Teaching Computer Programing as Knowledge Transfer: Some Impacts on Software Engineering Productivity

Orlando López-Cruz[1], Alejandro León Mora[1], Mauricio Sandoval-Parra[1], Diana Lizeth Espejo-Gavilán[1],

[1] Universidad El Bosque, Av. 9 132-A-01 Bloque M. Piso 3, Bogotá D.C. 110121, Colombia
{orlandolopez, alejandroleon, mauriciosandoval, despejo}@unbosque.edu.co

Abstract. Programming skills of software engineers that affect software development productivity are central to any of the computing disciplines. While literature focuses on how to teach novice programmers, the aim of this research is to show how to strengthen programming skills of programmers by effectively transferring knowledge to those who had bad experiences when learning computer programming or have not developed enough programming skills to get a productivity standard. Since software engineering is a knowledge-intensive application discipline, a knowledge transfer process is conducted to improve the productivity of computer programmers involved in software engineering projects. An *ad-hoc* methodology allowed to follow-up changes that revealed that improvements in the capability to absorb new external knowledge increases overall productivity of individuals in software development teams. This finding may be useful for software companies looking for increasing their productivity.

Keywords: Knowledge Transfer, Knowledge Management, Teaching Computer Programming, Software Engineering.

1 Introduction

Computing disciplines have been identified as Computer Engineering, Information Technology, Information Systems, Software Engineering and Computer Science [1] but neither students nor businesses differentiate them [2]. Even worse, in some countries, academic programs do not hold any of those names but 'systems engineering' which, according to INCOSE denotes "*an engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder's needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire life cycle*" [3], this is to say that "systems engineering" refers to a broader body of knowledge than just software engineering, computer programming, or any of the computing

J. Mejia et al. (eds.), *Trends and Applications in Software Engineering*, Advances in Intelligent Systems and Computing 537, DOI 10.1007/978-3-319-48523-2_14

disciplines. However, any of these disciplines require to develop computer programming skills, especially when people is going to be involved in software development projects.

Many studies spend efforts to make distinctions between these disciplines, but this research focuses on something that they share in common. They share the need to produce computer programs (i.e. software) whether as an art [4-6] or science [7] or, more plausible, as engineering [8]. The issue tackled in this study is not how to make the software seem to do what is supposed to do, but how software can be produced minimizing programmers time. This research draws the attention to the fact that behind computer program development there are computer programmers. People that is responsible to produce software from the requirements phase to the operations and maintenance phase. People in need of training on software development methodologies. People in need of interacting with others in respectful ways in a team work.

Many other studies have been conducted in order to overcome difficulties involved in teaching programming in an introductory course [9, 10] but, to the best of our knowledge, no studies report on results enforcing computer programming abilities in programmers (coming from any computing discipline) involved in a project. A question arose at this point: What is the impact of computer programming knowledge transfer on software productivity? This led to involve senior students of a "*systems engineering*" program in a real software development project, working at a client place next to experienced developers. Individuals were immersed in a stress-controlled but real software development and implementation environment.

The aim of this paper is to show results of a research focused in increasing productivity of programmers by improving their programming skills from a knowledge-based centered process. In addition, this is new because studies are centered to engage freshmen students in programming disciplines but not to retain workforce and improve their skills.

This paper is structured as follows. First, the phases defined to guide the research, then a succinct section of the relationship between the concepts of knowledge management and software is introduced. Then, the sections of results analysis and conclusions are developed.

2 Methodological Issues

In order to conduct a research regarding the real world either to explain it or transform it, qualitative and quantitative approaches are sides of the same coin: both, a qualitative approach to meanings and a quantitative approach to facts are needed [11]. In fact, The distinction between qualitative and quantitative research methodologies is ideological [12]. This is to say that complex phenomena, such as software development, should be studied both in its quantitative dimensions, and its qualitative dimensions as well. Even more, that an emphasis on quantitative or qualitative issues is not a priority. A different enriched methodological approach must be addressed when dealing with complex environments [13, 14] or phenomena such as the one being reported in this paper.

2.1 The Phases of the Research

The methodology is structured in four phases. (i) Preparation and selection phase consists of two parallel steps: first, determination of the environment to conduct the experiment. The right environment is a real software development project that may provide space to conduct tutoring activities. And, second, recruiting of senior students of an undergraduate program of a computing discipline from those with the lowest grades in computer programming courses. An assessment of students´ abilities is conducted to state a baseline of knowledge. Candidates are interviewed to validate their negative attitude towards computer programming. From this interviews arise assessment categories (Table 1). These categories are more specific than those presented in other studies [15].

(ii) During immersion phase, selected individuals are exposed to a real project that is being conducted. They are informed about the problem to be solved by the product resulting from software development. Tutoring meetings are scheduled in order to improve their programming abilities.

Table 1. Categories to assess knowledge internalization of each subject of study.

Id	Category	What to assess
C1	Interest to learn computer programming.	Basic interest to learn to program computers by himself or by means of the support of a tutor.
C2	Computer programming language knowledge.	The level of understanding and usage of the Java® programming language.
C3	Teamwork	Previous experience working with unknown people in team activities.
C4	Project control	Compliance with assignments, advances on assignments, deadlines, and deliverables (as well in programming code as in documentation).
C5	Computer technology	Specific knowledge of computer technology to use during project development both in software development and communications and collaborative software supporting development activities.
C6	Methodology	Knowledge and usage of the methodology used in the software development project.
C7	Frameworks learning	Understanding of *PrimeFaces* and *Spring* frameworks. Usage of this frameworks in the project.
C8	Database modeling and database management from code.	Learning to manage the database from Java code.
C9	Object Oriented Programming	Knowledge of the object-oriented programming paradigm in practice.
C10	Usability basic practices	Analysis, learning and usage of basic usability practices to produce a functional and easy to use software application.

(iii) The third phase is a hands-on learning process and follow-up. Each individual is instructed to complete a field diary. By means of field diaries, monitoring and guidance are provided. Advances are followed-up.

(iv) The fourth Phase consists of the final assessment procedure to compare actual abilities and knowledge that each individual actually exhibits with respect to the knowledge baseline that was recorded in phase one.

These phases conform a scene to induce improvement in knowledge for each individual involved in this study, as well as a general stage for an ongoing assessment process.

2.2 Issues in Conducting the Phases of This Research

When in phase one, besides looking for an actual software development project, an environment allowing effective tutoring to individuals is required. In addition, this should not disturb actual software development and avoid client concern or annoyance. This was not easy to set. However, the research was conducted in a real scenario where the client was asking to develop a software piece to capture data on medical variables from biomedical devices, including manual data gathered when using devices such as tensiometers and stethoscopes at a pediatric intensive care unit. The assessment categories introduced in Table 1 were refined by reviewing personal logs of selected students.

In order to recruit volunteers, students from a senior cohort were selected according to their low performance (low grades) in computer programming courses. In addition, those that were selected were asked to express their opinion in relation to programming in practice. Those who expressed dissatisfaction toward computer programming, low level of knowledge of programming languages and programming paradigms were preferred. Interest to learn and their abilities to work as team members were determined by means of questionnaires. This allowed to identify the initial programming abilities of the individuals and to set a baseline. Finally, some of them were selected. Follow-up was done by inspection of development of user stories and field diaries, and recording and tackling difficulties arising in the ongoing project

This paper reports results from two of the volunteer students that were involved in the experimental process. They never worked before as a part of a software development team. They received user stories that were refined by group meetings including members of the development team and final users from the client organization, by using activity (hands-on experience) records and personal logs (one for each individual). This last element was crucial to assess changes occurred in individuals, especially when comparing to the baseline assessment.

The process was divided into “learning stages” and “stages for practice”. During learning stages, individuals were asked to read chapters of different books on object oriented programming, or contents of web pages, in order to complement or refresh programming concepts. Meetings were held to provide support regarding some topics required to proceed to develop. Stages of practice were guided by IEEE 1074 tailoring the software development life cycle [16]. At the first (practice) stage, individuals showed low performance. This situation led to a delay in user stories deployment. The researchers were not disappointed with this gross result, as this was supposed to happen. Patience was definitely worthwhile. After individuals involved progressively with the project and feeling confident with themselves, a work team was consolidated.

However, in spite of the fact that individuals devoted about 25 hours per week to produce code and documentation deliverables of the project, for practice and learning stages, it took longer to accomplish. However, results encouraged individuals to

improve compliance with deadlines to the point they accomplished to deploy user stories as soon as the client were requiring them. This enthusiastically encouraged individuals to look for the development of deliverables in order to get user satisfaction.

A category, "interest to learn" (Table 1) was central to the development of the classification from individual logs reviews, deliverables from user stories summing up thirty-one modules [17]. The overall involvement of the subjects in the project was divided into four stages, identified as stages 1 to 4, each one of a three months period. Each stage was followed immediately by another.

During the first stage, subjects were trained and self-trained on the computer technologies required to be engaged in the project. This allows individuals to adapt daily to the work of software development.

3 Knowledge Transfer and Software

Recent attention has been paid to knowledge transfer in software engineering [15] either understanding knowledge as a main asset in software organizations [18 p.26, 19 p. 105] or because it is a knowledge intensive discipline [18, 20] as well as a computer programming skill [9] demanding activity. Even so, the software industry has been recognized as an "engineering" endeavor but of a different kind [8]. The reason is that software is manufactured once (then deployed many times) and it is essentially an abstract product, or at least with no physical component. Software as a product is more like a book of poems than a bridge. Both are produced once, but the physical dimension of a bridge is necessary for its usage while the physical part of a book (paper, ink and so on) may be abstracted, for instance, by publishing it as an electronic book. In addition, this implies that statistical quality control may not be applied to the software production process [21]. The essence of the book is the knowledge that has been codified: the poem. Even better, the codification of knowledge is what makes the essence of the book. The same applies to software, even when it is maintained [22]. Nonetheless, up to this point, there is nothing completely new.

What is new is to focus this research on the 'workforce' to produce software. Software development is a creative process that is conducted by human beings at their intellectual level. In this context, software engineers (or computer scientists, or "systems engineers") and poets or writers are alike. Their challenge is to conduct intellectual processes to produce a result that is a unique instance of a class of abstract objects.

4 Results Analysis

In order to respond the question asked in the introduction, assessment records were plotted on a graph (Fig. 1). Both subjects (subject 1 and subject 2) were exposed to the same project in four stages (listed from 1 to 4 horizontally in Fig. 1 for subject 1, and in Fig.2 for subject 2).

Since the research is focused in what is changed in a specific environment (i.e. the interest is focused more on the ongoing process than on final stand-alone results), the changes between stages were observed. Therefore, for each stage 1 to 4, the same categories C1 to C10 are assessed. Each category was graded from 1 to 10 (arrayed vertically). Grades from 1 to 3 were considered Low, 4 to 7 Medium, and 8 to 10 High.

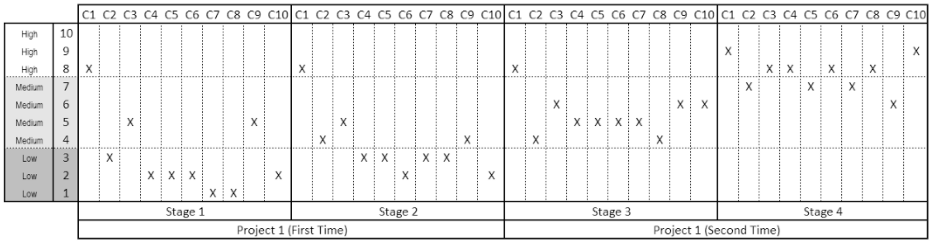

Fig. 1. Knowledge transfer assessment for subject A. Horizontally, four stages are depicted (*stages 1 to 4*). During stages 1 to 4 each of the ten categories in Table 1 were assessed.

Subject 1 in stage 1 got just one item High (C1), two items Medium (C3 and C9), and the remaining items were graded Low (Fig. 1). During stage 2 the item C1 continued High, while item C2 passed from Low to Medium. C3 and C9 stayed Medium with a little local decrease of C9 from 5 to 4. While the items C4,C5,C6,C7,C8, and C10 remained Low, it was encouraging the local change of its grading from 'very low' values to values higher in the same interval.

Stage 3 for subject 1 was a qualitative jump in knowledge categories assessment from mainly Lows to mainly Mediums. And stage 4 led to an unexpected mainly Highs and upper Mediums (Fig.1). Just C3, C5, C7, and C9 of subject 1 remained Medium. Examining each of the items, C1: Teamwork, C5: Computer Technologies, C7: Frameworks learning, and C9: Object Oriented Programming were the items in upper Medium.

For the case of subject 2 (Fig. 2), assessment of the categories C1 to C10 during stage 1was not qualitatively different from subject 1. This means that grades for categories being assessed were mainly Low. Just C3, C9, and C10 were in the Medium Interval. However, C9 and C10 were at the lowest Medium grade.

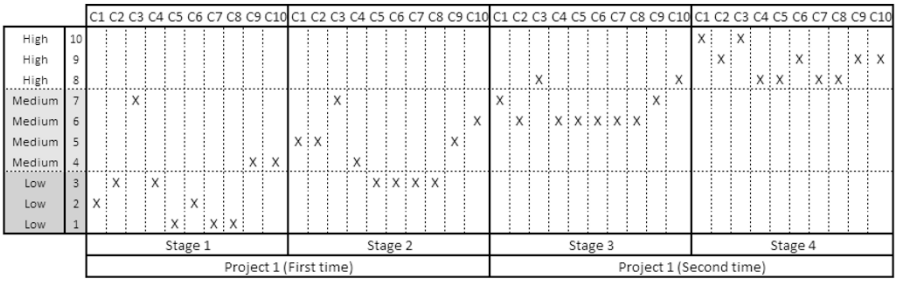

Fig. 2. Knowledge transfer assessment for subject B. Horizontally, four stages are depicted (stages 1 to 4). During stages 1 to 4 each of the ten categories in Table 1 were assessed.

During stage 2, there was a significant improvement in grades in the overall set of categories. They were assessed mainly in the Medium interval and just C5, C6, C7, and C8 were at the Low interval but at the upper grade of the interval.

Stage 3 proved to be an important improvement for subject 2. Must of the categories were at the upper Medium interval, and C3 and C10 were at the lower High interval. But stage 4 revealed outstanding results for this subject. Every category was graded at the High interval. Just four out of ten categories, C4, C5.C7, and C8 were at the lowest grade of the High interval.

The lowest graded categories at stage 4 for subject 2 were C4: Project control, C5 Computer Technology, C7: Frameworks learning, and C8: Database modeling and database management from code. When comparing this list with the lowest categories at stage 4 for subject 1, C5 and C7 are in common.

In addition, it may be observed that the category C3: teamwork was not fully developed, that may be explained by a low slope during stages 1 and 2, which in turn could be explained by externalities. From stage 1 to stage 3 the increase was not significant, but from stage 3 to stage 4 an unexpected and relatively significant increase was observed in the overall set of categories being assessed.

In order to check the consistency of this assessment, an additional measurement was considered: the user stories deliverables. These were increasing from stage to stage also, which means that subjects achieved higher levels of productivity as time goes by.

Subjects selected for this research exhibited poor or limited computer programming skills. Both subjects reported in this study continued during the first two stages of the experimental process to display real difficulties on a range of fundamental skills for integrating to a software development project, not just because of their low programming skills but because of their low profile in abilities like teamwork and project control. This is because the simple model of knowledge transfer [23] consisting of agent A making knowledge available to the environment of Agent B, as it were the classical data communication model [24], does not reveal the essence of knowledge transfer. Knowledge transfer is not a matter of data communication as in information theory. For knowledge to be effectively transferred, the receiver (Agent B) must not be a passive agent, but must exhibit the dynamic capability to absorb the knowledge [25, 26] available to make it productive.

The results shown in stages 3 and 4 support the statement that absorptive capacity of individuals or organizations [25, 26] must be developed before knowledge may be exploited by the receiver. In this context, the capability to increase the number of deliverables by the subjects of study involved in a software development project.

5 Conclusion

This paper has introduced a hands-on experiment to teach computer programming while “learners” are involved in a “true” software development project. It is worth noting that while the current interest of many researchers is focused in novice programmers, this experiment was conducted over individuals of a computing discipline expressing negative experiences towards computer programming with low-

level computer programming skills. This is to say, that the focus is to find practical ways to improve skills –capabilities of agents- to "produce" software deliverables of the project, not just coding programs and ensuring their correctness [7]. Or to phrase it another way, this research is focused to improve individual skills from those who has previous knowledge of computer programming and, however, despite of it, they have not reached some productivity standards.

The experiment conducted was not a set of training sessions or the development of an educational course syllabus. The experiment was a knowledge transfer process. What was ensured was the process of developing the capabilities of the receiving individuals to make computer programming knowledge productive in a real environment. The results observed on the individuals under study allow infer that productivity was significantly increased in a relatively short period of time, as a result of a controlled process of knowledge transfer.

From the experiment that was conducted, it was found that the ten categories (Table 1) that defined the set of assessment parameters showed that in subjects under study reveal a knowledge absorption process and knowledge seizing by exploiting developed (and developing at the individual level) programming skills in a real software engineering project environment. Individuals were involved on a part-time basis in this study. It could be thought that on a full-time basis an improvement in software productivity may be achieved in a shorter time.

A generalization of this finding is still an issue because knowledge is not a matter of data accumulation, but a cognitive process. Knowledge transfer could not be measured directly, so proxy variables such as those of the categories in Table 1 were measured to obtain an indirect estimate of the knowledge effectively transferred. This opens an opportunity to conduct research in working teams of real software engineering projects about productivity increase by improving absorptive capacities regarding specific categories in a similar way as the categories (Table 1) involved in this study.

Acknowledgments. Authors express their gratitude to Hospital Santa Clara, Bogotá D.C. for authorizing access to their software project at the Pediatric Intensive Care Unit. Especial thanks to Dr. Armando León Villanueva, Pediatric Lung Care and Pediatric Pulmonologist of the Pediatric Intensive Care Unit, and Dr. Maria Claudia Guzmán Diaz Pediatrician of Universidad El Bosque and 'Hospital Cardiovascular del Niño' San Mateo, Cundinamarca. Authors would further like to thank three anonymous reviewers for their useful comments and feedback which have helped to write the final version of this paper.

References

1. ACM-IEEE, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science - December 20, 2013. ISBN: 978-1-4503-2309-3*. 2013, United States of America: The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society.

2. Courte, J. and C. Bishop-Clark, *Do students differentiate between computing disciplines?* ACM SIGCSE Bulletin. **41**(1): p. 29-33. (2009)
3. INCOSE. *What is Systems Engineering. The International Council on Systems Engineering.* http://www.incose.org/AboutSE/WhatIsSE.
4. Knuth, D.E., *The art of computer programming: Sorting and searching*. Vol. 3, Reading, Massachussets: Addison-Wesley. 426-458. (1999)
5. Knuth, D.E., *The art of computer programming Fundamental Algorithms*. Vol. 1, Reading, Massachussets: Addison-Wesley. World students series. (1973)
6. Knuth, D.E., *The art of computer programming: Seminumerical algorithms*. Vol. 2, Reading Massachussets: Addison-Wesley (1973).
7. Gries, D., *The science of programming*. Springer Science & Business Media. (2012)
8. Bryant, A. *It's engineering Jim… but not as we know it: software engineering—solution to the software crisis, or part of the problem?* in *Proceedings of the 22nd international conference on Software engineering*. 2000. ACM. (2000)
9. Rubiano, S.M.M., O. López-Cruz, and E.G. Soto. *Teaching computer programming: Practices, difficulties and opportunities*. in *Frontiers in Education Conference (FIE), 2015. 32614 2015. IEEE*. (2015)
10. Plonka, L., et al., *Knowledge transfer in pair programming: An in-depth analysis.* International journal of human-computer studies. **73**: p. 66-78. (2015)
11. González López, J.L. and P. Ruiz Hernández, *Investigación cualitativa versus cuantitativa:¿ dicotomía metodológica o ideológica?* Index de Enfermería. **20**(3): p. 189-193.(2011)
12. Monzón Laurencio, L.A., *Ni cualitativo ni cuantitativo: un estudio hermenéutico analógico sobre la metodología de la investigación. (*2011)
13. Mejía, A., et al. *Ser directo puede traerte problemas, pero ser indirecto también: Las realimentaciones en dinámica de sistemas cualitativa y cuantitativa*. in *Artículo aceptado para el Congreso Latinoamericano de Dinámica de Sistemas*. (2007)
14. Aceros, V., et al., *¿Qualitative or quantitative? That's not the question: a method for developing dynamic hypotheses.* , in *Proceedings of the 9th Latin American System Dynamics Conference*. Universidade de Brasília: Brasilia. (2011)
15. Camacho, J.J., J.M. Sanches-Torres, and E. Galvis-Lista, *Understanding the Process of Knowledge Transfer in Software Engineering: A Systematic Literature Review*, in *The International Journal of Soft Computing and Software Engineering [JSCSE]. Special Issue: The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13], Doi: 10.7321/jscse.v3.n3.33 e-ISSN: 2251-7545*. 2013: San Francisco State University, CA, U.S.A. p. 219-229. (2013)
16. Fitzgerald, B., N. Russo, and T. O'Kane, *An empirical study of system development method tailoring in practice.* ECIS 2000 Proceedings, p. 4 (2000)
17. Cresswell, J.W., *Research design Qualitative, quantitative and mixed methos approaches*. Sage Publications. (2009)
18. Rus, I. and M. Lindvall, *Knowledge management in software engineering.* IEEE software. **19**(3): p. 26. (2002)
19. Mathiassen, L. and P. Pourkomeylian, *Managing knowledge in a software organization.* Journal of Knowledge Management. **7**(2): p. 63-80. (2003)
20. Ward, J. and A. Aurum, *Knowledge management in software engineering-describing the process*, in *Australian Software Engineering Conference, 2004. Proceedings. 2004*. IEEE. p. 137-146. (2004)
21. Basili, V.R. and G. Caldiera, *Improve software quality by reusing knowledge and experience.* MIT Sloan Management Review. **37**(1): p. 55. (1995)
22. Batista Dias, M.G., N. Anquetil, and K.M. de Oliveira, *Organizing the knowledge used in software maintenance.* J. UCS. **9**(7): p. 641-658. (2003)

23. Ajith Kumar, J. and L. Ganesh, *Research on knowledge transfer in organizations: a morphology.* Journal of knowledge management. **13**(4): p. 161-174. (2009)
24. Shannon, C.E., *A mathematical theory of communication.* ACM SIGMOBILE Mobile Computing and Communications Review. **5**(1): p. 3-55. (2001)
25. Lopez-Cruz, O. and N. Obregon Neira, *Diseño de la capacidad de absorción en las organizaciones: propuesta de un nuevo constructo y literatura*, in *Congreso Nacional e Internacional en Innovación en la Gestión de Organizaciones, Abril, 2016*, F.R. Santoyo, Editor. Universidad Central: Bogotá. p. 222-237.(2016)
26. López-Cruz, O. and N. Obregón Neira, *Design of the Organizational Absorptive Capacity: A New Construct Proposal and Literatures.* In publishing, (2016)