Difficulties in Solving Ill-Defined Problems: A Case Study with Introductory Computer Programming Students

Andréa Mendonça1, Clara de Oliveira2, Dalton Guerrero1, Evandro Costa1
1Computing and System Department, Federal University of Campina Grande, Paraíba, Brazil
2Informatics Department, Federal University of Santa Catarina, Santa Catarina, Brazil
andrea@dsc.ufcg.edu.br, clara@inf.ufsc.br, dalton@dsc.ufcg.edu.br, ebc.academico@gmail.com

Abstract - In this paper we report a case study carried out in the first academic semester of 2008 with novice programming students of the Computer Science Course at the Federal University of Campina Grande. This case study aimed at observing and evaluating strategies adopted by students on ill-defined problem solving. The results of this case study showed that students found it hard to perform problem statement exploratory reading and interpretation, formulate questions to enlighten the problem, analyze the problem constrains and error occurrence, write tests to check non-obvious situations and also to register effectively the new problem information acquired by discussing with client. The present paper also reports on some pedagogic possibilities to improve the Introductory Programming Teaching.


INTRODUCTION

Introductory Computer Programming students deal with problems. Usually, these problems are well defined and are meant to encourage the learning of programming concepts, language syntax, and program codification. However, the problems of the “real world” are ill-defined1 and require skills that go beyond codification, such as: identification, acquirement and the recording of relevant information, dialog with clients, and analysis of the problem constraints. As can be observed, all previously mentioned skills are related to requirements elicitation and specification.

In the traditional computing courses curricula, these skills are developed only for Software Engineering disciplines, Analysis and Systems Design, or other disciplines with equivalent denominations - usually comprising the forward disciplines of the curricula offered by the end of graduation. However, there are evidences of this being ineffectiveness: graduated students still face difficulties in dealing with ill-defined problems [1].

On an attempt to minimize this difficulty, some teachers and researchers advocate the adoption of ill-defined problems in the early stages of graduation [2]. We also agree with this point of view. To solve ill-defined problems demands the implementation of non-trivial skills by the students, which requires some considerable amount of time.

Ill-defined problem solving requires the adoption of a series of new teaching strategies – entirely different from the traditional one – to be employed by professors when teaching the novice programming student. The students’ difficulties will be tackled by these new strategies, making possible the development of new skills. In this scenario, some questions must be answered: Which are the difficulties students have to face when dealing with ill-defined problems? What kind of strategies do they use for problem specification? Since the dialog is a fundamental tool for elucidating ill-defined problems, how do these students conduct a dialog with clients? What kind of strategies do they adopt to find problem constraints? Can tests be used as part of these strategies?

To answer these questions and to make a full picture of students’ difficulties in dealing with ill-defined problems, we carried out a case study with novice programming students of the Computer Science Course at the Federal University of Campina Grande. The case study design, its implementation and results are reported in the present paper. Furthermore, we have also presented some related works and a pedagogical analysis showing possible ways of improving the teaching of programming to novice students.

RESEARCH QUESTIONS

Our case study aims at answering the following research questions:

- [RQ1] - What strategies do novice programming students adopt for problem specification?
- [RQ2] - How do these students conduct the dialog to obtain information and to enlighten the problem?
- [RQ3] – Are tests sound strategies adopted by students to discover problem constraints and program errors?
The design, execution and also the analysis of results of the present case study followed the guidelines suggested by Kitchenham et al. [3].

**Subjects**

Our case study was carried out at the end of the first academic semester of 2008 with the participation of ten (10) students from the Programming Lab Discipline (CS1) of the Computer Science Course.

The student group was made up of seven (07) men and three (03) women; the average age being 18 years old. Their participation was conducted in a voluntary basis, and motivated by the perspective of university curricula improvement.

The CS1 discipline duration was of 60h and its scope was focused on language syntax and program codification, under traditional teaching practice. Pascal programming language and Turbo Pascal 7.0 environment were adopted. There were 28 students in the class, i.e., the case study sample represented 37.5% of the total students’ population.

**Objects**

To answer the previously mentioned research questions, students were required to solve an ill-defined problem whose statement was:

“Develop a program to calculate the net wage of a store's sale assistants. These sale assistants are paid a wage plus sales commission after deductions, such as income tax.”

The proposed problem statement has some “faults” (intentionally placed), described as follows:

- Lack of Information: The rules to calculate sales commission and income tax were not given; the other taxes, besides income tax, referred to in the statement by “deductions” requires elucidation; the rules to define the net wage, and also the information that should be read (input) and printed (output), with their respective constraints were not given either;
- Ambiguities: That is the case of the expression “sales commission” – Does sales commission refers to the store's total sales, or to the sales just made exclusively by a particular seller? And the phrase “after deductions” - Does it refers to just the sales commission or to wage plus sales commission?

The lack of information and some ambiguities were inserted intentionally in order to motivate the students to reflect and built questions on the problem specification. In this case study, the researcher played the role of both observer and client, implementing the participant observation strategy.

For this problem, were established eight (08) test cases, which the student’s program should attend to in order to get the client’s acceptance. Test cases did not cover all the possible errors, but, as we have observed, these were sufficient to verify if the student had specified and solved the problem properly.

**Execution**

This case study was done towards the end of the CS1 discipline so that a better evaluation of the problem solving strategies learned by these students could be obtained.

Firstly, the problem statement showed in the previous section was given to the students. The researcher suggested the following situation: “Imagine that I am your client and I need a program to solve this problem. In case of doubt, feel free to consult me”.

There were no limitations of time. In addition, the students could freely search on the Internet for information; consult books or any other didactic material. Students were provided with a computer, pencil, pen and paper, with the purpose to observe all procedures employed by them in the problem-solving task. Then, the students should develop the program by using Turbo Pascal 7.0. After completion of the program, the researcher, playing the role of a client, ran some tests in order to confirm the program acceptance. If the program did not pass the tests, the students should detect the requirements that were not satisfied, and then correct the program. Every time a student delivered a program for the researcher to run the tests, this was interpreted as being a new version of the program. Thus, it was possible to collect all versions of the program codified by the student until all requirements were discovered and implemented.

This process came to end when the program passed all tests. After this, an interview was held with the students. The interview was unstructured, and it helped to clarify what happened during problem solving observation.

**Data Collection Procedure**

In the present case study, the following data were collected from the students:

- Annotation on behavior and on the activities developed during problem solving;
- Written records of interviews;
- Source code of all program versions;
- Audio recordings of all interactions.

The researcher collected the data during the problem solving operation carried out by the students. The combination of different data gave us a much more rounded picture of both the students’ behavior and the adopted strategies. This technique is known as “triangulation”.

**References for Analysis**

It is important to observe the strategies employed by students for the specification of ill-defined problem. Those strategies refer to the ordered actions aiming at achieving a certain goal: in this case, the complete specification of the given ill-defined problem.
Based on the work of Zabala [4], Coll et al. [5], Polya [6] and also on the principles of Requirements Engineering, we defined some strategies for problem specification (Table 1). These strategies were used as references to observe and analyze the students’ behavior.

**TABLE I**

**STRATEGIES FOR PROBLEM SPECIFICATION.**

<table>
<thead>
<tr>
<th>STRATEGY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploratory</td>
<td>Comprehension of language and terms that specify the problem. Identification of relevant information, ambiguities, contradictions and lack of information.</td>
</tr>
<tr>
<td>Reading</td>
<td></td>
</tr>
<tr>
<td>Interpretation</td>
<td>Contextualize and establish relations between the problem’s relevant information; identify the needs of the client; and enlighten the meaning of information to solve the problem.</td>
</tr>
<tr>
<td>Information</td>
<td>Observation; information search and dialog with the client.</td>
</tr>
<tr>
<td>Acquisition</td>
<td></td>
</tr>
<tr>
<td>Analysis</td>
<td>Examine the constrains and boundaries of the problem; investigate the possible errors that may occur; and make inferences.</td>
</tr>
<tr>
<td>Organization</td>
<td>Organize and register the informations in an appropriated way; update the problem statement to clarify the client’s needs.</td>
</tr>
</tbody>
</table>

**VALIDITY**

In this section, we discuss the threats to the present research validity:

Internal validity. The interviews revealed that 30% of the students had previous experiences with programming concerning two circumstances: some students withdrew, and others had already been taught programming. According to these students, their previous experiences focused on syntax learning and codification. This fact does not represent a threat to the validity of our study, for our goal is not to establish causal relationships.

External validity. The limited number of subjects does not allow generalizations outside the scope of the present study. On the other hand, we expect that these results to act as a preliminary study that allows the formulation of further hypotheses to be tested in subsequent experiments.

Construct validity. For this case study data were gathered from many different sources (audio recordings, source code, interviews, among others). This data collection promotes an increase in the reliability of the results, avoiding errors and biases.

**ANSWERING THE RESEARCH QUESTIONS**

The students took, on average, 86 minutes to conclude the program with approval in all tests. None of them elicited all requirements on the first version of the program. Only 20% of the students finished the program on the second version. But 50% of them needed more than 4 versions to satisfy all requirements. Based on the collected data, we described the results taking into account the research questions mentioned previously.

**[RQ1]** - What strategies do novice programming students adopt for problem specification?

It was observed in the present research that most of the students (80%), after reading the problem statement, started immediately to codify the program or to write the algorithm, without carrying out, previously, an analysis of the problem statement. For instance, Figure 1 shows the first version of a student’s program (identified here as “Student J”). The student did not make any questions, and built the first version of the program based only on the problem statement and on his belief concerning what should be done. The student extracted keywords from the problem statement and defined them as variables in the program, for example: income tax (line 7), net wage (line 8) and sales commission (line 5).
presented in the problem statement. This behavior stems from the habit of dealing with well-defined problem.

The results obtained from our case study will be detailed as follows, taking as reference the strategies presented in Table 1.

- **Exploratory Reading** - The students had not explored previously the problem statement.

  The students had difficulties to identify omitted information and constraints, e.g.: in the first program version, 70% of the students ignored the existence of some other taxes, even though these had been written implicitly in the problem statement – “(...) after deductions; such as income tax”. Also 70% of students ignored the possibility of other data to be required as input, for example, the seller ID. Although 90% of the students asked about the rules for calculating the sales commission and income tax, these calculations were the most obvious evidence of lack of information in the problem statement. Another interesting aspect of this is that they rarely explored the business domain vocabulary, even though they did not recognize the terms in the problem statement. Such behavior justifies the fact that over 50% of the students needed to codify more than four versions of the program so as to meet all the client’s requirements.

- **Interpretation** - Most students ignored the existence of a client and created their own interpretation of the problem statement.

  As mentioned above, 70% of the students in the first version of the program did not ask anything concerning the expected inputs and outputs. Consequently, two situations occurred: they only defined as input or output what was explicitly reported by the problem statement; or what they have themselves defined as necessary, imposing a personal interpretation to the problem. The inference generation process based on the student’s previous knowledge was related to reading comprehension [7]. In this particular case, it brought undesirable implications, because the students ignored the need of problem specification. This situation is exemplified in Figure 1.

- **Information Acquisition** - The difficulties with exploratory reading and interpretation endanger the information acquisition process.

  Obviously, if the students do not realize that there is a demand for information, they do not attempt to develop strategies to obtain it. The strategies for information acquisition were supported by dialog and submission of the program to the researcher (client). Most of the students submitted their program expecting that, on the occurrence of failures, the missing requirements would be pointed out. This demonstrates, among other things, a lack of reflection on what should be done and the absence of proactive behavior.

Regarding the dialog, the answer to the second research question (RQ2) that comes next in the text.

[RQ2] - How do these students conduct the dialog to obtain information and to enlighten the problem?

Generally, dialog occurred on demand: with a few questions before the codification and some others questions along the ongoing process. Some questions were related to the design of the program, such as:

Student B: Must I use procedures and functions?
Researcher: Not necessarily. This isn’t a program requirement. It is up to you to choose the best way to structure your program.
(Minutes latter...)
Student B: Can I use a flag statement to exit the program?
Researcher: Yes, you can.

After delivering the first version of the program, the students were usually surprised with the lack of information in the problem statement:

Student E: What is wrong with my program?
Researcher: Well, I need other inputs.
Student E: What kind of inputs?
Researcher: The seller’s ID number.
Student E: But this wasn’t written here! (Pointing to the problem statement)
Researcher: It is true, but I still needed the seller ID number.

Given the number of imprecise questions asked by the students, we noticed that it was necessary to improve the dialog skills. In the below dialog, Student A did not know the concept of gross wage, and instead of ask objectively about it, he made imprecise questions, maybe trying to infer the concept.

Student A: Does the gross wage vary, doesn’t it?
Researcher: Could you explain it better?
Student A: Does the gross wage change in relation to each seller?
Researcher: Yes, it does.
Student A: But what really is gross wage?

- **Analysis** - The students checked only the obvious test cases for problem constraints.

  The evaluation of this strategy answers the third research question:

[RQ3] – Are tests sound strategies adopted by students to discover problem constraints and program errors?
Some of the students’ difficulties included perception and elicitation of problem constraints. Their tests covered only the obvious cases. The students were unable to explore the remaining constraints and the possible failures. Often, the students wrote and ran only two or three tests at the end of the implementation. Some tests contemplated user interface aspects instead of program correctness. For example, 90% of the students in the first program version read the variables without showing any concern about constraints over them, as shown in Figure 1 (lines 12-15).

In the interviews, the students reported that they could rarely predict error situations about their programs. Some of them confessed that, when they were taught the Programming Lab discipline, they performed tests only when requested. According to these students, the deadline for program submission is usually short and, consequently, they are often left with very little time for the tests.

The results of our case study pointed towards greater emphasis on the necessity of paying attention to test teaching in introductory programming disciplines. These tests, besides promoting an improvement in program quality and in problem comprehension, they represent extra practice in answer to demands placed by the software industry.

- **Organization** - The students did not re-elaborate the problem statement.

The new information records were compiled in the form of small annotations, regardless of style or organization. Abbreviations, mathematic symbols and arrows linking blocks of information were used. As a result of the poor information record, occasionally the same questions were repeated at different stages of the problem resolution process. The students could not imagine that the registration of requirements was a kind of “contract” between developer and client, defining what should be done.

It this case study, 80% of the students wrote the proceedings for sales commission calculus wrongly. This is showed in the following dialog:

**Student E:** Will the user give the commission percentage?

**Researcher:** No. The sales commission should be calculated over the seller’s total sales.

**Student E:** How should I make the calculus?

**Researcher:** The commission is equal to 3% of the total sales up to $10,000.00 plus 4% over what exceeds this value.

Based on this dialog, the student made the following annotation:

\[
\text{If } \text{sales} \leq $10,000.00 \rightarrow 3\
\text{Else if } \text{sales} > $10,000.00 \rightarrow 4
\]

According to student’s annotation, if the seller’s sales were equal to $12,000.00, then the sales commission should be equal to 4% over this value. But, this calculus is wrong.

The correct way to perform the calculus, according to the researcher’s guidelines, should be: 3% over $10,000.00 plus 4% over $2,000.00.

It is important to notice that the student’s annotation also revealed difficulties of text interpretation and lack of attention to details.

**RELATED WORKS**

Difficulties with problem specification were also enhanced by other researches. Eckerdal, McCartney and Moström [8], for instance, analyzed the software design produced by students nearing completion of their Computer Science degrees, and their study showed that most of the evaluated students did not seem to understand what sort of information a software systems design should include, and how this information should be communicated. The above-mentioned authors’ conclusion was: “Perhaps we are wrong, and should offer students more experiences in dealing with under-specified tasks” [8, p. 407].

McCracken *et al.* [9], in a multi-national, multi-institutional study of assessment of programming skills of first-year CS students, observed that the abstraction of the problem to be solved from the problem statement represented the greatest difficulty for students. In his study, the problem specification included details that were not relevant to the problem solution, which made it difficult, in the author’s opinion, for many students to achieve the problem objectives.

Blaha *et al.* [10] describes a study with novices, graduating seniors and educators about recognition of ambiguity in the design process. Their results revealed that fewer novices recognized ambiguity, and many of them simply made assumptions, rather than explicitly ask question about ambiguous issues.

Ill-structured problem solving approach was also enhanced in the last ASEE/ICEE Frontiers in Education Conference - 2007, when it was ministered a panel entitled: “Ill-Structured Problem Solving in Engineering Education” by Daniels, Carbone, Hauer and Moore [11].

The results of these researches support the evidences shown in our case study, i.e., novices have difficulties in dealing with ill-defined problems. Taking into account that such predicament is not restricted to beginners alone, the teacher’s responsibility in reformulating the curricula becomes crucial.

**PEDAGOGIC POSSIBILITIES TO IMPROVE INTRODUCTORY PROGRAMMING TEACHING**

The results obtained with this case study, associated with the researches previously mentioned in the Related Work Section, call for changes in the introductory programming teaching methodology.

Researchers have been working out some alternative teaching approaches. One of these is Meyer, who proposed an approach called *Outside-In* [12]. The *Outside-In* approach is based on the four following principles: object-oriented paradigm, the use of Eiffel programming language, the utilization of libraries and software reuse. The author...
emphasizes the concept of *contract*, and makes use of it to teach students how to deduce the behavior of the modules they have been asked to develop.

Oliveira [13] proposes an educational project-centered approach entitled *Thematic Oriented*. Knowledge, according to this approach, must be treated from the synthesis level to the analytical, allowing students to develop different skills of problem solving, and experience different levels of abstraction, from problem comprehension to its solution.

Mendonça et al. [14] conceived an approach to introductory programming called *Problem Oriented Programming* (POP). In POP, students must deal with ill-defined problems; develop activities of requirements elicitation and specification. Those activities will support the problem enlightenment and the construct of a program to satisfy the expectations of the client (teacher).

These pedagogic practices can be applied to the teaching of introductory programming, and will contribute to overcome some difficulties as the ones presented in our case study.

**CONCLUSION AND FUTURE WORK**

This paper described a case study carried out with novice programming students in order to elicit their difficulties in ill-defined problem solving.

The results of this case study will contribute to (i) the increase of empirical research in computer education; (ii) the composition of a body of knowledge regarding the difficulties of students, and the challenges concerning programming education, and (iii) the guiding, the designing and the use of new theories, teaching new strategies and educational technologies whose aim is to improve programming learning and narrow the gap between the university and the software industry.

It would be interesting if this study could be replicated by teachers in other institutions. This would consolidate a multi-national and multi-institutional research about the difficulties novices may encounter when solving ill-defined problems.

This ill-defined problems case study is our contribution to knowledge module, taking the “Thematic Oriented” Curriculum approach as its central educational focus.

**ACKNOWLEDGMENT**

We would like to thank the volunteer students who have participated in the research. We want to extend our special thanks to Elloá B. Guedes. We would also like to thank the Foundation of Research Support of the State of Amazonas (FAPEAM) for its financial support.

**REFERENCES**


