Comparison of Student Experiences with Plan-Driven and Agile Methodologies

Carol A. Wellington, Thomas Briggs, C. Dudley Girard
Department of Computer Science, Shippensburg University
Shippensburg, PA 17257
cawell@ship.edu, thb@ship.edu, cdgira@ship.edu

Abstract - In Fall of 2004, we offered two software engineering courses: one in plan-driven methodologies and one in agile methodologies. In these courses, the students work on large projects in teams of 14 to 16 students using variants of Team Software Process (TSP) or Extreme Programming (XP). In order to compare the students’ experiences with these methodologies, the team in the plan-driven course and one of the agile teams were given the same problem statement. Throughout the semester, we measured team cohesion and individuals’ attachment to the project. To measure team cohesion, we modified the Group Environment Questionnaire that has been shown to accurately reflect team cohesion in sports teams. We also developed some of our own cohesion metrics and a measure of attachment to the project. While the GEQ showed no significant difference between the teams, our measures showed higher overall cohesion in XP, but higher sub-team cohesion in TSP. At the end of the semester, we also compared the functionality of the applications the teams developed and a variety of code metrics measuring the quality of their code and its design. While the team’s developed approximately the same amount of functionality, in general, the XP team’s code had better metrics. The TSP team required much more code to accomplish the same functionality because, although they had a strong design, their implementation did not leverage inheritance as the design expected.

Index Terms – Software Engineering Education, Team cohesion, Extreme Programming, Team Software Process.

INTRODUCTION

The software engineering concentration offered by Shippensburg University includes two product development courses. The goal of these courses is to give our students to experience widely varying development processes. We want them to be aware that the software development process creates an “ecosystem”[6] of a software development project by defining things like room layouts, people issues, and communications channels. These decisions can directly affect the quality of the resulting product. For example, issues relating to managing people have been shown to affect architecture decisions[5]. Our program is designed to make students aware of these important side effects that process choices have.

While ecosystem issues affect choice of methodology, we are interested in the reverse affect: how the choice of methodology affects the communications channels in the ecosystem. In particular, we are examining the effect of methodology choice on team cohesion and product quality. Prior work has shown that the personalities of team members have an effect on team cohesion within XP[9]. We are studying how the methodology itself affects team cohesion and quality.

To begin this study, we designed an experiment involving students in two software engineering classes: Traditional Life Cycle (TLC) and Testing and Extreme Programming(XP). In these classes, the students work in teams developing a product for a customer. In the semester in question, we had one team in each class working on the same problem statement with the same customer. The teams consisted of 15 and 16 upper division computer science students, respectively. Throughout the semester, we administered surveys to observe the team cohesion over time. At the end of the semester, we compared the quality of the source code and product the students delivered.

TEAM COHESION

We are interested in cohesion because it is associated with success[4]. Within software engineering, cohesion leads to increased communication and knowledge sharing[10]. This should lead to improved product quality.

Before embarking on measuring team cohesion, it is useful to carefully define the concepts associated with cohesion. First, a “team” is not just a group of people. It is important that a team is a group of people that share an identity and a purpose (for the rest of this paper, the terms “group” and “team” refer to teams that meet this criteria). “Cohesion” is the degree to which the team sticks together as they pursue the team’s purpose. Clearly, software engineering teams meet our criteria and this definition of cohesion has value because it is the degree to which they work together and should reflect some aspect of their ability to succeed.

The amount of cohesion that a team exhibits changes over time[4]. Within a software engineering organization, cohesion can be affected by many things: personality of team members[9], length of time the team has been together, overcoming obstacles together, etc. Therefore, it is important to measure cohesion over time and the changes in cohesion may be more insightful than its actual value.
Within software engineering, cohesion can be thought of in two very different ways: the social attachment within the team and the team's connection to the project itself. Carron, et al.[4] called these social (S) and task (T) cohesion, respectively. In addition, we can think of these types of cohesion at two levels of granularity: at the individual level and for the team as a whole. Carron, et al.[4] called these Individual Attractions to the Group (ATG) and Group Integration (GI), respectively. Combining these two dimensions of cohesion results in four measures of aspects of team cohesion:

- GI-T: The team’s attachment to the task
- GI-S: The team’s social connection
- ATG-T: Individual attachment to the task
- ATG-S: Individual connection to the team

**Measures of Team Cohesion**

In order to increase our confidence in our results, we have used a number of tools to measure team cohesion. These were administered in the form of two surveys: Group Environment Questionnaire and a survey we developed.

**Group Environment Questionnaire**

Carron, et al.[3] believe that all four aspects of cohesion can be measured by measuring the team and individual perceptions of cohesion. They developed the Group Environment Questionnaire a survey that measures GI-T, GI-S, ATG-T, and ATG-S. This questionnaire was devised for sports teams and has been shown to highly correlate with perceptions of cohesion in that environment. Because some of the questions were specific to sports teams, we have modified those questions to reflect software engineering teams.

**Peer Rankings**

At five milestones in the projects, we administered a survey in which the students ranked their teammates in two ways. First, they ranked each team member affect on the team’s success on the following scale:

1: detrimental to project success
2: useless to project success
3: average impact on project success
4: above average impact on project success
5: critical to project success
N/O: No Opinion on effect

Second, they were asked, “If you were picking a team for a new project, which of your teammates would you like to include?” This question required a binary response for each teammate; they were not allowed the “No Opinion” option.

From these questions, we gathered a number of statistics following standard cohesion analysis experiments[1]. We analyzed the averages of the peer rankings in the first question and the number of peers being selected in the second question. These give us a measure of individual perceptions of the degree of “value” of each team member, but not a direct measure of how closely the team is connected.

To get a measure of the similarity in responses, we calculated interrator agreement[1] for ranking made for each team member. First, responses of N/O were given a ranking of zero and, for the \(i^{th}\) team member, the standard deviation of the rankings given to that team member was computed (\(s_i\)). The maximum value that this statistic can achieve is denoted \(s_{\text{max}}\). Interrator agreement for the team member is computed as follows:

\[ ira_i = \frac{s_{\text{max}} - s_i}{s_{\text{max}}} \]

This gives a measure of similarity of response about a particular individual. The range of this statistic is 0 to 1 and increases with the similarity of the rankings. This gives us a strong measure of the closeness of a team and can also be used within subsets of the team to measure their cohesion within the context of the team.

Finally, we looked for mutual relationships within the answers to the binary question. These are pairs of teammates who each selected the other to be on their team. The number of such relationships should give another perspective on the closeness and mutual respect within the team.

**Project Interest**

Our survey also asked two questions related to the interest the individual had in pursuing the project:

1. On a scale of 1 to 5, where 1 means “not at all” and 5 means “very much,” to what extent do you want to work on the next release of this project?
2. If it didn’t affect your grade in this or any of your other classes and if you could pick your team, how many more or less hours would you work on this project a week? Zero means no change in your workload.

We found that the second question was not insightful as the magnitude of the responses depended on the number of hours the students were currently working on the project. As we did not have reliable numbers for hours they were working (students are not good at measuring themselves), we did not include the results of this question in our analysis.

**Measures of Quality**

We were interested in two types of quality: source code/design quality and product quality. We measured source code quality with a number of standard code metrics: method length, cyclomatic complexity, nested block depth, number of parameters/method, number of attributes per class, and weighted methods/class.

We compared product quality by comparing the amount of functionality produced and the usability of the user interface. Because these are subjective measures, we had the students make these comparisons.
EXPERIMENT DESIGN

The students in this experiment were enrolled in two classes: TLC and XP. The TLC team consisted of 17 upper division students and used a variant of the Team Software Process (TSP) [8]. After completing a use case model for the requirements and a high-level architecture, the team was divided into three sub-teams of five or six members each corresponding to the main components of the architecture. The XP team consisted of 16 upper division students using most of the twelve XP basic practices[2]. After a two-week “spike” in XP techniques, we completed three four-week iterations. During each of the iterations, we played the Planning and Iteration Planning Games, the students used test-driven development and pair programming, and the iteration culminated with a presentation to the customer.

The students in both teams had roughly equivalent academic experience with the exception that some of the member of the XP team had completed the TLC course and therefore had experience in larger projects. Both teams were given the same problem statement and the same customer to control as many variables as possible. The students were aware of this experiment and were instructed not to talk about the projects with students in the other teams.

The GEQ was administered twice: midway through the project and at the end of the project. Our survey (including peer rankings, peer selection and project interest) was given five times through the project.

RESULTS

Social Cohesion Measures

The results of the GEQ survey are shown in Figure I.

![FIGURE I] GEQ RESULTS

In general, these results show very little difference between the individuals’ perceptions of the social cohesion of the team (GI-S) and the individuals’ connection to the team (ATG-S). Both teams show slight increases in individual connection to the team, but that change is probably not significant.

Table 1 and Table 2 show the results of the first and last ranking surveys of the TLC team. Each row of the table is the responses given by a student. The students are ordered by subsystem, so the boxes represent rankings given by students in the same subsystem. Initially, very few rankings are given to students outside of a subsystem. That number does not increase dramatically by the end of the semester, but the increases seen are related to the architecture of the software being developed.

The three subsystems are: rules (students A-G), GUI students (H-M), and structures (students N-R). The GUI layer interacts with both of the underlying layers, but the interface to the rules subsystem was much simpler than the interface of the structures subsystem. In addition, the structures sub-team considered their software to be a “service” layer and were very interested in determining what functionality would help the GUI layer. The rules subsystem provided an interface that they thought would suit anyone who wanted to use rules. This meant that the rules sub-team had very little interaction with the other sub-teams. By the fifth survey, the GUI and structures teams show much more interaction while the rule subsystem is relatively isolated. Members of other sub-teams are ranking members of the rule sub-team, but rules members are still not confident enough in the capabilities of other sub-team members to rank them. In fact, the rules people rank fewer outside people at the end of the project than they did at the completion of requirements analysis.

Figure II shows the average percent of teammates that each individual chose to rank. The XP team shows a significant increase in this statistic after the first survey with
little change from that point in the project. The first survey was given very early in the first iteration of the project, so it is not surprising that the students had not formed opinions about all of their colleagues, yet. By the end of the first iteration, the second point at which they were surveyed, they have opinions about virtually all of their colleagues. In addition, with the exception of a slight decline after the first survey, the average rank they gave their colleagues didn’t change throughout the project.

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R
A | 4 | 3 | 4 | 3 | 2 | 4 | 4 | 3 | 2 | 4 | 3 | 2 | 4 | 3 | 2 | 4 | 3 | 2
B | 5 | 4 | 5 | 4 | 3 | 4 | 5 | 4 | 3 | 4 | 5 | 4 | 3 | 4 | 5 | 4 | 3 | 4
C | 4 | 3 | 4 | 3 | 1 | 3 | 5 | 4 | 3 | 4 | 3 | 1 | 3 | 5 | 4 | 3 | 4 | 3
D | 4 | 3 | 4 | 3 | 2 | 3 | 4 | 3 | 2 | 3 | 4 | 3 | 2 | 3 | 4 | 3 | 2 | 3
E | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5
F | 5 | 4 | 5 | 4 | 3 | 4 | 5 | 4 | 3 | 4 | 5 | 4 | 3 | 4 | 5 | 4 | 3 | 4
G | 4 | 3 | 4 | 3 | 1 | 3 | 4 | 3 | 1 | 3 | 4 | 3 | 1 | 3 | 4 | 3 | 1 | 3
H | 3 | 4 | 4 | 3 | 4 | 3 | 4 | 4 | 3 | 4 | 3 | 4 | 4 | 3 | 4 | 4 | 3 | 4
I | 3 | 4 | 4 | 3 | 4 | 3 | 4 | 4 | 3 | 4 | 3 | 4 | 4 | 3 | 4 | 4 | 3 | 4
J | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4
K | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4
L | 3 | 4 | 4 | 3 | 3 | 4 | 4 | 4 | 3 | 3 | 4 | 4 | 3 | 3 | 4 | 4 | 3 | 3
M | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5
N | 4 | 3 | 4 | 4 | 4 | 3 | 3 | 4 | 3 | 4 | 3 | 4 | 3 | 4 | 3 | 4 | 3 | 4
O | 3 | 3 | 4 | 4 | 4 | 3 | 3 | 4 | 3 | 3 | 3 | 4 | 3 | 3 | 3 | 4 | 3 | 3
P | 4 | 4 | 5 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 4 | 4 | 4 | 4 | 5 | 5 | 5
Q | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 5 | 4 | 3 | 1 | 4 | 4 | 4 | 4 | 4 | 4
R | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 3 | 4 | 5 | 3 | 4 | 5 | 3 | 4 | 5 | 3

TABLE 2
TLC RANKING SURVEY NUMBER 5

The TLC team ranked a much smaller percentage of their colleagues throughout the semester. The colleagues they chose to rank initially came almost entirely from their subsystem. As the semester progressed, the number of colleagues an individual ranked that were outside of their subsystem increased, but, by the end of the semester, they were still unwilling to rank half of the members of the team. This shows that the subsystem organizational structure that resulted from dividing the design according to the architecture has limited the interaction between the engineers.

While the percent of teammates they rank is higher within a subsystem in the TLC team than the XP team, Figure III shows that the percent of colleagues they would select for a future team is roughly the same within a TLC subsystem as within the entire team in XP. In fact, a number of the selections made outside of a subsystem were the result of students who selected every teammate for each survey. When those individuals are eliminated, the percent of students outside the subsystem that get selected is less than 10%.

The rankings survey can also be used to find lack of cohesion. In particular, each team had individuals that were given both high and low rankings in a single administration of the survey. For example, one member of the XP team was given ranks of 5 (critical) and 2 (useless) by different teammates in the same administration of the survey. For every case, these individuals
became the source of conflict within the team. Those conflicts ranged from mild debates over design decisions to arguments over the general direction the team was taking. The XP class had a second team working on a different project. That team did not have individuals with conflicting rankings and did not have the level of conflict seen in these teams.

Attachment to the Project

In Figure I, both teams, at both times in question, scored more highly on the task-related measures of cohesion than on measures of social cohesion, and the TLC shows higher task cohesion than then XP team.

In fact, the TLC team showed an increase in GI-T. This implies that individuals perceived the team’s interest in the project as increasing. However, the GEQ measure of individual attachment to the project (ATG-T) in unchanged. Our question that directly asks how much they want to continue working on the project, shows a different picture: Figure V shows that individuals on both teams are losing interest as the project continues. The TLC team shows a slight increase in this interest in the fourth administration of the survey. This corresponds with the beginning of the implementation phase of the project and could be a result of the fact that students are more comfortable with implementation than previous phases of the project. It is interesting to note that the XP team showed a significant increase in interest at the same time. This corresponded with the initiation of the third iteration of their project. The anecdotal explanation for this came from the students. They were now comfortable with the process and were finding the project repetitive. Completing “yet another iteration” was less interesting than starting something new. This was somewhat relieved after the third iteration when the deliverable began to show significant functionality.

Productivity

The TLC developed 157 production LOC per engineer while the XP team only developed 88 production LOC per engineer. The TLC team also developed 145 test LOC per engineer while the XP team only developed 117 test LOC per engineer.

While it appears that the TLC team developed a much larger system than the XP teams, the students agreed that the amount of functionality delivered by both teams was about the same. This apparent inconsistency was explained by a thorough analysis of the TLC code. Even though they had carefully designed their system and that design included a well-structured class hierarchy, their implementation has put almost all of the functionality in the leaves of that hierarchy which resulted in a large amount of “copy and paste” code. In fact, the rules subsystem had implemented 32 rules and generalized none of their behaviors. They had detected this about two weeks before the end of the semester and had eliminated over 1000 LOC before these statistics were gathered. The lesson is that a good design is not always implemented well!

Source Code Quality

Table 3 shows the code metrics for the source code developed by the two teams. For each of these metrics, larger numbers imply less well-structured code.

Clearly, the XP team’s code has consistently better metrics. In fact, the metrics where the XP team’s code is worse than the TLC team’s code result from portions of the code that were written by a GUI builder. The XP team took refactoring very seriously. In fact, in the second and third iterations, they reserved about a third of their project velocity for major refactorings. The result was a well-designed system even though it was built incrementally.

The XP team developed more test LOC per production LOC, but not significantly. In fact, while they were supposed to be using strict test driven development, large portions of their code are not covered by the test cases they developed. They found it very difficult to develop tests for the GUI portions of the system and, while the technique spike included the humble dialog box design pattern[7], they did not understand it well enough to utilize it early in the semester.
Later, they had difficulty refactoring the system toward that pattern.

<table>
<thead>
<tr>
<th>TABLE 3</th>
<th>CODE METRIC COMPARISON</th>
<th>XP</th>
<th>TLC</th>
<th>max team</th>
<th>larger</th>
</tr>
</thead>
<tbody>
<tr>
<td>max method length</td>
<td></td>
<td>71</td>
<td>70</td>
<td>XP</td>
<td>1%</td>
</tr>
<tr>
<td>avg method length</td>
<td></td>
<td>6.23</td>
<td>8.193</td>
<td>TLC</td>
<td>31%</td>
</tr>
<tr>
<td>avg Cyclomatic Complexity</td>
<td></td>
<td>7</td>
<td>1.59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max Cyclomatic Complexity</td>
<td></td>
<td>2</td>
<td>1.84</td>
<td>TLC</td>
<td>16%</td>
</tr>
<tr>
<td>max nested block depth</td>
<td></td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max num parameters</td>
<td></td>
<td>4</td>
<td>5</td>
<td>TLC</td>
<td>25%</td>
</tr>
<tr>
<td>Max num attributes</td>
<td></td>
<td>5</td>
<td>7</td>
<td>TLC</td>
<td>40%</td>
</tr>
<tr>
<td>avg weighted methods per</td>
<td></td>
<td>12</td>
<td>23</td>
<td>TLC</td>
<td>92%</td>
</tr>
<tr>
<td>avg weighted methods per class</td>
<td></td>
<td>8.64</td>
<td>11.29</td>
<td></td>
<td>31%</td>
</tr>
<tr>
<td>avg weighted methods per class</td>
<td></td>
<td>3</td>
<td>5</td>
<td>TLC</td>
<td>4%</td>
</tr>
<tr>
<td>Max classes per packet</td>
<td></td>
<td>51</td>
<td>53</td>
<td>XP</td>
<td>5%</td>
</tr>
</tbody>
</table>

Product Usability

It is interesting to note the differences in the user interfaces developed by the two teams. At the end of the semester, both teams agreed that the TLC team, who had thought about the entire application during requirements analysis, had developed a much more consistent user interface. In the XP user interface, you could detect which features were associated with each iteration. They had a tendency to add a new frame for each new piece of functionality instead of modifying the existing capabilities of the system. This led to a very cumbersome user interface. The students concluded that they needed to “refactor” the user interface!

CONCLUSIONS

We have compared the experiences of students using two very different development methodologies. Within that comparison, we looked at team cohesion, attachment to the project, productivity, and source code quality.

Within our study, we can make a number of interesting conclusions. It is interesting to note that the TLC team showed equal or higher scores for every aspect of cohesion at both administrations of the GEQ survey. However, the study indicates significant lack of cohesion across subsystems in TLC, it is likely that those interfaces will have a higher defect rate than code within a subsystem. Organizationally, assigning individuals to coordinate activities between subsystems should increase inter-sub-system cohesion and begin to address this problem. However, our results show that XP’s philosophy of everyone owning the source code increases overall team cohesion.

From a class management perspective, our measurements of cohesion can be used to detect pockets of lack of cohesion that can predict potential conflicts. This could allow preemptory management of the situation.

While our teams developed a similar amount of functionality, the code developed by the XP team demonstrated consistently better design strategies as they refactored often and thoroughly. In the future, we would make an effort to ensure they understood the humble dialog box design pattern to improve their ability to follow test driven development. For the TLC class, in the future, we will know to watch to ensure that functionality is being generalized within the designed class hierarchy. A metric or tool that detected “copy and paste” code could help ensure that a good design is implemented appropriately.

REFERENCES

[11] Syed-Abdulla, Sharifah, Holcombe, Mike, and Gheorghe, Marian, Agile Methodology in Practice Memorandum 03-04, University of Sheffield Computer Science Department, 2