2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)

Towards a Knowledge warehouse and expert system for the automation of SDLC tasks

Ritu Kapur
Department of Computer Science and Engineering
Indian Institute of Technology Ropar
Email: ritu.kapur@iitrpr.ac.in
Under the supervision of
Balwinder Sodhi
Department of Computer Science and Engineering
Indian Institute of Technology Ropar
Email: sodhi@iitrpr.ac.in

***Abstract*—Cost of a skilled and competent software developer is high, and it is desirable to minimize dependency on such costly human resources. One of the ways to minimize such costs is via automation of various software development tasks.**

Recent advances in Artificial Intelligence (AI) and the availability of a large volume of knowledge bearing data at various software development related venues present a ripe opportunity for building tools that can automate software development tasks. For instance, there is significant latent knowledge present in raw or unstructured data associated with items such as source files, code commit logs, defect reports, comments, and so on, available in the Open Source Software (OSS) repositories.

We aim to leverage such knowledge-bearing data, the latest advances in AI and hardware to create knowledge warehouses and expert systems for the software development domain. Such tools can help in building applications for performing various software development tasks such as defect prediction, effort estimation, code review, etc.

***Index Terms*—Automated Software Engineering, Software Maintenance, Data Mining, Supervised Learning**

I. Motivation

The success of a software development project depends to a great extent on the skills and experience of the software engineers working on the project [1]. Moreover, the cost of skilled engineers contributes towards a major part of the overall project cost. Thus it is highly desirable to reduce the dependency on such skilled resources, and also make them more effective. One of the ways to achieve such a goal is through building tools and techniques which can automate (fully or partially) the software development tasks, and this is the broad focus of our work.

A. Basic tenets behind our system

The gist of it can be stated in the form of the research hypothesis and research steps specified in this section.

1) Hypothesis: Broadly, the hypothesis we are working with can be stated as follows:

a) Software development is a creative process, and outcomes of a software development task depend greatly on the skills of the engineers.

b) There is plethora of useful information existing about completed and actively developed software. This information can be leveraged to build tools and techniques for automating software development tasks, thus helping in reducing costs. For example, OSS projects available at GitHub, programmer discussions at StackOverflow, and so on offer rich sources of data and other artifacts which can be leveraged.

2) Research Steps: The basic research steps to achieve the goals outlined in preceding paragraph can be listed as follows:

a) Identify sources from where we can extract useful information about software development tasks in completed or active projects.

b) Extract relevant information from the above sources which can guide the decision making process while developing a software. The decisions here include both technological and managerial decisions. For example, estimation of development effort [2], identification and localization of defects [3], and so on.

c) Using the above information, build suitable AI based tools to achieve the aforementioned goals.

Poor software development design decisions often lead to poor quality software [4] resulting in a high maintenance cost [5], [6], [7], whereas a poor software development effort estimate can cause significant losses for the developer [8]. Therefore, it is desirable to build various tools and techniques which can aid in such decision-making activities and reduce or possibly eliminate the uncertainty.

B. Software comprises of basic software elements

A software comprises of certain basic elements [9] such as component, connector, framework, library, etc. Thus, the effort for developing software can be computed by adding up the effort spent on building its basic software elements. Further, the use of developer activity information to build proficient effort estimation models [10]. But, as per our knowledge and the reviewed literature, no such generalized taxonomy or software development effort estimation metrics exist.

978-1-7281-3393-5/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSSP.2019.00011

C. Programming design decisions affect software quality

Researchers in [4] have shown that the coding practices adopted by the programmers significantly influence the quality and reliability [11] of the software. Further, software metrics are the most prominent method for representing programming decisions [12]. But, very few software metrics exist which can adequately capture the design decisions information.

D. Leveraging information available in OSS repositories

The increased use of online collaboration tools and services has resulted in the accumulation of large volumes of information about software development and related activities. Such sources of raw/unstructured knowledge include but not limited to:

- The platforms such as StackOverflow [13] that allow professional developers to discuss a variety of software development issues.
- OSS hosts such as GitHub [14], Apache Software Foundation (ASF) [15], etc., that allow OSS communities to collaboratively develop industrial strength software.
- Software defect reporting portals such as Apache Bugzilla [16], Eclipse Bugzilla [17], etc., that provide information about the defects reported against various OSS repositories.
- Vendor specific documentation portals such as MSDN [18], IBM Developer Network [19], Oracle Technology Network, etc., that describe the best practices as well as usage patterns for a variety of technologies developed by those vendors. Such technologies often serve as the core fabric for building customized products and solutions for a variety of domains.

Such raw content contains the wealth of latent knowledge that can be leveraged to build various useful automated software systems. However, the challenge is that such raw content first needs to be transformed into a form which is suitable for automatic knowledge analysis, reasoning, and recommendation.

II. Related Work

Two of the works that deal with defect prediction similar to ours are [20] and [21]. The former trains a Deep Belief Networks on semantic features extracted from source code, and uses it for defect prediction. [21] employs a textual similarity approach for performing the same task. However, there are important differences and gaps:

1) While building the feature sets for training the ML models, [20] considers the mere presence or absence of programming constructs in the input.
2) As per our knowledge, none of the existing works explore the characteristics of defects associated with source files.
3) Most of the works have relied on a small volume (less than five projects) of source code and defect reports if any.

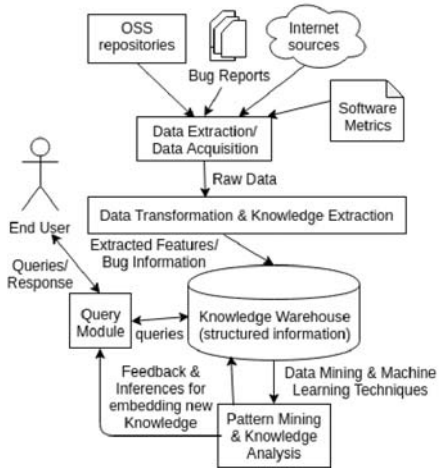

Fig. 1. Schematic diagram of the proposed methodology

III. Objectives

Major objective of our work is to build a set of novel algorithms and tools for:

1) Defining metrics related to software development tasks and artifacts. For instance, such metrics will capture the information associated with items such as programming design decisions, and programmer activity, and so on.
2) Building software artifacts which allow performing automatic extraction of raw un/semi-structured data from different sources as highlighted in §I-D.
3) Building software artifacts which allow performing automatic transformation of un/semi-structured data into a structured form such as a dataset or a Knowledge-Warehouse.
4) Building software artifacts such as datasets, taxonomies, ML models, and so on which are required for use by various AI algorithms that perform automatic analysis and reasoning on such data for answering domain specific queries.
5) Building consumer-facing service APIs which allow 3rd parties to build intelligent services and tools that exploit our knowledge warehouse and models. For instance, one may build a recommender system for validation of architectural design decisions.

IV. Methodology

The methodology, shown in Fig. 1, is explained in the following steps:

1) **Data extraction.** Data is collected from various Internet sources such as OSS repositories, Defect (or bug) reporting portals, etc., available at various online platforms.
2) **Data transformation and knowledge extraction.** The raw data collected from various online sources in the previous step is transformed into a required format and knowledge is extracted from it in the form of relevant

features. The extracted features are stored in the form of a structured knowledge warehouse.

3) **Build a knowledge warehouse.** The knowledge warehouse stores all the extracted features in a structured manner and is used to derive various new patterns and facts by the Pattern Mining Knowledge Analysis and the Query Module component, as shown in Fig. 1.
4) **Build a pattern mining query analysis system.** The extracted knowledge is then analyzed using various Machine Learning (ML) techniques to discover some important patterns, if they exist, in the information collected. The patterns discovered are sent both to the query module and the knowledge warehouse. The Knowledge warehouse stores these patterns as newly discovered knowledge for utilization in the future whereas the query module uses it to answer different user queries.

V. Research Challenges and Scope

The work focuses on the implementation of various Information Retrieval and ML techniques for knowledge discovery, representation and analysis for applications to the domain of Software Engineering. The raw input for such knowledge comes from the various sources as described in §I-D.

A. Proposed Scope and Deliverables

The current scope of our work is limited to the *design* and *development* phase of software development. The work will lead to the following deliverables:

1) Scraping module to collect information from various online sources such as defect reporting portals, OSS repository hosts such as GitHub, etc.
2) Feature Extraction Module to extract features from raw information extracted from various online sources.
3) Knowledge warehouse, containing the relevant information, in a structured format.
4) New software development-specific software metrics.
5) New software development-specific software taxonomies for various software.
6) Discovery of interesting patterns that exist in the data collected in the Knowledge warehouse.
7) Domain specific ML models built using the patterns discovered in the previous step and different ML techniques.
8) APIs and user interfaces that allow querying pertinent information from the knowledge warehouse and via ML models.

B. Research Challenges

Some of the research challenges are highlighted below:

1) The raw input for our system is text data available online. The existing Natural Language Processing techniques are unable to adequately handle the text that pertains to the software development domain. For instance, a polar word/phrase that uses such a vocabulary may or may not be considered polar from common English language perspective. For example, consider the following sentence:

   ``The following piece of code takes a huge amount of memory and CPU and takes very long to produce results.''

   A narrative similar to the above sentence about a piece of code is highly likely to be considered negative. However, most existing Natural Language Processing based tools would label this sentence as either a "neutral" or "positive" [22].
2) Further, there is a large volume of pertinent information available in the form of graphical images, audio, and video formats. How to integrate the non-text sources of input data is a challenge.
3) The current scope of the project is limited to only open source information available sources. However, there is a significant volume of information (e.g., defect reports, source code, design-documentations, etc.) available in closed-source projects which can provide useful knowledge.
4) When dealing with defect prediction, it is difficult to estimate the presence/absence of functionality related defects. As such, we are currently dealing with the defects pertaining to only the non-functional requirements (e.g., reliability, performance, security, etc.) of a software.
5) In our current work that was have carried out, a key assumption has been the validity of defect reports. Though it is a reasonable assumption, for many projects and for "confirmed" defects, there are always chances of a defect report being inaccurate.

VI. Research Plan

I have completed three years of my Ph.D.; the first half of which was focused on clearing the preliminaries required at my university. As outlined in the schematic diagram shown in Fig. 1, the current work focuses on two major problems:

- Estimating the defectiveness of source code
- Estimating the time and development effort required to build a software

A. Estimating the defectiveness of source code

Maintenance of software, particularly identifying and fixing the defects, contributes significantly towards the overall cost of developing and deploying a software application. To reduce this cost and solve the problem of design time , we [23] have designed a Defect Estimator for Source Code (DESCo) to predict the defects associated with source files. The DESCo system not only predicts the defectiveness associated with an input source file but also the characteristics of the associated defects. To build DESCo, we perform the following steps:

1) Define PROgramming CONstruct (PROCON) metrics to capture the design decisions.
2) Extract various OSS repositories and the associated defect reports from various online portals.
3) Build a dataset by extracting PROCON metrics' values by processing the source files present in various OSS repositories (fetched in the previous step).

4) Use prominent ML techniques to build different models trained on the dataset.

A copy of PROCON dataset is available at https://goo.gl/Tgyiqu. DESCo system interface is shared at http://desco.webhop.me:5500. We have shown that SVM with radial kernel performs the best (averaged F1 score of 0.807) for identifying a source file as potentially defective or not. Similarly, RF model with 7 estimators performs best (F1 score 0.694) when predicting bugs annotated with highest priority [23]. The work has been validated by comparing it with the state-of-the-art methods [20], [24] and datasets (PROMISE repository and others) [25]. This entire work is complete and submitted (under review) to ACM Transactions on Software Engineering and Methodology.

B. Software Development Effort Estimation for building a software

Effort estimation plays a critical role in planning and monitoring the software development process. We design an effort estimation system by utilizing the effort information of software built in the past (discussed in §I-B). To build this software, we follow the steps listed below:

1) Define the software development effort estimation metrics which capture the relevant effort information.
2) Extract the OSS repositories and the associated metadata information from various online portals.
3) Build a dataset by extracting the metrics' values by processing the data associated with various OSS repositories. These metrics are based on the developer activity information associated with various OSS repositories.
4) Use prominent ML techniques to build different ML models trained on the dataset.

We have completed building the software development effort estimation dataset, which is available at https://goo.gl/ZKxzHc. I plan to complete the model-building and API part in 1-2 months, and then submit this work to IEEE Transactions on Software Engineering.

C. Work plan for future

In Future, we plan to explore more such problems which help in building efficient expert systems to automate the software development tasks and support the decision making process. Such automated systems not only help to save important resources (such as time, cost, etc.) but also help in performing the tasks at better precision levels. The software artifacts, thus created, can be further used to build tools and techniques related to several automated software engineering areas like bug localization [26], [21], code review and recommendation [27] and program repair [26] etc.

REFERENCES

[1] D. Leonard-Barton, "Core capabilities and core rigidities: A paradox in managing new product development," *Strategic management journal*, vol. 13, no. S1, pp. 111–125, 1992.

[2] T. K. Abdel-Hamid, "Adapting, correcting, and perfecting software estimates: a maintenance metaphor," *Computer*, vol. 26, no. 3, pp. 20–29, 1993.

[3] M. Famelis and M. Chechik, "Managing design-time uncertainty," in *Model Driven Engineering Languages and Systems (MODELS), 2017 ACM/IEEE 20th International Conference on*. IEEE, 2017, pp. 179–179.

[4] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "Learning natural coding conventions," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 281–293.

[5] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*. ACM, 2000, pp. 73–87.

[6] B. P. Lientz and E. B. Swanson, "Software maintenance management," 1980.

[7] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, vol. 55, no. 8, pp. 1397–1418, 2013.

[8] K. Molokken and M. Jorgensen, "A review of software surveys on software effort estimation," in *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*. IEEE, 2003, pp. 223–230.

[9] B. Sodhi, "Towards an architectural element recommender system."

[10] J. J. Amor, G. Robles, and J. M. Gonzalez-Barahona, "Effort estimation by characterizing developer activity," in *Proceedings of the 2006 international workshop on Economics driven software engineering research*. ACM, 2006, pp. 3–6.

[11] J. D. Gannon and J. J. Horning, "Language design for programming reliability," *IEEE Transactions on Software Engineering*, no. 2, pp. 179–191, 1975.

[12] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[13] StackOverflow, "Stack overflow - where developers learn, share, build careers," 2018. [Online]. Available: https://stackoverflow.com/

[14] GitHub, "The world's leading software development platform · github," 2018. [Online]. Available: https://github.com/

[15] A. S. F. (ASF), "Welcome to the apache software foundation!" 2018. [Online]. Available: https://www.apache.org/

[16] ——, "Bugzilla - apache issues - the apache software foundation!" 2018. [Online]. Available: https://bz.apache.org/bugzilla/

[17] Eclipse, "Bugzilla main page - eclipse bugzilla," 2018. [Online]. Available: https://bugs.eclipse.org/bugs/

[18] Microsoft, "C coding conventions (c programming guide)," 2018. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions

[19] IBM, "Best practices for using java api code - ibm," 2018. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSWSR9_11.6.0/com.ibm.pim.app.doc/code/java/pim_con_bestpract.html

[20] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 297–308.

[21] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 14–24.

[22] B. Sodhi and S. Sharma, "Using stackoverflow content to assist in code review," *arXiv preprint arXiv:1803.05689*, 2018.

[23] R. Kapur and B. Sodhi, "Estimating defectiveness of source code: A predictive model using github content," *arXiv preprint arXiv:1803.07764*, 2018.

[24] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[25] PROMISE, "Promise repository," 2018. [Online]. Available: http://openscience.us/repo/defect/

[26] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 802–811.

[27] Y. Malheiros, A. Moraes, C. Trindade, and S. Meira, "A source code recommender system to support newcomers," in *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual*. IEEE, 2012, pp. 19–24.