Recommended Thresholds

- LOC - <100 Lines of Code is recommended for a module, because the larger the number of lines the harder it will be to understand/follow and maintain the code.
- Executable Statements - recommended <50 just as with LOC the larger the number of statements the more complex the code becomes to understand and maintain.
- Comment Percentage - 20 to 30% of the code is recommended, this helps make the code understandable. This percentage however may be misleading. The code should be inspected to see if the comments are meaningful comments. Comments such as "this works but I don't know how" are not useful. Also check to make sure the comments are not just old code that is commented out.
- Complexity (v(G)) - Complexity for a module should be less then 10. Between 10 and 20 is medium risk. If the complexity (greater than 20) becomes too high then the code no longer is understandable.
- Number of methods (CSO) - ≤ 20 preferred, ≤ 40 acceptable per class. The counting tool included explicit constructors and destructors in the method counts, so these thresholds are inflated. Taking that into account, the recommended number of actual implemented methods translates to under 10 per class.
- Weighted Methods per Class (WMC) - ≤ 100 acceptable. The number of methods and the complexity of those methods are a predictor of how much time and effort is required to develop and maintain the class. While the NOM may be inflated by the beneficial use of constructors, WMC provides a better idea of the true total complexity of a class.
- Response for Class (RFC) - ≤ 100. We have seen very few classes with RFC over 50. If the RFC is high, this means the complexity is increased and the understandability is decreased. The larger the number of methods that can be invoked from a class through messages, the greater the complexity of the class, complicating testing and debugging. Making changes to a class with a high RFC will be very difficult due to the potential for a ripple effect.
- RFC/CSO ≤ 5 for C++, ≤. 10 for Java. This adjusted RFC metric does a good job of sifting out classes that need extensive testing, according to developer feedback. The Java language enforces the use of classes for everything, which automatically drives up the value of this metric.
- Coupling Between Objects (CBO) - ≤ 5. A high CBO indicates classes that may be difficult to understand, reuse or maintain. The larger the CBO, the higher the sensitivity to changes in other parts of the design and therefore maintenance is more difficult. Low coupling makes the class easier to understand, less prone to errors spawning, promotes encapsulation and improves modularity.
- Depth in Tree > 5 means that the metrics for a class probably understate its complexity. DIT of 0 indicates a "root"; the higher the percentage of DIT's of 2 and 3 indicate a higher degree of reuse. A majority of shallow trees (DIT's < 2) may represent poor exploitation of the advantages of OO design and inheritance. On the other hand, an abundance of deep inheritance (DIT's > 5) could be overkill, taking great advantage of inheritance but paying the price in complexity.