

Software Requirement Criteria based on Human Errors

Fuqun Huang

Institute of Interdisciplinary Scientists

Seattle, WA, USA

Email: huangfuqun@insins.org

Abstract—Software requirement specifications have been observed to largely impact the dependability and the cost of software systems in software development and certification phases. Inappropriate specification of software requirements can cause software developers' erroneous mental representations, thus leading to defects that propagate into subsequent development phases. Understanding the human error mechanisms of software requirement representation is significant for reducing the defects originated from requirements. This paper proposes a theory on the human error mechanism of software requirement, and derived two new criteria to avoid requirement specification triggering the human errors of developers. The criteria were validated by an experiment. Results show that: 1) once a requirement specification contained the error-prone scenarios of the two proposed criteria, developers indeed committed corresponding errors; 2) violating the proposed criteria tended to cause common defects, which are the defects introduced by two or more developers in the same way.

Keywords—software requirement; human error; requirement quality; software certification; N-version programming

I. INTRODUCTION

Understanding how software requirement specifications trigger human errors of developers is significant in software engineering. Software requirement as a primary input to developers, is an essential factor that impacts how developers design and implement the software system, and how they may commit errors. These errors then manifest themselves as software defects and propagate to subsequent phases, leading to an increase of cost on defect detection and fixing, as well as a decrease of dependability and chance of success in software certification.

Established criteria for software requirement specification mainly include unambiguous [1, 2], correctness, completeness and consistency [3, 4]. These criteria have been intensively studied in software requirement engineering with enormous progress made. However, there is little research on promoting requirement representation strategies based on human error theory, though the importance of human factors has been recognized [5].

To the author's best knowledge, there are two research groups who studied human errors associated with software requirement. Anu and Walia et al. developed a human error taxonomy to improve requirement quality [6]. Their research also found that students who received knowledge training on human error taxonomy introduced less defects in writing requirements [7]. Huang has been focusing on uncovering the deep cognitive error mechanisms of how requirement specifications cause developers to commit errors in programming, including software design and implementation [8]. Huang's previous controlled experiment found that a requirement conforming to all the traditional criteria

amazingly triggered many programmers (23 out of 55) commit the same error in the same way [8]. This discovery leads to a more systematic theory presented here:

This paper proposes a model on how requirements trigger human errors of software developers, and draws two new criteria for reducing the human error-proneness of requirement specifications. The criteria are applied and validated in a case study.

II. RELATED WORK

Many factors may contribute to a requirement representation that tends to trigger human errors, for instance, the ambiguity [1, 2] or other deficiencies [9] of the languages used to represent the requirement, the incorrect, incomplete or inconsistent requirements introduced during requirement elicitation or writing process [3, 4]. Features such as granularity [10], text styles [11] and diagrams [12] may also be related to error-proneness through affecting a reader's perception and attention allocation.

The importance of introducing human error theory to software engineering for defending against software defects has been recognized in the recent decade. Ko and Myers introduced human error theories to investigate how inappropriate design in programming tools cause programmers' cognitive errors, for the improvement of human-computer interface design [13]. Huang and Liu first introduced human error theories to software engineering, and proposed a framework that contains a series of new methods to defend against software defects based on human errors [14]. Huang et al. conducted a series in-depth studies on defect prevention based on human errors [15, 16], defect prediction based on human errors [17, 18], defect detection strategies [19], and diversity design [20]. Anu, Walia, Hu, Bradshaw and Carver has been focus on using human error taxonomy to improve requirement quality [6]. Nagaria and Hall [21] interviewed developers about the situations of the skill-based errors and how they mitigate such errors. Couceiro and Madeira et al. [22] developed techniques to identify suspicious code by monitoring programmers' biometrics.

Despite the interesting results obtained in this emerging field, the current studies on human errors in software requirement engineering are either limited to taxonomies or in-depth study on one single error mode [8]. There still lacks a systematic theory on how *requirements* trigger human errors of *developers*, and the criteria for reducing the error-proneness of requirement specifications.

III. CONCEPTS AND THEORIES

A. Concepts

The concepts used in this paper are defined as follows:

Defect: an incorrect or missing step, process, or data definition in a computer program (adapted from [23]).

Error: an erroneous human behavior or cognitive process that leads to a software defect. Errors are classified at a finer-grained level in psychology as mistakes, slips or lapses [24].

Error Mode (EM): a particular pattern of erroneous behavior that recurs across different activities, due to the cognitive mechanisms shared by all humans [24].

Error-prone scenario: the circumstances under which a human error tends to be triggered. An error-prone scenario in software requirements could be a specific form of a single specification item; it could also be a specific combination or structure of multiple specification items.

B. Huang's theory on Human Error Mechanism of Software Requirement

The author proposes a model of how software requirement triggering the human errors of developers is shown in Figure 1. The model consists of the following principal components: requirement specification, error-prone scenarios, and software defects. The theory underlying the model is:

When a snippet of conditions contained in the requirement specification matches an Error-prone Scenario of a Human Error Mode, it tends to trigger a developer to introduce a defect corresponding to the HEM. Some defects can be caused by one single Error-prone Scenario (EPS), while others may be introduced by a combination of several EPSs.

The theory is inspired from a core philosophy in James Reason's work: "errors take a limited number of forms" [24]. Human errors may appear enormously diverse in different contexts, but the cognitive mechanisms that govern them are limited and have largely been validated by psychologists. Reason has summarized the error patterns that have been validated by psychologists before 1990 in his book [24]. With this belief in mind, Huang tested Byrne and Bovair's Post-Completion Error [25], an error mode that had not been discovered yet at the time of Reason's book [24], in software development context [8]. The experimental result shows that the error occur just as Reason and Huang's theories expect.

Due to the fact that the psychological theories are difficult to apply for practical purposes, in this paper Huang proposes the new concept Error-Prone Scenario (EPS) to model the contexts that tend to trigger a human error, and represent the EPSs in a structural way using pseudo codes. Thus, an EPS is the "bridge" that connects the knowledge of human errors in psychology domain to a defect occurred in a developer's specific development context, forming a coherent and completed causal map of why and how the defect is caused by the interaction between cognitive error and the context.

Based on Huang's theory on Human Error Mechanism of Software Requirement, *software requirement specification should avoid forming Error-prone Scenarios, or amended strategies should be used if an Error-prone Scenario is unavoidable.*

IV. HUMAN ERROR-BASED CRITERIA FOR SOFTWARE REQUIREMENT SPECIFICATION

Two Error-prone Scenarios should be avoided in software requirement specifications:

A. Post-completion Error-prone Scenarios

Error mode: Post-completion error [25]

Error Mode Description: If the ultimate goal is decomposed into sub-goals, a sub-goal is likely to be omitted under the following conditions: the sub-goal is not a necessary condition for the achievement of its super-ordinate goal, and the sub-goal is to be carried out at the end of the task. One well-known example is that, if the work flow of an Automatic Teller Machine (ATM) is designed as that withdrawing bank card is the last step, people are prone to leave after taking the cash but forget to withdraw their card from the ATM.

Error-prone scenario:

```

IF      Task A = {Task A.1, Task A.2}
WHEN    <Task A.1 is the main subtask>,
          AND
          <Task A.2 is not a necessary condition to Task A.1>,
          AND
          <Task A.2 is the last step of Task A >;
THEN    Humans tend to omit Task A.2.

```

B. Avoiding Selectivity Error-prone Scenarios

Error mode: Selectivity

Error Mode Description: Psychologically salient, rather than logically important task information is attended to [24]. "Selectivity" in programming means that when a developer is 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 develop solutions. For example, if there are several requirement items scattered at different places in a document, for a single task, developer may fail to notice some information and perceive a wrong representation of the task.

Error-prone scenario:

```

IF      Current task contains features
          FeTi = {Saliencei, LogicImportancei}
          and FeTj = {Saliencej, LogicImportancej}
WHEN    Saliencei > Saliencej;
THEN    People tend to notice a FeTj;
WHEN    LogicImportancei > LogicImportancej;
THEN    An error tends to be introduced due to
          the omission of FeTi.

```


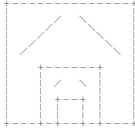
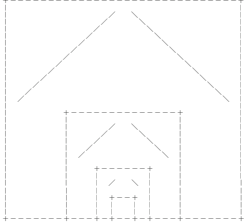
V. APPLICATION AND VALIDATION

This section demonstrates how to apply the proposed two criteria in software development contexts, and validate the criteria. The demonstration will focus on how to match the EPSs to a specific requirement context (in subsection B). The validation concerns whether these EPSs triggered developers to commit corresponding errors in the real development case (in subsection C).

A. The Requirement Specification

A programming task called the “Jiong” problem. The requirement of the “Jiong” problem is shown in the second column of Table I. Note that the first column of Table I (the line numbers of the specification) is added by the author for the displaying purpose for the reader of this paper only, it was not in the original requirement specification.

TABLE I. THE REQUIREMENT SPECIFICATION OF THE “JIONG” PROBLEM

Line1	Print a Chinese word “jiong” in a nested structure.
L2	Inputs
L3	There is an integer in the first line that indicates the number of input groups.
L4	Each input group contains an integer n ($1 \leq n \leq 7$).
L5	Outputs
L6	Print a word “jiong” after each input group, and then print a blank line after each “jiong” word.
L7	Sample Inputs
L8	3
L9	1
L10	2
L11	3
L12	Sample Outputs
L13	
L14	
L15	
L16	
L17	
L18	

B. Matching the Requirement Specification to Error-Prone Scenarios

1) Post-completion error

In the process of solving the “jiong” problem, the developers’ ultimate aim (superordinate goal) is to compute and print the image of the “jiong”. The task of printing a blank line after the “jiong” is a subgoal, but it is carried out at the end of the task. This is a typical scenario of post-completion error, where the subgoal is likely to be omitted, thus, developers may forget to print the blank line after the “jiong”. How the conditions in the “jiong” requirement matches the error-prone scenario of “Post-completion Error” is demonstrated in Figure 2.

Defect: the blank line after a “jiong” is missing.

2) Selectivity

How the conditions in the “jiong” requirement matches the error-prone scenario of Selectivity is demonstrated in Figure 3.

There are two passages in the requirement specification that contain information on the formats of inputs and outputs. The first passage is located under the bullets “**Inputs**” and “**Outputs**”. Under the bullet “Input”, the format of input is specified as “there is an integer in the first line, which indicates the number of input groups. Each input group contains an integer n ($1 \leq n \leq 7$).” Under the bullet “Output”, the format of output is specified as “print out a word ‘jiong’ after each input group, and then print out a blank line.”

The other passage relevant to the formats of inputs and outputs is located under “**Sample inputs**” and “**Sample outputs**”. Using “Sample inputs” and “Sample outputs” is a conventional part of specification in the programming contest, widely used across different tasks and different annual contests. The “Sample inputs” and “Sample outputs” here are static: they cannot reveal the required sequencing of inputs and outputs. However, if the participants only notice information under the “Sample inputs” and “Sample outputs”, they may interpret the specification in two ways: print all the “jiong” words together after reading in all the inputs; print each “jiong” word once an integer indicating the nesting level of the corresponding “jiong” word is input.

This typical scenario is prone to trigger the error mode of “Selectivity”. For the same task content, pieces of information are scattered at different places. Under such circumstances, it is hard for a person to collect all the information, exclude redundant information and form an accurate mental representation. As a result, one may selectively notice only a fraction of the whole body of information which are psychological salient, and thus commit errors. Under such situation, some developers may just notice the information of “Sample inputs” and “Sample outputs” as they are visually presented, and visual representations are generally more likely to capture people’s attention, and thus, wrongly interpret the specification as “print all the ‘jiong’ words together after reading all of the inputs.”

Defect: The “jiong”s are printed together only after all of the inputs have been entered.

C. Validation

1) Background

The “jiong” requirement specification was presented to developers in a programming contest. A total of 55 students in computer science submitted programs for the “jiong” problem. The study here on human error-based requirement specification criteria was combined with other studies with different theories and subsets of data. The other studies include the study on the correlation between cognitive styles, personality traits and the number of defects a developer tends to commit [20], and defect prediction based on Human errors [8, 17].

The author needs to emphasize that integrating multiple lines of studies into a large well-designed data collection procedure is very important for research of human factors in software engineering, especially when a researcher aims to discover causal mechanisms. As we know, there are many factors that influence a person’s programming performance. Some of the factors are correlated factors; some factors are causal but random and unpredictable; the most significant factors are those that are causal and predictable. A large dataset (containing different dimensions of information) produced by the same group of developers and collected in a same procedure form a baseline, which is essential to

comparing various factors, and testing the performances of various theories.

The detailed programming environment setting and all the defect data can be found in [20]. This paper only presents the data relevant to the topic of this paper, shown in Table II.

TABLE II. THE EXPERIMENTAL RESULTS

Error Mode	Defect Description	OC	ROC	Average ROC
Post-completion Error	The blank line after the “jiong” is missing	23	41.2%	22.7%
Selectivity	The “jiong”s are printed together only after all of the inputs have been entered.	2	3.6%	

2) Metrics

The metrics in Table II are defined as follows:

- **Occurrences (OC)** of a defect refer to the number of participants who introduced the defect.
- **Rate of Occurrences (ROC)** is calculated by the Occurrences of a defect divided by the total number of participants, which is 55 in this case. ROC measures the extent of how commonly a defect is introduced by different developers.
- **Average ROC** measures the average extent of how commonly a set of defects are committed by different developers. Average ROC is calculated by:

$$\frac{\sum_i ROC_i}{n} \quad (1)$$

where ROC_i is the Rate of Occurrences for error i , n is the total number of errors in the group of interest. In this case, we can compare the Average ROC of the defects associated with the proposed Error-prone Scenarios to that of the defects caused by skill-based errors. Skilled-based errors are the errors occurring in skill-based performance, which follows from the statement of an intention, “rolls along” automatically without conscious control. Examples of skill-based activities in programming include typing a text string, or compiling a program by pressing a button in the programming environment. Skilled-based errors are caused by environmental factors that influence a developer’s attention, such as interruption. Huang considers the skilled-based errors in programming are unpredictable at the requirement phase. The experiment results showed that all of the skill-based errors (e.g. typo errors) were committed by a single developer.

3) Results

The experiment show that 23 out of 55 (41.8%) developers introduced the defect of “forgetting to print a blank line after each word” corresponding to the “post-completion error”, in the same way as observed by psychologists in other tasks. Meanwhile, 2 out of 55 (3.6%) developers introduced the defect associated with the Selectivity error.

The results confirmed that: 1) once a requirement specification contained the error-prone scenarios of the two proposed criteria, developers indeed committed corresponding errors, in the ways as Huang’s theory expected; 2) violating the proposed criteria tended to cause common defects, which are the defects introduced by two or more developers in the same way.

It is also interesting to compare the occurrences of these two defects to the defects caused by skilled-based errors.

The huge difference between of the Rate of Occurrence of the two criteria-related defects (22.7% on average) and that of skill-based defects (1.8%) suggests that the proposed theory and criteria seem to have captured the mechanisms of how requirement specification trigger developers’ human errors.

VI. DISCUSSIONS

A. Contributions

This paper proposed a theory on the human error mechanism of software requirement, developed and validated two new human error-based criteria for software requirement specification, in addition to the existing criteria of correctness, completeness, unambiguity and consistency.

These human error-based criteria have not been covered by current requirement quality guidelines, whereas seem to deserve the attention of software practitioners. It is notable that “printing a blank line” in the “jiong” requirement 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 contains no features prone to trigger a software development error. In fact, this requirement triggered significantly more developers to commit the error than any of other locations, and amazingly in the same way.

B. Implications

Once the error-prone representation is identified, one can use strategies to prevent it from triggering development errors. For instance, the requirement writer may consider changing the procedure of the requirement specifications if possible. Another simple strategy is to highlight (e.g. using bright colors and/or bold font) the places of post-completion 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 post-completion errors [26]. Though using styles to facilitate readers’ cognitive process is not new in software requirement engineering, the contribution here is to tell the writer exactly where should be highlighted in order to reduce the readers’ error-proneness.

The criteria can also be added to the checklist for software verification, validation and certification activities. These new criteria should help one to identify requirement flaws that may not be covered by traditional criteria such as ambiguity and inconsistency. These new criteria could also help code reviewers identify software design and coding defects that originated from requirements.

C. Follow-up Studies

This paper proposed a model on the human error mechanisms of software requirement, and developed two criteria of software requirement specifications to avoid triggering human errors of developers. The paper also demonstrated and validated the criteria in a requirement specification. Future studies will be focused on how the criteria manifested themselves in industrial projects based on the data collected in formal third-party software certifications. A more comprehensive report on the study will follow up.

Another follow-up article will be on N-version programming. This paper developed a theory that is

significant for N-version programming: under what situations different programmers tend to introduce the same error for the same requirement. Avoiding such situations would be beneficial for improving the fault tolerance of safety-critical systems that employ N-version programming.

VII. CONCLUSION

This paper proposed a theory on Human Error Mechanism of Software Requirement and two criteria for software specifications to reduce the likelihood of triggering developers' human errors. The paper also developed a new concept, error-prone scenario, to map a specific requirement context to human error patterns that psychologists have established.

The proposed idea seems promising in identifying requirement flaws that are not covered by traditional criteria, and helping code reviewers identifying program defects that originated from requirement. Follow-up studies will focus on how the proposed criteria manifest themselves in industrial projects.

REFERENCES

- [1] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry, "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," *Requirements Engineering*, vol. 13, pp. 207-239, 2008.
- [2] A. Ferrari, G. Lipari, S. Gnesi, and G. O. Spagnolo, "Pragmatic ambiguity detection in natural language requirements," in *Artificial Intelligence for Requirements Engineering (AIRE), 2014 IEEE 1st International Workshop on*, 2014, pp. 1-8.
- [3] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 14, pp. 277-330, 2005.
- [4] M. Sabetzadeh and S. Easterbrook, "View merging in the presence of incompleteness and inconsistency," *Requirements Engineering*, vol. 11, pp. 174-193, 2006.
- [5] K. Lauenroth and E. Kamsties, "People's Capabilities are a Blind Spot in RE Research and Practice," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2016, pp. 243-248.
- [6] V. Anu, G. Walia, W. Hu, J. C. Carver, and G. Bradshaw, "Using a cognitive psychology perspective on errors to improve requirements quality: An empirical investigation," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016, pp. 65-76.
- [7] W. Hu, J. C. Carver, V. Anu, G. S. Walia, and G. L. Bradshaw, "Using human error information for error prevention," *Empirical Software Engineering*, vol. 23, pp. 3768-3800, 2018.
- [8] F. Huang, "Post-completion Error in Software Development," in *The 9th International Workshop on Cooperative and Human Aspects of Software Engineering, ICSE 2016, Austin, TX, USA, 2016*, pp. 108-113.
- [9] M. Glinz, "Problems and deficiencies of UML as a requirements specification language," in *Proceedings of the 10th International Workshop on Software Specification and Design*, 2000, p. 11.
- [10] A. Rashid, A. Moreira, and J. Araújo, "Modularisation and composition of aspectual requirements," in *Proceedings of the 2nd international conference on Aspect-oriented software development*, 2003, pp. 11-20.
- [11] E. Kamsties, A. von Knechten, and R. Reussner, "A controlled experiment to evaluate how styles affect the understandability of requirements specifications," *Information and Software Technology*, vol. 45, pp. 955-965, 2003.
- [12] D. L. Moody, P. Heymans, and R. Matulevicius, "Improving the effectiveness of visual representations in requirements engineering: An evaluation of i* visual syntax," in *2009 17th IEEE International Requirements Engineering Conference*, 2009, pp. 171-180.
- [13] A. J. Ko and B. A. Myers, "A framework and methodology for studying the causes of software errors in programming systems," *Journal of Visual Languages & Computing*, vol. 16, pp. 41-84, 2// 2005.
- [14] F. Huang and B. Liu, "Systematically Improving Software Reliability: Considering Human Errors of Software Practitioners," in *23rd Psychology of Programming Interest Group Annual Conference (PPIG 2011)*, York, UK, 2011.
- [15] 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.
- [16] F. Huang and B. Liu, "Software defect prevention based on human error theories," *Chinese Journal of Aeronautics*, vol. 30, pp. 1054-1070, 2017.
- [17] F. Huang and L. Strigini, "Predicting Software Defects Based on Cognitive Error Theories," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2018, pp. 134-135.
- [18] F. Huang, "Human Error Analysis in Software Engineering," in *Theory and Application on Cognitive Factors and Risk Management-New Trends and Procedures*, ed: InTech, 2017.
- [19] Y. Li, D. Li, F. Huang, S.-Y. Lee, and J. Ai, "An Exploratory Analysis on Software Developers' Bug-introducing Tendency Over Time," in *The Annual Conference on Software Analysis, Testing and Evolution*, Kunming, Yunnan, 2016.
- [20] 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.
- [21] B. Nagaria and T. Hall, "How Software Developers Mitigate their Errors when Developing Code," *IEEE Transactions on Software Engineering*, 2020.
- [22] R. Couceiro, R. Barbosa, J. Durães, G. Duarte, J. Castelhana, C. Duarte, et al., "Spotting Problematic Code Lines using Nonintrusive Programmers' Biofeedback," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, 2019, pp. 93-103.
- [23] I. S. Board, "IEEE Standard Glossary of Software Engineering Terminology," vol. IEEE Std 610.121990, ed. New York, USA: The Institute of Electrical and Electronics Engineers, 1990.
- [24] J. Reason, *Human Error*. Cambridge, UK: Cambridge University Press, 1990.
- [25] M. D. Byrne and S. Bovair, "A working memory model of a common procedural error," *Cognitive science*, vol. 21, pp. 31-61, 1997.
- [26] P. H. Chung and M. D. Byrne, "Cue effectiveness in mitigating postcompletion errors in a routine procedural task," *International Journal of Human-Computer Studies*, vol. 66, pp. 217-232, 2008.

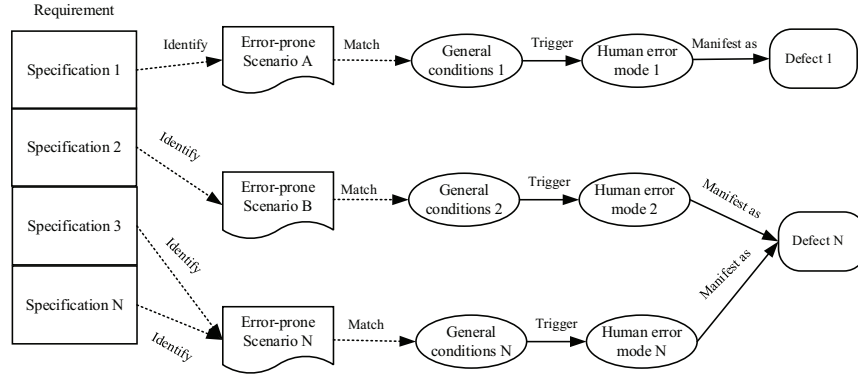


Figure 1. Huang's model on Human Error Mechanism of software requirement

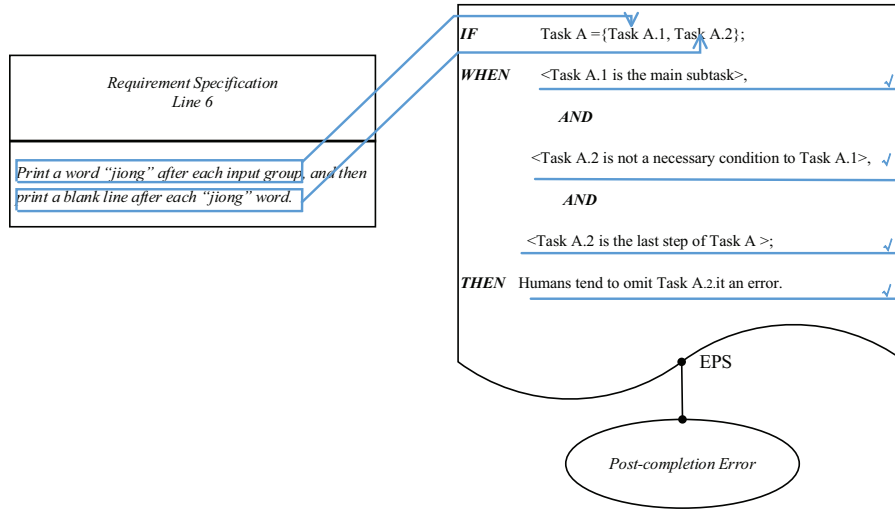


Figure 2. Identifying the Post-completion Error-prone Scenario in the "jiong" requirement specification

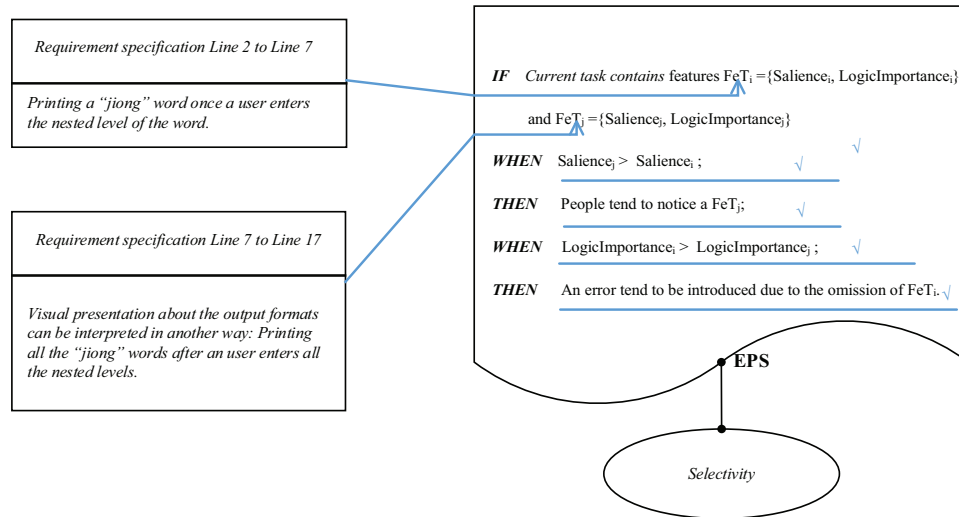


Figure 3. Identifying the Selectivity Error-prone Scenario in the "jiong" requirement specification