LETTER

Software Engineering Data Analytics: A Framework Based on a Multi-Layered Abstraction Mechanism

Chaman WIJESIRIWARDANA[†a)], ***Member and*** **Prasad WIMALARATNE**[†], ***Nonmember***

SUMMARY This paper presents a concept of a domain-specific framework for software analytics by enabling querying, modeling, and integration of heterogeneous software repositories. The framework adheres to a multi-layered abstraction mechanism that consists of domain-specific operators. We showcased the potential of this approach by employing a case study.

key words: *software analytics, mining software repositories, domain specific frameworks*

1. Introduction

With the increase in standard processes and productivity tool support, both the number of artifacts created in the software development process and the amount of available data has significantly grown. These vast amounts of data contain valuable information that could help supporting software developers, but it is impossible to examine everything manually. As a result, recent years have witnessed extensive studies on mining software repositories (MSR) to discover various types of valuable information about software projects [1], [2]. However, even with the existing MSR tools, software developers often suffer from a scarcity of techniques necessary for flexible querying and integration of custom data stored in such repositories towards accomplishing complex analysis tasks [3], [4]. Therefore, software researchers have to spend considerable time on integrating and synthesizing the information stored in repositories to perform software evolution analysis tasks. This has identified as an essential future direction in mining software repositories [5].

Existing frameworks that support MSR based on either generic query languages such as SQL or domain-specific languages that specifically designed for MSR. Approaches such as Gitana [6], AlitheiaCore [7] and MetricMiner [8] are based on the standard SQL syntax. These attempts are not explicitly facilitating MSR specific complex tasks such as *finding the number of critical issues resolved by the most frequent committer in a project* or *finding the bug introducing changes between two successful builds*. Boa [9] and QWALKEKO [10] are domain specific languages specifically designed for MSR. However, these approaches are built for querying particular data sources. Beyond that, such domain-specific languages are lacking the rich features of well-established generic query languages and not easily extended to facilitate heterogeneous data sources.

As a solution, we introduce a conceptual foundation of a domain-specific framework to support flexible querying and analysis of software engineering data. The framework adheres to an extensible multi-layered abstraction mechanism that consists of a collection of domain-specific operators. The domain-specific operators are built on top of the operators derived from relational algebra. The framework helps both researchers and practitioners in effectively exploring various kinds of information stored in both traditional and non-traditional repositories.

2. Approach

Software data analytics typically consists of three phases: It starts with collecting required data, e.g., by extracting data available on software repositories. Then the data needs to be preprocessed using various operations such as data cleaning or data filtering. Finally, the actual analysis task required for accomplishing the analysis goals can perform. To facilitate this workflow, this paper introduces a collection of software analytics specific operators, which could be meaningfully composed to build up the logic to accomplish analysis tasks.

2.1 Operator Stack

As depicted in Fig. 1, the operator stack consists of several layers, which could be extensible with new layers and a collection of corresponding operators. In the following, we briefly describe each layer and the rationale for separating it into multiple layers.

Basic operators: These operators directly derived from relational algebra. Relational algebra is a procedural query language, which operates on input relations. It consists with a set of fundamental operators such as `select`, `project`, `union` and `cartesian product`. Though several relational algebra theorems do not strictly hold in query languages such as SQL and LINQ, they are the native implementations of the underline concept of relational algebra. Therefore, we borrowed some ideas from such languages to identify basic operators supported by relational algebra. In this paper, operators such as `filter`, `select`, `join`, `sort`, `count`, etc. has been categorized as basic level operators. Such operators are useful in generating queries to perform simple tasks such as *counting the number of commits of par-*

Manuscript received April 5, 2018.
Manuscript publicized December 4, 2018.
[†]The authors are with School of Computing, the University of Colombo, Reid Avenue, Colombo, Sri Lanka.
a) E-mail: chaman@uom.lk
DOI: 10.1587/transinf.2018EDL8070

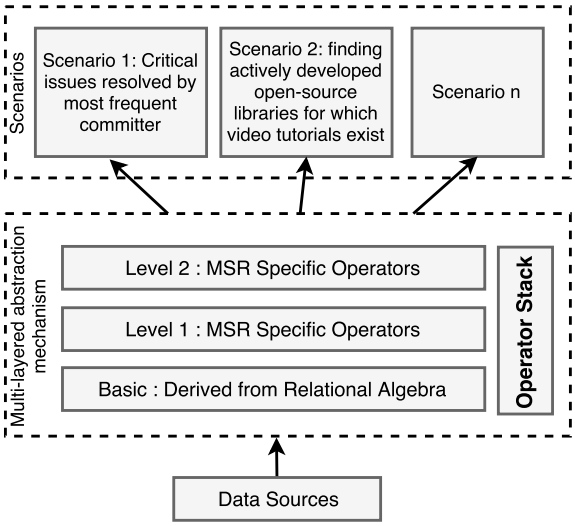

Fig. 1 Operator stack and the examples of supported scenarios.

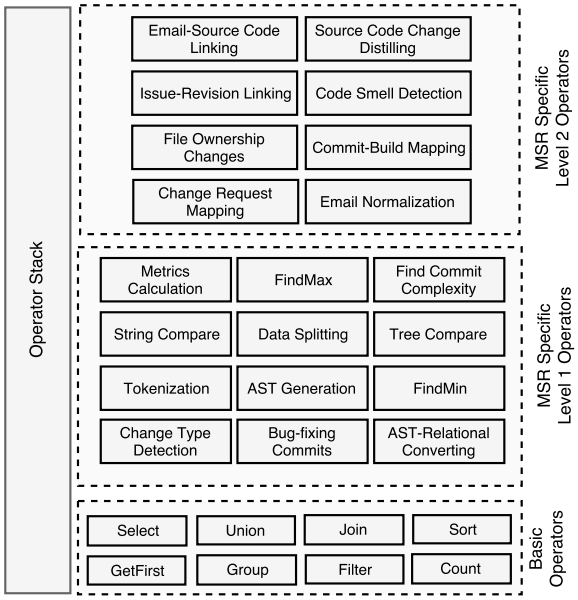

Fig. 2 Collection of operators derived from the literature.

ticular developer or *finding the successful builds within a particular period*.

Level 1 operators: Level 1 operators derived by combining basic operators without further processing to ease MSR specific tasks. Consider an analysis task of *finding the most frequent committer in a project*. It requires extracting commit details from a VCS, filtering the required fields, sorting them according to the number of commits, and selecting the committer with the highest number of commits. The whole process could be accomplished by using a single `FindMax` operator, which is made up of three basic level operators: `filter`, `sort` and `select`. Therefore, `FindMax` is categorized as a Level 1 operator in the proposed operator stack. It can be used to find, for instance, *finding the developer who fixed the most number of bugs*.

Level 2 operators: Software analysis tasks, by nature, are more complex than the examples described in the previous section. It may include complex tasks such as `issue-revision linking`, `change distilling` or `detecting code smells`. As a result, an operator such as `FindMax` could be used to perform a sub-task in a more complex analysis task. Therefore, we emphasize the critical need for more powerful operators as a single unit that is made up of a combination of basic and Level 1 operators to perform complex analytics experiments.

2.2 Repository of Operators

One of the main contributions of this paper is to introduce a collection of frequently used operators that are used to perform software analytics experiments. Initially, a literature review is conducted on the papers published on the International Conference on Mining Software Repositories[†] for the last three years (2015–2017). We carefully investigated the methodology section in each paper to identify the main analytics tasks carried out during the experiments. Our study was not limited to the baseline papers on the mentioned conference and the duration. We further traced back to find related work published outside the selected papers to identify the frequent nature of the operators. Figure 2 presents the identified operators from the literature review categorized into *basic*, *level 1*, and *level 2*. However, the operator repository would experience a continuous evolution with the time.

[†]www.msrconf.org/

2.3 Prototype Implementation

As a proof-of-concept, the operators are built using java languages. As of now, users can accomplish a software analysis task by writing a simple program by utilizing the available operators in the operator library.

3. Case Study Based Evaluation

A case study is employed to demonstrate how to build the logic to perform a software analysis task by utilizing the operators from the operator stack. The following analysis task showcases a possible interlinking of two software repositories by utilizing both the basic level and Level 1 operators from the operator stack.

Analysis Task: *Finding critical issues resolved by most frequent committer in a project.*

Measuring the performance of the developers is a non-trivial task for project managers, especially when the team size and project scope is large. Due to its subjective nature, finding an accurate, quantifiable metrics is a holy grail for managers. Simplistically, total lines of codes, number of commits, number of bugs fixed, or a meaningful combination of them could yield useful insights of performance.

Background: Software artifacts produced during the development of a software project are disconnected from each other. Establishing traceability links across artifacts is a

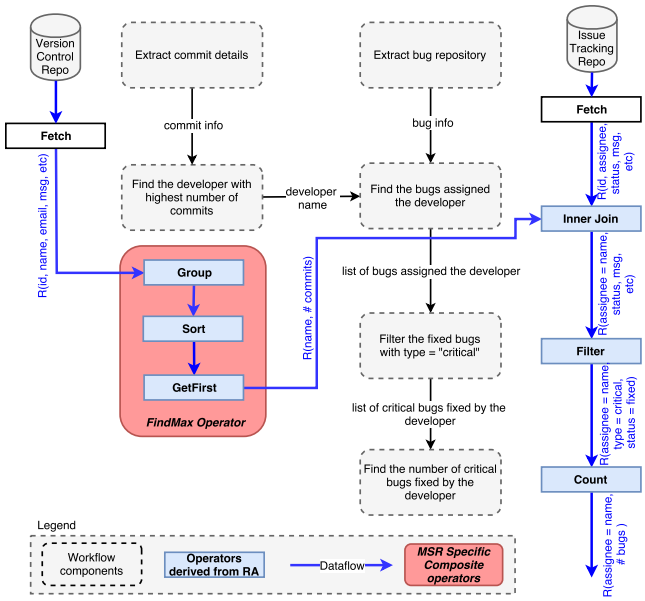

Fig. 3 Finding the number of critical issues resolved by the most frequent committer in a project.

Table 1 Summary of the experimental results.

Apache Project	No: of Commits	Frequent Developer	No: of bug fixes
Gora	1053	Developer A	52
Commons-lang	5171	Developer B	4
IO	2091	Developer C	0
Winx	1312	Developer D	2

key challenge in software maintenance [11]. To address this, gold rush for querying the interrelationship data across autonomous and heterogeneous software repositories has witnessed in several recent studies.

Implementation using operators: Figure 3 presents how to utilize the identified operators to perform the above analysis task. It demonstrates how the data is integrated from both version control and bug tracking repositories to find how many critical bugs have fixed by the most frequent committer.

We further tested the above scenario with four open source projects by using the prototype implementation. The prototype allows users to utilize the operators to perform the tasks directly. Therefore, it provides a great level of convenience to the users. Summary of the experimental results present in Table 1. The names of the developers keep anonymous on the table, however, can be exposed to the reviewers.

4. Conclusion

This paper presents a conceptual foundation of a domain-specific framework based on a multi-layered abstraction mechanism to support flexible querying and analysis of software engineering data. The framework supports querying, modeling, and integration of heterogeneous and autonomous software repositories. Main building blocks of the multi-layered abstraction mechanism are represented regarding a collection of basic and MSR specific operators. As future work, we plan to enrich the operator stack with more operators derived from the MSR research. Then the framework will be evaluated with more case studies by incorporating a diverse set of analysis tasks.

Acknowledgements

The authors of this paper gratefully acknowledge the financial support provided by the National Research Council (Grant No: NRC 15-74).

References

[1] H. Hemmati, S. Nadi, O. Baysal, O. Kononenko, W. Wang, R. Holmes, and M.W. Godfrey, "The MSR cookbook: Mining a decade of research," 2013 10th Working Conference on Mining Software Repositories (MSR), pp.343–352, IEEE, 2013.

[2] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, "How do professional developers comprehend software?," Proc. 34th International Conference on Software Engineering, pp.255–265, IEEE Press, 2012.

[3] A.E. Hassan, "The road ahead for mining software repositories," 2008 Frontiers of Software Maintenance, FoSM 2008, pp.48–57, IEEE, 2008.

[4] Y. Sakamoto, S. Matsumoto, S. Saiki, and M. Nakamura, "Visualizing software metrics with service-oriented mining software repository for reviewing personal process," 2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp.549–554, IEEE, 2013.

[5] J.M. Gonzalez-Barahona, G. Robles, and I. Herraiz, "Challenges in software evolution: The libre software perspective,"

[6] V. Cosentino, J.L.C. Izquierdo, and J. Cabot, "Gitana: A SQL-based git repository inspector," International Conference on Conceptual Modeling, Lecture Notes in Computer Science, vol.9381, pp.329–343, Springer, 2015.

[7] G. Gousios and D. Spinellis, "Conducting quantitative software engineering studies with Alitheia core," Empirical Software Engineering, vol.19, no.4, pp.885–925, 2014.

[8] F.Z. Sokol, M.F. Aniche, and M.A. Gerosa, "MetricMiner: Supporting researchers in mining software repositories," 2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM), pp.142–146, IEEE, 2013.

[9] R. Dyer, H.A. Nguyen, H. Rajan, and T.N. Nguyen, "Boa: A language and infrastructure for analyzing ultra-large-scale software repositories," Proc. 2013 International Conference on Software Engineering, pp.422–431, IEEE Press, 2013.

[10] R. Stevens and C. De Roover, "Querying the history of software projects using QWALKEKO," 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp.585–588, IEEE, 2014.

[11] R. Witte, Y. Zhang, and J. Rilling, "Empowering software maintainers with semantic web technologies," The Semantic Web: Research and Applications, Lecture Notes in Computer Science, vol.4519, pp.37–52, Springer, 2007.