Submission to WESS in May 2000

Do Design Patterns Improve Communication? An Experiment with Pair Design

Barbara Unger and Walter F. Tichy (unger,tichy@ira.uka.de)
Fakultät für Informatik, Universität Karlsruhe
D-76128 Karlsruhe, Germany

Abstract

One of the main advantages claimed for software design patterns is improved team communication. This paper reports on an experiment that tests this hypothesis. Team communication among pairs of designers with and without design pattern knowledge is observed in a maintenance setting and analyzed by protocol analysis. The results indicate that shared pattern knowledge leads to a more condensed explanation phase and a balanced give-and-take among team members during design work.

1 Introduction

Design patterns as described by Gamma et al. [3], Buschmann et al. [2], and others are quite popular. They are thought to improve software maintenance and team communication. The effects of design patterns on maintenance have already been tested in a number of experiments by the authors [4, 5, 6, 7]. This paper reports on an experiment regarding team communication.

Members of two-person teams received a program design for maintenance. One member of each team was given time to study the design beforehand. In a first phase, this team member explained the design to his partner. Subsequently, each pair of programmers discussed how to design given requirements changes into the existing program. Protocol analysis of the interactions were performed before and after each team took part in a design patterns course. The results showed that without shared pattern knowledge, there is no pronounced explanation phase and the discussions were dominated by one individual (not necessarily the person who studied the design beforehand). After the design patterns course, there is a clearly identifiable explanation phase followed by a balanced give-and-take during the actual design work.

These results confirm what Buschmann et al. describe as follows: "Pattern-sharing establishes a common vocabulary for design problems. It allows members of the growing community to identify, name, and discuss both problems and solutions more effectively [2, page xii]". The net effect of improved communication could be faster work, fewer errors, and lessened architectural drift.

Our results also have implications for pair programming, a practice in which two programmers work side-by-side, continuously collaborating on the same design, implementation, or test [1]. Our observations suggest that balanced team work does not simply "happen", at least not in the short term. Instead, a common understanding of problems and solution patterns must be developed. This common understanding, at least in pair design, might be boosted by knowledge of design patterns.

2 Description of the Experiment

The experiment tests the following hypothesis: If team members have common design pattern knowledge and vocabulary, they can communicate more effectively than without.

The experiment compares teams with and without pattern knowledge communicating about program designs. Verbal communication is captured with audio and video and the protocols analyzed. Communication is considered more effective if there are clear episodes of explanation by one member and balanced discussions during design work.

2.1 Design

For evaluating the hypothesis one needs to observe subjects in situations in which they are talking freely about programs or program designs. There are several ways to observe talking subjects, but one has to be careful that the communication remains natural. Think-aloud protocols or telling answers to an advisor are not natural for team work. Instead, we chose to group subjects into two-person teams and recorded their interactions while performing a design maintenance task.

For eliciting an explanation phase, one subject of each team is given one hour of extra time in advance. During this time, this subject studies requirements and design documents of the program. This team member will be referred to as the *expert*. Then the second team member joins as *novice*. The expert explains the program design to the novice. Subsequently, both team members collaborate on two maintenance tasks. This strategy provides a chance for observing two interesting communication phases: (1) The phase of explanation by expert to novice, and (2) the teamwork phase between two maintainers.

The independent variable is design pattern knowledge. First, teams are given a pretest, then they participate in a three-month lab course on design patterns, and finally they receive a posttest. The course provides practical exercises for all patterns relevant for the tests (as well as other patterns).

The experiment employs two different applications. Teams are divided into two experimental groups working on different applications. The applications are switched from pre- to posttest. For fairness to the students, the roles of expert and novice are also switched. Table 1 summarizes the experiment design.

	pretest	**posttest**
	program1	program2
group1	person1 = expert	person2 = expert
	person2 = novice	person1 = novice
	program2	program1
group2	person1 = expert	person2 = expert
	person2 = novice	person1 = novice

Table 1: Design of the experiment.

2.2 Applications and tasks

Both applications used in the experiment are from domains that are easy to understand.[1]

Application ChiCo is a front end for version control systems such as RCS, RCE, or others. ChiCo also controls file access on a team basis. It consists of 15 classes and 76 methods and contains the patterns Observer, Bridge, and Singleton.[2] The first maintenance task is to enhance the flat team structure into a hierarchy. Users from superprojects also have access to files in subprojects. The second work task is to display all files that are checked out for a user and to delete checked-out, but unchanged, work files.

Application Timmie is a time and defect tracking system for managing software projects and teams. Is consists of 13 classes and 102 methods. It is designed with Composite, Visitor, Chain of Responsibility, Observer, and Singleton. Only the team management and time tracking is provided. The first maintenance task is to add the defect management. The subjects' task is to add recording and tracking of defects, notification of the appropriate developers, notification of the teams for defect logging, and an undo mechanism. The second work task involves adding time accounting and defect logs of private projects. Private projects are included in private statistics but not in the overall project reports.

2.3 Subjects, teams, and groups

The 15 subjects were computer science graduate students in their 7.8^{th} semester on average (median 8^{th} semester). On a preliminary questionnaire they were asked about their previous programming experience. On average they had 5.8 years of programming experience (median 5.5 years) and on a five level scale they had been programming on average between 3000 and 30000 LOC. In the pretest, three students had practical experience with design patterns relevant for the experiment. Theses cases are discussed separately in the results (see Section 3).

The subjects were allowed to choose their team members. In the pretest, the partners of only one team did not know each other. Due to subject loss (subject s13 did not participate in the lab course, subjects s11 and s12 did not have time for the posttest), one new posttest-only team was established (T_8: s14, s15), see Table 2. The team members of this team did not know each other. The teams were randomly divided into groups.

subject	team in pretest	program in pretest	team in posttest	program in posttest
s1	T_1	Timmie	T_1	ChiCo
s2				
s3	T_2	Timmie	T_2	ChiCo
s4				
s5	T_3	Timmie	T_3	ChiCo
s6				
s7	T_4	Timmie	T_4	ChiCo
s8				
s9	T_5	ChiCo	T_5	Timmie
s10				
s11	T_6	ChiCo	–	–
s12			–	–
s13	T_7	ChiCo	–	–
s14			T_8	Timmie
s15	–	–		

Table 2: Team composition in pretest and posttest. Due to subject loss, T_6 and T_7 do not appear in the posttest and T_8 is newly established.

2.4 Experiment conduct

The pretest was performed in April 1999 and the posttest in July and August 1999. The test sessions

[1]The original documents including the program designs and work tasks are available upon request.

[2]For definition of these patterns, see [3]

were scheduled individually. They took place in a room where the two subjects had a table large enough for all documents, a flip chart for graphical explanations, chairs, paper and pencils. A video camera was set up in one corner of the room and an audio recorder was placed on the table. The experimenter was present throughout all tests for questions, but sat with the back to the team and worked on different things, in order not to interfere.

2.5 Threats to validity

This experiment is not a *controlled experiment* because several important variables were not controlled. Chief among these is learning effect. Subjects could have learned additional communication skills during the pretest. The subjects might have been unexperienced in team discussions and might therefore perform better in the posttest. There are two indicators that minimize this concern. First, the expert of team T_8 did not participate in the pretest, so was not exposed to this learning environment. Yet, the results of T_8 in the posttest are consistent with other teams. Second, most team members knew each other before, so there should be no effects from getting to know each other. One team, T_3, had worked together before for some time and had pattern knowledge. This team displayed the same characteristics in the posttest as the other teams, though not in the pretest.

There is clear evidence that there were large differences in interpersonal communication skills. Some teams worked together well while others performed poorly. Changing the expert/novice roles could influence team performance from pretest to posttest. It is unclear if and how this threat to internal validity biases the results.

External validity concerns are about the small program and task sizes, small team sizes, and the experience of the subjects.
Program and task size: In real situations, programs and tasks are more complex. But if there is a communication improvement it might help even more on complex tasks.
Team size: Larger team sizes increase the required communication so if there is a communication improvement in small teams the improvement is expected to help in larger teams.
Experience: The programming experience varied from small student exercises to students with their own software company. But in all teams except T_3, we observed similar communication improvements.

3 Results

One of the authors transcribed the audio and video protocols into written protocols and then coded them with a simple coding scheme. The coding scheme identifies the speaker and the type of utterance. See Table 3 for utterance types. Especially important are atomic assertions about the designs because they carry crucial design information. Sometimes answers are mixed with assertions ("Yes, because this is a visitor"). In this case the assertion is counted in addition to the question.

For graphical analysis of the coded protocols, each assertion by the expert is counted as +1 and each assertion by the novice as -1. A moving average over 70 data points is computed. The plot of the moving averages is called a *communication line.* If the contributions of the team members are equal, the moving average should be 0.

utterance type	example
atomic assertion	This variable counts the number of developers.
question	What is this variable for?
answer	yes, no
feedback	Ok, I understand.

Table 3: Types of talking and examples.

The communication lines appear in Figures 1 to 7.[3] In the pretest, all communication lines are heavily unbalanced (except for T_3), while most of the communication lines in the posttest exhibit a "hump" of explanation at the beginning and then become balanced around 0.

3.1 Results of the Pretest

We can categorize the communication lines from the pretest into two types of unbalanced communications: *expert dominated* and *novice dominated.* Expert dominance is clearly observable in teams T_4 and T_5 and partly in T_3, while the novice dominates in the others.

Expert dominance: The entire conversation is dominated by the expert. Either the novice does not gain enough knowledge from the explanation or he does not rely on this knowledge in the design task. Both situations mean that the explanation phase is ineffective.

Novice dominance: The novice takes over the conversation almost immediately. Two common situations occur: The novice generates his own hypotheses and confirms them on his own or asks the expert for confirmation. There is almost no explanation phase. The novice does not even listen to the expert. Again this means that the explanation phase is ineffective.

What is not obvious from the diagrams is that two teams did not complete their maintenance tasks. Team T_1 was stopped after the first work task because they didn't get the point of the task. The novice did not rely on the expert at all and often tried to get ad hoc information from the documents. He missed major points of

[3]Team T_7 was not evaluated because this was a foreign language pair and they sometimes switched to Russian.

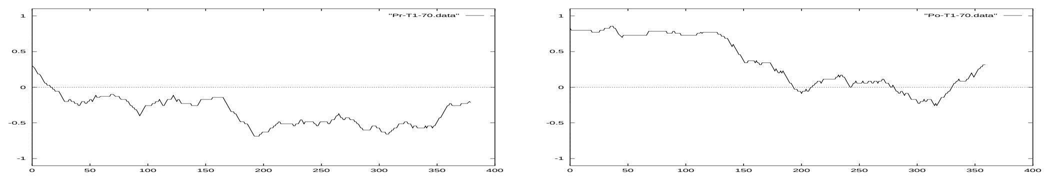

Figure 1: Communication lines of Team T_1. Left: Pretest. Right: Posttest.

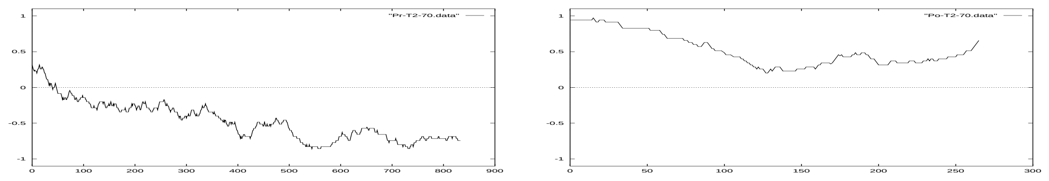

Figure 2: Communication lines of Team T_2. Left: Pretest. Right: Posttest.

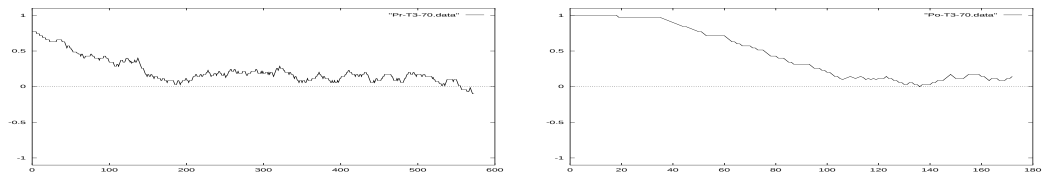

Figure 3: Communication lines of Team T_3. Left: Pretest. Right: Posttest.

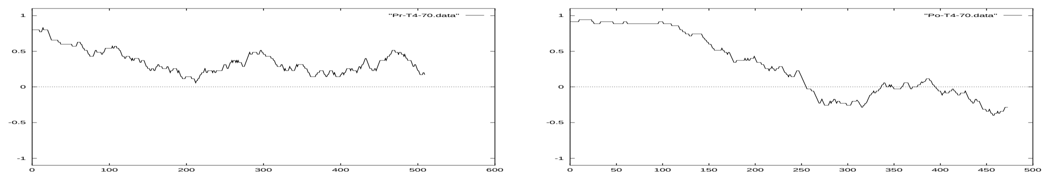

Figure 4: Communication lines of Team T_4. Left: Pretest. Right: Posttest.

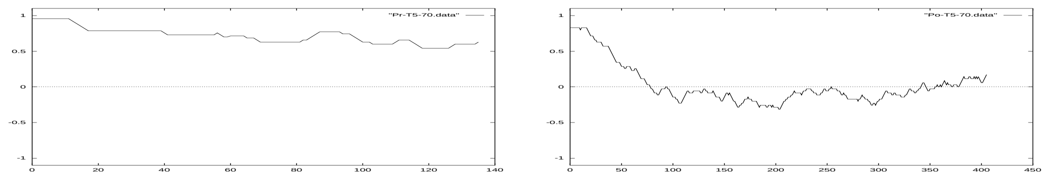

Figure 5: Communication lines of Team T_5. Left: Pretest. Right: Posttest.

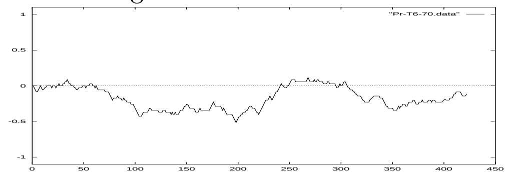

Figure 6: Communication line of Team T_6 in pretest.

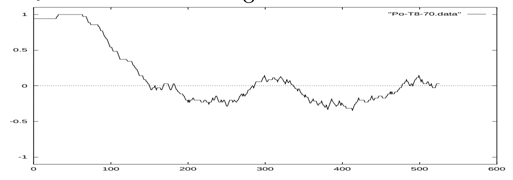

Figure 7: Communication line of Team T_8 in posttest.

the design. Team T_2 also stopped after the first work task because team members were too exhausted. They had worked for more than three hours. Others took only about 1 1/2 hours for both tasks. In this team the expert provided a lot of wrong information.

Team T_6 suffered from an uncooperative novice. He dominated the conversation even though he was aware that he did not understand the program. He frequently tried to break off and go home.

3.2 Results of the Posttest

In all posttest graphs, there is a clear explanation "hump" at the beginning followed by a balanced teamwork phase. The explanation phase is close to 1 and the teamwork phase alternates around 0. Even team T_8, whose expert did not participate in pretest, shows this behavior.

Especially interesting is team T_3. As mentioned, this team has extensive experience with team work and patterns. The explanation "hump" is already visible in the pretest, but in the posttest this "hump" is longer than the others and for about a fourth of the time it is perfectly 1.

One outlier in the posttest is T_2. This team's communication line is still unbalanced. The novice of this team was too tired for the experiment after having a long party the night before. This data can be ignored.

3.3 Implications

Comparing pretest and posttest, clear differences in the communication lines are observable. First, consider the teams with expert dominance in the pretest. All the communication lines are balanced in the posttest. Recall that the roles of the team members were switched from pre- to posttest. This behavior could be an indication that the pretest-novice is not as knowledgeable as the pretest-expert. Switching the roles, the weaker person can compensate by studying the materials beforehand, resulting in sufficient knowledge and confidence. So the balanced work phase in this case does not support the hypothesis.

Now consider the teams with novice dominance during pretest, i.e. teams T_1, T_2, T_6. In the posttest, after the explanation "hump", the dominating pretest-novice does not dominate the communication in the posttest **even though he now is the expert**. The implication is that the communication ability of the posttest novice, who was a weak pretest expert, must have increased. Reasons may be that either the weaker pretest-expert has now more confidence in his own knowledge gathered from the explanations, or the pretest-novice relies more heavily on the knowledge of his team member. The authors believe that a major part of the improvements are due to shared pattern knowledge.

The observed behaviors show that the communication abilities improved from pre- to posttest. Due to lack of control in this experiment, it is unclear how much learning effects besides the design pattern knowledge the lab course introduced.

4 Conclusion

The results show clear evidence that the communication lines of the design teams changed from pre- to posttest. Without shared pattern knowledge, the explanation phase is small or non existent, and the following communication is dominated by one individual. After a three month design pattern course a clearly identifiable explanation phase is visible followed by a work phase with balanced communication. These observations support the hypothesis that team members can communicate more effectively with design pattern knowledge.

As an implication for team design tasks, especially for pair programming, our observations suggest that collaborative team behavior not simply happen by chance. Instead a common understanding of problems and solutions must be developed. Design patterns can help established shared understanding for design work.

5 Acknowledgments

Thanks to Anneliese von Mayrhauser for helpful and fruitful discussions and to all our students who participated in this experiment.

References

[1] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison Wesley, Reading, PA, 1999.

[2] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stahl. *Pattern-oriented Software Architecture: A Systems of Pattern*. John Wiley and Sons, West Sussex PO19 1UD, England, 1996.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Professional Computing. Addison-Wesley, 1995 (deutsch 1996).

[4] Lutz Prechelt. An experiment on the usefulness of design patterns: Detailed description and evaluation. Technical Report 9/1997, Fakult"at f"ur Informatik, Universit"at Karlsruhe, D-76128 Karlsruhe, http://wwwipd.ira.uka.de/ prechelt/Biblio/, June 1997.

[5] Lutz Prechelt and Barbara Unger. A series of controlled experiments on design patterns: Methodology and results. In *Proc. Softwaretechnik '98*, volume 18 of *Softwaretechnik-Trends*, pages 53–60, Paderborn, Germany, August 1998. GI.

[6] Lutz Prechelt, Barbara Unger, and Michael Philippsen. Documenting design patterns in code eases program maintenance. In *Proceedings ICSE Workshop on Process Modeling and Empirical Studies of Software Evolution*, pages 72–76, Boston, MA, May 1997.

[7] Lutz Prechelt, Barbara Unger, and Douglas C. Schmidt. Replication of the first controlled experiment on the usefulness of design patters: Detailed description and evaluation. Technical Report wucs-97-34, Department of Computer Science, Washington University, St. Louis, MO 63130-4899, http://www.cs.wustl.edu/cs/techreports/1997, December 1997.