We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Chapter 2

Human Error Analysis in Software Engineering

Fuqun Huang

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.68392

Abstract

As the primary cause of software defects, human error is the key to understanding, detecting and preventing software defects. This chapter first reviews the state of art of an emerging area: software fault defense based on human error mechanisms. Then, an approach for human error analysis (HEA) is proposed. HEA consists of two important components: human error modes (HEM) and an undated version of causal mechanism graphs (CMGs). Human error modes are the general erroneous patterns that humans tend to behave in a variety of activities. Causal mechanism graph provides a way to extract the error-prone contexts in software development, and link the contexts to general human error modes. HEA can be used at various phases of software development, for both defect detection and prevention purposes. An application case is provided to demonstrate how to use HEA.

Keywords: human error analysis, software defect prevention, fault detection, causal mechanism graph, software quality assurance

1. Introduction

Software has become a major determinant of how reliable, safe and secure computer systems can be in various safety-critical domains, such as aerospace and energy areas. Despite the fact that software reliability engineering has remained an active research subject over 40 years, software is still often orders of magnitude less reliable than hardware. There are over 200 software reliability models, but each of which can apply to only a few cases. Based on scientific intuition, if there were a model that had captured the essence of an entity of interest, it should be able to describe the entity in a variety of contexts. It is necessary to reflect what have been overlook in the current research and practices in software (reliability) engineering.

INTECH
open science | open minds

© 2017 The Author(s). Licensee InTech. This chapter is distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. CC BY

Software, as a pure cognitive product [1, 2], does not fail in the same way as how hardware fails. Software does not have material or manufacturing problems, for example, corrosion or aging problems. How a software system performed in the last second could tell nothing about whether the system will fail or not in the next second; and people can hardly anticipate the consequences of a software failure until it happens. Drawing upon the notion of the cognitive nature of software faults, there is a need to build software dependability theories on the foundation of cognitive science.

As the primary cause of software defects, human error is the key to understanding and preventing software defects. Software defects are by nature the manifestations of cognitive errors of individual software practitioners or/and of miscommunication between software practitioners. Though the cognitive nature of software has been realized early in 1970s [3], significant progress has only been made in recent years on how we can use human error theory to defend against software defects [4].

This chapter reviews the new interdisciplinary area: Software Fault Defense based on Human Error mechanisms (SFDHE) and proposes an approach for human error analysis (HEA). HEA is at the core of various methods used to defend against software faults in the SFDHE area.

The chapter is organized as follows: Section 2 reviews the emerging area SFDHE; Section 3 proposes the method for human error analysis (HEA); Section 4 presents an application example; Section 5 makes conclusion.

2. The new interdiscipline: *S*oftware *F*ault *D*efense based on *H*uman *E*rror mechanisms (SFDHE)

2.1. History

Human cognition plays a central role in software development even if in the modern large projects [4–7]. A previous analysis on a large set of industrial data shows that eighty seven percent of the severe residual defects are caused by individual cognitive failures independent of process consistency [8]. Approaches for defending against cognitive errors are necessary to improve software dependability.

Software Fault Defense based on human error mechanisms [5], firstly proposed in 2011 by Huang [4], is an area aiming to systematically predict, prevent, tolerate and detect software faults through a deep understanding of the causal mechanisms underlying software faults—the cognitive errors of software practitioners. This is an interdisciplinary area built on integrative theories in software engineering, systems engineering, software reliability engineering, software psychology and cognitive science.

2.2. State of art

2.2.1. Human error mechanisms underlying software faults

The first phase of SFDHE is to identify the factors that influence software fault introduction, as well as how various factors interact with each other to form a software defect. The factors related to programming performance are traditionally studied in software psychology, with a thorough review in [9]. However, there is few study focusing on identifying factors that influence human errors in programming. One of Huang's recent experimental studies was devoted to comparing the effects of various human factors on fault introduction rate [7]. Results show that a few dimensions of programmers' cognitive styles and personality traits are related to fault introduction rate [7] as significantly as the conventional program metrics [10].

In order to study human errors in software engineering, there is a need to integrate general human error theories with the cognitive nature of software development. Huang [2] developed an integrated cognitive model of software design. Based on the cognitive model, a human error taxonomy was proposed for software fault prevention [2]. Another human error taxonomy was recently developed by Anu and Walia et al. [11] for with an emphasis on software requirement review. These human error taxonomies vary in details in order to achieve different purposes, however, they both place Reason's human error theory [12] as a fundamental theory.

A recent experiment [13] examined how an erroneous pattern called "postcompletion error" [14] manifests itself in software development. Postcompletion error is a specific type of human errors that one tends to omit a subtask that is carried out at the end of a task but is not a necessary condition for the achievement of the main subtask [14]. Postcompletion errors have been observed in a variety of tasks by psychologists, but there is a lack of empirical studies in software engineering. The author's experiment shows that 41.82% of programmers committed the postcompletion error in the same way. As the first attempt to link general human error modes (HEM) to programming contexts, the study has set a significant paradigm for investigating the human error mechanisms underlying software defects.

2.2.2. Software fault prevention based on human error mechanisms

A key activity of the traditional defect prevention process is to identify root causes. Root causes are generally classified into four categories: method, people, tool, and requirement; detailed causes are analyzed by brainstorming with cause-effect diagrams [15]. Such taxonomies are too abstract to be helpful for those organizations with little experience. Huang's human error taxonomy [2] has been used to advance the process of traditional software defect prevention [16, 17].

Huang [18] also developed an approach called defect prevention based on human error theories (DPeHE) to proactively prevent software defects by promoting software developers' cognitive ability of human error prevention. Compared to the conventional defect prevention that

focuses on organizational software process improvement, DPeHE focuses more on software developers' metacognitive ability to prevent cognitive errors. DPeHE promotes software developers' error prevention ability through two stages. In the first stage, DPeHE provides developers with explicit knowledge of human error mechanisms and prevention strategies. In the second stage, software developers use the provided strategies and devices to practice error regulation during their real programming practices. Through this training program, software developers gain better awareness of error-prone situations and better ability to prevent errors. This method has received very positive feedbacks from a variety of industrial users [18].

2.2.3. Software fault tolerance based on human error mechanisms

Independent development (i.e., development by isolated teams) is used to promote the fault tolerance capability in N-version programming. However, empirical evidence shows that coincident faults are introduced even if the redundant versions are truly built independently [19, 20]. Programmers are prone to make the same errors under certain circumstances, thus introducing the same faults at certain places. Huang [4] has been devoted to first understanding why, how and under what circumstances programmers tend to introduce the same faults, and then to seeking a scientific way to achieve fault diversity and enhance software systems' fault tolerant capability [4]. Huang's theory [7] relates the likelihood of identical faults to the "performance level" of the activity required from the programmers. Remarkably, the most frequent coincident fault does not occur at difficult task points that involve knowledge-based performance, but rather at an easy task point that involves rule-based performance [7].

2.2.4. Software fault detection based on human error mechanisms

Since the idea of using human error theories to promote software fault detections at various stages of software development lifecycle was presented in 2011 [4], significant progress has been made recently [11, 21]. Anu and Walia et al. [11] developed a human error taxonomy for requirement review, and positive effects on subjects' fault detection effectiveness were observed. Li, Lee and Huang et al. [21, 22] introduced human error theories to prioritize test strategies at coding and evolution phases.

3. Human error analysis

Human error analysis (HEA) is at the core process of various methods for defending against software faults in SFDHE. HEA can be employed at different phases during software development, for both defect detection and prevention purposes, shown in **Figure 1**. For instance, HEA can be used to promote requirement review, design review and code inspection. At requirement and design phases, HEA can also help one identify contexts prone to trigger software developers' cognitive errors at the next phase, so one can take strategies to prevent the errors.

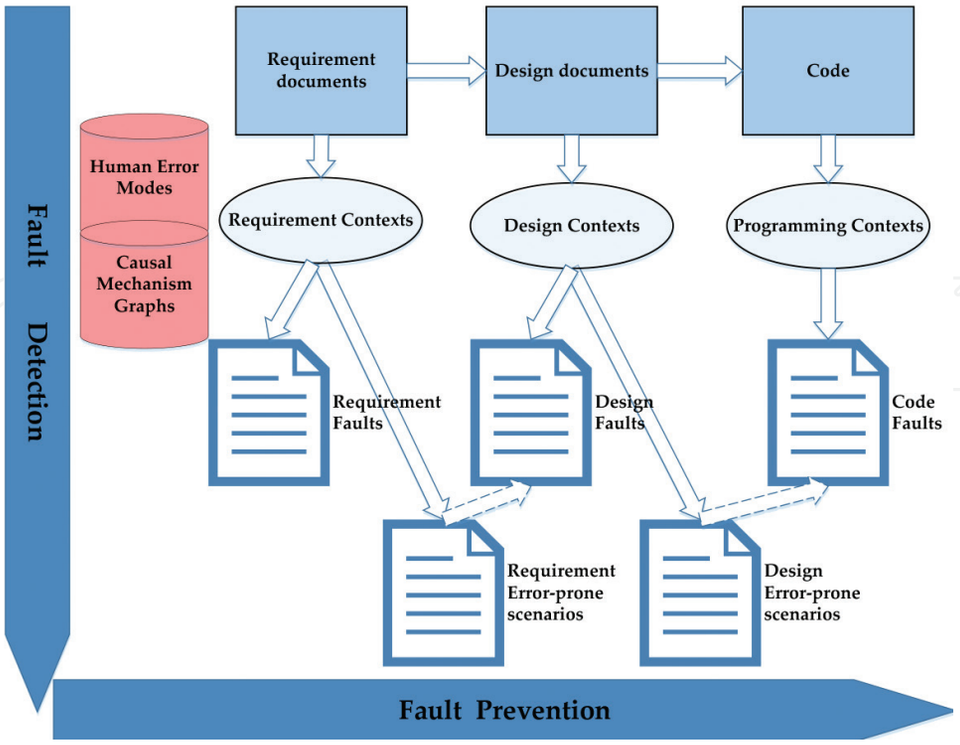


Figure 1. The framework of HEA in software engineering.

HEA consists of two components: human error modes (HEM) and causal mechanism graph (CMG). Human error modes are the erroneous patterns that psychologists that have observed to recur across diverse activities [12, 14]. CMG provides a way to extract a specific set of contexts of the artifact (e.g., requirement, design and code) under analysis to the general conditions that associates with a human error mode.

3.1. Human error modes

Though human errors appear in different "guises" in different contexts, they take a limited number of underlying modes [12]. A human ***error mode*** is a particular pattern of human erroneous behavior that recurs across different activities, due to the cognitive weakness that shared by all humans, for example, applying "strong-but-now-wrong" rules [12].

Understanding such recurring error modes is essential to identifying software defects and the contexts prone to trigger a human error. A sample of the error modes are describes in **Table 1**. These error modes were observed to manifest themselves in software development contexts in the author's previous experimental studies [5, 7, 13] or industrial historical data [8]. More software defects examples associated with these human error modes can be found in [18].

3.2. Causal mechanism graphs

The author recommends a graphic tool called causal mechanism graph (CMG) for causal mechanism modeling. CMG is a notation system firstly used to represent and model the

Error mode name	Explanation and scenarios
Lack of knowledge [2]	Software defects are introduced when one omits related knowledge, or even does not realize related knowledge is required. This error mode is prone to appear especially when the problem is an interdisciplinary problem.
Postcompletion error [13, 14]	The pattern of "post completion error" is that if the ultimate goal is decomposed into several subgoals, a subgoal is likely to be omitted under such conditions: the subgoal is not a necessary condition for the achievement of its corresponding superordinate goal; the subgoal is to be carried out at the end of the task.
Problem representation error	Misunderstand task representation material and simulate wrong situation model of the problem, due to the ambiguity of the material.
Apply "strong but now wrong" rules	People tend to behave the same way in a context that is similar to past circumstances, neglecting the countersigns of the exceptional or novel circumstances. In software development, this means that when solving problems, developers tend to prefer rules that have been successful in the past. The more frequent and successful the rule has been used before, the more likely it is recalled.
Schema encoding deficiencies	Features of a particular situation are either not encoded at all or misrepresented in the conditional component of the rule.
Selectivity	Psychologically salient, rather than logically important task information is attended to. In software development, "selectivity" means that when a developer solving problems, if attention is given to the wrong features or not given to the right features, mistakes will occur, resulting in wrong problem presentation, or selecting wrong rules or schemata to construct solutions.
Confirmation bias	People tend to seek for evidence that could verify their hypotheses rather than refuting them, whether in searching for evidence, interpreting it, or recalling it from memory. Others restrict the term to selective collection of evidence.
Problems with complexity	As problem complexity arises, error symptoms tend to occur such as delayed feedback, insufficient consideration of processes in time, difficulties with exponential developments, thinking in causal series not causal nets, thematic vagabonding, and encysting (topics are lingered over and small details attended to lovingly).
Biased review	People tend to believe that all possible courses of action have been considered, when in fact very few have been considered.
Inattention	Fail to attend to a routine action at a critical time causes forgotten actions, forgotten goals, or inappropriate actions. "Automatic processing" in software developing happens when no problem solving activities are involved, such as typing. Slips might happen without proper monitoring and error detection.

Table 1. Sample of human error modes (adopted from Ref. [18]).

complex causal mechanisms that determine software dependability, which encompasses different attributes, such as reliability, safety, security, maintainability and availability [23, 24].

A causal mechanism graph is capable of capturing logic, time and scenario features, which are essential to the description of interactions between various factors to produce an effect. The notations in CMG allow researchers to model causal mechanisms more accurately: logic

symbols allow for various logical combinations between causes or effects; the scenario symbol enables the identification of situations in which a relation is likely to exist; and time flow allows a number of cause-effect units to develop into a cause-effective chain. Moreover, notations are designed to capture the recurrent patterns of comprehensive causal mechanisms (e.g., activate and conflict).

CMG is especially suitable to represent one's cognitive knowledge, as it allows one to model the dynamic causal mechanisms in a robust way. This feature, combined with excellent reliability and validity [23], positions CMG as a powerful method to extract and model the

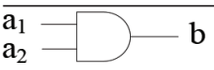
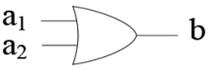
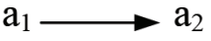
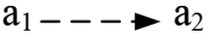
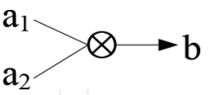
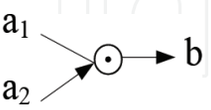
Symbol	Name	Description
	AND	Entity a_1 AND entity a_2 form entity b.
	OR	Entity a_1 OR entity a_2 form entity b.
	Subset	A set a_1 is a subset of a set a_2, that is, all elements of a_1 are also elements of a_2. "•" denotes the place where the connection ends, i.e., a_2 around the "•" is the set, while a_1 is the subset
	Element	An element a_1 is a singleton of the distinct objects that make up that set, S. "•" denotes the place where the connection ends.
	Property	A property a_1 is special quality or characteristic of an entity, S. "•" denotes the place where the connection ends.
	Cause	Influence describes the causal relations between two entities. a_1 causes a_2.
	Imply	Directed implication. When one variable implies another variable, it means dependency exists between the two variables (say a_1 implies a_2). Such dependency allows one to make inference about one variable according to another variable.
	Conflict	Effect b is present when a_1 is in conflict with a_2. The effect b is present only when these two factors (a_1 and a_2) are coupled, and where these two factors have different types of influences (e.g., positive versus negative).
	Trigger	Effect b is caused by "event a_2 Triggering event a_1."
	Human error mode	A general psychological error pattern.
	Context	The conditions contained in a software artifact that tend to trigger a human error mode.
	Top event (software defect)	The ultimate result (i.e., software defect) produced by the interactions between various contexts and human error modes.

Table 2. Sample notation for causal mechanism graph (Version E for human error analysis).

human error mechanisms underlying software faults. A sample of the CMG notation adapted for human error analysis is shown in **Table 2**.

3.3. An application example

An example of using CMG to perform human error analysis is shown in **Figure 2**. The proposed approach is applied on a software requirement called "Jiong" problem provided in Ref. [13].

A requirement segment is extracted, shown in **Figure 2**. To complete the "Jiong" problem, a programmer first needed to calculate the structure of a "Jiong" using a recursion or iteration algorithms (A.1 in **Figure 2**), and then print a blank line after the word (A.2 in **Figure 2**).

Using HEA, we see that this requirement segment contains three conditions: (1) A.1 is the main requirement; (2) A.2 is not a necessary condition to A.1; (3) A.2 is the last step of A. These three conditions consist a scenario that tends to trigger "postcompletion error." Postcompletion error is an error pattern whereby one tends to omit a subtask that should be carried out at the end of a task but is not a necessary condition for the achievement of the main goal [14].

This requirement was presented to student programmers in a programming contest in the previous study [13]. Results show that 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 psychologies in other tasks.

It is notable that "printing a blank line" is a very simple requirement and have been explicitly specified; this requirement is correct and clear. According to the current requirement quality criteria such as correctness, completeness, unambiguity and consistency, this requirement

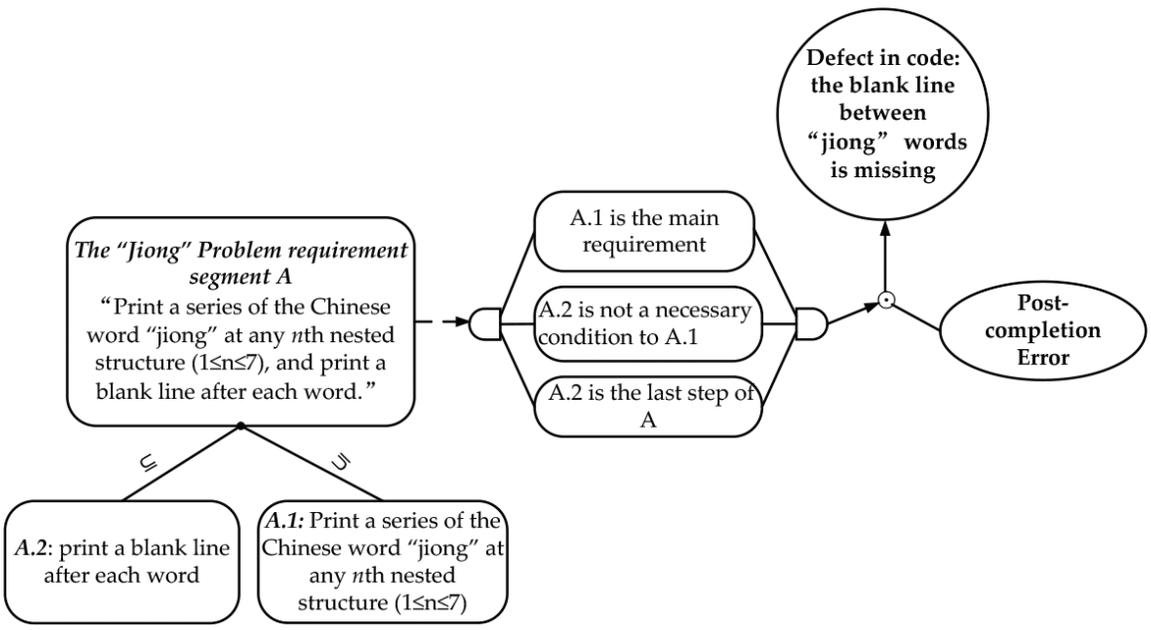


Figure 2. An example of human error analysis.

contains no features prone to trigger a software development error. In fact, this requirement triggered significantly more programmers to commit the error than any of other locations, and amazingly in the same way [13].

Once the error-prone representation is identified, one can use strategies to prevent it from triggering development errors. For instance, the requirement writer may highlight (e.g., using bright colors and/or bold font) the places of postcompletion tasks in the requirement documents ("printing a blank line after each word" in the "Jiong" case), since visual cues are an effective way to reduce postcompletion errors [25]. Though using styles to facilitate readers' cognitive process is not new in software requirement engineering, the contribution here is to tell the writer the exact location that should be highlighted, in order to reduce a developer's error-proneness.

4. Conclusion

This chapter emphasizes the necessity of understanding the cognitive nature of software and software faults, and reviews the emerging area of defending against software defects based on human error theories (SFDHE). An approach of human error analysis (HEA) is proposed to detect and/or prevent software defects at various stages of the software development life cycle. The application on a requirement review shows that HEA is able to identify an error-prone scenario that can never been captured by any existing criteria for requirement quality. HEA offers a promising perspective to advance the current practices of software fault detection and prevention.

Author details

Fuqun Huang

Address all correspondence to: huangfuqun@gmail.com

Institute of Interdisciplinary Scientists, Seattle, Washington State, USA

References

[1] Détienne F. Software Design–Cognitive Aspects. New York, NY: Springer-Verlag New York, Inc.;2002

[2] Huang F, Liu B, Huang B. 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

[3] Weinberg GM. The Psychology of Computer Programming. VNR Nostrand Reinhold Company; New York: 1971

[4] Huang F, Liu B. Systematically Improving Software Reliability: Considering Human Errors of Software Practitioners. In: 23rd Psychology of Programming Interest Group Annual Conference (PPIG 2011), York, UK; 2011

[5] Huang F. Software Fault Defense based on Human Errors. Ph.D., Beijing: School of Reliability and Systems Engineering, Beihang University; 2013

[6] Visser W. Dynamic Aspects of Design Cognition: Elements for a Cognitive Model of Design. France: INRIA; Research Report 2004

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

[8] Huang F, Liu B, Wang S, Li Q. The impact of software process consistency on residual defects. Journal of Software: Evolution and Process. 2015;**27**:625-646

[9] Huang F, Liu B, Wang Y. Review of Software Psychology (in Chinese). Computer Science. 2013;**40**:1-7

[10] Huang F, Liu B. Study on the correlations between program metrics and defect rate by a controlled experiment. Journal of Software Engineering. 2013;**7**:114-120

[11] Anu V, Walia G, Hu W, Carver JC, Bradshaw G. Using a Cognitive Psychology Perspective on Errors to Improve Requirements Quality: An Empirical Investigation. In:Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on; 2016, pp. 65-76

[12] Reason J. Human Error. Cambridge, UK: Cambridge University Press; 1990

[13] Huang F. Post-completion Error in Software Development. In The 9th International Workshop on Cooperative and Human Aspects of Software Engineering, ICSE 2016Austin, TX, USA; 2016, pp. 108-113

[14] Byrne MD, Bovair S. A working memory model of a common procedural error. Cognitive Science. 1997;**21**:31-61

[15] Card DN. Myths and Strategies of Defect Causal Analysis. In: Proceedings of the Twenty-Fourth Annual Pacific Northwest Software Quality Conference; 2006, pp. 469-474

[16] Mohammadnazar H. Improving Fault Prevention with Proactive Root Cause Analysis (PRORCA method). 2016

[17] Huang B, Ma Z, Li J. Overcoming obstacles to software defect prevention. International Journal of Industrial and Systems Engineering. 2016;**24**:529-542

[18] Huang F, Liu B. Software defect prevention based on human error theories. Chinese Journal of Aeronautics. 2017. In Press

[19] Knight JC, Leveson NG. An experimental evaluation of the assumption of independence in multi-version programming. IEEE Transactions on Software Engineering. 1986; **12**:96-109

[20] Avzenis A, Lyu MR, Schutz W. In search of effective diversity: A six-language study of fault-tolerant flight control software. In: Proceedings of the 18th International Symposium on Fault-Tolerant Computing, Tokyo, Japan; 1988, pp. 15-22

[21] Li Y, Li D, Huang F, Lee SY, Ai J. An Exploratory Analysis on Software Developers' Bug-introducing Tendency Over Time. In: The Annual Conference on Software Analysis, Testing and Evolution, Kunming, Yunnan; 2016

[22] Lee SY, Li Y. DRS: A Developer Risk Metric for Better Predicting Software Fault-Proneness. In: Trustworthy Systems and Their Applications (TSA), 2015 Second Inter national Conference on; 2015, pp. 120-127

[23] Huang F, Smidts C. Causal mechanism graph — A new notation for capturing cause-effect knowledge in software dependability. Reliability Engineering & System Safety. 2017;**158**:196-212

[24] Huang F, Li B, Pietrykowski M, Smidts C. Using Causal Mechanism Graphs to Elicit Software Safety Measures. In: 39th Enlarged Halden Programme Group Meeting (EHPG meeting at Sandefjord 2016); 2016

[25] Chung PH, Byrne MD. Cue effectiveness in mitigating postcompletion errors in a routine procedural task. International Journal of Human-Computer Studies. 2008;**66**:217-232

IntechOpen

IntechOpen

IntechOpen