Turnover-Induced Knowledge Loss in Practice

Martin P. Robillard
martin@cs.mcgill.ca
School of Computer Science
McGill University
Montréal, QC, Canada

ABSTRACT

When contributors to a software project leave, the knowledge they hold may become lost, thus impacting code quality and team productivity. Although well-known strategies can be used to mitigate knowledge loss, these strategies have to be tailored to their target context to be effective. To help software development organizations mitigate turnover-induced knowledge loss, we sought to better understand the different contexts in which developers experience this knowledge loss, and the resulting implications. We conducted qualitative interviews with 27 professional developers and managers from three different companies that provide software products and services. Leveraging the experience of these practitioners, we contribute a framework for characterizing turnover-induced knowledge loss and descriptions of the implications of knowledge loss, synthesized into 20 observations. These observations about knowledge loss in practice are organized into four themes, validated by the participants, and discussed within the context of the research literature in software engineering.

CCS CONCEPTS

• **Software and its engineering → Software development process management**; **Collaboration in software development**; *Software evolution*; *Maintaining software*; *Documentation.*

KEYWORDS

Knowledge loss, knowledge sharing, knowledge management, documentation

ACM Reference Format:
Martin P. Robillard. 2021. Turnover-Induced Knowledge Loss in Practice. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3468264.3473923

1 INTRODUCTION

When contributors to a software project leave the project, the knowledge they hold may become inaccessible, or *lost*, to the remaining team members [12, 38]. In software development, knowledge loss impacts quality [15, 32] and productivity [41], among other factors [42]. In the words of an experienced software developer we interviewed about the impact of knowledge loss: "We have to reverse engineer and sometimes we have to look for knowledge. We have to find something, which was probably written somewhere before, and the biggest impact is how fast we can deliver the solution".

Numerous high-level strategies exist to mitigate knowledge loss in software development, which can focus on personal, project, or technology factors [35]. These include, for example, learning by experimenting, maintaining a clear documentation process, or substituting verbal communication with an instant messenger. While it is useful to be informed of the options available, knowledge management practices have to be applied in a specific context. Only with a shared understanding of the context in which knowledge is lost can the practicalities and trade-offs involved in its mitigation become actionable. The goal of our research is to help software development organizations with the implementation of mitigation strategies for turnover-induced knowledge loss. For this purpose, we sought to better understand *1)* the different contexts in which developers experience turnover-induced knowledge loss, and *2)* the implications of knowledge loss in practice.

To learn about knowledge loss in practice, we conducted qualitative interviews with 27 professional developers and managers from three different companies that provide software products and services to commercial enterprises. We analyzed the interview transcripts and leveraged the statements of the participants to elaborate a systematic approach for describing knowledge loss contexts, and to capture descriptions of the implications of knowledge loss in practice. We provided all the participants with a summary of the findings and, of nine respondents, the majority confirmed the credibility and usefulness of these findings through a questionnaire.

The first contribution of this paper is a framework for characterizing turnover-induced knowledge loss, applied to 41 workplace situations described by the participants, and also validated by the participants. For example, we noted, among others, that the departure of a knowledge owner does not necessarily imply a complete disengagement from the project, and that internal transfers within a company can amount to a complete departure from the point of view of knowledge transfer. The second contribution is a set of descriptions of the implications of knowledge loss, synthesized into 20 observations organized into four themes, validated by practitioners, and discussed within the context of the broader research literature in software engineering.

The remainder of this paper is organized as follows. In Section 2, we present relevant concepts of knowledge management theory and survey the related work on knowledge loss in software engineering. In Section 3, we describe our research methods. Section 4 reports on knowledge loss contexts, and Section 5 reports on the implications of knowledge loss in practice. In Section 6, we present the results of the validation survey. We synthesize the insights and implications of the work and conclude in Section 7.

2 BACKGROUND

This investigation of knowledge loss in practice is informed by existing *theories of knowledge management* as well as related work on *knowledge loss in software engineering*.

2.1 Theory of Knowledge Management

The general goal of this work is to help improve **organizational learning** in software development organizations. Schneider summarizes organizational learning as the "learning of individuals, organization-wide collection of knowledge, and cultivation of infrastructure for knowledge exchange" [46, p.3]. Consistently with Schneider, we use Sunassee and Sewry's definition of **knowledge** as "the human expertise stored in a person's mind, gained through experience and interaction with the person's environment" [49].

Various models have been proposed to capture how knowledge flows within organizations [11]. As a reference model of knowledge management life-cycle, we use Nonaka and Takeuchi's theory of knowledge creation [36] as it is both well-known and enduring [11, p.79]. One of the main features of this model is the distinction between tacit and explicit knowledge. **Tacit knowledge** refers to knowledge that cannot easily be communicated through writing or verbalization [40], whereas **explicit knowledge** refers to the kind that can be codified.

Figure 1 shows the Nonaka and Takeuchi model [36]. The figure is a matrix where each cell represents a type of **knowledge conversion**. The rows represent the initial type of knowledge (tacit vs. explicit) and the columns represent the final type of the conversion (again, tacit vs. explicit). For example, **socialization** is the sharing of tacit knowledge directly from one person to another, whereas **externalization** is the conversion of tacit knowledge to explicit knowledge (usually by writing it down, but also through diagrams, etc.). Linking explicit knowledge is called **combination**. Finally, learning by doing leads to the **internalization** of knowledge. In the model, the spiral represents the learning process, with the diameter of the spiral representing the growing amount of knowledge acquired over time by repeatedly creating new knowledge through conversions.

Many books exist that describe **knowledge transfer practices** and **barriers to knowledge sharing** [12, 23, 37, 50]. While such reference texts are useful surveys of the general state of the practice of knowledge management, this paper offers a more focused view of the experience of knowledge loss in software projects, specifically in the context of turnover. We define **turnover** as the departure of a contributor from a software project in which they are actively engaged. Turnover can be voluntary or involuntary [48], and is generally associated with decreased project performance due in part to knowledge loss [15, 39].

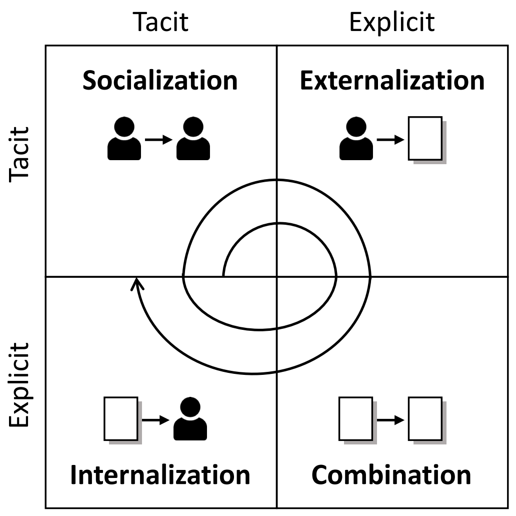

Figure 1: Nonaka and Takeuchi's model of knowledge creation. Adapted by permission from Springer Nature: Springer, Experience and Knowledge Management in Software Engineering by Kurt Schneider, © 2009 [46].

2.2 Knowledge Loss in Software Engineering

In software engineering, contributions to our understanding of knowledge loss can be organized in terms of *literature reviews*, *case studies*, and *data-mining studies*.

Literature Reviews. Nidhra et al. provide a literature survey of the knowledge transfer challenges and mitigation strategies in global software development [35], while Rashid et al. contribute a conceptual map of the literature on knowledge loss related to open-source software, summarizing, in particular, the different types of impact that knowledge loss has on open-source projects [42]. These include, for example, abandoned code and project instabilities. Also drawing on the literature, Kukko and Helander review 22 knowledge sharing barriers, and map them to different companies based on how the companies increased in size [22]. In terms of the overall literature, Manhart and Thalmann find that "the existing body of literature primarily focusses on the formal and the legal dimension of knowledge protection, while neglecting the tacit dimension" [27].

Case Studies. A number of studies directly investigated how knowledge gets lost in software projects. In particular, Feilkas et al. focus on how architectural documentation gets out of date through an industry case study [14]. Based on their experience with medium-size companies, Durst and Wilhelm emphasize the challenge of promoting knowledge management to a first class concern, noting that "management appears to be too busy with the day-to-day running of the business to give [the risk of knowledge loss] a high priority" [13]. In a similar vein, Viana et al. observed knowledge creation and loss within a company. They discuss, among others, the challenges of externalizing knowledge, and the issue that externalized knowledge may end up not being used [52]. Finally, through an interview-based case study, Mitchell and Seaman studied how knowledge flows within one company, identifying several obstacles, including "inability to efficiently locate explicit knowledge (documents) from the project's online database" [31, p.167], and a tension between engineers' perceived need to capture knowledge from impromptu interactions and the manager's belief that this is not cost-effective.

Data-Mining Studies. Researchers have also studied knowledge flow in software projects by mining software repositories. As general context for the study of knowledge loss, Sowe et al. provided an initial insight into knowledge sharing practices in open-source software through a study of mailing lists [47]. By mining project repositories, Mockus showed that "recent departures from an organization were associated with increased probability of customer reported defects, thus demonstrating that in the observed context the organizational change reduces product quality" [32]. Similarly, Bird et al. found that high levels of code ownership were associated with fewer defects [3]. Studying turnover in global open-source projects, Lin et al. observed that developers who produce documentation do not remain as long as coders in software projects [25]. Foucault et al. found that "the activity of external newcomers negatively impact software quality" [15]. As for knowledge loss specifically, Izquierdo-Cortazar et al. introduced the idea of modeling knowledge loss in terms of lines of code contributed by a developer who left the project [19]. This idea was further refined by Rigby et al. [43], who modeled the phenomenon probabilistically as a risk [20], and Nassif and Robillard, who added additional parameters to the analysis [33]. Other researchers have used various techniques applied to software repositories in an attempt to estimate the so-called "bus factor" metric, namely the number of key developers whose departure would threaten the project [2, 7], or to predict project productivity when accounting for knowledge loss [34]. Mirsaeedi and Rigby follow up on the investigation of knowledge at risk with the suggestion to distribute knowledge through code reviews to reduce the files at risk of turnover [30]. Although the use of metrics allows us to take into account a huge amount of process data to make observations, the modeling of knowledge in terms of source files is very approximate [45].

3 METHOD

We explored the experience of professional developers and managers dealing with turnover-induced knowledge loss situations by conducting qualitative interviews with participants recruited from three software development companies. We then transcribed the interviews and analyzed the transcripts qualitatively.

3.1 Companies and Participants

We recruited 27 participants from three companies (see Table 1). Company A is a multinational that develops enterprise solutions. The participants from Company A are members of a department in charge of software maintenance. Company B is a small company that develops and operates accounting software for affiliated companies. Company C is a small company that specializes in integrated on-line sales solutions. We recruited the participants via personal contacts. Our recruitment goal for this study was to involve participants from multiple companies and to interview a sufficient number of practitioners to be able to explore the knowledge loss phenomenon from multiple personal perspectives. Within these parameters, the number of participants was bounded pragmatically by the difficulty of diverting practitioners from their professional activities and the high cost of analyzing qualitative interviews.

Table 1: Characteristics of the study participants

ID	Role	Experience	Seniority
Company A			
A1-Man	Development Manager	●	●
A2-Man	Development Manager	●	●
A3-Man	Development Manager	●	●
A4-Man	Development Manager	●	●
A5-Man	Development Manager	●	●
A6-Arc	Software Architect	●	●
A7-Arc	Software Architect	●	●
A8-Dev	Software Developer	◓	●
A9-Dev	Software Developer	◓	●
A10-Dev	Software Developer	◓	●
A11-Dev	Software Developer	●	●
A12-Dev	Software Developer	●	●
A13-Dev	Software Developer	●	●
A14-Dev	Software Developer	●	●
A15-Dev	Software Developer	●	●
A16-Dev	Software Developer	●	●
A17-Dev	Software Developer	●	●
Company B			
B1-Exe	Company Executive	●	◓
B2-Man	Development Manager	●	●
B3-Man	Development Manager	○	○
B4-Dev	Software Developer	○	◓
Company C			
C1-Exe	Company Executive	◓	◓
C2-Man	Development Manager	○	◓
C3-Dev	Software Developer	◓	◓
C4-Dev	Software Developer	◓	○
C5-Dev	Software Developer	○	◓
C6-Dev	Software Developer	○	○

Table 1 provides, for each participant, their main role within the company, their level of experience, and their seniority within the company, all using generalized descriptors. We discretized the participants' level of experience and seniority into three levels. For general workplace experience, we distinguish between participants within five years of graduation or in their role as an IT worker ○, participants with over ten years of relevant industry experience ●, and participants in-between the two ◓. For seniority within the company, we distinguish between participants who have been with the company for two years or less ○, from participants who have been with the company for over seven years ●, and participants in-between the two ◓.

(1) Your official job title and brief description of your role and responsibilities;
(2) Brief summary of your work experience, including graduation year, years with Company X, years and roles with previous organizations (if appropriate);
(3) Can you think of situations where you needed information that only one or a few colleagues had, or that had been forgotten or lost?

Figure 2: Pre-study questionnaire

As the table shows, the three companies provide different environments for studying knowledge loss in practice. Participants from Company A are all veterans of the company. Company B presents a dichotomy, with two senior participants and two junior participants, with corresponding tenure time in the company. As as relatively new company, Company C presents a wide spectrum of expertise and seniority, from founding members in major positions of responsibilities to recent junior hires.

3.2 Data Collection

We sent participants a pre-study questionnaire (Figure 2) along with a consent form and a request to schedule an interview.

The goal of the interviews was to get rich descriptions of the participants' experience with turnover-induced knowledge loss. To this end, we employed an open-ended, loosely-structured style of qualitative interview [53], which consisted of asking participants to elaborate on the situations they referenced in Question 3 of the pre-study questionnaire. We conducted all 27 individual interviews through a videoconferencing software. We audio- and video-recorded the interviews. As none of the participants used screen sharing, we only analyzed the audio portion of the recordings. The author conducted all interview sessions. During the interview, the investigator limited his role to asking clarification questions and prompting the participant for additional details. The interviews lasted between 10 and 45 minutes (mean=25 SD=8.7). We conducted all but three of the interviews in English, with the remaining being conducted in French. This protocol was reviewed and approved by the Research Ethics Board of McGill University.

3.3 Data Analysis

We used an automated transcription software followed by a manual cleaning process to convert the audio recordings to textual transcripts. We structured the data analysis into two parts, with the first part focusing on understanding the *contexts* of knowledge loss and the second part focusing on the *implications* of knowledge loss from the point of view of practitioners.

Contexts for knowledge loss. We sought to better understand the *different contexts in which practitioners experience turnover-induced knowledge loss.* We analyzed the transcripts to identify data extracts in which a participant described a project departure context that resulted in knowledge loss. A *data extract* corresponds to a cohesive unit of action or verbalization by the participant, roughly equivalent to a few sentences in the transcript. We collected a set of 41 extracts about knowledge loss contexts from 25 participants across all three companies (two of the 27 participants had no contribution to this part of the study). By inspecting the data, we devised an original framework for characterizing knowledge loss contexts. This framework posits three orthogonal dimensions for describing the departure of a knowledge owner. The three dimensions are whether the departure was *permanent* or *temporary*, whether it was *sudden or anticipated*, and whether it was *complete or partial*. We selected these dimensions analytically, but based on the variation points we noted in the turnover situations described by participants. For each of the 41 knowledge loss contexts under study, we noted the type of departure using a small and closed set of codes created from a preliminary review of the data. These codes are reported as the content of the cells in Table 2. For the three dimensions, we used either one of the two applicable values (e.g., *complete* or *partial*), or the special value *unspecified* if the extract was not explicit about a dimension. As an example, we consider the following extract: "...and sometimes after [the handover] the team might disappear because they are not working on this project anymore [...]. They left, [...] they are assigned to another project and they are busy: we cannot even ask them questions..."—*A11-Dev*. We coded this extract as an *internal transfer* that was *permanent* and *complete*, and *unspecified* in terms of gradual vs. sudden.

Implications of knowledge loss. We sought to better understand *the implications of knowledge loss in practice*, using an inductive data analysis strategy. We structured our analysis using the reflexive thematic analysis framework proposed by Braun and Clarke [4]. Thematic analysis is a "method for systematically identifying, organizing, and offering insight into patterns of meaning (themes) across a data set" [4, p.57] that is organized in six phases. First, we studied all transcripts (*Phase 1: Familiarizing Yourself With the Data*) and *coded* all relevant data extracts using *open coding* (a procedure whereby an investigator associates a free-form label, or *code* to a data extract) (*Phase 2: Generating Initial Codes*). We followed a *process coding* approach, using gerunds to denote "observable and conceptual actions in the data" [29, p.75]. With this step, we identified 134 data extracts annotated with 22 distinct codes (e.g., *hoping that others remember*, or *trying to guess the intent of the code*). The investigator then studied the distribution of initial codes and grouped them into themes (*Phase 3: Searching for Themes*). As a result of this step, we mapped 20 of the 22 initial codes to the four themes described in Table 3 and dropped two because they were not well supported by evidence. The investigator then reviewed all coded data extracts, this time organized by theme as opposed to participant (*Phase 4: Reviewing Potential Themes*), and noted observations that capture the essence of the theme (*Phase 5: Defining and Naming Themes*). We completed Phase 6 (*Producing the Report*) concurrently with Phases 4 and 5. We use an example to illustrate our analysis process. During the initial coding, we noted the following extract as relevant: "...they're very, in their own world: it's hard to break that product out of there. [...] They're protective of the knowledge, they really don't want to share it that easily, because it's their bread and butter, their design."—*A6-Arc*. We then coded this extract as *facing knowledge sharing resistance*, one of the 22 initial codes. In the subsequent phase, we grouped this code with five related codes to create the theme *working with colleagues*.

Table 2: Contexts for turnover-induced knowledge loss. Each cell represents the type of context that has the attributes in the corresponding row and column. The pie charts visualize the hypothesized level of impact on knowledge loss, from high (•) to low (○). Contexts that were not directly recorded from participants are *in italics*.

	Permanent		**Temporary**	
	Sudden	**Anticipated**	**Sudden**	**Anticipated**
Complete	● Death, departure (minimum notice)	◕ Retirement, departure, internal transfer	◓ Unanticipated leave	◔ Weekend, vacations, anticipated leave
Partial	◓ Departure or internal transfer (minimum notice)	◔ Retirement, departure, internal transfer	◔ *Unanticipated leave*	○ *Weekend, vacations, anticipated leave*

3.4 Quality and Credibility

With the selection and application of our research method we strove for high standards of *quality* (are the findings thoughtful and useful?) and *credibility* (do they truly reflect the participants' experiences with a phenomenon?). These evaluation criteria are more relevant for a qualitative study than the usual *threats to validity* associated with quantitative studies [9, p.202]. The goal of this study is not to generalize a phenomenon observed in a sample to a population: instead we seek to synthesize the salient aspects of a multi-faceted human situation through rich descriptions.

To ensure that our findings had an adequate level of quality and credibility, we asked the original participants to review a preliminary version of this report and provide feedback (a practice also known as *member checking* [8]). After the data analysis was completed, we sent all 27 participants the preliminary report along with a feedback questionnaire. The report consisted of a one-paragraph introduction followed by a preliminary version of Sections 3–5 of this paper. One email invitation bounced and, of 26 recipients, we received nine responses with at least one participant for each company. We provide the survey details together with the results in Section 6.

Limitations. The number of participants in our study is inevitably limited, which creates the risk that we may have missed important themes or modulations of a theme. In particular, we did not attempt to assess the amount of saturation we achieved with our data set, as this practice is questionable in the context of reflexive thematic analysis [5]. In consequence, the themes we describe should be considered a shared but partial view of the knowledge loss experience. The quality of our observations is also affected by our use of interviews to elicit the participant's experience. We may have missed insights due to some participants imperfectly recalling their experience or verbalizing their thoughts. These limitations were, to the extent possible, mitigated through interview techniques (relevant prompts and questions), and triangulation between participants.

4 CONTEXT FOR KNOWLEDGE LOSS

We sought to better understand the workplace contexts that engender knowledge loss. To be able to reason about such contexts analytically, we devised a framework that characterizes project departure events along three dimensions: **permanent vs. temporary**, **sudden vs. anticipated**, and **complete vs. partial**. Table 2 presents the three dimensions with their corresponding contexts types and hypothesized level of impact on teams. We leverage our interviews to provide rich descriptions of each dimension. We provide additional data to validate the framework in Section 6.

4.1 Complete vs. Partial Departures

A *complete* departure means a knowledge owner is no longer available for knowledge sharing. The canonical case is that of "a guy passed away"—*A13-Dev*, but employees can also simply leave without trace: "I personally have no idea even how to contact him and where to find him or whatever"—*A1-Dev*.

In contrast, a *partial* departure means a knowledge owner remains available for knowledge sharing on a discretionary basis: "They ended up giving their numbers, they can call them whenever"—*A6-Arc*; "Fortunately he left personal contact information and he was nice enough, he said 'OK, now I'm retired I finally have some time in order to be able to write down [...] knowledge points that I think you should know in order to do well in supporting this particular product'."—*A15-Dev*.

Although developers who transfer internally [17] technically remain reachable, their departure can be considered *complete* if, as a result of their transfer, their knowledge is lost. "How can he forget? Because they developed it. You know they worked on that for very many years..."—*A12-Dev*; "So even if I send an email right now about the feature the developer built 10 years ago, he will be like 'what are you talking about? I have no idea"'—*A7-Arc*; "I told my colleague 'can you please help me on this because I don't know they did it.' He took time to check it. 'I'm so sorry I don't remember at all."'—*A12-Dev*.

Knowledge seekers may also have reservations about contacting departed knowledge owners, which contributes to making a departure *effectively* complete even when the knowledge owner remains accessible: "I guess there would have been a way to get in touch with him, but often times when a colleague has been gone for a little while, we try and not get involved too much, unless we have no choice."—*A9-Dev*; "So they developed this, they finish [...] so you know that application, it's not a priority anymore."—*A10-Dev*.

4.2 Sudden vs. Anticipated Departures

A departure is perceived as *sudden* if normal expectations about knowledge transfer cannot be met. Even when an employee gives their legal notice, their departure can be felt as sudden by their teammates: "Two weeks of notice to go and so that was hard because, the functionality was developed by him, but it was new. [...] So it was very surprising, because normally, to [transfer the knowledge] we have to plan and the other people should be available."—*A12-Dev*

Table 3: Implications of knowledge loss in practice: main themes

Theme	Observations
Lacking Guidance and Information	Practitioners can be left *feeling uncertain about a specification* or *guessing the intent of the code.* Practitioners may feel they lack *tacit knowledge* about how to solve problems, or relevant *domain knowledge.* Practitioners can feel they have to *learn on their own.*
Relying on Documentation	Documentation can *sometimes replace expert knowledge.* However, documentation can also be *missing* or *disorganized and hard to find or search.* The *format of the documentation may be unsuitable for the task*, or its *quality may be poor.* As either an alternative to or replacement for documentation, practitioners can *leverage the project history*, which has its own inefficiencies.
Working with Colleagues	It can be possible to *retrieve some knowledge from colleagues*, which can require *relying on informal personal networks.* However, *social knowledge transfer can require explicit investment*, and can be *inefficient* because of its *informality* and the need to *involve a lot of people.* The process can also be *complicated by personality factors.*
Recreating the Knowledge	Practitioners can attempt to *recreate the lost knowledge by doing their own research and self-directed learning.* However, *the high level of effort required for this activity can interfere with work. Recreating the knowledge can be a social activity*, which may *need to include some of the non-developer stakeholders.* Ultimately *some of the recreated knowledge becomes documented.*

Retirements and other *anticipated* departures, such as leaves and internal transfers, allow for some knowledge sharing: "we just plan around my vacation"—*C5-Dev*; "Before he left I asked him to identify pull requests that contain all the [relevant code]. And then I asked him to walk me through all these steps and those pull requests became kind of a reference implementation."—*C1-Exe*; "So you have still a 25 year gap on that, but they're starting to give what they can before retirement"—*A6-Arc*.

4.3 Permanent vs. Temporary Departures

Departures are *permanent* when the knowledge owner is not expected to return to the project: "so I can go back to the architect [...] who developed it 10 years ago, but it will not be useful because they moved on, you know"—*A7-Arc*. In contrast, with *temporary* departures, the knowledge owner is expected to return to the team. Although leaves and other temporary departures are not typically associated with turnover, we noted that, in practice, temporary departures can produce impacts similar to other knowledge loss contexts.

Temporary departures are problematic, for example, when the *information needs* cannot be anticipated in advance of a leave: "When people take vacation or are sick, requests that they would normally handle can come up and, there are a few of us that will try and take those over when they are gone, and sometimes they're requests that we have never experienced before and don't have any documentation for, so we don't know how to do them"—*B4-Dev*; "On weekend support you're responsible for any incoming critical customer incidents from any of these components or areas. So these are often in the times that you find that on the weekend you receive [many] customer incidents for an area which you know nothing about."—*A15-Dev*

In one interesting case, developers in Company C were contractually requested to decrease their involvement in a project and forgot crucial information they no longer used frequently: "From one day to the next [the client] decided to scale down development but to keep us on, but only a few hours per month. [...] At the beginning were were four full-time developers on this, so there was a lot of information we hadn't really documented, but that now that we only work on this eight hours per month, we realize that there are some information we forgot..."—*C2-Man*.

5 IMPLICATIONS OF KNOWLEDGE LOSS

We sought to better understand the experience of software development professionals faced with situations of knowledge loss. Through our analysis process, we determined that these experiences can be organized into four main themes. Table 3 summarizes the main observations we made for each theme. The observations are indicated in *italics* in the table and in **bold** in the detailed description of the themes.

5.1 Lacking Guidance and Information

What's the intent behind this code? What was he trying to do, here?—*A1-Man*

The theme *lacking guidance and information* generally captures the type of knowledge lost and the disorientation caused by the disengagement of a knowledge owner, beyond the usual needs that motivate information seeking [6, 21, 26].

Some participants described how they **felt uncertain about a specification**, indicating insufficiently externalized knowledge. For example, because of the departure of a team member: "we never knew what should be the right behavior"—*A8-Dev*. Although features may be documented, requests by customers can involve subtleties that are not covered by this documentation, in which case only the original designer was in a unique position to comment on whether a customer request is consistent with a feature's design *[A7-Arc]*: "Should we assume that the customer is right or wrong about what they are expecting? Because if the code is saying something else, we don't necessarily take it as this is the right and expected behavior [...] So most of the questions are to make sure that we are covering exactly the scenarios of the customers"—*A5-Man*. The question of distinguishing observed from specified behavior can motivate reaching out to knowledge owners who may still be partially available (Section 4.1): "We still have contact with the solution manager and [...] we still contact them and we still ask them 'is the functionality doing what it's supposed to do?"'—*A5-Man*.

Similarly, developers lacking a ground truth about the rationale for design decisions can also be left **guessing the intent of the code**: "It's been a while and most of the people who developed this have moved on so [...] there's a few things that were completely ignored and the question is, why did they ignore these?"—*A11-Dev*. These experiences can also be associated with a limitation of knowledge externalization, which is notoriously difficult for design rationale [18]. The implication is that developers must *recreate this knowledge* (Section 5.4): "What's the reason behind doing it in such way? So that reasoning here the person would have lost it and we would have to figure it out ourselves"—*A8-Dev*.

Participants also referred specifically to the **lack of tacit knowledge** caused by the departure of a knowledge owner. Describing how a developer about to move to another project was providing a knowledge transfer session, *A8-Dev* mentioned: "He might be able to give us the overview of that solution [with] certain technical points, but he would not be able to give us the [...] particular *knack* in that solution". Participant *A11-Dev* referred to this tacit knowledge as "Some *hints* that can drive a problem resolution", elaborating: "you can have the knowledge, it's there, but still there is some fine tuning in the heads of people". Some participants stressed the importance of practical experience: "so there's some documentation, but not much practical experience"—*A14-Dev*; "So handover session theory is good. But practice is absolutely a different thing. When somebody explains to you, you think you understood everything, you know everything. When it comes that a customer has this issue, this is a different story."—*A2-Man*

This observation is an important reminder of the limitations of externalized knowledge and that it cannot completely replace socialization and internalization in the process of knowledge creation (see Figure 1). In the words of a participant from Company B: "I had never had to really run the program and do these steps on my own. So when I had to do it, it was basically reading through those notes and having to figure out and understand what was what I meant by it."—*B2-Man*

Related to a lack of tacit knowledge, some participants indicated that they were impacted by the **lack of domain knowledge** owned by the departed colleague, e.g: "it's not like software knowledge, it's more of a business knowledge of the [customer]"—*A6-Arc*. Finally, some participants noted **having to learn on their own**: "[Alone] it will take much more time, but it's always possible [...] but unfortunately it's stressful..."—*A12-Dev*. In a different scenario, a developer who was the only employee with PHP knowledge left and his successor commented: "It wasn't that bad, but I did have to learn everything on my own because not necessarily anyone in the company was a PHP expert"—*C6-Dev*.

5.2 Relying on Documentation

Reading the code is not enough, right? You need really detailed explanations, and then that's when you would look for documentation. But in some cases it's either not good enough, the documentation, or it's not, well, it's not there anymore at all.—*A1-Man*

In the ideal case, **documentation can replace expert knowledge** *[C1-Exe]*. "If you have an SDD [Software Design Document] or if you have a document, even if it's not an SDD, but at least a document that's explaining the feature or what was developed [...] So if this is given and stated then we have no problem"—*A7-Arc*. However, among the observations collected, this experience was in the minority. Instead, most of the participants commented on the obstacles to using documentation. The challenges of documentation are well-known in the knowledge management literature: "Unfortunately, many documentation processes are broken, or nonexistent, and leaders don't realize it until the lost knowledge actually threatens operations." [12, p.89]. Issues with documentation have also been reported extensively in the software engineering literature (e.g., [1, 24, 51]). The following observations record the experience of developers specifically in the context of turnover.

In the worst case, **documentation can be missing**: "My boss would give me some pointers like "oh this guy did it I think he wrote some documentation somewhere", but I couldn't really find anything"—*C5-Dev*; "The moment the person leaves [the company], [...] they cleaned up everything, including the recordings. So the recordings were lost."—*A17-Dev*

Relevant information may be **disorganized and hard to find or search**: "It's an Excel file. Some of it is a Word document. Some of this is a recording from an old [proprietary format] that you're not even sure how to play anymore. [...] There's no standard basically, so it's very time consuming."—*A1-Man*; "If even the documentation was somewhere stored on some servers at the company by this custom dev team we might not even know where it is."—*A3-Man*. One participant even mentioned that they needed their expert in the first place, just to find the documentation: "[The information] is there, but you have to find it. So sometimes you have to ask the expert or colleagues with more experience just to point you to the place"—*A13-Dev*.

The **documentation format may be unsuitable for the task**. Numerous design trade-offs are involved when designing documentation [10], and the format employed may ultimately be mismatched to the information needs. This was noted in particular for video recordings: "and when we have specific questions and you have to go through the whole video one hour in order to extract some useful information and people, they just get fed up"—*A11-Dev*; In particular, documentation may be judged to be too "high-level" to palliate the loss of expert knowledge: "[the recording] was super high level, basically not really useful"—*A1-Man*. Finally, documentation may be ineffective at supporting knowledge internalization: "The documentation from development doesn't really help sometimes because it's really about how they designed it. The best knowledge is sometimes the one that you learn in the process"—*A10-Dev*.

In addition to not being in the right format, the documentation's **quality may not be sufficient**: "or we get the documentation, but, this is incomplete or not very well written, and it's confusing"—*A3-Man*, in particularly with respect to being up-to-date: "The document was not up to date, it's already probably lost some of the new enhancements"—*A4-Man*.

As either an alternative to or replacement for documentation, developers can **leverage artifacts from the project history** (email, issue reports, etc.): "if it's really a technical question, usually I can read the code and then find the answer or we use our support system"—*C1-Exe*; "after his departure I was trying to go through all his old solved incidents"—*A15-Dev*; "When again customers started asking us questions it was not easy to find this information because the person was gone. [...] Sometimes it's really hard to find the email which was there three years ago..."—*A2-Man*; "What I generally do is doing some search for previous incidents and find out how the colleague was solving that. So it give some bits and pieces."—*A14-Dev*.

5.3 Working with Colleagues

That's the ideal situation: that they have given enough knowledge to everyone else, like it's a beehive. You're hoping that everyone got a little piece of that knowledge so that it could be retrieved somehow.—*A6-Arc*

Participants commented on their attempts to **work with other colleagues to recover some of the lost knowledge**, i.e., to primarily rely on socialization to mitigate knowledge loss: "The only issue if they left [the company], then we can seek this help from other colleagues that are still around. Or some colleagues that got the knowledge transfer from this guy before he left"—*A11-Dev*; "You would look at the documents that exist, and pretty much every single time you can see the name of the authors [...] So we would try and get in touch with them once we saw that they were involved with the projects, just sending an email and see if they could get in touch with us"—*A9-Dev*. According to De Long, socialization is most appropriate to transfer "implicit know-how" type of knowledge [12, Figure 6.2].

Socialization-based knowledge conversion may require **relying on informal personal networks**: "Sometimes you can go through the personal touch because those who work on the same topic, they know who they are"—*A4-Man*; "So I go back to the developer: "ah please", because we have a very good relationship together, [our department] and development, and we know each other very well"—*A12-Dev*. However, **socialization-based knowledge transfer can require explicit investment**: "So a lot of the times the best way to start the whole process is you send three–four colleagues there for two weeks. They interact with them, they go out. [...] You start building a kind of human interaction before you can get any other knowledge out"—*A6-Arc*; "We build good relationship usually during the handover...."—*A5-Man*; "That's why you have a manager that hopefully knows a bit more the people involved in what group. But this is very unstructured, and that's why we have a mentor system"—*A1-Man*.

Although a potential safety net for turnover-induced knowledge loss, socialization is not without its inefficiencies. These **inefficiencies** can be associated with the **informality of the process**: "So you contact a person in [another country] that's been there for a while and he knows this person that knows, and he knows the other person..."—*A6-Arc*. Disengaged members also have other priorities "And sometimes after [the handover] the team might disappear because they are not working on this project anymore [...] we cannot even ask them questions: sometimes you send an email and it stays there for two weeks, three weeks"—*A11-Dev*. It may also be **necessary to involve many people** to cover all the knowledge required: "we had to involve too many stakeholders from development teams or previous product owners, or people who have a kind of small knowledge about that topic. So we had to get in touch with all of them to understand the behavior and then make a single piece of correction line."—*A8-Dev*; "I had [the leaver's advice] in the back of my mind as well and used that to help work with other people on our team with providing some information about what I was looking for to see if they could help fill in some blanks and they were able to."—*B2-Man*

Finally, **personality factors can also complicate the process**: "Sometimes you have this kind of human relation that they feel that, 'oh, I'm really busy, I'm going to come back to you later on, you know, when I feel like it"'—*A7-Arc*. A specific knowledge transfer barrier related to working with colleagues is reluctance to share the knowledge [16]: "They're protective of the knowledge, they really don't want to share it that easily, because it's their bread and butter, their design..."—*A6-Arc*.

5.4 Recreating the Knowledge

We had to rebuild the knowledge because we knew the only thing we know is what [the software] is supposed to do [...] But of course we wasted much more time than if we would have had the expert from the start.—*A1-Man*

Participants described their attempts at **recreating lost knowledge by doing their own research and self-directed learning**: "I used to use the [learning resources] from old colleagues [...] Because they were supporting [the software] and they were developing in it, so they gave hints in their books. So this kind of information I'm acquiring myself right now as well"—*A11-Dev*. Many participants pointed, however, that **the effort required for this activity interfered with their work**. Following an incomplete knowledge transfer session, a participant noted: "I remember I had to work for at least three or four days [...] maybe more than eight hours a day to be able to just understand, "OK, where is the issue"..."—*A12-Dev*; Other participants opined: "you have reverse engineering stuff. You have your code, you debug, and you learn so we can build up the knowledge back. But it takes some time."—*A13-Dev*; "But over time you [...] kind of reverse engineer, figure out how the application is designed. [...] So generally this kind of knowledge gap will slow down the support so you know the customer might wait..."—*A14-Dev*; "We allocated time for me and another person. We basically meet up every day and we try to see how it works inside..."—*A17-Dev*.

Participants also noted how **recreating the knowledge can be a social activity**, implying that the observations noted for the theme *working with others* may apply in this case as well. "And individual questions, you know. Gradually I would say gain some knowledge in that way. After that, people get better. We have some expertise in my team as of now."—*A4-Man*; "And the only way is to learn is by debugging, and come to our conclusion and try as a group too see what makes the most sense for the customer"—*A5-Man*.

In some cases, the other people involved may **need to include some of the non-developer stakeholders**, and in particular domain experts: "I started interviews with accountants. I started interviews with the original developer who is pretty good at sending me an email with kind of a stream of consciousness"—*B1-Exe*; "It's more of a meeting, myself and another person and my boss actually would then meet and go, and talk through what does it do? How does it work? How do we want it to work? And we would basically reverse engineer and decode things based on our knowledge of what we do inside of our company"—*B2-Man*.

As **some of the recreated knowledge becomes documented**, the issues of ensuring documentation accessibility and quality need to be considered: "Any way that I can find on the company's share drive, in whatever notes or a memo, I tried to collect and put them together and I try to restudy this software product on my own. But this is not an easy process."—*A15-Dev*; "Sometimes they're requests that we have never experienced before and don't have any documentation for, [...] and that's where our support documentation started building, as one of our team members had put together a document about different scenarios that can come up that they normally handle, that then we can use as a resource to use while they're gone."—*B4-Dev*

6 VALIDATION SURVEY

We sent a validation survey to all 27 participants, nine of whom responded. The survey was organized in three sections that each consisted of a Likert-scale evaluation followed by the free-form

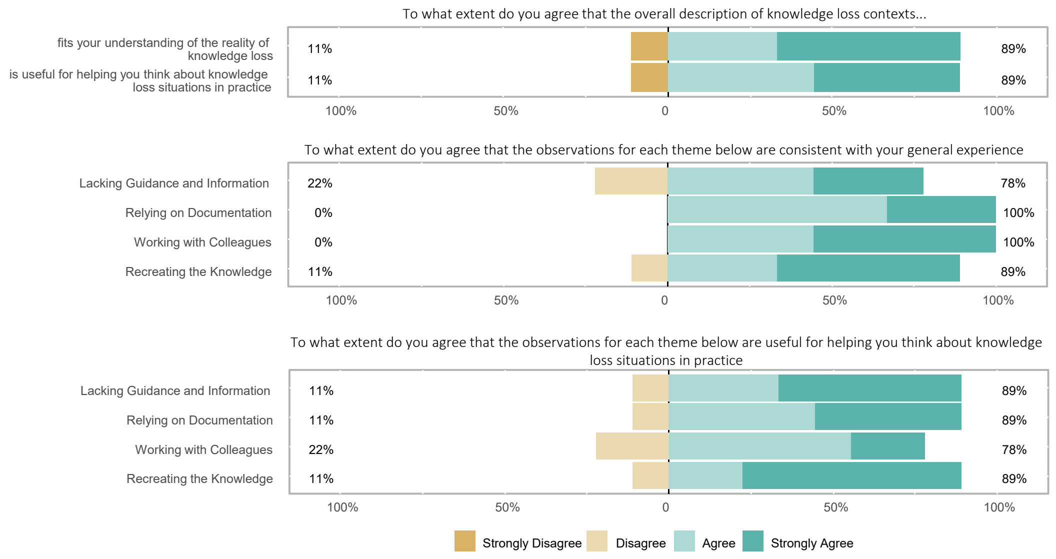

Figure 3: Validation survey responses for Likert-scale questions (n=9)

question "please justify your answer". Figure 3 shows the questions and answers to the Likert-scale portion of all three sections. In this section, we identify respondents by company only to maintain adequate anonymity.

Knowledge Loss Contexts. The first section addressed both the *credibility* and *usefulness* of the knowledge loss context framework described in Section 4. Eight respondents agreed that the description of the knowledge loss contexts fits their experience and was useful. The remaining respondent strongly disagreed to both statements, without comments. In addition to confirmatory comments, two respondents pointed to additional aspects of the knowledge loss context that would be useful to capture: "the quantity, quality, and uniqueness of the knowledge"—*C* and well as the "the size of the company and the development team"—*C*.

Knowledge Loss Implications–Credibility. The second section addressed specifically the *credibility* of the observations reported for the four themes. The observations for the themes *relying on documentation* and *working with colleagues* resonated with all respondents. For the theme *lacking guidance and information*, only one of the two detractors commented, stating: "I find the context of a developer or a development team having to figure out the *intent* of a piece of code troubling..."—*B*. This divergence may be due to the fact that the data supporting the observation about *guessing the intent of the code* was exclusively collected from participants in Company A, and the respondent may not have been exposed to these challenges as part of a smaller organization. About *recreating the knowledge*, the detractor indicated: "Can be done but it is too stressful and asks to invest a huge amount of time and the efforts are rarely appreciated"—*A*. This comment indicates that while the respondent disagrees with *the practice*, they actually speak of it from experience, thus confirming the observation. Other complementary insights shared by participants as comments to this section include: "Lacking guidance and information is the number one issue for us. To a point where we pay specific attention to this criterion in our code reviews. We ask for inline comments when we spot code that has no obvious intent"—*C* as well as "If knowledge is not externalized, it is often because we are unsure what the best format and location for that knowledge is"—*C* and "For lacking guidance and information, small companies may be more impacted because of the many responsabilities each of the employees have"—*C*.

Knowledge Loss Implications–Usefulness. The third section of the survey addressed the *usefulness* of the observations. For this section, seven respondents agreed or strongly agreed that the observations were useful for all four themes, and two respondents disagreed with two themes each. One detractor provided no justification, but for the second the justification does not appear to contradict the observations: "I do not agree exactly with everything about lacking guidance and information. I feel that developers need to be able to learn on their own..."—*C*. Although the observations capture the experience of developers in situations of knowledge loss, the statements make no judgement about mitigation measures, which may certainly include self-learning. As for feedback related to *working with colleagues*, the statement actually corroborates the experience of *A9-Dev* noted in Section 4.1: "When working with previous colleagues, I disagree in the sense that I would not feel comfortable contacting someone that is not working with the company anymore to retrieve information..."—*C*. Finally, one comment from a positive respondent highlights the role and potential value of knowledge re-creation in the spiral of knowledge: "I agree with all the different points mentioned but recreating the knowledge can be highly beneficial for the person at stake. In my personal opinion I learn best when I get hands-on experience with something. I find that relying on documentation and colleagues gives you a decent amount of understanding but to truly understand something it is best for me to recreate it. I agree that it is pretty time consuming"—*C*. It is interesting to note how this opinion is diametrically opposed to that of the respondent from Company A quoted in the previous paragraph.

7 CONCLUSIONS

With this study, we sought to better understand turnover-induced knowledge loss contexts and their implications. Based on interviews with 27 professional software developers and managers from three companies, we devised and applied a framework for characterizing knowledge loss contexts, and synthesized 20 observations about the potential implications of knowledge loss in practice. We conclude with a discussion of the broader insights we gathered from the study.

Insights From Knowledge Loss Contexts. The experiences we collected about knowledge loss contexts challenge a number of assumptions about turnover-induced knowledge loss in software development. The notion of turnover captured by the "bus factor" concept [2, 7, 43] assumes a sudden, complete, and permanent departure. Our study helped illustrate the different shades of turnover beyond those parameters. First, we saw how departed knowledge owners may remain available for knowledge transfer, thus mitigating the impact of knowledge loss. Conversely, we saw how internal transfers within a company can nevertheless amount to a complete departure from the point of view of knowledge loss, as the knowledge owners progressively lose their knowledge. The interview data also emphasized how the simple anticipation of a departure can play a major role in preventing knowledge loss. Finally, we observed that temporary leaves—not usually considered turnover—can have implications similar to knowledge loss as developers must resolve incidents under time pressure without the help of an expert.

Knowledge Loss and Knowledge Management. The 20 observations about the implications of knowledge loss listed in Table 3 capture important concerns of the participants regarding knowledge loss. We note that they cover practically all the forms of knowledge conversion illustrated in Figure 1. Observations about *lacking guidance and information* capture the challenges of sharing tacit knowledge, typically acquired through socialization. The theme *working with colleagues* focuses specifically on the socialization conversion, and *recreating the knowledge* also emphasizes the need to involve others in this process. Externalization, and to a lesser extent, combination, are reflected in the theme *relying on documentation*. The observations captured under this theme illustrate the barriers to knowledge transfer experienced when documentation is used for knowledge loss mitigation (as opposed to, for example, learning a new technology [28, 44]). The theme *recreating the knowledge* also touches on externalization, as participants have described their attempts to capture the new knowledge explicitly. Finally, the observations about tacit knowledge under theme *lacking guidance and information* show how some participants were adamant about the importance of knowledge internalization (learning by doing) in the knowledge loss mitigation process.

Future Work. This study was exploratory in nature. It sought to report on the experience of knowledge loss in software development using data excerpts that provide descriptions of the phenomenon as close to the participants' experiences as possible. While these descriptions help increase our awareness of the numerous facets of the phenomenon of knowledge loss in practice, much work remains to implement knowledge loss mitigation strategies in specific contexts. First, it will be useful to empirically assess the potential impact of turnover events on organizations. Our proposed framework hypothesizes this impact, but future work could empirically validate the levels, as well as help enrich the framework with additional dimensions for capturing knowledge loss. For example, it may be possible to identify the main type of knowledge lost through a specific departure event. Another promising direction for furthering this work will be to link the different context types with implications. For example, are internal transfers more strongly associated with certain types of implications than retirements? Finally, as the ultimate goal of this research is to help organizations manage knowledge related to software assets, a natural extension of this study will be to experiment with a selection of mitigation strategies for one or more knowledge loss implications determined with the help of the observations we contributed. To this end, the framework presented in Table 2 and the observations described in Table 3 can provide a shared vocabulary and a common ground for software development teams to engage in discussions of knowledge loss challenges and potential mitigation strategies.

ACKNOWLEDGMENTS

We warmly thank the company facilitators and participants for generously sharing their time and making this project possible. We are also grateful to Alexa Hernandez for reviewing the paper in detail, and to Deeksha Arya, Mathieu Nassif, Louis-Bertrand Varin, and the anonymous reviewers for their constructive comments. This work is funded by NSERC.

REFERENCES

[1] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software documentation issues unveiled. In *Proceedings of the 41st IEEE/ACM International Conference on Software Engineering*. 1199–1210. https://doi.org/10.1109/ICSE.2019.00122

[2] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2016. A novel approach for estimating truck factors. In *Proceedings of the 24th IEEE International Conference on Program Comprehension*. 1–10. https://doi.org/10.1109/ICPC.2016.7503718

[3] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't touch my code! Examining the effects of ownership on software quality. In *Proceedings of the Joint 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13th European Software Engineering Conference*. 4–14. https://doi.org/10.1145/2025113.2025119

[4] Virginia Braun and Victoria Clarke. 2012. Thematic analysis. In *APA Handbook of Research Methods in Psychology, Vol 2: Research Designs: Quantitative, Qualitative, Neuropsychological, and Biological*, Harris Cooper, Paul M. Camic, Debra L. Long, A.T. Panter, David Rindskopf, and Kenneth J. Sher (Eds.). American Psychological Association, 57–71. https://doi.org/10.1037/13620-004

[5] Virginia Braun and Victoria Clarke. 2021. To saturate or not to saturate? Questioning data saturation as a useful concept for thematic analysis and sample-size rationales. *Qualitative Research in Sport, Exercise and Health* 13, 2 (2021), 201–216. https://doi.org/10.1080/2159676X.2019.1704846

[6] Donald O. Case and Lisa M. Given. 2016. *Looking for Information: A Survey of Research on Information Seeking, Needs, and Behavior* (4th ed.). Emerald.

[7] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2015. Assessing the bus factor of Git repositories. In *Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering*. 499–503. https://doi.org/10.1109/SANER.2015.7081864

[8] John W. Creswell. 2003. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (2nd ed.). Sage.

[9] John W. Creswell. 2006. *Qualitative Inquiry and Research Design* (2nd ed.). Sage.

[10] Barthélémy Dagenais and Martin P. Robillard. 2010. Creating and evolving developer documentation: Understanding the decisions of open source contributors. In *Proceedings of the 18th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*. 127–136. https://doi.org/10.1145/1882291.1882312

[11] Kimiz Dalkir. 2017. *Knowledge Management in Theory and Practice*. MIT Press.

[12] David W. DeLong. 2004. *Lost Knowledge: Confronting the Threat of an Aging Workforce*. Oxford University Press.

[13] Susanne Durst and Stefan Wilhelm. 2011. Knowledge management in practice: Insights into a medium-sized enterprise's exposure to knowledge loss. *Prometheus* 29, 1 (2011), 23–38. https://doi.org/10.1080/08109028.2011.565693

[14] Martin Feilkas, Daniel Ratiu, and Elmar Jürgens. 2009. The loss of architectural knowledge during system evolution: An industrial case study. In *Proceedings of the 17th IEEE International Conference on Program Comprehension*. 188–197. https://doi.org/10.1109/ICPC.2009.5090042

[15] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rémy Falleri. 2015. Impact of developer turnover on quality in open-source software. In *Proceedings of the Joint ACM SIGSOFT Symposium on the Foundations of Software Engineering and European Software Engineering Conference*. 829–841. https://doi.org/10.1145/2786805.2786870

[16] Robert Gregory, Roman Beck, and Michael Prifling. 2009. Breaching the knowledge transfer blockade in IT offshore outsourcing projects - A case from the financial services industry. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*. 1–10. https://doi.org/10.1109/HICSS.2009.93

[17] Daniel S. Hamermesh, Wolter H. J. Hassink, and Jan C. van Ours. 1996. Job turnover and labor turnover: A taxonomy of employment dynamics. *Annales d'Économie et de Statistique* 41/42 (1996), 21–40. https://doi.org/10.2307/20066462

[18] John Horner and M. E. Atwood. 2006. Effective design rationale: Understanding the barriers. In *Rationale Management in Software Engineering*, Allen H. Dutoit, Raymond McCall, Ivan Mistrík, and Barbara Paech (Eds.). Springer, 73–90. https://doi.org/10.1007/978-3-540-30998-7_3

[19] Daniel Izquierdo-Cortazar, Gregorio Robles, Felipe Ortega, and Jesus M. Gonzalez-Barahona. 2009. Using software archaeology to measure knowledge loss in software projects due to developer turnover. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*. 1–10. https://doi.org/10.1109/HICSS.2009.498

[20] Murray E. Jennex and Alexandra Durcikova. 2013. Assessing knowledge loss risk. In *Proceedings of the 46th Hawaii International Conference on System Sciences*. 3478–3487. https://doi.org/10.1109/HICSS.2013.103

[21] Andrew J. Ko, Robert DeLine, and Gina Venolia. 2007. Information needs in collocated software development teams. In *Proceedings of the 29th IEEE/ACM International Conference on Software Engineering*. 344–353. https://doi.org/10.1109/ICSE.2007.45

[22] Marianne Kukko and Nina Helander. 2012. Knowledge sharing barriers in growing software companies. In *Proceedings of the 45th Hawaii International Conference on System Sciences*. 3756–3765. https://doi.org/10.1109/HICSS.2012.407

[23] Dorothy Leonard-Barton. 2015. *Critical Knowledge Transfer: Tools for Managing Your Company's Deep Smarts*. Harvard Business Review Press.

[24] Timothy C. Lethbridge, Janice Singer, and Andrew Forward. 2003. How software engineers use documentation: The state of the practice. *IEEE Software* 20, 6 (2003), 35–39. https://doi.org/10.1109/MS.2003.1241364

[25] Bin Lin, Gregorio Robles, and Alexander Serebrenik. 2017. Developer turnover in global, industrial open source projects: Insights from applying survival analysis. In *Proceedings of the 12th IEEE International Conference on Global Software Engineering*. 66–75. https://doi.org/10.1109/ICGSE.2017.11

[26] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology* 23, 4, Article 31 (2014), 1-37 pages. https://doi.org/10.1145/2622669

[27] Markus Manhart and Stefan Thalmann. 2015. Protecting organizational knowledge: A structured literature review. *Journal of Knowledge Management* 19, 2 (2015), 190–211. https://doi.org/10.1108/JKM-05-2014-0198

[28] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. 2018. Application programming interface documentation: What do software developers want? *Journal of Technical Writing and Communication* 48, 3 (2018), 295–330. https://doi.org/10.1177/0047281617721853

[29] Matthew B. Miles, A. Michael Huberman, and Johnny Saldaña. 2014. *Qualitative Data Analysis: A Method Sourcebook* (3rd ed.). Sage.

[30] Ehsan Mirsaeedi and Peter C. Rigby. 2020. Mitigating turnover with code review recommendation: Balancing expertise, workload, and knowledge distribution. In *Proceedings of the 42nd ACM/IEEE International Conference on Software Engineering*. 1183–1195. https://doi.org/10.1145/3377811.3380335

[31] Susan M. Mitchell and Carolyn B. Seaman. 2016. Could removal of project-level knowledge flow obstacles contribute to software process improvement? A study of software engineer perceptions. *Information and Software Technology* 72 (2016), 151–170. https://doi.org/10.1016/j.infsof.2015.12.007

[32] Audris Mockus. 2010. Organizational volatility and its effects on software defects. In *Proceedings of the 18th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*. 117–126. https://doi.org/10.1145/1882291.1882311

[33] Mathieu Nassif and Martin P. Robillard. 2017. Revisiting turnover-induced knowledge loss in software projects. In *Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution*. 261–272.

[34] Olivia B. Newton, Stephen M. Fiore, and Jihye Song. 2019. Expertise and complexity as mediators of knowledge loss in open source software development. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 63, 1 (2019), 1580–1584. https://doi.org/10.1177/1071181319631445

[35] Srinivas Nidhra, Muralidhar Yanamadala, Wasif Afzal, and Richard Torkar. 2013. Knowledge transfer challenges and mitigation strategies in global software development - A systematic literature review and industrial validation. *International Journal of Information Management* 33, 2 (2013), 333–355. https://doi.org/10.1016/j.ijinfomgt.2012.11.004

[36] Ikujiro Nonaka and Hirotaka Takeuchi. 1995. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press.

[37] Klaus North and Gita Kumta. 2014. *Knowledge Management: Value Creation Through Organizational Learning*. Springer. https://doi.org/10.1007/978-3-319-03698-4

[38] Salvatore Parise, Rob Cross, and Thomas H. Davenport. 2006. Strategies for Preventing a Knowledge-Loss Crisis. *MIT Sloan Management Review* 47, 4 (2006), 31–38.

[39] L. G. Pee, A. Kankanhalli, G. W. Tan, and G. Z. Tham. 2014. Mitigating the impact of member turnover in information systems development projects. *IEEE Transactions on Engineering Management* 61, 4 (2014), 702–716. https://doi.org/10.1109/TEM.2014.2332339

[40] Michael Polanyi. 1967. *The Tacit Dimension*. Anchor Books.

[41] Xiangju Qin, Michael Salter-Townshend, and Padraig Cunningham. 2014. Exploring the relationship between membership turnover and productivity in online communities. In *Proceedings of the 8th International AAAI Conference on Weblogs and Social Media*. https://www.aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/view/8097/8141

[42] Mehvish Rashid, Paul M. Clarke, and Rory V. O'Connor. 2019. A systematic examination of knowledge loss in open source software projects. *International Journal of Information Management* 46 (2019), 104–123. https://doi.org/10.1016/j.ijinfomgt.2018.11.015

[43] Peter C. Rigby, Yue Cai Zhu, Samuel M. Donadelli, and Audris Mockus. 2016. Quantifying and mitigating turnover-induced knowledge loss: Case studies of Chrome and a project at Avaya. In *Proceedings of the 38th ACM/IEEE International Conference on Software Engineering*. 1006–1016. https://doi.org/10.1145/2884781.2884851

[44] Martin P. Robillard and Kaylee Kutschera. 2020. Lessons learned while migrating from Swing to JavaFX. *IEEE Software* 37, 3 (2020), 78–85. https://doi.org/10.1109/MS.2019.2919840

[45] Martin P. Robillard, Mathieu Nassif, and Shane McIntosh. 2018. Threats of aggregating software repository data. In *Proceedings of the 34th IEEE International Conference on Software Maintenance and Evolution*. 508–518. https://doi.org/10.1109/ICSME.2018.00009

[46] Kurt Schneider. 2009. *Experience and Knowledge Management in Software Engineering*. Springer. https://doi.org/10.1007/978-3-540-95880-2

[47] Sulayman K. Sowe, Ioannis Stamelos, and Lefteris Angelis. 2008. Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems and Software* 81, 3 (2008), 431–446. https://doi.org/10.1016/j.jss.2007.03.086

[48] Meaghan Stovel and Nick Bontis. 2002. Voluntary turnover: knowledge management–friend or foe? *Journal of Intellectual Capital* 3, 3 (2002), 303–322. https://doi.org/10.1108/14691930210435633

[49] Nakkiran N. Sunassee and David A. Sewry. 2002. A theoretical framework for knowledge management implementation. In *Proceedings of the 2002 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology*. 235–245.

[50] Charles A. Tryon. 2012. *Managing Organizational Knowledge: 3rd Generation Knowledge Management ... and Beyond!* CRC Press.

[51] Gias Uddin and Martin P. Robillard. 2015. How API Documentation Fails. *IEEE Software* 32, 4 (2015), 68–75. https://doi.org/10.1109/MS.2014.80

[52] Davi Viana, Tayana Conte, Sabrina Marczak, Raymundo Ferreira, and Cleidson de Souza. 2015. Knowledge creation and loss within a software organization: An exploratory case study. In *Proceedings of the 48th Hawaii International Conference on System Sciences*. 3980–3989. https://doi.org/10.1109/HICSS.2015.477

[53] Robert S. Weiss. 1995. *Learning From Strangers: The Art and Method of Qualitative Interview Studies*. Free Press.