Increasing Consonance and Resonance in Agile Teaching Methodologies

Angela Guercio*, *Member, IEEE* and Paolo Maresca+, *Member, IEEE*

***Abstract* - In a cooperative environment technical excellence and high quality students' artifacts is what teachers strive to achieve while educating computer science students and facing the challenges of this new century. When agile techniques and accelerators and injected in the process in a cooperative environment the consonance and resonance in groups increases. This speeds up the learning process and the quality of the material produced by the students improves. Two observational studies at Kent State University at Stark and Ohio University are described in this paper. The studies observe the usefulness of using agile teaching techniques and analyze the quality of deliverables produced. A post questionnaire gathered students' feedback. The observation shows that cooperative learning produces better results than individual learning however consonance and resonance must be reached before the speed is achieved.**

***Keywords* – Collaboration, Agile Methods, Computer Science Education.**

I. Introduction

A group of working people can be seen as a strongly connected network with its own life and evolution. There are some aspects of programming in group that can be guided. In a group programming activity is important to define some objectives for the activity and let the harmonization process of the groups to self organize the group and to produce an evolution within the group itself.

A group, as a strongly connected network, can be seen as a vital system [2, 4, 5] that is able to evolve and modify its structure due to internal or external modification agents. If we consider the following definition of vital system: *"A set of components interacting with each other in a coordinated manner, directed and guided toward the pursuit of an end "* [25] it is clear that the network in question (the group) is intrinsically a vital system since in addition to being communicating and structured it can be reconfigured through shared goals.

*Angela Guercio is with the Department of Computer Science, Kent State University at Stark, aguercio@kent.edu

+Paolo Maresca is with the Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione (DIETI) of the University of Naples, "Federico II", Napoli, Italy, paomares@unina.it

However, in order for a vital system to be capable of achieving objectives, it is necessary that the relationships created inside the network can be qualified in terms of consonance and/or resonance. In music, consonance (from the Latin *cum+sonare* "to play with") provides for the listener the impression of stability and repose contrary to the impression of tension or clash obtained in dissonance. As in an orchestra, its members must reach a level of consonance before playing together. Then this consonance must be transformed in a resonance that lets the music vibrate and permeate throughout the air. In a similar way, we believe that a working team must reach a level of consonance before their work starts to resonate. The concept of consonance in a team refers to the potential mutual compatibility of the structures (groups), while that of resonance actualizes the concept of consonance by making possible an efficient operational collaboration that aims to achieve a shared goal. It is interesting to consider that the ability of an individual to interact with other subjects is characterized by the action of two forces [19]: the first one (consonance) which is fundamental to reach a state of harmony; the other one (competition) in opposition to the first one creates resistance to collaboration. The presence of these forces during the interaction implies, as the law of Requisite Variety states [2], that variety absorbs variety, i.e. a change of variety within a system aligns all the different varieties existing in the system. The group is comparable to a vital system whose varieties, in terms of categories of values that each participant in the group holds, must be aligned to be consonants.

A Viable System Approach (VSA) [6] offers schemes of interpretation useful for analyzing and governing the structure of the relationships and the process of interactions in these organizations, while taking into account the specific conditions of the relationships. In particular, for the alignment of the varieties of each individual, we suggest to check the condition of consonance at the level of the categorical values of each individual.

According to the model of the categorical variety [4, 5, 7, 8], the knowledge that identifies an effective system is constituted not only of objects or of

DOI: 10.5176/2251-3043_3.1.236

© 2013 GSTF

organized structures of information (such as databases), but above all of deeply rooted values, beliefs and opinions, as well as cognitive and interpretation schemes.

Common or harmonic values, present in cooperating people, act as facilitators of interaction between different actors and will accelerate the attainment of the consonance which is fundamental for the creation of an effective project-system characterized by informal and temporary relationships. Consonants categorical values generate a gravitational center which attracts the shared goal. What we observe and it is particularly interesting, is that in these cases there is less need for a role of government, because the alignment between the actors naturally emerges from the bottom.

The question is: "*How can individuals who possess different values, different patterns, and different cognitive and behavioral models interact effectively as nodes in a networking organization aimed at achieving shared goals*?"

Obviously this occurs through a process of spontaneous governance that starts from the bottom and leaves to its participants how to manage the use of the instruments and mechanisms of collaboration. The teacher as a coach must then train their teams to identify points of consonance and teach them to reach the level of resonance in the fastest way.

Working in group or in teams has become more than ever an essential part of the learning process in a teaching environment in this century whose challenges are generated by the globalization of our world. In particular, in the area of Computer Science, we must prepare students to the fast changing world requirements which expect cooperation across the globe and at the same time fast production of good quality artifacts. The authors believe that while teaching, we can observe the groups as real vital systems. As a consequence by working on elements that stimulate the concepts of consonance and resonance we are able to inject some accelerators in the groups' development: accelerators that can speed up both the learning process and the quality of the product.

With this goal in mind the observational studies presented in this work have been conducted. The first study was conducted at Kent State University at Stark and Youngstown State University with an activity that required computational thinking. 45 Computer Science students taking classes at different levels participated in the study. Some students worked in pairs, while others worked individually. The goal was to identify the quality of the work in a small group (a pair) with respect to the quality of the work performed by a single individual and to observe if a speed up in the learning process was detected in the pair with respect to the work of a single individual. A post questionnaire was used to gather feedback from the students about the experience.

The second observational study was conducted at Kent State University at Stark and involved 18 Computer Science students from the Software Engineering course. The students were paired to form small VSAs. The goal was to observe the group cooperation and the quality of the work in groups when stress factors are added such as the physical distance and the time constraints. Software tools were allowed to support the cooperative process and the design.

The paper is organized as follows. In Section 2 we discuss some of the issues in computer education raised by this century. In Section 3 we show how agile software techniques can be used in a teaching environment. In Section 4 we identify two agile techniques that will be used in the two observational studies that are discussed in Section 5. Finally in Section 6 we present the conclusions and our future research.

II. Issues in Education

Computer education is an extremely important topic in an era that develops so quickly and that has opened the doors to large communities of people and interests. As stated in the ACM Computer Science curriculum 2008 Computer Science education must seek to prepare students for lifelong learning that will enable them to move beyond today's technology to meet the challenges of the future. The Computer Science community recognizes that it is important to identify the fundamental skills and knowledge that all computing students must possess [1]. The ability to cooperate is one of them. E-learning has shown great potential in speeding the learning process among people physically distant. The explosion of massively open online courses (MOOCs) is providing and will continue to provide alternative avenues of learning for those who are looking for self-paced learning in their midst of their daily life and for those in search of an alternative to the costly education [27]. However in those courses beyond a strong self-discipline, cooperative projects across the distance require high collaboration abilities.

In this century where students work across the distance an updated curriculum must be accompanied by a proper set of teaching techniques that are aligned with the fast innovative process, that guide the learning experience, and help in the formation of the individuals. Teaching technical excellence and good design acquisition in a Computer Science curriculum require fostering cooperative abilities and achieving excellence in the quality of the students' deliverables in a timely manner. The question is "*while teaching Computer Science, are there techniques that can be*

© 2013 GSTF

set in place to foster cooperation and to improve the quality of material produced by the students given certain time constraints?"

In the attempt to answer this question we observe that there is a certain degree of similarity between the goals to be reached in software development and the goals to be reached in teaching. As a consequence we think that applying some software development methodologies and techniques in a teaching environment can be beneficial. In particular we believe that agile teaching methodologies, as we explain in next section, can be used to speed up the learning process and the quality of the material produced by the students.

III. The Agile Metaphor

In this section we first briefly recall the main aspects of the agile methodology and then we incorporate some of its techniques in a teaching environment to develop an agile teaching methodology.

Agile computing [3] is a new computing paradigm designed to overcome the needs of modern-day system software development. Teaching in an agile way involves students in an agile process that prepares them for the real world. Since agile processes are used in software development as well as in business and manufacturing [14] we believe that it is possible to use the agile teaching both in courses where programming and software design is required as well as in courses where computational thinking [32] is aimed.

The word agile refers to something that is fast to adapt, extremely flexible and quick in movement. In the area of Software Engineering, research has created efficient tools and methodologies for the software development that increase the production of software systems while maintaining high quality standards. Tools such as the Unified Modeling Language (UML) [12] have been used to help engineers in the design of software systems. Development methods such as structured methods, object-oriented approach, refactoring, etc., have been used to produce software that is more flexible and that stands the continuous changes produced by the dynamic set of requirements. The principles behind the Agile Manifesto [10] point the attention towards "individuals and their interactions" over "processes and tools", "working software" over "comprehensive documentation", "customer collaboration" over "contract negotiation", and finally "respond to changes" over "following a plan". Several agile techniques have been created and selected to favor such principles; for example pair programming, refactoring, works in team, short stand up daily meeting for quick update and continuous interaction, etc.. Most of those have become core principles and practice of eXtreme Programming (XP), an agile software development technique that fosters the principles of the Agile Manifesto [9, 11, 26].

We believe that in the classroom, teachers and students face some of the same difficulties that software engineers and customers face while developing software in the real world. In agile software development, small teams of engineers (often in pairs) produce quick executable deliverables that satisfy the customers' requests. In a teaching and learning environment students, as engineers, are required to produce deliverables (i.e. solution to problems, completion of homework, small programs, etc.) to satisfy the teacher's request who is their customer. We do understand that in reality the teacher often wears two hats. It plays the role of the customer, as the person that must be satisfied, as well as the role of the expert or the coach of the team while guiding the students in the process. However for simplicity we refer to the teacher simply as the customer.

As the customer, the teacher provides requirements to the students; the students interact with the teacher for clarifications of the requirements, for problem specifications, and for possible changes of such requirements. Then the students design and implement the solution that is later delivered to the teacher. In some cases, as in the case of a class project development, the teacher returns the material to the students with feedback that forces the students to adapt or rethink the produced material to create new deliverables. As in the case of the software development, high quality of the deliverables is expected and adaptability of the generated material increases the chance to produce the deliverable within the deadlines. For example, if during a project a student is collecting and printing large amount of material, it would be wise to modularize the collection of the material (i.e. in chapters, in sections, etc.) and avoid page numeration in order to be able to add the last minute additional material in place without reorganizing the whole collection.

The agile process is geared towards the satisfaction of the customer which becomes the success of the team or the company, the teaching process is geared towards the satisfaction of requirements set forth by the teacher which becomes the success of the student.

While pondering at these similarities we have asked ourselves: *can we increase the learning speed and quality of material produced by students by using agile techniques in a teaching environment?* In agile software development, good time management together with agile techniques is a good recipe for success. Similarly we expect in a teaching

© 2013 GSTF

environment that good time management together with some agile teaching techniques is a good recipe for the success of a course.

We wish to clarify that the methodology is only one of the components of the agile development. The other two components are the tools and the collaboration. A good balance between these components provides a really effective teaching agile process. The crucial point is represented by the collaboration. In a team the communication is very important. It is a process through which it is possible to coordinate the work of the team participants. This process requires the existence of consonance and resonance within the team. Only with these two characteristics we can be sure that the different activities performed by the team participants converge towards a common goal. In other words, the factors that influence the cooperative development are three:

1. the communication, which implies the existence of consonance and resonance, and aims to "harmonize" the team before their "performance"
2. the coordination, i.e. the set of activities required to conduct the work in an autonomous way, obtained by dividing the work in tasks or subtasks, by planning the meeting of verification, by structuring a plan of work and so on.
3. the cooperation, i.e. the set of activities that the team will perform together in order to reach the target goal.

With these ideas in mind we conducted two observational studies at two different universities by using computer science students of different levels of expertise. In addition to selecting two agile methodologies, we have chosen to leave the students free to choose both the tools required for the process as well as to establish their own communication. This activity to let free the students was then monitored with a questionnaire after the development.

IV. Selection of Agile Techniques

In this section, we briefly describe two core principles of eXtreme Programming (XP) [16], an agile software development technique, which have been used for two observational studies: pair programming, and refactoring. Both techniques have been applied in a cooperative environment and in one study stress factor such as high time constraints and distance have been added.

Refactoring [18] is the process of improving the design of code without changing the functionality. Problems in the low quality of code can be addressed by refactoring the source code. *Clean code* [21] is both maintainable and extensible, which are two benefits that are essential for high quality code. The process of refactoring is performed via an iterative process that analyzes each section of the code and applies the selected refactoring rules. Such rules perform minimal changes in the structure of the code but do not change its functionality. In the end the refactored code is easy to read, well-organized, modularized, and documented. Refactoring is a very common activity since most of the time developers work with existing code and allows for improving the quality of code produced.

Pair programming has gained interest as a tool that helps build better software in a more efficient and agile manner [15]. Pair programming is a process where two programmers, a driver and a tactician, synergistically work towards the solution. The driver controls the keyboard and focuses on the task of coding while the navigator observes and reviews the work of the driver and focuses more on the strategic architectural issues [30]. In reality in pair programming, there is more communication during the work than what is indicated by Williams [33]. While pair programming is beneficial, it is also controversial [29]. People either love it or hate it. Many studies have been conducted in pair programming both in a software development environment as well as in academic environment that identify its benefits and its downsides [13, 17, 20, 22, 23, 24, 28, 31].

While pure pair programming involve solely the activity of writing code in pair, we believe that the methodology used in pair programming can be extended and applied as a collaborative learning technique to computational thinking [32] activities and to learn-by-example activities.

The selection of these techniques for our observational study is dictated by the need to achieve in team cooperation products of high quality. We believe that while refactoring injects quality in the final product, the activity in pair injects speed during the production. Based on these observations we have chosen to extend pair programming to a computational thinking activity which is part of the first observational study described in this paper. The second observational study, instead, incorporates both pair programming and refactoring.

V. The Hailstone Sequence

In this first observational study we have extended the pair programming technique as a collaborative learning technique by replacing the pure coding activity with computational thinking activities and learn-by-example activities. While the study did not require any special programming ability, the use of analytical and computational thinking was required. The study involved students of an introductory CS course who were asked to answer a set of questions regarding the “hailstone sequence”. The hailstone

© 2013 GSTF

sequence starts with any positive integer and produces the next number in the sequence in the following way: if the number is odd, then multiply it by 3 and add 1; otherwise, divide it by 2. It is conjectured that, no matter what positive integer you start with, the hailstone sequence eventually reaches the pattern 4-2-1. This conjecture has yet to be proven. An algorithm in pseudo code and a webpage that simulated the generation of the hailstone sequence were provided.

See Table 1 for the questions asked in the study. The first 3 questions were designed to observe the behavior of the sequence. For example, Question 3 asked to "identify a starting number for which the hailstone sequence is at least 30 numbers long, to compute the length of that sequence, and to observe how long are the hailstone sequences associated with the new starting number if 1 is added and subtracted from that starting number." Question 4 was designed to identify the ability to apply specific knowledge in a more a general context. Finally question 5 was designed to observe computational thinking ability. In the questions students were asked to modify the hailstone sequence to generate an infinite sequence that starts at any given number and alternates an even number with an odd number.

TABLE 1. THE HAILSTONE STUDY

	Questions
Q1	What is the smallest starting number that generates a hailstone sequence with a length of at least 15?
Q2	Identify a starting number for which the hailstone sequence is at least 30 numbers long. What is the length of the sequence? If you add and subtract 1 from this starting number, how long is the hailstone sequences associated with the new starting numbers?
Q3	What is the length of the hailstone starting at 100? Starting at 200? Starting at 400?
Q4	In general if the hailstone sequence starting at some number *N* has length *L*, how long would the hailstone sequence starting at *2N* be? Explain your reasoning.
Q5	Can you write an algorithm in pseudo code similar to the one you have seen for the hailstone sequence for the generation of an infinite sequence that starts at any given number and alternates an even number with an odd number? (NOTE: There are many possible sequences that can be generated. Write an algorithm that is generic. The only requirement for the sequence is that the numbers must alternate odd and even numbers.)

With the exception of writing a small algorithm in pseudocode, no coding was required. A web page that contained the hailstone sequence generator was used during the experiment for the observation of the sequence's behavior.

The focus of this study is primary the cooperation and especially the consonance achieved by the team. Because of this the students were separated into two subgroups. One simulated a VSA of two components and the other a VSA with a single component. 45 students participated in the study, both undergraduate and graduate across at Kent State University at Stark, and Youngstown State University. Of the 45 students involved in the study, 15 worked individually and 30 were grouped in 15 pairs. Only one computer was available for each VSA pair. Each student in the VSA had to act alternatively as driver or navigator, and they exchanged roles at the beginning of each question. The role of the driver was to use the keyboard and to interact with the program and the machine. The navigator, on the other hand acted as a second pilot, observed the driver and engaged in discussion by proposing alternative paths or solutions, by correcting mistakes, or by guiding the driver throughout the activity. The students were asked to spend up to 4 minutes for each question. They timed themselves on paper for each question, reported their difficulty and confidence level in a small table and move on to the next question.

The answers to the study were graded as correct or incorrect and no partial credits were given. Therefore even a small mistake would make the answer incorrect. While this option would cut dramatically the number of accepted answers, it will help us to observe only the answers of best quality. It is worth noting that the rigor used for the evaluation of the responses served to check the degree of consonance reached by each VSA team.

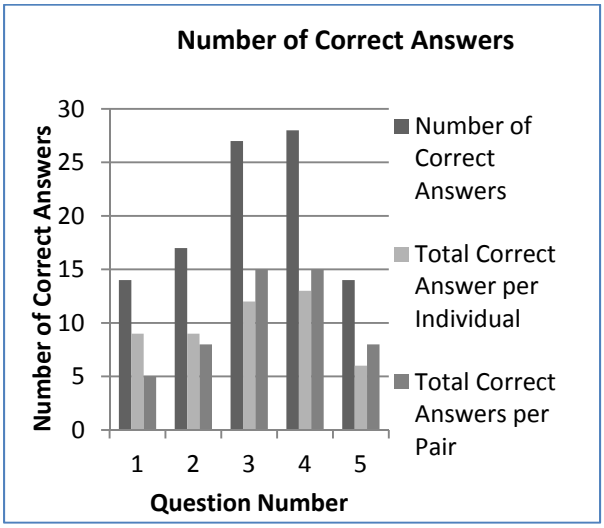

Fig. 1. Total number of correct answers.

As we can see in Figure 1, with the exception of questions 1 and 2, the total number of correct answers is greater in a pair than individually. We believe that this due to the fact that the pair at the

© 2013 GSTF

beginning had to establish a pattern of communication thus reducing the amount of time that would be spent in producing the correct answer. After the first training attempt in question 1 we see this training gap reducing in question 2 and it definitely disappears in all the remaining questions. This means that the team has reached consonance and the resonance is shown by the largest amount of correct answers per pair with respect to individual in question 3, 4, and 5.

This provides us with an empirical result of what we stated in the previous sections about cooperating vital systems. Before reaching a resonance which provides an "amplified" performance, the team is required to reach a consonance of the components of the team and this requires additional time than in single individuals. However, after this transitional stage a team offers better performance than a single individual.

If we observe the degree of difficulty perceived by each student per question we see that question 2 together with question 5 was considered among the most difficult questions. This makes us think that it is possible that the training gap would have been filled faster if an easier question had been encountered at the beginning of the study.

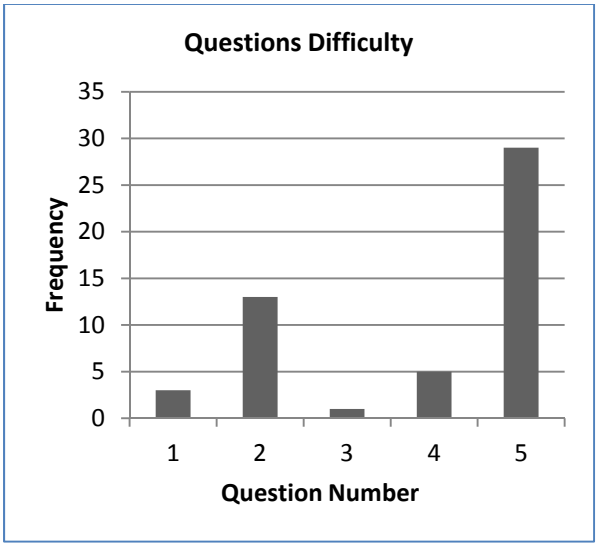

Fig. 2. Degree of difficulty per question.

The results also show that pairs produce quality results in less time than singles as it can be seen in figure 3, even though more communication is involved during their activity and that the extension of the pair programming technique can be used in other context were pure coding is not required.

The students' comments collected in a post questionnaire indicate that the majority of the students who worked in pair thought that the experience was beneficial. Some comments related to the question "Did you feel the pair programming environment helped you in this assignment?" are given below:

- "Yes working with another person helped us to figure out how to answer the questions"
- "Yes, we tend to complement each other"
- "Yes, reassurance and quicker work."
- "Yes, because I'm lost and he helped explain"

Only 3 students (out of 45) commented that pairing was slowing them down or that they didn't think they received any benefit from it. Here are their comments:

- "No I don't work well with other people and I felt my partner was holding me back."
- "It would be if we were both at the same level of programming."
- "Working in a pair had no effect on the difficulty of the assignment."

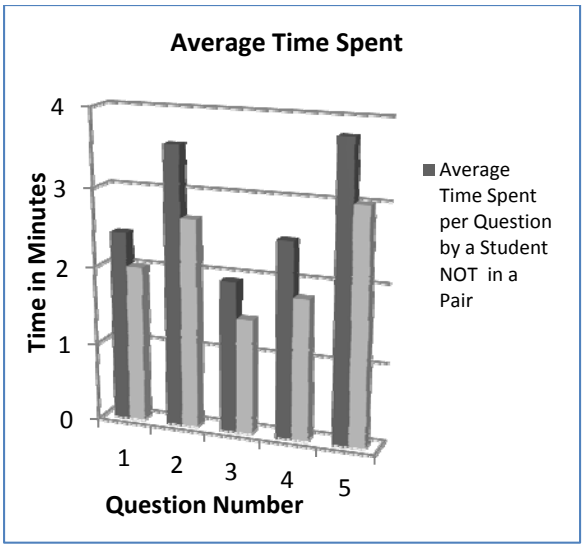

Fig. 3. Average time spent per question by pairs vs. individuals.

It may be interesting to observe why 3 VSA failed to cooperate. In other words, it should investigate better categorical values of those participants. In fact, an effective system of cooperation is composed primarily of deeply rooted values, beliefs and opinions, as well as cognitive schemas and interpretation. Common or harmonic values, available in people who cooperate, act as facilitators of interaction in order to accelerate the achievement of the consonance.

Categorical consonants values generate a gravitational center that attracts toward the shared goal. What is particularly interesting is that in these cases there is less need for a role of government, because the alignment between the actors naturally emerges from the bottom and that has been found in the majority of the groups. Discordant categorical values generate instead "cognitive traps", i.e. those

© 2013 GSTF

places where the VSA gives up due to obstacles to learning. These obstacles are caused by widespread opinions, prejudices, syndromes (e.g. "prima-donna" syndrome, too many details, search of the guilty person, etc.) and they do not allow the consonance and consequently the resonance.

The work of removing these obstacles before a working session may be of help to improve the results.

VI. Tools, Collaboration and Distance: the Refactoring Study

The previous study has identified an important problem that arises in pair activities: there is a communication gap at the beginning of the working activity which can or cannot be filled. When this gap is not filled the result is chaos and frustration of the team participants and the quality of the work suffers.

In an agile teaching environment where both the quality of the product and the time required for its production are essential the minimization of this gap is required.

The second study, named the *"Refactoring Study"*, was conducted at Kent State University at Stark to observe consonance and resonance of VSA under stress with some process accelerators.

The study used an agile application of pair programming activity in the context of refactoring. Nine pairs of CS undergraduate students from a Software Engineering course participated in the study. The participating students had never applied refactoring before.

The refactoring study was conducted over 48 hours. The code of a system which implemented a Video store that keeps track of rented movies by customers was used. The code consisted of 5 classes, 2 test cases, a makefile and a readme file containing information on how to build the system. Specific requirements of refactoring of part of the code were given. Additional questions required code comprehension and reverse engineering application. Out of the 8 questions, five of them were specifically on code refactoring. See Table 1 for the type of refactoring asked. As an example, the exact text of the Replace Temp with Query refactoring asked in Question 4 is presented below.

Question 4: Loops that do more than one thing at a time are more difficult to comprehend and extend in the future. The loop in method `statement` *is performing multiple duties; including accumulating the total charge for all movies. Perform a Replace Temp with Query refactoring to eliminate the variable* `totalAmount` *by creating a private method* `getTotalCharge` *in class* `Customer`*. Use a call to this new method where* `totalAmount` *is being output.*

TABLE II
THE REFACTORING STUDY

	Questions
Q1	Draw an initial UML class diagram
Q2	Write a unit test
Q3	Extract method refactoring
Q4	Replace temp with query refactoring
Q5	Move method refactoring
Q6	Replace type code with state/strategy refactoring-I
Q7	Replace type code with state/strategy refactoring-II
Q8	Draw the final UML class diagram from the code

To stimulate the consonance we added some stress conditions. In particular we challenged the communication by physically dividing each pair. The pairs were split in two different classes and communications was going only through virtual applications. The following accelerators were injected before the study:

- Two days before the study the students were informed of the study, how it would be performed, and who would be their partner. Students were informed that they were supposed to be able to share thoughts, code, diagrams, and any other data during the activity. At that time the students had the freedom to identify possible and preferred tools for communication and team cooperation.
- On the day of the study the students were informed that a 3 minutes face to face standing up pair meeting was performed before starting the study. In the meeting the students had to agree on the means and the tools of communication and cooperation that they would use. After that, the students spoke and cooperated only through the chosen digital means of communications.

During the study we observed that the students talked, shared their screen with their partner, wrote and shared code and diagrams, shared sketches and drawings when necessary. The team participants were asked to alternate the role of the navigator and the driver as pair programming requires. This requirement was used to share responsibilities within the group. At the beginning of the study a set of tools and applications including a common repository was made available for those groups that either did not reach an agreement or that they needed additional support; however no one was forced to use any of the listed applications.

All the teams returned the entire assignment within the given time with 62.5% of the teams scoring an A/A- grade, 37.5% of the teams scoring a B-/B/B+ grade. Of the 62.5%, 32.5 completed the

© 2013 GSTF

study with >95% accuracy, while the remaining 25% were in the range 90-95.

One team did not complete question 7, and 1 team did not complete question 8. Considering that nearly 50% of the groups stated in the post-questionnaire that more time should have been allotted to complete the work this result is highly encouraging.

In Fig. 4, we observe that students agreed on communications tools they were most familiar with. For code sharing 46% chose Skype, 28% used Email, while the remaining 36% used Eclipse, the Kent Dropbox, SVN, and Github. 32% used more than one application. For talking 50% preferred Skype over phone texting and email. Again 32% used more than one application. To share data, display, diagrams, et al, 60% of the students used Skype with respect to other applications. Only 25% of the students used more than one application.

If we observe the degree of difficulty (see Fig. 5) perceived by each student per question we see that the difficulty of question 2 is perceived by over 50% of the students and the value of difficulty continues to increase with question 3 and 4, to drop for question 5 before spiking in question 7. This means that the students had to face a degree of difficulty immediately as in the previous hailstone study. However this time we do not have the spike in Fig. 6 that we detected in the previous study, rather we observe that the students spent an amount of time proportional to the degree of difficulty of the question which is what we would like expect.

We think that the accelerators that we have injected in the process and the freedom of choice have played a speed up role in reaching the consonance in the VSA.

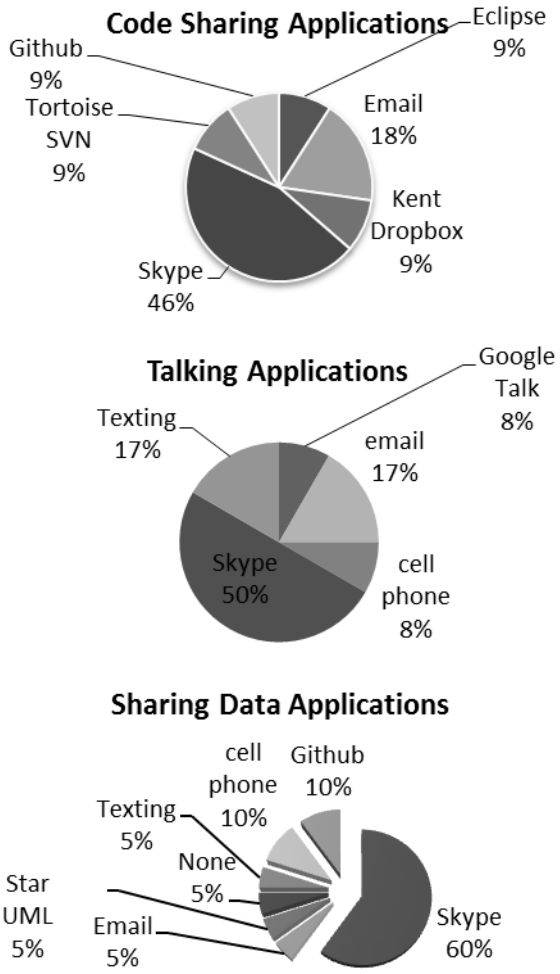

Fig. 4. The Collaboration Tools chosen by the students.

Qualitative information was extracted from the post questionnaires and the individual answers to the exercise. For those teams who completed the work it was found that there was at least one individual in these teams who was very well versed in programming. However, even these individuals who thought who were at a higher ability level confessed that their partner really helped them in achieving their goal faster and raise their limitation awareness. For instance one student mentioned: *"It was an interesting project, and it definitely made me feel like I need to learn more about C++. I understood exactly what the questions were asking me to do but when it came to the coding part Andrew was much more efficient than I was at that point."*

One group complained about the distance as a barrier to the work activity for the communication and they struggled to reach consonance. Here is the original comment: *"It was hard not being at to talk to Doug in person. We did not use the talk function on Skype and that could have helped too. We were stuck typing back and forth to each other. I think this would*

© 2013 GSTF

have gone more smoothly if we could have worked together in person."

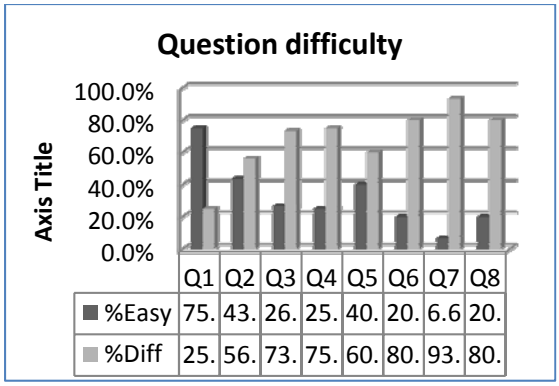

Fig. 5. Degree of difficulty per question.

The teams perceived the time constraint as these comments show: *"This assignment is much too difficult to finish in the time allotted. A week would have been much more sufficient to allow for technical issues and other problems that groups may have.."; "I thought this project would of worked better if we had more time, or that didn't take away so much of my time …"; "More time for the assignment would have made things easier".*

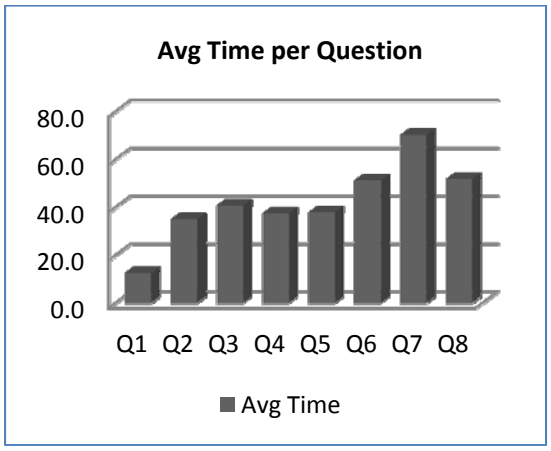

Fig. 6. Average time spent per question.

A benefit that emerges from this study is that students get to know their fellow students more and eventually interact more in class. For example one student mentioned that he didn't know his partner too well but he found he was a very competent partner.

VII. Conclusions and Future Research

Producing deliverables of high quality is what both teachers and students should try to achieve. In this paper, we show that when agile teaching techniques are used in Computer Science classrooms, the quality of the work produced by students greatly improves.

Two observational studies that use agile methodologies have been conducted. The study involved a total of 63 computer science students. The first study used and extension of pair programming as a collaborative learning technique. In the extension, the pure programming activity was replaced by computational thinking activities and learn-by-example activities. The study performed a comparison between deliverables produced by individuals working in pairs versus deliverables produced by singles and showed that cooperative learning produces better results than individual learning. Answers of better quality were produced in cooperative learning in a smaller amount of time even though more time was spent in communication especially at the beginning of the exercise when both the interaction and the protocol of communication had to be established (consonance).

The second study used the agile technique of refactoring in a collaborative environment. In the study we have injected some accelerators and added at the same time as stress factors the distance and a strict amount of time for the completion of the work. On the other side we gave them the freedom to choose the tools of communications to use in the process. The quality of the deliverables produces by the VSA was high and we have observed that the teams have quickly reached consonance and identified interpretative schema for the production of the solution, which let us think that the accelerators injected in the process have been beneficial. Empirically we have observed that they have overcome some of the time constraints by using any medium of communication they were very familiar with. While these results are encouraging, we are aware that there are still open problems to examine. The future research developments of this work will be towards the evaluation of the quality of the collaboration. The questionnaire that was delivered has examined aspects of the collaboration and the tools used in the refactoring observational study but did not examine how the teams, as vital systems, have been able to achieve the results, or to what extent they have cooperated. In other words, while observing Fig. 5 and 6 we recognize that, if we overlap them, there is a sort of pulse that makes them to oscillate in unison, as if to emphasize the drive to reach a harmony in the teams (understood as vital system) in response to a survival instinct in the team itself, necessary to complete the required tasks within the required time.

This dynamic should be more closely investigated and constitutes the third axis of this investigation, in addition to the tools and the collaboration

© 2013 GSTF

investigated in this observational study. This dynamic is part of the entropy of the vital systems that should be measured and that probably has a bell-shaped as shown in [4, 5, 19]. In other words understanding in which way the students have taken some decisions (and then answered the questions) cannot be separated from the categorical values they have adopted and from the interpretative schemes used. They can themselves be the process accelerators for their work, if intercepted and stimulated.

In fact, if we consider, that the students were left free to evolve independently in their group while working in a small and limited amount of time, it is clear that the way they operated was composed of 4 phases as shown in figure 7: chaos, complexity, complication and certainty [7, 8]. The figure, which represents the ideal curve of knowledge of a vital system, shows the entropy as a function of the information that they are involved in the process of resolution of the problem. It is clear that when the assignment is given, in that instant, the groups are in a state of chaos because they have not focused the problem yet. Immediately after, the disorder paradoxically increases because people do not know what to do (complexity) and they can only entrust and hold on the beliefs and values held by each individual to continue working (abduction). After that, the entropy is expected to decrease. This occurs when a hypothesis of solution is formulated. However it is necessary to check such hypothesis (complication). This requires the application of an interpretative schema that represents the working hypothesis which evolves into an interpretative schema of synthesis that represents the true inductive movement. If the verification has been successful then the interpretative schema of synthesis converges to the solution, the process is simplified, and the interpretative schema becomes reusable for similar problems (deductive movement).

During the process, the teams had to apply the following 3 approaches to knowledge: abduction, induction, deduction.

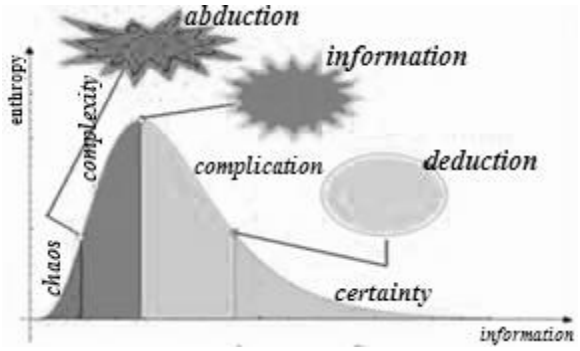

Figure 7. The 4 phases of the collaboration process

The first element, the abduction, descends from the knowledge that the individual of the vital system owns. It is influenced by the context in which he/she lives and depends on the variety information that the subject possesses. The abduction is a source of inexplicable knowledge, and irrational in some respects, that the students sometimes use to identify and create working hypothesis. The second element, the induction, is a process that, starting from special cases, derives a general law. However the process leads to a law that is not certain but only probable. Finally, the deduction is the logic process that deduces the required conclusion given some assumptions and some rules which ensure the correctness of the logical process.

In conclusion we can divide the curve in regions in which we can apply accelerators capable of facilitating the decrease of the entropy and the acquisition of the correct interpretative schemes in order to converge rapidly towards a solution. It is clear that the problem is not the chaos, which is in itself complex. The chaos is such if it is seen by the subject of the team in relation to his/her knowledge. By improving his/her knowledge the perception of the problem will improve.

ACKNOWLEDGEMENTS

Thanks to Dr. Michael Collard for the Video store assignment used in the Refactoring study. Special thanks to all the participants.

REFERENCES

[1] ACM Computer Science Curriculum 2008: An Interim Revision of CS 2001 - Report from the Interim Review Task Force,http://www.acm.org/education/curricula/ComputerScience2008.pdf, 2008.

[2] Ashby, W. R., *An introduction to cybernetics.* Chapman and Hall, London, 1956.

[3] Aydin, M., and Harmsen F., "An agile information systems development method", *Information Systems Journal, vol.* 12, no.2, 2004, pp.127-138.

[4] Barile S., *Management sistemico vitale*, Giappichelli, Torino, 2009.

[5] Barile S., The dynamic of Information Varieties in the Processes of Decision Making. In: Proceeding of the 13th WMSCI - World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, 2009.

[6] Barile, S. and Polese, F., "Linking Viable Systems Approach and Many-to-Many Network Approach to Service-Dominant Logic and Service Science", in *International Journal of Quality and Service Science*, vol.2, n.1, pp. 23–42, 2010.

[7] Barile, S., Saviano M., "Foundations of systems thinking: the structure-systems paradigm", in AA.VV., *Contributions to theoretical and practical advances in management. A Viable Systems Approach (VSA)*, International Printing Srl Editore, Avellino, 2011.

[8] Barile, S., Saviano M., "Qualifying the concept of systems complexity", in AA.VV., *Contributions to theoretical and practical advances in management. A Viable Systems Approach (VSA)*, International Printing Srl Editore, Avellino, 2011.

© 2013 GSTF

[9] Beck, K. *Extreme programming explained: Embrace change*. Addison Wesley, 2000.
[10] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M. et al., *Manifesto for Agile Software Development*. http://agilemanifesto.org/, 2001.
[11] Beck, K., and Andres, C. *Extreme Programming Explained: Embrace Change*. 2nd Edition. Upper Saddle River, NJ, USA: Pearson Education, 2004.
[12] Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language User Guide*, Addison Wesley, 1999.
[13] Bryant, S., Romero, P., and du Boulay, B. "Pair programming and the mysterious role of the navigator." *International Journal of Human-Computer Studies*, vol. 66, no.7, 2008, pp. 519-529.
[14] Cao, J. "Agile Computing." *C&C Research Laboratories NEC Europe Ltd.*, Rathausallee 10, D-53757 St. Augustin, Germany,*htpp://www.mit.edu/~caoj/pub/doc/jcao_t_agilecomp.pdf*, 2003.
[15] Coman, I., D., Sillitti, A., Succi, G. "Investigating the Usefulness of Pair-Programming in a Mature Agile Team." Agile Processes in Software Engineering and Extreme Programming in *Lecture Notes in Business Information Processing*, vol. 9, no.5, 2008, pp. 127-136.
[16] Conboy, K., Fitzgerald, B. "Method and developer characteristics for effective agile method tailoring: A study of XP expert opinion", *ACM Trans. Softw. Eng. Methodol*ogy. vol. 20, no.1, Article 2, July 2010, pp. 1-30.
[17] Dybå, T., Arisholm, E., Sjøberg, D., I., K., Hannay, J., E., Shull, F. "Are two heads better than one? On the effectiveness of pair programming." *IEEE Software*, vol. 24, no. 6, 2007, pp. 12–15.
[18] Fowler, M. *Refactoring: Improving the design of existing code*. Addison Wesley, 1999.
[19] Golinelli, G.M., *L'Approccio Sistemico Vitale (ASV) al governo dell'impresa*. Cedam, Padova, 2011.
[20] Höfer, A. "Video analysis of pair programming." *Proceedings of ICSE 2008*, 2008, pp. 37-41.
[21] Martin, R. *Clean Code*. Prentice Hall, 2009.
[22] McDowell, C., Hanks, B., and Werner, L. "Experimenting with pair programming in the classroom." *SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '03)*, 2003, pp. 60-64.
[23] McDowell, C., Werner, L., Bullock, H.E., and Fernald, J., Pair programming improves student retention, confidence, and program quality, *Communications of the ACM*, vol. 49, no. 8, 2006, pp. 90-95.
[24] Mendes, E., Al-Fakhri, L. B., Luxton-Reilly, A. "A replicated experiment of pair-programming in a 2nd year software development and design Computer Science course." *SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '06)*, 2006, pp. 108-112.
[25] Pellicano, M. *Il governo strategico dell'impresa*. Torino: Giappichelli, 2004.
[26] Succi, G., Marchesi, M. eds. *Extreme programming examined*. Pearson Education, ISBN-13: 978-0201710403, 2001.
[27] Vardi Moshe Y. "Will MOOCs Destroy Academia?" *Communications of the ACM*, 10.1145/2366316.2366317, vol. 55, no. 11, 2012, p. 5.
[28] Velasco Berba, V. "The pitfalls and perils of pair programming".http://ezinearticles.com/?The-Pitfallsand-Perils-of PairProgramming&id=356042, 2006.
[29] Williams, L. "But, isn't that cheating?" *Frontiers in Education (FIE '99)* Session 12B9, 1999, pp 26-27.
[30] Williams, L., and Kessler, R.. *Pair programming illuminated*. Addison-Wesley, 2003.
[31] Williams, L. "Lessons learned from seven years of pair programming at North Carolina State University." *Inroads: ACM SIGCSE Bulletin*, vol. 39, no.4, 2007, pp. 79-83.
[32] Wing, J. M., *Computational thinking*, Communications of the ACM (*CACM '06*), vol. 49, no. 3, 2006, pp. 33-35.
[33] Wray, Stuart. "How pair programming really works", *IEEE Software*, January/February, 2010, pp. 50–55.

Angela Guercio received her Ph.D. in computer science from Kent State University, Kent, OH in 2004, the M.S. in Computer and Information Sciences from the Knowledge Systems Institute, Chicago, in 2000 and the Doctor in Computer Science "cum laude" from the University of Salerno, Italy in 1984. She is currently an Assistant Professor at Kent State University in Ohio. She has been an Assistant Professor at Hiram College and Senior Research Associate at University of Salerno, Italy.

Dr. Guercio's research interests include programming languages, software-development environments, multimedia computing, web programming, and multimedia and visual languages. She is a co-author of several papers published in scientific journals and refereed conferences. She has been a winner of several awards and Fellowships for her research. She has chaired and participated in the organization of several international conference and coedited special issues of international journals. Dr. Guercio is a member of the IEEE, the IEEE Computer Society, and of the ACM.

Paolo Maresca is an Associate Professor of Sistemi per l'Elaborazione delle Informazioni, at the "Dipartimento di Ingegneria Elettrica e Tecnologie dell' Informazione (DIETI) of the University of Naples, "Federico II".

He is a member of the AICA and senior member of the IEEE and author of about 120 papers published in journal, conferences, books and magazines of national and international ICT topics. He is an Associate Editor and referee of international journals and coordinator of the Eclipse Italian community. He was awarded in 2011 for IBM faculty award and was IBM rational champion for 2012 and 2013.

© 2013 GSTF