Towards Building API Usage Example Metrics

Stevche Radevski*, Hideaki Hata*, Kenichi Matsumoto*
*Graduate School of Information Science
Nara Institute of Science and Technology, Nara, Japan 630-0101
Email: [stevche.radevski.sl1, hata, matumoto]@is.naist.jp

***Abstract*—It is not unreasonable to say that examples are one of the most commonly used knowledge sources when learning the usage and best practices of a new API. That being said, in many cases the examples provided on the APIs' website are lacking in quantity or quality, so developers have to resort to other information sources, namely blogs and coding forums. Moreover, there is no good way for API developers to measure anything concerning the examples they are creating.**

In order to resolve the problem of lacking examples information and feedback in their creation, our goal is to develop metrics for empirical measurement of examples and to offer support during the APIs example creation steps, and this paper represents the starting point towards that aim.

We have analyzed the source code examples provided on the API's website or GitHub directory for 7 popular API libraries written in Java and measured certain metrics, such as example coverage, example code to source code ratio, class coverage percentage, and problems that occur with the compilation and execution of existing examples. The purpose of this paper is to investigate the current situation of examples and provide a starting knowledge base for building an automatic tool for source code example metrics analysis.

I. Introduction

As the software engineering field matures, so does the development of new libraries, allowing for higher reuse of source code. In many of the cases there exists a library fit for most of the problems faced by the majority of software developers, so the need for learning new APIs occurs frequently and imposes a lot of learning difficulties. The large number of APIs used in projects does not allow for learning of a single API in details, therefore the importance of using examples as a quick reference or learning material increases even more. As a matter of fact, one study found that inadequate resources are a big obstacle in learning an API, where the lack of examples was rated as having the highest importance [1]. In their follow up study they validated their hypothesis of documentation being the biggest obstacle in learning an API [2]. Examples being a very important learning resource has also been mentioned in some previous studies [3], [4], among many others.

Ideally, sufficient and good quality examples would be located in one centralized source, such as on the APIs' websites or GitHub wiki pages. In such a case, the need for asking questions on coding forums and writing blogs about how a certain API should be used will certainly decline. Moreover, in such a condition, searching for examples and code snippets will be much more efficient, having only one place the developer needs to search at. It is not easy to argue what the reason for not providing sufficient examples is, but not knowing how good or how many examples are enough can be mentioned as one reason. It is certainly not important what method is used to create the examples, yet what is provided to the users of an API is the important part. There has been some very promising research in automatic generation and retrieval of examples from various sources [5], [6], [7], and also different recommendation systems and search engines have been proposed [8], [9]. The question that is imposed from this is, how can we say whether the examples that were created either by developers or tools are sufficient and of good quality or not?

Even after using tools and systems such as the ones mentioned before, to our knowledge, there is no way of evaluating the created examples and retrieving metrics concerning examples. There is a need to be able to measure and evaluate the result of such tools in a more empirical manner, and this is one of the motivations to develop metrics for examples. Moreover, if API developers have access to example metrics, they can get feedback on the information they provide to the potential users of the API they develop. After all, the end result of months of work on an API should be rewarded with full utilization of the abilities the API provides by other developers with minimal effort. This is especially true for new APIs that do not have any supporting information on coding forums and blogs, so they depend fully on the documentation and examples provided by the API developers. Example metrics can also point at code that will be called frequently and is of higher risk, how potential users would use the API, and what part of the API is less likely to be used. Finally, making such metrics public for an API can even show the ease of learning and usability of the API at hand. That being said, some of the possible metrics are example coverage, example quality, repeated calls to same methods (recurring coverage), usefulness, and example age, among many other plausible metrics.

In order to investigate some example metrics and several characteristics of the examples on the web, we have examined 7 popular API libraries written in Java. We examined the examples they have provided on either their website/GitHub page, or within the source code in a package names "samples" or " examples". We manually extracted all the examples, analyzed them by utilizing already existing tools for test coverage, and measured coverage and other derived metrics. We found that examples provided in the mentioned resources have around 30% coverage on average, where even a small amount (2.8% example lines of code to non-comment lines of code ratio) of example code can achieve such coverage,

and in some cases even the first example provided reaches up to 26% of coverage. Also, we found that using non-existent dummy code, thrown exceptions, and inconsistent naming of same variable in the same example were the most common problems during the compilation and execution of examples.

II. Methodology

Based on our goals to examine the current situation of examples and their potential metrics, we have derived three research questions:

- **Research Question 1:** What is the example coverage by the examples provided only by the API developers?
- **Research Question 2:** How well are individual classes covered by examples?
- **Research Question 3:** What kind of problems exist when trying to compile and execute example source code?

In order to measure example coverage and derive other metrics, we used the IntelliJ IDEA integrated development environment (IDE), along with Atlassian Clover tool for tests code coverage. In order to use the tool for measuring the coverage of examples, we simply used examples as if they were unit tests, and ran them using the tool. The study was executed in 3 stages: library selection and source code retrieval, examples extraction, and example characteristics analysis and report generation. Stage 2 and 3 were done in parallel in order to avoid repetition in processing. In the next sections the Clover tool and each of the stages are explained in details.

A. Atlassian Clover

Atlassian Clover[1] is a Java code coverage tool by Atlassian. It is a commercial product that is free to use for open-source projects. In this research the Clover-for-IDEA tool was used, Clover tool for IntelliJ IDEA. Some details about how the code coverage is calculated will be described further in this section. The source of the information on how the code coverage is calculated originates from Atlassian Clover website[2] and it is described as it appeared at the time of the writing of this paper.

By definition, code coverage is the percentage of code that is covered by tests. Clover has 3 types of coverage analysis:

- **Statement** - statement coverage measures whether each statement is executed.
- **Branch** - Branch coverage measures which possible branches in flow control structures are followed.
- **Method** - Method coverage measures if a method was entered at all during execution.

The total coverage calculated by Clover is done using the following equation:

$$TPC = \frac{(BT + BF + SC + MC)}{(2 * B + S + M)} * 100\%$$

$BT =$ branches that evaluated to "true" at least once

$BF =$ branches that evaluated to "false" at least once

$SC =$ statements covered

$MC =$ methods entered

$B =$ total number of branches

$S =$ total number of statements

$M =$ total number of methods

Even though using the equation above and Clover as a tool may not have been the ideal way of calculating coverage in terms of example coverage, it seemed a good starting point for our purpose and research aim.

B. Study Structure

1) Selecting Libraries: In order to investigate the present situation of example metrics, we selected 7 API libraries written in Java as our data source. There were several criteria that were important in the selection of the libraries, namely: the library had to be written in Java, and meant to be used in Java. By doing this, the execution of examples as tests was simplified, having both examples and development language being Java. The library had to be open-source. The library had to be smaller than 40,000 LOC in order to make manual extraction of examples and execution plausible. Another criterion was choosing well-established libraries, with established testing and building environment, and mature wikis/websites. Even though this implies a bias in the data source, it made sure that certain documentation exists and the measurement of example metrics is possible with the existing tools for test coverage. Finally, we made sure to include libraries having varying purposes.

The libraries were chosen from the top 100 Java libraries based on their usage as found on Takipi blog[3]. Using the list, we selected 7 libraries that matched our criteria, deeming it enough for the purposes of this study. The chosen libraries are shown in Table I.

2) Example Source Code Extraction: During this stage, the examples source code was extracted from one or more of the following locations: the website of the library, the GitHub page, and from inside the source code directory in a folder named "samples", "examples", or the equivalent. Example sources aside from these were not considered for this study. All locations that contained example source code were retrieved and documented separately for each library.

After locating all the examples, the next step was extracting the example source code and running it as a test. For this purpose, all the projects were individually loaded in IntelliJ IDEA, the original tests were run to check if the coverage works, and then all tests and calculated coverage were erased. One or more test files and test methods for holding the example source code were created depending on how the examples were structured at the source of each project.

[1]https://www.atlassian.com/software/clover/overview

[2]https://confluence.atlassian.com/display/CLOVER/About+Code+Coverage

[3]Alex Zhitnitsky, 'We Analyzed 60,678 Libraries on GitHub Here are the Top 100', *The Takipi Blog* [web blog], 14 April 2015, http://blog.takipi.com/we-analyzed-60678-libraries-on-github-here-are-the-top-100 (accessed 27 November 2015)

TABLE I
COVERAGE OF LIBRARIES

Library Name	Coverage %	First Example Coverage %	LOC	NCLOC	Examples NCLOC (ENCLOC)	ENCLOC/NCLOC Ratio %
cglib	46.7	26.3	13,991	9,140	213	2.33
guice	35.5	25.7	23,295	12,230	125	1.02
easymock	35.0	17.6	10,291	4,511	432	9.58
xstream	31.5	21.6	34,889	20,468	545	2.66
gson	29.6	14.3	13,649	7,392	208	2.81
commons-logging	21.5	13.2	6,436	2,604	24	0.92
commons-io	4.6	0.9	27,723	9,681	37	0.38

The example source code was clearly shown in all of the libraries' source location by providing it in a distinguishable section from the natural language explanation. These sections will further on be referred to as example blocks. The example blocks that had a successor example block built on top of the original one (a superset of the original example block) as a consequence of explanation flow in natural language were removed, and the successor was added.

The example blocks were copied one by one, and executed every time a new example block was added. In case there was a problem with compilation and execution, the cause and type of error was noted, assisting in the analysis for stage 3 of this research. After that, the problem was resolved without adding any source code related to the library being analyzed. For example, if there was a dummy class missing, it was created in a way to just satisfy the compilation errors. In case there was unhandled exception, it was again handled just so it passes compilation and execution. For more details on the problems that occurred, please refer to Table II.

3) Examples Characteristics Analysis and Report Generation: During the third stage several characteristics of the example source code were observed, all "tests" were run again, and the Clover reports were generated. We wanted to investigate what problems would occur if the code was to be mined using a tool and executed. The problems that occurred during execution and compilation of the examples can be found in Table II. Other metrics that were either generated by Clover or derived from the example source code and the report can be found in Table I and Figure 1.

In order to document all the compilation and execution problems, notes were taken after each unsuccessful execution of an example block during stage 2 of this research. After the entire example source code was executed for a particular library, all the different problems that occurred were compiled in a table. The same procedure was done for every library, and the results were compiled as shown in Table II. Since our goal was not to measure how many times a certain problem occurs in a single library, but what kind of problems occur in how many libraries, the problems are documented on a library granularity. It is important to note that the explanation of the examples using natural language was not considered unless it pointed out to the solution of the problem that occurred while executing the example block, or it stated that the example block will have that problem and gave the solution for it.

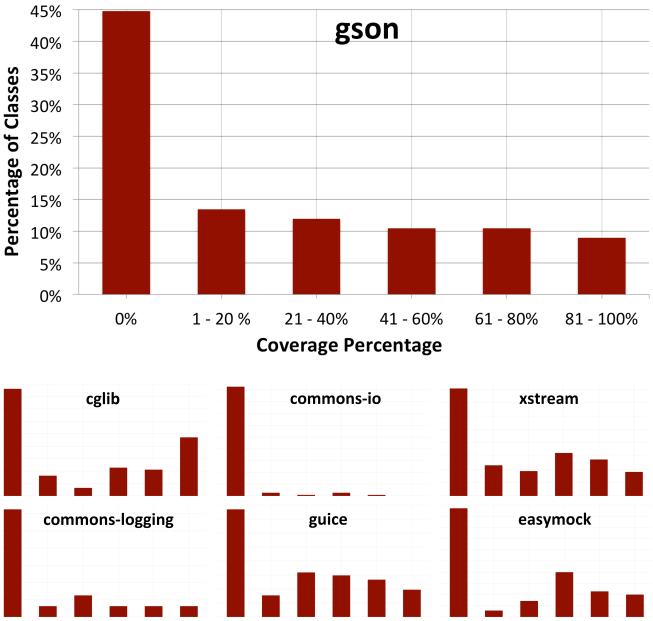

Fig. 1. Classes coverage

The last analysis that was done is calculating what is the coverage percentage for each class. Using some already generated results concerning class coverage by Clover, we compiled all the information for each library, and then calculated the percentage of classes that had coverage within a certain range. The results from the analysis can be found in Figure 1.

III. RESULTS

Once the three stages of the study process were finished, we obtained results to answer our 3 research questions.

A. Research Question 1

The results for the research question 1 are summarized in Table I. What we can see is that, on average, the coverage of examples provided by the library developers is around 30%. Whether that is sufficient or not cannot be discussed at this point, since further research on the topic is needed. We can also see that the Apache Commons libraries, *commons-io* in particular, have very low coverage and a very small number of examples. Arguably, the low coverage would result in higher

percentage of questions on coding forums and a larger number of blog posts for these libraries.

Another interesting result is concerning the *easymock* library. Even though they have provided a relatively large amount of examples (almost 10% example to source code ratio), their coverage is not any higher compared with the rest of the libraries that have much smaller ratio. What this may mean is there has been a lot of repetition in their examples, yet many other functionalities were ignored. This is a good example of how example metrics can be useful to library developers.

Furthermore, we can notice that even one fourth of the code was covered only by the first example. The first example refers to the very first example provided on the website, synonymous to "Hello World" application in many cases. Once again, it is still early to discuss what the implications of that are and how this information can be used, requiring further research.

B. Research Question 2

The results for the research question 2 are summarized in Figure 1. Note that the y-axis range is relative to the library, so it would be more reasonable to look at the ratio between different coverage percentages. Our reason for calculating this was to see if there are classes that do not appear at all in the examples or on their execution path. A very surprising result is that more than half of the classes on average have no coverage at all. Moreover, only less than 20% of the classes have more than 60% of coverage.

What this means is that there are large number of classes for which the users cannot learn anything about just by looking at the examples. Furthermore, these results may imply which classes are the true core of the library, and which classes may never be used, or used very infrequently. The coverage percentage is certainly alarming, even if we consider that there may be classes that are not necessary to be covered by an example. Once again, being at this early stage in the research, it is still not clear what a good coverage percentage is and what is the best way to calculate it, but these results certainly give an insight about it.

C. Research Question 3

The results for the research question 3 are summarized in Table II. It is important to state that some of the problems occurred more than once in a single library, but since our aim was to investigate what kind of problems occur when running examples and how common it is among different projects, we did not count the number of occurrences.

Even though using dummy data that does not exist comes to no surprise, the other problems such as exceptions, syntax errors, and a single variable named differently in the same example were not expected. The very frequent occurrences of these problems show that many of the examples have not been run or compiled by the developers or they are just outdated, which complicates things for the users of the library. Moreover, since examples are expected to reflect the "best practices" of a library [2], problems like these should not exist and are not acceptable. However, it is not necessarily the fault of developers, it may just be that developers are not aware of the state of their examples and have no efficient way of checking it, further emphasizing the importance of example metrics and support tools.

TABLE II
PROBLEMS DURING COMPILATION AND EXECUTION

Problems that occurred	**Num. of Libraries**
Using data/methods that do not exist in dummy data/classes	5
Exception thrown	5
Same variable with different name in same example	4
Syntax error (different from variable naming)	3
Unhandled exception preventing from compilation	3
Ellipsis used	2
No imports included and were not automatically detected	1
Methods used in example depricated	1

IV. Related Work

Although we did not find any studies that directly research the topic of example metrics, we did find several studies that are related to some measurement of examples. One study examined some of the characteristics for what a good example is based on their analysis on StackOverflow[4] [10]. Even though this study is not related to coverage measurement, it does provide good information in building other example metrics for quality of examples.

Another study investigated the API documentation on the internet, that is, where it occurs, how well it is covered, and how blogs are involved in the software development process, through the *jQuery* library [11]. In order to calculate the coverage, they checked whether the name of the method they searched for appears on the site, and made a judgment based on that. A study by two of the same authors of the previous paper explored the coverage of the *Android* API, *GWT* API, and *Java language* API with the aim of measuring APIs coverage on StackOverflow [12]. Once again, they measured coverage of classes by whether the name appeared in a post or not, which can be quite imprecise. A third study that did some kind of example coverage calculation is they calculated the number of times a method appears in different examples in their database for the *Android* API [13].

There may have been more studies that have used some kind of example coverage calculation, but we did not find any study that directly researches the topic. The coverage calculation in the previously mentioned studies was done as a side effect, means to achieving their research goal. These studies implicitly show the need for example metrics and

[4]http://stackoverflow.com

some of the ways the metrics can be applied in academia. Should there have existed example metrics, it would have been relatively easier to do the studies in this section and to do more detailed measurement.

V. Future Work

Being at the early stages of our research, there is a lot more to be done. Further in this section, the short-term and long-term research goals will be described.

The short-term goal followed by this research is building example metrics and tools to automatically mine for examples and calculate the built metrics. These metrics can be very simple for a start, and can be extended further on as the research matures. We believe that these metrics by themselves will be a very valuable asset to both library developers and to anyone else writing coding examples, blogs, and similar. Since there is no research related to correlations between example metrics and library success, usage, and library design, among other potential correlations, it is difficult to discuss about other indirect benefits from example metrics at this stage.

The long-term goal is creating an example database site, containing full examples for most of the libraries that exist. Borrowing concepts from open-source software development, this service should be community-based, where the community contributes working examples and the main developers do the review. This is especially important for new libraries that will not have any source code snippets anywhere on the Internet, so the community contributing in creating those examples can be crucial for the success of the library. Of course, any of the example generation tools mentioned in the introduction can be used whenever applicable. This database of examples can also be used in less-obvious ways, such as creating tests from the examples [7].

This system will be based and it will revolve around the example metrics we have discussed earlier. Although similar to existing websites such as StackOverflow, there are key differences such as the purpose, and the shift in the involvement of the community from answering asked questions to giving fully working answers (examples) for potential, future questions.

VI. Conclusion

Several studies have shown the importance of examples in the process of learning and using an API, for some being a quick way to get things done, where for others they represent best practices of using an API. We need more information on examples, where knowing how useful the provided examples are, how well they cover a certain API, and what is the quality of the examples are some of the questions that need answering. Our research goal is to develop metrics for examples to answer some of those questions. Throughout this paper a number of applications were shown, such as how the metrics can be used for example generation tools evaluation and how metrics can be used in assisting API developers in providing better and more examples.

In order to investigate some of the potential metrics, we analyzed 7 API libraries written in Java in order to investigate the current situation of examples, test the applicability of example metrics, and find out what problems can happen if examples are to be mined by a tool, so the problems can be mitigated. We found that the coverage provided by API developers on their websites is relatively low. Moreover, over half of the classes on average had no coverage whatsoever. Finally, we found that there are many problems that occur while compiling and running examples, such as exceptions, inconsistent naming, and using inexistent methods and data, which makes the usage of those examples more difficult and time consuming. Finally, we talked about some of our future goals and where our research is headed.

Acknowledgment

This work has been supported by JSPS KAKENHI Grant Number 26540029, and Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers: Interdisciplinary Global Networks for Accelerating Theory and Practice in Software Ecosystem.

We would also like to thank Professor Daniel M. German for the fruitful discussion and suggestions we received concerning this research.

References

[1] M. P. Robillard, "What makes APIs hard to learn? answers from developers," *IEEE Software*, vol. 26, no. 6, pp. 27–34, 2009.

[2] M. P. Robillard and R. Deline, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[3] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, S. R. Klemmer, and S. Francisco, "Two Studies of Opportunistic Programming : Interleaving Web Foraging , Learning , and Writing Code," *ACM Conference on Human Factors in Computing Systems*, pp. 1589–1598, 2009.

[4] F. Shull, L. Filippo, and V. R. Basili, "Investigating reading techniques for object-oriented framework learning," *IEEE Transactions on Software Engineering*, vol. 26, no. 11, pp. 1101–1118, 2000.

[5] R. P. L. Buse and W. Weimer, "Synthesizing API usage examples," *Proceedings - International Conference on Software Engineering*, pp. 782–792, 2012.

[6] S. M. Nasehi and F. Maurer, "Unit tests as API usage examples," *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pp. 1–10, 2010.

[7] M. Gaelli, R. Wampfler, and O. Nierstrasz, "Composing tests from examples," *Journal of Object Technology*, vol. 6, no. 9, pp. 71–86, 2007.

[8] L. Wang, L. Fang, L. Wang, G. Li, B. Xie, and F. Yang, "APIExample: An effective web search based usage example recommendation system for Java APIs," *2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings*, no. 60803010, pp. 592–595, 2011.

[9] J. Stylos and B. a. Myers, "Mica: A web-search tool for finding API components and examples," *Proceedings - IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2006*, pp. 195–202, 2006.

[10] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming Q&A in StackOverflow," *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pp. 25–34, 2012.

[11] C. Parnin and C. Treude, "Measuring API documentation on the web," *Proceeding of the 2nd international workshop on Web 2.0 for software engineering - Web2SE '11*, pp. 25–30, 2011.

[12] C. Parnin, C. Treude, L. Grammel, and M. Storey, "Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow," *Georgia Tech Technical Report*, 2012.

[13] J. E. Montandon, H. Borges, D. Felix, and M. T. Valente, "Documenting APIs with examples: Lessons learned with the APIMiner platform," *2013 20th Working Conference on Reverse Engineering (WCRE)*, no. Section V, pp. 401–408, 2013.