CloneCognition: Machine Learning Based Code Clone Validation Tool

Golam Mostaeen
University of Saskatchewan
Saskatoon, Canada
golam.mostaeen@usask.ca

Jeffrey Svajlenko
University of Saskatchewan
Saskatoon, Canada
jeff.svajlenko@usask.ca

Banani Roy
University of Saskatchewan
Saskatoon, Canada
banani.roy@usask.ca

Chanchal K. Roy
University of Saskatchewan
Saskatoon, Canada
chanchal.roy@usask.ca

Kevin A. Schneider
University of Saskatchewan
Saskatoon, Canada
kevin.schneider@usask.ca

ABSTRACT

A code clone is a pair of similar code fragments, within or between software systems. To detect each possible clone pair from a software system while handling the complex code structures, the clone detection tools undergo a lot of generalization of the original source codes. The generalization often results in returning code fragments that are only coincidentally similar and not considered clones by users, and hence requires manual validation of the reported possible clones by users which is often both time-consuming and challenging. In this paper, we propose a machine learning based tool 'CloneCognition' (*Open Source Codes:* https://github.com/pseudoPixels/CloneCognition ; *Video Demonstration:* https://www.youtube.com/watch?v=KYQjmdr8rsw) to automate the laborious manual validation process. The tool runs on top of any code clone detection tools to facilitate the clone validation process. The tool shows promising clone classification performance with an accuracy of up to 87.4%. The tool also exhibits significant improvement in the results when compared with state-of-the-art techniques for code clone validation.

CCS CONCEPTS

• **Software and its engineering** → *Maintaining software.*

KEYWORDS

Code Clones, Validation, Machine Learning, Artificial Neural Network, Clone Management.

ACM Reference Format:
Golam Mostaeen, Jeffrey Svajlenko, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. 2019. CloneCognition: Machine Learning Based Code Clone Validation Tool. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3338906.3341182

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-5572-8/19/08…$15.00
https://doi.org/10.1145/3338906.3341182

1 INTRODUCTION

Studies show that code clone is one of the major reasons for creation and propagation of software bugs throughout the system, inflating software maintenance costs [6, 13]. Recent research shows that a software system on average contains around 7% to 23% of clones of its overall code base [11, 14]. Detection of such code clones thus provides a better understanding of the software systems in addition to facilitating the overall software maintenance task [6, 13]. Hence, extensive research in this domain resulted in around 200 different proposed tools and techniques [1, 2, 4, 7, 13, 15, 19] for code clone detection until 2017 [18]. In general, these tools target to maximize returning all possible pairs of code clones from a given software system [6, 13].

With the target of detecting any possible pair of code clone while comprehending all possible source code modifications, code clone detection tools undergo a lot of generalization of the original source code, such as pretty-printing [13, 15], normalization of the identifiers [7, 15], forming syntax tree [10] and so on, -prior to applying the corresponding matching algorithms. These generalizations hence result in clone detection tools to return pairs to code fragments that are only coincidentally similar and not considered essentially clones by users [9, 12, 21]. While several factors, such as templating, Library/API calling protocols, general language or algorithmic idioms and so on in the implementation process creates enough similarities among code fragments to be detected by clone detection tools, they are not often considered clones by users [9, 21]. For these considerations, users often need to manually validate the reported code clones from a clone detection tool [21]. The manual validation often becomes a time-consuming and laborious work, as the software systems evolve over time [16].

In this paper, we present a machine learning based clone validation tool *CloneCognition* to automate the validation process. The tool uses an *Artificial Neural Network* (ANN) classifier to learn and predict the human validation pattern towards automating the code clone validation problem. The classifier model for validation is trained from manually validated clone sets from IJaDataset 2.0 [3] - a large inter-project dataset of open-source projects. The classifier model learns by mimicking the human validation patterns for feature extraction and shows an accuracy of up to 87.4% for automatic validation. For testing the generality of classification, the model was rigorously evaluated against different experimental setups, such as multiple code clone detection tools, artificial code clones, different open source projects -where it shows promising results. The proposed clone classification approach also shows better performance when compared against existing state-of-the-art techniques for code clone validation [21]. While in this paper, we focus on the tool aspect of the proposed automatic clone validation approach, we refer the readers to our original paper [12] for details.

Without reinventing the wheel of clone detection techniques, the tool runs on top of any code clone detection tools to automate the validation process. The reported possible clones from a clone detection tools are provided to CloneCognition as input, where they are validated against the classification model. For every possible pair of clones, the tool calculates the corresponding probabilistic score for automatic validation and final report generation. While the classification model is trained on diverse and evolving manually validated dataset by multiple expert judges, CloneCognition also provides a framework for manual validation of code clones towards building a case-specific dataset for model training and validation by the tool [12].

CloneCognition follows a service-oriented architecture so that it becomes compatible with any existing clone detection tool irrespective of the language or platform differences in the implementation of the tools via simple service call invocation (e.g., *RESTful* services). The tool is publicly accessible from a cloud server[1], and also a local instance of it can be obtained from GitHub[2].

2 RELATED WORKS

Management of the large-set of detected clones has been one of the active research topics in the code clone detection domain over the last few years [17, 20, 21]. Yang et al. studied the similar problem for code clone validation in their work - FICA [21]. Their work leverages '*Term-Frequency - Inverse Document Frequency*'- (TF-IDF) for the purpose for Type 2 clone validation. However, in our proposed method we targeted the validation of even beyond Type 2 clones, having more complex structures such as Type 3. We conducted several evaluation studies for comparison, where our proposed method shows better and promising results (e.g., as presented in Section 5).

Svajlenko et al. [17] used unsupervised machine learning for clustering similar code clones. Kapser and Godfrey [8] studied on the categorization of detected clones in hierarchy files and directories based on location proximity. While the total number of clones to be analyzed by those methods still remain same, our method in contrast targets on user-specific classification for clone filtering.

Besides, several visualization techniques have been proposed for clone management, such as Scatter plot [7], Hasse diagram [5], an aspect browser-like view [20], hierarchical graphs [4] and so on. Note that, our proposed tool can further aid in such clone management by providing additional dimensions, such as clone validation score.

3 CloneCognition WORKING METHODOLOGY

In this section, we present a brief discussion of the working procedure of the tool, while we refer to our original paper for detail studies envisioning the tool [12].

3.1 Clone Validation with Machine Learning

3.1.1 Dataset Formulation. The machine learning model of CloneCognition is trained on the clone sets of IJaDataset 2.0 [3] - a large inter-project dataset of open-source projects. For the machine learning model generalization, CloneCognition used five different publicly available and state-of-the-art tools- NiCad, Deckard, iClones, CCFinderX and SourcererCC for detecting clones independently from IJaDataset 2.0.

3.1.2 Feature Extraction. We conducted several experiments and studies on the feature selection and extraction for the machine learning classifier. While in this paper we focus on the tool aspects of our research studies on clone validation, we refer the readers to the original paper for details presentation on the feature sets, such as feature selection, classification data distribution in terms of features, feature scores, cluster information and so on [12]. Existing works on clone validation only address up to Type 2 clones and uses token level sequence matching, such as *n-gram* [21]. However, with an attempt to extending the validation process beyond Type 2, we use three level of normalization of the fragments for similarity feature extractions - *line similarities with Type 1, 2 and 3 levels of normalization, token level similarities in Type 1, 2 and 3 level of normalization* [12]. Besides, from our investigation study we found that, human judges in addition to the similarity measures assess different visual structural attributes, such as - *fragment size differences -with different consideration on larger versus smaller fragments, unmatched braces, function intersected* -which are also considered in the feature set of CloneCognition.

3.1.3 Training the Machine Learning Classifier Model. CloneCognition works on top of the reported code clones from a code clone detection tool. The reported clones from a clone detection tool are sent to CloneCognition for validation and report regeneration based on the validation result from the machine learning model. CloneCognition uses *Artificial Neural Network* (ANN) model, $f(\mathbf{x})$ for learning the human validation patterns towards automating the clone validation task (e.g., Fig. 1(b)). For a given test clone pair set, CloneCognition extracts features set, $\mathbf{x}_t$ to activate on the learned model function, $\hat{\mathbf{y}}_t = f(\mathbf{x}_t)$. ANN being a probabilistic classifier predicts and returns, $\hat{\mathbf{y}}_t$- the vector containing the probability of a submitted clone pair for being true positive and false positive e.g., as in Eq. 1.

$$\hat{\mathbf{y}}_t = f(\mathbf{x}_t) = (Pr[\mathbf{y}_t = (1,0)], Pr[\mathbf{y}_t = (0,1)]) \tag{1}$$

From our several studies on classifier selection and their corresponding configurations, we found that ANN performs better with one hidden layer comprising of 107 nodes, softmax activation function for the output layer and converges within a range of 500 to 600 epochs giving an accuracy of 87.4% [12].

3.1.4 Classification Decision Configuration and Clone Validation Report Generation. The trained classifier, $\hat{\mathbf{y}}_t = f(\mathbf{x_t})$ assigns probability values- $Pr[\mathbf{y}_t = (1,0)]$ and $Pr[\mathbf{y}_t = (0,1)]$ -respectively for the true positive and false positive clone classes. Users can tune the clone classification decision by setting different values of- $\gamma[0,1]$, which is used by CloneCognition to make the prediction decision. A test clone pair is validated as true positive, if $Pr[\mathbf{y}_t = (1,0)] \geq \gamma$. The default value of γ is 0.5, such that CloneCognition makes true

[1] http://p2irc-cloud.usask.ca/cloneCognition
[2] https://github.com/pseudoPixels/cloneCognition

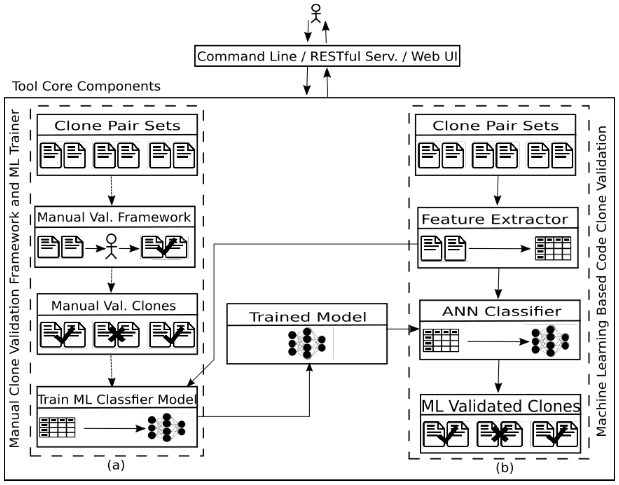


Figure 1: High-Level Workflow of CloneCognition.

positive prediction if, $Pr[\mathbf{y}_t = (1,0)] \geq Pr[\mathbf{y}_t = (0,1)]$. CloneCognition calculates and returns several reports on the clone validation statistics, such as precision, true positive counts, false positive counts, true positive ratio and false positive ratio. The validation responses are also recorded as CSV format for the corresponding test code clone pairs for later use.

3.2 High-Level Architecture of the Tool with its Components

Fig. 1 demonstrates the core components and schematic diagram of *CloneCognition*. The components can be classified into two major groups - manual validation framework for maintaining the cycle of supervised learning (i.e., Fig. 1 (a)) and clone validation with machine learning models (i.e., Fig. 1 (b)).

Towards creating a custom training dataset for case-specific scenarios [12, 21] or enriching the existing training dataset, CloneCognition provides a framework for a cycle of supervised learning (e.g., Fig. 1 (a)). Here, user-specified new clone pair sets are iteratively presented to the users for validation responses towards building manually validated dataset. The labeled dataset is later used for corresponding feature extraction and training the CloneCognition model (e.g., as presented in Section 3.1.2 and 3.1.3 respectively). The trained model is finally saved (e.g., after model serialization, such as *pickling*) for later automatic clone classification.

As presented in Section 3.1.4, Fig. 1 (b) demonstrates the different components of the tools for automatic clone validation report levering the trained machine learning model. For new set of clone pairs, *CloneCognition* uses feature extraction module for prepare the feature vector, which is fed to the trained model for generating the machine learning validated clone reports.

The tool follows a Service-Oriented architecture, so that the components can be individually accessed via *RESTful* API or Web UI in addition to command line as illustrated in the figure. We present brief discussion on the corresponding services and usage of the tool in the following section.

4 CloneCognition USAGE

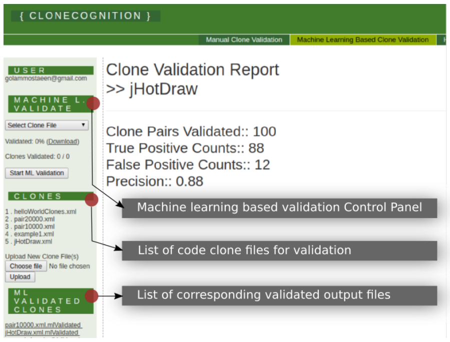
While the architecture allows accessing the technical features from command lines, we developed a user interface on top of the RESTful services to ensure easier manipulation and visualization of the validated clones. In this section, we present the major technical features provided by the CloneCognition tool.

4.1 Automatic Clone Validation using Machine Learning

For automatic validation, the reported sets of code clone pairs from a clone detection tool are provided to CloneCognition as input (for example, uploaded from the user interface as presented in Fig. 2). The tool extracts the feature sets for clone classification from the input sets of clone pairs. The extracted feature set for a corresponding clone pair is then activated against the trained machine learning model. CloneCognition sends the clone pairs with the corresponding classification probability values, as corresponding response for the clone validation requests. A code clone pair is validated as either true or false positive based on the probabilities assigned by the classifier. As noticeable from the figure (i.e., Fig. 2), the clone validation can also be tuned by the user set threshold for validation decision making of the classifier [12]. In addition to the corresponding clone validation scores, CloneCognition also returns the validation overall validation statistics, such as total true positive clones, precision and so on of the input reported possible clone pair sets, which are often useful for the evaluation of clone detection tools [21].

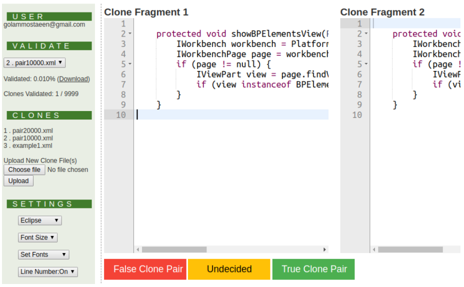
4.2 Manual Validation and Model Building

Fig. 3 demonstrates the manual validation framework of CloneCognition. The reported code clones from a clone detection tool are imported to CloneCognition, where the possible code fragments of corresponding clone pairs in iterations are presented to the user in a comparison view. Users mark a presented clone pair as any of - *'False Clone Pair'*, *'True Clone Pair'* or *'Undecided'* from their manual judgment of the presented clone pairs. CloneCognition records the manual responses for the corresponding clone pairs, -which are exportable as *.csv* format from the tool. The manual validation framework by the tool provides two folds advantages. First, the manual responses can be used for enriching the clone validation


Clone Validation Report
>> jHotDraw

Clone Pairs Validated:: 100
True Positive Counts:: 88
False Positive Counts:: 12
Precision:: 0.88

Figure 2: CloneCognition User Interface- Automatic Clone Validation with Machine Learning.



USER
golammostaeen@gmail.com

VALIDATE
2 . pair10000.xml ▾
Validated: 0.010% (Download)
Clones Validated: 1 / 9999

CLONES
1 . pair20000.xml
2 . pair10000.xml
3 . example1.xml

Upload New Clone File(s)
Choose file No file chosen
Upload

SETTINGS
Eclipse ▾
Font Size ▾
Set Fonts ▾
Line Number:On ▾

Clone Fragment 1

```

    protected void showBPElementsView(F
        IWorkbench workbench = Platforr
        IWorkbenchPage page = workbencl
        if (page != null) {
            IViewPart view = page.find\
            if (view instanceof BPEleme
        }
    }

```

Clone Fragment 2

```

    protected voic
        IWorkbencl
        IWorkbencl
        if (page !
            IViewF
            if (vi
        }
    }

```

False Clone Pair | Undecided | True Clone Pair

Figure 3: CloneCognition User Interface- Manual Validation and Model Building.

benchmark dataset, and hence evaluating the result quality of a clone detection tool. Second, the manual validation responses are used by CloneCognition for further improving the machine learning model in a cycle of supervised learning.

CloneCognition also supports training the machine learning classifier towards improving the model from the obtained manual validation patterns. The responses from the newly validated clones are used by CloneCognition towards a cycle of supervised learning.

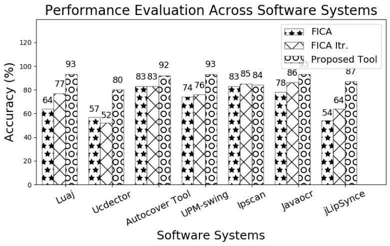


Figure 4: Comparison of CloneCognition with Related Works on Clone Validation. Validation accuracy calculated from known dataset and compared with existing methods.

5 EVALUATION

The CloneCognition classification model shows a promising clone validation accuracy of 87.4%, when evaluated using 10-fold cross validation against IJaDataset 2.0 [3]. Besides, for attaining confidence in the classification performance, the clone validation was tested for different use-cases, such as with different open source software systems, code clone detection tools, multiple users, artificial clones, existing methods and so on, - where the classification model shows promising results. For example, Fig. 4 shows the snippets of comparison of the tool with related state-of-the-art methods [21] for clone validation. The graph plot illustrates that CloneCognition outperforms the existing methods for clone validation across reported clones from different software systems. We also studied the result quality of the classification model of CloneCognition. For example, Fig. 5 shows the *Receiver Operating Characteristic* (ROC) curve of the tool. The result illustrates that CloneCognition attains up to 0.87 *Area Under ROC* (AUC), which is promising. Fig. 6 demonstrates a better result quality of CloneCognition when compared to the existing methods for code clone validation [21]. The box-plot graph illustrates that the proposed tool exhibits better overall median precision, recall, and F1-score in comparison to the existing methods of clone validation. CloneCognition also shows better consistency in its result qualities in comparison to the existing methods- where the existing methods exhibit higher variances in corresponding validation results as depicted in Fig. 6.

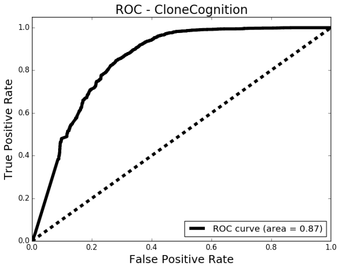


Figure 5: ROC curve of CloneCognition for clone classification.

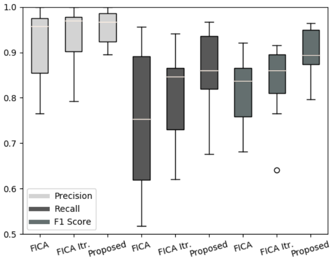


Figure 6: Result quality comparison with related works.

6 CONCLUSION

In this paper, we proposed a machine learning based tool CloneCognition for automating the time-consuming and laborious works of code clone validation. The tool shows promising clone classification results with an accuracy of up to 87.4% and also outperforms the existing state-of-the-art clone validation techniques. The CloneCognition tool offers several advantages in terms of code clone validation, such as i) integration with any clone detection tool as validation layer, ii) easier validation process from simple user interface, iii) precision measurement of clone detection tools, iv) possibilities of important knowledge discoveries and visualization in terms validated clone sets and corresponding validation scores.

ACKNOWLEDGMENT

This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), and by two Canada First Research Excellence Fund (CFREF) grants coordinated by the Global Institute for Food Security (GIFS) and the Global Institute for Water Security (GIWS).

REFERENCES

[1] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. 2007. Comparison and evaluation of clone detection tools. *IEEE Transactions on software engineering* 33, 9 (2007).

[2] Ekwa Duala-Ekoko and Martin P Robillard. 2007. Tracking code clones in evolving software. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on.* IEEE, 158–167.

[3] Ambient Software Evoluton Group. [n.d.]. IJaDataset 2.0. http://secold.org/projects/seclone.

[4] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. 2007. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th international conference on Software Engineering.* IEEE Computer Society, 96–105.

[5] J Howard Johnson. 1994. Visualizing textual redundancy in legacy source. In *Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research.* IBM Press, 32.

[6] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. 2009. Do code clones matter?. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on.* IEEE, 485–495.

[7] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. 2002. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering* 28, 7 (2002), 654–670.

[8] Cory Kapser and Michael W Godfrey. 2004. Aiding comprehension of cloning through categorization. In *Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of.* IEEE, 85–94.

[9] Cory Kapser and Michael W Godfrey. 2006. " Cloning considered harmful" considered harmful. In *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on.* IEEE, 19–28.

[10] Rainer Koschke, Raimar Falke, and Pierre Frenzel. 2006. Clone detection using abstract syntax suffix trees. In *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on.* IEEE, 253–262.

[11] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. 2017. CCLearner: A Deep Learning-Based Clone Detection Approach. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on.* IEEE, 249–260.

[12] G. Mostaeen, Jeffrey Svajlenko, Banani Roy, Chanchal K. Roy, and K. Schneider. 2018. On the Use of Machine Learning Techniques Towards the Design of Cloud Based Automatic Code Clone Validation Tools. In *Source Code Analysis and Manipulation, 2018. SCAM 2018. 18th IEEE International Working Conference on.* IEEE.

[13] Chanchal K. Roy and James R. Cordy. 2007. A survey on software clone detection research. *Queen's School of Computing TR* 541, 115 (2007), 64–68.

[14] Chanchal K. Roy and James R. Cordy. 2008. An empirical study of function clones in open source software. In *Reverse Engineering, 2008. WCRE'08. 15th Working Conference on.* IEEE, 81–90.

[15] Chanchal K Roy and James R Cordy. 2008. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on.* IEEE, 172–181.

[16] Jeffrey Svajlenko, Judith F Islam, Iman Keivanloo, Chanchal K Roy, and Mohammad Mamun Mia. 2014. Towards a big data curated benchmark of inter-project code clones. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on.* IEEE, 476–480.

[17] Jeffrey Svajlenko and Chanchal K Roy. 2016. A Machine Learning Based Approach for Evaluating Clone Detection Tools for a Generalized and Accurate Precision. *International Journal of Software Engineering and Knowledge Engineering* 26, 09n10 (2016), 1399–1429.

[18] Jeff Thomas Svajlenko et al. 2018. *Large-Scale Clone Detection and Benchmarking.* Ph.D. Dissertation. University of Saskatchewan.

[19] Robert Tairas and Jeff Gray. 2006. Phoenix-based clone detection using suffix trees. In *Proceedings of the 44th annual Southeast regional conference.* ACM, 679–684.

[20] Robert Tairas and Jeff Gray. 2009. An information retrieval process to aid in the analysis of code clones. *Empirical Software Engineering* 14, 1 (2009), 33–56.

[21] Jiachen Yang, Keisuke Hotta, Yoshiki Higo, Hiroshi Igaki, and Shinji Kusumoto. 2015. Classification model for code clones based on machine learning. *Empirical Software Engineering* 20, 4 (2015), 1095–1125.