Evaluating Student Participation in Open Source Software Development with an Annotation Model

Robert Charles[1] and Yonglei Tao[2]
[1]LORIA, Campus Scientifique, BP 239, 54506, Vandoeuvre Cedex, France
email: abiodun-charles.robert@loria.fr
[2]Grand Valley State University, Allendale, MI 49401, USA
email: taoy@gvsu.edu

ABSTRACT
While team work is an indispensable experience for computer science students, evaluating their performance in a project becomes a great challenge for the instructors. The basic assumption in a collaborative work is that each member of the collaboration has something to contribute. It is assumed that no member of the team is non functional. But how do we measure the contribution of each member of the team? An effective documentation tool is helpful. In this paper, we discuss why we need such a tool through a case of an open source software-based project in a computer science course. We then propose an annotation model AMIESDev (***A***nnotation ***M***odel for ***I***nformation ***E***xchange Software Development) to assist in evaluation of students' contribution in a collaborative work. The model can also be used to monitor the progress in the work.

KEY WORDS
Model, evaluation, software project, annotation, collaboration, open source

1. Introduction

Many efforts are being made these days to benefit from the propositions of open source software. A primary objective of open source software is to make programming experience open and accessible to all who may be interested. As such, open source software provides a vast pool of resources to satisfy diverse needs of computer science instructors.

Open source software offers a great promise to the computer science education community [23]. Some computer science instructors use open source software, for example, Linux and Apache, in support of teaching [14]. Others focus on the open source philosophy and development paradigm, such as, cooperating over large distances to produce professional software and improving its quality through peer review and rapid evolution of source code [5]. In addition, attempt to get students involved in open source software development has been made so as to gain knowledge and skills which otherwise would not be possible in conventional project activities.

Academic projects based on open source software give students unique experience with understanding, using, and modifying existing programs to meet new requirements. However, evaluation of students' participation in such a collective work becomes a great challenge for the instructors. We need to measure the contribution of a team as well as each member of the team in order to evaluate students' performance in an adequate fashion. Obviously, an effective documentation tool is necessary.

In this paper, we discuss why we need such a tool through a case of an open source software-based project in a software design course. We then propose an annotation model AMIESDev (***A***nnotation ***M***odel for ***I***nformation ***E***xchange Software Development) to assist in evaluation of students' contribution in a collaborative work. The model can also be used to monitor the progress in the work.

This paper is organized as follows. Section 2 describes an open source software-based project we created for students in a software design class. Section 3 discusses our proposal of an annotation system, AMIESDev. Section 4 presents its features and architecture. Section 5 discusses using our AMIESDev system in documenting students' contribution in a software project. Finally, section 5 concludes the paper.

2. A Case of Open Source Software-Based Project

A primary goal for computer science instructors is to prepare students for the real challenges they will face in professional software development. In addition to formal lectures, projects are an indispensable component in computer science courses.

As technologies evolve, future developers are more likely to work from existing or partial systems in order to build new ones. Hence, it is important for students to gain experience with complex programs they cannot possibly redevelop from scratch [3] [12].

2.1. Background

We developed a software redesign project based on an open source product JUnit. Our objective is to have students learn by doing practical design techniques, specifically, how to use instances of design patterns as building blocks and compose them together to build an object-oriented application.

JUnit is a testing tool that encourages unit testing, a key component of the test-driven development in eXtreme Programming (XP) [9]. It is widely used in industry as well as in the computer science curriculum [22].

As a matter of fact, JUnit is more than a tool. It is worth studying in its own right. JUnit demonstrates a skillful and well-motivated use of design patterns by experts [2].

Moreover, JUnit is well-documented. Explanatory articles on its website provide a helpful insight into key design decisions, making it possible to reuse and modify the original design.

2.2. A Design Process

A significant part of a redesign project is the understanding and analysis of the original design. JUnit is an exemplary software product. It enables us to teach design techniques as preparing students for the project.

We use JUnit to show software design via composition of design patterns. Roughly, such a process begins with identifying concerns for the intended application. Doing so allows us to focus on individual concerns separately. Applicable design patterns are then selected to address individual concerns. An instance of an applicable pattern describes collaborating objects as well as their roles in the system. Finally, instances of selected patterns are composed into the intended application via key objects that share common roles.

We adopted many examples from [2] to show decisions encountered by the original designers and criteria used to choose among alternatives.

2.3. Project Requirements

In the redesign project, we asked students to explore alternative solutions to meet slightly different requirements. New requirements given in the project are as follows:

a. Organize test cases as a list, rather than a tree.
b. Use the factory method pattern, instead of reflection, to accommodate user-defined test classes.
c. Use the observer pattern to handle the one-to-many relationships between the test result and the test cases.

Note that the above changes leave certain portions in the original design intact. Students usually work on a design project from scratch. In this project, however, they were asked to redesign particular portions of a product. They needed to answer a set of new questions, such as which classes would be affected, which classes could be reused, and which classes should be modified. Students had to not only propose a solution but also show how the solution fits into the given context.

In addition, we asked students to work in teams. We also asked them to document their solutions in UML (Unified Modeling Language). Since implementation is not required in this project, it is crucial to show both the structural and behavioral aspects to demonstrate a working solution.

2.4. Issues to Address

Students' comments on their project experience were generally positive. Software redesign provides students with a unique learning opportunity. In addition to basic considerations, students must deal with additional constraints imposed by the existing product. Added complexity created a more realistic, motivating situation in which students learn design techniques.

On the other hand, however, evaluating the performance of each team as well as individual student becomes a great challenge for the instructors. A flexible documentation tool is certainly helpful. Specifically, we need the following capabilities from such a tool in order to do evaluation in an adequate way.

1. Identify elements from the original design.
2. Identify elements resulting from modification to the existing ones.
3. Identify elements that are new.
4. Identify individual contributions to the final design.

In addition, we found defects in every team's solution but there was little in common among different teams. We realized it would be beneficial for students to review each other's solution due to the complementary nature of those defects. As a result, students would be able to improve their solutions via peer review, just like what happens in actual open source software development. Hence, it is also desirable for the documentation tool to allow other teams to express comments.

3. A Model for Evaluating Contribution in a Collective Work

Annotation systems provide features that are helpful for the situation discussed above. A desirable documentation

tool for instructional purposes can be built around an annotation system.

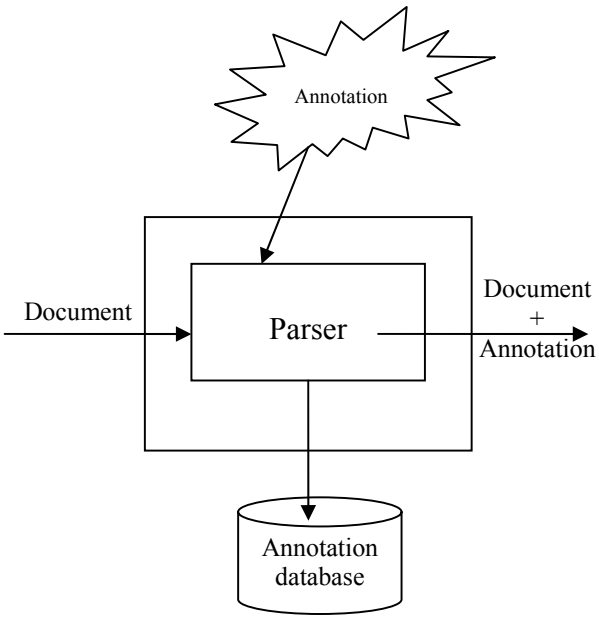

Figure 1: A generalized web annotation system

Annotation is defined as an act of interpreting or evaluating a document. Interpretation or evaluation is of a specific context and is expressed on the document Annotations normally take a different form and look compared to the original document. The different in look may be noticeable in form of character used, font, style, color or additional signs and images that do not form part of the original document. A document for annotation can include various entities like punctuations, words, images, artifacts terminologies, phrases, sentences, passages, collection of homogeneous documents, a collection of heterogeneous documents.

We consider a collection of documents as being homogeneous. Each document can be treated separately and each with related uniform properties. A document can be seen as a collection of heterogeneous documents in the sense that individual members that form this document differ in their properties and features.

In a generalized annotation system shown in *figure 1* as in the case in most annotation system, the concerned is the production of annotation based on the document sent to a parser (an annotation engine) and then the result [15]. The question of "why" is usually not addressed. The question of "how" is the general concern. For example, some programming languages like Java make available special routines, plug-in or API to enable the "how" of annotation [11].

The basic components of most annotation systems are (a) the document, (b) the annotation parser and (c) the resulting annotated document. The aspect of storage of annotation is not even applicable in some of these tools. In some cases, for example Amaya, provision is made for storage of annotation in either a local or a remote location [16].

A research group in France presented a tool called Dinosys [4] that was meant to apply annotation as a mean of sharing resources among students. The approach in Dinosys was more of explorative and there was no provision made for the evaluation of students' participation. Vasudevan and Palmer [20] proposed an annotation framework to be customizable to support variety of document management function, and to be non-intrusive to enable easy insertion into enterprise Intranets or the public Internet. This approach was good but not good enough for evaluative purpose.

We developed AMIESDev to assist in evaluation of students' contribution in a team project. AMIESDev can be viewed as an instruction-oriented version of the annotation system described in the work of Robert and David [18]. We can also use it to assist in monitoring progress in a project.

4. Architecture of AMIESDev

We in this section discuss key features and the architecture of AMIESDev. AMIESDev is based on AMIE (***A**nnotation **M**odel for **I**nformation **E**xchange*) which is a conjunction of annotation characteristic based on observations and needs in an information retrieval system. The basic components of AMIE are the user (a decision maker), document and time in an annotation system conceived to for decision support [19]. Decisions are made based on information aggregated from a set of document and time.

AMIESDev consists of five parts (a) a user is a member in a collaborative workspace (b) Program repository (c) annotation database (d) contribution (e) time.

4.1. User

The user is identified with the following parameters:

- Usercode (which can be his official school code)
- SurName
- FirstName
- PostalAddress
- EmailAddress
- City
- Usergroup (programming or academic group he belongs)

4.2. Program Repository

Program repository is a database of program. It may also be a link to program and not the programs themselves. Program names must be unique. It that can be identified with the following parameters:

- Programcode (Program.Attempt.Semester.Version.Status)

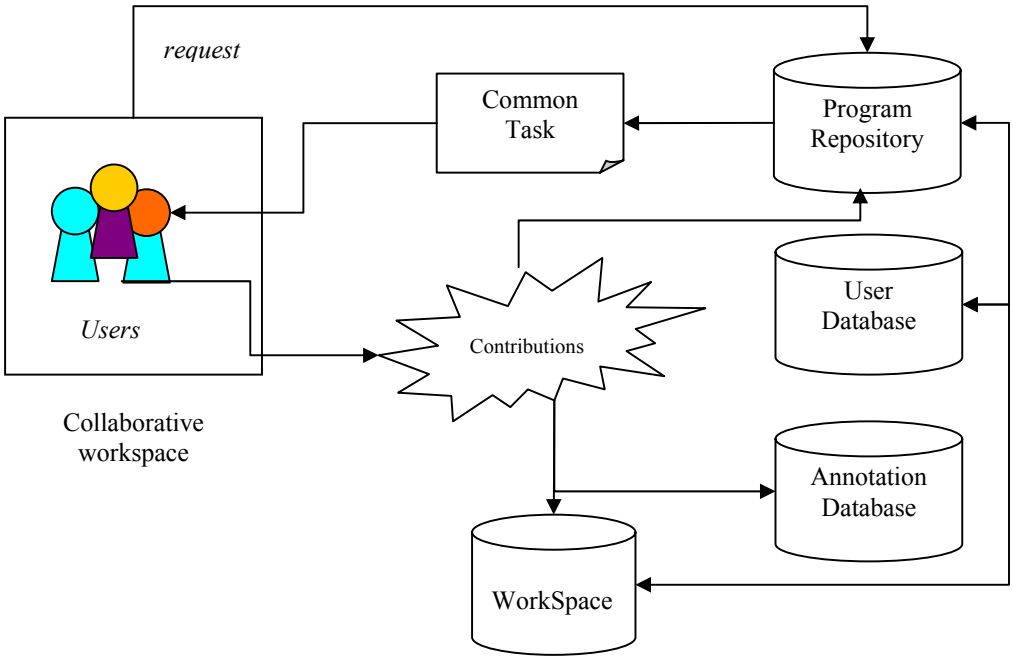

Figure 2: Architecture of AMIESDev

  Example JUnit.5.F.2006.005.InProgress
  JUnit development, Fifth Attempt, Fall Semester, 2006, version 005, Closed/New/ InProgress
- Type (Script, program, class, applet, etc…)
- Domain (operating system, application, wordprocessor, internet, security, etc…)
- Requirements (Library, OCX, DLL, Includes, etc...)

4.3. Annotation database

- Date of project initiaton
- AsociatedCreator (same as usercode in 4.1)
- ProgramReferenced (Same as Programcode in 4.2)
- AnnotationCode
- DateandTime of annotation creation
- Objective of annotation (Personal, Assignment, etc…)
- AnnotationType (Delete, Update, Addition)
- Annotation (Suggested proposition: This is the full program or function, call, applet, script, method suggested by user)
- Associated file (URL or path)
- Location (Node to place proposition (Module name, Calls, Function, Subrouting etc… which consist of module name, line number in module)
- CallParameters (Calls to the module)
- Returns (Output from the module)

4.4. WorkSpace

- SessionCode
- AnnotationCode (Same as in 4.3)
- UserCode (Same as in 4.1)
- OlderProgramCode (Same as in 4.2)
- NewerProgramCode (Modification to OlderProgramCode)
- DateandTime

A user in a collaborative workspace will normally have identification (usercode). He uses the identification code to request for a particular program from the software repository. The user is granted that program as a common task if he is a member of the collaboration. He is free to edit, modify the program. Instead of sending the modified program back as a replacement in the repository, we propose that, what he added or deleted is stored as a parameter of his contribution. Realize that a newer version of the modified program is created with a newer version number while the older version of the modified program is kept in the repository.

The newer program is returned to the repository with newer version code. User's identity is stored in the workspace with date and time and the code of the program he worked on (the older and the newer codes). A code is created for the proposition he made (in form of annotation). Annotation database receives the proposition as "annotation" and the parameters linked to the proposition.

5. Application to open source initiatives

With this model, it is possible to know the frequency of participation of students/users (from WorkSpace database). It is possible for student to decide exactly what version of program he will want to work with since newer programs are always created from existing ones. We can evaluate the methodology of each student/user from the annotation database. This is because a contribution of each user is stored separately independent of the global program. It is possible to monitor the growth of the program (from the program repository). We can also study the growth of each user from a set of his contributions in the annotation database.

From the WorkSpace database, we can see the period that is most favorable for particular user of the entire participants. A contribution that is judged "unnecessary" can be identified and eliminated because there is no overwrite of program. It is easy to understand the specific issue(s) learnt by user by comparing his initial "annotation" with the latest. This can be a way of rewarding students who made substantial progress in the joint project. It can be seen also if a user is static in his learning activity.

If this model is used for successive years, comparison can be made across years, across groups, across class and across period of time. We can even have several programs with different perspectives (eg. JUnit, C++, Fortran, Java, VB, etc…) in the system, and we can measure the participation across programs.

6. Conclusion and Perspectives

This study briefly describes the effort of a collaborative work in a university environment using JUnit. A model AMIESDev was conceived based on the parameters of the user, the program, added or modified text (annotation) which can be included in the pool of work in a collaborative workspace. We have also demonstrated how this model can be used to evaluate the participation of student/user in a collaborative program development. We did emphasized that performance within group can be evaluated as well as performance across years. This model can be well suited for other public domain open source initiatives like in LINUX. What is now remaining is its practical implementation.

Bibliographies

[1.] Apple Development Connection, 2006, Open Source Overview, http://developer.apple.com/opensource/overview.html

[2.] Beck Kent and Gamma Erich, 2002, JUnit A Cook's Tour, http://junit.sourceforge.net/doc/cookstour/cookstour.htm.

[3.] Cohen, J., "Updating Computer Science Education", *CACM*, Vol. 48, No. 6, June 2005, PP.29-31.

[4.] Desmontils E. , Jacquin C. and Simon L., Dinosys : un outil d'annotation pour l'enseignement à distance sur le Web, Colloque "Miage et e-mi@ge", Marrakech, Maroc, mars 2004.

[5.] Faber Brenton, "Educational Models and Open Source: Resisting the Proprietary University", Proceedings of SIGDOC '02, Oct. 20-23, 2002, Toronto, Canada, pp 31-38.

[6.] Gilbert David, 2002, A user guide for GnuCash, Simba Management Limited http://www.object-refinery.com/gnucash/gnucash-US.pdf (27/06/2006)

[7.] Gonzalez-Barahoma Jesus M., 2000, Free Software / Open Source: Information Society opportunities for Europe? Working group on Libre Software, version 1.2, http://eu.connecta.it

[8.] Gonzalez Guadamuz Andrés, 2005, Legal challenges to open source licences,

[9.] JUnit Web Page, http://junit.org

[10.] Lemyre Pierre-Paul and Willemant Richard, 2006, The legal issues surrounding free and open source software: Challenges and solutions for the government of Québec, http://www.cirano.qc.ca/pdf/publication/2006RP-04.pdf

[11.] Martini F., 2005, Les Annotations de Java 5.0, http://adiguba.ftp-developpez.com/tutoriels/java/tiger/annotations/annotations.pdf 21/07/2006)

[12.] Meyer Bertrand, "Software Engineering in the Academy", IEEE Computer, May 2001, pp 28-35.

[13.] Michelle Levesque, 2004, Fundamental issues with open source software dev elopement, Peer-Reviewed Journal on the Internet, http://www.firstmonday.org/issues/issue9_4/levesque (13/07/2006)

[14.] Nelson Daniel and Ng Yau Man, "Teaching Computer Networking Using Open Source Software", Proceedings of ITiCSE '00, July, 2000, Helsinki, Finland, pp 13 – 16.

[15.] Ovsiannikov Illia, Arbib A. Micheal, Mcneil H. Thomas, 1999, Annotation technology, International Journal of Human-Computer Studies, 50, 329 - 362

[16.] Quin Vincent and Vatton Irène, 1997, An Introduction to Amaya, http://www.w3.org/TR/NOTE-amaya (20/07/2006)

[17.] Ragib Hasan, 1999, The History of Linux, http://www.linux.co.uk/Pages/aboutlinux

[18.] Robert Charles and David Amos, 2006, Annotation and its application to information research in economic intelligence, Advances in Knowledge Organization (10), pages 35-40

[19.] Sidhom, S., Robert, C., David A., 2005, De l'information primaire a l'information a valeur ajoutée dans le contexte numerique. Revue maghrébine de documentation et d'information, vol 1, pages 95-118, Tunis, 2005, Tunisie

[20.] Vasudevan Venu and Palmer Mark, 1999, On Web Annotations: Promises and Pitfalls of Current Web Infrastructure, Proceedings of *the 32nd Hawaii International Conference on System Sciences*

[21.] Wheeler A. David, 2005, Why open source software / Free software (OSS/FS, FLOSS, or FOSS)? Look at the numbers, http://www.dwheeler.com/oss_fs_why.html (27/06/2006)

[22.] Wick Michael, Stevenson Daniel, and Wagner Paul, "Using Testing and JUnit Across The Curriculum", Proceedings of ACM SIGCSE, Feb. 23-27, 2005, St. Louis, Missouri, pp 236-239

[23.] Wolf, Marty, J., Bowyer, K., Gotterbarn, D., and Miller, K. (2002). "Open source software: intellectual challenges to the status quo", in SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education, pp. 317—318, ACM Press, New York, NY, USA.