Association for Information Systems
AIS Electronic Library (AISeL)

MCIS 2010 Proceedings

Mediterranean Conference on Information Systems (MCIS)

9-2010

CHANGING STUDENTS PERCEPTION REGARDING SOFTWARE DOCUMENTATION

Aharon Yadin
Academic College of Emek Yezreel, aharony@yvc.ac.il

Ilana Lavy
Academic College of Emek Yezreel, ilanal@yvc.ac.il

CHANGING STUDENTS PERCEPTION REGARDING SOFTWARE DOCUMENTATION

Aharon Yadin, aharony@yvc.ac.il

Ilana Lavy, ilanal@yvc.ac.il

The Max Stern Academic College of Emek Yezreel

Abstract

Being aware to the important role of proper software documentation on one hand and being acquainted with the students' views regarding this issue on the other, we decided to examine the effects of facing the students with bad documentation or the lack of it on their views, as represented by a software project they have to design, develop and test.. This research was performed within a software engineering workshop for Computer Science students. For addressing the soft skills issues required by the industry, the course was delivered as a workshop with various (inter and intra) team based activities.

The objective of outlining the importance of software maintainability issues was achieved through a hybrid team-based role play. The workshop consists of three assignments, following a typical software design and development process, in which each team had to continue the work performed by another team, thus creating a dependency between the team members as well as between the teams as might happen during real life maintenance. The main research study objective was to examine the effect of employing this kind of a hybrid team-based role-play and peer-review on the students' learning process regarding product documentation for future maintainability. Data referring to the students' perceptions is presented and analyzed in addition to student reflections on the workshop which demonstrate their expanded understanding of documenting the design and application process.

Keywords: Team-based role play, software engineering, peer review, documentation

INTRODUCTION

The term software engineering appeared first at a NATO conference in 1968 (Naur et al.,1968) and it was intended to ignite discussions on the process of developing correct, testable, and understandable computer programs. At that time, the "software crisis", that was partially caused by the rapid developments in computer technologies combined with more, complex user requirements, and the lack of "engineering" methodologies for software development (Veldwijk et al., 1992). Since then the process of software engineering has matured and is accepted as a proven learning discipline. The Software Engineering course is an important part of the Computer Science (CS) and Information Systems (IS) curricula, however many students regard it as less applicable in their future careers (Burge, 2007). Information systems and software based systems in general, require follow-up maintenance due to the existence of potential bugs that will have to be corrected, the high likelihood of functional enhancements to be introduced to the programs. For lowering the costs associated with these ever increasing needs for software maintenance adoption of proper software engineering methodologies is required. Thus software maintainability plays a critical function in the software engineering process, not unlike the role of software development itself.

Cognizant of the students' difficulties regarding non-technical knowledge such as critical thinking, interpersonal and team based skills, the Software Engineering workshop structure employs many inter-team and intra-team activities. Furthermore, to raise the students' awareness to the importance of documentation and the role it plays in maintainability, the workshop employs an incremental life-cycle involving each team in three activities: design (including documentation), development, and testing. However, unlike the ordinary software development life-cycle, in which each team performs the three activities for the same project the workshop structure employs a team-based role play. By team-based role play we mean that the design, development and testing is swapped among the teams. Initially all the teams are given the same project, and each team prepares his own design. The second assignment consists of developing the system according to the design specifications, but each team develops a system that was designed by a different team and not the system it has designed. The third assignment consists of defining the test specifications and testing the system, however, once again, each team tests a system designed by one team and developed by another. When proceeding to the next assignment, the team is required to ignore their prior knowledge or ideas and to concentrate only on the system as it has been designed (or developed) by another team of their peers.

The main research study objective was to examine the effect of employing this kind of a team-based peer-review on the students' learning process in a software engineering workshop with special emphasis on their perception regarding documentation. This paper describes the workshop structure and the quantitative and qualitative results obtained from employing it.

1 THEORETICAL BACKGROUND

Software development is a collaborative task that employs teams of developers working together (Cheng et al., 2003). To enhance software readability and maintainability, software engineering practitioners have been striving to improve existing tools and methodologies. Among these various practices, documentation -- used to describe the required software, its structure, logic and performance -- is high on the list (de Souza et al., 2007; Das et al., 2007). Without the proper documentation, the future maintainer will find it extremely difficult to understand the system or its processes. Poor and missing documentation is a major contributor to software quality degradation and aging, compounding an increase in maintenance costs (Kajko-Mattsson, 2008). In spite of this, many students fail to recognize the importance of documentation on maintainability (3).

Software engineering is an integration of many practices, methodologies and tools. One of the initial practices is software documentation. However, even at present many systems are still developed and released without proper documentation (Daich, 2002). In response, several methodologies have been developed to address the unsolved problem of the documentation lag (Clements, 2005). The Agile

Manifesto, for example, puts greater emphasis on the developed product while ignoring detailed documentation. This methodology welcomes change and stresses fast delivery of useful software, based on the close collaboration between developers and customers.

Software documentation is a general term that refers to two types of documentation: (1) documenting the user requirements that provide the basis for designing the system to be developed, and (2) documenting the software to be developed, or that was developed, for aiding development or future maintenance activities. These maintenance activities, according to many studies, are the most expensive part in the software development life-cycle (Seacord et al.,, 2003). The Agile methodology is helpful in reducing the amount of work needed during the first documentation process (requirement elicitation and project development). However, for future maintenance of the developed software, proper documentation is still required. For that reason, the documentation required to the Agile methodology is done at the end of the project and remains inadequate for maintenance purposes (Brolund et al., 2006)). For years educators and practitioners have stressed the importance of documentation (during the design phase or after project completion), however many projects are still released without proper maintenance documentation.

1.1 Software Maintenance

A common definition for maintenance is that it is performed after product delivery. Software maintenance, as well, refers to the activities carried out after the development's completion. However, software maintenance is very different from "ordinary" equipment maintenance. While in other engineering disciplines maintenance in intended to fix a problem (Canfora et al., 2000) and keep the equipment running so it will continue to provide the original functionality, software maintenance, in many cases is required to enhance the functionality based on the ever changing requirements due to the operational environment, the competition, and the business climate. Meeting these new and changing requirements is unique and basic software characteristic, as defined in Lehman's laws of software evolution (Lehman, 1980; Lehman, 1984). Furthermore, software is constantly being modified to utilize new hardware equipment and for integration into new environments.

Introduction of the software development life-cycle has led several researchers to consider activities related to software maintenance, which are initialized while the software is being developed and not only after delivery. Additionally, some researchers emphasize that starting the maintenance activities after completing development leads to an unnecessarily more complex and costly task (Schneidewind, 1987; Osborne et al., 1990). Others define software maintenance as a mix of activities, some performed after delivery and some performed during development. The pre-delivery activities include the necessary planning for the post-delivery activities (Pigoski, 1997).

1.2 Students' Perception regarding maintenance

There has been a great deal of improvement in software development over the last decade. Many new techniques, methodologies, languages and tools have been created to advance the development processes. Software maintenance, however, lags behind mainly due to its reactive nature. Introducing systemic approaches to software maintenance is inherently problematic (Dias et al., 2003). The required software maintenance (error corrections or introductions of new features) cannot be postponed or circumvented. By nature software maintenance is a disorganized process which deteriorates the software's architecture (Lehman's second law of software evolution (Lehman, 1980)). This deterioration is due in part to missing knowledge which is required for maintenance. In addition, any changes introduced deteriorate the system architecture further, making future maintenance even more difficult. The lack of correct and updated documentation is one of the main causes for this missing knowledge.

During their first and second years of study, students become acquainted with these facts, however in spite of the lecturers' efforts; software documentation continues to be insufficient. More troubling is the fact that students do not assimilate the need for, and importance of, proper documentation (Burge,

2007). Many students consider the development stage as the most important activity in the software development life cycle, totally ignoring the fact that successful software will have to be maintained for a long time.

1.3 Peer Review in Higher Education

Peer review is a form of external evaluation carried out by professional colleagues (Yadin et al., 2008). Peers can be experts in the field but can also be classmates who poses the same level of knowledge and assess the work of fellow students. Peer review is a widely practiced form of certifying quality in higher education (Herndon, 2006), and has been described as a formative evaluation process in which participants work collaboratively to strengthen a product (Keig et al., 1994). Peer review is generally said to encourage critical examination, promote the exchange of ideas, reduce non-academic interference, guide academic discourse, and reinforce academic values (Berkencotter, 1995). Peer review assumes the existence of norms by which a peer's work may be judged. Through critical examination, norms are used to compare a peer's work to accepted practices. If a peer's work deviates significantly from accepted norms, then an attempt to correct it will likely occur. Being aware of the advantages of peer review, it has been incorporated as an integral part of the workshop. The inter-team and intra-team peer review was used to enhance the students' learning abilities and to enforce critical thinking. However, In addition to the common peer review, the workshop requires the students not only to evaluate and assess their peers work, but also build on it. The students' success in performing their assignments depends on their ability to understand the work or their peers. This elevates the assessment process to a new and more important level.

2 THE STUDY

In what follows we discuss the study performed while addressing the participating students' the workshop's structure including the assignments and the grading scheme.

2.1 About The Study Participants

The workshop is a mandatory course taken during the second year of study. A total of twenty-six college students participated in the present study. In the workshop the students were divided into seven teams (five teams of four students and two teams of three students).. At this stage the students have already learned software modeling, UML usage, etc. In addition to the standard topics of the software engineering course, one of the workshop's important objectives is to prepare the students for their Final Project and the real world challenges they will face.

2.2 The course

The systems engineering workshop's general objectives are to introduce software development life cycle concepts to the students while enhancing their understanding of documentation and product maintainability. Since software is considered one of the most complex systems produced by humans (Lenic et al., 2004), students have to adopt proper working procedures for lowering the development risks and the high maintenance barriers. Documenting their ideas and thoughts during the design phase is crucial for future understanding of the software to be developed.

Other objectives relate to (1) practical understanding of the software development stages required for development of a modern Information System; (2) implementing these stages in a small project; (3) understanding the problems associated and caused by working in teams, and (4) developing the required "soft skills" (critical thinking, team work, interpersonal relationships, etc). For that reason, the workshop augments knowledge and understanding gained in current and previous courses, and is practical, "hands-on," and team- based.

All seven teams received and worked on an identical project. The project was a general description of a required system that was to be developed (an Internet based electronic auctions, or e-bidding system). As part of the first assignment, the students had to study the existing systems, address and assess various alternatives, and suggest ways (and a software based system) of providing an agreed upon functionality. The workshop structure followed the software development life-cycle and was based on three incremental assignments.

Each assignment required personal and individual work followed by team activities (in person or by using various collaborative tools).The students had four weeks for each assignment throughout the process the students consulted their instructor (via email, the workshop web site, and personal meetings) on various issues related to their assignment. In order to reinforce the importance of documentation and maintainability, the teams were engaged in role-based development in which the teams shared all responsibility for their success. While a specific project was designed by one team, developed by another team and tested by a third team, in the end each team had to work not only on the three stages of the assignment but on each one of the three design solutions (Figure 1). This way, each team was involved in developing a system designed by another team while trying to understand some of the undocumented intentions expressed in the design. This forced them to seek help from the designing team. On the other hand, this developing team had to help another team that was trying to develop the system based on their design document. The interdependence of these stages was stressed and made apparent to all teams. This workshop structure was designed to enhance the students' understanding regarding the importance of documentation through their own experience.

Figure 1 depicts the workshop's structure. The long horizontal rectangles represent the seven versions of the same project, while the three vertical columns represent the assignments (A1, A2 and A3). As can be seen, each project consists of the three assignments performed by three different teams. Each team, on the other hand, worked on all three assignments, each one belonging to a different project.

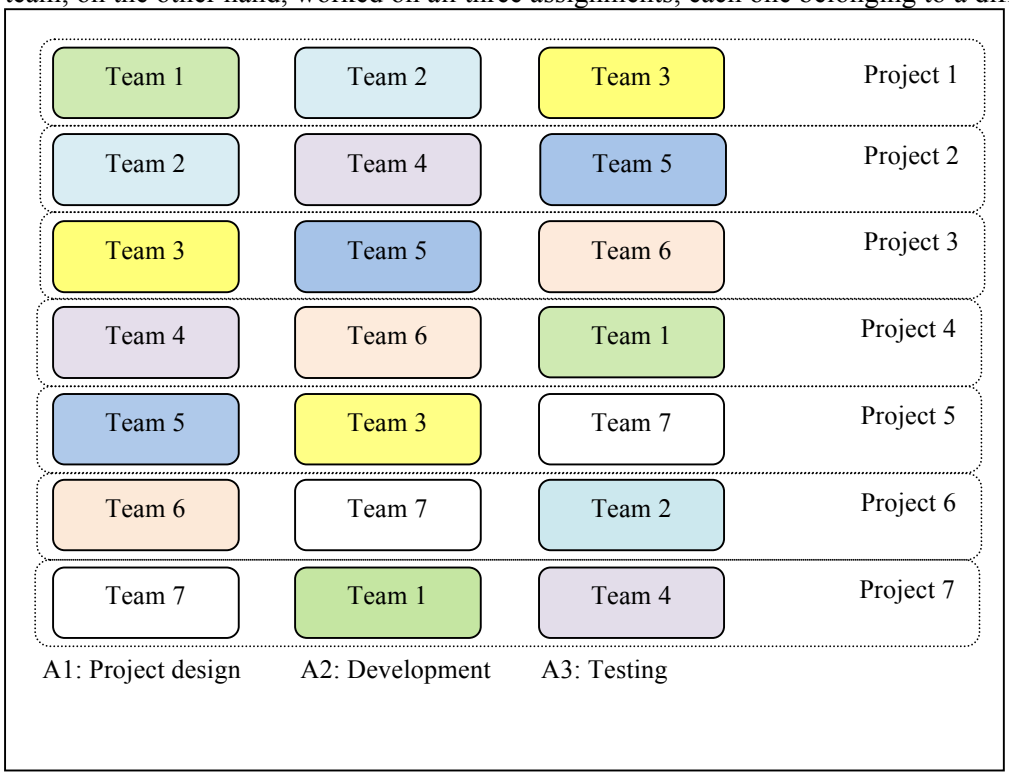

Figure 1: The Workshop's Structure

The workshop requirements included two types of deliverables: (1) team assignments, and (2) a personal assignment.

2.3 Team Assignments

The software development life-cycle activities were divided into three team based assignments: (1) project definition and design; (2) project development, and (3) project testing.

2.3.1 Project Definition and design

The first assignment started with a very brief description of the project, the functionality and the required development activities. The students studied available e-bidding systems, documenting their functionality, and used them as a basis for the system they were required to develop. Since such a large project cannot be completed during the semester, the students had to identify at least five different users to be supported by the system and for each user a set of Use-Cases had to be defined. In addition to the Sequence Diagrams supporting these Use-Cases, the students had to define the non-functional requirements associated with these Use-Cases. The system analysis phase (which is part of this assignment) included a high level design (System architecture and the Class Diagram) as well as a detailed design (Activity Diagram followed by a Program Design Language definition for the described functionality). All these activities required a great deal of individual work as well as collaborative work in which each student assessed and approved the work performed by other team members.

2.3.2 Project Development

The second assignment consisted of the development of the system according to the Project Definition and design document (the first assignment). However, instead of developing the system according to their own design, each team had to develop the system as it was defined by another team. The developing team had to carefully follow the document they received, ignoring all their prior knowledge or ideas they may have expressed in their first assignment. Small code modifications were permitted, provided that the definition in the document they received was erroneous and could not be implemented. After completing the development, each team had to compile a 'difference' document, outlining the changes between the implementation and the document as received, with special emphasis placed on the reason behind these changes. At this stage stress is not placed on enhancing the product to be developed, but rather on developing it according to the exact specifications outlined in the definition document. An additional document which was part of this assignment was a short evaluation of the first assignment's quality as it was reflected in the implementation. The last document to be submitted as part of this assignment was a Unit Test Plan for each of the methods developed.

2.3.3 Project Testing

The third assignment consists mainly of the testing phase. The students have to implement the Unit Test Plan as was designed by the previous team. Due to time constraints the workshop addresses only part of the required project development, so for testing the software pieces developed by the previous team, the testing team had to include additional developments (a test generator and a stub). These additional developments were required for building the testing infrastructure for the developed software pieces. As part of this assignment the team is required to correct mistakes that were discovered during the testing. The corrected code has to be tested once again. This process is repeated until everything runs according to the specifications, as outlined in the project definition document (the first assignment of this project). This third assignment also includes the testing report that summarizes the problems discovered and their corrections. In addition, this assignment includes a system test plan with at least ten detailed test cases. This plan is for the Quality Assurance staff, so it has to be detailed and based on the system functionality as derived from the project definition document (first assignment). The last part of this assignment is a quality test plan that concentrates on

the non-functional attributes of the system, with a special emphasis on the metrics to be used or defined.

2.4 Personal Assignment

The personal assignment is mainly an activity summary report, in which each student describes (1) the work done during every stage of the project; (2) his/her part in these activities; (3) the problems they (as a team) encountered during the project and (4) the problems he/she encountered personally. There is also a short reflection on the workshop, as well as a one sentence summary about the workshop's results. The last part reflects on the work distribution among the team members (100 points that the student divides between the other team members to express their relative contribution toward each of the three assignments).

2.5 The Workshop Grading Scheme

Since one of the important workshop goals is to strengthen team work, most of the grades are based on the team's activities. Each of the first two assignments makes up 33% of the grade, while the third, which is simpler, comprises 24%. The personal report, including the short reflection, contributes an additional 10%.

The grading scheme took into consideration the work distribution as was described by each team member. Five points (out of the 90 points allocated for team activities) were used as floating points among the team members, based on their average contribution to the team's success.

3 LEARNING PROCESS EVALUATION METHODOLOGY

The evaluation method included a comparison between two questionnaires. The first questionnaire was part of a survey conducted during the workshop's first lecture, in which students were asked to rate their perception regarding the relative importance of the three project phases expressed by the planned assignments. A similar survey was conducted during the last lecture producing the second questionnaire. Since the end of semester questionnaire was identical to the questionnaire used in the first lecture, its intention was to measure the workshop's influence regarding the perceived importance of the three phases and especially the importance of documentation and testing on the software engineering activities. In addition, the evaluation process analyzed the student's reflections on their workshop experiences.

For implementing a successful inter-team role play, the workshop was highly structured. In addition, pre-defined templates were used for all the team based assignments. However, in contrast to these pre-defined templates the personal reports were composed of free style answers. The only data provided were the points to be addressed in these reflections. This open format encouraged students to concentrate on the issues s/he felt were important and offered a better understanding of the students' achievements during the workshop.

4 RESULTS AND DISCUSSION

In what follows we present data and discuss the effect of the workshop's structure on the students' perceptions regarding the importance of documentation on the project's success, as well as their reflections regarding the benefits of the workshop.

4.1 The Assignment's Relative Importance

The first questionnaire results are outlined by Figure 2. It is no surprise that CS students regarded development as the most important activity (70% of the project). Testing was perceived to be of

secondary importance (16%) and documenting the design phase was perceived to be the least important (14%) in the design and development of software.

The students explained the relatively low importance of documentation by citing the fact that the methodology and the tools used (UML – Unified Modeling Language as well as the code itself) provide all the necessary documentation. The results obtained in this survey were no different when compared to the results from the previous year. These consistent results were the trigger for the workshop structure and one of its objectives was to convince students, through their own practical experience about the importance of documentation.

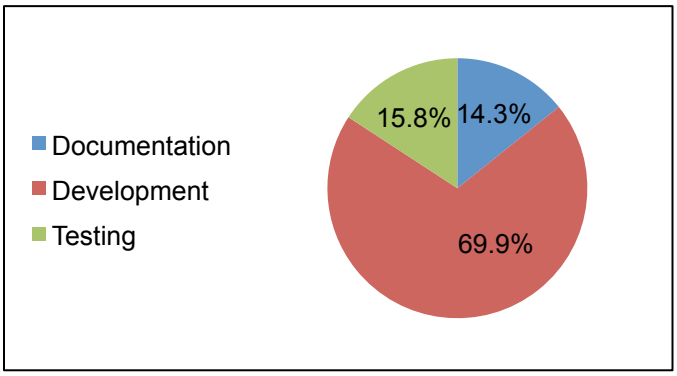

Figure 2: Relative Assignment Importance (1st Lecture)

As was demonstrated by the first questionnaire (Figure 2), most students view development as the most important activity of a project (70%). However, the second questionnaire revealed that the students began to realize the importance of the subsequent components and the role they play in determining the project's success. Therefore, by the semester's end, development's perceived importance was reduced by 31% (to 48% of the project), while the relative importance of the documentation assignment increased by 59% (from 14% in the first questionnaire to 23% in the second questionnaire). Testing's perceived importance also increased -- by 83% (from 16% in first questionnaire to 29% in the second).

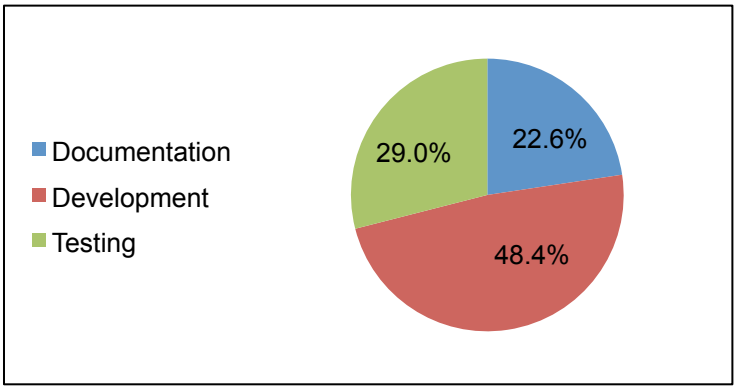

Figure 3: Relative Assignment Importance (Last Lecture)

There are many factors affecting the relative importance of the various life cycle stages and the amount of time required for each one. These factors, for example may include customer requirements, project type, the software development life cycle methodology, the programming languages and CASE tools, etc. In most cases the development stage requires less than 30% of the project estimated time. Glass (2003) uses 20% for requirements elicitation, 20% for design, 20% for coding and 40% for testing. The requirements elicitation is not part of this workshop since the project and its general requirements were predefined and the students had to study available solutions and decide which parts to design and implement. At the end of the semester, the students still regard coding (development) as the most important component, but it is significantly (31%) less than its importance at the beginning of

the semester. The end of semester percentages are closer to the numbers used by researchers and practicing software engineers.

The change in the students' perception regarding the relative importance of the various project components is directly linked to the workshop's structure. There were many instances during the second stage of the workshop, in which the students were trying to drop the design specifications they received from the previous team, claiming these specifications will not produce a viable solution and the project cannot be developed. In all cases it proved to be wrong. The solutions described in these design specifications provided a workable solution, however they were not properly documented, which hampered the students understanding. After discussing the design specifications with the responsible team, the project was developed as intended, with some minor modifications. This misunderstanding was repeated on the third stage, in which students had to design and execute the system testing. However, for designing the test environment and the test scenarios a full project understanding was required. In addition to the extra work needed due to the missing documentation it also changed the students' perception regarding the importance of the non-development activities.

The significant increase in the perceived testing importance (83%) can be explained by the fact that during the third stage the student had to design and develop the stub and the scenarios for the system to be checked. In all cases where the system did not function according to the specifications, the testing team had to correct the code and run it once again. This means that the testing team had to be familiar with the design specifications as well as with the developed code. The testing students acted as the "gate-keepers" making sure that only the fully functional system is released. Performing this task properly, in the workshop, requires some additional analytical skills for finding and correcting various bugs which may have been introduced during the development or the design stages. Unlike the real world situation, where in case of problems, Quality Assurance people usually return the system to the developers, here the testing team had to fix it by themselves, which led to their higher appreciation of the testing task and its elevated importance in the development process.

4.2 The Student's Perspective

Analysis of the students' reflections revealed emphasis of three main issues: (1) the importance of documentation; (2) team-based activities and (3) contribution to future vocation.

4.2.1 The Importance of Documenting the Project

Improving the students' understanding regarding documentation and the role it plays in the project and its future maintainability, was addressed by many of the reflections. For example:

"I understood (unfortunately through bad experience) the importance of a development project's documentation."

"It was only during the workshop that I began to grasp the importance of understanding and documenting the requirements."

From the above students' excerpts we can conclude that they developed a sense of appreciation for documentation, mostly arising from the need to spend many more of their own resources when it was missing. Furthermore, without proper documentation, the project may not be successful and might not deliver the expected outcome. The fact that they realized, for example, that undocumented specifications are misleading is consistent with Williams (2001) stressing that students no longer view the teaching staff as their sole conduit of technical information.

4.2.2 Team-Based Activities and Implications

Students pointed out several advantages regarding their experience of working in teams, as well as what was required of them. Here are some common reflections:

"Working in a team provided me with many new views and possibilities for solving the problem."

"The most important lesson I learned during the workshop was to accept my friends' criticism and provide constructive feedback."

"The success and failure of the project depended mainly on the team members' activities and not on any single member."

From these reflections we learn that in general students found the teamwork method helpful in developing their critical thinking (receiving and providing constructive feedback) and in improving their ability to cooperate. However, in their reflections students also pointed out the shortcomings they experienced in team-based activities. For example:

"Team work can be a blessing, but sometimes it can also be a curse…"

4.2.3 The Workshop's Contribution to Future Vocation

Here are some student reflections regarding the contribution of the workshop's assignments to their future employment.

"So far we learned that the most important stage in the project is the development. Here I understood that the process is equally important."

We conclude that the students found the detailed documentation very helpful. Furthermore, the understanding gained by working in teams helped them think as developers and enhanced the process of reaching the problem solution.

5 CONCLUDING REMARKS

From the students' reflections and the results received regarding the differences in their perception as reflected in the two questionnaires, it can be concluded that the workshop raised the students' levels of understanding (Biggs, 1996), and as a result helped them cope successfully with the given workshop assignments. The role-based development, in which the students had to assume responsibility for activities partially performed by others, exposed them to ideas which were different from the ones they had decided to use in their own solutions. Especially, this workshop structure effected the students' appreciation of documentation and the role it played in their own success. This exposure, in many cases, made them rethink their task and prompted them to look for better, more efficient solutions. The collaborative team work exposed each team member to various ideas expressed by his/her peers and as a result caused additional thinking about available solution alternatives.

REFERENCES

Berkencotter, K. (1995) The power and perils of peer review. Rhetoric Review, 13 (2), 245-248

Biggs, J. (1996) Enhancing teaching through constructive alignment. Higher Education, 32, 347- 364

Brolund, D. and Ohlrogge (2006), Streamlining the Agile Documentation Demonstration for the XP 2006 Conference. XP Vol. 4044 of Lecture Notes in Computer Science, 215-216, Springer.

Burge, J (2007) Exploiting Multiplicity to Teach Reliability and Maintainability in a Capstone Project. 20th Conference on Software Engineering Education & Training (CSEET'07).

Canfora, G. and Cimitile, A (2000) Software Maintenance. http://www.compaid.com/caiInternet/ezine/maintenance-canfora.pdf. Accessed June 2009

Cheng, L.T., Hupper, S., Ross, S. and Patterson, J (2003) Jazzing up Eclipse with Collaborative Tools. Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange, Anaheim.

Clements, P (2005) Comparing the SEI's Views and Beyond Approach for Documenting Software Architectures with ANSI-IEEE 1471-2000. Technical Note .CMU/SEI-2005-TN-017, http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn017.pdf. Accessed June 2009

Daich, G. C (2002) Document diseases and Software Malpractice. http://www.sstc.online.org/Proceedings/2003/PDFFiles/p961pap.pdf Accessed June 2009

Das, S. Lutters, W. G., and Seaman, C. B (2007). Understanding documentation value in software maintenance. Proceedings of the 2007 Symposium on Computer human interaction for the management of information technology, March 30-31, 2007, Cambridge, Massachusetts

de Souza, S. C., Anquetil, N., and de Oliveira, K. M (2007) Which documentation for software maintenance? Journal of the Brazilian Computer Society, 13 (2), 31-44.

Dias, M. B. G. Anquetil, N. and de Oliveira, K. M., (2003) Organizing the Knowledge Used in Software Maintenance. Journal of Universal Computer Science 9 (7), 641-658.

Glass, R. (2003) Facts and Fallacies of Software Engineering, New York: Addison-Wesley.

Herndon (2006) Peer Review and Organizational Learning: Improving the Assessment of Student Learning. Research & Practice in Assessment 1 (1), 1-7.

Kajko-Mattsson, M (2008) Problems in agile trenches. Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, October 09-10, 2008, Kaiserslautern, Germany.

Keig, L. and Waggoner, M. D. (1994). Collaborative peer review: role of faculty in improving college teaching. Washington D.C.: ERIC-ASHE.

Lehman, M. M. (1980) Lifecycles and the Laws of Software Evolution. Proceedings of the IEEE, Special Issue on Software Engineering, 19:1060-1076.

Lehman, M. M (1984). Program Evolution. Journal of Information Processing Management, 19 (1):19-36.

Lenic, M., Zorman, M., Povalej, P. and Kokol, P. (2004) Alternative measurement of software artifacts. ICCC 2004 Second IEEE International Conference on Computational Cybernetics, 2004, 231 – 235

Naur, P. and Randell, B (1968) Software engineering: Report of a conference sponsored by the NATO Science Committee. Garmisch, Germany: Scientific Affairs Division, NATO. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF. Accessed on June 2009

Osborne, W. M., and Chikofsky, E. J., (1990) Fitting Pieces to the Maintenance Puzzle. IEEE Software, 7 (1):11-12.

Pigoski, T. M., (1997) Practical Software Maintenance – Best Practices for Managing Your Software Investment. New York: Wiley & Sons.

Schneidewind,N. F (1987) The State of Software Maintenance. IEEE Transactions on Software Engineering, SE,13 (3): 303-310.

Seacord, R. C., Plakosh, D. Lewis, G, A (2003), Modernizing Legacy Systems – Software technologies, engineering processes, and business practices. New York, NY. Addison-Wesley.

Veldwijk, R. J., Boogaard, M, and Spoor, E. R. K (1992) Assessing the Software Crisis – Why Information Systems Are Beyond Control. Vrije Universiteit, The Netherlands. ftp://zappa.ubvu.vu.nl/19920006.pdf Accessed on June 2009.

Williams, L. (2001) In support of student pair-programming. In the Proceedings of the 32nd SIGCSE technical symposium on Computer Science Education. Charlotte, North Carolina.

Yadin, A. and Lavy,I. (2008) Integrated Formative Assessment as a vehicle towards meaningful learning in Systems Analysis and Design workshop. ICIS 2008 SIG-ED, Paris, France.