Measuring Software Process Activities in Student Settings

Éric Germain Michaela Dulipovici Pierre N. Robillard

École Polytechnique de Montréal
Laboratoire de recherche en génie logiciel
C.P. 6079 succ. Centre-ville
Montréal, QC H3C 3A7 Canada
+1 (514) 340-4711 ext. 4815

{ eric.germain, mihaela.dulipovici, pierre-n.robillard } @ polymtl.ca

Abstract

The recent emergence of various software processes facilitates process activity measurement and provides an opportunity for improving project planning as well as estimating related activities. As such, process measurement of effort spent during each activity is required to better understand the relationship between the various activities. The measurement of activity effort can lead to better understanding of the roles of these activities within the process discipline. Measurement of activity effort can also help to validate the prescribed activities and enable some modifications of the process disciplines.

However, direct measurement of activities performed can lead to measurement interference with the experiment. Predefinition of activities to be used by participants for data recording can only be made under the hypothesis that they will restrict their actions to the prescribed activities. Also, use of activity as a basis for measurement leads to process data that cannot be compared directly between different processes. Therefore, a classification is needed that will allow easy data entry and map naturally to many different process activity sets while avoiding interference.

This paper describes the process measurement that takes place every winter semester in the Software Engineering Studio class at École Polytechnique de Montréal. Each project lasts 675 hrs-pers. and is completed in parallel by teams of 5 students. Each of the three to five teams, depending on enrolment, base their work on the same formal specification and use a well-defined software process.

The paper presents the activity classification that was used in the 2001 setup. The software process used was a variation of the Unified Process for EDUcation (UPEDU), derived from the Rational Unified Process (RUP) in collaboration with Rational Software. Challenges and difficulties encountered are presented with a brief overview of the findings from the measurements taken.

Keywords

Software process discipline, software process activity, workflows, software life cycle, software measurement, experimental software engineering, project planning and estimating

1. Introduction

The purpose of a software engineering process is to improve the software development cycle by defining the various activities that are being done and by specifying the various artifacts that are being built. A well-defined software process specifies clearly the interrelationship between the activities and the artifacts. Some large software development organizations have a dedicated software engineering process group to define and maintain an efficient process.

One model that is gaining wider acceptance within the software industry is the Rational Unified Process (RUP). This model of software process is based on disciplines and their corresponding activities. Commercial processes like the Rational Unified Process (RUP) define all the disciplines and activities that are likely to be needed in a software development project [3]. The software process engineer will then customize the process to the needs of the project. Appropriateness of the new derived process is appraised by the level of acceptance of the activities by the team members and by the quality of the resulting product.

The ultimate goal of any software process is to improve the management of the software development project and the quality of the resulting software product. In defining a software process it is often assumed that the most important task is to identify the activities to be performed by the team. Process designers have a comprehensive view of the process activities and can see readily where each activity fits within the ensemble. However, software developers have rarely such a wide view and see activities only as needed. Developers may not see readily the purpose of each of a set of activities whenever these are closely related. One can see that there is no point of defining a process composed of many activities if developers get confused on the meaning or the purposes of these activities.

Empirical studies are needed to validate the prescribed activities proposed in a software process. This paper presents a case study where three projects are analyzed in order to validate the activities that compose the software process. This case-study was done in an university environment. Three teams of five (5) students developed a software product from the same formal specification. Each team used a software process specifically designed for the project and derived from the Unified Process for EDUcation (UPEDU), a software process based on the Rational Unified

Process (RUP). Effort spent on each activity defined by the process was carefully recorded. A comparative study was performed between the three teams [5]. All the teammates had a specific training on every activity of the process. All the projects were successful in the sense that they met all the requirements, succeeded customer tests and delivered the completed set of artifacts required by the software process. Challenges and difficulties encountered are presented with a brief overview of the findings from the measurement process.

2. Project Description

The application in which present measurements were taken was carried out within the framework of the bachelor level course "Studio in Software Engineering" given at École Polytechnique de Montréal. This course is different from a conventional one by its objective and its teaching method. The general objective of the studio is to learn software development through a project-oriented product. The teaching method is based mainly on work teams. All the teams are in competition on the same project and have to follow the same software development process, i.e. UPEDU. [7] This course does not contain a final exam but a class presentation and a formal acceptance test session.

The 13-week project is a Client-Server application programmed in Java software language, representing a Time Monitoring Tool (TMT) for software development teams. TMT could be used by any software development team. Depending on the users of the recorded data, the purpose of the TMT is to help developers record the time spent on various software development activities and managers to validate their planning, budgets and schedules and produce various reports. The TMT system to be developed is a stand-alone tool that is integrated within the organization's Intranet. The tool consists of four major components: a Developer Client Module, a Manager Client Module, a Server Module, and a Database. All components must execute on a Windows NT environment. The specifications were presented according to the IEEE std 830-1993: IEEE Recommended Practice for Software Requirements Specifications [2].

Each student was responsible for filling out his records capturing the time spent by a teammate student on a specific activity. One team member was responsible for collecting every week their teammates' completed records and for integrating the data into the team's database. The main fields of effort recording information are presented in Table 1.

Table 1. Format of effort recording information

Worker Name
Worker
Activity
Artifact
Date
Start Time
End Time
Duration

For each record the students inserted a short description of the entered data. The duration of the activity was automatically calculated. All the data presented in this article were obtained through analysis of these team databases.

The students attending this course are seniors and should have completed the required software engineering courses. Each student was expected to spend at least 135 hours on this project for a total of 675 person-hours for a 5-person team. The class included 3 teams. Team A and Team B were composed of 5 students each. Team C was composed of 4 students. Figure 1 shows the distribution of the time spent by each team for the project duration. It must be noted that Team C members have offered a quite steady effort pattern through the semester and were able to complete their project easily within the time frame prescribed, Team A and B members showed a more "student classic" pattern with a significant break in the middle and lots of work in the very last weeks. Nevertheless, all three end products conformed to the specifications.

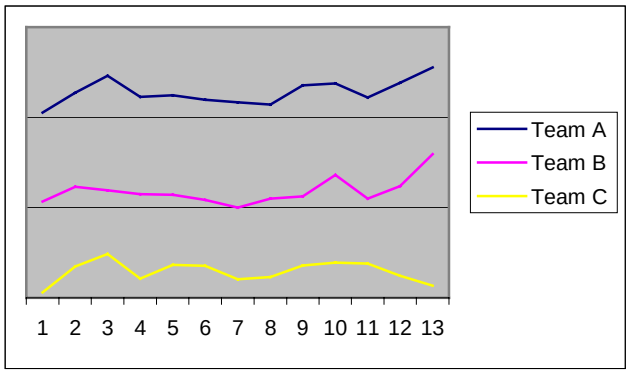

Figure 1. Effort distribution for each team

3. Process Definition

The process used for this project course is an adaptation of the UPEDU (pronounced Yoopeedoo), an acronym for Unified Process for EDUcation [7], which is derived from RUP [3]. UPEDU introduces students to the proper software process activities and will enable them to understand the role of processes in software development projects and to select activities, which are more specific to a given task. Most students have little industrial software development experience, and many of the process activities have little meaning for them. However, this behavior could be similar to any professional developer that joins a new team or a new project. The difficulty in adapting a process is to maintain a good balance in the process activities in terms of the various conceptual viewpoints they represent. The presence of too many activities can make some activities perceived as redundant while there should be enough activities to build a good software process. Software professionals designed the process used in this project. They had good experience in both software engineering process and course projects. This process was believed to be well adapted until the results of this case study were known.

UPEDU is made of 4 engineering disciplines (Requirements, Analysis & Design, Implementation, and Testing) and 2 management disciplines (Configuration & Change Management, and Project Management). A discipline is also composed of artifacts and roles.

UPEDU is also based on an iteration approach. An iteration is a time frame defined by a milestone for delivering some of the artifacts. It is expected that activities from each of the disciplines are performed in each iteration. Table 2 presents the milestones of each of the four iterations and their duration in weeks.

Table 2. Distribution of iterations during project duration

Week	Iteration
1	1 Validate requirement And Plan project
2	
3	
4	2 Complete analysis and build architecture
5	
6	
7	3 Build product
8	
9	
10	
11	4 Test and deliver
12	
13	

This study focuses on validating the activities that composed the process. Each activity is composed of three different cognitive tasks. For example activity "Define Use Case" is made of the intellectual task of thinking about the Use Case and the specification. The second intellectual task is to draw these Use Cases or to write the documentation related to this activity. The last intellectual task related to this activity is to participate in a technical review meeting on the Use Case.

The activities as defined by the UPEDU are concerned only by the intrinsic intellectual task of this activity. Thus, all tasks involving document writing or modification have been extracted from basic-level activities and grouped into a single, cross-discipline "Write Documentation" activity. Also, all "Review" activities from every discipline have been merged into a new generic "Review Artifact" activity.

4. Results Overview

An initial analysis step has been performed on recorded activities. Basic extraction and grouping of activities has been performed in order to better reflect the three cognitive tasks of activities as discussed in section 3. Thus, all "Review" activities from every discipline have been merged into a new generic "Review Artifact" activity. Also, all tasks involving document writing or modification have been extracted from recorded activities and grouped into a single, cross-discipline "Write Documentation" activity. Figure 2 shows a pie chart displaying the contribution of each discipline to the whole project based on the recorded activities. This chart shows that most of the effort is spent on the implementation and the review activities. Each other activity account for less that 10% of the effort.

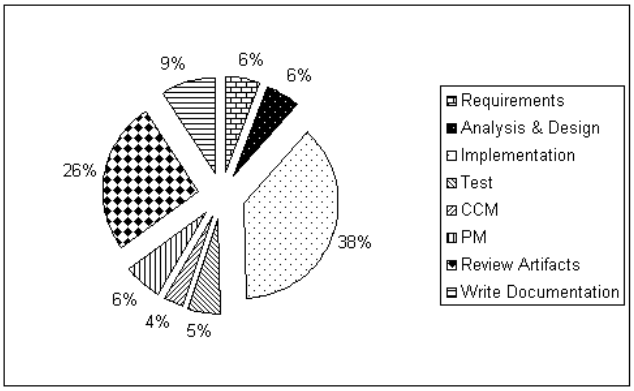

Figure 2. Effort repartition by discipline, initial data

It is expected that all teams will have performed some amount of effort in every activity. However, analysis shows that this expected and prescribed behavior was not the case. Some activities that were qualified as coherent were well understood and performed by every team. Some other activities were well performed by only two of the teams, meaning that one team did not see the purpose of these activities, which were call confused. Finally some other activities were performed by only one team and completely neglected by the two other teams; these activities were labeled as ambiguous. Each activity was assigned a quality status attribute based on one of these situations. Table 3 shows classification of every activity.

Table 3. Quality Status by Activity

Discipline	Activity	Quality Status
Requirements	Obtain Client Requirements	Coherent
	Find Actors and Use Cases	Coherent
	Structure Use Case Model	Ambiguous
	Detail Use Case	Confused
Analysis & Design	Architectural Analysis	Confused
	Use Case Analysis	Coherent
	Use Case Design	Coherent
	Class Design	Coherent
Implementation	Plan System Integration	Ambiguous
	Implement Component	Coherent
	Fix Defect	Confused
	Perform Unit Test	Confused
	Integrate System	Ambiguous
Testing	Plan Test	Coherent
	Design Test	Confused
	Execute Test	Confused
	Implement Test Component	Ambiguous
	Evaluate Test	Ambiguous
Configuration & Change Management	Create Baseline	Coherent
	Create Workspace	Ambiguous
	Define CM Environment	Confused
	Make Changes	Coherent
Project Management	Plan Phases and Iterations	Coherent
(cross-discipline)	Review Artifacts	Coherent
	Write Documentation	Coherent

4.1 Coherence

Coherence is established for a given activity when effort has been recorded for every team within that activity. Activity "Find Actors and Use Cases" is a typical example of coherence (see figure 3). Effort distributions of coherent activities clearly show that this activity is well understood by team member and they know what they are doing when recording this activity.

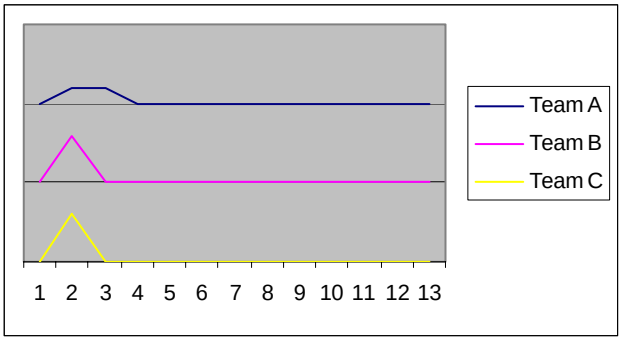

Figure 3. Effort patterns for activity "Find Actors and Use Cases" (max. = 5 pers.*hr)

4.2 Confusion

An activity is labeled confused when only two of the three teams have recorded any effort. Confusion can take one of two forms. The first confusion form occurred when the effort patterns shows some similarity. The second confusion form occurred when the effort patterns are not similar. The "Architectural Analysis" activity is an example of first confusion form (see figure 4). Peak efforts for Team A and Team C occur at week 4 and 5 respectively, which is a short time period within iteration no 2. The "Execute Test" activity is an example of second confusion form (see figure 5). Peak efforts for Team A and Team C occur at week 13 and 10 respectively. That represents a three week period that spans across two iterations. It is unlikely that such a lag is due only to short-term scheduling or productivity variations.

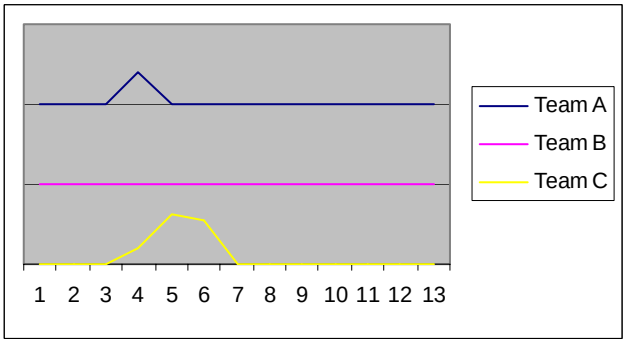

Figure 4. Effort patterns for activity "Architectural Analysis" (max. = 10 pers.*hr)

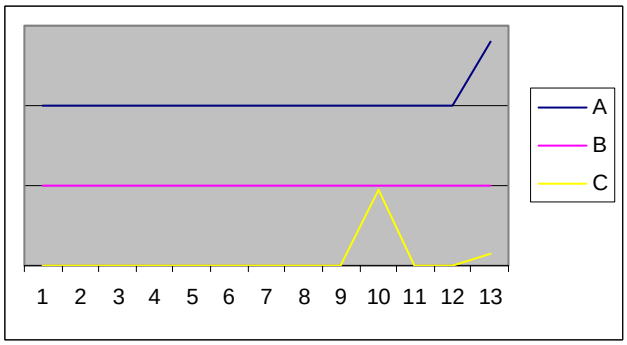

Figure 5. Effort patterns for activity "Execute Test" (max. = 10 pers.*hr)

4.3 Ambiguity

Pattern ambiguity occurs when, for a given activity, no effort has been recorded at all for two out of the three teams. This situation occurs when participants misperceive the nature of an activity, assign the work for which the activity was prescribed to another activity and do not assign any other effort to the former.

Activity "Integrate System" (see figure 6) is an example of activity ambiguity since only team B recorded any occurrence of that activity through the project.

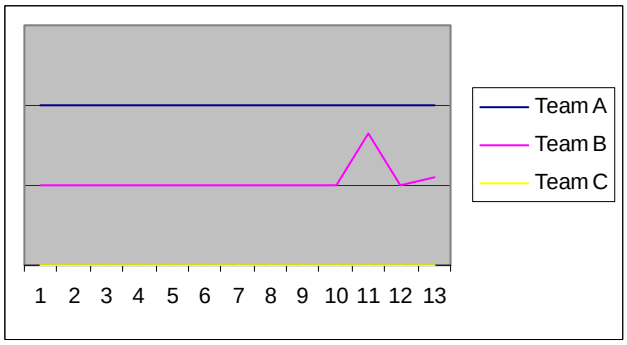

Figure 6. Effort patterns for activity "Integrate System" (max. = 20 pers.*hr)

The following comments the ambiguous activities found in this case study. It is believed that the low challenge level in component integration in the project compared to component development brought participants to record integration related work under activity "Implement Component". The same reasoning also applies to activity "Implement Test Component" in relation to activity "Execute Test". In the case of activity "Evaluate Test", it is suspected that this activity may have been simply overlooked by two teams in their struggle to deliver a complete, yet bug prone software product before the prescribed deadline. Even though it is not a recommended approach, activity "Structure Use Case Model" may have been performed implicitly during the realization of activity "Detail Use Case". Finally, the purpose of activity "Create Workspace" may not have been understood at all considering the fact that participants had been exposed very little to the concepts of Configuration Management in their curriculum.

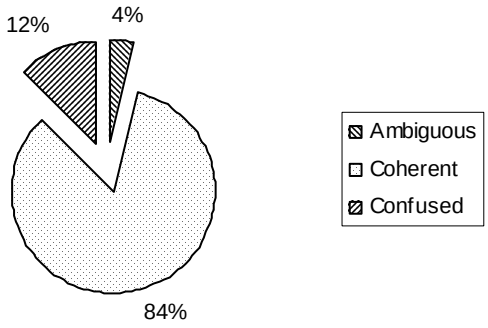

Figure 7. Effort distribution of raw activities among categories

It must be noted that confused and ambiguous activities are not the result of a bias team. All teams contribute to these activities in no systematic ways.

Figure 7 shows the distribution of actual effort among categories. Coherent activities represent 84% of the total. However, as shown in Figure 8, this percentage drops to 58% when the first five more effort-intensive activities are removed from the set. Thus, 19 process activities have a level of incoherence of 42% on average. This illustrates the fact that a vast majority of process activities are not very well understood by the developers. Nevertheless, the end products delivered by all the teams were conforming to the original requirements. Therefore we must question the relevance of the activity classification and reconsider the process implementation methodology itself.

There is actually no easy path to the identification of the right activities to be part of the software process. In particular, a problem seems to lie with our measurement methodology. Direct measurement of activities performed can lead to measurement interference with the experiment. This phenomenon is likely to occur whenever one tries to measure some activity for which some attributes are already explicitly defined in the process. People performing the activities might tend to alter the data they are entering in the system if they depart too much from the prescribed process, just to show they act accordingly with the instructions they were given.

In our experiment, students had to submit artifacts at specified dates, and those artifacts were the basis for their evaluation. However, it was probably unclear for them whether the process activities were mandatory. In fact, we presumed that the natural way to generate or modify the prescribed artifacts was to perform the activities that are related to these artifacts in the scope of the process. Our results bring a significant doubt about this assumption.

5. Conclusion

Empirical studies on process activities are a keystone element of process improvement. These studies should go in parallel with the formalization and standardization of software engineering process.

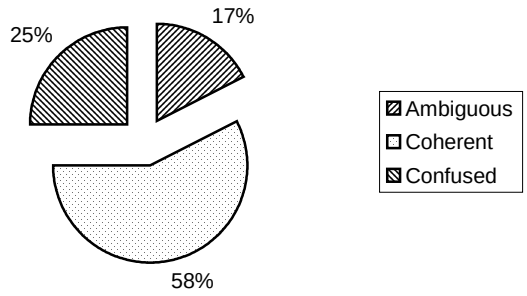

Figure 8. Effort distribution of activities among categories, without the five most effort-intensive ones

Effort measurement is a challenging task that must performed with great care. It is sometimes insufficient to implement a very well defined process as it may indeed be misunderstood. As we have shown, a well-defined process like UPEDU can be implemented while the proportion of activities that are not well understood by the developers remains significantly high.

There are some ways that should be investigated in order to overcome this situation. One is to pay special attention to defining activities in a greater level of detail than what is usually encountered, especially when it comes to defining the boundaries between activities that are closely related (e.g. Analysis vs Design). For some of these activities, that task may seem very difficult to achieve because of their inherent confusion. Another possibility is to define activities that are independent of process disciplines but are rather related to fundamental human cognitive activity. Such an approach can however be implemented only in the context of a paradigm shift in what is used for process measurement purposes. The use of artifacts as the sole basis for process execution control may bear many advantages over the use of activities in that context.

As a note of caution, the results of this study are derived from student projects realized in a very well defined context. The goal of the study is to show the necessity of defining an adequate measurement context when performing process measurement. Therefore generalization of these results to industrial projects can only be made with great care.

6. Acknowledgments

This project would of course not have been possible without the participation of all the students who took the Software Engineering Studio course at Winter 2001. We are grateful to Houcine Skalli for his work as Teaching Assistant during the course, and to Martin Robillard who provided the formal specification for the software to be built.

This work was supported in part by National Sciences and Engineering Council of Canada under grant A0141.

7. References

[1] El Emam, K, Drouin, J-N and Melo, W. SPICE: The Theory and Practice of Software Process Improvement and Capability Determination. IEEE CS Press, 1998.

[2] IEEE Std 830-1993. Recommended Practice for Software Requirements Specifications. 32 pp.

[3] Kruchten, P. The Rational Unified Process: An Introduction. Addison-Wesley, 2000.

[4] Paulk, M. C., Curtis, B., Chrissis, M. B. and Weber, C. V. Capability maturity model for software, version 1.1. Technical Report CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie-Mellon University, February 1993.

[5] Robillard, P.N. Case study analysis of Measuring Effort in a Software Engineering Process. Proceedings of MCSEAI'2000 (Fes, Morocco, November 2000).

[6] Robillard, P.N. The Role of Knowledge in Software. Communications of the ACM, 42, 1 (January 1999), 87-92.

[7] Robillard, P.N., Kruchten, P. and d'Astous, P. YOOPEEDOO (UPEDU): A Process for Teaching Software Process. Proceedings of CSEET (Charlotte, NC, USA, February 2001), 18-26