1. **Introduction**
2. **Topics**
   1. **Open Source Software**

Compared too many other design phenomenon; Open Source Software (OSS) can be seen as relatively recent or new trend. That’s not to say collaboration between designers hasn’t always been there, and it’s common knowledge that an idea like “free sharing” has existed far longer than most modern technologies. That being said, the actual label of “open source” was first applied at a strategy session in Palo Alto, shortly after the announcement for the release of the Netscape source code [0.1]. The session was inspired by the idea that this release had created an opportunity to discuss and advocate a design process that as of that point wasn’t being openly discussed. As a result of this announcement, many groups began to adopt this new methodology and a community began to build.

In general OSS is software where the source code is made openly available to the public [0.2]. As such anyone can inspect, modify, or even further enhance the existing code. This isn’t necessarily limited to simple inspection, and can involve greater development and cooperation among designers. Basically OSS can considered any software whereby a user can download, change, and then redistribute the software back to the prime repository. It doesn’t generally concern itself with concepts like licensing as it is believed this allows for greater independence between software vendors [6]. The concept isn’t necessarily universal, but is more often the commonplace than the exception. This in itself makes it very different from “classical” software, which not only restricts its source code, but also holds tight licensing among its products. The idea mentioned here can be further extended to the point that OSS clearly has a very different development process than traditional software (example: Waterfall Design Model).

The process itself is relatively simple, but becomes more sophisticated as the project increases in size [11]. Generally, an individual or group will have an idea, and will begin basic development on it. They will then put the source code online, and allow individuals to edit and make changes to the already existing code. In some cases an entire community is created upon this, and the design process begins to refine itself. Error reports or new feature requests will be posted, and contributors will make alterations to fill these requests. The rise of version control systems (example: GitHub) have greatly simplified this process, and have allowed OSS to flourish. One can now find a large variety of projects among OSS, which can include:

* Operating Systems: UNIX/POSIX
* Web servers: Apache and JAWS
* Compilers: GNU C/C++
* Editors: GNU emacs

**2.2 Quality of Software**

Quality can generally be seen as the standard of something determined by the measure against something similar. In the case of software, it’s the measure of a particular software design compared to other types or minimum accepted values. These measurements and comparisons vary greatly, and as of yet there is currently no universal standard [1]. There are also debates on when software quality analysis should take place, as it’s beneficial for different users at different times. The entire process should be considered for designers, while for end-users the result is really the only concern. For the latter, the average user only considers whether the product performs the tasks they require.

In terms of quality assessment, there are two commonly used practices for analysis: models and metrics. Models tend to be based on the hierarchal design structure, and usually take inspiration from the ISO9126 design standards [5]. In this case models tend to be more useful for designers as they analyze quality from a multitude of perspectives throughout the entire creation process. If based of the ISO9126 then they tend to consider:

* Functionality
* Reliability
* Usability
* Maintainability
* Portability
* Efficiency

Clearly many aspects play off of each other, and a precise picture is painted. Metrics on the other hand are specific measurements for analysis, and can be used by anybody [12]. It’s important to note that metrics are usually part of the greater process for models, but can also be used as separate calculations. To be clear on the separation; models don’t always lead to direct measurements, whereas metrics are themselves direct measurements.

Now as implied earlier quality is not a fixed and universal property of software. It depends greatly on the context and goals of its stakeholders, and generally one needs to be clear and precise with the quality specification. Overtime, quality of software changes and if the software does not adjust accordingly, its quality will decay [0.3, I...II]. The only real solution to such a problem is continuous quality control, and the use of models or metrics are generally a way to combat this problem. The advantage of OSS becomes clear here, as it utilizes continuous releases and designers from many different backgrounds to resolve errors.

**2.3 Metrics for Measuring Open Source Software Quality**

Software metrics are measurements which act as a scale for representing certain aspects of the software [1]. As mentioned previously, they are direct measurements and are useful for determining whether a specific design corresponds to particular characteristics. Metrics aren’t limited to closed software either, and can be applied to OSS. The reason is that static code analysis doesn’t change for OSS source code, and the resulting software isn’t ultimately any different [2]. In laymen’s terms, the product is the basically the same as any other piece of software from a physical perspective.

Utilizing metrics has many advantages, and can help designers in better responding to feedback [4]. In the case of fast operating environments, with shorter deadlines and less resources (like most OSS projects) metrics become all the more important. The reason for such a case is that metrics can analyze many aspects of a project, and help designers determine where they are faulting [3]. Metrics are mainly used from a static code point, but also can analyze errors and even documentation. While this is incredibly useful, metrics also have a setback in that most don’t have a “gold standard” for acceptable values [12]. There are benchmarks, however most haven’t been widely accepted and it’s recommended to test only those which are widely documented.

Another issue is that many research papers don’t seem to agree on when analysis should occur (example: the design process or final product). Models make the case for the doing it throughout the process, and this will be touched on in the next section. A point is made in a paper on the SQO-OSS quality model on the difficult of properly analyzing functionality, as one cannot know the prime directives of the creators [7]. This further limits certain metrics, as they become only useful to designers and not end-users themselves. One clear consensus exists and that is towards the analysis of source code. For OSS, the justification is that source code it is always available, therefore it should be analyzed [6]. The ISO9216 quality standard is often used as a complete reference for the looking at the quality of software. However, it’s also been stressed that much of is redundant and repetitive. To be safe it is suggested in one article to at the very least utilize at least one metric for coupling, cohesion, size, and inheritance to be safe [12].

OSS itself allows for software quality to be examined transparently and without many barriers [2]. The problem here is that while static code analysis remains the same, there are issues analyzing the big picture. Basically, there aren’t commonly accepted metrics that consider enough for it to be fully useful for OSS. A major example is community which will be discussed in its own section later on. Some metrics which are greatly documented and commonly used are:

A complete listing of metrics can be found at an ISO9126 site [0.4].

**2.4 Models for Measuring Open Source Software Quality**

**-**can take advice from models for metric and community use

**2.5 Community Involvement**

**2.6 Quality Testing Methods and Tools**

1. **Conclusion and Research Proposal**
   1. **Conclusion**
   2. **Research Proposal**

# There are a lot of researches on the quality of traditionally developed software or Closed Source Software (CSS). OSS are said to be possibly of comparable quality with CSS. However, there are very few studies investigating common software quality factors for various types of Open Source projects [12]. Open Source Projects have their own characteristics which might affect their quality:

a) Different levels of programmers. Open Source Project is joined by different levels of people who are interested in the open source project. In Closed Source Project, programmers are well organized in one organization.

b) Frequent beta releases. A cornerstone of open-source is short feedback loops between users and core developers, which typically result in frequent “beta” releases, e.g., several times a month. Although this schedule satisfies the end-users who receive quick patches for bugs they found in earlier betas, it can be frustrating to other end-users who want more stable, less frequent software releases.

c) Platforms-independence. Another cornerstone of open-source software is its platform-independence, which stems from its roots in the open systems—rather than proprietary systems—community. Support for platform-independence, however, can yield the task of keeping an open-source source software base operational despite continuous changes to the underlying operating system and compiler platforms.

d) Support for many compile-time and run-time configurations. The availability of the software in open-source projects encourages core developers to increase the number of options for configuring and subsetting the software at compile-time and run-time. Although this flexibility enhances the software’s applicability for a broad range of use-cases, it can also greatly magnify QA costs due to the combinatorial number of code paths that must be regression tested.

As the OSS paradigm makes progress within these organizations any potential software procurer is tasked with some important questions which, currently, cannot be answered with any real assurance:

* Many OSS projects are very similar. How do we choose between them? Which is the most appropriate system for the company’s IT infrastructure?
* How can we distinguish the “good” and “bad” projects?
* How can we reason about the quality of a software product in order to trust its future development?

Unfortunately these organizations often have nothing more than word-of-mouth on which to base their judgments of OSS products. With 1097071 projects currently hosted on SourceForge it is understandable that products of excellent quality may be overlooked. It is possible to supplement the word-of-mouth tradition with more formal metrics to evaluate the OSS.

**Objective(s) or Activities**

### The objectives of this project include:

# Investigate suitable metrics to measure important software quality factors and choose those suitable for OSS

# Select a range of representative Open Source projects for measurement

# Evaluate the software of these projects according to the selected quality factors

# Using statistic tools to analyse and present the result

**References**