4-2008

Teaching Successful "Real-World" Software Engineering to the "Net" Generation: Process and Quality Win!

William L. Honig
Loyola University Chicago, whonig@luc.edu

Teaching Successful "Real-World" Software Engineering to the "Net" Generation: Process and Quality Win!

William L. Honig
Loyola University Chicago
whonig@luc.edu

Abstract

Software engineering skills are critical for students seeking careers as software developers. However, academic course content often fails to teach practical, "real-world" software engineering as it is done in large organizations. Further, the proclivities of the current generation leave students disinclined to the disciplines of process and quality. Academics seldom use the Team Software Process (TSP), a leading methodology of global industries. Four years of data indicate that student teams using TSP can achieve industry levels of productivity and reasonable quality levels. Further, results from 23 teams and over 200 students indicate that these Net-Generation students developed an understanding for the value of discipline, data collection, metrics, and quality measures. The Team Software Process is recommended to other academic programs seeking to bring real-world software engineering into the classroom and improve teaching for the Net Generation.

1. Introduction

Studies of the modern generation have identified distinctive characteristics and differences in their learning approaches and mechanisms. They often search for information and problem solutions in a "haphazard" fashion [1]. When using information, they have been described as superficial and experiencing "knowledge fragmentation" [2]. Their penchant for rapid attempts at finding solutions without planning has been linked to hand-held games and called "Nintendo over Logic" and a symptom of their "zero tolerance for delays" [3].

Various labels, such as Generation X, Y, or Z, and Millennials, have been coined for these students; here they are termed the "Net Generation". The Net Generation's mindsets lead them to aberrant approaches when immersed in software creation. They have been observed doing "repeated hasty designs, followed by futile patching" [4]. When teaching introductory programming classes, the author has struggled to overcome student's desire to work by trial and error, and to make rapid, iterative attempts to fix programs. In brief, the Net Generation of programmers seem to be hasty, undisciplined, and always rushing – they seldom take time to plan or analyze but have plenty of time to perform ad hoc work with great energy.

Software engineering, particularly in the real world of industry and professional software organizations, requires discipline, planning, and controlled decision making. It often requires teams of developers to work closely together and make joint decisions on what to do and when to do it. This approach is diametrically opposed to the Net Generation mindset and their normal ways of doing things. As a result, bringing the real world of software engineering into academic programs is challenging [5, 6].

The Team Software Process (TSP) is a comprehensive methodology for team programming and has been widely used in real-world organizations [7]. TSP has been used by other academics to support a "case study" approach [8], and to implement "capstone" courses [9, 10]. However, TSP is not widely used in software engineering courses and its effectiveness at teaching the Net Generation has not been explored.

This paper reports a successful multiple year experiment using TSP in a graduate-level software engineering course. Section 2 details the use of TSP in the course and the structure to bring the "real world" into the classroom. The productivity and quality results of 23 large student teams, presented in Section 3, show they performed similarly to industry software groups. These students also achieved an understanding for the value of process, discipline, and quality which counters some of the Net Generation's typical approaches to programming (Section 4). Sections 5 and 6 conclude with suggestions to other academics seeking to bring the real world into the classroom software engineering experience.

2. Software engineering course structure

Students enter the graduate software engineering course as experienced software developers typically with some prior experience of team software development. In the course, students are organized into teams, work as both developers and in an assigned role according to the TSP rubric, produce software for a real client, and self-manage their project in a real world setting.

TSP defines five specific roles and functions: Support Manager, Quality/Process Manager, Planning Manager, Development Manager, and Team Leader. Students are assigned to one of these roles by the faculty member based on their backgrounds; students typically remain in the assigned role for the entire course (14 to 17 weeks). All students also function as Software Developers creating assigned parts of the system. The author also assigns members to teams to create a mix of cultures and genders. Student teams have from seven to twelve members, similar in size to many real world software teams. The course uses a specially designed academic version of the TSP [11].

The team works on a single implementation project for the full course. Two complete development cycles are carried out on a schedule defined by the faculty member (unlike the real world, the course calendar is an unchangeable schedule). The teams begin work with a brief problem statement and select the functions to be implemented in each cycle. The teams complete one or more phases of the TSP each week (Launch, Strategy, Plan, Requirements, Design, Implementation, Test, and Postmortem) and all phases are done sequentially in each development cycle.

The projects are new each semester and have a real-world client, typically a university staff group. Client representatives meet with the class two to four times during the semester to review team-generated requirements and other deliverables and to see demonstrations of the team's systems. Clients respond to team questions about system functions using an online discussion board.

The instructor edicts strict rules for team operation and behavior to enhance the real-world feeling of the experience. Teams are required to meet at least weekly outside the class sessions at an assigned time and place. The team meetings are face-to-face (as opposed to online or various asynchronous communications) and attendance is taken and reported to the faculty member. Strict team discipline is enforced by requiring meeting agendas, minutes, and recording of action items for tracking. The instructor makes unannounced visits to the team meetings to coach and answer questions on TSP. The TSP scripts give structure to the team's weekly activities with deliverables and exit criteria for each week.

3. Team productivity and quality results

The TSP requires weekly collection of extensive data including time worked on specific activities, size of documents and code produced, and problems, bugs, or defects found. Student teams are graded on the accuracy of their data collection to encourage accurate record keeping.

By the end of the course students show proficiency at analysis of this data to understand how their team is performing (see section 4). The data collected by 23 student teams over four years show the teams achieving reasonable real-world quality and productivity results.

All results reported here are based on the second cycle of team development. Cycle one data exhibits much greater variations as the teams are learning the process and perfecting their data collection methods.

3.1. Student teams achieve real-world software productivity

The cycle two productivity for each team is shown in Figure 1 in lines of code per hour. The total lines of code include new software written in cycle two and lines from cycle one that were modified. The total hours include all time worked in the cycle, including time spent on planning, requirements, and postmortem (not just time coding and testing). This productivity measure is similar to those used in industry to measure overall team productivity during a project.

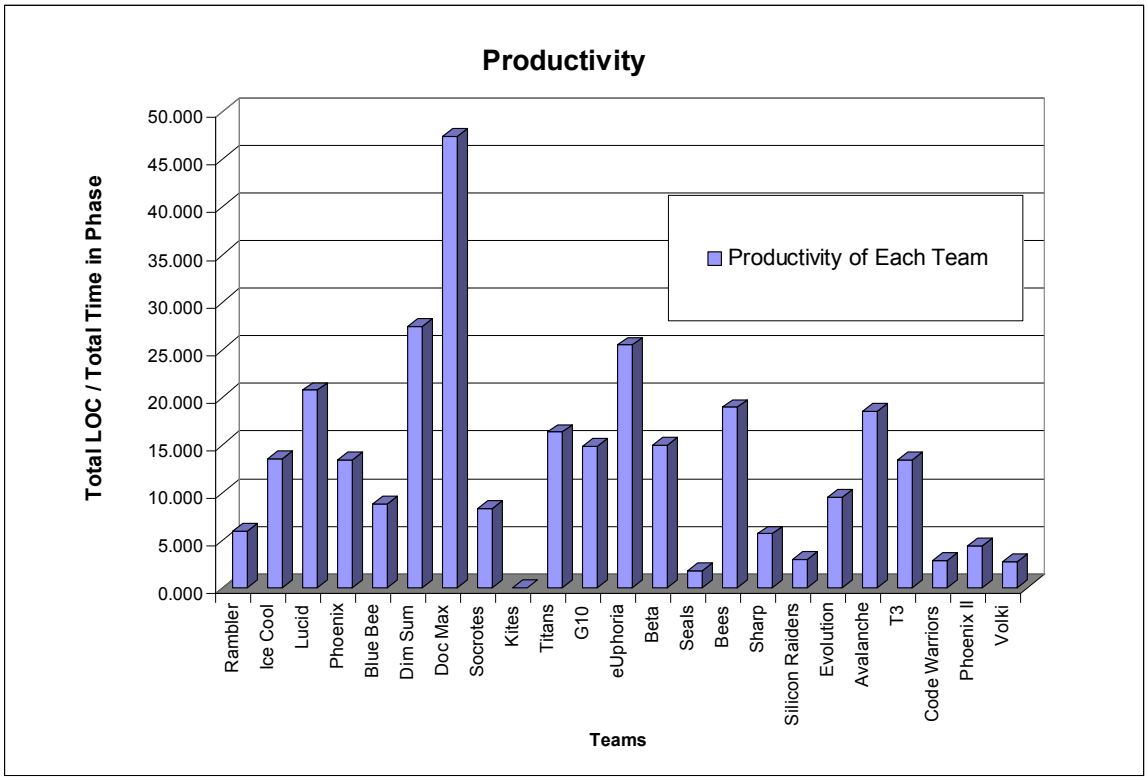

Figure 1. Team productivity, cycle 2, source lines of code per hour

The student teams typically created software at 10 to 20 source lines of code per hour; in contrast, students working alone or on two or three person projects, can create code at least two or three times faster. However, the student productivity rate with TSP is similar to that seen in many industrial software organizations.

When using TSP with only a few weeks of training, the student teams achieve a software productivity rate similar to real-world organizations. This similar rate reassures students that the time spent on process, which appears very unusual and contrived to them, is not a waste of time.

3.2. Student quality varies greatly

Tracking and analyzing product quality during the development process is also a key part of the TSP. Teams are required to record each bug or defect found (in all team deliverables including documents) and report and summarize the defects weekly. Again, this activity is quite alien to today's generation of students who seem more intent on quickly making numerous changes until something works reasonably well.

Defects are totaled for the entire cycle as a high-level measure of product quality; Figure 2 shows the total defects for each team at the end of cycle two. Defects are normalized to the total size of the product in lines of code. Figure 2 displays team results as a probability of a single line of code having an error; the teams average about 0.02. In more common terms, the team's product quality at the end of cycle two is about twenty errors per every thousand lines of code.

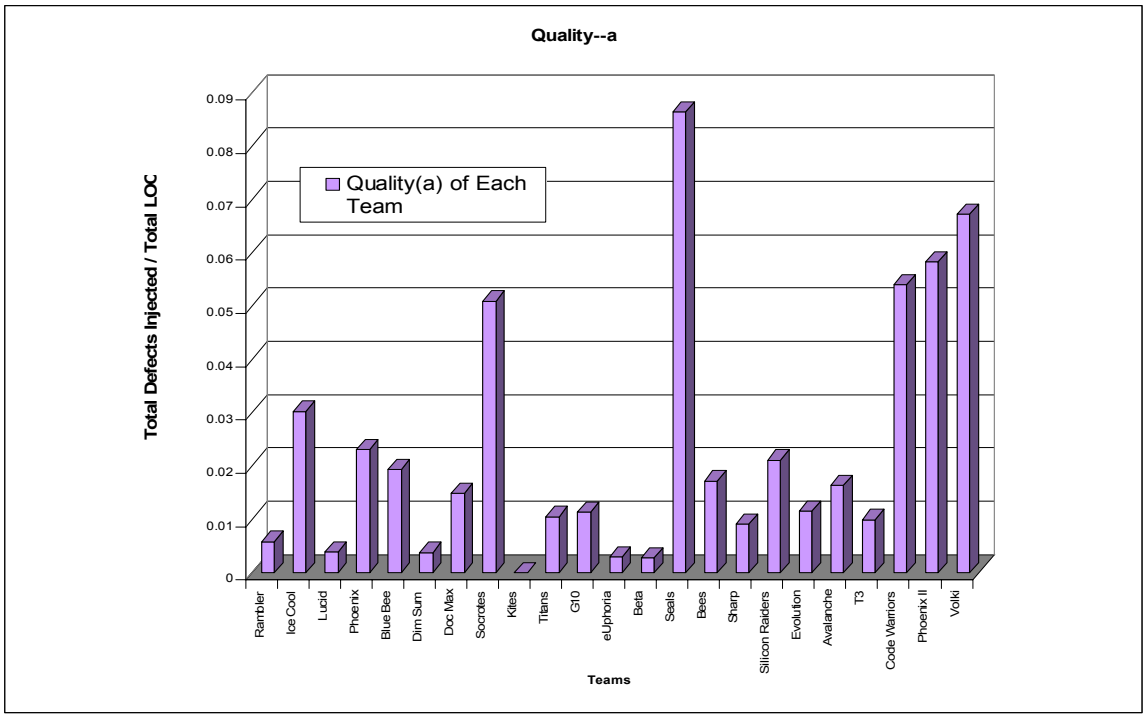

Figure 2. Team product quality, cycle 2, defects injected per line of code

It is important to note that this measure of product quality is, in effect, measuring the product problems found during development. At the end of the course, the product is "done"; however, it has not been put into use by the customers (except for very limited testing). In the real world, quality measures would include defects found during the actual use of the product in the field. For the student projects, these unfound problems are not included in the total quality measure.

Figure 2 also shows a wide variation in defect rates for the various teams. Team bug finding is very dependent on the effectiveness of their code inspections and product testing. The Net Generation, as noted in the introduction, is not naturally inclined to seek full and careful results. The teams are taught basic testing heuristics and encouraged to use fault seeding; however, their actual ability to find defects is also influenced by the time left for testing as the course comes to a close.

Nevertheless, the product quality is reasonable in comparison to real-world teams, especially those who are just beginning process-driven quality programs. Seeing and explaining the defect information is a key part of making the modern student aware of the value of quality in a product.

4. Student outcomes and the value of process and quality

The quality and productivity results raise an interesting question: are the class participants developing an appreciation for process driven software engineering or are they simply doing what is necessary to get through the class? Although students may listen and understand an attempt to expose them to real-world development practices, does the course create any change in their thinking about what makes a software project "well done"?

Student perception of the course is shown by informal feedback. The course is popular despite being viewed as a great deal of work and requiring considerable team meeting time outside of class. Students also report heightened interest in systems analysis, design, and testing techniques, probably due to experiencing the impact of these skills on their team's project.

However, the real issue is whether students value the metrics and data collection process and would use it to improve future projects. Such data driven decision making is not readily chosen by many of the Net generation who may prefer to simply do what is thought "right" at the moment.

TSP data collection and analysis is based on a number of forms which the students generate each week. An end-of-course survey asked students to rate the value of these forms; since there are many forms in TSP, they were asked to pick the forms most useful to projects such as theirs. Forms related to tracking product quality (counting defects and tracking changes) were perceived by more than half the students to have value. The specific TSP forms selected were:

- Configuration Change Request (CCR) – seeking approval of changes to fix defects
- Inspection Report (INS) – measuring productivity and counting defects in code inspections
- Defect Log (LOGD) – details for each defect found, including probable cause
- Configuration Status Report (CSR) – weekly summary of all changes for the week

Surprisingly, the configuration management reports were perceived as high value; these forms detail the implementation of processes designed to manage change (and in some cases defer changes to later development cycles). Working in large teams using TSP, students of the Net Generation developed an appreciation for limiting their natural tendency to make changes rapidly without consulting others.

Similarly, the finding and tracking of software problems would not normally capture the attention of the Net Generation. The course structure successfully taught students the value of focusing on bugs and problems, instead of considering defects something to be fixed quickly and hidden.

5. Conclusions

Students completing the course developed an appreciation for and acceptance of process and data analysis as a central and critical element of software development. They acquired an

understanding for metrics and reported their progress weekly, overcoming some of their tendencies to work quickly without regard to the consequences, results, and history.

Twenty three teams of over 200 graduate students used the TSP successfully and achieved productivity and quality results that compare reasonably with real-world teams. The TSP weekly measurement and reporting process effectively forced students to focus on process instead of putting all their energies into writing and testing code quickly in an ad hoc fashion.

Other academic organizations seeking ways to prepare students for real-world software development may consider the Team Software Process (TSP) an effective tool for software engineering courses. TSP proactively pushes students of the Net Generation into first experiencing and then appreciating an engineering and process driven way to create software.

6. Questions and Further Investigations

Students completing the described course developed an appreciation for and acceptance of process and data analysis as a central part of the software development process and seem better equipped to join real-world teams. However, the student teams still struggle with getting an early handle on product quality. The teams determine the quality of the product from the data as opposed to predicting and controlling the quality during development. TSP includes useful measures of quality including defect rates in document reviews and initial coding inspections (many summarized weekly on Form SUMQ). Although students generate these metrics and use them as a measure of results, they do not successfully use the information to improve product quality. Thus, further work is needed to determine effective ways to increase student understanding of quality control metrics earlier in the development cycles.

Today's real-world software development also includes distributed teams [12], outsourcing [13], and predetermined architecture and software reuse. TSP and the classroom methods described here can evolve to be even more realistic by including appropriate exposure to these in the team experience.

7. References

[1] S. Bodi, "How Do We Bridge the Gap Between What We Teach and What They Do?", Journal of Academic Librarianship, 28(3), May-June 2002, pp. 110-115.

[2] B. Costello, R. Lenholt, J Stryker, "Using Blackboard in Library Instruction: Addressing the Learning Styles of Generations X and Y", The Journal of Academic Librarianship, 30(6), November 2004, pp. 452-460.

[3] D. Oblinger, "Boomers, Gen-Xers, and Millennials: Understanding the 'New Students' ", EDUCAUSE Review, 38(4), July-August 2003, pp. 36-40ff.

[4] D. Ginat, "Hasty Design, Futile Patching, and the Elaboration of Rigor", Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Dundee, Scotland, June, 2007, pp. 161-165.

[5] N. Tadayon, "Software Engineering Based on the Team Software Process with a Real World Project", Journal of Computing Science in Colleges, 19(4), April 2004, pp. 133-142.

[6] M. Nauman, M. Uzair, "SE and CS Collaboration: Training Students for Engineering Complex Systems", Software Engineering Education & Training, CSEET '07, Dublin, Ireland, July, 2007, pp. 167-174.

[7] W. Humphrey, "The Team Software Process", Technical Report CMU/SEI-2000-TR023, The Software Engineering Institute, Carnegie-Mellon University.

[8] T. Hilburn, M. Towhidnejad, "A Case for Software Engineering", Software Engineering Education & Training, CSEET '07, Dublin, Ireland,, July 2007, pp. 107-114.

[9] R. Conn, "A Reusable, Academic-Strength, Metrics-Based Software Engineering Process for Capstone Courses and Projects", Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '04, Norfolk, Virginia, USA, March, 2004, pp. 492-496.

[10] B. von Konsky, M. Robey, "A Case Study: GQM and TSP in a Software Engineering Capstone Project", Software Engineering Education & Training, CSEET '05, Ottawa, Canada, April, 2005, pp. 215-222.

[11] Humphrey, W., Introduction to The Team Software Process, Addison-Wesley, New York, 2000.

[12] D. Petkovic, G. Thompson, R. Todtenhoefer, "Teaching Practical Software Engineering and Global Software Engineering: Evaluation and Comparison", Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Balogna, Italy, 2006, pp. 294-298.

[13] W. Honig, T. Prasad, "A Classroom Outsourcing Experience for Software Engineering Learning", Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Dundee, Scotland, June, 2007, pp. 181-185.