Towards Fixing Documentation Smells

Dear developer, thank you for being a part of this study!

APIs are interfaces to reusable software libraries and frameworks. APIs typically are supported by official documentation. A good documentation can facilitate the proper usage of an API, while a bad documentation can severely harm its adoption.

Several research show that API official documentation can often be incomplete, incorrect, and outdated.

This survey is focused to facilitate a research on a special kind of documentation problem that is Documentation Smell i.e. presentation related issues in documentation.

The survey will take 15-20 minutes to complete. All the responses will be anonymized and we will not share your personal information with anyone. You can opt out of the survey at any time.

Thank you very much for your participation!

You can opt out from this survey at any time by emailing the researchers below. You can decide to remove your response from our study within 30 days after you respond to this survey (and you give your email address in the response for us to track your responses). For removal request, please send email to the researchers below.

Sincerely,

Gias Uddin (gias.uddin@ucalgry.ca) &
Junaed Younus Khan (junaedyounus.khan@ucalgary.ca)

* Indicates required question

1. By participating in this survey, you consent that you are at least 18 years of age and * that you understood that we will not store any personally identifying information about you, that the participation to this survey is entirely voluntary, and that you can opt out from the survey at any time.

   Mark only one oval.

   - ◯ Yes
   - ◯ No

Information About You

In this section, we ask some questions about you that include your current profession, experience, and email address.

2. Current Profession *

   Mark only one oval.

   - ◯ Software Developer
   - ◯ Technical lead
   - ◯ Student
   - ◯ Research engineer
   - ◯ Faculty member
   - ◯ Other: _____

3. Year(s) of experience in software development *

Mark only one oval.

- 0 - 2
- 3 - 5
- 6 - 8
- 9 - 11
- 12 - 14
- 15 or more

Documentation Smell

A documentation smell is a bad design or presentation issue in documentation that does not make the documentation incorrect, but makes it difficult to understand it and use the underlying documentation unit (e.g. a method/function).

We have identified 5 types of documentation presentation smells. They are:
1. Lazy
2. Fragmented
3. Excessive Structural Info
4. Bloated
5. Tangled

This survey is focused on collecting your thoughts towards fixing these five types of smells. For this purpose, you will be shown some the smelly documentations and their fixed (non-smelly) versions in the next few sections. Please provide your valuable feedback on them.

1. Lazy Documentation

Definition: Documentations that contain very small information to convey to the readers. Does not contain any extra information except what can be perceived directly from the function name.

Let's have a look at a Lazy Documentation.

Method Prototype
public MouseMotionListener getMouseMotionListener()
Documentation
Implementation of ComboPopup.getMouseMotionListener().

Now, see the fixed non-Lazy version of the same documentation.

Method Prototype
public MouseMotionListener getMouseMotionListener()
Documentation
Returns the existing MouseMotionListener object. If there is no existing MouseMotionListener object, then creates a new one and returns that.

4. Do you agree with the fixed Non-Lazy version? *

   Mark only one oval.

   - ◯ Strongly Agree
   - ◯ Agree
   - ◯ Neutral
   - ◯ Disagree
   - ◯ Strongly Disagree

5. Do you agree that the fixed Non-Lazy version will help to increase developers' productivity? *

Mark only one oval.

- ◯ Strongly Agree
- ◯ Agree
- ◯ Neutral
- ◯ Disagree
- ◯ Strongly Disagree

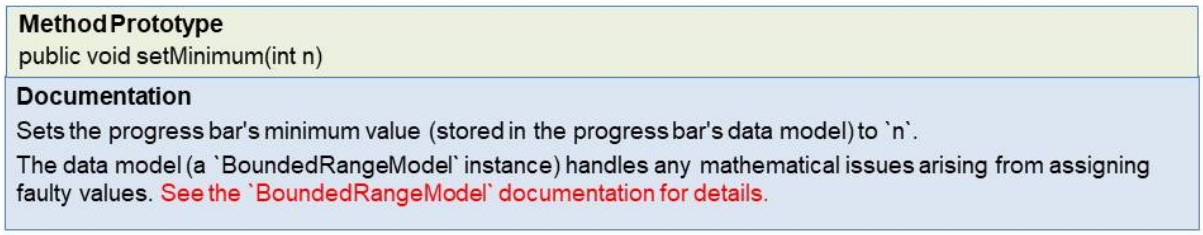
6. Please justify your selection and write any additional suggestion you have in couple of sentences. *

_____

2. Fragmented Documentation

Definition: Documentations where the information of documentation (related to an API element) is scattered over too many pages or sections.

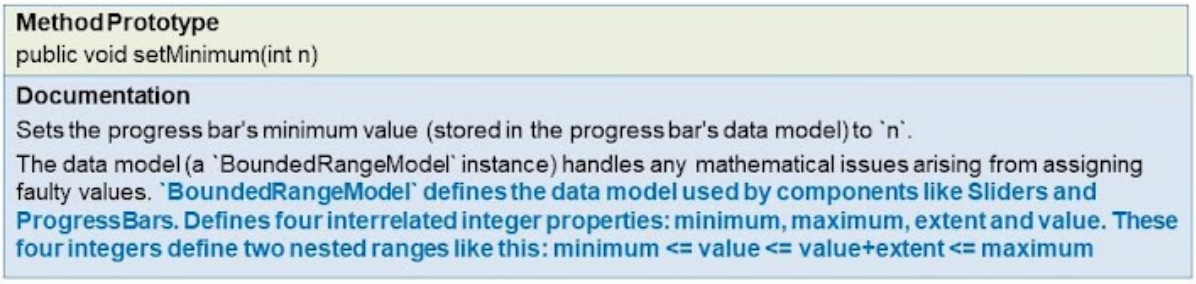
Let's have a look at a Fragmented Documentation. [Red marked part is suggesting potential Fragmented smell.]



Method Prototype
public void setMinimum(int n)

Documentation
Sets the progress bar's minimum value (stored in the progress bar's data model) to `n`.
The data model (a `BoundedRangeModel` instance) handles any mathematical issues arising from assigning faulty values. See the `BoundedRangeModel` documentation for details.

Now, look at the fixed Non-Fragmented version of the documentation. [Blue marked part is suggesting fix to the Fragmented smell.]



7. Do you agree with the fixed Non-Fragmented version? *

   Mark only one oval.

   - ◯ Strongly Agree
   - ◯ Agree
   - ◯ Neutral
   - ◯ Disagree
   - ◯ Strongly Disagree

8. Do you agree that the fixed Non-Fragmented version will help to increase developers' productivity? *

   Mark only one oval.

   - ◯ Strongly Agree
   - ◯ Agree
   - ◯ Neutral
   - ◯ Disagree
   - ◯ Strongly Disagree

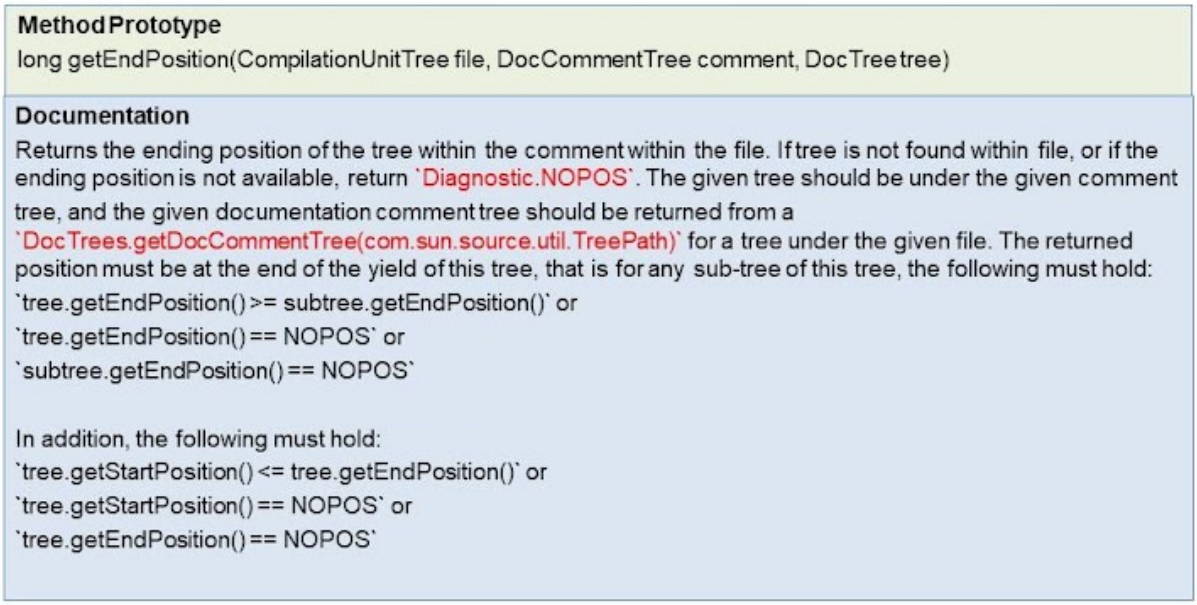
9. Please justify your selection and write any additional suggestion you have in couple of sentences. *

---

3. Excessive Structural Info

Definition: Documentations with description that contains too many structural syntax or structure. For example, the Javadoc of the java.lang.Object class Javadoc lists all the hundreds of subclasses of the class.

Let's have a look at a Documentation with Excessive Structural Info. [Red marked parts are suggesting potential Excessive Structural smell.]



Method Prototype

long getEndPosition(CompilationUnitTree file, DocCommentTree comment, DocTree tree)

Documentation

Returns the ending position of the tree within the comment within the file. If tree is not found within file, or if the ending position is not available, return `Diagnostic.NOPOS`. The given tree should be under the given comment tree, and the given documentation comment tree should be returned from a `DocTrees.getDocCommentTree(com.sun.source.util.TreePath)` for a tree under the given file. The returned position must be at the end of the yield of this tree, that is for any sub-tree of this tree, the following must hold:

`tree.getEndPosition() >= subtree.getEndPosition()` or

`tree.getEndPosition() == NOPOS` or

`subtree.getEndPosition() == NOPOS`

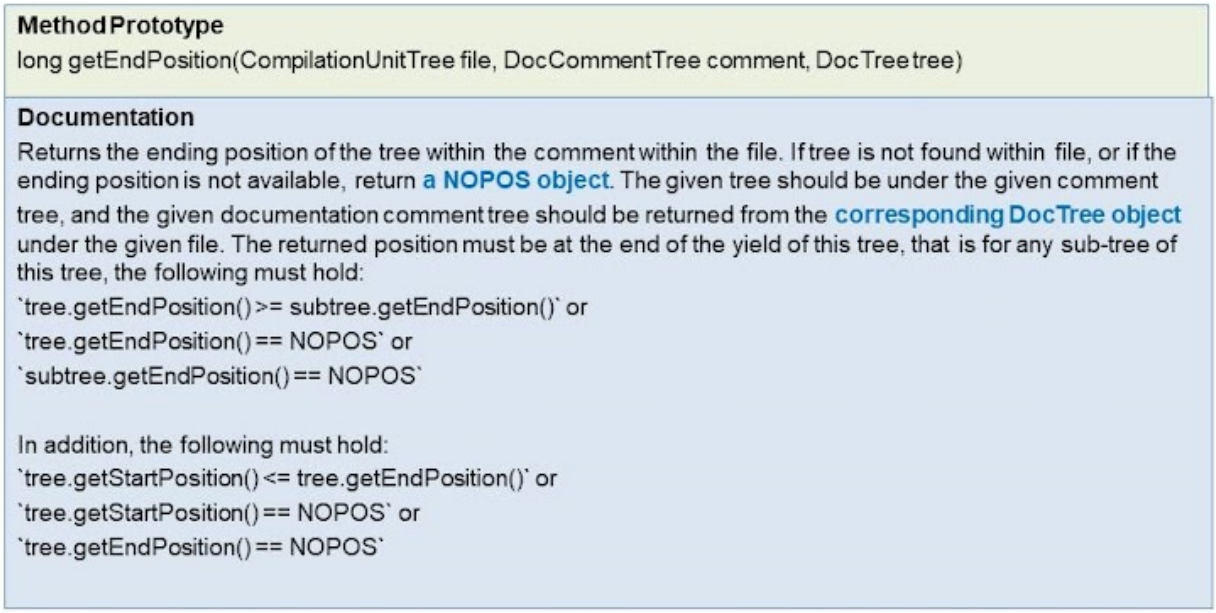
In addition, the following must hold:

`tree.getStartPosition() <= tree.getEndPosition()` or

`tree.getStartPosition() == NOPOS` or

`tree.getEndPosition() == NOPOS`

Now, look at the fixed Non-Excessive Structural version of the documentation. [Blue marked parts are suggesting fix to the Excessive Structural smell.]



Method Prototype

long getEndPosition(CompilationUnitTree file, DocCommentTree comment, DocTree tree)

Documentation

Returns the ending position of the tree within the comment within the file. If tree is not found within file, or if the ending position is not available, return a NOPOS object. The given tree should be under the given comment tree, and the given documentation comment tree should be returned from the corresponding DocTree object under the given file. The returned position must be at the end of the yield of this tree, that is for any sub-tree of this tree, the following must hold:

`tree.getEndPosition() >= subtree.getEndPosition()` or

`tree.getEndPosition() == NOPOS` or

`subtree.getEndPosition() == NOPOS`

In addition, the following must hold:

`tree.getStartPosition() <= tree.getEndPosition()` or

`tree.getStartPosition() == NOPOS` or

`tree.getEndPosition() == NOPOS`

10. Do you agree with the fixed Non-Excessive Structural version? *

Mark only one oval.

- ◯ Strongly Agree
- ◯ Agree
- ◯ Neutral
- ◯ Disagree
- ◯ Strongly Disagree

11. Do you agree that the fixed Non-Excessive Structural version will help to increase developers' productivity? *

Mark only one oval.

- ◯ Strongly Agree
- ◯ Agree
- ◯ Neutral
- ◯ Disagree
- ◯ Strongly Disagree

12. Please justify your selection and write any additional suggestion you have in couple of sentences. *

_____

4. Bloated Documentation

Definition: Documentations that are excessively lengthy and verbose. Hence, boring to read and understand.

Let's have a look at a Bloated Documentation. [Red marked parts are suggesting potential Bloated smell.]

Method Prototype

public float[] toCIEXYZ(float[] colorvalue)

Documentation

"Transforms a color value assumed to be in this ColorSpace into the CS_CIEXYZ conversion color space.

This method transforms color values using relative colorimetry, as defined by the ICC Specification. This means that the XYZ values returned by this method are represented relative to the D50 white point of the CS_CIEXYZ color space. This representation is useful in a two-step color conversion process in which colors are transformed from an input color space to CS_CIEXYZ and then to an output color space. This representation is not the same as the XYZ values that would be measured from the given color value by a colorimeter. A further transformation is necessary to compute the XYZ values that would be measured using current CIE recommended practices.

The ICC standard uses a device independent color space (DICS) as the mechanism for converting color from one device to another device. In this architecture, colors are converted from the source device's color space to the ICC DICS and then from the ICC DICS to the destination device's color space. The ICC standard defines device profiles which contain transforms which will convert between a device's color space and the ICC DICS. The overall conversion of colors from a source device to colors of a destination device is done by connecting the device-to-DICS transform of the profile for the source device to the DICS-to-device transform of the profile for the destination device. For this reason, the ICC DICS is commonly referred to as the profile connection space (PCS). The color space used in the methods toCIEXYZ and fromCIEXYZ is the CIEXYZ PCS defined by the ICC Specification. This is also the color space represented by ColorSpace.CS_CIEXYZ.

The XYZ values of a color are often represented as relative to some white point, so the actual meaning of the XYZ values cannot be known without knowing the white point of those values. This is known as relative colorimetry. The PCS uses a white point of D50, so the XYZ values of the PCS are relative to D50. For example, white in the PCS will have the XYZ values of D50, which is defined to be X=.9642, Y=1.000, and Z=0.8249. This white point is commonly used for graphic arts applications, but others are often used in other applications.

To quantify the color characteristics of a device such as a printer or monitor, measurements of XYZ values for particular device colors are typically made. For purposes of this discussion, the term device XYZ values is used to mean the XYZ values that would be measured from device colors using current CIE recommended practices.

Converting between device XYZ values and the PCS XYZ values returned by this method corresponds to converting between the device's color space, as represented by CIE colorimetric values, and the PCS. There are many factors involved in this process, some of which are quite subtle. The most important, however, is the adjustment made to account for differences between the device's white point and the white point of the PCS. There are many techniques for doing this and it is the subject of much current research and controversy.

Some commonly used methods are XYZ scaling, the von Kries transform, and the Bradford transform. The proper method to use depends upon each particular application.

The simplest method is XYZ scaling. In this method each device XYZ value is converted to a PCS XYZ value by multiplying it by the ratio of the PCS white

point (D50) to the device white point.

Xd, Yd, Zd are the device XYZ values
Xdw, Ydw, Zdw are the device XYZ white point values
Xp, Yp, Zp are the PCS XYZ values
Xd50, Yd50, Zd50 are the PCS XYZ white point values
Xp = Xd * (Xd50 / Xdw)
Yp = Yd * (Yd50 / Ydw)
Zp = Zd * (Zd50 / Zdw)

Conversion from the PCS to the device would be done by inverting these equations:

Xd = Xp * (Xdw / Xd50)
Yd = Yp * (Ydw / Yd50)
Zd = Zp * (Zdw / Zd50)

Note that the media white point tag in an ICC profile is not the same as the device white point. The media white point tag is expressed in PCS values and is used to represent the difference between the XYZ of device illuminant and the XYZ of the device media when measured under that illuminant. The device white point is expressed as the device XYZ values corresponding to white displayed on the device. For example, displaying the RGB color (1.0, 1.0, 1.0) on an sRGB device will result in a measured device XYZ value of D65. This will not be the same as the media white point tag XYZ value in the ICC profile for an sRGB device.

Now, look at the fixed non-Bloated version of that documentation.

Method Prototype

public float[] toCIEXYZ(float[] colorvalue)

Documentation

"Transforms a color value assumed to be in this ColorSpace into the CS_CIEXYZ conversion color space.

This method transforms color values using relative colorimetry, as defined by the ICC Specification. This means that the XYZ values returned by this method are represented relative to the D50 white point of the CS_CIEXYZ color space. This representation is useful in a two-step color conversion process in which colors are transformed from an input color space to CS_CIEXYZ and then to an output color space. This representation is not the same as the XYZ values that would be measured from the given color value by a colorimeter. A further transformation is necessary to compute the XYZ values that would be measured using current CIE recommended practices.

Converting between device XYZ values and the PCS XYZ values returned by this method corresponds to converting between the device's color space, as represented by CIE colorimetric values, and the PCS.

Some commonly used methods are XYZ scaling, the von Kries transform, and the Bradford transform. The proper method to use depends upon each particular application.

The simplest method is XYZ scaling. In this method each device XYZ value is converted to a PCS XYZ value by multiplying it by the ratio of the PCS white

point (D50) to the device white point.

Xd, Yd, Zd are the device XYZ values

Xdw, Ydw, Zdw are the device XYZ white point values

Xp, Yp, Zp are the PCS XYZ values

Xd50, Yd50, Zd50 are the PCS XYZ white point values

Xp = Xd * (Xd50 / Xdw)

Yp = Yd * (Yd50 / Ydw)

Zp = Zd * (Zd50 / Zdw)

Conversion from the PCS to the device would be done by inverting these equations:

Xd = Xp * (Xdw / Xd50)

Yd = Yp * (Ydw / Yd50)

Zd = Zp * (Zdw / Zd50)

13. Do you agree with the fixed non-Bloated version? *

Mark only one oval.

- ◯ Strongly Agree
- ◯ Agree
- ◯ Neutral
- ◯ Disagree
- ◯ Strongly Disagree

14. Do you agree that the fixed non-Bloated version will help to increase developers' productivity? *

Mark only one oval.

- ◯ Strongly Agree
- ◯ Agree
- ◯ Neutral
- ◯ Disagree
- ◯ Strongly Disagree

15. Please justify your selection and write any additional suggestion you have in couple of sentences. *

_____

5. Tangled Documentation

Definition: Documentations with a complex description that is difficult to understand.

Let's have a look at a Tangled Documentation.

Method Prototype

public static Integer getInteger(String nm, int val)

Documentation

Determines the integer value of the system property with the specified name.

The first argument is treated as the name of a system property. System properties are accessible through the `System.getProperty(java.lang.String)` method. The string value of this property is then interpreted as an integer value using the grammar supported by `decode` and an `Integer` object representing this value is returned.

The second argument is the default value. An `Integer` object that represents the value of the second argument is returned if there is no property of the specified name, if the property does not have the correct numeric format, or if the specified name is empty or `null`.

In other words, this method returns an `Integer` object equal to the value of:

> `getInteger(nm, new Integer(val))`

but in practice it may be implemented in a manner such as:

> > Integer result = getInteger(nm, null);

> return (result == null) ? new Integer(val) : result;

>

to avoid the unnecessary allocation of an `Integer` object when the default value is not needed.

Now, look at the fixed non-Tangled version of that documentation.

Method Prototype

public static Integer getInteger(String nm, int val)

Documentation

Determines the integer value of the system property with the specified name.

The first argument is treated as the name of a system property. System properties are accessible through the `System.getProperty(java.lang.String)` method. The string value of this property is then interpreted as an integer value using the grammar supported by `decode` and an `Integer` object representing this value is returned.

If there is no property of the specified name, if the property does not have the correct numeric format, or if the specified name is empty or `null`, an `Integer` object that represents

the value of the second argument is returned

In other words, this method returns an `Integer` object equal to the value of:

> `getInteger(nm, new Integer(val))`

16. Do you agree with the fixed non-Tangled version? *

Mark only one oval.

- ◯ Strongly Agree
- ◯ Agree
- ◯ Neutral
- ◯ Disagree
- ◯ Strongly Disagree

17. Do you agree that the fixed non-Tangled version will help to increase developers' productivity? *

Mark only one oval.

- ◯ Strongly Agree
- ◯ Agree
- ◯ Neutral
- ◯ Disagree
- ◯ Strongly Disagree

18. Please justify your selection and write any additional suggestion you have in couple of sentences. *

_____

Thank You

Thank you for taking the time to complete this survey. We truly value the information you have provided. Your responses will contribute to our study of documentation smells.