A Teaching Experience on Software Reengineering

Ricardo Pérez-Castillo, Ignacio García-Rodríguez de Guzmán, Felix García and Mario Piattini
Alarcos Research Group, University of Castilla-La Mancha
Paseo de la Universidad 4 13071, Ciudad Real, Spain
{ricardo.pdelcastillo, ignacio.grodriguez, felix.garcia, mario.piattini}@uclm.es

***Abstract*—Software maintenance is recognized as an important knowledge area within the most common international curricula in software engineering. Despite this fact, and its importance in the industry, software maintenance and supporting techniques such as reengineering are hardly ever taught in practical lessons. This paper presents a reengineering teaching experience conducted during two last years in lab sessions by using reverse engineering and code generation tools. The experience was carried out by merging traditional methods (such as teaching lessons) with a practical exercise in laboratory. The teaching-learning process was qualitative- and quantitatively assessed by comparing results between an initial and final evaluation, as well as between the experiences conducted during two last years with different syllabus of courses. In fact, the effect of the experience in both syllabi proved to be effective. The reported results show that students do not know reengineering as a software maintenance technique although their satisfaction with the experience was high or very high (62%) or medium (30%). The key learned lessons are that students recognized the usage of reengineering tools as very convenient for their performance as future practitioners and the need to devote additional time in classroom to learn such tools.**

Keywords—Software Engineering, Maintenance, Reengineering, Practical Experience, Evaluation

I. Introduction

Software maintenance is the stage of software development that requires more effort and resources [10]. In fact, the maintenance effort is between 70 and 80% while only the 20-30% of time is spent on other development stages during the software life cycle [8, 13]. Software maintenance is a key activity to correct, adapt, migrate or improve existing software systems [7]. Software maintenance firstly is on the side of the users' satisfaction and secondly extends the lifespan of software systems, which implies a higher return of investment.

There are many approaches and techniques in the literature for carrying out software maintenance. One of the most widespread techniques is software reengineering [1], which has been successfully applied in last two decades. Reengineering advocates obtaining improved versions of an existing system by reusing existing software artifacts in order to preserve the business rules embedded in the system under maintenance [12]. Reengineering process consists of three stages [4]: (i) reverse engineering, which analyses existing software and identifies the different components and their interrelationships to build one representations of the system at a higher degree of abstraction; (ii) restructuring, which takes the previous system's abstract representation and transforms it into an enhanced representation of the system at the same abstraction level by preserving the external behavior; and (iii) forward engineering, generates physical implementations of the target system at a low abstraction level from the restructured system.

This approach is also known as the horseshoe reengineering model [9] due to the abstraction degree is modified throughout the three stages (see Figure 1).

Regarding academia, software maintenance is also perceived as an interesting topic. Indeed, software maintenance, and particularly reengineering, is included in well-known international curricula. The sixth chapter of SWEBOK (Software Engineering Body of Knowledge) [6] is fully devoted to software maintenance. The Software Engineering Curriculum proposed by ACM and IEEE [3] defines an area of knowledge for software evolution with an estimated duration of 10 hours (2.24% of total time).

Despite the importance of software maintenance in industry and academy, teaching about software maintenance and reengineering must deal with two main challenges [5]: (i) software maintenance is often presented in the academia as an additional and routine activity outside the scope of software development process; and (ii) there is no much research concerning software maintenance and reengineering teaching in comparison with other software development stages.

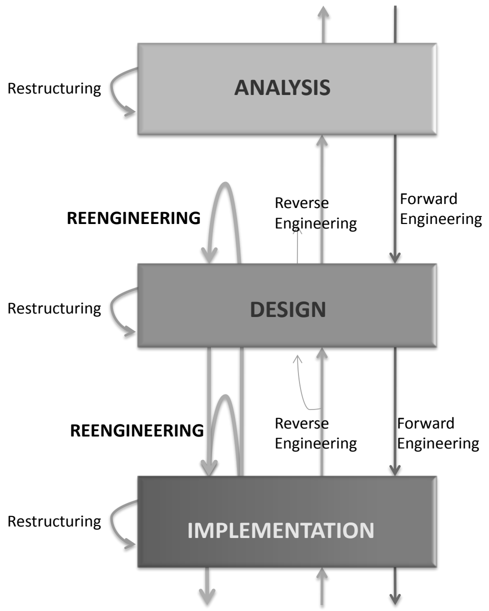


Figure 1. The horseshoe reengineering model.

This work attempts to address these challenges by providing a practical teaching experience about software reengineering within a software engineering degree in a Spanish university. The experience consisted of teaching a lesson about software reengineering and maintenance in a theoretical class as well as a collaborative practical exercise during two lab sessions. The main objective of this investigation is to analyze whether the proposed experience is pedagogic and therefore improve the teaching-learning process of software reengineering and software maintenance in general. During the teaching experience the students carried out an initial evaluation test and a final test as well. Such tests allowed us to know which topics were easily learned and which were the most common learning problems presented by the students. This experience has been replicated during the two last years in the same university within two different syllabi, since a new plan was implemented in the last year. Hence, comparative results concerning both years are also presented.

After analyzing obtained results, the teaching-learning process proved to be efficient since marks that students obtained in the post-questionnaire improved regarding their initial evaluations. Various learned lessons were additionally reported. Firstly, the most common problem for students was the integration of new functionalities in the target systems during software reengineering. Secondly, the majority of the students reported a high satisfaction about learning and managing new software applications for supporting reengineering.

The remaining of the paper is organized as follows: Section 2 summarizes the main international curricula on software engineering concerning reengineering. Section 3 presents in detail the teaching experience. Section 4 analyses the results obtained in the practical experience. Finally, Section 5 provides the conclusions and learned lessons of this work.

II. REENGINEERING IN INTERNATIONAL CURRICULA

Currently, there are several international curricula focused on the profession of Software Engineering such as SWEBOK, Computing Curricula, ICF-2000 (IFIP / UNESCO) or ISCC, among others. However, SWEBOK [6] and ACM / IEEE SE [3] are probably the most relevant ones. In this section it is presented how these curricula address the maintenance process and more specifically, reengineering as an essential method to support this process.

SWEBOOK (Software Engineering Body of Knowledge), which was designed for the accreditation of university curricula and certification of professionals, identifies a core body of knowledge that characterizes the discipline of Software Engineering. SWEBOK is divided into 10 knowledge areas, among which is included the Software Maintenance Area. Regarding maintenance, SWEBOK considers foundations, key concepts, and the processes and techniques for maintenance (see Figure 2). The software maintenance area in the SWEBOK curriculum includes three techniques for software maintenance (see right side of Figure 2). Reengineering is one of these three techniques. Anyway, the remaining techniques (program comprehension and reverse engineering) are covered by the reengineering concept.

The Computing Curricula of ACM / IEEE-CS provides the guidance for curriculum development of careers of computer science and engineering. The Computer Curricula is comprised of several parts: a master volume and additional volumes for specific disciplines. One of these volumes, the SE 2004 [3], is particularly focused on the description of software engineering curriculum. It is important to point out that (from all areas of knowledge or topics of interest that cut across all disciplines covered by the Computing Curricula) the discipline of software engineering described in SE 2004 is the one that estimates more effort to address the Software Maintenance area.

The SE 2004 is divided into twelve knowledge units. One of these knowledge units is directly related to the maintenance and reengineering: the unit SE7 entitled Software Evolution (see Figure 3). This knowledge unit is divided into two parts: evolution process and evolution activities. The Software Evolution unit includes, among other, software maintenance, the characteristics of software maintenance, reengineering, legacy systems and reuse of software. SE 2004 additionally specifies the certain time that should be dedicated in each unit. In case of software evolution, 14 hours have to be dedicated. Together with desirable time, SE 2004 specifies the attributes using the Bloom's taxonomy [2] refereeing to knowledge (k), comprehension (c), and application (a); as well as the topic's relevance to the core, which is represented as essential (E), desirable (D), or optional (O).

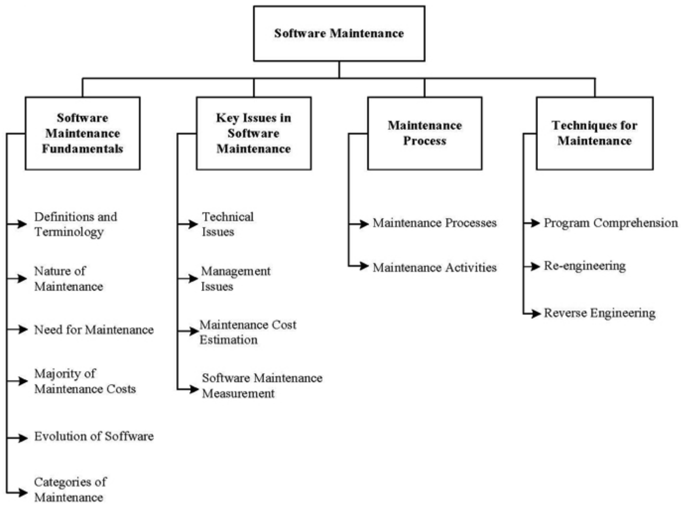
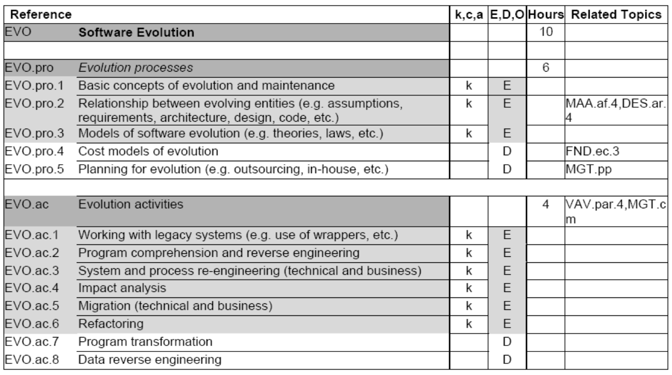

Software Maintenance
Software Maintenance Fundamentals
Key Issues in Software Maintenance
Maintenance Process
Techniques for Maintenance
Definitions and Terminology
Nature of Maintenance
Need for Maintenance
Majority of Maintenance Costs
Evolution of Software
Categories of Maintenance
Technical Issues
Management Issues
Maintenance Cost Estimation
Software Maintenance Measurement
Maintenance Processes
Maintenance Activities
Program Comprehension
Re-engineering
Reverse Engineering

Figure 2. Knowledge areas of software maintenance from SWEBOK [6].



Reference		k,c,a	E,D,O	Hours	Related Topics
EVO	**Software Evolution**			10	
EVO.pro	*Evolution processes*			6	
EVO.pro.1	Basic concepts of evolution and maintenance	k	E		
EVO.pro.2	Relationship between evolving entities (e.g. assumptions, requirements, architecture, design, code, etc.)	k	E		MAA.af.4,DES.ar.4
EVO.pro.3	Models of software evolution (e.g. theories, laws, etc.)	k	E		
EVO.pro.4	Cost models of evolution		D		FND.ec.3
EVO.pro.5	Planning for evolution (e.g. outsourcing, in-house, etc.)		D		MGT.pp
EVO.ac	Evolution activities			4	VAV.par.4,MGT.cm
EVO.ac.1	Working with legacy systems (e.g. use of wrappers, etc.)	k	E		
EVO.ac.2	Program comprehension and reverse engineering	k	E		
EVO.ac.3	System and process re-engineering (technical and business)	k	E		
EVO.ac.4	Impact analysis	k	E		
EVO.ac.5	Migration (technical and business)	k	E		
EVO.ac.6	Refactoring	k	E		
EVO.ac.7	Program transformation		D		
EVO.ac.8	Data reverse engineering		D		

Figure 3. Knowledge unit for software evolution from ACM / IEEE SE [3].

The two mentioned curricula show how the software maintenance process receives as much importance as other processes studied in Software Engineering. Specifically, both curricula consider re-engineering as the main method to support software maintenance. However, when checking current computer curriculum in some European universities it can be seen that in most cases: (i) maintenance is briefly theoretically studied, and (ii) reengineering is hardly ever studied, and may even not be mentioned in the majority of related units.

III. TEACHING EXPERIENCE

This section explains in detail the teaching experience conducted during two last years for assessing the teaching-learning process concerning a reengineering practice. Firstly, the context in which the experience was conducted is introduced. Secondly, the proposed reengineering practice is depicted by providing the teaching resources.

A. Context

The teaching experience was conducted in last two years (2011 and 2012) with the particularity that a new syllabus was introduced for the computer science degree in the last year. In fact, one of the goals established for this experience was the comparison of results obtained in both plans, since these plans entails some important differences such as the course in which the experience was carried out, or the related subjects previously treated by students. The reengineering teaching experience was conducted in the first semester of the course 2011/2012 and 2012/2013 in the Software Engineering subject.

2011 syllabus. In 2011 plan, this subject was taught in the Computer Science BSc with specialization in Management (ITIG) and Systems (ITIS) in the Computer Science Faculty (Escuela Superior de Informática) in Ciudad Real at University of Castilla-La Mancha. In 2011 plan, this subject was taught in an annual term (first and second semester) in the third course (over a total of three years) of the degree and it implies 10 ECTS (European Credit Transfer and Accumulation System). 80 students were enrolled in the subject during that year: 34 students in ITIG and 46 students in ITIS.

2012 syllabus. A new syllabus was introduced in 2012 for the Computer Science BSc at University of Castilla-La Mancha. In 2012 plan, the Software Engineering subject was taught in a half year term (in the first semester) in the second course of four ones in total. This subject implies 6 ECTS. 96 students were enrolled in the subject during 2012.

B. Practice Description

The teaching experience (see Figure 4) first consisted of a theoretical seminar in classroom during one hour and in which the basis and fundaments of reengineering as well as its relationship and contribution to the software engineering field in general and maintenance field in particular were tough. The practical exercise was then carried out in laboratory. The practice took two hours in which the reengineering tools to be used were explained in approximately one hour. After that, the students applied them to solve a practical exercise.

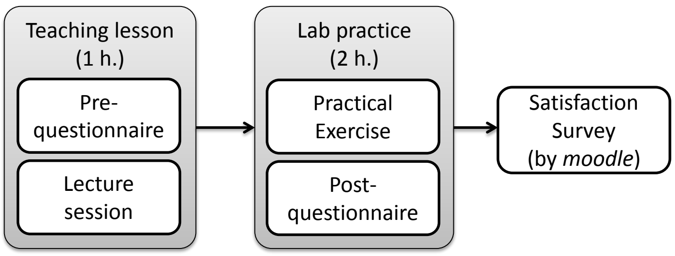


Figure 4. Overview of the teaching experience.

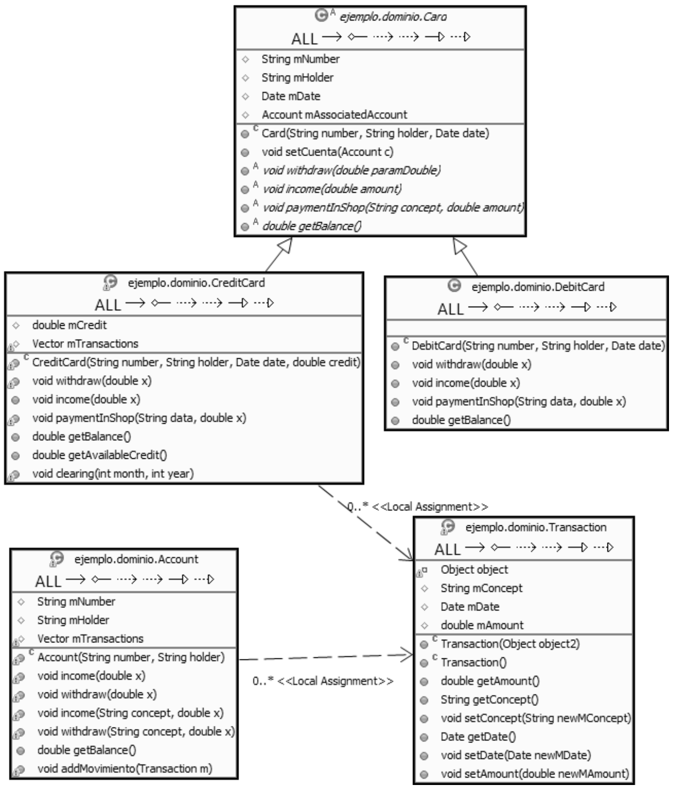

ejemplo.dominio.Card
ALL
String mNumber
String mHolder
Date mDate
Account mAssociatedAccount
Card(String number, String holder, Date date)
void setCuenta(Account c)
void withdraw(double paramDouble)
void income(double amount)
void paymentInShop(String concept, double amount)
double getBalance()
ejemplo.dominio.CreditCard
ALL
double mCredit
Vector mTransactions
CreditCard(String number, String holder, Date date, double credit)
void withdraw(double x)
void income(double x)
void paymentInShop(String data, double x)
double getBalance()
double getAvailableCredit()
void clearing(int month, int year)
ejemplo.dominio.DebitCard
ALL
DebitCard(String number, String holder, Date date)
void withdraw(double x)
void income(double x)
void paymentInShop(String data, double x)
double getBalance()
0..* <<Local Assignment>>
ejemplo.dominio.Account
ALL
String mNumber
String mHolder
Vector mTransactions
Account(String number, String holder)
void income(double x)
void withdraw(double x)
void income(String concept, double x)
void withdraw(String concept, double x)
double getBalance()
void addMovimiento(Transaction m)
0..* <<Local Assignment>>
ejemplo.dominio.Transaction
ALL
Object object
String mConcept
Date mDate
double mAmount
Transaction(Object object2)
Transaction()
double getAmount()
String getConcept()
void setConcept(String newMConcept)
Date getDate()
void setDate(Date newMDate)
void setAmount(double newMAmount)

Figure 5. UML class diagram for the existing informatin system.

The practice in laboratory consisted of the conduction of a reengineering process to modify an existing system. The artefact provided to students was a set of five java executable files (.class), without related documentation. The existing system concerns the bank domain, which contains 5 classes (see Figure 5). Firstly, there is an inheritance tree for specifying *CreditCard* and *DebitCard* from the super class *Card*, which represent a bank card for doing transactions and payments in shops (class *Transaction*). Each card is associated with a bank account (class *Account*), which consider a customer and a set of transactions.

The goal of the practice was to build an improved version of the system which had to fulfil new requirements and to include related documentation (see Appendix IV). Since the existing system was written in Java, the mandatory requirements for carrying out the exercise were Java as programming language and UML as modelling language.

In order to collect the necessary feedback, three questionnaires were designed to evaluate this experience (see Figure 4):

Pre-Questionnaire, which was filled in by the students before the theoretical seminar and its goal was to assess the previous knowledge of the students about reengineering in relation to the previous subjects passed by them. The questionnaire consisted of multiple choice questions and was composed of three theoretical and two practical questions (see Appendix I).

Post-Questionnaire, which was filled on at the end of the practice and included multiple choice questions about Reengineering to assess the acquired knowledge by students as a consequence of the experience. This questionnaire was answered via Moodle platform and included five similar theoretical questions about software maintenance and reengineering as well as the two practical cases of the Pre-Questionnaire in order to have a comparison of results obtained in these cases (see Appendix II).

Final Survey, in which the opinions of students about the experience were collected by using the Moodle platform. The main aim was to assess whether the students agreed about to include this experience as a regular content or unit in the software engineering subject in future years as well as their opinion concerning positive and negative aspects of the experience.

The execution of the teaching experience took place without problems in both years. 68 students participated (30 ITIG, 36 ITIS) in the 2011 experience, which means a participation of 85% of enrolled students. 70 students in total attended to the 2012 experience, which means a participation of 73% of enrolled students. The delivered documentation and software code provided by students was completed in all the cases, thus it was not necessary to discard any of them. The obtained results are analysed in the following section.

IV. Results

This section presents a quantitative and qualitative analysis of the obtained results. Section A analyses results of initial evaluation. Section B discuses results obtained in the practice of laboratory. Section C examines results obtained in the final evaluation by comparing such results with the initial evaluation ones. Section D analyzes how related subjects passed by students affect to the obtained results. Section E focuses on the comparison of results obtained in two different years. Finally, Section E shows the opinion of students regarding the teaching experience.

A. Initial Evaluation

Regarding conceptual questions of pre-questionnaire (see Appendix I) about software maintenance and reengineering fundaments and basis, it was observed that over 70% and 76% of students (respectively from 2011 and 2012 experience) had a slight or well-formed idea about software maintenance. However, only 22% and 13% of students had an idea about how reengineering works (respectively based on the second question of pre-questionnaire of 2011 and 2012 experience). Furthermore, concerning the third question so that students establish relationships between software maintenance and reengineering; only 18% and 17% of students (respectively from 2011 and 2012 experience) were able to relate software maintenance and reengineering. These results clearly indicate students do not know that reengineering is as a technique of software maintenance, which justifies the teaching experience on this area.

The comparison between results collected from 2011 and 2012 experience shows that 2012 students have a clearer idea of software maintenance than 2011 ones, while 2011 students proved to have a better preliminary comprehension of reengineering than 2012 ones.

Concerning practical cases, students obtained good results taking into account they have never learn reengineering. 2011 students achieve a success ratio (i.e., the percentage of students that choose the right answer) of 62% and 66% for the two case studies respectively (see appendix III). 2012 students even reach a higher success ratio with 71% and 86% for the two practical cases. 2011 and 2012 results have in common that success ratio obtained in the second case was higher than the results obtained in the first case.

B. Lab Practical Exercise

Table I shows the marks obtained for each student group in both experiences (2011 and 2012). Each group was formed with three to five students. Table I also shows the evaluation criteria that were not fulfilled, which are later depicted in Table II. The mean of marks obtained in 2011 was 8.18 with a standard deviation of 0.9, while the 2012 experience provided a mean of 8.45 with a standard deviation of 1.02. These marks indicate that students in both years did not find many difficulties for addressing the proposed practice. Despite of this fact, it was realized a set of common mistakes that students systematically repeated which are described as follows.

Table II provides the description of evaluation criteria that were repeatedly failed as well as its frequency during the experience. On the one hand, the most common error consisted of the wrong integrations of the necessary, new *Customer* class with the remaining classes of the system during the restructuring stage. It was due to (C2) the absence of the necessary dependencies with other classes (which was repeated by the 93% and 100% of the 2011 and 2012 students respectively); and (C3) owing to the absence of associations, i.e., the types of attributes representing customers were not modified appropriately (which was made by the 41% and 43% of the 2011 and 2012 students).

On the other hand, with a lower repetition, compilation and source code mistakes were also detected as common mistakes (C5), which affected in most cases to the absence or wrong launcher class that students had to implement (C3). In fact, the launcher class did not work in the 44% and 49% of the 2011 and 2012 students. Finally, 31% and 34% of 2011 and 2012 students respectively contextualized in a wrong way some of the reengineering stages during the steps made during the practice (see C4 in Table II).

TABLE I. MARKS BY STUDENT GROUPS OBTAINED IN THE PRACTICAL EXERCISE

Year	Group	Marks [0-10]	Common Mistakes				
			C1	C2	C3	C4	C5
2011	1	9.0		♦			
	2	8.0		♦	♦		
	3	9.0		♦			
	4	8.0		♦	♦	♦	
	5	7.5	♦	♦		♦	
	6	8.5		♦	♦		
	7	9.0		♦			
	8	8.0	♦	♦			
	9	9.0		♦			
	10	7.5	♦	♦	♦		
	11	8.5		♦		♦	
	12	7.5	♦	♦	♦		
	13	6.0	♦	♦	♦	♦	♦
	14	9.0	♦				
2012	1	9.5		♦			
	2	8.5	♦	♦		♦	
	3	9.5		♦			
	4	9.0		♦	♦		
	5	9.0		♦		♦	
	6	7.5	♦	♦	♦		♦
	7	7.5	♦	♦	♦	♦	
	8	9.0		♦	♦		
	9	8.0	♦	♦	♦	♦	
	10	6.0		♦	♦		♦
	11	6.5	♦	♦	♦	♦	
	12	9.5		♦			
	13	8.5		♦			
	14	7.5	♦	♦	♦	♦	
	15	9.0		♦			
	16	9.0		♦			
	17	9.5		♦			
	18	9.5		♦			
	19	8.5	♦	♦	♦		
	20	8.0	♦	♦	♦	♦	

TABLE II. MISTAKES AND THEIR FREQUENCIES IN THE PRACTICAL EXERCISE

ID	Evaluation criteria	Percentage	
		2011	2012
C1	The *Customer* class is not well integrated (loosing associations)	41.2%	42.9%
C2	There is a lack of dependencies with the new *Customer* class	92.6%	100.0%
C3	The *Launcher* class does not work to test the system.	44.1%	48.6%
C4	Misleading documentation regarding reengineering stages.	30.9%	34.3%
C5	Source code and/or compilation mistakes	7.4%	8.6%

The comparison between results collected from 2011 and 2012 experience shows that 2012 students obtained worse results than 2011 students. This contrasts with the results obtained in the pre-questionnaire with theoretical concepts, in which 2012 students obtained better results than 2011 ones.

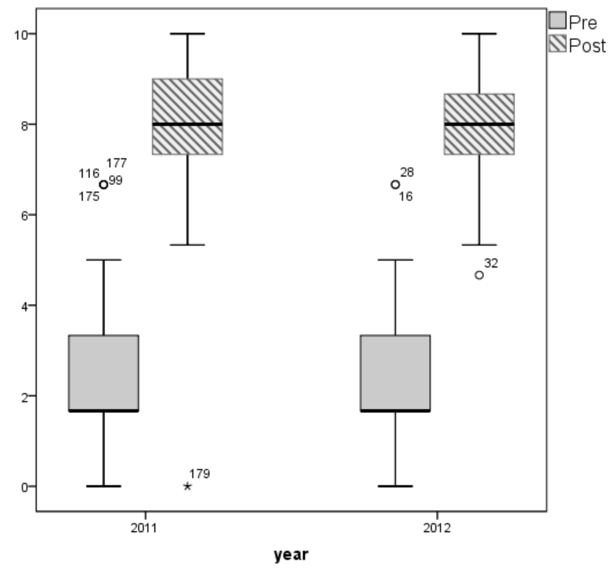


Figure 6. Box plot for reengineering understanding comparison.

C. *Post Evaluation*

The results retrieved at the end of the experience by means of the post questionnaire were compared with the initial evaluation. The comparison was made by analyzing the marks obtained from the evaluation of three first questions of the pre-questionnaire (see Appendix I) and the first block composed of four questions of true/false type of post-questionnaire (see Appendix II). This comparison shows that the students' perception about reengineering significantly changed in both years. In 2011, the 70% of students made a right definition of reengineering in post-questionnaire (a mark greater or equal to 5), which was an increase of 66% from the initial 4%. In the 2012 experience, the initial comprehension of 15% increased 68%, since the final comprehension about reengineering in post-questionnaire was 83% (see Figure 6). Despite the final understanding was better in 2012 than in 2011, the increase of understanding was 66% and 68% in both experiences. As a result, the teaching-learning experience was equally effective in both years.

Additionally, both the pre-questionnaire as the post-questionnaire were qualified with a mark between 0 and 10 for each student. It was observed that the score obtained from the post-questionnaire in 2011 was 7.8 on average, while the mean of marks obtained in pre-questionnaire was only 2.2. The marks of 2012 students followed a similar trend with a mean of 7.9 and 2.0 for the post- and pre-questionnaire respectively. Figure 7 shows the distribution of marks obtained from the two questionnaires in both years. The marks in all the cases followed a normal distribution with the mentioned means and standard deviations of 1.4 for post-questionnaires of both years and 1.5 and 1.9 for pre-questionnaires of 2011 and 2012. These results demonstrate that the proposed reengineering teaching experience increased the knowledge about reengineering and software maintenance of students. The influence of the teaching-learning process additionally was uniform, since the distribution of marks in the post-questionnaire had standard deviations less than standard deviations obtained after pre-questionnaires.

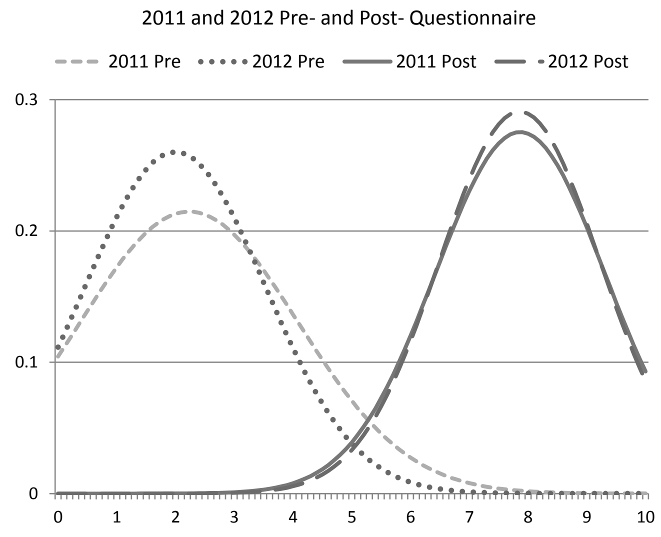


Figure 7. Distribution of marks obtained in the initial and last evaluation

Moreover, the difference results obtained in the two practical case studies made in both questionnaires was assessed. For this purpose, a Student's test was carried out with the results of both questionnaires for each case (see Table III). In 2011, there is a mean difference between the initial and final evaluation with a significance degree of 99% in both practical cases. The mean difference is the same in the two case studies (-0.212). The negative difference means that the average score obtained from the post-questionnaire was greater than the score at the beginning of the experience in 2011. In fact, the effect size values demonstrate that the students improved more in the second case study (-0.522) than in the first one (-0.496).

In 2012 experience, there is a significant mean difference only in the first practical case with a significance degree of 99% (see Table III). In this case, there is a mean difference of -0.185 with an effect size of -0.468. These results are very similar to 2011 results. However, the second practical case in 2012 did not report a significant difference. Although the score obtained in post-questionnaire was greater than the pre-questionnaire score, there is no a significance value above 95%. As a result, in this case nothing can be stated about the improvement of results after the reengineering teaching experience.

TABLE III. COMPARISON OF RESULTS OBTAINED IN PRACTICAL CASES.

		Mean Difference	Standard Deviation.	T-Student	Effect Size	Significance
2011	Practical Case I	-0.212	0.57	-3.03	-0.496	0.004
	Practical Case II	-0.212	0.41	-4.18	-0.522	0.000
2012	Practical Case I	-0.185	0.46	-3.21	-0.468	0.002
	Practical Case II	-0.077	0.37	-1.69	-0.258	0.096

D. Effect of related subjects

Another important aspect that has been assessed is the effect of related subject to the teaching-learning process in this experience. In order to quantify this effect, related subjects passed by each student were collected together with the score and marks obtained in each activity (i.e., pre- and post-questionnaire and the practical exercise).

The related subjects under study were (i) Fundaments of programming; (ii) Data structures; (iii) Advanced programming; and (iv) Databases. Since the syllabus was changed in 2012, the nature and duration of these subjects was different (see Table IV). Hence, different effects in results are expected.

The analysis of the effect of related subjects was carried out by applying the anova statistical test. The anova test analyzes the variance of various sub-samples with respect to a factor. Thus, the null hypothesis is H_0: $\mu_1 = \mu_2 = \mu_n$, while the alternative hypothesis means that there is a significant difference between the means of sub-samples, i.e., H_1: $\mu_1 \neq \mu_2 \neq \mu_n$. In this case, various *anova* tests were carried out by choosing as factor if a certain related subject was (or was not) passed. Also, different *anova* test were conducted by each score or mark obtained in all the different activities in both years.

After applying the anova test, the null hypotheses of all the tests cannot be rejected since the significance values were greater than 0.05. As a consequence, related subjects and its different nature do not affect to the results obtained in both experiences. This means that the different results obtained in both years could be explained due to the random effect.

Despite random effect, Table V provides all the cases in which students who passed a particular subject obtained better results in a certain activity. Table V shows that 2011 students who had passed most related subjects obtained better results in the practical exercise than students who had not passed them. Another interesting insight is that, at the contrary 2011, 2012 students who had passed most related subjects did not obtained necessarily better results. However, these students obtained better results in practical cases of post-questionnaires of 2012 (see Table V).

TABLE IV. NATURE OF RELATED SUBJECTS

Subjects	2011			2012		
	Course	ECTS credits	Duration	Course	ECTS credits	Duration
Fundaments of programming	1st	13	Annual	1st	6	1st Sem.
Data structures	2nd	10	Annual	2nd	6	1st Sem.
Advanced Programming	2nd	7.5	2nd Sem.	1st	6	2nd Sem.
Databases	3rd	7	Annual	2nd	6	2nd Sem.

TABLE V. EFFECT OF RELATED SUBJECTS IN OBTAINED RESULTS

Year	Score / Marks		Fundaments of Programming	Data Structures	Advanced Programming	Databases
2011	Pre	Questionnaire		↑		↑
		Case I				↑
		Case II	↑		↑	↑
	Post	Questionnaire		↑	↑	
		Case I	↑			↑
		Case II		↑	↑	
	Practical Exercise		↑	↑	↑	
2012	Pre	Questionnaire	↑	↑		
		Case I		↑	↑	↑
		Case II	↑		↑	
	Post	Questionnaire				↑
		Case I	↑	↑	↑	↑
		Case II	↑	↑	↑	↑
	Practical Exercise					

TABLE VI. ANOVA TEST RESULTS FOR THE EXPERIENCE YEAR (2011/2012)

Score / Marks		Quadratic Mean	F-value	Effect Size	p-value
Pre	Questionnaire	1.472	0.511	0.122	0.476
	Case I	0.322	1.444	-0.190	0.232
	Case II	1.317	7.526	-0.477	0.007
Post	Questionnaire	0.002	0.001	0.005	0.975
	Case I	0.141	1.229	-0.177	0.270
	Case II	0.079	0.910	-0.170	0.342
Practical Exercise		1.368	1.971	-0.241	0.163

E. *Comparison based on Two-Year Experience*

Besides of previous analyses, a comparison between results obtained in both experiences was conducted. An *anova* test was carried out for assessing the effect of the experience year (2011 and 2012). Table VI provides results of the *anova* test, which shows that only the second practical case of the pre-questionnaire is affected by the year with a significance degree of 95%. In this case, since the effect size is negative (-0.477) the results obtained in the second practical case were better in 2012 than 2011.

In remaining cases, the null hypotheses cannot be rejected, i.e., different results obtained in both years are due to the random effect. Anyway, Table VI provides the effect size values, which quantify the difference between results. Positive values indicate 2011 results were better than 2012 and *vice versa*.

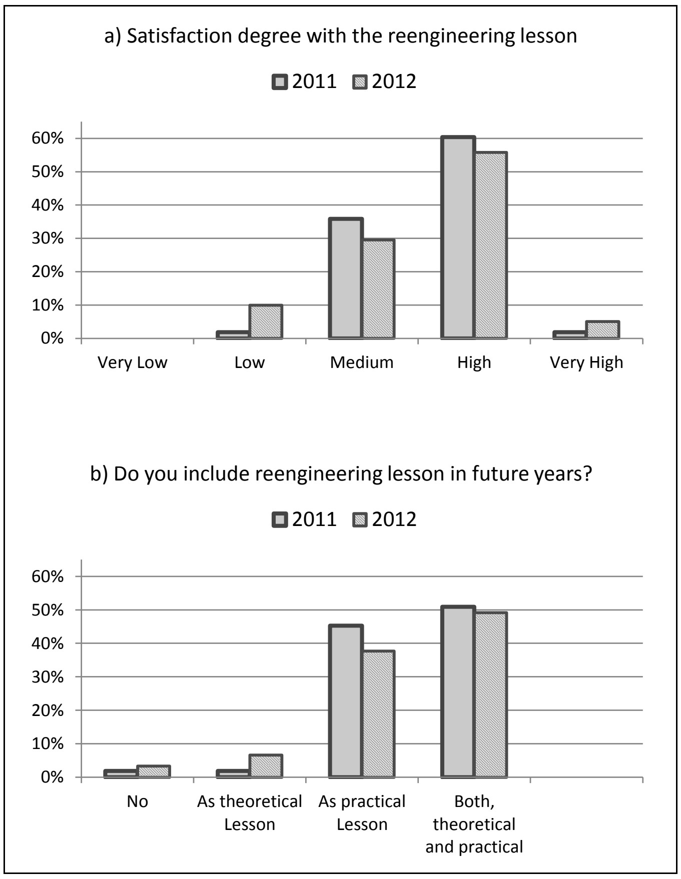


Figure 8. Results of the satisfaction survey

F. *Satisfaction Survey*

At the end of teaching experience, a feedback survey was distributed via the corporative *Moodle*-based system in order to obtain the students' opinion about the teaching experience. When it was asked if they would include the maintenance and reengineering in the program for future years, between 49% and 51% of 2012 and 2011 students indicated that they will include both theoretical as practical lesson. Furthermore, between 37% and 45% would include software maintenance and reengineering at least in practical lessons (see Figure 8 a).

Concerning the question to know their satisfaction (see Figure 8 b), it was quantified with a scale between 1 and 5 (i.e., very low, low, medium, high and very high). In 2011 experience, 60% of students had a high satisfaction (4), 35% a medium satisfaction (3), 2% very high (5). Whilst, the 2012 survey reported that 56% of students had a high satisfaction (4), 30% a medium satisfaction (3), 5% very high (5). Satisfaction of 2012 students was less concentrated among medium and high values than 2011 results. Nevertheless, there were more students with a low and very high satisfaction.

This questionnaire also asked for the positive and negative points of this experience. Most students provided as good points that: (i) they learned how to reuse code to avoid green-field software developments in every case; and (ii) they positively evaluated the knowledge and usage of new reengineering-based tools, especially such tools related to the reverse engineering stage as de-compilers, which they had never used.

Finally, the students almost unanimously stated as a negative aspect the lack of time during the practice of laboratory to end the practical exercise as well as the bit time dedicated for the explanation of reengineering tools to be used in laboratory. The lack of time can be partially explained by the time spent on the assessment. Cost of assessment is often stolen from teaching, and therefore also from learning [11].

V. CONCLUSIONS

Teaching in software engineering must be addressed to enable students to develop their future work in current professional environment, which are demanding tasks and skills that are sometimes not covered within current academia curricula. A vast number of software engineers are nowadays working in software maintenance tasks by applying software reengineering. Unfortunately, reengineering often receive a bit attention in terms of teaching.

This paper presents a teaching experience conducted within a software engineering subject of a computer science degree. As a result, initial evidences showed: (1) a lack of previous knowledge by students about software maintenance and reengineering, and (2) how students can deal with this gap by a practice specifically designed to acquire the necessary, basic training about software maintenance and the application of software reengineering. The positive results of this two-year experience will be helpful to introduce the special issues related to software maintenance and reengineering into the subject of the computer science degree.

Among the lessons learned, which could be applied to future repetitions of the teaching experience, we included:

1. During the development of the practice in the laboratory, the major challenge for students was the introduction of new functionalities in the target system. The hardest difficulties were regarding the integration of new functionalities into the new systems during the restructuring stage.

2. One of the most common faults made by students was the confusion of reengineering stages, since students did not demonstrate to have a clear comprehension of frontiers between reverse engineering, restructuring and forward engineering. To mitigate this threat, the explanation of reengineering stages may be extended in future experiences.

3. During practical exercise, a common bad practice carried out by most students was that they almost directly modified and restructured the existing information system at code level instead of at UML level.

4. Students identified the lack of time as a handicap to conclude the practical exercise with better results. Therefore, new experiences may be carried out with additional time.

5. Students expressed their approval about the use of new tools like de-compilers and other reverse engineering applications to generate UML design models. Nevertheless, students indicated the necessity of an in-depth explanation of such tools. An additional lesson related to these tools could be included in future teaching experiences.

6. Concerning the two different syllabi in which the experience was conducted, we learned that reengineering could be taught at an elementary level in first courses or could be addressed in depth in last courses as we tested in 2011 experience. These both experience have therefore provides us with a better understanding about how to include reengineering in the new syllabus.

ACKNOWLEDGMENT

This work was supported by the FPU Spanish Program and the R&D projects PEGASO/MAGO (TIN2009-13718-C02-01), MAESTRO (Alarcos Quality Center) and GEODAS-BC (TIN2012-37493-C03-01).

REFERENCES

[1] Bianchi, A., D. Caivano, V. Marengo, and G. Visaggio, Iterative Reengineering of Legacy Systems. IEEE Trans. Softw. Eng., 2003. 29(3): p. 225-241.

[2] Bloom, B.S., Taxonomy of educational objectives; the classification of educational goals. 1956, New York: Longmans, Green.

[3] Computing Curriculum Project, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (SE2004). http://sites.computer.org/ccse/. 2004, IEEE Computer Society & ACM.

[4] Chikofsky, E.J. and J.H. Cross, Reverse Engineering and Design Recovery: A Taxonomy. IEEE Softw., 1990. 7(1): p. 13-17.

[5] El-Ramly, M., Experience in teaching a software reengineering course, in Proceedings of the 28th international conference on Software engineering. 2006, ACM: Shanghai, China. p. 699-702.

[6] IEEE Computer Society, Guide to the Software Engineering Body of Knowledge (SWEBOK). http://www.computer.org/portal/web/swebok. 2004.

[7] ISO/IEC, ISO/IEC 14764:2006. Software Engineering -- Software Life Cycle Processes -- Maintenance. http://www.iso.org/iso/catalogue_detail.htm?csnumber=39064. 2006, ISO/IEC.

[8] ISO/IEC, ISO/IEC 12207:2008 - Systems and software engineering -- Software life cycle processes. 2008.

[9] Kazman, R., S.G. Woods, and S.J. Carrière. Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. in Proceedings of the Working Conference on Reverse Engineering (WCRE'98). 1998: IEEE Computer Society.

[10] Lehman, M.M., On understanding laws, evolution, and conservation in the large-program life cycle. Journal of Systems and Software, 1979. 1: p. 213-221.

[11] Schagaev, I., E. Bacon, N. Folic, and N. Ioannides, Curriculum Design, Development and Assessment for Computer Science and Similar Disciplines.

[12] Sneed, H.M., Planning the Reengineering of Legacy Systems. IEEE Softw., 1995. 12(1): p. 24-34.

[13] Sneed, H.M., Estimating the Costs of a Reengineering Project. Proceedings of the 12th Working Conference on Reverse Engineering. 2005: IEEE Computer Society. 111 - 119.

Appendix I. Pre-Questionnaire

1. – What do you know or understand about the concept of 'software maintenance'? When do you think software maintenance is applied within the software development lifecycle?

2. – What is software reengineering and in which cases reengineering should be used?

3. – Which are the relationships between software maintenance and reengineering?

** The pre-questionnaire is completed with the two practical cases depicted in Appendix III.*

Appendix II. Post-Questionnaire

1. – Tick with T (True) or F (False) the following statements: (KEY: T, F, T, F, F, F)

- Reengineering can be used to carry out the maintenance of information systems.
- Reengineering is a type of engineering applied through CASE tools.
- Reengineering can obtain improved versions of information systems.
- Reverse Engineering = Reengineering
- Reengineering = Refactor
- Reengineering = Migration

2. – Indicate which of the following factors are a reason to launch the maintenance of an existing information system. (KEY: D)

A. To adapt the system to new technologies, programming languages, etc.

B. To incorporate new functionalities or meet new requirements in the system.

C. To solve/mitigate faults and bugs in the system (maintainability).

D. All factors are right.

3. – Indicate which of the following factors are a reason to apply reengineering with an existing information system. (KEY: D)

A. To adapt the system to new technologies, programming languages, etc.

B. To incorporate new functionalities or meet new requirements in the system.

C. To solve/mitigate faults and bugs in the system (maintainability).

D. Any factor between (A) and (B).

E. Any factor between (A), (B) and (C)

4. – Indicate which of the following factors are a reason to withdraw an information system and develop a new one to replace it. (KEY: C)

A. Absolutely never. It is always better to make reengineering

B. When the goal is to obtain high quality systems, which cannot be obtained through reengineering

C. Preferably never. It is better to make reengineering unless the cost of a development from scratch is less than to continue maintaining it.

** The post-questionnaire is completed with the two practical cases depicted in Appendix III.*

Appendix III. Practical Cases.

Below are two case studies. For each case you are the project manager and are in charge of the decision making of such software engineering projects. Please, provide the most appropriate answer in each case:

CASE 1. There is a system built using a structured development methodology and was written in C. The goal is to migrate the mentioned system to another system developed according to the object-oriented paradigm, which have to be written in Java. (KEY: C)

A. To do nothing. If the system works well, why should we change it to Java? Migration may imply very high costs.

B. Since the goal is a mandatory requisite, several programmers with a C and Java expertise should work to obtain a new system from scratch.

C. Various programmers with a C and Java expertise should work in the new system. Looking ahead, an object-oriented development will be easier to modify (due to inheritance, polymorphism, etc.). The investment made now will be returned in future cheaper developments.

D. The migration is not necessary. The requirement specification will be collected through customer interviews so that a new Java-based system can be built according a green-field development. The comprehension of the C code is too much expensive and is not absolutely necessary.

CASE 2. There is a selling management system that stores all the customer and product information in plain text files. As a consequence, the possibility to change the data model is being evaluated so that a relational database can replace the data access based on text files. Please, select the most appropriate choice. (KEY: B)

A. If the existing system is connected to a database, certain features may not work properly after the incorporation of the relational database. The creation of a new system supporting the same functionalities with a relation database is a better option.

B. Adding the relational database is the better option, but the sole change in the system would be the creation of a database agent to handle the data access to/from the existing system.

C. If the system works well, why should a relation database replace the data management based on plain text files? In the future, when a new version of the existing system is built the relational database will be incorporated but not before.

APPENDIX IV. PRACTICAL EXERCISE STATEMENT

For this exercise, a reengineering project should be carried out in group. Each group will take as input software artifact a set of executable files (*.class) which belong to an existing, obsolete information system. The documentation of the system is missing and its functionality is unknown.

The goal is to obtain an improved version of the system, together with the documentation based on UML diagrams, in which the following changes have to be made:

- To add the Customer class (with ID, first and last name, as well as birthday date). This new class must be used in all cases in which a String-type variable represents a customer.
- To develop a Main class to allow operating main functionalities of the improved system.

The group have to follow and document all the three reengineering stages (reverse engineering, restructuring, and forward engineering) to obtain the enhanced information system.

As a result, the group will upload the following deliverables:

- A portfolio describing the steps followed in each reengineering stage by indicating the problems found. Furthermore, software artifacts obtained in each stage will be listed.
- Lessons Learned
- UML Diagrams
- Source Code of the improved information system