Castro, T.; Robertson, D.; Fuks, H.; Castro, A. **Identifying the Need to Intervene: Analysis and Representation of Interaction Patterns in Group Programming Learning**. Proceedings of 17th CRIWG Conference on Collaboration and Technology (CRIWG'11). Paraty, RJ: Lecture Notes on Computer Science LNCS, 2011. v. 6969.

Disponível em: http://groupware.les.inf.puc-rio.br/work.jsf?p1=8432

Identifying the Need to Intervene: Analysis and Representation of Interaction Patterns in Group Programming Learning

Thais Castro[1,2,3], David Robertson[2], Hugo Fuks[3], Alberto Castro[1]

[1] Federal University of Amazonas, Computer Science Department, Manaus, Brazil
{thais, alberto}@dcc.ufam.edu.br
[2] University of Edinburgh, School of Informatics, Edinburgh, UK
dr@inf.ed.ac.uk
[3] Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
hugo@inf.puc-rio.br

Abstract. This paper focuses on a supporting strategy for enhancing distributed and computer-mediated group programming learning. Based on a real-world research setting that started two decades ago, we have exploited a particular context characterized by: (i) a close analysis of artifacts produced by learners; (ii) a collaborative approach to learning, combined with (iii) a team-based approach to programming; and (iv) the use of a Progressive Learning Scheme for group programming learning. These elements are discussed as rationale for the analysis and representation of forum-based discussion logs generated within a case study carried out with first year undergraduate computing students. This analysis allowed us to develop a means of coordinating group programming on a distributed, agent-based platform using group programming stereotypes from conversation analysis. These stereotypes were defined using interaction patterns within a process calculus.

Keywords: collaborative learning, distributed learning environments, improving classroom teaching, teaching/learning strategies, group programming.

1 Introduction

Programming learning is a cognitive activity that requires a high level of abstract reasoning [25]. This usually causes students to get stuck somewhere in the introductory course, turning programming learning into a painful experience. In order

to mitigate this problem, we take the view that introductory programming courses should make use of collaborative learning methods in addition to usual learning strategies and pedagogical practices. The assumption that supports these methods states that students learn more when they are able to interact with their peers, for achieving a unified solution to a given problem [8].

In undergraduate introductory computing courses the use of internet-based groupware represents broadened opportunities to introduce good practices in the students' learning process, such as the recording of every interaction among students within their groups while solving exercises and of each student's code evolution (individually or within her group). Activity recording is an essential requirement for keeping track of their progression, making it possible for the teacher or any other classmate to give appropriate feedback during the exercises.

Students who work in groups supported by collaborative Learning Management Systems (LMS) usually produce a large amount of conversation logs including code fragments, footprints of the decision making processes, work methodology and the use of their own development patterns, which might be filtered and categorized to be further analyzed by the teacher and other peers, who may then give students the appropriate feedback. Filtering should be done automatically, by tools which could point out desired or undesired patterns in conversation or code fragments that could lead, in the case of desired patterns, towards the intended goal. An undesired pattern could be identified by analyzing the groups' conversation logs and then realizing that they are not getting any further in their search for a solution.

Using LMS and the sort of filtering tools described above could help to improve students' abilities to collaborate and to compensate the fact that they are not co-located and therefore lack all the advantages of a face-to-face conversation, such as body language. In order to partially address this issue, we replace students' interaction described by the logs for a more structured representation, having a collection of conversation patterns as a starting point. These conversation patterns could then be used alongside with any other collaborative programming scenario.

The aim of the research reported here is to improve the current supporting strategies for group programming learning described on Section 3. This paper focuses on the analysis of conversation logs for extracting and representing stereotypes on how students organize themselves in order to act under a collaborative scheme. These stereotypes could then be used by the aforementioned software tools for a large class of applications. Our work makes use of a multi-agent environment and its representation language for describing students' interactions that took place within a LMS. The interaction patterns were represented in LCC (Lightweight Coordination Calculus) [19] which is a language developed to improve coordination in distributed multi-agent platforms.

In order to situate the research presented here within its application scenario, we describe its context in the following section based on previous research experiences in programming learning and collaborative systems. Subsection 2.1 presents a few other researches on group programming learning, emphasizing what elements have been used in our work while Subsection 2.2 describes a progressive learning scheme for gradually introducing collaboration into programming learning courses. In Section 3 we describe the case study carried out to analyze the progressive scheme, which generated the conversation logs further analyzed. Continuing the case study description, Subsection 3.1 presents the evaluation method used to analyze the

conversation logs, and in Subsection 3.2 they are instantiated and examined. Finally, in Section 4, we identify the need to intervene in students' interactions.

2 Related Work

At UFAM, a Brazilian University, there is an ongoing twenty year experience with introductory programming teaching using the functional programming paradigm. The emphasis is in problem solving, as described in [3]. Besides the functional approach, this research group conducted experiments using collaborative methods to represent problem solving knowledge [13], [16], [22].

It is a common belief [2] that there is a strong link between a software engineer's performance and her academic background. Thus, in introductory undergraduate programming courses it is mandatory to use techniques based on problem solving, given that problem solving is an essential ability for software engineers. Moreover, given that problem solving is a collaborative activity, training for collaboration is an important part of undergraduate curricula. Thus, in order to collaborate, students should be aware of problem solving strategies. Having that in mind, a pilot study for probing which were the most difficult topics in the introductory course [3] was applied to two freshmen groups in 2004.

In their weekly practical class, students got one or two exercises that had been previously solved during the theoretical class. They could consult the Internet or any other bibliography and discuss the exercise with other students or trainees. From roughly 80 students who attended the course, 10 constituted the observation group. The pilot study follow-up was based on a qualitative analysis of their annotations, before and after solving the exercises. After every practical class, the researchers read the students', trainees' and research assistants' annotations (5 assistants observed the observation group while they were solving the exercise). Based on their understanding, researchers triangulated annotations in order to find out which students should be invited for an individual interview. The interview is based on the clinic method protocol, as proposed by Jean Piaget and explained in [8].

During the aforementioned pilot study analysis researchers repeatedly stumbled while following changes done to the code. In order to solve this tracking problem a tool called AAEP [1] was developed for keeping track of the students' code evolution by the pairing of versions. Later on, for the purpose of classifying code evolution in one of these three categories, namely, syntactic, semantic and refactoring another tool called AcKnow [5] was developed.

A new case study aimed at observing how students organized themselves within a group and how each group solved the exercise, revealing that way their major difficulties [4] was conducted in 2007. However, whenever students work in groups it is difficult to know who is responsible for each code fragment and who is collaborating with whom. Besides that, according to the literature in Cognitive Science [7], students learn more when they are organized in groups. Programming is definitively a cognitive activity [25] requiring a special setting, given by the combination AAEP-AcKnow. Based on the findings of this case study, we proposed a group programming progressive learning scheme [6], which gradually introduces collaboration into students' practical exercises. That was applied to a new case study that took place in 2008, which is the one considered below in this paper.

At this point, it is worthwhile to review the literature looking for the many attempts that have been made towards the adoption of collaborative practices in programming learning. Those considered relevant to this work are described in the following subsection.

2.1 Group Programming Learning

Freudenberg, Romero & du Boulay [9] define as intermediate talking the pair's discussion about 'chunks of code' during a pair programming session. Their findings point out that both driver and navigator work at the same level of abstraction. On the opposite way, some articles [14], [12] claim that one of the benefits of using pair programming in introductory courses is the lack of balance in abstraction level between the two actors. Either way, intermediate talk seems to be relevant for programming in pairs. Moreover, whether keeping track of intermediate talks in a pair programming set or more open conversation in a group setting, both approaches emphasize the importance of using computational tools to filter conversation.

Stahl [23] describes how the ConcertChat tool was used for group problem solving in maths. It comprises a whiteboard and a chat tool, relating chunks of the chat log to drawings or texts on the whiteboard. This combo avoids vagueness in conversation, fostering collaboration in this way. Stahl uses the term group cognition to refer to the distributed level of cognition among group members. Moving on to pair work, Peres & Meira [17] affirm that whenever students work in pairs supported by a computational tool they build a thinking and action system that positively reflects in their cognition system as a whole. In their work, they evaluate the use of an educational tool by pairs of students using the conversation analysis method [11]. Their findings point out that although there are conversation breakdowns, they did not seem to interfere in the learning process, reassuring this way the importance of keeping track of contextualized conversations and whatever product versions result from them.

Pastel [15] presents the use of group programming for program learning. It describes an experiment conducted over four laboratory sessions during a data structure course. Each session proposed a different programming method. In the first one, students had to program individually; in the second they work in pairs, as in a pair programming session; in the third session they discuss their pair programmed codes from the previous session with two colleagues originated from two other pairs for choosing the best code, possibly adding or subtracting bits of code. The final practice takes place within a group of four students that are free to choose whatever method suits them. It is worth mentioning that there was a case study conducted at UFAM in 2007 [4] that achieved similar results to Pastel's.

The commonality found among the aforementioned works is the use of some kind of mechanism for capturing the conversation exchange that takes place among the students for inferring clues that may guide teacher's intervention in the next session.
Based on the comparative literature review described above and after examining the findings of our own previous case studies we have identified the need for some sort of transition scheme, which could encourage students to work collaboratively. This scheme is described next.

2.2 A Programming Progressive Learning Scheme

One obstacle that fleshed out from the 2007 case study was the difficulty that freshmen showed in having any initiative and ability to collaborate. Based on the Handbook of Cooperative Learning Methods described in [21] and in the literature about collaborative strategies [13], [16] and [22], we proposed a group programming progressive learning scheme followed by a case study from 2008 for evaluating the scheme [6]. This case study data is used as the basis for the analysis and representation of the interactions in the following sections.

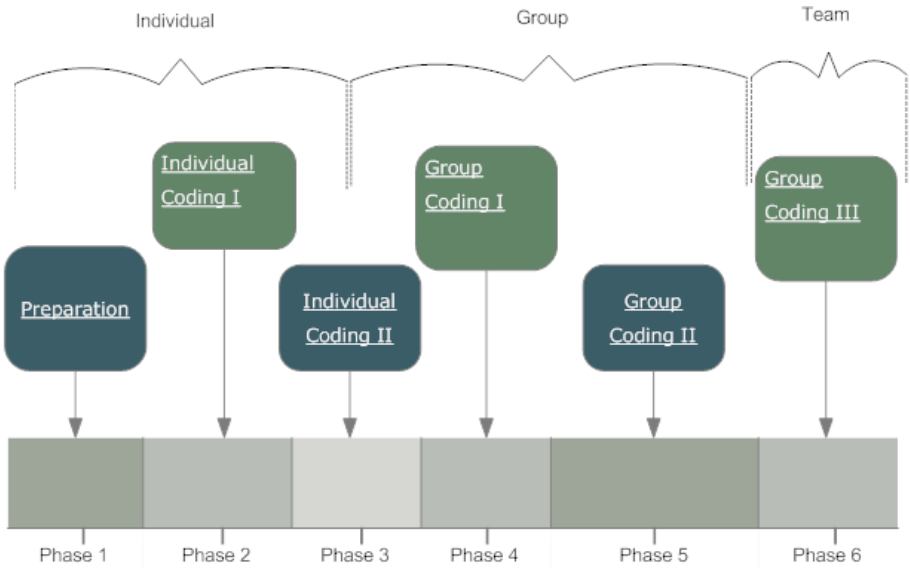

Figure 1 - The Programming Progressive Learning Scheme

Figure 1 illustrates the group programming progressive learning scheme adapted from [6] that covers a transition from individual to group programming, in a scenario that begins at phase 1, with preliminary laboratory sessions dealing with introductory problems and clarifications on the methodology. Phase 2 consists of individual solutions and recordings. In phase 3 group work starts by choosing the best individual solution. In phase 4 problem solving turns collaborative: the teacher defines the tasks and the group appoints the actors for each activity. In phase 5 the group's responsibility encompasses task division. Finally, in phase 6 there is a development task where groups compete in a way that resembles real work situations.

3 A Case Study in Group Programming Learning

This case study was conducted during the first semester of 2008 and applied to 60 Computer Science and 50 Computer Engineering first year undergraduate students. As the progressive scheme suggests, each programming topic involves it own representation need, demanding at least one exercise. Thus, it is important to extract information from every activity proposed during the introductory course, inferring the

knowledge representation used in each programming topic. Given that the proposed activities imply coding, companion reports and conversation records, the improvement evidences we expected are presented below as supporting strategies for each artifact, apart from the last one, which is traversal to the former three.

a. Code evolution: as students make their own attempts at solving the exercise, they keep all code versions in a log for post analysis. AAEP and AcKnow provide a quantitative analysis for each solution, making inferences on the code for identifying the use of desired or undesired programming techniques, and to make calculations using the timestamps between versions and confronting them with the classification of code changes in order to find out what types of changes students have realized within that time interval.

b. Conversation follow-up: for each exercise, students have to agree about task division and the method they are going to use to tackle the problem. Before each code version release there is a discussion, which leads the group to a successful code or to a pitfall. By keeping track of these two conversations, the teacher may preempt a possible pitfall as the conversations take place by giving some advice.

c. Report: after finishing the programming exercise, students are required to write an informal report containing, besides the code itself, all the information about the development process. They are encouraged to write about their difficulties and findings. This relevant information must be compared with the resulting information provided by AAEP and AcKnow (see item 'a' above) to check whether code improvement is taking place.

d. Use of search tools and inference techniques to observe their transition from individual to group solution: this is necessary for evaluating how students cope with the progressive learning scheme.

Considering the group programming progressive learning scheme presented earlier in this section, in phases 1 and 2, students start learning how to collaborate using a chat tool. The whole class took part in the discussion, but coding was left as an individual activity.

Next, in phase 3, coding remained an individual activity, but by then they had to choose the best code from the group justifying why that code was the chosen one. Almost all groups generated long conversation logs and discussions have run deep in eliciting the requirements for best code candidates. The conversations demonstrated that students were quite comfortable in testing each other's code by comparing them with the course's understanding of efficiency and efficacy.

In phase 4 they allocated the pre-divided parts of the exercise to all group members and then they discussed about the exercise itself in order to prepare their wiki-based group report (comprising the individual coding and the combined group coding together with the suggested problem solving method). Their forum-based conversation logs show at this point that although some groups could collaborate in a rudimentary way they lacked the ability to solve the problem as a group, passing the responsibility to join code fragments to a self-appointed group member. Almost half of the groups just made available their individual members' coding. Their logs also showed that group members had no understanding about the exercise as a whole. Only two groups understood the task at hand and really solved the exercise together, suggesting changes in each other's codes, whenever necessary. One group did not use

the forum tool justifying that they had a face-to-face discussion. One other group did nothing.

Phase 5 asks the groups to split the exercise into functional parts for distribution among group members. That entailed heated discussions showing that they moved to a next level in their collaboration practices. One possible reason for this improvement was that the groups that previously had difficulties or misunderstandings about how to conduct themselves in a collaborative way, got feedback pointing out where within the problem solving process they were not able to collaborate. Another possible reason was that students got progressively more involved on their activities as they kept solving the exercises posed within the progressive scheme.

Phase 6 of the progressive scheme resembles a programming marathon, where they work in small groups of three or four members. In this phase, group formation was left for the students based on their preferences. Surprisingly, not that many new groups were formed. During the four hour collocated marathon record keeping was not mandatory, although they had to write a wiki-based report.

Given that too many different digital artifacts were produced during this case study, we opted to focus on the second supporting strategy, the conversation follow-up, taking into account the overall analyses of the three other supporting strategies. It is worth mentioning that the first supporting strategy was extensively discussed in [5] and the others will be discussed in future work, based mainly on [10] and [18].

3.1 The Evaluation Method: Representing Interaction Patterns

Following the case study application, a conversation analysis took place largely based on the concept of speech acts by Searle [20] and used in this work for finding stereotypes. According to this theory any given student could start a message thread having an intention in mind, for instance, "Hey folks! I've done mine and I found out that it was simple". Here the objective is just to inform, so his utterance could be treated as pertaining to the Informing category. On Table 1, translated from Portuguese, there is a description of each message thread corresponding to an utterance category found in the conversation log, followed by its instance. For the sake of analysis, each group conversation log was shortened based on a local concept of extended speech acts, namely interaction patterns, comprising the initial turn and its relevant complements. They present the teacher with an overall idea about group behavior. Whenever pre-defined categories of interaction patterns were identified the message body was further analyzed in order to check whether the categorized pattern had been correctly identified.

We borrow the definition of interaction patterns from [26] which states that they are representations for interactions, described by the sections: name, sensitizing pictures, intent, context, problem, scenario, symptoms, solution, dynamics, rationale, check, danger spots, known uses and related patterns. In this paper we describe our interaction patterns by their name, sensitizing picture and intent, representing them in a representation language (LCC) in order to more easily identify their known uses instantiated by the stereotypes.

Table 1. Interaction Patterns and their Instances

Category	Instance
Making an artifact available	"My functions…"
Informing	"Guys, the problem isn't so difficult…"

Clarifying	"I couldn't log in before."
Confirming	"I've annotated it already…"
Asking	"Does someone want to add something else on the report?"
Suggesting	"…everyone should try to create a solution to each question on his own way…"
Calling attention	"Hey, Guys! Let's do the exercise!"
Pointing a mistake	"I think you made a mistake when you defined the output as the type int…"
Explaining	"…What I did was to use the 2nd question that…"

Stereotypes are the combination of different interaction patterns. According to the teacher's analysis, some of these stereotypes have already been identified and classified as good or bad as can be seen on Table 3 in Section 4. Good stereotypes are those that usually lead to a successful collaborative coding effort, while the bad ones are those that lack a collaborative effort, eventually leading just to a member code or even to an incorrect one. In order to more effectively identify and act upon stereotypes, the diagram represented in Figure 2 illustrates how interaction patterns from Table 1 are represented. However, stereotypes do not easily emerge from conversation logs. Two main reasons for that are the occurrence of conversation breakdowns not properly restored within an online discussion and the complexity of natural dialogue. Thus, some structured representation for the discussion would be beneficial to deal with it. One way to address the problem of defining, matching and dealing with stereotypes detected within the forum-based discussion logs is by using a formal representation language with a high dose of expressiveness.

A logical-based language designed for dealing with communication protocols in a multi-agent environment was chosen for representing interaction patterns. The LCC (Lightweight Coordinate Calculus) [19] is a notation already used within different applications under a platform named Open Knowledge (www.openk.org) that will help to address the coordination problem. Although being a very expressive language, LCC keeps the simplicity needed to represent the interactions properly by using inference resources provided by the Open Knowledge platform.

Typically, as shown in Figure 2, every process starts by a group member deciding to initiate a message thread, triggering that way an interaction pattern. The initiator automatically becomes the coordinator for that interaction pattern. Then a virtual agent called broadcaster broadcasts the message to everyone else subscribed to that conversation. If after having read the message one decides to answer it, she automatically becomes the evaluator by sending the message back to the coordinator via broadcaster.

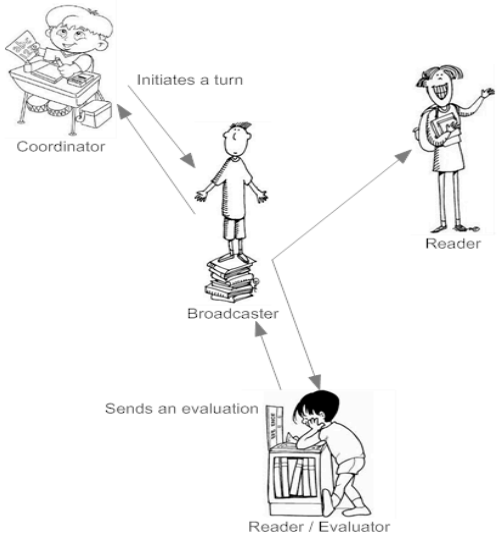

Fig. 2 - Formal Representation for Conversation

As can be seen, the elements that appear in Figure 2 show up in the LCC code fragment below, instantiated for the interaction pattern Clarifying.

```
1 a(clarifier,C) ::=
2      a(broadcaster(X,L,Er),B) <-- new_clarification(X,L).
3 a(broadcaster(X,L,Er),B) ::=
4      (information(X) => a(reader,R) <-- L=[R|Rs] then
5            Er=[E|Es] <-- evaluation(X,E) <= a(reader,R) then
6            a(broadcaster(X,Rs,Es),B)) or
7      null <-- L=[] and E=[].
8 a(reader,R) ::=
9      information(X) <= a(broadcaster(X,_,_),B) then
10     (evaluation(X,E)=>a(broadcaster(X,_,_),B)<--agree(X,E) or
11     evaluation(X,E)=>a(broadcaster(X,_,_),B)<--
do_query(X,E)).
```

The interaction pattern Clarifying requires three roles: clarifier, who starts the interaction; broadcaster, who sends the message to everyone subscribed to the interaction and reader, who reads the message and can be turned into evaluator, replying to the clarification. In this case the interaction pattern complements are lines 10 and 11.

This code fragment could be read as follows:

1. `a(clarifier,C)::=`. A statement giving the clarifier role an instance 'C'. The '`::=`' means the definition of this role starts after it.
2. `a(broadcaster(X,L,Er),B) <-- new_clarification(X,L)`. A statement calling another role instantiated by 'B', who will, given some string 'X', a list of readers 'L' and evaluators 'Er', broadcast the message to every agent subscribed to the interaction if there is a new incoming message, represented by '`new_clarification(X,L)`'.
3. `a(broadcaster(X,L,Er),B) ::=`. Gives the broadcaster role the identifier 'B' and after that its definition is started.

4. `(information(X) => a(reader,R) <-- L=[R|Rs] then.` Sends the information 'X' to a reader if there are readers in the list 'L'.
5. `Er=[E|Es] <-- evaluation(X,E) <= a(reader,R) then.` Builds a list of evaluators 'Er' if a reader 'R' sends an evaluation 'E' referring to the information 'X'.
6. `a(broadcaster(X,Rs,Es),B)) or.` Calls the broadcaster again for the next readers 'Rs' and evaluators 'Es'.
7. `null <-- L=[] and E=[].` This statement turns to null value when there aren't any more readers or evaluators in the lists 'L' and 'E'.
8. `a(reader,R) ::=.` Similar to the clarifiers, this statement gives the reader role an instance 'R'.
9. `information(X) <= a(broadcaster(X,_,_),B) then.` The information 'X' is received from the broadcaster 'B'.
10. `evaluation(X,E) => a(broadcaster(X,_,_),B) <-- agree(X,E) or.` It is valid if the reader agrees to the information 'X', sending some short agreeing statement.
11. `evaluation(X,E) => a(broadcaster(X,_,_),B) <-- do_query(X,E)).` If the previous statement is not true, the reader doesn't agree with the information 'X' and he sends a query 'do_query' containing his evaluation 'E', possibly a query.

Other interaction patterns ask for different complements. There are cases, such as Informing that requires more than one kind of reply. However, there are cases such as Making an Artifact Available that not necessarily have any complement.

The teacher analyzes the interaction patterns presented by each group and define whatever sequence she considers successful as a stereotype. A knowledge base is then created for each exercise comprising the group conversation log, interaction patterns and stereotypes. Whenever an unidentified stereotype shows up during the forum-based conversation, the teacher has to look at the case in depth and give feedback to the group before the task is completed. She may then include the new stereotype in the knowledge base.

The analysis presented in the next subsection shows how identifying the interaction patterns in the forum-based logs make them more traceable for further interventions.

3.2 Identifying Interaction Patterns in the Forum-based Discussion

Phase 4 of the progressive scheme presented in Figure 1 stands as a bottleneck because for the first time during the course, students are urged to think about a whole problem and its solution collectively. During the problem solving process they have to accommodate each other's views and negotiate whenever there is a conflict of understanding. The conversation fragments below refer to exercise 5, a classical allocation problem. It is worth noticing that the teacher proposes a specific method for solving it. Some groups adapted that method in order to solve the exercise according to their own characteristics.

All conversation logs shown below were translated from Portuguese. Hence, some misspellings and inappropriate use of English were placed in the text to give the reader a more accurate view of the conversation.

Group 1 neither discussed the problem understanding nor the problem solving, missing the division of work and the process of joining all the individual solutions together to build a unique group solution; each group member made available his own solution. This group's behavior could be represented by a sequence of Making an Artifact Available interaction pattern. They produced individualized solutions and it can be seen from their codes that there was no overall agreement or understanding about the problem as a whole.

One member of group 2, in a self-appointed way, led the group. Another one assumed the division of work. The forum-based discussion about the general understanding was rudimentary, gaining some strength whenever the matter was the coding. The last topic in the conversation log dealt with suggestions. The leader (StD2) examined all codes and pointed out what could be improved in a specific one, as shown in fragment below.

StK2 HAS TO CORRECT THE FUNCTION AND TO REDO POLYA'S STEPS, StD2 AND StF1 HAVE TO CORRECT POLYA'S STEPS.

StD21, I've already written above what is necessary in your case.

StF2, the inputs are a list of tuples, even when the function you're working with uses the previous function's result. I've corrected there in Polya's steps. E.g.: Inputs: [("dr. A", 4, 23), ("dr. B", 1, 13), ("dr. C", 3, 27)]. Outputs "dr. B"

StK2, your function is wrong, it has to return the complete list of tuples, as the doctor's tuple required in the input will have the 2nd term -1 and the last one will be the number of the patient added to the input.

Examples: Inputs: "dr. A" 28 [("dr. A", 4, 23), ("dr. B", 1, 13), ("dr. C", 3, 27)] Outputs: [("dr. A", 3, 28), ("dr. B", 1, 13), ("dr. C", 3, 27)]

In the conversation fragment above dealing with the coding a proper conversation takes place. The group discussion begins with the interaction patterns Asking and Explaining followed by Making an Artifact Available or Suggesting. This is a much desired stereotype that led the group to an excellent result.

Group 3 basically followed the same route: one student was self-appointed as group leader and was responsible for the division of work. The forum-based discussion regarding the coding started with a General Question about the understanding of the exercise that the leader emailed to the teacher. At some point, a student made available his code and report and it apparently made another student uncomfortable. She reacted by immediately posing a question, starting with a Clarifying, as shown in the fragment below.

StV3 Hey folks! I've done mine and I found out that it was simple. (Clarifying)

```
aux_lesses x xs = [ y | y <- xs , y < x ]
index_less xs = [i | i <- [0..length xs-1], aux_menores (xs!!i) xs == []]
```

But even if it was simple, I'd like you to look at the end of the function "index_less xs" (aux_lesses (xs!!i) xs == []), 'cause it was there where I had more difficulties.

I was trying to do ...

StF3 Well, mine was pretty short, I thought it was odd, but I think it is complete, as it was a simple question.

What I did was to use the 2nd question that shows the lowest indices and use them to show the doctor with fewer patients. It follows:

```
doctors_fewer_patients = disp!! index_less
```

StV3 Man, explain that in details….

Group 4 had more than one member leading the group. One student proposed a topic about some peculiarity she has found in the exercise. Based on that, another one

discussed the nature of the exercise and a third one suggested a slightly different problem solving method from the one proposed by the teacher. Everybody followed the third member's suggestion triggering discussions about each other's solution. The fragment below registers the moment that a student figures out the nature of the exercise. Then, another one proposes a way to do it gaining the leadership of the solving process.

StJ4	I think the problem is sequential… I believe the best way of dealing with the exercise is doing it sequentially, each function reusing the previous functions, creating extra ones, when it is the case.
…	…
StR4	As far as I can see everybody in the group agrees about the fact that the problem is sequential and consequently reuses every item in the following... So, because there is not much time left, ideally everyone should try to create a solution to each question on his own way, and then we separate the best ones, to combine them in order to write one group solution…

Despite the fact that there were two initial antagonistic ideas for solving the exercise as a group, the leading members have found an agreeable way of doing it, addressing the discussion with many Suggesting after some Informing and Clarifying. It proved to be a good way of dealing with different opinions, but it took longer than necessary. The group achieved a good result, but with an extra help from the teacher, they would have probably reached an agreement earlier in time and would have solved the exercise faster.

All members of Group 5 made their codes available straight away and this seemed to be the result of a face-to-face conversation. Apparently, from the fragment below, the forum tool was only used to call one member to join the chat session.

StR5	We are elaborating the report with all the functions you did. StY5 and I need you to join the chat session we've created to discuss this exercise because here in the forum it takes too long.

Group 6 did not discuss the exercise as a whole, keeping the conversation restricted to two members and only about the last item of the exercise.

Group 7 roughly followed groups 2 and 3 way of solving the exercise. One student led the discussion making his code available and asking for the division of work. Another student subdivided the work and asked everyone else to make their solutions available as soon as possible, together with their respective explanations. All members complied. A third member of the group read all the codes, finding a mistake and its probable cause. Something that deserves attention is that at the end of the forum-based discussion, one student remarked that he had not done his part yet but needed no help. Only at the very end of the discussion he made his solution available.

This group came up with another successful stereotype, beginning the forum-based discussion with Asking, followed by Making an Artifact Available, Suggesting and Explaining, and finishing with Making an Artifact Available. The group achieved an overall great result, using all required data structures and following the suggested way to solve the exercise.

Initially, Group 8 discussed about General Questions related to the exercise. One member complied with another member's Suggestion and everyone else continued reusing each others' functions, although some other questions about the exercise were posed. The group remained collaborating, except for one member, who did not take part in the forum-based conversation. From the fragment shown below, it is clear that he did his part without reusing or revising the functions that had been made available.

StF8	I've done the 5th question. As I haven't use the codes which have been done

before my code is big, but, basically it has the same functions done until the 4th one.

Unusually, there were too many Asking interaction patterns during the forum-based discussion. However, the discussion also had many Making an Artifact Available and Explaining intercalated with the Asking(s), which proved to be another good stereotype.

Group 9 presented a bad stereotype similar to Group 1, but with absolutely no discussion. Everyone made his own codes available and one member was responsible for the tests and group solution.

Group 10 didn't do the exercise.

The way each group worked on their first group exercise fleshes out clues about the causes of possible difficulties in collaboration and also evidences of successful use of an informal collaboration method.

In the aforementioned description of the way each group collaborated while solving exercise 5, most of the groups tried to collaborate and behave as a group, but they did not achieve group programming *nirvana*, i.e., they tried but they didn't get there because they did not discuss each other's approach for the exercise. The code fragments were mostly individualized solutions. However, the registering of the group exercises were very helpful because they made it clear for the teacher how to identify each group's difficulties, helping her to intervene whenever it deemed necessary, s preparing them for the next exercise.

Other groups, namely Groups 2 and 4, performed their work without any difficulties. Group 4 even used a new problem solving method, which was beyond the expectation at this point. Groups 1 and 9 completely ignored the fact they were doing their exercise as a group and only Group 10 failed to do the exercise.

In the following section there is a discussion informing teachers on how to deal with the emerging stereotypes from the forum-based discussion logs in order to properly intervene when using the group programming progressive learning scheme.

4 Identifying the Need to Intervene

Basically, in order to know when to intervene teachers should always: (i) identify the students who are not corresponding to their expected participation within their groups, because that entails that these students have some difficulties in performing the required task; (ii) identify the bad stereotypes, aiming at helping students whenever they get stuck. Teachers' observation should focus on: the relationship among interaction patterns presented on Table 1, the group programming progressive learning scheme used as guidance for the exercises posed during the introductory course, the roles students perform within their groups and the identification of stereotypes that act as guidelines for teacher's intervention.

The clues for intervention emerge whenever: (i) an interaction pattern repetitively appears within the forum-based discussion; (ii) only one or two group members hold the floor, even if they are using different interaction patterns and (iii) the combination of interaction patterns fosters bad stereotypes. Table 2 describes the clues extracted from the forum-based discussion for the first case (i) showing the corresponding intervention.

Table 2. Clues for Acting Upon Repetitive Interaction Patterns (Case(i))

Interaction Pattern	Clue	Intervention
Making an Artifact Available	There is no turn taking	Tell group members to inspect each other's code
Informing	Wasting time with irrelevant issues	Tell group members to restart the discussion
Asking	Lack of leadership	Tell group members to choose a leader

In case (ii), when only one or two group members hold the floor, the clues for intervention come from noticing that a couple of senders dominate the forum-based discussion.

In case (iii), when the combination of interaction patterns doesn't lead to collaboration, group members are probably on a wrong path caused by a bad stereotype. Table 3 describes the clues extracted from the forum-based discussion for case (iii) showing the corresponding interventions.

Table 3. Clues for acting upon the stereotypes (case (iii))

Stereotype	Clue	Intervention
Asking, Calling Attention and Suggesting	When this stereotype appears before a Making an Artifact Available takes place or if it loops for approximately 10 turns, the group is struggling to understand the task	Tell group members to subdivide the task as recommended.
Informing, Clarifying	They have difficulties in focusing on the exercise, thus they are not really solving the task. Instead of it, they're either talking about something not that relevant or about some other unrelated subject	Tell group members to pick whatever is relevant for completing the task and focus on the developing process.

Working with a computational representation for stereotypes, it is possible, just by examining the forum-based discussion, to find potential pitfalls while the groups are still in the process of finding a solution for an exercise. In this case, when such feedback opportunity is identified, and according to an existing knowledge base, the teacher can intervene in the process, helping to improve collaboration within the group. This way, groups achieve a more accurate solution, as it happens with any other collaborative activity [24].

5 Conclusion

Web-based environments, especially LMSs, are important resources for supporting both conventional and distance learning, whatever the educational setting considered. This intense and diverse use of LMS' interaction tools might present unexpected obstacles for their users, as the breakdowns that occur on the logical sequences within a conversation due to synchronization differences, but might also present new opportunities for intervention and support during collaborative learning activities.

In this paper, we proposed a web-based record-oriented collaborative and progressive scheme to allow knowledge elicitation on the learning process of group

programming and identification of opportunities for teacher's intervention. First, we showed how the progressive learning scheme could be used to incrementally advance students towards collaboration in group programming learning. Moving from individual coding solutions to group solutions using forum-based discussions as a means for supporting problem clarification, task division and the coding process, successful group members understand that collaboration is the way to go.

Then, inspired by the speech acts theory we proposed a set of interaction patterns as a means for identifying the underlying intention in group members' turns within the forum-based discussion. Starting by figuring out the intention behind group members' posts, the researcher looks for the initial turn and its relevant complements that comprise the interaction pattern. Good stereotypes are sequences of interaction patterns that show up within the forum-based discussion whenever members successfully collaborate for reaching a group coding solution. Whenever bad or uncategorized stereotypes show up within the conversation, an intervention opportunity emerges. For instance, the repetitive use of an interaction pattern within the forum-based discussion log offers opportunities for teacher's intervention given that it is expected that these patterns vary during turn taking, engaging all group members in the solving process. For instance, during the discussion that took place to solve exercise 5 some groups started repetitively using the Making an Artifact Available interaction pattern, without any further commentary. That gave the teacher a clue for intervention, immediately calling the groups' attention for the need to collaborate. According to the progressive scheme, instead of just making one's code available, students should collaborate discussing all individual codes in order to arrive at a unified code solution.

Programming learning, like many other intellectual activities, benefit from collaboration. However, group programming learning also has to tackle several difficulties that turn up when developing abilities related to effective group participation. These difficulties are not always easily identified whereas the groups are still discussing about the task itself. The deployment of the group programming progressive learning scheme gives the teacher a collection of inspection points where she is able to identify difficulties, as those concerning conversation breakdowns, acting on individuals or groups whenever she deems necessary. These inspection points, represented here by a set of interaction patterns, can be inferred and treated automatically, thus, generating automatic responses for the groups. We treated the interaction patterns automatically, representing them in LCC, contributing towards run time interventions from any actor involved in the process and consequently, the creation and further maintenance of a knowledge base about group programming learning. In the exercises that followed exercise 5 discussed here, some groups presented similar bad stereotypes in their conversation logs, needing the same type of intervention. This strengthens the idea of keeping a knowledge based framework with supporting strategies, methodologies like the progressive scheme and a set of interaction patterns in order to identify new stereotypes.

Stereotypes should be continuously added to the knowledge-based framework in order to be transformed into group programming learning intervention's heuristics in the future. The ultimate aim of the approach is, in addition to enhancing group programming learning, to decrease teachers' workload; to improve their ability to supply feedback and thus to improve overall learning outcomes.

Acknowledgments. Hugo Fuks has a research grant from CNPq and "Cientista do Nosso Estado" grant from FAPERJ. This work used resources from GroupwareMining Project – Proc.575553/2008-1, CNPq/CT-Amazônia n.055/2008.

6 References

1. Almeida N., F. A.; Castro, T.; Castro, A. N. (2006). Utilizando o Método Clínico Piagetiano para Acompanhar a Aprendizagem de Programação. In the proc. of the XVII SBIE. Brasília: Gráfica e Editora Positiva Ltda, 2006. v. 17. p. 184-193.
2. Brooks, F. The Mythical Man-Month (anniversary edition), (1995). Addison-Wesley Longman Publishing Co.
3. Castro, T.; Castro, A. N.; Oliveira, R. S.C.; Boeres, M. C. S.; Menezes, C. S. (2005). Enhancing Programming Understanding through Conceptual Schemas in Introductory Courses. CLEI Electronic Journal. Vol. 8, Num. 2, p. 4. (http://www.clei.cl/cleiej/).
4. Castro, T., Fuks, H., Spósito, M. & Castro, A. (2008). The Analysis of a Case Study for Group Programming Learning. ICALT - Proc. of the 8th IEEE International Conference on Advanced Learning Technologies, July 1-5, Santander, Spain.
5. Castro, T., Fuks, H., Castro, A. (2008). Detecting Code Evolution in Programming Learning. In the proc. of the 19th Brazilian Symposium on Artificial Intelligence. Series: Lecture Notes in Computer Science, Vol. 5249. Sub library: Lecture Notes in Artificial Intelligence, pp.145-156.
6. Castro, T., Fuks, H., Castro, A. (2008). Programming in Groups: a Progression Learning Scheme from the Individual to the Group. FIE - Proc. of the 38th Annual Frontiers in Education Conference. IEEE Catalog Number: pp F1F15-F1F20.
7. Cohen, S. (1997).What Makes Teams Work: Group Effectiveness Research from the Shop Floor to the Executive Suite. Journal of Management, Vol. 23, No. 3, 239-290.
8. Delval, J. (2002). Introdução à Prática do Método Clínico: descobrindo o pensamento das crianças. ARTMED Press.
9. Freudenberg, S., Romero, P., du Boulay, B. (2007). 'talking the talk': Is intermediate-level conversation the key to the pair programming success story? In Proc. of Agile 2007, IEEE Computer Society, pages 84-91.
10. Gerosa, M.A., Pimentel, M., Fuks, H. and Lucena, C.J.P. (2006). Development of Groupware based on the 3C Collaboration Model and Component Technology. 12th International Workshop on Groupware – CRIWG Lecture Notes on Computer Science LNCS 4154, Springer-Verlag, pp. 302-309.
11. Hutchley, I & Wooffitt, R. (2008). Conversation Analysis, 2nd edition. Polity Press. USA.
12. McDowell, C., Werner, L., Bullock, H., Fernald, J. (2004). The Impact of Pair Programming on Student Performance, Perception and Persistence. In the proc. of the International Conference on Software Engineering, pp. 602.
13. Mendonça, A. P. ; Castro, A. N. ; Mota, E.S. ; Silva, L. S; Pereira, V. L. S. (2002). Uma Experiência com o uso de Mapas Conceituais para Apoiar o Método da Controvérsia Acadêmica. In: XXII CSBC - VIII WIE, Florianópolis-SC-BR. SBC Press, v. 5. p. 99-107.
14. Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., Balik, S. (2003). Improving the CS1 experience with pair programming. In Proceedings of the 34th SIGCSE technical symposium on Computer science education., pp. 359 – 362.
15. Pastel, R. (2006). Student assessment of group laboratories in a data structures course. In Journal of Computing Sciences in Colleges, v. 22, issue 1, pp. 221 – 230.
16. Pereira, V. L. S.; Castro, A. N. ; Mendonça, A. P. ; Silva, L. S. (2002). Análise do método Jigsaw de aprendizagem cooperativa através da utilização de mapas conceituais. In: XXII CSBC - VIII WIE, Florianópolis-SC. SBC, v. 5. p. 181-188.

17. Peres, F., Meira, L. (2003). Educational software evaluation centered on dialogue: interface, collaboration and scientific concepts. In Proceedings of the Latin American conference on Human-computer interaction. Pp. 97 – 106.
18. Pimentel, M., Escovedo, T., Fuks, H. and Lucena, C.J.P. (2006). Investigating the assessment of learners' participation in asynchronous conference of an online course. 22nd ICDE - World Conference on Distance Education. Publisher: ABED, Rio de Janeiro, Brazil, Sep, 3-6.
19. Robertson, D. (2004). Multi-agent Coordination as Distributed Logic Programming. Lecture Notes in Computer Science, vol. 3132/2004. Pp. 77-96.
20. Searle, J. (1969). Speech Acts: an Essay in the Philosophy of Language, 1st edition, 31st printing (2009). Cambridge University Press. USA.
21. Sharan, S. (1999). Handbook of Cooperative Learning Methods. The Greenwood educators' reference collection. Praeger Publishers.
22. Silva, L. S; Castro, A. N. ; Mendonça, A. P. ; Pereira, V. L. S. (2002). Mapas Conceituais como suporte à estratégia de Investigação em Grupo: Uma experiência na Universidade. In: XXII CSBC - VIII WIE, Florianópolis-SC. SBC, v. 5. p. 163-172.
23. Stahl, G. (2006). Supporting group cognition in an online math community: a cognitive tool for small-group referencing in text chat. Journal of Educational Computing Research.
24. Vygotsky, L. S. (1978). Mind in Society: the Development of Higher Psychological Processes. Harvard University Press. London, UK.
25. Weinberg, G. (1971). The Psychology of Computer Programming. Computer Science Series. Litton Educational Publishing.
26. Schummer, T. and Lukosch, S. Patterns for Computer-Mediated Interaction. Wiley Software Patterns Series. 2007.