Int. J. Engng Ed. Vol. 24, No. 4, pp. 671–680, 2008
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Teaching Agile versus Disciplined Processes*

PIERRE N. ROBILLARD and MIHAELA DULIPOVICI
Department of Computer and Software Engineering, École Polytechnique de Montréal, 2500, Chemin de Polytechnique, Montréal, Québec, H3T 1J4, Canada. E-mail: pierre-n.robillard @polymtl.ca

Project courses are an important component of some software engineering curricula. They are capstone projects where teams of students experience the various practices for developing software. Instructors play the roles of coaches in guiding the students during the various phases of their project. Nowadays, software development processes fall into two major paradigms. The Disciplined software process paradigm defines best practices and their relationships on the basis of roles, activities and artifacts. The Agile process paradigm, which is based on values of simplicity, communication, and feedback, uses simple practices to enable a team to tune the practices to their unique situation. The two process paradigms have great value in general and one is likely to be more efficient than the other in any specific development project. However, it could be interesting to find out how each of these process paradigms performs in learning environments. To achieve this we conducted an observational study in an academic environment. Six teams of four students developed their own versions of a software product based on the same requirements. Three teams used a Disciplined process and three teams used an Agile process. This study is based on four observations: the quality of the implementation of the requirement, the total project effort, the process activity effort and the product size. The data to support each of these observations are presented. In this study, however, the Disciplined paradigm provides less project implementation with a better realization of quality. This study indicates that the more efficient approach for capstone projects for inexperienced students in software engineering would be a Disciplined process paradigm.

Keywords: software engineering; software process; disciplined process; agile process; capstone project

INTRODUCTION

THE FUNDAMENTAL SOFTWARE DEVELOPMENT PROCESS is made up of basic cognitive activities that enable the successive information to crystallize from requirements into source code, which is the ultimate artifact. There are many ways of reaching this goal and software processes are a collection of practices that provide some guidance that could be helpful in a collaborative environment to help focus team work [1]. The Disciplined process emphasises requirements, design and various intermediate artifacts to capture part of the information crystallization process and to facilitate cognitive synchronization and sharing of information. The Agile process paradigm puts most emphasis on the source code that should be built gradually and information is shared and synchronized by direct verbal communication.

In a pragmatic sense, a software process is the set of activities and artifacts that are required to realize a software project. However, there is no general acceptance of a set of unique process principles. A software process can be viewed from different perspectives. One of the early perspectives was from a management point of view, that a software development project could be characterized as a waterfall, a spiral or an incremental software process. Later the Disciplined software process was introduced. This approach to software development promotes a discipline-based engineering process involving an effective definition of the activities to be performed, artifacts to be produced and roles to be played. This paradigm involves the production of artifacts to support most practices, effective communication and knowledge transfer [2].

In response to the work put into producing a variety of artifacts, there has been a rapidly growing interest in Agile methodologies [3, 4]. The Agile software development philosophy puts verbal communication as one of its main values and emphasizes a minimalist approach to written documentation. The 'Agile Software Development' breaks with a number of traditional software engineering practices and prescribes some special practices instead. Some of the key concepts shared by the Agile approaches specifically downplay the importance of written artefacts and emphasize verbal communication. Extreme Programming (XP) [5] is a prominent member of the family of Agile development methodologies.

Previous studies have demonstrated the advantages and the disadvantages encountered by using either of the methodologies in an academic context [6]. There is no common acceptance of the best way for team-mates to communicate and share information during software development. It is reasonable to assume that, in some circumstances, it might be better to adopt one process rather that the other [7]. Even if it is recognized that choosing the appropriate development process can improve the product quality, there is little empirical evidence of a direct link between the development

* Accepted 13 January 2008.

process and the quality of the software product functionalities that are realized. There is also little evidence about the learnability of a given software process. This paper presents studies that evaluate the types of impact of software processes in the learning environment.

The study reported here is unique in that it uses, in parallel, in the same project-based course, the two processes that provide Disciplined and Agile paradigms to software development. The purpose is to evaluate the benefits of each process paradigm and eventually determine if one process paradigm is better than the other in a learning environment based on capstone projects.

The next section presents the studio environment. It defines the student profiles involved in this observational study. It describes the project to be implemented and the two process paradigms, Disciplined and Agile, each used by three teams of students. Then we present the data and the four observations resulting from this empirical study, which are: the quality of the implementation of the requirements, the total project effort, the process activity effort and the product size. The concluding discussion stresses the importance of the Disciplined process in the learning environment and the role of the Agile process for mature students.

STUDIO ENVIRONMENT

By using the results of previous research as a foundation [8], a studio-based teaching and learning approach was adopted for the observational study. The Software Engineering Studio at the École Polytechnique de Montréal is an optional capstone, project-oriented course offered during the last term to seniors (fourth-year students) in software engineering. Teams of students must develop an operational software product based on software requirement specifications provided by the instructors. They must also follow one of the two prescribed software engineering processes: Disciplined or Agile. Participants thus acquire experience in building an operational software product through the disciplines of analysis, design, implementation, testing and management according to a process perspective. The goal of this project course is to teach students the realities of teamwork, milestones, deadlines and the use of process activities.

Student profile

According to Gagné, for a better learning, two conditions have to be satisfied: the external conditions (the particular teaching environmental techniques for facilitating learning) and the internal conditions (the prerequisite knowledge or capabilities that a student must possess) [9].

With regard to the internal conditions, all students have almost the same background. They are full-time students, who have completed a four-year curriculum in software engineering, and some of them have also completed a four-month internship in industry. They are mature students that will be junior professional engineers at the end of the semester. The course enrolment is limited to 24 participants who form six teams. Participants are selected on the basis of their cumulative records. These students are from the first quarter percentile.

With regard to the external conditions, this project course counts as two regular courses. Each student has to allow 270 hours during the 15-week semester for this course. Dedicated lab spaces are reserved for six-hours a week of mandatory team work, where instructors meet the teams. Students can (and in fact they do) meet more often in the same lab space, which they can reserve. The artifacts produced throughout the project were deposited in the project repository and validated. All student teams are in competition on the same project, developing their own versions of the system based on the given specifications. Evaluation is on a team basis [10].

A portfolio—a collection of students' collaborative work generated during the Studio—exhibits the students' effort and progress. It contains the entire set of artifacts specific to the development process, as well as the time slips of the team members. The 'effort' was measured from the students' records of their time spent on each activity or artifact produced.

Ethical considerations were an important issue. Our research protocol was approved by an independent committee at École Polytechnique de Montréal, which is mandated for supervising research with human subjects, and they provided an Ethic Certification for this research. All subjects involved in this study were duly informed of the recording as well as of the nature of the study and they all signed the letter of agreement required by the ethic certification. Ethical issues were handled according to the Canadian policies for research involving humans (NSERC 2005) [11].

Project description

The project goal was to develop a Web-based meeting management system aimed at organizers of meetings where the numbers and geographic dispersion of participants make scheduling difficult. The software product to be developed, named REPLAN, is a planning system for physical, phone or virtual meetings, using the Web as communication support. It allows meeting coordinators to send availability requests to a set of individuals so that each one can specify their personal availability periods. The set of availability periods would then be graphically represented using a special calendar tool that would allow a coordinator to visualize the relevant information at a glance, making the scheduling decision easier to take. REPLAN would allow groups of people to plan meetings by using a common database containing the available information from each of these them.

In terms of technology, the Java/Servlet/JSP family of technologies had to be used as the

foundation for all products. Two computer servers were allocated: the first—for hosting both a file system and a Tomcat Application Server, and the second—for hosting a commercial database server to be accessed by the students' applications. In the dedicated laboratory, a Java environment, a database client and all necessary software were installed on each PC station.

Process paradigms

Two different software processes providing two different approaches to software development were used: a Disciplined process called UPEDU (Unified Process for Education) [12, 13] and an Agile process derived from XP (Extreme Programming) [14]. There is no consensus as to what exactly constitutes a Discipline and what constitutes an Agile process. An Agile Manifesto was put forward in 2001, but many variations exist with variable conformity to some principles of the Agile Manifesto. UPEDU is a framework that defines a Discipline process and a large variety of Discipline processes can be defined based on different sets of artifacts and activities. This study does not compare a specific UPEDU process and an Extreme programming process, but rather the underlying philosophies.

The UPEDU is an adapted version of the Rational Unified Process (RUP). The UPEDU software process framework is well defined, and all the activities and artifact templates, as well as a case study, can be viewed on the UPEDU website. Only the relevant activities and artefacts that are useful for the studio project are retained to enable students to focus on the essential components of the Discipline software process. The UPEDU's students had been trained on the UPEDU in a previous course and the Agile's students were trained on the Agile philosophy. Since we are not comparing a specific UPEDU process with a specific AGILE process here, we use the D-process for the Disciplined process derived from the UPEDU framework and the A-process for the Agile process derived from the Agile philosophy.

Three teams (team-D) were assigned to the UPEDU process and formed the D-process group (D for Disciplined) and three teams (team-A) were assigned to the process based on Agile philosophy and formed the A-process group. All students were enthusiastic about their team selection and their process group. Team selection was proposed to the students and care was taken to balance the team in terms of students' marks and student internship experience. The process alternatives were presented to the students as a course option and not as the basis for an observational study.

There is a systematic difference between the two process paradigms for the requirements capture. D-process is a use-case driven development process. 'Use-case' describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies the user's goal. A scenario—in an example of a use-case—captures the series of interactions between the user and the system needed for the function to be achieved. Scenarios may be depicted using UML diagrams. A use-case model consists of all the actors in the system and all the different use-cases through which the actors interact with the system. The model thereby describes the totality of the functional behaviors of the system. Agile defines a 'user-story' as the smallest amount of information required to allow the stakeholders to define a path through the system. A user-story is a short description of some piece of desired functionality. It contains just enough text to explain what is to be done so that the stakeholders and the developers can both understand it in the same way. However, it is possible that developers and stakeholders could each have a different understanding of a user-story. Misunderstandings were resolved by discussions. As the Agile manifesto states 'the most efficient and effective method of conveying information to and within a development team is face-to-face conversation.' Table 1 shows the number of use-cases and user-stories developed by each Disciplined and Agile team respectively.

Table 1. Number of use-case and user-stories used by the three Disciplined (D) and the three Agile (A) teams

Teams	D1	D2	D3	A1	A2	A3
Use-case	16	17	23			
User-stories				22	15	10

The difference between the numbers of use-cases is due to the degree of granularity in decomposing the functionalities of the system. For example, for the use-case 'Create a meeting' defined by the team D1, team D3 proposes three use-cases, with more detailed information. We found the same difference in the case of Agile teams. As no levels of detail were imposed, team A3 chose to present their user-histories at a more synthesized level than the other two A teams.

Although each process paradigm provides different tools to analyze the specifications, they do not manage the level of abstraction.

OBSERVATIONS

To better understand the role of process paradigms in student project development we based our study on four observations: the quality of the implementation of the requirements, the total project effort, the effort in the process activity and the product size. The data to support each of these observations are presented and the impacts of the process paradigms on these observations are discussed.

Quality of implementation of requirements

The product REPLAN was specified with 109

Table 2. Extract from REPLAN requirements

Requirement number	Requirement statement
3.5.5.3.	Information on the availability of the participants for the ranges defined by the initiator will be posted in the window called 'Availabilities'.
3.5.5.3.1.	The component 'Availabilities' will take the form of a calendar.
3.5.5.3.2.	The availabilities mentioned by the participants but located outside the ranges defined by the initiator will be excluded from the posting.
3.5.5.3.3.	The user participant will be able to browse within the component in order to visualize or modify the state of a definite period of time
3.5.5.3.4.	The window will include a component of text in which the user could insert information for the initiator. Only the participant will be able to modify the contents of the text field.

requirements statements. Table 2 shows a sample of typical REPLAN requirements.

One way of evaluating the quality of a software product is to compare its implemented functionalities with those of the requirements specification statements. The basic assumption of the proposed measurement method is that software experts can make a coherent judgment on the achievement levels related to each specification statement. We used a three level scale to quantify the experts' judgments, as shown in Table 3.

According to this scale, a team that has successfully implemented all the requirement statements will have a total score of 218 points, which is two points for each of the 109 requirement statements.

No team actually succeeds in completely implementing all the requirement statements of REPLAN. Each product had a different number of implemented requirements and a different quality scale level. As an example, Fig. 1 shows the level of each requirement for products D1 and A1. The requirements are numbered on the x-axis from f1 to f109 and their evaluated level of implementation is presented on the y-axis by the levels 0, 1 or 2. It can be seen that some requirements have been successfully implemented by the two teams (e.g. requirements f6 to f16), while others have not been implemented by any of the teams (f61 to f66); finally, some requirements are implemented differently by different teams.

A given requirement may have been implemented at various levels even within the same process group. In order to measure the implementation level for a given process group we recorded the level for each requirement in a three-dimension vector, representing the three teams within a process group. For example, a vector level Dx(2,1,0) means that the three teams D1, D2 and D3 had levels 2,1 and 0 respectively for the requirement statement x.

Table 3. Level scale for quantifying the experts' judgments on requirement implementation

Level	Definition
2	This requirement is successfully implemented
1	This requirement is partially implemented
0	This requirement is not implemented

The requirements vector can be expressed by its magnitude definition. For example, the level vectors Dy[2-1-1] and Ay[1-2-1] of the two process groups D and A have equal magnitude from the point of view of our analysis goal, since the order of the vector components is arbitrary. The higher the level of magnitude, the better the implementation for the process group. A vector magnitude of 3.46 (square root of 12) means that this requirement was successfully implemented by the three teams of this group. It should be noted that using the magnitude as the distance measure biases the results in favour of full implementation, e.g. distance (0-0-2) = 2 while distance (1-1-1) = 1.73.

Figure 2 shows the difference in magnitude value between D and A process groups for each requirement (|D|-|A|). Each requirement statement is numbered on the x-axis from 1 to 109. Requirement statements with a zero value (1 to 4, 6 to 17, etc.) have the same vector magnitude for the D and A process groups, which means that the two process groups performed in the same way; there is no information as to how successful the implementation was. A difference value in the positive direction indicates a better implementation by the D process group, while a difference value in the negative direction indicates a better implementation by the A process group.

We observed that for one set of requirements the D-process group seems more successful, while for another set of requirements the A-process group seems more appropriate. This observation indicates the possibility of a link between the quality of the requirements realization and the process used. A thorough analysis on these two sets of requirements was carried out.

The set of requirements from statements 86 to 93 and requirement statement 19 are systematically better implemented by the D-process group. These requirements are related to the management of the calendar, which is dynamic. It is presented using different colors (green: everyone is available, yellow: some are available, red: only few are available) the dates that are more suitable for scheduling the meeting depend on participant availability. The data statuses vary as the partici-

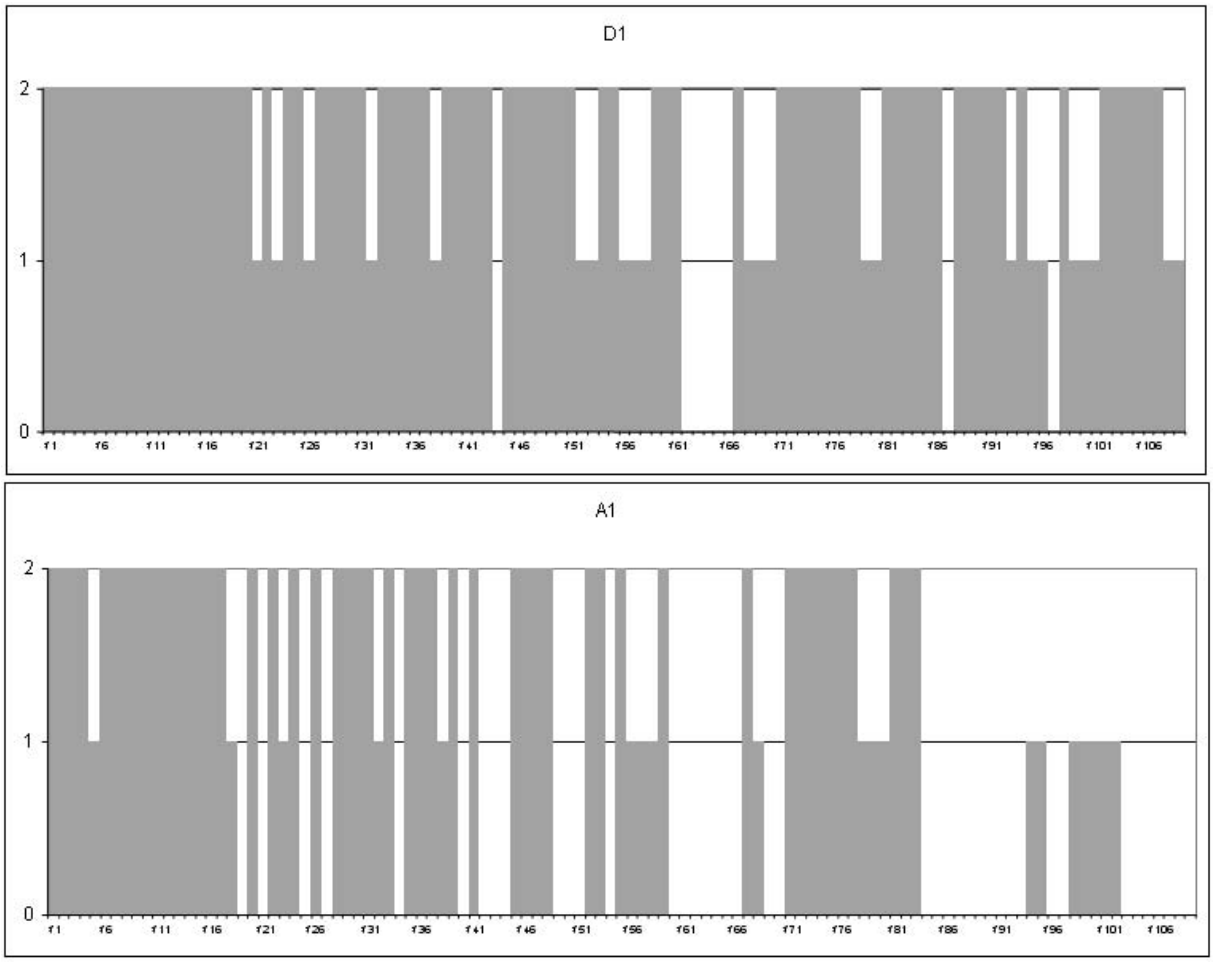

Fig. 1. Requirements implementation levels for the D1 and A1 process teams.

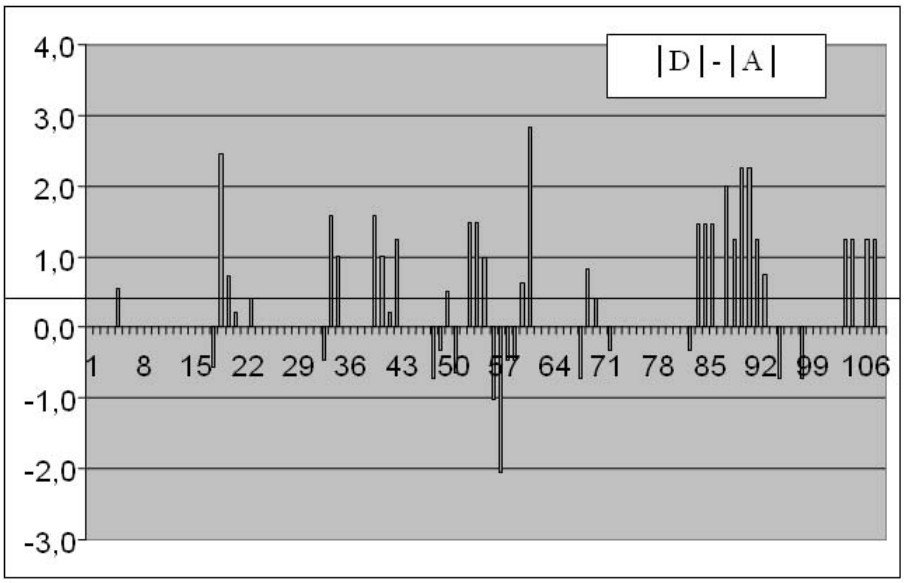

Fig. 2. Difference in vector magnitude between D and A process groups for each requirement statement.

pants send in their availability, which could be at four levels: confirmed presence, likely to be present, likely to be absent and confirmed absence. D-process teams used interaction diagrams to model these requirements and design the algorithm to manage the four level statuses and then color the corresponding dates according to the decision algorithm output. A-process teams did not see the difficulties related to the dynamic aspects of these requirements until they were ready to code it.

By then it was too late to deal with this level of difficulty and they did not implement these calendar features.

We find that process paradigms, in a student environment, may have a measurable impact on the quality of implementation of certain types of requirements. Requirements that specify dynamic functions were more successfully implemented by a disciplined approach requiring formal design artifacts. The difficulties inherent to these requirements are seen early in the process and can be thoroughly analyzed. Detailed scenarios, use-case, sequence, collaboration and state diagrams seem more appropriate for representing the details of each function. The diagrammatic representation seems to make it easier to carry out this kind of requirement. All D-process teams implemented these requirements, while none of the A-process teams succeeded. The A-process group did not have enough experience to foresee the difficulties related to these kinds of requirements.

The A-process group obtained a slightly higher score for the quality of implementation of the requirement statements specifying static functions. It seems that for this kind of requirement, succinct user-stories are suitable since most of the details are provided by direct communication between the client (a role played by the instructor) and the development team members. All screen outputs implemented by the A-process group were better looking.

Total project effort

All data used for this observational study were collected through on-line effort slips that were filled in by the participants. The data collection scheme included the following data elements for each effort slip:

- Participant ID
- Date
- Activity performed (one short sentence in free format)
- Output artifacts
- Effort expended (with a half-hour granularity)

Participants had been trained to fill out the effort slips correctly. Effort slips were checked regularly by the instructors in order to ensure their validity. Slip correction requests were issued to participants upon detection of anomalies. A customized software tool was used to record the data from each participant.

This analysis takes into account only the effort that is relevant to the software process activities. The effort that was not taken into consideration is, for example, effort spend on training, on setting up the development environment and on preparing the project presentations for mid-term and final deliveries. Figure 3 shows the effort measured in hours spent by each team on their project. The total effort ranges from fewer than 400 hours to more than 700 hours. Teams are listed in decreasing order of total effort.

The three disciplined process teams, D-process group, spent similar effort on the project while Agile process teams, A-process group, experienced a large spread. Disciplined processes were easier to manage since iterations and milestones were defined beforehand. Projects based on Agile processes required globally more effort. The instructors speculate that some of the extra effort spent by the A-process teams may come from taking greater care in the implementation of the user interface, which were visually better than for the D-process teams.

The process paradigms in this course project environment have a measurable impact on the

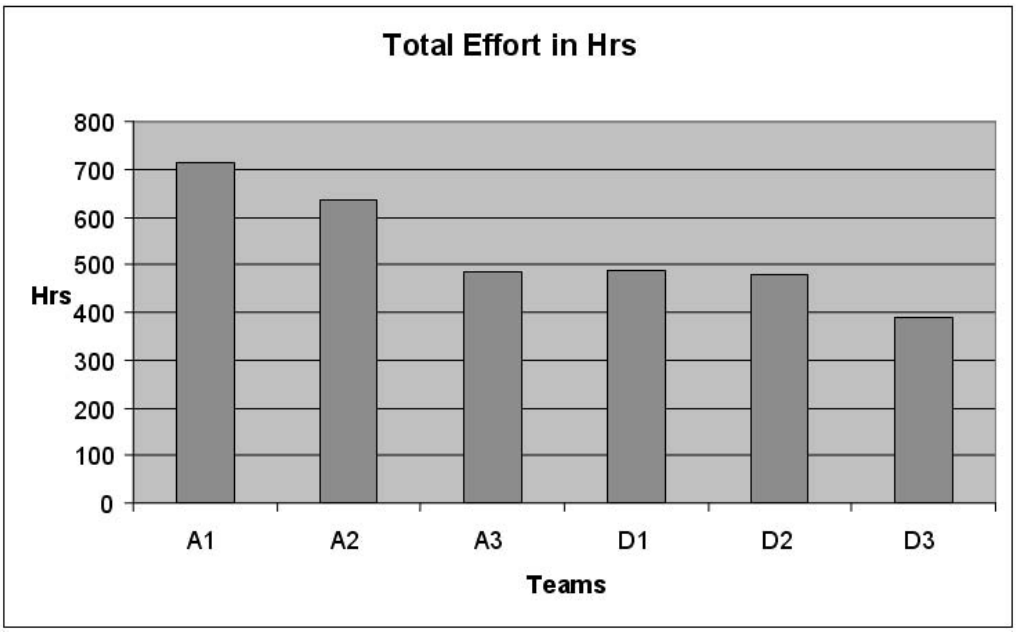

Fig. 3. Total effort in hours spent on the project for each of the A and D process teams.

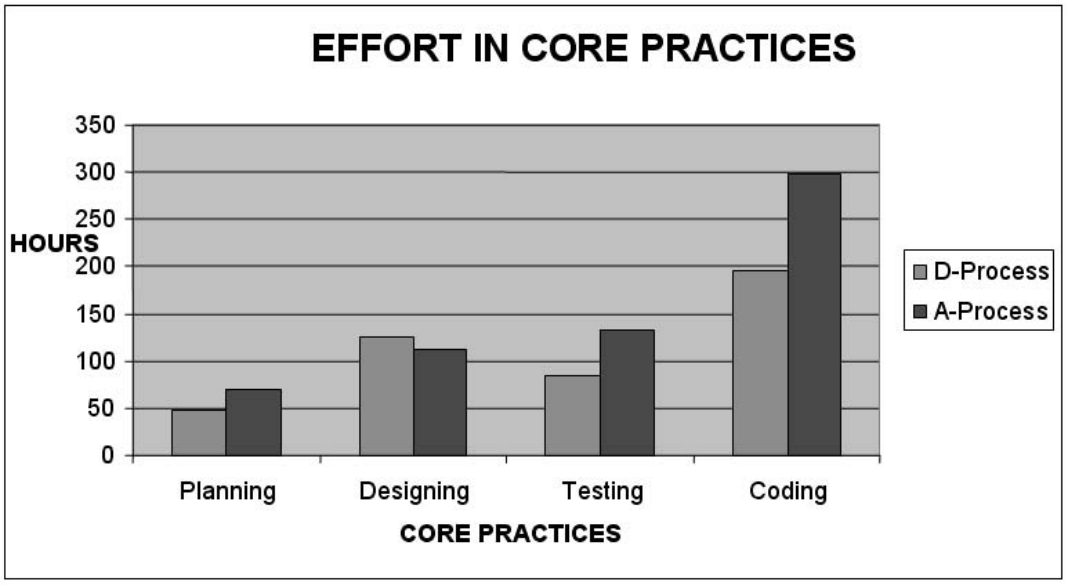

Fig. 4. Effort spent in each of the four core practices for each of the process groups.

effort required to implement the product. A-process groups spend much more effort that D-process groups. The next step is to find out in which type of activities this extra effort is spend.

Process activity effort

This section analyses the distribution of effort within the process activities. Both process paradigms are based on successive iterations. The Discipline D-process is based on activities played out by the roles in order to deliver appropriate artifacts according to defined schedules. Conversely, the Agile A-process stresses the embracing of change, collaboration, and early product delivery. A working code is considered the most important artifact of the development process—not the documentation, which exists only to support the development of the working software. The Agile process requires relatively few detailed artifacts. In order to study in more detail the effort spent in the D-process and A-process groups, we categorized the activities into four core practices areas that were required by any team developing software, regardless of the process paradigm. These core practices covered planning (P), designing (D), testing (T), and coding (C). Most of these practices are fairly well-known and self-explanatory with further details widely available in the literature [15].

Figure 4 presents the effort spent on average by the three teams from each of the two process groups within the four core practices. Planning includes all the effort required by the team-mates to manage their activities. There are more planning activities in the A-process group because they have more iterations to plan and they have to reassign their tasks more often as the project progresses. There is a greater effort in designing activities from the D-Process group because they have to produce design artifacts according to defined templates. There is a greater effort in testing activities from the A-process group because the Agile philosophy encourages a test driven approach. Finally there is much more effort put into coding by the A-process group.

The profile of the effort in the core practices for the A and D process groups is as expected. There is more effort put into Designing practices by the D-process group while there is more effort put into Testing and Coding practices by the A-process group. However, does the extra effort from the A-process group compensate for the extra effort in Designing practices from the D-process group? To better understand the meaning of the coding effort we must look at the product sizes.

Product size

The set of artifacts specific to development is related to code construction. It is by far the most effort demanding set of artifacts in the A-process. The Agile process strives for simple design and uses a simplifying technique whereby the programmers examine whether the code reflects the design they envisaged.

Surprisingly, the size of the developed product is somehow related to the process used. Figure 5 shows the size of a line of code (LOC) on the left-hand y-axis and the number of classes (NOC) on the right-hand y-axis. Teams are ordered according to Fig. 3. All D-process products have a larger number of classes (line in the upper part of the graphic) and are globally smaller than A-process products in terms of lines of codes (column high). There are many factors to explain the product size. One factor is that all the teams in the D-process group used design patterns, while none of the teams of the A-process group used them. Another factor is that none of the teams of the A-process group used refactoring because of

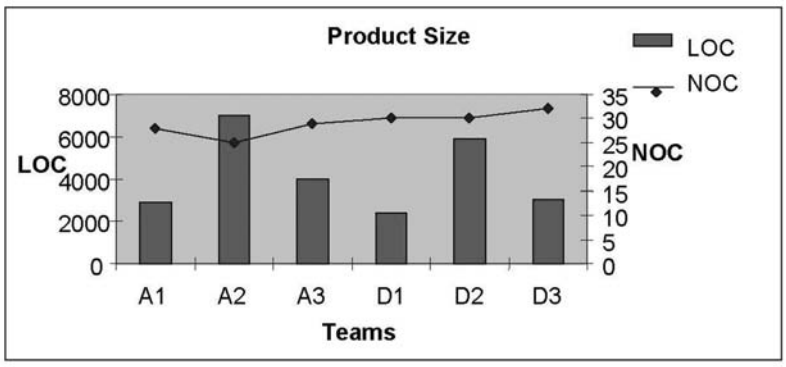

Fig. 5. Project size for each team, A and D, in line of code (LOC) and number of classes (NOC).

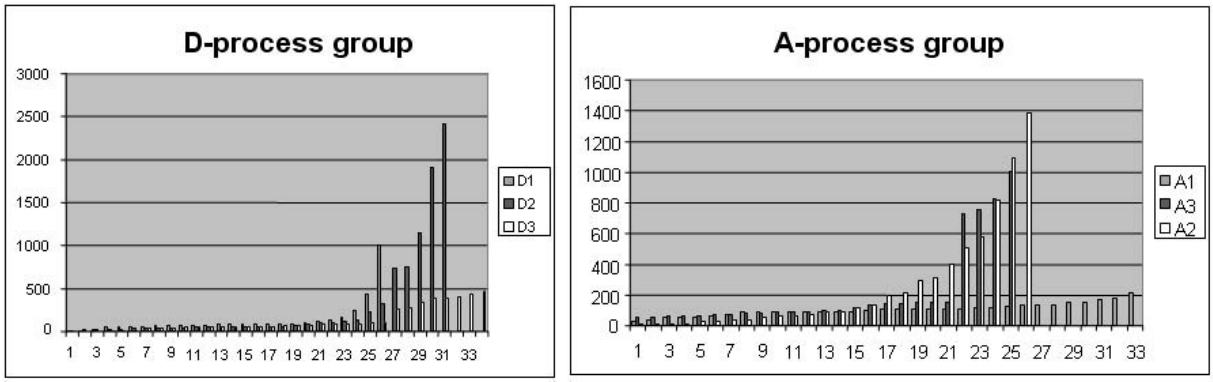

Fig. 6. Number of line of code per class for the D-process and A-process products.

the lack time and experience to do it. There was a large amount of duplicated coded in the A-process products.

Figure 6 shows the number of line of code for each of the classes of the D and A process group products. The architecture of the products developed does not seem to be related to the process groups. D3 and A1 products are both implemented based on a large number of small classes while the other D and A products have few very large classes.

CONCLUSION

This observational study is one of a kind and the conclusions are not based on extensive statistical analysis. However, it is performed according to the tradition of software engineering empirical research [16]. Although the observations are made within the student environment, some of the conclusions may be applicable to the professional environment.

The degree of validity of this study has been increased by relying on senior students enrolled in their last semester. The line between these students and novice professionals is becoming blurred [17]. From studies that have been conducted to evaluate the difference between software engineering students and the professional software developers used as subjects in empirical studies, it has been found that the differences are only minor. It has, in fact, been concluded that software engineering students may be used instead of professional software developers under certain conditions. This result does not, however, contradict the assumption that final-year software engineering students are qualified to be subjects in empirical software engineering research [18].

The aim of this observational study is to better understand the impact of process paradigms on students' ability to complete capstone projects. Disciplined and Agile process paradigms both have some positive features and some drawbacks. The same set of requirements was given to six teams of four senior students. For this elective project course, students were selected based on the quality of their academic records and the uniformity of their background. They were all full-time students enrolled in the same four-year computer engineering program with a major in software engineering. Half of the students, some on each team, had some four month internship work experience. Students were assigned, on a voluntary basis, to a Discipline process team or an Agile process team. Time slips filled out by the participants on a daily basis were used to record the effort spent on various activities. All the artifacts produce by the team were also recorded.

The time slips' data recording is the weak link in this observational study. Five approaches were used to improve the reliability of the time slip

data. First, students were trained in filling them in and they did a data analysis based on time slips from a previous year's projects during the prerequisite process course. This time slips analysis made them aware of the usefulness of the information recorded. Second, time slips were validated on a weekly basis and any anomalies were reported to the students and corrective actions were suggested. Third, time slips recording was presented to the students as a means of making them aware of the activities they were taking part in while developing a software product and it was part of the learning experience of this course project. Fourth, 25% of the team marks were allowed for the quality of the time slips. Quality is defined as the meaningful description of the activities and accurate activity duration. Fifth, students were informed that time slips content and its analysis would not be used for team evaluation purposes, but they could eventually be used for research purposes at the end of the semester. This was formally stated in a signed document that also authorized us to use their data for research purposes according to the ethic protocol for research with human subjects at École Polytechnique.

The observed facts based on this single course project of comparing Agile and Discipline processes is that the Agile paradigm required more effort, produced more code, implemented fewer functionalities but realized a nicer interface. The process paradigms seem to have little impact on the requirements analysis and the product architecture.

This severe conclusion must be interpreted cautiously. The following is a personal interpretation of the results.

The Agile software development paradigm has verbal communication as one of its main values and relaxes the importance of written documentation. Cockburn contends: 'the most effective communication is face-to-face, particularly when enhanced by a shared modeling medium' [19]. Using an Agile process, students apply 'hot communication media' to communicate necessary information when working together. Even though, as J. Smith mentions [20]: ' . . . in the books about the XP approach, the terms 'artifact' do not appear in the index. . . . it's not difficult to read through the text and pick out references that are artifacts'. A fundamental message is that artifacts should be produced only when they add the best possible value to the project.

We believe that the Agile process paradigm required experienced developers. Experienced developers might foresee difficulties in the implementation of some requirement and do a proper analysis before the coding phase. They are also likely to have enough experience with pattern designs to use them efficiently. They would do the required refactoring to improve the module implementations. Finally, experienced team workers are likely to reduce the time needed for planning activities.

The Disciplined process provides a recipe for inexperienced developers. Life cycle phases are well defined and milestones clearly identified. Templates for the various artifacts required for the analyses and design activities are provided. Team-mates are aware at the start of the project of all the steps and artifacts required to complete the project. Iterations are easier to manage.

Developing a software product is an opportunistic thinking endeavor [21]. The Discipline process provides the necessary guidelines for supporting the various steps of problem solution leading to coding. With a disciplined process students learn how to crystallize the information at various abstraction levels, which enables them to share this information in an organized way. They are more aware of the roles they are playing in the collaborative project. Once they learn and understand how software development works they are more likely to work by themselves without the guides provided by the disciplined process.

The last consideration is the nature of the knowledge [22] involved in each of the process paradigms. The Disciplined process is based on declarative knowledge that is acquired by studying concepts and theory, such as roles, artifacts (UML), and activities, while the Agile process is based on procedural knowledge that is acquired from experience, such as verbal communication to exchange information.

In conclusion, students should first be taught the Disciplined process to enable them to understand the making of software through the crystallization of guided information and information sharing. We believe that the Agile process project should be designed for Senior students with some industrial experience (through internships), who have had previous experience with a Disciplined process oriented project.

Acknowledgments—We are grateful to Éric Germain who was the instructor for this project and participated in the preparation of the Studio. This project would not have been possible without the participation of the students enrolled in the 'Software Engineering Studio' course at École Polytechnique de Montréal. This work was supported in part by NSERC grant A-0141.

REFERENCES

1. D. Socha and S. Walker, Is Designing Software Different from Designing Other things?, *Int. J. Eng. Ed.*, **22**(3), 2006, pp. 540–550.
2. P. Kruchten, *Unified Process: An Introduction*, Addison Wesley, Reading, Mass, USA, (2000).
3. A. Cockburn, *Agile Software Development*, Addison Wesley, Boston, Mass, USA, (2002).
4. Agile Alliance website: http://www.agilealliance.org.

5. K. Beck, *Extreme Programming Explained*, 2nd edn, Addison Wesley, Boston, Mass, USA, (2007).
6. E. Germain and P.N. Robillard, Engineering-based process and agile methodologies for software development: a comparative case study, *Journal of Systems & Software*, **75**, 2005, pp. 17–27.
7. K. Beck and B. Boehm, Agility through discipline: a debate, *IEEE Computer*, **6**, 2003, pp. 44–46.
8. E. Germain, P.N. Robillard and M. Dulipovici, Process activities in a project based course in software engineering, *Proceedings of the IEEE 32nd Frontiers in Education Conference (FIE'2002)*, S3G–7, (2002).
9. R. M. Gagné, *The Conditions of Learning and Theory of Instruction*, Holt, Rinehart & Winston, New York, (1985).
10. G. G. Mitchell and J. D. Delaney, An assessment strategy to determine learning outcomes in a software engineering problem-based learning course, *Int. J. Eng. Educ.*, **20**(3), 2004, pp. 494–502.
11. NSERC, Natural Sciences and Engineering Research Council of Canada. Tri-Council Policy Statement: Ethical Conduct for Research Involving Humans. In. Interagency Advisory Panel on Research Ethics (2005): http://www.pre.ethics.gc.ca/english/policystatement/policystatement.cfm.
12. P. N. Robillard, P. Kruchten and P. d'Astous, *Software Engineering Process with the UPEDU*, Addison Wesley, Boston, Mass, USA, (2003).
13. Ecole Polytechnique de Montréal , UPEDU website: http://www.upedu.org.
14. Manifesto for Agile Software Development, Agile Alliance website: http://www.agilealliance.org
15. R. Martin, *Agile Software Development: Principles, Patterns, and Practices*, Prentice-Hall, Englewood Cliffs, NJ, (2002).
16. W. F. Tichy, Hints for reviewing empirical work in software engineering, *Empirical Software Engineering*, **5**, 2000, pp. 309–312.
17. J. Carver, , L. Jaccheri and S. Morasca, Issues in using students in empirical studies in software engineering education, *Proceedings of the Ninth International Software Metrics Sympososium (METRICS'03),* pp. 239–249, (2003).
18. M. Höst, B. Regnell and C. Wohlin, Using students as subjects—a comparative study of students and professionals in lead-time impact assessment, *Empirical Software Engineering*, **5**, 2000, pp. 201–214.
19. A. Cockburn, *Agile Software Development*, Addison Wesley, Boston, Mass, USA, (2002).
20. J. Smith, *A Comparison of RUP and XP*, Rational Edge, (2001).
21. P. N. Robillard, Opportunistic problem solving in software engineering, *IEEE Software*, (Nov/Dec), 2005, pp. 60–67.
22. P. N. Robillard, The role of knowledge in software, *Communications of the ACM*, **42**(1), 1999, pp. 87–92.

Pierre N. Robillard is a professor in the department of Computer and Software Engineering at the École Polytechnique de Montréal. He is the author of several textbooks on software engineering. He holds a Bachelor's degree in Physics from Université de Montréal, a M.A.Sc. from University of Toronto and a Ph.D. from Université Laval. His research interests include the software engineering process, the modeling of cognitive activities and software workplace learning. Dr. Robillard is a professional engineer and is member of the IEEE and the IEEE Computer Society, the Association of Computing Machinery, and the European Association of Cognitive Ergonomics.

Mihaela Dulipovici is a lecturer in software process and software engineering at the École Polytechnique and the École des Hautes Études Commerciales, both in Montréal. She holds an M.A.Sc. degree in software engineering from École Polytechnique and is actually enrolled in a Ph.D. program. Her research interests include the user-centered software engineering process and the HCI. She is member of the IEEE.