Method

Xerces-J was chosen as the source of these metrics. Only Java source code was analyzed. Weighted Methods per Class was calculated with by summing each class's methods' cyclomatic complexity.Unweighted Methods per Class are also represented in the data as MPC. Each aggregate measure was calculated using an average of the classes only, based on the output from the analytics tool.

Evaluation

Weighted Methods per Class

The trend for WMC is not clear: in the small sample size given, versions 2.8.0 and 1.4.4 suggest that the WMC is much higher than the other two versions. The unweighted trend is a bit clearer: only the most recent version analyzed represents a departure from the general trend of about 10.6 unweighted methods per class. This suggests that there was effort decreasing the cyclomatic complexity for version 2.5.0. Examining version 2.9.0, there was a 4% decrease in the unweighted metric (from 2.8.0) corresponding to about 6% reduction in the weighted metric. This means there was a small decrease in the average cyclomatic complexity.

Depth of inheritance tree

Remaining stable from 1.4.4 to 2.5.0, DIT dipped in 2.8.0 and regained about half of that decrease with 2.9.0. All values are clustered between 1.85 and just over 2, indicating that most classes have about 1 or 2 parent types. This indicates a mild downward trend in the per-class level of inheritance in the project.

Number of Children

NOC drops from over 0.45 from 1.4.4 by 22% to about 0.35 in 2.8.0 and decreases by about 2% to 0.01 in 2.9.0. This demonstrates a general trend downwards, with decreasing recent reductions. This could demonstrate an increase in the difficulty in implementing further reductions. Taken together with the downward trend of DIT, it seems the project has refactored classes which had common superclasses into a less deep but wider hierarchy. This is supported by the supplemental Number of Classes data, suggesting almost double the number of classes exist in version 2.9.0 as in 1.4.4.

Coupling Between Objects

CBO starts at about 6 in 1.4.4, rising by over a third reaching almost 8.5 in 2.8.0 before dropping to about 8 in 2.9.0. As with NOC, 2.5.0 appears to represent a transition between 1.4.4 and 2.5.0. Because this data largely mirrors NOC, it seems there may be a relationship. Taken with DOT and NOC, the data suggests that the flattening of the class hierarchy may have necessitated a larger inter-dependency of each new class.

Lack of Cohesion of Methods

LCOM rises from about 31.5% in version 1.4.4 by about 9% to nearly 34% in version 2.8.0, before dropping again by nearly the same amount to 2.9.0. This indicates that progress was made in decreasing the cohesion between 1.4.4 and 2.8.0, but that progress was lost in the 2.9.0 implementation. There is no immediately obvious explanation as to why cohesion would drop so precipitously.

Examples

Weighted Methods per Class

Consider the methods for `org.apache.xerces.jaxp.DocumentBuilderImpl`:

Method	Weight	Cyclomatic Complexity
`DocumentBuilderImpl` (constructor)	1	2
`getDOMImplementation`	1	1
`getDOMParser`	1	1
`isNamespaceAware`	1	1
`isValidating`	1	1
`newDocument`	1	1
`parse`	1	4
`setDocumentBuilderFactoryAttributes`	1	4
`setEntityResolver`	1	1
`setErrorHandler`	1	2

Summing the cyclomatic complexity gives the WMC for that class of 18.

Depth of inheritance tree

Consider the class `org.apache.xerces.jaxp.DocumentBuilderImpl`. It inherits from `javax.xml.parsers.DocumentBuilder` which then inherits from `java.lang.Object`, giving the class's DIT of 3.

Number of Children

- Consider the class `org.apache.xerces.jaxp.DocumentBuilderImpl`. Since there are no classes which inherit fram this one, its NOC is 0.
- Consider the class `org.apache.xerces.parsers.DOMParser`. The only class which extends this one is the sample/demonstration class `dom.DOMAddLines`, so its NOC is 1.

Coupling Between Objects

Consider the class `org.apache.xerces.jaxp.DocumentBuilderImpl`. Its implementation requires referencing the following classes, which total 20, giving the CBO number.

```
java.util.Hashtable
java.util.Enumeration
java.io.IOException
javax.xml.parsers.DocumentBuilder
javax.xml.parsers.DocumentBuilderFactory
javax.xml.parsers.ParserConfigurationException
org.w3c.dom.Document
org.w3c.dom.DOMImplementation
org.w3c.dom.DocumentType
org.xml.sax.XMLReader
org.xml.sax.InputSource
org.xml.sax.SAXException
org.xml.sax.SAXParseException
org.xml.sax.SAXNotRecognizedException
org.xml.sax.SAXNotSupportedException
org.xml.sax.EntityResolver
org.xml.sax.ErrorHandler
org.xml.sax.helpers.DefaultHandler
org.apache.xerces.parsers.DOMParser
org.apache.xerces.dom.DOMImplementationImpl
```

Lack of Cohesion of Methods

Consider the class `org.apache.xerces.jaxp.DocumentBuilderImpl`. In the following table, each method has an X for each variable which is used.

Method	**V1**	**V2**	**V3**	**V4**	**V5**
`DocumentBuilderImpl` (constructor)	-	-	X	X	X
`getDOMImplementation`	-	-	-	-	-
`getDOMParser`	-	-	X	-	-
`isNamespaceAware`	-	-	-	X	-
`isValidating`	-	-	-	-	X
`newDocument`	-	-	-	-	-
`parse`	-	X	X	-	-
`setDocumentBuilderFactoryAttributes`	-	-	X	-	-
`setEntityResolver`	X	-	-	-	-
`setErrorHandler`	-	X	-	-	-
Count	1	2	4	2	2

Yielding:

Instance variable count: 5
Method count: 10
Instance variable use count: 11
Giving the following LCOM:

$$\frac{(1/5*11-10)}{(1-10)} = 86.7\%$$

Recommendations

Ovarall, Weighted (and unweighted) class method counts, DIT are all trending in positive directions, while LCOM, NOC, and CBO suggest additional investigation should be given to aspects of cohesion.

Summary

The project has undergone an explosion in the quantity of classes. At the same time, each class holds fewer classes in more recent versions. This suggests the software has moved non-coherent methods out of classes, which is somewhat confirmed by the LCOM trends. It is perhaps the case that this was done to allow dependency injection, allowing for classes holding fewer methods to be referenced instead of local methods.

In particular, the project seems to have difficulty maintaining cohesion, seen as a factor in both WMC and LCOM metrics. This suggests developers ought to be vigilant in ensuring functionality is being placed in the right classes, but may also represent a system with a high proportion of DTO-style objects with few responsibilities. This possiblity is also reflected in the class count.

The project may be struggling with coupling, also a possible secondary effect to the trend in class count.