STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

ON INDIVIDUAL PROJECTS IN SOFTWARE ENGINEERING EDUCATION

MILITON FRENŢIU, IOAN LAZĂR, AND HORIA F. POP

Abstract. A study of the individual projects is performed and some thoughts on the importance and the development of these projects are given. The projects are written by second year undergraduate students as a requirement in their curriculum, and by graduate students working for their companies, for comparison. The principal components method is used as Data Analysis Technique. Some consequences on the education activity are considered.

Keywords: Student Projects, Education, Software Engineering, software metrics, measurement, data analysis technique. fuzzy clustering

1. Introduction

Undergraduate computing courses having a project component (that attemps to convey some of the aspects of a real-life development project) often concentrate on the final product, rather than the process by which it is achieved [17]. However, it is important that the students should see the necessity of all phases of the life-cycle of a project, they must be accustomed to produce the documents for all these phases. We think it is important for undergraduate students their first project be PROCESS-ORIENTED activity.

There are two projects in the curricula of undergraduates in Computer Science at Babes-Bolyai University. First is an individual project, and a second is a Team project in the third year. Both projects are closed projects [13]. Also, it is supposed that each student will conceive an application connected to the diploma work at the end of studies (fourth year). The aim of individual projects at the Babes-Bolyai University is to give the students the opportunity to accomplish a simple project fulfilling all the steps of the life-cycle [5]. It is planned in the third semester, after the students have learned Algorithms and Pascal programming in the first

Received by the editors: December 10, 2003.

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques*; I.5.1 [**Computing Methodologies**] : Pattern Recognition – *Models* – *Fuzzy set*; G.3 [**Mathematics of Computing**] : Probability and Statistics – *Data analysis*.

semester, and Object-oriented programming (in C++), and Data Structures in the second semester.

The subject runs in the first semester of the second year, and is the first project done by an undergraduate student. It is a small project, of about a few hundreds up to 3000 statements. The main purpose is to make the students accustomed to finalize a project and to write all needed documents: requirements, specification, design, code, testing, and user manual, and, very important, to obey the deadlines. Summarising, the main objectives of the individual projects are:

- going through all steps of the life-cycle;
- observing the deadlines;
- writing a clear documentation for all steps;
- verifying and validating the program; inspections and reviews are considered very important means of eliminating errors;
- showing to the students when their programs are insufficiently tested.

The project extends over a fourteen week semester and in each week there is one hour practicum session in a computer laboratory. Students are also expected to complete an extra work at home to meet the requirements of the subject.

At every deadline the student have to deliver the corresponding document [5]. Missed deadlines are penalized; it is considered a decrease of the grade for the corresponding activity by one point for every late week.

Individual project features (document outlines) and the included aspects (each to be completed by students in two weeks) are:

Requirements specification: a conceptual model, functional requirements, user interface requirements, error handling;

Design specification: system architecture, and detailed module specification;

Test plan: a system level testing plan, a detailed unit test plan, and test results;

Coding: correct translation of the design and commented source code;

User manual: an introduction, general principles for using the software, tutorial on how to use the software, a list of software's functions and simple installation instructions;

Project evaluation: comments on the quality of the actual product (what was done well and what could be improved in future projects).

A complete documentation for each of the above mentioned phase is required to support the undergo software engineering process.We require a clear and complete documentation from the students, and we consider this as one of the main gain acquired by students through this project. Their attention has to be turned from immediate coding to correct approach of all life-cycle phases, with an accent on

complete and accurate specification and clear and correct design. Is well known that the most difficult errors, and their greatest part, as well, are not due to bad coding, but to incorrect design. We must convince the students by all means that a serious programmer is not the one that starts coding immediately as he gets the requirements. Good programming means not jumping any step of the life cycle, a good design is crucial for the next project, and the documents are needed for the entire activity as early as possible.

An important activity required to the students was the inspection of the documents written by their colleagues. It is known [6, 10, 14] that many shortcomings of the software process may be eliminated by inspecting all the steps of the life-cycle, starting with the requirements. This activity persuades the students on the necessity and importance of documments for all phases, and gets them accustomed to analyse other people's documents. Therefore, forcing the students to inspect the documents of other people is important not only for software engineering profession, but also for education. By inspection, students have the opportunity to see how others think and write programs, and to discover some of their errors. Also, the teacher has the possibility to discuss with the students about errors, shortcomings and difficulties, to improve the software process by various discussions and questions about them. The teacher has the possibility to provide to students a better way of solving problems, to force them to elliminate defects, and to improve their ways of programming.

Some important 'real-life' project features [12] are: (a) working with real users, (b) developing a working prototype, (c) completing a running system and (d) writing a formal verification and validation report. This individual project does not address the issues of 'real-life' projects, but it is mainly educationally oriented. The students do participate, during their third year of academic studies, to a group project which is 'real-life' oriented.

2. Observed attributes and Data collection

The study is based on 28 finished programs produced by second year undergraduate students as part of their requirements curriculum, and other 8 real products produced by graduates students at their software companies. The observed attributes were estimated by master students attending the course "Software Metrics". Certainly the measures associated to the attributes are the subjective evaluations of these master students about the corresponding attributes. We may accept that the postgraduate students are not experienced programmers, but they have finished a similar project three years earlier, and other two projects in their third and fourth year. Many of them are also working for software companies. Moreover, their evaluation was inspected by the first author and a few corrections were made (where obviously erroneous evaluations have been noticed).

The postgraduate students form a Master group in "Component-Based Programming" that studies the subject "Software Metrics". The definitions of the above considered attributes were given there, and they are inspired by, and can be found in the literature [2, 9, 14]. As an exercise they had to evaluate one project as a requirement for their seminaries at "Software Metrics", and theirevaluation was discussed with the first author of this paper. The analysed projects of the undergraduate students were seen beforehand and graded by some other teaching assistants and their grades are not influenced by this study.

Therefore, students involvement is twofold. On one side the undergraduate students of the second year learned to specify, design, code and test a complete program, on the other side, the master students learned to analyse software documents, to measure software attributes. Data collection is an important software measurement activity, and, since it is not easy to obtain access to real projects, this was another possibility to get used to evaluate software attributes. And we remarked it was a very useful activity, since the master students were very critical about the analysed projects, about the clarity of the documents and of the design, about the absence of the comments in the code. And they remembered they had the same shortcommings three years earlier.

The projects were analysed observing the attributes given in Table 1.

A1	requirements description	A18	number of classes
A2	good specification	A19	number of methods for all classes
A3	function points	A20	changeability (modifiability)
A4	clarity of design	A21	structuredness
A5	correctness of design	A22	testability
A6	completness of design	A23	reliability
A7	diagrams of design	A24	efficiency
A8	modules specification	A25	extensibility
A9	algorithms description	A26	adaptability
A10	lines of code	A27	clarity of documentation
A11	no. of comments	A28	maintainability
A12	good use of comments	A29	simplicity
A13	good use of free lines	A30	usability
A14	indentation	A31	portability
A15	good names	A32	quality
A16	readability	A33	average of weighted methods per class
A17	comprehensibility	A34	depth of inheritance

TABLE 1. The attributes observed for the software projects analysis

The attributes A10 and A11 were automatically measured by computer. All the others were estimated by master students attending the course "Software Metrics".

All metrics have the values in the interval [0, 10], where 0 is for "very bad" (or not present at all), and 10 for "excellent".

Certainly, these grades are subjective estimates on the projects. Nevertheless, we consider that the dependence between attributes is preserved in these data, and the strong dependence between almost all attributes and the knowledge of the authors (reflected in A34) is preserved. We may use these data and the results to draw some useful conclusions on the organisation of the software development process and the way it may be improved.

The attribute A12 refers to the documentation done by comments. It takes into account if the specification of each module is reflected through comments, if the meaning of each variable and object is explained by comments, if the invariants and other important explanations are given by comments.

3. Data Processing

We have analyzed a data set composed of 36 projects, characterized by 34 software metrics attributes. The 36 projects consist of 28 educational undergraduate projects and 8 real projects (namely, 12, 13, 21, 25, 28–31). In order to save the editorial space, the whole data set, as well as complete computational results, are not published here, but are available from the authors[1]

We have run a few experiments in order to detect the proper relations among the data. We were interested in studying the fuzzy cluster substructure of the data set, as well as the fuzzy cluster substructure of the set of attributes [16]. A special note with respect to attributes 18, 19, 33 and 34. These are characteristic to object-oriented approached projects, and are not relevant for other projects. As such, we have considered our study in two scenarios: on one side, we have considered the value of these attributes to be zero for the projects without object-orientedness. On the other side, we have marked the values of these attributes as missing.

In the first case we have used the Fuzzy Divisive Hierarchic Algorithm, with the Euclidean metric [15], and in the second case we have used the optimal completion strategy as outlined in [7], but in the same framework of the Fuzzy Divisive Hierarchic Algorithm. The results in the two cases are practically identical, allowing us to consider as reasonable fact to assign a value of zero to an object-oriented attribute in the case of a non-object-oriented project.

The final defuzzyfied hierarchy corresponding to the optimal fuzzy cluster substructure of the set of projects is described in Table 2. We remark grouping of all student projects in subclasses of the class 1.1.1, and the separation of the industrial

[1]The primary data is not given here, but can be seen, together with full data analysis results, at the web address `http://www.cs.ubbcluj.eo/ mfrentiu/articole/project3.html`.

projects in the other classes, along patterns of similarity to the student projects. Thus, projects 12, 13, 21, 25 have been grouped as class 1.1.2, as the most similar projects to the student group; project 31 forms class 1.2.1, and projects 28 and 30 form class 1.2.2, both classes considered more distant to the student group. Finaly, project 29 forms class 2., showing a clear separation of the whole set of projects. On the other side, the student projects have been further divided, the most notable split being among projects 3, 5, 7, 8, 15, 32–34, 36 (class 1.1.1.) and 1, 2, 4, 6, 9–11, 14, 16-20, 22-24, 26, 27, 35 (class 1.2).

Class	**Members**
1.1.1.1.1.1.	5 8
1.1.1.1.1.2.	3 15 33
1.1.1.1.2.1.1.	7
1.1.1.1.2.1.2.1.	34
1.1.1.1.2.1.2.2.	36
1.1.1.1.2.2.	32
1.1.1.2.1.1.	1 17 20
1.1.1.2.1.2.1.1.1.	26
1.1.1.2.1.2.1.1.2.	35
1.1.1.2.1.2.1.2.	6 27
1.1.1.2.1.2.2.	9 16 22 23
1.1.1.2.2.1.1.1.	18
1.1.1.2.2.1.1.2.	2
1.1.1.2.2.1.2.1.	11
1.1.1.2.2.1.2.2.	14 19
1.1.1.2.2.2.1.1.	24
1.1.1.2.2.2.1.2.	4
1.1.1.2.2.2.2.	10
1.1.2.1.1.	13
1.1.2.1.2.	25
1.1.2.2.1.	12
1.1.2.2.2.	21
1.2.1.	31
1.2.2.1.	30
1.2.2.2.	28
2.	29

TABLE 2. Final defuzzyfied partition corresponding to the optimal fuzzy cluster substructure of the set of projects

The final defuzzyfied hierarchy corresponding to the optimal fuzzy cluster substructure of the set of attributes is given in Table 3(a). The attributes given in

paranthesis are not clear-cut members of those classes, but have relevant fuzzy membership degrees so that they should be taken into account, as well.

We first remark that quite a large number of attributes have not been split. The attributes 1, 2, 4–9, 13–17, 20–32 (class 1.1.1.1.1) may actually correspond to a single inherent property of programming projects, property that is expressed through more different attributes. Other than these attributes, attribute 12 has membership degree 0.42 (as compared to 0.56, the membership degree to its class (!)), and attribute 33 has membership degrees of 0.27 (as compared to 0.30, the membership degree to its class (!)).

Other than this large class, we identify a few relevant groups: the most significant separation is of attribute 10 (class 2), the program size (lines of code). At the next level, we identify attribute 11 (class 1.2.2), associated to the program size as well (lines of comments), and attribute 19 (class 1.2.1), representing the total number of methods. A supporting member of class 1.2.1 is attribute 3, with quite a relevant membership degree of 0.30 (as compared to 0.70, the membership degree to its class).

On the other side of the tree, we identify the class 1.1.1.1.2 (formed by attributes 12 – comments accuracy, 33 – weighted methods per class, and 34 – depth of inheritance; attribute 7 is supporting member, with a membership degree of 0.30, as compared to the membership degree of 0.69 to its class); class 1.1.1.2 (attribute 18 – number of classes); class 1.1.2 (attribute 3 – function points; attribute 19 is a supporting member, with a membership degree of 0.33, as compared to the membership degree of 0.67 to its class).

In order to verify the relevance of the attributes in the presence or absence of industrial projects, we have removed the industrial projects from our attributes classification. The results are given in Table 3(b). As in the precedent case, the attributes given in paranthesis are not clear-cut members of those classes, but have relevant fuzzy membership degrees so that they should be taken into account, as well.

We remark an extremely similar hierarchical clustering structure. If we consider the shared attributes (i.e. those with mixed mebership degrees), the similarity is even higher. Thus, the main group of attributes, 1, 2, 4–9, 12–17, 20–32, formerly the class 1.1.1.1.1, forms now the class 1.1.1.1.

As well, the attributes 18, 19, 33, 34, formerly roughly with classes 1.1.1.1.2 and 1.1.1.2, form now the class 1.1.1.2. This class has the attribute 12 as a supporting member, with a membership degree of 0.30 (as compared to the membership degree to its own class, 0.69).

We may conclude that industrial projects have a marginal, even insignificant influence on the classification of the project attributes, confirming the overall influence of knowledge level to the way people approach the software process.

On the other side, by taking into account the supporting members of fuzzy classes, we may finally consider five main groups of attributes:

- 10 (lines of code);
- 11 (lines of comments);
- 3, 19 (program complexity);
- 18, 33, 34 (object-oriented attributes);
- 1, 2, 4-9, 12-17, 20-32

Even if only few projects out of a total of 36 are object-oriented, three of the four object-oriented attributes group together (18, 33, 34), and the fourth, the total number of methods for all classes (19), is consistently grouped together with the function points (3), both showing an impact of the overall program complexity.

Class	**Members**
1.1.1.1.1.	1 2 4 5 6 7 8 9 13 14 15 16 17 20 21 22 23 24 25 26 27 28 29 30 31 32 (12, 33)
1.1.1.1.2.	12 33 34 (7)
1.1.1.2.	18
1.1.2.	3 (19)
1.2.1.	19 (3)
1.2.2.	11
2.	10

(a)

Class	**Members**
1.1.1.1.	1 2 4 5 6 7 8 9 12 13 14 15 16 17 20 21 22 23 24 25 26 27 28 29 30 31 32 (12)
1.1.1.2.	18 19 33 34 (12)
1.1.2.	3 (19)
1.2.	11
2.	10

(b)

TABLE 3. Final defuzzyfied partition corresponding to the optimal fuzzy cluster substructure of the set of attributes: (a) with all the 36 projects considered; (b) only with the undergraduate projects

Figure 1 presents the 2D projection of the set of 36 projects along the first two principal components, as determined using the well-known Principal Components Analysis applied to the correlation matrix. The figure clearly displays three categories of projects: the two isolated industrial projects (25 and 29), a second well-separated group of industrial projects (12, 13, 21, 28, 30, 31) and the homogenous group of student projects. It is important to note that the main group of educational projects presents a clear linear trend, supporting the conclusion that most of the considered attributes correspond to a single factor, identified as the overall level of knowledge of the student.

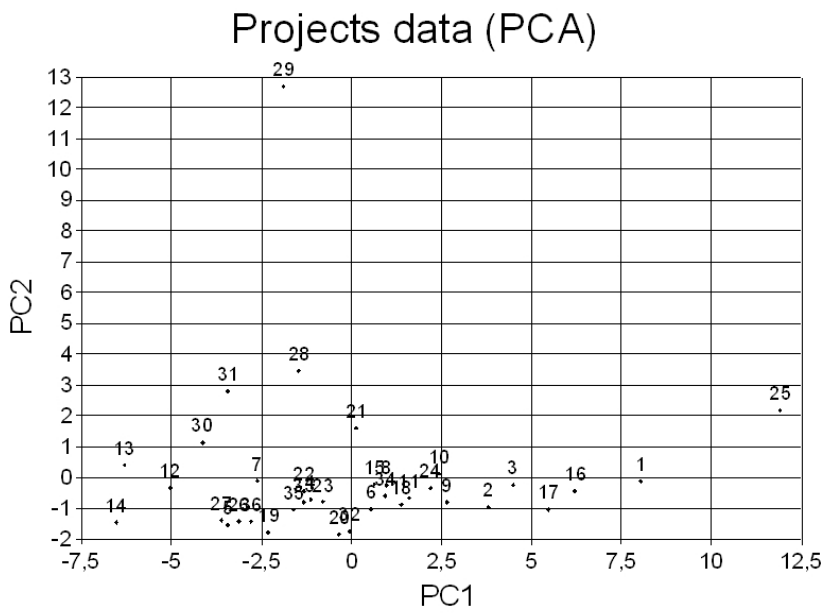

FIGURE 1. Principal Components Analysis of the set of 36 projects. The set is projected along the first two principal directions

4. Conclusions

In this paper we aimed to comparatively study educational and industrial projects. We acknowledge the fact that there are different factors contributing to these projects. On one hand, the purpose of educational projects is to train students into approaching a problem in an ordered manner. As such, the students will have to use different knowledge gathered at different classes. On the other hand, the purpose of industrial projects is to produce final software products that solve a real problem, required by a real customer, and distributed to the real world. These differences among the purposes of the two groups of projects contribute to their grouping in different fuzzy classes.

As it has been expected, the fuzzy clustering procedure confirms that all the qualitative attributes are dependent on the programmers general knowledge, the main such factor.

It is interesting to remark that, generally, the set of attributes have been split along the lines of object-orientedness. Apparently, the object-oriented software metrics are consistent with the function points attribute; the two size attributes

group closely, while all the others are grouped in a single class, allowing us to conclude that the students follow in the same manner all the programming rules and habits they are taught. We conclude that the object-oriented software metrics attributes measure, actually, the complexity of the object-oriented project. It is, thus, needed to concentrate on providing object-oriented attributes that measure the quality of an object-oriented project, as well.

We have analyzed software products made by undergraduate students. We are confident that the results cannot be extrapolated to large software systems. But they can certainly be used towards a better students formation and may be used as effective didactic materials, especially with the "Software Metrics" course. Even if we insist with the first year students on the necessity to develop their own programming style, to obey a few important programming rules [4], the students are skeptical, they are simply happy that their programs "work", they do not like to write comments, or to insist on a good design and Pseudocode algorithms, or documentation.

By analyzing the primary data, we may observe that students do not like writing comments. However, a certain progress is remarked from one generation of students to the subsequent: the necessity of comments appears strongly with this year's students as opposed to the last year's and two years' ago [3].

As well, we remark that undergraduate students refrain from writing complete documentations. Their documents are generally superficial, written on short notice, only to fulfill a requirement lined out by the professor. The students do not generally consider writing project documentations as part of their natural thinking process, required in order to produce effective projects. As well, project documentations are often mixed, in the sense that ideas that are naturally part of the design and implementation process are included in the specification documentation. Nevertheless, at the end of the activity the majority of the students consider that the main gain obtained through this project is understanding the importance of, and learning to write a complete documentation.

Completely different is the approach of real projects programmers (in our study, graduate students) on the necessity of a complete and correct documentation, its usefulness, and the effect of an adequate programming style on final projects.

We remark that the theoretical and practical knowledge of the average student has improved from one generation to the other. This is a confirmation of the success of our stepwise approach towards students education of projects development.

We observe that very few undergraduate students practice object-oriented programming. They have learned it in the second semester of the first year, but are not used to practice it, or they do not feel its necessity or advantages. Why is it so, is a natural question we must try to answer.

We consider useful to provide the students with an effective model, possibly the best project of the previous generation. On the other side, it may be useful to consider as project topics for a part of the students, the requirement to work on improving projects already written by students of the previous generation. This should be a step in the right direction, of improving students knowledge on developing projects.

The continuous improvement of the educational process is the ultimate purpose and goal of any teacher. And aquiring correct habits of developing a programming product is one of the major issues a computer science graduate will have to face. On the other side, a thorough study of the process of projects development, and complete data collection and its accurate interpretation should become a part of our educational activity.

We intend for the next year to improve these projects, asking to the new generation to maintain the existing projects, correcting the possible discovered errors, adding new functions, and adapting them to the changing environments.

Acknowledgements

We are acknowledging the support of postgraduate students of the group "Components Based Programming" for theirhelp in analysing the projects.

References

[1] M. Fagan, Design and Code Inspections to Reduce Errors in Program Development, IBM Systems Journal, 15 (3), 1976.

[2] N.E. Fenton, Software Metrics. A Rigorous Approach, Int. Thompson Computer Press, London, 1995.

[3] M. Frentiu, H.F. Pop, The Study of Dependence of Software Attributes using Data Analysis Techniques, Studia Universitatis Babes-Bolyai, Informatica, 47 (2), 2002, 53–60.

[4] M. Frentiu, On programming style, Technical report, Babes-Bolyai University, Department of Computer Science, http://www.cs.ubbcluj.ro/ mfrentiu/articole/style.html

[5] M. Frentiu, H.F. Pop, Documents produced at the individual project, Tehnical report, Babes-Bolyai University, Department of Computer Science, http://www.cs.ubbcluj.eo/ mfrentiu/articole/project.html

[6] T. Gilb, D. Graham, Software Inspection, Addison-Wesley, 1993

[7] R.J. Hathaway, J.C. Bezdek, Fuzzy C-Means Clustering of Incomplete Data, IEEE Transactions on Systems, Man, Cybernetics, Part B: Cybernetics, 31 (5), 2001, 1062–1071.

[8] C. Ho-Stuart, R. Thomas, Laboratory Practice with Software Quality Assurance. Proc. of the 1996 International Conference on Software Engineering: Education and Practice, IEEE, 1996, 220–225.

[9] ISO 9126, Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use, http://www.iso.org

[10] J.C. Knight, E.A. Myers, An Improved Inspection Technique, Comm. ACM, 36 (11), 1993, 51–61.

[11] O. Laitenberger, A Survey on Software Inspection Technologies, Handbook

[12] W.W. McMillan. What Leading Practitionners Say Should Be Emphasized in Students' Software Engineering Projects, IEEE, 1999, 177–185.
[13] M. Newby, An Empirical Study Comparing the Learning Environments of Open and Closed Computer Laboratories, Journal of Information Systems Education, 13 (4), 303–314.
[14] D.L. Parnas, A.J. van Schowen, S. Po Kwan, Evaluation of Safety-critical Software, Comm.A.c.M., 33(6), 1990, 636–648.
[15] H.F. Pop, Intelligent Systems in Classification Problems, Ph.D. Thesis, Babes-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, 1995.
[16] H.F. Pop, SAADI: Software for fuzzy clustering and related fields, Studia Universitatis Babes-Bolyai, Series Informatica 41 (1), 1996, 69–80.
[17] V.E. Veraart, S.L. Wright, Experience with a Process-driven Approach to Software Engineering Education. Proc. of the 1996 International Conference on Software Engineering: Education and Practice, IEEE, 1996, 406–413.
[18] H. Younessi, D.D. Grant, Using CMM to Evaluate Student SE projects. Proc. of the 1996 International Conference on Software Engineering: Education and Practice, IEEE, 1996, 386–391.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, RO-400084 Cluj-Napoca, Romania

E-mail address: mfrentiu@cs.ubbcluj.ro

E-mail address: ilazar@cs.ubbcluj.ro

E-mail address: hfpop@cs.ubbcluj.ro