OO and Usability Metrics

1 What do we want to measure?

A key goal is one of assessing attributes that relate to such concepts as ”separation of concerns” and ”information hiding” such as

- The interactions between elements
- The way that elements are grouped

Complications added by the OO paradigm include:

- Inheritance
- Polymorphism
- The class-instance distinction

2 Chidamber and Kemerer’s metrics

- The six C&K metrics are the most widely used OO metrics. Defined by employing a model related to the major features of the ”object model” as well as to established concepts such as coupling and cohesion
- The concepts provide a set of indirect measures used to assess different object attributes. The C&K metrics then provide direct measures that are meant to act as surrogates for the concepts

Definition: Surrogate

Something we can measure that we believe relates to the property of interest

2.1 What are they used for?

- To identify the classes that are most likely to contain faults
- Identify where changes may have increased the likelihood of errors occuring

2.2 WMC Weighted methods/class

Formula for this is

$$WMC = \sum_{i=1}^{n} c_i$$

where c_i is the complexity of method i

- Main rationale for this metric is that methods are properties of objects, and so the complexity of the whole is a function of the set of individual properties
- C&K suggest that the number of methods and their combined complexity reflects the effort required to develop and maintain the object + possible impact on children
- Weights are measures that are considered to relate to the static complexity of each method by using such attributes as length, and metrics such as cyclomatic complexity
- If all weights are set to 1, this reduces to a count of methods

Usefulness of WMC:

- For weights a key issue is the need to devise some way of assigning meaningful values to these that can be extracted from the design/code
- Commonly used are V(G), LOC or simply a value of 1
- An increase in WMC is a reasonably good indicator of the likelihood of there being an increase in defects for that class

2.3 DIT: Depth of inheritance tree

DIT is basically a count of tree height from a node to the root of a tree:

- A measure that identifies how many ancestor classes can potentially affect a given class
- Deeper trees implicitly constitute greater design complexity since they require an understanding of more superclasses
- Wider trees are more loosely coupled, but may also indicate that the commonality between classes is not exploited well
- DIT offers no significant predictive ability for fault proneness