INTERACTIVE INTELLIGENT STYLE GUIDE

George Bina, oXygen XML Editor

george@oxygenxml.com

@georgebina

TCUK 2019

About me

#xml #oxygenxml #syncrosoft #opensource #jing #ditaot #dance #tango

Style guide

A set of rules that people should follow when writing content

The style guide can be…

> a **text** document

> a **word processor** document

> one or more **HTML** documents

to be published on the web, thus accessible through a link

> one or more structured **XML** documents

to enable web publishing and other automated processing

Example of a possible issue

Code blocks with long lines may look ugly or they may even become incorrect

Code block in DITA

Sample

Here we have a sample code:

```
public void testGetAttributes() throws Exception{
  String schemaStr =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
    "<xs:schema xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" targetNamespace=\"oxygenNS\" xmlns:oxy=\"oxygenNS\">\n" +
    "    <xs:attribute name=\"attr1\" default=\"defaultVal\" type=\"xs:string\"/>\n" +
    "    <xs:attribute name=\"attr3\" default=\"defaultVal\" type=\"xs:string\"/>\n" +
    "    <xs:element name=\"elem\">\n" +
    "        <xs:complexType>\n" +
    "            <xs:attribute ref=\"oxy:attr1\" default=\"overwriteVal\"/>\n" +
    "            <xs:attribute name=\"attr2\" type=\"xs:token\" fixed=\"20\"/>\n" +
    "            <xs:attribute ref=\"oxy:attr3\" use=\"required\"/>\n" +
    "        </xs:complexType>\n" +
    "    </xs:element>\n" +
    "</xs:schema>";
```

Code block in PDF

Sample

Here we have a sample code:

```
  public void testGetAttributes() throws Exception{
    String schemaStr =
      "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
      "<xs:schema xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
 targetNamespace=\"oxygenNS\" xmlns:oxy=\"oxygenNS\">\n" +
      "    <xs:attribute name=\"attr1\" default=\"defaultVal\" type=
\"xs:string\"/>\n" +
      "    <xs:attribute name=\"attr3\" default=\"defaultVal\" type=
\"xs:string\"/>\n" +
      "    <xs:element name=\"elem\">\n" +
      "        <xs:complexType>\n" +
      "            <xs:attribute ref=\"oxy:attr1\" default=\"overwriteVal\"/
>\n" +
      "            <xs:attribute name=\"attr2\" type=\"xs:token\" fixed=
\"20\"/>\n" +
      "            <xs:attribute ref=\"oxy:attr3\" use=\"required\"/>\n" +
      "        </xs:complexType>\n" +
      "    </xs:element>\n" +
      "</xs:schema>";
```

Line split at whitespace

Sample

Here we have a sample code:

```
public void testGetAttributes() throws Exception{
  String schemaStr =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
    "<xs:schema xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" targetNamespace=\"oxygenNS\" xmlns:oxy=\"oxygenNS\">\n" +
    "    <xs:attribute name=\"attr1\" default=\"defaultVal\" type=\"xs:string\"/>\n" +
    "    <xs:attribute name=\"attr3\" default=\"defaultVal\" type=\"xs:string\"/>\n" +
    "    <xs:element name=\"elem\">\n" +
    "        <xs:complexType>\n" +
    "            <xs:attribute ref=\"oxy:attr1\" default=\"overwriteVal\"/>\n" +
    "            <xs:attribute name=\"attr2\" type=\"xs:token\" fixed=\"20\"/>\n" +
    "            <xs:attribute ref=\"oxy:attr3\" use=\"required\"/>\n" +
    "        </xs:complexType>\n" +
    "    </xs:element>\n" +
    "</xs:schema>";
```

PDF code block may become incorrect

Sample

Here we have a sample code:

```
public void testGetAttributes() throws Exception{
  String schemaStr =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
    "<xs:schema xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
targetNamespace=\"oxygenNS\" xmlns:oxy=\"oxygenNS\">\n" +
```

ERROR!

```
public void testGetAttributes() throws Exception{
  String schemaStr =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
    "<xs:schema xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
    targetNamespace=\"oxygenNS\" xmlns:oxy=\"oxygenNS\">\n" +
```

Limiting the code blocks line length

Sample

Here we have a sample code:

```
public void testGetAttributes() throws Exception{
  String schemaStr =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
    "<xs:schema xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" "+
    "  targetNamespace=\"oxygenNS\" xmlns:oxy=\"oxygenNS\">\n" +
    "  <xs:attribute name=\"attr1\" default=\"defaultVal\" +
    "    type=\"xs:string\"/>\n" +
    "  <xs:attribute name=\"attr3\" default=\"defaultVal\" +
    "    type=\"xs:string\"/>\n" +
    "  <xs:element name=\"elem\">\n" +
    "    <xs:complexType>\n" +
    "      <xs:attribute ref=\"oxy:attr1\" default=\"overwriteVal\"/>\n" +
    "      <xs:attribute " +
    "        name=\"attr2\" type=\"xs:token\" fixed=\"20\"/>\n" +
    "      <xs:attribute ref=\"oxy:attr3\" use=\"required\"/>\n" +
    "    </xs:complexType>\n" +
    "  </xs:element>\n" +
    "</xs:schema>";
```

DITA

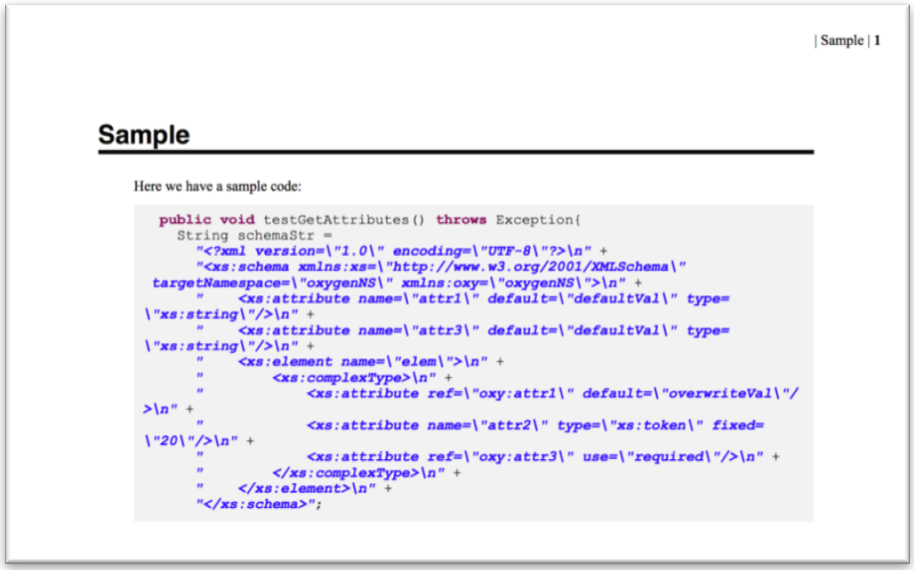
PDF

Sample

Here we have a sample code:

```
public void testGetAttributes() throws Exception{
  String schemaStr =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
    "<xs:schema xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" "+
    "  targetNamespace=\"oxygenNS\" xmlns:oxy=\"oxygenNS\">\n" +
    "  <xs:attribute name=\"attr1\" default=\"defaultVal\" +
    "    type=\"xs:string\"/>\n" +
    "  <xs:attribute name=\"attr3\" default=\"defaultVal\" +
    "    type=\"xs:string\"/>\n" +
    "  <xs:element name=\"elem\">\n" +
    "    <xs:complexType>\n" +
    "      <xs:attribute ref=\"oxy:attr1\" default=\"overwriteVal\"/>\n" +
    "      <xs:attribute " +
    "        name=\"attr2\" type=\"xs:token\" fixed=\"20\"/>\n" +
    "      <xs:attribute ref=\"oxy:attr3\" use=\"required\"/>\n" +
    "    </xs:complexType>\n" +
    "  </xs:element>\n" +
    "</xs:schema>";
```

Comparison (PDF)

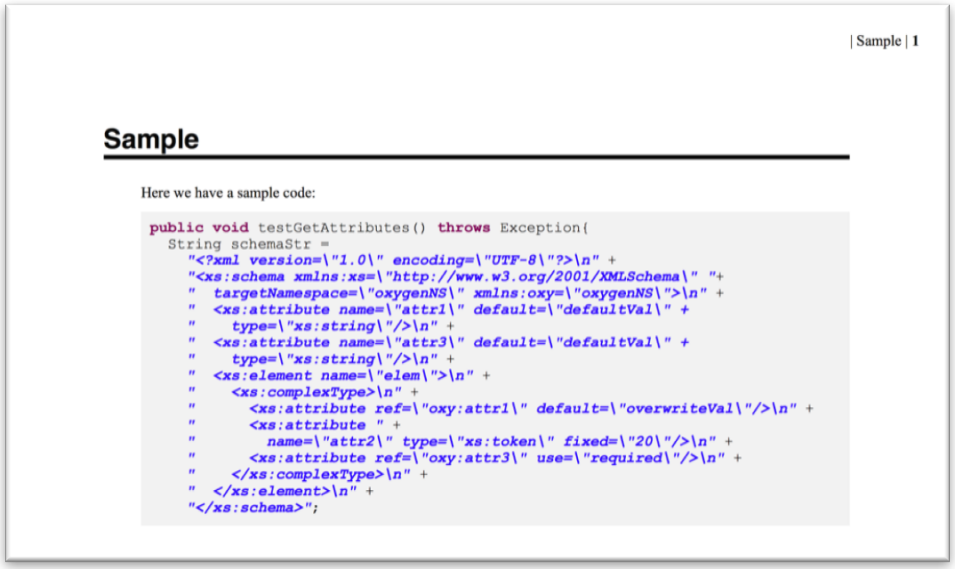


| Sample | 1

Sample

Here we have a sample code:

```
 public void testGetAttributes() throws Exception{
   String schemaStr =
     "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
     "<xs:schema xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
 targetNamespace=\"oxygenNS\" xmlns:oxy=\"oxygenNS\">\n" +
     "    <xs:attribute name=\"attr1\" default=\"defaultVal\" type=
\"xs:string\"/>\n" +
     "    <xs:attribute name=\"attr3\" default=\"defaultVal\" type=
\"xs:string\"/>\n" +
     "    <xs:element name=\"elem\">\n" +
     "        <xs:complexType>\n" +
     "            <xs:attribute ref=\"oxy:attr1\" default=\"overwriteVal\"/
>\n" +
     "            <xs:attribute name=\"attr2\" type=\"xs:token\" fixed=
\"20\"/>\n" +
     "            <xs:attribute ref=\"oxy:attr3\" use=\"required\"/>\n" +
     "        </xs:complexType>\n" +
     "    </xs:element>\n" +
     "</xs:schema>";
```



| Sample | 1

Sample

Here we have a sample code:

```
public void testGetAttributes() throws Exception{
  String schemaStr =
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
    "<xs:schema xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" "+
    "  targetNamespace=\"oxygenNS\" xmlns:oxy=\"oxygenNS\">\n" +
    "  <xs:attribute name=\"attr1\" default=\"defaultVal\" +
    "    type=\"xs:string\"/>\n" +
    "  <xs:attribute name=\"attr3\" default=\"defaultVal\" +
    "    type=\"xs:string\"/>\n" +
    "  <xs:element name=\"elem\">\n" +
    "    <xs:complexType>\n" +
    "      <xs:attribute ref=\"oxy:attr1\" default=\"overwriteVal\"/>\n" +
    "      <xs:attribute " +
    "        name=\"attr2\" type=\"xs:token\" fixed=\"20\"/>\n" +
    "      <xs:attribute ref=\"oxy:attr3\" use=\"required\"/>\n" +
    "    </xs:complexType>\n" +
    "  </xs:element>\n" +
   "</xs:schema>";
```

#ugly #incorrect

#nice #correct

We should record this rule

Code blocks lines should be limited to a certain maximum length

Why do we need a style guide?

Try to avoid repeating mistakes

Style guides are great, but…

… we can reach a point where we have a lot of rules and an overhead to get started

Too many rules!

At some point, we may not be able to keep track of all the style guide rules

Overhead

One should learn all the style guide rules before writing!

Solution?

How can we avoid learning all the rules, even before we start writing?

Solution: automate rules

Automatically detect when your content does not follow a specific rule

If possible, offer one or more solutions the user can choose from

Cars – automate checks and solutions

Intelligent cars:

- Issue a warning if you are about to hit another car or a large object
  - Prepare and even apply the brakes for you
- Issue a warning when you cross a lane guideline without signaling
  - Turn the wheel to keep you on your lane

We should benefit of similar support!

There are technologies designed for doing exactly that:

- Schematron
- Schematron Quick Fix

Schematron

One possible choice for style guide rule automation

What is Schematron?

- An ISO Standard

  ISO/IEC 19757 – DSDL (Document Schema Definition Language) Part 3: Rule-based validation

- A very simple schema language

  Less than 10 main elements, about 20 elements in total

- A different kind of schema

  Defines business rules, not the document structure

Why Schematron

- Designed to check patterns in structured documents
- Support for text processing through regular expression checks
- With the addition of SQF it can provide automatic fixes
- Can be organized in a way that splits responsibilities in roles
- Can link to style guide (a rule may contain a *@see* attribute)
- Can be integrated within a structured style guide

SQF – Schematron Quick Fix

Provide actions the user can choose in order to correct the reported problem

- similar to choosing a correct word when you have a spelling mistake -

Sample automated rule

Let’s look at some automated rules with Schematron and SQF

Possible fixes for user guide rules

- No dynamic image scaling (image/@scale)
  - Remove the @scale attribute
- Avoid “oXygen” in index terms
  - Delete “oXygen” from the index term
- No “;” at the end of list items
  - Remove “;”
  - Repalce “;” with some other character
- Index terms are allowed only in prolog
  - Move index term in prolog
- Topic IDs should match the topic filename
  - Set the topic ID to match the filename-based pattern
- No link text equal with the referred URL
  - Remove the link text
- Consecutive lists
  - Join the two consecutive lists
- Definition lists should be inside paragraphs
  - Wrap the definition list in a paragraph
- Suggest specifying a language on code blocks
  - Set the @lang attribute
- Reports consecutive notes of the same type
  - Change the note type
  - Merge their content
- Report empty elements
  - Remove the empty element
- Report tables with more cells than the declared columns
  - Remove the additional cells
- Sections should have IDs
  - Add @id attributes on sections
- Sections should have titles
  - Add title element in sections
- No text directly inside a section
  - Delete the text
  - Wrap the tect in a paragraph

Information Architect and Developer roles

Schematron can be organized to split responsibilities in different roles!

Profiles

Information Architect

- Good domain knowledge
- Knows what rules are needed
- No technical background
- No Schematron or SQF knowledge

Developer

- No domain knowledge
- Knows how to build rules
- Good technical background
- Schematron and SQF knowledge or ready to learn them

Workflow

- Information Architect decides to add a new rule to the style guide
- IA sends a request to the Developer to implement an automatic check
- The Developer creates the Schematron code to implement that check

Generic rules

Avoid starting an "indexterm" element with "oXygen"

Delete "oXygen"

Avoid starting a "note" element with "Note:"

Delete "Note:"

…

→

Avoid starting a "*parent*" element with a "*fragment*"

Delete "*fragment*"

"*parent*" and "*fragment*" are the generic rule parameters

Generic rules library and rules instantiations

Generic rules library (Developer)

avoidStartFragment
parent
fragment

…

Actual rules (Information Architect)

avoidStartFragment
parent = “indexterm”,
fragment = “oXygen”

avoidStartFragment
parent = “note”,
fragment = “Note:”

…

Improved workflow using generic rules

- Information Architect decides to add a new rule to the style guide
- If an existing generic rule cannot be used to automate that new rule
  - IA sends a request to the Developer to implement a new generic rule
  - The Developer creates the Schematron code to implement that generic rule
  - The new generic rule becomes available in a generic rules library
- IA identifies the generic rule and sets the corresponding parameters to automate the style guide new rule

Improved workflow advantages

- Less interaction between Developer and IA, only when a new generic rule is needed
- IA is in control of the actual rules, being able to modify them easily as needed, just by setting values to the corresponding parameters
- Limits the need for a Developer role, even to zero, if using an existing library of generic rules

Intelligent Style Guide

Why keep rules separate from style guide?

Embed rules into a structured style guide

indexing.dita

How to use indexing

Index terms should be defined only in the metadata section of a topic.

Because the user guide refers to our product, we should not use the product name in index terms.

Rule	avoidStartFragment
parent	indexterm
fragment	oXygen

avoidStartFragment

parent = “indexterm”,

fragment = “oXygen”

DIM
Dynamic Information Model

An example of an intelligent style guide

http://www.github.com/oxygenxml/dim

DIM example

Document containing a problem

task prolog metadata keywords indexterm

Sample

Short Description: test

Metadata

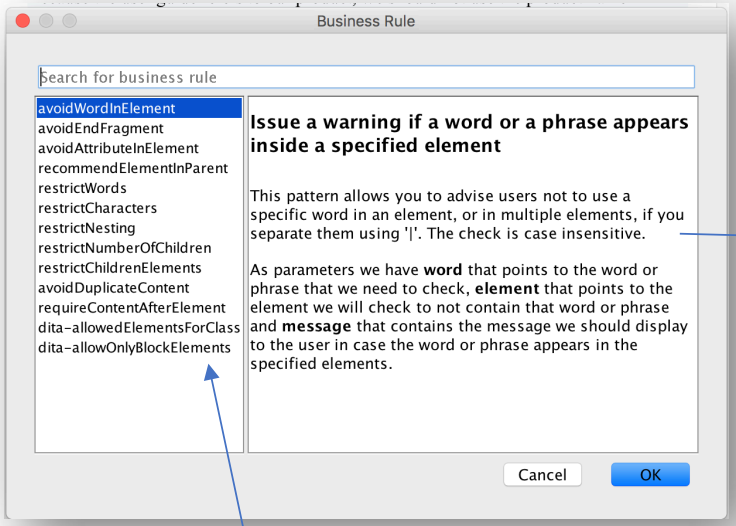
Keywords:

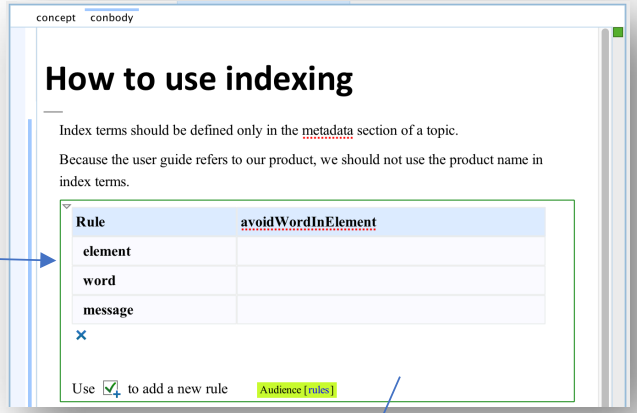
(oXygen validation)

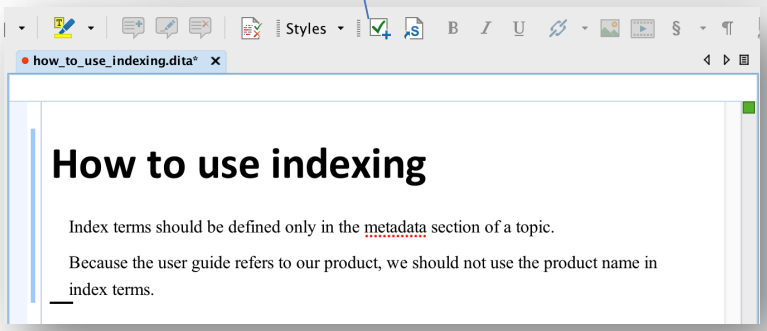
1. First command

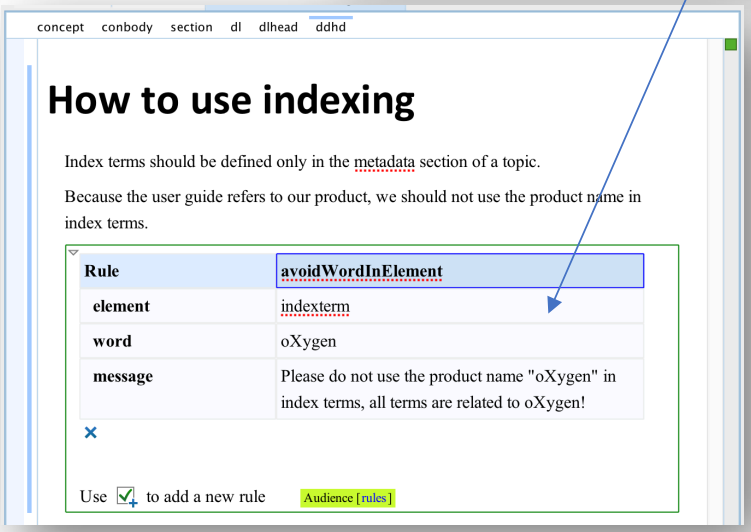
Text Grid Author

Adding a new rule





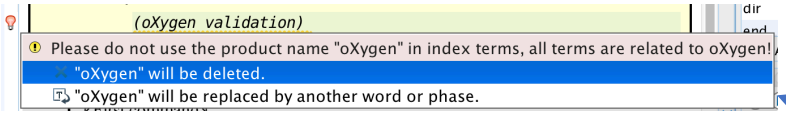


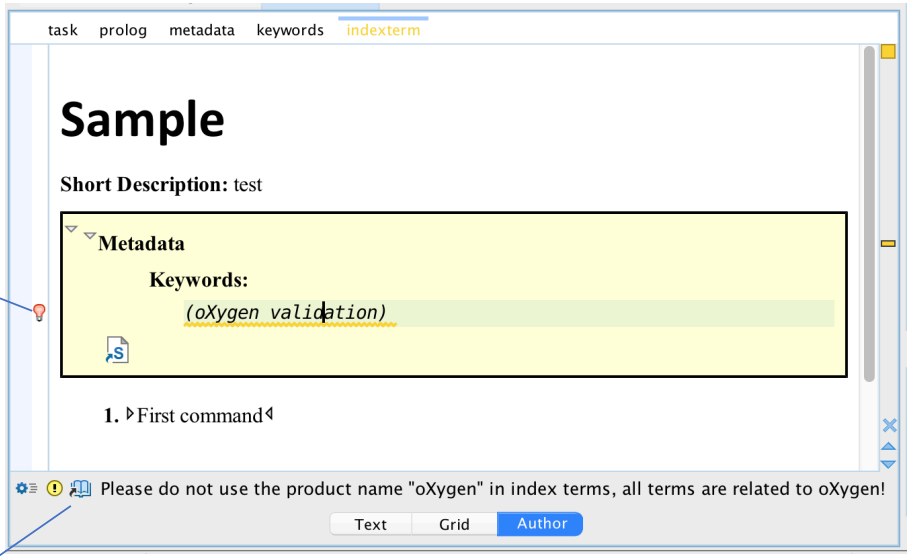


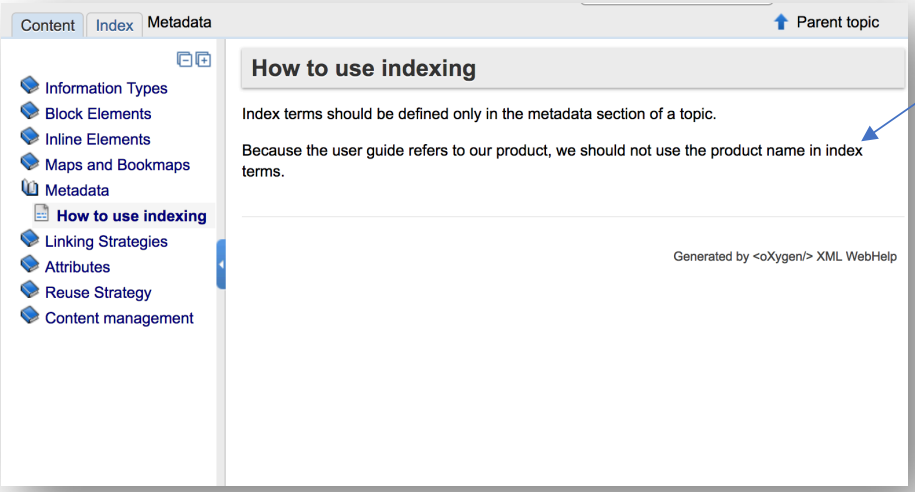
Schematron rule extracted from style guide

```
<!--Generated from how_to_use_indexing.dita.
 -->
<pattern is-a="avoidWordInElement"
         see="http://example.com/styleguide/webhelp/how_to_use_indexing.html">
   <param name="element" value="indexterm"/>
   <param name="word" value="oXygen"/>
   <param name="message"
          value="Please do not use the product name &#34;oXygen&#34; in index terms,
                 all terms are related to oXygen!"/>
</pattern>
```

Detected issue, link to style guide, and fixes



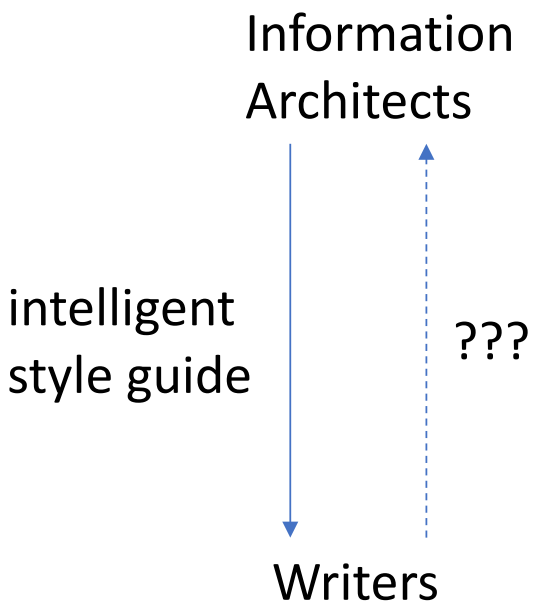




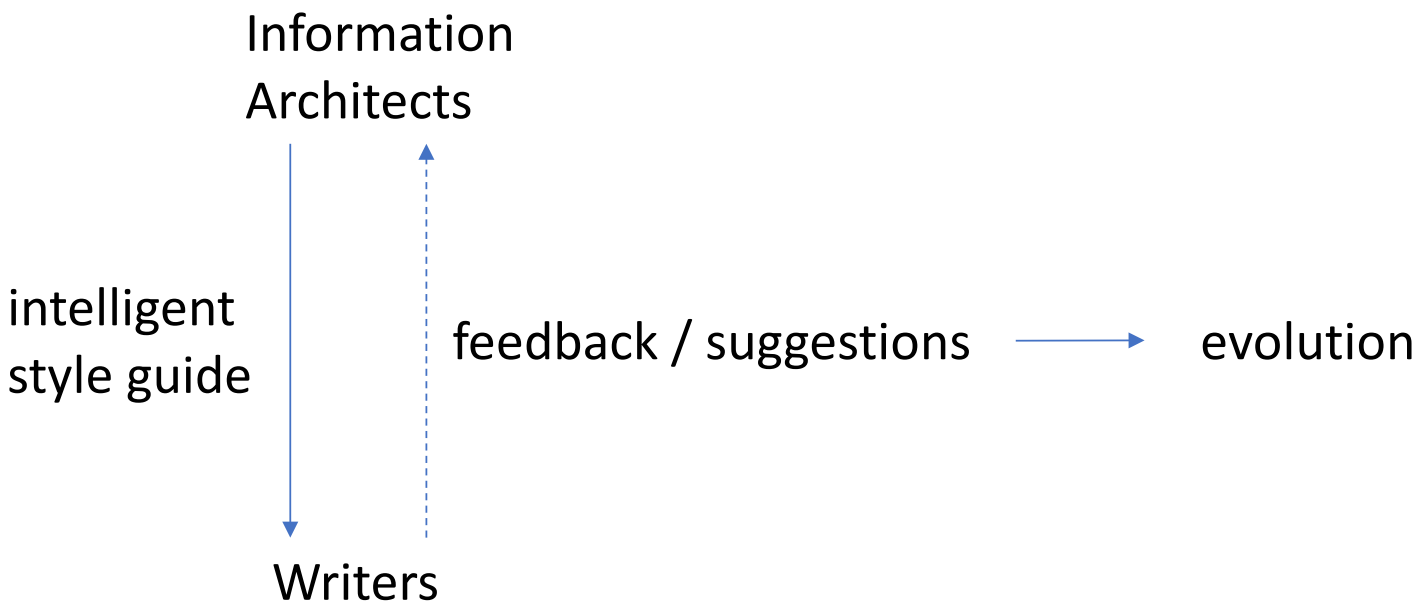
Communication

Intelligent style guides allow information architects to communicate with writers

Interactive - dialog



Interactive - dialog

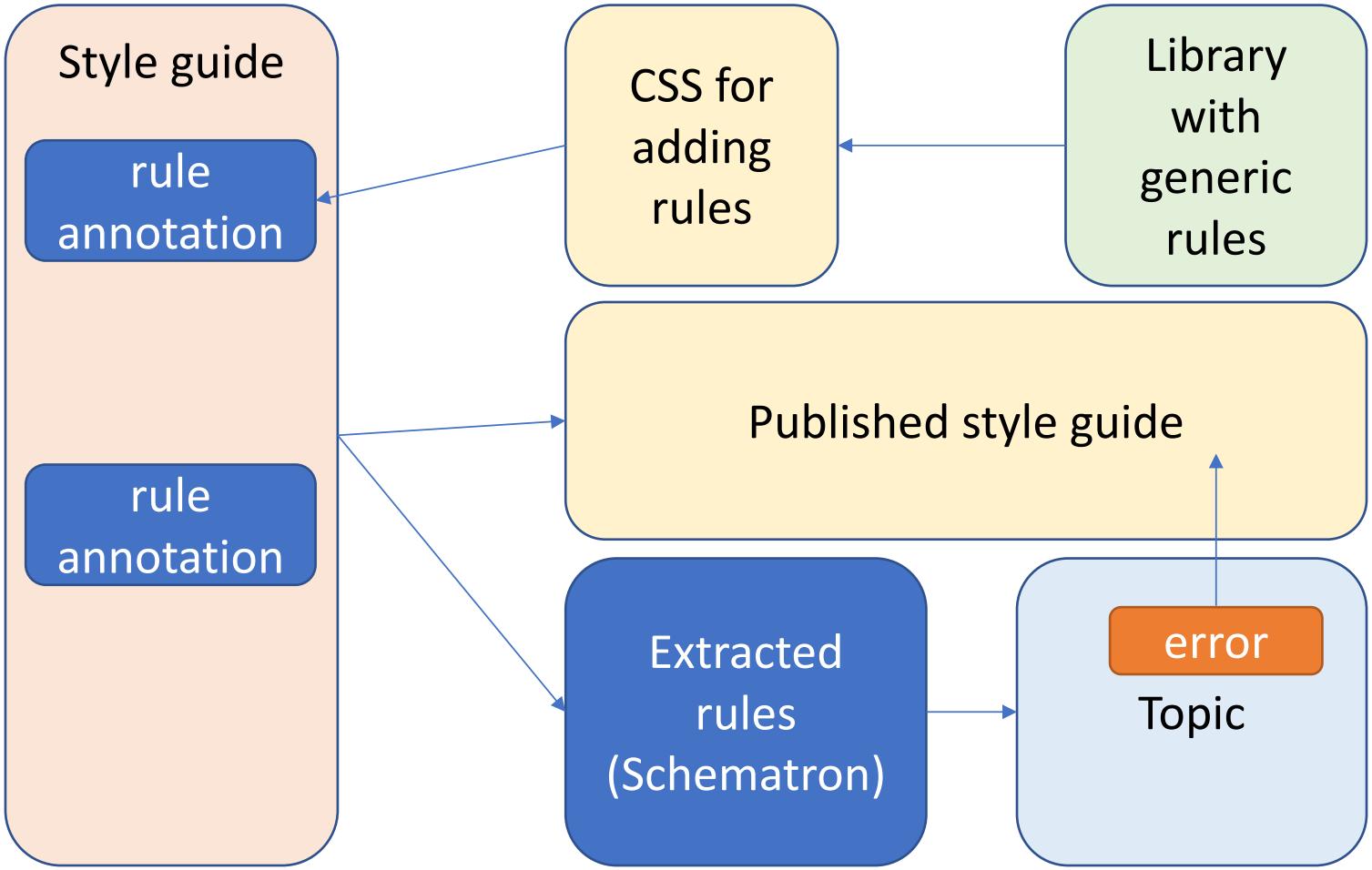


Integrated Style Guide

An example of an interactive intelligent style guide

https://github.com/oxygenxml/integrated-styleguide

Overview


Style guide
rule annotation
rule annotation
CSS for adding rules
Library with generic rules
Published style guide
Extracted rules (Schematron)
error
Topic

Interactive Intelligent Style Guide

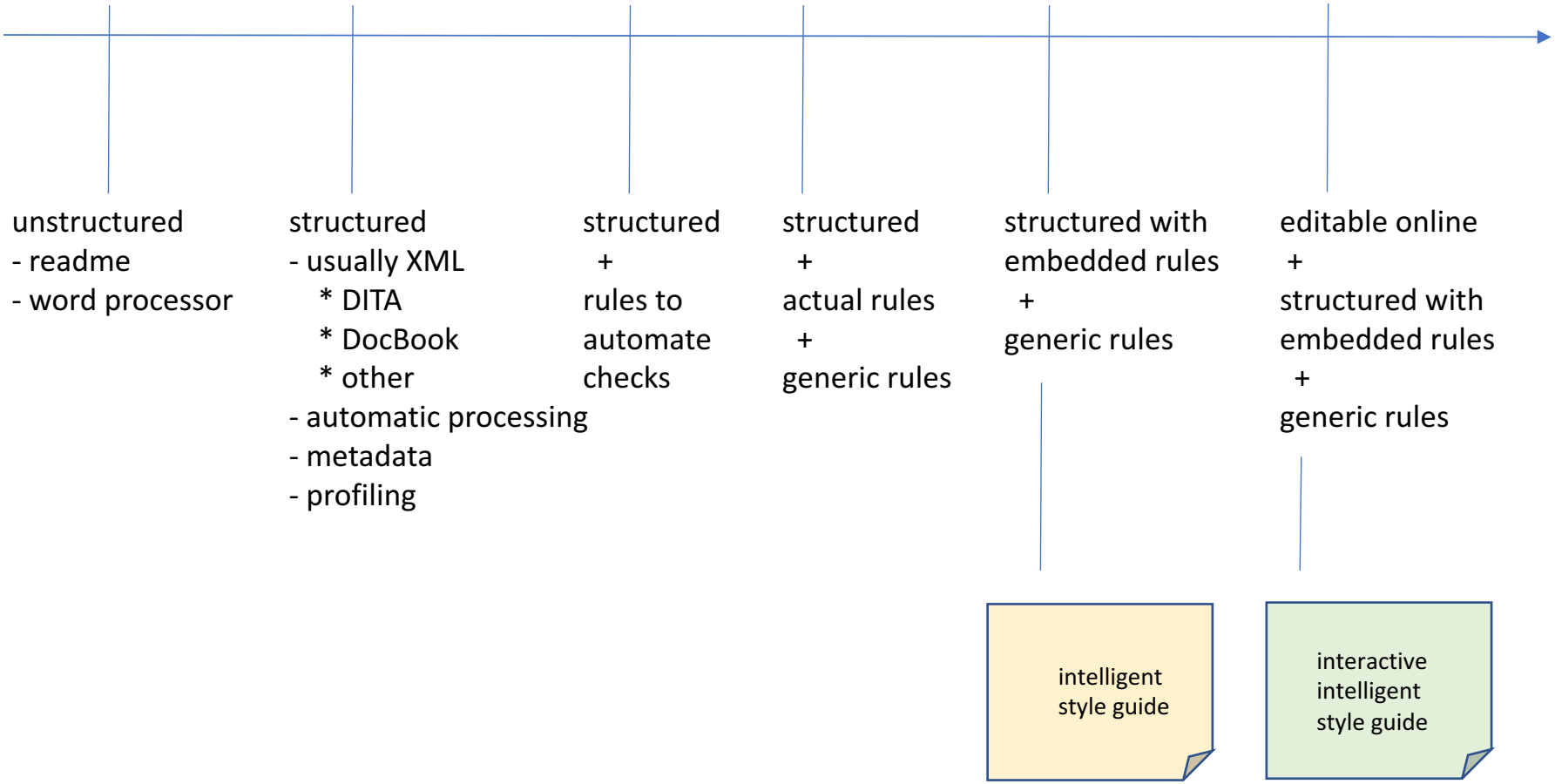
- Open source, hosted in a Git repository on GitHub
  https://github.com/oxygenxml/integrated-styleguide
- Published on GitHub Pages at
  https://oxygenxml.github.io/integrated-styleguide/
  http://oxygenxml.github.io/integrated-styleguide/rules/rules.sch
- Integrated with a CI (Travis) to automatically publish HTML and the extracted Schemaron rules
- Editable online with oXygen Web Author demo service, using the Edit links on each style guide page
- Customized UI to easily add new rules by instantiating a generic rule

DEMO

See the integrated style guide in action

- add a new rule
- see how the style guide is re-published and Schematron rules are generated
- use the style guide on an XML document and see how it notifies the writer

Style guide format evolution



Thank you!

Questions?

George Bina
george@oxygenxml.com
@georgebina