The Ramp-Up Problem in Software Projects:

A Case Study of How Software Immigrants Naturalize

Susan Elliott Sim
Department of Computer Science
University of Toronto
10 Kings College Rd.
Toronto, Ontario, Canada M5S 3G4
416-978-1685
simsuz@cs.utoronto.ca

Richard C. Holt
Department of Computer Science
University of Waterloo

Waterloo, Ontario, Canada N2L 3G1
519-888-4567 x4671
holt@plg.uwaterloo.ca

ABSTRACT
Joining a software development team is like moving to a new country to start employment; the immigrant has a lot to learn about the job, the local customs, and sometimes a new language. In an exploratory case study, we interviewed four software immigrants, in order to characterize their naturalization process. Seven patterns in four major categories were found. In this paper, these patterns are substantiated, and their implications discussed. The lessons learned from this study can be applied equally to improving the naturalization process, and to the formulation of further research questions.

KEYWORDS
empirical study, software maintenance, new employees, process

1. INTRODUCTION

The title of Brooks' book, *The Mythical Man-Month*, neatly sums up a software team management concept: adding personnel to a project actually decreases productivity in the short term due to the start-up costs of new team members.[3, 4] Despite this difficulty, it is often necessary to add new employees to replace personnel or to grow the team in order to take on additional work. Studying the ramp-up process for these recruits helps us identify ways to make this transition easier. The benefits of making this adjustment more manageable include: employees who are productive sooner; fewer distractions for senior team members who act as mentors; and the flexibility to add new employees as they are needed. Such improvements are particularly beneficial to a growing team that expects the arrival of many new hires or to an aging team backfilling positions that are open due to attrition.

When software maintainers begin work on a project, they face the daunting task of re-tooling themselves for a new job. This task usually requires them not only to learn about the system, but also to adapt to the new working environment. In order to become productive they need to learn:

- programming languages or dialects
- tools
- the intricacies of a software system
- project jargon
- development processes
- coding standards
- background information about the problem domain
- team dynamics—who does what
- organizational structure
- how to obtain resources and supplies

This is an incomplete list of potential areas in which a new recruit may have to make adaptations. The specifics will vary according to the team and the recruit.

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 adaptation, re-tooling, start-up, ramp-up or bringing someone up to speed.

In too many cases, little preparation has been put into a software immigrant's training, beyond assigning them to a senior developer who acts as a mentor. This person is expected to help the new recruit become productive by providing whatever guidance she or he needs. DeMarco and Lister[4] observe "We all know that a new employee is quite useless on day one or even worse than useless, since someone else's time is required to begin bringing the new person up to speed." They go on to estimate that this cost is about twenty percent of the cost of an employee who works for two years, more if the project is complex.

While there exists a significant body of work from the fields of organizational psychology, business administration and occupational training on the acclimation of new employees, this work tends to focus on domains such as education, medicine, manufacturing, and administration. Wilson[12] is a detailed study of how registered nurses responded to the stresses of starting employment at a hospital. Van Maanen and Schein [13] developed six bipolar scales to characterize the socialization of new hires. Practical advice on designing orientation sessions for recruits is given in Beeler [2]. These papers are but a minuscule selection from the literature of these areas. Unfortunately, there has been little work done on new employees in technology oriented domains, such as engineering or computing.

Studies have been undertaken in software engineering and cognitive psychology on working with legacy systems. There are some publications which give practical advice on working on undocumented software systems[6, 10] and anecdotes from practitioners and consultants[3, 4]. The most significant contribution comes from Berlin [1] 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 tend to provide not only answers to apprentices' questions, they also give 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. As an antidote, Berlin suggests capturing the information that mentors convey in documentation or an intensive course for apprentices.

Since the naturalization process is understood from different perspectives in a piecemeal fashion, we undertook a study to characterize the process as a whole, a perspective that includes orientation, socialization, and the acquisition of domain knowledge. An exploratory case study methodology was used because we were investigating a relatively uncharted area.[14] This research design is not used to test hypotheses using constructs formulated, a priori. Rather, it is used to build a body of knowledge about an area that is not well understood. An exploratory case study begins with a rationale and a direction. Our research rationale was to acquire an understanding of the naturalization process of immigrants for the purpose of redressing any shortcomings. The direction we chose to take to forge this understanding was to interview a number of software immigrants about their naturalization experience.

This approach would allow us to consider naturalization in context and to formulate theories about the process based on interview data. Another benefit of using this methodology is that each immigrant examined can be viewed as a replication of the case study, and as such contribute to the formulation of analytic generalizations. These inferences can stand on their own merit, or they can be used later as the basis for quantitative investigations. Probabilistic generalizations can not be made using only the quantitative data from case studies, and, indeed, should not, since this type of result is reserved for more experimental research designs. Since the logic of case studies differs significantly from the logic of experimentation, they use different approaches to preserve internal and external validity, construct validity, and reliability.[5, 14]

Using this method, we were able to characterize the naturalization process and make inferences about the features observed. In section two, the methods we used to collect and analyze data are described. In section three, we detail the seven patterns in four major areas that we found. The lessons that can be learned from these patterns by both practitioners and researchers are discussed in section four.

2. METHOD

- A multi-case study was performed with four respondents, all software immigrants to a single team. By interviewing subjects, we hoped to identify commonalties and differences in their experiences, and to infer naturalization patterns from this comparison. In this study, our goals were to:

- describe the naturalization process;

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 1: Summary of Respondent Characteristics

- 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 identify strengths and weaknesses in the naturalization process.

The unit of analysis in this study is a single software immigrant. Data was collected using structure interviews, and was analyzed using qualitative data analysis methods. Variables of interest were identified using a pattern matching technique. A data matrix was populated with these variables to articulate cross-case patterns.[9] In the following three subsections, we will describe, in order, the development team, the data collection procedure, and that data analysis techniques that were used.

2.1 Research Setting

We studied software immigrants to a development team maintaining a legacy system at a very large computer company. The software is approximately fifteen years old and had has twelve major releases. There are approximately 250 000 lines of source code in 1000 files. Developers used workstations running the AIX operating system (an IBM UNIX variant). Senior developers on this project estimate that it takes six to twelve months for a new team member to become fully productive. Through most of its history the development team consisted of approximately ten people, but in recent years its size has nearly doubled. The growth of the team provided us with an opportunity to study a relatively large number of software immigrants.

2.2 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"[5], 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 1.

Structured interviews were used with all subjects, in which standard questions were asked and the respondents 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 respondents anonymity 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 concerned 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

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 2: Summary of Question Set Usage

followed them through their naturalization. The usage of these question sets with the respondents is summarized in Table 2. These question sets can be found in Appendix A.

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.

2.3 Data Analysis

Since a single investigator conducted all of the interviews, we were able to formulate hypotheses throughout the study, using a method of constant comparison.[5] 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 Appendix B, 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.

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 are related to a theoretical proposition. Seven propositions, or "patterns" in 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.[5, 9, 14]

3. RESULTS

In this section, the findings of the study are discussed. It begins with a narrative overview of the naturalization process, then it continues 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 this 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 fixed endpoint. 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 section, patterns in the naturalization process will be discussed. The pattern is substantiated with details from the cases, then its implications are discussed, and related to the literature, where possible.

3.1 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.

3.1.1 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 navigating them through the 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 length of the initial contact period was less for subjects whose mentors who 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 an introductions at the lunch table and during coffee breaks. Newcomers need to develop an awareness of their fellow team members, and their areas of responsibility, so that they can turn to the appropriate consultant when necessary.

3.1.2 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 paper. 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; 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 in Berlin[1]. Their interactions with their mentors were also highly interactive, and 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.

3.2 Difficulties Outside of the Software System

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

3.2.1 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 an average of eight identifications, accounts or registrations; to do their job; they reported having (11, 11, 5, 5) identifications.

Only S3 had a fully functioning workstation on the first day of work. Respondents had to wait (3, 6, 0, 1) weeks, an average of two and a half 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 he 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 undocumented system, in no case 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.

3.2.2 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 has to invest the time to learn how perform system administration tasks, 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 need to be requested separately.

3.3 First Assignments

- **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.

3.3.1 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.

3.3.2 Implications

Given the types of information conveyed by mentors, small, non-critical tasks are appropriate first assignments for software immigrants. Even in the absence of pressure from the team, respondents tended to push themselves 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.

3.4 Predictors of Job Fit

- **Pattern 6**: Programmers who prefer to use bottom-up comprehension approaches have a smoother naturalization that those who don't.

- **Pattern 7**: There needs to be a minimal interest match between immigrants and the software system.

3.4.1 Evidence

Cases S1-3 are still working on the software team, but case S4 is 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 two 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 cordial 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."

3.4.2 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 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 in DeMarco and Lister.[4]

4. APPLICATION OF THE PATTERNS

Looking at the patterns across the cases, one of the conclusions we drew was that software immigrants could benefit from an intensive course on that focused on high-level system details. Such a course could: complement the lessons given by mentors; present important core information to all immigrants; reduce some of the variability between mentors; and decrease the time commitment required by mentors. Immigrants could attend this course outside of the department in a location where lack of resources is not an issue. Such a course could replace mentoring, since classroom lectures tend to isolate immigrants from the rest of the team. As software maintainers, they will eventually need to consult fellow experts when trying to solve problems. Therefore, it would not be desirable or even possible to eliminate mentor or consultant relationships entirely.

In June, we undertook a pilot study in which we designed such a course and administered it to three software immigrants with six weeks or less experience. Three senior developers were interviewed to help select curriculum for this course. They were asked "What does a newcomer to the team need to know in order to become productive?" Two of the three senior developers consulted felt that it was important for immigrants to acquire an understanding of the system to work effectively. Of these, one of them emphasized the importance of conveying design rationale to immigrants. The third developer consulted was primarily concerned with the practical knowledge required. Since mentors already present this information, we decided not to overlap the lessons.

This course was based on two hours of videotaped talks previously given by senior project developers, and a software architecture visualization tool, Software Bookshelf.[6] These materials were chosen with the eventual goal of making the course self-guided.

The reactions to the course were encouraging. On the course evaluations, the participants remarked that they would have like more information on the overall system architecture and main data structures. Based these results and the suggestions made by the immigrants, we found that the material was pitched at the right level, although some of the content needed fine-tuning. Four months after subjects completed the course, they were given an e-mail questionnaire that asked them to evaluate to the course in retrospect. They found parts of the course very helpful. Although they did not find the course directly applicable to their daily work, they often thought about the concepts they had learned.

The course developed should not be used as a substitute for the current mentorship system. Mentors also serve as the basis for a social introduction to the team that a course cannot replace. These consultation sessions remain very effective for transmitting low-level information where no documentation exists and for conveying design rationale.

Other teams could evolve similar courses for immigrants to their legacy systems. The course could be self-directed or taught in a lecture format, depending on the goals of the course, available resources, arrival rate of immigrants, and personalities on the team. The course or courses could be aimed at different points in time during an immigrant's naturalization. Teams could select curriculum appropriate for their problem domain, but a standard course should focus on core information to ensure its applicability across the team.

Such a course should be short, and consequently intense and highly directed. It should be presented in a manner that requires little intervention on the part of experts. This could be done using videotaped presentations or short talks from a number of team members or some combination of these approaches. Finally, the current naturalization process could be co-opted into helping to prepare material for such a course. Informal lectures by experts could be videotaped for viewing by future immigrants. New hires could record information they uncover during their naturalization, adding to the documentation available on the software system.

5. CONCLUSIONS

The study undertaken in this paper used a case study methodology to describe the naturalization process of software immigrants. The study was replicated with four newcomers to a single development team. Seven patterns were identified in cross-case analysis; they are:

- **Pattern 1**: Mentors are 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.
- **Pattern 3**: Administrative and environmental issues were a major source of frustration during naturalization.

- **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.
- **Pattern 6**: Programmers who prefer to use bottom-up comprehension approaches are more appropriate for the job of software maintenance.
- **Pattern 7**: There needs to be a minimal interest match between immigrants and the software system.

These cases identified not only the strengths and weaknesses in the process, but also areas for future research. The lessons learned can be applied both to the naturalization process itself, and to future research.

A theme that repeats itself across patterns, almost as a meta-pattern, is that the key to improving the naturalization process is to minimize frustration. This tactic is already used in some areas, for example, the use of mentors, and the tight coupling of lessons and initial tasks. Software immigrants did not speak negatively bout tasks that were difficult, only about tasks that were frustrating. While newcomers' early excitement and motivation carries them over many obstacles, it's best to maintain these positive feelings for as long as possible. While strategy may not directly reduce the time they need to naturalize, it may do so indirectly by keeping immigrants enthusiastic about their work.

The obstacles faced by respondents outside of learning about the software system led us to the next conclusion: any efforts to facilitate the naturalization process cannot be limited to technical solutions such as "more documentation" or "better tools". Improvements must encompass modifications to the organizational processes surrounding the arrival and integration of new developers to the team. Such changes were outside the scope of this study, and may require revisions not only at the team level, but also at the organization level.

As is often the case, this study posed many more questions than it answered. While software immigrants expended much effort resolving administrative issues, to what extent is this typical of team veterans? Some developers have estimated that they spend only 20% of their time programming; the other 80% is spent doing things so they can. Is this adage true, and how do software immigrants fit in? It was observed that mentors needed to spend a great deal of time with their charges. How much of a time commitment is needed? Is less time needed on systems that are better documented?

The use of an exploratory case study methodology to examine software processes is effective for two reasons. First, beyond anecdotal evidence, there have been few studies that document what actually happens on a development project. Researchers need to build an understanding of the processes already in place, so that they can create innovations that are more likely to be adopted. Case studies allow us to examine and document software processes in context. Second, this methodology allows the construction of theories from qualitative data. While case studies are not a universal solution for empirical research, they can be used forge an understanding of many hitherto unexamined phenomena. Investigations like this have the potential to bridge the gap been between the practice and theory of software engineering.

ACKNOWLEGEMENTS

This research was generously supported by CSER, ITRC, and IBM Canada Ltd. We would like to the thank the software immigrants who generously gave of their time to participate in this study and the senior developers for their helpful advice. Thanks also to Stephen Perelgut, and Gary Farmaner who helped with the studies.

REFERENCES

[1] L.M. Berlin. Beyond Program Understanding: A Look at Programming Expertise in Industry. *Empirical Studies of Programmers, Fifth Workshop*, pages 6-25, Palo Alto, USA., 1993.

[2] C. Beeler. Roll out the welcome wagon: structuring new employee orientations. *Public Management*, Vol. 76, August, pages 14-17, 1994.

[3] F.P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Addison-Wesley, 1995.

[4] T. DeMarco and T. Lister. *Peopleware: Productive Projects and Teams.* Dorset House Publishing, 1987.

[5] K. M. Eisenhardt. Building Theories from Case Study Research. *Academy of Management Review*, Vol. 14, No. 4, pages 532-550, 1989.

[6] S.D. Fay and D.F. Holmes. Help! I Have to Update an Undocumented Program. *IEEE Conference on Software Maintenance*, pages 194-202, Washington DC, USA, 1985.

[7] P. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. Müller, J. Mylopoulos, S. Perelgut, M. Stanley, and K. Wong. The Software Bookshelf. *IBM Systems Journal*, forthcoming.

[8] D.C Littman, J. Pinto, S. Letovsky, and E. Soloway. Mental Models and Software Maintenance. *Empirical Studies of Programmers, First Workshop*, Washington DC, USA, pages 80-98, 1986.

[9] M.B. Miles and A.M. Huberman. *Qualitative Data Analysis: An Expanded Sourcebook, Second Edition*. Sage Publications, 1994.

[10] T.M. Pigoski and C.S. Looney. Software Maintenance Training: Transition Experiences. *IEEE Conference on Software Maintenance*, pages 314-318, Montréal, Canada, 1993.

[11] W.J. Ray. *Methods: Toward a Science of Behavior and Experience, Fourth Edition.* Brooks/Cole Publishing Company, 1993.

[12] R.M. Wilson. *Patterns of Response to the Demands of Starting New Employment.* Master of Arts Thesis, OISE, University of Toronto, 1972.

[13] J. Van Maanen and E.H Schein. Towards a theory of organizational socialization, in *Research in Organizational Behavior*, edited by B.M. Staw, Vol. 1, pages 209-264, JAI Press, 1979.

[14] R. K. Yin. *Case Study Research: Design and Methods, Second Edition.* Sage Publications, 1994.

APPENDIX A: QUESTION SETS

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 your 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?

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.

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?

APPENDIX B: VARIABLES USED IN ANALYSIS

- 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