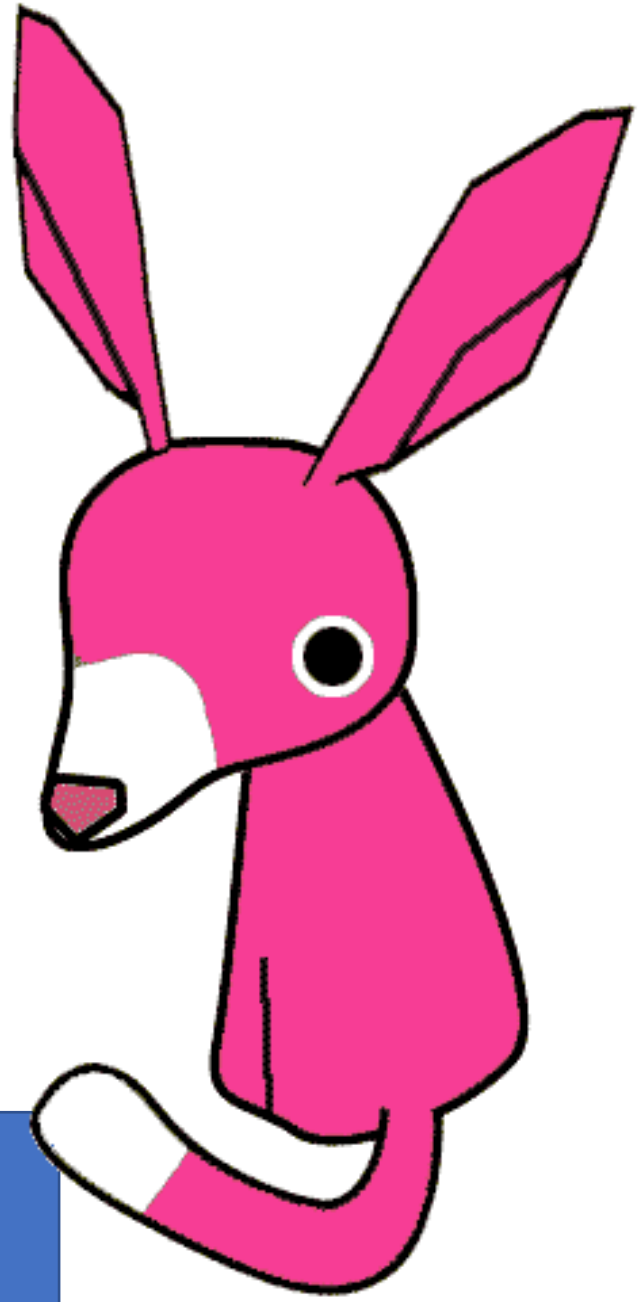
Schematron Tutorial

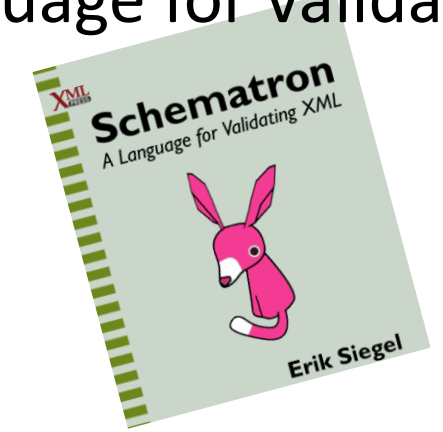
Declarative Amsterdam
November 7 and 8, 2022



While waiting, maybe you can do some preparations?
Go to **https://da2022.xatapult.com** for instructions!

Who Am I?

- Erik Siegel
- Content Engineer and XML specialist
- One-man company: Xatapult
  - Groningen, The Netherlands
- Author of "Schematron – A language for validating XML"

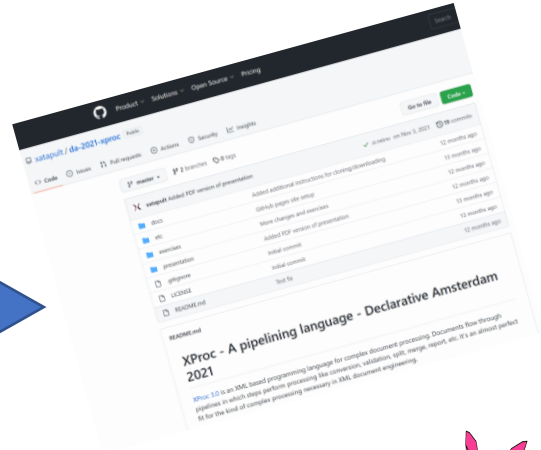


- Contact:

erik@xatapult.nl
www.xatapult.com
www.linkedin.com/in/esiegel/
+31 6 53260792

Tutorial format

- Bit of theory
- Exercise!
  - Basic instructions: http://da2022.xatapul t.com/
  - Presentation, code and instructions: https://github.com/xatapult/da-2022-schematron
  - Easiest to follow using oXygen, all prepared
  - All exercises contain instructions in **`instructions.pdf`**
  - Solution and explanations in **`solution/`** subfolder
- Repeat

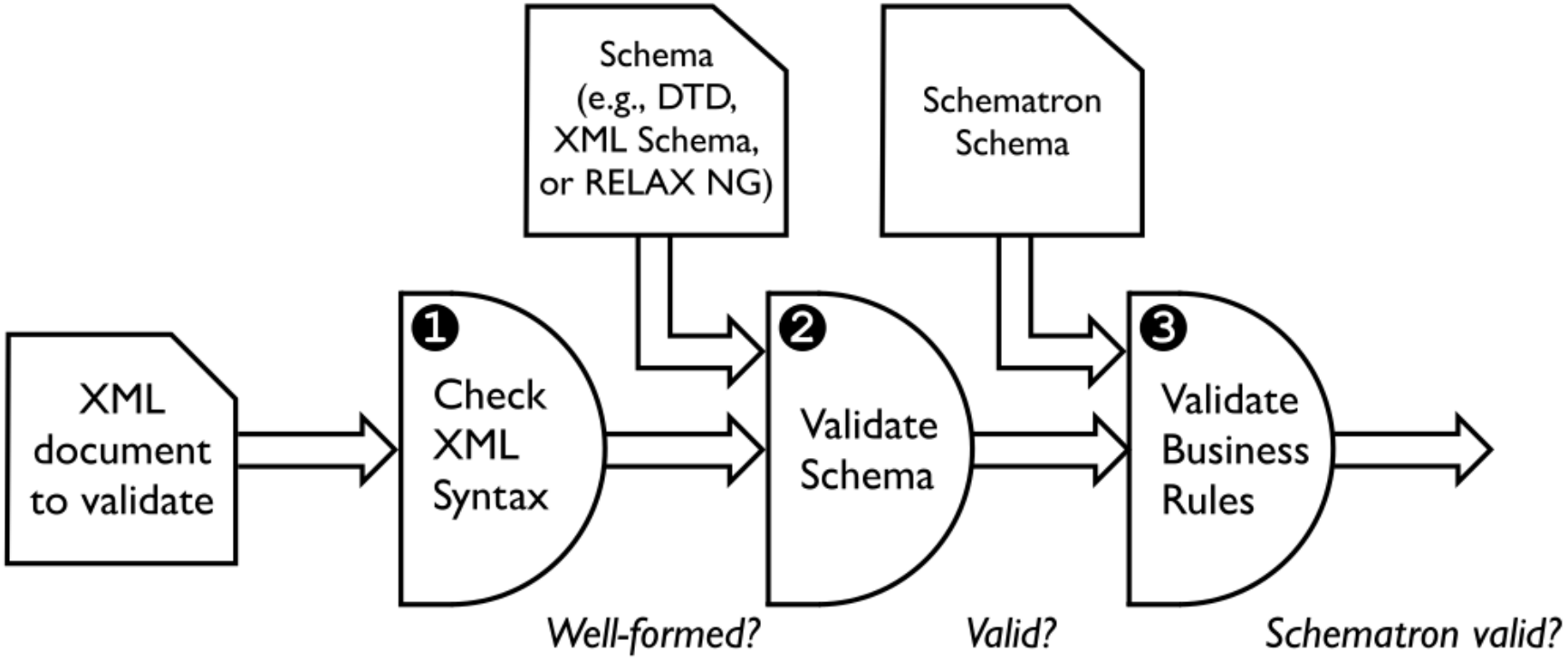


All instructions and explanations are collected in **`exercises/syllabus.pdf`**

Schematron highlights

- A formal XML based schema language
- Simple but powerful
- Can go way beyond the "classic" validation langua
- Two types of rules:
  - Assertions
  - Reports
- Messages in your own words!
- XPath based expressions
- Incorporates XSLT keys and functions
- Has a predefined XML based output format (SVRL)

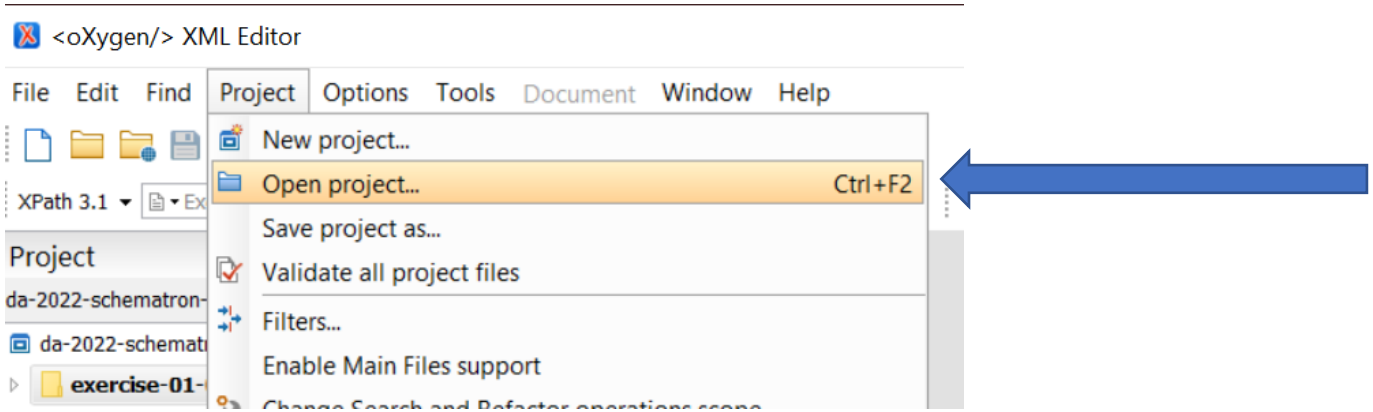
Schematron in context



👉 Hands-on: Installation and pre-flight check #1

- Done all preparations?
  - Download/cloned the GitHub repository for this tutorial? https://github.com/xatapult/da-2022-schematron
  - A Schematron processor ready? Preferably oXygen…

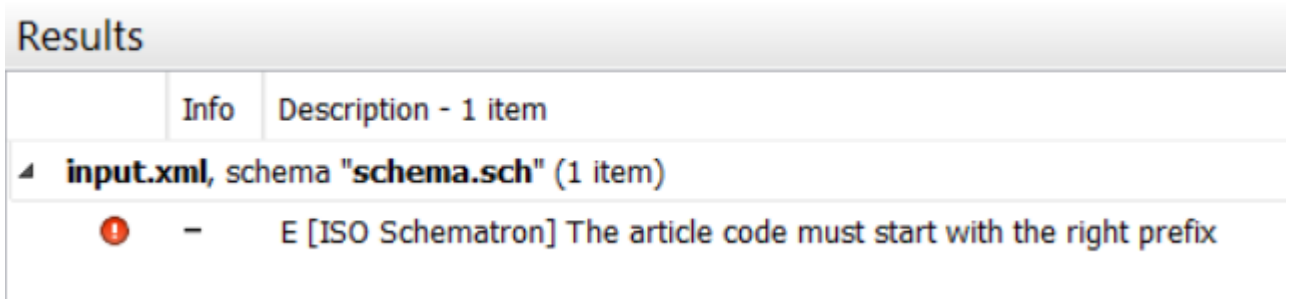
- When using oXygen:
  - Open the oXygen *project*: …/**`exercises/da-2022-schematron-exercises.xpr`**



👉 Hands-on: Installation and pre-flight check #2

- When using oXygen:
  - Open the file …**`/exercises/exercise-01-01/input.xml`**:

  - Validate it by pressing the button on the toolbar
    - Or by using the menu option Document > Validate > Validate
    - Or by pressing ctrl-shift-V
  - You should see:



  - Have a look at **`schema.sch`**, the Schematron schema used

Schematron fundamentals

An empty Schematron schema

```
<schema xmlns="http://purl.oclc.org/dsdl/Schematron" queryBinding="xslt3">

  <pattern>

  </pattern>

</schema>
```

Patterns and rules

- A Schematron schema consists of one or more *patterns*: **`<pattern>`**
  - Every pattern is applied to every node in the document being validated
- A *pattern* consists of zero or more *rules*: **`<rule context="…">`**
  - A rule has a **`context`** attribute, containing an XSLT match pattern
  - Only the first rule that matches in a pattern fires

```
<schema xmlns=http://purl.oclc.org/dsdl/Schematron queryBinding="xslt3">

  <pattern>
    <rule context="thing[@material eq 'wood']"> … </rule>
    <rule context="thing"> … </rule>
  </pattern>

</schema>
```


Assertions and reports

- When a rule fires, the node fired upon becomes the context item
- A rule consists of zero or more
  - asserts: **`<assert test="…">`** - Activated when the test expression is false
  - reports **`<report test="…">`** - Activated when the test expression is true
- The contents of the element is issued as validation message

```
      <thing code="T12345" price="75.25"/>
```

```
<rule context="thing">
  <assert test="starts-with(@code, 'T')">Codes must start with a T</assert>
  <report test="xs:double(@price) gt 100">High priced thing found!</report>
</rule>
```

👉 Hands-on: Assertions and reports

- All files in: …/**`exercises`**/**`exercise-02-01`**

- Read the instructions in: **`instructions.pdf`**
- Open the files:
  - Document to validate: **`input.xml`**
  - Template Schematron schema: **`template.sch`**
- Fill in **`template.sch`** according to the instructions
- To try it out in oXygen, **`input.xml`** is automatically validated by **`template.sch`**

Possible solution in: **`solution/solution.sch`** (short explanation in **`solution/explanation.pdf`**).

Rule processing revisited

- Assume we have an XML document with **`<book>`** and **`<magazine>`** elements, underneath some root element
- We have specific rules for books and magazines, but also rules that apply to *all* elements
- Will this work?

```
<pattern>
  <rule context="/*/book">
    … (asserts and reports for book elements)
  </rule>
  <rule context="/*/magazine">
    … (asserts and reports for magazine elements)
  </rule>
  <rule context="/*/*">
    … (asserts and reports for all elements)
  </rule>
</pattern>
```

👉 Hands-on: rule processing

- All files in: …/**`exercises`**/**`exercise-02-02`**

- Read the instructions in: **`instructions.pdf`**
- Open the files:
  - Document to validate: **`input.xml`**
  - Template Schematron schema: **`template.sch`**
- Improve in **`template.sch`** according to the instructions
- To try it out in oXygen, **`input.xml`** is automatically validated by **`template.sch`**

Possible solution in: **`solution/solution.sch`** (short explanation in **`solution/explanation.pdf`**).

What's the result of a Schematron validation?

- Schematron has an XML reporting language called SVRL: *Schematron Validation Reporting Language*
- In an IDE like oXygen, you don't see it
- Using the command line, you will
- Useful in toolchains and automated processing
  - For instance, create a custom report

SVRL example

```
<schematron-output xmlns="http://purl.oclc.org/dsdl/svrl">
  <active-pattern documents="…/parcels-schematron-invalid.xml"/>
  <fired-rule context="/*"/>
  <failed-assert location="/Q{}parcels[1]" test="…">
    <text>The total weight is too high</text>
  </failed-assert>
  <active-pattern documents="…/parcels-schematron-invalid.xml"/>
  <fired-rule context="parcel"/>
  <successful-report location="/Q{}parcels[1]/Q{}parcel[1]" test="…">
    <text>The parcel's date must be more than 10 days before the delivery date</text>
  </successful-report>
  <fired-rule context="parcel"/>
</schematron-output>
```

More meaningful messages: `<value-of>`

The `<value-of select="…"/>` element allows you to add values from the validated document to your messages

```
<rule context="/*/*">
  <assert test="string-length(@code) eq 4">A code must be 4 characters long</assert>
</rule>
```

```
<rule context="/*/*">
  <assert test="string-length(@code) eq 4">
    The code <value-of select="@code"/> is invalid. It must be 4 characters long.
  </assert>
</rule>
```

👉 Hands-on: better messages

- All files in: …/**`exercises`**/**`exercise-02-03`**

- Read the instructions in: **`instructions.pdf`**
- Open the files:
  - Document to validate: **`input.xml`**
  - Template Schematron schema: **`template.sch`**
- Enhance **`template.sch`** according to the instructions
- To try it out in oXygen, **`input.xml`** is automatically validated by **`template.sch`**

Possible solution in: **`solution/solution.sch`** (short explanation in **`solution/explanation.pdf`**).

Solution for the additional changes in: **`solution/solution-extended.sch`**

Variables: `<let>`

- Declare a variable using the `<let>` element:

```
<let name="code-value" value="@code"/>
```

- Allowed as children of:
  - `<schema>` (context item document node)
  - `<pattern>` (context item document node)
  - `<rule>` (context item node matched upon)

- Reference it (like in XSLT) with a `$` prefix:

```
<assert test="$code-value eq 'IMPORTANT'">
<value-of select="$code-value"/>
```

👉 Hands-on: variable usage

- All files in: …/**`exercises`**/**`exercise-02-04`**

- Read the instructions in: **`instructions.pdf`**
- Open the files:
  - Document to validate: **`input.xml`**
  - Template Schematron schema: **`template.sch`**
- Enhance **`template.sch`** according to the instructions
- To try it out in oXygen, **`input.xml`** is automatically validated by **`template.sch`**

Possible solution in: **`solution/solution.sch`** (short explanation in **`solution/explanation.pdf`**).

Declaring namespaces: `<ns>`

- Namespaces for expressions *must* be declared with `<ns>` elements:

```
<ns uri="http://www.w3.org/1999/xhtml" prefix="xh"/>
```

- The "normal" XML way of declaring namespaces will not work for expressions:

```
<schema … xmlns:xh="http://www.w3.org/1999/xhtml">
```

👉 Hands-on: Declaring and using a namespace

- All files in: …/**`exercises`**/**`exercise-02-05`**

- Read the instructions in: **`instructions.pdf`**
- Open the files:
  - Document to validate: **`input.xml`**
  - Template Schematron schema: **`template.sch`**
- Write **`template.sch`** according to the instructions
- To try it out in oXygen, **`input.xml`** is automatically validated by **`template.sch`**

Possible solution in: **`solution/solution.sch`** (short explanation in **`solution/explanation.pdf`**).

Basic Schematron: Wrap-up

- Root element **`<schema>`** with namespace **`http://purl.oclc.org/dsdl/schematron`**
- A Schematron schema consists of:
  - **`<pattern>`** (checked for all nodes in the document)
    - **`<rule>`** (only the first on that matches fires)
      - **`<assert test="…">`**
      - **`<report test="…">`**
- Enhance the messages using **`<value-of select="…">`** elements
- Create variables using **`<let name="…" value="…">`** elements
- Define a namespace using **`<ns uri="…" prefix="…">`** elements

Abstract patterns

Phases

Diagnostics

Abstract pattern fundamentals

- Abstract patterns are macros
- Parameters alter their instantiations

```
<pattern abstract="true" id="id-of-abstract-pattern">
  …
</pattern>


<pattern is-a="id-of-abstract-pattern">
  <param name="some-parameter" value="some text"/>
</pattern>
```

Abstract pattern input example

```
<tables>

  <!-- HTML table: -->
  <table>
    <tr>
      <td>Yes!</td>
      <td>No!</td>
    </tr>
  </table>

  <!-- Calendar table-like structure: -->
  <year>
    <week>
      <day>Monday</day>
      <day>Tuesday</day>
      <!-- Etc. -->
    </week>
  </year>

</tables>
```

Abstract pattern example

```
<pattern abstract="true" id="table-pattern">
   <rule context="$table">
     <assert test="$row">
       The element <value-of select="local-name()"/> is a table structure.
       Tables must contain the correct row elements.
     </assert>
   </rule>
   <rule context="$table/$row">
     <assert test="$entry">
       The element <value-of select="local-name()"/> is a table row.
       Rows must contain the correct cell elements.
     </assert>
   </rule>
 </pattern>
```

- Abstract pattern parameters share the $... syntax with variables
- Will be replaced with *text*
- Abstract pattern parameters are not declared in any way

Abstract pattern instantiation example

```
<!-- Pattern for HTML tables: -->
<pattern is-a="table-pattern" >
  <param name="table" value="table"/>
  <param name="row" value="tr"/>
  <param name="entry" value="td"/>
</pattern>

<!-- Pattern for a calendar table-like structure: -->
<pattern is-a="table-pattern">
  <param name="table" value="year"/>
  <param name="row" value="week"/>
  <param name="entry" value="day"/>
</pattern>
```

The **`value`** attribute is *not* an XPath expression, just text!

👉 Hands-on: Using an abstract pattern

- All files in: …**`/exercises/exercise-03-01`**

- Read the instructions in: **`instructions.pdf`**
- Open the files:
  - Document to validate: **`input.xml`**
  - Template Schematron schema: **`template.sch`**
- Enhance **`template.sch`** according to the instructions
- To try it out in oXygen, **`input.xml`** is automatically validated by **`template.sch`**

Possible solution in: **`solution/solution.sch`** (short explanation in **`solution/explanation.pdf`**).

Selecting active patterns: phases

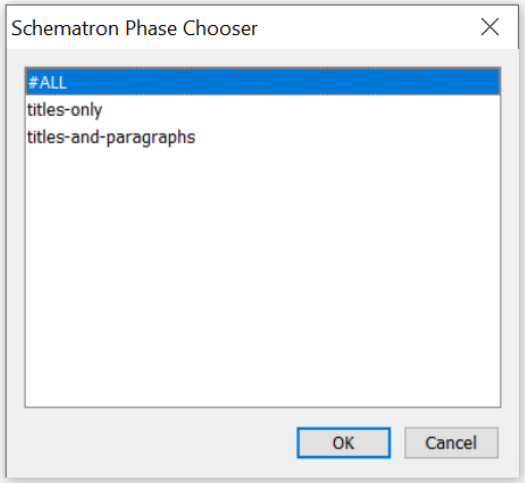
```
<phase id="normal-only">
  <active pattern="normal-rules"/>
</phase>

<phase id="normal-and-special">
  <active pattern="normal-rules"/>
  <active pattern="special-rules"/>
</phase>

<pattern id="normal-rules">…</pattern>
<pattern id="special-rules">…</pattern>
<pattern id="very-special-rules">…</pattern>
```

How to select a phase?

- Add a default phase with the **`defaultPhase="…"`** attribute (on the root element)
- oXygen will automatically recognize that there are phases and show a dialog to choose one:



- On the command line: **`-p`** ***`phase`*** option

👉 Hands-on: Using phases

- All files in: …/**`exercises`**/**`exercise-03-02`**

- Read the instructions in: **`instructions.pdf`**
- Open the files:
  - Document to validate: **`input.xml`**
  - Template Schematron schema: **`template.sch`**
- Enhance **`template.sch`** according to the instructions
- To try it out in oXygen, **`input.xml`** is automatically validated by **`template.sch`**

Possible solution in: **`solution/solution.sch`** (short explanation in **`solution/explanation.pdf`**).

Reusing messages: diagnostics

```
<pattern>
  <rule context="…">
    <assert test="…" diagnostics="message-1"/>
    <assert test="…" diagnostics="message-1"/>
  </rule>
</pattern>

<diagnostics>
  <diagnostic id="message-1"> (the message…) </diagnostic>
</diagnostics>
```

Using this mechanism, you can also issue multiple messages!

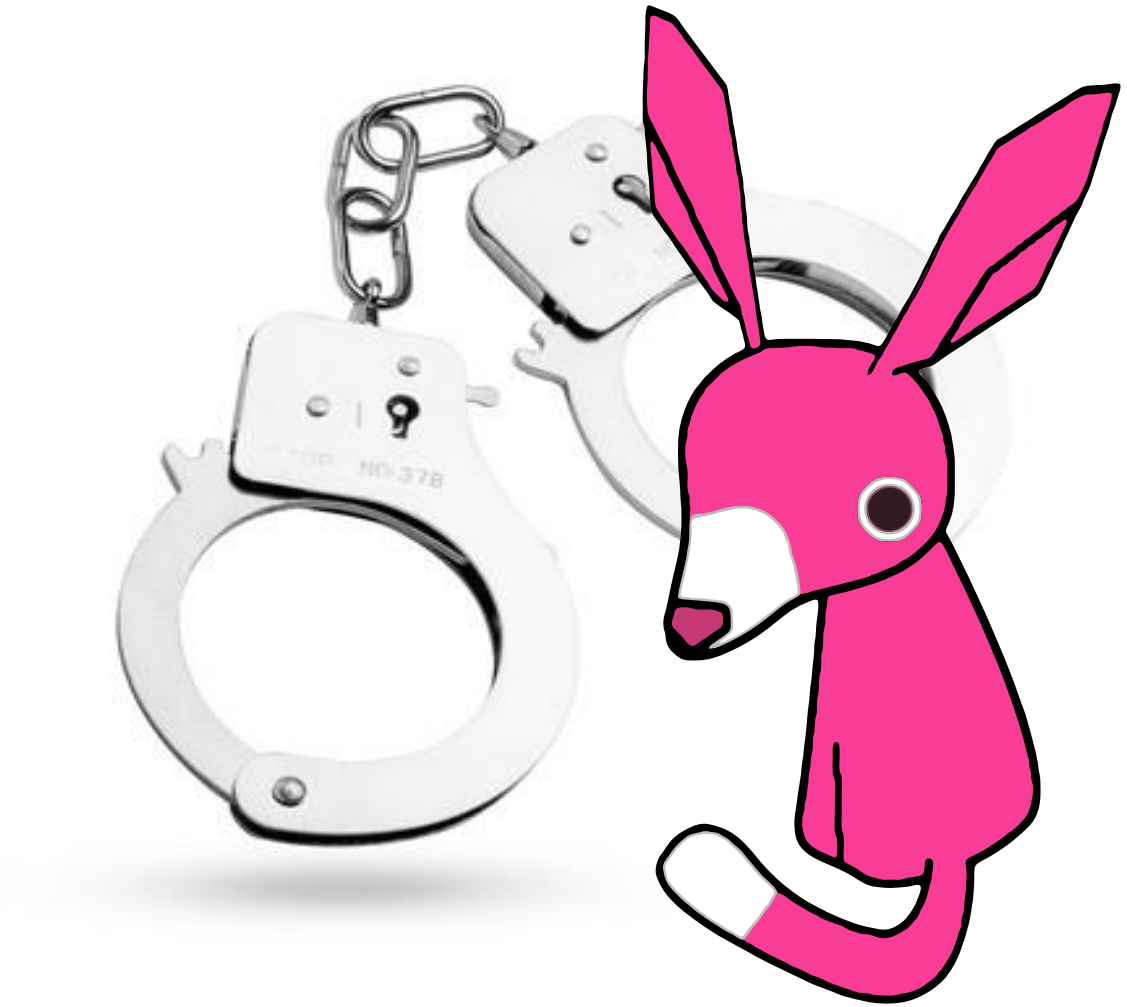
👉 Hands-on: Reusing messages

- All files in: …/**`exercises`**/**`exercise-03-03`**

- Read the instructions in: **`instructions.pdf`**
- Open the files:
  - Document to validate: **`input.xml`**
  - Template Schematron schema: **`template.sch`**
- Enhance **`template.sch`** according to the instructions
- To try it out in oXygen, **`input.xml`** is automatically validated by **`template.sch`**

Possible solution in: **`solution/solution.sch`** (short explanation in **`solution/explanation.pdf`**).

Query Language Binding and using XSLT



Schematron is a container language with a *configurable* query language for its expressions:

```
<schema xmlns=http://purl.oclc.org/dsdl/Schematron queryBinding="xslt3">

  <let name="department-code" value="/inventory-list/@depcode"/>
  <pattern>
    <rule context="article">
      <assert test="starts-with(@code, $department-code)">
        The article code (<value-of select="@code"/>) must start with the right
        prefix (<value-of select="$department-code"/>) for <value-of select="name"/>
      </assert>
    </rule>
  </pattern>

</schema>
```

Query Language Binding or QLB

The **queryBinding** attribute

Specify the Query Language Binding using the **queryBinding** attribute on the root element (default value: **xslt**):

```
<schema xmlns=http://purl.oclc.org/dsdl/Schematron" queryBinding="…">
  …
</schema>
```

Reserved and **defined** Query Language Binding names:

- **exslt**
- **stx**
- **xslt, xslt2, xslt3**
- xpath, **xpath2, xpath3,** xpath31
- xquery, xquery3, xquery31

Supported Query Language Bindings (by oXygen and SchXslt):

- **xslt, xslt2, xslt3**

Query Language Binding in practice…

- Only for XSLT (and maybe XPath)
- Advice:
  - Use **`xslt2`** or **`xslt3`**
  - Always specify the QLB (using the **`queryBinding`** attribute)
    - Otherwise, you get the default value: **`xslt`**
      - which means XPath 1.0
        - which is rather limiting…

The xslt2/xslt3 Query Language Binding

- Use XPath 2.0 or 3.1 expressions
- Use **`xsl:key`** to define keys and the **`key()`** function for lookups
- Add your own XSLT functions with **`xsl:function`** and use them in XPath expressions

- Officially no other XSLT constructs, like:
  - **`xsl:include/xsl:import`**
  - **`xsl:template`**
  - Global XSLT variables
- But this usually works fine…

```
<orders>
  <item id="bolts" price="5.49">A box with 20 bolts</item>
  <item id="nuts" price="3.78">A box with 20 nuts</item>
  <!-- … many, many more items… -->
  <order>
    <ordered-item id-ref="bolts" quantity="5"/>
    <ordered-item id-ref="nuts" quantity="10"/>
  </order>
  <!-- … many, many more orders… -->
</orders>
```

Example of using xsl:key

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" queryBinding="xslt3">

  <xsl:key name="item-ids" match="/*/item" use="@id"/>

  <pattern>
    <rule context="ordered-item">
      <assert test="exists(key('item-ids', @id-ref))">
        The referenced item <value-of select="@id-ref"/> does not exist
      </assert>
    </rule>
  </pattern>
</schema>
```

Example of defining a function - 1

```
<things>
  <thing name="thing 1" type="A125" price="17.25"/>
  <thing name="thing 2" type="A125" price="17.26"/>
  <thing name="thing 3" type="X96" price="89.34"/>
  <thing name="thing 4" type="Y78" price="10.01"/>
</things>
```

```
<type-codes-and-prices default-price="10.0">
  <data type="A125" price="17.25"/>
  <data type="X96" price="89.34"/>
</type-codes-and-prices>
```

Example of defining a function - 2

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" queryBinding="xslt3">

 <ns uri="#functions" prefix="f"/>

 <xsl:function name="f:get-price" as="xs:double">
   <xsl:param name="type" as="xs:string"/>
   <xsl:variable name="prices-document" as="document-node()"
     select="doc('type-codes-and-prices.xml')"/>
   <xsl:variable name="data-element-for-type" as="element(data)?"
     select="$prices-document//data[@type eq $type]"/>
   <xsl:choose>
     <xsl:when test="exists($data-element-for-type)">
       <xsl:sequence select="xs:double($data-element-for-type/@price)"/>
     </xsl:when>
     <xsl:otherwise>
       <xsl:sequence
         select="xs:double($prices-document/type-codes-and-prices/@default-price)"/>
     </xsl:otherwise>
   </xsl:choose>
 </xsl:function>
```

Example of using a function

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" queryBinding="xslt3">

  <ns uri="#functions" prefix="f"/>

  <xsl:function name="f:get-price" as="xs:double">
    …
  </xsl:function>

  <pattern>
    <rule context="thing">
      <let name="expected-price" value="f:get-price(@type)"/>
      <assert test="$expected-price eq xs:double(@price)">
        The price for <value-of select="@name"/> should be
        <value-of select="$expected-price"/>
      </assert>
    </rule>
  </pattern>

</schema>
```

👉 Hands-on: Using XSLT functions

- All files in: …/**`exercises`**/**`exercise-04-01`**

- Read the instructions in: **`instructions.pdf`**
- Open the files:
  - Document to validate: **`input.xml`**
  - Template Schematron schema: **`template.sch`**
- Enhance **`template.sch`** according to the instructions
- To try it out in oXygen, **`input.xml`** is automatically validated by **`template.sch`**

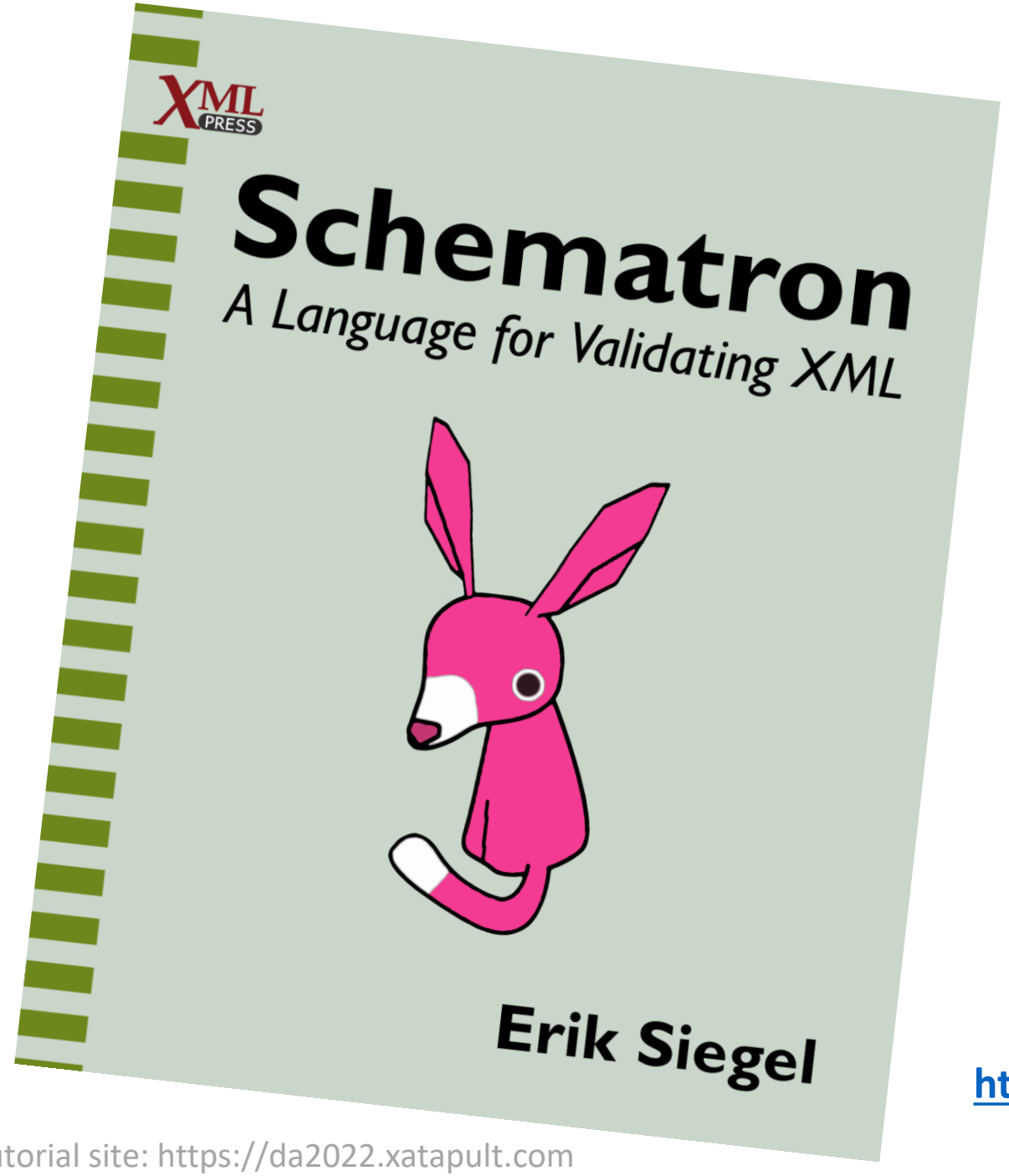
Possible solution in: **`solution/solution.sch`** (short explanation in **`solution/explanation.pdf`**).

Wrap-up

- Schematron is defined as flexible with regards to its query language
  - The Query Language Binding or QLB
- In practice: **`xslt`** (and therefore XPath) only
- Set this using the **`queryBinding`** attribute. Recommended values:
  - **`queryBinding="xslt2"`**
  - **`queryBinding="xslt3"`**
- Allows for XPath 2.0 or 3.1 expressions
- XSLT2 and XSLT3 type query bindings also allow XSLT keys and functions
- Other XSLT features are officially unsupported but usually work

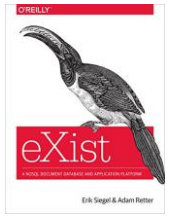
Wrap up

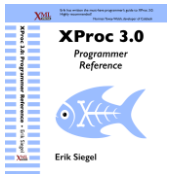
- Schematron is a simple but powerful XML based validation language
- It consists mainly of patterns, rules, asserts and reports
- Messages can be tailored to your (or the user's) needs
- It has mechanisms for re-use of code, among which abstract patterns
- It can incorporate XSLT keys and functions (and, unofficially, more)



- Introductions
- Schematron in context (of other validation languages)
- How to apply Schematron (with SchXslt, in Oxygen, etc.)
- Basics (patterns, rules, asserts/reports, value-of, variables, namespaces)
- Advanced (diagnostics, phases, abstract rules/patterns, includes)
- Query Language Binding
- Additional features (markup, flags, roles, etc.)
- Some further examples and recipes
- Appendices:
  - XPath technology primer
  - Introduction to namespaces
  - Schematron & SVRL reference
  - Introduction to SQF
  - Additional reading

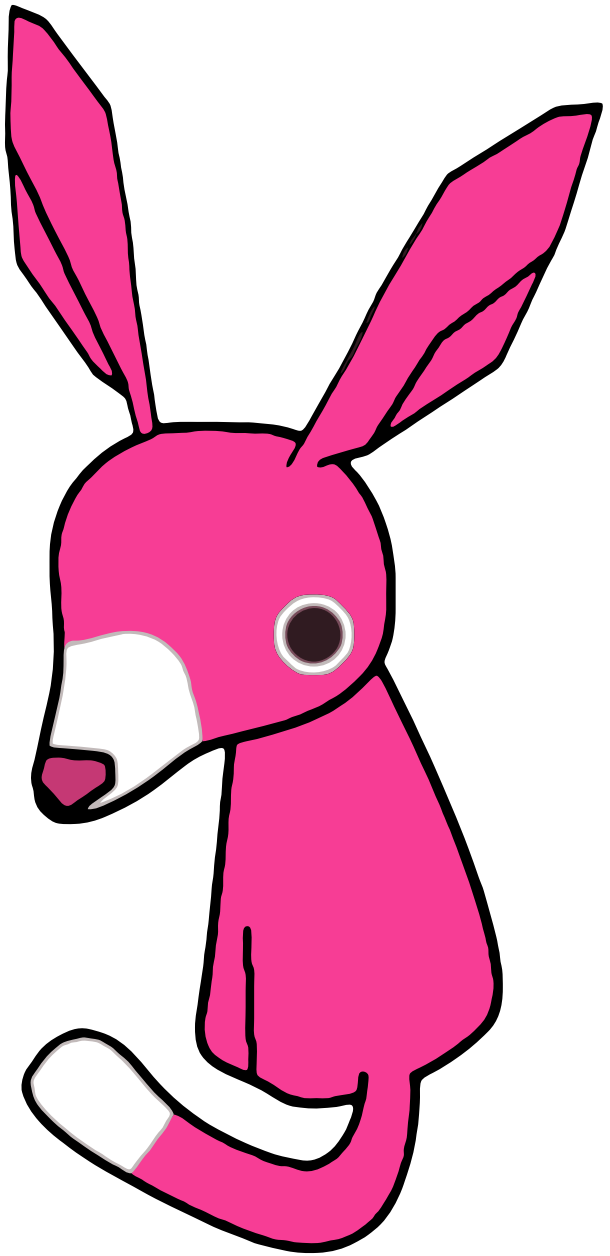
My other books





https://xmlpress.net/publications/schematron/

Questions?



XATAPULT
CONTENT ENGINEERING

Erik Siegel
www.xatapult.com
erik@xatapult.nl