NAutoBuilder: A NLP Based Tool to Build Software Automatically

Foyzul Hassan
Department of Computer Science
University of Texas at San Antonio
Email:foyzul.hassan@gmail.com

***Abstract*—In Software Engineering, even with recent research on automated tools , human involvement is required in hole life cycle of software development. Automation of this task will assist developers and also help researchers for better analysis of the software. Our research goal is build software repositories automatically from software configuration and readme files using Natural Language Processing techniques. The result is a system that will build software without intervention of human activity and it will set other configuration related tasks to build and run the software.**

I. Introduction

During the past decade, as more software developers have their projects hosted in open software repositories (e.g., Github [1], Sourceforge [2], Google Code [3]), large open software repositories become available to software engineering researchers and provide valuable resources for developing novel software engineering techniques. A large number of recently developed techniques are based on analyzing and mining data (code, version history, bug reports, emails, etc.) from software repositories, and the software projects in software repositories also serve as desirable subjects for the evaluation of numerous software engineering techniques [4] [5].

Since the code base is typically the functioning part in software projects, in both scenarios mentioned above, it is often necessary to apply certain types of program analysis on the code base of software projects in the software repositories. Furthermore, to analyze a large number of software projects from software repositories, it is important to automate the whole process of analysis by reducing or eliminating human intervention.

However, while the majority of program analysis techniques require a successful build process or a well built software project as their input, this requirement is not easy to satisfy for software projects collected from open software repositories. Open software repositories typically store and maintain only source code of software projects, so the projects must be built before they can be fed into various program analysis tools. Although it is possible to apply partial program analysis on only the available source code, without the resolution of dependencies, and relevant types / bindings, it is difficult or impossible to precisely perform some basic tasks such as counting API calls or constructing a call graph, with only partial program analysis. For example, for a statement foo().bar();, if foo() is an unresolved library method, we are not able to know its return type. So, we have to assume that all methods with a name ”bar” can be invoked, while actually, only the methods with a name ”bar” in the subtypes of the return type of foo() can be invoked. Also, we are not able to know if bar() is an API call or not, and what API it refers to. Furthermore, many well-maintained popular frameworks (e.g., Soot [6], Wala [7], and LLVM [8]) require built software or a successful build process, and a large number of program analysis techniques have been developed on these frameworks. Thus, without have the software projects built, these techniques and corresponding tools cannot be reused.

To sum up, automatic building of software projects provides a desirable foundation to support a large variety of software engineering research tasks based on open software repositories. With this foundation, it will be easier for software engineering researchers to design more sophisticated and effective program-analysis-based techniques, and larger-scale evaluation.

II. Related Works

Our research work is the first research effort on the automatic building of software projects from open software repositories using NLP techniques. The project is related to recent research efforts on the analysis of build configuration files, and mining software repositories. Existing research on analysis of build configuration files mainly falls in three categories: dependency analysis of involved path expressions, migration of build configuration files, and empirical studies.

On dependency analysis, Gunter [9] proposed a Petri-net based model to describe the dependencies in build configuration files. Adams et al. [10] proposed a framework to extract a dependency graph for build configuration files, and provide automatic tools to keep consistency during revision. Most recently, Al-Kofahi et al. [11] proposed an a fault localization approach for make files, which provides the suspiciousness scores of each statement in a make files for a building error. The most closely related work in this category is SYMake developed by Tamrawi et al. [12]. SYMake uses a symbolic-evaluation-based technique to generate a string

dependency graph for the string variables/constants in a Makefile , automatically traces these values in maintenance tasks (e.g., renaming), and detect common errors. Compared to SYMake, the proposed project plans to develop build configuration analysis for a different purpose (i.e., automatic software building). Therefore, the proposed analysis estimates run-time values of string variables with grammar-based string analysis instead of string dependency analysis, and analyzes flows of files to identify the paths to put downloaded files and source files to be involved.

On migration of build configuration files, AutoConf [13] is a GNU software that automatically generates configuration scripts based on detected features of a computer system. AutoConf detects existing features (e.g., libraries, software installed) in a build environment, and configure the software based on pre-defined options. By contrast, our proposed project tries to adapt the build environment, and resolve build errors which are not expected by developers. Most recently, Gligoric et al. [14] proposed an approach to automate the migration of various building configuration files to CloudMake configuration files based on building execution with system-level instrumentation. While their approach requires a successful execution of the original build configuration files, our proposed project mainly handles the cases with NLP techniques.

Mining Software Repositories. In the field of mining software repositories, our research is specifically related to online software search, and large scale analysis of software projects. On online software search, a number of research efforts have been made on searching for relevant libraries given certain queries. Such research efforts include CodeWeb, component-ranking. Previous researchers also studied searching for similar applications of a given application , and searching for code samples of APIs . Compared to the above research efforts, automatic software building searches libraries with signatures so the assessment of search results is straightforward. However, since it is prohibitively expensive to download and try all search results, a proper ranking strategy is required to enhance search efficiency.

On large-scale analysis of software projects, most previous researchers apply partial program analysis or meta-data analysis to mine software repositories or to perform empirical studies. Our work may further enhance these research efforts by allowing more in-depth program analysis techniques to be applied to a large number of software projects. Researchers also have been aware of the importance of diversity and reproducibility of empirical studies and evaluations based on software repositories. We believe that our work may provide some benefits to the diversity and reproducibility of such tasks.

III. Conclusion

The proposed research work provides a framework to automatically build arbitrary software projects in open software repositories. Such techniques will enable more precise and complete program analysis on a large number of software projects and versions in open software repositories, and thus will benefit software engineering techniques that mines and analyzes these repositories, as well as large scale evaluation of program-analysis-based software engineering techniques. By reporting building failures related to portability flaws of software projects, the project may also enhance the portability of software projects in open software repositories.

References

[1] P. Charles, "Project title," https://github.com/charlespwd/project-title, 2013.

[2] "The sourceforge story," http://web.archive.org/web/20110716044546/http://itmanageme accessed: 2012-04-12.

[3] "Google code project hosting," https://code.google.com/hosting, accessed: 2009-08-06.

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

[5] H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 19, no. 2, pp. 77–131, 2007.

[6] P. Lam, E. Bodden, L. Hendren, and T. U. Darmstadt, "The soot framework for java program analysis: a retrospective."

[7] "Ibm. the t. j. watson libraries for analysis (wala)," http://wala.sourceforge.net, accessed: 2012-03-20.

[8] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, ser. CGO '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 75–. [Online]. Available: http://dl.acm.org/citation.cfm?id=977395.977673

[9] N. Aoumeur and G. Saake, "Dynamically evolving concurrent information systems specification and validation: A component-based petri nets proposal," *Data Knowl. Eng.*, vol. 50, no. 2, pp. 117–173, Aug. 2004. [Online]. Available: http://dx.doi.org/10.1016./j.datak.2003.10.005

[10] B. Adams, H. Tromp, K. De Schutter, and W. De Meuter, "Design recovery and maintenance of build systems," in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, Oct 2007, pp. 114–123.

[11] J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen, "Fault localization for build code errors in makefiles," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. New York, NY, USA: ACM, 2014, pp. 600–601. [Online]. Available: http://doi.acm.org/10.1145/2591062.2591135

[12] A. Tamrawi, H. A. Nguyen, H. V. Nguyen, and T. N. Nguyen, "Symake: A build code analysis and refactoring tool for makefiles," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: ACM, 2012, pp. 366–369. [Online]. Available: http://doi.acm.org/10.1145/2351676.2351749

[13] "Gnu autoconf - creating automatic configuration scripts," http://www.gnu.org/software/autoconf/manual/index.html, accessed: 2015-10-25.

[14] M. Gligoric, W. Schulte, C. Prasad, D. van Velzen, I. Narasamdya, and B. Livshits, "Automated migration of build scripts using dynamic analysis and search-based refactoring," in *Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications*, 2014.