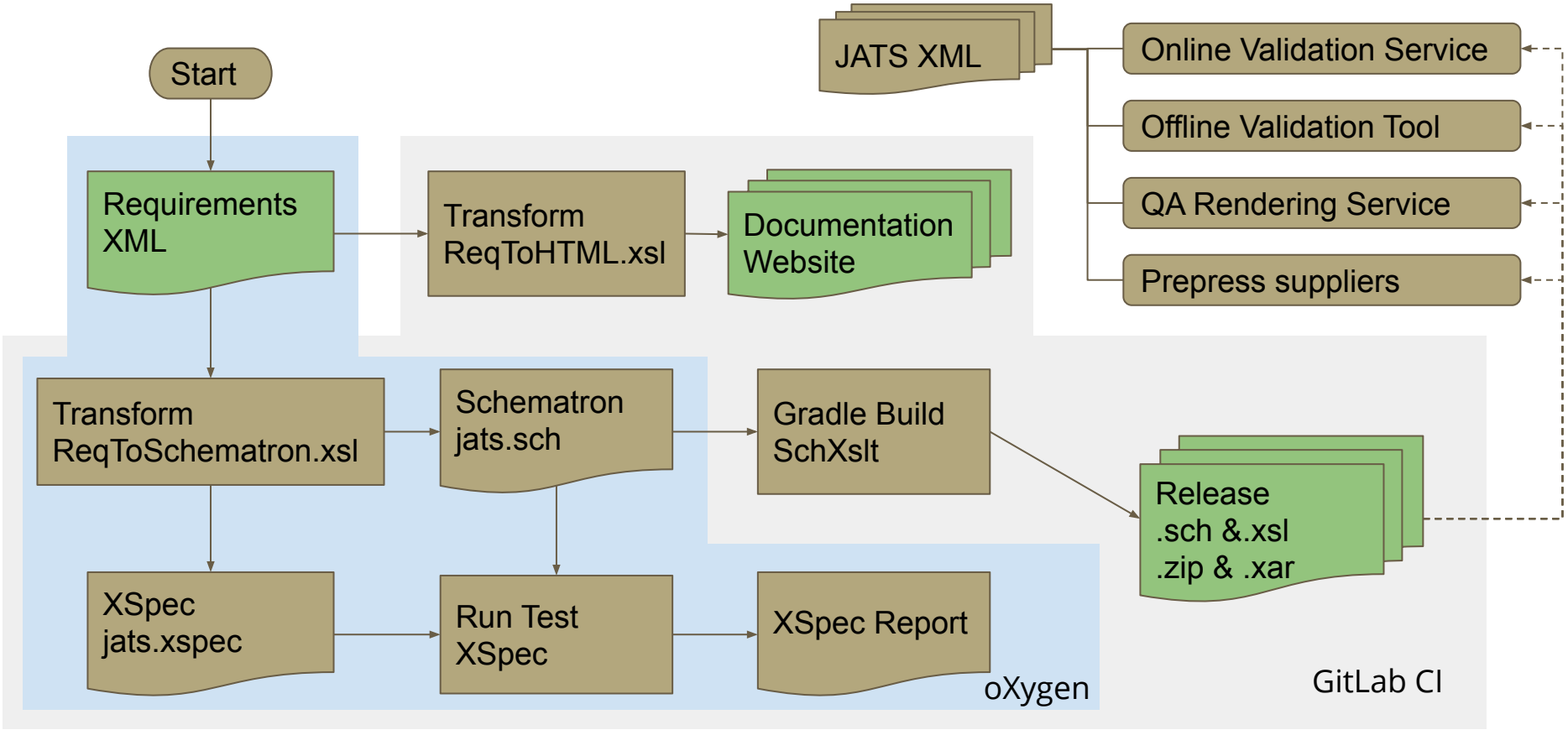
Narrative:

1. Who we are briefly (0:30, TV)
2. The project - context and what we set out to do (0:30, V)
3. Flowchart - start with a picture (1:00 V)
4. Requirements format - word doc template (0:30, V)
5. XML requirements format - immediate needs (1:00, T)
6. Generating Schematron (5:00, T)
    a. XPath contexts and shadowing
7. Generating XSpec (5:00, T)
8. Generating Documentation (1:00, T)
9. QA Beyond Schematron
    a. Parsing XML within Schematron (2:00, V)
    b. Testing the DOCTYPE declaration (1:00, V)
    c. Testing physical presence of defaulted attributes (1:00, V)
    d. Testing character encoding (1:00, V)
    e. Adding grammatical constraints (2:00, V)
10. Continuous Integration (1:00, T)
11. Conclusion (0:30, TV)

Timings:
Total Est.: 23:30 minutes

Generally 125 - 150 spoken wpm
Total Est.: 2,500 words

# Self-Generating Quality Control



# Requirements XML - Example.

```
<?xml version="1.0" encoding="UTF-8"?>
<schematron>
        <topic id="JATS-0003">
                <title>Appendices</title>
                <rule id="JATS-0003-001">
                        <Message xml:space="preserve">Appendix &lt;app/&gt; must contain a label &lt;label/&gt;</Message>
                        <Phases>
                                <phase>converted content</phase>
                        </Phases>
                        <Level>Error</Level>
                        <contexts>
                                <context>app-group/app</context>
                        </contexts>
                        <xpath>label</xpath>
                        <xspec>
                                <expect-not-assert eg="e001" location="/app-group/app"/>
                                <expect-assert eg="ce001" location="/app-group/app"/>
                        </xspec>
                </rule>
                <examples>
                        <example xml:space="preserve" id="JATS-0003-e001">
<app-group/>
                        </example>
                </examples>
        </topic>
</schematron>
```

Corresponds to a word document

Error message

Rules

Applicable phases

Applicable contexts

Rule logic

Links rule to specific nodes in examples and counter-examples.

Examples (which demonstrate a correct sample) and counter-examples (which contain the relevant error)