Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp285-288)

Lessons learned from an open-source University project

P. BASDARAS, K. CHALKIAS, A. CHATZIGEORGIOU, I. DELIGIANNIS, P. TSAKIRI, N. TSANTALIS
Department of Applied Informatics
University of Macedonia
156 Egnatia str., 54006 Thessaloniki, Greece
GREECE
achat@uom.gr http://csse.uom.gr

Abstract: - Open-source software development has become a widespread trend within the software engineering community and has begun to attract the attention of other disciplines as well. To help students understand the practices used within the open-source movement and to initiate an effort for logging all aspects of the software development process, our Laboratory has set up an open-source project targeting undergraduate and postgraduate computer science students. The main aim of the project is to systematically record architecture and code related information as well as "soft" issues related to the individuals that take part in the development. The paper discusses results and conclusions drawn from this project.

Key-Words: - open-source, software development, software engineering, student projects, bet prediction and analysis, software architecture, software metrics

1 Introduction

According to the Open Source Initiative (OSI) [1] the basic idea behind open source software is that when programmers are free to read, redistribute and modify the source code for a piece of software, the software evolves. As it can be observed from the amount and quality of code committed to the numerous available repositories, the open-source movement has a significant momentum. The release of the source code of Netscape's Navigator back in 1998 is considered widely the first spark of the open-source community.

Considering the widespread acceptance of open-source as a philosophy, it is reasonable to expect Computer Science departments to incorporate relevant courses in their curricula. Students should become familiar with the philosophical, technological, legal and social issues related to open-source software. Participation in an open-source project would be the ideal vehicle for demonstrating these aspects; however, software writing is on a voluntary basis and students cannot be forced to take part, considering also the tight schedules of most academic semesters.

An open-source software project can also be a rich source of information concerning the development process. By keeping track of various software versions, development effort, programmer capability, lines of code, type of changes etc, a large database of project related data can be formed. Analysis can highlight the software evolution in terms of architectural changes, development effort versus skills, bugs versus size and numerous other associations between software parameters.

In respond to these challenges and possibilities the Computational Systems and Software Engineering Lab (CSSE) of the Department of Applied Informatics at the University of Macedonia, Greece, has initiated an effort to create the infrastructure for an open-source software project. The main aim of the project is twofold: a) to provide the means for students who wish to participate in a project that is developed according to the rules and practices of the open-source community and b) to collect data concerning all aspects of the software development process that can be qualitatively and quantitatively analyzed and correlated.

2 Project Description

Substantial thought has been devoted to the selection of the project domain, so as to motivate as many students as possible. Secondly, the selected project should be expandable in a number of axes, to create the potential for a large number of releases. Typical choices for an academic environment, such as Information Systems (CRMs, ERPs etc) have been discarded, considering the need for voluntary work.

The area that has been finally selected is that of bet analysis and prediction, something, which at least for male students, proved to be a very exciting and "hot" topic. This was clearly evident from the participation in the first group meetings. Moreover, software for organizing and analyzing bets can be expanded to many directions. Representative examples are automatic coupon management, statistical analysis of past data, visualization in

This work was funded by the program “ARCHIMIDIS II – EPEAEK II” which is supported by the European Union and the Greek government. The particular research grant was awarded to Project 2.2.6.

appropriate formats, use of artificial intelligence for prediction, information retrieval through Web agents, risk analysis etc. An important issue was also that many students had prior knowledge of the domain from previous experience and these students could lead the rest of the developers. The project was named "OpenBet" combining the philosophy of open source with bet analysis. The project's homepage is located at [2]. The selected programming language for the implementation is Java due to the following reasons:

- it is an object-oriented language and as such it is modular and appropriate for open-source development
- the undergraduate students in our department are familiar with Java
- a large number of available API's facilitates the implementation

The project has been active for almost a year, considering also organizational activities prior to the beginning of the development. The development is based on a widely used CVS (Concurrent Versions System) which is managed by a core team of students. The administrative team is responsible for checking the submitted code (primarily whether the system remains functional) and for performing major architectural modifications. Every student is free to enhance the functionality in any direction he wishes and also to perform corrective or perfective maintenance. However, the project's homepage maintains a "to do list" with useful additions. The project has been announced and the required tools explained to both undergraduate and postgraduate students in our department.

To aid in the collection of data concerning software development, each participant, in order to obtain an account, has to provide information regarding his age, studies, experience in programming languages etc. Moreover, when committing a class (the smallest piece of code that can be committed), each developer has to fill in the following log form:

```
Class Name:
Consumed Time (mins):
Maintenance
    adaptive:
    corrective:
    perfective:
Locality of change
    Local:
    Propagated:
    origin of propagation (list):
```

In this way, various pieces of information concerning the affected classes can be investigated, i.e. the development effort, the type of change performed to the class (adaptive corrective, perfective [3]) and whether the change was local or propagated from another class.

To support the open-source project two programs have been developed by students of the Lab. In particular, a program that compares different versions of the developed software, which automatically extracts the new, deleted or modified methods/attributes and also categorizes method modifications between successive versions. In addition, another program was written to analyze the aforementioned log forms and to systematically record and visualize the corresponding information submitted by each developer.

3 Observations

A number of general conclusions can be drawn concerning the overall progress of the specific open-source project which aimed at the voluntary participation of students:

- The participation of students (regardless of whether they commit or download code) helps them to comprehend the basic practices and tools of open-source software development.
- As it would be expected, development activities are carried out by a limited number of students. As the project evolves and becomes more complex, the number of students who are able to participate is also decreasing. An exception is the case of simple requirements that have limited correlation to the rest of the functionality. Such requirements can be implemented at any time even by "newcomers".
- Whenever new requirements were published or announced, a period of relatively intense programming activity followed. At other times, a slowdown is observed and according to our feedback this is mainly because students are not used to take initiatives for adding new functionality.
- The number of bugs in the developed software is limited, possibly because of the effort of each participant to produce correct code, taking into account the closeness of the community in this particular project.
- All students have been very responsible in following all formal and informal rules that have been agreed and in filling in the required logs and internal code documentation.
- Multiple generations can be produced in a relatively limited time period providing data for analyzing the development process. However, the amount of new functionality in each version varies depending on the programmer.
- Although incentives have been given to the students in order to increase participation, further

"advertisement" should be made emphasizing the differences of an open-source project from other software projects during academic courses.

4 Results

In this section results concerning product and process metrics will be presented. The system to be analyzed has evolved through 44 functional commits, while the last version consists of 13 classes (classes have been added as well as removed during the project).

The number of lines of code (LOC) per class per version is shown in Figure 1. In general, the size of individual classes has not significantly changed; the system size increases (almost linearly) mainly due to the addition of new classes as the system evolves. The total number of operations in all system classes (NOO) increases also linearly as shown in Figure 2.

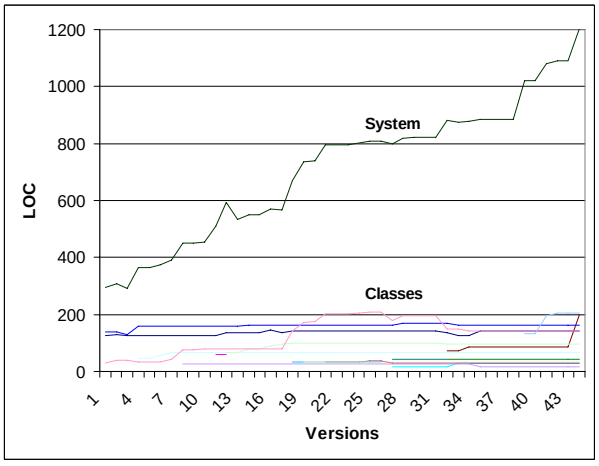

Figure 1: Lines of Code per class per version

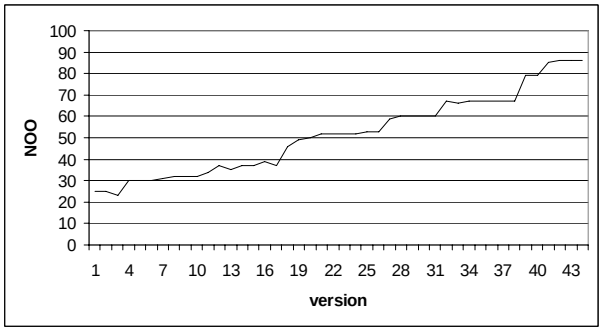

Figure 2: Total number of operations per version

To assess the evolution of the system architecture three representative and widely used metrics have been employed: Coupling Between Objects (CBO) for quantifying the interdependency between classes, Lack of Cohesion in Methods (LCOM) for evaluating the relevance of functionality between methods in each class and Weighted Methods per Class (WMC) for assessing the complexity of each module [4]. The results are graphically depicted in Figure 3.

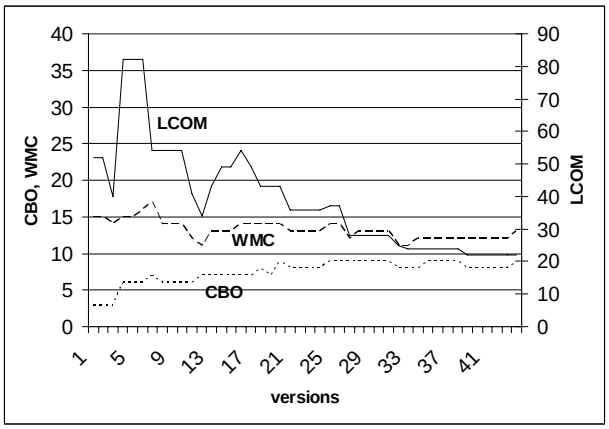

Figure 3: Evolution of software metrics

As it can be observed, the system gradually matures in terms of cohesion (cohesion increases). This is probably due to the fact that initially code was placed in a few classes regardless of the functionality, while during the course of the project distribution of the functionality to several classes has been performed. On the other hand, coupling increases slightly since no effort has been devoted in reducing the dependency between classes. Finally, a small reduction of method complexity can be observed, which might also be explained by the distribution of functionality to several classes.

As already mentioned, data has also been collected based on the logs filled in by the programmers during each commit. Figure 4 shows the number of commits per class. It becomes obvious that a number of classes required significantly more modifications than others.

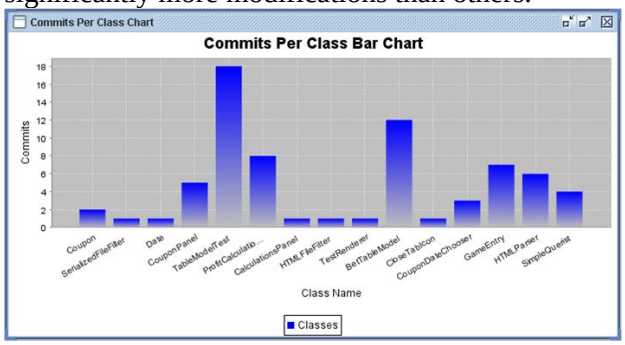

Figure 4: Number of commits per class

The observations from the latter figure are in agreement to those drawn from Figure 5 which displays the development time devoted to each class. Again, there is no uniform distribution of effort, something which possibly implies that there is ground for further improving the design in terms of responsibility distribution among the classes. On the other hand, such data clearly illustrates why monitoring an actual industrial project is crucial, so than capable programmers are allocated to the most time and effort consuming modules.

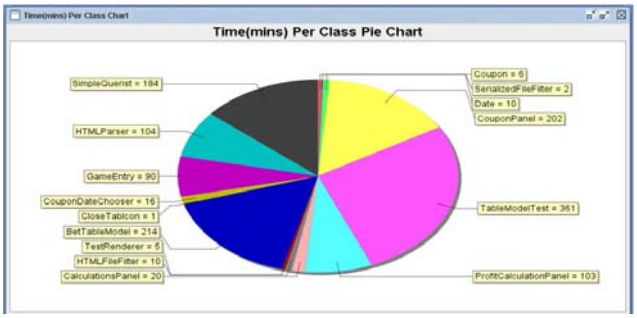

Figure 5: Development Time per class

Figure 6 summarizes the number of changes of each type, made to each class. The percentages indicate the fraction of changes of a given type in a specific class over all changes of the same type. The value on the top horizontal axis indicates the absolute number of changes of each type in each class. As it would be expected, adaptive changes constitute the largest part of maintenance while it is confirmed that corrective maintenance is limited.

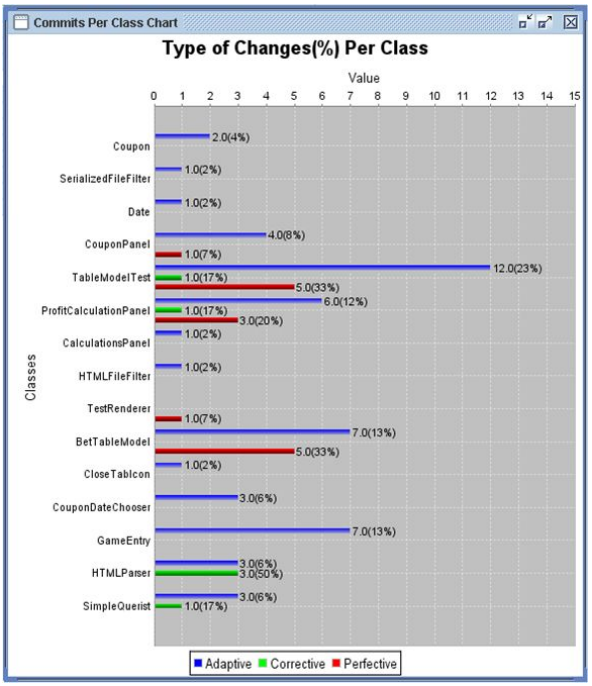

Figure 6: Type of Change per class

One of the most interesting aspects in getting feedback from the developers, is the ability to keep track of propagating changes, that is, modifications to a class that affect also other classes. For example, changing the signature of a method would enforce all clients of this method to change to modify the corresponding method invocation [5]. The extraction of this information by manual inspection would be an extremely difficult and error prone process. The corresponding results concerning propagated changed in the history of OpenBet are shown in Figure 7. An integer *n* at cell (*x*, *y*) indicates that *n* changes made to class *y* have propagated to class *x*.

	Coupon	SerializedFileFilter	Date	CouponPanel	TableModelTest	ProfitCalculationPane	CalculationsPanel	HTMLFileFilter	TestRenderer	BetTableModel	CloseTabIcon	CouponDateChooser	GameEntry	HTMLParser	SimpleQuerist
Coupon															
SerializedFileFilter															
Date															
CouponPanel						1				1					
TableModelTest	1								1	2		1			
ProfitCalculationPanel	1									2			1		
CalculationsPanel															
HTMLFileFilter															
TestRenderer										1					
BetTableModel	1					1							1		
CloseTabIcon															
CouponDateChooser			1		1										
GameEntry															
HTMLParser															
SimpleQuerist															

Figure 7: Propagated changes: origin and destination

The corresponding tool is also capable of providing statistics concerning individual developer participation, such as the number of commits, time and type of changes per author. Finally, it also displays all software versions as a tree, annotated by information concerning the author, date, affected classes etc.

By correlating the results, further conclusions can be drawn which might be surprising to the students at a first glance. For example, the largest classes (in LOC) are not necessarily the ones in which the most development effort has been devoted (there is however a positive correlation between the two). Such analysis, however, should be carried out with care, considering the evolution of the system and the functionality of each class.

5 Conclusion

An open-source software project has been initiated in the Department of Applied Informatics at the University of Macedonia, Greece. The main aim of the project was to motivate students in participating in a small open-source community. The results indicate that such projects are beneficial both from a pedagogic point of view and as a tool for collecting data concerning the software development process.

References:

[1] Open Source Initiative, http://www.opensource.org
[2] OpenBet, http://www.openbet.gr
[3] C. Ghezzi, M. Jazayeri and D. Mandrioli, *Fundamentals of Software Engineering*, 2nd edn, Prentice Hall, Upper Saddle River, NJ, 2003.
[4] S. R. Chidamber, C.F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, 20 (6) June 1994, pp. 476-493.
[5] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, "Predicting the Probability of Change in Object-Oriented Systems", IEEE Transactions on Software Engineering, 31 (7) July 2005, pp. 601-614.