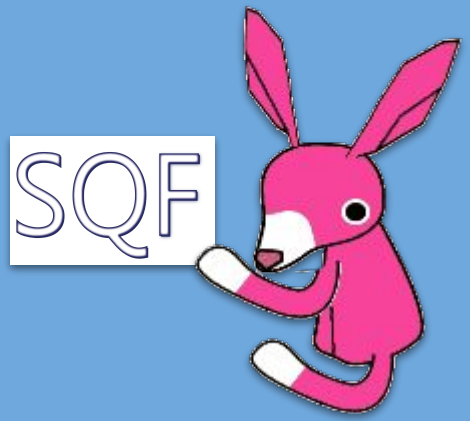
The Power of Schematron Quick Fixes



Octavian Nadolu
octavian_nadolu@oxygenxml.com
@OctavianNadolu

Nico Kutscherauer
kutscherauer@data2type.de
@nkutsche

<oXygen/> XML Editor

data2type

Overview

- Schematron QuickFix Language
- SQF - Second Draft
- Use Cases: QA, Efficiency, SQF/Schematron Interactions

What Is SQF?

Schematron QuickFix (SQF) is an extension of the ISO Schematron standard

User-defined fixes for Schematron errors

SQF History

- XML Prague 2014 – the SQF idea started to take shape
- February 20, 2014 - W3C Community Group "Quick-Fix Support for XML"
- April 2015 - First Draft
- March 2018 – Second Draft

Schematron Quick Fixes Spec

W3C Community Group Draft Report

Schematron Quick Fixes Specification

Quick-fix support for XML Community Group – Draft March 2018

This version:
schematron-quickfix.github.io/sqf/spec/SQFSpec.html
Latest version:
schematron-quickfix.github.io/sqf
Editors:
Nico Kutscherauer
Octavian Nadolu

Abstract

Schematron QuickFix is an extension of Schematron (standard *ISO/IEC 19757-3:2016*). Schematron QuickFix enables developers to define QuickFixes for Schematron errors. QuickFix implementations should present these QuickFixes for the reported Schematron errors to the user. The user can then decide which QuickFix fixes the error in an acceptable way.

www.w3.org/community/quickfix

schematron-quickfix.github.io/sqf

SQF Second Draft – March 2018

- Multilingual support for quick fixes
- Generate quick fixes dynamically
- Changed sqf:keep in sqf:copy-of
- Added @flags for sqf:stringReplace
- Redesign the content model of sqf:fix
- ...

Multilingual Support in SQF

- The name and description of a quick fix are defined by sqf:title and sqf:p elements
- The @ref attribute specifies IDs or keys for alternative localization

```
<sqf:fix id="addBone">
   <sqf:description>
      <sqf:title ref="fix_en fix_de">Add a bone</sqf:title>
      <sqf:p ref="fix_d_en fix_d_de">Add a bone as child element</sqf:p>
   </sqf:description>
   <sqf:add node-type="element" target="bone"/>
</sqf:fix>
```

Localization Using Diagnostics

- Implementation of quick fix localization using Schematron diagnostics
  - A diagnostic element is used for each language
  - The @ref attribute refers diagnostic IDs

```
<sqf:fix id="addBone">
   <sqf:description>
      <sqf:title ref="fix_en fix_de">Add a bone</sqf:title>
   </sqf:description>
   <sqf:add node-type="element" target="bone"/>
</sqf:fix>
....
<sch:diagnostics>
   <sch:diagnostic id="fix_en" xml:lang="en">Add a bone</sch:diagnostic>
   <sch:diagnostic id="fix_de" xml:lang="de">Fügen Sie einen Knochen hinzu</sch:diagnostic>
</sch:diagnostics>
```

Localization Using Property Files

- Implementation of quick fix localization using Java Property Files
  - A property file for each language “fileName_lang.xml”
  - The @ref attribute refers the property key

```
<sqf:fix id="addBone">
   <sqf:description>
      <sqf:title ref="dog.addBone.title" xml:lang="en">Add a bone</sqf:title>
      <sqf:p ref="dog.addBone.p">Add bone element as child</sqf:p>
   </sqf:description>
   <sqf:add node-type="element" target="bone"/>
</sqf:fix>
```

```
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
   <entry key="sample.dog.addBone.title">Füge einen Knochen hinzu</entry>
   <entry key="sample.dog.addBone.p">Der Hund wird einen Knochen erhalten.</entry>
</properties>
```

Generate Quick Fixes Dynamically

- Added support to generate quick fixes dynamically using the @use-for-each attribute
- Generate a quick fix for each match of the @use-for-each attribute

```
<sqf:fix id="removeAnyItem" use-for-each="1 to count(li)">
   <sqf:description>
      <sqf:title>Remove item #<sch:value-of select="$sqf:current"/></sqf:title>
   </sqf:description>
   <sqf:delete match="li[$sqf:current]"/>
</sqf:fix>
```

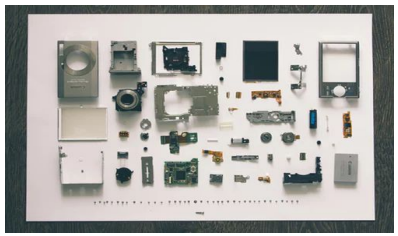
Used in Multiple Domains

Financial

Insurance

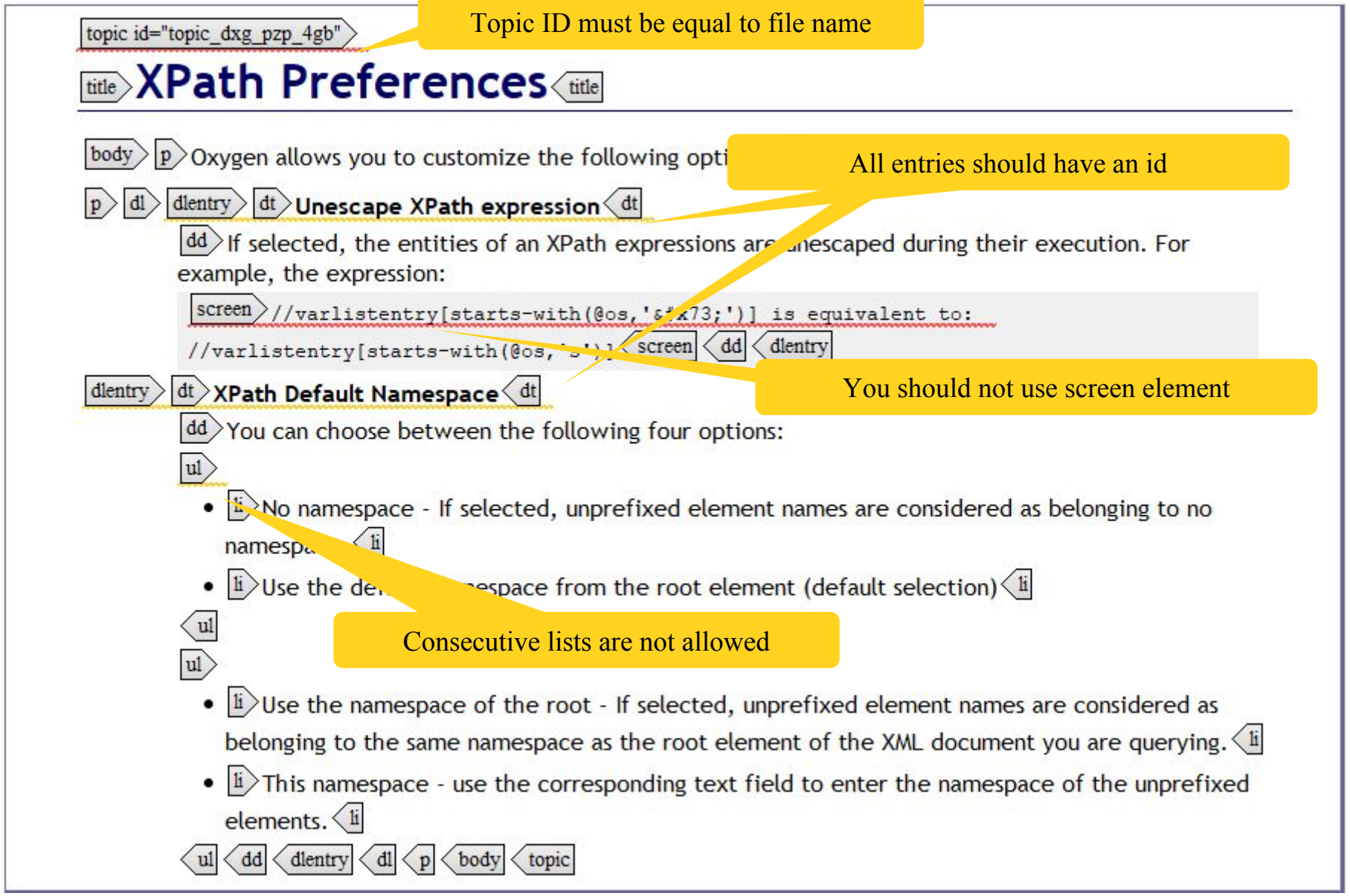
Government

Technical publishing



Use Cases

1. Quality Assurance



Users

Content Writer – A subject matter expert, but doesn't know XML very well

XML Expert – Knows XML structure, but doesn't know much about the actual content

Using the quick fixes created by the XML Expert the Content Writer can correct the problem

Example

Topic ID must be equal to file name

Set the file name as the topic ID

All entries should have an id

Add id to the current entry

Add id to the all entries from document

You should not use screen element

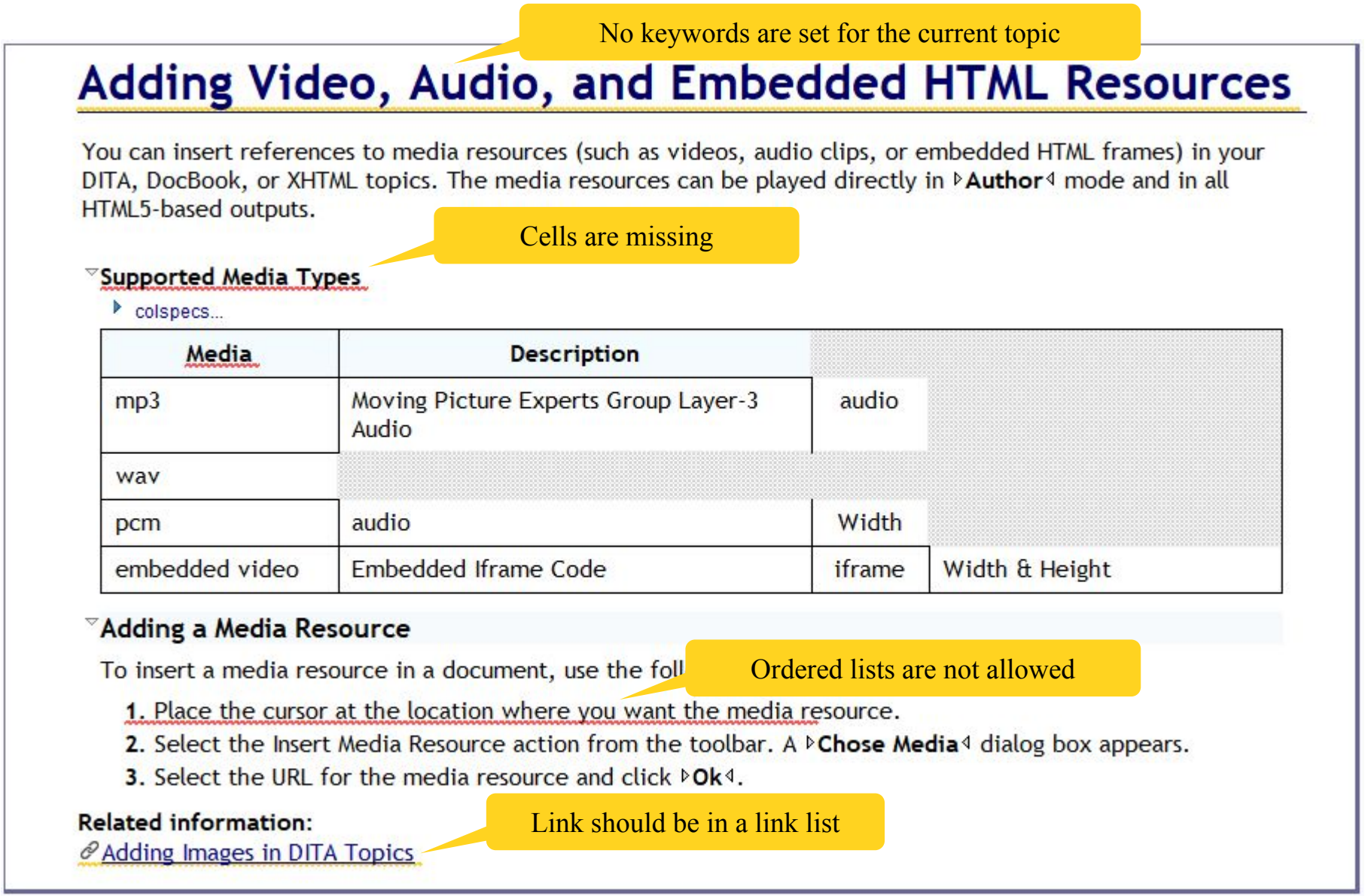
Replace screen with codeblock

Consecutive lists are not allowed

Merge lists into one

Add a paragraph after first list

2. Efficiency



Example

No keywords are set for the current topic

Add keywords for the current topic

Cells are missing

Add enough empty cells on each row

Ordered lists are not allowed

Convert ordered list to unordered list

Link should be in a link list

Create a link list with the current link

3. Library of Quick Fixes

- SQF basic operations: add, delete, replace, stringReplace
- A library of generic quick fixes, such as:
  - Wrap/Unwrap element content
  - Rename element
  - Merge

Abstract SQF

Rename element abstract quick fix:

```
<sqf:fix id="renameElement" role="replace">
   <sqf:param name="element" abstract="true"/>
   <sqf:param name="newName" abstract="true"/>
   <sqf:description>
      <sqf:title>Rename '$element' element in '$newName'</sqf:title>
   </sqf:description>
   <sqf:replace match="." target="$newName" node-type="element" select="node()"/>
</sqf:fix>
```

Abstract SQF

Abstract pattern:

```
<sch:pattern id="renamePattern" abstract="true">
   <sch:rule context="$element">
      <sch:assert test="false()" sqf:fix="renameElement">
         Element '$element' not allowed, use '$newName' instead.
      </sch:assert>
   </sch:rule>
</sch:pattern>
```

Pattern instantiation:

```
<sch:pattern is-a="renamePattern">
   <sch:param name="element" value="orderedlist"/>
   <sch:param name="newName" value="itemizedlist"/>
</sch:pattern>
```

```
<sch:pattern is-a="renamePattern">
   <sch:param name="element" value="ol"/>
   <sch:param name="newName" value="ul"/>
</sch:pattern>
```

4. Ignoring in Schematron

- Schematron has errors, warnings, informations
- Warnings are like Schrödinger's cat:
  - If you don't look at it, they are good and bad at the same time
  - Good warning needs to be marked
    ➡ **Ignoring**

Source: https://www.leifiphysik.de/

- Ignoring in Schematron is not a big deal:
  - Mark the context with a PI or a specific attribute
  - Exclude marked nodes from your tests.

Ignoring in Schematron

- Example from our XSLT styleguide

```
<sch:pattern id="p_01_unused_items" xmlns:d2t="http://data2type.de/">

    <sch:rule context="*[@d2t:ignore = 'p_01_unused_items']"/>

    <sch:rule context="xsl:template[@name]">

        <!-- Check template usage -->
        <sch:assert test="..."/>
```

  - Elements with the `d2t:ignore` attribute and the value `p_01_unused_items` will be ignored.
  - For convention the value is equal to the pattern ID.
  - With the specific namespace `http://data2type.de/` name conflicts are avoided

➡ To many conventions to be practicable with Schematron only

Ignoring in Schematron

- Problem in practice: the user has to know a lot:
  - That it is possible to ignore the warning
  - That you have to ignore with an ignore attribute.
  - The ***exact*** URL of the specific namespace
  - The ***exact*** pattern ID of each Schematron warning

➡ Ignoring with pur Schematorn is not practical

Ignoring with SQF

- SQF makes ignoring practical
- A simple QuickFix helps:

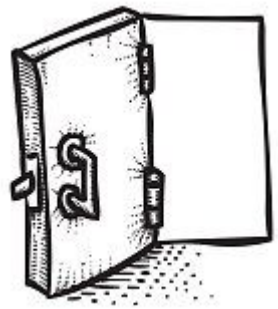
```
<sqf:fix id="ignore_p_01_unused_items">
    <sqf:description>
        <sqf:title>Ignore this warning</sqf:title>
    </sqf:description>
    <sqf:add match="." target="d2t:ignore" node-type="attribute"
        select="'p_01_unused_items'"/>
</sqf:fix>
```

  - The user has just to use the QuickFix, to ignore the warning

Ignoring with SQF

- Remarkable:
  - QuickFix is not used to fix the document
  - SQF and Schematron interacts
    - Schematron checks the document
    - QuickFix makes a manipulation
    - On next validation circle Schematron reacts depending on that manipulation

➡ Door opener for new ideas



5. Guided Development

- Example XSD Guide
  - Multiple ways to express the same thing in XSD
  - Inexperienced users has problems to follow a design pattern
    - *Google based development* instead of *Venetian Blind* or *Salami Slice*
  - The XSD Guide leads the user through XSD based on a selected design pattern.
  - Based on Schematron, SQF and XSLT only

XSD Guide – Demo

https://github.com/octavianN/thePowerOfSQF/Samples/xsdguide/

Guided Development

- How does it works?
  - Interaction by `xs:annotation/xs:appInfo/d2t:xsdguide`
  - User input by `<sqf:user-entry>`
    - Edit Content with @default
  - Multiple choice selections by:
    - Different QuickFixes
    - Generic QuickFixes (@use-for-each)
    - Drop-down menu in User Entry
      (Escali plugin only feature: by
      `sqf:user-entry/@default` returning a sequence)
  - Complex tasks by using XSLT functions

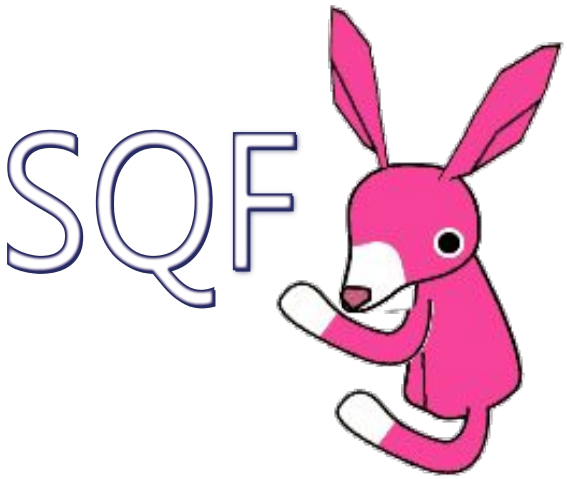
Guided Development

- XSD Guide is just an example
- There are other large and confusing standards in the XML world
  - With user without a deep understanding
  - You can provide shortcuts for verbose syntax
  - You may have to work with PIs for interactions

SQF Became a Powerful Language

and

Made Schematron More Powerful



SQF Implementations

- <oXygen/> XML Editor validation engine
- http://www.oxygenxml.com
- Escali Schematron engine
- http://schematron-quickfix.com/escali_xsm.html
  - Escali Schematron command-line tool
  - Oxygen plugin for invoking Escali Schematron

Resources

- Schematron Quick Fix specification
  http://schematron-quickfix.github.io/sqf
- Schematron QuickFix site
  http://www.schematron-quickfix.com/
- Schematron site

  http://schematron.com/
- Schematron specification
  https://standards.iso.org/ittf/PubliclyAvailableStandards
- Samples
  https://github.com/octavianN/thePowerOfSQF

THANK YOU!

Any questions?

octavian_nadolu@oxygenxml.com
@OctavianNadolu

contact@schematron-quickfix.com
@nkutsche