

Chapter 4: Studies of PBS Users

4.1 Overview

Although a number of Software Bookshelves had been constructed, they had not been evaluated as a software comprehension tool. A Software Bookshelf had been constructed for a commercial software product, and usability studies were conducted with the team maintaining the product. The intention was to evaluate how PBS helped them with their daily work, but for myriad reasons it was difficult to find PBS users. Many of the reasons were organizational, but some were due to shortcomings in the tool. The studies of PBS users, their findings, and implications for PBS are documented in this chapter.

4.2 The Software Project

A PBS had been populated for a compiler maintained by a team at IBM Canada Ltd. in anticipation of a migration to C++ from a PL/I dialect. The software was originally created fifteen years ago and has had twelve major releases. It consisted of approximately 250 000 lines of code in 1000 files. The system was supported by a team of ten developers.

Source code, some documentation, and Software Landscapes had been put onto the bookshelf. The system's fundamental abstract data structure and a small number of subsystems were well documented, but aside from the Software Landscapes there was little on the remainder of the system.

The Software Bookshelf was designed with three groups of users in mind: newcomers to a project who needed to learn about the system, project experts who required a reference on the software, and project managers who wanted to track the maintenance effort. Ideally, detailed studies should have been performed on each of these user groups, but this was not possible due to organizational constraints.

The newcomers, or software immigrants, were relatively easy to study, since they did not yet have a lot of responsibilities and consequently had time available to spend with a researcher. As a result, a detailed, formal study was conducted with them using a method that was sociological in origin. They were expected to use PBS a great deal, but they did not for the reasons that are presented in Section 4.3.3. What started out as a study of tool use, concluded in a broader characterization of the acclimation process for new employees. Presented in this chapter is a subset of the results already published elsewhere [Sim98a].

The project veterans tended to be quite busy, and were only available for occasional meetings. The findings on this user group are based on informal meetings and conversations. The project manager, while eager to help, was probably the busiest person on the team, and as a result there are no findings for this user “group”. In this case, missing a single individual, unfortunately resulted in the omission of an entire class of users. Nonetheless, valuable lessons were learned from studies of the other two user groups.

4.3 Software Immigrants

New staff members are usually experienced programmers who already have a rich set of skills and background knowledge. Despite their personal assets, they often lack basic knowledge about the specific project. For these reasons, we call these new team members “software immigrants”, since their experience is analogous to those of people who arrive in a new land and need to learn its language and culture. Software immigrants are often referred to by other terms such as newcomers, newbies, recruits, new hires, rookies, and even “fresh blood”. Novice is an inappropriate term since it implies a lack of experience. Extending this analogy, the process by which software immigrants adapt to a new project is called “naturalization”. Others may call it acclimation, re-tooling, start-up, ramp-up or bringing someone up to speed.

Studies have been undertaken in software engineering and cognitive psychology on working with legacy systems. There are some experience reports which give practical advice for working on undocumented software systems[Fay85, Pigosk93] and anecdotes from practitioners and consultants[Brooks95, DeMarc87]. The most significant contribution comes

from Berlin [Berlin93] who studied the interaction patterns between mentors and apprentices at the conversation level and found that mentoring is a highly effective way to transmit information about the system. Mentors provide not only answers to apprentices' questions, but also explanations of design rationale. Their conversations tend to be highly interactive in nature, using techniques such as confirmation and re-statement to verify that a message has been passed correctly. While mentoring has its merits, it tends to be a time-inefficient method to train a software immigrant because it results in a net decrease in team productivity in the short term. As an antidote, Berlin suggests capturing the information that mentors convey in documentation or an intensive course for apprentices.

4.3.1 Method

A multi-case study was performed with four respondents, all immigrating into a single team. By interviewing subjects, we hoped to identify commonalities and differences in their experiences, and to infer naturalization patterns from this comparison.

In this study, our goals were to:

- describe the naturalization process,
- identify shortcomings and successes of the process, and
- characterize the strategies software immigrants used to adapt to the new job.

In order to highlight areas that would profit from modification or improvement, we must first identify strengths and weaknesses in the existing naturalization process.

The unit of analysis in this study is a single naturalization. The rationale for this choice is that each participant could be studied more than once as they naturalized to a different projects and teams. Data was collected using structured interviews, and was analyzed using qualitative data analysis methods. During analysis, variables of interest were identified using a pattern matching technique. A data matrix was populated with these variables to articulate cross-case patterns [Miles94]. In the following three subsections, we describe the data collection procedure, and the data analysis techniques that were used.

4.3.1.1 Data Collection

Interviews were conducted from February 1997 to June 1997 with four respondents. Data collection began with S1 and S2 shortly after they joined the company. As the study proceeded, S3 and S4 were identified as relatively new software immigrants, and were willing to participate in the study. Consequently, using “controlled opportunism” [Eisenh89], they were interviewed using a sub-set of the questions used with the first two respondents. At the time of interview, S4 was on an educational leave of absence. The background of each respondent is summarized in Table 4.1.

Case	Interview Frequency	Experience on Team at Time of Interview	Highest Level of Educational Attained	Previous Work Experience
S1	Every 3 weeks for 4 months	0-4 months	Masters in CS (compilers)	4 co-op work terms
S2	Every 3 weeks for 4 months	0-4 months	Masters in CS (compilers)	3 years as Windows system programmer
S3	Once	7 months	Bachelors in CE	2 years with an optimizing compiler
S4	Once	8 months (on leave)	Doctorate in CS (artificial intelligence)	Summer jobs

Table 4.1: Summary of Respondent Characteristics

Structured interviews were used with all respondents. They were asked standard questions and were allowed to elaborate as appropriate to their situation. All interviews were conducted by a single investigator and were tape-recorded. Prior to being interviewed, respondents signed consent forms. All raw data is kept confidential, and the anonymity on the respondents is maintained.

Three sets of questions were used: the first set of questions inquired about the respondent’s background, both educational and industrial; questions from the second set probed the respondent’s growing understanding of the software system and naturalization process in progress; and the last set explored the respondent’s naturalization experience in retrospect. Question set one was used during the first interview with a respondent, and set three during the last. With S3 and S4, these occasions coincided. Question set two was used only with S1 and S2 as we followed them through their naturalization. The usage of these question sets

with the respondents is summarized in Table 4.2. These question sets can be found in Figures 4.1-4.3.

Cases	Question Set 1 Used During:	Question Set 2 Used During:	Question Set 3 Used During:
S1, S2	First interview	Interview every 3 weeks	Last interview
S3, S4	Only interview	No	Only interview

Table 4.2: Summary of Question Set Usage

At the end of the four months, we concluded our interviews with S1 and S2 because we felt that the immigrants had reached a plateau in their naturalization. This is not to say that they were completely familiar with the software system, but rather they had settled into a stable work routine and would be making a steady transition to being fully productive team members.

4.3.1.2 Data Analysis

Since a single investigator conducted all of the interviews, hypotheses were formulated throughout the study, using a method of constant comparison [Eisenh89]. After data collection concluded, notes and recordings made during the interviews were reviewed entirely. During this stage, seventeen variables of interest in five major areas of inquiry were identified using cross-case comparisons. It is important to note that the variables used were not scalar, but quantitative. A “value” assigned to a variable could be numerical, but textual descriptions and lists are also valid. The variables are listed in Figure 4.4 , and the areas are:

- respondent characteristics,
- orientation and training,
- difficulties outside of learning about the system,
- timing and type of tasks given, and
- approaches used to understand the system.

Question Set One: Subject's Background

1. What is your educational background?
2. What experience have you as a professional software developer? What kinds of projects did you work on? What tools and languages did you use?
3. Are there any educational materials that you found particularly useful such as books, manuals, guides, course, videos ?
4. What do you enjoy most about your work?
5. What do you dislike most about your work?

Figure 4.1: Question Set One**Question Set Two: Observing the Naturalization Process**

1. What is your current assignment? What have you been working on over the last week?
2. How did you gather information about the problem?
3. What resources did you use? What documentation did you read? Who did you consult?
4. What new things did you learn over the last week?
5. What new tools did you use over the last week?
6. Did you use Software Bookshelf? Include information about how and why if appropriate.
7. Over the last week, what have you done to become more familiar with the software system?
8. Draw a diagram of your current understanding of the system.

Figure 4.2: Question Set Two**Question Set Three: Recalling the Naturalization Process**

1. How long have you been working at this job?
2. What administrative issues did you have difficulties with? (i.e. badges, logins, machines, payroll, etc.)
3. How many different computer systems do you have to use to do your job?
4. How many different tools or applications do you have to use to do your job?
5. What technical issues did you have difficulties with? (i.e. missing background knowledge)
6. What difficulties did you encounter when learning about the system you are working on?
7. How long did it take you to become comfortable with your new environment? (i.e. office, building, cafeteria)
8. How long did it take you to figure out office numbers?
9. How long did it take to become productive?

Figure 4.3: Question Set Three

Data from the interviews were used to assign values to these variables and this information was put into a data matrix. A pattern matching technique was used, in which several pieces of information from one or more cases were related to a theoretical proposition [Eisenh89, Miles94, Yin94]. Seven propositions or “patterns” were found. Some of the propositions were grouped together because their causes or effects were tightly linked. These patterns will be presented in the next section.

- educational background
- work experience
- orientation
- training
- mentoring relationship
- IDs acquired
- computer systems used
- tools used
- time to fully functioning workstation
- system administration tasks reported
- initial task
- time until initial task assigned
- time until working independently
- shortcomings of technical background
- approach to learning system
- time to comprehend office numbering system
- other

Figure 4.4: Variables Used During Analysis

4.3.2 Results

In this subsection the findings of the study are presenting beginning with a narrative overview of the naturalization process, then continuing with analytic results. Counts of some variables will be presented, where relevant, using the following notation: (A, B, C, D) units. This tuple indicates a count of A units for S1, B units for S2, and so on. There are sufficiently few cases that it is possible to present all the data, and this notation allows us to do so compactly.

When software immigrants began work, they were each assigned a mentor. Only S3 received a three-hour formal orientation session from the human resource department; the remainder

received informal orientations from their manager. Some respondents attended external formal courses, but they did not find them relevant to their work; respondents attended (0, 1, 0, 2) courses. Mentors acted as primary sources of information to software immigrants, and they passed on a wide range of information to respondents. This information tended to be practical low-level information, such as file naming conventions, system set-up, and pointers on tool usage.

The first two weeks were focused on administrative issues, that is, providing the software immigrant with the equipment, tools, and user identifications necessary to do his or her job. Half the respondents received their first task after two weeks, the other half after three. These first tasks tended to be isolated modifications to the software, or open-ended investigations with no predetermined goal. After four months, five in the case of S4, respondents were working independently of their mentors on tasks that had gradually increased in scope. Although respondents did not yet have a thorough understanding of the system, they were on their way to acquiring one. In the words of S3, “I’m fairly comfortable now. I can read the code and understand it. ...I know where to look for problems, and that’s half the battle and I know who to consult when I don’t.”

In the remainder of this subsection, patterns in the naturalization process will be discussed. The pattern is substantiated with details from the cases, then its implications are discussed and, where possible, related to the literature.

4.3.2.1 Mentoring

- **Pattern 1:** Mentoring is an effective, though inefficient, way to teach immigrants about the software system.
- **Pattern 2:** Lack of documentation forces software immigrants to rely on mentors or consultants.

Evidence

When respondents joined the team, each was assigned a mentor who helped them with all aspects of naturalization. This assistance ranged from providing basic information about the software system, to workstation system administration, to steering them around food choices in the lab's cafeteria. Initially, mentors spent many hours a day with their charge. This time may have been lumped together into a long lecture or it may have been spread out over two or three question and answer sessions in a day. This frequency was maintained for about two weeks and then tapered off quickly. The intensity and duration of the initial contact period was less for subjects whose mentors were working on time-critical tasks. Although contact with their mentors decreased over time, it never stopped completely as maintainers often consult experts about esoteric parts of the software system. By four months, S1's interaction with his mentor consisted of a short question every two days or so. In contrast, S4 had a steady ongoing contact with her mentor because they worked closely together on the same problems.

There is a paucity of documentation for this system; what information does exist resides primarily in the minds of those developers who designed the system architecture and continue to maintain it. S3 stated, "Most people operate under the assumption that there are no documents, so you shouldn't try asking for one." This shortage means that for immigrants, their mentors become their primary source of information about the software system.

Beyond passing on knowledge, mentoring fills a social function as well. Mentors act as a means for integrating an immigrant into the social life of the software team, by providing them with introductions at the lunch table and during coffee breaks. Newcomers need to become conscious of their fellow team members and their areas of responsibility, so that they can turn to the appropriate consultant when necessary.

Implications

A major drawback of mentoring is that it is very time consuming for the senior developer, a phenomenon discussed in the introduction of this chapter. Despite the inefficiencies of mentoring, it may not be possible, or even desirable, to eliminate the system. Mentors

function as more than mere repositories of data about the legacy system; the information they provide extends into the administrative and social domains as well. In light of the lack of documentation, it is important to identify who the experts are to new team members.

If changes are to be made to the naturalization process, the mentoring system should be complemented, but not replaced. The experiences of software immigrants in this study were consistent with those found by Berlin [Berlin93]. Like the apprentices in that study, these software immigrants also had interactions with their mentors that were highly interactive, in which they received timely feedback about their comprehension of the software. Efforts should be made to reduce the time commitment required by mentors, so they can still maintain their productivity, while providing adequate guidance to a software immigrant. As a result, an immigrant who has a mentor with a busy schedule, can still receive the necessary training.

4.3.2.2 Difficulties Outside of the Software System

- **Pattern 3:** Administrative and environmental issues were a major source of frustration during naturalization.

Evidence

In every case, almost the entire first two weeks were spent dealing with administrative and environmental issues. These difficulties included setting up their computers, configuring software, acquiring access to systems or tools. In many instances, there was overhead involved in performing simple tasks. Respondents had to maintain (11, 11, 5, 5) identifications, accounts or registrations to do their job.

Only S3 had a fully functioning workstation on the first day of work. Respondents had to wait (3, 6, 0, 1) weeks for fully functioning machines. S4 had a computer on the first day, but had to spend a week configuring it to be usable. S2 did not even have a workstation on his desk for the first three weeks, and then needed another three weeks to set it up to meet his needs.

Sometimes these problems are interrelated, as recalled by S1, “I tried to [set up backups for my machine], but I got stalled because I had to register my machine. So when that comes back, I’ll continue...” Although his computer was basically operational after three weeks, S1 had to deal with system administration problems throughout the study.

Items ranging from user identifications to light bulbs had to be requested. Some requests could be serviced quickly but most requests required an overnight wait. Once, when S2 returned to his office with a binder, his office mate asked him, “Where can I apply to get a binder?” Ironically, binders, unlike so many other supplies, did not need to be requested.

Although respondents worked hard to comprehend a large under-documented system, at no time did they describe the task as frustrating. In contrast, frustration was a word that every respondent used with respect to at least one system administration task. This difficulty could be attributed to respondents’ lack of experience performing system administration, or the feeling that machine problems were keeping them from their real jobs—programming. Regardless of the causes of this sentiment, it is a problem common to software immigrants during naturalization. Later discussions with the project manager indicated that difficulties with the lack of computing resources were experienced by all members of the team.

Implications

The problems with administrative and environmental issues, particularly the computing resource shortage, would be worth addressing for this team, since benefits would be felt not only by software immigrants but also by veterans. Some real productivity gains could be made here if developers were not distracted by administrative issues. It is not very efficient for every team member to invest the time to learn how to perform system administration, an activity not directly related to writing code. Many of the processes could be streamlined or combined; for example, user identifications for a set of tools could be linked so that access to them does not need to be requested separately.

4.3.2.3 First Assignment

- **Pattern 4:** Initial tasks were open-ended problems or simple bug repairs, that were begun no earlier than two weeks after a software immigrant's arrival.
- **Pattern 5:** Mentors tend to pass on low-level information about the software system.

Evidence

Shortly after respondents had functioning machines, they received their first assigned task, which occurred at (3, 4, 2, 2) weeks. These initial assignments tended to be limited in scope and complexity, and did not have a fixed deadline. Three of the respondents were given open-ended problems to explore, for the purpose of improving the compiler's performance. S3 was given a bug repair that had been screened for excessive complexity by his mentor. S4's first assignment was to add a feature to a subsystem, and she recalls, "It was a small enough project and I didn't have to know anything else about the rest of the code. So it was a matter of modifying, maybe three or four files... It didn't seem very challenging, but looking back, I appreciate the fact that they gave me something so isolated. It allowed me to gain familiarity with at least those four files."

Three of the four mentors concentrated on conveying low-level information to immigrants. These lessons tended to concentrate on the subsystem that an immigrant would be working on and as a result tended to focus on knowledge that was immediately useful. Only S1's mentor began with high-level system design concepts, but even these lessons were limited to a single subsystem. By concentrating on pragmatics, software immigrants were able to start working with source code quickly.

Implications

Clearly, patterns four and five are closely related: Given the types of information conveyed by mentors, small, non-critical tasks are appropriate first assignments for software immigrants, and vice versa. Even in the absence of pressure from the team, respondents tended to push themselves to contribute. S1 observed this in himself, saying, "Sometimes it's me trying to do several things at the same time: trying to set up my machine and ...be a little bit productive for

the team.” In such situations, the additional demands of a task with a tight deadline is unnecessary. The relationship between these two patterns can be viewed as symbiotic. Any modifications to one pattern, must be reflected in the other. Clearly, the initial task needs to provide an opportunity for software immigrants to use the lessons learned.

4.3.2.4 Predictors of Job Fit

- **Pattern 6:** Programmers who prefer to use bottom-up comprehension approaches have a smoother naturalization than those who don't.
- **Pattern 7:** There needs to be a minimal interest match between immigrants and the software system.

Evidence

At the end of the study, cases S1-3 were still working on the software team, but case S4 was on a temporary educational leave. This provides an opportunity to examine the differences between a team member who may pursue other interests, and ones who are satisfied working as software maintainers on a compiler. The three key differences were S4's inclination to take a top-down approach to comprehending the software system, and her lack of previous experience with compilers, coupled with her depth of background and interest in another field.

Immigrants were trained up from simple tasks to more complex ones. Consequently, software immigrants acquired their understanding of the software, one subsystem at a time, in other words, in a bottom-up fashion. S1-3 took this approach when they tackled a problem by reading the source code or by profiling the subsystem. In contrast, S4 preferred to take a top-down approach, although there were no real tools or documentation that supported this line of inquiry. She said, “The system was humungous and I didn't know what comes first or anything. So the only way to do it is to dump everything [execution traces]. I didn't do that from the beginning, but I found it really frustrating because I wouldn't know what was actually being done. You need to know... or you don't know where to start.”

S4's background also differed from those of the other respondents. During their Masters degrees, S1-2 both wrote theses in the area of compilers. S3 had previous experience working on a highly similar software system. S4 had completed a Doctorate in artificial intelligence. She indicated this was another reason she did not find her work compelling, "I had spent four years working on my Ph.D. and I got hired into an area that had nothing to do with my Ph.D. I just never found it fascinating. They knew that when they hired me. ...They just wanted some one they felt could pick things up quickly."

At this point, it must be stated that S4 was not an unsuccessful software maintainer. Although she is on leave, she has not given any indication that she will not return. When describing her work, she included as many low-level details of the software system as S1-3. She was able to handle tasks that were as complex as the ones given to other respondents. Furthermore, throughout the interview she emphasized that despite the interest mismatch she had congenial relations with the development team. She stated, "The actual group was amazing. I think I was very fortunate to be in that group," and "...it was a positive experience. I don't regret working there."

Implications

Any improvement in job fit is, indirectly, an improvement on the naturalization process, since reducing a possible turnover rate decreases the time spent in this area by the team as a whole. When hiring new employees to be software maintainers on a large project, managers should look for at least a minimal interest match and a preference to work with system details in a bottom-up fashion. This is not to say that immigrants without these characteristics are certain to fail or leave, but they will face greater frustration in their early months on the job, a time that has its own share of difficulties. A newcomer with a strong interest match is more likely to be buoyed by a high level of initial excitement about the position, a feeling that does much to mitigate many of the frustrations he or she may face. Indicators of an interest match could be experience in a related field, or it may be as simple as an expressed preference. A scheme to give employees choices in the work they undertake is proposed by DeMarco and Lister [DeMarc87].

4.3.3 Application of the Results to PBS

Taken as a group the patterns provide a coherent explanation for why software immigrants use PBS so little. They spent very little time accumulating domain level knowledge about the system, usually about two weeks. During this time, software immigrants were struggling to get a computer set up. For as long as they did not have a fully functioning computer, software immigrants could not access PBS. By the time they had a computer on their desktop, immigrants were assigned a maintenance task by their mentors and they began tackling it immediately. These initial tasks were localized and only required immigrants to be familiar with a small number of files in a single subsystem. When software immigrants could access PBS, a Software Landscape was the only information available on the subsystem they had been assigned to work on. These diagrams were too abstract to help them understand an algorithm or a specific source file. After some experiments with PBS, they found that it lacked the information they needed for their assigned tasks and put the tool aside.

Software Landscape's main strength is that it reduces the complexity of a software system that is inherent in a directory of source files, by presenting a picture of its conceptual organization. While S3 found PBS to be an invaluable resource, he was the only respondent who had a computer on his desk from the outset. He reported that PBS probably saved him about two weeks of time accumulating background knowledge. Despite these positive early experiences, S3 gradually stopped using the tool after his initial learning period. He described PBS as a tool of "last resort" that he turned to when all other information sources were exhausted.

The evidence regarding the efficacy of PBS for software immigrants is inconclusive. Although there were some positive and negative comments, there is sufficient evidence to make a strong argument, in either direction, for it as a program comprehension tool. The only unequivocal result is that PBS did not fit well with how software immigrants are naturalized on this development team. At a time when software immigrants would have benefited from the information that PBS excels at delivering the tool was not available due to organizational

constraints. In order to keep PBS relevant throughout a software immigrant's naturalization, it needs to be able to provide information that will help them with their initial tasks as well. Some mechanism needs to be added that can provide structural information at a lower level than the Landscapes do. Furthermore, this mechanism should complement the Software Landscapes, so that PBS can provide a more complete picture of a software system.

4.4 Project Veterans

The second PBS user group studied were project veterans. It was expected that this group would use PBS as a reference, that is, to look up information about relationships between modules or to validate their knowledge of the software system. It was difficult to find software maintainers from this category using PBS in their daily work, so it was necessary to use more innovative ways of evaluating PBS as a program comprehension tool for them. Techniques that were used included formal demos, email, meetings to validate Landscapes, and "coincidental" meetings. This study could be best described as informal and consequently it does not have systematic results. However, the findings are intriguing and highlight possible directions for future research.

Demos of Software Bookshelf were organized at IBM internally and at their annual conference, CASCON. On these occasions, the concepts would be explained to team members and their reactions to the Landscapes were noted. Email was exchanged with developers to ask them about how they deal with specific maintenance tasks. Two of the researchers developing the Software Bookshelf arranged meetings with senior team members to validate the system decomposition used in Software Landscapes [Tzerpo96]. Interestingly, during one of these meetings a bug was identified using only Landscapes. Ambushes occurred when a researcher would see a software maintainer in the hallway, cafeteria, parking lot, or in her or his own office, and ask for her or his thoughts on PBS. The remainder of this section presents comments that were either common to many developers or uncommon enough to be interesting.

4.4.1 Questions about Edges

After the basic Software Bookshelf concepts, such as the Table of Contents, a Software Landscape and the meaning of boxes and lines, were explained the first question that a developer would ask was “What’s this edge? Where is it in the source?” They wanted to know what line or lines of source code were responsible for putting a particular edge on the Software Landscape. Developers had more questions about edges than nodes, because the source artifact represented by a node was clear—it was either a file or a collection of files. The source code represented by a box could be accessed just by clicking on the Landscape. There is no mechanism in `lsview` to click on an edge and display the source code it represented.

The question “What’s this edge?” is difficult to answer for a number of reasons. First, the low-level factbase only has information about entities such as variables, functions, and files, but not source lines. Even this information is induced to the file level to create the factbase for a Software Landscape. As a result, variables and functions aren’t even represented on a Landscape, only the artifacts, such as boxes and arrows, that imply their presence are. Second, it’s not clear that “what’s this edge?” is really the question that maintainers want answered. Suppose that it was possible to bring up a list consisting of the lines of source code that an edge represented and the list was similar to the following:

```
asm.c:65:          if(memory[i] == NULL)
codegen.c:168:     memory[startMSP] = Iord;
machine.c:59:     fprintf( sinkFile, "%8hd",
memory[i] );
```

This list only leads to other questions such as: What variable type is “memory”? What functions are these lines from? Upon examination, the “What’s this edge?” has deeper implications beyond adding a feature to the `lsview` Java applet.

4.4.2 Maintainers’ Comments on Anomalies

When presented with a Software Landscape, developers exhibit a range of reactions. The strongest reactions were from developers who thought they saw an obvious error in the diagram. These “surprises” raised questions and comments. Sometimes these anomalies were true tool errors, for example a mistake was made during the clustering process and a file was

placed into the wrong subsystem. Sometimes these anomalies were a mismatch between the developer's mental image of how the system should be drawn and the representation in the Software Landscape. Other times the anomalies were actual errors in the TOBEY implementation.

The implication of these criticisms is that there is an objective and a subjective element to evaluating Software Landscapes. Both elements can manifest themselves in both the psychological and technical domains. From a psychological standpoint, the objective aspects are apparent in the design principles that should be followed when displaying visual information. The subjective aspect arises when the visual representation is compared with a developer's mental picture of a software system. From a technical standpoint, the objective element is relevant when evaluating whether files have been placed in the appropriate subsystem. The subjective arises in the definition of subsystems in the first place. All four aspects (the cross product of objective/subjective and technical/psychological) may be sources of comments from software maintainers.

4.4.3 Journalism-Style Questions

Some of the questions that senior maintainers asked when repairing a defect or evaluating a set of Software Landscapes were surprising. A senior software maintainer who had been working on a project for a long time sometimes asked journalism questions: who, what, when, where, why and how. When looking at source code, a possible sequence of questions and answers were:

What is this thing?
 Who did this?
 Oh, I remember, this was when X fixed bug Y.

Or

This looks like Z wrote this.
 When was that? How long had Z been working then?
 What was he trying to do?
 He probably did this because...

These questions are indicative of an effort to recover design rationale. If a senior maintainer can determine the circumstances surrounding a change, she or he can often infer why a change

was performed a particular way. In these situations, history is being used to make up for the dearth of documentation on this software project. For example, a fact extractor could be applied to configuration management or version control tools to collect historical information.

4.4.4 Code Migration

It was expected that PBS would assist software maintainers as they migrated subsystems from the PL/IX programming language to C++. Those who were involved in the migration effort used Software Landscapes to verify some of their decisions rather than as a driver of the process. There were three reasons for this reticence. One, maintainers did not completely trust Software Landscapes, as they trusted their tried-and-true software tools such as `grep` or `find`. While this reluctance was understandable, it was nonetheless disappointing. Two, maintainers who were sufficiently senior to be entrusted with the task of designing the ported subsystem already had good mental models of the software system as a whole. As a result, they did not feel that they needed to rely on Software Landscapes for a decomposition. However, they did check the Landscapes after the design was complete to verify that there were no surprises. Three, maintainers who were responsible for doing the actual migration were working at a level too low to be helped by Software Landscapes. The task required them to ask a lot of “What’s this edge?” type questions that could not be answered using the existing PBS tools.

4.5 Summary

The results from these two user studies have a number of implications for refining PBS as a code comprehension tool. The points that are most relevant to this thesis are summarized in this section.

- Software maintainers want to relate pictorial elements to lines of source code.
- Low-level information is needed in PBS to complement architectural diagrams.

Evidence for this conclusion can be found in the prevalence of the question “What’s this edge?” While abstraction can be helpful for learning new concepts, it does not help developers perform day-to-day maintenance tasks. Their jobs centre around modifying a large

corpus of source code. From studying how software maintainers work, we make the following two observations:

- Software maintainers use goal-directed knowledge acquisition.
- Information is gathered by searching and asking questions.

Developers who were studied spent very little time learning for the sake of knowing. Even software immigrants spent only two weeks on open-ended study. It is more often the case that maintainers want to acquire a specific piece of knowledge for a particular maintenance task. Maintainers find this information by searching source code or by asking their peers. Asking questions is a key part of the problem solving process. Developers have been observed asking questions aloud while working alone, and providing the answers themselves.

There are many conclusions that can be drawn from these lessons, two of them are:

- PBS needs a search tool that lets users make queries about source code.
- Additional studies are necessary to develop and understanding of what searches programmers perform on source code and how.

In order to keep PBS relevant to software maintainers beyond the initial open-ended learning period, a search tool needs to be added. The tool needs to be able to answer the “What’s this edge?” question, and all those that follow. The primary purpose of this tool would be to search source code for information that programmers need to perform maintenance tasks. There are two aspects of this problem that need to be examined more closely. First, an understanding of source code searching for maintenance tasks needed to be developed. To this end, a survey of programmers was undertaken and is described in the next chapter. Second, the problem of source code as a search domain presents a technical challenge. A collection of search tools and code analysis tools is examined in Chapter 6. Based on the findings from these two lines of investigation, a design for a searching tool is presented in Chapter 7.