Predicting Software Defects based on Cognitive Error Theories

Fuqun Huang
Institute of Interdisciplinary Scientists
Seattle, WA 98105, USA
huangfuqun@insins.org

Lorenzo Strigini
City, University of London
London, EC1V 0HB, UK
strigini@csr.city.ac.uk

***Abstract*—As the primary cause of software defects, human error is the key to understanding and perhaps to predicting and preventing software defects. However, little research has been done to forecast software defects based on defects' cognitive nature. This paper proposes an idea for predicting software defects by applying the current scientific understanding of human error mechanisms. This new prediction method is based on the main causal mechanism underlying software defects: an error-prone scenario triggers a sequence of human error modes. Preliminary evidence for supporting this idea is presented.**

Keywords—software defect prediction; human errors; cognitive errors; defect prevention; causal analysis

I. INTRODUCTION

Software defect prediction plays a significant role in software engineering by forecasting error-prone program locations and providing guidance for risk control techniques such as software testing. It seems intuitively true that if we can accurately predict an adverse event, we can also effectively prevent it. However, this expectation has not yet been met by the existing software defect prediction approaches, though many have been proposed over the last 45 years [1-5].

The predictors used in the existing defect prediction models can be categorized into three groups: program metrics, testing metrics, and development process metrics [1]. These predictors are then related to defect density by various methods, which have been evolving from simple correlative functions [2] to multivariate approaches such as regression analysis [3], data mining [4] and machine learning algorithms [5]. Despite significant progress in these models, the relations they identify are about correlation rather than causality. The causal mechanisms underlying software defects are not examined in detail. As a result, the current fault prediction models can only provide outputs like "module A is more likely to contain defects than module B"; they are unable to predict the exact location and form of a software defect, which could allow focused and effective preventive actions.

As the primary cause of software defects, human error is the key to predicting and preventing software defects. Programming is a special type of writing, conducted by programmers [6]; software defects are by nature the manifestations of cognitive errors of individual software practitioners and/or of miscommunication between software practitioners [7, 8]. However, a theory is missing on how software defects are introduced by cognitive error mechanisms, and how we can use this theory to predict software defects.

This paper proposes to predict software defects through a deep understanding of cognitive error mechanisms. While current prediction methods mostly suggest software modules to which special attention should be dedicated (especially in testing, or sometimes in reorganizing design before coding) as especially defect-prone, the proposal here is for ways to predict specific kinds of defects at specific steps in programs, and thus to recommend amendments to specifications, or focused checks in code inspection, or test cases.

II. COGNITIVE ERROR MODEL OF SOFTWARE DEFECTS

The cognitive error model of software defect describes how a software defect is caused by human errors (Fig.1). The model includes the main causal factors that determine a human error [9]: the nature of the task (including both its *content* – what is to be programmed – and its *representation* – how this is described to the person who must perform the task), the nature of the programmer (mainly his/her current available knowledge base [10]), and human error modes, which are the general mechanisms governing humans' erroneous cognitive performance.

A human ***error mode*** is a particular pattern of erroneous human behavior that recurs across different activities, due to the cognitive weakness shared by all humans, e.g., applying "strong-but-now-wrong" rules [9].

To predict an error, the nature of the task and individual should be analyzed together, because these two factors interact. For instance, the same task could be easy for one person but difficult for another person. This integrated feature of a task and a human individual is modeled by a "scenario". More specifically, an ***error scenario*** is the set of circumstances under which an error is committed. A *scenario* contains not only the exterior conditions surrounding an individual (e.g. the content and representation formats of the task), but also the interior cognitive conditions relevant to individual's performance, e.g. his/her knowledge relevant to the performance.

The mechanism underlying a software defect is that the *scenario* has triggered a set of *human error modes*. Some defects can be caused by a single mechanism, while others may be introduced by a combination of several mechanisms.

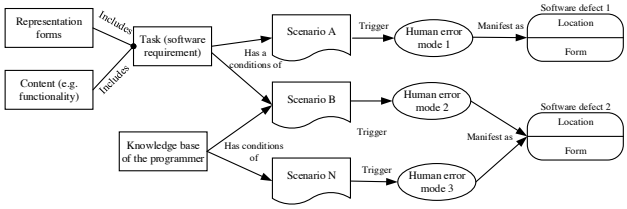

Figure 1. A cognitive error model of software defects

In summary, *error mode* describes "***why***" a defect is introduced; *error scenario* concerns "***when***" (under what circumstances) a defect is introduced; *error mechanism* integrates all the aspects of "***how***" a defect is introduced.

III. Predicting The Exact Locations and Forms of Software Defects

Based on the cognitive error model described above, software defects can then be predicted through identifying in the current programming context (e.g., a program specification) general conditions that are likely to trigger human error modes--the erroneous patterns that psychologists have observed to recur across diverse activities [9, 11].

Some suggestive evidence for the potential of this form of prediction comes from an error reported in [12]. A "post-completion error" scenario triggered many programmers to introduce the same defect in the same form at the same location. "Post-completion error" is an error pattern whereby one tends to omit a sub-task that should be carried out at the end of a task but is not a necessary condition for the achievement of the task's main goal [11]. The general conditions that trigger a post-completion error are described in the left side of Figure 2.

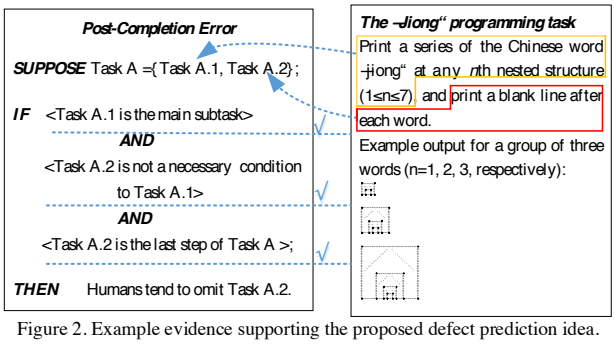

Figure 2. Example evidence supporting the proposed defect prediction idea.

A programming task called the "Jiong" problem (described on the right-hand part of Figure 2) was presented to student programmers in a programming contest. In its specification, the experimenter recognized a task feature likely to trigger a "post-completion error" mode. To complete the "jiong" problem, a programmer first needed to calculate the structure of a "jiong" using a recursion or iteration algorithm (mapping to the main subtask TA.1 on the left side of Fig.2), and then print a blank line after the word (mapping to TA.2 on the left side of Fig.2).

23 out of 55 (41.8%) programmers committed the error of "forgetting to print a blank line after each word", in the same way as observed by psychologists in other tasks. It is notable that "printing a blank line" is a very simple sub-task and was explicitly specified in the requirement; this requirement would be most unlikely to lead to a defect according to traditional prediction models; but in fact, more programmers made a mistake at this program location than at any other locations, and amazingly in the same way.

This case shows that once a scenario matches the general conditions that tend to trigger a human error mode, it indeed appears to provoke programmers to introduce the corresponding defect, in the form and location expected. Therefore, it suggests that software defects can be predicted by identifying the *scenarios* that tend to trigger *human error modes*. This kind of evidence is of course very preliminary. A comprehensive case study will be presented in our forthcoming journal article.

IV. Discussions and Conclusion

This paper proposed the idea of predicting software defects based on defects' cognitive nature. Compared to traditional prediction models that provide a relative likelihood that a program module may contain defects, the proposed idea emphasizes predicting the exact location and form of a possible defect through identifying *scenarios* that tend to trigger *human error modes*. The preliminary evidence shows that this idea is achievable.

The proposed approach to prediction can be highly rewarding in software engineering because if the location and form of the likely defect are predicted, one can *prevent* the defect in a real sense. Huang's previous study [13] suggests that systematic instruction on human error is beneficial to programmers' ability to prevent defects; however, how to relate a programmer's general knowledge of human errors to his/her task at hand remains a challenge. The proposed method is a prime candidate for addressing this challenge.

Future studies will be needed to extract more human error modes, represent the general conditions that tend to trigger these error modes, and design a practical scenario analysis method for identifying the features of tasks and programmers that tend to introduce software defects. Most importantly, the proposed approach of defect prediction would find its place as a technique for debugging specifications, and studies will be needed to show whether it will outperform or enhance the effectiveness of other methods for specification inspection.

References

[1] N. E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Trans. Software Engineering*, vol. 25, pp. 675-689, 1999.

[2] F. Akiyama, "An Example of Software System Debugging," *Information Processing*, vol. 71, pp. 353-379, 1971.

[3] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *27th International Conference on Software Engineering*, 2005, pp. 284-292.

[4] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *Software Engineering, IEEE Transactions on*, vol. 33, pp. 2-13, 2007.

[5] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, vol. 81, pp. 649-660, 2008.

[6] G. M. Weinberg, *The Psychology of Computer Programming*: VNR Nostrand Reinhold Company, 1971.

[7] F. Détienne, *Software design - cognitive aspects*. New York, NY, USA: Springer-Verlag New York, Inc., 2002.

[8] F. Huang, B. Liu, and B. Huang, "A Taxonomy System to Identify Human Error Causes for Software Defects," in *The 18th international conference on reliability and quality in design*, Boston, USA, 2012, pp. 44-49.

[9] J. Reason, *Human Error*. Cambridge, UK: Cambridge University Press, 1990.

[10] F. Huang, B. Liu, and Y. Wang, "Review of Software Psychology (in Chinese)," *Computer Science*, vol. 40, pp. 1-7, 2013.

[11] M. D. Byrne and S. Bovair, "A working memory model of a common procedural error," *Cognitive science*, vol. 21, pp. 31-61, 1997.

[12] F. Huang, B. Liu, Y. Song, and S. Keyal, "The links between human error diversity and software diversity: Implications for fault diversity seeking," *Science of Computer Programming*, vol. 89, Part C, pp. 350-373, 2014.

[13] F. Huang and B. Liu, "Software defect prevention based on human error theories," *Chinese Journal of Aeronautics*, vol. 30, pp. 1054-1070, 2017.