Error Simulation in a Maturity Environment for Software Engineering Teaching

Luiz Carlos Begosso[1,2] and Luiz Ricardo Begosso[1]
1 Fundação Educacional do Município de Assis (FEMA)
Av. Getúlio Vargas, 1200 Assis, SP Brazil 19807-634
{lbegosso, begosso}@femanet.com.br
http://www.fema.edu.br
2 Faculdade de Tecnologia de Ourinhos (FATEC)
Av. Vitalina Marcusso, 1400 Ourinhos, SP Brazil 19917-206

Abstract. The software industry blames universities for their graduates lacking necessary skills, meaning that just the possession of knowledge does not necessarily imply the competent performance required by the profession. Professional maturity in software engineering is one of software industry's major needs and in-office training can not address this need as efficiently as university education can. This paper exposes the experience reached in a software engineering course in which students develop their software projects in a mature environment. The emphasis here is to use a tool inside a maturity environment that can simulate user errors when operating the software, so that students can learn how to prevent them in their software projects.

1 Introduction

In the last ten years, the software industry has been concerned with a maturity movement, dedicating a significant share of resources to worker training in quality models such as SEI-CMM, ISO, PSP and others. In Brazil, many software organizations implemented those models, but one of their main obstacles was the professional culture in which workers were not used to working in a quality environment and thus had bad habits that needed to be changed.

The software industry blames universities for their graduates lacking necessary skills, meaning that just the possession of knowledge does not necessarily imply the competent performance required by the profession. Bach [1] emphasizes the difference between what software engineers do and what they should do. Although most software engineering books are centered on methods, computer science undergraduate courses experience difficulties in teaching abilities and attitudes.

There must be an effort for computer science courses to prepare their students for long term needs and non-technical skills, such as social, interpersonal, managerial and strategic skills. So these courses must teach cognitive, attitude and skill characteristics, which are considered the maturity characteristics for software engineering [2].

This paper exposes the experience reached in a software engineering course in which students develop their software projects in a mature environment. This is important because usually students make their projects as a non-experienced developer behavior. This way, they do not know the main kind of errors the user can make when operating the software. So this paper also presents a computer tool that simulates human performance under error, the S. PERERE, allowing students to put into practice errors that they did not know about, so that they can prevent them in their projects.

S. PERERE simulates human behavior during interaction with computer software. So this paper contributes to the possibility of minimizing the problems related to computer users and to the students' capacity for preventing human errors.

We believe that the simulator S. PERERE has a great potential to contribute to the studies of software development in a mature environment. This contribution is related to use the simulator, which prints out the several possibilities of human errors associated with some task. So the student can evaluate each possible situation and increase functionalities of his or her software, in this way permitting a better integration of the user with the software under development.

This paper is organized as follows: Section 2 describes the Maturity Environment; Section 3 describes the tool S. PERERE; Section 4 presents a project case and Section 5 presents the conclusions.

2 The Software Engineering Maturity Environment

Numerous studies address the software industry no-return path towards maturity through the establishment of software development processes [3] [4] [5]). This movement is strengthened by a global market requirement for quality in software development, not only as a means of establishing productivity and competitive advantages but in many cases as a safe means of exchanging software products within a globally distributed organization.

The 1990 decade was characterized by application of SEI's Capability Maturity Model in many software organizations in order to respond to the international tendency. Only recently, however, the SWEBOK – Software Engineering Body of Knowledge – definition process by SWECC has established the need to move software development quality and maturity from the level of training to the level of education.

Nonetheless, people have always been the fulcrum in a quality improvement process. Organization employees must be trained to understand the value of their cooperation and their responsibility in organization improvement, as training is an important issue in the quality improvement process.

In this direction, many software development organizations promote training programs so that their employees can access the necessary knowledge to engage in a quality improvement process. Software quality training intends to increase the employee's ability to develop his or her work in the organizational way, at the same time increasing efficiency and reducing rework.

However, employee training costs can become very high, and in some cases, especially for small companies, this cost can overwhelm the quality improvement effort. Considering natural personnel rotation at IT companies, permanently-offered training may be necessary. In some software industry centers competition is reinforced and personnel rotation is maximized.

Knowledge and practice in a mature environment is one of the desired professional characteristics in such situations. As a consequence, it is now important to change the inclination of computer science courses from just presenting software engineering concepts to presenting those concepts within a maturity environment, so that the practice of software development quality can be incorporated into these students before they are delivered to the job market. Educational organizations must quickly take the mature professional formation activity from industry's hands in order to provide the demanded professional profile.

Part of this project defined a pedagogical structure in order to permeate a computer science undergraduate curriculum with Software Maturity concepts. This pedagogical structure has been implemented and tested for 5 years by Begosso and Filgueiras [2]. The Maturity Environment definition started from the identification of learning objectives for cognitive, skill and attitude concepts, after evaluation of concepts embedded in SEI CMM, SWEBOK and also the requirements of the software industry as presented in software literature. SEI CMM's KPAs and SWEBOK's KAs have been scrutinized in order to identify the required concepts, which were mapped to related courses in a computer science curriculum.

In the beginning of this project, a broad literature review provided the desired profile for a software engineer, which unfortunately is not currently developed in computer science courses. Table 1 summarizes this profile, grouped regarding the significant learning objective categories.

Table 1 – Software Engineering Maturity Characteristics

Cognitive Characteristics	Attitude Characteristics	Skill Characteristics
SE Best Practices	Professional Market Acknowledgement	New Practice Adaptability
Documentation	Professional Ethics	Work Environment Adaptability
Project Management	Continued Education	Oral and Written Communication
SE Methods	Humanistic Perspective	Experience in Maturity Environ.
Quality Standards	Sociability	Software Engineering Tools
		Other's work maintenance
		Team Work

The Maturity Environment has been implemented in a controlled way, so that it would be possible to assess the change in student's maturity after this process. The assessment method should evaluate the student progress towards the characteristics in Table 1.

The evaluation mechanism is a 92-question form that is filled in by last term computer science students. The questionnaire explores cognitive, attitude and skill learning objectives, asking the student to evaluate his or her behavior when developing software engineering projects.

Each discipline inside the Maturity Environment must have its own learning environment which addresses its specific requirements. When developing their projects, students conduct work using a Spiral Model approach. So they plan, implement, test and validate. In the validatation phase, they use the tool S. PERERE, described in the next section.

3 Description of the Simulator S. PERERE

A simulator of human performance, developed by Begosso [6] and named S. PERERE, Simulation of Performance in Error, is a human behavior computational simulator whose main objective is to produce, in a random way, human error states. S. PERERE is a human action simulator that considers the error. Some important characteristics of the simulator are: it is possible to explore human error diversity under interaction with software; and human error is treated as an expression of human variability.

When developing their software, students must consider all possible kind of errors that can happen in the interaction between humans and software. Generally, only the most critical errors are considered or those that take the system to undesirable situations.

3.1 Behavior Units

The definition of a set of elementary behaviors is necessary to restrict the complexity of the human performance simulator. Berliner et al [7] suggested a taxonomy of elementary human behavior and defined a set of verbs to represent perceptive, cognitive and action processes.

The behavior units of the cognitive process were defined by authors as: *Calculate, Choose, Decide, Compare, Interpolate, Verify* and *Remember*. The behavior units of the perceptive process were defined as: *Inspect, Observe, Read, Monitor, Scan, Detect, Identify* and *Find*. Finally, the behavior units of the motor process were defined as: *Move, Hold, Push/Pull, Attach, Give, Remove, Discard, Give back, Position, Adjust, Type* and *Install*.

These verbs are implemented in S. PERERE as the set of possible human behaviors. Any task simulated by S. PERERE must be defined in terms of these verbs.

3.2 Human Error

Several attempts to define “human error” are found in the literature; however, it seems that there is no agreement among the authors on a unique definition for the term.

In this paper, we will use the definition proposed by Reason [8], who considers that erroneous actions include all situations in which a planned sequence of physical or mental activities failed to obtain a result and those errors can not be attributed to interventions of external causes. Reason [8] proposes that the erroneous actions can be of two kinds: involuntary and intentional actions.

Involuntary actions are those that deviate from planned intentions and, thus, don’t reach their goals. This can happen in situations when the task, is done in an automatic way: someone misplaces a tool, for example. Those are named *slips* by Reason [8].

Intentional actions occur as planned and still can be considered as erroneous, if they fail in achieving the desired result. The task is performed consciously: the worker selects the right tools but is mistaken about the object to be repaired. Those errors are named *mistakes*.

From Reason’s work, Begosso [6] implemented some human errors into S. PERERE, which will be considered in this paper: omission, repetition, inversion and perceptive confusion.

3.3 Specification of S. PERERE

S. PERERE has mechanisms to simulate several kinds of human behavior: it can represent knowledge to perform a task, to be aware of the environment and update its situation awareness, as well as to act on the environment. To reach this objective, it is necessary to use a cognitive architecture that is able to produce elementary human behavior and can be affected by errors.

Cognitive architectures are computational improvements of aspects inherent in the cognitive, perceptive and motor process of human beings. The architecture that supports S. PERERE is the ACT-R, maintained by the ACT-R Research Group from Carnegie Mellon University, used to help the development of intelligent systems.

S. PERERE is made of the trigger module, the disturber module and the pre-processor module. Data input to S. PERERE is a task description, composed of task elements from the Berliner et al [7] taxonomy that translate the correct (expected, assumed) behavior of a person carrying out that task.

Also, the initial state of the mental model is input to S. PERERE. This allows the system to recognize the starting point for the accomplishment of the task. Concerning the output, S. PERERE generates the task affected by the disturbances, as well as a list of disturbances that occurred.

A brief description of each module follows.

3.3.1 Pre-Processor Module

This module reads each part of knowledge stored in the system and understands it as a unit of elementary behavior, according to Berliner et al [7]. Moreover, the relation also enables the pre-processor to read the production rules to generate the task to be simulated in a syntactically correct way for the performance, in the conditions set forth in the cognitive architecture.

3.3.2 Trigger Module

S. PERERE's trigger module must represent the mechanics of triggering the error.

S. PERERE enables the user to configure the trigger mechanism by selecting one type of error for a specific behavior. For example, the error of omission may be selected for a typing behavior. This working option for S. PERERE establishes the random choice of knowledge from declarative memory, one which includes the motor process of typing, obviously, and results in the omission of said task step.

3.3.3 Disturber Module

The Disturber Module is responsible for simulating the task affected by errors. The module has mechanisms for simulating errors in perceptive and motor processes.

As soon as the disturber reads the disturbance chosen by the trigger module, it sends to the pre-processor the disturbance to be included in a certain behavior, and the disturbed task is output. In other words, one can say that the disturber generates erroneous situations that impact the simulated behaviors for performing the task.

For each error that is generated, S. PERERE creates a text file, syntactically correct from the point of view of the ACT-R language, to be run in the environment of cognitive architecture.

3.4 Error Specification

In order to understand the error generation in S. PERERE, a specification, in structured English, of the errors generated by the simulator, follows:

Omission

The disturber chooses at random one of the task elements and omits the production that would execute that task.

Repetition

The disturber chooses at random one of the task elements and repeats its execution.

Inversion

The disturber chooses at random one of the task elements and inverts its order with the immediately next element.

Perceptive Confusion

The disturber triggers the production rule after visualizing the object and selects the next element physically located next to the one visualized.

4 A Project Case

It was necessary to create some cases in which students develop software projects with the objective to evaluate students' maturity growth inside the maturity environment. This session presents an example where a student group developed a project under the rules of the environment and, at validation phase, they ran the simulator S. PERERE, which generated human errors over the developed software. The main result expected in this case is to observe if the students can see how user error can affect the execution of their software, and so how they can improve the quality of their products.

The project was a commercial system for product sales control and this example concerns an operation over an interface where the user must input data to the system and then press the "Confirm" key.

For this case, the simulator will generate two kinds of errors that the user could have undertaken: omission and perceptive confusion.

To illustrate the operation, we have considered the following simple task for the user:

i. The user verifies the initial interface for data input and presses the "New Record" key.

ii. The user types data for product code, name, and price.

iii. The user presses the "Confirm" key to write the new record on the database.

For the generation of omission in typing, the simulator chooses at random one of the typing information items for the specified task. In this case, it omitted the field name and verified whether the software developed by the students accepts this condition. If so, the students are advised about it and they make sure that all required fields are typed.

For the generation of perceptive confusion error, after typing all the required fields the user should press the "Confirm" key. However, the simulator tries to press a neighboring key that is activated. For example, if the neighboring key is "Delete", the user can make an error without wishing to do it. If so, the students are advised about it and make sure that only permitted keys are activated.

The two situations illustrated in this example indicate the presence of a latent error status in the software and contribute to the learning environment.

5 Conclusion

In the development of students' projects, it was possible to observe the generation of errors for the categories omission, repetition, inversion and perceptive confusion, which enabled students to learn about these errors and prevent them in their software.

The mechanisms created in this work have permitted the growth of students' software engineering maturity as intended. The evaluation method was successful in instrumenting this process and indicating that this growth is really happening, and more than that, has succeeded in pointing where changes must be made for the next applications.

Simulations generated over the example task, although very simple, allow us to think positively about S. PERERE's potential in generating errors on the simulated behavior.

The authors believe that this research contributes to creating an undergraduate teaching environment to form a mature professional for the software development process. The process is open and easily reconfigurable, so that it can be used in other disciplines.

References

1. J. Bach, What Software Reality is Really About, *Computer* **32**(12), 148-151, (1999).
2. L. R. Begosso and L. V. L. Filgueiras, Environment for Maturity Development in a Computer Science Graduation Program. In: Proc. of the International Conference on Software Engineering Research and Practice, (Las Vegas, USA, 2002), pp. 400-405.
3. L. R. Begosso and L. V. L. Filgueiras, Implantação de CMM Nível 3 para ensino de Ciência da Computação Orientado à Qualidade. In: Proc. of Symposium on Software Technology'99 – 28th JAIIO, (Buenos Aires, Argentina, 1999), pp. 21-25.
4. C. Jones, *Patterns of Software Systems Failure and Success* (International Thomson Computer Press, USA, 1996).
5. S. H. Kan, *Metrics and Models in Software Quality Engineering*, (Addison-Wesley Publishing Co, USA, 1995).
6. L. C. Begosso, PERERE: Uma Ferramenta Apoiada por Arquiteturas Cognitivas para o Estudo da Confiabilidade Humana, doctoral thesis, (Sao Paulo University, Sao Paulo, 2005).
7. D. C. Berliner, D. Angell, J. Shearer, Behaviors, measures and instruments for performance evaluation in simulated environments. In: *Symposium and Workshop on the Quantification of Human Performance*, (The University of New Mexico, Albuquerque, 1964), p.277-296.
8. J. Reason, *Human Error* (Cambridge University Press, Cambridge, 1990).