

# A Qualitative Study on Project Landscapes

Barthélémy Dagenais<sup>†,\*</sup>, Harold Ossher<sup>‡</sup>, Rachel K. E. Bellamy<sup>‡</sup>, Martin P. Robillard<sup>†</sup>, Jacqueline P. de Vries<sup>‡</sup>

School of Computer Science<sup>†</sup>  
McGill University  
Montréal, QC, Canada  
{bart,martin}@cs.mcgill.ca

IBM T.J. Watson Research Center<sup>‡</sup>  
P.O. Box 704  
Yorktown Heights, NY 10598  
{ossher,rachel,devries}@us.ibm.com

## Abstract

*When developers join a project, they find themselves in a new project landscape and must orient themselves quickly. To investigate the nature of this project landscape, and how we could help newcomers orient themselves, we have started an exploratory study using grounded theory. We primarily collect our data by interviewing experienced developers who recently joined ongoing projects. We are already seeing some patterns emerge. For example, it seems that newcomers find it more important to be able to experiment with the system early on than to have up-to-date and complete documentation.*

## 1. Introduction

When parachutists land, they look at the surrounding landscape to get oriented, focusing initially on prominent features like mountains and rivers, and then examining smaller details in some desired direction. They, or anyone else operating within the landscape, are constantly affected by it; the terrain enables or inhibits navigation and progress, and often facilitates or limits particular activities. We hypothesize that a software development project has a *project landscape*, which affects day-to-day work, and that when software developers join a new project, they must become oriented within the project landscape as quickly as possible. This paper describes the preliminary stages of an exploratory study into the nature of project landscapes.

Based on our own experience, we started with some initial thoughts about project landscapes. We expected the landscapes of different projects to differ significantly, but to have some common kinds of features, such as system architecture, development process, tools, and communication strategies. Our aim is to study project landscapes from scratch, however, unbiased by these thoughts. We are interviewing developers who recently joined software projects

about their integration experiences. Though the project landscape affects all project members, two reasons led us to focus on developers joining new projects: the things a newcomer needs to learn to become productive should be good indicators of key features of the project landscape, and a newcomer's knowledge about the project is still fresh in their mind. To avoid making detailed assumptions based on our initial thoughts, we have adopted the grounded theory methodology.

In the past, studies on project integration have been performed with new employees joining their first software development projects [2, 7]. Because these studies were performed with junior and recently-hired developers, many of the difficulties they encountered related to the newness of the corporate culture and the difference between academic and industrial environments. To focus on the project landscape, our study involves developers with varying degrees of experience in the field and within their company.

We hope that, by the end of this study, we will have a better understanding of the nature of project landscapes and new team members' orientation within them, and that this will inspire work on tools and methods that will ease the situation for both new team members and their mentors. Specifically, we are interested in answering three main research questions: (1) what are the key, prominent features in a project landscape, (2) what obstacles do new team members face, and (3) what orientation help could be provided?

## 2. Methodology

To investigate team member integration in software development projects, we based our study on the grounded theory methodology as described by Corbin and Strauss [3].

**Grounded Theory.** Grounded theory is a qualitative research methodology that employs *theoretical sampling* and *open coding* to formulate a social theory "grounded" in the empirical data. For an exploratory study like ours, grounded theory enables us to start from general research questions and refine the questions, and the instruments, as the study progresses. For example, as opposed to traditional sampling

\*This research was conducted while the author was working at the IBM T.J. Watson Research Center.

Participant	IBM yrs of Experience	Project Age	Project Area
PA	0	>5 years	Mgmt. Tools
PB	0	>1 year	Web App.
PC	0	>10 years	Middleware
PD	0	>1 year	Dev. Tools
P1	9	6 months	Dev. Tools
P2	5	2 weeks	Dev. Tools
P3	7	>5 years	Mainframe
P4	26	>1 year	Mainframe

**Table 1. Participants**

techniques (e.g., random sampling), grounded theory allows us to refine our sampling criteria throughout the course of the study to ensure that the selected participants are able to answer the new questions that have been formulated. Analysis of the data, collected through interviews in our study, is performed using open coding: we assign codes to sentences or paragraphs and we define them as the study progresses. We also place the codes in an evolving hierarchy: at the end of the study, we hope to find a code that will be at the top of the hierarchy and that will form a theory.

**Participants.** We began our study by selecting junior developers who had joined IBM within the prior six months or who were interns, due to availability. The first four participants always mentioned organizational issues<sup>1</sup> rather than project-specific issues, even if they happened only in the first few weeks of work, e.g., being directed to the wrong building on arrival. After another two interviews, we realized that experienced developers often join a project that is just starting, and whose landscape is still unformed. Substantial differences between integration into new and ongoing projects led us to further refine our selection criteria to interview only experienced developers who had recently joined an ongoing project (at least 6 months old).

We learned about team member integration in all our interviews, but a benefit of the grounded theory approach is that our evolving selection criteria ensured that we could focus on narrower research questions, purposely leaving other important questions for future work. Table 1 shows details of the participants we have interviewed so far.

**Instruments.** We use two instruments in our study. First, we ask each participant to draw a sketch of their team and project that could be used to introduce another developer, with similar skills, to the key aspects of the participant’s job. The second instrument is a one-hour semi-structured interview performed on the phone. The sketch makes the participant think about their project and prepares them for the interview. At the end of the interview, we ask the participant to describe their sketch, effectively walking us through their project landscape. In accordance with the grounded theory methodology, both instruments evolve as the study pro-

<sup>1</sup>We use the term “issue” liberally to mean any problems, orientation aid, or information need mentioned by participants.

gresses. For example, we have an interview guide containing eight common questions, such as “What was your first day’s experience like?” and “Can you describe some specific examples that show how you found what you needed to get started on the project.” As we learn more about the various issues participants encounter, we modify common questions and add more specific questions. Because those specific questions could potentially introduce a bias, we add them at the end of the interview. For example, we learned that one participant used the project’s bug repository during the first day to learn about the implementation of various features. We thus appended a question about the usefulness of bug systems for learning about a project.

**Analysis.** Between each interview, we produce a summary that we code and discuss. The summary helps us reflect on the research questions and select the next participants and the questions to ask them. For practical reasons, interviews are transcribed in parallel, and as they become available we perform several passes of coding. To code a transcription, we read each sentence or paragraph and if it helps us answer our general research questions, we summarize the underlying concept with a code (one to three words). If the code representing the concept does not exist yet, we create an entry in our list of codes and add a definition. At the end of a few coding passes, at least one other person in our team reviews the codes, their definitions, and how they were assigned. We then discuss whether we should add, delete or modify codes. For example, after a few participants mentioned that diagrams were not that helpful, it became clear that we needed new codes to qualify “usefulness.”

### 3. Preliminary Insights

Although we have performed only eight interviews to date and are still at a preliminary stage in our study, we are beginning to form answers to our research questions.

At the end of each interview, we ask the participant to describe their sketch. We found this sketch walkthrough to be an excellent description of a project landscape and a tentative answer to our first research question: these sketches often contain small architectural diagrams, overviews of the organization hierarchy, lists of tasks to accomplish, lists of technologies used, etc. We plan to perform a more systematic analysis of these sketches as the study progresses.

We are also seeing patterns emerge from our interviews. We believe that some of the following patterns could lead to the development of useful tools or confirm the value of existing tools.

**Learning by Experimenting.** Four participants mentioned that they primarily learned about their project by writing prototypes and tests exercising the features of the software system. It seems that no matter how many pages of documentation they read, what really helps them is to be able

to run the code. When we asked P1 what information was the most valuable in getting him up to speed, he answered: *“For me it’s really being able to compile your sources to run the product, to run the things and from that point on, you can easily find out how on a technical level, you can easily try to change things yourself, get some experience... what happens if you change that?”*

Unfortunately, for most of the participants, getting their development environment set up was also one of the major difficulties they encountered. For example, it took P1 two weeks to run some code, and after two months P2 said: *“To be honest with you, I don’t consider my development environment completely set up right now. It works for certain tasks, the tasks that I’m involved right now, but I would not be able to immediately switch over and debug and investigate the code that some of the other team members have been working on”*. P1 and P2 mentioned that getting all the required dependencies with the correct versions was difficult. P1, P2 and P4 also mentioned that running code that had just been developed by their new colleagues took them many days of work because there were a lot of manual and complex configuration steps to perform.

Although documentation on the projects’ wikis (e.g., dependency lists) initially helped the participants, most of them primarily relied on personal communication with colleagues to finally get their environments set up: PD and P1 did this through a meeting, P2 through Netmeeting and phone, and P4 by phone. Most of them mentioned that there were always some little details, not documented anywhere, that prevented them from experimenting with the system.

These insights point towards key elements of a project landscape: source code, but more importantly, the ability to quickly execute it and experiment with it. A tool (e.g., a sandbox), or a flexible landscape that allows newcomers to experiment safely with the system early on, would greatly help developers orient themselves. A map pointing to resources would not have sufficed for the participants we interviewed. This corroborates one of the arguments made by Newell and Galliers that learning involves two activities; learning about something (cognitive learning) and learning to do something (behavioral learning) [6]. These authors mentioned that the first activity is often emphasized during knowledge transfer at the expense of the second.

**Colleagues’ Expertise and Interests.** When we started to interview senior developers, we quickly observed a difference from the junior developers in the information they were seeking. All senior developers mentioned, without being specifically asked, that it was important for them to know the expertise and the interests of their colleagues. This was never mentioned by the junior developers.

Although P1, P2 and P4 wanted to learn more about their colleagues, their reasons differed. P1 wanted to learn about the amount of expertise: *“In the end you have to find out*

*who you can rely to and who is making good calls based on experience, and there is a lot of difficult calls”*. P2 wanted to learn about the technical interests of his colleagues so that questions could be directed to the right one. P4 mainly wanted to know about outside interests, to build stronger relationships and team spirit. They all mentioned that it takes time to learn about the various team members and that it requires personal interactions (e.g., meetings and lunches, but not email nor chat). P1 used the bug system as an alternative to learn about who was working on what, to get a sense of his colleagues’ expertise.

These responses provide evidence that the expertise and interests of colleagues are key features of a project landscape. According to our participants, tools like wikis containing the biographies of team members do not provide enough aid to orientation, and no tool can replace face-to-face meetings. We believe, though, that a tool displaying information about what team members are doing could help newcomers. For example, a dynamic page could list the bugs team members recently fixed and the technical web sites and documents they recently visited

**Distributed Teams.** One of our initial selection criteria was to study only co-located teams to avoid the special issues introduced by distributed teams. After trying to find such teams within IBM, we realized that pure co-located teams are the exception and that partially-distributed teams are the rule. In fact, only PD was in a pure co-located team.

All participants mentioned that working in a distributed team is challenging. For example, an important milestone for P1 and P2 was when they met their remote team members in person; it was only then that they felt they were part of the team. P4 also mentioned that he preferred to meet his colleagues and collaborators in person, but that was increasingly difficult to do. The junior developers did not mention this need and were comfortable with chat, emails and conference calls. While perhaps a generational difference, we believe it is a difference in project role, as senior developers’ responsibilities required more remote interaction.

Our third research question addresses practices and tools that could help newcomers orient themselves. Many of these should be applicable to co-located and distributed teams alike, though many issues are more acute in the distributed case. Although remote colleagues are a key part of a project landscape, no tool or quick fix can fill the need to meet them in person and interact with them regularly, an observation that corroborates the findings of Armstrong and Cole on distributed work [1].

**Software Documentation.** In 2003, Lethbridge et al. performed a survey and on-site observations to find developers’ needs for and usage of documentation [5]. As in this previous study, our participants often mentioned that documentation (e.g., design documents, tutorials or code examples) were slightly outdated and incomplete, but as PD said,

“those wikis, they fall out of date, they’re not exactly correct or whatever. But that’s fine, because that’s what I expect them to be as wikis, but also at the same time that means I’m not going directly to them, all the time... because they don’t feel like a first point of knowledge transfer.” Participants generally understood that maintaining documentation is time-consuming and did not mention outdated documentation as a major issue. To cope with the missing or outdated pieces of documentation, participants often relied on emails, mailing lists, or plain trial and error: they considered that coping with incomplete or erroneous documentation was part of their job.

The participants mentioned, though, that older documentation systems that provide large collections of documents were not at all helpful. The inability to search or categorize the content of the documents was a major issue, an observation that corroborates Lethbridge et al.’s conclusions. Our participants said that this issue caused them to have to ask colleagues how to find relevant documents or which document contained the information they were looking for.

Software documentation is an important part of a project landscape. Our study suggests that the navigation aids available to help newcomers find and access relevant documentation are key. Some of the patterns we found point towards more dynamic navigation aids than standard maps or index documents, e.g., tools that help newcomers search and index documents, experiment with the system, or learn about their colleagues’ current expertise and interests.

## 4. Related Work

Studies have been done in the past in related areas such as project integration, knowledge acquisition by software engineers, and knowledge transfer in general.

Begel and Simon observed eight graduate students during their first months of work at Microsoft [2]. The authors found that most of the difficulties encountered by the students came from their inexperience with a corporate environment. Sim and Holt interviewed four recently-hired developers at a big software company and identified seven integration patterns [7]. Results from these studies can help organizations better orient their new employees. By interviewing more experienced developers, we focus on project-specific issues, in line with our research questions.

Regarding knowledge acquisition, we presented the work done by Lethbridge et al. on software documentation [5]. Although this study was performed in 2003, when wikis were less popular than today, most of their findings on outdated and unreliable documentation still hold. In our interviews to date, though, we found that these problems are less severe than they appeared to be when this study was published. Ko et al. observed 17 developers to learn about their information needs when performing a maintenance task [4]. They summarized the results as 20 ques-

tions, such as “What code caused this program state?” and “What have my coworkers been doing?” These questions are helpful in determining whether an issue encountered by a newcomer is part of a general software engineering task or is due to project integration.

Finally, there exists a large body of literature on knowledge transfer and learning. Newell and Galliers surveyed 13 projects in different areas (e.g., healthcare, utilities) and found several reasons why knowledge transfer among organizations is problematic [6]. Although our study focuses on individuals, we hope to find similar patterns and solutions and build on these previous studies.

## 5. Conclusion

When developers join a project, they must orient themselves within the project landscape. In this paper, we presented the study on project integration we are currently performing. By following an approach based on grounded theory and by interviewing developers who recently joined a project, we want to discover the nature of project landscapes and what features aid or inhibit integration. For example, early results suggest that newcomers find it more important to be able to experiment with the system than to have up-to-date and complete documentation. The long-term goal is to understand and improve the project landscapes in which software developers work.

## References

- [1] D. J. Armstrong and P. Cole. Managing distances and differences in geographically distributed work groups. In P. J. Hinds and S. Kiesler, editors, *Distributed Work*, pages 167–186. The MIT Press, 2002.
- [2] A. Begel and B. Simon. Struggles of new college graduates in their first software development job. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 226–230, 2008.
- [3] J. Corbin and A. C. Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, 3rd edition, 2007.
- [4] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proceedings of the 29th international conference on Software Engineering*, pages 344–353, 2007.
- [5] T. C. Lethbridge, J. Singer, and A. Forward. How software engineers use documentation: the state of the practice. *IEEE Software*, 20(6):35–39, 2003.
- [6] S. Newell and R. Galliers. Knowledge transfer: Short-circuiting the learning cycle? In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, pages 149–b, 2006.
- [7] S. E. Sim and R. C. Holt. The ramp-up problem in software projects: a case study of how software immigrants naturalize. In *Proceedings of the 20th international conference on Software engineering*, pages 361–370, 1998.