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 super-classes
- 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

2.4 NOC: number of children

NOC is a count of the number of immediate subclasses of a class that exists in the class hierarchy:

- Concerned with the scope of properties and the number of subclasses that will inherit the methods of the parent class
- The number of children gives an idea of the potential influence that a class has on the design, and hence of the amount of testing of methods required
- A large number may indicate a poor abstraction in the parent class, or that is is used in a variety of settings

This has some ability to discriminate with regard to effects of changes

2.5 CBO: coupling between objects

This is the count of the number of couples that exist between a class and other classes

- Coupling is assumed to exist if one class uses methods or instance variables of another
- Excessive coupling can be considered as being detrimental to modular design and likely to prevent reuse
- May require consideration of the form and strength of coupling involved

This has moderate predictive ability with regard to change proneness

2.6 RFC: Response for a class

RFC provides a measure that seeks to recognise the influence of the ”immediate surroundings of as class”

- The response set of a class is the set of methods accessed by the set of methods belonging to an object of that class
- The more methods can be invoked from an object, the greater its complexity in terms of the effort needed to comprehend its operation and the level of understanding required to test it

Research suggests some correlation with the likely number of defects/fault proneness

2.7 LCOM: Lack of cohesion in methods

LCOM is based upon the concept of cohesion. The original definition (LCOM1) was a count of the number of methods that do not share attributes

- If some methods use one subset of state variables, while others use another subset, then there is a lack of cohesion and maybe the class should be split into two
- LCOM is then a measure of the number of disjoint sets of methods of a class based upon this use of state variables

The value for LCOM should be 0 where a class is fully cohesive

LCOM2 was later defined as:

$$\text{Number of pairs with no common attributes - number of pairs with common attributes}$$

3 Pattern decay and grime

Definition: Grime

The build up of unrelated artefacts in the classes that play roles in the implementation of a design pattern

Metrics used to assess grime can also be used to help assess the related issue of technical debt

4 Cognitive dimensions

Dimension	Interpretation
Abstraction	Types and availability of abstraction mechanisms
Hidden dependencies	Important links between entities are not visible
Premature commitment	Constraints on the order of doing things
Secondary notation	Extra information provided in means other than formal syntax
Viscosity	Resistance to change
Visibility	Ability to view components easily
Closeness of mapping	Closeness of representation to domain
Consistency	Similar semantics expressed in similar syntactic forms
Diffuseness	Verbosity of language
Error proneness	Notation invites mistakes
Hard mental operations	High demand on cognitive resources
Progressive evaluation	Work to date can be checked at any time
Provisionality	Degree of commitment to actions or marks
Role-expressiveness	The purpose of a component is readily inferred

4.1 Using cognitive dimensions

Essentially this is a reflective set of metrics. In using them to assess design notations, there are limitations when using them for evaluation, including:

- Vague definitions
- CDs define properties, not if they are bad or good
- A lack of clear evaluation procedures

However CDs do provide a framework that can be used for analysing and describing visual structures. And they remain a valuable tool in the absence of more developed models and metrics

5 Design notations

The following are issues in diagrammatic notations used for software design such as UML

- Symbol redundancy: where there are alternative forms that can be used
- Visual distance: allowing easy discrimination between the symbols used
- Primacy of shape: shape is a good discriminator for symbols.
- Textual differentiation: used to differentiate excessive graphic complexity