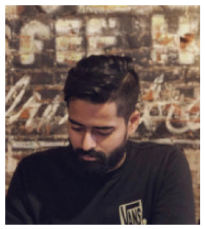
ICPC '19: 27th IEEE/ACM International Conference on Program Comprehension, May 24--25, 2019

Improving Source Code Readability: Theory and Practice



Sarah Fakhoury Devjeet Roy Sk. Adnan Hassan Venera Arnaoudova

Washington State University, USA

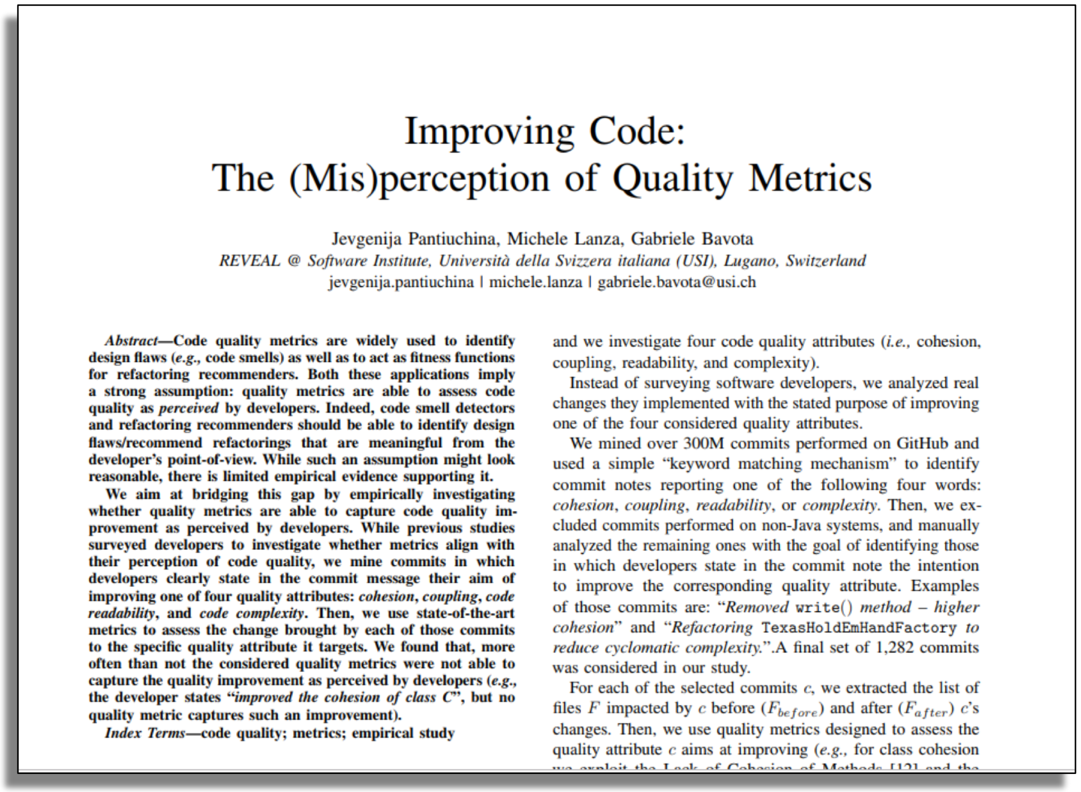
WASHINGTON STATE UNIVERSITY

Code Quality Metrics

- Almost all software projects are evaluated for **code quality**
- Researchers have developed several metrics to **measure** code quality
  - Complexity
  - Maintainability
- Tools rely on these metrics to recommend quality improvements

Quality Metrics in Practice

- Pantiuchina et al. ICSME'18
- Investigated if developers perception of quality in practice aligns with metrics defined in literature
- Often, State-of-the-art metrics are unable to capture quality improvements



Improving Code: The (Mis)perception of Quality Metrics

Jevgenija Pantiuchina, Michele Lanza, Gabriele Bavota
REVEAL @ Software Institute, Università della Svizzera italiana (USI), Lugano, Switzerland
jevgenija.pantiuchina | michele.lanza | gabriele.bavota@usi.ch

***Abstract*—Code quality metrics are widely used to identify design flaws (*e.g.,* code smells) as well as to act as fitness functions for refactoring recommenders. Both these applications imply a strong assumption: quality metrics are able to assess code quality as *perceived* by developers. Indeed, code smell detectors and refactoring recommenders should be able to identify design flaws/recommend refactorings that are meaningful from the developer's point-of-view. While such an assumption might look reasonable, there is limited empirical evidence supporting it.**

We aim at bridging this gap by empirically investigating whether quality metrics are able to capture code quality improvement as perceived by developers. While previous studies surveyed developers to investigate whether metrics align with their perception of code quality, we mine commits in which developers clearly state in the commit message their aim of improving one of four quality attributes: *cohesion*, *coupling*, *code readability*, and *code complexity*. Then, we use state-of-the-art metrics to assess the change brought by each of those commits to the specific quality attribute it targets. We found that, more often than not the considered quality metrics were not able to capture the quality improvement as perceived by developers (*e.g.,* the developer states "*improved the cohesion of class C*", but no quality metric captures such an improvement).

***Index Terms*—code quality; metrics; empirical study**

and we investigate four code quality attributes (*i.e.,* cohesion, coupling, readability, and complexity).

Instead of surveying software developers, we analyzed real changes they implemented with the stated purpose of improving one of the four considered quality attributes.

We mined over 300M commits performed on GitHub and used a simple "keyword matching mechanism" to identify commit notes reporting one of the following four words: *cohesion*, *coupling*, *readability*, or *complexity*. Then, we excluded commits performed on non-Java systems, and manually analyzed the remaining ones with the goal of identifying those in which developers state in the commit note the intention to improve the corresponding quality attribute. Examples of those commits are: "*Removed* `write()` *method – higher cohesion*" and "*Refactoring* `TexasHoldEmHandFactory` *to reduce cyclomatic complexity.*".A final set of 1,282 commits was considered in our study.

For each of the selected commits c, we extracted the list of files F impacted by c before (F_{before}) and after (F_{after}) c's changes. Then, we use quality metrics designed to assess the quality attribute c aims at improving (*e.g.,* for class cohesion

Quality Metrics in Practice

- Metrics are unable to capture majority of instances where improvements where made by developers in practice

- Numbers as high as as 85% accuracy are reported for evaluation of metrics that are defined in literature [1]

Discrepancy between results from research and the interpretation of metrics in practice

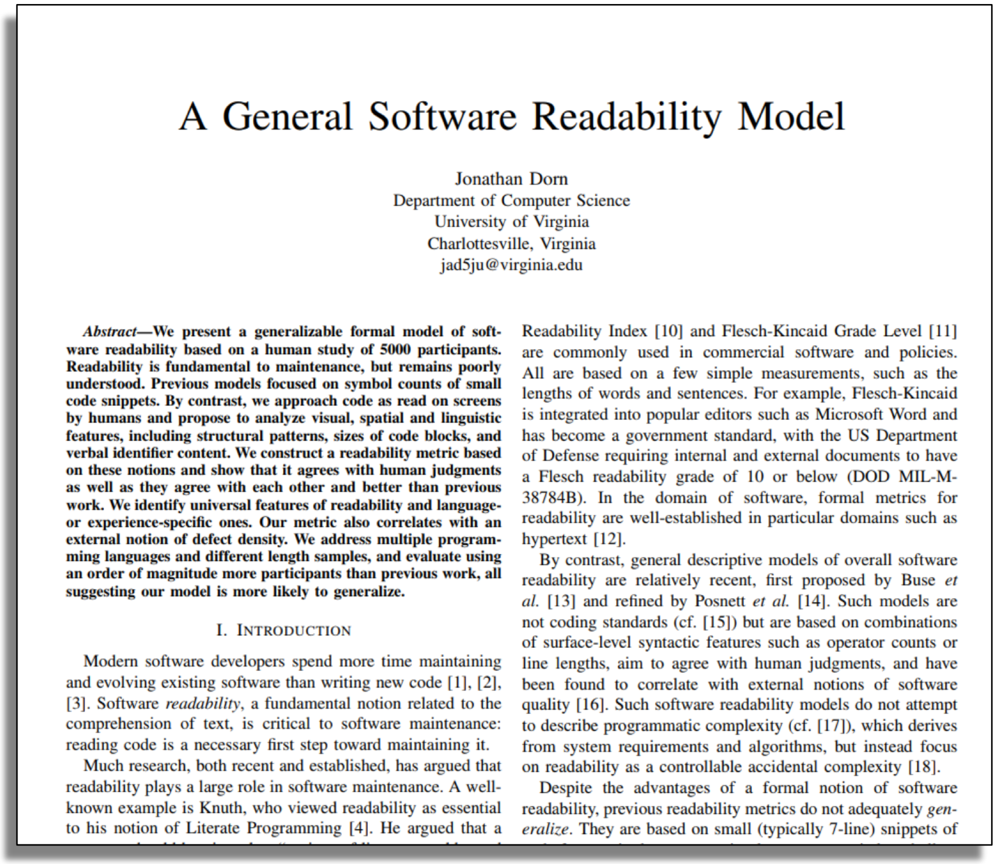
[1] Simone Scalabrino, Mario Linares-Va´squez, Rocco Oliveto, and Denys Poshyvanyk. A comprehensive model for code readability. *Journal of Software: Evolution and Process*, 30(6):e1958, 2018.

Code Quality: Readability

- Fundamental part of software maintenance
  - Impacts comprehension
  - Time-intensive
- Multiple models have been defined to measure readability
  - Consider different aspects of the source code (structural features, lexical, etc.)
- Usually models are evaluated by surveying developers

Readability Models

- Dorn in 2012
- Considers **visual**, **spatial**, and **linguistic** aspects of the source code
- Code Snippets evaluated by surveying 5,000 participants



A General Software Readability Model

Jonathan Dorn
Department of Computer Science
University of Virginia
Charlottesville, Virginia
jad5ju@virginia.edu

***Abstract*—We present a generalizable formal model of software readability based on a human study of 5000 participants. Readability is fundamental to maintenance, but remains poorly understood. Previous models focused on symbol counts of small code snippets. By contrast, we approach code as read on screens by humans and propose to analyze visual, spatial and linguistic features, including structural patterns, sizes of code blocks, and verbal identifier content. We construct a readability metric based on these notions and show that it agrees with human judgments as well as they agree with each other and better than previous work. We identify universal features of readability and language- or experience-specific ones. Our metric also correlates with an external notion of defect density. We address multiple programming languages and different length samples, and evaluate using an order of magnitude more participants than previous work, all suggesting our model is more likely to generalize.**

I. Introduction

Modern software developers spend more time maintaining and evolving existing software than writing new code [1], [2], [3]. Software *readability*, a fundamental notion related to the comprehension of text, is critical to software maintenance: reading code is a necessary first step toward maintaining it.

Much research, both recent and established, has argued that readability plays a large role in software maintenance. A well-known example is Knuth, who viewed readability as essential to his notion of Literate Programming [4]. He argued that a

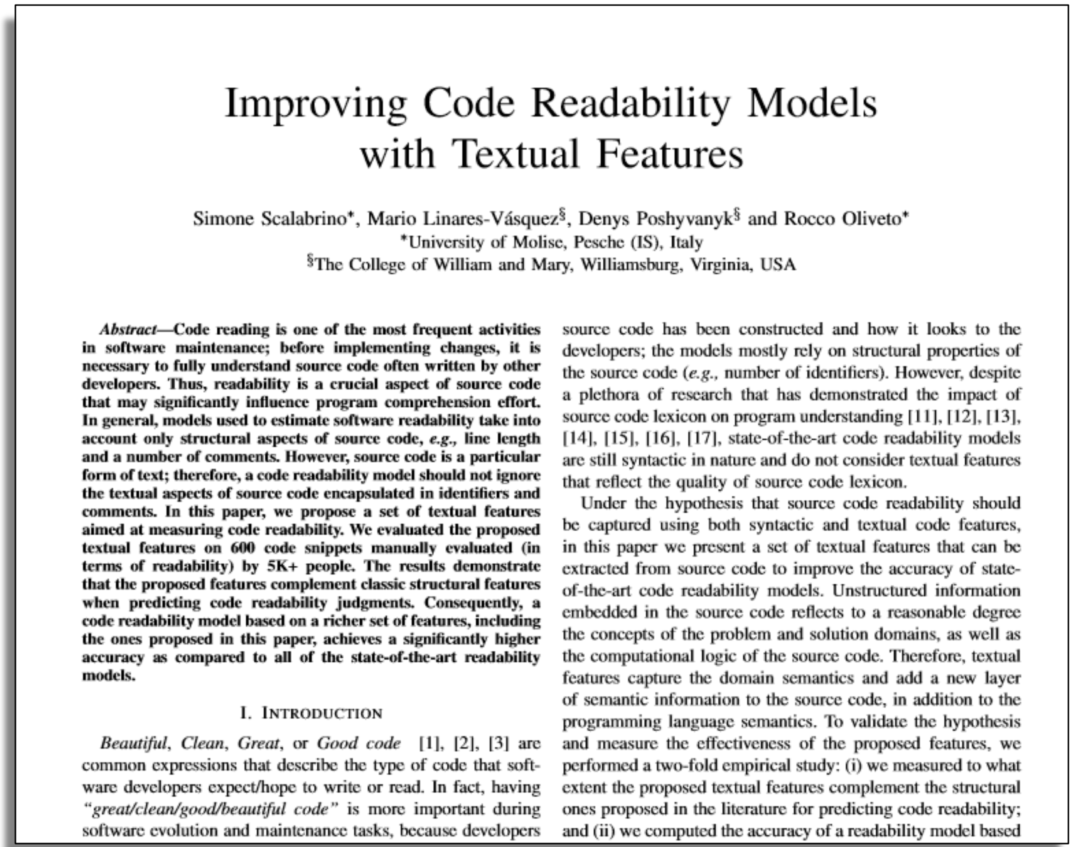
Readability Index [10] and Flesch-Kincaid Grade Level [11] are commonly used in commercial software and policies. All are based on a few simple measurements, such as the lengths of words and sentences. For example, Flesch-Kincaid is integrated into popular editors such as Microsoft Word and has become a government standard, with the US Department of Defense requiring internal and external documents to have a Flesch readability grade of 10 or below (DOD MIL-M-38784B). In the domain of software, formal metrics for readability are well-established in particular domains such as hypertext [12].

By contrast, general descriptive models of overall software readability are relatively recent, first proposed by Buse *et al.* [13] and refined by Posnett *et al.* [14]. Such models are not coding standards (cf. [15]) but are based on combinations of surface-level syntactic features such as operator counts or line lengths, aim to agree with human judgments, and have been found to correlate with external notions of software quality [16]. Such software readability models do not attempt to describe programmatic complexity (cf. [17]), which derives from system requirements and algorithms, but instead focus on readability as a controllable accidental complexity [18].

Despite the advantages of a formal notion of software readability, previous readability metrics do not adequately *generalize*. They are based on small (typically 7-line) snippets of

State of the art Readability Models

- Scalabrino et al. ICPC'16
- Model based on metrics capturing the quality of **source code lexicon**
- Evaluated on snippets from Dorn's data set



Improving Code Readability Models with Textual Features

Simone Scalabrino*, Mario Linares-Vásquez§, Denys Poshyvanyk§ and Rocco Oliveto*
*University of Molise, Pesche (IS), Italy
§The College of William and Mary, Williamsburg, Virginia, USA

***Abstract*—Code reading is one of the most frequent activities in software maintenance; before implementing changes, it is necessary to fully understand source code often written by other developers. Thus, readability is a crucial aspect of source code that may significantly influence program comprehension effort. In general, models used to estimate software readability take into account only structural aspects of source code, *e.g.,* line length and a number of comments. However, source code is a particular form of text; therefore, a code readability model should not ignore the textual aspects of source code encapsulated in identifiers and comments. In this paper, we propose a set of textual features aimed at measuring code readability. We evaluated the proposed textual features on 600 code snippets manually evaluated (in terms of readability) by 5K+ people. The results demonstrate that the proposed features complement classic structural features when predicting code readability judgments. Consequently, a code readability model based on a richer set of features, including the ones proposed in this paper, achieves a significantly higher accuracy as compared to all of the state-of-the-art readability models.**

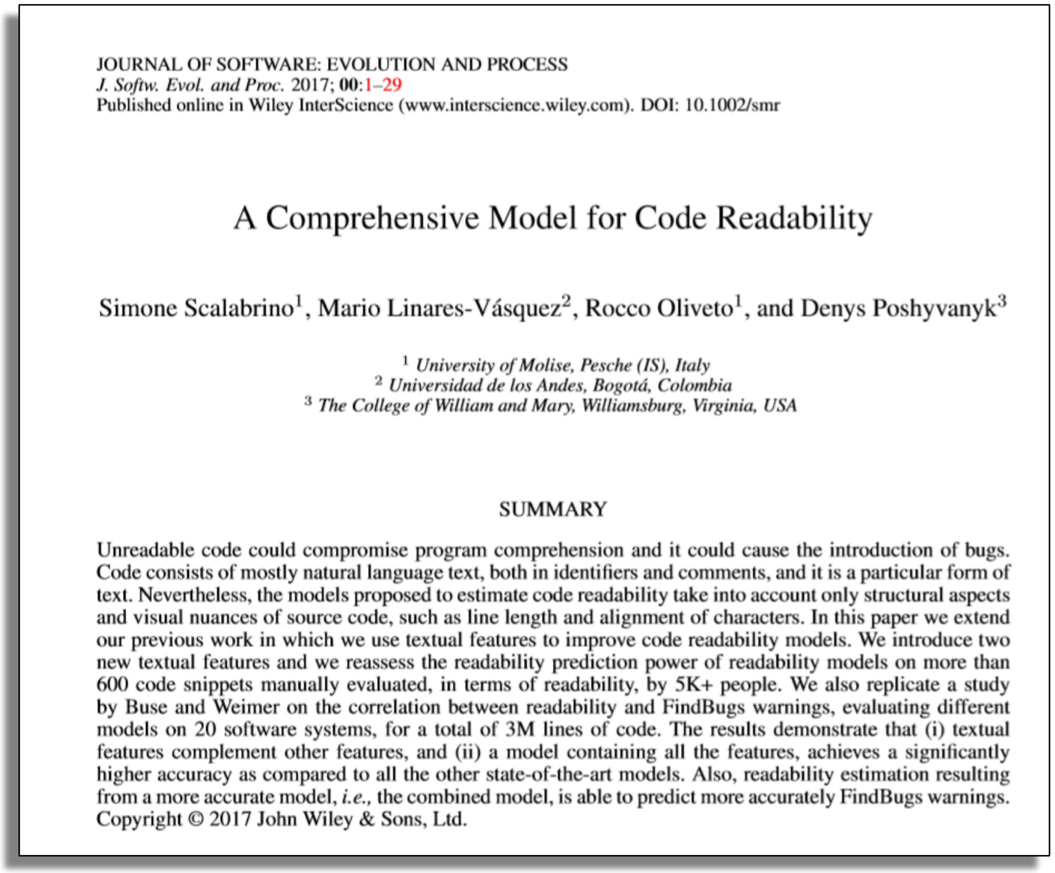
I. Introduction

Beautiful, *Clean*, *Great*, or *Good code* [1], [2], [3] are common expressions that describe the type of code that software developers expect/hope to write or read. In fact, having *"great/clean/good/beautiful code"* is more important during software evolution and maintenance tasks, because developers source code has been constructed and how it looks to the developers; the models mostly rely on structural properties of the source code (*e.g.,* number of identifiers). However, despite a plethora of research that has demonstrated the impact of source code lexicon on program understanding [11], [12], [13], [14], [15], [16], [17], state-of-the-art code readability models are still syntactic in nature and do not consider textual features that reflect the quality of source code lexicon.

Under the hypothesis that source code readability should be captured using both syntactic and textual code features, in this paper we present a set of textual features that can be extracted from source code to improve the accuracy of state-of-the-art code readability models. Unstructured information embedded in the source code reflects to a reasonable degree the concepts of the problem and solution domains, as well as the computational logic of the source code. Therefore, textual features capture the domain semantics and add a new layer of semantic information to the source code, in addition to the programming language semantics. To validate the hypothesis and measure the effectiveness of the proposed features, we performed a two-fold empirical study: (i) we measured to what extent the proposed textual features complement the structural ones proposed in the literature for predicting code readability; and (ii) we computed the accuracy of a readability model based

A comprehensive model of readability

- Scalabrino et al. JSEP 2017 extended their original work
- Combined features from multiple models: Scalabrino, Dorn, Buse & Weimer, Posnett
- Model that considers all features outperforms the state-of-the-art models on their own



JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. 2017; **00**:1–29
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

A Comprehensive Model for Code Readability

Simone Scalabrino[1], Mario Linares-Vásquez[2], Rocco Oliveto[1], and Denys Poshyvanyk[3]

[1] *University of Molise, Pesche (IS), Italy*
[2] *Universidad de los Andes, Bogotá, Colombia*
[3] *The College of William and Mary, Williamsburg, Virginia, USA*

SUMMARY

Unreadable code could compromise program comprehension and it could cause the introduction of bugs. Code consists of mostly natural language text, both in identifiers and comments, and it is a particular form of text. Nevertheless, the models proposed to estimate code readability take into account only structural aspects and visual nuances of source code, such as line length and alignment of characters. In this paper we extend our previous work in which we use textual features to improve code readability models. We introduce two new textual features and we reassess the readability prediction power of readability models on more than 600 code snippets manually evaluated, in terms of readability, by 5K+ people. We also replicate a study by Buse and Weimer on the correlation between readability and FindBugs warnings, evaluating different models on 20 software systems, for a total of 3M lines of code. The results demonstrate that (i) textual features complement other features, and (ii) a model containing all the features, achieves a significantly higher accuracy as compared to all the other state-of-the-art models. Also, readability estimation resulting from a more accurate model, *i.e.,* the combined model, is able to predict more accurately FindBugs warnings.
Copyright © 2017 John Wiley & Sons, Ltd.

Readability Metrics in Practice

- Models are evaluated by surveying external developers outside of a development environment

Could this be why there exists a discrepancy between results from research and the interpretation of metrics in practice?

Research Question #1

Are state-of-the-art readability models able to capture readability improvements in practice?

Readability Models Used

- **Scalabrino's Model.** Uses measures metrics that measure the quality of source code lexicon as a proxy for readability.[1]

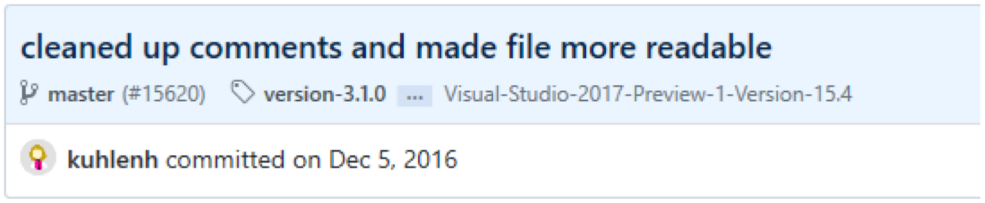
- **Dorn's Model.** Uses visual, spatial, alignment and linguistic aspects of the source code. [2]

- **Combined Model.** Proposed by Scalabrino *et al.* as a combination of multiple state of the art readability models considering both linguistic and structural aspects of the source code. [1]

[1] Simone Scalabrino, Mario Linares-Va ́squez, Rocco Oliveto, and Denys Poshyvanyk. A comprehensive model for code readability. *Journal of Software: Evolution and Process*, 30(6):e1958, 2018.
[2] Jonathan Dorn. A general software readability model. Master's thesis, University of Virginia, 2012.

Methodology: Data Collection

- **63** engineered open source Java projects from GitHub
- Mined **548** commits making readability improvements, using keyword matching (readable, easier to read… etc)
- Saved version of file before readability improvement was made and after



cleaned up comments and made file more readable

master (#15620) version-3.1.0 … Visual-Studio-2017-Preview-1-Version-15.4

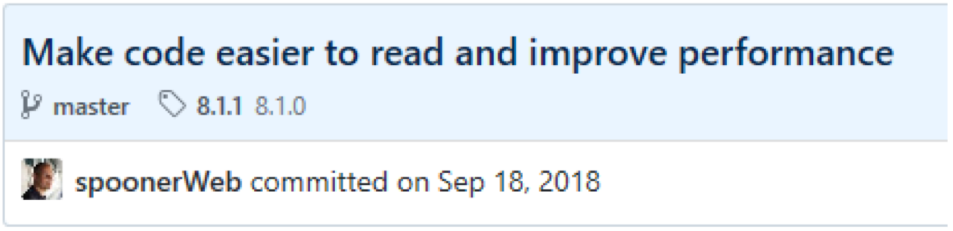
kuhlenh committed on Dec 5, 2016

Methodology: Data Collection

- Mined commits that do not contain readability improvements from the same projects

- Manually validated to ensure the commit described readability improvements

- Multiple changes in one commit are manually detangled



Make code easier to read and improve performance

master 8.1.1 8.1.0

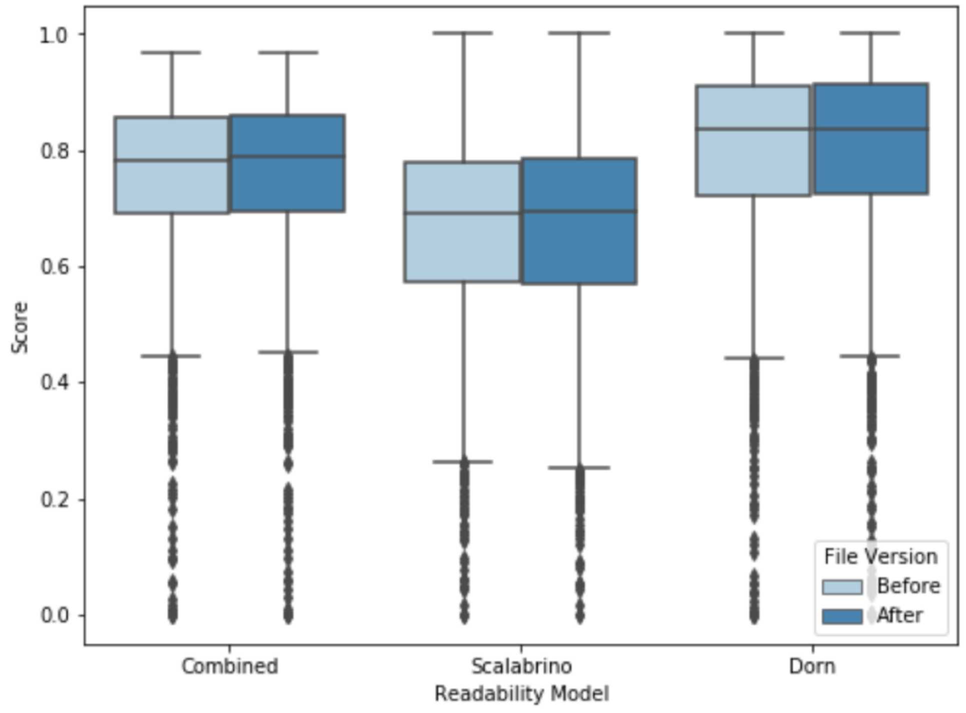
spoonerWeb committed on Sep 18, 2018

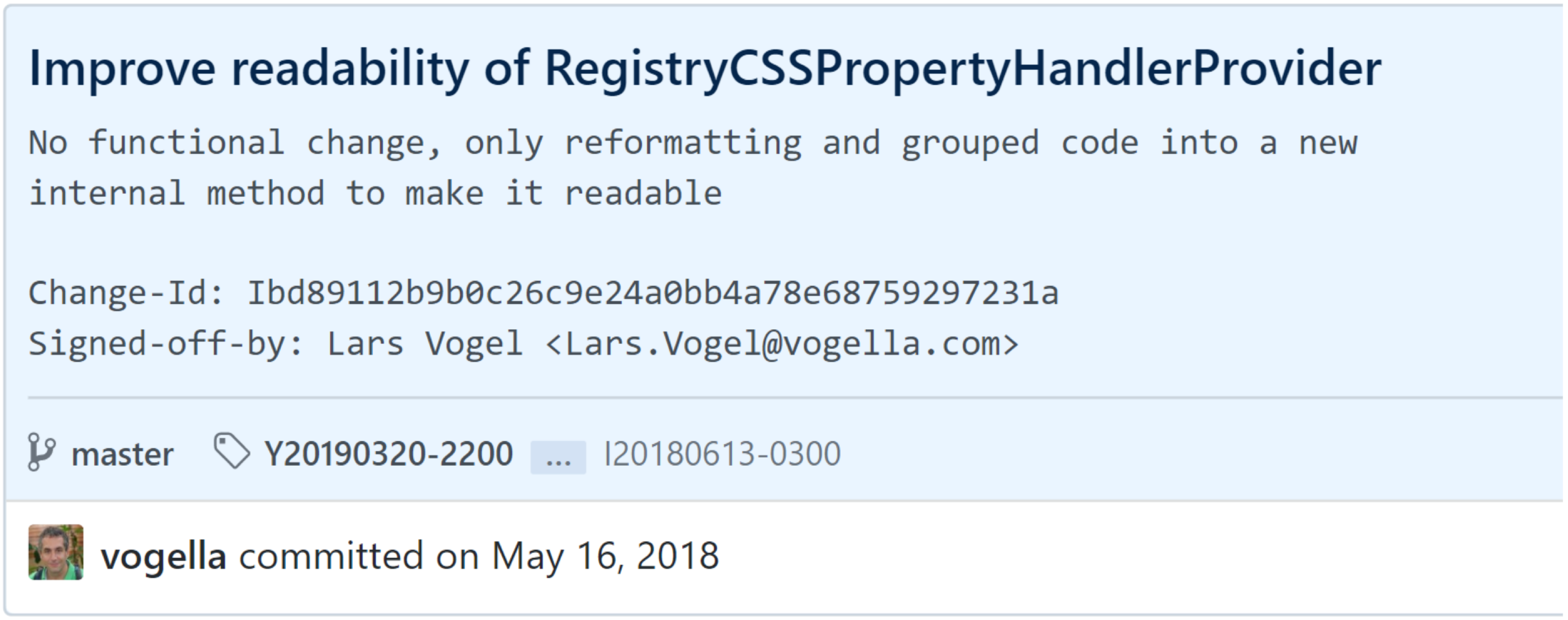
Models detect 35.7% - 40.7% of files have readability improvements

Average **72%** agreement between models

None of the models reported statistically significant changes in readability scores

Validating the results found by Pantiuchina et al.





Improve readability of RegistryCSSPropertyHandlerProvider

```
No functional change, only reformatting and grouped code into a new
internal method to make it readable

Change-Id: Ibd89112b9b0c26c9e24a0bb4a78e68759297231a
Signed-off-by: Lars Vogel <Lars.Vogel@vogella.com>
```

master Y20190320-2200 ... I20180613-0300

vogella committed on May 16, 2018

All models register slight decrease in readability

Open Questions

Do we need **models** that are **more sensitive** to types of changes made in practice?

If so, what metrics can **capture** these readability improvements made i**n practice**?

Research Question #2

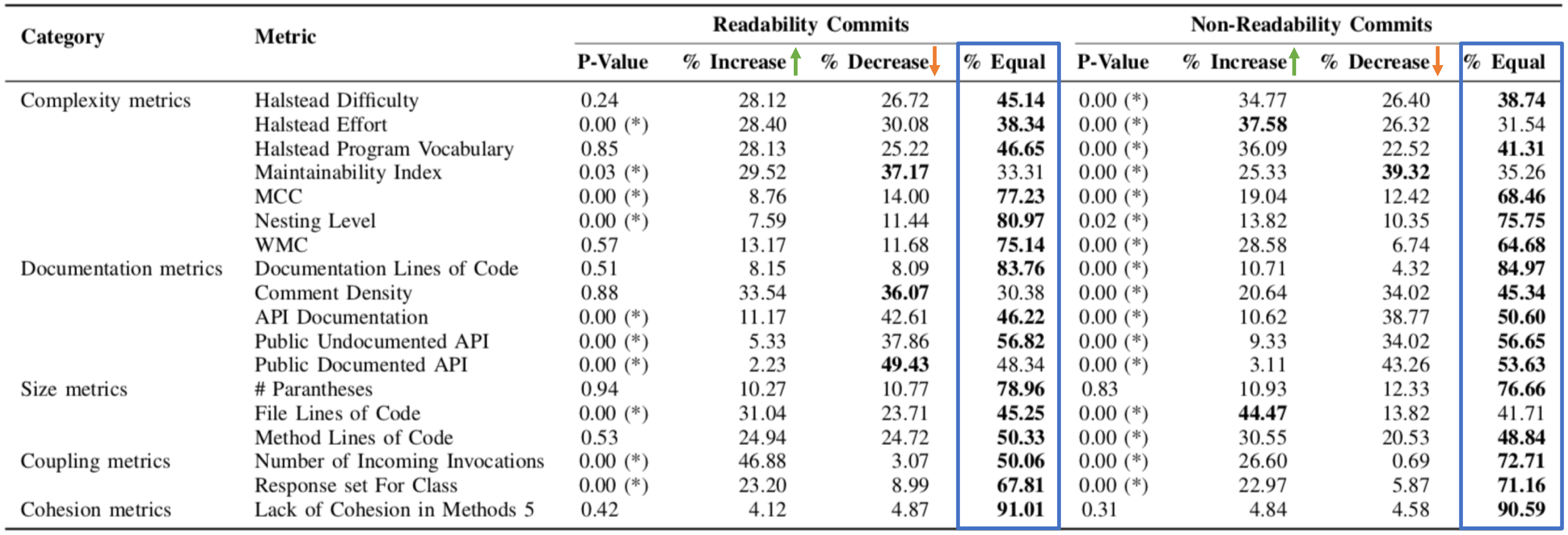
Which source code metrics **are able** capture improvement in the readability of source code, as perceived by developers **in practice**?

Static Analysis Tools: Source Meter

- Collects a variety of source code metrics
  - Cohesion
  - Complexity
  - Coupling
  - Documentation
  - Inheritance
  - Size

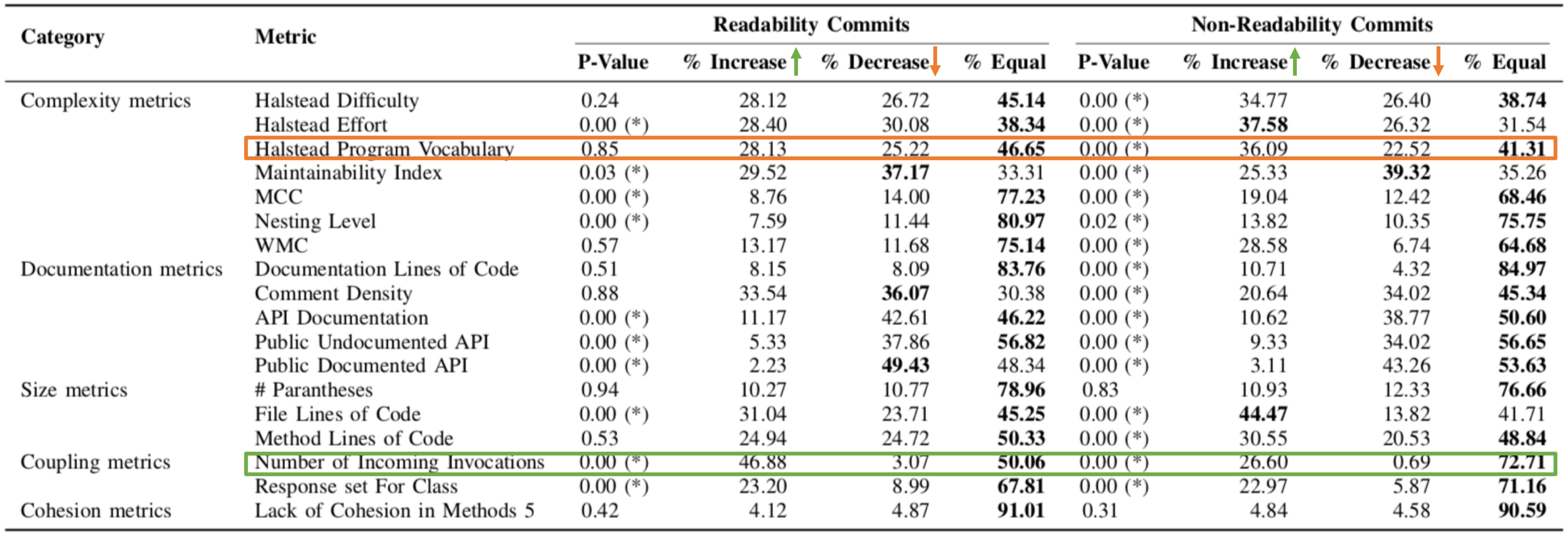
[1] https://www.sourcemeter.com/

Results: Source Meter



Category	Metric	Readability Commits				Non-Readability Commits			
		P-Value	**% Increase↑**	**% Decrease↓**	**% Equal**	**P-Value**	**% Increase↑**	**% Decrease↓**	**% Equal**
Complexity metrics	Halstead Difficulty	0.24	28.12	26.72	**45.14**	0.00 (*)	34.77	26.40	**38.74**
	Halstead Effort	0.00 (*)	28.40	30.08	**38.34**	0.00 (*)	**37.58**	26.32	31.54
	Halstead Program Vocabulary	0.85	28.13	25.22	**46.65**	0.00 (*)	36.09	22.52	**41.31**
	Maintainability Index	0.03 (*)	29.52	**37.17**	33.31	0.00 (*)	25.33	**39.32**	35.26
	MCC	0.00 (*)	8.76	14.00	**77.23**	0.00 (*)	19.04	12.42	**68.46**
	Nesting Level	0.00 (*)	7.59	11.44	**80.97**	0.02 (*)	13.82	10.35	**75.75**
	WMC	0.57	13.17	11.68	**75.14**	0.00 (*)	28.58	6.74	**64.68**
Documentation metrics	Documentation Lines of Code	0.51	8.15	8.09	**83.76**	0.00 (*)	10.71	4.32	**84.97**
	Comment Density	0.88	33.54	**36.07**	30.38	0.00 (*)	20.64	34.02	**45.34**
	API Documentation	0.00 (*)	11.17	42.61	**46.22**	0.00 (*)	10.62	38.77	**50.60**
	Public Undocumented API	0.00 (*)	5.33	37.86	**56.82**	0.00 (*)	9.33	34.02	**56.65**
	Public Documented API	0.00 (*)	2.23	**49.43**	48.34	0.00 (*)	3.11	43.26	**53.63**
Size metrics	# Parantheses	0.94	10.27	10.77	**78.96**	0.83	10.93	12.33	**76.66**
	File Lines of Code	0.00 (*)	31.04	23.71	**45.25**	0.00 (*)	**44.47**	13.82	41.71
	Method Lines of Code	0.53	24.94	24.72	**50.33**	0.00 (*)	30.55	20.53	**48.84**
Coupling metrics	Number of Incoming Invocations	0.00 (*)	46.88	3.07	**50.06**	0.00 (*)	26.60	0.69	**72.71**
	Response set For Class	0.00 (*)	23.20	8.99	**67.81**	0.00 (*)	22.97	5.87	**71.16**
Cohesion metrics	Lack of Cohesion in Methods 5	0.42	4.12	4.87	**91.01**	0.31	4.84	4.58	**90.59**

Results: Source Meter



Category	Metric	Readability Commits				Non-Readability Commits			
		P-Value	% Increase↑	% Decrease↓	% Equal	P-Value	% Increase↑	% Decrease↓	% Equal
Complexity metrics	Halstead Difficulty	0.24	28.12	26.72	**45.14**	0.00 (*)	34.77	26.40	**38.74**
	Halstead Effort	0.00 (*)	28.40	30.08	**38.34**	0.00 (*)	**37.58**	26.32	31.54
	Halstead Program Vocabulary	0.85	28.13	25.22	**46.65**	0.00 (*)	36.09	22.52	**41.31**
	Maintainability Index	0.03 (*)	29.52	**37.17**	33.31	0.00 (*)	25.33	**39.32**	35.26
	MCC	0.00 (*)	8.76	14.00	**77.23**	0.00 (*)	19.04	12.42	**68.46**
	Nesting Level	0.00 (*)	7.59	11.44	**80.97**	0.02 (*)	13.82	10.35	**75.75**
	WMC	0.57	13.17	11.68	**75.14**	0.00 (*)	28.58	6.74	**64.68**
Documentation metrics	Documentation Lines of Code	0.51	8.15	8.09	**83.76**	0.00 (*)	10.71	4.32	**84.97**
	Comment Density	0.88	33.54	**36.07**	30.38	0.00 (*)	20.64	34.02	**45.34**
	API Documentation	0.00 (*)	11.17	42.61	**46.22**	0.00 (*)	10.62	38.77	**50.60**
	Public Undocumented API	0.00 (*)	5.33	37.86	**56.82**	0.00 (*)	9.33	34.02	**56.65**
	Public Documented API	0.00 (*)	2.23	**49.43**	48.34	0.00 (*)	3.11	43.26	**53.63**
Size metrics	# Parantheses	0.94	10.27	10.77	**78.96**	0.83	10.93	12.33	**76.66**
	File Lines of Code	0.00 (*)	31.04	23.71	**45.25**	0.00 (*)	**44.47**	13.82	41.71
	Method Lines of Code	0.53	24.94	24.72	**50.33**	0.00 (*)	30.55	20.53	**48.84**
Coupling metrics	Number of Incoming Invocations	0.00 (*)	46.88	3.07	**50.06**	0.00 (*)	26.60	0.69	**72.71**
	Response set For Class	0.00 (*)	23.20	8.99	**67.81**	0.00 (*)	22.97	5.87	**71.16**
Cohesion metrics	Lack of Cohesion in Methods 5	0.42	4.12	4.87	**91.01**	0.31	4.84	4.58	**90.59**

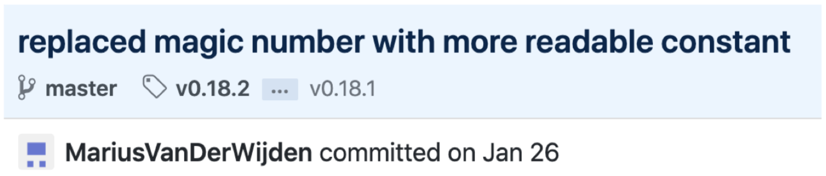
Some metrics such as Number of Incoming Invocations can be used to **complement** readability models

Research Question #3

What **types** of **changes** do developers perform during readability improvements?

checkstyle

- Checks source code adherence to configurable rules
- Used the two style configurations provided by checkstyle: google and sun checks
- Enabled a check for magic numbers

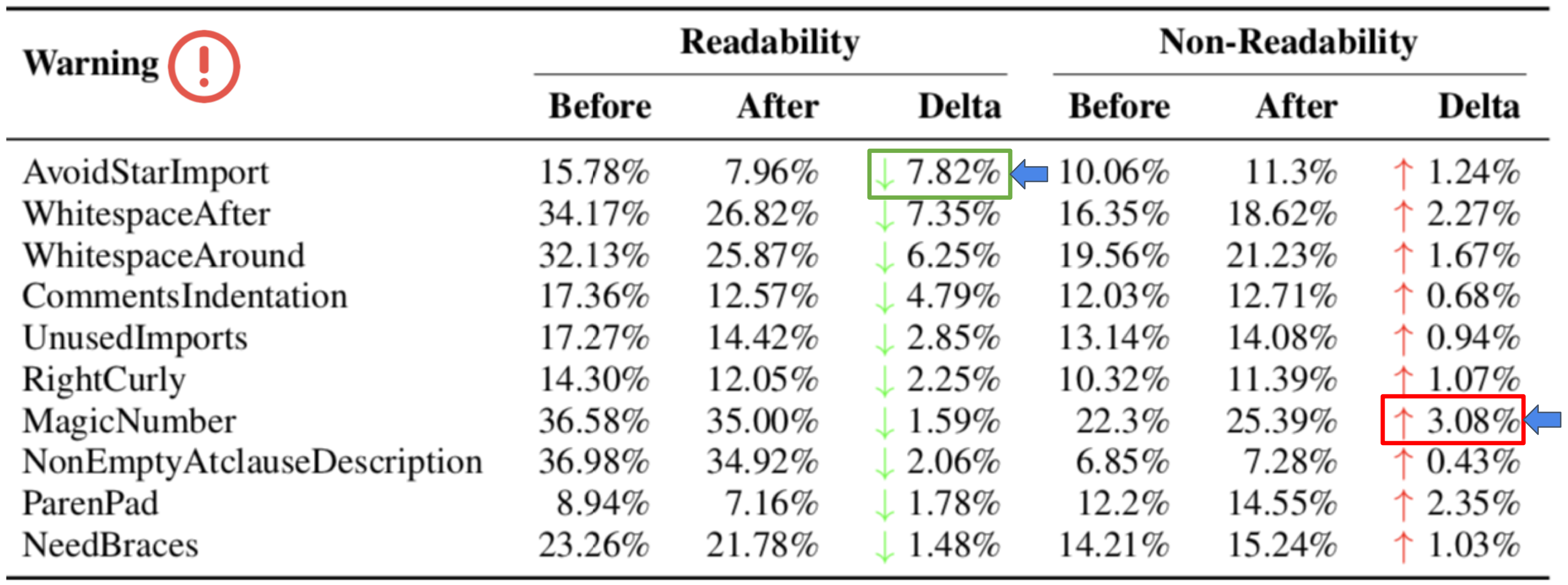

replaced magic number with more readable constant

master v0.18.2 ... v0.18.1

MariusVanDerWijden committed on Jan 26

[1] http://checkstyle.sourceforge.net/

Results: Checkstyle



Warning	Readability			Non-Readability		
	Before	After	Delta	Before	After	Delta
AvoidStarImport	15.78%	7.96%	↓ 7.82%	10.06%	11.3%	↑ 1.24%
WhitespaceAfter	34.17%	26.82%	↓ 7.35%	16.35%	18.62%	↑ 2.27%
WhitespaceAround	32.13%	25.87%	↓ 6.25%	19.56%	21.23%	↑ 1.67%
CommentsIndentation	17.36%	12.57%	↓ 4.79%	12.03%	12.71%	↑ 0.68%
UnusedImports	17.27%	14.42%	↓ 2.85%	13.14%	14.08%	↑ 0.94%
RightCurly	14.30%	12.05%	↓ 2.25%	10.32%	11.39%	↑ 1.07%
MagicNumber	36.58%	35.00%	↓ 1.59%	22.3%	25.39%	↑ 3.08%
NonEmptyAtclauseDescription	36.98%	34.92%	↓ 2.06%	6.85%	7.28%	↑ 0.43%
ParenPad	8.94%	7.16%	↓ 1.78%	12.2%	14.55%	↑ 2.35%
NeedBraces	23.26%	21.78%	↓ 1.48%	14.21%	15.24%	↑ 1.03%

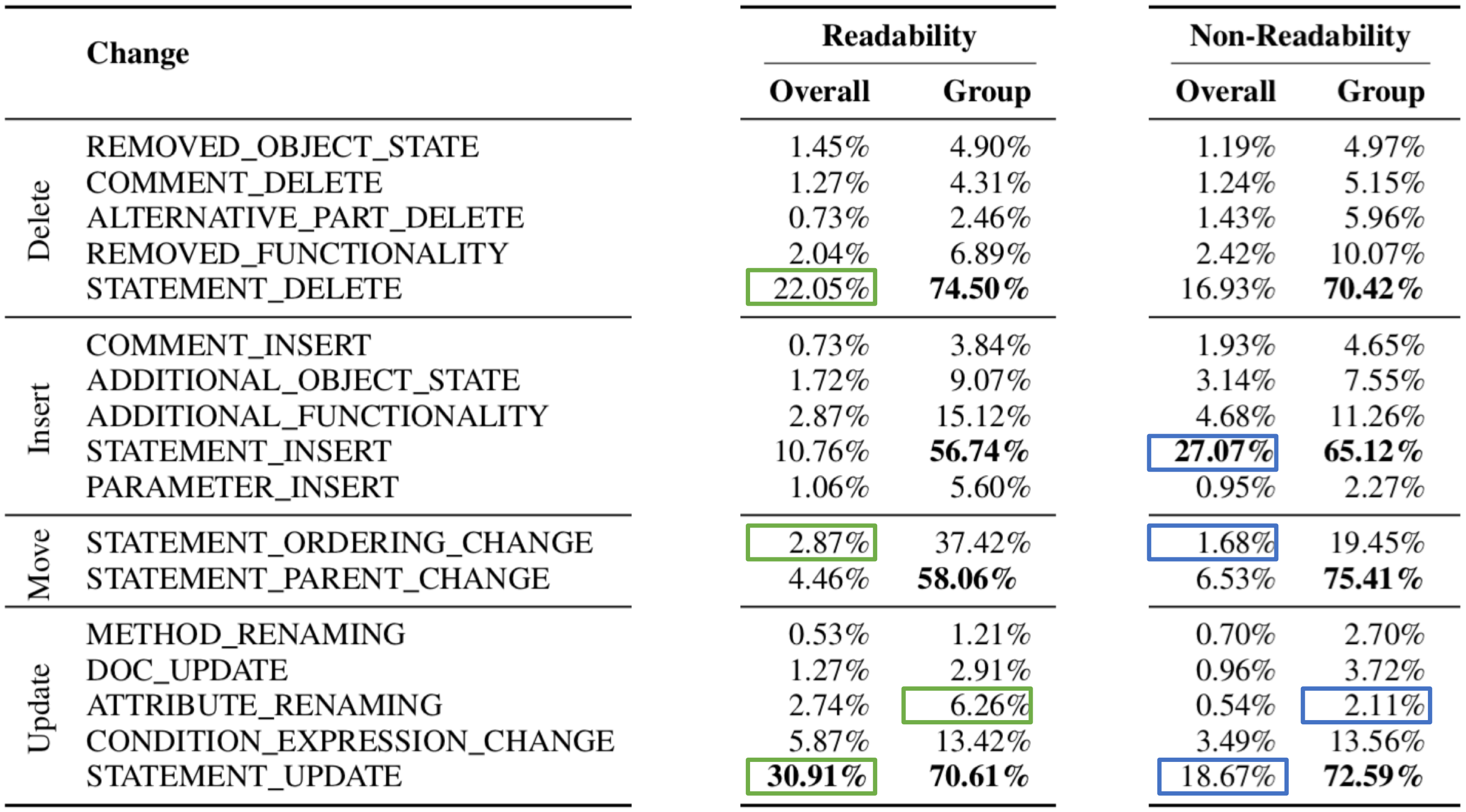
- Readability commits **fix problems** that pertain to imports, white spaces, and braces
- Non-readability commits **introduce** these problems

ChangeDistiller

- Categorizes statement level changes in source code
- Can detect 41 changes in 4 categories: move, update, insert and delete.
  - Comment Insert
  - Attribute Renaming
  - Statement Ordering Change

[1] https://bitbucket.org/sealuzh/tools-changedistiller/wiki/Home

Results: ChangeDistiller



	Change	Readability		Non-Readability	
		Overall	Group	Overall	Group
Delete	REMOVED_OBJECT_STATE	1.45%	4.90%	1.19%	4.97%
	COMMENT_DELETE	1.27%	4.31%	1.24%	5.15%
	ALTERNATIVE_PART_DELETE	0.73%	2.46%	1.43%	5.96%
	REMOVED_FUNCTIONALITY	2.04%	6.89%	2.42%	10.07%
	STATEMENT_DELETE	22.05%	**74.50%**	16.93%	**70.42%**
Insert	COMMENT_INSERT	0.73%	3.84%	1.93%	4.65%
	ADDITIONAL_OBJECT_STATE	1.72%	9.07%	3.14%	7.55%
	ADDITIONAL_FUNCTIONALITY	2.87%	15.12%	4.68%	11.26%
	STATEMENT_INSERT	10.76%	**56.74%**	**27.07%**	**65.12%**
	PARAMETER_INSERT	1.06%	5.60%	0.95%	2.27%
Move	STATEMENT_ORDERING_CHANGE	2.87%	37.42%	1.68%	19.45%
	STATEMENT_PARENT_CHANGE	4.46%	**58.06%**	6.53%	**75.41%**
Update	METHOD_RENAMING	0.53%	1.21%	0.70%	2.70%
	DOC_UPDATE	1.27%	2.91%	0.96%	3.72%
	ATTRIBUTE_RENAMING	2.74%	6.26%	0.54%	2.11%
	CONDITION_EXPRESSION_CHANGE	5.87%	13.42%	3.49%	13.56%
	STATEMENT_UPDATE	**30.91%**	**70.61%**	18.67%	**72.59%**

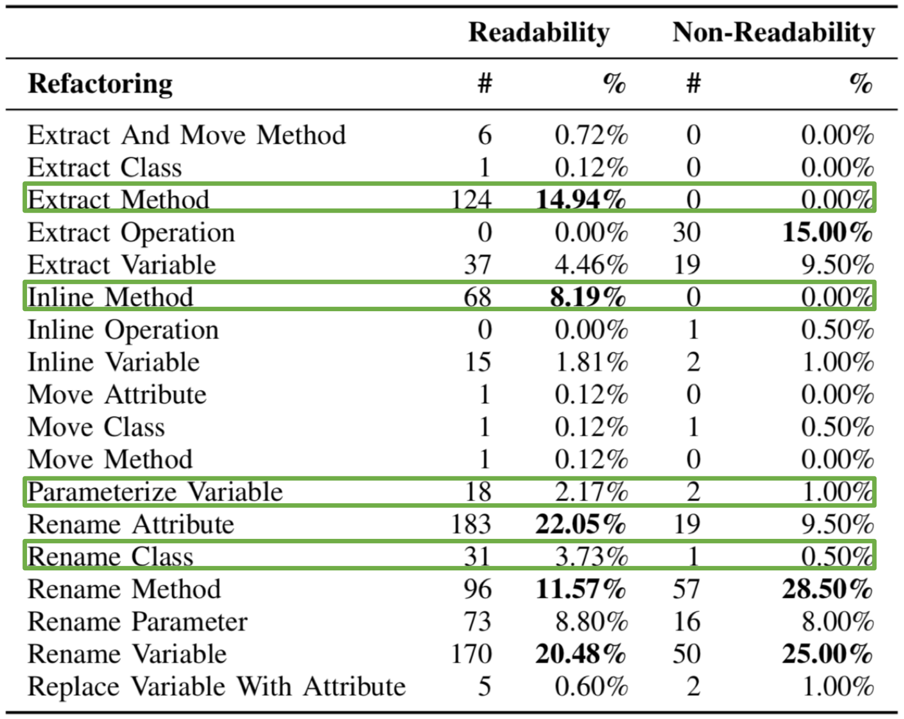
RefactoringMiner

Detects refactorings across the history of a Java project and provides a high level overview of the types of changes being made.

- Extract Method
- Parameterize Variable
- Move Method

[1] https://github.com/tsantalis/RefactoringMiner

Results: RefactoringMiner



	Readability		**Non-Readability**	
Refactoring	**#**	**%**	**#**	**%**
Extract And Move Method	6	0.72%	0	0.00%
Extract Class	1	0.12%	0	0.00%
Extract Method	124	**14.94%**	0	0.00%
Extract Operation	0	0.00%	30	**15.00%**
Extract Variable	37	4.46%	19	9.50%
Inline Method	68	**8.19%**	0	0.00%
Inline Operation	0	0.00%	1	0.50%
Inline Variable	15	1.81%	2	1.00%
Move Attribute	1	0.12%	0	0.00%
Move Class	1	0.12%	1	0.50%
Move Method	1	0.12%	0	0.00%
Parameterize Variable	18	2.17%	2	1.00%
Rename Attribute	183	**22.05%**	19	9.50%
Rename Class	31	3.73%	1	0.50%
Rename Method	96	**11.57%**	57	**28.50%**
Rename Parameter	73	8.80%	16	8.00%
Rename Variable	170	**20.48%**	50	**25.00%**
Replace Variable With Attribute	5	0.60%	2	1.00%

Refactorings such as extract method, inline method, parameterize variable, and rename class are **specific** to readability commits, and **almost non-existent** for non-readability commits

Takeaways

- There is a need for models that can capture readability improvements in practice
- Some metrics considered in existing readability models do not capture readability improvements in practice
- In addition to static analysis tools, warnings from code style tools and types of changes made could be considered for readability models

Thank You!