SCIENCE CHINA
Information Sciences

October 2019, Vol. 62 200104:1–200104:3
https://doi.org/10.1007/s11432-018-9854-3

• LETTER •
Special Focus on Software Automation

AI-boosted software automation: learning from human pair programmers

Xin PENG[1,2*], Zhenchang XING[3] & Jun SUN[4]

[1]*School of Computer Science, Fudan University, Shanghai* 201203*, China;*
[2]*Shanghai Key Laboratory of Data Science, Fudan University, Shanghai* 201203*, China;*
[3]*Research School of Computer Science, Australian National University, Acton* ACT 2601*, Australia;*
[4]*School of Information Systems, Singapore Management University, Singapore* 178902*, Singapore*

Received 18 December 2018/Revised 31 January 2019/Accepted 19 March 2019/Published online 3 September 2019

Citation Peng X, Xing Z C, Sun J. AI-boosted software automation: learning from human pair programmers. Sci China Inf Sci, 2019, 62(10): 200104, https://doi.org/10.1007/s11432-018-9854-3

Dear editor,
Software automation [1] aims to automatically generate computer programs from formal or informal requirement descriptions. It covers a variety of transformations of different spans, including generating programs from natural-language requirements, requirements specifications, or design specifications. Traditionally software automation is achieved through logical reasoning and rule-based transformation [1]. Although the transformation from high-level programming languages to their executable forms has been fully automated, automatic generation of programs from their requirements is still hard due to the informality, non-operationality, and incompleteness of the requirements [2].

The progress of software automation can be boosted by the development of big data and AI (artificial intelligence) techniques. For example, open source communities such as GitHub[1)] host hundreds of millions of projects with various kinds of software development data such as source code, revisions, issues, emails; online forums such as Stack Overflow[2)] record dozens of millions of questions and answers on a wide range of topics in programming. Moreover, companies such as Google have accumulated billions of lines of code in their repositories to support tens of thousands of developers around the world [3]. On the other hand, some recent studies have revealed that most software is natural, and thus, like natural language, is also likely to be repetitive and predictable [4]. Based on the huge amount of software development data and knowledge, one can naturally expect that most of the common functionalities have been implemented and shared by others, and most of the common problems in software development have been reported and solved by others. Based on this assumption, many researchers have explored ways of data-driven intelligent software development, which leverages the big software development data and AI techniques such as deep learning for software automation.

To date data-driven intelligent software development has made it possible to automate some specific tasks in software development such as recommending the next API based on code context [5], generating API usage sequences for a given natural language query [6], and generating GUI skeleton from UI design image [7]. These tasks only account for a small part of software development. End to end automated program generation from requirements is possible for specific types of small programs such as data manipulation programs (which can be induced from input-output pairs), but is unrealistic for industry-scale software systems in

* Corresponding author (email: pengxin@fudan.edu.cn)
1) https://github.com.
2) https://stackoverflow.com.

general due to the following challenges.

• Creative and uncertain nature of software. Understanding the requirements and generating architecture design of these systems require creativity and to deal with a great deal of uncertainty. Developers often need to think carefully to understand and refine user requirements into concrete functionalities and business logics. They also have to consider a sound software architecture that satisfies the desired non-functional requirements such as performance, reliability, and extendability. Moreover, the requirements and architecture involve a great deal of uncertainty incurred by the ever-changing user requirements and runtime environments.

• Domain diversity. The software projects in open source and industrial repositories have high diversity in their business and technical domains. These projects belong to a large variety of business domains (e.g., finance, e-business, education, entertainment) and are built on a large variety of languages, libraries, frameworks, and platforms (e.g., Java, Spring, Android). Although the total amount of software development data is huge, the data that a specific project can learn from may be limited considering the business and implementation diversity. Moreover, different aspects (e.g., different libraries and frameworks, common and project-specific logics) are often interweaved together, even in a small piece of code.

• Data quality. The quality of much software development data is questionable due to both essential and accidental factors. Although the principle of separation of concerns is widely accepted, code scattering and tangling phenomena is common in open-source and industrial software systems. On the other hand, the text content (e.g., class/method/variable names, comments) of programs, which is used in tasks like code recommendation and program comprehension, is often not expressed in a consistent and normative way. For example, a recent study by Liu et al. [8] revealed that a large part of the commit messages that are used as references are noisy, for example they may be bot messages generated by tools or trivial messages containing little or redundant information.

Therefore, it may be more realistic to expect human-AI cooperative automation for industry-scale software systems. It means that developers still follow existing development processes such as agile development and an AI assistant behind the scene acts as a pair programmer that provides the required helps when the developers encounter difficulties. The purpose of this kind of cooperative automation is not to replace developers by end to end automation, but achieve better efficiency and quality by reducing repetitive work and helping novice developers think and work like experienced ones.

To understand the requirements for the AI assistant, let us first consider how human developers work and cooperate. Given a development task, developers need to achieve the required goal (e.g., implementing a new feature or improving an existing one) based on their development knowledge and understanding of the current code context. Usually they can also resort to various development resources such as code bases, API documentation, online forums. The reason why they encounter difficulties in the task usually lies in the gap between the developers' knowledge and the goal of the task. For example, the developers may not know the calculation principle of offset in the canvas of a text editor or the APIs that can change the color of the text on the canvas. Due to the knowledge gap it is often hard for them to find the required solution even though it can be implied from existing code, documentation, or online discussions. Bridging the knowledge gap therefore becomes the main task of the AI assistant.

The way how pair programmers communicate with and help each other can help us further understand how the AI assistant should work. Here are some thoughts that are inspired by the cooperative work of pair programmers.

Interactive clarification and explanation. When a pair programmer understands the problem of the other one and provides suggestions, he/she usually needs to interactively clarify the intention and explain the suggested solution. For example, when the pair programmer suggests to use Java StringBuffer to construct the text content read from a file, he/she may explain that StringBuffer is thread safe and this explanation can increase the trust on the suggestion. To recommend the solution for the next step, he/she may clarify the intention, for example, by asking whether to print the text content or show it on the screen. Even when recommending a code fragment the pair programmer may explain the parameters and other implementation details that need to be adapted to the task requirements and local code context. Without this kind of clarification and explanation, it is hard for the AI assistant to understand the goal of the developers, make them trust the suggestions, and help them successfully apply the solutions.

Stepwise refinement. Developers often follow a non-sequential order of thinking and editing [5]. For example, a developer may first write the body of a file manipulation functionality and then consider its condition (e.g., whether the file exists). When developers provide suggestions for others they usually understand the problem and develop the solution from a refinement process. They could first suggest some core APIs with sample code for feedback to clarify the intention, and then determine and complete the implementation details gradually, for example configuring variable values and adding initialization, resource cleaning, and exception handling code. The AI assistant needs to follow a similar process to understand the intention of the developer and suggests the required solution, and avoids to get into the details from every beginning.

With the required background knowledge. An implied premise of pair programming is that the pair programmers share the required background knowledge, including both technical and business knowledge. For example, when talking about the thread safety of string APIs the pair programmers need to both understand the underlying technical concepts such as process/thread, thread safety, and buffering and know that thread safety is only meaningful when multiple threads access and modify the string. Similarly, when choosing APIs for reading/writing excel files, the pair programmers need to know concepts like sheet, row, column, cell and their relationships. To efficiently communicate with the developers and provide accurate suggestions, the AI assistant needs to be equipped with knowledge such as API knowledge graph [9] and other technical or business knowledge graphs.

On-demand solution granularities or forms. Developers need solutions and suggestions of different granularities or forms in different situations. When an implementation (e.g., a code fragment or a set of files) for a similar functionality can be found there is no reason to suggest the code line by line; instead, the pair programmer could recommend the whole reference implementation and suggest the required modifications. In some cases, the knowledge gap of the developers lies in an API, a technical principle, or the format of a string variable, thus the pair programmer needs to provide different forms of suggestions and solutions such as API recommendation, explanation of technical principles and string format. Therefore, the AI assistant needs to understand the knowledge gap of the developer and provides suggestions and solutions of different granularities or forms in an on-demand way.

To conclude, end to end automated program generation is unrealistic for industry-scale software systems due to the creative and uncertain nature of software, diversity of technical and business domains, and data quality issues. A more realistic expectation is human-AI cooperative automation, where an AI assistant acts as a pair programmer and provides the required helps. How the AI assistant should work can be understood by observing the way how pair programmers communicate with and help each other. Some inspirations include interactive clarification and explanation, stepwise refinement, background knowledge, and on-demand solution granularities or forms.

Acknowledgements This work was supported by National Key Research and Development Program of China (Grant No. 2016YFB1000801).

References

1 Xu J, Chen D, Lv J, et al. Software Automation (in Chinese). Beijing: Tsinghua University Press, 1994
2 Mei H, Zhang L. Can big data bring a breakthrough for software automation? Sci China Inf Sci, 2018, 61: 056101
3 Potvin R, Levenberg J. Why Google stores billions of lines of code in a single repository. Commun ACM, 2016, 59: 78–87
4 Hindle A, Barr E T, Gabel M, et al. On the naturalness of software. Commun ACM, 2016, 59: 122–131
5 Nguyen A T, Nguyen T N. Graph-based statistical language model for code. In: Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE-15), Florence, 2015. 858–868
6 Gu X D, Zhang H Y, Zhang D M, et al. Deep API learning. In: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE-16), Seattle, 2016. 631–642
7 Chen C Y, Su T, Meng G Z, et al. From UI design image to GUI skeleton: a neural machine translator to bootstrap mobile GUI implementation. In: Proceedings of the 40th International Conference on Software Engineering (ICSE-18), Gothenburg, 2018. 665–676
8 Liu Z X, Xia X, Hassan A E, et al. Neural-machine-translation-based commit message generation: how far are we? In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE-18), Montpellier, 2018. 373–384
9 Li H W, Li S R, Sun J M, et al. Improving API caveats accessibility by mining API caveats knowledge graph. In: Proceedings of the 34th IEEE International Conference on Software Maintenance and Evolution (ICSME-18), Madrid, 2018. 183–193