Communication Strategies for Mentoring in Software Development Projects

Shreya Kumar
Michigan Technological University
1400 Townsend Drive
Houghton MI 49931 USA
+1 906 487 2209
ShreyaK@mtu.edu

Charles Wallace
Michigan Technological University
1400 Townsend Drive
Houghton MI 49931 USA
+1 906 487 3431
Wallace@mtu.edu

ABSTRACT

As with professionals in all engineering disciplines, software developers new to a project must be given the implicit and explicit knowledge they need to be productive, in an effective and appropriate way, due to fluid team dynamics, geographical distribution, and other factors. As part of a broader study of communication in software development, we focus here on communication strategies for mentoring. We explore some examples of mentoring-oriented communication, in an educational setting and in an open-source consortium of academics and professionals. We plan to draw out recurring patterns of communication between mentors and protégés.

Categories and Subject Descriptors

K.6.31 [**Project and People Management**]

General Terms

Management

Keywords

Communication Patterns, Mentoring, Software Project Communication.

1. INTRODUCTION

In collaborative creative endeavors like software development, newcomers must be brought up to speed not only on matters of fact but on deeper issues of rationale and motivation. The concept of mentor – the experienced guide, conveying knowledge and "know-how" to the protégé – is a time-honored tradition in management. Whether through established, codified practices (e.g. explicit mentoring initiatives by professional engineering organizations [1]) or the more implicit processes captured in legitimate peripheral participation [2], mentors provide instruction, counseling and interaction to impart understanding in a way that "reading the manual" (or the source code) cannot.

Software development, however, occupies a unique position in this space, due to its innately fluid and fast-changing nature. Software teams are formed and reformed at a rapid pace, in response to evolving requirements, business alliances, and personnel changes. Moreover, the flexibility afforded by software development, exemplified most vividly by open-source projects [3], allows theoretically limitless numbers of collaborators, problematizing the notion of team altogether. In this context, the concept of mentor must be expanded beyond its customary definition. Mentoring relationships may be ad hoc and transitory, with little or no clear delineation between those eligible for mentor status and those seeking mentorship. Begel and Simon [4] discuss the importance, advantages and challenges of mentoring for novices in the software industry.

Several scholars have identified communication as a central aspect of the mentoring process. Beyond the "simple exchange of information and accomplishment of ability" which is the primary goal of mentoring, Kalbfleisch [5] likens the process of establishing a mentoring relationship to "the initiation of friendships and love relationships in terms of communicating appropriate relational expectations". Buell [6] expands on this idea by categorizing mentoring relationships in terms of cloning, nurturing, friendship and apprenticeship, and noting the importance of "turning points" where the nature of the mentoring relationship changes [7].

In this paper, we explore the communication choices that developers make as they initiate and conduct mentoring activities. Our study samples include a student software development project, with regular face-to-face interactions with a client/mentor, and a globally distributed open source development project that primarily communicates via email. We apply our notion of communication patterns [8, 9] to characterize mentoring activities, employing Buell's mentoring models [6] as a guide.

2. COMMUNICATION PATTERNS

Earlier we have introduced the notion of a communication pattern language, both as a tool to study project communication and as a means of representing repeatable communication practices [8, 9]. Our motivation has been driven by pedagogical concerns: we seek to expose the complexities of project communication to budding software engineers, and to equip them the means to think analytically about their communication choices. In this paper, we discuss the strategies related to mentoring that we discovered in different types of software projects. As we observe occurrences of the same strategies more reliably in different types of projects, we will establish them as patterns.

As Alexander explains in his seminal paper on software design patterns, "[e]ach pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." [10] Patterns denote the essence of solutions to problems without overspecifying them - a key advantage when dealing with the fluid nature of the "softer" aspects of software development.

CHASE'14, June 2 – June 3, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2860-9/14/06...$15.00
http://dx.doi.org/10.1145/2593702.2593712

Coplien and Harrison take advantage of this in their work on organizational patterns. A pattern language approach has been used to analyze the recurrences of configurations of roles in software organizations, "using patterns in a generative way" [11].

Each communication pattern describes a set of properties associated with a communicative act. These properties, inspired by the classic "Kipling questions" (Who, What, Where, When, Why and How) include: power differential and roles between participants (who); synchrony and frequency of communication (when); presence of physical or virtual "subject matter" or tools (what); medium of communication (how). Patterns may define particular *genres* of communication (for instance, client demo, requirements gathering session, burndown chart), but others describe properties that cut across genre (for instance, given the "brainstorming session" genre, whether to perform the act synchronously with a facilitator, synchronously with all participants acting collectively, or asynchronously with participants providing input independently). In this way, patterns can be overlaid on one another, and a single communication act can be the combination of multiple patterns. Here we examine how different mentoring relationships employ different strategies of communication and how the strategies are affected by the project context (for instance, the medium of communication).

Our data on student projects comes from earlier ethnographic studies conducted at our institution [12, 13]. We use a grounded theory approach [14] to identifying patterns in our project data, conducting open coding on our first pass, identifying the instances of mentoring-related communication that emerge, then selecting for strategies used in these mentoring activities. In our current focus on mentoring, a "turning point" in the mentoring relationship is met with a change in communication strategy.

3. STUDENT MENTORING

In this section, we describe our observations of the "Nurturing model" [6] of mentoring and the communication acts and strategies associated with it in two student software projects, where the communication context was characterized by face to face interaction and accessibility to the client/expert/mentor.

Our two student software projects were each a semester long each and consisted of a team of three software engineering students working on a US Navy-sponsored project named "Seabase". The project centered on development of a controller for a ship-mounted crane and involved conversion of some legacy code. The client was "Hank", a professor in the mechanical engineering department who originally developed some of the legacy code. With fresh, inexperienced teams and a short project duration, it is difficult to establish repeatable practices for project work. Students did however have the benefits of physical colocation and a readily available and involved client.

In the two student projects – Seabase I and Seabase II, we witness the "Nurturing Model" [6] of mentor relationship where the mentor facilitates an environment for the protégé to learn and provides help and encouragement with guidance, as opposed to the "Cloning Model" where the mentor issues commands to be followed. So how does the "Nurturing" relationship manifest itself in communication strategies? We see two strategies – an initial *Mentor as Interrogator* strategy and a more mature *Mentor as Oracle* strategy, supported by a variety of communication tactics.

Mentor as Interrogator: The classic view of mentor, as illustrated memorably in Socratic dialogue, is mentor as asker of questions, carefully chosen to reveal gaps in knowledge or provoke awareness among protégés. In this pattern, the interrogation is typically followed by advice or sharing of strategies to overcome the identified gap. In his exchange with Seabase II team member "Bob" in Table 1, Hank is trying to determine Bob's plan for soliciting information from another student team. In fact, for the first few weeks of the project, most of the team's meetings with Hank followed this pattern interspersed with giving advice and taking progress updates. Interestingly, Hank's mentoring in this exchange is encouraging Bob to think strategically about his upcoming communication with the team.

TABLE 1. **MENTOR AS INTERROGATOR**

Seabase II – Week 4 (Bob expresses that he might meet a different student team that the SE team need data from)
Hank: What do you hope to get out of that meeting?
Bob: See how they are testing code, and how they are using Simulink
Hank: The reason to have that meeting with them was to understand their system, right? Specifically.
Bob: Trying to see what values they are using for Simulink
Hank: Like sensors...
Bob: Yes, like sensors, values for testing
Hank: So what you might want before that meeting is the things you need, like the sensors list that you would need. You won't have a list and they won't have a list, but since you folks have the diagram for that block. Do you know from that what values you will need?
[They look at the diagram and discuss some input parameters like sway, swing angle, hoist, lock, etc.]
Hank: It might be a good idea to have this picture when you talk to them.
(Bob and Hank together make a list of the values)
Hank: Ok, here is a low-level question. How do you want to go about it when you meet them then? You are at the interface, you provide them the list of things you need. Keep in mind that they are a senior design team just like the platform team. It would be good if you have that dialogue with them. I guess what I am trying to say is that you might not get a quick answer. They should know, but they might not.

Mentor as Oracle: This is the strategy of learning in the presence of a mentor with the protégé posing questions and the mentor answering them. The Seabase II software engineering team, when tasked with learning MATLAB, spent some time trying to learn from their client and technical expert, Hank's directions and reference material, but solicited Hank's time to ask specific questions about the language and platform where Hank (in the role of the mentor) resolved their queries through demonstration.

In Week 7 of Seabase II, Denise, a leading member of the SE team, meets Hank to learn MATLAB, which was a project requirement. After going through the reference material and code examples that Hank shared, the team is having trouble with a specific portion of the code regarding the damping mode block. Hank suggests using one block to calculate the damping values instead. The team watches Hank work on his computer as he demonstrates how some flags are being set in different S-functions of the code. He then tells the team they can choose to use whichever method they like best. Denise then asks about the placement and detection of logical breaks in the code to structure it better and Hank makes suggestions to structure the code better.

We observe a use of the *Over The Shoulder Learning* [15] tactic in the context of the workplace where the team observes Hank as he works step by step through examples of flag setting in the code. This is easily implemented in the face-to-face synchronous setting available to Seabase II.

In Seabase II, we observe an unmistakable "turning point" in Hank's relationship with the team, from the "Nurturing Model" to the "Friendship Model"[6]. In Buell's conception, the Friendship Model is characterized by "collaborative, reciprocal, mutual engagement" and weak or nonexistent hierarchy in the mentor-protégé relationship.

The turning point in Seabase II occurs during a meeting where team member Denise brings an elaborate hand drawn chart to depict data dependencies between blocks of the original legacy code. The chart was something Hank had repeatedly requested of the Seabase I team and finally found to his satisfaction with the Seabase II team. The hand-drawn chart plays a crucial role in demonstrating commitment to the client. Interestingly, the chart originated as a pedagogical tool for Denise, helping her to "get her head around" the legacy code. As such, it is messy and difficult for other readers to understand; however, Denise takes advantage of the synchronous, face-to-face communication with the client to "talk him through" the document, thereby mitigating any confusion caused by its hand-drawn nature. We call this tactic *Artifact Facilitated Discussion*.

TABLE 2. ARTIFACT FACILITATED DISCUSSION – MENTOR AS INTERLOCUTOR

Seabase II – Week 6 – [Denise and Hank are looking at her chart together.] Denise: This is where we need some help. So this is what happens in the code [pointing at Denise's chart] [Denise explains on her chart that she has color coded based on which blocks are her responsibility and how the chart describes the blocks.] Hank: Can you show me some example within the code? This is great. Don't throw this out. Is this hand-drawn?
Week 9 - [Hank and Denise are looking at the chart and Denise is explaining how init runs and affects other S-functions. Hank asks what some of the functions do, especially init. Denise explains the purpose] Hank: Oh that is sweet! That makes sense now. So when this one is high, that value becomes high and this one goes low, that value is low. I finally get it.. what is setup? [Denise explains what setup is.] Hank: I love it. I love it! The beauty of something like this is that I can understand it. Someone with a high level of knowledge of how the code or the function works can look at it and completely understand it.

Artifact Facilitated Discussion: This tactic is observed when the presence of an artifact, like a diagram, or piece of code, or design document becomes the center of discussion and facilitates and captures the understanding of the participants. It is associated with communication situations where participants have large gaps in their shared knowledge, where the problem of articulating the question and discovering the right question to ask is difficult.

It is typically found in a synchronous communication setting, so participants can confirm understanding with each other through the "catalyst" of the artifact. This is also an example of incidental learning [16] in the presence of an artifact - the hand drawn chart that Denise made to trace flow of code module dependencies. In this pattern, the presence of the artifact allows for more questions to be asked and promotes collaborative learning. The ability to point at places on the artifact to explain or better ground one's questions is valuable.

Mentor as Interlocutor: Denise's chart initiates a turning point in Hank's relationship with Denise, toward a "Friendship Model" of mentoring. In communication terms, a new strategy emerges – one in which questions arise from both the mentor and protégé and they play off each to share knowledge. Most meetings from the Artifact Facilitated Discussion onwards followed this strategy, where the team, implicitly led by Denise would brainstorm with Hank about strategies for arriving at solutions to identified obstacles.

4. OPEN SOURCE MENTORING

In our student project case studies, we find that the student teams clearly benefit from the physical colocation of mentor and team and frequent synchronous communication – factors that facilitate more traditional mentoring approaches. How do mentoring strategies change when this easy access to communication is not available? In this section, we describe the mentoring strategies we observed in an open source software project where the communication landscape was drastically different. It allows us to study similar mentoring strategies in contrasting contexts to appreciate the essential attributes of the mentoring strategies that work for different situations.

We are currently studying an open source visualization software project with developers distributed geographically (primarily in Europe and South America) and varying in their levels of experience and of commitment to the project. Communication is conducted almost exclusively on a common list serve. The project has an implicit core group of programmers, who often take on an implied mentoring stance for the "newcomers".

We observe the same mentoring models as the student project – Nurturing and Friendship. The Nurturing model is seen typically between the experienced programmers and the newcomers and the Friendship model exists between the core programmer group. We witness turning points where novice programmers become experts and switch to a mentor role from that of a protégé. However, in this asynchronous medium of communication, we focus on how these mentoring models translate into communication patterns.

We also observe the *Mentor as Interrogator* and *Mentor as Oracle* mentoring strategies very often, where when a new or less experienced programmer would pose a question, typically the host would acknowledge it, start with appreciation and encouragement and then pose questions to arrive at the core of the issue. When satisfied that the question is valid and properly articulated, the host would typically answer with a solution along with advice, often with code detail and steps to follow.

Email is a less than ideal, asynchronous form of communication and the project members have to use it even for interaction that is typically conducted face to face. In the student projects, an important recurring communication strategy was the *Artifact Facilitated Discussion*, which is impossible in the open source project as the project members are geographically distributed and spread across different time zones. We examine how they cope to still facilitate incidental learning.

The participants are typically proficient programmers, many of whom are well versed with the library they are working on with many years of experience, code as part of the email body very frequently becomes part of the conversation. On a closer, more qualitative look, we observe the *Code As Conversation* pattern, where participants on the forum ask a question related to the code and paste a code snippet in their email. In turn, respondents also use code in their email to share or propose solutions, along with some text as explanation. This exchange is often used to arrive at implementation strategies, make design decisions or even to debug code together.

TABLE 3. CODE AS CONVERSATON

("Novice" replies with corrections) thank you for the navigation. There is the script: <script code pasted in email>
("Host" appreciates the script) Thanks! Now, it will be easier to review :)
("Host" critiques and guides "novice" gently towards other solutions) Hi, I looked at it a bit. It's a start, but I think the direction is not quite right yet. Let's take a look at one of your examples: <code example here> I like that you are using a matrix model. But, what is not so clean is mixing shapes and elements. Right now, you are creating elements within the definition of the shape (i.e., #instVarNames). A rule of thumb should be that shapes should be interchangeable. Consider the following Mondrian example: <code example here>

In Table 3, we observe a combination of the *Code As Conversation* pattern and the *Mentor as Oracle and Interrogator* mentoring strategies. We share an excerpt from an email thread started by a programmer with comparably less experience on the project than some of the senior team members. The "novice" programmer wants the "host" to review his code implementation. The "host" starts with encouragement and shares the correct procedure for collaboration. When the "novice" creates the correct script, the "host" programmer informs him that he will examine his code, he soon replies with some comments – he talks to him about the direction of the solution and uses the novice's code example to illustrate what should be different. He then shares his own code example to demonstrate how to accomplish what the "novice" was attempting.

Although the mentoring models observed in both the student and open source projects have similarities, the change of medium to email only affects the tactics. When tone is not easily conveyed and the ability to point at a collectively viewed artifact is missing, we see that both the "host" and "novice" carefully craft responses with lots of information included, sometimes step by step directions to overcome the lack of a face to face interaction, where even partially articulated questions accompanied with gestures and pointing convey one's meaning. Pointing at common code is replaced by copying and pasting code fragments. We see that "Over The Shoulder" learning is not possible but "incidental learning" assisted by the code fragments takes place.

Finally, we note that protégés eventually become mentors as an example of the "turning point" where we see a "novice" programmer used to pose questions to the forum very frequently accompanied by statements like "I am very confused" and "I am sure this is a stupid question". Over the years, the "novice" has turned into an "expert" where we notice him responding to and encouraging "novices" with statements like "It is great that you are working on… and let us know if you need any help".

5. FUTURE WORK

Our initial work on these project case studies has elicited instances of strategies and tactics; as we examine a broader base of projects, we expect to discover more instances of similar communication activities, thus allowing us to establish true patterns of communication. We also plan to develop a method to assess mentoring strategies. Objectively rating a communication act or strategy is difficult, as communication contexts are varied and the notion of success or failure of communication strategies is complex. However, the structure of our patterns do provide a rubric for determining fitness to a particular context, by matching the attributes of the communication pattern with attributes of the communication context.

6. REFERENCES

[1] National Society of Professional Engineers, 2002, *Mentoring Guide for Small, Medium, and Large Firms.*

[2] Lave, J., Wenger, E., Situated learning: Legitimate peripheral participation, Cambridge University Press, 1991.

[3] Yu, Y., Benlian, A., Hess, T., 2012, *An Empirical Study of Volunteer Members' Perceived Turnover in Open Source Software Projects*, Hawaii International Conference on System Science (HICSS), IEEE, 3396-3405.

[4] Begel, A., Simon, B., 2008, *Novice software developers, all over again*, International Workshop on Computing Education Research, ACM, 3-14.

[5] Kalbfleisch, P. J., 2002, *Communicating in mentoring relationships: A theory for enactment*, in Communication Theory 12, 63-69.

[6] Buell, C., 2004, *Models of mentoring in communication*, in Communication Education 53.

[7] Bullis, C., Bach, B. W., 1989, *Are mentor relationships helping organizations? An exploration of developing mentee-mentor-organizational identifications using turning point analysis*, in Communication Quarterly 37, 199-213.

[8] Kumar, S., Wallace, C., 2013, *A tale of two projects: A pattern based comparison of communication strategies in student software development*, Frontiers in Education Conference, IEEE, pp. 1844-1850.

[9] Wallace, C., Kumar, S., 2013, *Communication patterns: a tool for analyzing communication in emerging computer science educational practices*, ACM Technical Symposium on Computer Science Education, ACM, pp. 729-729.

[10] Alexander, C., The timeless way of building, Oxford Univ. Press, New York, 1977.

[11] Coplien, J. O., Harrison, N. B., Organizational Patterns of Agile Software Development, Prentice-Hall, Inc., 2004.

[12] Brady, A., Seigel, M., Vosecky, T., Wallace, C., 2008, *Speaking of Software: Case Studies in Software Communication*, in Software Engineering: Effective Teaching and Learning Approaches and Practices.

[13] Vosecky, T., Seigel, M., Wallace, C., 2010, *Making and Acting*, in Qualitative Research in Technical Communication, 276.

[14] Glaser, B. G., Strauss, A. L., The discovery of grounded theory: Strategies for qualitative research, Transaction Books, 2009.

[15] Twidale, M. B., 2005, *Over the shoulder learning: supporting brief informal learning*, in Computer Supported Cooperative Work 14, 505-547.

[16] Cook, C. R., Scholtz, J. C., Spohrer, J. C., 1993, *Empirical Studies of Programmers: Fifth Workshop*. Norwood (NJ): Ablex Publishing Corporation.