Full: A New Graduate Course Structure for Addressing Human Errors in Software Development

Fuqun Huang
Department of Computer Science
Western Washington University
Bellingham, USA
huangf2@wwu.edu

***Abstract*— This research-to-practice full paper introduces a novel graduate course, Human Errors in Software Development (HESD) based on a metacognition framework. Software defects pose a significant threat to the reliability and safety of computer systems, incurring trillions of dollars in costs globally. Addressing and rectifying defects in programs present a formidable challenge for Computer Science (CS) students. Given the critical role of cognition in software development, there is a pressing need for a course designed to train students to address human errors in various cognitive activities of software development. To our knowledge, there is currently no semester-long nor quarter-long university course offering comprehensive training to students on handling human errors in software development.**

HESD equips students with a profound understanding of the cognitive mechanisms underlying human errors in software development. It aims to enhance students' awareness and cognitive abilities to proactively prevent human errors in software development and, consequently, reduce defects in programs they develop. HESD comprises three stages: Stage I provides students with explicit knowledge of human errors in software development; Stage II fosters students' awareness and regulation abilities to effectively address human errors during software development; Stage III encourages students to apply acquired knowledge and skills in diverse contexts.

The newly designed course was delivered to master's students at a large public university over a 14-week semester. Nine students in Software Engineering were enrolled in the course. Comprehensive surveys were used to evaluate the course's attractiveness and usefulness to students. The average satisfaction score was 4.2 (Min = 3, Max = 5, SD = 0.8) in a five-point Likert scale (where 1 means "very dissatisfied" and 5 means "very satisfied"), signifying that the students were quite satisfied with the course. A survey designed for assessing human error knowledge, awareness and regulation ability was filled out by the students before the course and at the end of the course. Results showed that the course has significantly enhanced students' Human Error Knowledge in software development by 153%, improved Human Error Awareness in software development by 62%, and elevated students' Human Error Regulation by 63%. The students found the course to be highly interesting, practical, and valuable.

Keywords—computer science curriculum, human error, software development, cognition

I. INTRODUCTION

Programming is a cognitive-intensive activity [1], with human errors playing a significant role in causing the defects (or bugs) in programs [2]. Identifying and fixing defects is one of the most challenging hurdles for students learning programming [3]. To ease these barriers and enhance students grasp of programming, it is crucial to equip them with methods and strategies to address human errors in software development.

While human errors has a substantial impact on software defects, their exploration within the realm of Computer Science Education (CSE) has been largely neglected. In CSE, the term "error" is commonly studied as incorrect snippets of source code [4] (equivalent to "defects" in this paper) or within the context of "compiler error messages indicating incorrect status of program executions [5, 6]. The human errors discussed in this paper are one category of root causes of defects, falling within the purview of psychology.

In the software engineering domain, there have been several studies aimed at enhancing software quality, incorporating human error theories to improve software diversity [7], forecast defects [8], and review software requirements documents [9]. However, the interdisciplinary topic of teaching Computer Science (CS) students the psychological mechanisms behind software defects and cognitive strategies to address human errors in programming remains largely unexplored.

To our knowledge, there is currently no semester-long nor quarter-long university course offering comprehensive training to students on handling human errors in software development, while there are three "short-term" training sessions related to human errors. Anu et al. [10, 11] have proposed a human error taxonomy for software requirements in an educational setting, focusing on writing and reviewing requirements. Huang et al. have conducted a training session on human error prevention [12], but the initiative primarily catered to professional developers. Huang has recently designed a two-week training program for teaching CS graduate students to identify defects in program based on human error theories [13]. These studies encountered challenges arising from the brief duration of training, including the constraints related to the scope of knowledge and the promotion of the trainees' cognitive awareness and regulatory abilities.

It is crucial for CS students to comprehend the underlying mechanisms of human errors causing software defects, enabling

them to develop the necessary awareness and regulatory skills essential for addressing such errors during software development.

This paper reports the experiences of the first university course focused on Human Errors in Software Development (HESD) with master's students at a large public university (University of BLINDED). HESD integrates software engineering methods and psychological knowledge, aiming to reduce software defects in various activities involved in software development. In a new interdisciplinary course, the primary challenge lies in identifying and integrating materials across multiple disciplines. Therefore, this **research-to-practice category full paper** focuses on the course structure (detailed materials can be found in the referred literature or openly accessible after the paper's clearance from the double-blinded review) and the evaluation and feedback received from students after one semester of implementing the course.

II. TERMINOLOGIES

A set of core concepts used in the paper is defined as follows:

Defect is an incorrect or missing step, process, or data definition in a **computer program** (adopted from "fault" in IEEE Standards [14]). It is worth noting that defects in this paper refer to those manifested in the computer program but are not limited to coding defects. These defects may originate from a design or even further upstream, during the process of understanding requirements.

Human Error (HE) refers to an erroneous human behavior that results in a software defect. "Human Error" is a specific area of study in cognitive psychology that focuses on the cognitive mechanisms of human erroneous behaviors, with a representative work by J. Reason [15].

Human Error Mode (HEM) refers to a recurring pattern of erroneous behavior stemming from a shared cognitive mechanism among humans. For instance, "applying 'strong-but-now-wrong' rules [15] is a HEM wherein individuals exhibit a predisposition to apply a rule that has been frequently and successfully employed in past experiences, even if it is less appropriate for the current situation.

Human Error Mechanism refers to the interactions between a HEM and the development context that result in a software defect.

The HESD course uses Rasmussen's performance levels to organize different HEMs. Rasmussen classify cognitive activities into three levels: Skill-based (SB), Rule-based (RB) and Knowledge-based (KB) performances. Different performance levels have different cognitive characteristics, thus have different error modes [15].

Skill-based (SB) performance follows from the forming of an intention and "rolls along" automatically without conscious control. Skill-based activities in programming include typing a text string, compiling a program by pressing a button in the programming environment.

Rule-based (RB) performance is applicable for tackling familiar problems. It is typically controlled by stored rules or procedures that have been derived from a person's experiences. The mind matches patterns in the situation at hand to the preconditions for such stored rules, allowing quick selection of actions. In programming, there are many rule-based performances, such as programming the printing of a string line, and defining a variable in one's familiar programming language.

Knowledge-based (KB) performance comes into play when one faces novel situations, and no rules are available from previous experiences. At this level, actions must be planned using an analytical process. Errors at this level arise from resource limitations and from incomplete or incorrect knowledge. In programming, cognitive performances such as constructing the mental model of the system so as to understand a specific programming task, and trying to figure out a solution for a novel problem are knowledge-based performance.

III. THE COURSE STRUCTURE

The HESD course was designed based on a metacognition framework, with the ultimate goal of enhancing students' awareness and regulation of their own cognitive processes related to human errors.

Metacognition is the process of reflecting on one's own thinking, encompassing the awareness and regulation of one's cognitive processes [12, 16].

Meta-cognition enhances cognitive task performance, a key trait in high-performing individuals across learning, problem-solving, and engineering design [17-20]. Researchers use meta-cognition training to improve cognitive performance such as learning ability [21, 22], math problem-solving [23-26] and engineering design [20, 27]. Huang and Liu [12] effectively used a meta-cognition framework to train professional programmers in preventing human errors in software development. Despite its focus on professional programmers and a limited scope, their study suggests meta-cognition as a reliable framework for designing training programs to mitigate human errors in software development. Consequently, the designer of the HESD course (the author of this paper) adopted meta-cognition as the fundamental framework for content design and organization.

Additionally, the designer enhanced the framework by emphasizing the specific domain of human errors in software development. Specifically, HESD prioritizes training students in three key aspects:

Human Error Knowledge: This entails explicit knowledge about human error theories in psychology, encompassing categories of human error modes, the cognitive mechanisms behind these modes [15], and the latest knowledge of how these modes manifest as software defects in programs [8].

Human Error Awareness: This refers to the mental state of recognizing a situation in which a human error has been committed (for error detection) or realizing a situation in which one is prone to committing an error (for error prevention).

Human Error Regulation: This involves the activity of using appropriate strategies to avoid or prevent human errors in software development [12].

The Scope of Human Errors in this paper: Human errors can be committed by individuals engaged in various activities.

This paper primarily focuses on errors committed by student programmers in **software development**, including understanding requirements, design, and coding.

Fig.1 illustrates the structure of the HESD course based on the meta-cognition of human errors framework. The HESD course comprises three stages aimed at achieving incremental improvements in students' abilities to address human errors in software development. Stage I emphasizes explicit knowledge, Stage II promotes students' awareness and ability to regulate human errors in software development, and Stage III trains students to apply the acquired knowledge and skills to new contexts.

A. Stage I-Acquiring Human Error Knowledge in Software Development

The main goal of Stage I is to convey cutting-edge knowledge in Human Error in Software Development (HESD) to students. As this knowledge is explicit, the primary teaching method involves in-class lectures. However, owing to the interdisciplinary nature of this field, grasping the concepts and theories can be challenging for students. To enhance student engagement and facilitate a better understanding of human error theories in psychology, videos and interactive games are incorporated into lectures. Examples include confirmation bias card games and working memory capacity tests [12].

Stage I consists of six units, with one week allotted for each unit. The details are as follows:

Unit 1 (Week 1): Fundamental related Concepts and Theories in Software Engineering (SE)

This unit delves into fundamental concepts and theories related to software defects in SE. It introduces key terminologies such as defect, error, and failure, along with metrics like defect density, failure rate, program metrics, and software reliability metrics. Established practices in the SE domain are explored, drawing from well-established standards [14, 28] and the software reliability handbook [29].

Unit 2 (Week 2): Software Design Cognition

Focusing on the "correct side" of software design cognition, this unit explores how software engineers think during software development. Key concepts, including schemata and cognitive models [30], are discussed. A recommended reading assignment is Visser's report on the dynamic aspects of design cognition [31].

Unit 3 (Week 3): Fundamental Human Error Theories

This unit provides an overview of general human error theories established by psychologists, with Reason's book "Human Errors" [15] serving as the primary source. Reason's work is widely accepted in the emerging area of human error causes for software defects, offering a systematic exploration of why humans commit errors.

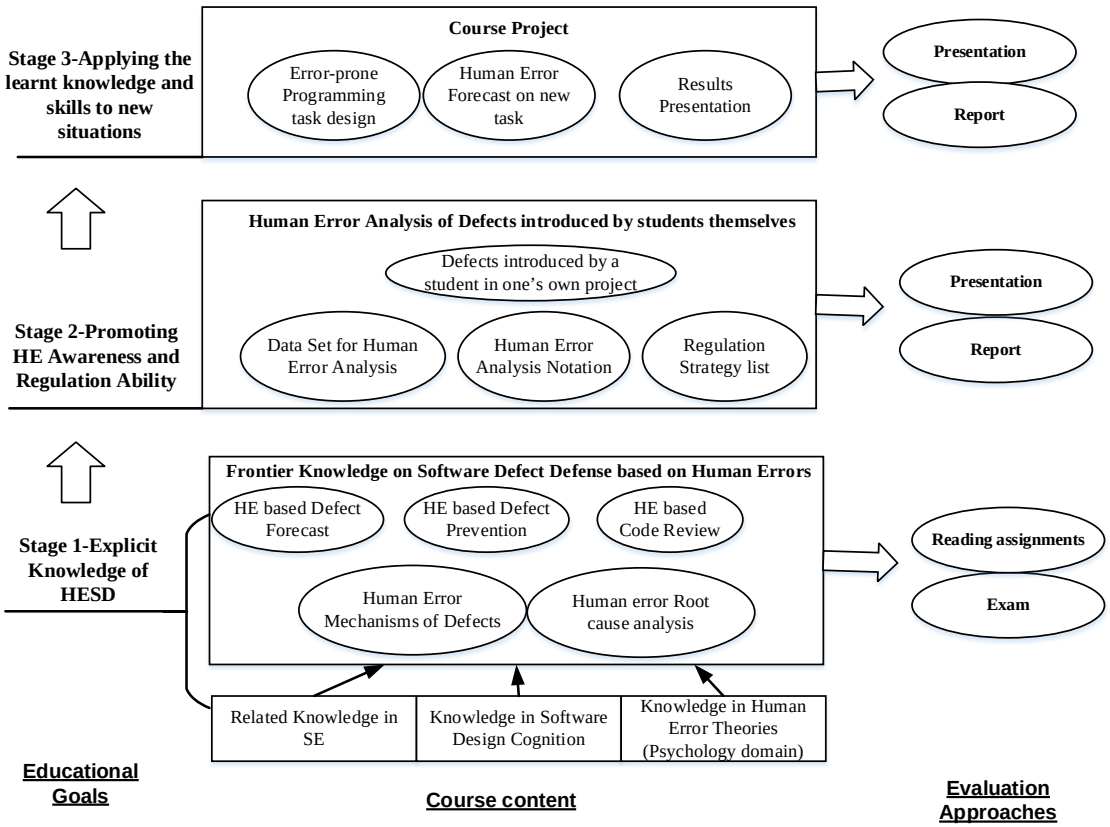

Fig. 1. The HESD course structure

Unit 4 (Week 4): Frontiers on Human Error Mechanisms of Software Defects

This unit focuses on how human errors manifest themselves as software defects, i.e., the human error mechanisms of software defects . It presents the key theories on Human Error Modes, Error-Prone Scenarios (conditions under which a HEM tends to occur) [8], Human Error Analysis, and examples of defects caused by human errors are presented [32].

Unit 5 (Week 5): Frontiers on Software Defect Forecast and Prevention based on Human Errors

This unit instruct students on improving situational awareness and employing suitable strategies to prevent human errors during software development. The main sources for this lecture are [12] and [8].

Unit 6 (Week 6): Frontiers on Code Reviews based on Human Errors

This unit instructs students in identifying defects by pinpointing the scenarios in code that are prone to triggering errors committed by developers. Exercises during the class utilize sample requirements and code presented in [33].

Knowledge Evaluation (Week 7): Mid-term Examination

In Week 7, a mid-term examination is conducted to assess how effectively students have absorbed the explicit knowledge taught in Units 1-6.

B. Stage II - Promoting Awareness and Regulation Abilities to Address Human Errors in Software Development

While acquiring the explicit knowledge taught in Stage I is crucial, it is not sufficient for students to prevent human errors in their daily work. Consequently, Stage II is designed to elevate students' situational awareness of error-prone contexts and enhance their capability to avoid and detect human errors through appropriate cognitive strategies. This is achieved through field practices that leverage a human error root cause analysis list to identify the human error mechanisms behind real defects introduced in the students' own development projects. Additionally, students apply a list of cognitive error regulation strategies [12] to their ongoing development tasks. Hands-on training and field practice are the primary teaching methods in this stage, which is covered in Unit 7 lasting for three weeks.

Unit 7 (Weeks 8-10): Field Practices for Improving Awareness and Regulation Ability in Tackling Human Errors in Software Development

This unit offers hands-on training on human error analysis for defects introduced by students in their daily software development work. A template (Table I), comprising software defect description, human error modes, and human error analysis is used to facilitate these activities. Students are encouraged to utilize the learnt strategies for regulating cognitive errors and record the use cases.

The students are encouraged to record a defect by filling in the first column of Table I each time they find a defect. Then they record the SD activity in which they find the defect by filling in the 2nd column and describe the defect in 3rd column of Table I. After that, They select one or more human error modes that caused the defect in the 4th column. Table II provides a sample of the Human Error Modes used in the course. Finally, they provide detailed explanation for why they select the human error modes, and how the SD contexts interact with the human error modes to cause the defect.

TABLE I. THE TEMPLATE AND STUDENTS' SAMPLE OUTPUTS FOR HUMAN ERROR ANALYSIS

Defect ID	SD activities in which the defect appeared	Defect description	Human Error Modes [a]	Human Error Analysis [b]
DM-10	"My own programming experience"	"Accepting dates with wrong format"	"HEM1-Applying 'strong-but-now-wrong' rule AND HEM7-Confirmation bias "	"Developing an API which will receive requests from international clients, one of the inputs was a date. The most common date in format in Portugal is day, followed by month, followed by year. However, for this specific case, the intended format would be month, day and then year."
DM-11	"reviewing other people's software"	"Confusion between 2 similar fields"	"HEM 2-Rule Encoding Deficiencies / HEM 9-Perceptual Confusions"	"Objects that represent contracts have 2 different fields, one representing the dossier number, and another representing the contract number. In different systems and business areas, those 2 fields have different names (authorization number, number, dossier number, contract number). While developing a new functionality to output the contract number, a bug was introduced such that the dossier number was returned instead."
DM-13	"reviewing other people's software"	"Iterating through an array for indices which are out of range"	HEM 3-Lack of knowledge / HEM 7-Confirmation bias	"The goal of the algorithm was to iterate through the first 7 items of a list of values (first week of the month) in order to retrieve and store some information regarding those items. However, it was not mandatory that the list had at least 7 items, which could result in trying to iterate through the first 7 items of a 6 item list (impossible)."

[a.] Please choose one or more Human Error Modes below if possible, and provide detailed descriptions if the causes are not coved by the provided HEMs.

[b.] Please describe how the HEMs interact with other factors to cause a software defect. You can use English to describe and/or the human error mechanism graphs in [32].

TABLE II. A SAMPLE OF HUMAN ERROR MODES

ID	Human Error Mode	Brief Descriptions	Performance Level
HEM1	Applying "strong-but-now-wrong" rule	The more frequently a rule has been used before, the more likely to be retrieved in a similar current situation.	RB
HEM2	Rule Encoding Deficiencies	Relevant rules in one's knowledge base was incomplete or incorrectly represented.	RB
HEM3	Lack of knowledge	The knowledge that is essential for solving a new problem is not available or incomplete.	KB
HEM4	Difficulties with exponential developments	Humans tend to construct linear models when exponential models are required to understand a situation in reality.	KB
HEM5	Selectivity	Psychologically salient, rather than logically important information is paid attention to.	KB
HEM6	Biased review	Humans tend to believe that all possible courses of action have been considered, when in fact only a subset have been considered.	KB
HEM7	Confirmation bias	People tend to seek evidence that could verify their hypotheses rather than refuting them	KB
HEM8	Omissions following interruptions	Original sequence of activity is picked up one or two steps further along after an interruption.	SB
HEM9	Perceptual Confusions	People mistakenly use/apply/fetch/recall object B for object A, because: B looks like A, B is in the expected location of A, or B does a similar job of A.	SB
HEM10	Post-Completion Error	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. Forgetting email attachment is an example.	Cross-level

C. *Stage III-Applying the Learnt Knowledge and Skills to New Contexts*

This stage equips students with abilities to address human errors in software development in new contexts using the knowledge they have gained. Stage III is fulfilled through Unit 8 spanning 4 weeks.

Unit 8 (Week 11-14): Interdisciplinary Course Project

The course project comprises three components. In Weeks 11-12, each student designs a new programming task embedded with three or more Error-Prone Scenarios and shares them in the course repository. In Week 13, each student chooses a task designed by another peer, performs defect early forecasting, and records the predicted defects while explaining the human error mechanisms underlying them. In Week 14, all students submit a report and present their designed programming tasks along with the defect forecasting results. This approach allows every student to compare their defect forecasting results with the Error-Prone Scenarios embedded by the task designer.

IV. COURSE DELIVERY

A. *Prerequisites*

To enroll in this course, students are required to have successfully completed two undergraduate courses, which include an Introductory Programming course and a course on Data Structures.

B. *Students Enrollments*

Nine professional master's students in Software Engineering from the University of XX were enrolled in the course. According to their responses to the entry survey, all participants held a Bachelor's degree in Computer Science or a closely-related field. Each considered themselves professionals in the industry, boasting an average of 3.1 years of full-time industry experience (Minimum (Min) = 1 year, Maximum (Max) = 6 years, Standard Deviation (SD) = 1.6 years). Their average experience in software development was 4.1 years (Min = 0, Max = 7, SD = 2.1 years). In software quality assurance, they averaged 1.6 years of experience (Min = 0, Max = 7, SD = 2.6 years). Notably, two students had management experiences of 1 and 10 years, respectively, while the remaining seven students had no prior management experiences. Three of the participants estimated that they had reviewed 1000-5000 defects from other people, while the other six students focused more on software development.

C. *Evaluations of Students' Performance*

Each of the three stages included an evaluation of the students' performances. Stage I underwent evaluation through an examination because this stage focuses on explicit knowledge learning.

The students' performances in Stage II were assessed based on the quantity and quality of the dataset for human error analysis, as well as the application cases of the error regulation list. This evaluation took the form of a report and presentation.

For Stage III, the evaluation was centered on the quality of project reports and presentations.

All evaluations were graded on a numerical scale from 0 to 20, with 10 as the minimum passing grade. Additionally, grades were assigned according to the equivalent ECTS grading scale (from E to A): E-Sufficient (10-11), D-Satisfactory (12-13), C-Good (14-15), B-Very good (16-17), A-Excellent (18-20).

V. Evaluation and Students' Feedback

The designer of HESD employed multiple approaches to assess the course and gather feedback from the students. The first approach involved using a survey to directly request students to rate the effectiveness and satisfaction of the course, as detailed in Section V.A. To mitigate potential response bias in students' direct assessments, the course designer utilized another survey, outlined in Section V.C, to gather empirical evidence for evaluating the effects of the course on students' awareness and regulation abilities in addressing human errors in software development. Participation in the assessment surveys was voluntary for the students, and they were not provided with any credit incentive for completion. Six out of the nine students voluntarily completed the assessment surveys.

A. Students' Reviews on the Course Effectiveness

The effectiveness of HESD was initially assessed through a survey gauging students' satisfaction, a commonly employed method for evaluating a new course. The instructor utilized five-point Likert scale questions, with results presented in Table III.

TABLE III. The effectiveness of the course in students' views

Questions	Statistics [a]			
	Mean	**Min**	**Max**	**SD**
How interesting do you consider the course?	4.7	4	5	0.5
How important do you consider teaching students about the mechanisms of human errors and error-preventative strategies in software development?	4.7	4	5	0.5
The extent to which the course enriched your knowledge of human errors in SD, compared to the knowledge you already had before?	4.3	4	5	0.5
The extent to which do you think the course has promoted your awareness and regulation of human errors in software development?	4.3	4	5	0.5
Overall	4.5	4	5	0.5

[a] The 5-point Likert scale is defined as 1-Not at all 2-Slightly 3-Moderately 4-Very 5-Extremely

B. Students' Satisfaction with the Course

At the conclusion of the course, students were invited to voluntarily rate their levels of satisfaction. The satisfaction levels were assessed using a five-point Likert scale: 1-Very dissatisfied, 2-Dissatisfied, 3-Neutral, 4-Satisfied, and 5-Very satisfied.

The average satisfaction score was 4.2 (Min = 3, Max = 5, SD = 0.8), signifying that the students were quite satisfied with the course.

C. Evaluating Students' Human Error Metacognition in Software Development

The instructor evaluated the impact of the course on students' metacognitions of human errors, encompassing Human Error Knowledge, Human Error Awareness, and Regulation. She further designed a survey for assessing metacognition of human errors, shown in Table IV.

The survey comprises 7 questions on the 5-point Likert scale, focusing on the quantitative assessment of the metacognition of human errors in software development.

The assessment used a survey commonly employed for evaluating metacognition [24, 34]. The details of the questions are described in Table IV. Q1-Q3 focus on evaluating students' Human Error Knowledge, Q4-Q5 focus on Human Error Awareness, and Q6-7 focus on Human Error Regulation in SD.

The survey was administered at the beginning of the course, and the same survey was completed by students who volunteered at the end of the course, which was 14 weeks later. The students were explicitly instructed not to refer to their initial responses when completing the survey at the end of the course. The instructor deemed 14 weeks long enough for students to forget initial ratings, boosting confidence in response independence.

TABLE IV. The survey for assessing metacognition of human errors

Metacognition dimensions	Questions		Responses on 5-point Likert scale				
			1	***2***	***3***	***4***	***5***
Human Error Knowledge in software development	Q1	Your current knowledge of Human Errors in Psychology	None	Scarce	Basic	Adequate	Abundant
	Q2	Your current knowledge of Programming Cognition	None	Scarce	Basic	Adequate	Abundant
	Q3	Your current knowledge of Human Errors in Software Engineering	None	Scarce	Basic	Adequate	Abundant
Human Error Awareness in software development	Q4	During software engineering activities in the past two weeks, the extent to which were you aware of your own cognitive process (i.e. meta-cognition), e.g., realizing an error-prone situation?	Not at all	Slightly	Moderately	Very	Extremely
	Q5	In the past two weeks, how often have you used cognitive strategies in your software development or other software activities?	Never	Rarely	Sometimes	Often	Always
Human Error Regulation in software development	Q6	During the past two weeks in SE activities, how often had you been aware of a situation in which you were going to commit an error (Error-Prone Scenario), before a software defect was confirmed at the technique levels, e.g. through compiler or incorrect outputs in tests?	Never	Rarely	Sometimes	Often	Always
	Q7	During the past two weeks in SE activities, how often have you used cognitive strategies to prevent errors in software development?	Never	Rarely	Sometimes	Often	Always

The effects of HESD course on students' metacognitions of human errors in software development are presented in Table V.

TABLE V. THE RESULTS OF THE ASSESSMENT ON STUDENTS' METACOGNITION OF HUMAN ERRORS

Metacognition Dimensions		Entry			Exit		
		Mean	*Min*	*Max*	*Mean*	*Min*	*Max*
HE Knowledge	Q1	1.5	1	2	3.3	3	4
	Q2	1.5	1	3	3.8	3	4
	Q3	1.5	1	2	4.2	4	5
HE Awareness	Q4	1.8	1	3	3.7	3	4
	Q6	2.3	1	4	3.0	2	4
HE Regulation	Q5	2.0	1	3	3.0	2	4
	Q7	1.7	1	3	3.2	2	4

The average level of each metacognition dimension at the time of survey completion was calculated using (1):

$$D = \sum_{j=1}^{J} \sum_{i=1}^{I} S_{ji} \Big/ (I \times J) \quad (1)$$

where D is the average score of students' metacognition dimension (i.e. knowledge, awareness, and regulation) of human errors at the time of filling out the survey, *i* indicates the question number, I represents the total number of questions for a metacognition dimension, *j* indicates the students' sequence number, J represents the total number of students, and S_{ji} means the score of the question i given by the student j.

The average improvement for a metacognition dimension was then calculated by equation (2):

$$IMP_d = D_{exit} - D_{entry} \quad (2)$$

The average improvement for each metacognition dimension is shown in Fig. 2.

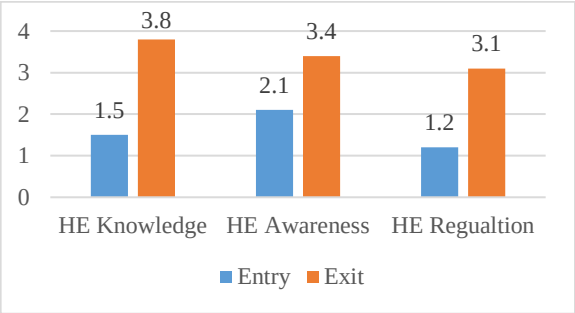

Fig. 2. The impact of the course on the students' Metacognition of human errors in software development

The course has significantly enhanced students' Human Error Knowledge in software development by 153%, increasing from an average of 1.5 to 3.8 on the 5-point Likert scale.

The course has also improved students' Human Error Awareness in software development by 62%, rising from an average of 2.1 to 3.4 on the 5-point Likert scale.

Furthermore, the course has elevated students' Human Error Regulation by 63%, advancing from an average of 1.9 to 3.1 on the 5-point Likert scale.

Overall, the HESD course has been identified as effectively promoting students' metacognition of human errors in software development, with the average score increasing from 1.8 to 3.5 on the 5-point Likert scale. This shift indicates a move from the "never-rarely" range to the "moderately-very" range.

D. Feedbacks through Open Questions

The instructor incorporated three open questions in the Course Exit Survey to gather qualitative feedback from the students for future course improvements. The questions and results are outlined as follows:

1) The contents that students like the most about the course

a) Interesting and Enriching: "The topics of the course are super **interesting**" (Respondent #1, R1); "The premise of the course is most **enriching**" (R6); "First impact of being introduced to human errors and the reasons behind them."(R4)

b) Practical: "I really liked the **practical** lessons where we could put theory in practice." (R3); "The content is to the point, **practical and transversal** to the practice of Software Engineering." (R6)

c) Useful: "the strategies for preventing Human Errors, which are **extremely useful** software engineering"(R2); "Being able to apply what we've learned with real life scenarios such as the tasks for designing a program, or understanding errors in someone else's problem / instructions." (R4); "I've especially liked how the theory gets translated into the practice, so students get to understand how the meta-cognitive process can positively impact the quality of code produced."(R6)

2) The contents that students dislike the most

The students considered the reading assignments in Stage I were a bit heavy: "the amount of articles to read was too much."(R3); "Lectures for explicit knowledge training as in the beginning the readings were a bit heavy."(R4).

The course assumed that enrolled students have programming experience, although some of the professional master's students hold other roles, such as project managers.

"The programming tasks should also take into consideration that some people have no background in engineering."(R1)

"I think that some of the programming tasks were a bit complex for someone who is not that familiar with algorithms."(R4)

3) Students' suggestions for improving the course in the future

The students' suggestions for future improvement tend to focus on the infrastructure and challenges of remoted learning during the Covid-19 period: "The Professor should have a better infrastructure." (R1) "In person classes could improve the learning experience, probably." (R2) "The material exposed to the students should use the university student platform" (R5).

VI. Discussions and Conclusion

This paper introduces a novel graduate course designed to equip students with the skills needed to address human errors in software development. Given that human errors significantly contribute to defective programs authored by students, the proposed course concentrates on enhancing students' abilities to produce reliable programs by fostering their metacognition in human error knowledge, awareness, and regulation strategies. The course was delivered to a cohort of professional master's students at a large public university, demonstrating its effectiveness in advancing students' proficiency in addressing human errors in software development. The students found the course to be highly interesting, practical, and valuable.

In light of the students' feedback, educators should be mindful that the course assumes students possess basic programming experience, and there is room to reduce the load of interdisciplinary reading assignments. Exploring the extension of the course into an elective undergraduate offering is a worthwhile consideration for the near future.

Acknowledgment

The author expresses her gratitude to Prof. Mário Zenha-Rela for his support of delivering the course. The author also appreciates Claudio Mindouro, Daniel Matias, Gabriela Pinhao, Joana Maria, Kuno Schatzer, Luis Oliveira, Paulo Mueller, Victor Carvalho, and Vitor Garcia for enrolling in the course, with special thanks to those who volunteered to share their anonymized outputs.

References

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

[2] 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.

[3] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," *Acm sigcse bulletin*, vol. 37, pp. 14-18, 2005.

[4] D. McCall and M. Kölling, "A new look at novice programmer errors," *ACM Transactions on Computing Education (TOCE)*, vol. 19, pp. 1-30, 2019.

[5] T. Barik, D. Ford, E. Murphy-Hill, and C. Parnin, "How should compilers explain problems to developers?," in *Proceedings of the 2018 26th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2018, pp. 633-643.

[6] J. Prather, R. Pettit, K. H. McMurry, A. Peters, J. Homer, N. Simone, *et al.*, "On novices' interaction with compiler error messages: A human factors approach," in *Proceedings of the 2017 ACM Conference on International Computing Education Research*, 2017, pp. 74-82.

[7] 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.

[8] F. Huang and L. Strigini, "HEDF: A Method for Early Forecasting Software Defects based on Human Error Mechanisms," *IEEE Access*, vol. 11, pp. 3626 - 3652, 2023.

[9] 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.

[10] V. Anu, G. Walia, and G. Bradshaw, "Incorporating human error education into software engineering courses via error-based inspections," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, pp. 39-44.

[11] V. Anu, G. Walia, G. Bradshaw, and M. Alqudah, "Developing and Evaluating Learning Materials to Introduce Human Error Concepts in oftware Engineering Courses: Results from Industry and Academia," in *2019 IEEE Frontiers in Education Conference (FIE)*, 2019, pp. 1-9.

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

[13] F. Huang, "Promoting Students' Cognitive Ability to Identify Human Error-Prone Scenarios in Programs," in *2023 IEEE Frontiers in Education Conference (FIE)*, 2023, pp. 1-9.

[14] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," vol. lEEE Std 610.121990, ed. New York, USA: The Institute of Electrical and Electronics Engineers, 1990.

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

[16] B. Shneiderman and R. Mayer, "Syntactic/semantic interactions in programmer behavior: a model and experimental results," *International Journal of Computer and Information Sciences*, vol. 8, pp. 219-238, 1979.

[17] L. H. Swanson, "Influence of metacognitive knowledge and aptitude on problem solving," *Journal of Educational Psychology*, vol. 82, pp. 306-314, 1990.

[18] S. Bryant, "Rating expertise in collaborative software development," in *Proceedings of the 17th Workshop on the Psychology of Programming Interest Group*, 2005, pp. 19-29.

[19] B. Berardi-Coletta, L. S. Buyer, R. L. Dominowski, and E. R. Rellinger, "Metacognition and problem solving: A process-oriented approach," *Jounral of Experimental Psychology: Learning, Memory, and Cognition*, vol. 21, pp. 205-223, 1995.

[20] R. Hargrove, "Assessing the long-term impact of a metacognitive approach to creative skill development," *International Journal of Technology and Design Education*, pp. 1-29, 2012/01/01 2012.

[21] M. V. J. Veenman, B. H. A. M., and V. H. P. Afflerbach, "Metacognition and learning: conceptual and methodological considerations," *Metacognition Learning*, vol. 1, pp. 3-14, 2006.

[22] C. A. Gama, "Integrating Metacognition Instruction in Interactive Learning Environments," Doctor of Philosophy, University of Sussex, 2004.

[23] S. K. Teong, "The effect of metacognitive training on mathematical word-problem solving," *Journal of Computer Assisted Learning*, vol. 19, pp. 46-55, 2003.

[24] B. Kramarski and Z. R. Mevarech, " Enhancing mathematical reasoning in the classroom: The effects of cooperative learning and metacognitive training," *American Educational Research Journal*, vol. 40, pp. 281-310, 2003.

[25] A. H. Schoenfeld, "Learning to Think Mathematically: Problem Solving, metacognition, and Sense-Making in Mathematics," in *Handbook for Research on Mathematics Teaching and Learning*, D. Grouws, Ed., ed New York: MacMillan, 1992, pp. 334--370.

[26] V. Pennequin, O. Sorel, I. Nanty, and R. Fontaine, "Metacognition and low achievement in mathematics: The effect of training in the use of metacognitive skills to solve mathematical word problems," *Thinking & Resoning*, vol. 16, pp. 198-220, 2010.

[27] O. Lawanto, "Metacognition changes during an engineering design project," in *Frontiers in Education Conference, 2009. FIE '09. 39th IEEE*, 2009, pp. 1-5.

[28] ISO/IEC/IEEE, "Systems and software engineering-Vocabulary," vol. ISO/IEC/IEEE 24765, ed, 2010.

[29] M. R. Lyu, *Handbook of Software Reliability Engineering*. California: IEEE COMPUTER SOCIETY PRESS, 1996.

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

[31] W. Visser, "Dynamic aspects of design cognition: elements for a cognitive model of design," INRIA, France, Research Report2004.

[32] 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.

[33] F. Huang, B. Zhao, and H. Madeira, "A New Code Review Method based on Human Errors," in *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*, 2022, pp. 321-332.

[34] M. Boekaerts and L. Corno, "Self-regulation in the classroom: a perspective on assessment and intervention," *Applied Psychology*, vol. 54, pp. 199-231, 2005.