Abstract - The technical literature in cognitive science informs that working in groups reaps more benefits than working alone. We are seeing a variety of innovative group programming methodologies like the agile methods. This work examines the findings of a pilot study carried out during the first academic semester of 2007. The subjects were 110 freshmen computing and engineering students taking their first programming course at the Federal University of Amazonas. Based on these findings, it is proposed a programming progression learning scheme, from individual (current practice) to group programming (the desired practice based on the literature review). Such transition is necessary for students are not used to programming in groups. In order to evaluate such progression learning scheme a case study is outlined.

Index Terms – group programming, group modeling, programming learning.

INTRODUCTION

The search for knowledge about programming skills is recurrent in this research area, starting with Dijkstra [6] in his work “On the Teaching of Programming” and in Weinberg [16], which addresses the psychological aspect of programming learning. Although there is a lot of research on the subject, the milestones of programming learning are not fully elicited yet and that is why it is so difficult to understand which elements contribute to the knowledge acquisition in programming.

Regarding learning, independently of the area under consideration, it is known [14] that collaboration promotes in the individual the development of skills and strategies for problem solving that are of extreme importance for the construction of knowledge in that domain. That is why we endeavour to adapt these collaboration techniques to the domain of programming learning.

Collaboration has become a necessity both in the job market and in education. Particularly in software development companies, the increasing demand for products and services leads to an exacerbation of competition, prompting these companies to search for convergence in their activities. This is reflected in the formation of development teams and in the need for collaboration within the team and among teams. As to education, when collaborating students can see the point of view of their pairs and build a joint solution for the problem in question. It is easier to perceive the effect of collaboration when one has access to the records in a virtual learning environment and can analyze a substantial difference in the refinement of solutions, as described in [1].

In programming learning it is necessary to adopt a model or scheme to guide the development of strategies and to follow-up the group development. Such model should suit both domains: problem solving and collaboration. As we could not find this kind of model in the literature, the bibliographical review in the next section is based on programming learning, both individually and within groups. Then, in the following section there is a description of a pilot study for group programming carried out in the first semester of 2007. Then, as a result of this pilot study’s analysis, it is proposed a programming progression learning scheme followed by a case study’s setting for group programming learning.

PROGRAMMING AND GROUP PROGRAMMING LEARNING

Learning concepts and methods for the construction of computer programs is not trivial, as it requires the use of high level abilities and a high dose of abstract reasoning. In [6] it is emphasised that programming involves reasoning more than any other ability. But programming is also an engineering task, once it deals with the production of artefacts that must satisfy quality requirements and be subjected to verification.

In [7] it is pointed out that in introductory courses students seldom learn problem solving techniques. What normally happens is that students find great difficulty in applying their previous skills. This ends up becoming a source of fear and frustration, thus fostering evasion. In the work described in [5] an effort to develop a new programming course based on laboratory sessions, several activities were planned such as online discussions, programming exercises in pairs, reading of texts from the Internet, reflection annotations, diary entries and collaborations using the pair review process to criticise colleagues’ replies to a given topic.

The latter article deals with the transformation from a methodology involving theoretical and practical classes to one that uses only practical classes, with distinct activities well distributed among the sessions. In spite of the preoccupation with the technical aspects of programming, the search for knowledge about programming skills is recurrent in this research area, starting with Dijkstra [6] in his work “On the Teaching of Programming” and in Weinberg [16], which addresses the psychological aspect of programming learning. Although there is a lot of research on the subject, the milestones of programming learning are not fully elicited yet and that is why it is so difficult to understand which elements contribute to the knowledge acquisition in programming.
with developing questionnaires to create in the students the habit of reflection, such methodology proved inefficient to detect confusion in the grasping of concepts. As to the programming exercises carried out in pairs there is no evidence of an improvement in performance as a result of this technique given that there was no record of pair activities or even a control group.

Following the learning evaluation line, the work presented in [4] describes a combination of some evaluation techniques in an introductory computer course and demonstrates through statistical analyses the differences and relationships among these techniques. The course was planned following the premise that before learning how to program students must be able to solve problems. Therefore, firstly students solve problems without the use of a programming language and later learn how to use it to represent solutions.

After the initial phase of problem solving, the course continues with six laboratory sessions where students must individually solve problems, being allowed to consult their pairs whenever they deem necessary. It must be pointed out that problem complexity increases as the sessions advance. Once this phase is over, small groups (2 to 4 members) propose a case study consisting in the development of a program (normally a game). A learning evaluation is conducted statistically comparing student performance in the laboratory sessions, case study and controlled individual practices without consultation (such as tests) carried out twice throughout the course.

The contribution of the aforementioned work is the statistical analysis of correlations among the evaluation mechanisms used. Although it is important to know if the students are being evaluated by the most efficient methods for their learning, other factors are disregarded, as it occurs with the nature of the work developed in groups – for instance, there is no control of the development of each work, making it impossible to ascertain if a given task was performed by a single student, which would cause an error in the correlations.

Once again using evaluation models to teach programming, the work described in [10] uses the students pre-evaluation as a basis to categorise them in learning stages as defined in Bloom’s taxonomy. From there the course is formatted in such way as to offer differentiated activities for students in the different stages of training.

In an effort to identify aspects that can facilitate programming learning, the work described in [7] states that through students’ answers to questions such as “what is programming?” it is possible to define a presentation order for the programming paradigms. The authors depart from the principle that students need to know what programming learning really is in order to actually learn. The majority of the answers surveyed suggest that they must first be exposed to a more structured reasoning before being presented to object abstractions in OOP.

What is really necessary to facilitate programming learning is still an open question. Although the aforementioned articles attempt to find an answer to this question, there are no works in the reviewed literature that have succeeded in establishing undisputed methods and techniques for group programming learning. On the other hand, there have been initiatives whose focus is to create and maintain student interest in the course using concepts of extreme programming, already widely used in development teams in the software industry.

A PILOT STUDY FOR GROUP PROGRAMMING

A pilot study on introductory programming course was carried out in the first semester of 2007 [3] with two groups of students enrolled in the first semester of the Computer Science and Computer Engineering undergraduate courses at the Federal University of Amazonas.

The objective of this case study was to propitiate students the experience of developing solutions for complex problems through the distribution of tasks, negotiation, composition of partial solutions and successive refinements. This was achieved working in groups of up to 5 members who were also responsible for recording the activities developed along the several work stages, using a version control environment denominated AAEP [1]. At the end of the course, as a final activity, a group task was posed. The solution had to be supplemented by the record of the interactions among team members. Then, the analysis and interpretation of these data based on classification, codification and tabulation processes took place.

As described in [3], the groups experienced several difficulties mostly related to the coding of the proposed solution in the Haskell language. Although solution planning was very difficult for the students for its outcome depends on their coordination and interaction, the main difficulty reported was related to skills in programming techniques and knowledge of the specific language. This entails that programming learning can be more efficient when conducted in a group following a model or scheme that facilitates this process.

Besides developing code and keeping the record of interactions the students also answered a questionnaire, which was submitted to a quantitative analysis. Finally, they spoke freely of their difficulties and of the work’s development stages, being these comments subsequently submitted to a qualitative analysis.

As a reflection of the teacher researcher in this process, the performance of the groups was generally satisfactory. Comparing the implemented codes and their companion reports with the researcher’s annotations it is possible to find a match, which often indicates difficulties in putting together the parts of the program developed individually in independent modules to build and refine the complete program.

The analysis of this pilot study makes evident that programming in groups is enjoyable for students, increasing for this reason their interest on the proposed activities. It also pointed out the importance of a gradual transition from individual to group tasks, because students reported many difficulties in carrying on their group projects.
A PROGRAMMING PROGRESSION LEARNING SCHEME

As described in the previous sections, there have been many attempts to facilitate programming learning. Some initiatives are geared to more practical classes associated with continuous follow-up, while others try to keep focus on especially prepared exercises. Nevertheless, all the analyzed reports still reveal significant gaps regarding programming learning.

Based on a 15 years experience with programming learning [2], use of collaboration methods to represent knowledge about problem solving [11] [12] [15] and this pilot study, we believe programming in groups is a difficult task, mainly because students are not used to work in groups. That is why, for carrying on group activities, such as programming, it is necessary to have a model to guide this activity or, in case of an inexistence of such a model, a progression programming learning scheme, as the one we propose next.

The analysis of a case study described in [1] resulted in the conception and development of the AAEP, a student solution follow-up tool, which allows, should the professor deem it necessary, the retrieval of a comparison between 2 of any versions by a single student. This tool was designed to attend essentially to the teacher’s demands. Therefore, access to management functionalities such as user and problem records and solution analysis is restricted to the latter. The other functionalities such as program source-code elaboration, edition and test associated with commentaries characterizing each version are accessible by both students and teachers.

We used the AAEP [1] in classroom to support group programming. This has become instrumental towards adapting a collaboration model for group programming learning that involves the following actions:

- Checking whether students search for existing code for similar problems before trying to solve a specific problem, a possible evidence of the importance of problem solving and programming by example;
- Making the solution planning process more explicit by revealing the importance of recording code versions;
- Observing whether group interaction leads to faster solutions;
- Observing incorporations on the practice from the individual to the group, focusing on possible behavioural changes;
- Analysing student behaviour within the group.

Figure 1 illustrates a programming progression learning scheme that defines a progression from individual to group programming, in a scenario that begins at phase 1, with a preparation that involves laboratory sessions dealing with introductory problems and clarifications on the methodology. Phase 2 consists of individual solution and recording. In phase 3 group work starts by deciding the best individually constructed solution. In phase 4 problem solving turns collaborative: the teacher defines the tasks and the group defines the actors for each activity. In phase 5 the group is also responsible for task definition. Finally, in phase 6 there is a development task where the groups compete in a format that mimics real work situations.

This scheme is currently being evaluated at the Federal University of Amazonas. The choice for the division into 6 phases was based on our 15 years programming teaching experience using functional languages. It considers a 75 hours course taught in one semester.

The problem solving phases described in Figure 1 follow this sequence and phase numbers because, as we have cited before, besides our prior experience in programming learning, we have specifically conducted case studies for the use of collaboration methods to represent knowledge about problem solving [11] [12] [15] and found out that the number of problem solving phases is adequate for a 60 hour course on introductory programming.

GROUP PROGRAMMING LEARNING: A CASE STUDY

Based on our experience with education and, more specifically, programming education, we have elaborated and applied a work project to survey behaviours in group programming learning that follows the group programming learning progression scheme defined at the end of the previous section, in order to define the next steps for this research. Amongst other possibilities, the application of such programming scheme can provide subsidies for the elicitation of requirements for a piece of groupware acting within this domain.

According to what was made evident in the previous case studies, students usually start computing courses without knowing how to collaborate. Another way to see that is that they assemble materials instead of elaborating syntheses.

In the introductory programming course, classes were divided into practical and theoretical ones. The latter were taught once a week and the practical ones were split between laboratory practice and external practical classes. These classes were used for group work, and there, digital recording...
of the interactions was mandatory. A tool to support collaboration was provided, comprising the entire content created before and during the course, following the directives found in [8] [9] and being aware of the problems described in [13]. Table 1 describes the planning of the practical activities, according to the phases and weeks shown in Figure 1. Next, we present sample exercises by phase. These exercises come from the UFAM programming exercises database.

1. **Preparation** – “A group of four friends wants to purchase a product that costs more money than they have. In order to buy the product, the group decides that the closing amount would be divided proportionally to how much each one already has, considering that those who contributed with more money could always get some more. Describe how you can solve this problem coding in Haskell, indicating how much each one will have to pay.”

2. **Individual Coding I** – “Given 2 points, p1 and p2, located in the Cartesian space, which define a line. Determine this line’s equation.”

3. **Individual Coding II** – “Given 3 points, p1, p2 and p3, located in the Cartesian space, determine whether they constitute a triangle and, if it is the case, determine its area.”

<table>
<thead>
<tr>
<th>Phases</th>
<th>Week in the Course</th>
<th>Content Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual: Preparation and Individual Coding I</td>
<td>1 e 2</td>
<td>Initial survey: students answered a questionnaire available in the environment of support to the group work.</td>
</tr>
<tr>
<td>Group: Individual Coding I</td>
<td>3</td>
<td>Definition of skill levels: Practical laboratory activity using the AAEP with a predefined session, using basic geometric problems (90 minutes) and analysis of results based on the resolution time.</td>
</tr>
<tr>
<td>Group: elaboration of group solutions using Group Coding I</td>
<td>5</td>
<td>Resolution of 1 exercise in Haskell, with individual records taken as notes.</td>
</tr>
<tr>
<td>Group: systematization of the processes of group solution planning using Group Coding II</td>
<td>6</td>
<td>Analysis and selection of individual solutions; refinements; collaborative notes about the process as chats and digitally recorded face-to-face conversations.</td>
</tr>
<tr>
<td>Team: competition using collaborative development (Team Coding)</td>
<td>8</td>
<td>Problem 1 - students solve problems according to previous phases. Group members review each others codes and write notes.</td>
</tr>
<tr>
<td></td>
<td>9 e 10</td>
<td>Problem 2 - one part for each student, but the professor supplies the division of work.</td>
</tr>
<tr>
<td></td>
<td>11, 12 e 13</td>
<td>Problem 3 - the students themselves must use modularization techniques to divide the work and to choose who will develop which part, recording the decision-making process.</td>
</tr>
<tr>
<td>Process Evaluation</td>
<td>14</td>
<td>Programming marathon style with a prize at stake. It consisted of: external observation by Master’s or more advanced students; use of a tool for monitoring group activities; and stages for resolution of the marathon problem: record of things to observe; record of each element of the solution; answering a questionnaire.</td>
</tr>
<tr>
<td>Process Evaluation</td>
<td>16</td>
<td>Application of questionnaire and interviews.</td>
</tr>
</tbody>
</table>

4. **Group Coding I** – “Consider 2 points, p1 and p2, located in the Cartesian space. We are interested in identifying whether the following relationships between them are applicable: (a) it is possible to draw a line passing by p1 and p2 and parallel to the abysses line; (b) it is possible to draw a line passing by p1 and p2 and parallel to the ordinate line; (c) p1 and p2 are in the same quarter; (d) p1 and p2 are in different quarters; (e) p1 and p2 are in opposite quarters; (f) p1 and p2 are in adjacent quarters.”

5. **Group Coding II** – “In a clinic, as soon as a patient arrives at the hospital, she receives an attending number. There are always three available doctors per shift, and they will receive incoming patients depending on the number of patients a doctor already has in her to attend list.”
The doctor who has fewer patients in her list gets the next one.

Using tuples, we can define the following input: available_doctors ("(dr. A", 4, 23), "(dr. B", 1, 13), "(dr. C", 3, 27)), where the 2nd term of each tuple refers to the number of patients in that doctor’s list and the 3rd term refers to the last patient attended by that doctor. Based on this input, write a script in Haskell that, for a given incoming patient, choose in which doctor’s list she should be allocated to.

6. **Team Coding** – “In a blood database, there is a donor’s record that includes the donor’s social security number (SSN), sex (S), age (A), blood type (BT), RH factor (RH), date (DD) and the amount of blood (AG) (250 or 500 ml). The blood is kept in 250 ml fix capacity recipients. Hospitals require blood (H) everyday. Each request indicates the blood’s characteristics (type and RH factor) and the requested amount (RA). It is known that men and women must wait different minimum time intervals between donations. For men, it is 2 months and for women, it is 3. The maximum and minimum ages for donations (for both sexes) are 60 and 15, respectively…”

Problem modularization and function definition in Haskell are the concepts involved in the preparation phase. The individual solving phase requires the same level of expertise, added up with more mathematical reasoning due to the nature of geometric problems. The next phase, group coding I, requires the use of conditionals in addition to the knowledge about geometric problems. In the group coding II phase, students must use tuples and lists to solve an introductory problem. Finally, in the team coding phase, students must solve a problem that involves tuples, lists and recursion.

From the first group coding phase onwards students have to work in groups. These groups are formed in the third week of practices, after the application of a supervised laboratory session. The requirement adopted is that for each group to be as heterogeneous as possible, with a minimum of 5 members. No more than 2 have formal experience in programming, 1 may have some experience (for example, web script programming) and 2 to 3 may lack experience in programming.

As shown in the sample exercises above, after group formation, the exercises increase in complexity according to the progression scheme and course contents. This enables the gradual shift from a work rhythm based on individual practices to the incorporation of collaborative practices of program development.

**CONCLUSION**

This article presents the setting for a case study and a progression learning scheme to support group programming, based on the resulting analysis of a pilot study conducted with undergraduate computing students at the Federal University of Amazonas, during the first academic year of 2007.

The academic semester that the aforementioned case study is being carried out to evaluate the progression scheme is still in course. Even within the beginning of the semester, it was possible to identify acceptability in the laboratory sessions, that we believe will motivate students to accomplish all the planned activities.

Analyzing the digital content produced during this introductory course, it will be feasible to elicit the strategies used by the groups to collaboratively code. Finally, we will proceed with this investigation in order to formalize a collaboration model for group programming learning.

**ACKNOWLEDGEMENTS**

This project is partially sponsored by the Brazilian Ministry of Science and Technology through the CTInfo project grant nº 550865/2007-1. Hugo Fuks is sponsored by CNPq individual grant nº 301917/2005-1 and also receives grant from the FAPERJ project "Cientistas do Nosso Estado". It also receives resources from project ColabWeb – Proc.553329/2005-7, CNPq/CT-Amazônia n.27/2005.

**REFERENCES**


