Clone Swarm: A Cloud Based Code-Clone Analysis Tool

Venkat Bandi*, Chanchal K. Roy† and Carl Gutwin‡
University of Saskatchewan
Saskatoon, Canada
*venkat.bandi@usask.ca, †chanchal.roy@usask.ca, ‡carl.gutwin@usask.ca

***Abstract*—A code clone is defined as a pair of similar code fragments within a software system. While code clones are not always harmful, they can have a detrimental effect on the overall quality of a software system due to the propagation of bugs and other maintenance implications. Because of this, software developers need to analyse the code clones that exist in a software system. However, despite the availability of several clone detection systems, the adoption of such tools outside of the clone community remains low. A possible reason for this is the difficulty and complexity involved in setting up and using these tools. In this paper, we present Clone Swarm, a code clone analytics tool that identifies clones in a project and presents the information in an easily accessible manner. Clone Swarm is publicly available and can mine any open-sourced GIT repository. Clone Swarm internally uses NiCad, a popular clone detection tool in the cloud and lets users interactively explore code clones using a web-based interface at multiple granularity levels (Function and Block level). Clone results are visualized in multiple overviews, all the way from a high-level plot down to an individual line by line comparison view of cloned fragments. Also, to facilitate future research in the area of clone detection and analysis, users can directly download the clone detection results for their projects. Clone Swarm is available online at clone-swarm.usask.ca. The source code for Clone Swarm is freely available under the MIT license on GitHub.**

***Index Terms*—Software Maintenance, Clone Detection, Code Clone Visualization**

I. Introduction

In software development, the practice of copying and pasting code fragments in multiple files is quite common. These code fragments are called code clones, and they can be syntactically, semantically or lexically similar. Whether cloning code is a useful practice to follow or not is a topic of debate for many software engineers, with both sides having strong arguments in favour [1] [2] [3]. Broadly speaking, however, the adverse effects of code clones cannot be ignored. Clone fragments can often cause update anomalies, erratically increase the program size and also cause bugs to propagate as bug fixes need to be duplicated across all the clone fragments. Thus, having a clear picture of the clones that exist in a system is crucial for a complete and in-depth understanding of the subject system [4]. As stated in a previous study [1], re-factoring clones in a system requires making informed decisions, and clone analysis systems can assist software maintenance engineers in this task through visual aids.

Several tools have been developed and refined over the last decade for the detection and analysis of code clones in a software system. Some are command-line based tools that provide a textual output such as NiCad [5], SimCad [6], Siamese [7] and SourcererCC [8] while others provide graphical output in addition to the textual results such as VisCad, Clone-World, iClones, and XIAO [9] [10] [11] [12] [13]. Comprehending software clones can often be challenging for large scale projects due to the volume of the data. This can however be offset using visualizations, as humans are intuitively good at identifying patterns. Overall, tools that offer visual representations of code clones are found to be more effective in conveying the information about the subject system, its working, and its overall structure [14]. However, most clone analysis systems that currently exist are restricted in their use due to the lack of cross-platform support, complicated setup processes, and a steep learning curve involved in using these systems. In their need to offer an in-depth clone analysis, most of these tools often provide a plethora of options that can end up confusing the average end user. This has meant that there is a reduced adoption of these tools in real-world scenarios outside of the clone community.

To address this limitation, we present Clone Swarm, a web-based code clone analysis tool. Clone Swarm is built as extension to the NiCad Clone detection system [5] that has been widely used for detecting Type 1, 2 and 3 clones with high precision and recall [15]. Clone Swarm runs NiCad on the cloud and provides the results through an easily accessible web interface, thus allowing software developers and maintainers to interactively explore code clones in their project without having to run the clone detection system themselves. Clone Swarm can analyse any publicly available git repository and offers support for projects in several languages such as C, C#, Java, and Python. Clone Swarm offers multiple overviews and can let users filter what they need to see through a tiered approach. From the top, a project level overview shows the general state of the system by representing the project structure in a radial layout and uses heat-map colour variations to indicate the level of cloning. At the second stage, a file level overview provides information on the number of clones that exist within a file and finally at the bottom tier an individual fragment level overview lets users compare two clone fragments line by line with syntactical highlighting. Finally, Clone Swarm offers users an option to download the clone detection results of their projects directly for future research and analytics.

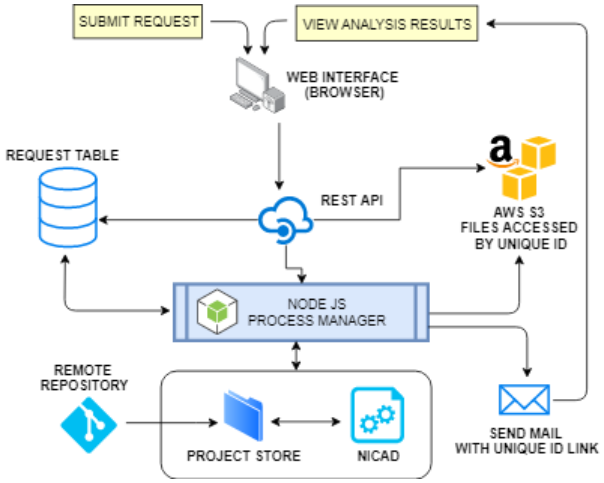

Fig. 1. Architecture diagram of Clone Swarm

II. SYSTEM DESCRIPTION

Clone Swarm is meant to be used primarily by developers involved in the maintenance of software systems. However, it can also be used by program managers to get a general overview of the state of the software system at a given point of time. Clone Swarm does this by visualizing the state of clones in a project along with the inherent project structure by displaying file names along with the folder names in which the clones exist (as shown in Figure 2). After this users can select a particular file to see all the clones that exist in it along with additional information on how big those clones are, the number of times they are duplicated and the file location of these other fragments as shown in Figure 3. Finally, developers can zoom into a particular clone set and look up each clone fragment individually or compare it with a similar fragment, in a separate overview panel (as shown in Figure 4). Clone Swarm consists of a remote back-end system that detects clones and a front end GUI that visualizes the detected clone results. A high-level architecture diagram of Clone Swarm is shown in Figure 1.

III. USAGE SCENARIOS

The results of most clone detection system are purely textual and can often be large in volume depending on the size of the subject system, and this can be a significant challenge in understanding the available clone information. Additionally, large scale data representation can be a challenging problem when the datasets that are being visualized have several different aspects to them. To visualize clones in a software system, we have to show where the clones exist in the system, how many there are of them and how big these clones are in terms of lines of code. To characterize our design problem we relied on the research framework created by Clone-World [10] which in turn was built on a compiled list of clone analysis tasks [14]. The framework provides 5 basic scenarios in which a clone analysis tool can be used: Identifying clone classes for refactoring, Visualizing changes over time, Clone tracking, Analyzing clone distribution in a software system and Clone usages and structural analysis. However some of the scenarios involve studying the evolution of clones over time across different system versions and require use of additional tools such as SPCP-Miner [16] which study clone evolution combined with clone detection. This is beyond the scope of a simple and easy to use clone analysis tool designed for a novice user and so we narrow down and modify our objectives and focus only on the following 3 major scenarios:

S_1. **Clone Distribution**: This scenario is helpful when project managers or software owners need to understand the general health of a software project and the number of clones that exist in the system. This analysis could include looking at the level of severity of code duplication and identifying the regions or modules where clones exist in the system.

S_2. **Clone Tracking**: Bugs can propagate through clones and so when developers fix a bug in a single file they would also need to identify and fix the same bug in all the cloned copies as well. So developers need to be able to search and analyse clones in a single file either based on the clone classes or the level of similarity.

S_3. **Clone Structural Analysis**: When comparing and merging the different cloned copies, developers need to know their exact location(line numbers) and also look at the actual difference in the source files.

IV. USER INTERFACE

Clone Swarm is accessible as a website and has two tabs: a **Home** tab for submitting new requests and a **Dashboard** tab for viewing the results of the clone detection program. On the homepage, a user can submit a new request by entering the URL of the Git repository and their email address to which a notification will be sent once the processing is complete. Users are also required to key in the level of granularity that they need for the clone detection and the language in which the software system is programmed. Clone Swarm currently offers two levels of granularity: Block Level and Function Level and can process systems written in four different programming languages: C, C#,Java, and Python. After a user submits a request on the frontend, a POST call is made, to the backend API and the request is added to the processing queue. The threshold for NiCad is set to 30% by default, but users will be given an option to modify this in a future build on this project (threshold allows for near miss detection and thus clones that may differ by up to 30% of their normalized lines are detected by NiCad).

Once the processing is complete, the user is mailed a link with a unique code that identifies their project, and the link redirects users to the Dashboard tab in Clone Swarm. On this tab, the clone detection results for the project are fetched from the S3 AWS Store using the unique code available on the link and displayed in several overviews.

In presenting clone results we adopt a strategy of incremental visualization and filtering at multiple levels to assist users in their analysis. In implementing this strategy, we also take

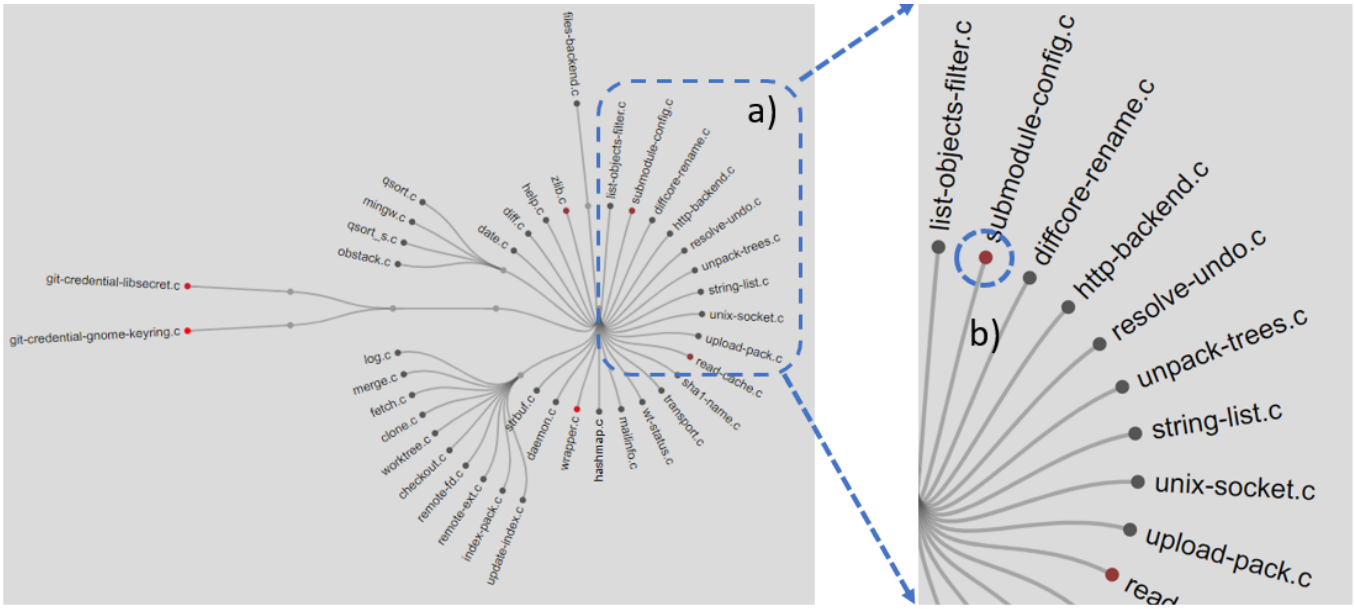

Fig. 2. General Overview Plot, a) Zoomed in view of a particular region, b) File with higher level of cloning visualized by a deeper shade of red compared to the other files in its vicinity.

into consideration design recommendations made by previous clone analysis tools like VisCad [9]. Clone Swarm offers three types of overviews that are described in the following sections.

A. General Overview Plot

This panel is the first level at which the clone information of a project is available and is intended to be used in scenario S_1 to provide an idea about the general health of the project system in terms of code clones as shown in Figure 2. Because of the nature in which code fragments tend to be copy-pasted, code clones can end up being scattered in multiple files across a large project system, and the best way to present this information is by visualizing the clones along with the different subsystems and their hierarchy in the project. However, visualizing this hierarchical structure in a regular tree would take up too much space can end up warping into a long rectangular structure when there is deep nesting in a project. To solve this dilemma, we represent the information in a circular fashion. Not only does this maximize space utilization but also makes the chart easier to understand because of the inherent concentric ring pattern that reduces visual clutter.

In this plot, we render every file or folder that has a clone in it as a node and then arrange these nodes in a radial layout with the root of the project resting at the center. Files or folders that are in the same level of hierarchy in the project structure sit on the same circular level in the plot. The radial layout is built with the D3 library [17] that implements a linear-time variant of the Reingold-Tilford algorithm [18]. The nodes are colored on a varying scale of black to bright red to indicate the level of cloning in the file or folder. This can assist users in easily identifying the files that have the highest amount of cloning in them. Each node can be inspected further by

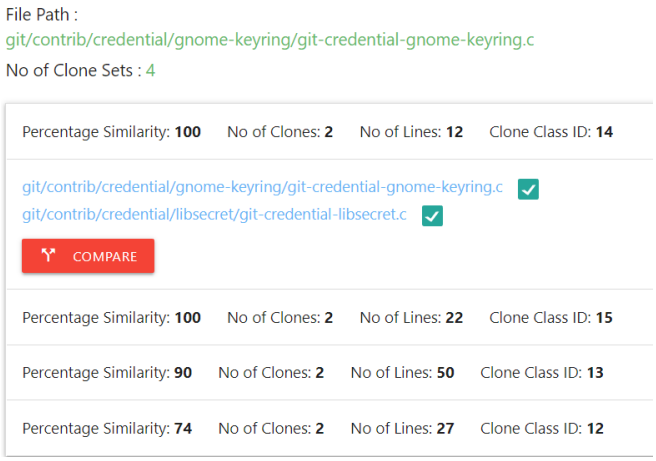

Fig. 3. File Level Overview of clones in a single file.

double-clicking it which opens the clone information for that file in the File level overview panel. This feature currently, however, is only available for files, which are terminal nodes in the radial layout(and not inner nodes which are folders). Every terminal node is also marked with the name of the file for easier identification and for other nodes users can hover their mouse pointer over a node to see its name. Additionally, the plot is rendered using simple vector graphics which makes it scale invariant, meaning that it can be panned and zoomed for closer inspection without a loss in image quality. We also offer two buttons, one for setting a dark theme and the other for resetting the zoom level of the plot.


```
static void usage(const char *name)
{
    struct credential_operation const *try_op = credential_helper_ops;
    const char *basename = strrchr(name, '/');

    basename = (basename) ? basename + 1 : name;
    fprintf(stderr, "usage: %s <", basename);
    while (try_op->name) {
        fprintf(stderr, "%s", (try_op++)->name);
        if (try_op->name)
            fprintf(stderr, "%s", "|");
    }
    fprintf(stderr, "%s", ">\n");
}
```

```
static void usage(const char *name)
{
    struct credential_operation const *try_op = credential_helper_ops;
    const char *basename = strrchr(name, '/');

    basename = (basename) ? basename + 1 : name;
    fprintf(stderr, "usage: %s <", basename);
    while (try_op->name) {
        fprintf(stderr, "%s", (try_op++)->name);
        if (try_op->name)
            fprintf(stderr, "%s", "|");
    }
    fprintf(stderr, "%s", ">\n");
}
```

Fig. 4. Fragment Level Overview of a clone set.

B. File Level Overview

This panel offers information regarding clones present in a particular file as is meant to be used for tasks in scenario S_2, as shown in Figure 3. By default, it shows the information for the file in the project with the largest number of clones, but this can be changed by double-clicking on a node in the general overview plot to view its information instead. The clone pairs are grouped into sets called classes where any two fragments from a set can form a clone pair. These clone classes are then ordered based on the level of similarity in the clone fragments. Users can click on a particular clone fragment to view its code or select and compare two fragments by selecting the files and clicking the compare button.

C. Fragment Level Overview

The third and final panel is hidden by default but can be invoked using interactions on the File Level Overview panel as mentioned in the subsection above. This level is intended to be used in both S_2 and S_3 to first identify a file and then analyse the set of clones within it. It offers syntactical highlighting of the code based on the programming language of the subject system and also provides line level numbering so that a code fragment can be easily looked up in tandem with the file path in the original project codebase. The syntax highlighting feature is built by extending the library Prism.JS [19] for the 4 major languages that Clone Swarm supports. Apart from the single view it also lets users compare two clone fragments side by side which can be useful in checking for false positives and also in making decisions regarding whether a particular fragment should be refactored or not.

V. SERVER ARCHITECTURE

The backend of Clone Swarm consists of three separate modules, each with a distinct part involved in processing a clone detecting request.

The first layer, as shown in Figure 1, is a REST API that is exposed to the Internet and has two open endpoints, one for submitting a new request and another for viewing the processed results. When a user submits a new request the information regarding the project such as its Git URL, programming language, the contact mail address and the level of granularity needed in clone detection are all stored as a record in the request table. Then a processed state flag is set to false to indicate that the record is yet to be processed. Apart from this information, every request is also tagged with a unique reference code that is stored along with the record. After this, the server then sends a trigger to the second module. When a request to view the results of an already processed system is made, the server checks the AWS S3 store for the XML file using the unique access code and serves it back to the web interface where it is parsed and displayed.

The second module in the backend is the processing manager and is built using Node.JS in a single threaded model. It has two states, an ingestion state, and a waiting state. By default, the processing manager is in a waiting state but switches into an ingestion state when it receives a trigger from the first module. If the module is already in the ingestion state, it ignores the trigger. After switching into an ingestion state, the processing manager reads the first record from the request table that has the processed flag set to false and fetches the source code of the request submission from its remote repository and stores it in the Project Store. When cloning the project from the remote source, we only clone it to a depth of 1 to reduce the load on the server and network demand. After this, the process manager triggers NiCad with a given set of input parameters based on the values in the request record such as the programming language and granularity. Because of the inherent complexity in running NiCad and the demand it can place on the server we limit the processing manager to a single thread. However, when we have more than 20 records waiting to be processed, we can spin up to 4 instances of the NiCad module.

When the processing is complete, the processed flag of the record is set to true, and a email message is sent to the notification email provided earlier along with a link tagged with the unique access code identifying the request. NiCad provides the clone detection results in a convenient XML format which is stored in an Amazon S3 bucket store with a name tagged with the aforementioned unique access code. The processing manager then starts to process the next record in the wait queue, and if there aren't any, it switches back into the

waiting mode. The architecture of the processing manager was inspired by the design of Commit Guru, a tool that analyses software systems and predicts risky commits [20]. The third and most important module of Clone Swarm is NiCad [5] a free, open-source tool that performs the actual clone detection. NiCad is effective at detecting both exact and near-miss clones which are clones that share some degree of similarity. It detects clones based on a hybrid model that uses a combination of syntactic and semantic characteristics and thus has both high precision and high recall [15], [21].

We built the backend using a stateless REST API model with the results being stored in a remote S3 AWS(amazon web services) store to facilitate loose coupling and scalability. If the demand for the backend increases, we can create several instances of the backend system and deploy them behind a load balancer.

VI. LIMITATIONS AND FUTURE WORK

Clone Swarm is built as an extension to NiCad and so is limited in the number of features it can offer due to the limitations of the underlying clone detection system. Clone Swarm only offers support for 4 major languages and not all of them have multiple granularity levels. In future we would like address these limitations by extending our backend architecture to other clone detection systems that have been developed recently like SourcerCC [8] and Siamese [7] that are faster and scalable to larger systems.

VII. CONCLUSION

Despite being a hot topic in software engineering, code clone analysis tools have had minimal adoption in the real world outside of the clone community, possibly because of their esoteric nature and the complexity involved in setting them up for regular use. In this paper, we address this problem and present Clone Swarm, a code clone analysis tool that lets software developers detect and interactively explore the clones in their project. Clone Swarm uses NiCad [5] to detect the clones on a remote server and lets users submit requests and view results through a web-based visualization. Clone Swarm lets users explore their code in several tiered overviews and offers a top to bottom approach to drill down on erroneous clone fragments. Clone Swarm is available for use for free at **clone-swarm.usask.ca**. Researchers can use Clone Swarm to analyse any open source repository. Also, clone analysis results of 15 popular open source projects are available on the homepage of Clone Swarm for researchers interested in understanding software clones along with an option to download the results for further exploration.

ACKNOWLEDGMENT

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and additionally by two Canada First Research Excellence Fund (CFREF) grants sponsored by the Global Institute for Water Security (GIWS) and the Global Institute for Food Security (GIFS).

REFERENCES

[1] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, "An empirical study of code clone genealogies," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 5. ACM, 2005, pp. 187–196.

[2] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta, "An empirical study on the maintenance of source code clones," *Empirical Software Engineering*, vol. 15, no. 1, pp. 1–34, 2010.

[3] F. Rahman, C. Bird, and P. Devanbu, "Clones: What is that smell?" *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 503–530, 2012.

[4] C. K. Roy, M. F. Zibran, and R. Koschke, "The vision of software clone management: Past, present, and future (keynote paper)," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, 2014, pp. 18–33.

[5] J. R. Cordy and C. K. Roy, "The nicad clone detector," in *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*. IEEE, 2011, pp. 219–220.

[6] M. S. Uddin, C. K. Roy, and K. A. Schneider, "Simcad: An extensible and faster clone detection tool for large scale software systems," in *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*. IEEE, 2013, pp. 236–238.

[7] C. Ragkhitwetsagul and J. Krinke, "Siamese: scalable and incremental code clone search via multiple code representations," *Empirical Software Engineering*, pp. 1–49, 2019.

[8] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: Scaling code clone detection to big-code," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 1157–1168.

[9] M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Viscad: flexible code clone analysis support for nicad," in *Proceedings of the 5th International Workshop on Software Clones*. ACM, 2011, pp. 77–78.

[10] D. Mondal, M. Mondal, C. K. Roy, K. A. Schneider, Y. Li, and S. Wang, "Clone-world: A visual analytic system for large scale software clones," *Visual Informatics*, vol. 3, no. 1, pp. 18 – 26, 2019, sI: Proceedings of PacificVAST 2019.

[11] Y. Dang, S. Ge, R. Huang, and D. Zhang, "Code clone detection experience at microsoft," in *Proceedings of the 5th International Workshop on Software Clones*. ACM, 2011, pp. 63–64.

[12] Y. Dang, D. Zhang, S. Ge, C. Chu, Y. Qiu, and T. Xie, "Xiao: Tuning code clones at hands of engineers in practice," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 369–378.

[13] Z. M. Jiang, A. E. Hassan, and R. C. Holt, "Visualizing clone cohesion and coupling," in *13th Asia-Pacific Software Engineering Conference (APSEC 2006), 6-8 December 2006, Bangalore, India*, 2006, pp. 467–476.

[14] H. A. Basit, M. Hammad, and R. Koschke, "A survey on goal-oriented visualization of clone data," in *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*, Sep. 2015, pp. 46–55.

[15] J. Svajlenko and C. K. Roy, "Evaluating clone detection tools with bigclonebench," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 131–140.

[16] M. Mondal, C. K. Roy, and K. A. Schneider, "Spcp-miner: A tool for mining code clones that are important for refactoring or tracking," in *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 2015, pp. 484–488.

[17] M. Bostock, V. Ogievetsky, and J. Heer, "D3 data-driven documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.

[18] C. Buchheim, M. Jünger, and S. Leipert, "Improving walker's algorithm to run in linear time," in *Graph Drawing*, M. T. Goodrich and S. G. Kobourov, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 344–353.

[19] Prism.JS, https://prismjs.com/, accessed: Sept. 2019.

[20] C. Rosen, B. Grawi, and E. Shihab, "Commit guru: Analytics and risk prediction of software commits," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 966–969.

[21] C. K. Roy and J. R. Cordy, "A mutation/injection-based automatic framework for evaluating code clone detection tools," in *2009 International Conference on Software Testing, Verification, and Validation Workshops*, April 2009, pp. 157–166.