# OpenSSL 3.0.0: An exploratory case study

James Walden
waldenj1@nku.edu
Northern Kentucky University
Highland Heights, Ohio, USA

## ABSTRACT

*Context:* The OpenSSL project released version 3.0.0 in September 2021. This release was a departure from previous versions of OpenSSL in several ways, including a new versioning system and the first use of public software design documents.

*Objective:* The goal is to compare code quality of version 3.0.0 with the previous major release using the GrimoireLab toolset.

*Method:* We developed a new backend for Graal, a component of GrimoireLab, to use the `cqmetrics` C code metrics tool. We also modified Graal to add the capability to perform monthly samples of a project. We collected monthly snapshots of the two branches of OpenSSL and computed code metrics for each snapshot.

*Results:* While the code base grew substantially in each version, code complexity and use of problematic language features both decreased. The only negative indicator of code quality was an increase in style inconsistency.

## CCS CONCEPTS

• **Software and its engineering** → *Designing software*; *Software design engineering*; *Software reliability*.

## KEYWORDS

mining software repositories, software metrics

**ACM Reference Format:**
James Walden. 2022. OpenSSL 3.0.0: An exploratory case study. In *19th International Conference on Mining Software Repositories (MSR '22), May 23–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3524842.3528035

## 1 INTRODUCTION

The OpenSSL library is a key component of Internet infrastructure. While the project was neglected by its many users prior to the discovery of the Heartbleed vulnerability [5], support from the Linux Foundation's Core Infrastructure Initiative allowed rapid improvement in code quality and security after 2014 with the release of versions 1.0.2 and 1.1.0 [12].

OpenSSL 3.0.0, released on September 7, 2021, is the first release to use OpenSSL's new versioning system [1], in which the first digit indicates a major release, the second digit a minor release, and the third digit a patch to fix a bug or vulnerability. Prior OpenSSL versions used all three digits to indicate the major and minor release. Patches were indicated by a letter following the version number. The major release immediately prior to 3.0.0 was 1.1.1, which added support for TLS 1.3 and several new cryptographic algorithms like SHA3. Version numbers beginning with 2.0 had been previously used for OpenSSL's FIPS (Federal Information Processing Standards) 140 module, so they were skipped in the new versioning scheme.

The goal of this case study is to understand how development of OpenSSL 3.0.0 differed from development of the previous major release, 1.1.1. Version 3.0.0 was the first release of OpenSSL built with public design documents [9]. The release announcement described two additional improvements, including a 94% increase in documentation and a 54% increase in the amount of test code [2].

OpenSSL 3.0.0 introduced providers, a new abstraction for collections of algorithm implementations [10]. Four providers are included: Default, Legacy (for older algorithms), Engines (with a compatibility layer for engines from older OpenSSL versions), and FIPS. Providers make algorithms available through OpenSSL's high-level API. Low-level APIs supported by older versions were deprecated. Many smaller changes were made, including changes to data structures and error handling.

This study compares how code quality changed during the development of OpenSSL versions 1.1.1 and 3.0.0 using static code metrics. Our choice of metrics was guided in part by Lehman's laws of software evolution [6]. We analyze the second law (increasing complexity) using code complexity metrics and the sixth law (continuing growth) using code size metrics. We also examine language feature use and style metrics. Our research questions were:

**RQ1** How did the size and complexity of OpenSSL evolve differently during the development of versions 1.1.1 and 3.0.0?

**RQ2** How did the use of programming language features and coding style evolve differently between the two versions?

## 2 APPROACH

We used the GrimoireLab [4] toolset to compute software metrics. Graal is the component of GrimoireLab that performs source code analysis. In order to collect a broader variety of code metrics than were currently supported by Graal [3], we developed a new backend for Graal to support the `cqmetrics` tool [1]. This tool has been previously used to study OpenSSL [12].

We discovered a bug in Graal's unit test suite that caused many unit tests to return successful results regardless of the actual test results. We submitted a pull request to fix this bug [2].

Graal provides options to select commits by `git` branch, date ranges, and files. Due to the many thousands of commits made during OpenSSL 3.0.0 development, we needed to sample a subset of commits. We modified Graal to add the capability to select the

[1] https://github.com/chaoss/grimoirelab-graal/pull/110
[2] https://github.com/chaoss/grimoirelab-graal/pull/103


*MSR '22, May 23–24, 2022, Pittsburgh, PA, USA*

ACM ISBN 978-1-4503-9303-4/22/05…$15.00
https://doi.org/10.1145/3524842.3528035

**Table 1: Size and Complexity Changes**

| | OpenSSL 1.1.1 | | OpenSSL 3.0.0 | |
|---|---|---|---|---|
| Metric | Change | Percent | Change | Percent |
| Number of Files | 168.00 | 16.3% | 553.00 | 46.2% |
| Lines of Code | 68,339.00 | 18.8% | 183,605.00 | 42.6% |
| Number of Functions | 2,068.00 | 25.5% | 5,581.00 | 54.9% |
| Mean Function Length | −1.08 | −8.2% | −1.23 | −10.2% |
| Cyclomatic (mean) | −0.59 | −9.8% | −0.52 | −9.6% |
| Halstead (mean) | −187.00 | −16.2% | −269.37 | −27.9% |

**Table 2: Language Feature Use**

| | OpenSSL 1.1.1 | | OpenSSL 3.0.0 | |
|---|---|---|---|---|
| Metric | Change | Percent | Change | Percent |
| Number of gotos | 1,369.00 | 19.0% | 3,857.00 | 45.0% |
| goto density | 0.00 | 3.2% | 0.00 | 4.2% |
| Number of CPPs | 3,036.00 | 10.7% | 9,447.00 | 30.1% |
| C preprocessor density | −0.01 | −4.0% | −0.02 | −6.5% |
| Number of includes | 921.00 | 17.4% | 3,970.00 | 63.9% |
| CPP include density | 0.00 | 1.8% | 0.01 | 17.8% |
| Number of conditionals | 169.00 | 12.2% | 450.00 | 29.0% |
| CPP conditional density | 0.00 | −2.7% | 0.00 | −7.3% |

first commit made each month, so that we could construct monthly time series for each version. We plan to submit a pull request for this feature.

We examined both versions of OpenSSL from creation of the version's branch creation to the public release of the version. The OpenSSL 3.0.0 branch was created on the same day that version 1.1.1 was released. There are 26 months of OpenSSL 1.1.1.1 data and 37 months of OpenSSL 3.0.0 data. We analyzed snapshots of monthly code metrics starting from the creation of each branch through the announcement of a final, public release in `git` commit messages.

## 3 EXPLORATORY ANALYSIS

OpenSSL 3.0.0 is a significant redesign of OpenSSL, focused on the new providers abstraction. However, it retains backwards compatibility, deprecating but not removing low level APIs and retaining the Legacy provider to maintain access to old algorithms. This resulted in OpenSSL 3.0.0 being the largest version of OpenSSL, with 614,738 lines of code. It is 42.6% larger than OpenSSL 1.1.1, with its 431,113 lines of code. Significantly expanding a project while improving code quality is a challenge. We examine how well OpenSSL 3.0.0 rose to that challenge through our two research questions below.

### 3.1 Size and Complexity

Our first research question focuses on the how the evolution of code size and complexity differed between versions 1.1.1 and 3.0.0 of OpenSSL. We find that while OpenSSL 1.1.1 substantially increased the size of the library, OpenSSL 3.0.0 added almost three times as much code. In both major releases, the number of functions increased substantially while mean function length decreased, showing an increased use of smaller functions. Many programmers find that smaller functions are easier to understand [7].

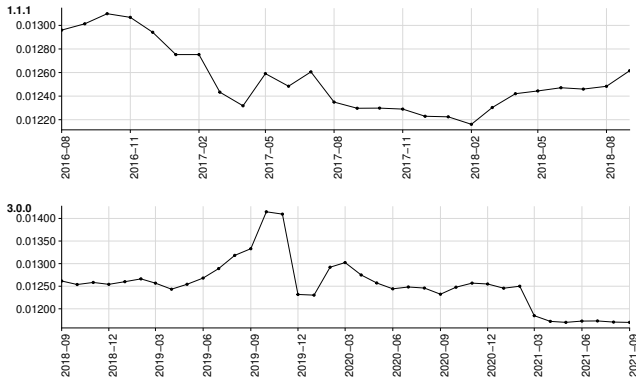
Although both versions added a large amount of code, code complexity decreased over the course of their development. The mean cyclomatic complexity changed by -0.59 (-9.8%) in version 1.1.1 and by -0.52 (-9.6%) in version 3.0.0. The mean Halstead complexity changed by -187 (-16.2%) in version 1.1.1 and by -269.37 (-27.9%) in version 3.0.0. Code size and complexity metric changes over the development of both releases are summarized in Table 1. It seems plausible that the greater design effort focused on version 3.0.0 allowed code complexity to decrease while code size substantially increased.

### 3.2 Language Feature Use

Our second research question focuses on the how the evolution of programming lanaguage feature usage and coding style differed between versions 1.1.1 and 3.0.0 of OpenSSL. We examined language feature metrics as counts and densities. Densities refer to the count of a language feature per thousand lines of code. Normalizing metrics by lines of code allows us to make fair comparisons between the differently sized code bases.

An empirical study of GitHub projects suggested that use of `goto` for specific purposes like error handling was considered good practice by open source developers [8]. However, using C preprocessor conditionals for portability has been blamed for increasing the difficulty of reading and maintaining code [11].Changes in both features over the development of both versions are summarized in Table 2.

**Figure 1: Preprocessor Conditional Density**




Use of `goto` statements increased in both absolute numbers and density for both releases. The density of `goto` statements increases by 3.2% in version 1.1.1 and 4.2% in version 3.0.0 over the course of their development. Usage of `goto` in OpenSSL is focused on error handling, as approximately 80% of `goto` labels in both versions are `err`. The next two most common labels, `end` and `done`, make up another 10% of usage, further supporting the argument that the vast majority of `goto` statements are forward jumps to handle errors or deallocate resources.

While the use of C preprocessor directives increased in absolute terms, the number of such directives compared to lines of code decreased for both versions, as we can see in Table 2. However, the density of include directives increased slightly in version 1.1.1 (1.8%) and substantially (17.8%) in version 3.0.0. Combined with the increase in the number of files, this points to the code base being refactored into a larger number of modules in version 3.0.0.

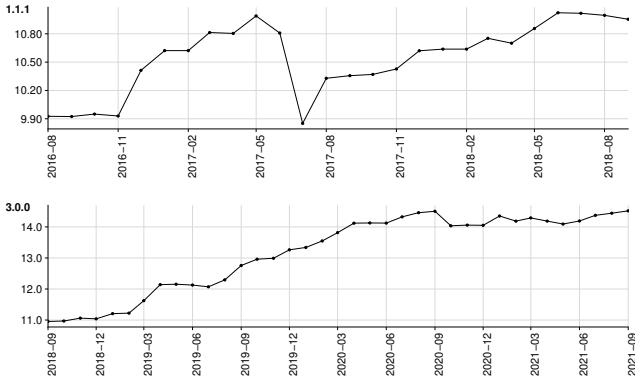
The use of potentially problematic preprocessor conditionals decreased in both version 1.1.1 (-2.7%) and 3.0.0 (-7.3%). While the trend of preprocessor conditional use was generally downwards, we can notice peaks and troughs during the development process in Figure 1. There are seven commits in the 3.0.0 branch that mention preprocessor conditions between September 2019 and March 2020. One describes staging a change but preventing it from taking effect by using preprocessor statements, which caused a short term increase in preprocessor conditionals that ended once the conditionals preventing the change were removed.

### 3.3 Style Consistency

The `cqmetrics` packages computes a style inconsistency metric. This metric measures inconsistency for 19 style rules. For each way to format a particular construct, it measures the times the rule is applied in the one way (e.g., adding a space before a brace) and the times the rule is applied in the other way (e.g., omitting the space).

OpenSSL reduced style inconsistency tremendously in 2015 from 57.0 to 4.0 by automatically reformatting the code base [12]. Since 2015, style inconsistency has drifted consistently upwards. During development of OpenSSL 1.1.1, style inconsistency increased by 10.4% from 9.93 to 10.95. Style inconsistency increased more rapidly during the development of OpenSSL 3.0.0, with a total increase of 32.5% to a final value of 14.4. The evolution of inconsistency shown in Figure 2 show almost continual increases, especially in version 3.0.0.

**Figure 2: Style Inconsistency**




## 4 CONCLUSION

We compared the evolution of OpenSSL versions 1.1.1 and 3.0.0 using a new backend that we developed for Graal to compute static code metrics with `cqmetrics`. OpenSSL 3.0.0 was developed using public design documents and a focus on more documentation and unit tests. We observe substantial improvements in code quality during the development of 3.0.0 even as 183,605 lines of code were added, which is three times the number of lines added for 1.1.1.

Code complexity decreased for both versions, even as code size increased. Mean Halstead complexity decreased by 27.9% in version 3.0.0 and by 16.2% in 1.1.1. OpenSSL 3.0.0 showed substantially larger increases in the number of files (42.6% compared to 16.3%) and `#include` statement density (17.8% compared to 1.8%) than 1.1.1. These changes point to the code base being refactored into larger number of modules. The use of potentially problematic preprocessor conditionals decreased faster in version 3.0.0 compared to 1.1.1 (7.3% compared to 2.7%) suggesting that version 3.0.0 is using safer techniques for portability. Style inconsistency increased for both versions, with a noticeably larger increase for version 3.0.0 (32.5% compared to 10.4%).

Future work includes use of new data sources to help connect design documents to observed changes in code quality. We did not have time to examine the pull requests and issues, which may show connections between design and code that could not be found in commit messages and mailing lists. GrimoireLab supports accessing both types of data. Another data source for future consideration are additional code metric tools supported by Graal. We also want to deepen our investigation of why code metrics have changed. Static code metrics can change due to new programming language features and language usage, as well as from modifications to the software engineering process.

## ACKNOWLEDGMENTS

We thank the MSR 2022 Hackathon track PC members and reviewers for their comments. We especially thank Jesus M. Gonzalez-Barahona, who was always available to answer questions.

## REFERENCES

[1] Matt Caswell. 2018. *The Holy Hand Grenade of Antioch.* OpenSSL Management Committee. https://www.openssl.org/blog/blog/2018/11/28/version/
[2] Matt Caswell. 2021. *OpenSSL 3.0 Has Been Released!* OpenSSL Management Committee. https://www.openssl.org/blog/blog/2021/09/07/OpenSSL3.Final/
[3] Valerio Cosentino, Santiago Duenas, Ahmed Zerouali, Gregorio Robles, and Jesús M González-Barahona. 2018. Graal: The Quest for Source Code Knowledge. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM).* IEEE, 123–128.
[4] Santiago Dueñas, Valerio Cosentino, Jesus M Gonzalez-Barahona, Alvaro del Castillo San Felix, Daniel Izquierdo-Cortazar, Luis Cañas-Díaz, and Alberto Pérez García-Plaza. 2021. GrimoireLab: A toolset for software development analytics. *PeerJ Computer Science* 7 (2021), e601.
[5] Zakir et al. Durumeric. 2014. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement.* ACM, 475–488.
[6] Manny M Lehman. 1996. Laws of software evolution revisited. In *European Workshop on Software Process Technology.* Springer, 108–124.
[7] Robert C Martin. 2009. *Clean code: a handbook of agile software craftsmanship.* Pearson Education.
[8] Meiyappan Nagappan, Romain Robbes, Yasutaka Kamei, Éric Tanter, Shane McIntosh, Audris Mockus, and Ahmed E Hassan. 2015. An empirical study of goto in C code from GitHub repositories. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering.* ACM, 404–414.
[9] OMC. 2020. *OpenSSL 3.0.0 Design.* OpenSSL Management Committee. https://www.openssl.org/docs/OpenSSL300Design.html
[10] OMC. 2021. *OpenSSL migration guide.* OpenSSL Management Committee. https://www.openssl.org/docs/man3.0/man7/migration_guide.html
[11] Henry Spencer and Geoff Collyer. 1992. # ifdef confirmed harmful or portability experience with C news. In *Proceedings of the Summer 1992 USENIX Conference.* USENIX, 185–198.
[12] James Walden. 2020. The impact of a major security event on an open source project: The case of OpenSSL. In *Proceedings of the 17th International Conference on Mining Software Repositories.* ACM, 409–419.