The Synergy of Human and Artificial Intelligence in Software Engineering

Tao Xie
North Carolina State University
Raleigh, NC, USA
xie@csc.ncsu.edu

***Abstract*—To reduce human efforts and burden on human intelligence in software-engineering activities, Artificial Intelligence (AI) techniques have been employed to assist or automate these activities. On the other hand, human's domain knowledge can serve as starting points for designing AI techniques. Furthermore, the results of AI techniques are often interpreted or verified by human users. Such user feedback could be incorporated to further improve the AI techniques, forming a continuous feedback loop. We recently proposed cooperative testing and analysis including human-tool cooperation (consisting of human-assisted computing and human-centric computing) and human-human cooperation. In this paper, we present example software-engineering problems with solutions that leverage the synergy of human and artificial intelligence, and illustrate how cooperative testing and analysis can help realize such synergy.**

I. Introduction

Various software-engineering activities traditionally require intensive human efforts and impose burden on human intelligence. To reduce human efforts and burden on human intelligence in these activities, Artificial Intelligence (AI) techniques, which aim to create software systems that exhibit some form of human intelligence, have been employed to assist or automate these activities in software engineering [22]. Example AI techniques employed in software-engineering activities are constraint solving [10], [15] and search heuristics [39] used in test generation and machine learning [16], [18] used in debugging, natural language processing [20], [35], [45] used in specification inference, and knowledge engineering [4] used in various software-engineering tasks along with search-based techniques broadly applied in search-based software engineering [13]. These AI techniques typically accomplish more than automating repetitive well-defined subtasks by instilling simulation or emulation of how human intelligence would have been able to help address these software-engineering activities. Comparing to human intelligence, these AI techniques implemented in software systems are able to handle a much larger scale of tasks (e.g., the scale of the data under analysis).

On the other hand, human intelligence has also been leveraged in the broad context of human-centric software engineering [19], [38]. Typically human's domain knowledge can serve as starting points for designing AI techniques. Furthermore, the results of AI techniques are often interpreted or verified by human users. Such user feedback could be incorporated to further improve the AI techniques, forming a continuous feedback loop.

Our recently proposed cooperative testing and analysis [37], [38] include human-tool cooperation (consisting of human-assisted computing and human-centric computing) and human-human cooperation. In human-assisted computing [37], tools are on the "driver" seat and users provide guidance to the tools so that the tools could better carry out the work. In contrast, in human-centric computing [30], users are on the "driver" seat and tools provide guidance to the users so that the users could better carry out the work. Human-human cooperation is often in the form of crowdsourcing [5].

We next present some example software-engineering problems with solutions on the spectrum from leveraging human intelligence to artificial intelligence, and illustrate how cooperative testing and analysis (e.g., human-assisted computing and human-centric computing) can help realize the synergy of human and artificial intelligence.

II. Example Problems

Test generation. Traditionally, test generation relies on human intelligence to design test inputs that can satisfy testing requirements such as achieving high code coverage or fault-detection capability. On the other hand, automatic test generation [36] relies on tool automation to automatically generate test inputs that can satisfy testing requirements. In human-assisted computing, test-generation tools are in the driver seat and may face challenges in test generation; for some of these challenges, users can provide guidance such as writing mock objects [17], [24], [31] or writing factory methods for encoding method sequences to produce desirable object states [29]. In human-centric computing, users are in the driver seat and may get guidance from tools on subtasks such as what assertions need to be written and what method sequences need to be written [23] or translating natural-language test descriptions to executable test scripts [25].

Specification generation. Traditionally, specification generation relies on human intelligence to write specifications. In the recent decade, automatic specification inference has emerged to infer likely specifications from runtime traces [1], [8], source code [27], [28], and natural language artifacts [20], [35], etc. In human-assisted computing, recent research [33] leverages simple user-written specifications to more effectively infer sophisticated specifications. In human-centric computing, users write specifications based on confirming or rejecting

likely specifications recommended by tools such as Agitator [2].

Debugging. Traditionally, debugging relies on human intelligence to locate faulty code and fix the code. Previous research [40], [41] uses automatic debugging to automatically isolate failure-inducing inputs, etc. Recent research [34] attempts to automatically fix faulty code. In human-assisted computing, recent research [32] seeks guidance from users when tools generate imperfect fixes (e.g., causing unexpected impact). In human-centric computing, previous research [14] visualizes code locations associated with suspicion levels of being faulty so that users can get guidance on where to inspect [21]. Recent research [12] mines suspicious patterns as starting points for users to conduct investigation.

Programming. Traditionally, programming relies on human intelligence to write programs for implementing some functionalities. In recent years, program synthesis [11] intends to automatically synthesize programs for implementing some functionalities. In human-assisted computing, program synthesis by nature relies on users to provide user intents (e.g., functionalities to be implemented) in various forms. In human-centric computing, various tools are developed for recommending to programmers example API usage [26], [44] or code completion [3].

III. Conclusion

To reduce human efforts and burden on human intelligence in software-engineering activities, Artificial Intelligence (AI) techniques have been employed to assist or automate these activities. Typically human's domain knowledge can serve as starting points for designing AI techniques. Furthermore, the results of AI techniques are often interpreted or verified by human users. Such user feedback could be incorporated to further improve the AI techniques, forming a continuous feedback loop. Through example problems in software engineering, we have illustrated our recently proposed cooperative testing and analysis, especially human-tool cooperation (consisting of human-assisted computing and human-centric computing). Such cooperative testing and analysis help realize the synergy of human and artificial intelligence in software engineering.

Furthermore, such cooperative testing and analysis can also play important roles in successful technology transfer and adoption. Our previous successful efforts [6], [7], [9], [12] on putting software analytics [42], [43] in practice highlight the importance of investigating how practitioners take actions on the produced insightful and actionable information, and provide effective support for such information-based action taking. In particular, we need to rely on better AI techniques such as knowledge representation, machine-human interaction, and understanding of human cognitive tasks. We also need to leverage human intelligence to take actions on the produced analytic results.

Acknowledgment

This work is supported in part by NSF grants CCF-0845272, CCF-0915400, CNS-0958235, CNS-1160603, an NSA Science of Security Lablet grant, a NIST grant, a Microsoft Research Software Engineering Innovation Foundation Award, and NSF of China No. 61228203.

References

[1] G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. In *Proc. POPL*, pages 4–16, 2002.

[2] M. Boshernitsan, R. Doong, and A. Savoia. From Daikon to Agitator: lessons and challenges in building a commercial tool for developer testing. In *Proc. ISSTA*, pages 169–180, 2006.

[3] M. Bruch, M. Monperrus, and M. Mezini. Learning from examples to improve code completion systems. In *Proc. ESEC/FSE*, pages 213–222, 2009.

[4] S. K. Chang. *Handbook of Software Engineering And Knowledge Engineering: Recent Advances*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2005.

[5] N. Chen and S. Kim. Puzzle-based automatic testing: bringing humans into the loop by solving puzzles. In *Proc. ASE*, pages 140–149, 2012.

[6] Y. Dang, D. Zhang, S. Ge, Y. Qiu, and T. Xie. XIAO: Tuning code clones at hands of engineers in practice. In *Proc. ACSAC*, pages 369–378, 2012.

[7] R. Ding, Q. Fu, J.-G. Lou, Q. Lin, D. Zhang, J. Shen, and T. Xie. Healing online service systems via mining historical issue repositories. In *Proc. ASE*, pages 318–321, 2012.

[8] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, 2001.

[9] Q. Fu, J.-G. Lou, Q.-W. Lin, R. Ding, Z. Ye, D. Zhang, and T. Xie. Performance issue diagnosis for online service systems. In *Proc. SRDS*, pages 273–278, 2012.

[10] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed automated random testing. In *Proc. PLDI*, pages 213–223, 2005.

[11] S. Gulwani. Dimensions in program synthesis. In *Proc. PPDP*, pages 13–24, 2010.

[12] S. Han, Y. Dang, S. Ge, D. Zhang, and T. Xie. Performance debugging in the large via mining millions of stack traces. In *Proc. ICSE*, pages 145–155, 2012.

[13] M. Harman. The current state and future of search based software engineering. In *2007 Future of Software Engineering*, pages 342–357, 2007.

[14] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proc. ICSE*, pages 467–477, 2002.

[15] J. C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, 1976.

[16] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan. Bug isolation via remote program sampling. In *Proc. PLDI*, pages 141–154, 2003.

[17] M. R. Marri, T. Xie, N. Tillmann, J. de Halleux, and W. Schulte. An empirical study of testing file-system-dependent software with mock objects. In *Proc. AST*, pages 149–153, 2009.

[18] A. Michail and T. Xie. Helping users avoid bugs in GUI applications. In *Proc. ICSE*, pages 107–116, 2005.

[19] G. C. Murphy. Human-centric software engineering. In *Proc. FSE/SDP Workshop on Future of Software Engineering Research*, pages 251–254, 2010.

[20] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar. Inferring method specifications from natural language API descriptions. In *Proc. ICSE*, pages 815–825, 2012.

[21] C. Parnin and A. Orso. Are automated debugging techniques actually helping programmers? In *Proc. ISSTA*, pages 199–209, 2011.

[22] C. Rich and R. Waters. *Readings in artificial intelligence and software engineering*. Kaufman Publishers Inc., Los Altos, CA, USA, 1986.

[23] Y. Song, S. Thummalapenta, and T. Xie. UnitPlus: Assisting developer testing in Eclipse. In *Proc. ETX*, pages 26–30, 2007.

[24] K. Taneja, Y. Zhang, and T. Xie. MODA: Automated test generation for database applications via mock objects. In *Proc. ASE*, pages 289–292, 2010.

[25] S. Thummalapenta, S. Sinha, N. Singhania, and S. Chandra. Automating test automation. In *Proc. ICSE*, pages 881–891, 2012.

[26] S. Thummalapenta and T. Xie. PARSEWeb: A programmer assistant for reusing open source code on the web. In *Proc. ASE*, pages 204–213, 2007.

[27] S. Thummalapenta and T. Xie. Alattin: Mining alternative patterns for detecting neglected conditions. In *Proc. ASE*, pages 283–294, 2009.

[28] S. Thummalapenta and T. Xie. Mining exception-handling rules as sequence association rules. In *Proc. ICSE*, pages 496–506, 2009.

[29] S. Thummalapenta, T. Xie, N. Tillmann, J. de Halleux, and Z. Su. Synthesizing method sequences for high-coverage testing. In *Proc. OOPSLA*, pages 189–206, 2011.

[30] N. Tillmann, J. D. Halleux, T. Xie, S. Gulwani, and J. Bishop. Teaching and learning programming and software engineering via interactive gaming. In *Proc. ICSE, Software Engineering Education (SEE)*, 2013.

[31] N. Tillmann and W. Schulte. Mock-object generation with behavior. In *Proc. ASE*, pages 365–368, 2006.

[32] X. Wang, L. Zhang, T. Xie, Y. Xiong, and H. Mei. Automating presentation changes in dynamic web applications via collaborative hybrid analysis. In *Proc. FSE*, pages 16:1–16:11, 2012.

[33] Y. Wei, C. A. Furia, N. Kazmin, and B. Meyer. Inferring better contracts. In *Proc. ICSE*, pages 191–200, 2011.

[34] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming. In *Proc. ICSE*, pages 364–374, 2009.

[35] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie. Automated extraction of security policies from natural-language software documents. In *Proc. FSE*, pages 12:1–12:11, 2012.

[36] X. Xiao, S. Thummalapenta, and T. Xie. Advances on improving automation in developer testing. In *Advances in Computers*, volume 85, pages 165–212. 2012.

[37] X. Xiao, T. Xie, N. Tillmann, and J. de Halleux. Precise identification of problems for structural test generation. In *Proc. ICSE*, pages 611–620, 2011.

[38] T. Xie. Cooperative testing and analysis: Human-tool, tool-tool, and human-human cooperations to get work done. In *Proc. SCAM, Keynote Paper*, pages 1–3, 2012.

[39] T. Xie, N. Tillmann, P. de Halleux, and W. Schulte. Fitness-guided path exploration in dynamic symbolic execution. In *Proc. DSN*, pages 359–368, 2009.

[40] A. Zeller. Yesterday, my program worked. today, it does not. why? In *Proc. ESEC/SIGSOFT FSE*, pages 253–267, 1999.

[41] A. Zeller. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann, 2009.

[42] D. Zhang, Y. Dang, J.-G. Lou, S. Han, H. Zhang, and T. Xie. Software analytics as a learning case in practice: Approaches and experiences. In *Proc. MALETS*, pages 55–58, 2011.

[43] D. Zhang and T. Xie. Software analytics in practice: mini tutorial. In *Proc. ICSE*, page 997, 2012.

[44] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei. MAPO: Mining and recommending API usage patterns. In *Proc. ECOOP*, pages 318–343, 2009.

[45] H. Zhong, L. Zhang, T. Xie, and H. Mei. Inferring resource specifications from natural language API documentation. In *Proc. ASE*, pages 307–318, 2009.