**Title**: Metrics for Measuring the Quality of Object-Oriented Software

**Link**: <http://delivery.acm.org.libproxy.auc.ca/10.1145/2510000/2507311/p66b-singh.pdf?ip=199.212.55.169&id=2507311&acc=ACTIVE%20SERVICE&key=FD0067F557510FFB%2E2E114FAB5F912086%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=939957675&CFTOKEN=22411919&__acm__=1495551123_66186ebadebfcf7a06c0c6f881edc6c7>

**Summary:**

The paper presents five parameters for measuring software of object-oriented programming, and some associated metrics for each one. The article takes the approach that “anything which can’t be measured can’t be controlled”, and in part shows that changes to a piece of software overtimes can reduce the quality. It draws the conclusion that software becomes more complex over a series of releases, and quality therefore decreases over time.The following metrics and parameters were used:

* LOC (lines of code): represents the size of the software, and states that the greater the size the less understandable it will be (measure of understandability)
* CC (cyclomatic complexity): measure the number of linearly independent paths through a program, and low values are considered better (measure of complexity)
* CBO (coupling between objects): metric for a class is the count of those classes which the given class is coupled, and higher values are seen as bad (measure of efficiency reuse, and complexity)
* LCOM (lack of cohesion): the degree to which methods within a class are related to one another, and maximized cohesion is seen as a positive (reuse, efficiency, complexity)
* WMC (weighted methods per class): indicator of how much time and effort is require to develop/maintain a class, and low is seen as greater polymorphism (measure of understandability, complexity, and reuse)
* RFC (response for a class): measure the number of methods that are invoked in response to a message received by an object of the class, and shows how much communication there is between classes which can complicate debugging/testing (measure of understanding and complexity)
* MI (maintainability index): measures the ease with which we can maintain software or how easily we can make changes to the software; and high is considered as better (measure of maintainability)
* NOC (number of children): measures the number of direct descendants for each class, and classes with large number of children are considered to be more difficult to modify and usually require more testing (measure of efficiency and reusability)
* DIT (depth of inheritance tree): the maximum length from the class node to the root of the tree, and is measured by the number of ancestor classes (measure of reusability and complexity)

**Limitations**: It doesn’t show a benchmark for “good quality” software, many of the results seems generalized, it doesn’t clearly explain how larger corresponds to lower quality, and doesn’t actually show how it actually tested the metrics. It also performs it’s testing on different versions of the same software which seems strange to me as the end product is what ultimately matters.

**Note:** All the metrics used showed a loss in software quality over the history of versions, and it is especially important that their testing utilized an OSS called JFreeChart. A metric like MI could be especially useful since it’s a combination of a couple metrics.

**Important Note**: An article like this should be a main focus of interest because it’s lacking a lot of things that we could consider. It doesn’t consider the community involvement in OSS or security measurements, and actually limits itself in that it’s only performed on a single piece of software. We could actually mention what we utilize for testing, and improve upon their flawed conclusion.

**Title**: Comparing Software Metric Tools

**Link**: <http://dl.acm.org.libproxy.auc.ca/citation.cfm?id=1390648&CFID=942414749&CFTOKEN=24620810>

**Summary**:

The article expresses the fact that different metric tools interpret and implement the definitions of software differently. Basically, what this means is that the tools could have heavy variation even though they are showing the exact same metrics. The importance of this variation is quite relevant to scientific study where we wish to get as accurate of results as possible. The primary purpose of the article is to answer two questions:

1. Do metric values of given software systems and metric definitions depend on the metric tools used to compute them (confirmed in the results)?
2. Does the interpretation of metrics values of given software system as induced by a quality mode depend on the metrics tool (partially confirmed).

The article goes on to consider some limitations in their study which involves lack of universality among tools, the actual cost/availability of tools, the fact that not all tools measure the same thing, and that there’s no “gold standard” for valid metrics. For the most part they adhere to their self-imposed limitations and their results are quite clear in that there is heavy variation. The article states that much of this comes down to scope and what the tools consider in regards to the extent of the individual metrics. As an example, a few of the tools consider built in API objects and comments for things like LOC and inheritance tree. The following are the metrics that were tested on three different pieces of software:

* CBO (coupling between object classes)
* DIT (depth of inheritance tree)
* LCOM-CK (lack of cohesion of methods)
* LCOM-HS (lack of cohesion of methods)
* LOC (lines of code)
* NOC (number of children)
* NOM (number of methods)
* RFC (response for a class)
* WMC (weighted methods per class)

It should also be noted that ten metric tools were tested out as this presented the largest subset of available tested metrics.

**Limitation**: The article doesn’t really answer whether a metric itself is useful/relevant, and focusses on OOD. It also has the issue of scope which was mentioned earlier in the summary.

**Note**: It points out important notes that should be considered when doing our testing, like the fact that we can’t rely on using multiple tools. It doesn’t say that the fact that they different results is a bad thing, but more that these results can’t be relied on if we’re using multiple tools. It should also be noted that they had much higher variations on larger software.

**Important Note**: The article refers to the ISO-9216 quality model and states that to adhere to the model there should be at least one metric relating to couple, cohesion, size, and inheritance. Tests should clearly be run multiple times are variation in results are potential.

**Title**: Using Pirate Metrics to measure success of open source projects

**Link**: <https://opensource.com/business/16/6/pirate-metrics>

**Summary**:

The article discusses the use of “Pirate Metrics” (redeveloped from another model) to measure community involvement and retention in open source software projects. It mentions that metrics for community involvement are especially useful for smaller projects to ascertain if it’s heading down a good trajectory. These metrics measure the following:

* Acquisition: A user visits the website or repository
* Activation: A user downloads the product
* Retention: A user opens up an issue in the community
* Referral: A user stars the repository on GitHub (useful user feedback)
* Revenue: A user regularly opens issues or contributes to code

**Limitation**: It may be difficult to measure some of these specific metrics on our end. This idea is advertised as more of a self-reflection framework whereby a developer may analyze their own work. It’s also designed from a business framework, and the question of actual

**Note**: It has some interesting notes regarding community, which seems to be a common metric for addressing open source quality (indirect quality). The consensus seems that the more activations you have, the more the implied open source product can be viewed as having “good quality”.

**Title:** The size of open-source communities and its impact upon activity, licensing, and hosting

**Link**: <http://redmonk.com/dberkholz/2013/04/22/the-size-of-open-source-communities-and-its-impact-upon-activity-licensing-and-hosting/>

**Summary**:

The post makes an interesting discussion on the fact that once a project reaches a certain number of contributors, it appears to behave differently than a smaller project. An interesting point is made that the vast majority of active projects have few actual committers, and actual trend shows a linear decrease when compared to projects over number of committers. Basically, projects with a high number of community members are in the vast minority when compared to projects with few. The author uses a comparative chart/spreadsheet to show that committers tend to be more active, as a whole, in larger projects. These larger projects also tend to become more self-hosted, and pay attention to things like licensing. The conclusion is that a larger project (based on community) behaves much more actively and progressively than one which is small by comparison.

**Limitation**: While this is an excellent way of explaining why a larger community can be better, it doesn’t necessarily “prove” that more committers equals a better product. He also utilizes an external dataset from Ohloh which he himself admits has limitations. It makes a case for licensing, which isn’t a proven enough discussion point in correlation to quality.

**Note**: An interesting trend is shown here in that the vast majority of projects have a smaller connection of actual committers. This is an interesting fact as it makes logical sense, especially in that the number of lower quality projects should be in the majority. Another idea to consider is that since large community based projects have more involvement, error resolution should theoretically be improved and therefore the actual quality should be greater (needs validation).

**Important Note**: The article makes a clear case why larger projects are generally more successful, or at the very least perform different. This definitely makes a case for a community based software metric, and also why it’s important to consider. It also shows that GitHub is still an excellent choice for choosing projects, as it has gotten over some of the inherent issues of other sites like Sourceforge (for large projects).

**Title**: Open Source Metrics

**Link**: <https://opensource.guide/metrics/>

**Summary**:

The article makes a case for measuring certain metrics when creating open source projects. Utilizing metrics helps creators in responding to user feedback, see project popularity, understand how the project is used, etc. The primary reference the author uses is based upon the metric graphs that appear on GitHub projects. They divide the metrics into the following categories:

* Discovery: is the project being seen by people
* Usage: is the project being downloaded/cloned
* Retention: are people regularly committing to the project

**Limitation**: These metrics are quite useful for checking community involvement, however the examples they use are limited to GitHub (they provide minor alternatives). Similar to previous community articles, this is more directed towards actual creators and how they can better their own projects (possible limitation on research application).

**Note**: The final metrics in the “Retention” section have some points on issue turnover rate (how often issues are resolved), which in itself may be another useful metric.

**Important Note**: Community involvement, as later articles point out, is a clear indication of success. We should consider some type of metric here when performing our tests (downloads correspond to quality). GitHub stars may also be useful for our testing grounds as they don’t measure quality, but can allow us to get an initial assumption for picking a variation of OSS.

**Title:** QA Metrics: The Value of Testing Metrics Within Software Development

**Link**: <https://www.getzephyr.com/resources/whitepapers/qa-metrics-value-testing-metrics-within-software-development>

**Summary**:

The article makes the point that as a result of shorter deadlines and lower resources, metrics become all the more important. That being said it also explains that metrics are only useful when taken at the whole (only measure certain aspects). The article further states that while there are no universal metrics for all products, there are some important ones which can be divided as followed:

* Project: for the purpose of analysing whether a project is meeting an objective (general)
  + Requirement Coverage: percentage of number of requirements covered divided by the total number of requirements
  + Defect Open and Close Rate: percentage of number of defects found before delivery divided by total defects (defects found before and after)
* Department: provide generalized day-to-day analysis metrics
  + MTTD: number of issues detected over total execution time
  + MTTR: number of issues fixed over total coding time
  + Defect Removal Efficiency: percentage of number of pre-release defects divided by number of total defects
* Company: analyzes delivery to consumer
  + Customer Reported Issue Percentage: percentage of total number of defects found by customer divided by total number of product defects

**Limitation**: Getting some of the data from a few of these metrics may be difficult, or even impossible. Some the metrics imply knowledge of day-to-day activities which may limit the result.

**Note**: I refrained from mentioning some of the metrics (especially those in the “Company” section) since they either apply to multiple projects together or are impossible to test without restricted data.

**Title**: Effects of the Number of Developers on Code Quality in Open Source Software

**Link**: http://dl.acm.org.libproxy.auc.ca/citation.cfm?id=1852864&CFID=942414749&CFTOKEN=24620810

**Summary**:

The article uses a number of metrics to test whether the quality is effected by the number of developers. They state the motivation of this study is driven by the fact that many people suggest quality is directly affected by more or less developers. Their results seem to show that for their given test subjects, there is no correlation. They use the following metrics for testing:

* Cyclomatic Complexity
* Lines of Code per Function
* Comment Density
* Maximum Nesting

**Limitation**: The article doesn’t actually show how testing was performed, and doesn’t reveal their findings. There’s also a limitation to their premise that the number of developers doesn’t affect quality, as their scope seems too small to justify.

**Note**: This is a useful work for contradictory purposes, and also shows that some of the metrics discussed earlier are commonly used.

**Important Note**: This is one of the first articles to discuss comment density, which it states should be about 30%. This is a metric that should be considered for static quality, and maintainability.