Universidad de Castilla-La Mancha
Escuela Superior de Informática
Departamento de Informática

PhD Thesis

Enhancing Collaborative Learning
Using a Simulated Student Agent

July 2001

Student : Aurora Vizcaíno Barceló
Supervisors: Dr. D. Benedict du Boulay
Dr. D. Manuel Prieto Méndez
Dedicatoria

A José Luis y Aurora, mis padres, por toda su dedicación y amor incondicional. Me gustaría que cada palabra de esta tesis fuera una frase de agradecimiento para ellos.

A Javier por soportar con resignación mis ausencias y animarme a seguir adelante.
Acknowledgements

How often I thought that I was walking this long road alone, and now I realise how many other people were walking at my side or smoothing the path for me.

I will try to name them in chronological order so as not to forget anybody.

I would like to thank Manuel Prieto, the first person to encourage me to do this thesis and who supported me throughout.

Many thanks to Pierre Dillenbourg for helping me with the groundwork to this thesis.

Many thanks to Jesús Favela who was at my side along the whole length of the road.

Many thanks to Ben du Boulay I was extremely lucky to meet you!! His supervision has been invaluable to me.

Coral Calero was another of the people who, apart from travelling the whole road with me, also picked me up when I fell down and warned me about rocks in the road ahead, thus preventing me from falling down again.

I am extremely grateful to the Alarcos group whose support was very important. I particularly wish to thank Mario for his help and advice, and Paco for all those encouraging chats. I hope to be able to have another coffee with you soon!!

My most sincere thanks to Sally whose help with the English was indispensable.

Many thanks to Francisca Perea for the help with the statistics, and for all the good times we have spent together in our office.

Many thanks to Camelia and Alfonso for allowing me to use their students in my experiments. And, of course, thanks to all the students who took part in the experiments.

A very special thanks to Mercedes Velasco without whose help HabiPro would not exist.

I am also very grateful to Juan Pablo Soto for his help at the experimental stage.

Thanks to Amy Soller for her suggestions for Chapter 2.

Thanks to all my friends who always encouraged me and believed in me.

And finally, many thanks to anyone else who may have directly or indirectly contributed to the finalisation of this thesis.
ABSTRACT

1 INTRODUCTION

1.1 Collaborative Learning and Computer Supported Collaborative Learning. Some challenges

1.2 Programming

1.3 Hypothesis and Goals

1.4 Structure of the Thesis

2. COLLABORATIVE LEARNING AND COMPUTER SUPPORTED COLLABORATIVE LEARNING

2.1 Collaborative Learning

2.1.1 Constructivist and Sociocultural Theory

2.1.2 What is Collaborative Learning?

2.2 Computer Supported Collaborative Learning

2.2.1 The Role of the Computer

2.2.2 CSCL and CSCW

2.2.3 Characteristics of CSCL

2.2.4 Advantages of CSCL

2.2.5 Challenges of CSCL

2.3 Pedagogical Agents
2.3.1 Simulated students in CSCL environments 33
2.3.2 Limitations of Simulated Students 34
2.3.3 Examples of Applications 35
2.4 Summary 43

3 LEARNING AND TEACHING PROGRAMMING

3.1 Difficulties to Teach and to Learn Programming 45
3.1.1 Subtasks of Programming 46
3.1.2 Procedural learning 48
3.1.3 Students Difficulties in Procedural Aspects of Programming 48
3.2 Psychology of Programming 50
3.3 Programming Activities 51
3.3.1 Analysis and Design 52
3.3.2 Coding 54
3.3.3 Program comprehension 55
3.3.3.1 How are programs comprehended? 55
3.3.3.2 Programming comprehension models 56
3.4 Debugging 58
3.4.1 Types of bugs 60
3.4.2 Locating Bugs 62
3.4.3 Bugs versus Misconceptions 65
3.5 Maintenance and Documenting 66
3.4 Systems to Teach and to Learn Programming 66

3.4.1 Analysis of the Systems 72

3.5 Summary 75

4. MODELLING A SIMULATED STUDENT TO DETECT SITUATIONS THAT HAMPER COLLABORATION: IMPLEMENTATION IN THE HABIPRO SYSTEM

4.1 Model of the Simulated Student 77

4.1.1 Description of the Model 78

4.1.2 The Student Model 79

4.1.3 Group Model 81

4.1.4 The SSBM. 82

4.2 Roles of the Simulated Student 86

4.2.1 Passive Students 86

4.2.2 Disoriented Students 89

4.2.3 Inefficient Conversations 90

4.3 HabiPro 92

4.3.1 HabiPro Interface 92

4.3.2 HabiPro Architecture 95

4.3.3 The Simulated Student in HabiPro 99

4.3.4 HabiPro a System to Develop Good Programming Habits 107
5. EVALUATION

5.1 Introduction ................................. 116

5.2 Objective ................................ 116

5.3 Hypothesis ................................ 117

5.4 Design of the Experiment .................. 117

5.4.1 Subjects ................................ 119

5.4.2 Procedure ............................... 120

5.4.3 Materials ............................... 122

5.4.4 Dependent Variables .................... 123

5.5 Results .................................. 124

5.5.1 Does the Simulated Student Influence the Rate of Solving Problems .... 127

5.5.2 Comparison with the Results of the Pre-Test .................. 133

5.5.3 Did The Simulated Student Detect when Students Needed Help to Solve the Exercises? ............ 135

5.5.4 Students’ Assessment of the Assistance Offered by Their Partners .. 139

5.5.5 Did the Simulated Student Detect Passive Students? ............. 142

5.5.6 Students’ Assessment of the Degree of their Partners’ Participation 145

5.5.7 Did the Simulated Student Detect Off-topic Conversations ....... 147

5.5.8 Students' Impressions .................... 149
Computer Supported Collaborative Learning offers students many advantages. Learners may work with other students, and can therefore exchange ideas, and reflect upon other points of view. However, when students work in a group certain circumstances may occur that hamper collaboration or learning. For example, when there is a passive student in a group, the group's performance is usually lower than when all the students participate actively. Solutions to control these "negative" situations should be researched.

In this thesis we present a possible solution to the problem: using a Simulated Student to monitor students' behaviour in order to detect and avoid negative situations. This work describes the architecture of the Simulated Student and explains its implementation in a real system, HabiPro. The roles and interventions of the Simulated Student to avoid three circumstances: passive students, off-topic conversations and students who have difficulties in finding the correct solution, are also outlined.

HabiPro is a collaborative, synchronous and distributed system that was implemented to help students to develop good programming habits, principally: debugging, programming comprehension and style of programming, three very important but rarely taught aspects on programming courses. HabiPro enables students to solve programming problems collaboratively even though students are in different geographical locations since users communicate with each other via a chat window.

In order to validate the effectiveness of the Simulated Student's detection of these three situations an experiment was performed. Twenty-two pairs of students worked with two HabiPro versions. One version contained the Simulated Student and the other version did not. The experiment showed that the Simulated Student detected and solved the problems in the majority of the cases; in addition students solved more exercises using the version with the Simulated Student. The experiment also revealed that students considered HabiPro a good learning method.
CHAPTER 1: INTRODUCTION
Chapter 1. Introduction

Many experiments with or without new technology have demonstrated the benefits of Collaborative Learning. When students learn in collaboration their motivation for learning is generally greater than when they study alone. They explain and justify their opinions, reflect upon their knowledge, and encourage each other to work. However in Collaborative Learning, circumstances can also arise that decrease the benefits of work in collaboration or impede that these benefits exist. We call these circumstances "negative situations". Most of these negative situations are related to human behaviour. In Computer Supported Collaborative Learning (CSCL) these negative situations also take place. In fact, CSCL has inherited them from collaborative learning environments without computers. The question is, can CSCL avoid or correct any of these negative situations?

1.1 Collaborative Learning and Computer Supported Collaborative Learning. Some challenges

Many studies have established that group learning can be highly efficient: Collaboration fosters the development of self-regulatory skills (Blaye and Light, 1995). Students learning in small groups encourage each other to ask questions, explain and justify their opinions, articulate their reasoning and elaborate and reflect upon their knowledge (Soller et al., 1998).

The educational value of students' collaboration has led to the development of Computer Supported Collaborative Learning tools (Lund, Baker, and Baron, 1996). CSCL tools, besides incorporating pedagogical strategies that encourage learning, include new characteristics in order to support communication and collaboration while students are learning.

On the other hand, in collaborative environments, circumstances which reduce the benefits of work or study in a group may arise. For example, when there is a passive student in a group, motivation for working usually decreases since the rest of the students may feel uncomfortable working with a person who does not take part either in helping with or in solving the exercises. Another example of a negative situation is when one student begins to chat about other topics not related to the topic to be learned. In this case, the students'
attention is diverted towards another subject and it is sometimes difficult to redirect their attention back towards the exercises.

CSCL systems focus principally on supporting different types of communication, improving the transmission speed or providing situations where students have to collaborate. Despite the fact that students do not necessarily have the social skills they need in order to collaborate effectively (Cappozzi, Rothstein and Curley, 1996) few collaborative systems have been designed to support or guide students in learning these skills.

Currently some research groups are beginning to investigate how collaborative applications can detect negative situations and attempt to avoid them. The University of Pittsburgh is working on the EPSILON project studying a specific situation that they call "a failure knowledge sharing episode". This is defined as a segment of interaction during which one team member has not shared new knowledge with the group. The episode is considered effective if one or more students also learn the new knowledge and is otherwise thought to be ineffective (Soller and Lesgold, 2000). Another system that attempts to correct negative situations is presented by Okamoto (Okamoto, 1996; Inaba and Okamoto, 1997). They describe a system that can diagnose and coordinate flows of discussion through students' conversations. In the Monterrey Institute of Technology, Constantino-González has developed COLER (Constantino-González and Suthers, 2000). This is a Web based collaborative learning environment in which students can solve database-modelling problems while working synchronously in small groups at a distance. COLER controls students' interventions, and acts when the degree of intervention is not adequate.

The first two systems base their study of the learners on an analysis of groups' conversations. Because of this their chat interactions are structured and do not enable free communication. However, structured chat may hinder or slow students' learning, since besides thinking about the problem students have to think about the type of intervention that they are going to make.
Chapter 1. Introduction

We argue that, besides the conversations, collaborative systems can use other types of information such as an index or frequency of proposals corrected by students or students' mistakes, in order to obtain a model of the students' behaviour.

Situations that devalue the benefits of collaborative learning are many. The following is a list which describes circumstances that may be negative in collaborative environments. Although some research is being done in this field, it is an area where there is still much room for investigation.

- Feeling of Individualism. This feeling is the opposite of interdependence. When, in a group, a student thinks that s/he is more intelligent than the others and tries to solve the problems by him/herself without consulting or collaborating with other group members the feeling of interdependence disappears. A feeling of individualism emerges in the student who does not collaborate, and the rest of the members in the group feel frustrated.

- Insufficient Cooperation. Collaboration is not the same as cooperation. If students divide the work and each student solves one subtask, cooperation rather than collaboration exists. In this case, unequal learning can occur, and the benefits of work in collaboration reduce since students work individually on a sub-problem.

- Passive member. When a member does not take part in solving exercises. This situation is negative for two reasons. The passive students usually learn less than they would if they were working actively in the group, and the group members feel uncomfortable working with a passive student.

- Tempted member. A group member tries to persuade his/her other companions to give up the task.
Off-topic conversations: Sometimes, when people are working or learning in a group, one or various members begin to talk about topics that are not related to the problem that they must solve. This may result in students reducing their productivity, or in one or more of them thinking that they are wasting their time.

Another negative situation is related to the problem solving strategies used by the students. On some occasions, groups waste a lot of time trying to solve a problem in the wrong way. This causes them to feel unmotivated.

Our research is centred on three specific situations which take place in collaborative environments frequently. The three circumstances are briefly described below, a more detailed description can be found in chapter 4:

When students talk for extended periods about topics not related to the problems to be solved. In this situation, the students' attention is diverted from the problem and it may be difficult to reinstate their concentration. Besides, students who are interested in solving exercises could feel that they are wasting their time. As Sipusic et al. (1999) claims, more interaction among participants in a collaborative learning group would be beneficial for learning, however one exception is if the discourse is mostly off-topic and detracts from the time and effort devoted to learning.

When students are passive. This situation is negative for the passive members and for the group. It is negative for the passive member because s/he learns less and for the group because the feeling of interdependence disappears. A team's learning potential is maximised when all the students actively participate in the group' discussions (Soller et al., 1998).

When a group wastes a lot of time trying to solve a problem in an incorrect way. This situation can became negative because students lose motivation and may even abandon the activity.
Many of the above situations are avoided or corrected when a teacher, coach or tutor monitors learning in the group. A coach can intervene and indicate what is the cause of a peer or group interaction impasse (Katz and O'Donnell, 1999). The presence of human coaches playing the above roles can improve the efficiency of communication and learning in CSCL applications. However, it is not always possible to have one coach per group to monitor their work. Nevertheless, it is possible to add a Simulated Student which plays the role of a "responsible student". This role would be similar to that of a real student but would at the same time prevent the occurrence of the three negative situations or try to reduce their effects when they have already arisen. Designing and implementing an efficient Simulated Student which detects the three previous situations and avoids their negative effects has been one of the critical goals of this thesis.

1.2 Programming

In order to test the efficiency of the Simulated Student, a learning system with a specific domain was chosen. Several reasons led us to choose programming:

- Programming is a topic where students have a high level of failure. Last year at the Castilla-La Mancha university 339 students were registered in the first course of programming, but only 131 learners went to the exam and of them, 90 passed. Learning Programming is a procedural process, characterised by needing practice as well as theory. It is a topic that must be learnt "by doing" rather than memorising. To learn programming means to acquire several skills, for instance abstraction or debugging. However, in most degrees, more hours are dedicated to theoretical classes than to practical ones. Students often complain about the small amount of laboratory hours that they have in which to practice programming. In summary, two reasons why programming is a difficult topic to learn are the procedural character of the discipline and the fact that programming involves a great variety of subtasks and types of specialised knowledge (Pennington and Grabowski, 1990).
Chapter 1. Introduction

- Programming is a suitable topic to learn in collaboration. When students begin to learn programming, apart from buying a book about the language that they are using, on many occasions they also attend courses and if they have access to Internet, they join message lists, or news and work groups where they can ask questions or seek advice from other programmers. So students look for the experience and collaboration of other people. This shows that in learning programming, collaborative techniques are often used because students join together to write programs and to seek advice about their doubts in a spontaneous and natural way (Vizcaino et al., 2000). Moreover, professional programmers generally program in collaboration with other work-mates, so it seems sensible that students should get used to working in a group.

- It was relatively easy to do experiments and tests with our own students so we could obtain feedback quickly.

- The fact that both thesis supervisors have a wide background in learning programming was a further incentive.

Many applications have been developed to help students to learn programming or create and debug programs. Some examples are: Lisp Tutor (Anderson and Reiser, 1985), which is a tutor to program in Lisp. When students deviate from the correct solution path, the tutor gives specific feedback and requires the student to try again. Talus (Murray, 1986) relies on its abilities to reason out computational semantics to perform algorithm recognition, infer code teleology and to automatically detect and correct non-syntactic errors in student programs written in a restricted, but nontrivial, subset of LISP. Ceilidh (Benford et al., 1993) is a framework which provides computer-based support for course administration and the teaching of computer programming such as C, C++ or Pascal.

It is surprising that even though programming is a convenient topic to learn in collaboration, none of the above tools have collaborative characteristics. On the other hand most of them have been designed to correct syntactic and semantic problems, and only Ceilidh has a part focus on the style of programming. A good style of programming helps
Chapter 1. Introduction

us to understand a program, and it is easier to demonstrate that it is correct (Adam and Laurent, 1980). Because of this, students should learn and develop a good style of programming at the same time as they learn the syntax and semantics of a programming language.

Program comprehension and debugging are activities involved in learning programming and both activities need the development of new skills. Surprisingly, these tasks are seldom taught explicitly by teachers or learning applications.

The system presented in this thesis, HabiPro (from the Spanish "Habitos de Programación"), has tried to fill the gap that existed in programming applications. The system focuses on developing debugging skills in the students, training students to understand programs developed by other programmers and showing students the advantages of using a good programming style. This is another important aspect of this thesis, and for this reason, chapter three has been dedicated to the psychology of programming.

1.3 Hypothesis and Goals

The sub-goals which we concentrated on in order to develop this thesis were:

- The study of the role of the computer in educational environments, specifically in collaborative learning circumstances. The main goal was to look for possible gaps in CSCL and discover what aspects could be improved.
- The design of a model for a Simulated Student which would control and avoid negative situations, thus augmenting the benefits of collaborative learning.
- The search for a topic to teach, that would be suitable for collaborative learning.
- The study of the psychology of programming, and comparison and analysis of different systems to learn programming in order to detect their advantages and deficiencies.
- The design and implementation of a collaborative system to develop good habits in programming.
- Implementation of the model in the collaborative system developed previously.
Chapter 1. Introduction

- The study of the different types of empirical validation.
- The design of experiments in order to validate the hypothesis.
- Experimentation and validation of the Simulated Student and of the system.
- Analysis of the results and conclusions.

These sub-goals may be summarised in one principal goal: Design and implementation of a Simulated Student for collaborative learning systems in order to detect and avoid negative situations that hamper collaborative learning.

In order to reach the principal goal a hypothesis was made:

Hypothesis: It is possible to design and to implement a Simulated Student for collaborative learning systems in order to detect and avoid negative situations that hamper collaborative learning.

If the sub-goals, and consequently the principal goal, could be achieved then the original hypothesis would be considered correct.

1.4 Structure of the Thesis

The following chapter (Chapter 2) reviews research on Collaborative Learning and Computer Supported Collaborative Learning. Different types of software agents incorporated into CSCL are analysed. The chapter concludes with a description of the role of Simulated Students in learning environments. Chapter 3 gives a brief review of the philosophy of programming and explains why programming, in many cases, is a difficult discipline for students to learn and for teachers to teach. The chapter finishes with a description of the main features of the most well-known applications designed to teach and to learn programming. Chapter 4 describes, in detail a general model of a Simulated Student and the particular Simulated Student which was implemented in HabiPro, a system to develop good programming habits. This chapter also presents the difference between HabiPro and other systems that have already been designed. Chapter 5 describes the experiment performed in order to check whether the Simulated Student played its roles
correctly, and to assess students' attitudes towards HabiPro. Finally, Chapter 6 reviews whether the sub-goals of the thesis and in consequence the main goal have been achieved. Finally future work is described.
CHAPTER 2. COLLABORATIVE LEARNING AND
COMPUTER SUPPORTED COLLABORATIVE
LEARNING
This chapter presents a review of Collaborative Learning and of Computer Supported Collaborative Learning (CSCL). The first section provides a brief description of collaborative learning. The second section presents Computer Supported Collaborative Learning and explains the different roles that computers have played in CSCL. CSCL characteristics and challenges are also presented. The third section describes the use of pedagogical agents, and focuses on the different behaviours that a simulated student can adopt. The chapter finishes by describing several applications with simulated students.

2.1 Collaborative Learning

In both psychological and educational research, there is considerable interest in small group as opposed to individual learning. Much research has established that group learning can prove highly effective with several advantages over individual learning. This section explains the theoretic bases of collaborative learning and some possible definitions.

2.1.1 Constructivist and Sociocultural Theory

Many pedagogical and psychological theories have dealt with collaborative learning but two major theoretical perspectives have dominated research in this area: sociocognitive and constructivist theories derived from Piaget and sociocultural theory derived from Vygotsky. In the next paragraphs both theories are briefly explained.

Although in his later writings Piaget was largely concerned with individual development, in his earlier work (Piaget, 1932) he emphasised the importance of social interaction on individual development of formal thought. The constructivist theories are based on learning as an active process where learners construct new ideas or concepts based on their previous and current knowledge. Piaget presents a new learning conception. The conception of knowledge as a construction that a student performs through an interaction with the medium. Related to the constructivist perspective and to social learning is the research of the so-called neo-Piagetians, which has emphasised the importance of peer interaction for cognitive development (Doise and Mugny, 1984).

The second major theoretical influence on research into collaborative learning comes from Vygotsky. A central idea in Vygotsky’s theory is that all individual development has its
Chapter 2: Collaborative Learning and Computer Supported Collaborative Learning

origins in social processes (Vygotsky, 1962; Vygotsky, 1978). Whereas a Piagetian approach to collaborative learning sees social interaction as providing a catalyst for individual change, in research from a Vygotskian perspective, inter-psychological processes are not merely catalysts for change but are themselves internalised by the individuals involved. This means that attempts to coordinate perspectives and co-construct hypotheses to arrive at a joint solution are more valuable than simply having differences in perspective. Discussion is all-important, since Vygotsky held that socially-based dialogue formed the foundation for subsequent "inner dialogues" which reflect thought and action. There is thus a causal relationship between social and the individual processes.

Another aspect of Vygotsky's theory is the "Zone of Proximal Development" (ZPD). Vygotsky defined ZPD as: "The discrepancy between a child's actual mental age and the level he reaches in solving problems with assistance" (Vygotsky, 1962).

A fundamentally important feature of the ZPD is the necessity for collaboration or assistance from another more able partner. The need for this more able learning partner arises from the belief that the activities which form a part of the child's education must be beyond the range of his/her independent ability. Within a Vygotskian paradigm effective instruction involves the teacher (or more able peer) acting as a partner, enabling the child in her pursuit of success. This requires the provision of appropriately challenging activities and the right quantity and quality of assistance (Luckin and du Boulay, 1999). Vygotsky's sociocultural approach to learning and the ZPD are successfully employed in the study of computer supported collaborative learning environments.

2.1.2 What is Collaborative Learning?

The broadest definition of collaborative learning could be: "Collaborative Learning is a situation in which two or more people learn or attempt to learn something together" (Dillenbourg, 1999). However, as Dillenbourg claims, each element of this definition can be interpreted in different ways. For instance "two or more" may be interpreted as a pair, a small group (3-5 subjects), a class (20-30 subjects), a community, etc. The same analysis could be done for the words "learn something" or "together". This suggests that is very
difficult to find an adequate definition of collaborative learning, not least, because it is very
difficult to find unique definitions of collaboration and, of course, of learning.

It is not productive to list all the different definitions of collaboration or learning, because
this is not the goal of this thesis; for more information see (Dillenbourg, 1999; Littleton and
Häkkinen, 1999). Nevertheless, it is important to explain the differences between two
words that (at least in Spanish) are used interchangeably: cooperation and collaboration.
Many authors (Cohen, 1994; Dillenbourg et al, 1996; Pauli and Reusser, 1997) distinguish
collaborative learning from cooperative learning. In cooperative learning there is a
distribution of tasks among the students of the team so that each student contributes to the
common goal by working on a distinct subtask. However in collaborative learning
environments students have to work all the time on one and the same problem, there is not
an explicit partition of task. A spontaneous partition of roles may emerge where the person
who most dominates the topic plays the actor’s role while the others are observers that
control the situation. The observers can contribute with criticisms or propose ideas
(Miyake, 1986). For example, in computer supported tasks, the participant who controls the
mouse tends to be an “executor” while the other persons are likely to be “reflector/s”
(Blaye et al., 1991).

If defining collaborative learning is a difficult task due to the different nuances that
collaboration and learning can have, a more difficult task is to design an effective
collaborative learning situation, with or without computer. The following is a list of eight
factors that should be taken into account when a collaborative situation is being designed
(Bannon, 1995):

- The nature of the collaborative task: puzzle solving, editing a newsletter.
- The nature of the collaboration: peer, teacher-student, student-computer.
- The number of collaborators: 2, 3, 50.
- The previous relationship between collaborators: how many shared experiences
  have they had.
Chapter 2: Collaborative Learning and Computer Supported Collaborative Learning

- The motivation for collaboration: intrinsic interest, experimenter defined task, money.
- The setting of collaboration: classroom, workplace, home.
- The conditions of collaboration: physically co-present, computer-mediated.
- The time-period of collaboration: minutes, hours, days, weeks, years.

When the collaborative learning situation involves using computers, further factors must also be considered. The next section explains the relationship between computers and collaborative learning and lists the new factors, which must be taken into account in collaborative learning applications.

2.2 Computer Supported Collaborative Learning

Although research into collaborative learning has a fairly long history, dating back at least to the early work of Piaget and Vygotsky, it is only about a decade ago that the computer based learning systems started to include specific characteristics in order to support collaborative learning.

Computers are often used in schools with small groups of pupils working jointly around the machine. For example, Jackson, Fletcher and Messer (1986) report a survey of microcomputer use in a primary school in which a high percentage of the teachers surveyed reported using groups of children rather than individuals at the computer. Peer interaction around the computer is therefore a widespread practice, although such collaboration may occur for no better reason than scarcity of computing resources (Crook, 1987). Anderson, Mayes and Kibby in (Anderson et al., 1995) complain about such collaboration, saying that an important limitation of such collaboration is the fact that the programs involved are often designed with the single user in mind, rather than a group of users. They affirm that simply arranging a group of students or pupils around a computer running a traditional CAL (Computer Aided Learning) program designed for use by individuals and exhorting them to collaborate will not guarantee success. The application used in collaboration must be designed for this purpose, taking into account that the tasks to be performed enhance the
collaboration, that the system supports communication and collaboration among group members and favours the cognitive processes of group learning.

Besides the fact that people used computers in groups for pragmatic reasons another reason that led to the design of collaborative learning applications was the fact that some computer-implemented tasks designed for individual use, such as those involving discovery learning, were enhanced by collaborative rather than individual use (Anderson et al., 1995).

Damon and Phelps (1989) claim that peer collaboration can stimulate the challenges of discovery learning, create an engagement rich in mutual discovery, reciprocal feedback, and frequent sharing of ideas, and can provoke deep conceptual insights. Combining peer collaboration with discovery learning, therefore, should result in a highly stimulating learning environment. Strathtutor (Anderson et al., 1995) was designed in order to test Damon and Phelps’ ideas. Strathtutor was a learning-by-browsing system with different mechanisms for navigation among frames. Pairs of students had to learn different concepts about glaciation by exploring the Strathtutor hypertext. One conclusion was obtained from this work. It was demonstrated that the combination of a conceptually challenging hypertext and the requirement to explore this jointly with another learner produced a highly effective learning event. Unfortunately the authors did not research whether it was possible that the same learning gains might have resulted from these subjects working independently. Other applications such as Envisioning Machine (Roschelle and Teasley, 1995), and HyperCard (Blaye and Light, 1995) were developed in order to test the efficiency of collaborative learning applications for peers. There have been numerous studies that have compared the performance of people working individually versus those working in pairs with computers. Blaye et al. (1991) found that pairs performed better than individuals on a computer-based problem and that this beneficial effect transferred to later individual work. Similar beneficial effects have been reported by Amigue (1989), Amigue and Agostinelli (1992), Mevarech, Stern and Levita (1987) and Mevarech (1993). However, Jackson and Kutnich (1996) report a study on the effects of peer interaction which found that individuals performed better than pairs. They conclude that the beneficial effects of peer interaction are dependent on the nature of the task.
Others have investigated the types of group interaction and how that effects learning. One set of studies has compared people working competitively with those working cooperatively. For example, Johnson, Johnson and Stanne (1985) and Johnson and Johnson (1986) found that children who worked cooperatively were more successful in solving a complex computer-based problem that children who worked either competitively or individually and that they performed better on subsequent individual post-tests.

Developments in technology, such as networks, faster computers and more concretely, shared screen facilities, interactive video or groupware applications provides new opportunities for CSCL. For example, collaborative learning applications can enable students to collaborate even though they are in different geographical locations.

2.2.1 The Role of the Computer

This section explains the roles the computer can play in collaborative learning. Bannon (1995) refers to Kurland and Kurland (1987) and to Collins' (1984) work where various roles of the computer are explained: It can allow one to simulate situations that would be impossible in the real world; it can maintain traces of student actions that can be used in improving problem-solving strategies; it can rectify the process of thinking, not just the product, it can help create functional learning environments where the student can acquire goals and knowledge while pursuing goals that is meaningful for them. The authors in these studies referred to applications for individuals but many of these roles can be also extended to collaborative applications.

It is possible to divide the computer's roles into three levels. At the most basic level, the computer can be used simply as a data-gathering tool that can support the investigation of collaborative processes between people. Collaboratories belong to this level. In this case, the computer makes the task of the researcher easier but does not really affect the collaborative process per se. Because of this, this role cannot really be considered under the rubric of what CSCL implies. Examples of collaboratories are: The Collaboratory for Environmental Molecular Sciences Laboratory (http://www.emsl.pnl.gov:2080/docs/collab/) and Spectro-Microscopy Collaboratory
Chapter 2: Collaborative Learning and Computer Supported Collaborative Learning

(http://www-itg.lbl.gov/BL7Collab/). The first one joins experts of different disciplines: mainly physics and chemistry in one electronic community in order to collaborate in research and development of technology for conserving the environment. The second one enables collaborators distributed in nine research institutes to access three analytic tools which give information about materials. More innovative work provides a microworld on the computer with which students can maintain collaborative interactions. People Power (Dillenbourg and Self, 1995) is a special example of tool with a microword, since the student collaborates with a computerised learner (co-learner), and both student and co-learner learn together by using the same microword.

At a second level the computer can also be seen as a ‘tutor’ with whom the pupil interacts, or even collaborates. The conception of a computer as a tutor has received a lot of attention over the past decade, especially through the concept of the computer as a coach that could help the student in understanding a problem domain by pointing out flaws in the students’ conceptual model evidenced by their responses to problems (Bannon, 1995)

A third level is the use of the computer specifically as a medium or resource for collaborative learning. The computer can help students to communicate and collaborate on joint activities, providing assistance in the coordination process. Pauli and Reusser (1997) affirm that the role of computers in collaborative learning environments is considered as a special case of possible support structures in educational environments for collaborative learning. An example of support structure are the interfaces: Interfaces can induce a specific distribution of roles between learning partners and help to foster social interaction (O’Malley, 1992; Blaye et al., 1991). Interfaces can serve to scaffold the executive and regulative aspects of the collaborative task. On the other hand, aspects of the software can modify the socio-cognitive dynamics between the learning partners. For instance Soller and Lesgold, (1999) propose a new Collaborative Learning Model. This model identifies the characteristics exhibited by effective collaborative learning teams and, based on these characteristics, suggest strategies for promoting effective student interaction. Another example where the computer is a resource for collaborative learning is the system described by Inaba et al., (2000). This system detects an appropriate situation for a learner to join in a
collaborative learning session, and the system forms a learning group where each member is assigned a reasonable learning and social role.

2.2.2 CSCL and CSCW

Computer supported collaborative learning is a subcategory of Computer Supported Collaborative Work (CSCW). However it is important to clarify the differences between them.

CSCW is defined as a computer-based network system that supports group work in a common task and provides a shared interface for groups to work with (Ellis et al. 1991). Collaborative learning is defined as groups working together for a common purpose (Resnick, 1995). CSCW tends to focus on communication techniques themselves, and on facilitating group communication and productivity. On the other hand CSCL focuses on what is being communicated and on scaffolding or supporting students in learning together effectively. But both areas, CSCW and CSCL, have in common that they are multidisciplinary and must take account of not only the techniques of the groupware but also their social, psychological, organisational, and learning effects.

![Diagram showing the relationship between CSCW and CSCL](image)

Figure 1. Relationship between CSCW and CSCL

2.2.3 Characteristics of CSCL

CSCL adds the advantages that the new technology offers to the advantages that collaborative learning has: it enables students to work in a group, although they are in different places, to share information and interchange ideas. However not all applications which permit synchronous or asynchronous communication and which store information can be defined as CSCL applications.
Many applications have appeared with the goal of teaching. One example includes virtual classroom systems which support e-mail, video-conferencing and other mechanisms that allow students to interact with the teacher or a partner in a distributed way. Another example is hypermedia systems such as Mosaic (AndreeSEN, 1993), which have mechanisms to structure large amounts of information and also permit the integration of several formats such as: graphics, voice, video and, of course, allow information sharing. Both virtual classroom systems and hypermedia systems are very useful for accessing information, but neither of them offers mechanisms to help students to understand new information more easily. One characteristic of CSCL is that it must give support in order for students to understand the new information, relate it with the previous information that they have already had, and compare and identify different interpretation of the same information (Wan and Johnson, 1994).

Another characteristic of CSCL systems is that they normally do not guide students strongly during the learning session. This is an important difference from early Intelligent Tutoring Systems such as Lisp Tutor (Anderson and Reiser, 1985) which intervenes as soon as the student types in some incorrect code. In many situations quick feedback did not help students to learn to think and solve problems by themselves. Immediate feedback was seen as a critical feature in the first generation of educational software (Dillenbourg et al., 1996). For this reason, as Crook (1998) explains, experts in education were very critical of such systems as pedagogical tutors because these systems restrict students’ experience. The style of the feedback is important Lepper et al., (1990) found that students chose to solve more difficult problems when teachers used indirect feedback such as questions than when teachers used a more direct style. These results suggest that the use of subtle cues to guide and support students maximises the motivation to learn (Merrill et al., 1992).

CSCL applications, by contrast, encourage students to discuss possible solutions via a shared Notebook as in CoVis (Rammamurthy et al., 1995) or other communication modes (text, video, audio, animation) as in CSILE (Scardamalia et al., 1994). In this way students have the possibility of reflecting on their own beliefs and knowledge and obtain a more lasting learning.
A very important aspect of collaborative applications is communication. Text has been the most used technique, principally because it needs few technological requisites. Currently, many applications add support for video or audio, trying in this way to support a wider range of communication channels in order to facilitate collaborative processes such as, for instance, reflection. Communication aspects can affect collaborative attitude. For instance, research has shown that establishing face-to-face contact seems to be important during reflection stages Dillenbourg et al. (1996)

2.2.4 Advantages of CSCL

CSCL has inherited the advantages that collaborative learning has. The following paragraphs describe them.

Students working in small groups tend to learn more of what is taught and retain it longer than when the same content is presented in other instructional formats. Students who work in collaborative groups also appear more satisfied with their classes (Beckman, 1990; Chickering and Gamson, 1991; Collier, 1980; Cooper et al., 1990; Johnson and Johnson, 1987; Johnson, Johnson and Smith, 1991).

Collaboration in planning and problem solving activities fosters the development of self-regulatory skills (Blaye and Light, 1995). Improvements in general categorisation and classification skills have also been shown through collaborative learning in memory or taxonomy tasks (Rogoff, 1990). It seems then that collaborative learning can produce a variety of gains in both general abilities as well as in more domain specific skills.

Damon and Phelps (1989) claim that peer collaboration can stimulate the challenges of discovery learning, create an engagement rich in mutual discovery, reciprocal feedback, and frequent sharing of ideas, and can provoke deep conceptual insights. Combining peer collaboration with discovery learning, therefore, should result in a highly stimulating learning environment.
Chapter 2: Collaborative Learning and Computer Supported Collaborative Learning

Classroom learning improves significantly when students participate in learning activities with small groups of peers (Brown and Palincsar, 1989; Doise, Mugny and Perret-Clermont, 1975). Students learning in small groups encourage each other to ask questions, explain and justify their opinions, articulate their reasoning, and elaborate and reflect upon their knowledge, thereby motivating and improving learning (Soller et al., 1998).

In a collaborative environment three types of learning come into view:

- Learning by teaching which facilitates learning by externalisation of one's understanding. Students who teach other students learn more and better (Berliner, 1989; Palthepeu, Greer and McCalla, 1991).
- Learning by diagnosing which deepens understanding by diagnosing other learners (Ikeda and Mizoguchi, 1995). A variant of this type is learning by observing a companion learner's behaviour.
- Learning by open discussion which facilitates thinking capability through interaction. Theories of learning have long suggested that dialogue has an important role to play in shaping conceptual development (Ravenscroft and Pilkington, 2000).

Another important advantage of CSCL is that students can work in a group although they are in different places. This is an issue which enables that people with disabilities to work with other people, even though they are at home.

CSCL applications are also very useful for distance learners, learners who cannot attend traditional courses and are geographically distributed. Distance learners often complain that they do not know other students enrolled on the same course so they cannot ask questions or interchange ideas with each other. Collaborative applications make it possible for distance learners to know each other via chat or e-mail and they can work in collaboration. In Verdejo and Barros (1998) an architecture to explore collaboration at a distance is described. The architecture enables students to work collaboratively in small groups to carry out learning activities previously designed by a teacher. An asynchronous communication was used; there were two reasons for this choice: distance learning students
rarely have compatible schedules to permit real time sessions. Furthermore, the kind of learning considered needs thought and reflection: an asynchronous communication is well suited in order to foster reflection because it allows each student to work at his own pace.

2.2.5 Challenges of CSCL

CSCL has many advantages, as has been seen in the section above. However for efficient collaboration to take place certain conditions must apply. Simply to form a group of people and ask them to solve a problem is not enough to ensure that an efficient collaboration will appear. The literature describes characteristics that are exhibited during effective collaborative learning interactions. Seven of these characteristics are explained as follows:

- **Interdependence**: collaboration is more efficient when students in the group perceive that their goals are positively correlated such that an individual can only attain his goals if his team members also attain their goals (Soller et al., 1998). Salomon (1995) affirms that all collaborations are based on a genuine opportunity for interdependence; if one student has some information that another student needs, and the second student has information that the first one needs, there is a interdependent relationship so in this case the collaboration may be successful.

- **Interaction among learners**. The educational benefit that a learner gets through a collaborative learning process depends mainly on the interaction among learners. The interaction is partly influenced by relations among members of the learning group, which suggests that how to form an effective group for the collaborative learning is critical to ensure educational benefit to the members (Inaba et al., 2000). An individual’s learning achievement in a team can often be determined by the quality of his communication in the group's discussions (Johnson, Johnson, Holubec, 1990).

- **Building a shared representation of the problem**. CSCL is efficient if two people together succeed in forming a joint, unique, single cognitive system. Dillenbourg sums up this sentence with the formula "1+1=1" which means that two people share a single representation of the problem (Dillenbourg, 1996).
Chapter 2: Collaborative Learning and Computer Supported Collaborative Learning

- Adequate collaboration activities. Collaborative activities are not always useful. Because of this a collaborative learning environment must detect those activities that are not useful for learning and try to motivate other activities that help students to learn (Burton, Brna, Pilkington, 2000).

- Participation: A team's learning potential is maximised when all the students actively participate in the groups' discussions. "Building involvement in group discussions increases the amount of information available to the group, enhancing group decision making and improving the students' quality also increases the likelihood that all group members will learn the subject matter, and decreases the likelihood that only a few students will understand the material, leaving the others behind" (Soller et al., 1998, pp 187).

- Social Grounding: Social grounding is the mechanism by which two participants in a discussion try to elaborate the mutual belief that their partner has understood what they meant to a criterion sufficient for the current purpose (Clark and Brennan, 1991). Social grounding is improved when students play and interchange different roles such as: questioner, motivator or facilitator. It is assumed that there is a strong link between cognitive processes and related roles. Analysis of experiments revealed that students in effective collaborative learning teams naturally take different speaking turns by playing characteristic roles such as such as questioner, facilitator, and motivator (Soller et al, 1998; Okamoto, 1996).

- Performance Analysis and Group Processing: Group processing exists when groups discuss their progress, and decide what behaviours to continue or change. Group processing can be facilitated by giving students the opportunity to individually and collectively assess their performance. During this self evaluation, each student learns individually how to collaborate more effectively with his teammates, and the group as a whole reflects on its performance (Soller et al., 1998).
The above conditions for effective learning do not always appear when a group is working. Sometimes other factors intrude which instead of encouraging collaboration and learning decrease them. These types of situations as called “negative situations” and they are characterised by debilitating the group.

Salomon (1995) enumerates phenomena that debilitating a group such as: The “free rider” effect whereby one team member leaves it to the others to complete the task (Kerr and Bruun, 1983); The “sucker effect” whereby a more active or able member of a team discovers that he or she has been taken for a free rider by other team members (Kerr, 1983); the “status sensitivity” effect whereby high ability or very active members take charge, and thus have an increasing impact on the team’s activity and products (Dembo and McAuliffe, 1987); or the “ganging up on the task” phenomenon, whereby team members collaborate with each other to get the whole task over as easily and as fast as possible (Salomon and Globerson, 1987).

Other “negative situations” have been detected in collaborative learning and described in the CSCL literature. The next paragraph is a summary of negative situations, many of them are conditions that prevent the positive situations.

- Individualism, this is the opposite of interdependence. When a student thinks that s/he is more intelligent that the others in the group and tries to solve the problem by him/herself without consulting or collaborating with other group members the feeling of interdependence disappears, and a feeling of individualism emerges in the student who does not collaborate and a feeling of frustration in the rest of members in the group.

- Several situations can occur that prevent a shared representation of the problem. For instance, students may divide the labour and each student solves one subtask in isolation. In this case there is no shared representation because there is not collaboration but cooperation. Dillenbourg (1996) sums up cooperation with the formula 1+1=2. Another possible situation where a shared representation of the problem does not exist is in the case where a group member does not want to collaborate because s/he does not
want to work at that moment or in that group. In this case we propose another formula \(1+0=1/2\). Another situation occurs when a group member tries to persuade a companion to give up the task. For this case a possible formula could be \(1+-1=0\).
Related to this aspect is the ineffective knowledge sharing episode, defined by Soller and Lesgold (2000) as a segment of interaction during which one teammate tries to share new knowledge with the group but the rest of the group members do not learn the new knowledge.

- **Social grounding:** Students do not always play adequate roles in a collaborative environment or even if the students play correct roles it may happen that there are not enough students to play all the necessary roles.

- **Participation:** Sometimes one or more group members do not participate actively. It is possible that a student does not participate but still observes the companions' actions and reflects on those actions so is acquiring new knowledge and in consequence is learning. This situation would not really be a negative situation because the student finally learns. On the other hand, if a student is not interested in solving the problem or in learning and probably s/he is in the group under obligation this is a negative situation. The student possibly will not learn, or learn less than the rest of the group. As many researchers indicate, the information that is received but is not used during the learning process, is difficult to remember when need it (Schank and Kass, 1996). On the other hand, having a passive student in the group can produce situations that decrease learning such as loss of interdependence, or the feeling of having a "free rider" in the group.

- **Related communication:** Sometimes, when people are working or learning in a group, it may happen that one or more members begin to talk about topics that are not related to the problem that they must solve. If this situation is not extended it can be convenient because it helps people to get to know each other and feel more comfortable in the group. But if the situation continues for a long time it may be prejudicial for several reasons: students get away from the point and change their attention to another point.
and it is sometimes very difficult that students get back to solve the exercises. Second, if one student wants to continue working s/he can feel that s/he is wasting his/her time and may reject working in a group in future due to a bad experience. Third, students need more time to do the same task than other groups where inefficient conversations don't arise.

- Another potentially negative situation is related to pedagogical strategies. On some occasions groups waste a lot of time trying to solve a problem in an incorrect way. The fact that students try different strategies is a good pedagogic technique because students learn from their experience (Piaget, 1932), and a central part of the learning process occurs when students attempt to apply the instructional material to solve problems for themselves (Anderson, 1983; Anzai and Simon, 1979). Important learning may occur when students encounter obstacles, work around them, and explain to themselves what worked and what did not (Ohlsson and Rees, 1991; Anzai and Simon, 1979). However, this type of learning has potential cognitive and motivational pitfalls. Students trying to solve problems can expend much time and effort pursuing blind alleys because of errors or poor strategies. Of course, in some cases students may learn something valuable while searching for a solution. In many cases, however, such episodes leave students confused and frustrated. So if a group does not obtain feedback after spending a lot of time working on a task, members may lose motivation, and they may even abandon the activity.

Analysing the above negative situations we can observe that most of them are produced by social behaviour, in other words, these situations are not directly caused by introducing a computer into a collaborative learning environment. CSCL has inherited the possibility of negative situations from collaborative learning without computers. The main goal of this thesis is to explore whether CSCL can avoid or correct some of these situations.

Many of the previous situations are avoided or corrected when a teacher, coach or tutor monitors learning in the group. A coach can intervene and indicate what is the cause of a peer or group interaction impasse (Katz and O'Donnell, 1999). In a collaborative problem-
solving situation the coach can observe students’ actions and dialogue and interact with them while they are problem solving, not just afterwards. When students reach an impasse, the coach is challenged to determine why. The coach should intervene when she detects a shared cognitive deficit such as when students are referring to different domain objects that have the same or similar names. So a coach has to detect and diagnose general signs of trouble such as a shared knowledge deficit or miscommunication. After detecting and diagnosing a problem the coach has to correct it, addressing the problem and possibly tailoring his intervention to the specific nature of the impasse. So a coach performs the following actions: Detection of cues, impasse diagnosis and correction.

The presence of a human coach playing the above roles can improve the efficiency of communication and learning in CSCL applications. However it is not always possible to have a coach per group to monitor their work.

On the other hand, collaborating with a coach or teacher has some disadvantages. For instance synchronous distributed applications have the advantage that they allow students to work in a group although they are in different locations. Students have to reach an agreement only about what time they are going to connect to the application. If the application requires a teacher to connect at the same time, students cannot decide for themselves what time they are going to work but depend on the timetable of the teacher. And if another group wants to work at the same time two teachers will be needed and so on. So if teachers are a limited resource it could mean a problem.

Another disadvantage of working with a coach is that when they are working with a teacher some students feel that they are obliged to work in a group and their behaviour is not so spontaneous as when working only with companions. Dillenbourg (1999) claims that collaboration is more likely to occur between people of a similar status than between a boss and his/her employee, or between a teacher and pupils. In addition, Goodman et al. (1998) demonstrated that the interaction between students is greater than interaction with a teacher. When students receive advice from a peer, students usually reflect on and consider the proposal but if the proposal is from a teacher, students do not generally query it. Goodman
et al. (1998) illustrate, in Table 1 the difference between a student-tutor exchange in a standard intelligent tutoring system and a student-peer exchange in an intelligent tutoring system with a learning companion.

<table>
<thead>
<tr>
<th>Student/Tutor Dialogue</th>
<th>Student/Peer Dialogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student: What do you think I should do next?</td>
<td>Student: What do you think I should do next?</td>
</tr>
<tr>
<td>Tutor: Link an explanation to some evidence</td>
<td>LuCy: Link an explanation to some evidence. What do you think?</td>
</tr>
<tr>
<td></td>
<td>Student: I don't think that is a good idea.</td>
</tr>
<tr>
<td>LuCy: What would you suggest?</td>
<td>Student: Choose the best explanation.</td>
</tr>
<tr>
<td>LuCy: Which one do you think is best?</td>
<td>Student: A failure in the satellite's ground station forced the orbit change.</td>
</tr>
<tr>
<td>LuCy: Why would you choose that as the best explanation?</td>
<td>Student: I have found conclusive evidence supporting this explanation.</td>
</tr>
<tr>
<td>LuCy: I don't think that's a good reason</td>
<td>LuCy: Ask the instructor.</td>
</tr>
</tbody>
</table>

Table 1: Comparison of student/tutor dialogue vs. Student/peer dialogue

The student asked the same question in both the student/peer dialogue and student/tutor dialogue. The answer is the same in both cases, but the learning experience is very different. In the student/tutor case, the tutor gave a factual response. In the student/peer case, the learning companion gave the same information but invited conversation. It also encouraged the student to explain and articulate this thinking, which exposed a misconception. Of course, a different tutor might have given a response more like LuCy's.

In the experiment performed by Goodman and his colleagues (1998), LuCy was an interactive software peer who answered student's questions and offered suggestion much

---

like a human peer would. LuCy encouraged the student to justify decisions, consider and articulate alternative courses of action with the purpose of encouraging student reflection and articulation of past actions and future intentions (Goodman et al., 1998). LuCy collaborated with a single student. There is not yet a version for groups.

An interesting situation in collaborative learning environments would be that in each group there was a simulated peer with the expertise of a teacher and with the skills necessary to play the roles of detection, diagnosis and correction. This situation would have the advantage that the rest of the group members would not feel that they were working with a teacher or a coach.

Software agents are becoming more frequent in computer systems. LuCy is an example of a software agent. Agents are characterised by playing roles similar to human behaviour. Could an agent be the ideal student that avoids negative situations in CSCL?

2.3 Pedagogical Agents

The move from intelligent tutoring systems to pedagogical agents began about ten years ago, when researchers began to explore new types of interactions between computers and students (Johnson, 1999). For instance Chan and Baskin (1990) developed a simulated learning companion, which acts as a peer instead of a teacher. Dillenbourg (1996) investigated the interaction between real students and computer-simulated students as a collaborative social process. Chan (1996) has investigated other types of interactions between students and computer systems, such as competitors or reciprocal tutors. Dillenbourg et al. (1997) divides agents into three types:

- Sub-agents are autonomous software entities which carry out tasks for the user. Some examples of tasks accomplished by sub-agents include e-mail filtering, calendar managing and finding people sharing interests.
- Co-agents perform the same actions as the human user they interact with. For instance, the idea of a co-learner has been used within various learning paradigms: collaborative learning (Dillenbourg and Self, 1992; Chan and Baskin, 1988), reciprocal tutoring (Chan, 1996) or learning by teaching (Ur and Van Lehn, 1995; Ramirez, 1999).
• Super-agents provide solutions and monitor the action of users. In CSCL super-agents monitor the interaction among users.

We focus on the type of agent that simulates the student's behaviour and interacts with him/her as if it was a real student. In Dillenbourg's taxonomy, this kind of agent would be classed as a co-agent and (also called a simulated student).

Simulated students have been used to collaborate with real students. VanLehn, Ohlsson and Nason (1994) claim that simulated students may make it possible to improve upon traditional teaching methods where students work together in pairs or small groups. Simulated students can also be used to help teachers and instructional developers.

• Teachers can practice the art of tutoring by teaching a simulated student. Using a simulation instead of a real student allows teachers to see how their actions affect that student's knowledge, to undo their actions, and to try their skills on students with varying prior knowledge and learning strategies. Simulated students have only recently become available, even though the idea is not new (Doak and Keith, 1986). Most existing systems train teachers in activities that do not directly involve students' learning. Simulation-based systems exist that train teachers to diagnose arithmetic bugs (Brown and Burton, 1978; Nason, 1991), to organise reading groups (Shelley and Sibert, 1991) and to manage classroom motivation (Wood, 1991). Now that technology is available for simulating learning, simulations can help train teachers in activities that directly involve students' learning. An example of a tool with simulated students is Prodigy (Nason, 1993). Prodigy is a teacher training system that enables teachers to develop their skill in diagnosing students who have difficulty in fraction arithmetic. The core of Prodigy consists of a set of simulated students. Each simulated student was constructed on the basis of several hours of detailed empirical data from a particular human student who had difficulties with fractions. Each simulated student within Prodigy mimics the behaviour observed in the corresponding human student at a fine grained level. For example, the simulations mimic the handwriting and the writing speed of the observed students. Each simulated student exemplifies a different set of
bugs. The computational rules that drive the behaviour are implemented in a general and robust way, so that the simulated students behaves sensibly even if it is given fraction problems which were never given to the human students that they are based on. A session with Prodigy begins with the user selecting a simulated student. The user poses problems to that student and observes the resulting behaviour. Preliminary trials indicate that student teachers do gain an increased appreciation of the complexity of the relation between observable behaviour and the underlying procedural knowledge.

- Students can learn in collaboration with a simulated student. Because the simulated student can be simultaneously an expert and a co-learner, it can scaffold and guide the human's learning in subtle ways. The simulated peer learner can play different roles. Dillenbourg and Self's (1992) system teaches a student how to rearrange electoral districts to maximise the number of seats held by a political party. The human student and the simulated student are working for the same party, and they argue over which redistricting tactics are most likely to increase their party's winnings. Hintze' (1991) system is configured more like a help system than a collaborating partner, even though Hintze calls it a co-learner. The human student tries to discover the function of the keys on a fancy telephone. The system checks the student's hypothesis and sometimes suggests one of its own. Palthepu, Greer and McCalla's (1991) system acts like an interactive knowledge acquisition tool for inheritance hierarchies. The student tells the system facts, such as "mammals have legs", "humans are mammals" and "dolphins are mammals". The system asks questions in order to complete the inheritance hierarchy (e.g., "Are humans and dolphins the only mammals?"). All three systems are similar in that they contain only two components: a learning module that updates the simulated student's knowledge base, and a dialogue control module that decides what to say next. None of the systems contain an expert module, so none know when the students have adopted a misconception or are wasting time exploring unproductive territory.

- Instructional developers can test their instruction on simulated students. Unlike formative evaluations with real students, a simulating-based evaluation can indicate exactly what piece of the instruction caused which pieces of knowledge, and thus help
developers troubleshoot their instructional designs early in the design process. An illustration of a simulated-based formative evaluation is Sierra. Sierra is a machine learning program designed to model young children's learning of arithmetic, elementary algebra equation solving and other simple mathematical calculational skills (VanLehn, 1987, 1989). Sierra learns from solved examples. For instance, the subtraction lessons of three major textbooks were each given to Sierra. All three textbooks introduced borrowing in problems that had only two columns, such as

\[
\begin{array}{c}
85 \\
-27 \\
\hline
\end{array}
\]

This gave Sierra trouble. It could see that the digit borrowed from was always in the upper left corner (the 8, in this case), but it did not know how to generalise that location. Was it supposed to be the top digit in the left-most column or the top digit in the column adjacent to the columns initiating the borrowing? Sierra ignored the explanations that accompany borrowing because they required an understanding of the base-ten system, which Sierra (and many young children) lacked. When Sierra was given 3-columns problems that require borrowing in the units column, such as

\[
\begin{array}{c}
745 \\
-127 \\
\hline
\end{array}
\]

it would not know whether to decrement the tens columns (because it was adjacent to the column causing the borrowing) or the hundreds column (because it was the left-most column). This confusion causes Sierra to get struck (reach an impasse). Students and Sierra employ a variety of ways to work around the impasse, including omitting the borrow-from action, omitting the whole borrow, or just picking one column and hoping that it is the right one for the borrow-from action. The different repairs to the impasse correspond to different bugs. In one large study, four of the predicted bugs were observed, and two of them were moderately common, occurring in the work of six students each. Thus, Sierra was able to explain how several observed bugs were acquired.
2.3.1 Simulated Students in CSCL Environments

This section examines the role of simulated students in collaborative learning using computers. The following five points illustrate possibilities for the use of simulated students in CSCL.

- A simulated student can play a role that increases learning. For instance, Webb (1989) found that learning correlated more with the number of times a student answered a question of another student than with the number of times a student asked a question. Suppose one of the students in a pair learning situation is a simulated student. It could ask questions of the human student. Answering the questions should increase the human student's learning.

- In traditional collaborative work when a teacher has placed students in a group and given them a task, there is a little control over the group’s interactions because the teacher can only spend a fraction of his or her time with that group. On the other hand, with a simulated student as part of the group, all kinds of pedagogically beneficial interactions can be staged from within the group itself—thought provoking questions can be asked, taciturn students can be prodded to speak, bad ideas can be questioned, small slips can be caught before they have serious consequences, attention can be directed away from areas that are already mastered and towards areas where students are ripe to learn (VanhLehn, Ohlsson and Nason 1994).

- A simulated student can have one thing that a real student can never have: a second knowledge base that is nearly complete and correct. When deciding how to act, it refers both to its "own" knowledge base and to this expert knowledge base. This expertise is necessary for the teacher when he or she guides a small group, so it should be necessary for the simulated student as well. The lack of such expertise in a group composed only of human students dooms it to be less effective than one with a simulated student, in principle at least (Webb, 1989).
To sum up, we can say that simulated students can avoid some of the negative situations described in the previous section. For instance a negative situation was related to a deficit in social grounding, produced because there were not enough students to play all the roles. A simulated student could solve this problem playing the roles that are not being played by anybody, and even changing these roles depending on the situation.

A simulated student can also monitor the group and correct misunderstandings, detect miscommunications, etc. Another important advantage is that each group may have a simulated student. This simulated student is available at any time, so students do not need to worry about what time the connection is or if the partner is busy.

2.3.2 Limitations of Simulated Students

The disadvantages of simulated students are related to their design. There are limitations on the participation of simulated students in peer learning that are due to the current state of technology. The state of the art in natural language processing, and speech in particular, is not sufficient to allow machines to participate in the kinds of open-ended discussions that groups are famous for. This means that machines will only be able to participate in discussions that are mostly formal or where participants indicate the type of the contributions they make.

Ragnemalm (1996) describes the complexities involved in understanding natural language conversations between human peers, and the difficulties in developing a learning companion that could communicate freely with a human student. However VanLehn, Ohlsson and Nason (1994) illustrate with Cascade, a simulated student, that a natural interaction can be held with only limited natural language processing. The scenario also illustrates that in a formal task domain, such as physics, a pedagogically useful dialogue can take place without much natural language processing. Indeed, there are many times when the use of a text medium for a conversation implies using more concise language and less subtle cues than are used in real human-to-human conversations. This suggests that it is probably not necessary to fully understand how human-to-human dialogue works in order to build a peer problem solving system.
2.3.3 Examples of Applications

Even within the limitations of technology, several systems have been developed involving either an agent, an artificial companion, a virtual student or a simulated student. The name used is not the important thing. The most important thing is that these new partners have been introduced with the idea of helping students in the difficult task of learning. This section presents examples of simulated students which have been used with the aims of improving learning or fostering reflection.

Integration-Kid (Chan, 1991) was the first system built as a learning companion system. A learning companion is an artificial student who interacts with the human student and learns under the guidance of the computer teacher. Thus, the learning companion performs the learning task at about the same level as the student, and both the student and the companion exchange ideas while being taught by the computer teacher (Chan and Chou, 1995). Integration-Kid explores various patterns of interactions through different protocols of learning activities of the agents, such as, cooperation, modelling, and observing. The learning companion in this system is capable of collaborating or competing with the student. The performance of the learning companion in Integration-Kid is governed by a subset of problem solving expertise and some faulty knowledge. In the process of learning, this problem solving expertise is expanded and the faulty knowledge is tuned by simply deleting and adding knowledge units. Integration-Kid proved the feasibility of learning companion systems and demonstrated that there are simulated learning interactions which are not possible with a computer tutor only. This work started a discussion about the characteristics that learning companions should have: What expertise should the learning companion posses? How should it be implemented, using machine learning or simulation? What role(s) could the companion take?

From the above issues perhaps the most important are the expertise and the role that such a companion agent should have if it is going to be of educational value to the student. Integration-Kid explored the roles of collaborator and competitor, and sub-optimal expertise for the learning companion. These are only two of the roles and one expertise level that a learning companion could have. There are many other roles and expertise levels
that these agents could take. The limits are only defined by human peer learning (Ramirez, 1999).

**EduAgents system** Hietala (1996) incorporates two types of teaching agent as well as companion agents. One type of teacher agent is a behaviouristic one, while the other type has a more constructivistic approach to teaching. The learning companion agents enrich the learning situation by taking an active part in the session. They start with a small amount of domain knowledge and adapt to the learner's level of skills, for instance, by autonomously observing and extracting more knowledge from the dialogue between the learner, the teacher agents and itself.

An experiment was performed in order to test the efficiency of the system. Three (13 year old) students had three 45 minutes sessions. One interesting finding was that when the subjects were guided by the more behavioristic teacher they switched to collaboration with the companion more often than when they were working with the more constructivistic teacher. A closer analysis of the messages sent to the companion agent revealed that the subjects did not actually want to cooperate as such but rather wanted to obtain clear advice from the companion agent in order to be able to advance in their own problem solving.

The time that the students spent with the collaboration tool diminished from session to session. During the first session the average use of the collaboration tool was 6.5 minutes, while it was only 3 minutes during the third of the 45 minutes sessions. The most typical of the messages sent to the companion agent during the first session was "Please suggest a new operation" while in the third session it was "What do you think about my answer?". Thus the subjects preferred to treat the companion agent as a tutor or as an assistant teacher. After an incorrect answer from the companion agent the use of the Collaboration Tool became less frequent.

The results from the experiment provoked many questions such as: Do subjects prefer the more capable companion agents or will the companion agent they choose depend on the difficulty level of the task? Or what is the nature of the social interaction between a human
Chapter 2: Collaborative Learning and Computer Supported Collaborative Learning

student and the computerised agents? An answer to the first question can be found in Ramirez (1999) where an analysis of expertise in learning companions is described.

Steve is a advanced prototype designed to interact with students in networked immersive virtual environments, and has been applied to naval training tasks such as operating the engines aboard US Navy surface ships (Johnson et al., 1998). Immersive virtual environments permit rich interactions between human and agents; the student can see the agents in stereoscopic 3D and hear them speak, and the agents rely on the virtual environment tracking hardware to monitor the student's position and orientation in the environment.

Steve's learning environment can be simultaneously inhabited by multiple students, each of whom play the role of a different crew member aboard a simulated ship. Steve agents may be used to assist individual team members, or play the role of missing team members. This requires having each Steve agent understand how the roles of the various team members interact and depend upon each other. Steve is an example of an animated agent; this kind of agent is not dealt with in this thesis.

People Power is a system with a co-learner (computerised learner) which takes the role of a collaborator (Dillenbourg and Self, 1992). This system investigates how the co-learner and the students co-generate reflective social dialogue through argumentation or negotiation.

The previous examples used simulated students in order to help students to solve problems, or investigate social collaboration between a computer and a human, in the case of later systems, but none of them deals with the topic of avoiding negative learning situations. All previous systems, except Steve, are designed for only one user: none of the systems deals with the issue of how a companion can collaborate with two or more students. More recent work is researching how avoid some of the negative situations and how to use a companion in collaborative environments; an example is the EPSILON project.
Chapter 2: Collaborative Learning and Computer Supported Collaborative Learning

The EPSILON (Encouraging Positive Social Interaction while Learning ON-Line) project is an interdisciplinary initiative to develop a smart software agent that can intelligently and adaptively provide pedagogical support to students learning collaboratively on-line (Soller, Cho and Lesgold, 2000). The EPSILON project is currently investigating one of the previously explained negative situations that they call "a failure knowledge sharing episode". This is defined as a segment of interaction during which one team member has not shared new knowledge with the group. The episode is considered effective if one or more students learn the new knowledge and ineffective otherwise (Soller and Lesgold, 2000). An example that clarifies this definition is:

- John has completed a critical portion of the task that is new to his teammates.
- John has not discussed or explained his actions.
- John’s teammates have not questioned or asked for explanations of his actions.

As a result, the agent might intervene, asking the group to explain John’s actions. In this way the agent detects if other group members understood the action.

Another system that attempts to correct negative situations is presented by Okamoto and his colleague (Okamoto, 1996; Inaba and Okamoto, 1997). They describe a system that can diagnose and coordinate flows of discussion through the students’ conversations.

The supporting functions of the system are as follows:
1. To make users participate actively in the discussion.
2. To facilitate group discussion smoothly in the desired direction (such as finding an adequate solution to a problem).

The system has a chat area where students discuss different solutions to the problems. Each time a student writes a sentence s/he has to say what type of sentence it is and select the appropriate button in the intention menu (there are ten categories). The system can diagnose four states where it should intervene. The states are as follows:
Chapter 2: Collaborative Learning and Computer Supported Collaborative Learning

1. The divergent state of discussion.
2. The impasse state of discussion.
3. The state in which a participant gives little utterance for a while.
4. The state in which a question is not solved.

When the system decides to give the participants some advice, it tries to classify all the participants’ roles into the following categories:

- The participant who leads the discussion process.
- The participant who has the initiative in the current topic.
- The participant who understands the contents of discussion well.
- The participant who does not understand the contents of discussion.
- The participant who often asks questions.
- The participant who gives little utterance.
- The participant who offers propositions with which the other participants do not agree.
- The participant who does not belong to the above categories.

Then, the system gives appropriate advice to each participant depending on his role in the discussion.

The system has two workspaces. One is the Personal Work Space, which is a closed private area and cannot be seen by others. The participant can use several tools for problem solving in this space; e.g., a word processor, a drawing tool, and an ITS. Another space is the Collaborative Work Space shared by all participants.

This system is not focused on a specific domain. It is a general application which can be used with any tutor. The authors do not present experiments that show whether the application helps students to learn more or faster; they centre their work mainly on productive discussion between students but they do not study the effects of a productive discussion in a collaborative learning environment. Unfortunately the application needs the students to indicate explicitly what type of intervention they are making. So, without the
students' help, the system cannot detect what role is played by each student, and if the student describes an intervention incorrectly the system cannot detect it. On the other hand, if the goal of the system is to help students to solve problems it can be counter-productive to obligate students to think about what type of intention their utterance has. This can distract the students' attention and reduce their performance. We think that besides the conversations the systems could also use other mechanisms such as frequency of intervention, type of solutions, frequency of success, in order to detect the students' role.

A similar application is presented by Inaba (2000). She describes a collaborative learning support system that detects an appropriate situation for a learner to join in a collaborative learning session. To fulfil this objective, she has considered the following points:

1. How to detect the appropriate situation to start a collaborative learning session and to set up the learning goal.
2. How to form an effective group which ensures educational benefits to the members of the group, and
3. How to facilitate the desired interaction among learners in the learning group.

There are many factors that characterise a collaborative learning session. When should the learners start a collaborative learning session? What subject matter should they learn in the session? How should the session progress? Who should join in the session? What educational benefit should be expected for each learner through the session?

They chose concepts to represent a collaborative learning session through a survey of learning theories and studies on collaborative learning. As a result, they set up five primitive concepts to characterise a session:

- **Trigger** means the desired situation for each learner to start a collaborative learning session.
- **Learning material** means the subject matter.
- **Learning scenario** means how the collaborative learning session progresses.
- **Learning group** means a group type and each learner's role in the session.
Chapter 2: Collaborative Learning and Computer Supported Collaborative Learning

- Learning goals can be specified as two kinds of goals: “common goal” as a whole group and “personal goal” for each learner.

To form a learning group means to choose learners who join the group as members and to assign a specific role in the group to each member. They classify members into two kinds of role-holders: the members who play Principal Role (PR-members) and the members who play Secondary Role (SR-members). Each role is defined as follows:

PR: A PR-member is expected to gain the main educational benefit through the session. The PR role is usually played by the learner who first proposes to engage in collaborative learning.

SR: A supporting role for the PR. A SR-member helps the PR-member attain his/her I-goal. So PR is Learning by Teaching and the SR member is expected to acquire new knowledge by being taught by the member who plays the Peer Tutor.

The system has different agents to support collaborative learning dynamically. When the program detects the situation for a learner to shift from individual learning mode to collaborative learning mode, it forms a learning group each of whose members is assigned a reasonable learning goal and a social role which are consistent with the goal for the whole group.

Each agent should have the ability to realise the following functions:

1. Setting up an appropriate learning goal for a learner.
2. Forming a learning group to enable the learner to attain the learning goal, and negotiating with other agents to reach an agreement: the formation of a collaborative learning group within which each member of the group can obtain educational benefit.

This system does not control if, when the second member joins the collaboration session, the second member plays his role correctly. Although it may be convenient for the first student to work in collaboration, this may not be the case for the second student. The
system should also check whether the negative situations take place when students are working in the group.

Constantino-González has developed COLER (Constantino-González and Suthers, 2001). This is a Web-based collaborative learning environment in which students can solve database-modelling problems while working synchronously in small groups at a distance. Her work seeks to facilitate effective collaborative learning interactions. A coach has been implemented as a personal assistant to each client. The coaches are pedagogical agents that encourage students to discuss and participate in collaborative problem solving. The personal coach views the student's private workspace and the shared workspace in order to help to prevent missed opportunities for collaborative learning. The knowledge for coaching collaboration consists of the ability to recognise relevant learning opportunities and to provide advice that encourages students to exploit these opportunities.

An evaluation of the students' reaction to the coach's advice was carried out. Findings indicate that 40% of the total advice was taken by the students, 28% was applicable but ignored and 21% was no longer needed due to changes in the situation. Student response to 11% of the advice could not be determined. Questionnaires were administered after the collaborative problem-solving phase. Students suggested additional types of messages and indicated that sometimes advice should be given to the whole group instead of just to individuals. Perhaps using a global coach which gives public advice, instead of a coach per student would have avoided this problem. In fact, the authors are investigating the development of a global coach with the ability to inspect all students' private workspaces as well as the shared workspaces. This would improve the system since it would avoid students having two conversations in the chat space, one, private, with the coach and another, public, with the teammates, which could become confusing for them. On the other hand, having a personal coach could decrease the students' freedom and perhaps even their collaboration because a student might give priority to the coach's advice instead of the teammate's advice.
Chapter 2: Collaborative Learning and Computer Supported Collaborative Learning

2.4 Summary

This chapter introduced collaborative learning, and briefly summarised the educational techniques which have influenced this type of learning. It went on to explain how a few decades ago computers were used by groups of students for reasons of economy and scarcity of resources, but that the present reasons for which they are used by groups are pedagogic ones. New technology, and more concretely CSCL technology helps students to learn in groups. The characteristics, advantages and challenges of CSCL have been analysed. Finally, the chapter explained how agents and simulated students have become a new component of collaborative applications. Several advantages of using simulated students in collaborative systems have been described.

We propose to take advantage of the benefits of using a simulated student in a collaborative system, in order to avoid some challenges that working in a group brings. This thesis is focused on three specific problems or challenges. There are two reasons why we chose these three problems:

- The experiments carried out in our research group (see Vizcaíno et al., 2000; Vizcaíno and Prieto, 2000) in order to detect problems in collaborative learning, and the review of the CSCL literature (e.g. Eichinger and Anderson, 1991; Soller et al., 1998; Sipusic et al., 1999; Nishimori et al., 2000) indicate that these three situations arise with great frequency.
- There is not enough research about how these situations could be avoided or corrected using CSCL systems.

The three negative situations are described in detail in section 2.24 and here are cited briefly,

1. Students sometimes, waste time talking about topics not related to the problem in hand. This makes other group members feel that they are wasting their time.
2. Passive students are students who for different reasons do not take a full part in solving the problem. This is unhelpful for the passive students themselves and for the rest of the
group. It is prejudicial for the student because s/he does not learn and it is unjust for the rest of the group because they may feel they are working for the other person without obtaining anything in return.

3. Lost situation. When after a time students do not find a solution to the problem they can lose motivation and sometimes abandon the problem.

We have developed a collaborative application with a simulated, though not animated, student, which by using different techniques, not just the conversations, obtains information about the roles played by the student and tries to detect the three above mentioned situations. When it detects a negative situation it attempts to correct it and avoid its effects. Our system extends previous work in that:

- The simulated student uses different techniques to diagnose the role of each student such as measuring the frequency of participation of each student, checking the solution proposed by each student and checking conversational content against keywords. Earlier systems used only the conversations as a method of obtaining information.
- The system enables free communication between students, whereas most current systems enable only structured conversations, which slow the communication process and misinterpret students' dialogue when students use the interface buttons incorrectly (Constantino-González and Suthers, 2000).
- The system focuses on developing skills. Traditional systems teach basically declarative knowledge.
- The system has been evaluated in order to check whether the simulated student plays its role correctly and whether the students learn more, or faster when collaborating with the simulated student. There is no similar evaluation of the earlier systems.

In order to test the efficiency of the simulated student we had to choose a particular domain to teach. Because our system focuses on developing skills, the domain should be procedural where students could solve problems. The next chapter explains why programming was the domain chosen for the collaborative application.
Chapter 3: Learning and Teaching Programming

The goal of this chapter is to explain why programming is a discipline that often presents problems in its learning. Program comprehension and debugging are activities involved in learning programming and both activities need the development of new skills. However these tasks are seldom taught explicitly. This chapter studies the difficulties that students have in debugging and program comprehension and how both can be learnt. New technology can be used to help students to learn programming or, more concretely, to develop the skills necessary to become a good programmer: several applications have been developing with this goal, nine of them are analysed in this chapter.

3.1 Difficulties to Teach and to Learn Programming

Programming is a discipline difficult to teach and to learn. For example, last year of the 339 students who were registered in the subject of programming in our university, only 131 students went to the exam; the rest gave up the discipline. And only 90 passed. The elevated index of failure in this subject was one of the reasons why we developed a system to help students to learn programming. The following paragraphs explain why programming a discipline complex to teach and to learn.

Learning to program in any language is not an easy task and programming teachers are well aware of the myriad difficulties that beset beginners (du Boulay, 1986; Rogalski and Samurçay, 1990). However programming is also a subject difficult to teach. In programming there are many abstract concepts like recursion or data structures, and it is not easy task to find clear examples to demonstrate some concepts. Good pedagogy suggests that teaching should use examples from real life to help the students remember the examples and concepts more easily. In programming is very difficult to explain same abstract concepts with examples based on real life.

Programming differs from related domains such as mathematics or physics in two ways. First, there are no everyday intellectual activities that can form the basis for spontaneous construction of mental models of programming concepts such as recursion or variables. Second, programming activity operates on a physical machine which may not be transparent in its functioning for learners (Rogalski and Samurçay, 1990).
Programming is also a complex skill to learn where even languages designed for the novice such as Basic and Pascal contain many traps for the unwary (du Boulay, 1986). This discipline is complex to learn for two main reasons:

- Programming involves a great variety of subtasks and types of specialised knowledge that are necessary to perform effectively (Pennington and Grabowski, 1990).
- Programming is a procedural discipline which must be "learning by doing".

The next sections explain these two issues in more detail.

3.1.1 Subtasks of Programming

Computer programming may be characterised as a design of tasks. A skilled programmer must comprehend the problem to be solved by the program, design an algorithm to solve the problem, code the algorithm into a conventional programming language, test the program and make modifications in the program once it is completed. Success at the programming tasks requires knowledge of the external problem domain (e.g. statistics, finance, communications), knowledge of design strategies to develop and implement algorithms, knowledge of programming languages, knowledge of computer hardware features that affect software implementation, and knowledge of the manner in which the program will be used. In Figure 2 are described the tasks of computer programming: understanding the problem, design, coding and maintenance. With each subtask Pennington and Grabowski (1990) have associated certain mental products and knowledge domains.
### Figure 2: The tasks of programming (adapted from Pennington and Grabowski, 1990, pg 47)

<table>
<thead>
<tr>
<th>Programming subtasks</th>
<th>Knowledge domains</th>
<th>Mental representations</th>
<th>External representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding the problem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain knowledge (e.g. statistic, banking)</td>
<td>Situation model</td>
<td>Requirements document Specifications document</td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design strategies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming Algorithms and methods</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design language</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solution model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plan representation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design document</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coding</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming language</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming conventions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program Representation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintenance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All knowledge domains</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Debugging, testing strategies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequent kinds of error</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All representations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All documents</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The programming subtasks described above have multiple interconnections that make them difficult to separate in practice.

The crucial dimensions in the activity of programming are processing and representation. There are two ways individuals can move from a real-word problem to program text implementable on a given device. A real-world problem can first be solved in the domain and then translated into program text. Or, alternatively, it can be approached in the programming language and applied to the real-world object. When problem solving takes place on a real-world object, processing precedes representation, even if properties of the objects in the target programming language intervene in the choice of solution. This approach is closely related to the 'computational' programming paradigm where a program is defined as a succession of computations. The 'functional' perspective where the program is seen as a function that needs to be decomposed is more infrequent. When problem solving initiates in the programming text, the structuring of data and relationships between pieces of data is the core of programming activity.
Chapter 3: Learning and Teaching Programming

Studies on the acquisition of programming skills, have mainly centred on the processing dimension. There are at least three reasons for this: 1) the historical role of procedural languages 2) the importance of planning in programming design (Hoc, 1988) and 3) the productive role of the organisation of actions as a first programming model. In addition, problems given to novices are often directly defined in terms of programming entities, such as numerical data and variables in the well-known sum problems or explicitly in terms of a given programming language, such as: 'define a function called list-sum. Given a list of numbers, list-sum returns the sum of these numbers'.

3.1.2 Procedural learning

In a traditional classroom where the teacher explained a topic and students had to remember it, learning was largely a transference of information and did not help students learn how to study procedural issues.

Procedural disciplines are those that involve both practical and theoretical parts. Procedural learning requires theoretical learning but also the application of practical or operative activities (Castañeda, 1995). An example of a procedural discipline is mathematics where students have to learn through problem solving exercises. On the other hand a clearer example of declarative discipline is history. Student are able to learn history without necessity of attending classes or practising the ideas learnt, thought another view of learning history is that it comprises precisely those skills involved in being a historian.

In programming, students need to understand important concepts such as recursion or data structures. However the most important thing is that students know how to use these concepts in practice. So students need to develop new skills and habits and when students are not used to learning by practising, programming can became a very difficult discipline.

3.1.3 Students Difficulties in Procedural Aspects of Programming

The previous subsections outlined the general problems of teaching and learning programming. This section presents studies about specific procedural problems that learners had when they were learning programming.
Perkins and Martin (1986) conducted a series of interviews with students using Basic. They describe how the main difficulties were 'fragile knowledge' and 'neglected strategies'. The former is knowledge that the student has, but fails to use, when it is needed. Evidence that the student possessed the knowledge is not simply the fact that they had been taught it, but that on nearly 50% of occasions where hints had to be given to students, the student went on to solve the problem, even though the hint did not contain the appropriate knowledge.

A component of the difficulty seems to be that the students were not reading the program to discover what it actually did ('close tracking' or 'parsing'). This is an example of a 'neglected strategy', a general problem-solving strategy that should be, but is not, used. Mistaken strategies also exist, as can be seen in one of their students who appeared to be obsessed with using syntactic features taught recently, even when the problem only required simple structures taught and mastered some weeks before (Gilmore, 1990).

Perkins and Martin (1986, p225) summarise their novices' problem as 'fragile knowledge exacerbated by a shortfall in elementary problem-solving strategies', suggesting by way of conclusion that we should not view programming as an opportunity for the development of general problem-solving skills, since they are themselves required for successful programming. Although an appropriate teaching of programming could develop new skills or develop skills that students already have.

White (1988) conducted a study with Prolog programmers, some of whom had previously learnt Pascal. Although it is not surprising that interference effects occurred, what is interesting is that Prolog novices were able to use the appropriate terminology for Prolog at the same as using an inappropriate strategy from Pascal.

Bonar and Cunningham (1988) in describing their use of the intelligent tutoring system Bridge, comment on how their students were quite successful at developing an outline solution using plan-based concepts, suggesting that they had acquired and understood the contents of the programming plans. However, the bottleneck for the students came when trying to translate the plan-based outline into actual Pascal code.

Thus, it seems that having knowledge of the plans alone is inadequate, it is understanding how to use them that counts. Possessing knowledge is not the only problem that novice
Chapter 3: Learning and Teaching Programming

Programmers have. In a number of cases they show that they have the knowledge, but also that they do no know how to use it adequately. It seems that programming may be rather like riding a bicycle in that without practice it cannot be mastered (Gilmore, 1990).

3.2 Psychology of Programming

In the previous sections some aspects of the difficulties to teach and to learn programming have been outlined. We could see that different programming processes are related to cognitive process. The field of the Psychology of Programming was developed in order to study the cognitive processes that programmers, in general (novices, intermediates or experts) have, partly in order to propose strategies that programmers can use to improve their efficiency. Although the psychology of programming is not the focus of this thesis, it has a bearing on the domain chosen for the collaborative system we have investigated.

The psychology of programming studies strategies that help programmers to program in a safe way, for instance that, debugging programs takes considerable time. There are always errors and it is a painstaking task to track them down and correct them. Programmers have all experienced this: even if they have not found solutions as such to the problem, they have nevertheless worked out ways of playing it safe. The psychology of programming studies these strategies.

The history of the psychology of programming dates back to the 1960s, probably to a French work which analysed business programmers' behaviour in terms of top-down and data-oriented programming methodology. The broad range of individual differences in the structuring of programs designed to solve the same problem led to the conclusion that a programming methodology was needed for programmers in order to teach them how to use high-level representations of information and control flow to correct for the saliency effects of low-level machine constraints. This conclusion is consistent with more recent studies of planning which indicate that programmers need support to process schematic representations before implementing them in concrete devices (Hoc, 1988).
Chapter 3: Learning and Teaching Programming

The decade of the 60s was the first generation of programming. In this decade most studies were produced by computer scientists who developed a normative approach to what they considered to be the most powerful programming concepts. During the 1970s, the second generation of programming studies was dominated by a number of empirical studies, comparing diverse types of languages. The third generation was initiated in the early 1980s by a wide-ranging debate on the theoretical and methodological grounds of the psychology of programming. Pioneering efforts were followed by an increase in the number of psychologists studying programming, especially notation and debugging aspects (Green, 1980). At the same time, some computer scientists in the cognitive science field were developing cognitive approaches to programming.

For some it may seem strange that a relationship between cognition and programming exists. Ormerod (1990) explains three ways in what cognition and programming are related. First, cognitive psychology is based on a 'computational metaphor', in which the human mind is seen as a kind of information processor similar to computer. Second, cognitive psychology offers methods for examining the processes underlying performance in computing tasks. Third, programming is a well-defined task, and there are an increasing number of programmers, which makes it an ideal task in which to study cognitive processes in a real-world domain.

3.3 Programming Activities

Programming includes a wide range of activities: analysis and design, coding, program comprehension, debugging, maintenance and documenting. Although all them are very important this section focuses mainly on program comprehension and debugging. Both of them need training and present particular problems for novice students. It is interesting to note that although a larger part of the professional activity of a computer programmer consists of debugging programs, in programming courses debugging is not often treated as a skill that needs to be taught. Generally the novice has to acquire the skill to correct bugs on his own (Kessler and Anderson, 1986). Program comprehension is another topic that, although it involves important activities such as identifying important program parts and
inferring relationships between them, is seldom discussed explicitly with the novice programmers.

3.3.1 Analysis and Design

Analysis and design are the first two stages of software development. Analysis consists of outlining the task that the program will have to perform by defining all its requirements. Design consists of finding a program structure that can actually meet these requirements, and in specifying how the program will work with a sufficient level of detail for the implementation to take place. In theory, a design can be run (and its validity tested) on pen and paper.

Not much research has been conducted on Analysis. By contrast, many studies have been conducted on Design, which all make the distinction between procedural languages and object-oriented languages.

Procedural Languages

Many language paradigms exist, such as logic, functional, procedural or object oriented languages, but most research has focused on procedural languages. In a procedural language, the programmer specifies a list of instructions s/he wants the computer to execute. These instructions are put into separate groups called functions and/or procedures, and a function can ask another function to be executed.

Lee and Pennington (1994) listed three types of design strategy for procedural languages:

- Stepwise refinement (Wirth, 1974; Adelson & Soloway, 1985) consists of splitting the main problem into sub-problems, and again, in a top-down breadth-first manner.
- Kant and Newell (1984) described a similar model called 'Incremental design': a programmer very quickly builds a schematic solution, which is then developed using 'successive refinement', and tested by symbolic execution and/or test-case execution.
- Opportunistic design (Green, 1980) occurs when the designer switches back and forth between different levels of abstraction, in an apparently unorganised, opportunistic way.
Rist (1996) proposed a more general view of design. In his opinion, design can be seen as a search process; that is, it consists in 'choosing a place to start, and searching from that location along a specific type of link'. The nature of the design will depend on which kind of starting position and type of link the designer uses. Once a terminal position is reached, the designer starts from another position, and so on until s/he has followed a 'complete trajectory through system structure'.

Object-oriented languages

In an object-oriented (OO) language such as Java, C++, or Eiffel, a programmer specifies 'classes' which will be instantiated during program execution into 'objects'. These classes are supposed to model a real word situation or a concept. For example, in a snooker simulation, the programmer may create three classes (snooker table, ball, and cue), but the program will be made of 24 objects (one table, one cue and 22 balls). Each class specifies what characteristics the objects have and the actions that they can perform, their methods. A method can refer to its own class methods and variables, and to other classes as well.

Furthermore, object-oriented languages introduced the concept of inheritance. When a class A inherits from another class B, A will have all the variables and methods of B plus some others. Each object-oriented language has further refinements on the concept of inheritance, such as abstractness, multiple inheritance, public/private methods and variables, and so on.

Object-oriented designs differ in many ways from procedural language designs. As Gibson (1990) underlines, they describe the environment rather than the goals or functions of the program, by building hierarchies of classes and objects. Rosson and Alpert (1990) found that programmers tend to come up with more consistent designs for object-oriented languages that for procedural languages. Lee and Pennington (1994) and Kim and Lerch (1992) both found evidence that designing with an object-oriented language is quicker than with a procedural language, even if the designer has to spend more time analysing the problem.
Chapter 3: Learning and Teaching Programming

As far as novice OO designers are concerned, creating classes seems to be an important problem (Pennington, Lee, and Rehder, 1995). Novices first spend a lot of time hesitating about which classes they should create, before defining the methods and considering the goals of the classes. Moreover, they have difficulty in 'associating objects with procedures' (Détienne, 1997), i.e. mixing object-oriented concepts and procedural concepts which are both used in object-oriented languages.

3.3.2 Coding

Coding used to be considered as the most important part of software development, but with higher-level languages and bigger programs, design is now seen as the crucial, 'smartest' part of the development process. Coding and design were seen as completely separate activities. In fact, they are deeply intertwined. On the one hand, a designer may think that his design is sufficiently detailed if it can be fairly simply translated into code. On the other hand, implementing a design in actual code will raise new issues and lead to modifications in the design. Gray and Anderson (1987) and Visser (1987) found that programmers alternate constantly between writing and evaluating their code, and evaluation sometimes leads to changes in the design.

Object-oriented languages were said to be more 'natural' from a design and programming point of view. This claim was rejected by Davies, Gilmore, and Green (1995), who found that expert programmers classify classes not according to object-oriented aspects such as class relationship, but according to functional aspects. Object-oriented languages still seem more natural for some aspects of design as the construction of a simulation model of a real world situation, and for coding.

Many studies compared expert programmers and beginners. Batra and Davis (1992) found that experts build more complete representations of the problem before creating a solution. They also use more discourse rules (Davies, 1990) and programming plans, but also organise them differently (Davies, 1995). They can produce alternative solutions before choosing the best one.
Chapter 3: Learning and Teaching Programming

3.3.3 Program Comprehension

Program comprehension is an important part of computer programming skill, both from a practical and a theoretical perspective. It is a complex cognitive skill, which involves the acquisition of a mental representation of program structure and function.

From a theoretical viewpoint, comprehension involves the assignment of meaning to a particular program, an accomplishment that requires the extensive application of specialised knowledge (Navarro-Prieto, Cañas and Bajo, 1996).

From a practical viewpoint, the ability to understand programs written by others is an important component of a programmer's skill set. During program comprehension the programmer finds her/himself performing programming tasks such as debugging, modification and in general code reuse. Pennington (1987) estimated that more than 50% of all professional programmer time is spent on program maintenance tasks that involve modification and updates of previously written programs. Because other programmers most often write the programs, comprehension plays a central role in this endeavour.

3.3.3.1 How are Programs Comprehended?

Early research on program comprehension was based mainly on the so-called Plan Theories of programming and consisted of studying the stereotypical knowledge stored in long term (plans, schemas) which programmers would use when they are trying to understand a program. The main assumption of plan-based theories was that the development of expertise is associated with the gradual accumulation of plan knowledge over time. The first approaches, like Shneiderman (1976) and Soloway and Ehrlich (1984), were based on Schema theories (Schank and Abelson, 1977) and the studies of chess expertise (DeGroot, 1965). These models proposed that program understanding is a top-down process in which experts access hierarchically organised plan knowledge. Therefore when programmers are trying to understand a program they first look for the plan that best matches this program. Once the programmer has chosen the plan that corresponds to the program s/he is reading, this plan guides the rest of the program comprehension process. On the other hand, Rist (1996) proposed a more general model in which a top-down process is combined with a bottom-up process.
In general, the studies conducted by the proponents of these theories supported the idea that expert possess structured Long Term Memory (LTM) knowledge. However, later studies failed to reproduce the earlier results or to prove the predictions from the theories (Widowsky and Eyferth, 1986; Vessey and Weber, 1984), suggesting that the existence of plans is not sufficient to explain the complex task of program understanding. In particular, studies in reading behaviour such as that of Robertson, Davis, Okabe, Fitz-Randolf (1990) pointed out that eye movements through the code give support to Rist's model of non linear understanding, but also showed evidence of strategic search. Also, some empirical results showed differences between languages in terms of the types of errors committed by the subjects (Gilmore and Green, 1988) which show the necessity of taking language variables into account.

3.3.3.2 Programming Comprehension Models

There are different strategies to understand a program. The next paragraphs present the main model of program understanding.

**Hypothesis testing:** Brooks (1983) described a model of program understanding based on hypothesis testing. To understand a program, the programmer makes a global hypothesis about what the program does. Then he splits this hypothesis into sub-hypotheses, and so on, until the hypotheses attain a level of detail which make them comparable to the documentation available, for instance code, text, etc. A low-level hypothesis can thus be confirmed or disconfirmed by the documentation, in which case this hypothesis and its neighbours will have to be modified.

**Top-down model:** this model was described by (Soloway, Ehrlich and Bonar, 1982) and Soloway and Ehrlich (1984). It is quite similar to the model of Brooks, except that it is based on plans. They identify three types of programming plans: strategic plans (global methods used by the program to achieve a goal), tactical plans (locals methods used to achieve a local goal) and implementation plans (language dependant implementations of a tactical plan). Here the focus is on goals and plans since the internal representation of the program is a decomposition of the overall goal into different plans, which are themselves
local goals that are achieved by other sub-plans, and so on. They insist on the role of plans and discourse rules in program comprehension.

**Bottom-up model:** Pennington (1987) explained that schemas play a role in program comprehension and explanation, but that they are not the memory structure. The role of schemas in internal representations of programs has not been, in her opinion, proven. Her model of program comprehension is based on the building of two internal models: the program model (which is a control-flow model abstracted from the code by spotting plans and independent chunks) and the situation model (which is a data-flow abstraction based on hypotheses which are checked against the program model). The programmer begins to build the program model by analysing the code (bottom level). Then he can start building the situation model, a model which requires some domain knowledge. It is completed once the top-level hypothesis, that is the overall program goal, is reached. Shaft and Vessey (1996) observed programmers trying to comprehend familiar and unfamiliar programs. They found that some programmers used a top-down methodology in both cases, others a bottom-up methodology, while others again used a top-down strategy for a familiar program and a bottom-up strategy for an unfamiliar one.

**Integrated meta-model:** von Mayrhofer and Vans (1995) built a theory which integrates the top-down model, the program model, and the situation model. Its fourth basic component is the knowledge base of the programmer.

All these models are focused on code understanding, without taking into account design understanding, and nothing is said about the use of different types of documentation. It seems that program understanding depends on the subject's task and on his knowledge of the domain and of the program itself.

Program comprehension is a very important skill that novice students must develop. Program comprehension means that students know the syntax of a language (if the program is written in a specific language), understand the semantics and use the previous mentioned
strategies for comprehending the whole meaning. The system that will be presented in the next chapter tries to develop program comprehension skills in novice students.

3. 3.4 Debugging

Novice programmers often have misconceptions about programming language syntax and semantics and they lack the expert's knowledge about how to analyse program specifications and design and implement algorithms. Because of this novice programmers introduce many different bugs into their programs.

Johnson (1985) describes an experiment where a Pascal compiler, that students used in an introductory programming course, was modified. When students attempted to compile a program a copy of their program was saved on tape in order to analyse the number, and types of bugs. It was found that in programs with a mean number of 44 source lines, the mean number of bugs per program was 3.81. This fact shows that the range of frequency of bugs in novices programs is quite high. Students need to debug their program in order to find and correct the mistakes. Debugging is an important task that students must learn to perform at the same time that they learn to program.

Debugging is an even more difficult task when the program to be debugger is written by someone else. Programmers often report that instead of debugging someone else's program, they would rather write their own version (Katz and Anderson, 1988). The task of debugging another person's code is more difficult than the combined tasks of writing and debugging one's own program. And different research shown that programmers use a different debugging strategy when they are debugging their own program than when they are debugging someone else's programs.
Debugging is a particular instance of a very general problem-solving process namely troubleshooting. (Katz and Anderson, 1988). Figure 3 provides a high-level model of what is involved in troubleshooting.

![Flowchart diagram](image)

**Figure 3.** Simplified model of general troubleshooting. (Adapted from Katz and Anderson, 1988, pp 353).

Generally, a person must first come to understand or have a representation of the device being repaired. The person then usually tests the device in some way, if only to observe that incorrect behaviour is produced. The person must then locate the error in some way and, once the error is found, repair it. After allegedly repairing the device, the person should test the device to be assured that correct behaviour is now produced. If the device still acts incorrectly, further location and repair may be needed. Thus, according to this model, troubleshooting contains four steps (understanding, testing, location, and repair), repeating
the last three steps if necessary. A similar characterisation of troubleshooting has been suggested by Morris and Rouse (1985).

It is clear that this model is an oversimplification of troubleshooting. The model neither addresses the issue of how the processes interact nor how the processes are instantiated in a particular domain or in a particular situation.

Debugging may be thought of as just a specific instance of troubleshooting, but instead of fixing the problems in a device, the errors in a computer program are sought and corrected. Other researchers (Gugerty and Olson, 1986, Kessler and Anderson, 1986) have shown that debugging may be characterised as consisting of subprocesses that correspond to the model of general troubleshooting just presented.

When debugging a programmer must a) test the program and detect that it is not behaving incorrectly, b) locate the erroneous line or line of code, and c) rewrite the buggy code. Also, if a person had not written the program originally, or had written it a long time ago, that person would probably need to first comprehend or understand the program.

3.3.4.1 Types of Bugs

The bugs made by novice students can be of very different kinds and have been categorised by different researchers in different ways. For example Katz and Anderson (1988) divide the types of LISP bug into five categories.

- a) Goal errors, which show up as missing pieces of code,
- b) misrepresentation errors, which show up as a misunderstanding about the problem statement or confusion over the roles of variables,
- c) intrusion errors, which show up as pieces of code that would have worked in a previous program, but are inappropriate in the current program;
- d) misconceptions, which reflect a misunderstanding about an aspect of LISP, and
- e) syntactic errors.
Bental (1995) performed a study about the queries that students ask a human tutor when their programs did not work correctly. Students used a functional language similar to Lisp, called SML. A total of 134 messages were received and they were categorised based on Wertz' categorisation of bugs in novice LISP programs (Wertz 1982). Wertz applied this classification to a range of novice exercises and divided bugs into lexical, syntactic, semantic, teleological and conceptual bugs. The following list shows Bental's results.

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax of SML</td>
<td>18</td>
</tr>
<tr>
<td>Semantics of SML</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Recursion</td>
<td>19</td>
</tr>
<tr>
<td>Functions as values</td>
<td>13</td>
</tr>
<tr>
<td>Data types</td>
<td>9</td>
</tr>
<tr>
<td>Pattern Matching</td>
<td>5</td>
</tr>
<tr>
<td>Others</td>
<td>7</td>
</tr>
<tr>
<td>Type Signatures</td>
<td>12</td>
</tr>
<tr>
<td>Style</td>
<td>18</td>
</tr>
<tr>
<td>Debugging</td>
<td>13</td>
</tr>
<tr>
<td>Interpreting the Question</td>
<td>15</td>
</tr>
<tr>
<td>General Incomprehension of SLM</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>134</td>
</tr>
</tbody>
</table>

Bental counts lexical (typographical) errors among the syntactic bugs because the majority of these errors cause syntax problems. Semantic and teleological bugs are counted together because the distinction between the two varies for different languages. Some core features of SML semantics (such as data types) might be considered teleological in a language such as LISP.

Style and type signatures were added as categories because students asked specific questions about them.

Interpreting the question is related to Wertz's category of conceptual bugs. Spohrer and Soloway (1986) note that interpretation is a frequent source of difficulties for novices.
In Bentall's study many students specifically asked for help to interpret the error messages given by the SML compiler and to locate bugs. Debugging and error location are essential skills, since few people write correct programs first time and novices less so than experts. Many students had little idea how to go about locating errors.

Bentall's classification concentrated on basic programming skills rather than another aspects of software design. The kinds of questions that students ask suggest that automated analysis of novice programs will become an increasingly important part of support for the teaching of computer programming. Students require help with the syntax and the semantics of the programming language, with programming style, and with the requirements of the task itself. They also need help to develop debugging strategies (Bental, 1995).

"Many recent systems have focused on the problem of diagnosing student errors in programming. Few have concentrated explicitly on helping novices learn to debug. While the systems aim at supporting novices, they generally avoid issues connected with teaching the novice how to debug" (Brna, Hernandez and Pain, 1999, pp 383).

The system presented in this thesis tries to train novice students to debug programs written by other people. The system shows students different programs with typical mistakes, so students have to find the mistake, and to correct it.

3.3.4.2 Locating Bugs

When programmers look for a bug, they seem to work from a mental representation of the program as well as from the listing of the program itself. This mental representation is probably built during coding and is maintained by the salience of the actual, written code. The representation appears to contain information regarding the subjects' interactions in writing a line or whole section of code as well as presuppositions concerning the code in general.

In an experiment with novice Lisp programmers Katz and Anderson (1988) found three general strategies that the students used to locate a bug: a) a simple mapping from the program's behaviour to debug, b) hand-simulation, and c) causal reasoning.
The simple mapping strategy means that the program’s buggy behaviour pointed to the bug that produced this behaviour. A typical example of this strategy is when a subject saw the error message “ERROR: eval: undefined functions course”. The student wondered aloud why the computer though that “course” was a function (course was a variable name), and started looking at instances of the word course in the program. This strategy may become more difficult to use when a program has a large number of occurrences of a particular variable.

Hand-simulation consists of the student executing the program as the computer would and looking for inconsistencies between what occurs in the actual function and what is expected as per the subject’s representation of the function. Jeffries (1981, 1982) observed this strategy being used by both expert and novice PASCAL programmers.

The causal reasoning strategy involves looking at the information obtained in the testing of the function (e.g., the program’s output or the results of tracing a subfunction) and reasoning about what might be causing the bug. It is the students’ representation of the program, as well as general programming knowledge, that seems to guide the search for the bug, as opposed to hand simulation in which the actual, written code guide the search.

Programmers produce a wide range of behaviours when debugging their programs. From this observation it is clear that debugging is not a single activity, but a set of activities, each component of which may be performed differently depending on the situation.

Stacey (1995) suggests that a tip for finding errors is to check for common errors. The problem is that novice programmers do not know the most frequent errors. In the system that we have developed we have tried to present the most common bugs so students can familiarise themselves with them, and learn how correct and avoid them. Another tip is to talk to someone else about the problem. Although all programmers are used to debug their program using this tactic, it is interesting that most systems developed to teach programming or debug programs are designed for individual students and do not enable
communication with other people. In section 3.5 of this chapter several systems are studied, and this topic is dealt with in more detail.

When programmers are looking for a bug two psychological effects can arise that prevent programmers finding the bugs. These effects are Psychological Set and Psychological Distance.

Psychological Set is the phenomenon of seeing what you expect to see. Some features of psychological set are:

- Students learning "while loops" often expect a loop to be continuously evaluated, not just at the top or bottom. Therefore, they expect that the loop will terminate as soon as the while condition becomes false. They confuse the while loop with the natural language while.
- Errors in assignment statements are about three times as hard to find as errors in arrays or more complex interactions. Simple errors are often overlooked.
- It is hard to see some errors, for instance, a programmer used the variables SYSTSTS and SYSSTSTS without realising that he had misspelt the second one, they were both supposed to be the same variable.

A way to combat psychological set is by using good programming practice or also called good programming style. Examples of good programming style are

- Formatting the program. This helps to read the program and understand it.
- Commenting. To add comments helps to understand the program and to reflect about what the program does.
- Variables Names. Choosing sensible names of variables can avoid psychological set effect and also avoid the problems of the psychology distance explained in the next paragraph.
- Routine Names. Choosing names that describe the functionality of the routine helps understand the program.
Research has shown that good programmers mentally slice away parts of the program that are not relevant during debugging. This narrows the search space and finds the errors more quickly, but, may lead to the ignoring of the relevant part of the code.

Another problem that often arises when programmers are debugging is Psychological Distance. Psychological Distance is the ease with which two items can be differentiated. Examples of different distances:

<table>
<thead>
<tr>
<th>First variable</th>
<th>second Variable</th>
<th>Psychological Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>StOppt</td>
<td>StOppt</td>
<td>almost invisible</td>
</tr>
<tr>
<td>Shiftrf</td>
<td>Shiftrt</td>
<td>almost none</td>
</tr>
<tr>
<td>Claims1</td>
<td>Claims2</td>
<td>small</td>
</tr>
<tr>
<td>gcountr</td>
<td>ccountr</td>
<td>small</td>
</tr>
<tr>
<td>Product</td>
<td>Sum</td>
<td>large</td>
</tr>
</tbody>
</table>

During debugging it is helpful check for problems caused by insufficient psychological distance between similar variable names and/or routine names. A good habit is to choose variable and routine names with large distances to avoid problems.

Problems of both psychological distance and psychological set show the relation between a good style of programming (comments, suitable names, format) and debugging. If a person has good programming habits s/he should have less problems in debugging his/her programs. However teachers focused on teaching the syntax and semantics of programming frequently forget to teach good programming style. One of our goals is that students should be aware of the importance of having a good programming style. The system that we have designed try to develop this skill in students.

3.3.4.3 Bugs versus Misconceptions

Misconceptions are psychologically plausible misunderstandings of the student that account for bugs manifested in the student’s programs. Common misconceptions include (Joni et al., 1983):
• Overgeneralized Concepts: The student believes a programming language construct is applicable in situations where it is not.

• Inability to Use Multiple Constructs Together Correctly. For example, the student can use loop variables and array indices separately but cannot coordinate their use in a loop to search through the elements of an array.

• Incorrect Analogies. The student treats the programming language like English, believing that the computer understands the tokens in his program.

Accurate bug detection is an important first step in the inference of these misconceptions. Misconceptions cannot be inferred without some means of detecting the bugs that provide evidence for their existence.

Misconceptions are manifested as one of two kinds of bug. The first kind, nonstylistic bugs, result in program nontermination or termination with incorrect values or an error. Using ADD instead of SUB, or omitting a conditional test can result in this kind of error. The second kind of bug, stylistic bugs, are errors in programming style whose only effect are to render code more verbose, less efficient, or less intelligible to other programmers. Examples include non-mnemonic identifiers, unnecessary function definitions that redefine primitives, and unnecessary conditional tests.

3.3. 5 Maintenance and Documenting

Software maintenance is an often overlooked area of software development. It has been reported to absorb up to 70% of the information systems budget (Harrison, 1987). Documenting is the final activity and it is also a crucial element of software engineering. Both maintenance and documenting are very important activities but they will not deal in this research.

3.4 Systems to Teach and to Learn Programming

Several systems have been developed within the area of teaching and learning programming. Some systems even have been focused on debugging programs. Frequently, however, these tools are based on an advanced understanding of the language, and are not really useful to train absolute novices. This section describes the main features of nine
programs designed to teach programming or help students to debug their programs. This is followed by a section that analyses these programs from the point of view of the ways of interaction they have with students and the skills they teach.

**LAURA** (Adam and Laurent, 1980) was built in order to correct, or at least to localise semantic errors. In order to debug a program it is absolutely necessary to have information on what it is intended to achieve. This information is given to the system by means of an implementation of the algorithm to be used. This implementation is supposed to be correct and it is called program model. The program that the system has to debug is an implementation of the same algorithm, syntactically correct, but which may contain semantic errors. The system will compare the two implementations.

In this context, the ability to apply program transformations, either systematically or according to heuristics, is a determining factor. It makes it possible to recognise that a certain part of one program calculates the same functions as a certain part of the other. These parts then can be identified. If a total identification is possible, the student program must be declared correct. Otherwise, the errors are limited to the unidentified parts. The system may recognise an error, express a diagnostic and correct the errors by itself in order to carry on with identifications.

The **LAURA** system can determine that two programs calculate the same functions, which means that they will produce the same results with round-off errors (this is a particular form of equivalence). Thus this work also concerns the problem of automatic checking for the equivalence of two programs. It also makes obvious the great power of program transformations due to their application and especially their activation by an automatic system. This is only possible if the goal is clearly expressed. This is the case for **LAURA** since it considers debugging from the angle of a comparison between two programs, the model and the candidate. It must be pointed out that without any more information than the program model and the student program, some differences cannot be interpreted.

**Lisp Tutor** (Anderson and Reiser, 1985) was built around a model-tracing paradigm. That is, the tutor has built into it an ideal student model, a production-system model of the
different ways a skilled student should write LISP code, that monitors the actual student coding a function. Each production corresponds to the student either typing or planning a portion of the function. When the student deviates from the correct solution path, the tutor gives specific feedback and requires the student to try again (from the most recent production firing, not from the beginning of the function). Student errors that have such remediation are referred to as diagnosed bugs.

If the tutor does not recognise a student’s input as correct or as a bug, the tutor displays “I don’t understand that” and asks the student to try again. These errors, student inputs for which no feedback exists, are referred to as undiagnosed bugs. After entering two undiagnosed errors for a particular input, the student is given the answer that the tutor expected (via the ideal student model) and an explanation of why the answer is correct. Students may also obtain a particular answer and an explanation of the answer at any time while coding a function by pressing an explain key. In sum, there are four actions a student may take that are relevant to each coding-production: entering a correct answer, entering a diagnosed error, entering an undiagnosed error, and typing the explain key. The tutor records each student interaction in terms of the productions possible, the student’s input, and the tutor’s response.

PROUST (Johnson, 1985) uses an intention-based approach to identify and explain bugs written by beginning programmers. It is on-line aid for novice programmers. PROUST has been designed to operate in an ordinary interactive environment. The students edit and compile their programs; whenever a program passes through the compiler without errors, it is automatically passed over to PROUST before it is executed. PROUST then analyses the program for non-syntactic bugs.

For each programming problem that the students are assigned, a problem description is prepared, using a special problem description language. Currently these problem descriptions cannot be developed by the course instructor, but must instead be prepared by someone who is familiar with PROUST’s knowledge base. The problem descriptions are collected into a library. When the student submits a program for analysis, PROUST retrieves the corresponding problem description from the library. Using the problem
description as a guide, PROUST determines what exactly the student's program is intended to do, and how it was intended to do it, identifying bugs in the process. For each bug in the program, PROUST determines what the probable cause is, and uses this information in describing the bug to the student. If the bug was due to a misconception, then PROUST describes the misconception in English.

Proust and Lisp Tutor have two different approaches to automated feedback. The Lisp Tutor "watches" the student as s/he writes the program and intervenes as soon as the student types in some incorrect code. Conversely, Proust takes a complete (syntactically correct) program and suggests corrections to it (Bental, 1995).

Talus (Murray, 1986) relies on its abilities to reason about computational semantics to perform algorithm recognition, infer code teleology and to automatically detect and correct nonsyntactic errors in student program written in a restricted, but nontrivial, subset of LISP. Solutions can vary significantly in algorithm, functional decomposition, role of variables, data flow, control flow, values returned by functions, LISP primitives used, and identifiers used. Solutions can consist of multiple functions, each containing multiple bugs.

The relationship of Talus to a complete intelligent tutoring system is: The student modelling component infers misconceptions from this data and the tutorial expert uses this information to devise remedial instruction that is mediated by the dialog manager. The tutorial expert may directly display the bug corrections, generate a hint, provide a counterexample, or monitor and guide the debugging process in some other manner. The courseware module, represents the corpus of material to be taught.

DAN (Laubisch, Eisenstadt, 1992) Debugging Assistant for Novices, is a suite of program analysis routines which can debug computer programs in the domain of labelled-directed-graph manipulation. DAN was designed in order to make SOLO learning a relatively painless experience for novices. SOLO is primarily a data base manipulation language. User-defined procedures can invoke primitives which add, remove, retrieve or print out structures in a relational data base. Based in part on the existing programmer's apprentice and debugging projects, the system makes several novel contributions. The system deals
with argument passing, recursion, and side-effects. Symbolic evaluation of the user's code produces an effect description which is compared with an idealised effect description, derived from a library plan. The library plan uses domain-specific knowledge to allow the user a great deal of freedom to invent alternate approaches to the programming task at hand. By using effect description mismatches, a variety of hints for correcting the code can be given for the student.

**Capra** (Verdejo, Fernandez, Urretavizcaya, 1993) Capra is an intelligent tutor to teach novice students the design of elementary programs. The system offers interactive sessions following a Socratic style which combines explanation and verification of the different concepts of the domain. Under a mixed-initiative dialogue the system guides the student through its problem solving activities. The tutor dynamically plans a sequence of tutoring actions taking into account the current student's interactions expressed either explicitly or inferred through the communication process.

The main architecture of the system consists of three modules: The Tutor, the Knowledge-based Debugger and the Interface. During a teaching session the Tutor puts forward exercises to check the student's comprehension of concepts. The selection of the problems is carried out based on: the student model, the concept to be checked and the current state of the session. To understand and supervise the pupil during the problem-solving activities the Tutor has an interactive knowledge-based Debugger. The interface acts as the communication module between the student and the system.

A student can ask questions, propose activities or comment tutor proposals and the tutor can makes comments, corrects or explain some concepts.

A diagnosis module checks the student's mistake and establishes their importance taking into account the type of error detected, the affected concept and the student's model. The system then decides whether it is necessary to interrupt the student activity.

The most important features of Capra are 1) it enables interactive guidance during the resolution process, in particular, even if errors are detected the system can follow and partially understand what the student is doing. 2) Different strategies for interrupting the user can be used.
Chapter 3: Learning and Teaching Programming

Ceilidh (Benford et al, 1993) is a framework which provides computer-based support for course administration and the teaching of computer programming. Ceilidh has the ability to assess student’s programs, to provide feedback on them and to provide on-line assistance when the student asks for help. In Ceilidh, interactive tutorial help is provided via electronic mail to a human tutor. The Ceilidh environment has been developed to teach languages including C, C++, Pascal and SML (a functional programming language similar to LISP).

Ceilidh presents an exercise to the students. It also presents a file containing a framework for the code. The student edits the framework file to complete the functions. The student can then to run the functions s/he has written on tests of the students’ own devising and on pre-set test data. When the student is satisfied with the exercise, s/he can submit the program to Ceilidh for assessment. Ceilidh assigns a mark to the program and informs the student. Each student can see the marks s/he has achieved and if the student is unhappy with the mark can ask for help from a human tutor by sending electronic mail from within Ceilidh.

Ceilidh’s assessment is based largely on the correct compilation and on dynamic testing. Some marks are also awarded for the style of the program (absence of redundant, correct type signature with each function).

Marking is only of limited use as feedback. If a program does not compile Ceilidh does not offer any extra help.

ELM-ART II (Weber and Specht, 1997) is an adaptive, knowledge-based tutoring system on the WWW that supports learning programming in LISP. The system supports adaptive navigation as individualised diagnosis and helps in problem solving tasks. Adaptive navigation support is achieved by annotating links. Additionally, the system selects the next best step in the curriculum on demand. All interactions of the learner with ELM-ART II are recorded in an individual learner model. ELM-ART II enables example-based programming. It encourages students to re-use the code of previously analysed examples when solving a new problem. The hypermedia form of the course and, especially, similarity links between examples help the learner to find the relevant examples from his or her previous experience.
ELM-ART II can predict the student's way of solving a particular problem and find the most relevant example from the individual learning history. If the student fails to complete the solution to the problem, or if the student cannot find an error that was reported when evaluating the code, s/he can ask the system to diagnose the code of the solution in its current state. The system gives feedback by providing a sequence of help messages with increasingly detailed explanation of an error or suboptimal solution. The sequence starts with a very vague hint of what is wrong and ends with a code-level suggestion of how to correct the error or how to complete the solution. The system also enables the students to write a message to the tutors asking for help or feedback. A chat window to talk to other students that are using the system at the same time and a discussion list are also available.

**TADP** (Training Aid for Debugging Prolog) (Brna, Hernandez and Pain, 1999) was modelled based on the SOPHIE system, designed to find faults in electronic circuits (Brown, Burton and deKleer, 1982). The main goals of TADP are: encouraging a methodical approach to debugging by supporting activity consistent with a top-down methodology; criticising the student's performance; and detecting whether the student has learned some of the basic debugging principles.

TADP presents a goal together with a buggy program. The student uses a tracer with only a few, simple functions (creep, skip, retry). The student has to identify the type of error. TADP seeks to diagnose the student's performance. The system can give advice if the student is not being consistent, and it can also ask the student for the correct behaviour of some procedures. If the student fails to identify the bug, the system shows the correct answer to the student. Currently, TADP only presents programs with a single bug.

### 3.4.1 Analysis of the Systems

In this subsection the above systems are analysed from different points of view:

- Feedback, when and how each system gives feedback to the learners.
- Communication and collaboration, Do the systems enable communication or collaboration with other learners or with teachers?
- What kind of skill does each system develop?
The Lisp Tutor and Proust are good examples to show opposite ways of offering feedback. The Lisp Tutor is characterised by the way feedback is given at the moment students make a mistake. There are many studies about the value of giving immediate feedback or if it is better that the student learns from their own experience. Bangert-Drowns et al. (1991) claim that a system that provides too much guidance may interfere with the active nature of learning by doing. By contrast, Anderson, Boyle and Reiser (1985) stress the importance of providing error feedback while the buggy knowledge that led to the error is still active and available. The focus of Proust is based on the idea of letting the student work alone without interrupting his/her work. Proust reviews the student's complete program and gives feedback only when the student has finished the task.

An intermediate focus is the utilised by Capra which, depending on the importance of the error, does or does not give immediate feedback. If the error is not very important Capra continues processing until it finds a more important mistake or until the student finishes the task.

There is no simple answer concerning the ideal learning feedback. There are clear advantages to both exploration and guidance (see Merrill et al., 1992). Furthermore, the most appropriate type of learning situation may depend upon factors such as the confidence and ability of the learners, whether they are working alone or in a group, and how easily students can elicit information on their own in the domain to evaluate the success of their reasoning.

Another aspect where the studied systems are different is in how they offer feedback. Talus has different ways; it generates a hint, provides a counterexample or guides the debugging process. DAN and ELM-ART II are similar to Talus in this respect and they also give hints or display counterexamples in order to help students correct their code or answer. When LAURA finds an error it expresses exact and clear diagnostics and corrects it. On other occasions it will mark off one part that contains the mistake and the student herself has to identify what the cause of the mistake is.
Chapter 3: Learning and Teaching Programming

Proust in contrast with Capra does not have sophisticated pedagogical expertise, (it was a future project that the author wanted to perform), so Proust gives information about the mistake but cannot choose between different pedagogical strategies. Capra can choose between different pedagogic strategies for offering feedback.

Another different and unusual way for giving feedback is that proposed by Ceilidh. It gives a mark for the efficiency of the program. The students know that the higher the mark the better their program is. This type the feedback is very subtle because it only gives one idea about the whole efficiency but it does not offer information about the kind of mistake or the place where they are.

In section 3.3.4.2 it was explained that a tip to debug programs is to ask other students or teachers. Frequently the mistake that prevents the program from working is a simple misspelling but the psychological set or psychological distance phenomena mask the mistake. On the other hand, another student or the teacher can often detect the mistake with a quick look at the program. Unfortunately only Ceilidh and ELM-ART II enable students to contact other people: Ceilidh enables students to contact a teacher. In this way students can seek advice from the teacher or show him/her their programs. ELM-ART II enables students to e-mail the tutor and even to chat with other students connected to the program. Although adding a chat to the program is a good idea, it has a disadvantage. It is possible that when a person is interested in joining the chat because s/he needs quick feedback, nobody is connected. This fact can be a little frustrating. Capra does not enable communication with a teacher but it does enable communication between the student and the intelligent tutor system.

Although, as it was explained in the previous chapter, work in collaboration has many advantages, none of the analysed systems has been designed to help students to learn in collaboration. However, when people begin to learn programming, apart from buying a book about the language that they are using, and attending courses if they have access to Internet, they join message lists, or news and work groups where they can ask question or advice from other programmers. So, the students look for the experience and collaboration
of other people. This suggests that in programming learning through collaborative
techniques are often used because students work together to write programs and to take
advice about their doubts in a spontaneous and natural way.

Another lack of the current systems is their focus. Most of them diagnose semantic and
syntactic mistakes and some systems even automatically correct them. The question is, can
a student learn to debug his/her own or others programs using these systems? It is difficult
to answer this question but it is a fact that the current systems with, of course, the exception
of TADP, do not show students programs written by other people so it is difficult for the
student learn how to debug other people's programs. It is also obvious that most of the
authors have worried about teaching their systems to understand programs but little has
been done in order to teach students program comprehension. The style of programming is
another important aspect which is often overlooked. It was explained in this chapter how
style of programming affected the process of program comprehension and debugging. Most
of the systems analysed here focus on syntactic or semantic aspects. Only Ceilidh takes
style of programming into account and reduces the mark if the program exhibits a poor
style of programming. A summary of the systems described in the section is shown in Table
2.

3.5 Summary

This chapter has reviewed the problems that learning and teaching programming present.
Program comprehension and debugging are two important tasks that seldom are take into
account when programming is taught. Both tasks have been described in detail in this
chapter since the system that will be explained in the next chapter focuses on train students
to understand and debug programs.

Although many systems for teach programming have been developed, few of them enable
communication with other people, none of them enables work in collaboration, and they
seldom teach a good style of programming.
<table>
<thead>
<tr>
<th>Functionality</th>
<th>LAURA</th>
<th>LISP TUTOR</th>
<th>PROUST</th>
<th>TALUS</th>
<th>DAN</th>
<th>CAPRA</th>
<th>CEILIDH</th>
<th>ELM-ART</th>
<th>TADP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugging real programs, bias toward programmers</td>
<td>Helps students write programs in LISP.</td>
<td>To identify and explain bugs written by programmers.</td>
<td>During learning, collaboration is automatic.</td>
<td>When the task is finalised, the user receives feedback.</td>
<td>When the task is finalised, the user receives feedback.</td>
<td>Depending on the importance of the error, the system detects errors.</td>
<td>When the student asks for assessment, the system gives marks.</td>
<td>Depending on the importance of error and the correct solution.</td>
<td>Teaching top-down debugging methodology.</td>
</tr>
<tr>
<td>Focus on semantic errors</td>
<td>Displays solution</td>
<td>Displays solution</td>
<td>Displays solution</td>
<td>Displays solution</td>
<td>Displays solution</td>
<td>Displays solution</td>
<td>Displays solution</td>
<td>Displays solution</td>
<td>Displays solution</td>
</tr>
<tr>
<td>Does this Enable Communication? NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Is this a Collaborative? NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Language FORTRAN</td>
<td>LISP</td>
<td>FORTRAN</td>
<td>LISP</td>
<td>FORTRAN</td>
<td>FORTRAN</td>
<td>FORTRAN</td>
<td>FORTRAN</td>
<td>FORTRAN</td>
<td>FORTRAN</td>
</tr>
<tr>
<td>Feedback (When) Immediate</td>
<td>When an error is detected</td>
<td>When an error is detected</td>
<td>When an error is detected</td>
<td>When the task is finalised</td>
<td>When the task is finalised</td>
<td>Depending on the importance of the error, the system detects errors.</td>
<td>When the student asks for assessment, the system gives marks.</td>
<td>Depending on the importance of error and the correct solution.</td>
<td>Teaching top-down debugging methodology.</td>
</tr>
<tr>
<td>Feedback (How) Corrects automatically</td>
<td>Corrects automatically</td>
<td>Corrects automatically</td>
<td>Corrects automatically</td>
<td>Corrects automatically</td>
<td>Corrects automatically</td>
<td>Corrects automatically</td>
<td>Corrects automatically</td>
<td>Corrects automatically</td>
<td>Corrects automatically</td>
</tr>
</tbody>
</table>

Table 2: Systems for teaching programming.
CHAPTER 4. MODELLING A SIMULATED STUDENT TO DETECT SITUATIONS THAT HAMPER COLLABORATION: IMPLEMENTATION IN THE HABIPRO SYSTEM
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

The previous two chapters surveyed the literature on CSCL and on the psychology of programming. The second chapter highlighted the need for more research on controlling students' behaviour in order to maximise the benefits of collaborative learning. The third chapter described some important skills that are seldom taught explicitly in programming courses, even though they are necessary for an efficient programming.

In this thesis we have proposed adding a Simulated Student to a collaborative environment in order to avoid some negative situations that hamper collaboration and thus reduce the efficiency of a learning situation. The first part of this chapter presents a model of the Simulated Student architecture and describes in a general way what roles it should accomplish. The second part explains how a Simulated Student has been implemented in a concrete system, HabiPro, following the model proposed. The Simulated Student's behaviour in HabiPro is also outlined. The third part describes how HabiPro focuses on those aspects of programming that are infrequently taught in programming courses. Both, the Simulated Student model and HabiPro were designed to fill the gaps that the previous chapter have illustrated.

4.1 Model of the Simulated Student

This section presents the architecture and behaviour of the Simulated Student. It describes the aims, roles and tasks of the Simulated Student at a level of abstraction independent of the possible implementations.

The desired behaviour of the Simulated Student influences the configuration of the model. In the model outlined, the Simulated Student has a similar status to the human students (see Figure 4). This is an important feature, which creates several advantages such as favouring a more comfortable environment for collaboration or encouraging the students to reflect. As was explained in chapter 2: students feel more at ease working with friends who have a similar level of knowledge to them. Besides, students interact more with a peer than with a teacher or monitor because they may doubt the information that a peer gives, but students do not normally doubt or reflect on the information that a teacher gives.
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

The Simulated Student of our model acts like a normal human student as much as possible, even to the point of proposing wrong solutions. So it tries to make the human students' learning process as natural as possible, allowing them to think of possible solutions and reflect on all proposals. When a negative situation takes place, the Simulated Student acts in a special role as a "responsible student" trying to prevent any decrease in collaboration or motivation.

The model described in Figure 4 shows the Simulated Student and the human students using the same methods of communication. This makes it different from previous models (Johnson et al., 1998; Goodman et al., 1998) where the students perceived the agents as an animated figure which used another window to communicate with the students. To use the same techniques of communication for both real and Simulated Student helps students to consider the Simulated Student as a partner. In the previous cases, students could see the Simulated Student as an "assistant" that the system offers.

4.1.1 Description of the Model

The Simulated Student model proposed has three main components: the individual Student Model (SM), the Group Model (GM) and the Simulated Student Behaviour Model (SSBM). It also has two complementary modules: the Information Manager and the Interface (see Figure 4). The Interface is the means of communication between the (real and simulated) students. The Information Manager module classifies the information and stores it in the student models and group model. The SSBM uses the information stored in the Group Model and in the Student Models to decide when and how the Simulated Student has to intervene. All the three main components are described in detail in the next few paragraphs.
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

Figure 4. Overview of the model showing flows of information

4.1.2 The Student Model

There is a huge literature on student modelling. VanLehn (1988) defined the student model as the component of an Intelligent Tutoring System (ITS) that represents the current state of a student. Another definition is that proposed by Self (1994) which states that the student model is an element that tries to give information about the students.

A generic student model is formed from a set of entities where each entity expresses information about the learner. The information that the student model contains depends on the goals of the ITS. In our case the goals are to detect and correct situations that decrease motivation and communication in collaborative learning.

If the student model is going to work in a collaborative application further entities must be considered such as individual student's goals or opinions about their partners (Paiva, 1997). The student model that we have designed contains new entities that allow a greater control of the students' behaviour. The entities utilised are:
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

**Frequency of interaction:** One challenge of groupware applications is to provide a collective and equitable benefit, as Grudin (1994) claims there is often a disparity between who does the work and who gets the benefit from it. Equitable and regular participation increases the amount of information available to the group, enhancing group decision making. Improving each student's frequency of interaction increases the likelihood that all group members will learn the subject matter, and decreases the likelihood that only a few students will understand the material, leaving the others behind (Soller et al., 1998). Because of this, controlling the frequency with which a student interacts with the system and with his/her groupmates is very important. The frequency of interaction is a critical factor for detecting passive students.

**Type of interaction:** The students' interaction may be of different types such as talking via chat (this can also be of different types, e.g.: proposal, question or an explanation) or solving exercises in the shared window. Knowing the type of interaction helps to characterise the student's role. In collaborative environments different roles arise, and these roles directly affect cognitive processes (Burton, Brma and Treasure-Jones, 1997).

**Level of knowledge:** The knowledge that a student has is a factor that an ITS should take into account since it would have to adapt its exercises, explanations and, in general, the processing of learning to the student's knowledge.

**Personal beliefs:** In collaborative situations the learner's beliefs are not only about the domain but also about the other learners. One student's belief about another can produce an increase or decrease in the Zone of Proximal Development. If a student thinks that his/her partner has more knowledge about a topic than him/her, s/he expects to learn more by working with this person than studying alone. Gracile (Ayala and Yano, 1995) is a system that uses mediating agents to exchange information about the students' skills and knowledge, trying to maximise the ZPD.

**Mistakes:** The detection of individual mistakes is very important to determine individual misconceptions. If one student makes less mistakes at the end of the session than s/he did at the beginning, this might indicate that learning has taken place.
4.1.3 Group Model

The group model is defined as a way of capturing those aspects that identify the group as a whole (Paiva, 1997). Different opinions exist about how to model the group. Paiva (1997) claims the group is something more than the sum of its parts. For her the group model must be constructed using the actions and beliefs with which the group is in agreement as a basis. A problem with this is how to initialise the group model when the session is beginning. Hoppe (1995) proposes an alternative using individual results to parameterise the group situation.

In this thesis the group model that has been used is based on Paiva's proposal because the amount of information stored in the entities proposed by her model is enough for the Simulated Student to be able to detect the negative situations and act efficiently. The entities are:

- Group knowledge: Beliefs that the group has. These are inferred from the group's actions.
- Group mistakes: The mistakes diagnosed from the group actions are group mistakes. "However, it may be the case that misconceptions that are ascribed to the group are not shared by all the individuals of the group, since the group beliefs are the "accepted" beliefs, and thus may not be held by all" (Paiva, 1997, page 218).
- Differences: The differences between the students are an important factor to consider. An example of difference is the conflict where one student supports theory P and another learner believes the theory not P. On many occasions it is convenient to use the differences between students to trigger possible discussions. This strategy is used by COLER (Constantino-Gonzalez and Suthers, 2000). A coach identifies important differences between students and encourages students to discuss these differences in ways expected to lead to learning.
- Preferences: To know what type of exercises students prefer or what kind of assistance is their preferred permits that the application adapt itself to the users.
4.1.4 The SSBM.

The Simulated Student Behaviour Model (SSBM) uses the information from the students models and the group model to decide how it must act. The architecture of the SSBM is displayed in Figure 5.

![SSBM Architecture Diagram](image)

Figure 5. SSBM architecture showing flows of information

The SSBM is formed of five components, the problem detector, the pedagogical module, the log of the session, the domain knowledge and the action generator.

Domain Knowledge, as its name indicates, contains information about the subject to be learnt. This information is necessary in order to know how much knowledge students must have at each moment and also to adapt the Simulated Student's actions.

The Problem Detector, through the information received from the student models, the group model and the knowledge domain, checks whether negative situations are taking place. This module, as Figure 5 shows, is formed of three sub-components. The first one, in blue, is in charge of control if off-topic conversations arise. The second component, the yellow one, monitors the group's learning process. The last one, in black blue, checks each student's participation in order to detect passive students. It is possible to modify these components or add new ones to avoid other, different negative situations. This is an advantage that this model has, thanks to its modularity.
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

The Pedagogical Module indicates what action the Simulated Student should carry out in order to avoid the problematic situation that the problem detector identified. Several factors are taken into account by the Pedagogical Module before it chooses an action. Apart from the nature of the problem detected, these factors are the individual and group features, and the Log of the Session. The Log of the Session stores all the interventions, even those of the Simulated Student. Having a record of interventions enables the system to know previous Simulated Student's answers and makes it possible not repeat answers or actions. The Log of the Session is also useful for analysing the (simulated or real) students' behaviour.

Examples of types of action that the pedagogical system can trigger are to motivate the students or to reinforce students' learning. The actions are classified into three groups, one group per negative situation; the squares in different colours inside the pedagogical module represent this.

For each type of action that the Pedagogical Module chooses, there exists a set of possible roles that the Simulated Student can play. Depending on the student models, the group model, the Domain Knowledge, and the Log of Session, the Action Generator chooses which role the Simulated Student should play. For example, if the Problem Detector detects a passive student, the pedagogical module can advise that the human student be invited to collaborate. The Action Generator decides how the Simulated Student should invite the student, perhaps with a direct invitation or with a question, etc.

The following paragraphs describe how the components of the Problem Detector works. Figure 6 displays the information that this module uses to detect off-topic conversations. The off-topic conversation detector is divided into three sub-components. First is the Conversation Processor which uses two types of inputs, the conversations and the type of interactions that students are undertaking (the last information is obtained from the student model). The Conversation Processor analyses whether, from the information processed, it can be considered that students are beginning conversations related to non work-related topics. To have off-topic conversations is not always negative. In some situations it can be advantageous in that it increases the feeling of friendship in the group or allows the
members to have a little break. Because of this, the second module is called the Situation Controller and analyses individual and group knowledge, the personal beliefs and the differences between the group members, in order to check whether it is appropriate to cut the off-topic conversation at that precise moment. The module Evaluator combines the output of the Conversation Processor and the Situation Controller and decides whether the Simulated Student should act or not.

![Diagram of conversation detection system]

**Figure 6. Off-topic conversation detector**

The module in charge of detecting passive students is displayed in Figure 7. To detect passive students, several issues must be taken into account since a student can be passive for different reasons. One student may actually have less knowledge and because of this participates less. Or a learner may just believe that the other members have more knowledge than him, although perhaps this is not so in reality, and for this reason prefer to be inactive.

Since the causes of passivity in students may be different, the Passive Students Detector needs different inputs to analyse the possibilities. The frequency and type of interaction of each student are very important for deciding whether one student is passive. However one student may intervene very little but successfully. This could mean that the student reflects on the problem before intervening, and therefore this student should not be considered a passive student. The student's level of knowledge is a critical factor in detecting whether the learner does not take part because he has nothing to propose or because he does not understand the question.
Another important input that the Passive Students Detector uses is the global degree of participation of the group. Comparing this information with the frequency of participation of each student helps to compare individual participation with that of the group. The passive student detector, besides indicating whether there is a passive student, must determine the motive of the passivity. This is the goal of the sub-component called the Selector of Passivity. The pedagogical module uses this information to choose which pedagogical techniques best fitted to that student at that moment.

![Diagram of Passive student detector](image)

**Figure 7. Passive student detector**

The "learning problem detector" (Figure 8) monitors the students' progress to decide when the Simulated Student should intervene. For instance, if the students propose correct solutions, the Simulated Student can ask about the solution in order to check whether the students really understand the solution or if they have just answered it correctly by chance. The group and individual knowledge indicates what topics the students understand individually and at a global level. Both the individual mistakes and the group's mistakes indicates which subjects the students do not comprehend. The preferences of the group are another parameter to be taken into account, since a group may always fail in the same kind of exercise because they do not know how to approach it appropriately, even though they understand the topic which the exercise is asking about.

The Learning Controller sub-module checks whether the students have problems with the topic in hand or whether the students have reached an appropriate level of knowledge (the domain rules indicates what degree of knowledge students should have at each moment).
When irregularities are found in the learning process the Irregularity Selector investigates what is producing the anomaly. This information is passed to the Pedagogical Module which decides what pedagogical support the Simulated Student should offer.

![Figure 8. Learning Problem Detector](image)

### 4.2 Roles of the Simulated Student

The previous section outlined the Simulated Student model and what information it uses in order to act. This section describes how a Simulated Student should behave when one of the three negative situations arises: passive students, students who do not find the solution after several attempts, and off-topic conversations. A subsection is dedicated to each situation. An experiment performed with teachers simulating the role of a Simulated Student (Vizeaino, 2001) helped us to determine the most appropriate behaviour to avoid each circumstance.

#### 4.2.1 Passive Students

Studies of efficient groups shows that the benefits of collaborative learning are achieved by active teams (Soller, 2001). Unfortunately, members of groups are not always active. A group may contain students with little motivation, or the opposite may even occur. That is to say, a hyperactive student does not give other students the opportunity to participate. Both situations are negative because they do not allow all students to improve their skills or knowledge through their experimentation and participation.
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

There may be various reasons why a learner does not take part in solving the exercises set:

- The student does not have the appropriate knowledge to deal with the problem.
- The student does not like working in a group.
- The student does not have enough self-confidence to propose ideas.
- The student does not have the opportunity to collaborate because the other members do not give him/her the chance.

Depending on the reason why the learner does not participate the Simulated Student must behave differently. For instance, if a student does not take part because s/he does not have sufficient knowledge it might be useful for the rest of the students to explain the solution to the problems to her/him. Therefore, double learning is taking place, learning from explanations and learning by teaching. It is commonly found that when students explain their proposals, their knowledge seems to be reinforced and internalised (see e.g. Palthepu et al., 1991).

On the other hand, if the problem occurs by the presence of a hyperactive student, it would be more opportune to offer other specific opportunities to collaborate, for example, using a turn taking protocol.

The Simulated Student should, through different sources of information, infer when there is a passive student and act in accordance with each case. In the following paragraphs different ways of acting depending on the kind of passive student are proposed. These behaviours are summarised in Table 3.

- **Students with insufficient knowledge.** Occurs when a learner does not participate because s/he does not have enough knowledge or his/her knowledge is inferior to that of the rest of the group members. The Simulated Student should pay special attention to the passive student's learning, trying to improve his/her knowledge or skills. Another alternative would be to detect what topics the passive student has more knowledge about, in order to invite him to participate when the questions deal with these topics. Therefore, the ZPD could be maximised. Another possible strategy would be to reduce
the level of difficulty of the problems, if all students have a similar degree of knowledge.

- **Shy students.** A Simulated Student may detect a passive student and find that his/her level of knowledge is similar to that of the other students. The reason why the student does not collaborate may be different to the previous case. Perhaps, the reason why the student is passive could be lack of self-confidence in proposing ideas or being frightened of proposing an incorrect solution. This lack of self-confidence or fear of giving wrong solutions are feelings difficult for the Simulated Student to detect. If the Simulated Student detects that the student has enough knowledge it can only suppose that the cause of the scant participation is shyness. Perhaps, the most appropriate behaviour in this situation is to try a strategy of increasing motivation. For instance, the Simulated Student could ask the students' opinion, congratulate them for their ideas when they do work and not give importance to the fact that a solution does not work. The role of the Simulated Student would be to foster student participation. Another possibility would be to suggest a participation protocol, such as a round robin protocol. However, this technique can have the disadvantage that it can make students feel uncomfortable because they are obliged to participate even though they have nothing to propose.

- **Hyperactive students.** The hyperactivity of one or more students may cause the other student/s not to collaborate. In this case, the Simulated Student should try to decrease the hyperactive student's participation and facilitate the participation of the rest of the students. To achieve this, it is possible to use several techniques, for example: explaining to the hyperactive student that they are working in a collaborative environment and that s/he can learn more by taking into account other opinions and reflection on them; advising techniques of work where everybody has to participate or even suggesting a turn protocol.

Table 3 summarises the roles and strategies that the Simulated Student can use in the event of detecting passive students. These strategies were the fruit of bibliographic suggestions (e.g. Johnson, Johnson and Stanne 1985; Johnson and Johnson, 1987; Crook, 1987;
Berliner, 1989), and our own experience and observation of students when they work in a
group.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Role</th>
<th>Strategy</th>
</tr>
</thead>
</table>
| Student with deficient knowledge. | • The SS asks other students to explain the exercises. It can check if a gain of knowledge has arisen.  
  • The SS investigates what topics the student demonstrates more knowledge about and invites her/him to explain these topics.  
  • The SS checks if it is appropriate to lower the level of difficulty of the exercises. | Learning by teaching.  
  Learning from an explanation.  
  Adaptation of the level of difficulty of the exercises to students' knowledge level. |
| Student with adequate knowledge. | • The SS motivates and invites the passive student to intervene.  
  • The SS suggest turn taking protocols. | To motivate students to participate.  
  Reinforce self-confidence. |
| Hyperactive student.             | • The SS moderates the hyperactive participation and encourages the rest of the students to participate.  
  • The SS suggests using turn protocols. | To guarantee equitable participation.                                  |

Table 3. SS Interventions to avoid passive

4.2.2 Disoriented Students

The following negative situation is related to the learning process. On some occasions
groups waste a lot of time trying to solve a problem in an wrong way. The fact that students try different ways is a good pedagogical technique because students learn from their experiences (Piaget, 1932), and a central part of the learning process occurs when students attempt to apply the instructional material to solve problems for themselves (Anzai and Simon, 1979; Anderson, 1983). Important learning progress may occur when students encounter obstacles, work around them, and explain to themselves what worked and what did not (Anzai and Simon, 1979; Ohlsson and Rees, 1991). However, this type of learning has potential cognitive and motivational pitfalls. Students trying to solve problems sometimes expend much time and effort pursuing blind alleys because of errors or poor strategies. Of course, in some cases students may learn something valuable while searching for a solution. In many cases, however, such episodes leave students confused and frustrated. So if a group does not obtain feedback after spending a lot of time working on a task, members may lose motivation, and even abandon the activity or begin to talk about

89
other topics causing some group members to feel that they are wasting their time.

The Simulated Student might avoid these negative effects by monitoring the students' knowledge and their learning process. When the Simulated Student detects that learners are not close to finding a solution it could give clues or explanations and even, if it is necessary, indicate the correct answer.

The presence of a Simulated Student in collaborative applications could also avoid the Group Think Effect, which is another negative situation that arises in collaborative environments. The Group Think Effect is produced when the group accepts an idea for social reasons or because it is easier to do so. If a Simulated Student asks why they accept a proposal or proposes wrong ideas with the goal of producing doubt, the Group Think Effect should decrease. Table 4 summarises situations that can take place in a collaborative learning process. The role of the Simulated Student and the pedagogic strategy used to control the problem are also shown.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Role</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students do not have enough knowledge so they don't know how to work.</td>
<td>• The SS gives hints or explains the exercises.</td>
<td>Proposing clues, or solutions but always with the goal of fostering students' reflection.</td>
</tr>
<tr>
<td>Students always try wrong solutions (perhaps they are trying to guess the solution)</td>
<td>• The SS explains why that solution cannot work.</td>
<td>To accustom the students to think about the advantages and disadvantages of a proposal.</td>
</tr>
<tr>
<td></td>
<td>• The SS tries to motivate the students (if it occurs that students are bored or tired).</td>
<td></td>
</tr>
<tr>
<td>Students have different points of view about the solution, and they propose different or even opposing answers.</td>
<td>• The SS helps the students to reflect on the different proposals.</td>
<td>To teach respect for different ideas and to think about their advantages or disadvantages. Learning by listening and learning by teaching.</td>
</tr>
<tr>
<td></td>
<td>• The SS encourages the student who proposes the solution to explain it.</td>
<td></td>
</tr>
<tr>
<td>Students propose correct solutions</td>
<td>• The SS checks that students really understand the solutions and that they did not arrive at it by chance.</td>
<td>Checking gain of knowledge</td>
</tr>
<tr>
<td></td>
<td>• The SS proposes a wrong solution to create doubt.</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. SS Interventions to help students learning

4.2.3 Inefficient Conversations

In order to test early versions of the implemented collaborative environment, an experiment was carried out (Vizcaíno, Favela and Prieto, 2001). Students were divided into groups of different sizes and were asked to solve the same set of exercises. Each group member used
a computer and could communicate with his/her peers via chat. The chat conversations were recorded. Upon analysing the conversations it was observed that in many cases students talked about topics that were not related to the exercises; for instance they talked about the football match that they saw the day before. When this type of conversation was brief, it presented no problem, in fact, it could even be advantageous on some occasions because it helped to relax the environment and made students more comfortable with the group. But if these interactions continued for a long time they could be prejudicial to the learning process. As Sipusic et al. (1999) claim, more interaction among participants in a collaborative learning group would be beneficial for learning. However one exception is if the discourse is mostly off-topic and detracts from the time and effort devoted to learning.

Off-topic conversations could be avoided if a member of the group controls that conversations not related to the topic in hand are kept short. This would be the role of a "responsible student" who wishes to take advantage of his/her time. The Simulated Student can play this role if no other student plays it. When the off-topic conversation continues for a long time, the Simulated Student must act to try to redirect the students' attention towards the exercises. Our experience shows that a good method to draw students' attention back to the exercises when they are talking about other topics is to propose solutions to the exercises or ask questions related to the problems. Table 5 outlines the situation and the Simulated Student's role.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Role</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students talk about other topics for a long time.</td>
<td>- The SS suggests continuing with the problems and asks questions or proposes solutions.</td>
<td>Drawing students' attention back towards the problems.</td>
</tr>
</tbody>
</table>

Table 5. SS Intervention to avoid off-topic conversations

Now that the roles of the Simulated Student have been explained in general, we can describe how the Simulated Student works in HabiPro a collaborative application to develop good programming habits in novice students.
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

4.3 HabiPro

HabiPro, from the Spanish Habitos de Programación (Programming Habits), is a collaborative, distributed, synchronous learning system. It is distributed because students can be connected to the system although they are in different geographical places. It is synchronous because students must work at the same time, and it is a collaborative environment because at least two students have to be connected in order for the system to work. The students also have to solve the problems in a collaborative way. In the time space taxonomy for groupware applications described by Ellis, Gibbs and Reln (1991) HabiPro occupies the blue square (see Figure 9).

![Figure 9 Groupware Time Space Matrix](image)

4.3.1 HabiPro Interface

HabiPro provides different spaces of work (see Figure 10). One of them is an unstructured chat window (right window) that permits communication among students. The Simulated Student also uses the chat window to communicate with the real students. The reason why the chat interaction is unstructured is to allow the students free communication and to avoid the situation of them having to think about the type of intervention that they are going to make as well as the exercises.
The bigger window on the left displays the problems to be solved. Below this problem area, we can see the answer windows, one per student. In these windows each student writes his/her proposal. Having one answer window per student permits the learners to know who has proposed each solution. They can use the chat window to decide which solution they think is the correct one, and when they reach an agreement they can check whether the solution is really correct.

Next to each answer window there are three buttons; one to check whether the proposal written in that window is correct, another to explain why the student agrees with that solution, and the last to explain why they do not agree. When students press the agree or disagree button in the chat window the sentence "I agree/disagree because...." appears and the students have to complete the sentence. The goal of these buttons was to facilitate the use of the chat window, so students would have to write less. Controlling who presses these buttons could be a method with which to monitor the types of student interactions.
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

However, experience has shown us that students did not use these buttons very often, so it was not a useful indicator.

The previous version of HabiPro (Figure 11) had only one shared answer window with the goal of students using the chat window to discuss possible solutions, and when an agreement was reached the solution was written in the answer window. Although this technique worked successfully it had the disadvantage that the system could not know how many proposals there were and who had made them since students wrote the proposals in the chat window and the system had no easy way to analyse these conversations. Therefore it was very difficult to obtain information to fill the students models.

![Figure 11 Previous interface with one answer window](image)

In the current version, with one answer window per student, the system can know who has written each solution and test whether the proposal is close to the real solution. In this way, information can be obtained about the students' knowledge. The application also has a collaborative awareness window (the small window in the bottom right hand corner) that shows the names of the students who are connected.
HabiPro can present three different categories of problem: finding the mistake in a program, predicting the result of a program, and completing a program. The different types of problems and their pedagogical goals will be dealt with in section 4.4.4

4.3.2 HabiPro Architecture

This section describes HabiPro's underlying architecture and its components. Figure 12 gives an overview of the architecture and shows that it is a client-server. The components of the client and the server will now be described.

The client is formed of four modules and the communication layer.

- Interface: The interface is the channel of communication used by the Simulated Student and by the real students. As was explained in the previous section, the interface is divided into different sections. The interface, besides being a communication medium is also a means of obtaining information. The information manager receives information exclusively from the interface.
- The information manager analyses the information and decides where it must be stored. For example, if the input is related to a particular student, the information should be sent to the student model. On the other hand, if the information is the result of shared work the information will probably be stored in the group model.
- Student model: This is a very important component since it stores information about the students. The different sub-components that form the student model are described in section 4.4.2.1.
- The collaborative awareness module gives collaboration awareness to the students. This part is very important in non-presence collaboration where students lack some advantages such as eye contact or knowing their companion's emotional state by her/his tone of voice. The awareness window indicates how many students are connected, their names and if it is possible, their photos.
- The communication layer: This enables the sending of messages to the server and the reception of messages from the server.

The server architecture is composed of three parts as well as the communication layer:
• The group model: This is an essential part of the system. The group model represents and characterises the group's behaviour. This information can be used to adapt the application to the students, to show information about the group to the group itself or to a teacher, or to compare group features to individual ones. The group model will be described in detail in section 4.4.2.2.

• The problem component: This is subdivided into two parts: the problems database where exercises are stored, and the answers component in charge of correcting the exercises. The application could be used to teach other topics by simply modifying this component.

• The Simulated Student Behaviour Model: This component decides how and when the Simulated Student should intervene. Its general architecture has already being described in section 4.1.4. The functions of the SSBM in HabiPro will be described in section 4.4.3.

• The communication component: This enables a group of students to be connected, and to be able to send and receive messages. This component has three main functions: (i) Connection, each time the server detects that a client wants to connect, the server has to create a new communication channel, and inform all the connected clients that a new client has connected. (ii) Disconnection: This occurs when a client abandons the session, the server informs the rest of the clients who has disconnected. (iii) The final function is to send control events and messages to the clients.

![HabiPro architecture diagram](image)

Figure 12. HabiPro architecture
4.3.2.1 The Student Model in HabiPro

Section 4.2.2 described the entities, that, in general, a student model should have. Most of the components mentioned above have been implemented in HabiPro. Other components such as personal beliefs are not taken into account by HabiPro. In order to prevent opinions that students may have about other students affecting collaboration, the students are encouraged to use false names when they connect to the program. Anonymity leads to more equitable levels of contributions (Hsi and Hoadley, 1997).

The sub-components of the student model store information that principally enables it to know the students' interactions and their learning progress. Controlling this information enables it to detect and avoid the three negative situations handled. The student model components are:

Frequency and type of interaction: Students may carry out two types of interventions. One is to participate in the chat conversation. Another is to propose a solution in the answer window. This entity monitors how often a student intervenes in the chat and in the answer window. Both parameters are important to determinate the passivity of a particular student.

Density of interaction: The density of interaction is the number of letters written by a student in an intervention. So, when the students writes in the chat window the system calculates the density of the interaction, in other words, the number of letters of the intervention. When students pass to the following exercise the system creates a new density of interaction counter. Therefore the system knows the density of interaction per exercise. This fact avoids the situation of a student who has passive behaviour not being considered passive as because the student had a high density of interaction in the first exercises.

Two more indicators can be calculated from the density of interaction. The total density of the student's work is obtained by adding up all the densities stored, and also the average density can be calculated by dividing the total density by the number of exercises. The three indicators are very useful in detecting passivity.
The frequency of interaction is not enough to decide whether a student is passive. It is possible that one student has an ordinary level of interaction in the chat window but his/her interventions are mainly sentences such as: yes, no, ok or of course, without proposing anything new. Hence, although his/her frequency of interaction indicates a normal level of interaction, s/he could possibly be a passive student.

On the other hand, it was found from our observation of the students' conversations, that the students with more leadership were those whose density of interaction was the greatest. This finding is understandable. Students who have more knowledge (or at least think that they have it) use longer sentences to explain their proposals or to comment on the rest of the solutions.

Another situation that shows the importance of considering the density of interactions is that produced when a student has a low level of frequency of interaction but a high density of interaction. This fact could mean that the student intervenes less than the other members but perhaps offers more or better information because it is the fruit of previous reflection. In this case the student must not be considered a passive learner.

A situation that could lead to misinterpretation would be that a student has a high density of interaction because s/he talks about other topics. However this situation is controlled by the off-topic conversations detector.

From these explanations it can be deduced that the density of interaction is an important factor to detect the passivity or hyperactivity of a student.

**Level of knowledge:** Having one answer window per student enables the system to deduce the level of knowledge mistakes of each student. By comparing the individual proposal to the real solution the system can check whether students are close to the solution. If this is so, the student probably understands the topic, or if the opposite occurs, then s/he probably has a low level of knowledge about the topic in hand.
4.3.2.2 Group Model in HabiPro

The more information the group model has about the students, the more precise it can be in determining what necessities and preferences the group has. In HabiPro four entities are sufficient to avoid the three negative situations.

Group interaction: This indicates the global participation of the group. It is calculated by adding all the individual frequencies of interactions. The Group interaction is useful in order to compare the degree of participation of each student with respect to the group as a whole. This information is also helpful for comparing the results of different groups and for calculating statistics.

Group density: This is the sum of the students' density of interaction. The group density helps us to compare roles within the group. For instance, if a group has a group density of 100, student A has a density of 75 and student B's density is 25, this suggests that student A is probably playing the role of leader, or at least, this student makes more comments and proposals than student B. So student B has more possibility of being considered passive than student A.

Group knowledge: Group knowledge generally is inferred from the group's actions. In order to infer the students' knowledge, HabiPro uses the information from the answer windows. Each student may propose solutions in the answer window, and through the chat students discuss their proposals and they choose which solution to check. This agreement could be considered as an indicator of the group's knowledge.

Group preferences: The type of exercises where students have more success, and the kind of help that they often ask for, are also factors to be taken into account. This information helps the system to adapt to the users.

4.3. 3 The Simulated Student in HabiPro

Based on the model described in this chapter we have developed a Simulated Student in order to detect three negative situations: inefficient conversations, passive students and
students who are unmotivated or lost because they cannot find the solution to the problem. The following sub-sections describe how the Simulated Student intervenes when a negative situation takes place in HabiPro. Each sub-section contains a table summarising the different types of intervention that the Simulated Student can offer, and some examples of interventions. Appendix 1 shows all the types of intervention of the Simulated Student in HabiPro.

4.3.3.1 Passive Students in HabiPro

HabiPro has two areas where students can interact: the chat window and the answer window. From the interaction of the students within these communication areas the frequency, density and type of interaction are calculated. By type of interaction we mean to distinguish between whether the intervention is a proposal of a solution (written in the answer window), or it is part of the conversation in the chat window.

The Detector of Passivity checks the frequency with which each student intervenes by proposing solutions in the answer window. If this parameter is inferior to the established threshold, (the thresholds can be the average of interventions or can be indicated by the teacher), the Simulated Student suspects that the student might be passive.

However, a low frequency of intervention in the answer window is not a clear indicator of passivity. For this reason, the Simulated Student also checks the frequency and density of participation in the chat window and the students' level of knowledge. These indicators help to detect:

- Whether the student's participation is also poor in the chat window. In this case if student's participation density is also low the student is considered passive.
- Whether, although the learner participates little in writing in the answer window, when s/he does participate s/he proposes ideas, therefore his/her density of interaction must be equal or higher than the average of the group density. In this case, the student would not be considered passive, because perhaps the student needs more time to reflect than the rest.
- Whether the student participates with only sentences such as: yes, no, maybe (low density of interaction). In this case the student would be considered passive.
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

When the Detector of Passivity, after studying all the indicators, considers that the learner is passive, the Selector of Passivity has to investigate what type of passivity is taking place. This sub-module checks the number of solutions that have been proposed by the student and the index of mistakes and successes obtained. The study of these parameters joined to the comparison of the specific (features of the student) and global (features of the group) parameters such as density, frequency of interventions or knowledge helps HabiPro to conclude why the passivity might have occurred.

The Simulated Student has two types of intervention, depending on the kind of passivity detected. In the case of the learner not having enough knowledge the Simulated Student focuses its interventions on helping the student to gain knowledge. The types of interventions are:

- Asking for feedback: The goal of this intervention is to look for feedback about the passive student's behaviour. In this way the Simulated Student tries to make the real students realise that there is a student who is participating less, in order for them to help him/her. The Simulated Student may ask questions related to the passive student's knowledge or directly about his/her behaviour (see Table 6).

- Asking the other students for justification: The Simulated Student tries to make the passive student acquire knowledge from the other students' explanations. For this reason the Simulated Student asks the other students with more knowledge to explain the solution.

- Checking knowledge: The Simulated Student checks whether the passive student's knowledge is improving.

When the Selector of Passivity determines that the passive student's knowledge is adequate the role of the Simulated Student is centred on increasing the student's self-confidence, encouraging the learner to participate and to share his/her knowledge. These types of interventions are briefly described below:
• Invitation to participate: The Simulated Student tries to encourage the supposed "shy student" to participate.
• Asking for feedback: The Simulated Student asks the passive student's opinion.
• Asking for an explanation: The Simulated Student tries to get the passive student's attention by asking him/her about his/her behaviour. This is an indirect way of telling the other students that there is a student who is not taking part as much as the rest.
• Asking about preferences: The Simulated Student solicits the passive student's interaction using indirect means, for instance, asking about his/her preferences.

Table 6 shows the types of interventions that the Simulated Student offers. In Appendix 1 the different sentences that the Simulated Student can offer are listed.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Type of Intervention</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student with deficient knowledge.</td>
<td>• Asking for feedback.</td>
<td>• Tom, didn't you understand the previous exercise? You look confused.</td>
</tr>
<tr>
<td></td>
<td>• Asking for justification.</td>
<td>• Peter, Do you mind explaining your solution to us, Tom and I don't understand it.</td>
</tr>
<tr>
<td></td>
<td>• Checking knowledge.</td>
<td>• Tom, this time you propose a solution in the answer window, ok?</td>
</tr>
<tr>
<td>Student with adequate knowledge.</td>
<td>• Invitation to participate.</td>
<td>• Ann, you are very quiet. What do you propose?</td>
</tr>
<tr>
<td></td>
<td>• Asking for feedback.</td>
<td>• What do you think about my proposal, Peter?</td>
</tr>
<tr>
<td></td>
<td>• Asking for explanations.</td>
<td>• John, you aren't joining in much. Are you tired?</td>
</tr>
<tr>
<td></td>
<td>• Asking about preferences.</td>
<td>• Don't you like this kind of exercise?</td>
</tr>
</tbody>
</table>

Table 6. Types of interventions in the case of passive Students

The case of hyperactive students has not been directly dealt with in HabiPro. However it is dealt with in an indirect way, since if in a collaborative environment there is a student whose participation is excessive, the Simulated Student would detect that the other students have an inferior degree of participation. So, the Simulated Student would act by inviting them to participate more in solving the exercises. In this way the effect of the hyperactive student could be decreased.
4.3.3.2 Disoriented Students

The Simulated Student monitors the students' learning in order to check whether learning is taking place or to detect learning problems. For example students may not understand a topic or, after several attempts to search for the solution, they are not close to the correct one. The following paragraphs describe the behaviour of the Simulated Student to ensure correct learning in HabiPro.

The Simulated Student controls both the individual student's knowledge and the group's knowledge through the information received from the answer windows. The Learning Controller checks how many times students try a solution and whether it is close to the correct one. When the Learning Controller detects that students do not have enough knowledge or they do not understand the problem, since they cannot find the solution after an adequate number of attempts, the Learning Controller invites the Simulated Student to act. Normally the interventions of the Simulated Student consist of giving a clue that helps the student to delimit the question or to reflect on the problem.

The number of trials that the Learning Controller considers adequate per exercise depends on the difficulty of the exercise and on the students' level of knowledge.

The Simulated Student suggests hints such as a real student might do, using words that students would use and sometimes without showing much confidence in its proposal. So, students reflect upon these ideas and reject them if they disagree. If the clue was given in the manner of a teacher using technical language and with more authority, perhaps the advantage of working with a Simulated Student would disappear.

In the case of students not understanding the clue proposed by the Simulated Student and continuing to try incorrect answers, the Simulated Student generally suggests the solution and an explanation of it, with the goal of students at least understanding the solution.

The Simulated Student, besides helping students to find the solution to the problem, also makes other types of intervention related to motivating students' learning or checking that
learning has taken place. For instance, when students find the solution at their first attempt, the Simulated Student congratulates them with the aim of maintaining their motivation. This is positive reinforcement. When people are congratulated for their success in their work they usually feel more motivated to continue working. With the goal that the Simulated Student behaves like a real student the congratulatory sentences have been copied from real students' conversations.

One disadvantage of working collaboratively in HabiPro might be that a student who knows the solution writes it in the answer window without explaining to the groupmates why the solution works. A consequence might be that the groupmates would not learn as much as they could. The Simulated Student tries to avoid this situation by checking whether students understand their solutions. For instance, the Simulated Student asks why a solution was proposed and why it is better than the others. In this way it can also verify whether students found the solution by chance.

Another Simulated Student's intervention is to advise the students to consult HabiPro's assistance (HabiPro has two types of help: clues and counterexamples) when the level of difficulty of the problem is high or when the Learning Controller determines that the students' knowledge is insufficient. Table 7 summarises the types of role that the Simulated Student can play to control student learning.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Type of Intervention</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students cannot find the solution.</td>
<td>Giving hints.</td>
<td>The index of an array starts with 0, doesn't it?</td>
</tr>
<tr>
<td></td>
<td>Asking for feedback.</td>
<td>Do you remember if the index of an array starts with 0?</td>
</tr>
<tr>
<td></td>
<td>Consulting HabiPro's help.</td>
<td>Why don't we have a look at the counterexample?</td>
</tr>
<tr>
<td>Even with the clues students cannot find the solution.</td>
<td>Giving solution with explanation.</td>
<td>The solution is j=0 because the index of an array starts with 0.</td>
</tr>
<tr>
<td>Students have different points of view.</td>
<td>Helping to analyse alternatives.</td>
<td>Perhaps, the mistake is in the index of the array instead of in the numbers contained in the array.</td>
</tr>
<tr>
<td></td>
<td>Expressing Disagreement.</td>
<td>I don't agree with Tom's proposal in this case the first sentence is not printed.</td>
</tr>
<tr>
<td>Students find the solution at the first trial.</td>
<td>Congratulations.</td>
<td>We are the best!!!</td>
</tr>
<tr>
<td></td>
<td>Checking students' knowledge.</td>
<td>Why was the solution j=0?</td>
</tr>
</tbody>
</table>

Table 7. Type of interventions to improve students learning
4.3.3.3 Off-topic Conversation in HabiPro

In order to detect whether off-topic conversations are taking place, the Conversation Processor contains different databases. One is a general database which contains words related to problem solving in programming. Another is a specific database that possesses words related to the specific exercises, one database per exercise. The final database, called the playful database, contains words related to conversations that young students might have in their free time (football in the case of Spanish boys, women, men...). Of course, this database can be modified depending on the environment where HabiPro is used.

When students write a sentence in the chat window, the Conversation Processor checks whether the conversation contains keywords stored in the specific database. If no keyword is found, the second step is to check if the student's conversation contains words belonging to the general database. In the case that none of the words in the conversation matches one or more words of the specific and general database, the Simulated Student supposes that students are talking about other topics. If this situation occurs once or twice the Simulated Student does not intervene because a short off-topic conversation usually helps students feel more comfortable in the group or to relax briefly. On the other hand, if the conversation lasts a while it might be negative, so this is when the Simulated Student acts. The Situation Controller is in charge of deciding whether the Simulated Student does or does not have to intervene. The Situation Controller tests the duration of the off-topic conversation, the group's knowledge and the group's preferences in order to determine whether the off-topic conversation took place and whether students do not like the type of exercise or if they do not understand it.

The goal of having a playful database is so that the Simulated Student can know what students are talking about. It can then finish that conversation with a sentence related to the off-topic conversation, with the objective of appearing to understand the conversation. In the case of it not being possible to know what students are talking about because none of the word in the conversation match with the words in the playful database, the Simulated Student tries to draw the students' attention back towards the exercise by suggesting a possible solution.
We performed an experiment with teachers imitating the role of a Simulated Student. The experiment showed that the most efficient interventions to close off-topic conversations were those that did not give an option to continue the dialogue, but only proposed a possible solution to the problem and ignored the student's comments about other topics (Vizcaíno and Prieto, 2000).

Table 8 summarises the two types of role of the Simulated Student, to close off-topic conversation and to draw students' attentions back towards the problems.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Type of Intervention</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students talk about other topics for a long time</td>
<td>• Closing the conversation.</td>
<td>• I don’t like football. Let’s finish this exercise.</td>
</tr>
<tr>
<td></td>
<td>• Giving a clue.</td>
<td>• I think we have to write “new” in the solution don’t we?</td>
</tr>
<tr>
<td></td>
<td>• Proposing a solution.</td>
<td>• The solution is 13, let’s try it.</td>
</tr>
<tr>
<td></td>
<td>• Encouraging to continue.</td>
<td>• Let’s try to solve all the exercises!!!!, ok?</td>
</tr>
</tbody>
</table>

Table 8. Types of intervention to avoid off-topic conversations

4.3.3.4 Other Interventions of the Simulated Student

The Simulated Student developed for HabiPro also intervenes when a new exercise is shown. There are several possible interventions: to give an opinion about the exercise, to ask about the goal of the exercise or even to propose a wrong solution in order to simulate a human student and so to prevent students always accepting its proposals without reflection.

The intervention of the Simulated Student a few minutes after an exercise is displayed decreases the likelihood that students begin to talk about other topics, since it starts the conversation on the right track and usually the students continue it. Besides if the silence is due to the difficulty of the exercise, the Simulated Student can use this intervention to open a way to attack the exercise. The different types of intervention that the Simulated Student may offer are presented in table 8.
Table 8. Types of intervention when a new exercise is shown.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Type of Intervention</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>A new exercise is shown</td>
<td>• Asking about the goal of the exercise</td>
<td>• I don't understand that the exercise is asking Do you understand it?</td>
</tr>
<tr>
<td></td>
<td>• To limit the mistake</td>
<td>• Perhaps the mistake is in the loop. What do you think?</td>
</tr>
<tr>
<td></td>
<td>• Propose a wrong solution</td>
<td>• The exit of the program is 5. Isn't it?</td>
</tr>
<tr>
<td></td>
<td>• Gives an opinion</td>
<td>• This exercise is different from the others.</td>
</tr>
<tr>
<td></td>
<td>• Express Uncertainty</td>
<td>• I don't find the mistake and you?</td>
</tr>
</tbody>
</table>

4.3.4 HabiPro a System to Develop Good Programming Habits.

In the previous section HabiPro has been presented as an application which incorporates a Simulated Student developed to avoid negative situations. This section shows that HabiPro is much more. HabiPro is a system designed to help novice students to develop programming skills.

Learning programming, as was explained in chapter three, is a difficult task. Because of this we have developed HabiPro with the goal of helping students to learn programming, or more concretely to develop the skills necessary to became good programmers.

Chapter three described that learning programming is a complex process where a set of sub-tasks must be learnt in order to learn programming. Some of the sub-tasks are seldom taught explicitly even though they are very important in order to become an expert programmer. HabiPro focuses on three sub-tasks rarely dealt with in programming courses: Program comprehension, debugging and style of programming. Program comprehension and debugging present particular problems for novice students. On the other hand, a good style of programming must be taught to novice students, so they become accustomed to using good practice from the beginning. When the opposite effect occurs and an poor habit is acquired, a double process has to take place: to delete the old habit and to implant a suitable new one.
4.3.4.1 Learning to Debug Using HabiPro

Although a large part of the professional activity of a computer programmer consists of debugging programs, in programming courses debugging is not often treated as a skill that needs to be taught explicitly.

Novice programmers often have misconceptions about programming language syntax and semantics and they lack expert knowledge about how to analyse program specifications and design and implement algorithms. These are several of the reasons why novice programmers introduce many different bugs into their programs.

All programmers must at some time find one or more mistakes which prevent their program from working. It is convenient for novice students to get into the habit of thinking about and predicting the kinds of mistake that prevent programs from working correctly. HabiPro trains students to debug programs through the exercise "Finding the mistake". So, at the same time as novice students learn the concepts of programming, they can practice how to find bugs and to rectify them. In this way students learn from their own experience. "Learning must be an active process" (Piaget, 1932).

Debugging is an even more difficult task when the program to be debugged is written by someone else. Programmers often report that instead of debugging someone else's program, they would rather write their own version (Katz and Anderson, 1988). HabiPro always display programs not written by the students themselves. Therefore HabiPro initiates students into the process of debugging and correcting programs written by other programmers. This is an aspect that HabiPro has in contrast with other previous systems developed to teach programming which only enable the learners' own programs to be debugged.

The mistakes that HabiPro displays in the programs are of different types with the goal of students practising different debugging strategies. One kind of exercise is focused on addressing logical and semantic mistakes as well as student's misconception. The aim of
this type of exercise is to help the students to remember concepts previously learnt and to practice them.

Another type of exercise is centred on syntactic mistakes. Learning and teaching programming is always related to using a programming language. So, an added difficulty in learning programming is to learn with a language with which to practice the concepts learnt.

Syntactic mistakes occur repeatedly in novice programs. Because of this HabiPro can display a program which contains the most frequent syntax mistakes made by novice students. This kind of mistake is usually detected by compilers. However, novice students frequently do not understand the messages generated by the compiler. For this reason HabiPro offers clues (students can ask to see clues or counterexamples) to show the message that the compiler would indicate if students run the program. The clue also explains what the compiler's message means and how the mistake can usually be corrected. So students start to familiarise themselves with compiler messages. The next version of HabiPro will enable students to compile programs.

Besides consulting clues, students can also ask to see related examples. These are correct programs similar to the exercise in hand. So, students have to use the technique of comparison and observation to detect which part of the example looks like their problem and relate the other example solution to their own problem. Cognitive theories suggest that learning within a problem-solving domain is facilitated when clear practical examples are shown, and the learning is improved if students are allowed to practice the schemas learnt by themselves (Cooper, 1990). Through the related example, HabiPro enables students to acquire or remember schemas and put them into practise to correct the problem in hand.

Another type of bug that HabiPro can display in its programs is the "Misspelling mistake". Misspelling is an error often made by programmers but normally difficult to find due to the

---

2 ("A schema is defined as a mental construct permitting problem solvers to categorise problems according to solutions modes" (Sweller, Chandler, Tierney, and Cooper, 1990, p 176).
Psychological Set effect (seeing what you expect to see). With this type of exercise we want students to understand that although programmers usually search for bugs in the more complex part of the code, the bug could simply be a misspelled word. A good style of programming can reduce mistakes of this type, by using names of variables with a great Psychological Distance, for instance.

In order to find a bug in a program students need to understand the program. Program comprehension is a very important topic that involves activities such as identifying important program parts and inferring relationships between them. However, program comprehension is seldom discussed explicitly with the novice. In the next section we will outline how HabiPro's exercises train students to understand programs.

4.3.4.2 Learning Program Comprehension Using HabiPro

Program Comprehension involves the assignment of meaning to a program. The ability to understand programs is an important component of a programmer's skill set. However, the ability to read programs is not a well-developed skill in most programmers. The result of this inability is also the inability to write a readable code (Stacey, 1995).

Studies have shown that there are significant differences between how expert and intermediate programmers research a program. Experts apply effective strategies, which allow them to optimise the use of schemas in programming. In contrast, intermediate programmers cannot restructure their knowledge because of their lack of strategies (Davies, 1991). The ability to comprehend a computer program is a skill that begins its development in the novice programmer and reaches maturity in the expert programmer.

HabiPro dedicates a type of exercise towards the training of students to understand programs. This is called "Predicting the result". HabiPro displays a program and the students have to guess what the result is; for instance, what is printed or how the variables change their values.

By solving this type of problem, students check whether they have enough syntactic and semantic knowledge to comprehend a program, and discover the difficulty of understanding
programs written by others. They also search their mental schemas to try to find the schema which matches the problem.

This kind of problem also helps students to realise the importance of writing comments and sensible variable names that help them and others understand the program. In this way, students indirectly learn the importance of using a good style of programming.

HabiPro possesses another type of exercise where program comprehension is dealt with but in a different way. This kind of exercise, called "Completing a program", shows a program where one line or word is omitted and students have to complete the program. If it is necessary for students to know the program output in order to understand what the program does, HabiPro shows it. In this case the mental process to comprehend the program is different. Students must use the program output in order to guess how the program works.

If there are various the solutions to the exercise, HabiPro only accepts the best one. So students learn that a problem can normally be solved using different techniques but in many cases one solution is better than the rest.

4.3.4.3 Learning a Good Style of Programming with HabiPro

A good style of programming helps to reduce the probability of making mistakes and facilitates the debugging process. Habits such as giving a correct format to the program, adding comments or choosing sensible names for variables help to reduce the Psychological Distance and Psychological Set, both effects of which were described in chapter three.

Usually, if a programmer has good programming habits s/he will make less mistakes and have less problems in debugging his/her programs. However, teachers focused on teaching the syntax and semantics of programming frequently forget to teach good programming style.

Besides teaching good programming habits in an indirect way, as has just been explained in above sections, HabiPro also has a set of exercises focused on showing students the importance of having a good style of programming. There are different kinds, for instance:
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

- Choosing a suitable name for variables, programs or functions.
- Comparing well structured programs and those with a good style of programming to programs written without taking style of programming into account. Also, trying to guess how the program works. In this way students realise that it is more difficult to comprehend a badly structured program.

We want to note that the programs displayed by HabiPro, in the other kinds of exercise, have been written with a suitable format and sensible names of variables with the goal of students learning a good style of programming by seeing correct programs.

In Appendix 2 all the types of exercises contained in HabiPro, including its clues and examples, are presented. Appendix 3 shows the different interfaces of HabiPro, displaying some of the exercises, clues and examples.

4.3.4.4 Differences between HabiPro and Previous Systems

As was seen in chapter three many tools have been designed to teach programming. HabiPro is different to the previous systems in three aspects: First, HabiPro is a collaborative system and none of the previously studied tools have been designed for the students to work in collaboration.

The application was designed to support collaborative learning for two reasons: First, when novice students begin to program they usually make several mistakes in their programs. However, it is often difficult for students to find their own mistakes (similarly, when a person writes a paper it is often difficult for him/her to find their own errors). Because of this, in many cases, students prefer to program in laboratories where there are other fellow students working on similar problems. This way they can discuss doubts, interchange ideas or ask for help to find the mistake/s that prevents their program from working correctly. This fact suggests that collaboration can be of great help when students begin to learn programming. Students can learn through the mistakes that their fellow students have made, and students are also more motivated to work than when they are programming alone.
When a person confronts a problem s/he normally prefers to confront it with another person or in a group.

The second motivation for the system being collaborative, is of a social nature: Professional programmers usually work in group and quality control techniques, such as code peer review, for instance, are recognised to be very effective practices. In Williams et al. (2000) the advantages of pair programming are enumerated, some of these advantages are: the improvement in the problem-solving process, the prevention of defects and even pair satisfaction. It is useful for students to become accustomed to working in a collaborative environment from the beginning. In this way, they can also learn the social skills necessary for their professional development.

The second difference between HabiPro and the other tools is that previous tools have focused on correcting student’s bugs or helping students to develop correct programs. HabiPro, on the other hand, trains students to debug programs written by them or by others, to comprehend programs, and to create programs with a good style of programming.

Third, most previous tools are focused on syntactic and semantic mistakes. Only Ceilidh takes the style of the program into account (absence of redundant, correct type signature with each function, etc). HabiPro tries to help students to develop good programming habits, and most of these habits are narrowly related to the style of programming. Some examples of good style habits are: adding comments, use of indentation that facility to understand the program and well structured programs. These small details help us to understand the programs and to correct them. Well structured programs become easier objects to deal with and, in addition, it is easier to prove that they are correct (Adam and Laurent, 1980).

Table 9 facilitates the comparison between the systems mentioned in chapter three and HabiPro. By looking at the table it is easy to detect other points where HabiPro is different. For instance: the types of feedback that HabiPro offers, or the programming language used.
Chapter 4: Modelling a Simulated Student to Detect Situations that Hamper Collaboration

4.4 Summary

In this chapter three important issues have been dealt with. The first part of the chapter described the model for the Simulated Student. The second part explained how the Simulated Student, which was implemented following the model, works in HabiPro. The last part outlined how HabiPro trains students to debug and understand programs and how it shows students the importance of programming with good programming habits.

The following chapter will describe an evaluation of the Simulated Student's avoidance of the three negative situations and the efficiency of HabiPro in developing good programming habits in students.
<table>
<thead>
<tr>
<th>System</th>
<th>Functionality</th>
<th>Feedback (When)</th>
<th>Feedback (How)</th>
<th>Language</th>
<th>Does this Enable Communication?</th>
<th>Is this Collaborative?</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAURA</td>
<td>Debugging real programs, focus on semantic errors.</td>
<td>Immediate When an error is detected.</td>
<td>Corrects automatically</td>
<td>FORTRAN</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>LISP TUTOR</td>
<td>Helping students to write programs in Lisp.</td>
<td>Immediate. When an error is detected.</td>
<td>Displays solution</td>
<td>LISP</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>PROUST</td>
<td>To identify and explain bugs written by programming beginners.</td>
<td>When the task is finalised.</td>
<td>Gives information about the mistake and corrects it.</td>
<td>PASCAL</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>TALUS</td>
<td>Recognition, detection and correction of the bugs.</td>
<td>When the task is finalised.</td>
<td>Gives hints, counterexamples, or guides the student.</td>
<td>LISP</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>DAN</td>
<td>Debugging assistance for novices.</td>
<td>When the task is finalised.</td>
<td>Gives hints and counterexamples.</td>
<td>SOLO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>CAPRA</td>
<td>Introduction of programming concepts.</td>
<td>Depending on the importance of the error.</td>
<td>Gives explanations about the mistakes. Different Pedagogy strategies.</td>
<td>PSEUDOCODE GO.</td>
<td>With the intelligent tutor</td>
<td>NO</td>
</tr>
<tr>
<td>CEILIDH</td>
<td>Assessment and administration of programming courses.</td>
<td>When the student asks for it.</td>
<td>Gives marks.</td>
<td>C++, C++, PASCAL and SML</td>
<td>With a teacher</td>
<td>NO</td>
</tr>
<tr>
<td>ELM-ART II</td>
<td>Teaching programming through examples, explanations and problem-solving in WWW.</td>
<td>When the student asks for it and when the system detects errors.</td>
<td>Gives hints and counterexamples.</td>
<td>LISP.</td>
<td>With the tutor and with other participants.</td>
<td>NO</td>
</tr>
<tr>
<td>TADP</td>
<td>Teaching top-down debugging methodology.</td>
<td>Depending on the importance of error and when the task is finalised.</td>
<td>Gives hints and display the correct solution.</td>
<td>PROLOG</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>HabiPro</td>
<td>Training students to debug, and to comprehend programs. Teaching a good style of programming.</td>
<td>When students do not find the solution in a suitable time, and when students ask for it.</td>
<td>Two types: The Simulated student: Giving hints. The system: Giving clues and counterexamples.</td>
<td>JAVA</td>
<td>With other students</td>
<td>YES</td>
</tr>
</tbody>
</table>

Table 9. Systems to teach or to learn programming
CHAPTER 5. EVALUATION
Chapter 5: Evaluation

The previous chapter described the model of the Simulated Student, its roles and its behaviour in HabiPro. This chapter presents an experiment designed to test whether the behaviour of the Simulated Student was adequate in each circumstance, and whether its behaviour improved the learning process.

5.1 Introduction

Both the Simulated Student model and HabiPro were developed using an iterative process in which evaluation was essential for the continuous refinement of the model and the system. The results obtained from early pilot experiments helped us to improve the prototypes and to produce the final version. Describing all these experiments is not of any great use. However it is critical to describe the main experiment which indicates how useful the simulated student and HabiPro are.

Experimentation, in general, can be divided into four steps: delimitation of the goals to be achieved through the experiment, proposition of the hypotheses, validation of the hypotheses and analysis of the results. The following sections describe all the four phases carried out in the experiment.

5.2 Objective

As argued in chapter two there are several situations that reduce the advantages of collaborative learning. We have proposed the utilisation of a Simulated Student to monitor negative situations and to avoid the effects of these situations. However, the proposal must be evaluated in order to test the efficiency of the Simulated Student. For this reason it will be convenient to observe how the Simulated Student reacts when faced with certain negative situations and how the behaviour of the Simulated Student affects the other students' learning. Other aspects that must be evaluated are: the students' assessment of the Simulated Student's interventions and how students feel when learning using HabiPro. Given these needs, the aims of the experiment described here where to explore:
Chapter 5: Evaluation

1.- The efficiency of the Simulated Student to detect and avoid off-topic conversations.
2.- The efficiency of the Simulated Student to detect and avoid passive students.
3.- The efficiency of the Simulated Student to detect problems in the learning process and its efficacy in solving the problem.
4.- The effect of the Simulated Student on the students' learning.
5.- Students' assessment of the Simulated Student.
6.- Students' opinion about their learning when using HabiPro.

5.3 Hypotheses

In order to carry out an investigation it is necessary to propose a problem and a potential solution to at problem. This solution is called an hypothesis. The hypothesis were:
Ho (Null hypothesis): Adding a Simulated Student to HabiPro does not affect the students' behaviour and learning.
H1: By adding a Simulated Student to HabiPro students could solve more exercises than using the version without the Simulated Student.
H2: By adding a Simulated Student to HabiPro the effect of passive students could be avoided or decreased.
H3: By adding a Simulated Student to HabiPro the duration of off-topic conversations could be decreased

5.4 Design of the Experiment

To test the hypotheses we designed an experiment in which students had to solve problems using HabiPro in two sessions. In the first session one group of students used a version of HabiPro with the Simulated Student and another group of students used a version without the Simulated Student. In the second session the students used the version of HabiPro that they had not used in the first session. The experiment is a within-subjects design. That means that the comparisons are made between two or more results obtained from different circumstances but always from the same group.
Chapter 5: Evaluation

The within-subject design is the best way to ensure that the groups that work in the different experimental conditions are as similar as possible, since in reality they are the same subjects. In our case, this means, that all subjects used both versions of HabiPro, with and without the Simulated Student. The comparison of their performance in each different situation allows us study the effect of the independent variable. This has the advantage of guaranteeing control of all the variables related to the difference between subjects which might contaminate the results of the investigation. Unfortunately there are several distorting effects that can arise in within-subject designs (León and Montero, 1997). These are:

- Learning effect: The fact that same subjects are involved in several sessions of an experiment may assist subjects to learn how to solve the task in the first session; in consequence they will always perform the task more correctly in the subsequent sessions. In our experiment the learning effect was not very strong because the problems shown in both sessions were different and even the versions of HabiPro were distinct.

- Tiredness effect: This factor was not meaningful in our experiment due to the fact it did not last long enough (each session lasted an average 60 minutes).

- Motivation effect: Motivation did not decrease in the second session, students enjoyed working with HabiPro in the first session and they wanted to work with the system again. Students also knew that the HabiPro exercises were very similar to the exam exercises, so the more they practised the easier it would be to pass the exam.

- Persistency effect: This kind the effect more usually occurs in psychology, pharmacy, or medicine experiments. The persistency effect is related to the duration of the effects of an experiment. For instance, when a research group is investigating the efficiency of a new pill for headaches they have to take into account, before repeating the session, that the effect of the pill may last several hours. This effect was not significant in our experiment since the type of task produced little persistence and besides between the first session and the second there was a convenient space of time (an average seven days).

In our experiment there were two independent variables: the version used (with or without the Simulated Student) and the session (first, second). Table 10 shows the advantages (A)
and disadvantages (D) that starting the session with each version could have. For instance, starting with the version of the Simulated Student has the advantage of having the help offered by this, however by chance the exercises of this version took a little bit longer and sometimes appeared more difficult.

<table>
<thead>
<tr>
<th>Session</th>
<th>Version With Simul. Student</th>
<th>Version Without Simul. Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESSION 1</td>
<td>D: Students have to became familiar with HabiPro.</td>
<td>D: Students do not have the Simulated Student's help.</td>
</tr>
<tr>
<td></td>
<td>D: The exercises are a little more difficult.</td>
<td>D: Students are not familiar with HabiPro.</td>
</tr>
<tr>
<td></td>
<td>A: Students are offered help by the Simulated Student.</td>
<td>A: Exercises are a little easier.</td>
</tr>
<tr>
<td>SESSION 2</td>
<td>D: Students have the Simulated Student's help.</td>
<td>D: Students may miss the Simulated Student's help.</td>
</tr>
<tr>
<td></td>
<td>D: Exercises are a little more difficult.</td>
<td>A: Students are familiar with the use of HabiPro.</td>
</tr>
<tr>
<td></td>
<td>A: Students are familiar with working with HabiPro.</td>
<td>A: Exercises are a little easier.</td>
</tr>
</tbody>
</table>

Table 10. Advantages and disadvantages of starting with each HabiPro version

5.4.1 Subjects

Forty-four students enrolled in the first course of the subject Introduction to Programming, in the first year of the Computers Science degree in Ciudad Real took part in the experiment. The students chosen had to have one factor in common: the same teachers in their theory and practice classes to avoid the possibility of some students knowing more because their teacher had explained more or faster, and thus a similar level of knowledge was expected. Those students who were repeating the course could not to take part in the experiment because they might have had more knowledge than the students who had just started to learn programming. The experience was part of their normal teaching. It was carried out in the same laboratory in which they attended their practical classes and at the same time that they were used to attending them.

Students were randomly divided into the two sub-groups, one subgroup started the experiment working with the version of HabiPro containing the Simulated Student and the
other subgroup with the version without Simulated Student. The sub-groups were also randomly divided into couples. So we had two subgroups of eleven pairs.

5.4.2 Procedure

Each couple taking part in the experiment attended two sessions about one week apart. The sessions lasted approximately one hour. Each pair had to solve programming problems using a different version of HabiPro in each session. So, the eleven couples that used the version without HabiPro in the first session used the version with HabiPro in the second session, and vice versa. Each student worked from a computer and they communicated with each other using the chat window: that means that members of a couple could be in different geographical locations.

All the students undertook a pre-test (see section 5.4.3) one week before the first session. The time that students used to answer the problems was not controlled, since the goal of the pre-test was to indicate the number of correct exercises that students could solve alone and without HabiPro.

Before starting the first session, students were introduced to HabiPro through a small demonstration. Students belonging to team A were introduced to the version of HabiPro with the Simulated Student. They were informed that they would solve the exercises by collaborating with two other students. Students did not know that one of their work mates was the Simulated Student. In this way we tried to avoid the situation of students considering the simulated student's proposal as being more appropriate or, on the other hand, ignoring it because came from an unreal student. Team B was introduced to the version without the Simulated Student. The teacher told students that they should solve the exercises by collaborating with one other student via the chat window.

Both versions of HabiPro, with and without the Simulated Student, contained nineteen exercises. The exercises of both versions are given in Appendix 2. However, the version with the Simulated Student contained exercises which were a little more difficult to solve and slightly longer. The exercises of the version with the Simulated Student contains arrays
and loops, topics that students often do not understand very well. This occurred by chance since we tried to prepare similar exercises for both versions. The fact of preparing the exercises for the version without HabiPro first and the exercises for the version with the Simulated Student second probably influenced us to think up exercises which were a little more difficult. Having longer and more difficult exercises in the version with the Simulated Student might cause students to solve less exercise than when using the other version.

Both versions recorded the students' actions and their participation in the chat window. Although we were aware that this technique was not too ethical, we considered that this was the only method of obtaining real information about students' reactions to the Simulated Student and it was the only way to check whether off-topic conversations were avoided. If students had known that all their interactions were stored, their behaviour would have been different and the results of the experiment would not have been so real. No actual student can be identified from the analysis presented here.

Students were encouraged to solve as many exercises as possible but always reflecting on the solution and ensuring that they understood it. It was explained that working with HabiPro helped them to discover frequent mistakes and to familiarise them with the type of exercises that would appear in their exam. Students were also motivated to work in a group, sharing their opinions, making proposals and helping other students with problems.

The importance of maintaining their anonymity was also explained to the student. We remind the reader that students connected to HabiPro with a false name with the goal that their personal beliefs did not influence the collaborative process.

The first session of the experiment was carried out during the fist hour of one practical class. For this reason, students did not feel that they were taking part in an experiment. They thought that they were practising programming using new technology. At our university students have two hours of laboratory practice per week in the first course. In this time, students practice the concepts explained in the theory class.
Chapter 5: Evaluation

When the first session finished students were individually invited to fill in a questionnaire where they gave their opinion about the participation and collaboration of their partners, their assessment of HabiPro and even recommendations or suggestions.

The second session was carried out the next time that the students had lab practice, in other words, one week later. Due to the fact that the students had to use the opposite version of the one that they used the first time, a small introduction to the new version was needed. In this case, team A was introduced to HabiPro without the Simulated Student and so on.

Students were informed that the different version contained different exercises and they were again motivated to solve the exercises and to work in collaboration. The mistakes related to the use of HabiPro that were detected in the fist session were commented on with the goal that they would not be repeated. An example of one mistake was using the answer window to communicate with the others instead of using the chat window.

When the second session finished, students individually filled in a questionnaire which was similar to the one in the first session, the only difference being that team A had to give their opinion about two partners (the human student and the virtual one) in the first session, and in the second session only about one partner, the human student. The opposite applied to team B.

5.4.3 Materials

Apart from HabiPro, whose design and types of exercises were described in Chapter 4, students were presented with a pre-test, and with a questionnaire. A description of them follows.

Tests

The pre-test contained 10 exercises similar to the exercises shown in HabiPro. The pre-test is presented in Appendix 4. This test was designed to measure general knowledge of programming.
Questionnaire

Students were asked to fill in a questionnaire at the end of the first and second experimental sessions. There were two types of questionnaire depending on the version of HabiPro used. The questionnaires are presented in Appendix 4.

The questionnaires were designed to measure the subjects' perceptions of their partners and of HabiPro as a learning tool, in particular as a tool for developing good programming habits.

The questions focused on students' opinions about:

1. The degree of help offered by the human partners.
2. The degree of help offered by the Simulated Student.
3. The feeling of having learnt.
4. The degree to which work in collaboration improved their learning.
5. Their general feeling towards HabiPro.

5.4.4 Dependent Variables

The following data was collected in the experiment, see Appendix 5 for example of the logs of interaction used to record the data.

1. Usefulness of the Simulated Student in passive student situations.
2. Usefulness of the Simulated Student in off-topic situations.
3. Usefulness of the Simulated Student in the student's learning process. Influence of the Simulated Student on the number of exercises solved.
4. The students' assessment of HabiPro.
5. Students' assessment of the Simulated Student.
6. Feelings of having learnt.
7. Number of exercises solved with the Simulated Student.
8. Number of exercises solved without the Simulated Student.
5.5 Results

5.5.1 Does the Simulated Student Influence the Rate of Solving Problems?

In this section a comparison is made of the number of exercises solved correctly in the different sessions. In this way we will test whether the students solved more exercises in one session than in the other one, or whether working with the Simulated Student influenced the number of exercises solved. The results obtained from the pre-test are also compared to the results obtained using HabiPro, with or without the Simulated Student, in order to see the influence of HabiPro on the students' learning.

Table 11 shows the number of exercises that each pair of students solved correctly. WSS stands for "Without Simulated Student", SS stands for "Simulated Student".

<table>
<thead>
<tr>
<th>Pair</th>
<th>Version used in the first session</th>
<th>Number of Exercises solved in 1st Session</th>
<th>Number of Exercises solved in 2nd Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WSS</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>WSS</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>WSS</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>WSS</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>WSS</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>WSS</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>WSS</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>WSS</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>WSS</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>WSS</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>WSS</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>SS</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>SS</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>SS</td>
<td>16</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>SS</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>SS</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>17</td>
<td>SS</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>SS</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>SS</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>SS</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>21</td>
<td>SS</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>22</td>
<td>SS</td>
<td>11</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 11. Number of exercises solved in each session
In order to know whether there are differences between the number of exercises solved in the first session and the number of exercises solved in the second session a test of normality was carried out. This test is useful to check whether both variables have or do not have a normal distribution. Depending on their distribution parametric or nonparametric tests should be used:

<table>
<thead>
<tr>
<th></th>
<th>Kolmogorov-Smirnov</th>
<th>Shapiro-Wilk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Statistic</td>
<td>df</td>
</tr>
<tr>
<td>Exercises Session 1</td>
<td>.225</td>
<td>44</td>
</tr>
<tr>
<td>Exercises Session 2</td>
<td>.204</td>
<td>44</td>
</tr>
</tbody>
</table>

Table 12. Test of Normality

Due to the sample being less than 50 the data of the Shapiro-Wilk test should be considered. This test indicates, with \( p = 0.010 \), that none of the variables has a normal distribution. Therefore a nonparametric test should be applied in order to see whether significant differences exist between the results of the first and second sessions. Two nonparametric tests are used: the sign test and the Wilcoxon signed-rank test. The McNemar test cannot be used because the variables are not dichotomous.

For each pair of observations, the sign test only uses the direction of the differences (positive or negative), while the Wilcoxon signed-rank test begins by ranking the differences without considering the signs, restoring the sign to each rank, and finally summing the ranks separately for the positive and negative differences. This test offers more information about the data than the sign test. The following tables show the results obtained after applying both test to number of exercises of Session 1, and number of exercises Session 2.

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercises Session 1</td>
<td>44</td>
<td>10.68</td>
<td>2.82</td>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>Exercises Session 2</td>
<td>44</td>
<td>10.86</td>
<td>2.31</td>
<td>7</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 13. Descriptive Statistics
# Chapter 5: Evaluation

## Wilcoxon Signed-Rank Test

<table>
<thead>
<tr>
<th>Exercises Session 2 - Exercises Session 1</th>
<th>N</th>
<th>Mean Rank</th>
<th>Sum of Ranks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative Rank</td>
<td>20(a)</td>
<td>21.50</td>
<td>430.00</td>
</tr>
<tr>
<td>Positive Rank</td>
<td>22(b)</td>
<td>21.50</td>
<td>473.00</td>
</tr>
<tr>
<td>Equals</td>
<td>2(c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a Exercises Session 2 < Exercises Session 1  
b Exercises Session 2 > Exercises Session 1  
c Exercises Session 1 = Exercises Session 2

Table 14. Ranks

### Experiments Session 2 - Experiments Session 1

<table>
<thead>
<tr>
<th>Z</th>
<th>.270(a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asymptotic Significance (2-tailed)</td>
<td>.787</td>
</tr>
</tbody>
</table>

a. Based on negative ranks.  
b Wilcoxon signed-rank test

Table 15. Contrast Statistic (b)

## Sign Test

<table>
<thead>
<tr>
<th>Exercises Session 2 - Exercises Session 1</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative Differences (a)</td>
<td>20</td>
</tr>
<tr>
<td>Positive Differences (b)</td>
<td>22</td>
</tr>
<tr>
<td>Equals(c)</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
</tr>
</tbody>
</table>

a Exercises Session 2 < Exercises Session 1  
b Exercises Session 2 > Exercises Session 1  
c Exercises Session 1 = Exercises Session 2

Table 16. Frequencies

### Experiments Session 2 - Experiments Session 1

<table>
<thead>
<tr>
<th>Z</th>
<th>-.154</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asymptotic Significance (2-tailed)</td>
<td>.877</td>
</tr>
</tbody>
</table>

a Sign Test

Table 17. Contrast Statistic (a)
Equivalent results are obtained with both statistics. However due to the fact that the size of the sample is smaller than 50, the Wilcoxon test is taken more into account. The Wilcoxon test indicates that the Asymptotic Significance is 0.787. This means that the probability of making an error if the null hypothesis was rejected is very high, for this reason the null hypothesis should be accepted. The null hypothesis is that the distributions are equal.

The next step is to check whether students solved more exercises with a particular version of HabiPro. In order to study this fact, the variable "version" is considered as a factor, its value indicates whether the students started to work with or without the Simulated Student. Thus, it is necessary to divide the initial sample into two sub-samples: students who used the version with the Simulated Student in the first session, students who used the version without the Simulated Student in the first session and the same division for the second session. The conditions of normality and variance are studied in both sub-samples. Table 18 shows the results.

<table>
<thead>
<tr>
<th>Version</th>
<th>Kolmogorov-Smirnov</th>
<th>Shapiro-Wilk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Statistic</td>
<td>df</td>
</tr>
<tr>
<td>Exercises Session 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>With Simulated Student</td>
<td>.125</td>
<td>22</td>
</tr>
<tr>
<td>Without Simulated Student</td>
<td>.348</td>
<td>22</td>
</tr>
<tr>
<td>Exercises Session 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>With Simulated Student</td>
<td>.165</td>
<td>22</td>
</tr>
<tr>
<td>Without Simulated Student</td>
<td>.295</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 18. Test of Normality
Chapter 5: Evaluation

<table>
<thead>
<tr>
<th>Exercises Session 1</th>
<th>Levene Statistic</th>
<th>df1</th>
<th>df2</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Based on Mean</td>
<td>.031</td>
<td>1</td>
<td>42</td>
<td>.861</td>
</tr>
<tr>
<td>Based on Median</td>
<td>.136</td>
<td>1</td>
<td>42</td>
<td>.714</td>
</tr>
<tr>
<td>Based on Median and with Adjusted df</td>
<td>.136</td>
<td>1</td>
<td>39.894</td>
<td>.714</td>
</tr>
<tr>
<td>Based on Trimmed Mean</td>
<td>.135</td>
<td>1</td>
<td>42</td>
<td>.715</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exercises Session 2</th>
<th>Levene Statistic</th>
<th>df1</th>
<th>df2</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Based on Mean</td>
<td>.588</td>
<td>1</td>
<td>42</td>
<td>.447</td>
</tr>
<tr>
<td>Based on Median</td>
<td>.121</td>
<td>1</td>
<td>42</td>
<td>.730</td>
</tr>
<tr>
<td>Based on Median and with Adjusted df</td>
<td>.121</td>
<td>1</td>
<td>30.788</td>
<td>.731</td>
</tr>
<tr>
<td>Based on Trimmed Mean</td>
<td>.217</td>
<td>1</td>
<td>42</td>
<td>.644</td>
</tr>
</tbody>
</table>

Table 19. Test of Homogeneity of Variance

The results of the Shapiro-Wilk test are different for each value of the factor "Version". The test indicates that the null hypothesis should be rejected (with a p value of 0.000) for the value "Without the Simulated Student", the null hypothesis is that the variable has a normal distribution. However, the test indicates a normal distribution in the case of the version having a value of "With Simulated Student". So independent of the session, the distribution is normal when the version with the Simulated Student is used and the distribution cannot be considered normal when students used the version without the Simulated Student.

The Levene Statistic shows that the level of significance is too big to refuse the null hypothesis (H₀: there is homogeneity of variance), hence the samples have a homogeneity of variance in both sessions.

The results obtained imply that a parametric study should be performed when the Simulated Student is used. In contrast, if the version of HabiPro without the Simulated Student is used a nonparametric study might be made.

The next study starts analysing the first sub-sample where the value of "version" is "with Simulated Student" in the first session and, of course, "without Simulated Student" in the second session. The following Table 20 shows the average number of exercises solved in each session.

128
Chapter 5: Evaluation

<table>
<thead>
<tr>
<th>Sub-Sample 1</th>
<th>Exercises Session 1</th>
<th>Mean</th>
<th>N</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>12.18</td>
<td>22</td>
<td>2.34</td>
<td>.50</td>
</tr>
<tr>
<td>Exercises Session 2</td>
<td>10.27</td>
<td>22</td>
<td>1.75</td>
<td>.37</td>
<td></td>
</tr>
</tbody>
</table>

Table 20. Statistics to measure the relationship among variables

We can see that there is a difference between means: students solved more exercises in the first session with the Simulated Student than in the second session without it. The next step is to test whether the difference is significant. The statistic used to compare means is the T-Student:

<table>
<thead>
<tr>
<th>Paired Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
</tr>
<tr>
<td>Mean</td>
</tr>
<tr>
<td>Lower</td>
</tr>
</tbody>
</table>

| Sub-sample 1 | Exercises Session 1 - Exercises Session 2 | 1.91 | 2.49 | .53 | .81 | 3.01 | 3.601 | 21 | .002 |

Table 21. Paired Samples Test

The T-Student test indicates that there is a significant difference between means. This argues that students do tend to solve more exercises using the version of HabiPro with the Simulated Student.

Let's see what happens with the second sub-sample, when students did not use the Simulated Student version in the first session but they did use it in the second session. The following table indicates that there is difference in the means.

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercises Session 1</td>
<td>22</td>
<td>9.18</td>
<td>2.46</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>Exercises Session 2</td>
<td>22</td>
<td>11.45</td>
<td>2.67</td>
<td>9</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 22. Descriptive Statistics

Since the sample has a difference of means, the next phase will be to check whether the difference is significant. Because the sample, in this case, does not have a normal
distribution, the nonparametric tests, the sign test and the Wilcoxon signed-rank tests, are again used.

<table>
<thead>
<tr>
<th>Wilcoxon signed-rank test</th>
<th>N</th>
<th>Mean Rank</th>
<th>Sum of Ranks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercises Session 2 - Exercises Session 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative Rank</td>
<td>2(a)</td>
<td>10.50</td>
<td>21.00</td>
</tr>
<tr>
<td>Positive Rank</td>
<td>20(b)</td>
<td>11.60</td>
<td>232.00</td>
</tr>
<tr>
<td>Equals</td>
<td>0(c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>22</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 23. Ranks

<table>
<thead>
<tr>
<th>Exercises Session 2 - Exercises Session 1</th>
<th>Z</th>
<th>Asymptotic Significance (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-3.450(a)</td>
<td>.001</td>
</tr>
</tbody>
</table>

Table 24. Contrast Statistic (b)

<table>
<thead>
<tr>
<th>Sign Test</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercises Session 2 - Exercises Session 1</td>
<td></td>
</tr>
<tr>
<td>Negative Differences (a)</td>
<td>2</td>
</tr>
<tr>
<td>Positive Differences(b)</td>
<td>20</td>
</tr>
<tr>
<td>Equals(c)</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 25. Frequencies

<table>
<thead>
<tr>
<th>Exercises Session 2 - Exercises Session 1</th>
<th>Exact Significance (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.000(a)</td>
</tr>
</tbody>
</table>

Table 26. Contrast Statistic (b)
The test shows that the difference between means is also significant in the case of students starting to solve the problems without the Simulated Student. From observing the contrast tests, the Wilcoxon or the signs tests it can be deduced that students solved less exercises in the first session without the Simulated Student than in the second session with it.

This is an important issue that shows that independently of using first the version with or without HabiPro, students tend to solve more exercises when they utilise the version with the Simulated Student. So the data support the hypothesis that the Simulated Student helps students to solve more exercises.

The next issue that is going to be analysed is whether using a specific version in the first session influences the results of the second session. The total number of exercises correctly solved in both sessions starting with and without the Simulated Student will be studied. Before this, it will be checked whether the sample follows a normal distribution if this is so, then a parametric study can be used. Thus the normality test and homogeneity of variance are performed.

<table>
<thead>
<tr>
<th>Kolmogorov-Smirnov</th>
<th>Shapiro-Wilk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic  df  Sig.</td>
<td>Statistic  df  Sig.</td>
</tr>
<tr>
<td>TOTAL   .147   44  .018</td>
<td>.902   44  .010</td>
</tr>
</tbody>
</table>

Table 27. Normality Test

<table>
<thead>
<tr>
<th>TOTAL</th>
<th>Levene Statistic</th>
<th>df1</th>
<th>df2</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Based on Mean</td>
<td>.464</td>
<td>1</td>
<td>42</td>
<td>.500</td>
</tr>
<tr>
<td>Based on Median</td>
<td>.008</td>
<td>1</td>
<td>42</td>
<td>.927</td>
</tr>
<tr>
<td>Based on Median and with Adjusted df</td>
<td>.008</td>
<td>1</td>
<td>29.795</td>
<td>.927</td>
</tr>
<tr>
<td>Based on Trimmed Mean</td>
<td>.070</td>
<td>1</td>
<td>42</td>
<td>.793</td>
</tr>
</tbody>
</table>

Table 28. Test of homogeneity of variance

131
In this case the sample does not follow a normal distribution. However it has homogeneity of variance, therefore a contrast nonparametric is performed.

### Mann-Whitney Test of both Session

<table>
<thead>
<tr>
<th>SESSION</th>
<th>N</th>
<th>Mean Rank</th>
<th>Sum of Ranks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Simulated Student</td>
<td>22</td>
<td>18.05</td>
<td>397.00</td>
</tr>
<tr>
<td>With Simulated Student</td>
<td>22</td>
<td>26.95</td>
<td>593.00</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>44</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 29. Ranks

<table>
<thead>
<tr>
<th></th>
<th>TOTAL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mann-Whitney U</td>
<td>144.000</td>
<td></td>
</tr>
<tr>
<td>Wilcoxon W</td>
<td>397.000</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>-2.313</td>
<td></td>
</tr>
<tr>
<td>Asymptotic Significance (2-tailed)</td>
<td>.021</td>
<td></td>
</tr>
</tbody>
</table>

a Variable of group: SESSION

Table 30. Contrast Statistic (a)

### Kolmogorov-Smirnov Test for two samples

<table>
<thead>
<tr>
<th>SESSION</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Simulated Student</td>
<td>22</td>
</tr>
<tr>
<td>With Simulated Student</td>
<td>22</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>44</td>
</tr>
</tbody>
</table>

Table 31. Frequencies

<table>
<thead>
<tr>
<th>Differences</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute</td>
<td>.636</td>
</tr>
<tr>
<td>Positive</td>
<td>.091</td>
</tr>
<tr>
<td>Negative</td>
<td>-.636</td>
</tr>
<tr>
<td>Kolmogorov-Smirnov Z</td>
<td>2.111</td>
</tr>
<tr>
<td>Asymptotic Significance (2-tailed)</td>
<td>.000</td>
</tr>
</tbody>
</table>

a Variable of group: SESSION

Table 32. Contrast Statistics (a)
Both contrasts show the same results: the total number of exercises solved in both sessions is bigger when students used the Simulated Student in the first session. This fact suggests that the Simulated Student in addition to helping students to solve exercises, teaches them to work in collaboration and with HabiPro. So, students that started to work with the Simulated Student in the first session solved more exercises in the second session than the students that did not use the Simulated Student in the first version. This is an important issue since it indicates that the Simulated Student improves the performance of students using HabiPro even in a posterior session.

5.5.2 Comparison with the Results of the Pre-Test

Two weeks before using HabiPro, students tried to solve exercises similar to those presented by the collaborative system, however they had to solve them individually and by hand. Figure 12 shows the number of exercises that students correctly solved in the pre-test. The majority of the students solved only four or five exercises correctly.

![Graph of pre-test exercises](image)

Figure 13. Exercises correctly solved in the pre-test
Comparing the number of exercises correctly solved individually with the number exercises solved using HabiPro helped us to know whether by using the system students improved their performance. The pre-test had ten exercises. Table 33 shows that the mean of exercises solved correctly is 5.36.

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Test exercises</td>
<td>44</td>
<td>5.36</td>
<td>1.75</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>TOTAL</td>
<td>44</td>
<td>21.5455</td>
<td>4.0889</td>
<td>16.00</td>
<td>33.00</td>
</tr>
</tbody>
</table>

Table 33. Mean of Exercises Solved in Pre-test

The following table indicates that the mean number of exercises that students solved correctly in the pre-test is rather smaller than by using HabiPro, independent of whether the Simulated Student was utilised or not.

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Test exercises</td>
<td>44</td>
<td>5.36</td>
<td>1.75</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Without Simulated Student</td>
<td>44</td>
<td>9.7273</td>
<td>2.1820</td>
<td>7.00</td>
<td>14.00</td>
</tr>
<tr>
<td>With Simulated Student</td>
<td>44</td>
<td>11.8182</td>
<td>2.5083</td>
<td>8.00</td>
<td>19.00</td>
</tr>
</tbody>
</table>

Table 34. Comparison of Means

In order to test whether the difference of means was significant the Friedman test was used. (although the table is not shown in this section to avoid making it too long, it was found that the number of exercises solved in the pre-test did not have a normal distribution). The Friedman statistic is a nonparametric test based on ranks within different cases. The scores for each variable are ranked and the mean ranks for the variables are compared. In this case the number of exercises solved in the pre-test case and in the cases of using or not using the Simulated Student are ranked. The mean ranks are calculated and compared for the three cases. The Friedman test is used to test the null hypothesis that there is no difference in the number of exercises solved in each case. The Friedman test statistic is approximately distributed as a chi-square distribution.
The value of the Chi-Square is 75.779, and p value < 0.005. Therefore the null hypothesis that there is no difference in the number of exercises solved in each case must be rejected. This means that HabiPro, even without the Simulated Student, improves the students' performance, and they solve more exercises correctly.

5.5.3 Did The Simulated Student Detect when Students Needed Help to Solve the Exercises?

One role of the Simulated Student is to help the students to solve the exercises when the learners do not have enough level of knowledge or they are lost. When this happens the Simulated Student gives clues, hints or proposes solutions close to the real one. In this section the degree of success of the Simulated Student in playing the role of adviser is analysed. Before analysing the results obtained when students used the version with the Simulated Student we are going to see with what frequency students needed help to solve the problems when they worked without the Simulated Student. The first row of Table 37
indicates each couple number. The second row indicates the number of times that each couple needed three or more attempts to solve the problem, this means that the couple proposed two wrong solutions.

| C. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | To. |
|----|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Nu | 3 | 4 | 2 | 2 | 3 | 4 | 2 | 1 | 2 | 3 | 1 | 3 | 2 | 3 | 5 | 4 | 3 | 2 | 3 | 2 | 3 | 59 |

Table 37. Number of times that students had problems in solving the exercises

Table 37 shows that in 59 situations students did not solve the exercises in the first two trials. These results will be commented on later. Table 38 shows the results obtained when students used the version with the Simulated student. The logs stored when students worked with this version were analysed in order to answer the following questions:

- How many times did the Simulated Student detect that students needed assistance to solve the exercises?
- Did the Simulated Student's intervention help students to solve the exercises?
- Did students always consider the Simulated Student's advice?
- How many interventions by the Simulated Student were necessary to solve the problem?
- Did the Simulated Student act when it was inappropriate to do so?

The first column Table 38 indicates each pair's number. The second column, called "number of times that students needed help", indicates how many times a pair had "problems" in solving the exercise. By having problems, we mean that the couple proposed two wrong solutions. This information was obtained from the stored logs. These contained all the answers written in the answer window (even the incorrect ones), all the conversation in the chat window and all the Simulated Student's interventions. The third column indicates how many times the Simulated Student detected the situation. The fourth column called "students solved the problem" shows how many times the Simulated Student's intervention seemed to help the students to solve the exercise. The fifth column indicates how many times the students ignored the Simulated Student's proposal. The sixth column indicates how many times the Simulated Student intervened in order to help students. The
last column indicates how many times the Simulated Student acted unnecessarily, in other words, when the Simulated Student considered that its help was necessary although it was not.

By comparing Table 37 and 38 we can see that students had more problems in solving the exercises when they used the version with the Simulated Student than in the other case. This fact might be because the exercises were more difficult and longer in the version with the Simulated Students.

<table>
<thead>
<tr>
<th>Couple</th>
<th>Number of times that students needed help</th>
<th>Detected</th>
<th>Students solved the problem</th>
<th>Students ignored the help offered</th>
<th>SS interventions</th>
<th>SS intervened unnecessarily</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Sum</td>
<td><strong>65</strong></td>
<td><strong>65</strong></td>
<td><strong>61</strong></td>
<td><strong>4</strong></td>
<td><strong>69</strong></td>
<td><strong>4</strong></td>
</tr>
</tbody>
</table>

Table 38. Number of Times that Students Needed Help and SS's Interventions

The logs of the version without the Simulated Student showed that in 68% of the cases students found the solution at the third or fourth attempt, and most times they needed to consult the help offered by the system. However, in the rest of the cases the students, instead of reflecting upon the problem, started to talk about other topics. The following
conversation is a typical example of a conversation that took place when students did not find the solution after several attempts.

Student1: The clues indicate that our solution is wrong.
Student2: I have no idea what the answer is.
Student1: Are you from Ciudad Real?
Student2: No, I am from Puertollano and what about you?

Students wasted a lot of time talking about other topics. This might be the reason why the students solved less exercises even though the problems were easier than in the version with the Simulated Student.

Now the data obtained in the case that the students used the Simulated Student version are analysed. The results show that the Simulated Student always intervened when it was necessary (100% successful), the logs indicated that when students proposed a wrong solution the Simulated Student acted by suggesting a solution or asking a question related to the solution. The intervention of the Simulated Student helped students to solve the problem in 93.8% of the cases, 61 times out of 65. However students ignored the Simulated Student's advice 6.15% of the times, hence in these cases the Simulated Student intervention's was not efficient.

From these results it is possible to deduce that one intervention from the Simulated Student was enough to help the students to solve the problem. Table 38 shows more interventions (69) because of the 4 times that the Simulated Student intervened unnecessarily. In section 5.6 the possible reasons why the Simulated Student acted when it was not necessary will be analysed.

The data obtained from the experiment support the hypothesis that the Simulated Student helps students to solve problems, because although students did not know how to attack the problem in many situations, the Simulated Student's interventions helped students to find the solution. In fact, they solved more exercises correctly than when they used the version
without the Simulated Student. So, in an indirect way, the first hypothesis is again supported.

The data that we have just analysed reflect how the Simulated Student influences the way that students solve exercises. However, it would be interesting to know the students' opinion of the help offered by the Simulated Student. This is described in the next paragraphs. It is important to remember that students did not know that one of their partners was the Simulated Student. They thought that they were working with two human students.

5.5.4 Students' Assessment of the Assistance Offered by their Partners

When students filled in the questionnaire they were invited to give their opinion about their partners' assistance. Students could express their opinion using five categories (1. Not very useful, 2. A little useful, 3. Useful 4. Quite useful, 5. Very useful). Figure 14 and tables 39 show the subjects' opinion.

Figure 14. Students' assessment of the assistance offered by the Simulated Student
(1=Not very useful, 5=Very Useful)
Overall the help offered by the Simulated Student was considered useful (Mean=3.30) although, as Figure 14 indicates, there is a variety of opinions. The measures of central tendency (mean and median) describe the centre, middle, or most typical value in a sample. In this case the value of the mean is equal to the mode, 4. The percentiles are values above and below which a specified percentage of cases fall. For example, the 25\textsuperscript{th} percentile is the value below which 25% of the cases fall and above which 75% of the cases fall. For instance, in this case, the vote of 10\% of the students is below 1.50.

Students were also questioned about the help offered by the other partner, (the human student). Figure 15 testifies that students considered the help offered by the real student more useful. The mean in this case is 3.95. In both, the previous and the current cases the percentage of students who considered the help offered by their class-mates as quite useful...
is the same, 34.1%. The mean and the mode are again equal, and in addition to this they have the same value as in the previous case, therefore the central tendency is similar in both cases. The fact that the Simulated Student's assessment has a similar tendency to the human student's assessment suggests that students did not realise they were working with a Simulated Student.

![Pie Chart](image)

Figure 15. Students' assessment of assistance offered by the human student
(1=Not very useful, 5=very useful)

<table>
<thead>
<tr>
<th>N</th>
<th>Valid</th>
<th>44</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>3.95</td>
<td></td>
</tr>
<tr>
<td>Std. Error</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>4.00</td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Desv. tip.</td>
<td>.96</td>
<td></td>
</tr>
<tr>
<td>Variance</td>
<td>.93</td>
<td></td>
</tr>
<tr>
<td>Rank</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Maximum</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>174</td>
<td></td>
</tr>
</tbody>
</table>

Table 40 a. Frequency Statistics
5.5.5 Did the Simulated Student Detect Passive Students?

This section presents an evaluation of the role of the Simulated Student in detecting passive students. As was done in the previous section first of all we analysed with what frequency passive students arose in the version without the Simulated Student. Table 41 indicates each couple number, in the first row. The last row shows how many times passive behaviour was detected per couple.

Table 41 shows that students had passive behaviour nine times and in all cases except one the passive student repeated his/her behaviour, (we must remember that a student was considered passive when s/he did not propose a solution during three or more exercises, and his/her participation in the chat window were little). By studying the logs where passive students were detected it was discovered that in seven cases the passive students' partners were students who proposed correct solutions very often. Perhaps the passive student did not participate because s/he did not want to demonstrate that his/her level of knowledge was less or because it was more comfortable to do so.

In the other two cases the members of the group had a similar level of knowledge. However one student took part less in solving the exercises because of shyness or for other personal reasons.
Chapter 5: Evaluation

The partners who worked with a student who participated little often asked the passive student question such as: "are you still alive?" or "are you there?" Although students answered the questions they seldom improved their participation.

The study carried out to analyse the Simulated Student's efficiency in detecting and encouraging passive students was similar to that presented in the previous section. So, similar questions were studied:

- How many times did a student show passive behaviour?
- How many times did the Simulated Student detect it?
- How many times did the Simulated Student solve the problem?
- How many interventions by the Simulated Student were necessary to solve the problem?
- Did the Simulated Student act when there was no passive student?
- How many times did the passive student relapse?

The following table shows the results obtained after studying the logs. The first column indicates each pair's number. The second shows the number of times that students demonstrated passive behaviour. We considered that a student was passive when s/he did not propose a solution during two consecutive exercises and the student did not take part in the chat or his/her participation was very poor, for instance solely with sentences such as: ok, yes, no. The third column represents the number of times that the Simulated Student detected a passive student. The fourth column indicates how many times the Simulated Student intervened. The fifth column expresses how many times the Simulated Student intervened unnecessarily because the student was not in fact passive. The last column shows how many times the passive student repeated passive behaviour sometime later in that session.
Chapter 5: Evaluation

<table>
<thead>
<tr>
<th>Couple</th>
<th>Number of times that it occurred</th>
<th>Detected</th>
<th>The problem was solved</th>
<th>SS's. interventions</th>
<th>SS intervened unnecessarily</th>
<th>The passive student repeated his/her behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Sum</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 42. Results related to the passive students' behavior

The last row of Table 42 summarises the results. In this case one less (eight instead of nine) passive student situations was detected than in the case of using the version without the Simulated Student. This issue was not expected since we thought that working with three students it was easier for students to be passive than in the case of working just with another student. By comparing Table 41 and Table 42 we can observe that the passive students repeated the passive behaviour more often when they worked with the version without the Simulated Student, perhaps because they were not motivated enough or because the Simulated Student helped to avoid that students repeating a passive behaviour.

The data in Table 42 show that the Simulated Student always detected passive behaviour when it took place. On all occasions its intervention caused the passive student to take part in solving the exercises. In fact, from the logs we observed that after the Simulated
5.5.6 Students' Assessment of the Degree of their Partners' Participation

At this moment it might be opportune to know the students' opinion about the degree of participation of the Simulated Student. Moreover the goal of the Simulated Student was not to participate much but to improve the learning process, so it should have intervened principally when a factor that could have negatively influenced collaboration or learning arose.

In the questionnaire students were asked about their partners' degree of participation. The students could choose between five categories: 1. Not very participative, 2. A little participative, 3. Participative 4. Quite participative, 5. Very participative. The following graphic shows the students' opinion of the Simulated Student's participation. It is easy to check that the students' opinions were very divided.

Figure 16 indicates that 61.4% of the subjects thought that the Simulated Student was participative, quite or very participative; by contrast 38.6% considered that the Simulated Student was not very or only a little participative.
Figure 16. Students' assessment of the Simulated Student's participation
(1=not very participative, 5=very participative)

Subjects also said their opinion about the participation of the real learner. The following figure shows the percentage of the answers.

Figure 17. Students' assessment of the human student's participation
(2=little participative, 5=very participative)
Chapter 5: Evaluation

As was expected, the real student obtained better results. The Simulated Student was designed to intervene in concrete situations, therefore when the working process was unproblematic the interventions were less frequent. An advanced natural language process would allow the Simulated Student to be able to intervene with more frequency. However it would be convenient to check how it influenced the learning process and whether it would improve the results obtained.

5.5.7 Did the Simulated Student Detect Off-topic Conversations?

The third goal of the Simulated Student was to detect and avoid off-topic conversations. To evaluate the Simulated Student's efficiency in playing this role, the following questions were taken into account:

- Did the Simulated Student detect off-topic conversations?
- How many times did the Simulated Student avoid this situation?
- How many times did the Simulated Student intervene in order to stop the conversations?
- Did the Simulated Student act when students were not participating in off-topic conversations?

Each column of the following table answers one of these questions. As in the previous cases the first column indicates each couple number. The second column indicates the number of times that off-topic conversations took place, the third column shows how many times the Simulated Student detected it, the fourth column expresses how many times the problem was solved, the fifth column indicates the number of times that the Simulated Student intervened and the last column counts the number of times that the Simulated Student intervened unnecessarily.
Chapter 5: Evaluation

<table>
<thead>
<tr>
<th>Couple</th>
<th>Number of times that it occurred</th>
<th>Detected</th>
<th>The problem was solved</th>
<th>SS's interventions</th>
<th>SS intervened unnecessarily</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td><strong>22</strong></td>
<td><strong>22</strong></td>
<td><strong>22</strong></td>
<td><strong>22</strong></td>
<td><strong>22</strong></td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td><strong>14</strong></td>
<td><strong>12</strong></td>
<td><strong>11</strong></td>
<td><strong>13</strong></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>

Table 43. Results related to off-topic conversations

The last row shows that students had off-topic conversations fourteen times. The Simulated Student detected twelve of these situations and avoided them in eleven cases. So its interventions was successful in 91% of the cases. One intervention was enough to stop the off-topic conversations in each case. Table 43 shows that there were thirteen interventions, one more than the amount of situations detected. This is because of an unnecessary intervention. The results indicate that in the majority of the cases (85.7%) off-topic conversations were detected and stopped. This fact supports the third hypothesis (H3: By adding a Simulated Student to HabiPro the duration of off-topic conversations could be decreased).
Chapter 5: Evaluation

We also analysed how many times students had off-topic conversation when they used the version without the Simulated Student. Table 44 shows that double the amount of off-topic conversations occurred. A conversation was counted as off-topic when the conversation had four or more interventions and the length of the sentences was greater than 10 letters. Conversations smaller or with less letter did not affect the results. However longer conversations might distract students or cause them to waste time.

The reasons why there was double the amount of off-topic conversation could be because:

- As has been already explained, sometimes when student did not solve the exercises they started to talk about other topics.
- Nobody controlled whether off-topic conversations took place

The first row of Table 44 indicates each couple number, the second row shows the number of times that off-topic conversations were detected. The fact that in this case there were double the number of off-topic conversation than when students used the version with the Simulated Student supports Hypothesis 3.

<table>
<thead>
<tr>
<th>C</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>To.</th>
</tr>
</thead>
<tbody>
<tr>
<td>nu</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>28</td>
</tr>
</tbody>
</table>

Table 44. Number of Times of Off-topic Conversation. Version Without Simulated Student

5.5.8 Students’ Impressions

This section summarises the data obtained from the questionnaire. The first aspect that is presented is the students' general impression of HabiPro as a learning system. The second sub-section is centred on checking whether students felt that they had learnt: did the students believe that the system helped them to learn? And the last sub-section focuses on the importance of working in collaboration. Were the students aware that collaboration could improve their learning?

5.5.8.1 Students' Reaction Towards HabiPro

Subjects were asked to comment on their impressions of HabiPro as a method of learning. The system was evaluated in five categories: 1. Very bad, 2. Bad, 3 Normal, 4. Good 5.
Excellent, all these categories had scales from 1 to 5, with 5 being the highest mark. Figure 18 shows the results obtained as percentages.

![Diagram](image)

Figure 18. Students' reaction towards HabiPro

(2=bad, 5=excellent)

<table>
<thead>
<tr>
<th></th>
<th>Frequency</th>
<th>Percent</th>
<th>Valid Percent</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>2</td>
<td>1</td>
<td>2.3</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>9.1</td>
<td>11.4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>24</td>
<td>54.5</td>
<td>65.9</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>15</td>
<td>34.1</td>
<td>100.0</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td>100.0</td>
<td>100.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 45. Frequency of students' assessment towards HabiPro

As can be deduced from Figure 18 and Table 45, the majority of the students thought that HabiPro is a good method with which to learn. None of the students considered that it was a very bad method for learning, and only 11.4% students (5 students) said that HabiPro was normal or a bad learning method. In future work we want to check whether there is any relationship between the students' assessments and either the number of exercises solved or their opinion about their partners.
In the questionnaire there was another question related to HabiPro's assessment. The question was: "in general, do you think that using the system helped you to learn programming". The subjects had to choose between three answers:

- Faster than by reading a book and working by myself.
- Similar to reading a book and working by myself.
- Slower than reading a book and working by myself.

Forty students chose the first answer, three students the second and only one the third. This means that students positively evaluated the help offered by the system and by their partners and they thought that they could learn more by using HabiPro than by reading a book.

5.5.8.2 Awareness of Having Learnt

Humans frequently learn without being aware of doing so. For this reason asking the students about their awareness of having learnt could be a double-edged weapon. However we considered that it was important to ask this question. So we could discern whether students conceived HabiPro as a way of learning or as an entertainment. The question presented to the students was: "Do you think that you learnt by using the system?" Again the answer could be chosen from five categories: 1 Nothing, 2 A little, 3 Normal, 4. Quite a lot 5. A lot. All these categories had scales from 1 to 5, with 5 being the highest mark. Table 46 of frequencies and Figure 19 show the results obtained.

<table>
<thead>
<tr>
<th>Valid</th>
<th>Frequency</th>
<th>Percent</th>
<th>Valid Percent</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2.3</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>27.3</td>
<td>27.3</td>
<td>31.8</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>47.7</td>
<td>47.7</td>
<td>79.5</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>20.5</td>
<td>20.5</td>
<td>100.0</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td>100.0</td>
<td>100.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 46. Frequency Related to the Awareness of Having Learnt
Figure 19. Awareness of having learnt
(1=nothing learnt, 5=a lot learnt)

Only two students (4.6%) answered that they had learnt a little or nothing. Thirty students (68.2%) considered that they had learnt quite a lot or a lot, and the rest thought neither one thing nor the other. The median and mode were calculated and both have the same value: 4. This means that the central tendency is 4. The next step would be to verify whether students had really learnt.

5.5.8.3 Did the students learn?

Students enrolled in the first course of Programming sat a partial exam which contained ten programming exercises. Seven of them were similar to HabiPro exercises. The exam was worth ten points. We compared the marks obtained by the students who used HabiPro to the marks of students who had not worked with HabiPro in order to see whether there were differences.

Thirty programming exams were randomly chosen from students who had worked with HabiPro and thirty from those who had not done so. We were aware that the results of this evaluation were not very significant for two reasons:

- Not all sixty students had the same practical or theory teachers. The fact of having a particular teacher might positively or negatively influence the results.
Chapter 5: Evaluation

- Students who used HabiPro were more used to solving that kind of exercises than the rest. This fact might affect the results positively.

Nonetheless, the data were analysed in order to see its tendencies. The results obtained will be checked with a new experiment where the influence of previous factors are avoid as far as possible.

Normality tests were carried out in order to see which statistic to use to compare the means:

<table>
<thead>
<tr>
<th>Number of exercises solved</th>
<th>Kolmogorov-Smirnov Statistic</th>
<th>df</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.098</td>
<td>60</td>
<td>.200</td>
</tr>
</tbody>
</table>

Table 47. Normality Test for whole the sample

<table>
<thead>
<tr>
<th>Did the students use HabiPro?</th>
<th>Kolmogorov-Smirnov</th>
<th>Shapiro-Wilk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Statistic</td>
<td>df</td>
</tr>
<tr>
<td>No</td>
<td>.137</td>
<td>30</td>
</tr>
<tr>
<td>Yes</td>
<td>.133</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 48. Normality Test for each sample

The normality tests indicate that the distribution is normal in total as well as in the sub-samples that have used HabiPro or not. The following figure shows the mean mark that students obtained.

<table>
<thead>
<tr>
<th>Did the students use HabiPro?</th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of exercises solved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>30</td>
<td>5.79167</td>
<td>2.03640</td>
<td>.37179</td>
</tr>
<tr>
<td>No</td>
<td>30</td>
<td>3.84167</td>
<td>1.24306</td>
<td>.22695</td>
</tr>
</tbody>
</table>

Table 49. Mean number of exercises correctly solved by each sample

By evaluating the means we can see that students that used HabiPro obtained better marks than students who did not work with it. The question is: is it possible to extend the
population to these results? Before answering this question the results of the T-Student statistic must be analysed.

<table>
<thead>
<tr>
<th>Number of exercises solved</th>
<th>Levene’s Test for Equality of Variance</th>
<th>t-test for Equality of Means</th>
<th>95% Confidence Interval of the Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal variances assumed</td>
<td>F</td>
<td>Sig.</td>
<td>t</td>
</tr>
<tr>
<td></td>
<td>8.727</td>
<td>.005</td>
<td>4.477</td>
</tr>
<tr>
<td>Equal variances not assumed</td>
<td>4.477</td>
<td>47.977</td>
<td>.000</td>
</tr>
</tbody>
</table>

Table 50. Independent Samples Test

The Levene statistic indicates with a level of significance of 0.005 that the variances are not equal. Therefore the value T=4.477, with a p=0 indicates that the means are different and higher in the case of using HabiPro. Therefore, we can conclude that the data support the hypothesis that students learn using HabiPro. However these results have to be verified in another experiment where the students have the same teachers and the conditions as similar as possible.

5.5.8.4 Alone or in a Group?

It is a fact that not everybody likes to work in a group. Some people prefer working or studying alone. We are aware of this reality and for this reason we thought it convenient to add to the questionnaire a question that indicated whether students considered that working with more students could improve their learning. So, they were asked: "Do you think that you would have learnt more alone?". This time there were only two possible answers: Yes or No. Figure 20 displays the answers.
Figure 20. Results to the question: "Do you think that you would have learnt more alone?"

The figure speaks for itself. Most of students believed that working with a partner helped them to learn. This is an important issue that testifies that students valued the help offered by their partners (even though one of them is a Simulated Student).

5.6 Discussion

In the following sections the results obtained from the experiment are commented on. The principal topics dealt with are the influence of the Simulated Student on the students, and the Simulated Student's efficiency in carrying out its three functions.

5.6.1 Simulated Student's Influence

As was described in section 5.5.1 the data supports the hypothesis that students tend to solve more exercises correctly when they use the version with the Simulated Student than when they use the version without it. However the experiment showed something that was not expected: The students who used the version with the Simulated Student in the first version solved more exercises in the second session, with the version without the Simulated Student, than the students that used the version without the Simulated Student in the first version. The reasons why this happened may be because the Simulated Student taught students to work properly in a group, since it encouraged passive students to work, stopped
Chapter 5: Evaluation

off-topic conversations and sometimes advised students to see the clues or examples when they had problems in solving the exercises. Perhaps, students learnt a correct way of working with HabiPro and although they did not have the Simulated Student in the second session they remembered, for example, that was useful to have a look at the clues or examples when they did not know something. This is not a proved issue, just a supposition. This supposition will be tested in further experiments.

Another result obtained was that students solved more exercises correctly in an hour using HabiPro than in the pre-test. The data indicates that students solved an average of 5 exercises correctly. However, with HabiPro students solved 9.7 in the case of using the version without the Simulated Student and 11.8 in the case of using the Simulated Student. The results show that HabiPro improves the student's performance even without the Simulated Student.

Although we would have expected some improvement after the pre-test because of learning, the difference of means between solving the exercises by hand or with HabiPro is very great, almost double. This fact led us to think about two variables that could influence the results.

- Students solved the pre-test individually.
- When students solved the pre-test they had never solved exercises of this type.

As was explained when we described the design of the investigation, one of the issues of validity of this kind of experiment is the learning effect. Perhaps although the exercises of HabiPro were different to the pre-test ones, this effect could have taken place and helped students to solved more exercises when they used HabiPro.

5.6.2 The Simulated Student's Role in Helping to Solve the Exercises

From the experiment it was observed that the Simulated Student always intervened when students could not solve the exercises. An example of the Simulated Student's intervention is shown in the following conversation. In this case the Simulated Student is proposing a
solution. It does not impose its idea, leaving the students free to check the proposal or to ignore it. The Simulated Student is Student3

Student1: I don't know how the "for" works.
Student2: Yes, I see that, we have tried a lot of possible solutions and none of them are correct.
Student3: I think that the index of the array must be 0. Let's try \( j = 0 \).
Student1: Yes!!! Now I remember that the index of an array starts with 0 in Java.

On the other hand, although the Simulated Student intervened when it was necessary, it also acted four times when it was unnecessary. When and why did this occur? The Simulated Student has no natural language capability hence it cannot understand the chat conversation. The Simulated Student uses the information from the answer windows and the number of times that students check a solution in order to decide when to act. So it may occur that the Simulated Student proposes a solution that the other student has just written in the chat window. This would be an unnecessary intervention and this is what in fact happened on the four occasions that the agent intervened inadequately. An example of this situation is shown by the following conversation:

Student1: The solution that we have tried doesn't work.
Student2: I think that the semicolon that is in the first line is not necessary.
Student3: I have found the mistake!!! It is the semicolon in the first line.

Students might think that the Student3's behaviour was strange because it proposed the same idea that had already been mentioned in another way. However, in most cases students thought that Student3 wrote the sentence in the chat window at the same time as Student2, but Student2's intervention arrived earlier. Everybody who has used a chat application connecting two or more people at the same time knows that such a chat conversation is not as logical as an oral conversation, since, except in the applications that use a turn taking protocol, chat users are not aware whether the others are writing at the same time, and neither of them knows in which order interventions will arrive. So, in this
case the expectations of working in a chat helped us to mask a possible defect of the Simulated Student. However, we have to admit that in one of the cases where the Simulated Student suggested something that had already been proposed, one of the students answered:

Student2: Student3, what are you thinking of? I have already proposed that solution.

Of course the Simulated Student did not answer, and so perhaps a possible off-topic conversation was avoided, since students continued solving the exercises normally.

Table 37 also showed that on four occasions students ignored the Simulated Student's proposal. As was previously explained, students are not obliged to obey the Simulated Student and they can choose between checking what it proposes or not doing so. In future work we investigate whether there is some relationship between the manner in which the Simulated Student proposes a solution and the probability that students accept or refuse it. The fact that the four times that suggestions were ignored occurred in different exercises might indicate that there is no particular relationship between the two factors.

5.6.3 The Simulated Student's Role in Detecting Passive Students

From the results obtained and described in section 5.5.4 it can be seen that there were eight cases of passive students. All of them were detected and the Simulated Student's intervention caused the passive student to participate in all the cases, although one student repeated his/her passive behaviour.

The Simulated Student intervened once unnecessarily. The log of the conversation was analysed in order to find out why this occurred. The log indicated that the student who was considered passive did not write anything in his answer window for two exercises and the density of interaction in the chat was low. However his interventions in the chat were useful, although not very long. Due to the short density of interventions and to the small amount of participation in the answer window the Simulated Student considered that was passive. Part of the conversation is presented:
Chapter 5: Evaluation

Student2: What is wrong in the problem?
Student1:j<5;
Student2: ok, I am going to try it. (Student2 writes j<5 in the answer window)
Student1: ok
Student2: Student1, you were right.
Student1: Thanks
Student3: Propose something Student1!, you normally hit the solution.

The conversation shows that the Student1 participated in the chat window. It was even he who proposed the solution, but his interventions were very short. This fact joined to the fact that Student2 wrote the solution in the answer window led the Simulated Student to think the Student1 was a passive student.

5.6. 4 The Simulated Student's Role in Detecting Off-topic Conversations

In both previous cases the number of times that a circumstance occurred was equal to the number of times that it was detected. This means that the Simulated Student was a 100% successful. However in off-topic conversations incident 85.7% of cases were detected. Analysing the logs it was observed that the cases not detected were those where the students' interventions were very short. For instance

Student1: Another exercise!!
Student2: I am tired
Student1: It is bad time
Student2: Yes, siesta time
Student1: Is this the eleventh one?
Student2: I think so

The Simulated Student was not programmed to check sentences formed of less than five words. It is not very usual for off-topic conversations to have just four of five words, besides this does not mean a great waste of time. The Simulated Student acted as was arranged. If it were necessary to avoid this situation it would be easy by just adjusting the
number of words or checking all the interventions without taking into account the length of the sentence.

The Simulated Student's intervention was useful in 91% of the cases (11 over 12). The example below shows how the Simulated Student acted in order to stop conversations.

Student2: Who are you?
Student1: I am Ana and you?
Student2: My name is Jose and what is your name Student3?
Student3: Let's finish the exercises and we can talk later
Student3: I think that one variable is not declared.
Student2: Yes, the "X" is not declared!!

The Simulated Student detected that students were talking about other topics and by proposing a solution tried to divert the students' attention back towards the exercises.

In one case the off-topic conversation problem was not solved since students ignored the Simulated Student's intervention and continuing talking about other topics. However the fact that the Simulated Student did not intervene again indicates that the duration of the conversation was not very long. If it had lasted longer the Simulated Student would have intervened a second time. It would have been interesting to see what would have occurred after a second intervention.

In the case of off-topic conversations there was an unnecessary intervention. The reason for this was that although students were talking about the problem they did not use words related to Java or solving the exercises and they sometimes used slang expressions. Because of this none of the words was found in the Simulated Student's database. This fact led the Simulated Student to think that an off-topic conversation was taking place. The following conversation and the Simulated Student's unfortunate intervention are shown.
Chapter 5: Evaluation

Student1: This looks really hard
Student2: I hadn't got a clue about the last one.
Student2: But, d'you know what they're asking?
Student1: Yeh, something like the last one
Student3: Stop the chat and let's think a little!!

Students had to face a new exercise and Student2 did not understand what the exercise was asking for. Student3 did not find keywords so it decided to act. If the Simulated Student had been able to process natural language this intervention would probably not have occurred.

5.7 Limitations of the Experiment

The experiment described in this chapter and the methods used in order to evaluate the Simulated Student and HabiPro might have several weaknesses. In this section the influence theses weaknesses on the results are analysed.

- By chance the exercises in the Simulated Student version were harder than the exercises in the version without the Simulated Student. This fact, depending on one's point of view, might be positive or negative. It is positive because even though students had to solve more difficult exercises they solved more than in the case of working without the Simulated Student. Thus supporting the hypothesis that the Simulated Student helps students to solve the exercises. On the other hand, perhaps if the exercises had been easier the Simulated Student would have participated less and in consequence it would have been less useful.

- The logs were analysed by the same person who designed HabiPro. There was not an independent analysis to judge the effectiveness of the Simulated Student's interventions. It is known that it is not suitable that the same group that implements a system evaluates it since might be more difficult to detect mistakes. In the same way we must take into account the possibility that the logs were subconsciously not analysed objectively.
Chapter 5: Evaluation

- The groups that were analysed in order to check learning gain did not all have the same teachers, thus a future experiment will have to be carried out in order to check whether the result obtained are valid.
- Students usually respond to questionnaires in the way they think the experimenter wants to hear. We tried to avoid this fact by asking students not to write their names on the questionnaires. People are usually more sincere when they do not need to give their name.

5.8 Summary

This chapter has presented and discussed the results obtained in the experiment to test the Simulated Student in HabiPro. The three alternative hypotheses were supported by the data and a new issue was found: the Simulated Student teaches students to work with HabiPro and improves the students' performance even though they work with the version without the Simulated Student.

The data obtained from the students' questionnaire showed a positive assessment towards the assistance and participation of their work-mates (even the Simulated Student), and towards HabiPro as a learning method.
This chapter analyses whether the goals introduced in the first chapter have been achieved. The chapter also describes the principal contributions of this thesis and the main papers that have been published which are related to this investigation. Finally, future work is described.

6.1 Goals Researched

In the first chapter the goals and sub-goals of this thesis were described. By achieving the sub-goals the main goal must be also achieved. The main goal was summarised as the following hypothesis:

- Goal 1. The study of the role of the computer in educational environments, specifically in collaborative learning circumstances.
  The literature related to educational computer science and especially to collaborative learning with and without computers has been surveyed. From this analysis it was observed that the field of CSCL presents some challenges (see chapter 2), for instance, how to control students' behaviour in order to maximise the advantages of working in collaboration. Several research groups are trying to control circumstances that harm collaborative learning. However there is still a lot of work to do in this field.

- Goal 2. The design of a model for a Simulated Student which would control and avoid negative situations, thus augmenting the benefits of collaborative learning.
  In order to deal with some inadequacies of CSCL we designed a Simulated Student in charge of monitoring students' behaviour and acting when situations that could be detrimental to collaboration or learning take place.
• **Goal 3:** The search for a topic to teach that would be suitable for collaborative learning.

The Simulated Student had to be implemented in a specific collaborative system to check whether it detected the problems and could solve them. Therefore the question of what domain was convenient for the collaborative system arose.

One of the topics of interest for our research group is how to help students to learn procedural disciplines. Procedural disciplines are frequently more difficult to learn than declarative ones since they include both a conceptual part and skill development. We had the opportunity of checking whether collaborative learning was a good method to facilitate procedural learning.

Different domains were considered and programming was the discipline chosen. Several reasons led us to this choice, two are mentioned here: Programming is a topic difficult to learn and even to teach. Also, novice students like programming with other people so it is often easier to debug programs and develop programming strategies collaboratively.

• **Goal 4:** The study of the psychology of programming, and comparison and analysis of different systems to learn programming in order to detect their advantages and deficiencies.

Once it was decided that the discipline to teach would be programming an extensive investigation of the learning process that programming involves and how this procedural discipline must be taught was necessary. This investigation indicated that important tasks related to learning programming were seldom explicitly taught.

Researching and studying the systems that have already been developed to teach programming was also a vital task to carry out. This helped us to develop a different focus on teaching programming. It was surprising to discover that almost none of the systems offered collaborative techniques.
• Goal 5. The design and implementation of a collaborative system to develop good habits in programming.

In order to help students to learn programming, and more concretely to practice programming skills, HabPro, a collaborative, distributed, and synchronous system was developed. The system was designed with the goal of students principally practising: debugging, program comprehension and style of programming. As was explained in chapter 3, these three tasks are rarely explicitly taught to novice students.

• Goal 6. Implementation of the model in the collaborative system developed previously.

A Simulated Student was implemented in HabiPro, following the model designed. The modularization of the model facilitated the implementation process.

• Goal 7. The study of the different types of empirical validation.

The empirical validation of the Simulated Student and of HabPro were necessary in order to check their usefulness in practice. In order to minimum the effects of the different characteristics of the population a within-subject experiment was chosen. Data was collected which supports the effectiveness of the Simulated Student.

Main Goal: The design and implementation of a Simulated Student for collaborative learning systems in order to detect and avoid negative situations that hamper collaborative learning.

Since the sub-goals have been achieved, it is possible to consider that the main goal is also achieved. So we can say that the initial hypothesis

may be answered affirmatively.

165
6.2 Contributions

This thesis has proposed a way of solving some challenges that CSCL faces. Concretely, this work suggests the use of a Simulated Student to monitor the students' behaviour in collaborative learning systems. This thesis is not just a proposal, it also describes the design of a model for the Simulated Student and the roles that it could play. A Simulated Student was implemented based on the model designed. The Simulated Student has been added to a collaborative, distributed and synchronous application, HabiPro, implemented to develop good programming habits. The usefulness of the Simulated Student and of HabiPro has been evaluated. The main contributions of this thesis are:

The model of a Simulated Student to avoid negative Situations. The main features of the model are: the Simulated Student is considered as one more student in the group and for this reason it uses the same communication methods as the real students. Secondly, the Simulated Student monitors factors such as the frequency, type and density of interactions to measure participation as well frequency of proposals corrected by students and students' mistakes in order to obtain a model of the students' level of knowledge.

- The verification, using a real system with real students, that collaboration with a Simulated Student may be beneficial towards the students' learning because:
  1. The Simulated Student can detect when off-topic conversations take place and prevent that these conversations lasting too long so that students do not waste their time on them.
  2. The Simulated Student can detect passive behaviour and encourage the passive student to participate.
  3. The Simulated Student can assist students to reflect upon the problems, thus solving more exercises in the same time.
  4. The Simulated Student can teach or accustom the students to work appropriately.
  5. Students reflect upon the Simulated Student's proposals avoiding the situation that sometimes occurs when students simply accept a solution uncritically just because it comes from a teacher.
• HabiPro, a collaborative, distributed and synchronous system to develop good programming habits (debugging, programming comprehension and programming style). The system focuses on programming aspects rarely dealt with in previous programming systems and even in programming courses. HabiPro, also supports and encourages collaborative work.

6.3 Results Published

Partial results obtained during the investigation have been presented at several international congresses, some of them are described below.


This paper describes situations that hamper collaborative learning effects and proposes a model for a Simulated Student.


What the advantages of using a Simulated Student in CSCL environments are.


The paper describes the tasks of a Simulated Student to guide students' learning process in a system designed to teach a procedural discipline (programming).

An experiment performed with HabiPro in order to check its usefulness was described in this paper.


This paper compares HabiPro with previous systems designed to teach programming and describes HabiPro's features.


This paper describes HabiPro's architecture, features and use.


This paper presents a method that, depending on students' arguments, decides which pairs must work together in a collaborative system.

International Conference on Artificial Intelligence in Education *AIED 1999*, Le Mans, France, July, pp 81-83.

This paper was a presentation of the thesis plan and its goals.


The paper describes the first version of HabiPro.


This paper describes the advantages of using student models in collaborative learning systems.


This paper describes a student and group model for collaborative applications.


This paper presents the first student model and group model that we designed for collaborative systems.

### 6.4 Future Work

The experiment described in the previous chapter focused on testing the efficiency of the Simulated Student when playing its three main roles: detecting passive students, detecting off-topic conversations and helping students to solve programming exercises. The marks
obtained in an exam helped us to study the influence that HabiPro may have had on the students, though there was the confounding factor of their being taught by different teachers. The next experiment will focus on checking more systematically what students have really learnt and whether students in fact develop good programming habits as a result of using HabiPro.

The results obtained from the experiment performed will be analysed in order to see whether there are further relationships, such as between the numbers of exercises solved and the students' assessment of HabiPro. The analysis reported here concentrated on the performance of HabiPro rather than how, and how well, students learned the particular programming skills needed to solve the exercises provided. The logs stored will used to examine the kinds of mistakes that the students made in each of the different sort of exercises and the type of debugging strategies they used, and whether these changed over the course of the interaction.

HabiPro will be improved by adding the facility for students to compile programs, so they will be able to check whether their programs work properly, and what it is more important, they will become more familiar with compiler messages.

HabiPro will be tested with other procedural disciplines, such as mathematics, in order to see how the domain chosen influences the results, and how easy it is to adapt the system to a new domain. To convert HabiPro to a new domain the database which contains the exercise should be modified, so for each new exercise introduced its solution/s, new clues and examples should also be added. To adapt the Simulated Student two things would principally be modified. The first would be to add words related to the new domain to the databases used to detect off-topic conversations. It would also be necessary to change the help offered by the Simulated Student, adapting it to the new subject. The sentences that the Simulated Student says when it detects off-topic conversation or passive students would not have to be changed and neither would how the Simulated Student detects off-topic conversations, passive students or lost students.
The data obtained from the experiment indicated some situations where the Simulated Student acted unnecessary or could not amend the situation. For example, the Simulated Student did not detect one off-topic conversation because it consisted of very short interventions. The Simulated Student will be improved in order to correct the problems detected. So, all the interventions will be checked independently of the length of the sentence. More words will be added to the Simulated Student's database in order to widen the possibilities of detecting off-topic conversations.

To prevent the Simulated Student from repeating something that has already been said we are exploring the use of techniques developed in the field of natural language processing. We have discussed this issue with Miguel Angel Ibarra, a doctoral student working in this field at CICESE, a Mexican research centre, who is in the process of analysing the conversations obtained from our experiments, in order to adapt the syntactic analyser for Spanish that he has developed to help the Simulated Student achieve a better understanding of the conversations (Ibarra, Favela and López, 2000).

The Queensland University of Technology in Brisbane, Australia and several Argentinean universities are interested in using HabiPro in their universities. It will be very useful to check the influence of the Simulated Student on subjects belonging to other cultures. Of course, the databases that the Simulated Student uses to detect off-topic conversation must be modified in order to make them adequate for new environments.

There is still a lot of work to do and a long road to travel in order to develop a perfect Simulated Student. However the most important thing is to take the first step.


Bennett, and H. F. Freidrich (Eds.), *Learning and Instruction: European Research in an 

How can an interactive learning environment be designed? *European Journal of 
Psychology of Education*, 7 (4), 325-337.

Press.

228, 456-462.

Learning from Hypertext. In *Computer Supported Collaborative Learning*. O'Malley (Ed.). 
Heidelberg: Springer-Verlag.

Supercomputing Applcation.


References


design of programming support tools, debugging aids and tutorial environments. In Y.
Science B.V.

effects of expertise and familiarity on the classification of object-oriented code. Human-
Computer Interaction, 10 (2and-3), 227-248.


Dembo, M.H., and McAuliffe, T. J. (1987). Effects of perceived ability and grade status on
social interaction and influence in cooperative groups. Journal of Education Psychology,
79, 415-423.

Détienne, F. (1997). Assessing the cognitive consequences of the object-oriented approach:
A survey of empirical research on object-oriented design by individuals and teams.
Interacting with Computers, 9, 47-72.


In Computer Supported Collaborative Learning. O'Malley (Ed.). Heidelberg: Springer-
Verlag


on Collaborative Learning. In P. Reiman and H. Spada (Eds.), Learning in Humans and
Machines: Towards an Interdisciplinary Learning Science (pp. 189-211). Oxford: Pergamon.


References


References


APPENDIX 1. Simulated Student’s Interventions

Sentences

Number 1
FirstMistake what does this sign mean?
SecondMistake perhaps its (&&) instead of || Otherwise the exercise always displays 'error', won't it?
Advanced passiveness you're very quiet. Say something
Medium passiveness do you understand this exercise?
Off-topic Let's solve all the exercises, ok?
Begin Exercise shall we start?
CorrectFirstExercise are you on exercise 2?

Number 2
FirstMistake the index of an array begins with 0, doesn't it?
SecondMistake shall we try j=0?
Advanced passiveness are you there? You're not saying anything
Medium passiveness we're going very quickly, why don't you propose a solution
Off-topic this doesn't seem very difficult
Begin Exercise do you understand the arrays?
CorrectFirstExercise we're made a good start!!!

Number 3
FirstMistake shall we have a look at the clues?
SecondMistake I've got it!!! The mistakes is the ";," at the end of the second sentence. Check
Advanced passiveness you're very quiet, say something
Medium passiveness why don't you propose a solution say something
Off-topic let's try finish this first, and then we can talk
Begin Exercise I can't see any errors. What about you?
CorrectFirstExercise this is going well
APPENDIX I. Simulated Student’s Interventions

Number
FirstMistake I don’t agree with j=0 because otherwise “primavera” wouldn’t appear.
SecondMistake How about trying j>=0
Advanced passiveness you tend to have good ideas. Why don’t you propose a solution?
Medium passiveness you could propose a solution too if you like
Off-topic we’ll talk about that later. Let’s try j>=0, ok?
Begin Exercise you have to indicate when to stop the loop, don’t you?
CorrectFirstExercise that was easy!!

Number
FirstMistake Let’s have a look at the example. Maybe it’ll help
SecondMistake I think it must be similar to what I’m sending you. Shall we check to make sure!!
Advanced passiveness what do you think, pal?
Medium passiveness if you have an idea tell us, ok?
Off-topic I think you have to put new int[7], what do you think?
Begin Exercise we might have to put “new” or something like that. I don’t know
CorrectFirstExercise why was it 7 and not 6? I thought the index started at nought

Number
FirstMistake the output of the method is an integer, isn’t it?
SecondMistake check mine.
Advanced passiveness why don’t you propose something?
Medium passiveness if I’m going too quickly tell me. This computer is really fast.
Off-topic I’ve got 5 right. Let’s have a think about this one and finish as soon as possible
Begin Exercise what do you think is missing?
CorrectFirstExercise another one!!!
APPENDIX I. Simulated Student’s Interventions

Number 7
FirstMistake and it seemed to easy!!
SecondMistake this program calculates the mean of 2 numbers. Shall we try with the mean
Advanced passiveness are you thinking?
Medium passiveness propose something, mate, don’t be shy ;)
Off-topic it calculated the mean doesn’t it?
Begin Exercise this exercise seems different from the rest
CorrectFirstExercise we’re the greatest ;))

Number 8
FirstMistake I think we should use ”j” instead of ”k”
SecondMistake shall we try my answer?
Advanced passiveness are you still there?
Medium passiveness what do you think?
Off-topic is ”j<5” the same as putting ”j<=4”? 
Begin Exercise which variable do you have to use i or j?
CorrectFirstExercise that was easy, wasn’t it?

Number 9
FirstMistake shall we look at the example or the clues?
SecondMistake I think it’s the reply. I sent you
Advanced passiveness have a go! You almost always get the right solution
Medium passiveness you’re allowed to make a suggestion if you want!!
Off-topic there can’t be many left. Let’s finish them!!!
Begin Exercise this computer is really fast. I’m on exercise 9, what about you?
CorrectFirstExercise we’ve already done 9 exercises!!!
APPENDIX 1. Simulated Student’s Interventions

Number 10
FirstMistake I think the mistake is in the "procesar" method the others are very easy and seem to be right. What do you think?
SecondMistake that's it!!! In the "procesar" method it puts "resta" instead of "restar"
Advanced passiveness say something, have a go
Medium passiveness how's it going? You don't say much!!
Off-topic why does it put "restar" in one place and resta in another?
Begin Exercise what's the mistake here?
CorrectFirstExercise Great!! I'd never have noticed that mistake!!!

Number 11
FirstMistake I don't agree
SecondMistake maybe there is a ";" missing in the "return suma" sentence
Advanced passiveness carry on making suggestions. You seem to know what you're doing.
Medium passiveness did you understand the last one. You seem a bit confused.
Off-topic the bloke next to me is on number 15. Let's go a bit faster. I don't want him to finish first.
Begin Exercise this exercise reminds me of one we did in the practical class.
CorrectFirstExercise we're experts now!!

Number 12
FirstMistake it's that genius again
SecondMistake check my reply
Advanced passiveness don't go to sleep now the exercise are more difficult
Medium passiveness come on mate!! Propose something. I'm half asleep
Off-topic the output is " El 3 es cierto"?
Begin Exercise do they want the program output for N=3?
CorrectFirstExercise ;)

199
APPENDIX 1. Simulated Student's Interventions

Number 13
FirstMistake I think it's this. " el 13"
SecondMistake it's the 13
Advanced passiveness how good are you at these exercises?
Medium passiveness don't you like these exercises? You don't seem to be taking part much
Off-topic we've only got to put 13, haven't we?
Begin Exercise it's another style exercise
CorrectFirstExercise how short!!!
APPENDIX 1. Simulated Student’s Interventions

Number 16
FirstMistake I think the solution is 5. Shall we try that?
SecondMistake Let’s try just writing 5
Advanced passiveness what do you think?
Medium passiveness send what you write to the answer window, because I’ve hardly received anything from you
Off-topic The teacher’s behind me. Carry on with the exercises
Begin Exercise the last pair is different, isn’t it?
CorrectFirstExercise that was really easy

Number 17
FirstMistake I think you have to put ‘void’ or something like that at the beginning of the ‘palindromo’ method
SecondMistake check mine. There’s not much time left
Advanced passiveness say something. I haven’t received anything from you
Medium passiveness why don’t you write a solution in the window?
Off-topic I think I’ve found the mistake, there’s a ”;” missing in the ”if” line
Begin Exercise what does charAt do?
CorrectFirstExercise I’m enjoying this

Number 18
FirstMistake what does ”!par” mean? I don’t understand boolean variables
SecondMistake It can’t be only ”par”?
Advanced passiveness do you like this kind of exercise? I don’t much
Medium passiveness is something wrong?
Off-topic one more and we’ll have done them all. We’re the greatest!! ;)
Begin Exercise I’m tired
CorrectFirstExercise brilliant!!!
## APPENDIX 1. Simulated Student's Interventions

<table>
<thead>
<tr>
<th>Number</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FirstMistake</strong></td>
<td>is this the kind of loop that you always have to do?</td>
</tr>
<tr>
<td><strong>SecondMistake</strong></td>
<td>shall we try 32, because &quot;num&quot; has decreased by one value after passing for the while loop while</td>
</tr>
<tr>
<td><strong>Advanced passiveness</strong></td>
<td>clever clogs!! Say something about the last exercise!!</td>
</tr>
<tr>
<td><strong>Medium passiveness</strong></td>
<td>you haven't said much. Why?</td>
</tr>
<tr>
<td><strong>Off-topic</strong></td>
<td>Let's do the last exercise</td>
</tr>
<tr>
<td><strong>Begin Exercise</strong></td>
<td>it could be output 3</td>
</tr>
<tr>
<td><strong>CorrectFirstExercise</strong></td>
<td>good work, everyone!!!!</td>
</tr>
</tbody>
</table>
Examples of Simulated Student's Interventions

This section displays (in English) three types of interventions: one which tries to avoid off-topic conversations, another which tries to help student to solve the exercises and the last one which encourages a passive student to participate.

![Image of a program and chat]

Figure 8. Simulated Student's intervention in trying to avoid off-topic conversations

In this case the Simulated Student acted twice in order to stop the off-topic conversations, but it finally, made that students talk about the exercises.

Figure 9 indicates a situation where students were having problems in finding the mistake in the program. In this case the Simulated Student proposed a solution. This helped the student called Juan to remember that the array must start with 0.

In Figure 10 the Simulated Student acted with the goal of encouraging the participation of a student that did not take part in solving the exercises.
APPENDIX 1. Simulated Student's Interventions

Figure 9. Simulated Student's intervention in trying to help students to solve the problem.

Figure 10: Simulated Student's intervention in trying to encourage the participation of a passive student.
**AP2.1 Version Simulated Student: Exercises**

**Problem**

Correct the mistake in the program:

```java
static void leerOpcion() throws IOException {
    char character;
    System.out.print("Introduce a or b");
    character = (stdin.readLine()).charAt(0);
    while (character != 'a' || character != 'b') {
        System.out.print("Error. Try again");
        character = (stdin.readLine()).charAt(0);
    }
}
```

**Answer1**

```java
}
```

**Answer2**

```java
 while (caracter != 'a' || caracter != 'b') {
```

**Clue1**

A double condition join by `||` is true when one of the two subconditions is true.

**Clue2**

If the user introduces an "a", the program will display an error message.

**Clue3**

**Clue4**

**Example**

```java
static void leerOpcion{
    int numero;
    System.out.print("Introduce a number from 1 to 3");
    numero = Integer.parseInt(stdin.readLine());
    while (numero != 1 & numero != 2 & numero != 3) {
        System.out.print("Error. Try again");
        numero = Integer.parseInt(stdin.readLine());
    }
}
```

```java
} // leerOpcion
```
Problem
Correct the mistake in the following program

```java
class Exercise2 {
    public static void main(String args[]) {
        int numeros[] = {0,2,4,6,8};
        int j;
        for(j=1; j<5; j+=2) {
            System.out.print(numeros[j]);
        } // for
    } // end main
} // end class
```

The program output is 048

**Answer1**
j=0

**Answer2**
for(j=0; j<5; j+=2) {

**Clue1**
Can you remember what value an array index begins with?

**Clue2**
The array index begins with 0.

**Clue3**

**Clue4**

Example
```
class Exercise {
    public static void main(String args[]) {
        int numeros[] = {0,2,4,6,8};
        int j;
        for(j=0; j<5; j++) {
            System.out.print(numeros[j]);
        } // for
    } // end main
} // end class
```

The program output is 02468
Problem

Find the mistake in this program

class Exercise3 {
    public static void main(String args[]) {
        boolean b;
        b = false;
        System.out.println("b es " + b);
        b = true;
        if (b) System.out.println("Esta sentencia no se ejecuta.");
        System.out.println("10>9 es " + (10 > 9));
    } // end main
} end class

Answer1
;

Answer2
public static void main(String args[])

Clue1
The error message that the compiler shows is: "the body of the program cannot be found"

Clue2
The body of a program must be between "{" "}"

Clue3
Perhaps the compiler cannot find the first "{" at the beginning of the program because it found another symbol before.

Clue4

Example
Problem

Complete the following program

class Exercise4 {
    public static void main(String args[]) {
        String estaciones[] = {"primavera", "verano", "otoño", "invierno"};
        int j = 3;
        while(................){
            System.out.println(estaciones[j]);
            j--;
        }  //end while
    }  //end main
}  //end class

The output is:
invierno
otoño
verano
primavera

Answer1

j>=0

Answer2

j>-1

Clue1

The while loop executes the body of the loop repeatedly as long as the condition is true.

Clue2

The loop must stop when the index array reaches 0.

Clue3

Clue4

Example

class Exercise {
    public static void main(String args[]) {
        String estaciones[] = {"primavera", "verano", "otoño", "invierno"};
        int j = 0;
        while(j<=3){
            System.out.println(estaciones[j]);
            j++;
        }  //end while
    }  //end main
}  //end class

The output is:
primavera
verano
otoño
invierno
Problem

Complete the following program

class Exercise5 {
    public static void main(String args[]) {
        int dias_semana[];
        
        ......................
        dias_semana[0]=1;
        dias_semana[1]=2;
        dias_semana[2]=3;
        dias_semana[3]=4;
        dias_semana[4]=5;
        dias_semana[5]=6;
        dias_semana[6]=7;
        System.out.println("El día de la semana "+dias_semana[0]+ "es lunes");
    } //end main
} //de la clase

Answer1
dias_semana=new int[7];

Answer2
dias_semana=new int [7];

Clue1
Has the number of elements in the array been indicated?

Clue2
Before assigning individual values to an array type variable, you have to reserve memory and indicate how many and what kind of elements the array will have.

Clue3

Clue4

Example
class Exercise {
    public static void main(String args[]) {
        double [] argumentos;
        argumentos=new double[2];
        argumentos[0]=Math.sqrt(5);
        argumentos[1]=Math.sqrt(7);
        System.out.println("Los argumentos son"+argumentos[0]+argumentos[1]);
    } //end main
} //de la clase
Problem

Complete the following method

```java
static int leeropcion() throws IOException{
    System.out.println("Introduce la opción deseada...
"n");
    int entrada=Integer.parseInt(teclado.readLine());
    while (entrada<1 || entrada>6) {
        System.out.println("Opción inAnswer1. Vuelva a intentarlo...");
        entrada= Integer.parseInt(teclado.readLine());
    }
}
```

// leeropcion

Answer1

return entrada;

Answer2

Clue1

Check whether the method returns a value

Clue2

Clue3

Clue4

Example

```java
static int resultado()
{
    resultado=0;
    while (resultado<5) \{ 
        resultado=resultado+1;
    \}
    return resultado;
}  // end resultado
```
Problem

Style exercise.
It is important to give significant names to both the variables and methods. The following method is called Calculate_. You must indicate what must be calculated.

class Calcula_{
    public static void main(String args[]) {
        int numeros[]=\{1,2,3,4,5\};
        double resultado=0;
        int i;

        for (i=0;i<5;i++)
            resultado=resultado+numeros[i];
        System.out.println("El resultado es"+resultado/5);
    } // main
} end class

Answer1
mean

Answer2
average

Clue1

Clue2

Clue3

Clue4

Example
Problem

Complete the following program

```java
class Exercise8{
    public static void main(String args[]){
        int dos_dimisiones[][]=new int[4][5];
        int i,j,k=0;

        for(i=0;i<4;i++)
            for(--------------){
                dos_dimisiones[i][j]=k;
                k++;
            }//for
        }//end main
    }//end class
```

Answer1

```
j=0; j<5; j++
```

Answer2

```
j=0; j<=4; j++
```

Clue1

The index `j` represents the five columns in the array

Clue2

Clue3

Clue4

Example

```java
class Exercise8{
    public static void main(String args[]){
        int dos_dimisiones[][]=new int[4][6];
        int j,i,k=0;

        for(j=0; j<4; j++)
            for(i=0; i<6; i++)
                System.out.print(dos_dimisiones[j][i]+" ");
        }//end main
    }//end class
```
Problem

Correct the mistake in the following program

```java
import java.io.*;

class Exercise9 {

    static BufferedReader teclado=new BufferedReader(new
    InputStreamReader(System.in));

    static int leerArgumentos (int opcion) throws IOException{
        int salida;
        salida=0;
        switch (opcion) {
            case 1: System.out.print("Introduce un 1");
                     salida=Integer.parseInt(teclado.readLine());
                     break;
            case 2: System.out.print("Introduce un 2");
                     salida=Integer.parseInt(teclado.readLine());
                     break;
        } //fin del switch
        return salida;
    } //fin leerArgumentos

    public static void main(String args[]) throws IOException{
        int opcion=2;
        int resultado;
        resultado=leerArgumentos();
    } //end main
} //end class
```

Answer1

```java
resultado=leerArgumentos(opcion);
```

Answer2

```java
opcion
```

Clue1

The compiler error is: leerArgumentos(int) cannot be applied to ( )

Clue2

When a method with parameters is called you must write the arguments
Example

static boolean numeroPar (int n){
    boolean solucion;
    if ((n%2)==0){
        solucion=true;
    }else
        solucion=false;
    return solucion;
} //fin numeroPar

public static void main(String args[]) throws IOException{
    boolean resultado;
    int numero;
    numero=Integer.parseInt(args[0]); //el numero lo lee por la linea de
    ordenes
    resultado=numeroPar(numero);
    if (resultado)
        System.out.println("El"+numero+"es par");
} //end main
There is an error in the following code. Correct it

```java
static double sumar(double a1, double a2){
    return a1+a2;
} // fin sumar

static double restar(double a1, double a2){
    return a1-a2;
} // fin restar

static void procesar (int opci) throws IOException{
    if (opci !=6) {
        double resultado=0;
        switch(opci) {
            case 1: resultado=sumar(3,5);
                    break;
            case 2: resultado=resta(5,3);
                    break;
        } // switch
    } // if
} // procesar.
```

**Answer1**

```java```
case 2: resultado=restar(5,3);
```

**Answer2**

```java```
restar
```

**Clue1**

**Clue2**

**Clue3**

**Clue4**

**Example**
Problem

Correct the mistake in the following program. (Write the complete line)

```java
import java.io.*;

class Series {
    public static void main(String [] args) throws IOException {
        BufferedReader leer = new BufferedReader(new InputStreamReader(System.in));

        System.out.print("Introduzca el valor de x: ");
        double x = Double.parseDouble(leer.readLine());
        System.out.println("sen("+x+") = "+Math.sin(x));
    } // fin main

    static double seno(double x) {
        int i;
        double suma, ultimo, termino;
        i = 0;
        suma = 0;
        termino = 0;
        do {
            termino = Math.pow(x, 2*i+1)/Math.pow(2*i+1, i);
            if (i == 0)
                suma = suma + termino;
            else
                suma = suma - termino;
            i++;
        } while (Math.abs(termino - ultimo) >= 0.001);
        return suma
    } // seno

} // clase series
```

Answer1

```java
return suma;
```

Answer2

```java
;
```

Clue1

```java`

Clue2

```java`

Clue3

```java`

Clue4

```java`

Example
Problem  

Predict the result of the following program if it is called: java Exercise 3 (in other words if the argument is the number 3)

```java
class Exercise {
    public static void main (String[] args) {

        int N, j;
        boolean cierto;
        N=Integer.parseInt (args[0]);
        cierto=true;
        j=2;
        while (j<=N/2 && cierto) {
            if (N%j == 0) {
                cierto = false;
            } 
            else 
                j++;
        }//end while 
        if (cierto) {
            System.out.println("El "+N+" es cierto");
        }//if
    } // method
} // end class
```

Answer1  

El 3 es cierto

Answer2

Clue1

Clue2

Clue3

Clue4

Example
Problem

Style exercise. Compare this program to the previous one. Write the number of the exercise that you have found easiest to understand?

class Primos_2 {
    public static void main (String [] args) {
        int N, j;
        boolean primo;
        N=Integer.parseInt (args[0]);
        System.out.println ("n"+"Primos comprendidos entre 1 y "+N+" : ");

        for (int i=1;i<=N;i++){
            primo=true;
            j=2;
            while (j<i/2 && primo) {
                if (i%j == 0) {
                    primo = false;
                }
                j++;
            }
            if (primo) {
                System.out.println ("El "+i +" es primo");
            }
        }
    } // method
} // class

Answer1

Answer2

Clue1

Clue2

Clue3

Clue4

Example
Problem
Complete the following exercise

```java
class Exercise14 {
    public static void main(String args[]) {
        int numero[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int j;
        for (j = 0; ......................) {
            if ((numero[j] % 2) == 0) System.out.print(numero[j]);
        } //for
    } //main
} //class
```

The program output is: 246810

Answer 1
j < 10; j++

Answer 2
j <= 9; j + 2

Clue 1
The syntax and definition of For loop is:
for (Initial value ; Test for the loop to continue; Increment )

Clue 2
Clue 3
Clue 4
Example
Problem
What is the output of the following program?

```java
class Exercise15 {
    public static void main(String args[]) {
        int numeros[] = {0, 2, 4, 6, 8};
        int j;
        for (j = 4; j >= 0; j -= 2) {
            System.out.print(numeros[j]);
        }
    }
}
```

Answer1
840

Answer2

Clue1
: An element in an array can be read at any time. The array doesn't have to be read in its entirety.

Clue2

Clue3

Clue4

Example
```java
class Example15 {
    public static void main(String args[]) {
        char caracteres[] = {'a', 'b', 'c', 'd', 'e', 'f'};
        int j;
        for (j = 5; j >= 0; j--) {
            System.out.print(caracteres[j]);
        }
    }
}
```

The output is:
Fedcba
APPENDIX 2. HabiPro Exercises

Problem

Style exercise.

Psychological Distance is the ease with which two items can be differentiated. It is convenient that the variables have a big psychological distance in this way mistakes can be avoided. Which pair does have the psychological distance bigger?

Write the number

Variable1
1. Stoppt
2. Shiftrf
3. Sumando1
4. gcount
5. Product

Variable2
St0ppt
Shifrtr
Sumando2
ccoung
Sum

Answer1

Answer2

Clue1

Clue2

Clue3

Clue4

Example
Problem
Correct the mistake in the following program

class Palindromo{
    public static void main (String [] args) {
        if (args.length==1) {
            String palabra=args[0];
            boolean palindro;
            palindro=palindromo(palabra);
            if (palindro)
                System.out.println("La cadena es un palíndromo.");
            else
                System.out.println("La cadena no es un palíndromo.");
        }
        else{
            System.out.println("Debe introducir un parámetro por la línea de
                ordenes");
        } //fin main

    static palindromo (String palabra){
        int prin, fin;
        boolean palin=true;
        prin=0;
        fin=(palabra.length())-1;
        while ((prin<fin) && palin) {
            if (palabra.charAt(prin) != palabra.charAt(fin))
                palin= false;
            else{
                prin++;
                fin--;
            }
        }
    return palin;
    } // palindromo
} //class

Answer1
boolean

Answer2
static boolean palindromo (String palabra) {

Clue1
A method must indicate if it returns a value.

Clue2

Clue3

Clue4
Example

static boolean numeroPar (int n) {
  boolean solucion;
  if ((n%2)==0) {
    solucion=true;
  } else {
    solucion=false;
  }
  return solucion;
} //end numeroPar

public static void main(String args[]) throws IOException {
  boolean resultado;
  int numero;
Problem

Correct the mistake in the following program.

```java
static void numero_es_par (int numero) {
    boolean par;
    if ((numero%2)==0) then
        par=true;
    else
        par=false;
    if (!par) {
        System.out.print("El numero "+ numero+" es par");
    }
}
```

Answer1

par

Answer2

(par)

Clue1

The symbol ! (logical not) does negative the value of a boolean variable (reverse a boolean)
Problem

Predict the output of the following program?

class Exercise19 {
    public static void main(String args[]) {
        int num = 3;
        do {
            System.out.print(num);
            num--;
        } while (num > 3);
        System.out.print(num);
    }
}

Answer 1

32

Answer 2

Clue 1

The "do-while" loop always is executed at least once.

Clue 2

Clue 3

Clue 4

Example
AP2.2 Version Without Simulated Student: Exercises

Correct the mistake in the following program:

class Exercise1 {
    public static void main(String args[]) {
        int a;
        long b;
        long c;
        int d;
        a=500000;
        b=2000;
        c=b*24*60*60;
        d=a*c;
        System.out.print(b);
        System.out.println(d+" Kilómetros.");
    }
}

Answer1 long d;

Answer2 The same program with significant names of variable is:

class Exercise2 {
    public static void main(String args[]) {
        int velocidadLuz;
        long dias;
        long segundos;
        int distance;
        velocidadLuz=500000;
        dias=2000;
        segundos=dias*24*60*60;
        distancia=velocidadLuz*segundos;
        System.out.print("En "+dias);
        System.out.print(" días la luz recorrerá cerca de ");
        System.out.println(distancia+" Kilómetros.");
    }
}

Clue1 The program output is: En 1000 días la luz recorrerá cerca de 25920000000000 Kilómetros.

Clue2 The variable distance stores a number too great, isn’t it?
Example

class Example1 {
    public static void main(String args[]) {
        int entero = 12;
        double pi = 3.1416;
        long enteroLargo = 2147483648;
        System.out.println("entero = " + entero);
        System.out.println("pi = " + pi);
        System.out.println("enteroLargo = " + enteroLargo);
    }
}
Complete the following program

```java
class Exercise2{
    public static void main(String args[]){
        int sum_uno, contador;

        for (contador=1; contador<=4; contador++) {
            sum_uno = sum_uno + contador;
        }
        System.out.println("suma=\n" + sum_uno);
    }
}
```

The program output is:

```
suma=15;
```

**Answer1**

```
sum_uno=5;
```

**Answer2**

```
sum_uno accumulates its own value plus the value of the contador variable.
```

**Clue1**

```
The for loop is executed four times.
```

**Clue2**

```
The sum_uno variable must be inicilizated.
```

**Clue3**

**Example**

```java
class Example2{
    public static void main(String args[]){
        int suma;
        suma = 0;
        for(int j=1;j<=5;j++) {
            suma = suma + j;
        }
        System.out.println("Suma=" + suma);
    }
}
```

The output is:

```
Suma=15
```

228
APPENDIX 2. HabiPro Exercises

What is the output of the following program?

class Exercise3 {
    public static void main(String args[]) {
        boolean b;
        b = false;
        System.out.println("b es "+b);
    }
}

Answer1 b es false

Answer2

Clue1 Boolean variables can only take the values "true" or "false"

Clue2 When we write a boolean variable with println, what is really written is "true" or "false"

Clue3

Clue4

Example
class Example3 {
    public static void main(String args[]) {
        boolean b;
        b = true;
        System.out.println("b es "+b);
    }
}

The program output is:

b es true
What is the output of the following program?

class Exercise4{
    public static void main(String args[]) {
        int numeros[]={0,2,4,6,8};
        int j;
        for(j=0;j<5;j+=2) {
            System.out.print(numeros[j]);
        }
    }
}

Answer1
048

Answer2

Clue1
An element of an array can be consulted at any time. There is no reason why it should be read in its entirely or in an orderly form.

Clue2
The syntax of loop For is:
for (Initial; Test; Increment )

Clue3
The block inside the "for" is executed while the Test is true.

Clue4
The Initial value is modified according to the Increment.

Example
class Example4{
    public static void main(String args[]) {
        int numeros[]={0,2,4,6,8,10,12,15};
        int j;
        for(j=0;j<5;j+=3) {
            System.out.print(numeros[j]);
        }
    }
}

The output is:

0612
What is the output of the following program?

```java
class Exercise5 {
    public static void main(String args[]) {
        int contador = 10;
        int salida = 0;

        while (contador < 12) {
            salida = contador + salida;
            contador = contador + 1;
        }
    }
}
```

**Answer 1**
21

**Answer 2**
The while loop is executed twice
The variable called salida stores its own value plus the contador value. Later the contador increases its value.

**Clue 1**

**Clue 2**

**Clue 3**

**Clue 4**

**Example**

```java
class Example5 {
    public static void main(String args[]) {
        int num, acumulador;
        num = 2;
        acumulador = 0;
        while (num <= 4) {
            acumulador = acumulador + num;
            num = num + 1;
        }
        System.out.println(acumulador)
    }
}
```

The program output is:

9
What is the output of the following program?

```java
class Exercise6 {
    public static void main(String args[]) {
        boolean b;
        b = false;
        System.out.println("b es "+ b);
        b = true;
        System.out.println("b es "+ b);
        if(b==true) System.out.println("Esta sentencia si se ejecuta.");
        b = false;
        if(b==true) System.out.println("Esta sentencia no se ejecuta.");
        System.out.println("10>9 es "+ (10 > 9));
    }
}
```

**Answer 1**
- `b es false`
- `b es true`
- `Esta sentencia si se ejecuta`
- `10>9 es true`

**Answer 2**

**Clue 1**
When we write a boolean variable with println, what is really written is "true" or "false".

**Clue 2**
The output of a relational operator is a boolean variable.

**Clue 3**

**Clue 4**

**Example**

```java
class Example6{
    public static void main(args[]){
        boolean b=true;
        System.out.println("b es "+b);
        System.out.println("10<9 es "+(10<9));
    }
}
```

The output of this example is:

- `b es true`
- `10<9 es false`
What is the output of the following program?

class Exercise7 {
    public static void main(String args[]) {
        for(int x=1; x<=5; x++) {
            System.out.print(x);
        }
    }
}

Answer1 12345
Answer2
Clue1 The syntax of loop For is:
for (Initial; Test; Increment )
Clue2 The block inside the "for" is executed while the Test is true.
Clue3 The Initial value is modified according to the Increment.
Clue4
Example
class Example7 {
    public static void main(String args[]) {
        int suma=0;
        for(int x=1; x<=10; x++) {
            suma+=x;
        }
        System.out.println("El valor de suma es " + suma);
    }
}

The output of this example is:
El valor de suma es 55
Correct the error in the following program

class Exercise8 {
    public static void main(String args[]) {
        boolean b;
        b = verdadero;
        System.out.println("b es "+b);
    }
}

Answer1
b = true;

Answer2
b = false;

Clue1
The range of values of a boolean data type are false and ....

Clue2

Clue3

Clue4

Example
class Example8 {
    public static void main(String args[]) {
        boolean b = true;
        System.out.println("b es "+b);
    }
}

The program output is:

b es true
Correct the error in the following program

```java
class Exercise9{
    public static void main(String args[]) {
        double x;
        int z;
        int w;

        x=3;
        y=45;
        z=7;
        x=Math.sqrt(y*z);
        System.out.println("El resultado es "+x);
    }
}
```

**Answer1**

int y;

**Answer2**

int y;

**Clue1**

How many variables have been declared?

**Clue2**

How many variables does a value have?

**Clue3**

Do the initialised variables coincide with the declared variables?

**Clue4**

All the variables used in a program must be declared.

**Example**

```java
class Example9{
    public static void main(String args[]){
        int a,b;
        double c;
        a=3;
        b=5;
        c=a+b;
        System.out.println("El resultado es "+c);
    }
}
```

The output of this program is:

El resultado es 8.0
Correct the error in the following program:

```java
class Exercise10{
    public static void main(String args[]) {
        boolean b;
        b = false;
        System.out.println("b es " + b);
        b = true;
        if(b) System.out.println("Esta sentencia no se ejecuta.");
        System.out.println("10>9 es " + (10 > 9));
    }
}
```

The output is:

b es false
10>9 es true

```java
Answer1
b=false;
```

```java
Answer2
```

**Clue1**
The condition of an "if" may be an expression which returns a boolean result or even a boolean variable.

**Clue2**
When we write a boolean variable with println, what is really written is "true" or "false".

**Clue3**
The output of a relational operator is a boolean variable.

**Clue4**

```java
class Example10{
    public static void main(String args[]) {
        boolean b=true;
        if(b!){
            System.out.println("Se cumple la condición");
        }else System.out.println("No se cumple la condición");
    }
}
```

The program output is:
No se cumple la condición.
Write the line which contains the mistake in the following program:

```java
class Exercise11 {
    public static void main(String args[]) {
        int x;
        x = 10;
        if(x == 10) {
            int y = 20;
            System.out.println("x and y: "+ x + " " + y);
            x = y * 2;
        }
        y = 100;
        System.out.println("x is "+ x);
    }
}
```

**Answer1**

```java
int y=20;
```

**Answer2**

```java
y=100;
```

**Clue1**

Do you remember what the scope of a variable is?

**Clue2**

A variable can't be used outside its scope.

**Clue3**

A block defines the scope of a variable.

**Clue4**

Have you checked whether the variables are used outside their scope?

```java
class Example11 {
    public static void main(String args[]) {
        int w;
        w=1;
        if (w==1){
            int x;
            x=3;
            System.out.println(x);
        }
        System.out.println(w);
    }
}
```

The program output is:

```
3
1
```
What is the output of the following program?

class Exercise12 {
    public static void main(String args[]) {
        int x=5;
        do{
            System.out.println("El valor de x es: "+x);
        } while(x<5);
    }
}

**Answer1**

El valor de x es: 5

**Answer2**

**Clue1**
The body of the loop is executed repeatedly whilst the "while" condition is true.

**Clue2**
The condition is tested after of executing the body of the while. The while body is always executed once.

**Clue3**
The "do while" syntax is:

do{
    // statements
} while(condition);

**Clue4**

class Example12{
    public static void main(String args[]{
        int num=2;
        do{
            System.out.print(num);
            num--;
        }while(num>0);
    }
}

The output is:

21
Correct the mistake in the following program:

```java
class Exercise13{
    public static void main(String args[]) {
        int numeros[]={0,1,2,3,4};
        int j;
        for(j=0;j<5;j++) {
            System.out.print(numeros[j]);
        }
    }
}
```

The program output is:

01234

**Answer1**

j<5

**Answer2**

j<=4

**Clue1**

The first element of the array is that which correspond to index 0, and the last one is that which correspond to the index (lengtharray - 1).

**Clue2**

The syntax of "for" is:
for (Initial ; Test ; Increment)

**Clue3**

The for body is executed while the Test is true..

**Clue4**

The initial value increases with the increment

**Example**

```java
class Example13{
    public static void main(String args[]){
        char vocales[]={'a','e','i','o','u'};
        for(int j=0;j<5;j++){
            System.out.print(vocales[j]);
        }
    }
}
```

The program output is:

aeiou
What is the output of the following program?

```java
class Exercise14{
    public static void main(String args[]){
        System.out.println("10>9 es "+(10>9));
    }
}
```

**Answer1**

10>9 es true

**Answer2**

The result of a comparison is a boolean value.

**Clue1**

The part which is not in "" is valued as an expression (relational expression in this case).

**Clue2**

**Clue3**

**Clue4**

**Example**

```java
class Example14{
    public static void main(String args[]){
        if (5<3)
            System.out.println(true);
        else System.out.println(false);
    }
}
```

The program output is:

false
Complete the following program:

class Exercise15{
    public static void main(String args[]) {
        int suma;
        
        for(int j=1;j<=5;j++) {
            suma=suma+j;
        }
        System.out.println("Suma="+suma);
    }
}

The output is:

Suma=15

**Answer1**

suma=0;

**Answer2**

**Clue1**

All the variables used in a program must be declared and initialized.

**Clue2**

**Clue3**

**Clue4**

**Example**

class Example15{
    public static void main(String args[]){
        int contador=0;
        while(contador<3){
            System.out.println(contador);
            contador++;
        }
    }
}

The program output is:
0
1
2
What is the output of the following program?

```java
class Exercise16{
    public static void main(String args[]) {
        int salida=0;
        int x=1;
        while(x<=10) {
            salida=salida+x;
            x=x+2;
        }
        System.out.println(salida);
    }
}
```

Answer1 25

Answer2

Clue1 The while loop executes the while body while the condition is true.

Clue2 The while loop evaluates the condition before executing the while body.

Clue3 The while syntax is:
```java
while(condition){
    // while body
}
```

Clue4

Example
```java
class Example16{
    public static void main(args[]){
        int j=0;
        while(j<3){
            System.out.print(j);
            j++;  
        }
    }
}
```

The output is:
012
Complete the following program

```java
class Exercise17 {
    public static void main(String args[]) {
        int numero[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int j;
        for (..........................) {
            if ((numero[j] % 2) == 0) System.out.print(numero[j]);
        }
    }
}
```

The output is:

246810

**Answer1**  
j=0;j<10;j++

**Answer2**  
j=0;j<=9;j++

**Clue1**  
The syntax of "for" loop is:
```java
for ( Initial ; Test ; Increment )
```

**Clue2**  
The sentences inside of the for loop are executed while the test is true.

**Clue3**  
The increment indicates how the initial value is going changing.

**Example**

```java
class Example17 {
    public static void main(String args[]) {
        char vocales[] = {'a', 'e', 'i', 'o', 'u'};
        for (int j = 0; j < 5; j++) {
            System.out.print(vocales[j]);
        }
    }
}
```

The program output is:

aeiou
What is the output in the following program?

```java
class Exercise18 {
    public static void main(String args[]) {
        int numeros[] = {0, 2, 4, 6, 8};
        int j;
        for (j=4; j>=0; j--) {
            System.out.print(numeros[j]);
        }
    }
}
```

**Answer1**
840

**Answer2**
An element of an array can be consulted at any time. There is no reason why it should be read in its entirely or in an orderly form.

**Clue1**
The Initial value in a for loop may be increased or decreased in any value.

**Clue2**
The syntax of "for" loop is:
for ( Initial ; Test ; Increment )

**Clue3**
The sentences inside the "for" loop are executed while the test is true.

**Example**
```java
class Example18 {
    public static void main(String args[]) {
        char caracteres[] = {'a', 'b', 'c', 'd', 'e', 'f'};
        int j;
        for (j=5; j>=0; j--){
            System.out.print(caracteres[j]);
        }
    }
}
```

The output is:
fedcba
Complete the following program:

class Exercise19 {
    public static void main(String args[]) {
        String estaciones[] = {"primavera", "verano", "otoño", "invierno"};
        int j = 0;
        while (.............) {
            System.out.println(estaciones[j]);
            j++;
        }
    }
}

The output is:

primavera
verano
otoño
invierno

**Answer1**  
j<4

**Answer2**  
j<=3

**Clue1**  
The while loop executes the while body while the condition is true.

**Clue2**  
The while loop evaluates the condition before executing the while body, this means that the while body may not be executed.

**Clue3**  
The syntax while loop is:

```
while(condition){
    // Statements
}
```

**Clue4**  

**Example**  
class Example19 {
    public static void main(String args[])
    {
        String semana[] = {"lunes","martes","miercoles","jueves","viernes","sábado","domingo"};
        while(j<7) {
            System.out.println(semana[j]);
            j++;
        }
    }
}
The output is:

lunes
martes
miercoles
jueves
viernes
sabado
domingo
APPENDIX 3. Use of HabiPro
**Connection**

HabiPro is a client-server application. The version used in the experiment and presented here only permits just the connection of two users.

When a student wants to work with HabiPro, s/he has to introduce the name or IP address of the computer where the server is running in the client window (the server and client applications can be in the same computer). The user also needs introduce his/her user name.

When the server receives the application for connection this sends a message to the client accepting the connection and informing him/her whether any users are already connected. At the moment when two users connected are they can start work.

**HabiPro Interface**

![HabiPro Interface](image)

Figure 1. HabiPro interface

Figure 1 shows HabiPro Interface. Students communicate with each other using the chat window. Each student’s intervention is represented in a different colour. The small window, on the right, indicates the names of the users connected. Alumno3 is the name that the Simulated Student has by default. Figure 1 shows an empty window next to the window
where the names are shown. This window is where students write what they want to send to the others, so it is part of the chat window.

On the left there is a big window. This shows the exercises that students have to solve. In this case the exercise is: "Completar el siguiente programa" (In English: "Complete the following program").

Behind the main window we can see three windows: The first one is titled "Escribe tu respuesta" ("Write your answer"). As the title indicates, that window is where students write a solution to the exercise. This written solution is not seen by the other student until that the student presses the button "enviar" ("send") next to the window. When the button is pressed the other user receives the student’s proposal in the window tilted "Respuesta 2", "answer 2". In the case the person who sent the answer was the Simulated Student, the proposal is shown in the window "Respuesta 3".

![Figure 2. Students' answers](image)

Figure 2 shows an example where all students (even the Simulated Student) have written and sent one solution. In this example the program contains a mistake, the students have to
APPENDIX 3. Use of HabiPro

find it and correct it. The program accepts that students write the whole line where the mistake occurs correctly or just the mistake itself (as Student 3 did).

In the answer window students should just write Java code. If they want to comment on anything about the proposal with the other students they can use the chat window for this.

Once the students have reached an agreement about which of the proposed answers is the best one, it is the moment to check the answer. Both students have to press the button "Chequear" ("Check") next to that solution. HabiPro will only check the answer when all students have pressed the button "check". In this way we try to avoid students working independently, so they have to discuss the solution and to decide which solution to check. In a previous version of HabiPro the answer could be checked by one student. So, for example, a student could write a solution and check it. In the case that the solution was correct, HabiPro displayed the next exercise. This fact caused some students to see a new exercise come up without knowing how the previous one was solved.

At the moment that all students press the check button HabiPro checks the answer and two things may occur: The answer is correct so HabiPro shows a window similar to that shown in Figure 3 and HabiPro displays the next exercise. Or if the answer in incorrect then HabiPro shows a window as in Figure 4.

![Figure 3. Window indicating correct answer](image)

HabiPro informs the students that the answer is incorrect and shows two buttons "Ayuda" ("Help") or "volver a intentar" ("Try again"). It is important to clarify that Figure 4 is just
shown to one group member (the first student connected to HabiPro). This person is in charge of consulting via chat if it is necessary to consult the "help" or if the group prefers to try to solve the exercise without "help" (in this case the button "Try again" must be pressed). The other student sees a window informing him/her that the solution was incorrect.

![Figure 4. Window that indicates that the answer is incorrect](image)

If students decide to consult the help, two new buttons arise (see Figure 5). One button gives clues about how to solve the problem and the second button displays a similar exercise to the problem that students have to solve, the solution is also displayed so students can compare the exercises and try to find the solution to their problem.
Figure 5: HabiPro Help

Figure 6 shows an example where HabiPro is displaying a clue, in Figure 7 a counterexample is being offered.
Corrige el error del siguiente programa

```java
salida=Integer.parseInt(leer.readLine());
break;
} //fan del switch
return salida;
} //fan leerelementos
public static void main(String args[]) throws IOException{
   int opcion;
   resultado=leerelementos();
} //fin main
```

Alumno1 > Chequea la respuesta 2
Alumno2 > Chequea la respuesta 2
Alumno2 > Chequea tu respuesta
Alumno3 > ¿Probamos con mi respuesta?
Alumno1 > Chequea la respuesta 3
Alumno3 > Este ordenador es muy rápido, yo estoy en el ejercicio 5, vosotros también?
Alumno1 > Si, yo también
Alumno2 > igual yo
Alumno1 > Chequea la respuesta 2
Alumno1 > chequea
Alumno3 > miramos el contraejemplo o las pistas?
Alumno2 > venga hay que verlas

Figure 6. HabiPro displaying a clue
Figure 7. HabiPro displaying a counterexample

**Disconnection**

The session finishes when one student presses the last button situated in the icon bar or when s/he chooses the option to disconnect (submenu of connection "Conexión" menu) in the menu bar.
**PRE-TEST Exercises**

1. **What is the output in the following program?**

   ```java
   class Exercise1 {
       Public static void main(String args[]) {
           boolean b;
           b=false;
           System.out.println("b es "+b);
       }
   }
   ```

2. **Complete the following program**

   ```java
   class Exercise2 {
       public static void main(String args[]) {
           int num;
           System.out.println("Esto es un num: "+num);
           num=num/2;
           System.out.println("El valor de num/2 es");
           System.out.println(num);
       }
   }
   ```

   **The program output is:**
   
   Esto es un num: 100
   El valor de num/2 es
   50

3. **What is the output in the following program?**

   ```java
   class Exercise3 {
       public static void main(String args[]) {
           boolean b;
           b = false;
           System.out.println("b es "+b);
           b = true;
           System.out.println("b es "+b);
           if(b==true) System.out.println("Esta sentencia se ejecuta.");
           b = false;
           if(b==true) System.out.println("Esta sentencia no se ejecuta.");
           System.out.println("10>9 es"+(10>9));
       }
   }
   ```
4. What is the output in the following program?

class Exercise4 {
    public static void main(String args[]) {
        for (int x = 1; x <= 5; x++) {
            System.out.print(x);
        }
    }
}

5. Correct the mistake in the following program:

class Exercise5 {
    public static void main(String args[]) {
        boolean b;
        b = verdadero;
        System.out.println("b es" + b);
    }
}

6. Correct the mistake in the following program:

class Exercise6 {
    public static void main(String args[]) {
        double x;
        int z, w;
        x = 3;
        y = 45;
        z = 7;
        x = Math.sqrt(x * y * z);
        System.out.println("El resultado es " + x);
    }
}
7. Write the line where there is a mistake and explain why

```java
class Exercise7{
    public static void main(String args[]){
        int x;
        x = 10;
        if(x==10) {
            int y=20;
            System.out.println("x and y:"+ x + " " + y);
            x = y * 2;
        }
        y = 100;
        System.out.println("x es" + x);
    }
}
```

8. What is the output in the following program?

```java
class Exercise8{
    public static void main(String args[]){
        int salida=0;
        int x=1;
        while(x<=10){
            salida=salida+x;
            x=x+2;
        }
        System.out.println(salida);
    }
}
```

9. Correct the mistake in the following program

```java
class Exercise9{
    public static void main(String args[]){
        int oper0, oper1, oper2;
        oper1 = 3;
        oper0 = 5;
        oper2 = oper1+oper0;
        System.out.println("El resultado es "+ oper2);
    }
}
```
10. Correct the mistake in the following program:

class Exercise10{
    public static void main(String args[]){
        int velocidadLuz;
        long dias;
        long segundos;
        int distancia;
        velocidadLuz=300000;
        dias=1000;
        segundos=dias*24*60*60;
        distancia=velocidadLuz*segundos;
        System.out.print("En "+dias);
        System.out.print(" días la luz recorrerá cerca de ");
        System.out.print(distancia+"Kilómetros.");
    }
}

APPENDIX 4. Materials
Questionnaire (without Simulated Student version)

1. Do you think the system help you to learn?
   Not at all 1 2 3 4 5 Very much

2. How would you describe the help offered by your colleague?
   Not very useful 1 2 3 4 5 Very useful

3. How would you describe your colleague's level of participation?
   Low 1 2 3 4 5 High

4. Do you think you would have learnt more working by yourself
   No Yes

5. If the answer was no, indicate why
   • My colleague helped me to think about other possible solutions.
   • My colleague helped me to understand concepts.
   • I was more motivated than I would have been working by myself.
   • Any other reasons:

   If the answer was yes, indicate why
   • I would have worked faster alone.
   • We have wasted time talking about thing not related to the exercises.
   • My colleague did not propose any solution.
   • My colleague did not take part much.

6. In general, do you think this system has helped you to learn programming.
   • Faster than reading a book and working alone.
   • The same as reading a book and working alone.
   • More slowly than reading a book and working alone.

7. How do you value the system as a learning method
   Bad 1 2 3 4 5 Good

8. Would you like to add any comments of your own
Questionnaire (Simulated Student version)

- Do you think the system help you to learn?
  Not at all 1 2 3 4 5 Very much

- How would you describe the help offered by your colleague ( )?
  Not very useful 1 2 3 4 5 Very useful

- How would you describe the help offered by your colleague ( )?
  Not very useful 1 2 3 4 5 Very useful

- How would you describe your colleague's level of participation ( )?
  Low 1 2 3 4 5 High

- How would you describe your colleague's level of participation ( )?
  Low 1 2 3 4 5 High

- Do you think you would have learnt more working by yourself
  No Yes

- If the answer was no, indicate why
  - My colleague ( ) helped me to think about other possible solutions.
  - My colleague ( ) helped me to understand concepts.
  - I was more motivated than I would have been working by myself.

- Any other reasons:
  If the answer was yes, indicate why
  - I would have worked faster alone.
  - We have wasted time talking about thing not related to the exercises.
  - My colleague ( ) did not propose any solution.
  - My colleague ( ) did not take part much.

- In general, do you think this system has helped you to learn programming.
  - Faster than reading a book and working alone.
  - The same as reading a book and working alone.
  - More slowly than reading a book and working alone.

- How do you value the system as a learning method
  Bad 1 2 3 4 5 Good

- Would you like to add any comments of your own
**First example**

********************************************************************************

**PROBLEM Nº 0**

Student1 connects
Student2 connects
Student2 > Hello
Student1 > hello
Student2 writes answer-->character=(stdin.readLine()).charAt(0);
Student2 > the correct line must be:
Student2 > character=charAt(0).(stdin.readLine());
Student1 > wait a minute, I'm going to check it
Student3 > let's see if we can do all the exercises ok?
Student2 > I think I've made a mistake
Student2 > readline shouldn't have a capital L
Student2 > readline??
Student1 > yes readLine
Student1 > has anyone got their notes with them?
Student2 writes answer-->character=(stdin.readLine()).charAt(0);
Student2 writes answer-->character=(stdin.readLine()).charAt(0);
Student2 > check answer2
Student1 > are you sure?
Student2 > what do you think student3??
Student3 > stop chatting and think a little
Student1 > readLine has a capital letter
Student1 writes answer-->character=(stdin.readLine()).charAt(0);
Student1 > I think that's is it. What do you think?
Student2 > should char be written with capitals or not
Student1 > not
Student1 > the mistake's in the while line
Student2 > while (character!='a' && character!='b') {
Student2 > this is the correct line
Student1 > yes
Student2 > it's && not ||
Student2 > write it in your answer window
Student2 > and send it
Student2 > press copy and paste
Student1 writes answer-->while(character!='a' && character!='b');•
Student1 > check?
Student2 > check your answer
Student3 > what does this sign mean ||?
Student1 asks for a clue
Student2 > are you receiving me?
Student2 > check answer2
Student1 > I haven't got an answer
Student2 writes answer-->while (character!='a' && character!='b') {
Student2 > that's it
Student1 > yes, but that's not the solution
********************************************************************************

**PROBLEM Nº1**

Student2 > yes it is
Student1 > pardon?
Student1 > the bloody machine said that it wasn't before
Student2 writes answer-->La salida del programa es 26
Student1 > is the array there for decoration or is it my imagination?
Student1 > in this exercise we have to correct the mistake in the program
Student2 > let's concentrate
Student2 > ok
Student2 writes answer---->for(j=0;j<5;j+=2) {
Student2 > for(j=0;j<5;j+=2) {
Student1 > does anyone know how += works?
Student2 > j+=2
Student2 > j=0 numeros[0]=0
Student2 > j=2 numeros[2]=4
Student2 > j=4 numeros[4]=8
Student2 > j is <5 in all cases
Student2 writes answer---->for(j=0;j<5;j+=2) {
Student2 > check answer2
Student3 > Student1 why don't you propose a solution? say something
Student1 > I think that j should be initialized at 0
Student3 > we've made a good start!!!
***************PROBLEM Nº2***************
Student2 writes answer---->[b] System.out.println("Esta sentencia se ejecuta.");
Student2 > check answer2
Student1 writes answer---->public static void main(String args[]){
Student1 > look at the main and you'll see an extra ;
Student2 > yes
Student1 > check?
Student2 > yes
Student1 > check answer2
Student3 > shall we have a look at the clues?
Student2 presses the help button
Student2 asks for a clue
Student2 > we have to delete the {
Student1 > I'm sorry
Student1 writes answer---->;
Student1 > check answer2
Student2 writes answer---->static void main(String args[]); se ejecuta.")
Student1 > check it now
***************PROBLEM Nº3***************
Student1 > this program is really difficult
Student1 > yes
Student2 writes answer---->[j<=0
Student2 > check answer2
Student2 presses the help button
Student2 asks for an example
Student2 asks for a clue
Student2 asks for the next clue
Student2 writes answer---->[j>=0
Student2 > check answer2
Student3 > how about trying j>=0
Student3 writes answer j>=0
Student2 presses the help button
Student1 > I think so
Student1 > check answer3
***************PROBLEM Nº4***************
Student1 > I think the dimensions of the array are missing
Student2 writes answer---->[dias_semana[]=new int[6];
Student2 > check answer2
Student3 > let's have a look at the example. Maybe it help
Student2 asks for a clue
Student1 > ok
Student1 asks for an example
Student2 writes answer---->[dias_semana[]=new int[7];
APPENDIX 5. Example of Log

Student2 > check answer2
Student3 > I think it must be something similar to what I'm sending you. Shall we check to make sure?
Student3 writes answer: `dias_semana=new int[7];`
Student2 presses help button
Student2 asks for an example
Student2 writes answer: `dias_semana=new int[7];`
Student2 > check answer2
Student2 > check answer3
Student2 writes answer: `dias_semana=new int[7];`
Student2 writes answer: `dias_semana=new int[7];`
**************************PROBLEM N°5**************************
Student1 > I don't know
Student2 writes answer: `return entrada;`
Student2 > check answer2
Student3 > another one!!!
**************************PROBLEM N°6**************************
Student2 writes answer: `sumar`
Student2 > check answer2
Student3 > and it seemed so easy!!
Student2 asks for help
Student2 > sumatoria??
Student2 > Sumar?
Student2 > Sumatorio?
Student1 > sumatoria
Student2 > maybe we have to write it the capital letters
Student2 writes answer: `Sumatoria`
Student2 > check answer2
Student3 > this program calculated the mean of 2 numbers. Shall we try with the mean?
Student3 writes answer: `media`
Student2 presses help button
**************************PROBLEM N°7**************************
Student3 > Student1, what do you think?
Student2 > I hadn't seen the /5
Student1 > j
Student1 writes answer: `j=0;j<5;j++`
Student1 > check the 1
Student1 > check answer2
Student3 > that was easy, wasn't it?
**************************PROBLEM N°8**************************
Student1 > yes
Student1 writes answer: `resultado=leerargumentos(opcion);`
Student1 > check the 1
Student1 > check answer2
Student3 > we've already done 9 exercises!!!
**************************PROBLEM N°9**************************
Student2 > bye
Student1 > bye
Student1 > it's been a pleasure
Student1 > I'm off
Student2 writes answer: `case 2: resultado=restar(5,3);`
Student2 > check answer2
Student2 > check!!!
Student3 > great!! I'd never have noticed that mistake
Second Example

***************PROBLEM N° 1**************
Student1 connect
Student2 connect
Student1 > hello
Student2 > hello?
Student2 > I can see several mistakes
Student1 > Shall we start?
Student2 > ok
Student2 > I can see 2 things
Student2 writes answer-->readLine
Student1 > which?
Student2 writes answer-->readLine
Student2 writes answer-->condition while
Student2 > have you seen my answer?
Student1 writes answer-->in the ||
Student1 > Yes
Student2 > I agree with your solution because it should be &&
Student1 > ok
Student1 > and also in the while loop
Student1 > because it reads a character again
Student2 > I think that's right
Student1 > you're right
Student1 > we're on the next question
Student2 writes answer-->readLine
Student2 > can you see my second answer (readline)?
Student1 > How can we go to the next question?
Student2 > I don't know
Student1 > check answer2
Student2 writes answer-->&&
Student2 > check answer2
Student2 > you'll have to check
Student1 > I've already checked it
Student2 writes answer-->&&
Student1 > and now?
Student2 > have you checked the answer?
Student3 > are you on exercise 2 yet?

***************PROBLEM N°2**************
Student2 > hello
Student1 > yes
Student2 > we're on exercise 2
Student2 > let's think
Student1 > there's a mistake in the "for"
Student1 > Yes
Student2 > yes
Student1 > it should start with the 0
Student2 > ok, write the answer
Student1 writes answer-->j=0
Student1 > check answer2
Student1 > Check your answer
Student2 > I don't agree with your solution because you've got to change the j++ as well
Student2 writes answer-->for(j=0; j<5; j++)
Student2 writes answer-->for(j=0; j<5; j++)
**APPENDIX 5. Example of Log**

Student2 > are you ok?
Student1 > yes why shouldn't I be?
Student2 writes answer-->for(j=0; j<5; j++)
Student1 > let's go on the next one
Student3 > the index of an array begins with 0, doesn't it?
Student2 > yes, it starts with 0
Student2 writes answer-->for
Student1 > check answer2
Student1 > check your answer
Student2 > when you send an answer you have to press the agree button
Student2 writes answer-->for
Student3 > shall we try j=0 ?
Student3 writes answer j=0
Student2 > check your answer
Student2 > check answer3
Student1 > I agree with your solution because it begins with 0
Student2 > check answer2

********** PROBLEM N°3 **********
Student2 > did you see what the solution to the last one was?
Student1 > I'm already on exercise 3
Student2 > I'm on number 3
Student2 > I can see a logical mistake
Student1 > what
Student1 > In the System.out.println
Student1 > print a logical value
Student2 > what's wrong with the system?
Student2 > I think that's correct
Student2 > The mistake is if(b)
Student2 writes answer-->if(b)
Student1 > are you sure?
Student2 > check answer2
Student1 > I don't agree with your solution because b is a logical value
Student2 > ;
Student2 > I can see something else
Student1 > what?
Student1 > I think the rest is ok
Student2 > in the main
Student2 > There shouldn't be a ";" at the end of the main
Student1 > yes
Student1 > I agree
Student1 > write the answer
Student1 writes answer--> public static void main(String args[])
Student1 > check answer2
Student2 > I agree with your solution because the ; is not necessary
Student3 > This is going well

********** PROBLEM N°4 **********
Student1 > come on
Student1 > are you on 4
Student2 writes answer-->j=0
Student1 > no
Student1 > but the points
Student1 > I don't like them
Student2 > The answer is j=0
Student2 writes answer-->j=0
Student2 writes answer-->j=0

264
Student2 > are you ok?
Student1 > ok
Student2 writes answer--->j>=0
Student2 writes answer--->j>=0
Student1 > I agree with your solution because j decreases until reaches 0
Student1 > check your answer
Student3 > that was easy

**************************PROBLEM No 5**************************
Student1 > yes
Student1 > let's do the next one
Student1 > yes
Student1 > something like this
Student1 > no
Student1 > you haven't pressed the check button
Student2 > I am on number 5
Student1 > the array dimension
Student2 > the new is missing
Student1 > check my answer
Student2 writes answer-->dias_semana=new int[5];
Student2 > check answer2
Student3 > shall we look at the example? Maybe it'll help us.
Student1 > ok
Student1 > check your answer
Student2 writes answer-->dias_semana=new int[7];

**************************PROBLEM No 6**************************
Student1 > I'm on number 6
Student2 > I'm on number 6
Student1 > the program returns a integer
Student1 > return entrada;
Student1 writes answer-->return entrada;
Student1 writes answer-->return entrada;
Student1 > check answer2
Student1 writes answer-->return entrada;
Student2 > I agree with your solution because you have to return the value
Student3 > Another one!!

**************************PROBLEM No 7**************************
Student1 > yes
Student1 > I like this
Student1 > There's a cast missing
Student1 > Yes
Student2 > Yes
Student1 writes answer-->resultado=(double)resultado+numeros[i];
Student1 > check answer2
Student1 > check your answer
Student2 > I don't agree with your solution because the problem is ok. I think we have to give the method a name
Student1 > yes, I agree
Student2 writes answer-->mean
Student2 writes answer-->mean
Student3 > we're amazing ;))

265
***************PROBLEM N°8***************
Student1 > I'm on number 8
Student2 > A loops one, easy
Student2 > what do you think?
Student2 > I think it is j
Student1 > I think it is i
Student1 > we'll have to fill in the next for, won't we?
Student1 > the next for
Student1 > no?
Student1 > j=0; i<5; j++
Student2 > ok
Student2 > write the answer
Student1 > j=0; j<5; j++
Student2 writes answer-->j=0; j<5; j++
Student1 writes answer-->j=0; j<5; j++
Student1 > check your answer
Student1 > check answer2
Student2 > I agree with your solution because you have to read the array with j.
Student3 > it was easy

***************PROBLEM N°9***************
Student1 > yes
Student1 > I'm on number 9
Student2 > on 9
Student2 > Let's think
Student1 > I'm trying
Student2 > I can see that
Student1 > me too
Student2 writes answer-->resultado=leerargumentos(opcion);
Student1 > you haven't passed the option
Student1 > check your answer
Student2 > ok
Student3 > now we're on 10!!!

***************PROBLEM N°10***************
Student1 > me too
Student2 > maybe the arguments should be double
Student2 writes answer--> 
Student1 > I don't know
Student1 > and the class?
Student2 writes answer-->resultado=sumar((double)3,(double)5);
Student1 > the word class is missing
Student1 > check your answer
Student2 > it's a piece of code
Student3 > I think the mistake must be in the procesar method the other methods are very easy and seem to be ok. What do you think
Student1 > yes
Student1 > but let's carry on thinking
Student1 > class is missing
Student3 > why does it put 'restar' in one place and 'resta' in another?
Student2 > they are functions
Student1 > that's the error
Student1 writes answer-->restar(5,3);
Student2 > that was the mistake
Student1 > check answer2
Student1 > check your answer
Student3 > that's it!! in the processar method puts resta, instead of 
restar, really simple, wasn't it?
Student1 writes answer case 2: resultado=restar(5,3);
Student1 > check answer3
Student2 writes answer-->resultado=restar(3,5);
Student2 > check answer2
Student2 > check your answer
Student2 > check answer2
Student1 > check answer3

***********************PROBLEM N°11***************
Student2 > 11
Student1 > I'm on number 11
Student2 > me too
Student1 > let's think
Student2 > the mistake is return suma?
Student2 writes answer-->return suma;
Student1 > why?
Student2 writes answer-->return suma;
Student2 > check answer2
Student3 > we're incredible!

***********************PROBLEM N°12***************
Student2 > that was easy
Student1 > too easy
Student1 > yes
Student1 >
Student1 > in the condition
Student2 > yes, it's the output for 3
Student2 > I think I know what it does. It calculates whether N is a 
primary number
Student1 > the program doesn't execute the while
Student2 writes answer-->El 3
A. Advantages
C Couple
CAL Computer Aided Learning
CSCL Computer Supported Collaborative Learning
CSCW Computer Supported Collaborative Work
D. Disadvantages
GM Group Model
ITS Intelligent Tutoring System
LTM Long Term Memory
Nu Number of times
OO Object-Oriented
SM Student Model
SS Simulated Student
SSBM Simulated Student Behaviour Model
To. Total
WSS Without Simulated Student
ZPD Zone of Proximal Development
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Relationship between CSCW and CSCL</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>The tasks of programming</td>
<td>47</td>
</tr>
<tr>
<td>3</td>
<td>Simplified model of general troubleshooting</td>
<td>59</td>
</tr>
<tr>
<td>4</td>
<td>Overview of the model showing flows of information</td>
<td>79</td>
</tr>
<tr>
<td>5</td>
<td>SSBM architecture showing flows of information</td>
<td>82</td>
</tr>
<tr>
<td>6</td>
<td>Off-topic conversation detector</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>Passive student detector</td>
<td>85</td>
</tr>
<tr>
<td>8</td>
<td>Learning Problem Detector</td>
<td>86</td>
</tr>
<tr>
<td>9</td>
<td>Groupware time space matrix</td>
<td>92</td>
</tr>
<tr>
<td>10</td>
<td>Current HabiPro interface</td>
<td>93</td>
</tr>
<tr>
<td>11</td>
<td>Previous interface with one answer window</td>
<td>94</td>
</tr>
<tr>
<td>12</td>
<td>HabiPro architecture</td>
<td>96</td>
</tr>
<tr>
<td>13</td>
<td>Exercises correctly solved in the pre-test</td>
<td>133</td>
</tr>
<tr>
<td>14</td>
<td>Students' assessment of the assistance offered by the Simulated Student</td>
<td>139</td>
</tr>
<tr>
<td>15</td>
<td>Students' assessment of assistance offered by the human student</td>
<td>141</td>
</tr>
<tr>
<td>16</td>
<td>Students' assessment of the Simulated Student's participation</td>
<td>146</td>
</tr>
<tr>
<td>17</td>
<td>Students' assessment of the human student's participation</td>
<td>146</td>
</tr>
<tr>
<td>18</td>
<td>Students' reaction towards HabiPro</td>
<td>150</td>
</tr>
<tr>
<td>19</td>
<td>Awareness of having learnt</td>
<td>152</td>
</tr>
<tr>
<td>20</td>
<td>Results to the question: &quot;Do you think that you would have learnt more alone?&quot;</td>
<td>155</td>
</tr>
</tbody>
</table>
Table 1: Comparison of student/tutor dialogue vs. Student/peer dialogue........................................ 28
Table 2: Systems for teaching programming................................................................. 76
Table 3. SS Interventions to avoid passive........................................................................ 89
Table 4. SS Interventions to help students learning ..................................................... 90
Table 5. SS Intervention to avoid off-topic conversations........................................... 91
Table 6. Types of interventions in the case of passive Students................................ 102
Table 7. Type of interventions to improve students learning...................................... 104
Table 8. Types of intervention to avoid off-topic conversations................................. 106
Table 9. Systems to teach or to learn programming...................................................... 115
Table 10: Advantages and advantages of starting with a determinate version of HabiPro.. 119
Table 11. Number of exercises solved in each session................................................ 124
Table 12. Test of normality............................................................................................. 125
Table 13. Descriptive Statistics....................................................................................... 125
Table 14. Wilcoxon Signed-Rank Test............................................................................ 126
Table 15. Contrast Statistic (b) ..................................................................................... 126
Table 16. Frequencies..................................................................................................... 126
Table 17. Contrast Statistic (a) ..................................................................................... 126
Table 18. Test of Normality............................................................................................ 127
Table 19. Test of Homogeneity of Variance................................................................... 128
Table 20. Statistics to measure the relationship among variables.............................. 129
Table 21. Paired Samples Test....................................................................................... 129
Table 22. Descriptive Statistics...................................................................................... 129
Table 23. Ranks................................................................................................................. 130
Table 24. Contrast Statistic (b)...................................................................................... 130
Table 25. Frequencies.................................................................................................... 130
Table 26. Contrast Statistic (b)...................................................................................... 130
Table 27. Normality Test................................................................................................. 131
Table 28. Test of homogeneity of variance.................................................................. 131
Table 29. Ranks................................................................................................................. 132
Table 30. Contrast Statistic (a)...................................................................................... 132
List of Tables

Table 31. Frequencies......................................................................................................................132
Table 32. Contrast Statistics (a)......................................................................................................132
Table 33. Mean of Exercises Solved in Pre-test..............................................................................134
Table 34. Comparison of Means......................................................................................................134
Table 35. Ranks.................................................................................................................................135
Table 36. Contrast Statistic (a).........................................................................................................135
Table 37. Number of times that students had problems in solving the exercises.........................136
Table 38. Number of Times that Students Needed Help and SS's Interventions............................137
Table 39 a. Frequency Statistics.......................................................................................................140
Table 39 b. Frequency with which students valued the SS' help......................................................140
Table 40 a. Frequency Statistics.......................................................................................................141
Table 40 b. Frequency with which students valued the Student's help..........................................141
Table 41. Number of times that there were passive students. Version without SS......................142
Table 42. Results related to the passive students' behaviour............................................................144
Table 43. Results related to off-topic conversations.........................................................................148
Table 44. Number of Times of Off-topic Conversation. Version Without Simulated Student.........149
Table 45. Frequency of students' assessment towards HabiPro......................................................150
Table 46. Frequency Related to the Awareness of Having Learnt....................................................151
Table 47. Normality Test for whole the sample................................................................................153
Table 48. Normality Test for each sample.........................................................................................153
Table 49. Mean number of exercises correctly solved by each sample..........................................153
Table 50. Independent Samples Test...............................................................................................154