UNIVERSITY OF CALIFORNIA,
IRVINE

Understanding How Information Flows in and out of Regularly Scheduled Software Maintenance Design Meetings: a Case Study

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Software Engineering

by

Adriana Meza Soria

Dissertation Committee:
Professor André van der Hoek, Chair
Professor David Redmiles
Assistant Professor Iftekhar Ahmed

2022

DEDICATION

To

I am dedicating this thesis to those who supported me the most during these five years of study. First, I dedicate this thesis to my mother, Bertha Alicia Soria Arteche, who has been a role model for me since my childhood; Her good examples have taught me to work hard for the things I aspire to achieve. I also dedicate this work to my father, Eduardo de Jesus Meza Lara. Although he is no longer in this world, my memories with him will continue to give me little moments of happiness throughout my life. I also want to dedicate this work to Jesus Alberto Alvarado Tovar, a constant source of support and encouragement during the challenges of graduate school. I am genuinely thankful for having him in my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my committee chair, Professor André van der Hoek, who has been a supportive mentor in good and bad times. Without his guidance and persistent help, this dissertation would not have been possible.

I would like to thank my committee members, Professor David Redmiles, and Professor Iftekhar Ahmed, whose knowledge of comparative literature and modern technology helped me to shape this thesis.

In addition, I thank Professor Marian Petre from The Open University, who provided much of her wisdom in the early stages of this project, and Professor Janet Burge from Colorado College, who has been a close collaborator in this project. Finally, I want to thank outstanding students who worked with me as undergraduate researchers during my Ph.D.: Taylor Lopez, Negin Mashhadi, Joshua Joseph Costa, Liz Seero, Emily Evans, Juwin Viray, Yuqi Huai, Seunghwan Hong, Nayoung Lee, and SangHoon Kim.

VITA

Adriana Meza Soria

EDUCATION

2018-2022	Ph.D. in Software Engineering University of California, Irvine Research Area: Software Design and Collaboration
2014-2016	M.S in Engineering (Summa Cum Laude) CETYS University, Tijuana, Mexico
2008-2013	B.S. in Computational Systems Engineering Instituto Tecnologico de Tijuana, Tijuana, Mexico

PROFESSIONAL EXPERIENCE

Summer 2021	Research Intern, MIT-IBM Watson AI Lab
2013-2017	Senior Software Engineer, Grupo Tress International
2012–2013	IT Assistant, IWAI Metal Mexico

TEACHING EXPERIENCE

Summer 2020	Professor, University of California, Irvine, U.S.A
2018-2022	Teaching Assistant, University of California, Irvine, U.S.A
2016-2017	Professor, CETYS University, Mexico
2014-2017	Professor, Autonomous University of Baja California (UABC), Mexico

HONORS AND AWARDS

2022	Recipient of Miguel Velez Scholarship (3rd)
2021	Latino Excellence and Achievement Award
2019	Recipient of Miguel Velez Scholarship (2nd)
2019	Recipient of Rosalva Gallardo Valencia Graduate Award
2018	Second place at AMIA Design Challenge
2017	Recipient of Miguel Velez Scholarship (1st)

REFERRED PUBLICATIONS

A. Meza Soria, A. van der Hoek, J. Burge, "Recurring distributed software maintenance meetings: toward an initial understanding," 2022 IEEE/ACM 15th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), 2022, pp. 21-25.

A. Meza Soria and A. van der Hoek, "The Design of an Experiment Concerning the Capture of Important Design Bits at the Whiteboard," 2021 ACM/IEEE 5th International Workshop on Human Factors in Modeling (HUFAMO), 2021.

A. Meza Soria and A. van der Hoek, "Collecting Design Knowledge through Voice Notes," 2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), 2019, pp. 33-36.

A. Meza Soria and A. van der Hoek, "Toward Collecting and Delivering Knowledge for Software Design at the Whiteboard," 2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), 2018, pp. 108-109.

OTHER PUBLICATIONS

B. Ryan, A. Meza Soria, K. Dreef, A. van der Hoek, " Reading to Write Code: An Experience Report of a Reverse Engineering and Modeling Course," 2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), 2022, pp. 223-234.

VOLUNTEER WORK

2018-2022	Member, Mexico Graduate Research Education Program
2019	Mentor, I-SURF summer program, UC Irvine
2019	Mentor, ExploreCSR workshop, CSULB and UC Irvine

ABSTRACT OF THE DISSERTATION

Understanding How Information Flows in and out of

Regularly Scheduled Software Maintenance Design Meetings:

a Case Study

by

Adriana Meza Soria

Doctor of Philosophy in Software Engineering

University of California, Irvine, 2022

Professor André van der Hoek, Chair

Meetings have always been a significant part of all types of work. Software development is no exception, with meetings of all kinds taking place daily. One type of meeting that is critical to software development is the Regularly Scheduled Software Maintenance Design Meeting (RSSMDM): a recurring meeting during which the primary product leads of a software development team consider emerging issues and new directions for an already deployed and functioning software system. To date, RSSMDMs have not been widely studied and particularly ignored is the perspective of the role of information in shaping the discussions in these meetings.

This dissertation contributes a foundational understanding of how information flows in and out of RSSMDMs through a single case study of ten such meetings at a healthcare software company. Through a thematic analysis, it particularly characterizes the variety of information that the participants in these meetings on the one hand share and on the other hand capture while they engage in their design work. In addition, the dissertation identifies the tools that the meeting participants use to share and capture information and the various ways in which they use the tools.

The results from the thematic analysis are varied, with several of the more important findings being: (1) many different types of information are shared, with the range much greater than what traditionally has been considered as important to capture for future use, (2) much of the information shared is fleeting, concerning the current state of the deployed software and the current state of its code base, (3) sharing is frequent, with new information shared on average once or twice a minute, (4) the diversity and frequency of information captured is much less than information shared, and (5) traditional design tools such as diagramming and sketching tools are not used in support of the meetings, displaced by the use of Confluence (a wiki-style knowledge repository) and Jira (an issue tracker). These and other findings establish a baseline for future research into RSSMDMs, provide insights into current practices, and offer suggestions for the development of improved tools to support participants with information sharing and information capture in RSSMDMs.

1 Introduction

Design not only takes place when a system is first envisioned. It is equally important thereafter as the system is enhanced and refined [1]. Such maintenance design [2], [3] manifests itself in all sorts of forms. At one end of the spectrum, it concerns small-scale, detailed design, for instance in an agile stand-up meeting when a team member requests help in deciding how to approach a given task [4] or via chat or Zoom when a developer in the midst of programming something hits a roadblock and consults a colleague to discuss approaches for overcoming the obstacle [5].

At the other end of the spectrum, it involves high-level design, for instance when the system needs to be significantly re-architected to eliminate costly accumulating technical debt [6] or when a major new enhancement needs to be planned out [7]. Such design is typically relegated to designated meetings.

In between, another form of maintenance design [2] plays out daily across software organizations everywhere: the type of maintenance design that concerns ongoing direction setting, shaping, and managing of a system on a day-to-day basis. This kind of maintenance design is addressed across regularly scheduled (once or twice per week) meetings that members of a development team attend, to, for instance, build an understanding of an urgent issue that has been reported early that day or review the design of a new feature a client has requested. The discussion of topics of this nature does not necessarily require a standalone meeting to be planned, nor are they lightweight enough to be dealt with during daily scrum meetings. However, a regular meeting to discuss as many maintenance design issues as possible during the allocated time is a suitable venue.

In this dissertation, I name this kind of meeting a Regularly Scheduled Software Maintenance Design Meeting (RSSMDM). Particularly for large software systems, where dealing with maintenance problems is part of the daily routine, these meetings are essential to discuss high-level aspects of day-to-day maintenance design problems.

The kind of people that participate in these meetings varies. On the one hand, there are core participants that coordinate and attend RSSMDMs on a regular basis. These core participants are all key internal stakeholders of the software products of the company because they are responsible for their long-term evolution and scalability. On the other hand, other non-core participants also join RSSMDMs, but on a more incidental basis, when they have something to discuss with the key internal stakeholders, or when their presence is critical to discuss a topic. Both core and non-core participants may have a highly technical (e.g., software architect, quality control engineer, developer) or less technical (e.g., manager, product owner) background. However, core participants are typically more experienced than non-core participants in terms of their knowledge of the codebase, the customer's background, procedures to interact with other teams, and more.

Information is particularly crucial to how the discussions of day-to-day design problems in RSSMDMs proceed. On the one hand, just like programmers need to have the right information available for them to make code changes [8] and software architects sometimes need to know the rationale associated with past decisions [9] to make new decisions, key internal stakeholders in RSSMDMs need information that exists prior to the meeting to support their design deliberations. Without it, the decisions being made may be flawed, the resulting code might be of inferior quality, time-sensitive issues may need to be postponed to a later discussion, and the design conversations themselves may be ineffective due to different understandings among the meeting participants.

On the other hand, key internal stakeholders need to capture the discussion outflow of RSSMDMs. It is well known that capturing the outflow of meetings is essential to provide continuation to the work being performed [10]. Previous studies report that developers may capture decisions as a set of informal diagrams during early-design meetings (e.g., [11]–[13]) or that they may keep more formal representations of decisions, alternatives, and sometimes some rationale in wiki repositories (e.g., [14]–[17]). However, little is known about what information is captured in RSSMDMs. Given that these meetings center on maintenance, it could be that in addition to decisions [18], [19], alternatives [20], and rationale [21], other information is captured as well (e.g., procedures on how to execute a database migration, good development practices to avoid architectural drift, internal team processes).

The goal of my dissertation is to build a foundational understanding of how information flows in and out of RSSMDMs. It seeks to characterize the variety of information that key internal stakeholders share during these meetings, the new information that they generate as part of discussing design work, and the tools that, as of now, they use to obtain prior information or capture discussion outflow during RSSMDMs.

Studies about developers' information needs (e.g., [22], [23]), the role of knowledge in software development [24], and software development meetings (e.g., [25], [26]) exist. However, lacking today is a study about prior information needed in RSSMDMs, as it is needed to, for instance, discuss the implications of a support case, propose a solution for a defect in the context of a deployed and functioning system, or to re-architect a testing pipeline. Moreover, what information is captured after discussing maintenance design for these kinds of situations has also not been studied.

A holistic approach to understanding these phenomena represents a gap in the software engineering literature to which this dissertation contributes directly. It is important to fill this gap because RSSMDMs serve a critical role in the long-term maintenance of many software products. By developing an understanding of how information flows in and out of these meetings, we may discover improvements to meeting practices and potentially design novel tools to better share and capture information.

As a first step toward understanding how information flows in RSSMDMs, my dissertation contributes an in-depth study of ten such meetings held by a group of team leads at a major healthcare software development company in early 2020. This group is responsible for a key healthcare software product that is used in hundreds of hospitals worldwide. The group consists of a product owner, two software architects, and a quality assurance engineer, all of whom are located in the U.S. and all of whom attend nearly every meeting. Other participants join from across the U.S. and India on a more incidental basis, and include a shadow product owner in India, a manager, and developers at a range of levels. I requested and was given copies of the WebEx recordings of ten regularly scheduled meetings[1] that took place between March 24th and July 16th, 2020, which were transcribed by a professional service. These videos and transcripts then, constitute the basis for my study.

[1] These meetings were provided to us in an opportunistic manner. We did not have an inclusion or exclusion criterion other than the meetings being scheduled regularly and concerning maintenance design. We settled on ten meetings, because it balances depth (in it being feasible to manually, line-by-line analyze ten hours of meetings) with breadth (in having a month of maintenance design issues available to examine and make sense of).

One could take many perspectives to perform a qualitative study of this nature. For instance, I could study what typical design elements the participants use (e.g., decisions, alternatives, constraints), how specific roles contribute to the discussions (e.g., software architects, product owners, developers), and more. In this dissertation, I particularly choose to look at information because it is foundational in supporting design deliberation. Below, I present the overarching research questions that guided this qualitative study. Each question targets a critical aspect of how information is shared and captured in RSSMDMs.

RQ 1. What prior information do participants share in RSSMDMs in the course of design deliberation?

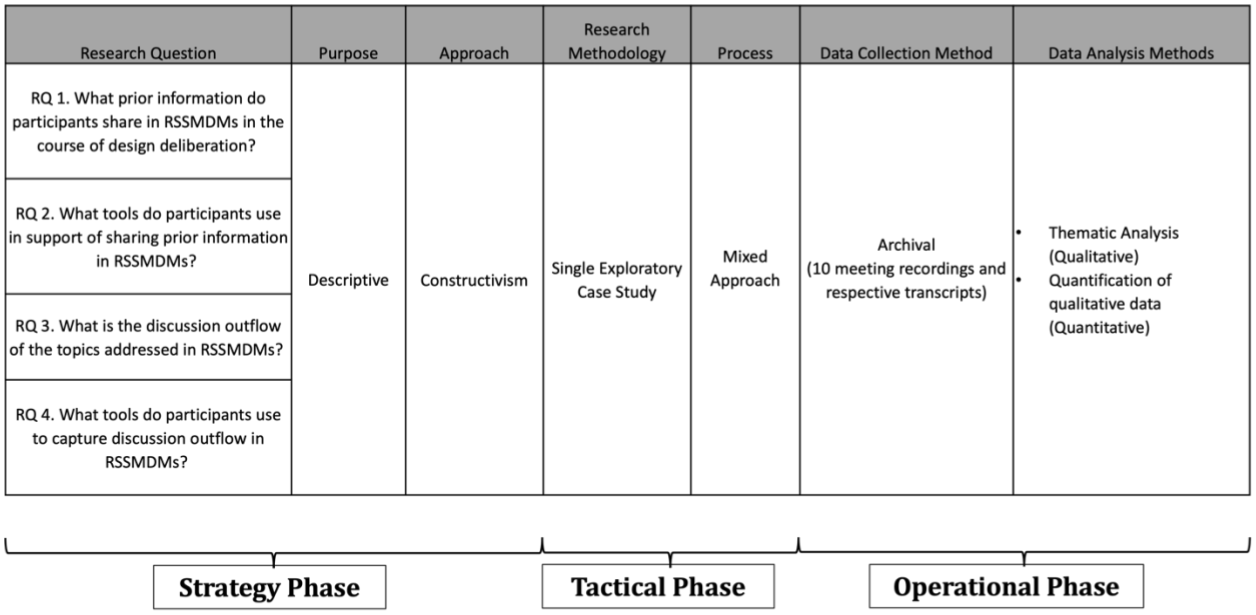
RQ 2. What tools do participants use in support of sharing prior information in RSSMDMs?

RQ 3. What is the discussion outflow of the topics addressed in RSSMDMs?

RQ 4. What tools do participants use to capture discussion outflow in RSSMDMs?

My approach to studying the ten RSSMDMs is grounded in a constructivist philosophical point of view that, rather than verifying previously established theories, centers on exploring and understanding a particular phenomenon in its natural setting [27]. Specifically, this work presents a single exploratory case study [28] in a major healthcare software company. The focus of the case study is a set of RSSMDMs in which a high-performing team (i.e., product owners, software architects, and lead developers) discuss day-to-day maintenance design issues and new directions in the context of a deployed and functioning system. Figure 1 shows the design of the study that I just described.

Figure 1. The design of the study.



Research Question	Purpose	Approach	Research Methodology	Process	Data Collection Method	Data Analysis Methods
RQ 1. What prior information do participants share in RSSMDMs in the course of design deliberation?	Descriptive	Constructivism	Single Exploratory Case Study	Mixed Approach	Archival (10 meeting recordings and respective transcripts)	• Thematic Analysis (Qualitative) • Quantification of qualitative data (Quantitative)
RQ 2. What tools do participants use in support of sharing prior information in RSSMDMs?						
RQ 3. What is the discussion outflow of the topics addressed in RSSMDMs?						
RQ 4. What tools do participants use to capture discussion outflow in RSSMDMs?						

Each meeting was first partitioned into the design topics that were deliberated (minimum two, maximum eleven). Over the course of the ten meetings, the key internal stakeholders attending discussed 45 different topics, which were significantly diverse in type. Example topics included a performance problem with a deployed instance of the software at one of the hospitals, a design review of a major proposed change by one of the developers, a discussion of the impact of a customer choosing to implement their own single sign-on solution, and an analysis of the testing environment and how it incurs unnecessary cloud expenses.

Four researchers, one of them myself, paired up in different configurations to analyze different aspects of the meetings' video recordings and their respective transcriptions by applying thematic analysis [29]. Thematic analysis is a method to qualitatively classify data into categories according to a coding scheme that can be developed in different ways (e.g., inductively, deductively, hybrid) [30]. Two inductive thematic analyses were performed in total. One centered on the prior information that the participants shared during the meetings and which tools (if any) they used to share it. The other one centered on the discussion outflow of the 45 topics that the participants discussed and which tools (if any) the participants used to capture it.

Together, the answers to the four research questions offer a comprehensive view of how information flows in and out of RSSMDMs, where there might be opportunities for improvement in terms of practices and tools to share prior information, and where there might be opportunities for improvement in practices and tools to capture the discussion outflow while at the meeting.

My dissertation provides the following contributions:

C1. A rich set of observations about information sharing practices and tool use in RSSMDMs. Understanding what kinds of prior information participants share to discuss design (e.g., architecture, deployment, testing), who shares it (e.g., product owners, software architects, developers), how it is shared (e.g., by request, voluntarily), and what tools are used to do so has important implications for future tool development and meeting practices in RSSMDMs.

C2. A rich set of observations about information capture practices and tool use in RSSMDMs. Understanding what kinds of outflow participants capture, whether they capture outflow throughout the discussion or at specific moments, identifying when discussion outflow is not captured in tools, who captures it (e.g., product owners, software architects, developers), how participants capture it (e.g., prompted, unprompted, requested by the tool driver), and what tools they use to do so has important implications for future tool development and meeting practices in RSSMDMs.

The remainder of this dissertation is organized as follows: Chapter 2 covers important related work. Chapter 3 describes the ten RSSMDMs studied. In Chapters 4 and 5, I present the results of analyzing how prior information is shared and how information is captured in RSSMDMs. In Chapter 6, I discuss threats to validity. Finally, Chapter 7 presents the conclusions I draw from this work and my suggestions for future work.

2 Background

My work relates to various research areas: software maintenance design, collaborative work in software development, software design studies, software development meetings, information studies, knowledge management, and design rationale. In this chapter, I discuss a representative sample of studies for each of these research areas.

2.1 Software maintenance design

Foundational work defines software maintenance as the process of modifying software systems after delivery to correct faults, improve performance, or adapt them to an environment that has changed [31]. Software maintenance was initially categorized into three types: adaptative (software maintenance to adapt the system to changes in its data environment), corrective (software maintenance to correct processing, performance, or implementation failures of the system), and perfective (software maintenance to perfect the system in terms of its performance, processing efficiency, or maintainability) [32]. This initial classification gave rise to other typologies and standards (e.g., [33]–[37]).

Articulating the key dimensions of software maintenance served as a baseline to further research in this area. As one example, some work centered on empirically studying how developers perform software maintenance activities (e.g., refactoring [38], program comprehension [39], debugging [40]). As another example, many empirical studies centered on how developers make code changes (e.g., identifying characteristics in vulnerable code changes [41], exploring how developers understand code changes [42], measuring the future impact of code changes [43]). Other work studied developers' information needs in their day-to-day programming. This topic in particular was studied not only empirically [44], but also through laboratory experiments [45].

The foundational work and empirical evidence led to the development of novel tools to improve software maintenance in practice. For instance, Dias et al. presented a novel approach to detect and untangle source code changes across different program versions [46] and Rastkar and Murphy proposed multi-document summarization techniques to generate a description of why code changed so that developers could understand the rationale behind code changes [8]. Other researchers proposed tools that support developers in visualizing different aspects of code changes (e.g., code understanding [47], code evolution [48], [49], debugging [50]). Yet other research led to tools that refactor code automatically [38] or address information needs that developers have while programming (e.g., [51], [52]).

Software maintenance has also been studied in relation to design. At the architectural level, for instance, researchers have proposed techniques (e.g., [53], [54]) and tools (e.g., [55], [56]) to recover the architecture of large systems based on their source code. Others have studied the role of software architecture in the evolution and quality of software [57]. My work in particular relates to software maintenance design [2] because the discussions held in the RSSMDMs that I am studying are all about maintenance design work. Understanding the types of maintenance work that exist, and the kind of design each requires, is fundamental to understanding the context of the discussions that constitute the basis of my study, as these discussions range from architectural design to low-level design to solve day-to-day maintenance issues.

2.2 Collaboration in software development

Many studies about collaborative work center on observing co-located physical environments in which software is developed. For instance, Mark studied the interaction of software developers working together inside dedicated project rooms (she calls them war rooms, shared physical spaces where team members work together synchronously [58]). Mark in particular named this kind of collaboration “extreme collaboration”, but others also refer to it as “radical collaboration” [59]. Researchers who have studied collaboration inside dedicated project rooms (e.g., [59], [60]) observed that it brings advantages in terms of coordination, communication, and learning to software development teams, which eventually reflects positively on their overall productivity.

Even though co-located spaces, and particularly dedicated project rooms, have been reported as beneficial to make coordination tasks agile in software development teams [61], the distribution of development teams around the world is a step that every company must eventually take as it grows. Sometimes companies distribute their development teams around the globe because they seek to conquer new markets, which requires maintenance and technical support in the time zones of those respective markets [62]. Other times, they do so because companies seek to reduce development costs and must outsource development tasks to economically favorable locations [63].

The distribution of software development teams creates new collaboration challenges [64] that need to be understood and addressed to make possible the gains that distributed software development offers [65]. For instance, Espinosa et al., confirmed that geographic distance has a negative effect on coordination [66]. They also observed that sharing information with team members at other locations is a way to mitigate coordination issues by establishing a common ground regarding development tasks and processes [66]. Moreover, studies have shown that a substantial amount of synchronous coordination in co-located software development teams takes place through informal encounters in public areas at the workplace (e.g., the water cooler, coffee room) [67][68]. Such unplanned interactions do not happen when team members are separated by distance. Therefore, synchronous collaboration is relegated to virtual, sometimes hybrid meetings [69], which often are scheduled on a regular basis, and that have become a crucial space to share information and build a common ground among distributed team members. My work studies one kind of such meetings, the RSSMDM, a regularly scheduled meeting to address software maintenance issues and new directions in the context of a deployed and functioning system.

2.3 Software design studies

The study of professional software designers and how they engage in design work has received steady attention over the years. Foundational work was rooted in personal experiences that, for instance, reflect on the importance of separating design and project management in large software development projects [70], establish important foundations for software architecture design [71], classified common design problems in software development [72], and identified known solutions for them (design patterns) [73].

Later work centered on observing software developers engaged in design either in situ (e.g., [74]–[76]) or as part of laboratory experiments (e.g., [77], [78]) to study how they make decisions, and how they represent these decisions as well as other design constructs (e.g., alternatives, assumptions, constraints). Regarding decisions, in [79], decision-making was studied in the context of agile versus non-agile organizations; Falessi et al. studied decision-making strategies (e.g., naturalistic, rational) suitable to resolve tradeoffs in software architecture design [9]; and Zannier et al. observed that the structure of design problems determines how rational or naturalistic (in the sense of Falessi et al.) decision making may become [76].

Regarding design representations, Cherubini et al. interviewed software developers at Microsoft to investigate why they make drawings [74]. They identified two scenarios in which developers make drawings to represent design, namely refactoring and code reviews. Moreover, they observed that design decisions were often externalized in temporary drawings that designers rarely capture in tools for long-term use, and that the reason for not doing so was the time such capturing would require. As a second example, Dekel and Herbsleb conducted observational studies of development teams working on design exercises [80]. They observed that developers preferred to use notations that deviate from standard UML to represent their designs. They also observed that team members rely on other communication mediums (e.g., conversation and gesturing) to interpret the sketches made. They claimed that without this additional non-visual information the drawings would be difficult to interpret and would have limited documentation potential.

The motivation underneath these and other studies of the use of drawings in software design (e.g., [75], [81]) informed the development of novel sketching tools to create, re-use, and annotate drawings that represent the design of software systems (e.g., SILK [82], Knight [83], DENIM [84], Calico [85], FlexiSketch [86]).

2.3.1 Studying Professional Software Design Workshop

In 2010, the Studying Professional Software Design (SPSD) workshop [87] took place at the University of California, Irvine. The SPSD was a workshop similar to the Design Thinking Research Symposium series [88], though it focused exclusively on software design. The goal of the SPSD workshop was to collect a foundational set of observations and insights about software design. Before the workshop, teams of professional software designers were tasked with designing a traffic simulator. They were given a problem prompt about an educational traffic simulator, a non-electronic whiteboard, and had one hour and fifty minutes to come up with the high-level design of it. They were asked to produce two design outcomes: the simulator's interface and a basic algorithm to run the simulator. The three design sessions were video recorded, and these videos were then shared with the SPSD workshop participants before the workshop.

The workshop led to a wide range of studies based on the videos and their respective transcripts. For instance, Christiaans and Almendra studied design decisions by classifying the transcripts of the three sessions using a decision-making framework [89]. They observed that decision making was most of the time cooperative, meaning the developers worked together for the sake of integrating their ideas. As another example, Matthews studied the use of assumptions to show how creative parts of design work, such as imagining nonexistent circumstances, are facilitated by designers' use of assumptions [90]. In a similar vein, Ossher et al. studied the sessions to understand how designers develop concerns [91]. They observed that concerns were revisited repeatedly throughout the design process, that designers use multiple notations to represent them, and that these representations evolve throughout the sessions to little by little become more detailed and formal.

Other studies about the SPSD workshop's data centered on studying design processes and activities. For instance, Baker et al. analyzed the videos of the three design sessions to track individual ideas that the designers generated over the course of their respective sessions [92]. They grouped the ideas identified into high-level subjects (a segment of work related to the same topic) and divided the sessions into cycles, which they termed "periods of time that begin with a moment of focus setting by the designers and that last until the next focus setting in the session". They observed three prevalent types of cycles in regards of the way subjects were discussed, single-subject cycles (various subjects are mentioned, but there is always a central one), paired-subject cycles (two subjects are considered together with equal relevance and depth), and low-depth cycles (instances in which many subjects are addressed without engaging with any one in depth). Overall, Baker et al. suggest that designers might benefit from being able to view two parts of a design-in-progress side by side, implying that subject switching is beneficial to the process.

Tang et al. also studied design activities, but with the goal of discovering practices that characterize effective design [93]. In contrast to Baker et al., they concluded that excessive switching between topics (subjects in the sense of Baker [92]) negatively impacts the overall use of time. They explain that excessive topic switching defocuses the design discussion, and that for this reason designers require more time to address all the issues that must be addressed. These authors suggest that having an outline about what must be discussed prior to the actual discussion might improve developers' understanding of the design problem and facilitate the proposal of solutions.

My work relates to the studies based on the SPSD workshop in terms of methods and contributions. Given that these studies laid an important foundation for how to analyze videos and transcripts about software design discussions, part of my methodology to analyze the set of RSSMDMs that I have been provided with is inspired by this prior work. Moreover, my work complements the SPSD workshop studies in two ways. First, while all the SPSD workshop studies center on the early design of a new system, my work addresses maintenance design (design in the context of a deployed and functioning system). Second, none of the SPSD workshop studies analyzed the design sessions by taking information flow as the central perspective, which my work does.

2.3.2 Beyond the SPSD workshop

The studies based on the SPSD workshop, as well as relevant work that preceded it (e.g., [74], [80]), served as a baseline for other studies about collaborative design in software engineering. For instance, Mangano et al. conducted a literature review to identify behaviors that characterize informal design using (physical) whiteboards [11]. They improved the implementation of an interactive whiteboard called Calico [85] to support all the behaviors identified in their literature review. They evaluated this improved version of Calico following an experimental protocol very similar to the one used in the SPSD workshop, with the difference that designers used a digital whiteboard tool instead of a physical whiteboard. They concluded that interactive whiteboard applications such as Calico have the potential to support designers to manipulate design content effectively during design sessions.

As a second example, Baltes and Diehl performed an exploratory study at three companies (all located in different countries and working on different software products) and combined this with an online survey to investigate how developers use diagrams [94]. They confirmed some of the insights observed by Cherubini et al. [74] and by Dekel and Herbsleb [80]: most diagrams were informal and contained some UML elements (though UML was not applied by the book). They also observed that the most common purposes to create drawings and diagrams were designing, explaining, and understanding.

Future work kept building upon this thread, for instance, by proposing to link drawings and diagrams to source code artifacts [95], automatically transforming paper drawings into digital ones and vice versa [96] and motivating the development of tools with novel features to support collaborative software design meetings (e.g., FLEXISKETCH TEAM [97], [98], the Interaction Room [99], [100]). The focus of these tools is to capture design information in drawings. However, important aspects of the design are also conveyed by discussing them [80]. My work seeks to inform the design of tools that consider both aspects.

2.4 Meetings in software development

Meetings have always been a significant part of working life and software development is no exception, with meetings of all kinds taking place daily. On a typical day, developers may attend various types of meetings, all with different purposes. For instance, they may attend the daily stand-up meeting [101] to share and provide an update of the work that they have done, or they may attend a design meeting to work on the design of a new software application. They may also attend sprint planning meetings [102] to define the work to implement as part of the next sprint, or retrospective meetings [103] to reflect on the way the previous sprint was handled.

During these different kinds of software development meetings, software design is addressed at different levels. For instance, aspects of low-level maintenance design (i.e., where to find a code example to avoid a memory leak) might be briefly discussed during an agile stand-up meeting [4]. As another example, high-level aspects about the early design of a new product might be discussed during a one-off design meeting.

2.4.1 The daily stand-up meeting

The daily stand-up meeting is one of the most applied agile practices [101]. However, while much has been written about it in all sorts of venues, it has not undergone many detailed empirical studies. Stray et al. are perhaps the exception in having dived deeply into it. They built an empirical understanding of this kind of meeting by analyzing the transcriptions of eight daily stand-up meetings from two software development teams [104]. They observed that even though the literature states that information sharing should focus on answering straightforward questions (i.e., what have I done? what will be done? what obstacles are in my way?), not all the time participants followed this approach by the book. In fact, they observed that 35% of the time (across all the meetings studied) was spent on design related discussions (e.g., gathering knowledge to fix issues, discussing possible solutions). Even though the nature of the meetings was to give short updates, some design work, most of it of a "low-level" nature, occurred.

Stray et al. also performed a grounded theory study of daily stand-up meetings to identify factors that may influence developers' attitudes towards how daily stand-up meetings are conducted at their workplace [4]. They observed that information sharing and the opportunity to discuss and solve problems contribute to having a positive attitude towards the daily stand-up meeting. Stray et al. also conducted a case study to identify practices that contribute to inefficient meeting management [105]. Participants reported that developers often engage in discussing details about issues on which they are working, instead of briefly mentioning the issues and scheduling a different time to discuss them in detail. They shared that this practice extends the meeting time considerably, which many developers perceive as inefficient. Another negative practice reported was that the developers sometimes used the daily stand-up meetings to report only to the scrum master, instead of sharing information with all team members.

Stray more deeply investigated these various aspects of daily stand-up meetings as part of her dissertation [106]. Her work lays a strong empirical foundation to understand daily stand-up meetings and problems that may arise. In a similar vein, my work seeks to build a foundation to understand RSSMDMs, but from a perspective focused on information, and how it flows in and out of the meetings. None of the studies performed by Stray et al. addressed this perspective.

2.4.2 Software design meetings

In Section 2.3.1, I introduced various design studies about the SPSD workshop (e.g., [80], [89], [90], [92], [93], [107]), all based on the same meeting setting of two designers working at the whiteboard on a complex software system, a traffic simulator. The design meetings conducted at the SPSD workshop were all centered on a single project and the nature of design work was that of "early-design" (the design of a new system), with participants brainstorming ideas, alternatives, and making decisions.

Olson et al. [25] conducted a study in two companies with a meeting setting quite similar, though not equal to the one studied at the SPSD workshop. In this setting, small groups of developers were sitting around a table with minimal tools such as whiteboards, flipcharts, paper, and pencil. The meetings centered on addressing the early design of large and complex software systems (e.g., internal systems, prototype ideas for future systems) that others, rather than the meeting participants themselves, would implement in future. Olson et al. analyzed ten videos of these design meetings from two companies. In my study, I analyze the same number of meetings. However, the meetings in my study are regularly scheduled and from the same organization. These characteristics are important because recurrence is part of the aspects that my study addresses.

One of the various aspects that Olson et al. studied in these meetings was design deliberation: discussions in which the designers generate ideas, discuss their pros and cons, and select some of the ideas discussed to be included in the final design. They used issues, alternatives, and criteria to describe the structure of design deliberations. They were interested in identifying moments at which designers stated questions explicitly, listed alternative solutions, or shared the rationale of why each alternative was good or bad. The focus of my work is not studying design discussions in terms of alternatives, decisions, and more, but to look at what information is needed to discuss them. Moreover, I look at the new information that they produce, and how they capture it.

Another difference between both the Olson et al. study and the SPSD workshop studies as compared to my study is the kind of design in which the participants engaged. The Olson et al. videos and the SPSD workshop videos were all about early design discussed at designated design meetings, and for which a deployed instance of the software being designed did yet not exist. My work addresses maintenance design [2] as part of regularly scheduled meetings, in which maintenance issues and new directions in the context of a deployed and functioning system are discussed.

2.4.3 Recurring meetings in software development

Recurring meetings are part and parcel in software development, with the daily stand-up meeting [101] the prototypical example. As another example, Grapenthin et al. observed agile sprint planning meetings [102]. They noted that 26% of new sprint tasks were discovered later in the meeting, rather than at the beginning when they were supposed to be all identified to then be planned. While daily stand-up (e.g., [4], [108]) and sprint planning meetings [102] are recurring software development meetings, with a clear purpose and structure, the research studies to date do not center on recurrence.

To the best of our knowledge, only one study has focused on studying recurrent meetings [109]. In this work, the authors argue that regularly scheduled meetings are interesting to study because of their repetition, which allows relationships, norms, and roles to form and evolve. They believe that the routine nature of regularly scheduled meetings lends a casual character that distinguishes them from one-time, topic-focused meetings. Even though this study does consider recurrence as an important perspective of analysis, it was not about software development meetings, and did not address how information flows in an out of the meetings, which my work does.

2.4.4 Distributed and hybrid meetings in software development

The study of distributed, and now hybrid, collaboration also speaks to software development meetings. In general, the massive shift to remote meetings due to the pandemic, and now back to hybrid, has led to a renewed interest from the research community (e.g., [110]–[112]). Most studies remain agnostic to the meeting purpose or topic, for instance investigating how shared meeting facilities for collocated participants in hybrid meetings influence communication [113], unpacking blended technological and conversational practices of inclusion and exclusion [114], studying the experience of the remote participants and how they can be better included in the conversations (e.g.,[115], [116]), and investigating the use of chats as an important side channel that helps inclusivity and coordination but equally can be a potential distraction that is difficult to keep up with [117], [118]. To date, none of these studies centers on information flow or the affordances that recurrent meetings bring.

The company that I am studying typically organizes RSSMDMs in a hybrid modality, with some participants physically present at the same location and others joining remotely via a conference call. However, given these ten meetings were all recorded right after the COVID-19 pandemic spread globally, all of them are fully remote, with participants joining from different physical locations.

2.4.5 Tool support for meetings

Much research has focused on developing tools to support software developers during meetings. Some examples are tools to capture design drawings and diagrams, such as the ones presented in Section 2.3 (e.g., FlexiSketch [97], [98], the Interaction Room [99], [100], Calico [85]). Other examples are tools to capture what is said and agreed upon during the meetings by, for instance, augmenting written notes with audio recordings (e.g., Filochat [119], Dynomite [120], the Audio Notebook [121]) or capturing audio clips from the discussion (i.e., OctoUML [122], KNOCAP [123]).

Rather than supporting specific meeting activities such as drawings or notetaking, other researchers proposed the creation of integrated meeting rooms able to offer a variety of meeting services. One example was LiteMinutes [124], an integrated meeting room that supports text notes taken on wireless laptops, slide images captured from presentations, and video recording from cameras in the meeting room. A few years later, the conceptual design of a meeting room called SMaRT [125] was proposed. The novelty of this design was that it required minimal explicit human-computer interaction to operate, given that many tasks were designed to be triggered automatically. To mention one additional example, Haller et al. introduced the NiCE meeting room in 2010 [126]. The central piece of this meeting room was an electronic whiteboard with advanced features to, for instance, automatically transfer data from PCs (i.e., PC screenshot) and physical paper (i.e., drawings) to the whiteboard's canvas, take snapshots of the whiteboard's canvas, and create overlays to juxtapose different drawings. Moreover, some whiteboard functionalities could be handled via a physical remote control (a tool pallet) to, for instance, change the color of the pens to draw.

A disadvantage of heavy-weight tools like LiteMinutes, SMaRT, and NiCE in comparison to light-weight tools such as notetaking apps (e.g., Microsoft OneNote [127], Evernote [128], Apple Notes [129], Google Keep [130], Notion [131], Obsidian [132]),word processors (e.g., Google Docs [133], Microsoft Office 365 [134]) and web whiteboards (e.g., Mural [135], Miro [136], Google Jamboard [137]) is the need of advanced technological devices and a physical space to operate. Considering the current state of the world (due to the COVID-19 pandemic), with many teams working from home [112], this requirement is unrealistic, or at least inconvenient.

Research on video conferencing practices (e.g., [138]–[141]) and technologies (e.g., [142], [143]) has been instrumental to virtual meetings becoming a reality. For instance, Geyer et al. proposed a collaborative workspace system called TeamSpace [144], [145], the focus of which was to support virtual meetings as part of a larger collaborative work process. This tool proposed the integration of meeting information from multiple meetings, enabling users to efficiently gain knowledge of both current and past activities. As a second example, Yankelovich et al. proposed the Meeting Central prototype, a suite of collaboration tools designed to support distributed meetings, with a minimalist design that provided only those features that have the most impact on distributed meeting effectiveness [146].

Nowadays, powerful videoconferencing platforms such as Zoom [147], Google Meet [148], and Cisco WebEx [149] are part and parcel in global software development. However, these are not the only tools that software developers use to work in a distributed fashion. As in co-located meetings, tools to capture information are also needed. Typically, developers combine the affordances of videoconferencing platforms with other tools by, for instance, sharing their screen to show what they are writing in a word processor (e.g., Google Docs [133], Microsoft Office 365 [134]), an issue tracking system (e.g., Jira [150], Zenhub [151], Bugzilla [152]), or a web-based corporate wiki (i.e., Confluence [153]).

Information management is such an important issue for meetings that some researchers have proposed the creation of meeting virtual assistants to automate information management tasks. For instance, the CALO Meeting Assistant (MA) [154], part of the CALO personal assistant system, provides distributed meeting capture and annotation together with automatic transcription and semantic analysis of multiparty meetings. As another example, Squartini and Esposito investigated the design of digital assistants able to process multimodal signals in real-time, to infer contextual information and support interaction in group activities [155]. They argue that assistants should act as co-workers, actively cooperating and contributing to the group's knowledge building, and pretending to work with the group rather than acting as passive data storage devices.

2.5 Information studies

Previous work centers on studying information in the context of software engineering from two main perspectives. One of these perspectives studies the information that developers need to perform their work on day-to-day basis. The other one studies the information that developers consider important to capture for future use. My work relates to these studies in two ways. First, I use some of the methods used in this previous work to analyze the information needs of participants during the meeting, as well as the information that they captured. Second, my work contributes to the body of knowledge about information studies because it seeks to study information and information capture in an unexplored context, RSSMDM. In this section, I present studies that take an "information needs" perspective to analyze software developers' activity, as well as studies that center on information capture for future use.

2.5.1 Information needs

Many studies have taken an "information needs" perspective to analyze software developers work with the goal of designing tools to support developers in efficiently searching information during different activities. These studies focus on three main aspects: 1) analyzing what kinds of information developers need, 2) observing what sources or tools they use to obtain such information, and 3) identifying the obstacles that prevent developers from obtaining information when so needed.

Ko et al., for instance, shadowed seventeen developers to observe their information needs while performing various development activities [22]. They identified 334 moments in which developers sought information and classified these moments into 21 different categories. Categories represent generalized information needs that cover specific questions or utterances that the subjects being studied said. For example, in this work, the utterance "*Originally, the repro steps said I need a blog count [as a test case] but I couldn't set one up, so I went back and forth*" was classified into the category "in what situations does this failure occur?". Ko et al., also identified the activities in which the developers engaged during the sessions (e.g., writing code, submitting code, triaging bugs, reproducing failures, understanding program behavior, reasoning about design, maintaining awareness) to contextualize the information needs that they observed.

While Ko et al. studied information needs in the context of multiple development activities, others focused on observing information needs associated with specific development tasks. For instance, Sillito, Murphy, and De Volder studied questions that developers ask when evolving a code base [156]. To investigate this phenomenon, they performed two independent studies. The first study was a laboratory experiment of newcomers working on an open-source project, the second study involved industrial developers working on the code base of their respective companies. The focus of both studies was observing the information that the developers needed to know about the code base they were working with when making a code change (e.g., a bug fix, an enhancement). They classified the information needs into 44 different categories, with each category based on a generalized version of similar specific questions that the participants asked. The authors also investigated the tools that the participants used to obtain the information needed [157]. A noteworthy finding was the identification of 78 questions that developers commonly have of the codebase, but for which tool support is typically lacking.

Breu et al. investigated questions that developers and end users ask as part of bug reports [23]. They identified 940 frequently asked questions in the content of 600 bug reports. Breu et al. classified the questions into eight categories (i.e., missing information, clarification, triaging, debugging, correction, status inquiry, resolution, process). Some noteworthy findings that stem from this study are that only 67.66% of the questions had an associated response, and that the type of question (the category) had a significant effect on the respective response rate. For instance, questions related to "corrections" (category that contains questions that discuss how to correct a bug) were more likely to be answered than questions related to "resolution" (a category that contains questions that ask whether a bug was resolved).

Other studies about information needs focused on understanding how developers search information in the web. For instance, Treude et al. analyzed 385 questions from Stack Overflow to explore which of them were answered and which were not [158]. Their preliminary findings show that questions that ask for instructions (classified as "how-to"), questions that inquire about unexpected behaviors (classified as "discrepancy"), and questions related to development or deployment environments (classified as "environment") often remained unanswered. In a similar vein, Xia et al. studied the information developers frequently search on engines such as Google, Mozilla, or Safari [159]. They collected search queries from 60 developers and surveyed 236 software engineers to investigate it. Their results report that the participants used these web browsers to search for unknown terminology, exceptions and error messages, code snippets to reuse, solutions to common programming bugs, and suitable third-party libraries that they could use to build software. To mention one additional example, Duala-

Ekoko and Robillard studied questions that arose when developers work with unfamiliar APIs. To investigate this phenomenon, they observed 20 developers working on programming tasks in which they had to use an API they were not familiar with. Their motivation to study this kind of information needs was to figure out how to evaluate tools that aim to improve API learning.

2.5.2 Information capture

Information capture tools have been developed for different contexts and with different kinds of functionality. For instance, in Section 2.4.5, I introduced various tools including tools to capture drawings and diagrams (e.g., FlexiSketch [97], the Interaction Room [99], Calico [85]), tools to augment written notes with audio recordings (e.g., Filochat [119], Dynomite [120], the Audio Notebook [121]), tools to capture audio clips from the discussion (i.e., OctoUML [122], KNOCAP [123]), and tools to capture and index meetings via video recordings (e.g., LiteMinutes [124], SMaRT [125], NiCE meeting room [126]).

Studies about information capture have been performed in other contexts as well. For instance, knowledge management tools were particularly popular as a research topic several decades ago (e.g., [160]–[164]). Much like design rationale, this research often centered on highly structured approaches to capture and retrieve knowledge (e.g., [17], [165]–[167]), though unstructured approaches based on hypermedia or wikis were also explored considerably (e.g., [14], [168]–[170]). Interestingly, these informal, lightly structured approaches appear to have had more influence than the highly structured approaches in practice, with the use of platforms such as Confluence [153] and lightweight architecture decisions records [171], [172] being popular choices. I provide further details about information capture in the context of knowledge management and rationale in Sections 2.6 and 2.7 respectively.

2.6 Knowledge management and knowledge capture

Knowledge management for software engineering aims to support knowledge flow across different phases of a software engineering process [162]. Foundational work in this area introduced critical terminology. For instance, Vasanthapriyan, Tian, and Xiang defined the concept of "knowledge" as the interpretation of information within its context [162]. Later, Polanyi classified this concept into two broad categories: "tacit knowledge" (intuitive, unarticulated, typically obtained by experience, reflection, or individual talent) and "explicit knowledge" (objective, rational, technical, articulated in files, documents, databases) [173]. Nonaka defined "knowledge management" as the means to create a learning environment to support knowledge creation and transfer [174].

Other work focused on identifying knowledge management needs and practices [164]. Rus, Lindvall, and Sinha, for instance, observed that knowledge tracking is a real problem in practice and argued that a structured way to manage knowledge could help organizations to leverage the knowledge that they possess [175]. As another example, Fischer et al., studied the role of knowledge in long-term indirect collaboration (a kind of collaboration in which designers who originally create design knowledge may never know the people that will use and update this information in future) [10]. These authors argued that indirect collaboration should be an aspect to consider in the design of knowledge management tools. To mention one additional example, Ward and Aurum performed two case studies to investigate knowledge management practices in two different organizations. They observed that, even though several structures to store knowledge exist in both companies, knowledge remained primarily tacit.

A strand of work at the intersection of knowledge management and software architecture centers on studying how to keep architectural knowledge. This term is commonly used to refer to the high-level design of software systems, either in terms of components and connectors [176] or in terms of design decisions, assumptions, and their associated rationale [177]. Some studies that stem from this strand of work focused on understanding various aspects of architectural knowledge management. As one example, Clerc, Lago, and van Vliet performed a survey to collect feedback on the importance of architectural knowledge for the daily work of practitioners [178]. As a second example, van Heesch and Avgeriou performed a survey with 53 professional software architects to explore how architects make architectural design decisions [179].

Foundational work about architectural knowledge in particular led to many tools to manage such knowledge [166]. Some researchers, for instance, proposed to create central repositories of design information (e.g., ADkwik [17], PAKME [165]) or to recover design information from artifacts such as code and design documents (e.g., ADDRA [180], DiscoTect [181], Revealer [182]). A common characteristic across these approaches was that an underlying structure to capture information always exist. PAKME [165], for instance, proposed to capture two types of knowledge: generic knowledge (e.g., patterns, general scenarios, quality attributes) and project-specific knowledge (e.g., concrete scenarios, architectural decisions). In contrast to PAKME [165], which comes with a prescribed fixed data model, CADDMS [183] allows end users to define their own knowledge model.

Unstructured approaches based on hypermedia and wikis were also explored considerably (e.g., [14], [168]–[170], [184], [185]). Clerc et al. studied global software development practices that could be implemented using wikis [14] and Sousa et al. studied how an organizational wiki could be used as a knowledge management tool from the point of view of two knowledge management models [168]. More recently, grey literature on lightweight architecture decision records [19] proposes the use of version control systems (similar to GitHub [186]) to keep decision records and their history [171], [172]. Some lightweight approaches to capture information have also been proposed to capture lightweight knowledge records during meetings (e.g., KNOCAP [123], OctoUML [122]). A common factor in these proposals is that knowledge is directly captured from conversations.

My work relates to this body of knowledge because it seeks to study knowledge management practices and what tools are used to capture knowledge in a particular kind of meeting, the RSSMDM.

2.7 Design rationale

One strand of research to which my work relates closely is that of design rationale, which has a long-standing history of seeking to understand and provide support for documenting the rationale behind software design decisions (e.g., [21], [187]–[190]). As already discussed, this work often limits itself to recognized concepts such as ideas, arguments, alternatives, constraints, issues, and decisions, with the resulting tools often structured accordingly (e.g., [160], [191]–[194]).

An extensive strand of research has focused on meeting capture to preserve important aspects of discussions for later, as motivated by the need to capture rationale.

Some of these tools are generic (e.g., the Audio Notebook [191], Dynomite [120]), and others are specialized toward software design [195]. Even though useful to avoid lost knowledge, most of these tools focus on capture and preservation only, with no features for bringing captured information back into future meetings, other than making the information available through search (e.g., [196]).

My work sets itself apart by focusing on all kinds of information that may be important in future and not just rationale, as well as by proposing the exploration of unstructured approaches to re-using (when so needed) and capturing discussion outflow in a lightweight manner during RSSMDMs.

3 The topics that are discussed in RSSMDMs

My dissertation contributes an in-depth single-exploratory case study [27], [28] that centers on ten RSSMDMs held by a high-performing development team at a major healthcare software development company. Important characteristics about these meetings are that they are regularly scheduled, are maintenance oriented, cover multiple topics (I use the term topic to refer to the discussion of an independent maintenance design issue during a meeting), and involve participants who are distributed. The ten meetings that I have studied each last about an hour, sometimes a bit less or more, and took place from March 24th to July 16th, 2020. In this section, I provide the background of the meeting participants, describe the meeting setting, and delve into the nature of the discussions held during the meetings.

3.1 The meeting participants

A total of twelve participants joined the ten RSSMDMs. Table 1 shows all the participants, preferred pronoun, role, and which meetings each participant attended. Note the different roles of the participants and the mostly even distribution of participants from the U.S.A. and India, the latter to where most development is outsourced. The first four

Table 1. Meeting participants

				Meetings attended									
Participant	Preferred pronoun	Role	Location	M1 Mar. 24th	M2 Mar. 31st	M3 Apr. 8th	M4 Apr. 14th	M5 Apr. 21st	M6 Jun. 5th	M7 Jun. 12th	M8 Jun. 19th	M9 Jul. 2nd	M10 Jul. 16th
1	she, her, hers	Product owner	U.S.A.	X	X	X	X	X	X	X			X
2	he, him, his	Software architect	U.S.A.	X	X	X	X	X	X	X	X	X	X
3	he, him, his	Software architect	U.S.A.	X	X	X	X	X	X	X	X	X	X
4	he, him, his	QA engineer	U.S.A.	X	X	X	X		X	X	X	X	
5	she, her, hers	Product owner	India	X		X	X		X				X
6	he, him, his	Manager	U.S.A.			X	X	X	X	X	X		
7	he, him, his	Developer	India	X		X	X	X	X		X		X
8	he, him, his	Developer	India				X	X	X			X	X
9	he, him, his	Developer	India				X		X			X	X
10	he, him, his	Developer	India		X			X					
11	he, him, his	Manager	India				X						
12	he, him, his	Infrastructure engineer	India								X	X	

participants (P1-P4) are the key internal stakeholders (as defined in Chapter 1) who run these meetings. This group of team leads is composed of a product owner (P1), two software architects (P2 and P3), and a quality assurance engineer (P4), all of whom are in the U.S. and attend these meetings on a regular basis. The two architects (P2 and P3) attended all ten meetings, the product owner (P1) missed only one, and the quality control engineer (P4) missed two of them. The rest of the attendees are all non-core meeting participants who attend the meetings on a more incidental basis. Most of these non-core participants are in India, except for a manager, who joins the meetings from the U.S.A. The participants from India include another manager, a shadow product owner, an infrastructure engineer, and developers at a range of levels.

The role of the key internal stakeholders in the discussions held in RSSMDMs is primarily that of experts, who propose ideas to solve complex design issues that have a long-term impact on the system (e.g., technical debt, performance issues, security issues). They often discuss topics of this nature by themselves, though non-core participants are welcome to be present and contribute if they want to. By virtue of being part of the organization longer, the key internal stakeholders have more knowledge about the codebase, customers, projects, and past decisions associated with them. Therefore, another responsibility that they have during the meetings is that of sharing knowledge (e.g., good programming practices, internal processes) with the non-core participants.

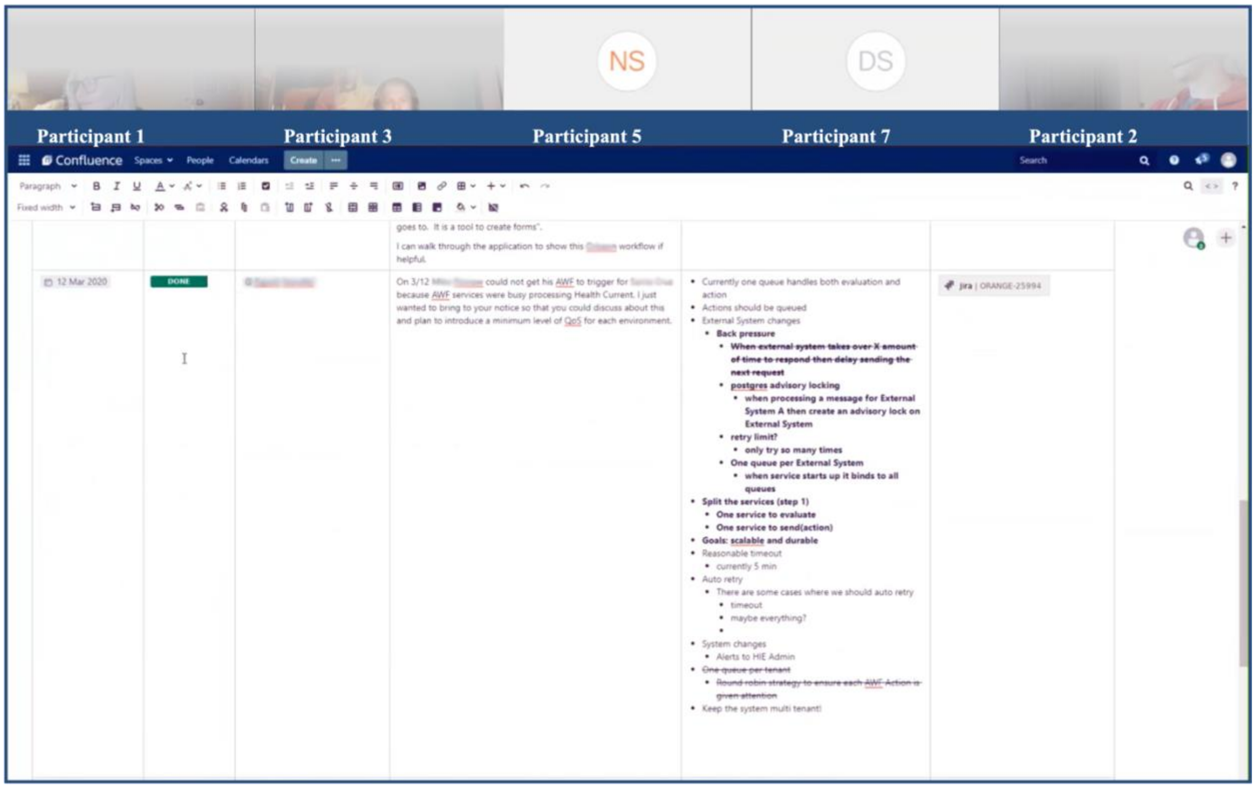
The role of non-core participants in the meetings is more that of an on-demand contributor. They may join the meetings all the times they want to, though they typically join when they need to or upon request of the key internal stakeholders. For example, non-core participants may attend the meetings because they need advice on how to handle issues to which they have been assigned, because their presence is critical to discuss a specific high-level maintenance issue, or to learn practices and processes that the architects want to establish.

3.2 The meeting setting

Figure 2 shows the look and feel of a typical meeting setting. The figure shows only five participants on the screen. However, seven other participants were connected to the call. The three participants with their camera on (P1-P3) are key internal stakeholders. The other two people in Figure 2 are non-core participants (P5 and P7).

The meetings are typically run based on an agenda, though this agenda is not always the same kind of artifact. Sometimes, for instance, the key internal stakeholders use a wiki page with a list of issue ticket links as the meeting agenda. Other times they use a plain wiki

Figure 2. Typical virtual setting of the meeting.



goes to. It is a tool to create forms".

I can walk through the application to show this [redacted] workflow if helpful.

12 Mar 2020

DONE

[redacted]

On 3/12 [redacted] could not get his AWF to trigger for [redacted] because AWF services were busy processing Health Current. I just wanted to bring to your notice so that you could discuss about this and plan to introduce a minimum level of QoS for each environment.

- Currently one queue handles both evaluation and action
- Actions should be queued
- External System changes
  - **Back pressure**
    - ~~**When external system takes over X amount of time to respond then delay sending the next request**~~
    - **postgres advisory locking**
      - **when processing a message for External System A then create an advisory lock on External System**
    - **retry limit?**
      - **only try so many times**
    - **One queue per External System**
      - **when service starts up it binds to all queues**
- **Split the services (step 1)**
  - **One service to evaluate**
  - **One service to send(action)**
- **Goals: scalable and durable**
- Reasonable timeout
  - currently 5 min
- Auto retry
  - There are some cases where we should auto retry
    - timeout
    - maybe everything?
    -
- System changes
  - Alerts to HIE Admin
- ~~One queue per tenant~~
  - ~~Round robin strategy to ensure each AWF Action is given attention~~
- Keep the system multi tenant!

Jira | ORANGE-25994

page with a list of bullets that a participant prepared before the meeting. In some meetings, the participants also use a pre-populated list of topics with questions and concerns that the participants want to discuss as the meeting agenda. They call this artifact “Ask an Architect”: a wiki page with a pre-loaded table template to which the team, and occasionally employees from other teams in the organization, can add to receive help from the architects (Figure 2 shows an example of what it looks like). Each row in this table contains relevant information for a topic that someone has requested to be discussed: a submission date, the topic status, who submitted the topic, the description of the topic, and the outflow of discussing it once the topic discussion commences (e.g., decisions made, agreements, procedures to establish).

In these RSSMDMs, the participants discuss as many topics as they can within the meeting timeframe (about an hour). When time is over, they roll the rest of the topics over to a next meeting. P2 (one of the software architects) typically coordinates the meetings by walking through the meeting agenda (e.g., an “Ask an Architect” wiki page, plain wiki page, list of tickets) and selecting topics to discuss based on the priority for the team. P2 typically provides or asks another participant to provide an overview of the topic to be discussed.

There are also occasions when non-core participants request to discuss topics that were not part of the meeting agenda. Non-core participants often make these impromptu requests when they need to discuss urgent matters (i.e., a support case that has arisen). The key internal stakeholders give priority to discussing these requests. The participant who made the request typically starts the discussion by providing the background of the issue. Then, the discussion focus switches to defining a plan of action to either obtain more information about the issue in question or brainstorm how to solve the problem.

3.3 Methodology

My dissertation aims to analyze how prior information is shared in RSSMDMs, what new information is captured as outflow, and what tools participants use to do either of these. For these analyses, it is critical to place prior information that is shared and the meeting outflow captured in the context of the discussions in which this information was shared or captured. The reason is that participants may share different kinds of information or capture specific kinds of outflow depending on the nature of the discussion. As one example, participants may mention deployment information (e.g., server logs, reports from monitoring tools) as part of the overview of a support case. Then, they may capture some decisions as to who should handle the case together with some high-level direction to do so in a (newly created) Jira ticket. As another example, participants may share knowledge about past solutions while reviewing the design of a new feature to integrate into the application, to then capture the feedback provided into an (existent) wiki page.

As a preliminary step to studying the meetings, two researchers (one of them myself) independently partitioned each meeting according to the different topics being addressed in it. Then, they compared their partitioning, discussed differences, and agreed on a unique list of topics. Detecting when the discussion of a new topic begins was relatively evident in these meetings. Sometimes, the participants would verbally end a discussion and signal a moving on to the next topic (i.e., "*… But as of now, my questions are done for this ticket. Yeah, and I move to another ticket which is around app integration feature.*"), would announce that they need to discuss something urgent (i.e., *"… there's a present issue at [client name], uh. I'd reckon we'll discuss that …"*) or would simply change the content shared on screen to implicitly signal they are about to start a new topic discussion (i.e., switching from one Jira ticket to another). Another aspect to consider along this partitioning process was recurrence: when topics were re-discussed within a meeting or at different meetings, we kept the same topic id in the respective conversation fragments. A preliminary fragmentation of the meetings into topic discussions was presented in [197], with the result below a refinement over those initial results based on a refined coding pass[2].

After partitioning the meetings into topics, I categorized all the topics identified in two ways. The first relates to the kind of work that the participants discussed (e.g., support case, integration of new functionality, practices and processes). The second relates to the overall purpose of the discussion (e.g., devising a solution, gathering knowledge, reviewing a design proposal). The process to categorize the topics according to these two perspectives was as follows: I first developed an initial categorization that a second researcher independently reviewed. This review led to suggestions for improvement. By discussing the suggestions and iterating through this process of one researcher updating the codes and another reviewing the updates, the final categorizations emerged.

3.4 The topics that were discussed

Across the ten meetings, a total of 45 topics were discussed. Table 2 shows the resulting list of topics after the last revision. The first column shows the identifier assigned to each topic; the second lists the meetings in which each topic was discussed; the third and fourth columns, respectively, show the name and description of topics. Note that the table

[2] The updates applied were: 1) merging topics T_3 and T_{41}, I assigned the same identifier to both discussions (T_3), and 2) re-numerating the identifiers of topics T_{42}-T_{46} to T_{41}-T_{45} respectively).

Table 2. List of topics discussed across the ten meetings.

Topic id	Meetings	Name	Description
T_1	M1, M4	AWS accounts for team members in India	Check the status of AWS accounts that the team members in India had requested.
T_2	M1	Client onboarding additional users	The team evaluates whether a client onboarding additional users may generate additional computational resources that the client should be charged for.
T_3	M1	Noisy neighbor	An architectural issue in which unusual traffic of some clients may affect others because the clients share computational resources.
T_4	M1	In house IdP	A client decides to build its own IdP (Identity Provider) solution instead of the one the team recommends. They evaluate the risks of this decision.
T_5	M2	Connect over public internet	Discussion centers on clarifying whether it is safe to connect one of their software products over the public Internet.
T_6	M2	Performance testing baggle	Discussion that centers on the design of major improvements to their testing suite.
T_7	M3	App loading response time	Discussion centers on reviewing the design of a performance test for certain functionality (e.g., scenarios to consider, input boundaries).
T_8	M3	Model mapping	The meeting participants provide feedback on the design of a new feature.
T_9	M4	Upgrade failure	The meeting participants walk through an issue and how it was handled to share with the rest of the team how to avoid the issue and improve their practice.
T_{10}	M5	Connectivity issues	The discussion addresses a connectivity issue with docker containers.
T_{11}	M5	Left behind testing environments	The meeting participants touch base on how they create and get rid of unused environments with the goal of improving the team practice in regards of deployment.
T_{12}	M5	Shut down unused environments	The meeting participants get rid of many unused environments during the meeting upon request of a manager.
T_{13}	M6	Accounts request	The team in India urgently requests more privileges in their accounts to monitor software.
T_{14}	M6, M8	APPSEC	One of the meeting participants shares the status of a project to integrate APPSEC (security software to check vulnerabilities) into their development pipeline.
T_{15}	M6	Broken provider shared functionality	The discussion centers on an open issue about a functionality that is not working. Participants have different ideas on what works and does not.
T_{16}	M6	Documentation template	The discussion focus is a proposal of a documentation template to track functional details of the system (how it works for the end-user) along with majority of design details (how it works internally).
T_{17}	M6	Health check	The participants discuss a ticket about a new development.
T_{18}	M6	Client SC update	The participants check the status of an open (but not urgent) issue with a client.
T_{19}	M7	Access from local environment	The participants discuss why a development account is not working as it should.
T_{20}	M7	Automating upgrades	The meeting participants discuss the establishment of a standard procedure to perform maintenance upgrades.
T_{21}	M7	Redis restart problem	The meeting participants walk through how a support case was improperly handled (because of Redis[3] DB being restarted), and how it should be handled correctly instead.
T_{22}	M8	Priorities	Discussion centered on the aspects that should be considered to define the priority of tickets.

[3] Redis is an in-memory data structure store, used as a distributed, in-memory key–value database, cache and message broker, with optional durability.

T_{23}	M8	First databank feature	The participants review the proposed design of a new feature.
T_{24}	M8	Infrastructure baggle	Participants discuss how to organize a huge list of tickets about infrastructure improvements.
T_{25}	M8	Quadrant	The meeting participants share a grid that will help them to prioritize tickets.
T_{26}	M8	Single sign-on discussion	The team schedules the discussion of Single sing-on technology and procedures for a future meeting.
T_{27}	M8	Ticket 24325	The participants discuss a ticket to reduce lock contention on task definitions.
T_{28}	M8	Ticket 26159	The participants discuss a ticket about their load balancer not being compliant with the company building.
T_{29}	M9	Ticket 23482	The participants discuss a ticket related to build the UI of a routine that monitors services and servers that they use.
T_{30}	M9	Ticket 23979	The participants discuss a ticket to monitor API requests that time out.
T_{31}	M9	Ticket 24021	The participants discuss a ticket related to Redis high CPU utilization.
T_{32}	M9	Ticket 24178	Participants discuss a ticket to set up Elasticsearch[4] logs.
T_{33}	M9	Ticket 25059 / 25590	The participants discuss tickets related to upgrading Elasticsearch.
T_{34}	M9	Ticket 25356	The participants discuss a ticket to track browser usage through Real User Monitoring (RUM)[5]. The ticket was already solved so they just close it.
T_{35}	M9	Ticket 26058	The participants discuss a ticket related to an Amazon Relational Database Service (RDS)[6] instance with CPU utilization spiking to 100% and staying there. They currently work around this problem by rebooting the instance, but a better solution is needed.
T_{36}	M9	Ticket 26440	The participants discuss a ticket to trigger an alert when a service hits its maximum scalability.
T_{37}	M9	Ticket 26626	The participants discuss a ticket about a mechanism to automatically collect metrics from containers.
T_{38}	M9	Ticket 26869	The participants discuss the development of a monitoring dashboard for an internal team that provides support to clients. They need to easily check the environments' information.
T_{39}	M10	Testing pipeline doubts	Discussion about concerns developers have with the kind of testing that should be done on a regular basis and what kind of technology should be used to create new test cases.
T_{40}	M10	Authentication across system	Discussion about an open issue at one of the client sites to integrate two different applications under the same authentication mechanism.
T_{41}	M6	End-to-end tests broken	Discussion about end-to-end tests that do not work as they should. The meeting participants share that there are many tickets to fix those issues, but that doing that work will take time.
T_{42}	M6	Embedding documentation	Discussion about ideas on how to improve documentation for end users inside their application.
T_{43}	M8	Conversation about defects	Brief discussion about the need of having a defect backlog meeting soon.
T_{44}	M8	Ticket 20441	Participants close a duplicated ticket to upgrade Elasticsearch. They kept the tickets discussed in T_{33}.
T_{45}	M9	Infrastructure bucket summary	Discussion to coordinate on the way to triage a long list of tickets about infrastructure.

[4] Elasticsearch is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents.

[5] Real user monitoring is a passive monitoring technology that records all user interaction with a website or client interacting with a server or cloud-based application.

[6] Amazon Relational Database Service is a distributed relational database service by Amazon Web Services.

is ordered by topic id; this is the identifier we use to track all the times a topic was discussed. For example, if a topic was discussed twice within a meeting or across different meetings, every discussion fragment will have the same topic id assigned.

The kinds of work addressed during the meetings were broad, with discussions ranging from high-level issues such as the re-design of components of the system's architecture to day-to-day issues such as support cases or the integration of new features. The purpose of the discussions was also varied. While some discussions were preliminary and centered on gathering knowledge to start new projects, other discussions sought to solve issues with ongoing projects. In the next sections, I analyze the 45 topics observed from two different perspectives: (1) the kind of work addressed in the meetings, and (2) the purpose of the discussions.

3.5 The kind of work addressed in the meetings

Table 3 presents the classification of the kinds of work discussed in the meetings. This categorization consists of nine different kinds of work that inductively emerged from my analysis of the discussions (conform the procedures described in Section 3.3). Each kind of work has a unique name (first column) and a description (third column), with the fourth column listing which topic discussions were of which kind and the fifth column listing the total number of topics in each category. The second column shows a meta-classification of the kinds of work observed according to the dimensions of maintenance work proposed by Swanson (ADAPTATIVE, CORRECTIVE, and PERFECTIVE). Note that not all topics discussed fit the maintenance dimensions of Swanson.

Table 3. The kinds of work addressed in the meetings according to Swanson's maintenance dimensions.

Type of work	Dimension of maintenance (Swanson, 1976)	Description	Topics	Topics per category
ADMINISTRATIVE TASK	N/A	Administrative tasks needed to provide maintenance to the software (it is not the maintenance work itself). For instance, tasks related to the management of development tools and services and tasks related to providing access to developers to these tools and services, so that they can use them to work on maintenance issues.	T_1, T_{12}, T_{13}, T_{19}	4
ARCHITECTURAL RE-DESIGN	ADAPTATIVE	Design work at the architectural level that centers on re-designing components of the architecture of a deployed and functioning system to prevent future failures (e.g., performance, security).	T_3	1
COORDINATION AND PLANNING	N/A	Coordinating the roadmap of future projects, planning the work to discuss in future meetings, and coordinating activities to carry out after the meeting.	T_{14}, T_{24}, T_{26}, T_{43}, T_{45}	5
INFRASTRUCTURE	PERFECTIVE	Improvements to monitor the software, upgrade the software stack, keep the software in compliance with standards and laws, or reducing costs in terms of the software stack.	T_{27-38}, T_{44}	13
INTEGRATION OF NEW FUNCTIONALITY	ADAPTATIVE	Features to meet new expectations, customer needs, and more. These new features require design and sometimes re-design of the front- and back-end of the system.	T_4, T_8, T_{17}, T_{23}, T_{42}	5
PERFORMANCE	ADAPTATIVE	Evaluating the performance of the system under circumstances different from the ones for which it was originally designed, so to adapt it to upcoming changes in the environment (i.e., higher demand because of additional customers onboarding)	T_2	1
PRACTICE AND PROCESS	N/A	Discussion on how to improve the way the team works by proposing new processes and practices or changes to them.	T_{9-11}, T_{16}, T_{20-22}, T_{25}	8
SUPPORT CASE	CORRECTIVE	The correction of misfunctions of the software due to wrong configurations, a defect in the code, or a deficient design that does not performs as expected in real circumstances.	T_5, T_{15}, T_{18}, T_{40}	4
TESTING	PERFECTIVE	Designing individual tests or an overall testing pipeline to detect processing inefficiencies and performance issues in the software before it is deployed.	T_6, T_7, T_{39}, T_{41}	4
				45

Below, I discuss a range of observations associated with the categorization shown in Table 3. For each, I first list the observation, and then illustrate the observation with an example obtained from the RSSMDMs. A first observation distilled from Table 3 is that:

Observation 1: All kinds of maintenance work (ADAPTATIVE, PERFECTIVE, CORRECTIVE) are addressed in the meetings.

In these meetings, only four topics address CORRECTIVE maintenance. All these discussions are about SUPPORT CASES that, most of the time, a participant requests to discuss, sometimes to obtain help from the key internal stakeholders in diagnosing the issue and other times to brainstorm ideas to solve the actual problem. The excerpt below (from T_{40}) is an example of how a participant requests to discuss a SUPPORT CASE this participant had been assigned.

> ***P7:*** *Okay, and some peculiar, uh, list of questions like there's a present issue at [Confidential], uh. I'd reckon we'll discuss that or…*
> ***P1:*** *Uh, sure. Go for it.*
> ***P7:*** *Okay, so. Um, I will share my screen and share the email chain, hang on. [P2], can I interrupt?*
> ***P2:*** *Yeah, go ahead. I'll –*
> ***P7:*** *So, the background here, in last week, uh, um, [customer] was trying to integrate …*

In comparison to CORRECTIVE maintenance, ADAPTATIVE maintenance was discussed twice as often. Specifically, the participants discussed how to integrate A NEW FUNCTIONALITY into the system (T_4, T_8, T_{17}, T_{23}, T_{43}), how to meet new PERFORMANCE requirements (T_2), and how to prepare the system's architecture (ARCHITECTURAL RE-DESIGN) to account for higher usage levels in future (T_3). The excerpt below illustrates the overview provided to kick off topic T_{23}, which addressed the integration of a new feature into the system.

> ***P7:*** *Can you all see the screen?*
> ***P2:*** *Yeah.*
> ***P7:*** *Okay. So, I mean most of us have the context. So, yesterday, we had a discussion in the team, and [name] briefed us about, uh, what we're trying to do here. So, during the discussion, we came out with seven approaches that I wanted to, um, gather feedback on, uh, depending upon how you want to handle. So, uh, basically, just for the interest of all here, uh, what we're trying to do here is to, uh, get the standard codes from [Confidential], uh, [audio cuts out] and, uh, take over this pain from the clients so that they don't have to, um – they don't have to upgrade it and maintain it each time that it'll change, which could be probably weekly, fortnightly, or monthly. So, essentially, taking over that pain away from the client and since it is common for all the, uh, clients, then we were thinking about what other areas we can, uh, take that up.*

Maintenance work of a PERFECTIVE nature was the dimension in which most topics were discussed (17 topics in total). These topics centered on improving the maintainability of the software by designing a better TESTING suite (four topics) and on improving components of the software stack (INFRASTRUCTURE) upon which the system runs (13 topics). At a first glance, PERFECTIVE maintenance seems to be discussed much more than CORRECTIVE and ADAPTATIVE maintenance. However, many topics classified as INFRASTRUCTURE (T_{27-38}, T_{44}) were discussed as part of a ticket triage session. During ticket triage sessions, discussions were relatively short in comparison to, for instance, discussing a SUPPORT CASE or an ARCHITECTURAL RE-DESIGN. Thus, more topics could be addressed within the usual meeting timeframe of an hour.

Observation 2: The participants also engage in discussing topics that are not maintenance work in the sense of Swanson, but that are critical for the team to operate.

The team also discussed topics that allowed them to improve their practice (e.g., how they handle issues, how they use resources, how they develop software), solve operative roadblocks (i.e., not having access to a server or tool) efficiently, and coordinate on a variety of activities. I classified these discussions into three groups, PRACTICE AND PROCESS, ADMINISTRATIVE TASK, and COORDINATION AND PLANNING.

As an example of discussions in which ADMINISTRATIVE TASKs were addressed, in T_{13}, P5 requests the key internal stakeholders to create additional accounts to access a monitoring service tool for the team members in India (i.e., *P5: "Is it possible to provide at least one or two, um, access, maybe to [Name] and, uh, to [Name] or any of the people?"*). Having access to run-time information is essential for the developers to be able to debug problems. P2 spends about five minutes sorting out who can have access to this tool (i.e., *P2: "So, everyone can get access – uh, access to the production account"*) and in creating the new accounts with the appropriate level of permissions. This discussion was efficient because the manager that had to approve the creation of the accounts and the associated role each account should have, was present in the meeting (i.e., *P6: "Not admin. So, let's do operational"*). Sorting this out via email or Slack could have taken much more time. The discussions of T_1 and T_{19} also centered on creating accounts for the development team.

In topic T_{12}, the participants centered on getting rid of unused servers, a clean-up activity they had not performed in a while and that the manager (P6) urged them to address. I provide detailed examples of both kinds of ADMINISTRATIVE TASKS in Section 3.6.

As an example of discussions in which the team coordinates and plans (COORDINATION AND PLANNING), in T_{14} P2 shared an update about a new project in which the team was expected to participate, the excerpt below illustrates the information shared by P2:

> ***P2:*** *Um, cool. Um, [project name] update, I just got this today. [person name] is always very bad planning ahead, I guess. I have a meeting tomorrow at 10:00 to go over next steps. Um, it looks like we'll be doing planning, essentially. Um, obviously it's up to the team to come up with shard sizing and endpoints an all that stuff, so what we'll probably start doing is crating Jira tickets for what we had planned and getting those together. Um, so we'll see.*

This information came from an email that only P2 received. However, it was important to share it with the rest of the team so that they could prepare for it. The discussions of T_{24}, T_{43}, and T_{45} were similar and involved announcements of upcoming tasks and work distribution. T_{26}, however, was a bit different in nature. During this discussion, the team centered on planning the discussion of a topic of interest for the team in India. I provide examples of some of these discussions in Section 3.6 as well.

The participants also engaged in discussing aspects of their internal practice (PRACTICE AND PROCESS). For instance, some discussions focused on creating socio-technical documentation (T_{16}) and a pair of discussions defined a systematic way to triage tickets (T_{22}, T_{25}). The participants also held discussions in which they walked through issues with the software to evaluate their actions and learn from them (T_9, T_{10}, T_{21}). Moreover, they discussed how to automate cumbersome activities by simplifying them (T_{11}, T_{20}). I provide examples of these topics in Section 3.6.

3.6 The overall purpose of the discussions the participants held

Table 4 presents the topics' classification based on the overall purpose of the discussions. The table consists of 14 categories of topic discussions, organized alphabetically. Each type of discussion has a unique name (first column) and a description (second column), with the third column listing which topic discussions were of which type and the fourth listing the total number of topics assigned to each category. Note that each topic discussion is only assigned to a single type. While some exhibited a flavor of more than one type (e.g., PROBLEM UNDERSTANDING with some DEVISING A SOLUTION interwoven), one type always dominated the other considerably in how much time was spent on it in the discussion. Moreover, team members often would make it clear what they needed out of a discussion (e.g., they stated they would like to have feedback on a proposed solution they were presenting – REVIEWING A DESIGN). In cases where a discussion also exhibited aspects of some other types of discussion (e.g., a team member who identifies a problem with a design – ASSESSING A PROBLEM – and who also shares some ideas for how

Table 4. Categorization of topic discussions (in alphabetical order) in terms of the purpose of the discussion, together with which topic discussions are of each type.

Purpose of discussion	Description	Topics	Topics per category
ASSESSING A PROBLEM	A new problem has arisen that must be addressed by the team, but the problem is not well understood; the primary goal of the discussion is to build a better understanding of what might be transpiring, how serious of a problem it is, etc.	T_{15}, T_{18}, T_{40}, T_{41}, T_{42}	5
AUTOMATING ACTIVITIES	Discussion centers on automating inefficient activities or parts of them (i.e., database upgrades, the destruction of testing environments).	T_{11}, T_{20}	2
CLARIFYING A MISUNDERSTANDING	Discussion centers on clarifying a question that arose concerning some past decision about planning, the architecture, the code, or deployment, and ensuring that everybody is on the same page; the clarification may regard an internal misunderstanding by someone on the team, or by someone outside the team.	T_5, T_{39}	2
DEFINING INTERNAL PRACTICES	Discussion centers on clarifying, defining, and sometimes documenting practices to improve the way the team handles internal activities (e.g., documenting their practices, prioritizing tickets).	T_{16}, T_{22}, T_{24}, T_{25}	4
DEVISING A SOLUTION	Given an issue—bug, deployment problem, newly needed functionality—that is already well-understood by at least one of the participants, the discussion centers on identifying one or more solutions for how to address the issue (e.g., bug fix, process change, proposed design).	T_3, T_6	2
GATHERING KNOWLEDGE	A discussion in which the team focuses on providing one or more of the team members with comprehensive knowledge and considerations that they will need to discuss the topic with the clients and/or executives in future; often focuses on complexity, feasibility, potential risks, and so on.	T_2, T_4	2
MANAGING ACCOUNTS	A discussion that centers on setting up accounts that the team members need to, for instance, access specific environments, use monitoring tools, development tools, or testing tools; sometimes, participants describe the process that should be followed to set up accounts, other times they execute the actions needed to set up the accounts.	T_1, T_{13}, T_{19}	3
MANAGING COMPUTATIONAL RESOURCES	A discussion about work that concerns steps to set up or clean up environments (server instances) that the team members use to work; typically, participants do not only discuss these steps, but act on them during the meeting.	T_{12}	1
PERFORMING A POST-MORTEM	Discussing a prior problem to understand what went right, what went wrong, and what the team could have done differently and should do differently next time to improve the experience for everyone involved; can concern process issues as well as coding issues.	T_9, T_{10}, T_{21}	3
PLANNING A FUTURE MEETING AGENDA	A meta discussion in which the team engages in a discussion as to what kinds of topics they should discuss in the next meeting; often it is spurred by one member needing a topic discussed, but results in an open conversation as to what else they might have ignored and should tackle.	T_{26}	1
PLANNING HOW TO TRIAGE TICKETS	A discussion that centers on figuring out an optimal way to triage long backlogs of tickets; it is not a ticket triage discussion, but a prior conversation to define how they will go about it.	T_{45}	1
REVIEWING A DESIGN PROPOSAL	A team member brings a proposed design solution to the team with the explicit objective to receive feedback and spot potential flaws early, so to be able to refine the design before some developer (or set of developers) is tasked with actually implementing it.	T_7, T_8, T_{17}, T_{23}	4
SHARING INFORMATION ABOUT FUTURE PROJECTS	A discussion that centers on providing information about future projects. Sometimes informally, other times as a formal announcement that includes important dates and milestones.	T_{14}, T_{43}	2
TRIAGING A TICKET	Assessing a ticket in the ticket management system (Jira) as to where it falls in terms of its priority for the team to address it; discussion typically considers severity of the issue, difficulty of implementation and/or code changes, cost-benefit, and more.	$T_{27\text{-}38}$, T_{44}	13
			45

to overcome the problem – DEVISING A SOLUTION), the participant stating the purpose helped us in deciding the primary type of discussion, resulting in the mapping shown in Table 4.

While closely related, the type of work and the discussion purpose are different. The former is higher level, with the latter applicable in more than one of the types of work. That is, a topic discussion may pursue a different purpose for the same type of work. For instance, for T_9 and T_{16} the kind of work was related to the team's internal way of doing things (PRACTICE AND PROCESS), but the purpose of each of these topic discussions was different. T_9 centered on walking through a recent issue (PERFORMING A POST-MORTEM) from which lessons could be learned to improve the team's practice. On the other hand, T_{16} focused on documenting a formal way to perform a task (DEFINING INTERNAL PRACTICES).

Across the multiple instances of each discussion purpose, I observed common characteristics that together define the essence of each discussion's purpose. Below, I discuss each of the 14 categories shown in Table 4 in more detail. I first show a list of common characteristics of each discussion purpose. Then, I provide an example that maps such characteristics to the context of a real topic discussed in the meetings.

ASSESSING A PROBLEM
- Participants request to discuss a problem that has recently arisen.
- Not all participants are aware that this problem exists.
- The outcome of the discussion can be a diagnosis, or a plan to obtain more information about the problem. The participants may also suggest to delegate work to other teams.

T_{15} is one of the discussions in which the main purpose is ASSESSING A PROBLEM. This discussion begins with P5 requesting to discuss an issue reported on the support Slack channel. Someone from the support team reported that certain functionality in one of their products was not working. The excerpt below shows how P5 requested to discuss this problem.

> ***P5:*** *It is very clear that when we worked on share feature, provider was a recipient as well. [Inaudible] and it worked, … so I'm wondering when did it break? And if it is broken, how come we didn't, uh – we didn't get any notification …*
> ***P3:*** *Anyone know about this? Broken provider sharing?*
> ***P5:*** *Yeah, the [inaudible] is going on the support channel. [name] has asked.*

The excerpt does not show it, but a formal ticket reporting the issue had not been created yet. Note that P5, who requested to discuss this issue, did mention that the problem had been reported recently, via a Slack channel. Also, note that many participants, including some of the key internal stakeholders, were not aware of this problem.

P5 shared her screen with an instance of the system where the issue could be reproduced to demonstrate it. After demonstrating the issue several times, one of the architects mentioned a number of possible causes. The excerpt below illustrates.

> ***P2:*** *It could be a bad user experience, I guess. So, that – I mean, it could be actual defect. It could be, uh, user experience, too. It could be, um, bad configuration, which is another user experience problem, or maybe a training problem.*

The outcome of this discussion was delegating the issue's diagnosis to the support team. The architects concluded that it was too early for them in the process to diagnose what exactly the problem was because the causes could be many. Their advice was to let the support team work on this issue a few more, so that they could run tests to potentially discard a wrong configuration or a training problem as the cause. They also advised P5 to wait for the support team to submit a ticket to engage in this issue.

CLARIFYING A MISUNDERSTANDING

- An internal misunderstanding among the team members or between the team and external people exists.
- The discussion centers on clarifying the misunderstanding.
- The outcome of this discussion is an improved understanding that the participants sometimes share with external people.

In T_5, P10 requests to discuss a misunderstanding between two external people (no team members) that was going on in a Slack channel. The external people requested that someone from the development team clarify whether or not an envisioned solution could go through the public Internet. The excerpt below illustrates.

> ***P10:*** *Hi team, good morning. So, I was listening to all of you, uh, patiently, [inaudible] and then, uh, because this is all – I mean, it is almost in end of the meeting. I – I just wanted two minutes. Uh, [P3], uh – uh – the reply which we got from [person name 1], uh, on that third-party – MC – MC, channel, right? Uh, so, shall I – shall I mention the same, uh, you know, to, uh, [inaudible]?*
> ***P3:*** *I mean, the only – the only thing I'd – the only thing that gives me hesitation is [person name 2] wrote in bold, "MC will not connect to public internet." What [person name 1] said is, "It will be secure over SSL," which is connected to the public internet.*
> ***P10:*** *Yeah.*
> *…*
> ***P2:*** *It was Slack – on our Slack channel. I was looking over at [person name 2]'s response. It looked pretty good, except for the public part.*

In the subsequent discussion, the architects determined that neither of the two external people was completely right or wrong. Instead, the right approach was a combination of both proposals: connecting the MC channel to the public internet, but over SSL. The architects then guided P10 in what he should reply on the Slack channel to clarify the misunderstanding.

DEFINING INTERNAL PRACTICES

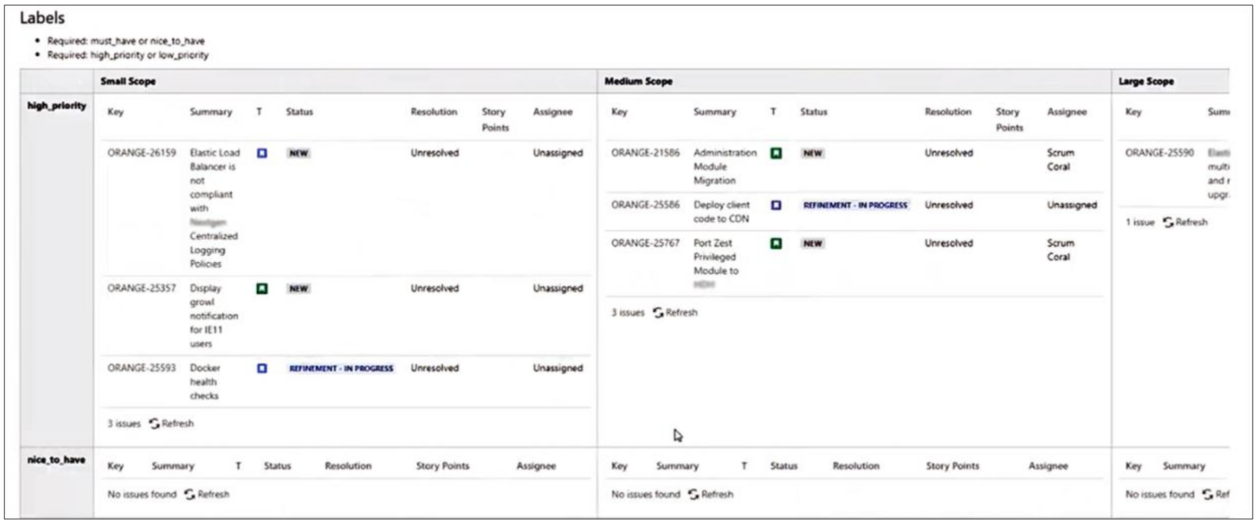
- Participants seek to standardize the way certain activities are handled.
- Sometimes, the participants use the meeting to brainstorm ideas towards defining a practice. Other times, they use the meeting to present practices previously worked out.
- The outcome of the discussion is the establishment of a new practice that all team members should follow.

T_{25} is an example of a discussion focused on DEFINING INTERNAL PRACTICES. Before this discussion, the key internal stakeholders defined a matrix to evaluate the priority and scope of tickets during ticket triage sessions. They had temporarily called it "the quadrant", though they were still unsure about using that name. The architects presented the matrix to the rest of the team on behalf of the product owner (P1), who was not present during this meeting. The idea was that, from that day forward, everybody ideally should use the matrix to systematically decide the priority and scope of tickets. The excerpt below illustrates how P2 introduced the matrix to the rest of the team. Then, P3 explained the motivation behind the matrix and how they should use it.

> ***P2:*** *... Um, P1 wanted us to look at this – it's no longer really a quadrant. Um, I don't know. What – what do you guys call this? A, uh...*
> *...*
> ***P3:*** *... At least then when we look at a ticket, we can say is this a small ticket or a large ticket? What's the general scope of the ticket? Do we wanna allocate a lot of time to it or a medium amount or just a little time? And then in that area, priority from highest to lowest, right? So that, when you go to here, you can say, okay, is there a – oh, here, look. Here's a – here's a small scope, nice to have that we can pull in because we're out of – you know, all we have are these large scope tickets left to do, right? And we don't – you know, we don't have the bandwidth to do a large scope. So, let's grab a small scope.*

Figure 3 shows what the matrix looks like. Note that the matrix juxtaposes priority (first column) and scope (other column headers). The items in the matrix cells are all tickets and their associated metadata (e.g., id, summary, status, resolution, points, assignment).

Figure 3. The matrix to evaluate the priority and scope of tickets.



Labels

- Required: must_have or nice_to_have
- Required: high_priority or low_priority

high_priority — Small Scope

Key	Summary	T	Status	Resolution	Story Points	Assignee
ORANGE-26159	Elastic Load Balancer is not compliant with [redacted] Centralized Logging Policies	🟦	NEW	Unresolved		Unassigned
ORANGE-25357	Display growl notification for IE11 users	🟩	NEW	Unresolved		Unassigned
ORANGE-25593	Docker health checks	🟦	REFINEMENT - IN PROGRESS	Unresolved		Unassigned

3 issues Refresh

high_priority — Medium Scope

Key	Summary	T	Status	Resolution	Story Points	Assignee
ORANGE-21586	Administration Module Migration	🟩	NEW	Unresolved		Scrum Coral
ORANGE-25586	Deploy client code to CDN	🟦	REFINEMENT - IN PROGRESS	Unresolved		Unassigned
ORANGE-25767	Port Zest Privileged Module to [redacted]	🟩	NEW	Unresolved		Scrum Coral

3 issues Refresh

high_priority — Large Scope

Key	Summ
ORANGE-25590	[redacted] multi and r upgr.

1 issue Refresh

nice_to_have — Small Scope

Key	Summary	T	Status	Resolution	Story Points	Assignee

No issues found Refresh

nice_to_have — Medium Scope

Key	Summary	T	Status	Resolution	Story Points	Assignee

No issues found Refresh

nice_to_have — Large Scope

Key	Summary

No issues found Ref

AUTOMATING ACTIVITIES

- Participants identify an activity performed on a regular basis that is executed inefficiently (e.g., it takes more time than it should, it costs the company more than it should).
- All the participants brainstorm ideas to automate the activity, or at least part of it.
- They sometimes search for information needed to evaluate the cost benefit of automation, or to validate if the ideas proposed to automate the activity are doable.
- The outcome of the discussion is the early design of an automatic process to perform the activity or part of it.

As one example of a discussion that seeks to AUTOMATE ACTIVITIES, in T_{11} P2 shared a concern about how the team members were creating environments and deployments for testing. P2 explained that it was important that developers destroy testing environments when these are not needed anymore because all these environments cost money to the company.

***P2:** … Um, one other item I wanted to bring up on this call, is, uh, I just wanna make sure that when environments and deployments are created for testing that they're – [Crosstalk] – within a timely manner. We're finding a lot of like orphaned deployments and a whole bunch of ground ups and all kinds of stuff. Those all cost money. I think we have seven ground ups running right now. And the [inaudible] cost alone is $800.00 a month. So, we're gonna start kind of – just sorta think of ideas to kinda shake that down, like fewer ground ups, um. Like what features do we need to add to ground up to maybe have one ground up for Dev, one ground up for Prod, or – and how do we ensure that? Cause we've found some deployments that are pretty old and some old environments that have been around for a bit.*

After describing the activity, and why it is inefficient, P2 proposed to automate this cumbersome task by defining a "time to live" for testing environments so that these are auto-destroyed two weeks after being first created. Other participants chimed in to consider and argue the proposal from various angles. P3, for instance, raised the point that the approach should be integrated into the software they use to handle deployments.

***P2:** And most of those should've only lived two weeks max. So, maybe we should have like a time to live.*
***P1:** Yeah.*
***P2:** Self-destruct after two weeks.*
***P1:** Is that possible?*
***P3:** That's a great idea especially if it's integrated into [name of the software they use to handle deployments], right?*

Then, P2 looked at examples of auto-destruction timeframes set in other tools (e.g., Redis, EC2[7], RDS) to assess what would be a good default for the auto-destruction of the environments created for testing, and how they could configure such auto-destruction automatically. Notice P3 also engaged in this conversation.

***P2:** So, like this – this [inaudible] has been running since [date]. I don't it it's…these – this is, uh, R3X large, so it's probably like $200.00 a month or something, um. Some of these like, I don't know [inaudible].*
…
***P3:** Are you looking at Redis?*

[7] Amazon Elastic Compute Cloud is a part of Amazon.com's cloud-computing platform, Amazon Web Services, that allows users to rent virtual computers on which to run their own computer applications.

P2: *Yeah.*
P3: *It's just – it's – it's – that – that—and – and we might be able to improve on this. Every deployment creates a little Redis instance and you talk – you mentioned that on that – on that – is the – the page – the – the Confluence page you were documenting. You might be able to change that. We just need a little – it's just a central place for key values for – for – for information about a deployment.*
P2: *Oh, here's all the – this is what I was looking at. Why was I thinking Redis, but –*
P3: *With EC2, yeah, it's run on EC2, yeah.*
P2: *Yeah. I think it has RDS as well. I don't know. So, let's start page.*

The outcome of the discussion was an improved automatic way to handle this task, as exemplified in the following excerpt.

P3: *And that's why – that's why I love that idea of – of the – of the time to, you know, self-destruction. You go into ground up. And if there's only one ground up, it's all managed right there. First of all, you're gonna see all of the environments. Anything we – you decide you need to create and wanted to run some tests, it's automatically gonna have [inaudible] of – of like three days. And you should have to go in there and say, "Give me another day. Give me another day. Give me another day," if you want it to stick around. Otherwise, boom, it autodestructs.*
P1: *Yeah. I like that idea, too, 'cause like so far, we just have it on the release checklist, and I ask – like I as – I'll ask every few weeks or so. And I get the same [inaudible]. Yeah.*

DEVISING A SOLUTION

- At least one participant fully or near fully understands the issue to be discussed (e.g., bug, deployment problem, newly needed functionality).
- Participants develop a collective understanding of the issue by asking questions and clarifying doubts.
- Participants engage in design deliberation: they propose alternatives, raise design issues, and evaluate scenarios. These episodes in the discussion are typically long.
- The outcome of the discussion is not a final design, but design directions that need further investigation (e.g., a ticket to work on a proof of concept, the skeleton of a process, a list of requirements).

T_6 is an example of how participants work on DEVISING A SOLUTION. During its discussion, they seek to design a new performance testing strategy. P3 shared the kind and volume of the performance testing that they had already performed. P3 also explained what, in his opinion, should be an expanded scope and objective of performance testing for their system. The excerpt below illustrates how P3 kicked off this discussion.

> ***P3:*** *Yeah, so – so, I just want to – breaking it down to the three things that we've got. Or two things that we've got, and one that we've... working on. And the three times – three kinds of performance test. One, a component performance test. Uh, thanks to – to our ex-colleague [person name], uh, we have a performance component test.*
> *...*
> *Um, and so, we've got an example, and, uh, I think that – it's already running in the – it's already running in the pipeline. All we have to do is, um, write more tests. So, use that as an example, to write more tests.*
> *...*
> *Uh, the second kinds of tests that we have are sort of like load and stress tests. These have been generally ad hoc tests that we've put together. Um, and done in such a fashion as to prove to ourselves that something is, uh, something will handle the load, or perform under certain conditions, right?*
> *...*
> *Um, so, that's second level. And then, a third level would be performance regression tests in an automated sort of, um, way, with a sort of full system – full system testing.*
> *Um, so, that being said, we have A and we have B, and we've got versions of C. So, I would propose that for this baggle, that we do what we can to get our, uh, uh, version C into our release process. As long as regression tests are really, I think this an important part for our release process, where we review the performance of the system before we release it.*

After P3 provided the topic's introduction, P4 (quality control engineer) chimed in to share some concerns about it. P4 illustrated these concerns with hypothetical scenarios. The excerpt below shows one of P4's contributions.

> ***P4:*** *Well, the one thing that I have a question on is, where are we doing the testing? Um, if it's still inside of, uh, an environment sped up by Jenkins? Is it gonna be done on a headless browser? Um, you know, so, like, my only concern with, let's say, performance testing – performance testing, um, like components that – like render time, for example – is what does the data look like, right?*

> *Uh, what's being rendered? And then, what's the environment that we're doing it in? Um, you know, because let's say, for example, like our patient summary. We're expecting it so you get one second render time, right? But that's many different components, essentially, with they're all running with the data size. And then, also, it may look different from the client's perspective, because they're on a different box, too. So, I think we have to kind of keep up with if –*
> *…*

The participants also discussed what should be the goal of performance testing. P3 explained that the amount of data and traffic of each customer would impact performance, and therefore, the processing time will be different for each customer. P3 proposed that the goal of performance testing should not be a fixed processing time, but a comparison of how the system behaves before and after a new release, and that this comparison should be made with each client. They should always ensure the system's performance is never worse than before a new release. P3 used one of the test examples already developed in Selenium[8] to illustrate this idea.

> ***P3:*** *Right, right. And everybody wants to say, "Oh, I've got this thing," like it has to render in two seconds. So, therefore, write your test to prove that it renders in two seconds. What I'm trying to say is, here, we can – you can log in and click on the button, and see whether or not it's good enough. But right here, what I want to be able to do is to make sure it doesn't get worse.*
> ***P4:*** *Sure, right? And I – I think that's respectable.*
> ***P3:*** *Yeah. I mean, so, so, like the – the test I have, you know, the test that I have, that is running Selenium, and it's recording the render times, it's against fake data, right? These are not necessarily, you know, uh, not necessarily modeled after exactly what we're gonna see in production. However, it's gonna stay the same, you know? And it's not trivial, right? There are hundreds of clinical messages – clinical items for these patients, right? Not thousands, not five or ten; but some relatively constant number. So, when that renders, you know, it's gonna be a value which may – it is sort of representative of what they're gonna see, but it won't – but the important thing is going forward, does it stay the same? Do we make some changes that causes it to get worse? And I think that's a pretty – you know, pretty good goal.*
> *…*

[8] Selenium is an open source umbrella project for a range of tools and libraries aimed at supporting browser automation.

At the end of the session, P1 urged the team to summarize what they discussed and decide on which components they should begin to work with. Note the relevance of the product owner's intervention in summarizing what has been discussed. Otherwise, the discussion of low-level technical details could have continued until the end of the meeting. At the end of the meeting, P1 also created some tickets. One of them focused on building a proof of concept (PoC) for component C. The excerpt below illustrates the summary of the whole discussion.

> ***P1:*** *So, are you saying that all we have to do is take what you wrote, point it at perf prod, and say we've met our goals for this baggle? Or is there other stuff?*
> ***P3:*** *I – we just have to work out exactly – yes.*
> *…*
> ***P1:*** *Okay. So, I know we only have like 18 minutes left in this meeting, and I still feel kind of fuzzy on – on what the – like plan to be for this baggle, and I feel like, um, it's – I'm trying to listen and understand what you guys are saying, and grab the actionable pieces; and what I had so far was, like, take what you wrote, [P3], from site, and [inaudible] for prod, and then – and the next step is to figure out a way, um, like to be able to incorporate our performance test into our really – or, whatever, uh, stage of the process we – we do for coding and releasing. Um, I don't know if – is that – is that – does that summarize it? Am I missing anything?*
> *P3: Yeah. Yeah, you know, it's – perfectly summarizes it.*

GATHERING KNOWLEDGE

- A participant shares external information about a project that may influence the context in which the software is or will be deployed.
- The key internal stakeholders discuss hypothetical scenarios about the software being used in the new context.
- During these discussions they typically share prior information, sometimes from their experience and knowledge of the codebase, other times from investigating what they need to know by checking server logs, monitoring tools, or performance graphs.
- The outcome of the discussion are factors that are relevant to make furher decisions about the project.

T2, for instance, is an example of a conversation that seeks GATHERING KNOWLEDGE. P1 (product owner) shares that one of their customers is planning to onboard additional users, which would potentially double the number of calls to one of their APIs. Then, she requests help from the architects to foreshadow whether performance issues may arise due to the new workload.

> ***P1:*** *So, uh, [customer name], uh, [company name 1] owner, reached out to me today, and in the next few months they were thinking of onboarding a few more, um, of their clients, which would potentially double the number of calls, um, sending it to [API name]. Are there any concerns?*

P2 then brought up a hypothetical scenario in which the new workload may require an increase in the size of the database of the customer, which eventually may generate unusual charges in terms of computational resources. He explained to P1 that increasing the database size is not difficult, though the monthly charges paid for database services would increase.

> ***P2:*** *What if they – what if they increase the size of their database? We're already heading 90% during –*
> ***P1:*** *Oh, they are?*
> ***P3:*** *That's okay. That's not hard to increase. It's just –*
> ***P2:*** *Yeah. That's not hard.*
> ***P3:*** *It's – it's a five-minute downtime – fifteen-minute.*
> ***P2:*** *You just – gotta pay –*
> ***P1:*** *Okay.*
> *…*
> ***P4:*** *I think we should charge those guys. [Laughs] Get some of the sweet [client] money.*

The product owner became particularly concerned about it and shared a second piece of external information (someone asked her to keep track), which led the architects to explain how resources are charged to customers.

> ***P1:*** *Well, [employee name] has asked me to keep track of the cost difference, um, between, you know – We know when they went live with their workflows and, um, you know, when they – when they weren't live.*

> …
> ***P2:*** *Um, well their costs won't change because, um, we haven't changed the size of their database. So –*
> ***P1:*** *So, we're not paying for – a small portion for each – for any of the transactions that they're sending?*
> ***P3:*** *It's – that's really hard to do.*
> ***P2:*** *You can't tell. [Laughs]*
> …
> ***P2:*** *It's shared across the entire system, so everyone shares and then –*

Then, the product owner proposed a second hypothetical scenario trying to figure out a way to track the additional cost that this customer may generate in terms of resources. This scenario led the architects to engage in a bit of investigation, with P3 looking at usage graphs to check the monthly resource usage of this customer.

> ***P1:*** *Sure. But if we see an increase in the month of – like, during the month of March, then we can confidently attribute it to the workflows that [company] turned on.*
> ***P2:*** *Well, [company 2] also doing more. They're turning on CDA feeds. Um –*
> ***P3:*** *It's these little, short spikes, which is probably a vacuum getting kicked off or something.*
> …
> ***P2:*** *Yeah. And maybe we could just even out those spikes, and then we wouldn't have to increase their – their database.*
> ***P3:*** *By a spike – by a spike, I mean, like, a 15-minute spike. That's [inaudible] . Yeah. You should look at the graph. You should look at their graph. It's like – [inaudible – crosstalk].*

The eventual takeaway of the discussion was that the product owner had sufficient information to continue talking with the customer about the potential consequences of increasing their usage.

MANAGING ACCOUNTS

- The developers make or follow up on an account request.
- They discuss the request details (e.g., which team members need accounts, what role they should have, who should provide approval).
- If creating the accounts is under control of the key internal stakeholders, they create them during the meeting; otherwise, they walk participants through the process to obtain the accounts.
- The outcome of these discussions are accounts with the permissions the developers need, or detailed guidance on how to obtain them.

As an example of a discussion focused on MANAGING ACCOUNTS, in T_{13}, P5 requested the creation of additional accounts to access a monitoring service tool for the team members in India. The excerpt below illustrates.

> ***P5:*** *Uh, I have a request. Uh, we already spoke about this, uh, the Cognito*[9] *access. I think we have two accounts in the [India] office, but I feel it's not enough. Uh, yesterday we struggled a lot because people who had, uh, access, they went – I mean, a little bit quickly, and the people who were actually working, they didn't have access to an account. So, is it possible to provide at least one or two, um, access, maybe to [Name] and, uh, to [Name] or any of the people?*

The participants spent several minutes discussing which team members needed accounts, and what roles should be granted to them. P2 mentioned he could provide accounts with an operational role right away. One of the managers (P6) then inquired about the difference between this and other roles available (read only, administrator).

> ***P2:*** *So, everyone can get a access – uh, access to the production account, I guess? Or choose some people. We can give them read only access, and then we can give them operational. read only is obviously read only, but operational allows them to, um, configure Cognito.*
> ***P5:*** *Okay.*
> *…*
> ***P6:*** *[Crosstalk] Why don't you plug in the entire team then, [P2]?*
> ***P5:*** *Yeah.*
> ***P2:*** *I can request the entire team. It just has to go through –*

[9] Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps.

P6: *Let's do that. I don't want to piecemeal this then.*
P2: *You wanna give them operational or –*
…
P6: *Out – outside of the operational if, you know, people are gonna get involved in, uh, doing any kind of production support, that's the minimum required, right?*

Based on the information provided by P2, the manager decided that an operational role fulfilled what the developers needed to do and approved creating the accounts for them with that role. P2 created these accounts right after. He also shared that, given that the operational role is under the control of their team, it was the best choice because they could edit the accounts' role and permissions as needed.

P2: *Operational only allows you read only and to modify Cognito. It's our own role though. We can change that role. Um, I would be hesitant to give everyone admin because –*
P6: *Not admin. So, let's do operational, and let's figure out what we're doing based on the support needs whether we need to add more privileges to the operational role.*
P2: *Yeah, and that is controlled by us. The other two are controlled by, uh, DevOps? So, we do have wiggle room with operational to add additional roles and privileges as – as we decide, so I think that – that's reasonable.*

MANAGING COMPUTATIONAL RESOURCES

- A housekeeping task exists that the team has ignored for some time and at least one participant inquires about it.
- The participants discuss the perils of doing the task at that moment. If no major roadblocks are anticipated, they attempt to take care of the task during the meeting.
- The outcome of the discussion is the completion of a housekeeping task.

In T_{12}, the conversation centered on MANAGING COMPUTATIONAL RESOURCES. P6 brings to the discussion the revision of various unused server instances that were running, costing money to the company, and that nobody was using nor taking the time to destroy. Revisiting these servers and revising whether they were still in use was an external request from another team. P6 inquired about why this task had not been handled. Thus, P2 decided to investigate it during the meeting.

> ***P6:*** *... The email that came out from I guess [other team] ... regarding shutting down, uh, the unused instances and so from that list provided, have you done anything of shutting down the ones that we – that were not used?*
> *[nobody answered]*
> ***P2:*** *Guess that's a no. Let's see.*

P2 opened the list of servers that the external team shared with them to review the status of each server. The architects and the manager then discussed the use of specific servers and whether they should keep them. Notice P6 considered several angles of the team's internal organization in an attempt to reduce the number of servers that they used.

> ***P6:*** *So, which ground up is, uh, [Team in India] using? Are they using the [instance name1] one?*
> ***P3:*** *We've got like three or four. There's like [instance name 2] and…the [instance name 3]?*
> *[Crosstalk]*
> ***P6:*** *You need three for two teams?*
> ***P3:*** *Yes.*
> ***P3:*** *Yeah, I mean, so for example, we've got [instance name 4] and [instance name 5]. We can get rid of ground up [instance name 5]. We don't need to have two, right, [inaudible].*
> ***P6:*** *What would it take for us to, I guess, trim down to one?*

P6 then asked questions to the developers present in the meeting to clarify what server instances were indeed needed.

> ***P6:*** *So – so, I guess the question then [P8] for you guys is why do you need three environments, [instance name 2], [instance name 3], and [instance name 6]?*
> ***P8:*** *We can manage with two but we created three because we have multiple demos so we will only have [instance name 2] and [inaudible].*

Across the ten meetings, only one topic was classified into this category. MANAGING COMPUTATIONAL RESOURCES is an activity they may not often discuss during the meetings. However, given that all people who may know whether a server was still required were present at the meeting in question, the manager decided it was a good moment to inquire about it.

The discussion of this topic (T_{12}) was intertwined with another discussion (T_{11}) that focused on defining a strategy to create testing environments in an efficient way (this is discussed in Section 3.9 and shown in Table 6.

PERFORMING A POST-MORTEM

- The software architects provide an overview of an issue that was handled in the wrong way.
- The software architects explicitly identify the actions that could be improved.
- The outcome of the discussion are best practices and/or processes to avoid similar situations in future.

T_9 is an example of a discussion in which participants PERFORM A POST-MORTEM. This kind of discussion begins with somebody providing an overview of a past problem and how it was handled. In topic T_9, for instance, P2 shared that the problem was that a database upgrade script failed during a migration. They considered this kind of issue a blocker because there was no easy way to roll back to the previous version, yet the upgrade to the next version could not be completed. This situation put the team in an uncomfortable position because it could not use an automatic process to make the upgrade anymore and needed to instead execute manual SQL commands to solve the issue, which they know is risky because many unforeseen issues may arise. Later in the conversation, P3 complemented the overview of the issue by explaining what caused this problem: the script was using the same connection to open a database cursor and to do some updates, which is a bad programming practice. The excerpt below illustrates the most important parts of the overview.

> ***P1:*** *Could we also just give like a like, brief overview of what the issue is and how I guess [inaudible – crosstalk] release.*
> *P2: … So, on Friday I was upgrading everyone to the latest release, which was 255, something like that. … Um, so the – the database upgrade was failing, basically. Um, so we created this ticket.*
> *…*
> ***P3:*** *And so, the way the script was written, it said, fetch all of the roles, and give me 20 at a time. Open a cursor, and then I'm gonna do some work. But in the script, it used the same connection to open the cursor that it did to do some updates.*

Then, the architects explicitly identified the actions that could be improved when a similar situation happens in future. In the above example, P3 mentioned the bad programming practice. This participant also mentioned this bad programming practice was not new, and that it had happened with other developers in the past. P2 then urged the team to define a process to handle future instances of such misuse of the database cursor, which he believed was another important improvement they could work on, given that the issue kept recurring and may well appear in future again.

> ***P3:*** *So, this is just a normal snafu that often gets us when we open up a cursor and don't create a second connection to do work. And it just so happened that this migration script suffered from this.*
> *…*
> ***P2:*** *so the ticket was created, but it still hasn't been fixed. So, we still cannot do any further releases …– is there anything that we could have done to have it fixed by now …*

After stating what could be improved, the stakeholders defined and explained a new best practice to avoid this kind of poor programming in future, and worked on defining a process to handle blocker issues that break the build. These two points were the outcome of this discussion. The excerpt below illustrates how P3 explained the new best programming practice, and P2 proposed some ideas to define a process to handle blocker issues.

> ***P6:*** *So, do we have anything in terms of a best practice of –*

P3: *Yeah. Whenever you open a cursor, that cursor has to be dedicated, and there's no – I mean, that connection has to be dedicated to that cursor. And the biggest problem is it won't break until you're in a situation where, you know, it returns multiple pages, and then all of a sudden it explodes, and dies– in a not so obvious way. Right? It just doesn't work. I think*
…
P2: *Would it make sense to create a [inaudible] ticket next time for tracking the – the issue? Or should we just ping everyone, hey guys, we have a blocker, please fix this ASAP? What would be the messaging?*

PLANNING A FUTURE MEETING AGENDA

- Participants request to discuss a certain topic in a next meeting.
- There is a brief discussion about the most important points that should be discussed about this topic, and who should attend the meeting.
- The key internal stakeholders document this information (e.g., they create an "Ask an Architect" question or they create a wiki page with notes).
- The discussion's outcome is the formal scheduling of the topic's discussion for a next meeting.

T_{26} is an example of discussions in which participants PLAN A FUTURE MEETING AGENDA. In this topic, P6 requested to schedule the discussion of Single Sign-On (SSO) integration into their solutions for a next meeting. Given this was a new goal of the company, P6 desired to clarify the expectations for the feature, the business rules, an architectural plan, and the toolset they could use to implement it. The excerpt below illustrates.

P6: *So, can – for the next meeting, can we talk about the whole SSO? Because, uh, there's like a – a lot of questions going with [person name]. And I'm not sure if, uh, we need to talk to [person name] or we need to decide first as – from an SSO perspective. It's, um, you know, SSO client, non-SSO client. Within SSO, how do we secure APIs versus UI? And then what do we wanna use at the tool set? To me, it seems like these should just be options from a configuration perspective. And depending on what the client likes, we should be able to configure and move on. So, I just wanted to have further discussion on that, and we need to come up with what is that architectural plan or patterns, and that's what the system will support, right?*
…

At the end of the discussion that then unfolded, P2 added the question to the "Ask an Architect" dashboard they will use for the next RSSMDM, with the team deciding to call the topic "SSO next steps". Then, the participants discussed important points that should be addressed in this discussion. P2 actively documented these points in "Ask and Architect". The excerpt below illustrates.

> ***P3:*** *Okay, so should we create a topic specifically around – what would we call this? SSO, uh – SSO…*
> *…*
> ***P3:*** *SSO next steps.*
> ***P2:*** *We can just – let's just make it an "Ask an Architect" question first just…*
> *…*
> ***P2:*** *– I think part of the confusion is that, um, we only support some single sign-ons. So, Okta and – what's the other one? – Auth0 fully.*
> ***P3:*** *I think – I think that's part of – I think part of what we need to discuss, right? – is what – what are the aspects that are preventing us from being more generic and require – do you know what I mean? I – I – I think – I think a lot of it is around trying to support basic auth, which I'd rather not do anyway.*
> ***P4:*** *Right.*
> *…*
> ***P2:*** *Yeah, I think that's the – that's the main rub is that, you know, if you are a user, you're fine. You can use any single sign-on. But if you wanna use basic auth, which is required for the API, then you need custom code. So, then we're writing custom code for every single sign-on that exists.*

Across the ten meetings, only one topic was classified into this category.

PLANNING HOW TO TRIAGE TICKETS
- A preliminary discussion in which participants briefly look at large backlogs to figure out how to start a triaging session.
- The participants propose high level groups to classify tickets, define the goal of each group, and more.
- The outcome of the discussion is the articulation of the goals for a future ticket triage session, and a strategy to work on a subset of the tickets.

As an example of the team PLANNING HOW TO TRIAGE TICKETS, prior to beginning a ticket triage session, P1 takes a moment to discuss high-level aspects of the tickets they are about to triage (T_{45}). Participants exchanged ideas about how and where to start. The excerpt below illustrates.

> ***P3:*** *Yeah, sounds good. How do we want – how do we want to do that prioritization – Do we want to have a manual, like [P12] suggested, the top 20? Start going through the list and see what makes it, makes the cut.*
> ***P12:*** *I'll make the list tomorrow. Maybe today you just go over the themes and just tell like what do you want from our prospective. You can say I want an alert, any services are stretched to its maximum limit, something like that. An alert based on, ah, um, something like Redis disc usage. You know, that could be a lower priority than the alert that you wanted to prioritize. You know, this is a lengthy list. Like more than – I think more than 100. It's definitely more than 100 items.*
> ***P1:*** *You did mention when you first introducing this list that you think we should tackle the monitoring section first. Is that correct? Like if we were going to pick one of these sections, you –*
> ***P12:*** *Yeah. I really – these are easier problems to solve. And some of them are really important to solve, as well. They do not require much coding. We just already have the data available somewhere. You just need to put it in a different place where someone can access it or alert them based on something.*

Note the team does not yet have an efficient way to go over it. P1 suggests starting with what P12 believed should be prioritized. After listening to the information that P12 shared, P1 defined the goal of the first group of tickets, and a strategy to go about them.

> ***P1:*** *Okay, so how about we just take maybe our goal for [baggle name] baggle is to improve our monitoring, mass produce tickets here, and, um, and then, ah, in addition to some of our other PTR work. So, maybe we can, in this meeting, like, as a group, go through this list and ask questions, discuss what each ticket might entail. And then pull like the top X number of tickets from this and put it in our infrastructure epic.*

Across the ten meetings, only one topic was classified into this category. This planning typically occurs prior to a ticket triage session that might require more than one meeting to be concluded. Note that even though coordination did not take the whole meeting, it did take some time.

REVIEWING A DESIGN PROPOSAL

- One or more participants share an initial design document.
- The rest of the team shares concerns about the design, which sometimes are evaluated through hypothetical scenarios.
- The outcome of the discussion are suggestions to refine the design.

In T8, for instance, P5 shared a design for some new functionality that had been requested. P5 did so by sharing the requirements and some technical concerns about what was possible in terms of the components already existing, and that must be modified.

> ***P5:*** *… We'll start with the results model story where the expectation is that currently, we do not support CD mapping. So, we have to support CD, um, messages for results, and also, um, current mapping for pathology is incorrect.*
> *…*
> *So, before we take a decision, I want to check whether – because it will involve separate privileges. And maybe, I just wanted to ask this group, will it impact anyone if we go with the separate, uh, placeholder here for re – results?*

In addition to providing an overview of the proposed design, P5 shared specific concerns about the design, some related to the design proposing to use separate privileges. The excerpt below shows the answers that some of the key internal stakeholders provided.

> ***P2:*** *You can use the, um – you don't need new privileges or anything. You can use the same API and add a new section. For example –*
> ***P5:*** *Mm-hmm.*
> ***P5:*** *– [module name 1] and [module name 2]. I believe all – and search settings, those all use the same API. They all use the same privilege.*
> ***P4:*** *Mm-hmm, yeah.*

Note that P2 (an architect) clarified that separate privileges are not needed and pointed out modules in which a similar implementation was followed. P4 (quality control engineer) confirmed the information that P2 shared.

The outcome of this discussion were multiple notes to refine the design before implementing it. One of them, for instance, stemmed from the information that P2 shared about the privileges and the code snippets the developers from India could look at to follow the same approach to handle privileges.

SHARING INFORMATION ABOUT FUTURE PROJECTS

- The team shares external information about future projects.
- Sometimes they make formal announcements about activities that will happen in other meetings. Other times, they share information informally, to create awareness about important milestones coming up.
- This discussion does not have a particular outcome; it is merely informative. However, participants may foreshadow activities in which they could assist.

Two times the topics discussed centered on SHARING INFORMATION ABOUT FUTURE PROJECTS. In T_{14}, for instance, P2 shared an update about a new project in which the team was expected to participate. This information came from an external meeting that only P2 attended. However, it was important to share the information with the rest of the team so that they could prepare for it.

> ***P2:*** *Um, cool. Um, [Confidential] update, I just got this today. [person name] is always very bad at planning ahead, I guess. I have a meeting tomorrow at 10:00 to go over next steps. Um, it looks like we'll be doing planning, essentially. Um, obviously it's up to the team to come up with shard sizing and endpoints and all that stuff, so what we'll probably start doing is creating Jira tickets for what we had planned and getting those together. Um, so, we'll see.*
> *Sounds like they want to plan a full year, which isn't really how we do things, so it is what it is. We'll create a backlog, and we'll hope that we can finish all of it and, you know, see how it goes. So –*

In the below, note how P7 chimed in to clarify whether they already had part of the work done, and how P3 offered his support in handling some of the activities. Even though some coordination takes place, the main purpose of this conversation was to share the status of the project with the team.

> ***P7:*** *You – you do have, um, high level shard sizes, right? From the last meeting?*

***P2:** Uh, those were, like efforts – I have, like, um, amount of effort, so we didn't – I don't know if I can translate effort to shard sizes, but I don't need – I don't think we need the sizing tomorrow. It sounds like we're gonna talk about sizing, so I'm hoping to have some time to create the Jiras and then maybe get the team's input on some of that stuff, so that's my plan anyways. We'll see what [person name] says.*
***P3:** Let me know. We can – we can also work on getting all of those on the PTR.*
***P2:** Yeah. Probably need to do that.*

TRIAGING A TICKET

- A ticket with some information exists, but it needs to be refined, prioritized, and possibly assigned.
- The participants quickly recap the ticket's history (e.g., what it is about, why it was created, to whom it was assigned).
- A brief design discussion takes place to define the goal and scope of the tickets. The goal of the discussion is to define the business value of the ticket, so that they can decide if it should be implemented soon.
- The participants update the ticket's information in parallel (e.g., business value, goal, description, the title).
- The outcome of this discussion is the ticket's refinement, and potential scheduling.

An example of a discussion centered on TRIAGING A TICKET is topic T_{37}. The participants briefly discussed a ticket to build a dashboard with application metrics during this discussion. This discussion began with P12 recapping the ticket's background. While doing so, he shared the screen with the ticket's information in Jira as backdrop. P1, the product owner, quickly reminded the participants of the goal of the conversation: defining the ticket's priority.

***P12:** Something that has been discussed before, if you wanted to track our applications metrics so that we can take better direction on how to shape them, I think there's enough information here. It's probably assigned to [person name], I think, yeah, it's assigned to [person name] and then they ask me for some information and then I give them.*
P1: Okay. So, does the group agree that this is still a must have? I'll wait – I'll wait for that answer.
…

After reviewing the tickets' information, the architects raised some questions about the ticket's goal. P2, for instance, pointed out that focusing development on detecting bad code would have more value for the development team. P3, on the other hand, was more concerned about implementation details. For example, he wanted to know which technology they could use to build the dashboard.

> ***P2:*** *What do, um – I guess, what does that afford us? What does it – allow us to kill unhealthy containers? Is that the main rev?*
> ***P12:*** *Yup. That's it.*
> ***P2:*** *Or does it also, um – is that the main outcome? Or do we also get more insight into when we deploy bad code? I'd be more interested in the bad code, so we know, you know, this release is bad, pull it back.*
> ***P12****: Right – Right now, this is just some counters and then I guess timers, like within how many 99 percentile completed. And then node .js event flag to detect anything code.*
> ***P3:*** *Are we going – Do these get – Where do these – Are we going to record them to Datadog*[10]*?*
> ***P12:*** *Right now – this ticket is for collecting only. Then we have the option of either sending to Datadog or I'm proposing using something called Prometheus*[11]*. That will be part two or step two.*
> *…*

The design conversations held in the context of ticket triage discussions are typically short. Overall, design is discussed at a high level, and details are only addressed when critical to define the ticket's priority. Once the ticket's priority has been decided, the participants may also assign the ticket to a sprint or baggle. The product owner typically makes these decisions based on the input that the architects provide. All this information is captured in the issue tracking system (Jira in this case), which is continuously shared on-screen during the session so that other participants can observe and correct what the person driving the tool was capturing or updating. The excerpt below illustrates.

[10] Datadog is a monitoring and analytics tool for information technology (IT) and DevOps teams that can be used to determine performance metrics as well as event monitoring for infrastructure and cloud services.
[11] Prometheus is a free software application used for event monitoring and alerting.

P3: *I mean, I like it. I like, you know – I think – I think it's – It will be very valuable. Um, I'm not sure if it is a top priority for this coming baggle.*
P12: I feel that.
…
P1: *Can we, if that's the case let's remove from must have and high priority labels. Um, and then maybe also that [person name] ticket, so that they don't, ah, yes. We pick a must have ticket over this one. Cool. Thank you.*

3.7 The relationship between the kind of work and the purpose of the discussion

As explained before, the kind of work is a high-level categorization that classifies the work that the participants addressed (see Section 3.5). The purpose of the discussion (see Section 3.6), on the other hand, is orthogonal, and typically fits with more than one kind of work. Table 5 shows the relation between the kinds of work addressed and the purpose of

Table 5. Relationship between the kind of work done and the purpose of each discussion.

	ADAPTATIVE (7/9)			CORRECTIVE (4/7)	PERFECTIVE (17/22)		N/A (17/28)			
Purpose of the discussion	ARCHITECTURAL RE-DESIGN	INTEGRATION OF NEW FUNCTIONALITY	PERFORMANCE	SUPPORT CASE	INFRASTRUCTURE	TESTING	ADMINISTRATIVE TASK	COORDINATION AND PLANNING	PRACTICE AND PROCESS	Total
ASSESSING A PROBLEM		1		3		1				5
AUTOMATING ACTIVITIES									2	2
CLARIFYING A MISUNDERSTANDING				1		1				2
DEFINING INTERNAL PRACTICES								1	3	4
DEVISING A SOLUTION	1					1				2
GATHERING KNOWLEDGE		1	1							2
MANAGING ACCOUNTS							3			3
MANAGING COMPUTATIONAL RESOURCES							1			1
PERFORMING A POST-MORTEM									3	3
PLANNING A FUTURE MEETING AGENDA								1		1
PLANNING HOW TO TRIAGE TICKETS								1		1
REVIEWING A DESIGN PROPOSAL		3				1				4
SHARING INFORMATION ABOUT FUTURE PROJECTS								2		2
TRIAGING A TICKET					13					13
Total	1	5	1	4	13	4	4	5	8	45

each discussion. The value in each cell represents the number of topics respectively classified into that respective combination of work and discussion purpose.

In terms of maintenance work (ADAPTATIVE, CORRECTIVE, PERFECTIVE), the participants held discussions in which six different purposes came to the foreground: ASSESSING A PROBLEM (4 topics), CLARIFYING A MISUNDERSTANDING (2 topics), DEVISING A SOLUTION (2 topics), GATHERING KNOWLEDGE (2 topics), REVIEWING A DESIGN PROPOSAL (4 topics), TRIAGING A TICKET (13 topics).

The team most often engaged in ASSESSING A PROBLEM (T_{15}, T_{18}, T_{40}) when issues with deployed instances of the software arose (SUPPORT CASES). ASSESSING A PROBLEM was also the goal of one of the discussions when the participants worked on INTEGRATING A NEW FUNCTIONALITY (T_{42}) or on creating the tests cases (TESTING) required for a given implementation (T_{41}).

Concerning shaping the strategy of upcoming projects, the participants held discussions with the purpose of GATHERING KNOWLEDGE. Two examples are T_4 (the client wishing to create their own IdP solution, with the product owner needing to talk to the client about not doing this but lacking technical knowledge that the team then provides to her so she can effectively engage) and T_2 (the company's customer wishing to onboard new users, with the product owner concerned about the computational resources that the new workload may generate). Somewhat unexpected, another discussion whose purpose also relates to strategy is TRIAGING A TICKET ($T_{27\text{-}38}$, T_{44}). The key internal stakeholders found these discussions particularly useful in re-defining the goal and scope of improvements they have had in their backlog for a long time, and that they desire to get done in an upcoming release. As one example, the excerpt below shows the discussion of T_{29}. P12 first reminded the architects that this ticket was originally created to develop a dashboard with monitoring information from EC2 for which they had the backend (information extraction) already developed. At present, they typically review the monitoring information that the routine extracts via a console. The discussion does not explicitly mention the word "dashboard", but the ticket shared on screen did show it. P12 also shared that having this dashboard at first was important to monitor overloads that were occurring very often, but since they added a cache to the application, the frequency of overloading issues decreased considerably. Thereby, a dashboard to look at this information was not urgent anymore.

> ***P3:*** *What about the monitoring for [Confidential]?*
> ***P2:*** *I think the most critical parts of our system.*
> ***P12:*** *So, the thing is we already have the information [audio cuts out] EC2, providing this console. But nobody looks at it. Usually, I look at it sometimes when I suspect that the errors are because of [inaudible]. But since we added cache, we have not seen this problem that much. So, that has been saving us.*
> *P3: That cache was, ah, very good. Um, you know, this is such a critical part of our system, I can't see us doing away with it anytime soon.*
> *…*
> ***P3:*** *Yeah, the load balancer is just throwing – all the errors on the load balancer, that was being caused by overloading it. Can we alert on it? Can we make this an alert? [Inaudible – crosstalk] Let's make this a must-have. Yeah. Must have small. And then – and then say –*
> ***P2:*** *Alert, as well. Not just – We'll never look at it if there's not an alert. There's too many dashboards.*
> ***P3:****Yeah, and we get so much. Want to change the – the title to, you know, alert for [Confidential] errors, something like that?*

P3 then concluded that the problem with monitoring in regards to the current context was that an alert to warn them when overloading occurs does not exist. P2 and P12 agreed with re-defining the goal of the ticket to address that. P12 then re-wrote the ticket's goal (and actually also changed the ticket's title) to configure an alert that detects and warns them when an overload occurs, instead of developing a dashboard to display monitoring information, which would be more complex and would take more time to develop.

Some of these maintenance discussions (e.g., DEVISING A SOLUTION, REVIEWING A DESIGN PROPOSAL) align with the concept of design work because participants incur in design deliberation more than anything else. For instance, when DEVISING A SOLUTION, participants engaged in proposing alternatives for a significant architectural change (T_3) or in discussing what kinds of tests should form the core of a renewed test pipeline (T_6). The excerpt below shows a discussion that involves a design reflection toward DEVISING A SOLUTION (T_6).

> ***P3:*** *… the test I that I have, that is running Selenium, and it's recording the render times, it's against fake data, right? These are not necessarily, you know, uh, not necessarily modeled after exactly what we're gonna see in production.*
> *…*
> *There are hundreds of clinical messages … clinical items for these patients, right? Not thousands, not five or ten; but some relatively constant number. So, when that renders, you know, it's gonna be a value which may … it sort of representative of what they're gonna see, but it won't … but the important thing is going forward, does it stay the same? Do we make some change that causes it to get worse? And I think that's pretty good goal.*
> ***P4:*** *Yeah, exactly.*

Note that P3 relies on a test example he has already coded to explain that testing should not address specific conditions at the client sites, but instead it should focus on a general performance target. P3 believes that their performance metrics should compare the previous performance state of the software (before updates have been introduced to the codebase) against the state obtained with the new release (once updates have been made to the codebase). Note that P4 (quality control engineer) agreed.

Discussions in which the participants engaged in REVIEWING A DESIGN PROPOSAL, for instance, centered on reviewing the design of a performance test for certain functionality (T_7) or focused on providing feedback about the design of a new feature (T_8).

The excerpt below, extracted from T8, illustrates how P2 provides feedback to P5 on a specific part of a design proposal that she presented.

> ***P2:*** *… You don't need new privileges or anything. You can use the same API and add a new section. For example …*
> ***P5:*** *Mm-hmm.*
> ***P2:*** *[module name] and [module name]. I believe all … and search settings, those all use the same API. They all use the same privilege.*

P2 also highlighted some coding examples they could check to code the privileges of this feature in the same way.

During discussions in which participants engaged in DEVISING A SOLUTION or REVIEWING A DESIGN PROPOSAL, the kind of work addressed typically covered an ARCHITECTURAL RE-DESIGN, INTEGRATING A NEW FUNCTIONALITY, or work related to TESTING.

For other discussions, design was also addressed, but at a smaller scale. For instance, while CLARIFYING A MISUNDERSTANDING, short design conversations were conducted to explain how the system works internally or how it could be configured. The excerpt below illustrates how as part of clarifying what kind of tests the team should apply after a recent update to the testing pipeline (T39), P3 mentioned all the kinds of testing they had. Note P3 did not go into details about it but did mention what components the developers could use.

> ***P3:*** *think there's just a – just a, you know, are – um, we gotta make sure that all the other tests – all the other tests are understood and written. If we take – you know, if we're not paying attention to we've got API testing, right. We've got Karma testing. We've got integration testing on the front end, right. So, uh, we could pull up [P4]'s document about it, right. It basically, everything else covers everything that should be in end-to-end.*

Moreover, when the purpose was GATHERING KNOWLEDGE for upcoming projects, participants walked through hypothetical design scenarios about how the system would behave under different circumstances. For instance, during T_2, P2 led the team to analyze this scenario:

> ***P2:*** *What if they – what if they increase the size of their database? We're already heading 90%.*

As another example, while TRIAGING A TICKET, tiny design discussions were conducted to summarize the background of issues and to re-define the goal and scope of the ticket in question. During the discussion of T_{38}, for instance, P12 explained the background of the ticket to triage, and the possible causes to investigate from a design perspective:

> ***P12:*** *Yeah, so, it's – So, when customers say that, you know, either [Conficential] queues are backed up, that means that his transaction [audio drops out] or they will say there is a general slowness in the UI. So, there are two things. First this is business transaction. Second thing is in either the Elasticsearch [inaudible] is busy or the entire search query system is really [audio drops out].*

In RSSMDMs, spending most of the meeting's time in design conversations is expected. For most topics (28 out of 45), the key internal stakeholders did lead the participants to spend a non-trivial portion of the meetings' time discussing maintenance design work (at different scales). However, discussions not related to design but to COORDINATION AND PLANNING, PRACTICE AND PROCESSES, and ADMINISTRATIVE TASKS that the team needs to operate also took place. The purpose of these conversations was more varied. The ADMINISTRATIVE TASKs performed, for instance, centered on creating accounts (MANAGING ACCOUNTS) that the developers needed and on getting rid of environments that nobody was using (MANAGING COMPUTATIONAL RESOURCES). Regarding COORDINATION AND PLANNING, activities ranged from planning the agenda of future meetings (PLANNING A FUTURE MEETING AGENDA) to sharing the status of projects (SHARING INFORMATION ABOUT FUTURE PROJECTS) and more. Finally, in terms of PRACTICE AND PROCESS, participants engaged in the automation of technical tasks (AUTOMATING ACTIVITIES), the definition of good practices (DEFINING INTERNAL PRACTICES), and the analysis of how the team reacts to issues with deployed instances of the software (PERFORMING A POST-MORTEM).

3.8 Additional observations

Additional observations were also obtained from analyzing the topics from the two perspectives previously introduced: (1) the type of work that was addressed, and (2) the purpose that the discussions had.

Observation 3: Some types of discussions are mundane.

Not all the discussions in which the team engages are as creative or as challenging as DEVISING A SOLUTION or ASSESSING A PROBLEM. The team also holds discussions that are more mundane. This does not mean they are less important; they just are more routine or low-key in the items being considered. MANAGING ACCOUNTS and COMPUTATIONAL RESOURCES, for instance, involve tasks that are outside the normal realm of development, but necessary to keep the overall effort going.

Observation 4: Several types of discussions commonly involve some form of investigation.

A common activity across several types of discussions is to investigate some aspects of what is being considered in detail. For instance, this excerpt highlights how, during a topic discussion in which they worked to understand why the software appeared to be slowing down under certain circumstances, the team used one of the tools in the AWS toolkit to study the CPU load, live:

> ***P3:*** *Do you see that?*
> ***P1:*** *Oh, wow.*
> ***P3:*** *So, what's on there? Zero instance. I just picked on that because it's 11 – Oh. That's – that's the reader. So, look at that. The CPU is higher on the read replica right now. Then if we look on their writer – Let's look at that over the last week.*

This regularly happened for ASSESSING A PROBLEM, but similar episodes were part of PERFORMING A POST-MORTEM, MANAGING COMPUTATIONAL RESOURCES, and DEVISING A SOLUTION. In each case, it was important to the proceedings of the conversation to obtain current information about the state of a deployed instance of the software, or to examine some of its historical behavior in terms of user behaviors and associated resource use.

In addition to examining data coming from the deployed system, the team sometimes goes back into its own history. For instance, on some occasions they browsed older Confluence pages or Jira issues, and on one occasion they even reverted to the Slack channel where some key information could be found.

Observation 5: The key internal stakeholders nearly always delegate.

The participants rarely fully resolve an issue that they discuss. Instead, the result from the discussion is typically some high-level assessment, guidance, design consideration, design decision, or directive. They may, for instance, perform a first level of triaging by assigning issues to particular releases, but then the details are still left to one of the team members.

3.9 Topic recurrence

An important characteristic of the meetings studied is that they are not single but multiple-topic oriented. In every meeting, more than one topic was discussed, though the number of topics discussed varied considerably (minimum two, maximum eleven). In the first meeting, for instance, the participants discussed five topics, while for the second one they only discussed two. On average, participants discussed between four and five (4.5) topics across the ten meetings studied. Figure 4 shows the topics discussed in each meeting in order and provides an idea of how long each discussion was.

Regarding recurrence, one can observe interesting phenomena by examining this distribution. For example, observe that:

Observation 6: The participants revisit topics.

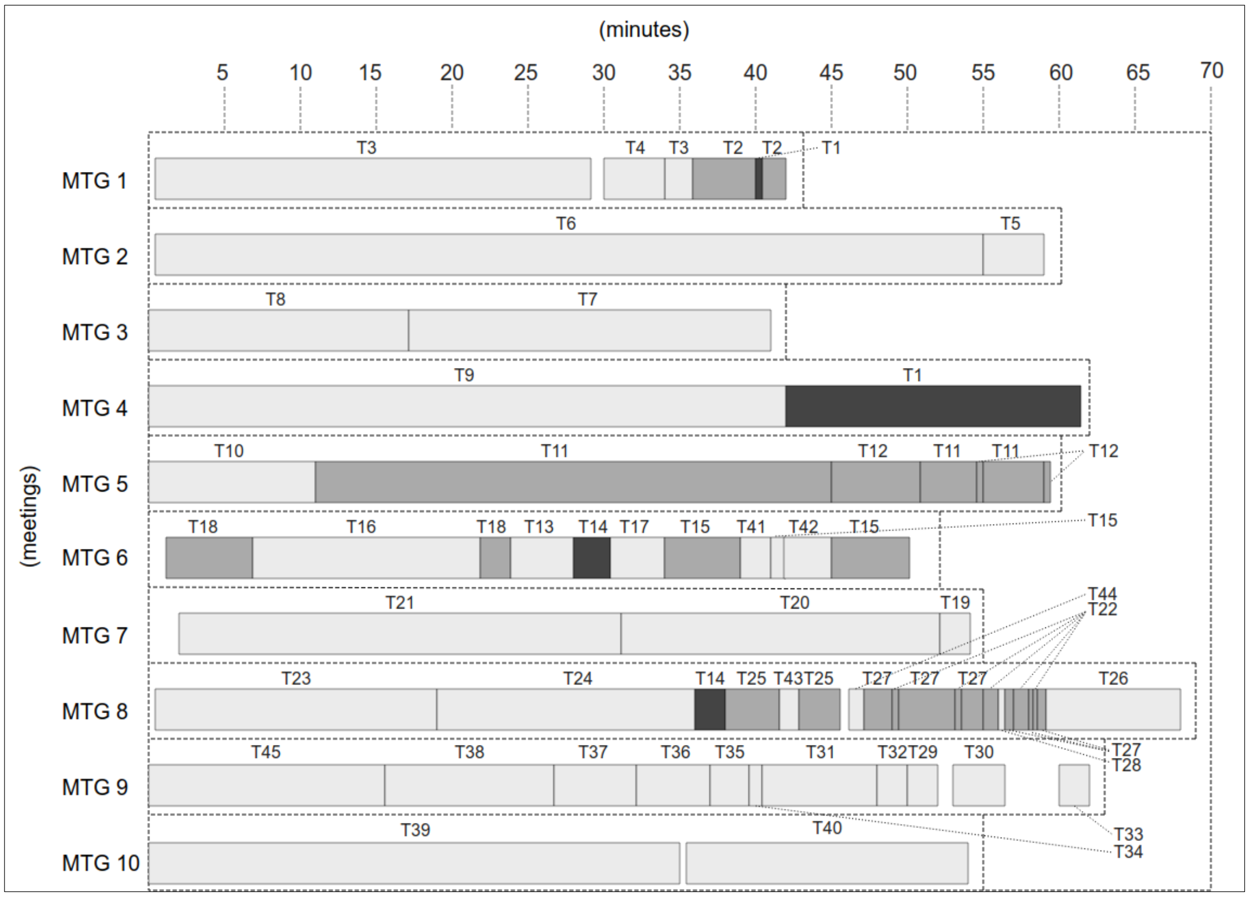
Figure 4 shows the topics that were re-discussed within the same meeting. Note that T_2 was re-discussed in the first meeting, and that topics T_{11} and T_{12} were re-discussed in the fifth meeting. Then, in the sixth and eighth meetings, topics T_{15} and T_{18}, and topics T_{22}, T_{25}, and T_{27} were respectively re-visited.

In some situations, certain aspects of one topic led participants to engage in discussing another one. For instance, in T_{15}, P5 requested to discuss a support case that was, at that time, active in the technical support Slack channel. The excerpt below illustrates the problem reported.

> ***P5:*** *I have a question for this group. Can I – uh, I'm just following, uh, the discussion, which is regarding shared, uh, functionality. [P2] has mentioned there that, um, provider never worked, so service providing, it's not possible. Actually, that will not work with provider. Maybe it is broken. It is very clear that when we worked on share feature, provider was a recipient as – as well. [Inaudible] and it worked, so I'm wondering why because we built this function so long ago, so I'm wondering when did it break? And if it is broken, how come we didn't, uh – we didn't get any notification then how did we know this ever?*

This conversation led the participants to discuss the flakiness of their end-to-end tests (T_{42}), which is supposed to be the way to prevent the issue reported in T_{15}. The discussion

Figure 4. Timeline of topics addressed per meeting. Shading from the lightest to the darkest identifies the topics that were discussed only once, rediscussed within the same meeting, and discussed in more than one meeting.



of T42 lasted about two minutes. Then, the participants returned to discussing the support case (T15). The excerpt below illustrates part of T42.

> ***P3:*** *Our full end-to-end tests are so broken that it needs to be a unit test and component test. It has to be down on the pyramid. We stack that pyramid so high that – that – that they break, and nobody can – too many are broken. It's too flakey and too complex end-to-end wise.*

Topics were also revisited after being officially concluded when new doubts about what had been said arose. This was the case for T18. Once the topic had concluded, and participants had already discussed another unrelated topic (T16), P8 raised one more question about T_{18} (i.e., *P8: "Oh, wait a moment. I have one more question regarding the [confidential] updates*"), which led the team to re-discuss certain aspects of T_{18}.

Another interesting pattern observed was that some of the topics revisited exhibited an intertwined nature, meaning the participants alternated the discussion of a pair of topics and made progress with both. Across the ten meetings, I observed intertwined topic discussions in two meetings. Topics T_{11} and T_{12} during the sixth meeting, and topics T_{22} and T_{27} during meeting eight.

In the sixth meeting, for instance, one of the architects brought as a concern that the creation of testing environments (virtual machines in which the software was deployed) was not being handled properly (T_{11}); Table 6 , timestamp [00:11:00] illustrates how this discussion began. Then, a manager brought up a question related to getting rid of various unused environments; this was an ADMINISTRATIVE TASK that IT requested the team to get done, but that they had left pending. The first fragment in timestamp [00:48:55] illustrates how a new discussion (T_{12}) began after this question was raised. P2, P6, and P3 worked on reviewing the excel sheet that lists the names of unused environments (T_{12}). They discussed these one by one and got rid of some of the unused server instances. P2 was the one initially deleting the unused instances, but at some point P3 began to help him. While performing this activity, P3, then drew the attention back to T_{11} (timestamp [00:51:01]), in which P6 also got involved by making key suggestions. Then, P3 interrupted to let P2 know he had eliminated one of the unused instances (timestamp [00:54:52]); this action was related to T_{12}. After this brief interruption, P6 continued discussing the overall strategy to create, use, and destroy environments (timestamp [00:55:00]). While the purpose of T_{11} was to define a good process to create, use, and destroy testing

Table 6. Example of two discussion intertwined.

Timestamp	Discussion about T_{11}	Discussion about T_{12}
[00:11:00]	***P2:*** *Um, one other item I wanted to bring up on this call, is, uh, I just wanna make sure that when environments and deployments are created for testing that they're – [Crosstalk] – within a timely manner. We're finding a lot of like orphaned deployments and a whole bunch of ground ups and all kinds of stuff. Those all cost money.* *…*	
[00:48:55]		***P6:*** *So – so, I had a question on costs not related to ground ups. The email that came out from I guess [confidential] regarding shutting down, uh, the unused instances and so from that list provided, have you done anything of shutting down the ones that we – that were not used?* *[moment of silence]* ***P2:*** *Guess that's a no. Let's see.* *…* ***P6:*** *So, why are we not getting rid of [instance name 4]and [instance name 5]?* ***P2:*** *I think we still use 'em as far as I –* ***P6:*** *But the reason that came up on the list is because they're unused for the last one week or something; the CPU utilization is less than one percent.* *…* ***P3***: *Dev [inaudible] just shut down, uh, delete [instance name 7] today. Now, it's got [instance name 8] on it.*
[00:51:01]	***P2:*** *So, right now, when you switch you can only switch to blue, but it'd be nice to be able to switch to any deployment that is on this list over here.* *…* ***P6:*** *So, are you saying, P2, that until we figure things out and do enhancements to ground up, we would have a minimum of two ground ups, one for – uh, three, one for each team?* ***P2:*** *Um, yeah, uh, that's …* *…* ***P6:*** *So – so – so, will we say that maybe for now we keep one for each team, and then with the target of, okay, over the next eight months, we need to figure out how we can work with just one.* *…*	
[00:54:52]		***P3:*** *Right. [inaudible] is gone, it is terminated.* ***P2:*** *Thank you.*
[00:55:00]	***P6:*** *So – so, I guess the question then P8 for you guys is why do you need three environments, [instance name 1], [instance name 2], and [instance name 6]?* *…*	

environments, T_{12} sought to complete a pending ADMINISTRATIVE TASK. Note that the domain of both topics was related, such a connection seemed enough to, in a natural way, go back and forth between both topics. A similar behavior was observed in [92].

There was only one case in which topics were not really re-discussed, but briefly interrupted. For example, the discussion of T_2 was interrupted by P2 in an attempt to start discussing a new topic (T_1). The excerpt below shows P2's attempt for starting a new topic.

P2: *Has, uh, [team in India] got their, uh, Amazon accounts yet?*
P1: *Oh, yeah. Wanted to, uh, follow up with that.*

Apparently, P2 assumed they had concluded the discussion of T_2. However, P3 ignored this request, and continued the discussion of T_2, which extended until the end of the meeting.

The participants also re-discussed topics from meeting to meeting. This form of re-discussion was much less frequent than re-discussions within the same meeting, and it actually happened only two times. One of them was T_1, the topic that P2 attempted to discuss in meeting six, but that in the end they did not because P3 had still a lot to say about the previous topic (T_2). They therefore had the discussion in a later meeting. This is not really a recurring discussion, as the full actual discussion all took place in meeting four.

The other discussion was T_{14}, the purpose of which was SHARING INFORMATION ABOUT A FUTURE PROJECT. On two occasions, P2 used the RSSMDM to broadcast information about this project. In this case, a real recurrence, though it was not reconsidering prior design decisions. Rather, it simply was further information sharing.

3.10 Summary

This chapter presents the kind of work the participants addressed and the overall purpose of the discussions held during the meetings. Across the ten meetings, a total of nine different types of work were addressed, and 14 different purposes were observed. This chapter also presents the relation between these two categorizations. While the kind of work is a high-level classification, the purpose of the discussion is orthogonal and typically fits with more than one kind of work. Though, all topics were assigned to only one type of work and overall purpose, the one that was most representative of what was done during each topic. From the analysis of these categorizations two high level observations stem: (1) all kinds of maintenance work (ADAPTATIVE, PERFECTIVE, CORRECTIVE) were addressed during the meetings, and (2) the participants also discussed other activities that were not maintenance related. Both observations provide an idea of how the meetings proceed, with participants primarily centered on addressing design but taking care of other activities that are also required to move work forward (e.g., ADMINISTRATIVE TASK, COORDINATION AND PLANNING, PRACTICE AND PROCESS). Other low-level observations were also obtained. One of the most remarkable is that participants did revisit topics within the same meeting and on a couple of occasions across meetings. Overall, topics revisited within the same meeting were often related, and participants alternated their discussion. Other times, they revisited them because someone still had concerns about it that had not been discussed.

4 Prior Information Shared during RSSMDM

Information is crucial to how the design discussions in RSSMDMs proceed. Some of this information is pre-existing to the meeting and has the potential to shape the design discussions held at the meeting as well as its outcomes. In this dissertation, I call this information prior information. The participants in an RSSMDM share and need prior information to understand problems, propose alternatives, and make design decisions. Without the right prior information, design conversations in RSSMDMs may turn ineffective; wrong assumptions could be made; or potential misunderstandings may arise. As a result, the design work may take the wrong path or time sensitive issues may need to be postponed to a later meeting.

Part of the goal of this dissertation is to characterize the variety of prior information that the participants in these meetings share. Another important part of this dissertation's goal is to describe the tools that participants use to share and/or obtain it. In Chapter 1, I introduced two primary questions that seek to describe how prior information flows into RSSMDMs (RQ 1 and RQ 2). Answering these two questions is the primary objective of this chapter. This chapter answers these two questions, together with a number of secondary questions. The answers provide a comprehensive, complementary view about how prior information flows into RSSMDMs.

RQ 1. What prior information do participants share in RSSMDMs in the course of design deliberation?

RQ 1.1 Do certain kinds of discussions require more prior information than others?

RQ 1.2 How often is prior information shared in RSSMDMs?

RQ 1.3 Is prior information shared spontaneously or upon request?

RQ 1.4 When prior information is requested, is the request answered?

RQ 1.5 Who shares the prior information?

RQ 1.6 How often is prior information shared that people not attending the meetings said?

RQ 2. What tools do participants use in support of sharing prior information in RSSMDMs?

The remainder of this chapter is organized as follows. Section 4.1 explains the methodology used to analyze the data. Section 4.2 presents the results associated to RQ 1. Sections 4.3 to 4.8, respectively present the results for RQ 1.1 to 1.6. In Section 4.9, I present the results that answer RQ 2. Finally, in Section 4.10, I discuss the implications of the findings for research, practice, and tool development.

4.1 Methodology

To answer the research questions proposed, I analyzed the transcripts and videos of ten meetings from a healthcare software development company (details about this dataset are provided in Chapter 3). The analysis was primarily performed on the transcripts, though the videos were used as a source for understanding tool use. For instance, on quite a few occasions someone on the team would share their screen to bring up the issue tracker (Jira) or present a graph with data collected by the run-time monitoring tools the team employs. Non-verbal interaction of this kind is only possible to detect by watching the videos.

As a preliminary step, I partitioned each meeting based on the topics being deliberated (I described this process in Chapter 3). Once the meeting had been partitioned into smaller topic discussions, a thematic analysis [29], [30] was performed to identify and categorize all the occasions prior information was mentioned for each topic. I describe this process below.

4.1.1 What to consider as prior information and what not

A first step in performing the thematic analysis was defining what should be considered prior information. Two researchers (one of them myself) defined a set of rules to identify prior information in the data. These rules stem from two perspectives: (1) considering as prior information statements that participants voluntarily share during the conversation (without being explicitly requested by someone), and (2) considering as prior information statements shared in response to an explicit request ("information needs" in the sense of [22]). I list the four rules below. Rules one to three stem from the first perspective; rule four stems from the second.

Rule 1. A characteristic of prior information is that information is pre-existing. It is not a random thought, an idea, or some plan for doing something. Rather, it is something that has been said before, a particular state of something, or some data that was observed (e.g., "*they did not have source queuing on*"). Prior information is most of the times verifiable, meaning that an artifact containing it exists somewhere (e.g., code, design document) or that someone (whom one could ask to verify such information) who is not present at the meeting said.

Rule 2. We generally looked for the smallest part of a spoken segment as the information; that is, information typically consists of short utterances and may well be just a small part of a full sentence or longer fragment of contribution by a participant. A single conversation turn sometimes contained multiple pieces of information. We therefore coded the precise fragments of information rather than sentences or complete turns, so that no fragments overlapped with one another.

Rule 3. We only tagged information if it was not merely a repetition of something that was said earlier, even if in slightly different words; each coded bit of information had to contribute something new.

Rule 4. We tagged for an "information need" if there was an explicit request for prior information (e.g., a question), if there was an implied request (e.g., an expressed doubt), or if someone acknowledged something was missing (e.g., *"I wish we..."*).

4.1.2 Classifying prior information

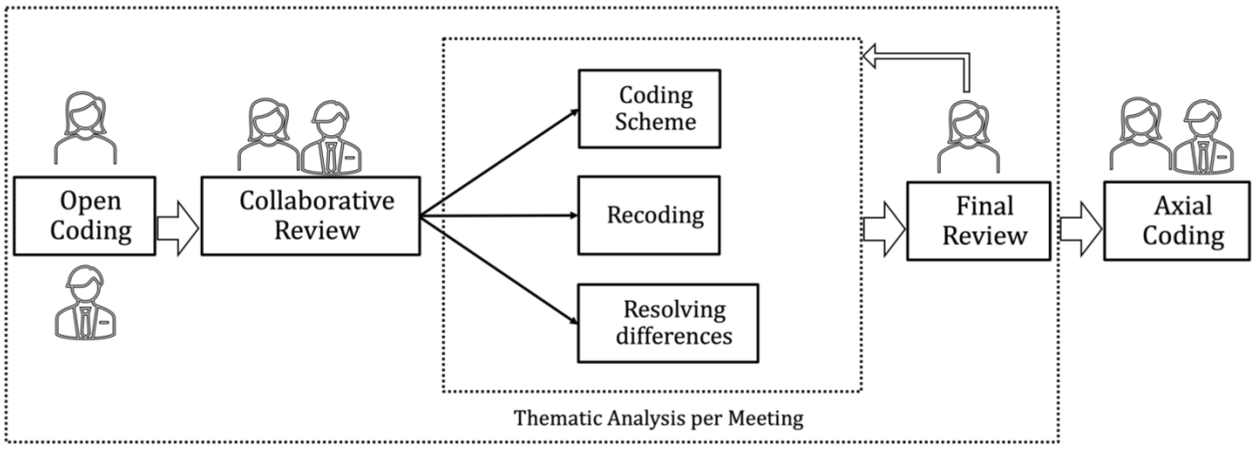
Three researchers performed an inductive thematic analysis [29] to categorize the kinds of prior information identified in the sessions. We followed the set of steps below to create a coding scheme to perform this categorization:

1. Open coding: two researchers (one of them myself) independently performed open coding on the first meeting, identifying each place where they felt information was being shared or requested in the conversation. We also independently categorized the pieces of information identified.
2. Collaborative review: we then met, compared, and discussed our respective findings, and created a first version of a coding scheme organizing the categories of information being shared in the meeting.
3. Final review: a third researcher reviewed the coding scheme and the assigned codes of the first meeting and gave feedback, which led to further changes to the coding scheme and codes assigned.

4. The two researchers who performed the coding of the first meeting (Step 1) then independently analyzed the second meeting. Thereafter, the two researchers compared and discussed the findings, which led to a refined coding scheme that the third researcher once more reviewed. The suggestions that the third researcher provided led the two researchers to several updates to the coding scheme, which they then reflected by updating the codes for the second and first meeting.
5. All other meetings were analyzed meeting-by-meeting following the process used for the second meeting, with any updates to the coding scheme reflected by recoding earlier meetings. With each new meeting, the coding scheme slowly but surely stabilized in terms of the categories it contained.
6. Axial coding: the two researchers (one of them myself) then performed axial coding, reviewing all the categories of the coding scheme one by one, examining the internal consistency of all assigned codes in each category as well as potential overlaps among categories. As a result, a few categories were merged, and several assigned codes were changed to be consistent with one another.
7. Once the ten meetings had been coded, an additional coding pass was done to detect segments that qualify as MISINFORMATION, segments in which prior information shared was corrected by the participants right after it was shared. In such cases, the two researchers (one of them myself) changed the information type that had been originally assigned to MISINFORMATION. MISINFORMATION was also added as a category to the coding scheme.

Figure 5 shows a visual representation of the process (steps one to six) that was followed.

Figure 5. Thematic analysis process.



The coding process previously described provided the basis for answering RQ 1. By juxtaposing the results obtained for RQ 1 with the topic categorizations done in Chapter 3, I then answered RQ 1.1 and RQ 1.2. To answer questions RQ 1.3, RQ 1.4, and RQ 1.5, other coding passes were performed to tag each already identified prior information fragments with several additional codes. These additional codes were distilled from what each of these questions seek to answer.

For RQ 1.3, for instance, two researchers (one of them myself) coded how each segment of prior information was shared. Specifically, this coding captures if the prior information was voluntarily offered without a particular nudge to do so (VOLUNTEERED), or if it was contributed in response to a request (ANSWERED).

The information contributed voluntarily (VOLUNTEERED) was information shared that someone else, not present at the meeting said, we identified those segments by adding an additional code, RECALL. All the other segments of this kind (VOLUNTEERED) were coded with the additional code NOT RECALL. In a similar vein, for the information contributed in response (ANSWERED) we added the additional code RECALL if the answer provided was something that someone not present in the meeting said. All other segments of this kind (ANSWERED) were coded as NOT RECALL. For all the segments coded as information ANSWERED, we also coded the statements requesting the information as REQUESTED. Table 7 shows the description of all these codes.

For RQ 1.4, I analyzed the transcripts once more to tag any parts of the conversation where participants requested prior information and a response was not issued. I added an additional code (NOT ANSWERED) to the segments tagged as requests (REQUESTED) for which a corresponding response was not provided (segments tagged as ANSWERED). During this coding pass I also noticed occasions in which requests of a certain type were made, and prior information of a different type was provided in response. In this case, I applied a different additional code (DIF TYPE) to the segments tagged as requests (REQUESTED).

Finally, to answer RQ 1.5, two researchers (one of them myself) coded the ROLE of the participants who shared, requested, or answered information requests. This involved the straightforward mapping of the speaker, which was information that was provided by

Table 7. The prompt of prior information.

Prompt	Description	RECALL	NOT RECALL
VOLUNTEERED	Information that is shared spontaneously as part of the meeting discussion; someone voluntarily refers to some information that they deem is important for the design discussion.	The participant explicitly mentions that the information shared is something someone else said (i.e., *"[person name] wrote in bold, [component name] will not connect to public internet."*).	The participant does not mention that the information shared is something someone else said.
REQUESTED	Information is requested if there is an explicit ask for it (e.g., a question), if there is an implied request (e.g., a meeting participant expresses a doubt about something), or if someone acknowledges something is missing from the conversation (e.g., "I wish we…").	NA	NA
ANSWERED	Information contributed in response to a request.	The participant explicitly mentions that the information contributed in response is something someone else said (i.e., *"I think he said it was reconnecting now"*).	The participant does not mention that the information contributed in response is something someone else said

the transcription service we used, to their ROLE. Table 8 shows the various roles of the participants in the RSSMDMs.

Additional coding was not required to answer RQ 1.6. Instead, an analysis of the segments that had been previously coded as RECALL and NOT RECALLED was performed.

To answer RQ 2, two researchers (one of them myself) performed an additional coding pass to classify the SOURCE of each prior information segment. This coding captures if the information contributed originated from the mind of a participant (PERSON) or if it was actually visible on the screen in the tool that they were using at the time (TOOL).

All qualitative coding (when it applied) was performed in MAXQDA [198], with 3368 codes assigned to prior information segments in order to answer RQ 1, RQ 1.3, RQ 1.5, and RQ 2. In addition, 694 codes were added to segments labeled as either VOLUNTEERED or ANSWERED to define whether the information shared was something someone else said (RECALL) or not (NOT RECALL), and 53 codes were added to some of the segments tagged as REQUESTED to answer RQ 1.4.

For confidentiality reasons, the healthcare software development company does not allow me to share the videos or transcripts; I do, however, have permission to share anonymized extracts from the discussions in this dissertation.

Table 8. The roles of participants who requested and shared prior information.

Role	Description
SOFTWARE ARCHITECT	A team member who is the primary person responsible for the high-level technical design choices and who enforces standards, including software coding and design standards.
DEVELOPER	A programmer on the team; generally they contribute code, test cases, and locally build and test the changes they make.
INFRASTRUCTURE ENGINEER	A team member who is responsible for choosing, adopting, and interfacing with key infrastructure that the software being developed uses (e.g., AWS, deployment tools, monitoring tools).
MANAGER	Role responsible for overseeing and coordinating the work of a number of developers on the team (also called a team lead).
PRODUCT OWNER	Carries overall responsibility for the project and its success; contributes particularly a business, finance, and customer-oriented perspective to the discussion.
QUALITY ASSURANCE ENGINEER	Responsible for quality assurance processes to ensure that the software adheres to appropriate standards before it is released.

4.2 What kinds of information are shared?

In this section, I present the results of the analysis conducted to answer RQ 1: what prior information do participants share in RSSMDMs in the course of design deliberation? Table 9 shows the final version of the coding scheme that emerged in classifying the kinds of prior information that participants shared across the ten meetings. A total of 36 different categories were observed (left column). The respective description of each category appears on the right column.

Table 9. The kinds of prior information observed (in alphabetical order)

Category	Description
ANALOGOUS SOLUTION	An example of how some programming/design problem was overcome elsewhere in some other system or systems, or in some other part of the system under development
ARCHITECTURAL FACT	A statement about how the system is designed and/or operates at a high level
ARCHITECTURAL QUALITY ASSESSMENT	An informed assessment about a high-level design aspect of the software
ARGUMENT	The reasoning for why a certain decision was made
BEST PRACTICE	A way of doing things in code, deployment, or development process that is commonly understood to be a good approach
CHANGE DIFFICULTY	An assessment of how complex/effort intensive it may be to perform a change that the team identified as needed
CODE FACT	A statement about how the system is designed and/or operates at a low level; the fact concerns a specific aspect of the implementation that one should (hypothetically) be able to trace to a specific location in the source code
CODE QUALITY ASSESSMENT	An informed assessment about an internal quality aspect of the source code
CUSTOMER CONTEXT	A fact characterizing some aspect of a customer or set of customers that helps in understanding the business side of the software
CUSTOMER COST	A fact about what charges are levied to a customer by the development organization
DEPLOYMENT FACT	A fact describing a concrete aspect of the static state of a deployed instance of the software
DEPLOYMENT MANAGEMENT	Information that may influence the actions that the team has undertaken or might still need to undertake to perform some work on/updates to the deployed software
DEPLOYMENT QUALITY ASSESSMENT	An informed assessment about an externally observable quality aspect of the deployed software
DEVELOPMENT PROGRESS	An understanding of what the development team has designed and/or implemented already and what it has not yet designed and/or implemented, typically in terms of specific features of the product they are talking about
DOCUMENTATION PROGRESS	An understanding of what the development team has documented and what it has not yet documented, typically in terms of features of the product or aspects of its internal development process
DOCUMENTATION QUALITY ASSESSMENT	An informed assessment about some aspects of the documentation of the software
EXTERNAL DEVELOPMENT PROGRESS	An understanding of what a client has designed and/or implemented already and what it has not yet designed and/or implemented
FUNCTIONALITY REQUEST	A request from another person, team, or part of the organization, or from a customer, for certain new functionality
GENERAL PROGRAMMING KNOWLEDGE	An insight about programming that is not tied to the software being developed and generally known to many developers
INFRASTRUCTURE FUNCTIONALITY	An understanding of how some part of some external software works; that software could be a component included in the software stack or some general infrastructure
INFRASTRUCTURE PROGRESS	An understanding of what has been designed and/or implemented already as part of some external software; that software could be a component included in the software stack or some general infrastructure

INTERNAL COST	A fact about what charges external service providers would levy against the development organization for hosting the software and/or support services
ISSUE	An identified problem with the software in terms of its source code and/or its current operation
ISSUE DETAIL	Additional observations towards a deeper understanding of the problem being discussed
MISINFORMATION	A wrong assertion about the system functionality, the code, specific characteristics of a deployed instance of the software, the customers, etc.
NON-FUNCTIONAL REQUIREMENT	An existing statement of a non-functional goal for the software that should be met (toward, e.g., performance, usability)
PEOPLE EXPERTISE	An understanding of the expertise and/or capabilities of an individual developer or team within the development organization
PRIOR ISSUE	An issue that was raised and deliberated in the past
PRODUCT METADATA	A non-code property of the product or part thereof that specifies information about the product that may be relevant to customers
RUN-TIME INFORMATION	An observed fact about the executing software in the form of operational data points and/or specific behaviors at a specific time/under specific circumstances
TEAM HOUSEKEEPING	An understanding of non-code tasks that have been completed, are in progress, or still to be performed
TEAM PROCESS	Information that pertains to how the team works together
TESTING FACT	A statement concerning the testing of the software, the details of how the team does so, and what it reveals about the software
TESTING MANAGEMENT	Information that may influence the actions that the team has undertaken or might still need to undertake with respect to testing the software
TESTING PROGRESS	An understanding of what parts of the software have and have not been tested
TESTING QUALITY ASSESSMENT	An informed assessment about some aspects of testing the software

At a high level, the types of information that participants shared include information pertaining to system execution (e.g., DEPLOYMENT FACT, RUN-TIME INFORMATION), the state of development (e.g., FEATURE REQUEST, DEVELOPMENT PROGRESS), the code itself (e.g., ARCHITECTURAL FACT, CODE FACT), the development process (e.g., TEAM PROCESS, TESTING MANAGEMENT), clients (e.g., CUSTOMER COST, CUSTOMER CONTEXT), and more (e.g., PRODUCT METADATA, INTERNAL COST). Note that the range of information not only covered facts about the code based nor just the state of deployed instances of the software, but the participants also mentioned information about the customers, as well as their own practice. In summary, I observed that:

Observation 7: The kinds of information shared in RSSMDMs vary wildly.

Given the fact that the design work by the team takes place in the context of a system that is deployed and in use by multiple customers, it is not surprising that information sharing is so broad. The high diversity in the types of information being shared reflects the different kinds of design challenges that arise in this particular design context, in which participants must ensure that all information needed to shape the design gets considered. Otherwise, updates to the software may take the wrong path (e.g., software updates may break functionality that was already working, software updates may slow down the overall performance of the system).

The result of applying inductive thematic analysis to the 45 topics discussed (as these were introduced in Chapter 3) across the ten meetings was a mapping of the 36 categories (in Table 9) to 694 conversation fragments in which participants shared prior information (either voluntarily or in response to an information request). Table 10

Table 10. Frequency of different kinds of information shared per meeting and in all the meetings.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	All meetings	
Category	#	#	#	#	#	#	#	#	#	#	#	%
RUN-TIME INFORMATION	11	6	1	2	14	3	12	0	17	0	66	9.5%
DEVELOPMENT PROGRESS	3	4	7	1	4	4	3	5	18	4	53	7.6%
CODE FACT	9	1	8	1	1	9	5	3	8	3	48	6.9%
DEPLOYMENT MANAGEMENT	3	1	1	5	13	7	10	2	4	1	47	6.8%
DEPLOYMENT FACT	2	6	3	0	16	2	2	0	11	1	43	6.2%
ISSUE DETAIL	2	0	1	5	13	8	7	1	4	2	43	6.2%
CUSTOMER CONTEXT	9	2	8	3	0	2	0	6	4	5	39	5.6%
INFRASTRUCTURE FUNCTIONALITY	7	1	0	0	5	2	2	6	6	7	36	5.2%
TEAM HOUSEKEEPING	0	0	0	21	1	9	4	0	0	0	35	5.0%
TESTING FACT	0	5	1	4	1	1	0	0	0	15	27	3.9%
ARCHITECTURAL FACT	4	7	5	0	0	0	4	1	3	1	25	3.6%
TEAM PROCESS	0	0	0	7	2	7	3	5	0	0	24	3.5%
ISSUE	4	0	3	1	2	1	4	1	4	1	21	3.0%
ARGUMENT	0	1	2	0	5	2	0	2	2	3	17	2.4%
TESTING PROGRESS	0	11	0	0	0	1	0	0	0	5	17	2.4%
CHANGE DIFFICULTY	1	1	2	1	2	0	1	3	5	0	16	2.3%
PRIOR ISSUE	0	0	0	1	3	4	2	1	2	0	13	1.9%
DEPLOYMENT QUALITY ASSESSMENT	3	1	2	0	2	0	0	0	4	0	12	1.7%
TESTING MANAGEMENT	0	3	1	0	2	0	0	0	0	6	12	1.7%
GENERAL PROGRAMMING KNOWLEDGE	6	1	0	1	0	1	0	0	2	0	11	1.6%
INTERNAL COST	1	0	0	0	9	0	0	0	0	1	11	1.6%
PEOPLE EXPERTISE	1	0	0	3	0	1	0	1	0	5	11	1.6%
BEST PRACTICE	1	0	0	4	0	2	2	0	0	1	10	1.4%
TESTING QUALITY ASSESSMENT	0	0	0	1	0	1	0	0	0	8	10	1.4%
MISINFORMATION	0	2	2	0	0	0	1	2	1	0	8	1.2%
ANALOGOUS SOLUTION	3	0	2	0	0	0	1	1	0	0	7	1.0%
DOCUMENTATION PROGRESS	0	0	0	2	0	3	0	0	0	1	6	0.9%
CUSTOMER COST	4	0	0	0	0	0	0	0	0	1	5	0.7%
FUNCTIONALITY REQUEST	0	0	2	0	0	0	0	2	1	0	5	0.7%
CODE QUALITY ASSESSMENT	0	0	1	0	0	0	0	1	2	0	4	0.6%
DOCUMENTATION QUALITY ASSESSMENT	0	0	0	0	0	4	0	0	0	0	4	0.6%
ARCHITECTURAL QUALITY ASSESSMENT	1	1	0	0	0	0	0	0	1	0	3	0.4%
PRODUCT METADATA	1	0	0	0	0	0	0	0	0	1	2	0.3%
EXTERNAL DEVELOPMENT PROGRESS	1	0	0	0	0	0	0	0	0	0	1	0.1%
INFRASTRUCTURE PROGRESS	1	0	0	0	0	0	0	0	0	0	1	0.1%
NON-FUNCTIONAL REQUIREMENT	0	1	0	0	0	0	0	0	0	0	1	0.1%
Total	78	55	52	63	95	74	63	43	99	72	694	100.0%

presents the information identified as being shared by participants across the ten RSSMDMs studied. The table includes the number of occurrences of each type of information per meeting, together with the totals across all meetings in absolute numbers and as relative percentages. Shading identifies the top five categories in each meeting as well as in all the meetings (if multiple types of information ranked as fifth-most, I marked them all).

Even though diverse kinds of information were shared, not all were shared that often. The shaded areas in the right most column show the kinds of prior information that were shared more often across the ten meetings. Note that all these kinds of information (RUN-TIME INFORMATION, DEPLOYMENT FACT, ISSUE DETAIL, DEVELOPMENT PROGRESS, CODE FACT, DEPLOYMENT MANAGEMENT) not only have the highest overall percentages (right column of Table 10), but also appear at least one time in eight (RUN-TIME INFORMATION, DEPLOYMENT FACT), nine (ISSUE DETAIL), or even ten meetings (DEVELOPMENT PROGRESS, CODE FACT, DEPLOYMENT MANAGEMENT). These kinds of information have interesting characteristics. First, note that most of them reflect the software being in use (RUN-TIME INFORMATION, DEPLOYMENT MANAGEMENT, ISSUE DETAIL, DEPLOYMENT FACT). In the excerpt below (extracted from T_2), for instance, one of the architects reminded the team that the current load on one of the servers involved was already near its maximum (RUN-TIME INFORMATION). Sharing this information was relevant to analyze how onboarding new people (e.g., users, patients) could affect the overall performance of the system:

> ***P2:*** *What if they – what if they increase the size of their database? <u>We're already heading 90% during [confidential].</u>*

A second characteristic about the kinds of information shared most often is that these also reflect the software being under active development (DEVELOPMENT PROGRESS, CODE FACT). For instance, the excerpt below (extracted from T7) illustrates how the team shares facts about the current state of the code as they deliberated what kind of testing should be covered for certain development (the fragment that we coded as prior information shared is underlined).

> ***P7:*** *So, essentially, that means that I'm going back to the API, [confidential] API, and it is responding back to me with a list of applications. So, that is one response time.*

In this case, the developer is verbally walking through how the current code works to discuss what should be tested. What motivated this discussion is covering the typical issues their customers experience before deploying the software in development.

These two characteristics boil down to the observation below:

Observation 8: The information that is shared most often relates to the system as deployed and under development.

The top-most categories in Table 10 all offer insight into what is happening with the system at the client site or into the current state of the code and the progress in its development. Given that the software is in use by clients and actively under development, this can be expected as the team reacts to problems arising at the deployed sites and has to keep into account what functionality has and has not been completed yet.

In relation to the body of knowledge provided in Chapter 2, the kinds of information that I observed being shared are quite different from what has long been stated as important to capture for later (e.g., decisions [18], alternatives [199], [200], constraints

[201]). Interestingly, I did not see these kinds of information being shared in the meetings. Instead, the team seems to rely on what in many ways is the manifestation of past deliberations in the code. That is, rather than referring to an underlying decision or constraint, they typically refer to the current state of the code. The following ARCHITECTURAL FACT (shared during T3), for instance, is clearly the result of an important decision the team made in the past:

> ***P2:*** *<u>So, because both, um, you know, tenants or clients, whatever, share the same computer layer, um, it is possible for one client to, uh, negatively affect – affect the other</u>…*

The original decision, which likely concerned a choice of architectural style and associated cloud-based infrastructure, shows through in the ARCHITECTURAL FACT, but the decision itself is not being recounted here.

As another example, I highlight ARGUMENT, information that shares a reason behind some past action. The following is an example (from T6), with a member recalling why they had chosen a certain flow:

> ***P4:*** *<u>I remember the general idea of why we wanted it, because we wanted the – the back end to feed the front-end information, so we didn't have to worry about hard-coding stuff. So, when the backend updated something, right, the front-end got it. That was the general idea.</u>*

Even though P4 did not explicitly acknowledge it as such, one may recognize this piece of information as an example of what previous studies (e.g., [21], [190]) have named rationale because it is the reason (why) a past decision was made. Despite their purported importance in the literature, such arguments were not recalled too often (only 17 times in total).

Observation 9: Several types of information historically advocated as important for design discussions were barely shared, though they sometimes show through in other types of information.

The meeting participants did not share past decisions, design goals, or alternatives, which are all elements that have a strong presence in the design rationale literature (e.g., [89], [199], [200]). While I did see some information being shared that traditionally is considered part of rationale (PRIOR ISSUES, ARGUMENTS, ISSUES, NON-FUNCTIONAL REQUIREMENTS), this represented just 7.4%.

Other important observations distilled relate to specific information kinds rather than to a subset of them. As one example, I observed that participants rely on information about their customers (CUSTOMER CONTEXT) to foreshadow what kind of effect decisions may have once these are implemented and deployed. In the excerpt below (extracted from T_2), for instance, the product owner brought up an important point as the team was debating whether and how to scale some service component (CUSTOMER CONTEXT):

> ***P1:*** *[customer name] reached out to me today, <u>and in the next few months they were thinking of onboarding a few more, um, of their clients, which would potentially double the number of calls</u>.*

This piece of information led the team to engage in discussing the impact and scope of an issue in terms of this new information.

Observation 10: The team performs its design work by accounting for what is happening at their customers.

Information of this kind (CUSTOMER CONTEXT, CUSTOMER COST) is most of the time shared by one of the key internal stakeholders. Sharing this kind of information allows the team to plan major updates to the codebase while keeping design work centered on the customer needs.

A second category from which I distilled an interesting observation is MISINFORMATION, which represents when someone shared some information that subsequently was corrected. These kinds of cases were rare but reflect important moments in the design discussions. Consider the following extract (from T8):

> ***P2:*** *I will say that, um, <u>[confidential] did not have a UI for the lab or the mappings, so.</u> It would be something new, I guess.*
> ***P1:*** *So, would the –*
> ***P5:*** *<u>[confidential]– [confidential] has it.</u> I don't know if we are talking the same, but, um, I don't know if you're talking about these mappings. Are you talking about this?*
> ***P2:*** *Yeah, I can show my screen really quick.*
> ***P5:*** *Yeah, and even – even this mapping is there. And it is configurable there.*

During a design discussion, one of the developers asserts that some part of the system does not have a UI (first fragment underlined). P5 corrects him (second fragment underlined), points out that it does, and works with the developer to then proceed and show the UI and the views it supports to the others in the discussion. Had P5 (a product owner) not brought up that the UI exists, the team might have gone down a design path that would be superfluous (e.g., designing a UI).

Observation 11: Participants do not always take for granted all the information that is shared and do point out when someone shares wrong information.

That said we only know when the participants did point out the information shared was wrong, not when they should have but did not. Even though the results show wrong information was shared a few times, it could be that it was shared more times, but the participants did not make the respective correction. Further research is needed to know how much MISINFORMATION is shared during the meetings, and if it represents a serious problem. The results obtained in this study only show that it does happen, wrong information is shared, and sometimes it is corrected.

4.3 Information shared in relation to the work done

In this section, I present the results of the analysis conducted to answer RQ 1.1: do certain kinds of discussions require certain kinds of prior information? To answer this question, I first placed the kinds of information shared in the context of the different kinds of work done in RSSMDMs, as determined in Chapter 3 (Section 3.5). Table 11 shows the information shared per kind of work. The shading identifies the five categories that were most frequently shared for each kind of work. Note that 420 times prior information was shared while doing some form of maintenance work (ADAPTATIVE, CORRECTIVE, PERFECTIVE). This accounts for approximately the 60% of the information that was shared during the meetings. The remainder of prior information (274) was shared as part of supportive activities that do not directly address maintenance work, but that are important for the team to get work done (e.g., creating accounts that some team members urgently need, sharing announcements about the status of projects, or establishing processes to improve activities that they typically do).

During discussions that concerned future modifications to the code base, such as performing an ARCHITECTURAL RE-DESIGN, solving a SUPPORT CASE, INTEGRATING A NEW FUNCTIONALITY, or updating the software stack (INFRASTRUCTURE), descriptions of how specific parts of the code work (CODE FACTS) appear particularly important: for all these types of work CODE FACTS ranked in the top five.

Table 11. Prior information shared per type of work.

	ADAPTATIVE (128)			CORRECTIVE (64)	PERFECTIVE (228)		N/A (274)			
Category	ARCHITECTURAL RE-DESIGN	INTEGRATION OF NEW FUNCTIONALITY	PERFORMANCE	SUPPORT CASE	INFRASTRUCTURE	TESTING	ADMINISTRATIVE TASK	COORDINATION AND PLANNING	PRACTICE AND PROCESS	All types of work
RUN-TIME INFORMATION	4		7	3	17	7	2		26	66
DEVELOPMENT PROGRESS	3	9		7	17	6		2	9	53
CODE FACT	9	6		6	9	7			11	48
DEPLOYMENT MANAGEMENT	2	1	1	4	4	1	10	2	22	47
DEPLOYMENT FACT	1		1	4	11	8	3		15	43
ISSUE DETAIL	2	1		10	5				25	43
CUSTOMER CONTEXT	5	8	3	6	4	9		1	3	39
INFRASTRUCTURE FUNCTIONALITY	6	7		2	6	7		1	7	36
TEAM HOUSEKEEPING				3			27		5	35
TESTING FACT						22			5	27
ARCHITECTURAL FACT	3	2	1	2	3	9		1	4	25
TEAM PROCESS				2		1	6	4	11	24
ISSUE	3	4		2	4		1	1	6	21
ARGUMENT		1		1	2	6	1	1	5	17
TESTING PROGRESS						17				17
CHANGE DIFFICULTY	1	2			3	3		3	4	16
PRIOR ISSUE		2		1	2				8	13
DEPLOYMENT QUALITY ASSESMENT	1		2		4	3			2	12
TESTING MANAGEMENT						10			2	12
GENERAL PROGRAMMING KNOWLEDGE	4	2			2	1			2	11
INTERNAL COST	1			1					9	11
PEOPLE EXPERTISE		1		6			3	1		11
BEST PRACTICE	1					1			8	10
TESTING QUALITY ASSESMENT						9			1	10
MISINFORMATION		4		2	1				1	8
ANALOGOUS SOLUTION	3	2						1	1	7
DOCUMENTATION PROGRESS				1			2		3	6
CUSTOMER COST		1	3	1						5
FUNCTIONALITY REQUEST		3			1			1		5
CODE QUALITY ASSESSMENT		1			2			1		4
DOCUMENTATION QUALITY ASSESMENT									4	4
ARCHITECTURAL QUALITY ASSESMENT	1				1	1				3
PRODUCT METADATA		1				1				2
EXTERNAL DEVELOPMENT PROGRESS		1								1
INFRASTRUCTURE PROGRESS	1									1
NON-FUNCTIONAL REQUIREMENT						1				1
Total	51	59	18	64	98	130	55	20	199	694

A particular interesting example in this regard concerns the only topic in which the team worked on an ARCHITECTURAL RE-DESIGN (T_3). In this discussion, facts about the code were shared even more often (13 times) than facts about the architecture itself (ARCHITECTURAL FACT, one time). The excerpt below shows the only ARCHITECTURAL FACT that was shared to justify why modifying the architecture was needed:

> *P2: So, because both, um, you know, tenants or clients, whatever, share the same computer layer, um, it is possible for one client to, uh, negatively affect – affect the other...*

The participants immediately follow up this high level ARCHITECTURAL FACT with highly technical descriptions about how the code works (CODE FACTs), as exemplified in the following excerpt.

> ***P2:*** *You trying to do an HTTP post to a TCP, which I have done before.*
> *…*
> ***P3:*** *Right. So, yeah, right now, it just – it handles an execute – an evaluation, and if it evaluates to true, it turns around and tries to send. And if it's blocked, that thread is blocked until it's able to send. Right? So, if the actions were queued, that would allow evaluations to continue.*

Note how the two architects on two different occasions mentioned specific parts of the code and how each works.

When the team worked on improving their practice (PRACTICE AND PROCESS), the participants typically walked through issues that were not handled properly. This is why ISSUE DETAIL ranked second during these discussions. The excerpt below shows a conversation focused on improving their practice (T9). P2 first introduced an ISSUE that had not been handled properly.

> ***P2:*** *So, on Friday I was upgrading everyone to the latest release, which was 255, something like that. Um, and one of our environments [electronic tone] would fail while trying to – to upgrade it. Um, so the – the database upgrade was failing, basically. Um, so we created this ticket.*

Then, P3 followed up by also sharing various details about this issue (ISSUE DETAIL). This information helped others to understand the big picture of the issue in question, as well as the actions that were taken. The excerpt below illustrates a couple of ISSUE DETAILs shared by P3 (fragments highlighted):

> ***P3:*** *And so, the way the script was written, it said, fetch all of the roles, and give me 20 at a time. Open a cursor, and then I'm gonna do some – some – some work. But in the script, it used the same connection to open the cursor that it did to do some updates.* *<u>Now, this always works if you get – if there are less than 20 because that's the batch size. So, in other words, if you say, give me up to 20, and it gives – and there are only 18, then it acts normal. But as soon as you go over 20, that cursor is still open. That query is still open, and you try to make an update, ppppt, everything fails.</u>*
> *So, this is just a – a – a normal snafu that often gets us when we open up a cursor and don't create a second connection to do work.* *<u>And it just so happened that this migration script suffered from this.</u>*

In this particular kind of discussions (PRACTICE AND PROCESS), the description of issues was interwoven with information about how the deployed instance of the software in which the issue occurred behaved (RUN-TIME INFORMATION). As one example, P2 said:

> ***P2:*** *<u>I'll say that script worked for, I don't know, like 25 to 30. It just failed for one. One environment it failed. All the other ones passed.</u>* *So, it would be hard to catch.*

Sharing this piece of information makes even more clear why the issue occurred, and why it is difficult to catch during testing. RUN-TIME INFORMATION was the kind of information most often shared during discussions that addressed work related to improving their practice (PRACTICE AND PROCESS).

An important observation that stems from the two examples previously introduced is that:

> **Observation 12:** Specific kinds of information are more relevant for certain kinds of work.

Table 11 shows not only these two, but other types of information commonly shared while discussing certain kinds of work. For instance, the types of information most often shared when the team worked on the design or implementation of test cases (TESTING) were

TESTING FACTs, TESTING PROGRESS, and information about how to set up or run tests in production (TESTING MANAGEMENT). Prior information from these three categories was shared much more often (22, 17, and 10 times respectively) than prior information of any other kind. In a similar vein, when the INTEGRATION OF NEW FUNCTIONALITY was discussed, the customer needs (CUSTOMER CONTEXT) as well as components of the software that could be used to implement that new functionality (DEVELOPMENT PROGRESS) were often shared.

I also analyzed how much information was shared per discussion based on its respective purpose (as introduced in Chapter 3, Section 3.6). Table 12 shows the information shared (fourth column), how many topics were discussed (third column), and the average number of prior information that was shared (most right column) per discussion purpose. The results are ordered in descending order, from purposes with the most average number of prior information shared to purposes with the least average number of information shared. The shading shows the five discussion purposes with the highest average number of information shared. The first column shows whether a

Table 12. Prior information shared in discussions with a certain purpose (results ordered by the average number of times information was shared).

Does the discussion involve design?	Purpose of discussion	Topics	Information shared	Avg # information was shared per discussion purpose
Yes	DEVISING A SOLUTION	2	99	49.5
Yes	PERFORMING A POST-MORTEM	3	112	37.3
Yes	AUTOMATING ACTIVITIES	2	66	33.0
Yes	CLARIFYING MISUNDERSTANDINGS	2	55	27.5
Yes	REVIEWING A DESIGN PROPOSAL	4	77	19.3
No	MANAGING ACCOUNTS	3	47	15.7
Yes	GATHERING KNOWLEDGE	2	27	13.5
Yes	ASSESSING A PROBLEM	5	64	12.8
No	MANAGING COMPUTATIONAL RESOURCES	1	8	8.0
Yes	TRIAGING A TICKET	13	98	7.5
No	DEFINING INTERNAL PRACTICES	4	29	7.3
No	PLANNING A FUTURE MEETING AGENDA	1	4	4.0
No	PLANNING HOW TO TRIAGE TICKETS	1	3	3.0
No	SHARING INFORMATION ABOUT FUTURE PROJECTS	2	5	2.5
		45	694	15.4

discussion purpose entails design work or not. Chapter 3 presents the characteristics each discussion has.

On the one hand, a considerable amount of information was shared while DEVISING SOLUTIONS (49.5), PERFORMING POST-MORTEMS (37.3), AUTOMATING ACTIVITIES (33.0), CLARIFYING MISUNDERSTANDINGS (27.5), and REVIEWING DESIGN PROPOSALS (19.3). Also note that only three categories out of the top five involve design work: GATHERING KNOWLEDGE (13.5), ASSESSING PROBLEMS (12.8) and TRIAGING A TICKET (7.5). Together, all the discussion purposes that involve some kind of design work account for approximately 86% of the information that was shared.

On the other hand, in discussions the purpose of which was merely coordinating or executing administrative tasks and that did not involve design work, prior information was barely shared. MANAGING ACCOUNTS (15.7), MANAGING COMPUTATIONAL RESOURCES (8.0), DEFINING INTERNAL PRACTICES (7.3), PLANNING A FUTURE MEETING AGENDA (4.0), PLANNING HOW TO TRIAGE TICKETS (3.0), and SHARING INFORMATION ABOUT FUTURE PROJECTS (2.5). The categories previously mentioned account for only about 14 % of the information shared across the ten meetings.

Observation 13: Discussions that involve design deliberation require more prior information than discussions that do not.

While information sharing also happened during non-design-centered discussions, clearly much more information is needed to convey design ideas. For example, while DEVISING A SOLUTION to improve the performance testing suite (T_6), 48 different pieces of prior information of 18 different kinds were shared. As one example, the excerpt below shows how P3 kicked off this topic's discussion.

P3: *Yeah, so – so, I just want to – breaking it down to the three things that we've got. Or two things that we've got, and one that we've... working on. And the three times – three kinds of performance test ... we have a performance component test. And we've got an example. It works great. It's caught one defect, one regression, that we – we put in at one point.*
...
Um, and so, we've got an example, and, uh, I think that – it's already running in the – it's already running in the pipeline. All we have to do is, um, write more tests.
...
Um, that's another place where we could be doing this. Uh, the second kinds of tests that we have are sort of like load and stress tests. These have been generally ad hoc tests that we've put together.
...
Um, so, that being said, we have A and we have B, and we've got versions of C. So, I would propose that for this baggle, that we do what we can to get our, uh, uh, version C into our release process. As long as regression tests are really, I think this an important part for our release process, where we review the performance of the system before we release it.

Just during this fragment of the discussion, four different pieces of prior information were shared, all of the kind TESTING PROGRESS.

4.4 Importance of prior information sharing in RSSMDM

In this section, I present the results of the analysis conducted to answer RQ 1.2: How often is prior information shared in RSSMDMs? To answer this question, I determined the average number of times prior information was shared per minute based on the meeting's duration (in minutes) and how much information was shared per meeting. Table 13 shows these results. The first row shows the duration of each meeting, the second row the total number of times information was shared, and the third row shows the average per minute.

Table 13. Meeting duration and information shared.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	Total
Duration (min)	43	60	42	62	61	52	55	69	63	55	562
Times Information shared	78	55	52	63	95	74	63	43	99	72	694
Times information was shared per minute	1.81	0.92	1.24	1.02	1.56	1.42	1.15	0.62	1.57	1.31	1.23

Note that for meeting eight, prior information was shared approximately once every minute. For all the other meetings, prior information was shared between one or two times per minute.

I also analyzed how much information was shared per topic discussion (all topics were introduced in Chapter 3). Table 14 shows this distribution, the first column shows the Topic Id, and the second the number of times prior information was shared. Topics are

Table 14. Prior information shared per topic.

Topic Id	Times prior information was shared
T_{11}	52
T_{3}	51
T_{6}	48
T_{39}	48
T_{21}	48
T_{10}	35
T_{1}	34
T_{7}	30
T_{9}	29
T_{38}	26
T_{40}	24
T_{8}	22
T_{23}	22
T_{18}	22
T_{36}	19
T_{2}	18
T_{16}	18
T_{31}	16
T_{20}	14
T_{30}	13
T_{13}	12
T_{15}	11
T_{4}	9
T_{24}	8
T_{12}	8
T_{5}	7
T_{35}	7
T_{29}	7
T_{14}	5
T_{41}	4
T_{26}	4
T_{45}	3
T_{42}	3
T_{37}	3
T_{17}	3
T_{34}	2
T_{32}	2
T_{27}	2
T_{25}	2
T_{33}	1
T_{22}	1
T_{19}	1
T_{28}	0
T_{43}	0
T_{44}	0
Total	694

organized in descending order, from the topic in which the most prior information was shared (T_{11}), to the topic in which the least information was shared (T_{33}, T_{22}, T_{19}). Note that only for three topics (T_{28}, T_{43}, T_{44}) out of 45 prior information was not shared.

Overall, the results obtained from the two analyses indicate that:

Observation 14: Information sharing is very frequent in RSSMDMs, on the order of once or twice a minute on average.

As one example of the high frequency with which prior information sharing happens in these meetings, prior information was shared 52 times in T_{11}. The discussion of this topic was intertwined with T_{12} (as explained in Chapter 3, Section 3.9). After being discussed the first time, T_{11} was re-discussed a couple of times more. Each of the times the topic was discussed, prior information was shared. Most of the shared information related to the software as deployed (e.g., DEPLOYMENT MANAGEMENT, DEPLOYMENT FACT), and TESTING.

As a second example, I discussed T_6 in Section 4.3. During this topic, prior information was shared about 48 times. The excerpt used as an example in Section 4.3 lasted approximately four and half minutes. During this short timeframe, prior information (all of the same kind) was shared four times, meaning that almost one piece of information was shared per minute.

As another example, in T_{38} information was shared 28 times. The excerpt below shows three pieces of information (ISSUE, RUN-TIME INFORMATION, FUNCTIONALITY REQUEST) that were shared to set the context of the discussion:

P12: *Um, so, <u>the thing is this morning some of the environments were down. The machine processing was at scale and then could not – could not serve the</u>*

<u>*request in time.*</u> <u>*The [confidential] wanted to have a dashboard or something to detect these kind of problems.*</u>

In this example, a developer first shared an ISSUE for the team to discuss (“*Um, so, the thing is this morning some of the environments were down.*”) then, a fact about the state of the run-time environment (RUN-TIME INFORMATION, “*The machine processing was at scale and then could not – could not serve the request in time.*”), to then explain that the customer wants to be able to monitor and detect when this issue happens again (FUNCTIONALITY REQUEST, “*The [confidential] wanted to have a dashboard or something to detect this kind of problems.*”). Note how, in a single paragraph, one following immediately after the other, three different kinds of prior information were shared.

Overall, these results and exmaples show how critical prior information is to the way RSSMDMs are conducted. Without the prior information being shared, the participants may need to make assumptions during their design process to move design work forward. In the context of a deployed and functioning system, however, such a practice may incur a high risk to the final design. Not only may the expectations of end users not be covered, but new bugs might be introduced to the software.

4.5 Shared spontaneously or upon request?

In this section, I present the results of the analysis conducted to answer RQ 1.3: Is prior information shared spontaneously or upon request? The results presented from Table 10 to Table 14 count all the times that information was shared. However, not all information was shared in the same way. Out of 694 total shared pieces of information, 587 were coded as VOLUNTEERED, that is, the team member shared the particular piece of information out of their own volition without any prompt by another team member. The participants sharing the prior information simply included the information in the course of making a contribution to the discussion.

The remainder (107) was shared by request (coded as ANSWERED, as explained in Section 4.1): before someone shared the information with the team, another person in the meeting asked for it, whether explicitly or implicitly.

Most of the excerpts I have previously used as illustrations were VOLUNTARILY shared, nobody asked for them. However, prior information was also shared upon the request of a participant. On most occasions, the request was explicit (with someone asking for it), as in the following excerpt:

> ***P2:*** *…Need to know – know your, uh, – your Linux pretty well, and how to use Vault, and get your – how to get the – all that. Use all – we have some scripts that we use here.*
> ***P3:*** *<u>Which – which script to use?</u>*
> ***P2:*** *We all have our like, our preferences. I use, uh, I think one of [confidential]'s Vault scripts.*

The request (fragment underlined) concerned a DEPLOYMENT MANAGEMENT piece of information by P3, which was answered instantly by P2. Interestingly, the answer did not point to a single script, but to the availability of a set of scripts. The discussion continued talking about the options.

In other cases, the request was implicit in the conversation, with the following a typical example (DEPLOYMENT FACT):

> ***P3:*** *If you try to do 250 con – concurrent requests, you're gonna get 429-ed because we've got every endpoint limits any site from making, <u>was it, 100 concurrent requests, right, 100 or 150 is the default. A hundred or 150 –</u>*
> ***P4:*** *I think it's a – I think it's 100. That's ss – good.*
> ***P3:*** *– a hundred concurrent requests for any site on any endpoint.*

Note how P3 never explicitly asked the team, mentioning two potential values simply as part of their narrative (fragment underlined). P4 felt compelled to interject and answered with the actual value. P3 did not skip a beat, continuing their train of thought with the clarified limit.

Observation 15: Information is most often shared spontaneously as part of the natural unfolding of the discussion, with a small portion explicitly requested.

I am not aware of any prior literature that has looked at this balance, but hypothesize that the fact that most information is shared of the participants' own volition might be an indicator of this team being highly experienced at conducting design deliberations. The opposite might also be true: the team is poorly performing in not soliciting the information that it truly needs.

4.6 Are requests for information answered?

In this section, I present the results of the analysis done to answer RQ 1.4: when information is requested, is the request answered? Table 15 shows all types of information that were requested, together with how often these were or were not answered. The table presents the number of occurrences of information requested (RQ), the requests that were not answered (NOT AN), and the requests for which information of a different kind was provided in response (DIF TYPE). Results are sorted based on the percentage of unanswered requests (%NOT AN).

The requests that went unanswered (NOT AN) should definitively be considered INFORMATION MISSING. However, to determine whether the requests for which answers of a different kind were provided (DIF TYPE) answered the question that was asked, further investigation is needed. Thus, whether information was MISSING or not will depend on each case basis.

Table 15. Frequency of different kinds of information being requested (#RQ), the number of requests that were not answered (#NOT AN), the number of requests for which information of the same kind was provided in response (#AN), and the ones for which information of a different kind (#DIF TYPE) was provided in response. Categories not listed had 0 requests.

Category	#RQ	#NOT AN	#DIF TYPE	#AN	%NOT AN
ANALOGOUS SOLUTION	3	3	0	0	100.0%
FUNCTIONALITY REQUEST	2	2	0	0	100.0%
NON-FUNCTIONAL REQUIREMENT	2	2	0	0	100.0%
ARCHITECTURAL FACT	2	1	0	1	50.0%
CHANGE DIFFICULTY	2	1	0	1	50.0%
DOCUMENTATION PROGRESS	4	2	0	2	50.0%
INFRASTRUCTURE FUNCTIONALITY	10	5	0	5	50.0%
INTERNAL COST	2	1	0	1	50.0%
PRIOR ISSUE	4	2	0	2	50.0%
TESTING MANAGEMENT	2	1	0	1	50.0%
CODE FACT	16	6	1	9	37.5%
CUSTOMER CONTEXT	7	2	0	5	28.6%
TEAM PROCESS	7	2	0	5	28.6%
TESTING FACT	7	2	0	5	28.6%
TEAM HOUSEKEEPING	20	5	0	15	25.0%
ISSUE DETAIL	9	2	2	5	22.2%
DEPLOYMENT MANAGEMENT	9	2	0	7	22.2%
DEVELOPMENT PROGRESS	9	2	0	7	22.2%
ARGUMENT	5	1	0	4	20.0%
DEPLOYMENT FACT	10	2	0	8	20.0%
ISSUE	2	0	2	0	0.0%
CUSTOMER COST	3	0	1	2	0.0%
RUN-TIME INFORMATION	6	0	1	5	0.0%
BEST PRACTICE	2	0	0	2	0.0%
DEPLOYMENT QUALITY ASSESMENT	1	0	0	1	0.0%
PEOPLE EXPERTISE	2	0	0	2	0.0%
TOTAL	148	46	7	95	31.1%

As an example of an information request that was not answered (NOT AN), the excerpt below shows a request for a NON-FUNCTIONAL REQUIREMENT that went unanswered.

> ***P2:*** *<u>Is there an NFR for how long it should take?</u> Because I think right now, it'd probably take uh, at least a maybe like a day to process all the results and set the new type. Or should we consider doing something with the – at query time for Elasticsearch?*
> ***P5:*** *Mm-hmm.*
> ***P2:*** *<u>Um, is – is there NFR for this, or is it okay for it take</u> – maybe that will help make the decision. Um, cause currently, we do set the value in Elasticsearch and Postgres. But we could probably change that to do something at query time depending on what the NFR is.*

Indeed, P2 asked twice in the above fragment (which we coded only once, since we did not code repetitions) and later asked again. The team deliberated what a potentially good limit might be based on a few analogous situations, but never provided a definite answer as to whether or not an NFR existed. In this case, however, the team at least considered the request and discussed it, as compared to other cases in which requests were simply ignored. For instance, in the following extract (also tagged as NOT AN), P6 makes a request concerning the review process, but P3's answer does not touch that process at all and neither does the discussion afterwards.

> ***P6:*** *Isn't that part of the quarterly review, the checklist is like, hey if I'm gonna be writing scripts then it's part of this process.*
> ***P3:*** *It's – this – this is not just restricted to strips – scripts. This is like, in our entire code base, [electronic tone] where we always open cursors, and we always iterate over them, and we always open up a second connection. It's just standard best practice. Only practice, I mean, it – it –*

As an example of an information request in which information of a different kind was issued as an answer (DIF TYPE), the excerpt below illustrates how P6 made a request about information related to CUSTOMER COST, which P2 answered by providing information related to the deployment of the software (DEPLOYMENT MANAGEMENT).

> ***P6:*** *Yeah. So, does – does that mean that if clients don't wanna be in this multi-tenant environment they will pay more and be in their own VPC?*
> ***P2:*** *It just means they wouldn't have a public endpoint.*
> ***P12:*** *Yeah. Nothing changes before it gets to the application. It's just how the client will connect to it.*
> ***P6:*** *Okay. But it will still be a shared architecture still?*
> ***P12:*** *Yeah. Nothing changes before it gets to the application. It's just how the client will connect to it.*

Note that nothing about how clients would be charged, whether they are part of a multi-tenant or dedicated environment, was said. Though, the fact "*they would not have a public endpoint*" seems to implicitly deliver a message, as it draws attention to the deployment configuration rather than the cost. P12 complements the answer, which led the conversation to a different path. In this case, it is not possible to confirm whether the answer fulfilled what was asked. Note that even though P6 confirmed to understand what P2 explained, information was requested a second time.

Observation 16: In approximately a third of the cases when information is requested, participants do not produce the requested information, with the discussion proceeding without it.

Table 15 also shows the number of requests that were answered (AN). This value was obtained by subtracting the number of unanswered requests (NOT AN) from the total number of requests made (RQ). This value does not match the number of segments that we coded as ANSWERS in the transcripts (we coded 107 segments as answers). The reason is that some requests had more than one corresponding answer.

Across all meetings, 26 types of information were requested and 23 of those had at least one occasion in which one of the requests went completely unanswered (not even an answer of a different type was issued). In total, 148 information requests were made, 46 of them went completely unanswered (NOT AN), and on seven occasions, prior information contributed in response to a request was of a different type (DIF TYPE). Situations of this kind occurred only for five categories of prior information: ISSUE (twice), ISSUE DETAIL (twice), CODE FACT (once), CUSTOMER COST (once), and RUN-TIME INFORMATION (once) for a total of seven times across the ten meetings. In general, when information requests were made and an answer was indeed issued, information of the same kind was provided in response.

I also broke down the results shown in Table 15 per meeting. In so doing, I did not find any patterns; requests for each type of information appear fairly evenly spread out across the meetings, as are generally the times that requests went unanswered. An exception to this even distribution is TEAM HOUSEKEEPING. In this category, 15 requests were raised during the fourth meeting. All these requests were discussed as part of T_1. During this discussion, participants touched base on the creation of some AWS accounts that the developers from the team in India had requested days before, and that they still did not have, yet they urgently needed to work on a support case. A particular characteristic of this discussion is that work was not delegated but done during the meeting. P2 created the accounts for the participants during the discussion. They went back and forth clarifying details such as who should have access (i.e., "*Well, I think everyone has [Confidential] by now, right?*"), what roles were needed (i.e., "*Which roles do you have?*"), and more. This explains why an uncommonly high number of HOUSEKEEPING requests were made in a single meeting.

4.7 Who shares the information?

In this section, I present the results of the analysis done to answer RQ 1.5: who shares the information? Table 16 presents the count and relative percentage of which meeting participants shared information (columns "#shared" and "%shared"), requested information (columns "#requested" and "%requested"), or provided answers to the

Table 16. Sharing, requesting, and answering per role.

Role	#shared	%shared	#requested	%requested	#answered	%answered	#answered (not recall)	#answered (recall)	#contributed voluntarily	#contributed voluntarily (not recall)	#contributed voluntarily (recall)
DEVELOPER	48	6.9%	12	8.1%	17	15.9%	17	0	31	30	1
DEVOPS ENGINEER	44	6.3%	0	0.0%	10	9.3%	10	0	34	33	1
MANAGER	12	1.7%	22	15.5%	5	4.7%	5	0	7	7	0
PRODUCT OWNER	45	6.5%	17	11.5%	7	6.5%	7	0	38	32	6
QA ENGINEER	24	3.5%	6	4.1%	3	2.8%	3	0	21	21	0
SOFTWARE ARCHITECT	521	75.1%	90	61.8%	65	60.7%	63	2	456	447	9
Grand Total	694	100.0%	147	100.0%	107	100.0%	105	2	587	570	17

requested information (columns “#answered” and “%answered”). In the table, meeting participants are organized by role. The count of information shared considers the information that participants shared without a previous request (“#contributed voluntarily”) as well as the information that was provided in response to a request (“#answered”). The count of information “#requested” comes directly from the segments that we coded as such in the transcripts, and the count of information “#answered” corresponds to the sum of segments coded as “#answered (not recall)” and “#answered (recall)”. The results in the column “#contributed voluntarily” (which forms part of the information shared) were obtained from summing up segments respectively coded as “#contributed voluntarily (recall)” or “#contributed voluntarily (not recall)”. The terms “recall” and “not recall”, which respectively identify the occasions in which participants quoted things other people said (recalls), and when they did not (not recall). In Section 4.1, I provide details about this coding.

The results presented in Table 16 clearly show that the two architects were central to the meetings: they shared nearly 75% of all information, they made over 60% of the information requests, and, between the two of them, they answered nearly 60% of the information requests. Given their responsibility of providing technical oversight for the product, this is not surprising. It is also not surprising given that both are long-time employees of the company and are at almost every single meeting of the team, which has allowed them to assimilate much knowledge about the product.

Even so, other meeting participants were regularly called upon, providing key input into the various design discussions, and helping diagnose potential issues with the deployed software. Given their more specialist roles, they spoke less than the two architects and made fewer contributions from the perspective of sharing information. When they did contribute information, however, it often was important:

> ***P12:*** *Yeah, <u>they'll be running their application just like we currently do, but it will not be exposed to internet. It will be exposed only in their private database [Confidential].</u>*
> ***P6:*** *Okay. But it will still be a shared architecture still?*
> ***P12:*** *<u>Yeah. Nothing changes before it gets to the application. It's just how the client will connect to it.</u>*
> ***P2:*** *It's like going through like the private door rather than the public door at the –*
> ***P3:*** *Sort of the – a private API.*
> ***P2:*** *– club.*

P12, who is an expert on infrastructure matters, explains the design of a security configuration for an API. It was critical for others on the team to hear what he had to say about the new configuration.

The role of the two product owners was less technical when it came to the discussions. The primary product owner, who resides in the U.S., often focused on steering the discussion at a high level. The below excerpt, for instance, represents a typical way in which she engaged, setting the stage for the ensuing discussion:

> ***P1:*** *I – I think – I have a question about – hopefully, it's quick – um, about the [confidential] workflows. <u>Uh, it looks like those are pretty, like, smooth sailing, no real issues when it comes to impacting, um, other clients. Right?</u>*

She requested an update on a recent upgrade to the software and if it eliminated a long-term performance issue (DEPLOYMENT QUALITY ASSESSMENT). One of the architects then engaged in a lengthy back-and-forth with her in which she further motivated the question by highlighting that more hospitals were planned to be onboarded and that she wanted to ensure that the software would be able to handle the additional load. When the team began to discuss, the product owner let the architect lead the discussion and said very little. From the perspective of information, however, she provided essential, context setting information enabling the team to focus and function.

Observation 17: A significant amount of information sharing is done by a limited number of participants.

Two of the software architects were responsible for over 75% of the information being shared. On the one hand, this result is not completely unexpected because it is the architects' express responsibility to lead the design and development from a technical point of view. On the other hand, the concentration of knowledge in just two of the twelve team members might be an issue of concern if these team members were to leave the company or transition to another team.

4.8 Prior information from other people

In this section, I present the results of the analysis done to answer RQ 1.6: How often is prior information shared that people not attending the meetings said? The aggregation of columns "#answered (recall)" and "#contributed voluntarily (recall)" from Table 16 shows that only 19 times out of 694 participants recalled what someone not attending the meeting said. As one example of these recollections, during T_5, the main point of the discussion was clarifying a comment made by two different people over a Slack channel:

> ***P3:*** *I mean, the only – the only thing I'd – the only thing that gives me hesitation is [person name 1] wrote in bold, "[Component name] will not connect to public internet." What [person name 2] said is, "It will be secure over SSL," which is connected to the public internet.*

In this case, clarifying whether the software could connect to the public internet was critical to make progress with this SUPPORT CASE.

As another example, while PERFORMING A POST-MORTEM to establish a new practice (T_{10}) the architects kick off the discussion by recapping what the issue was about as well as its status. To do so they shared prior information other people said:

> ***P3:*** *And, uh, so, we've resolved that, but apparently, <u>I know something [persona name] was saying that there's still – he's still seeing high amount of disconnect reconnects.</u>*

From the 19 times participants shared something said by someone not attending the meeting, seven times the information was shared during a SUPPORT CASE that was still open, and four times it was shared to improve a process (PRACTICE AND PROCESS) about SUPPORT CASES that were not handled in the ideal way. This means that recalls of what external people said were most often used to discuss a SUPPORT CASE or to improve an INTERNAL PROCESS.

Observation 18: Recollections of what external people said or what they did are indeed shared, but not very often.

Participants do not seem to re-state what others have said very often. However, these recollections seemed to be particularly relevant for the non-core participants (as introduced in Chapter 3) to articulate the background of topics that had recently arisen, and for which well-organized documentation did not exist. Conversation from Slack channels (e.g., support channel, devops channel), for instance, was particularly useful for this purpose. It seemed to help developers to articulate the background of a recent problem during the meeting so that the key internal stakeholders could provide feedback on how to solve it.

4.9 Are tools used to obtain prior information?

In this section, I present the results of the analysis done to answer RQ 2: What tools do participants use in support of sharing prior information in RSSMDMs? Table 17 shows the variety of tools that the participants of the meetings use to share information. Observe that participants use both development-oriented tools as well as general purpose tools. Also, note the use of some proprietary software (not a tool, but one of their products).

Table 17. The source of the prior information that was shared.

Category	Source	Description
DEVELOPMENT	CONFLUENCE	Project management dashboard for project planning and document sharing.
	JIRA	Issue tracking system used to record, manage, and prioritize problem tickets.
	DEPLOYMENT/MONITORING	Internal or external services (not part of AWS) to deploy software or monitor deployed instances of their software.
	AWS	Amazon Web Services (AWS) are cloud computing services provided by Amazon.
GENERAL PURPOSE	OFFICE SUITE (EXCEL)	Spreadsheet software that supports calculation and graphical presentation of data.
	EMAIL (OUTLOOK)	Outlook is a personal information manager web app from Microsoft consisting of webmail, calendaring, contacts, and tasks services.
	CHAT (SLACK)	Messaging app for business.
NA	PROPRIETARY	The company software products.

I also observed that while drawing and diagramming tools (e.g., whiteboards [13], [85], [86], CASE tools [202]–[204]) have been historically reported as beneficial during design meetings (i.e., create and/or show early sketches or UML diagrams), the participants did not use tools of this kind during the meetings. Instead, most prior information was obtained from a knowledge repository (Confluence) and an issue tracking system (Jira). A chat tool (Slack) and other development tools (e.g., AWS, testing tools) were used as well. In fact, only a few times did information come from a non-textual source. On one occasion, for instance, a participant actively demonstrated a defect in one of their products (PROPRIETARY).

Observation 19: Typical design tools were not used to share prior information in these meetings.

The fact that drawing tools were not used to share prior information during the meetings that I observed does not necessarily mean participants never use them. However, these results do point out that participants find text-based information sharing more practical during the meetings.

As part of this analysis, I also quantified the times prior information was shared through each tool. Table 18 shows the times that the information being shared by one of the meeting participants was visible to all others in the tool that had been brought up via screen share in WebEx. Results appear organized in descending order, both vertically and horizontally. Note that out of 694 pieces of information shared, only 75 times did participants rely on the information displayed in a tool. This means that tools were only used about 11% of the times information was shared; all other times information purely

Table 18. Tools used to share information. Categories not listed did not involve tool use.

Category	Jira	Confluence	E-mail/chat	AWS	Deploy/monitor	Proprietary	Office suite	Total
RUN-TIME INFORMATION		2	3	4	3			12
DEVELOPMENT PROGRESS		5	3		1	2		11
ISSUE	2	2	1					5
TESTING FACT					5			5
CODE FACT	1	2				1		4
FUNCTIONALITY REQUEST	2	2						4
ISSUE DETAIL			4					4
TESTING PROGRESS	4							4
CUSTOMER CONTEXT		2	1					3
DEPLOYMENT MANAGEMENT	1			1	1			3
ARCHITECTURAL FACT		2						2
DEPLOYMENT FACT				2				2
DEPLOYMENT QUALITY ASSESSMENT				1	1			2
PRIOR ISSUE	1		1					2
TEAM HOUSEKEEPING	1			1				2
TEAM PROCESS	1		1					2
TESTING MANAGEMENT	2							2
ANALOGOUS SOLUTION				1				1
CUSTOMER COST				1				1
DOCUMENTATION PROGRESS	1							1
INFRASTRUCTURE FUNCTIONALITY			1					1
INTERNAL COST							1	1
TESTING QUALITY ASSESSMENT	1							1
Total	17	17	15	11	11	3	1	75

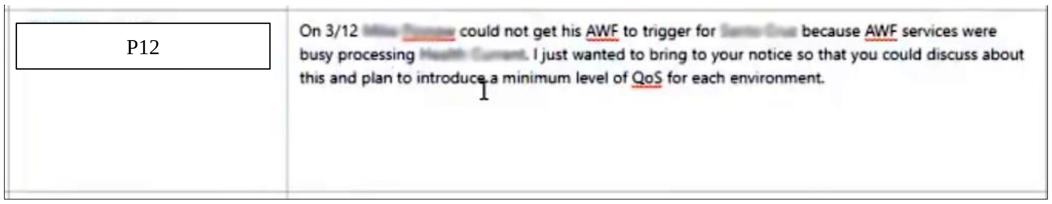
came from participants' own memory and knowledge[12]. Even though the information participants shared from their own memory they did not use a tool to show the information, relying on tools to share information would improve the accuracy of the information shared and avoid the risk of misinforming the team.

Observation 20: Tools, including everyday development tools, are a source of information for the meetings, though they are not the main source.

For the few occasions tools were used (75), I also looked into the ways participants use them to share prior information. As one example, participants typically set the stage for a discussion by bringing up issues in Jira, sharing a wiki page with notes in Confluence or referencing a conversation from a Slack channel. The excerpt below is an example of how a participant set the stage for a discussion with information previously captured in Confluence (see Figure 6).

> ***P2:*** *So, I think that's worth talking about. Um, basically, like 12 days ago, um, [Confidential] was trying to send, uh, automated workflows, uh, out of the [Confi-dential] system, but the entire system was being, uh, clogged by [Confidential] or, you know, another phrase for it is like a noisy neighbor.*

Figure 6. Information shared from Confluence.



P12

On 3/12 [redacted] could not get his AWF to trigger for [redacted] because AWF services were busy processing [redacted]. I just wanted to bring to your notice so that you could discuss about this and plan to introduce a minimum level of QoS for each environment.

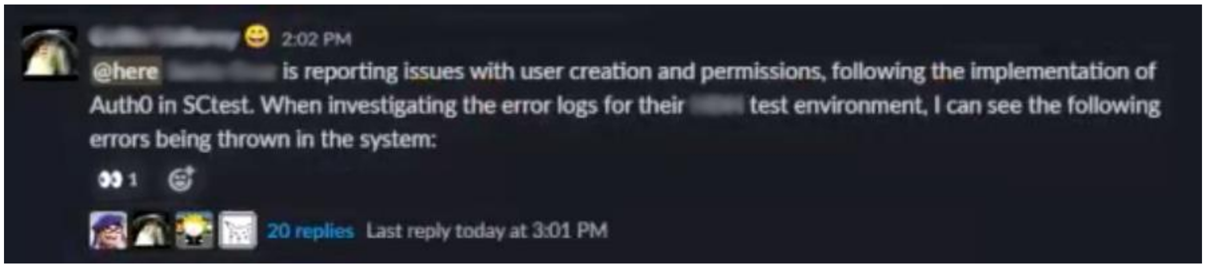
In this example, P2 shared the background of a reported ISSUE to kick off a root cause analysis by using some of the content captured in the Confluence page where P12

[12] A possibility exists that they looked it up on their computer without sharing, which is data we do not have; the team has an established practice to share screens though.

raised the issue and already contributed a few notes documenting the issue and its undesirable behavior. Information from Confluence and Jira was often used this way, though not all issues were introduced only from these tools. For quite a few discussions, the meeting participants would show and cite from the Slack support channel. As one example, a participant shared the following information about an issue, while showing the Slack conversation in which it was reported (see Figure 7).

> ***P2:*** *basically, <u>it looks like, um, we're getting a lot of errors coming out of the API.</u>*

Figure 7. Information shared from a Slack channel.

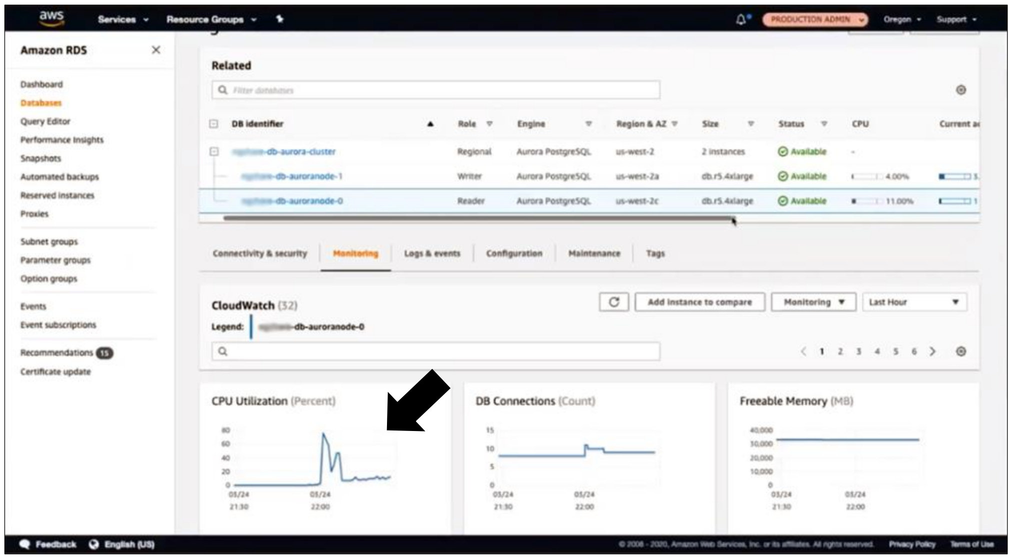


2:02 PM
@here is reporting issues with user creation and permissions, following the implementation of Auth0 in SCtest. When investigating the error logs for their test environment, I can see the following errors being thrown in the system:
20 replies Last reply today at 3:01 PM

Another use of tools was to provide illustrations that helped the team understand the behavior of the deployed system. They used either standard AWS tooling to gain insight into the resource use of their cloud application or would bring up a monitoring tool they had connected to their own logging infrastructure for detailed insight into code-level behavior. As one example, a meeting participant wanted to know whether an observed issue was still a problem (it was) and brought up AWS RDS (RUN-TIME INFORMATION, see Figure 8):

> ***P3:*** *<u>The CPU is higher on the read replica right now.</u>*

Figure 8. Information shared from AWS.



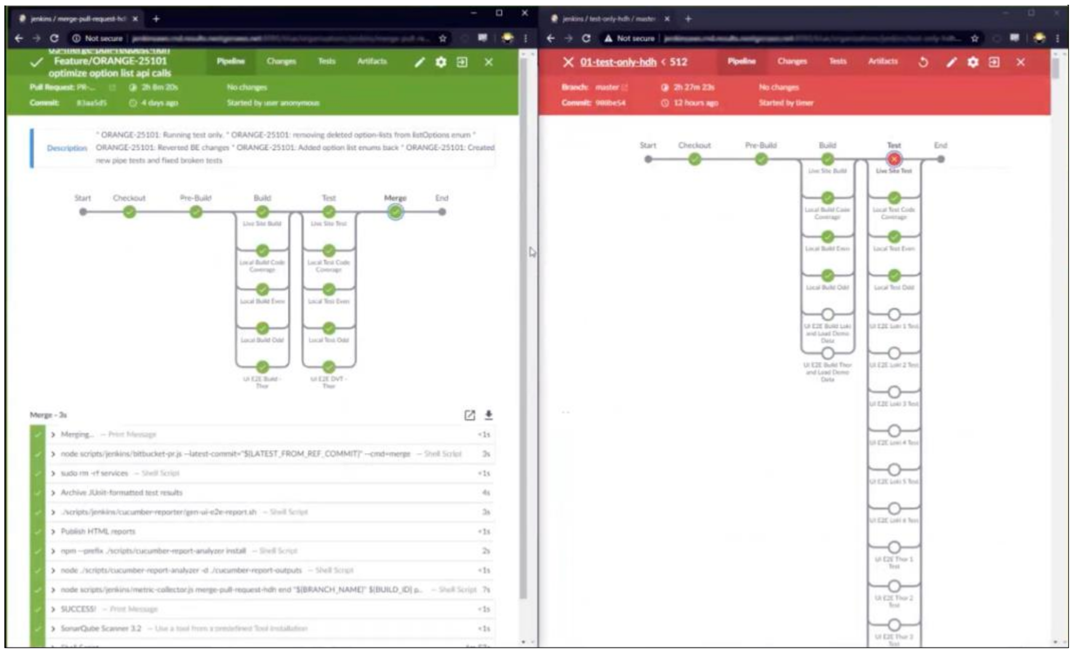
DB identifier	Role	Engine	Region & AZ	Size	Status	CPU	Current a
[redacted]-db-aurora-cluster	Regional	Aurora PostgreSQL	us-west-2	2 instances	Available	-	
[redacted]-db-auroranode-1	Writer	Aurora PostgreSQL	us-west-2a	db.r5.4xlarge	Available	4.00%	
[redacted]-db-auroranode-0	Reader	Aurora PostgreSQL	us-west-2c	db.r5.4xlarge	Available	11.00%	

Legend: [redacted]-db-auroranode-0

As another example, they would study test results to remind themselves of what the various parts of their test suite actually covered. Figure 9 shows two tests side by side for inspection. Various instances of prior information were shared based on the inspection of these two tests. The excerpt below shows what the architects said, the first line is a TESTING FACT shared spontaneously, the second an answer, and the third expanded detail (also contributed voluntarily):

> ***P2:*** *So, here are the two test runs, [P3], and it looks like DB – or sorry, Merge runs everything for UI.*
> ***P3:*** *Oh, okay. We got live site there?*
> ***P2:*** *Yeah.*
> ***P3:*** *Live site, local code, local test. Okay. It's just – it's just everything except all the, uh, mm, perfect.*

Figure 9. Information shared from development a tool.



* ORANGE-25101: Running test only. * ORANGE-25101: removing deleted option-lists from listOptions enum * ORANGE-25101: Reverted BE changes * ORANGE-25101: Added option list enums back * ORANGE-25101: Created new pipe tests and fixed broken tests

```
Merge - 3s
> Merging... — Print Message                                                                  <1s
> node scripts/jenkins/bitbucket-pr.js --latest-commit="${LATEST_FROM_REF_COMMIT}" --cmd=merge — Shell Script   3s
> sudo rm -rf services — Shell Script                                                         <1s
> Archive JUnit-formatted test results                                                         4s
> ./scripts/jenkins/cucumber-reporter/gen-ui-e2e-report.sh — Shell Script                      3s
> Publish HTML reports                                                                        <1s
> npm --prefix ./scripts/cucumber-report-analyzer install — Shell Script                       2s
> node ./scripts/cucumber-report-analyzer -d ./cucumber-report-outputs — Shell Script         <1s
> node scripts/jenkins/metric-collector.js merge-pull-request-hdh end "${BRANCH_NAME}" ${BUILD_ID} p... — Shell Script   7s
> SUCCESS! — Print Message                                                                    <1s
> SonarQube Scanner 3.2 — Use a tool from a predefined Tool installation                      <1s
```

In total, TESTING FACTS were shared three times: two were contributed voluntarily and one in response to an information request.

Finally, in terms of who was driving the tools, one person (P2) predominantly shared his screen since they navigated Confluence and Jira on behalf of the team during the meetings, but other tools were brought up and other participants would take turns sharing screen content when so relevant. Using information in the tools to set the stage was something almost all participants who acted as tool drivers did. However, using tools to share RUN-TIME INFORMATION or configuration details about deployed instances of the software was a way to use the tools that only the architects and some developers did.

4.10 Implications for research, practices, and tools

The results of the previous sections give rise to several observations concerning the current state of information sharing as part of RSSMDMs. These observations have implications for further research, as well as for the tools and practices surrounding meetings of this kind. In this chapter, I discuss these various implications by anchoring them in the observations and research questions introduced in previous sections.

4.10.1 Implications for research

Several findings of this work confirm results from previous studies. As one example, previous research has described software design as a knowledge intensive activity [24]. The fact that I observed *information sharing being extremely frequent (RQ 1.2) and that discussions that involved design work involved more information sharing than discussions that did not (RQ 1.1)*, reinforces this previous finding. On average, one to two pieces of prior information were shared per minute and about 80% of the times prior information was shared, it was done while discussing a design-related topic. Both these findings, empirically validate the knowledge intensive nature of design work that previous researchers have claimed.

Other findings obtained did not confirm but extend more than one existing body of knowledge. Regarding information needs (see Section 2.5), for instance, investigating what kinds of information are shared in these meetings (36 different kinds) describes which are the information needs of developers in RSSMDMs. In the same vein, the findings of this work set apart the kinds of information needed to discuss maintenance design (see Section 2.1) from information that has been found to be important to discuss design work in general (see Section 2.3). *While decisions [18], concerns [91], and constraints [201] have been historically advocated as important for design work, participants did not explicitly share them in these meetings, though these kinds of information sometimes show through in other types of information (RQ 1).* For instance, rather than explicitly mentioning past decisions and their associated rationale, the participants described the current state of the codebase

(CODE FACTS) to then share ARGUMENTS about why certain functionalities were coded that way. Even though some instances of decision-making and rationale show through in other kinds of information during the meetings (in a way akin to the one just described), these instances account for no more than the 7.4% of the information shared. In general, *the kinds of information developers relied on the most to discuss design were related to the system as deployed and under development (RQ 1).*

The results obtained in this study also serve as a baseline for future research in more than one area. Regarding rationale (see Section 2.7), for instance, this work raises an interesting question: how often is rationale information needed for other types of design work? While for maintenance design rationale was not shared that often, a different distribution might be observed for other kinds of design work (e.g., green-field design, architectural design, re-design to address technical debt). In relation to knowledge management (see Section 2.6), however, the fact that *a significant amount of information sharing was done by a limited number of participants (RQ 1.5)* raises the need to confirm whether this represents a risk. Given that these people may leave the team at some point, it would also be an important topic to study how teams prepare for such occasions.

In terms of information needs (see Section 2.5), various strands of future work may be followed as well. For example, *even though the absence of information did not appear to slow down the discussion when an information request was not met, the actual impact is unknown at this time (RQ 1.4)*. This raises some interesting research questions as well, for instance: Were the answers not really needed? Were answers deferred to the team doing more detailed design once it has been considered by the key internal stakeholders and assigned? Or did the absence of answers lead to later problems? Further study is necessary to understand the impact of MISSING INFORMATION in RSSMDMs.

Finally, a future research agenda should also address the limitations of the present study. Given that I studied a particular kind of design work (maintenance design, see Section 2.1) and setting (RSSMDMs) for one team of developers all from the same company, further research is required to generalize (if applicable) the findings obtained from this work. Future studies may, for instance, replicate this study with other teams that also held maintenance design meetings, or with teams that conduct other kinds of meetings in which, at a small or large scale, participants engage in discussing design work (e.g., green-field design meetings, SCRUM meetings).

4.10.2 Implications for practice

My research and findings also have important implications for practitioners in relation to the way they run maintenance design meetings, and perhaps design meetings in general. First, *the wild diversity in the types of information being shared in RSSMDMs (RQ 1)* reflects the interest of the key internal stakeholders in ensuring that all the information needed to shape design in the context of a deployed and functioning system gets considered. For instance, in the case of post-mortem discussions, the way the architects shared best practices and introduced procedures is a practice that many other experienced developers may find useful to implement. As a second example, the way in which the product owners intervened to share information about the customers in critical moments of the discussion is also a good practice that other product owners may want to apply to ensure *the team performs its design work by accounting for what is happening at their customers (RQ 1)*.

Second, regardless of who shares information, their background, experience, knowledge of the codebase or the customers, participants *do not always take for granted all the information that is shared and do point out when wrong information is shared (RQ 1)*. This is a good practice that team leads must encourage among the team members. Even though instances of MISINFORMATION were barely observed during these meetings, the times participants pointed out wrong information was being shared and actually provided the right information, saved the team from taking a wrong direction with the design. Such corrections are highly valuable for everyone.

Another important observation in relation to practice is that *a significant amount of information sharing was done by a limited number of participants (RQ 1.5)*. In these meetings, these participants were the software architects, who shared over 75% of the prior information. On the one hand, this can be expected because it is their express responsibility to lead the design and development from a technical point of view. On the other hand, the concentration of knowledge in just two of the twelve team members might be an issue of concern if these team members were to leave the company or transition to another team. From that perspective, it is important that software development teams consciously engage in knowledge sharing, and knowledge documentation, so that important knowledge about the system and the team's internal practice does not reside only in a few people. Instead, knowledge should be spread among several team members. While it is not necessarily harmful or risky that few people accumulate large amounts of knowledge, it is potentially harmful when only those people know it. A primary motivating factor for establishing the RSSMDMs in this company was precisely to enable such knowledge sharing, together with the meetings being the place to address issues that continued to emerge. Companies in which recurring meetings of this nature have yet not been installed, could justify the need of installing them to improve one or both of these situations.

4.10.3 Implications for tool support

This work also uncovered important implications for the development of better tools to assist prior information sharing during design meetings. First, some of the findings obtained point out where current tools fall short in regards to prior information sharing. For example, the fact that tools were only used approximately 11% of the times information was shared is a red flag to investigate why: Could it be that tools do not live up to their potential to assist developers with information sharing during meetings? Could it be that users found it difficult to find information in the tools during the meeting? It would be interesting to know why, *even though tools, including everyday development tools, were a source of information for the meetings, they were not the main source (RQ 2)*. As another example of tool support falling short relates to the fact *typical design tools were not used to share prior information (RQ 2).* Historically, design tools (see Sections 2.3, 2.6, and 2.7) have been reported as beneficial for design work. However, participants did not use tools of this kind during the meetings. This does not necessarily mean these tools are not important for this kind of design setting. Could participants find it useful to be able to look at diagrams of the systems architecture while discussing ARCHITECTURAL RE-DESIGNS, or while DEVISING SOLUTIONS for SUPPORT CASES? An experimental study of this team using diagrams of the architecture during discussions of this kind would be a starting point to investigate.

My work also uncovers implications for the development of better tools to support information sharing in RSSMDMs. Observing that *specific kinds of information were more relevant for certain types of work (RQ 1.1)* calls attention to creating tools able to pull information from multiple sources to share during the meeting. This research would extend the scope of previous work in knowledge repositories (see Section 2.6). While knowledge repositories were created to serve as a primary source of design information, my study shows that other tools, for instance issue tracking systems (i.e., Jira) were nearly equally important to obtain it. The creation of mechanisms to interconnect information between knowledge repositories (i.e., Confluence) and these other tools (e.g., Jira, Slack, AWS) might bring a positive change to developers' practice. In fact, some simple mechanisms towards this direction have already been implemented through plug-ins that enable the creation of links between Confluence and Jira. However, the integration of information snippets from other sources such as Slack channels, or AWS dashboards would be equally useful. In a similar vein, meeting assistants could support participants by suggesting relevant information for specific maintenance discussions based on the participants' conversation. For instance, while discussing the root cause analysis of a past issue, a smart meeting assistant could automatically share links to the tickets associated to that issue. As a second example, when utterances requesting RUN-TIME INFORMATION are made (i.e., "*what is the usual number of requests for that API?*"), an automated assistant could share the access to AWS dashboards containing this information. The state of the art in language processing techniques [205] is already able to identify certain utterances and trigger actions based on them (e.g., [206]–[209]), though utterances that implicitly represent an information need (i.e., "*I don't know what the NFR is for that"*) would be more challenging.

Another factor to consider in this regard is the way in which information sharing happens. In these meetings, I observed that *information is shared spontaneously as part of the natural unfolding of the discussion, with a small portion explicitly requested (RQ 1.3)*. These results point out a challenge for the design of tools that proactively provide information. While the formulation of questions could be used as a cue to trigger queries to obtain information from a knowledge base (i.e., Confluence), only about 15% of the information in these meetings was requested. The rest was voluntarily shared, meaning that new ways to detect information needed (not only based on questions) should be investigated. Finally, another factor to consider in the design of future tools supporting RSSMDMs are *the recollections from external people that were shared during the meeting (RQ 1.6)*. Even though participants did not share what others had said very often, these recollections were particularly relevant in some cases. For instance, while working on SUPPORT CASES, information from technical Slack channels (e.g., support channel, devops channel) was particularly useful to understand the problems being reported. It helped developers to provide the situation's background in order to obtain advice from the architects on how to solve it. In this regard, an interesting future research direction could be investigating the extraction of useful information snippets from Slack channels to be shared during RSSMDMs when so relevant.

5 Outflow Captured during RSSMDMs

Finding better ways to capture information generated in software development meetings is not a recent problem (see Section 2.4). Sometimes, a meeting participant is tasked with taking notes during the meeting. Other times, someone may take a picture of the whiteboard used at the end of the meeting and share it with the rest of the team or participants may simply rely on their memory to retain information during the meeting, to then create more formal notes or diagrams of what was designed afterwards.

Tools to assist meeting information capture have also been the subject of study. Some examples are tools to capture early design sketching (e.g., [13], [85], [86]) or multi-media recordings of the meeting (e.g., voice notes [123], full meeting recording [120], [121], [124]). Moreover, tools to document information from the meeting once is has concluded have been studied as well (see Section 2.4). However, lacking today is a study that classifies the new information that is generated during the meetings. In this dissertation, I call this information discussion outflow. The participants in RSSMDMs capture information generated during the meetings to document the work done for future use (e.g., individual work, subsequent meetings about the same topics, unforeseen future updates to the project). Without a proper way to capture discussion outflow, future design conversations in RSSMDMs may turn ineffective and excessive reverse re-engineering [210] and in some cases even re-work might be needed. As a result, future design work may take longer than it should.

My study fills this gap in the literature by describing what kinds of information are captured as discussion outflow in RSSMDMs. The participants in RSSMDMs may, for instance, capture information that contributes to understanding a problem, or generating a solution for it (e.g., decisions [18], alternatives [199], constrains [201]). They may also share a design proposal with the rest of the team with the goal of obtaining feedback to improve it, with the feedback made as annotations on the design proposal or simple notes on a Google doc.

Part of the goal of my dissertation is to characterize the variety of discussion outflow that the participants in these meetings generate and capture. Another important part of this dissertation's goal is to describe the tools and artifacts that participants use to capture it. In Chapter 1, I introduced two primary questions that seek to describe how discussion outflow is captured in RSSMDMs (RQ 3 and RQ 4). This chapter answers these two questions, together with a number of secondary questions. The answers provide a comprehensive view about how information flows out of RSSMDMs.

RQ 3 What is the discussion outflow of the topics addressed in RSSMDMs?

RQ 3.1 What kinds of discussion outflow are captured more often?

RQ 3.2 Is discussion outflow always captured?

RQ 3.3 Are certain kinds of discussion outflow captured during certain kinds of work?

RQ 3.4 Are certain kinds of discussion outflow captured for discussions of a certain purpose?

RQ 4 What tools do participants use to capture discussion outflow in RSSMDMs?

RQ 4.1 Are specific artifacts used to capture discussion outflow?

RQ 4.2 Are certain tools used more often to capture discussion outflow?

RQ 4.3 How do participants use tools during RSSMDMs?

RQ 4.4 What participants drive the tools to capture discussion outflow?

RQ 4.5 How are tool drivers prompted to capture discussion outflow?

The remainder of this chapter is organized as follows. Section 5.1 explains the methodology used to analyze the data. Section 5.2 presents the results associated to RQ 3. Sections 5.3 and 5.4 present the results of RQ 3.1 and RQ 3.2 respectively. In Section 5.5, I present the results of RQ 3.3 and RQ 3.4. Then, Section 5.6 presents the results for RQ 4 and RQ 4.1, Section 5.7 the results for RQ 4.2, and Section 5.8 the results for RQ 4.3. In Section 5.9, I present the results that answer RQ 4.4 and RQ 4.5. Finally, in Section 5.10, I discuss the implications of the findings for research, practice, and tool development.

5.1 Methodology

I analyzed the transcripts and videos of ten meetings from a healthcare software development company (details about this dataset are provided in Chapter 3). The analysis was primarily performed on the videos to detect when participants captured information in a tool. For instance, participants would typically share their screen to create a new ticket in an issue tracker system (Jira) to capture some information. As another example, they captured meeting notes in a wiki page with a pre-loaded template for that purpose (Confluence). Non-verbal interaction of this kind is only possible to detect by watching the videos. The transcripts were also analyzed to detect discussion outflow that was not captured in tools. For instance, on quite a few occasions someone on the team would task other participants with creating a support ticket, requesting new accounts for a team member after the meeting, or organizing a list of issues to be triaged in a future meeting.

As a preliminary step, I partitioned each meeting based on the topics being deliberated (I described this process in Chapter 3). Once the meeting had been partitioned into smaller topic discussions, a thematic analysis [29], [30] to identify and categorize all the occasions in which discussion outflow was captured for each topic was performed. I describe this process below.

5.1.1 What to consider as discussion outflow and what not

A first step to perform the thematic analysis was defining what information should be considered discussion outflow. Two researchers (one of them myself) develop a set of rules to identify discussion outflow in the data. These rules stem from the idea of considering as discussion outflow all moments in which information was captured in tools that were shared on screen. I list the three rules so defined below.

Rule 1. Consider as discussion outflow information that is captured in tools that are being shared on screen.

Rule 2. Every time participants typed information in a tool. For instance, a tool driver could uninterruptedly capture part of an idea for a while, then engage in discussion (no typing), to, thereafter resume typing to complete what was being captured. Each of those times outflow was captured it was tagged as a separate occasion.

Rule 3. Discussion outflow could be captured in three different ways: (1) the tool driver could voluntarily capture discussion outflow, (2) the tool driver could capture discussion outflow as per the explicit request of someone, or (3) tool drivers may capture information that they asked of other participants.

5.1.2 Classifying discussion outflow

Two researchers performed an inductive thematic analysis [29], [30] to categorize the kinds of prior information identified in the sessions. We followed the set of steps below to create a coding scheme to perform this categorization:

1. Open coding: two researchers (one of them myself) independently performed open coding on the first meeting, identifying each place where they felt information was being captured. We also independently categorized the pieces of information identified.
2. Collaborative review: we then met, compared, and discussed our respective findings, and created a first version of a coding scheme organizing the categories of information being captured in the meeting.
3. Final review: a third researcher reviewed the coding scheme and assigned codes of the first meeting and gave feedback, which led to further changes to the coding scheme and codes assigned.
4. The two researchers who performed the coding of the first meeting (Step 1) then independently analyzed the second meeting. Thereafter, the two researchers compared and discussed their findings, which led to a refined coding scheme that the third researcher once more reviewed. The suggestions that the third researcher provided led the two researchers to several updates to the coding scheme, which they then reflected by updating the codes for the second and first meeting.
5. All other meetings were analyzed meeting-by-meeting following the process used for the second meeting, with any updates to the coding scheme reflected by recoding earlier meetings as needed. With each new meeting, the coding scheme slowly but surely stabilized in terms of the categories it contained.

6. Axial coding: the two researchers (one of them myself) then performed axial coding, reviewing all the categories of the coding scheme one by one, examining the internal consistency of all assigned codes in each category as well as potential overlaps among categories. As a result, a few categories were merged, and several assigned codes were changed to be consistent with one another.

Figure 5 in Section 4.1.2 shows a visual representation of the process (steps one to six) that was followed.

Once an instance of discussion outflow was identified in the videos, the coding process previously described was followed to characterize the kind of information that was captured as outflow. This process provided the basis for answering RQ 3. A table showing the resulting kinds of discussion outflow observed is presented in Section 5.2. An analysis of these results per meeting and per topic provided the answers for RQ 3.1 and RQ 3.2, respectively. Then, by juxtaposing the results obtained to answer RQ 3 with the kinds of work and the purpose of the discussions (both introduced in Chapter 3), I answered RQ 3.3 and RQ 3.4.

To answer other research questions other coding passes were performed to tag each already identified prior information fragment with several additional codes. These additional codes were distilled from what each of these questions sought to answer.

For RQ 4 and RQ 4.1, two researchers (one of them myself) coded the TOOL that was used to capture information and the ARTIFACT (type of document) in which information was captured. To answer RQ 4.2, an analysis of the tools used per topic provided the answer. Then, to answer RQ 4.3, the researchers coded all the ACTIONS performed with the tools. All the actions observed are presented in Section 5.8.

In order to answer RQ 4.4, the same two researchers coded which participant was using the tool shared on screen to capture outflow (TOOL DRIVER), as well as the respective role this participant had (TOOL DRIVER'S ROLE). The codes applied to identify TOOL DRIVERS are the same we used to identify the meeting participants in Chapter 3 (see Table 1). The roles of the participants that attended these meetings were presented in Chapter 4 (see Table 8).

To answer RQ 4.5, the researchers focused on coding what prompted TOOL DRIVERS (participants driving the tool shared on screen) to capture (tool-based) outflow. They identified three different ways: PROMPTED, UNPROMPTED, and REQUESTED BY TOOL DRIVER. Table 19 shows the description of each of these codes. In addition, for each instance of discussion outflow coded as PROMPTED, the researchers coded the segment in which the PROMPT was made. Then, for each instance of discussion outflow coded as REQUESTED BY TOOL DRIVER, the respective REQUEST was coded as well. Table 19 also shows the description of these codes.

Table 19. The ways tool drivers capture information in a tool shared on screen.

Category	Description	Additional codes
UNPROMPTED	The tool driver was not explicitly requested by someone to capture discussion outflow.	NA
PROMPTED	The tool driver captured discussion outflow as per the explicit request of someone.	PROMPT: The segment in which capturing something is requested.
REQUESTED BY TOOL DRIVER	The information captured is something the tool driver asked for from other participants.	TOOL DRIVER'S REQUEST: The segment in which the tool driver requests son information.

For confidentiality reasons, the healthcare software development company does not allow me to share the videos or transcripts; I do, however, have permission to share anonymized extracts from the discussions and anonymized screenshots of the meetings in this dissertation.

All qualitative coding (when it applied) was performed in MAXQDA [198], with 167 codes assigned to discussion outflow segments to answer RQ 3, and 167 codes to answer RQ 4. For RQ 4.1, RQ 4.4, and RQ 4.5, 167 codes were added per question. To answer RQ 4.3 in particular, 167 codes were added to identify the times information was captured, and 65 additional codes were assigned to identify other ACTIONS done with the tools. Together, all these codes account for a total of 1067 codes. A few other codes were added to some segments to, for instance, identify the PROMPT or REQUEST made by TOOL DRIVERS (to complement RQ 4.5).

5.2 The kinds of discussion outflow that were captured

In this section, I present the results of the analysis performed to answer research question RQ 3: What is the discussion outflow of the topics addressed in RSSMDMs? Table 20 shows the different kinds of discussion outflow observed when new information was added or information was refined. Table 20 is organized as follows: the second and third columns show the different kinds of information captured and their respective descriptions. The first column shows a meta-classification of the kinds of discussion outflow observed. I alphabetically ordered the results in Table 20 by the "meta-classification" of the information categories (first column).

At a high level, the meta-classification of the 16 kinds of discussion outflow that were captured shows that participants captured DESIGN INFORMATION. Observe that only one type of information (PROBLEM BACKGROUND/CODE STATE) in this group relates to the problem space, the rest relates to various aspects of the solution proposed to fix a

Table 20. The kinds of discussion outflow observed when new information was captured, or information was refined.

Meta-classification	Category	Description
COORDINATION	DISCUSSION ITEM (TOPIC)	The information captured is a topic they will discuss during the session.
	ACTION ITEM	An action item is a single, clearly defined task that must be done.
DESIGN INFORMATION	PROBLEM BACKGROUND/CODE STATE	The background of a design problem to be discussed; the state of the codebase, and how things currently work.
	ISSUE/TICKET HIGH LEVEL DESCRIPTION	High level description of a development task for which the team decided to create either an issue or a ticket, but for which a detailed design had not been defined yet; this is the initial description added to a ticket.
	IDEA/ALTERNATIVE	Design ideas or alternatives to solve a problem or improve the state of the system.
	IMPLEMENTATION GOAL/SCOPE	The goal for some future functionality or update to the codebase is captured.
	IMPLEMENTATION ROADMAP	The roadmap of a major development update (i.e., testing pipeline)
	RATIONALE	The reason behind a design decision, proposal, or idea.
	REQUIREMENT	A new functionality desired.
	THINGS TO KEEP IN MIND	Feedback received on a design proposal.
TEAM PROCESS	ADMINISTRATIVE DECISION	The approval of administrative tasks during the meeting (i.e., approval to create new accounts).
	BEST PRACTICE	Documenting a best practice.
	PLAN OF ACTION FOR CERTAIN SITUATION	Documenting a set of steps to take.
	SITUATION'S BACKGROUND/STATUS	The background information of a topic to be discussed (i.e., the status of a support case).
PROJECT MANAGEMENT	IMPACT ESTIMATE	An estimate of how much impact a development could have on the end users, the codebase, or resources.
	SCHEDULING ESTIMATE	An estimate of how long a development would take.

problem (e.g., ISSUE/TICKET HIGH LEVEL DESCRIPTION, IDEA/ALTERNATIVE, IMPLEMENTATION GOAL/SCOPE, REQUIREMENT, THINGS TO KEEP IN MIND). The participants also captured some RATIONALE, sometimes to explain why the current state of code is what it is (in regards of the problem space), or to justify why certain feature had to be implemented in a certain way (in regards of the solution space). Information related to the team's ways of operating (TEAM PROCESS), information to coordinate future activities (COORDINATION), and some information related to PROJECT MANAGEMENT were also captured. PROJECT MANAGEMENT information was specifically about aspects of the code (e.g., IMPACT ESTIMATE, DEVELOPMENT ESTIMATE) while COORDINATION activities were more general (DISCUSSION ITEM, ACTION ITEM). In comparison to DESIGN INFORMATION, the kinds of information observed in these other groups (TEAM PROCESS, COORDINATION, PROJECT MANAGEMENT) were less diverse. An observation that stems from comparing the information that was shared (as introduced in Chapter 4) with the kinds of information that were captured (all shown in Table 20) is that:

Observation 21: The kinds of discussion outflow captured were less diverse in nature than the prior information that was shared.

While participants shared 36 different kinds of prior information during their discussions (see Chapter 4), only 16 different kinds of discussion outflow were captured. As an example of the kinds of design information captured, during T_6, the IMPLEMENTATION ROADMAP for a renovated performance testing suite was captured. This implementation roadmap was composed of five steps. The summary of one of the steps in this plan was captured as "*PoC on writing component test for FE*". Then, an associated piece of RATIONALE was also captured in relation to this step to justify why the step was needed, "*Biggest bump for our buck is increase component performance testing*". At the very end of this discussion, the participants also captured "*< 1 Spring*". They captured this information after estimating how long would the development of this step of the roadmap take (SCHEDULING ESTIMATE). During the same discussion, an ACTION ITEM in relation to the first step of the plan was also captured ("*To be ticketed*") to remind the team members a ticket for the work related to that step had to be created. According to the meta-classifications shown in Table 20, specific kinds of DESING, COORDINATION, and PROJECT MANAGEMENT information were captured during this discussion. Only TEAM PROCESS information was not captured.

I also informally explored if some of the prior information shared was captured. I did observe some occasions in which prior information shared was captured as part of a piece of discussion outflow in some topics. For instance, I witnessed the statement "*currently one queue handles both, evaluation and action*" captured as part of the problem's description (PROBLEM/BACKGROUND STATUS) of T_3. In this example, the information captured was a CODE FACT that one of the architects shared during the discussion. Given that a systematic method was not followed to establish connections between what was shared and what was captured, I do not include this as a formal finding of this work. However, this is an interesting research direction to explore in future. I discuss this idea in Section 5.10.

Another important part of this analysis was identifying the kinds of prior information shared that were completely absent in what was captured as discussion outflow. For instance, note that information about the system as deployed and under development (e.g., RUN-TIME INFORMATION, DEVELOPMENT PROGRESS, DEPLOYMENT MANAGEMENT, DEPLOYMENT FACT, as defined in Chapter 4) does not appear in Table 20.

Observation 22: Even though information about the software as deployed (e.g., RUN-TIME, DEPLOYMENT FACT) was the kind of information most often shared as part of the discussions, such information was not captured as discussion outflow.

Participants often shared information about the software as deployed and under development to build up the context of a problem. These kinds of information are typically fleeting, they seem to have only a temporal relevance in the discussion. T_3 (a discussion that addressed the ARCHITECTURAL RE-DESIGN of certain part of the software), offers an example. With the goal of completing the problem's description, participants shared different kinds of information about the deployed instance of the software in which the problem occurred. The excerpt below shows some of the prior information that was shared:

> ***P2:*** *...Um, so, basically, I think <u>it was backed up like 12 hours or something, because [service name] had an external system they were sending to. Um, it was responding very slowly, like 30 second responses, and there were millions of automated workflows that were trying to be sent out, but because it was so slow, it was just backing everything up.</u>*

While P2 shared this information to communicate how this issue affected the clients (RUN-TIME INFORMATION, underlined in the third paragraph), it was not captured anywhere, neither at the time P2 brought it up nor later.

5.3 Kinds of discussion outflow captured more often

In this section, I present the results of the analysis that answers research question RQ 3.1: What kinds of discussion outflow are captured more often? To answer this research question, I counted the times information was captured for each kind of discussion outflow presented Table 20 (second column). The result of this analysis was a mapping of the 16 categories (in Table 20) to 167 times in which the participants shared their screen and captured information as outflow of the discussion. Table 21 presents this mapping across the ten meetings. The table shows the times each type of information was captured per meeting (from column two to column 11), together with the totals across all meetings (column 12). Most importantly, it shows in how many meetings a kind of information was captured (last column). Every information kind that was captured at least once during a meeting was considered in this count. Shading identifies all the information kinds for which discussion outflow was captured at least once during a meeting, as well as, in the last column, the kinds of information captured in the most meetings. Results appear organized

Table 21. Kinds of discussion outflow that were captured per meeting.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	All meetings	Captured in # meetings
Category	#	#	#	#	#	#	#	#	#	#	#	#
IDEA/ALTERNATIVE	13	2	0	0	9	0	6	0	6	0	36	5
PROBLEM BACKGROUND/CODE STATE	3	1	0	0	7	0	0	0	3	0	14	4
IMPLEMENTATION ROADMAP	1	7	0	0	0	0	0	2	1	0	11	4
ACTION ITEMS	0	1	0	3	1	0	0	2	0	0	7	4
PLAN OF ACTION FOR A CERTAIN SITUATION	0	0	0	10	0	0	7	0	0	6	23	3
DISCUSSION ITEMS	0	0	0	11	5	0	0	3	0	0	19	3
RATIONALE	0	1	0	0	2	0	0	6	0	0	9	3
THINGS TO KEEP IN MIND	0	0	6	0	0	0	0	12	0	0	18	2
SITUATION'S BACKGROUND/STATUS	0	0	0	7	0	0	2	0	0	0	9	2
ISSUE/TICKET HIGH LEVEL DESCRIPTION	3	0	0	0	2	0	0	0	0	0	5	2
IMPLEMENTATION GOAL/SCOPE	1	0	0	0	0	0	0	0	3	0	4	2
SCHEDULING ESTIMATE	0	3	0	0	0	0	0	0	0	0	3	1
REQUIREMENT	0	0	0	0	3	0	0	0	0	0	3	1
IMPACT ESTIMATE	0	0	0	0	0	0	0	2	0	0	2	1
ADMINISTRATIVE DECISION	0	0	0	2	0	0	0	0	0	0	2	1
BEST PRACTICE	0	0	0	2	0	0	0	0	0	0	2	1
Total	21	15	6	35	29	0	15	27	13	6	167	

from the kind of information captured in more meetings to the kind of information captured in the least meetings.

The takeaway from the results in Table 21 is the total number of different kinds of information shared per meeting. Note that across the ten meetings the categories most often captured were: IDEA/ALTERNATIVE (five meetings), PROBLEM BACKGROUND/CODE STATE (four meetings), IMPLEMENTATION ROADMAP (four meetings), ACTION ITEM (four meetings), PLAN OF ACTION FOR CERTAIN SITUATION (three meetings), DISCUSSION ITEM (three meetings), and RATIONALE (three meetings).

An important observation that stems from these results is that:

Observation 23: DESIGN INFORMATION is what participants of RSSMDMs most often captured as discussion outflow.

Except for ACTION ITEMs (captured in four meetings) and the PLAN OF ACTION FOR CERTAIN SITUATIONs (captured in three meetings), all other information kinds do relate to design work. During half the meetings, the participants captured proposed solutions (IDEAS/ALTERNATIVE). Moreover, details about the background of design problems (PROBLEM BACKGROUND/CODE STATE) as well as the IMPLEMENTATION ROADMAP of solutions for those problems were captured in four meetings.

On the one hand, these results highlight that the team captures design information more than anything else during the meetings, which aligns with the general good practice that the literature recommends (e.g., [24], [211]). On the other hand, the fact that nothing was captured during the sixth meeting and that the kind of information that was most often captured (IDEA/ALTERNATIVE) was only captured in five meetings raises the concern of whether discussion outflow was indeed always captured. I elaborate more on this in the next section.

5.4 Understanding why discussion outflow is not always captured

In this section, I present the results of the analysis conducted to answer RQ 3.2 Is discussion outflow always captured? Towards answering this question, I analyzed how much discussion outflow was captured per topic.

Table 22 shows the results of this analysis. The left side of the table (first three columns) shows the meetings in which a topic was discussed (first column), the topic identifier (as defined in Chapter 3) for which information was captured (second column), and how often information was captured for each topic (third column). The results on this side of the table appear organized in descending order, from the topic(s) in which more times information was captured, to the topic(s) in which less times information was captured. The right side of the table (columns four and five) shows the meetings for which information was not captured. The fourth column shows in which meetings each topic was

Table 22. Discussion outflow captured per topic.

Topics for which discussion outflow was captured			Topics for which discussion outflow was not captured	
Meeting(s) in which the topic was discussed	Topic Id	Times information was captured	Meeting(s) in which the topic was discussed	Topic Id
M5	T_{11}	27	M1	T_{2}
M1	T_{3}	21	M1	T_{4}
M4	T_{9}	21	M2	T_{5}
M2	T_{6}	15	M3	T_{7}
M1, M4	T_{1}	14	M5	T_{10}
M8	T_{23}	12	M6	T_{13}
M7	T_{21}	9	M6 M,8	T_{14}
M9	T_{38}	9	M6	T_{15}
M8	T_{27}	8	M6	T_{16}
M3	T_{8}	6	M6	T_{17}
M7	T_{20}	6	M6	T_{18}
M8	T_{24}	6	M7	T_{19}
M10	T_{39}	6	M8	T_{22}
M5	T_{12}	2	M8	T_{25}
M9	T_{31}	2	M8	T_{28}
M8	T_{26}	1	M9	T_{30}
M9	T_{29}	1	M9	T_{32}
M9	T_{36}	1	M9	T_{33}
Information captured	18 Topics	167 times	M9	T_{34}
			M9	T_{35}
			M9	T_{37}
			M10	T_{40}
			M6	T_{41}
			M6	T_{42}
			M8	T_{43}
			M8	T_{44}
			M9	T_{45}
			Information captured	27 Topics

discussed, and the fifth column shows the topic identifier (as defined in Chapter 3). For all the topics on this side of the table information was never captured.

More than the exact number, or the average number of times information was captured per topic, what stands out from these results is the number of the topics in which discussion outflow was not captured at all. For 27 topics, no discussion outflow was captured. In fact, for all the topics discussed during the sixth meeting ($T_{13-18,}$ $T_{41,}$ T_{42}), zero instances of discussion outflow were captured.

Observation 24: Discussion outflow was captured for less than half the topics that were discussed across the ten meetings.

On the one hand, these results may mean that important information could have been lost (not captured). On the other hand, the results may also mean that capturing the outflow of certain discussions is not always required.

To better understand what these results mean, I organized the 27 topics for which no discussion outflow was captured per kind of work. Table 23 presents the resulting mapping. The first column shows the topic identifier, and from the second to the ninth columns I list each type of work (a categorization of topics per type of work was introduced in Chapter 3). I analyzed each of these discussions in detail with the objective of providing

Table 23. Topics for which discussion outflow was not captured per kind of work.

Topic Id	ADMINISTRATIVE TASK	COORDINATION AND PLANNING	INFRASTRUCTURE	INTEGRATION OF NEW FUNCTIONALITY	PERFORMANCE	PRACTICE AND PROCESS	SUPPORT CASE	TESTING	
T2					1				
T4				1					
T5							1		
T7								1	
T10						1			
T13	1								
T14		1							
T15							1		
T16						1			
T17				1					
T18							1		
T19	1								
T22						1			
T25						1			
T28			1						
T30			1						
T32			1						
T33			1						
T34			1						
T35			1						
T37			1						
T40							1		
T41								1	
T42				1					
T43		1							
T44			1						
T45		1							
Total topics	2	3	8	3	1	4	4	2	27

some additional insight into how harmful it might be not to capture discussion outflow. Below, I share some high-level observations in this regard.

Capturing discussion outflow did not seem to be that relevant during some discussions. For instance, the discussion of T_{13} (the team in India requested additional privileges in their accounts to monitor software) and T_{19} (a development account was not working as it should and the team worked on figuring out why) did not seem to require outflow to be captured because both addressed ADMINISTRATIVE TASKs in which actions were performed during the meeting, rather than being documented as ACTION ITEMs to take care of later. Other discussions in which capturing outflow did not seem needed concerned activities to coordinate work during the meeting or to broadcast information about future projects among the team members (COORDINATION AND PLANNING). As one example, during T_{14}, a brief announcement about the status of a project to integrate APPSEC (security software to check vulnerabilities) into the development pipeline was shared. As another example, during T_{45} a participant briefly explained the logistics of that meeting. For both discussions, agreements were not made, nor were significant doubts solved. The simple act of informing the participants about it, without documenting anything, seemed to be sufficient. That said, it is unknown whether others who were not at the meeting could have benefited had the shared information been captured.

During some topics capturing information seem to be relevant only to a specific participant. On a couple of occasions, one of the product owners (P1) raised questions about future projects for which development had not yet been started. One example is T_4, in which P1 shared with the team that a client decided to build its own IdP (Identity Provider) solution. P1 requested help to evaluate the risks of this decision in preparation for a future meeting with the customer and external team leads (not part of this team). Something similar happened during T_2, in which P1 desired to know whether a client onboarding additional users may incur an additional cost for which the client should be charged. The answers provided to these questions were not documented in any tool. However, at that moment, this information seemed to be relevant only for P1.

During some discussions no design information was captured, though participants did use tools to set metadata (e.g., dates, special tags to set the status or priority of a development). For example, during meeting eight, the team centered on planning various details of a future TICKET TRIAGE session. During this meeting, the participants used T_{28} (a ticket about a load balancer not being compliant with the company policies) and T_{40} (an open issue at one of the client sites to integrate two different applications under the same authentication mechanism) as examples to reflect on how to determine the value of tickets. Then, the actual triaging session was conducted during meeting nine. The participants started this meeting by providing an overview of the process to be followed (T_{45}) to then engage in the actual triaging. T_{30}, $T_{33\text{-}35}$, and T_{37} were some of the tickets reviewed during this session. Information was not captured in the tickets during any of these meetings. However, the participants did add some tags to classify them. Most of these topics appear classified as INFRASTRUCTURE (see Table 23), the triaging session conducted was about INFRASTRUCTURE tickets.

Discussion outflow was not captured during unplanned conversations. Some of the discussions for which outflow was not captured were not part of the original meeting agenda. For instance, unplanned topics that were discussed were often SUPPORT CASES (T_5, T_{15}, T_{18}, and T_{40}) that participants requested to discuss to seek guidance from the key internal stakeholders. Other unplanned discussions (T_{41}, T_{42}, T_{43}) were spontaneously started in relation to the previous topic. For example, during T_{42} the team brainstormed some ideas to improve the documentation embedded in the system for end users. This topic was spontaneously started after discussing a support case (T_{15}) in which a participant demonstrated an issue reported in the system. The participant actively used the system to reproduce the issue (while sharing the screen), then someone happened to ask if the documentation to use that functionality was easily available to end users. This was the comment that got the discussion of T_{42} started.

Not capturing discussion outflow did not seem to be that harmful for most unplanned topics. For discussions that addressed SUPPORT CASES, for instance, once an issue with the software is confirmed, a ticket is typically created, and the outflow of the discussion would be captured there plus any other relevant information. In the case of discussions that started in relation to the previous topic, the team was not even actively working on them. I only observed one unplanned topic for which important information to improve their practice may not have been captured (T_{10}). I discuss this topic in the next section together with other discussions about PRACTICES AND PROCESS for which information was not captured.

Topics that addressed the discussion of internal processes. During meeting eight, the participants discussed what aspects should be considered to define the priority of tickets (T_{22}). Moreover, they introduced an artifact they called "the quadrant" that was supposed to help the team with this task (T_{25}). The motivation behind these discussions was an upcoming triaging session in which they wanted to determine the priority and value of a long list of tickets. The participants brainstormed some ideas towards figuring out how to measure the value of tickets (T_{22}) and how to use “the quadrant” to make the process more systematic (T_{25}). However, the ideas to improve the process were not captured. As another example, during T_{10}, a post-mortem discussion about a connectivity issue with docker containers that troubled the team for a while was conducted. Important information about how to prevent and fix this issue (PRACTICE AND PROCESS) was provided. Interestingly, the participant who raised the topic was not sharing their screen at that time. In this case, I cannot confirm information was not captured (P2 might have captured outflow offline). However, the typical practice of sharing his screen and documenting outflow collaboratively was not followed.

Topics in which important feedback was provided and the developers were the tool drivers. As a first example, in T_{17}, a developer shared their concerns about how to design a new functionality for which the implementation goal and some early ideas had been previously documented. The participant shared his screen and showed the existing information. P3 then chimed in to share some concerns to consider in the design, though the developer did not capture these suggestions in the artifact shared on screen. As a second example, during T_{16}, a developer presented a template to capture socio-technical documentation. The developer shared his screen to show the template, then the key internal stakeholders provided feedback, which was also not captured at that time. As a third example, during T_7 a developer requested help with the design in performance testing for a feature he was soon to implement. The participant shared a document with all the ideas and information they had about the feature. The discussion then centered on providing feedback about it, with the architects proposing some scenarios to consider for testing, as well as some input boundaries to keep in mind. Interestingly, these ideas were not captured in the design document shared on screen.

A common factor among these discussions was that the tool drivers were all developers. While we cannot confirm information was not captured (they could have captured information on a physical notebook), it is interesting the developers did not follow the somewhat established practice of sharing their screen during these discussions. For future research, it would be worthwhile to investigate why they did not share their screen as other participants did most of the time, it would also be worthwhile to confirm whether information was captured, and if it was not captured, what was the reason.

Overall, it seems that:

Observation 25: Not every discussion generates outflow that needs to be captured.

For most of the topics in which discussion outflow was not captured it does not seem to be that harmful (21 out of 27). During discussions that seek to simply pass on information or coordinate how they will work that day, new design information was not generated. Thus, there was no need to capture something. For unplanned discussions such as SUPPORT CASES, documenting information seemed premature because many things were still inaccurate. Moreover, topics that were spontaneously brought up because of a certain relation with the previous topic discussed, were not things the team was actively working on at that moment. Thus, capturing outflow could be premature as well. Finally, for some of the topics reviewed during the triaging session, only metadata was set.

Overall, it seems that important information may not have been captured for only six topics (T_7, T_{10}, T_{16}, T_{17}, T_{22}, T_{25}). Interestingly, these discussions possessed at least one the next characteristics: (1) the topics addressed the discussion of internal processes (T_{10}, T_{16}, T_{22}, T_{25}), and/or (2) the developers were the tool drivers (T_7, T_{16}, T_{17},). An interesting future research direction in this regard would be checking with the participants whether information was captured offline, and if it was not, whether they believe some information from these discussions should have been captured.

5.5 Kinds of discussion outflow captured in relation to the work done and the discussions held

In this section, I present the results of the analysis conducted to answer RQ 3.3 and RQ 3.4. For RQ 3.3: Are certain kinds of discussion outflow captured while discussing certain kinds of work? I first placed the kinds of information captured in the context of the different kinds of work (as determined in Chapter 3, Section 3.5) that participants discussed across the ten meetings. Table 24 shows the kinds of discussion outflow captured per kind of work. The kinds of information appear grouped by meta-category (DESIGN INFORMATION, TEAM PROCESS, COORDINATION, PROJECT MANAGEMENT), and the kinds of work appear by the maintenance dimensions of Swanson (as introduced in Section 2.1).

First, note that information was not captured during PERFORMANCE discussions (e.g., T_2), and neither was it captured while discussing SUPPORT CASES (e.g., T_5, T_{15}, T_{18}, T_{40}). A common characteristic among topics that addressed these two kinds of work is that participants spontaneously requested to discuss these topics during the meetings. These discussions were not part of the meeting agenda, and formal documentation about them (e.g., tickets, a design in a wiki page) did not exist, so there was no apparent base to which to attach notes. Typically, too, verbal outcomes were expressed to be followed up by the

Table 24. Kinds of discussion outflow captured per kind of work.

		ADAPTATIVE (39)			CORRECTIVE (0)	PERFECTIVE (42)		N/A (86)			
Meta-category	Category	ARCHITECTURAL RE-DESIGN	INTEGRATION OF NEW FUNCTIONALITY	PERFORMANCE	SUPPORT CASE	INFRASTRUCTURE	TESTING	ADMINISTRATIVE TASK	COORDINATION AND PLANNING	PRACTICE AND PROCESS	All types of work
DESIGN INFORMATION	IDEA/ALTERNATIVE	13				6	2			15	36
	THINGS TO KEEP IN MIND		18								18
	PROBLEM BACKGROUND/CODE STATE	3				3	1			7	14
	IMPLEMENTATION ROADMAP	1				1	7		2		11
	RATIONALE					6	1			2	9
	ISSUE/TICKET HIGH LEVEL DESCRIPTION	3						2			5
	IMPLEMENTATION GOAL / SCOPE	1				3					4
	REQUIREMENT									3	3
TEAM PROCESS	PLAN OF ACTION FOR A CERTAIN SITUATION						6	1		16	23
	SITUATION'S BACKGROUND / STATUS							7		2	9
	ADMINISTRATIVE DECISION							2			2
	BEST PRACTICE									2	2
GENERAL	DISCUSSION ITEMS							2	3	14	19
	ACTION ITEMS						1	2	2	2	7
PROJECT MANAGEMENT	SCHEDULING ESTIMATE						3				3
	IMPACT ESTIMATE					2					2
	Total	21	18	0	0	21	21	16	7	63	167

developers handling the case, who in the case of confirming an issue with the software exists, would create a ticket with all the relevant information (as explained in Section 5.4). While these results do not necessarily mean information would never be captured when the team discusses SUPPORT CASES or PERFORMANCE, it highlights that:

Observation 26: It is unlikely that the team captures information as the outflow of maintenance topics that are brought impromptu to the conversation.

Another noteworthy observation stems from analyzing the kinds of information in terms of maintenance versus non-maintenance work. For instance, note that design feedback (THINGS TO KEEP IN MIND), the goal and scope of code to be developed (IMPLEMENTATION GOAL/SCOPE), as well as estimates of how much value a new feature or change may have in the eye of the customer (IMPACT ESTIMATE) or how long it would take to the team to develop it (SCHEDULING ESTIMATE) were only captured during maintenance activities. At the other end of the spectrum, however, most TEAM PROCESS information (SITUATION'S BACKGROUND STATUS, ADMINISTRATIVE DECISION, BEST PRACTICE) was only captured during non-related maintenance activities.

As part of this analysis, I also note that the topics to be discussed (DISCUSSION ITEMS) were captured for all non-maintenance categories (PRACTICE AND PROCESS, ADMINISTRATIVE TASK, COORDINATION AND PLANNING). In contrast, for maintenance related discussions, the topics to be discussed (DISCUSSION ITEMS) were not captured before the meeting.

Observation 27: Certain kinds of discussion outflow are more likely to be captured when maintenance work is addressed, while others are more likely to be captured when non-maintenance activities are addressed.

While these results do not necessarily mean feedback (THINGS TO KEEP IN MIND) may never be captured during non-maintenance discussions, or that TEAM PROCESS information may not be captured while discussing a maintenance topic, they do highlight certain kinds of information are typically the outflow of certain maintenance or non-maintenance activities.

I also placed the kinds of information captured in the context of the purpose of each discussion held (as determined in Chapter 3) with the goal of answering RQ 3.4: Are certain kinds of discussion outflow captured for discussions with a particular purpose? Table 25 shows the results of this analysis. It presents the kinds of information captured (grouped by a meta-category) per discussion purpose. The squares over the table identify the meta-categories of information, specifically DESIGN and TEAM PROCESS information, that were often captured as part of discussions that had a certain purpose.

The first cluster (located on the top-left side of the table) shows that most DESIGN INFORMATION was captured during discussions with the purpose of DEVISING A SOLUTION, AUTOMATING ACTIVITIES, TRIAGING A TICKET, or REVIEWING A DESIGN

Table 25. Kinds of discussion outflow captured per discussion purpose.

Meta-category	Category	DEVISING A SOLUTION	AUTOMATING ACTIVITIES	TRIAGING A TICKET	REVIEWING A DESIGN PROPOSAL	PERFORMING A POST-MORTEM	MANAGING ACCOUNTS	CLARIFYING A MISUNDERSTANDING	DEFINING INTERNAL PRACTICES	MANAGING COMPUTATIONAL RESOURCES	PLANNING A FUTURE MEETING AGENDA	Al discussion purposes
DESIGN INFORMATION	IDEA/ALTERNATIVE	15	15	6								36
	THINGS TO KEEP IN MIND				18							18
	PROBLEM BACKGROUND/CODE STATE	4	7	3								14
	IMPLEMENTATION ROADMAP	8		1					2			11
	RATIONALE	1	2	6								9
	ISSUE/TICKET HIGH LEVEL DESCRIPTION	3								2		5
	IMPLEMENTATION GOAL / SCOPE	1		3								4
	REQUIREMENT		3									3
TEAM PROCESS	PLAN OF ACTION FOR CERTAIN SITUATION					16	1	6				23
	SITUATION'S BACKGROUND / STATUS					2	7					9
	ADMINISTRATIVE DECISION						2					2
	BEST PRACTICE					2						2
GENERAL	DISCUSSION ITEMS		5			9	2		2		1	19
	ACTION ITEMS	1	1			1	2		2			7
PROJECT MANAGEMENT	SCHEDULING ESTIMATE	3										3
	IMPACT ESTIMATE			2								2
	Total	36	33	21	18	30	14	6	6	2	1	167

PROPOSAL. The second cluster (located in the middle of the table) shows that most TEAM PROCESS information was captured during discussions with the purpose of PERFORMING A POST-MORTEM, MANAGING ACCOUNTS, or CLARIFYING A MISUNDERSTANDING.

An observation that stems from these results is that:

Observation 28: Certain kinds of information are more likely to be the outflow of discussions with a certain purpose.

While capturing DESIGN INFORMATION was highly relevant for certain discussion purposes (DEVISING A SOLUTION, AUTOMATING ACTIVITIES, TRIAGING A TICKET, REVIEWING A DESIGN PROPOSAL), TEAM PROCESS information was relevant for others (PERFORMING A POST-MORTEM, MANAGING ACCOUNTS, CLARIFYING MISUNDERSTANDINGS). Moreover, note that some kinds of information were only captured for only one discussion purpose (the shading identifies them). For instance, THINGS TO KEEP IN MIND were only captured during discussions that addressed the INTEGRATION OF A NEW FUNCTIONALITY, BEST PRACTICES were only captured while PERFORMING A POST-MORTEM, REQUIREMENTS were only captured during discussions to AUTOMATE ACTIVITIES, and ADMINISTRATIVE DECISIONS were only captured during discussions related to MANAGING ACCOUNTS.

5.6 Tools and artifacts used to capture discussion outflow

In this section, I present the results of the analysis conducted to answer RQ 4 and RQ 4.1. For RQ 4: What tools do participants use to capture discussion outflow in RSSMDMs? Table 26 shows the variety of tools that the participants of the meetings use to capture information. The second and third columns show each tool's name and description respectively. The first column shows whether the tool is exclusively used for software development, or if it is a tool also used in other domains. This categorization was also used to classify the tools that participants used to share prior information (see Chapter 4).

Table 26. Tools in which discussion outflow was captured.

Category	Tool	Description
DEVELOPMENT	CONFLUENCE	Project management dashboard for project planning and document sharing.
	JIRA	Issue tracking system used to record, manage, and prioritize problem tickets.
	VSCODE	Visual Studio Code is a code editor for building and debugging modern web and cloud applications.

Various observations stem from these results in the context of the tools used to share prior information (see Chapter 4). First, note that more tools were used to share information than the ones used to capture it. Moreover, while the purpose of tools to share prior information was broad (DEVELOPMENT, GENERAL PURPOSE, PROPRIETARY), only DEVELOPMENT tools were used to capture discussion outflow: a knowledge repository (Confluence), an issue tracking system (Jira), and a code editor (VSCODE). The first two tools (Confluence and Jira) were also used to share prior information. Even though much less kinds of tools were used to capture discussion outflow, than the ones used to share information, this finding was not surprising, since many of the other tools used to share information present dynamic information (e.g., AWS, monitoring tools).

Observation 29: Few tools were used to capture information and all of them were development-oriented tools.

A second important observation stems from analyzing which DEVELOPMENT tools were used to capture discussion outflow. While drawing and diagraming tools (e.g., whiteboards [13], [85], [86], CASE tools [202]–[204]) have been historically reported as beneficial during design meetings to, for instance, draw early sketches or create UML diagrams, the participants did not use tools of this kind to capture information. During these meetings, participants are more likely to capture information in tools such as knowledge repositories (e.g., [165], [167]) or issue tracking systems [212], even during discussions in which new DESIGN INFORMATION is considerably produced.

Observation 30: Typical drawing and diagramming tools were not used to capture information during the meetings.

I also investigated what specific artifacts the participants created with these tools to answer RQ 4.1: Are specific artifacts used to capture discussion outflow? Table 27 presents the kinds of artifacts the participants created with each of the tools they used in order to capture discussion outflow.

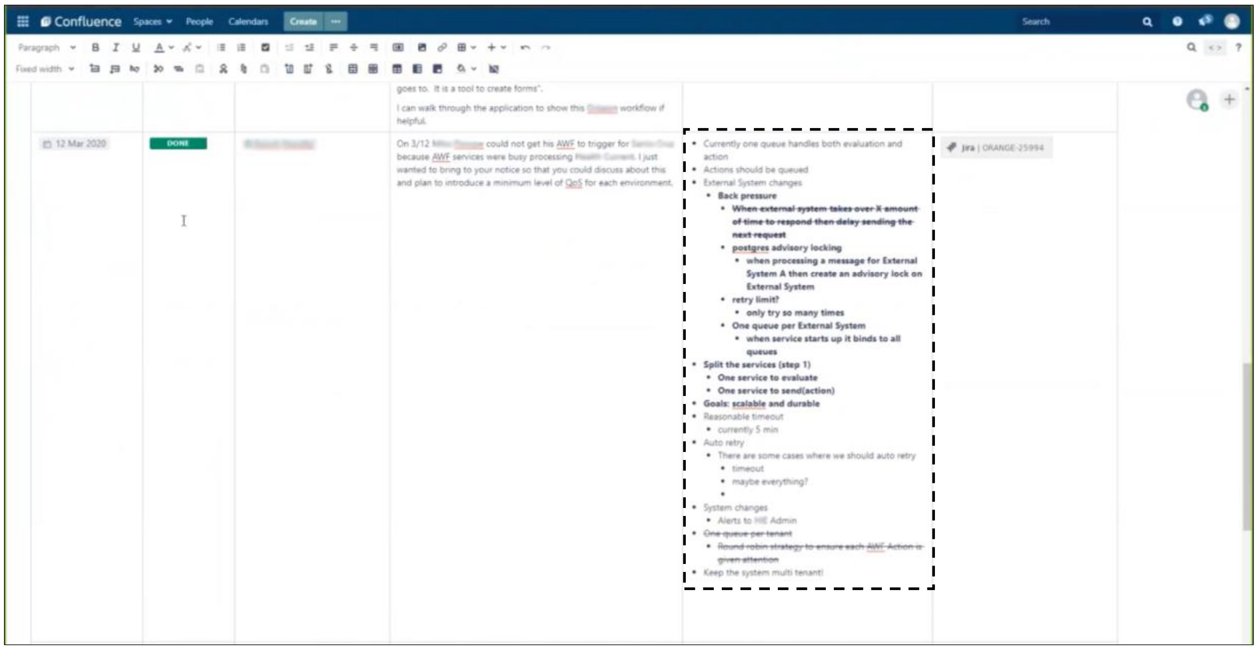
Table 27. Artifacts created to capture information for each tool used.

Tool	Artifact	Description
CONFLUENCE	ASK AN ARCHITECT WIKI PAGE	A wiki page with a pre-loaded table template to which the team, and occasionally employees from other teams in the organization can add to receive help from the architects (Figure 2 shows an example of what it looks like). Each row in this table contains relevant information for a topic that someone has requested to be discussed: a submission date, the topic status, who submits the topic, the description of the topic, and the outflow of discussing it once the topic discussion commences (e.g., decisions made, agreements, procedures to establish).
	MINUTES OF THE MEETING WIKI PAGE	A wiki page with a pre-loaded template to capture the minutes of the meeting. The template suggests titles and a placeholder for information that is typically capture during meetings (e.g., date, list of topics, action items).
	PLAYBOOK WIKI PAGE	A wiki page to document technical notes, procedures, or tutorials.
	WIKI PAGE	A wiki page without a pre-loaded template.
JIRA	TICKET	In the context of an issue tracking system (i.e., Jira), or any other service desk platform, a ticket is an event that must be investigated or a work item that must be addressed by the development team.
	ISSUE	In Jira Service Desk, Jira tickets entered by customers are referred to as “requests”, which the development team internally refers to as issues.
VSCODE	TEXT NOTE	Notes captured in a notepad app or lightweight code editor (not an IDE).

Observation 31: Participants use different artifacts to organize discussion outflow.

Observe that several different artifacts were used to capture information in Confluence (knowledge repository). During T_3 (the re-design of a major component of the architecture), for instance, information was captured in “Ask an Architect” (a Confluence wiki page with a pre-loaded template that the team itself had produced). Figure 10 shows

Figure 10. An "Ask an Architect" wiki page is one of the artifacts used in Confluence.


goes to. It is a tool to create forms".

I can walk through the application to show this [redacted] workflow if helpful.

12 Mar 2020 | DONE | [redacted]

On 3/12 [redacted] could not get his AWF to trigger for [redacted] because AWF services were busy processing [redacted]. I just wanted to bring to your notice so that you could discuss about this and plan to introduce a minimum level of QoS for each environment.

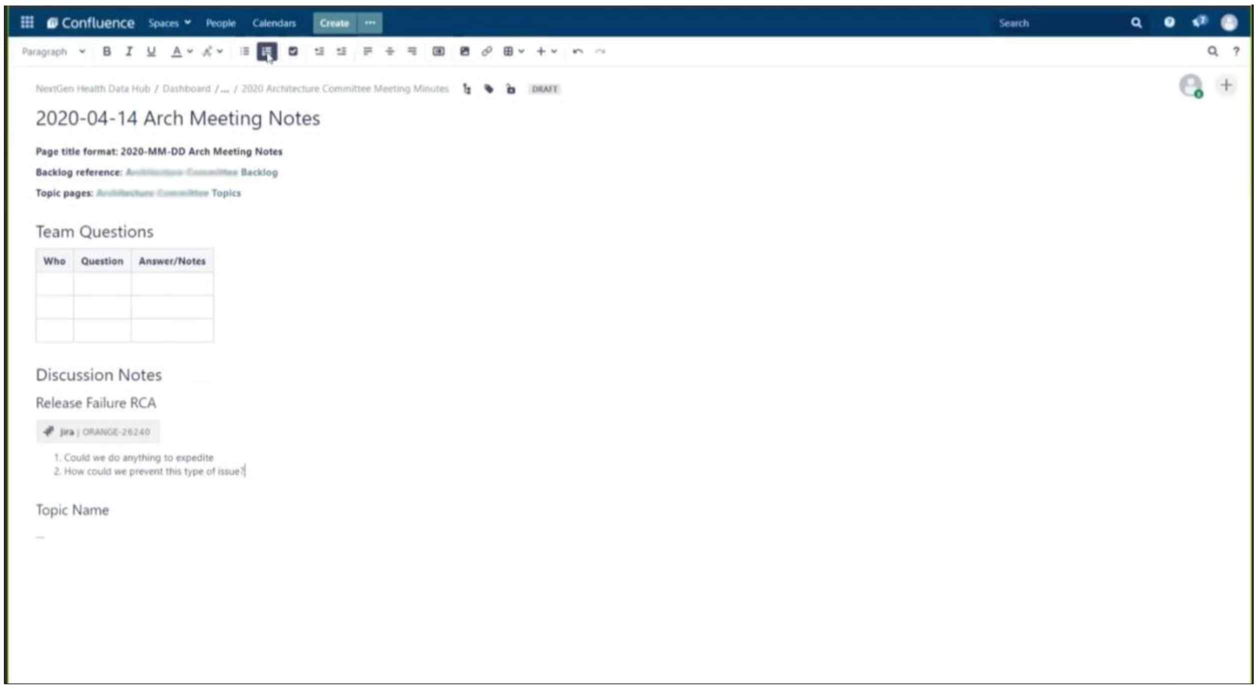
- Currently one queue handles both evaluation and action
- Actions should be queued
- External System changes
  - **Back pressure**
    - ~~**When external system takes over X amount of time to respond then delay sending the next request**~~
    - **postgres advisory locking**
      - **when processing a message for External System A then create an advisory lock on External System**
    - **retry limit?**
      - **only try so many times**
    - **One queue per External System**
      - **when service starts up it binds to all queues**
- **Split the services (step 1)**
  - **One service to evaluate**
  - **One service to send(action)**
- **Goals: scalable and durable**
- Reasonable timeout
  - currently 5 min
- Auto retry
  - There are some cases where we should auto retry
    - timeout
    - maybe everything?
    -
- System changes
  - Alerts to [redacted] Admin
- ~~One queue per tenant~~
  - ~~Round robin strategy to ensure each AWF Action is given attention~~
- Keep the system multi tenant!

Jira | ORANGE-25994

the look and feel of this wiki template, the information inside the dashed box is what was captured during the meeting.

Another wiki page template commonly used was the one to capture "the minutes of the meeting". Figure 11 shows an artifact of this kind being used to capture information

Figure 11. A "minutes of the meeting" wiki page is one of the artifacts used in Confluence.


NextGen Health Data Hub / Dashboard / ... / 2020 Architecture Committee Meeting Minutes DRAFT

2020-04-14 Arch Meeting Notes

Page title format: 2020-MM-DD Arch Meeting Notes

Backlog reference: [redacted] **Backlog**

Topic pages: [redacted] **Topics**

Team Questions

Who	Question	Answer/Notes

Discussion Notes

Release Failure RCA

Jira | ORANGE-26240

1. Could we do anything to expedite
2. How could we prevent this type of issue?

Topic Name

—

during a discussion in which the architects walked through an issue to educate the team on how to avoid it in future (T_9). During this discussion, participants also created a “playbook” to capture information on how to properly use database cursors, which was the reason behind the issue reported.

5.7 Tools used more frequently to capture discussion outflow

In this section, I present the results of the analysis conducted to answer RQ 4.2 Are certain tools used more often to capture discussion outflow? Table 28 shows the tools in which participants captured discussion outflow per topic (discussion outflow was only captured during 18 topics out of 45). The first column shows the “Topic Id” and the next columns, in pairs of two (“#Topics”, “# Times”), show the topics in which that tool was used and how many times respectively. The results appear organized by topic. Shading highlights all the tools used per topic.

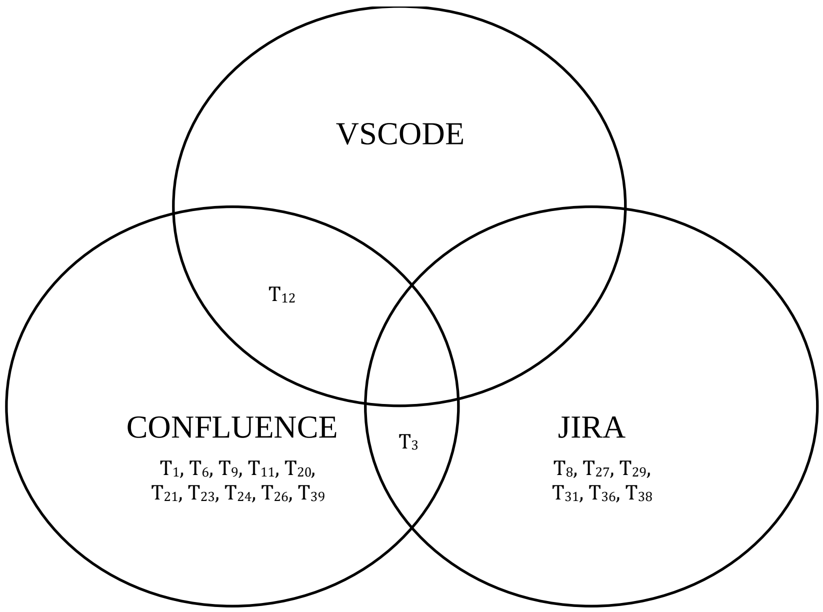
Participants used Confluence to capture information during 12 discussions, Jira during seven, and VSCODE for only one. Note that, except for T_3 and T_{12}, one tool was

Table 28. Tools used to capture outflow per topic.

	CONFLUENCE		JIRA		VSCODE	
Topic Id	# Topics	# Times	# Topics	# Times	# Topics	# Times
T_{1}	Y	14	N	0	N	0
T_{3}	Y	18	Y	3	N	0
T_{6}	Y	15	N	0	N	0
T_{8}	N	0	Y	6	N	0
T_{9}	Y	21	N	0	N	0
T_{11}	Y	27	N	0	N	0
T_{12}	Y	1	N	0	Y	1
T_{20}	Y	6	N	0	N	0
T_{21}	Y	9	N	0	N	0
T_{23}	Y	12	N	0	N	0
T_{24}	Y	6	N	0	N	0
T_{26}	Y	1	N	0	N	0
T_{27}	N	0	Y	8	N	0
T_{29}	N	0	Y	1	N	0
T_{31}	N	0	Y	2	N	0
T_{36}	N	0	Y	1	N	0
T_{38}	N	0	Y	9	N	0
T_{39}	Y	6	N	0	N	0
Total	12	136	7	30	1	1

typically sufficient to capture all the outflow generated per discussion. Figure 12 shows this distribution more clearly. Note that most topics appear in only one circle (which represents a tool), rather than at the intersection of two (when two or more tools were used).

Figure 12. Distribution of tools used per topic.

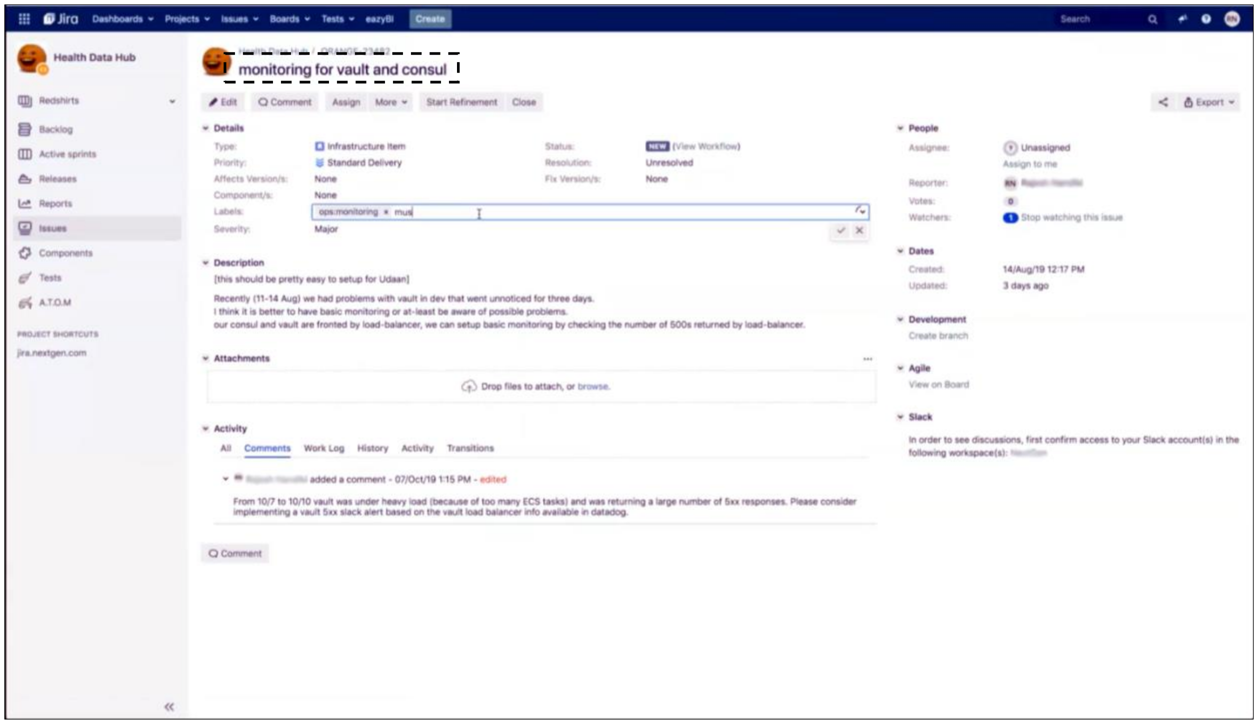

VSCODE
T_{12}
CONFLUENCE
T_1, T_6, T_9, T_{11}, T_{20}, T_{21}, T_{23}, T_{24}, T_{26}, T_{39}
T_3
JIRA
T_8, T_{27}, T_{29}, T_{31}, T_{36}, T_{38}

Confluence was used in conversations that had various purposes. For instance, it was used during discussions in which participants addressed the AUTOMATION OF ACTIVITIES (T_1, T_{11}, T_{20}) or discussions in which they PERFORMED A POST-MORTEM (T_9, T_{21}). As another example, it was also used to capture information while DEVISING A SOLUTION (T_6), REVIEWING A DESIGN PROPOSAL (T_{23}), DEFINING INTERNAL PRACTICES (T_{24}), PLANNING A FUTURE MEETING AGENDA ($T_{26,}$), or while CLARIFYING A MISUNDERSTANDING (T_{39}).

In contrast, most of the topics in which Jira was used (e.g., T_{27}, T_{29}, T_{31}, T_{36}, T_{38}) had the purpose of TRIAGING TICKETS. During these discussions, participants opened tickets previously created with the goal to assess their scope or impact, or to schedule them into a sprint. As one example, in T_{29}, the discussion's outflow was re-defining the scope of the ticket from developing a monitoring dashboard to developing an alert. The scope in this case was re-defined in the ticket's title. Figure 13 shows the original title of the ticket (*"monitoring for vault and consult"*), and Figure 14 shows the re-defined title (*"alert for vault and consult 5xx errors"*).

Both Confluence and Jira, were only used together to capture information during T3 (the architectural re-design of a component). During this discussion, the participants captured the problem's description (PROBLEM/BACKGROUND STATUS), various ideas to re-design the component in question (IDEA/ALTERNATIVE), and a plan to implement this major architectural change (IMPLEMENTATION ROADMAP). All this information was captured in Confluence. Figure 15 (left arrow) shows the sections of the wiki page (in "Ask an Architect") that the participants used to capture it. The information inside the dashed box is what was captured as outflow of this discussion.

Figure 13. Original information in the Jira ticket before refinement.



Health Data Hub / ORANGE-23482

monitoring for vault and consul

Details

Type: Infrastructure Item
Priority: Standard Delivery
Affects Version/s: None
Component/s: None
Labels: ops:monitoring × mus
Severity: Major
Status: NEW (View Workflow)
Resolution: Unresolved
Fix Version/s: None

Description

[this should be pretty easy to setup for Udaan]

Recently (11-14 Aug) we had problems with vault in dev that went unnoticed for three days.
I think it is better to have basic monitoring or at-least be aware of possible problems.
our consul and vault are fronted by load-balancer, we can setup basic monitoring by checking the number of 500s returned by load-balancer.

Attachments

Activity

[redacted] added a comment - 07/Oct/19 1:15 PM - edited

From 10/7 to 10/10 vault was under heavy load (because of too many ECS tasks) and was returning a large number of 5xx responses. Please consider implementing a vault 5xx slack alert based on the vault load balancer info available in datadog.

People

Assignee: Unassigned
Assign to me
Reporter: RN [redacted]
Votes: 0
Watchers: 1 Stop watching this issue

Dates

Created: 14/Aug/19 12:17 PM
Updated: 3 days ago

Development

Create branch

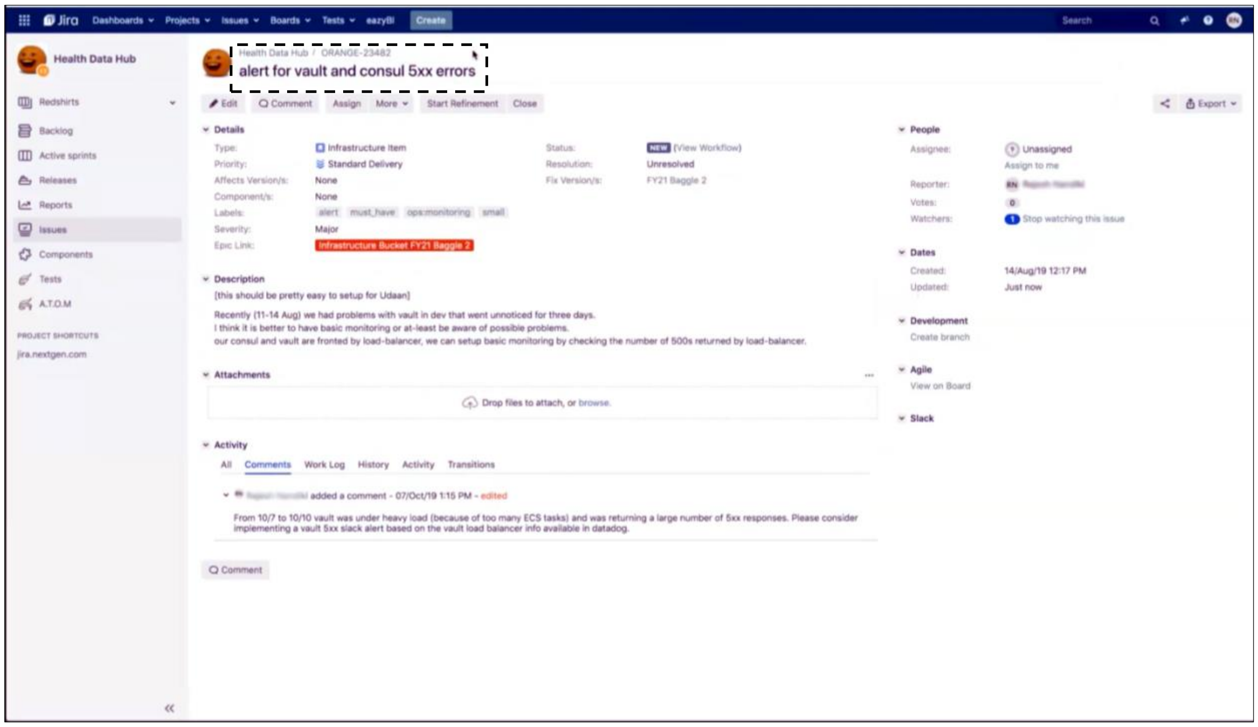
Agile

View on Board

Slack

In order to see discussions, first confirm access to your Slack account(s) in the following workspace(s): [redacted]

Figure 14. Redefined information in the Jira ticket.



Health Data Hub / ORANGE-23482

alert for vault and consul 5xx errors

Details

Type: Infrastructure Item
Priority: Standard Delivery
Affects Version/s: None
Component/s: None
Labels: alert must_have ops:monitoring small
Severity: Major
Epic Link: Infrastructure Bucket FY21 Baggle 2

Status: NEW (View Workflow)
Resolution: Unresolved
Fix Version/s: FY21 Baggle 2

Description

[this should be pretty easy to setup for Udaan]

Recently (11-14 Aug) we had problems with vault in dev that went unnoticed for three days.
I think it is better to have basic monitoring or at-least be aware of possible problems.
our consul and vault are fronted by load-balancer, we can setup basic monitoring by checking the number of 500s returned by load-balancer.

Attachments

Drop files to attach, or browse.

Activity

All Comments Work Log History Activity Transitions

[redacted] added a comment - 07/Oct/19 1:15 PM - edited

From 10/7 to 10/10 vault was under heavy load (because of too many ECS tasks) and was returning a large number of 5xx responses. Please consider implementing a vault 5xx slack alert based on the vault load balancer info available in datadog.

Comment

People

Assignee: Unassigned
Assign to me
Reporter: [redacted]
Votes: 0
Watchers: 1 Stop watching this issue

Dates

Created: 14/Aug/19 12:17 PM
Updated: Just now

Development

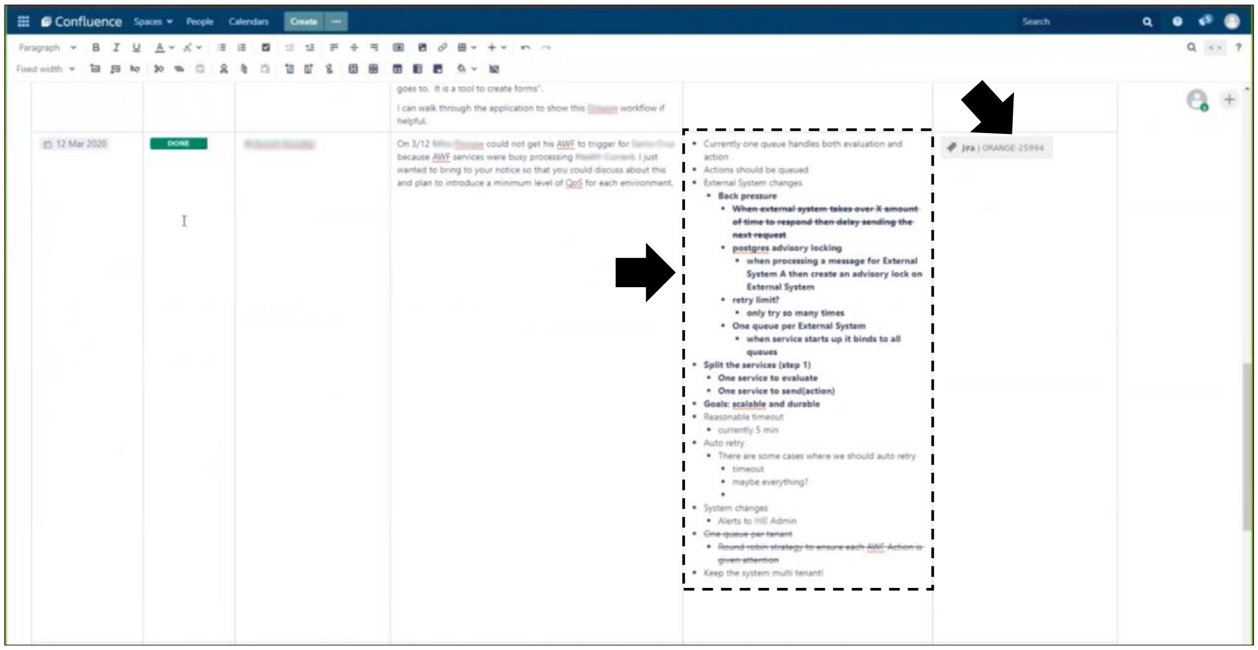
Create branch

Agile

View on Board

Slack

Figure 15. Discussion outflow captured in "Ask an Architect".



goes to. It is a tool to create forms".

I can walk through the application to show this [redacted] workflow if helpful.

12 Mar 2020 DONE [redacted]

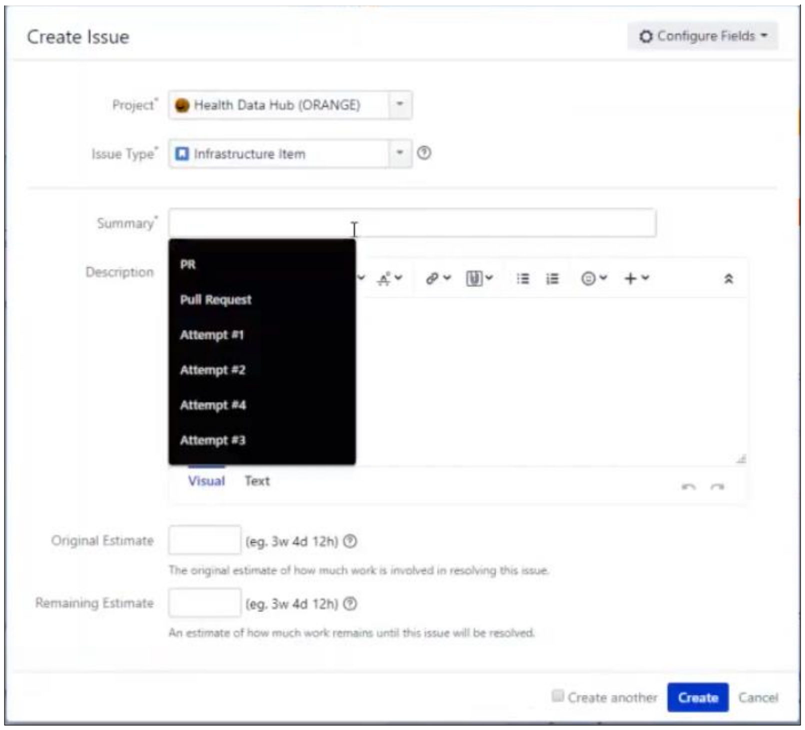
On 3/12 [redacted] could not get his AWF to trigger for [redacted] because AWF services were busy processing [redacted]. I just wanted to bring to your notice so that you could discuss about this and plan to introduce a minimum level of QoS for each environment.

- Currently one queue handles both evaluation and action
- Actions should be queued
- External System changes
  - **Back pressure**
    - ~~**When external system takes over X amount of time to respond then delay sending the next request**~~
    - **postgres advisory locking**
      - **when processing a message for External System A then create an advisory lock on External System**
    - **retry limit?**
      - **only try so many times**
    - **One queue per External System**
      - **when service starts up it binds to all queues**
- **Split the services (step 1)**
  - **One service to evaluate**
  - **One service to send(action)**
- **Goals: scalable and durable**
- Reasonable timeout
  - currently 5 min
- Auto retry
  - There are some cases where we should auto retry
    - timeout
    - maybe everything?
    -
- System changes
  - Alerts to [redacted] Admin
- ~~One queue per tenant~~
  - ~~Round robin strategy to ensure each AWF Action is given attention~~
- Keep the system multi tenant!

Jira | ORANGE-25994

The participants also created a Jira ticket during this conversation and captured the HIGH-LEVEL DESCRIPTION of the first activity they had decided to do, which was implementing a proof of concept (PoC). Figure 16 shows the ticket's creation, and Figure 17 shows the final state of this ticket, which contains a high-level description of the work to do (ISSUE/TICKET HIGH LEVEL DESCRIPTION).

Figure 16. Ticket created during the meeting.



Create Issue

Configure Fields

Project* Health Data Hub (ORANGE)

Issue Type* Infrastructure Item

Summary*

Description

PR
Pull Request
Attempt #1
Attempt #2
Attempt #4
Attempt #3

Visual Text

Original Estimate (eg. 3w 4d 12h)
The original estimate of how much work is involved in resolving this issue.

Remaining Estimate (eg. 3w 4d 12h)
An estimate of how much work remains until this issue will be resolved.

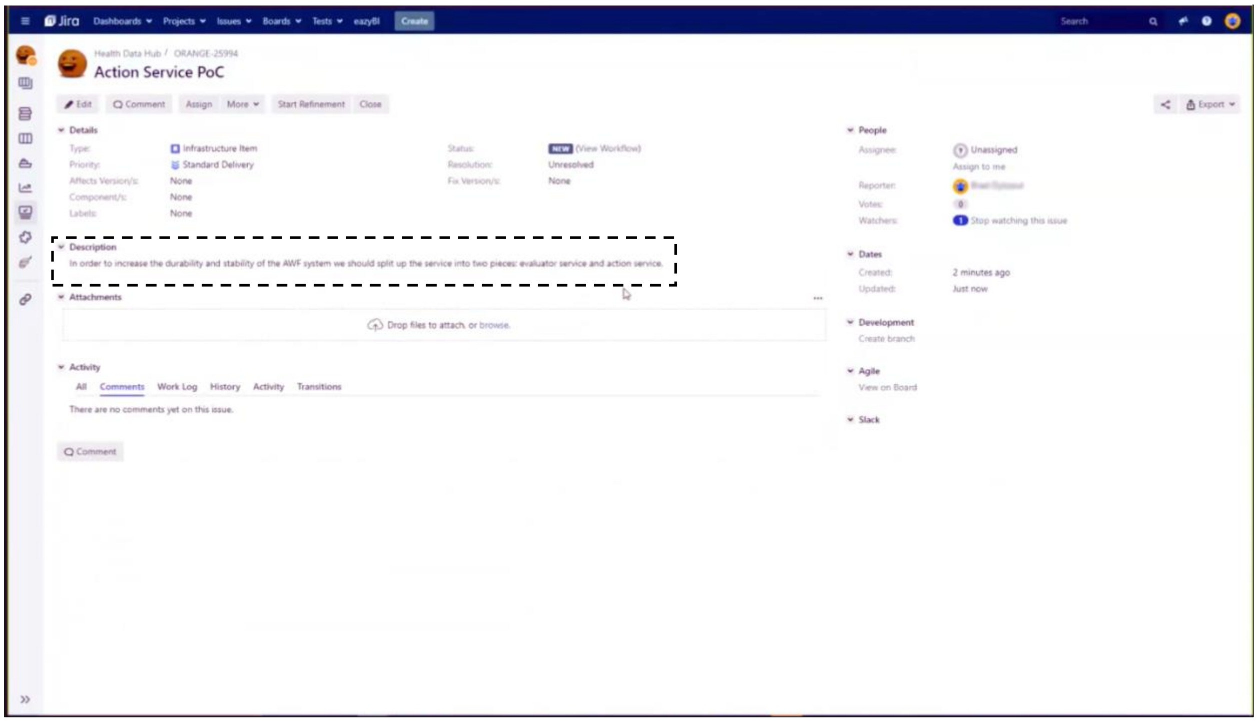
Create another Create Cancel

Figure 17 shows the ticket updated (the dashed box indicates the area in which information was captured). The excerpt below shows the ticket's description that the participants captured in the ticket.

> *"In order to increase the durability and stability of the AWF system we should split up the service into two pieces: evaluator service and action service"*

Note that even though the description is brief, it contains what this development will afford them ("*to increase the durability and stability of the AWF system*") as well as the summary of

Figure 17. Ticket after the high-level description was added.



Health Data Hub / ORANGE-25994

Action Service PoC

Details

Type:	Infrastructure Item	Status:	NEW (View Workflow)
Priority:	Standard Delivery	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		

Description

In order to increase the durability and stability of the AWF system we should split up the service into two pieces: evaluator service and action service.

Attachments

Activity

There are no comments yet on this issue.

People

Assignee:	Unassigned Assign to me
Reporter:	[redacted]
Votes:	0
Watchers:	1 Stop watching this issue

Dates

Created:	2 minutes ago
Updated:	Just now

Development

Create branch

Agile

View on Board

Slack

the work that needs to be done ("*we should split up service into two pieces: evaluator service and action service*"). A link to this ticket was also added to "Ask an Architect" to keep both artifacts connected (Figure 15 , right arrow identifies the link that was created).

The other topic in which more than one tool was used was T12. While most information was captured in Confluence, VSCODE was briefly used to fix up some text copied from an AWS dashboard. They then copied the text with the format desired to Confluence. Even though I cannot confirm it, it seems that the information captured in the code editor was not captured for a long term, but simply to fix its format. VSCODE was used only this time, nothing else was done with it.

Overall, these results show that:

Observation 32: Knowledge repositories are the tool that is used most often to capture information during RSSMDMs.

Three situations represent an exception: (1) discussions in which TICKET TRIAGING is the main activity, (2) discussions in which a plan to implement a major update (IMPLEMENTATION ROADMAP) is defined (T3) and tickets for some of the activities that this plan involves are created, and (3) discussions in which feedback (THINGS TO KEEP IN MIND) is captured as part of the same design artifact that is being presented (T8).

5.8 What do participants do with tools during the meeting?

In this section, I present the results of the analysis conducted to answer RQ 4.3: How do participants use tools during RSSMDMs? Table 29 shows all the actions that the two researchers observed the participants did with the tools. The first two actions, ADDING NEW INFORMATION and REFINING, are the ones that the two researchers who performed the coding considered as moments in which discussion outflow was captured. The rest are other actions the participants did with the tools in preparation for capturing information (SETTING UP), organizing the information that had been previously captured (REORGANIZING), classifying artifacts into bigger groups (SETTING METADATA), or linking different artifacts in which information had been captured (TRACING).

Table 29. The actions that the participants did with the tools.

Action	Description
ADDING NEW INFORMATION	Capturing brand-new information in a document (e.g., adding new information to an empty document, adding a new idea or thought to an existing document, adding a new paragraph in an existing ticket, etc.)
REFINING	Updating or altering existent information in the tool to make it more accurate (e.g., editing the content of an existent bullet point, updating the content of an existing paragraph).
SETTING UP	Opening or creating a specific file or document (e.g., ticket, wiki page) to capture information, or (re)defining the document's structure to capture information (e.g., changing table structure, adding bullets, titles). No meaningful content (discussion outflow) is captured.
REORGANIZING	Moving existing content without altering it (e.g., copy-pasting information from one section to another one, changing the spacing.)
TRACING	Connecting different documents via links (e.g., adding ticket numbers to the minutes of the meeting).
SETTING METADATA	Adding or changing the metadata of a document (e.g., tags, status, the date).

Table 30 shows the times each tool and respective artifact was used in a certain way (ACTION). The first and second columns show the tools and artifacts used, and the third and fourth columns show the times tools were used to capture information (ADDING NEW INFORMATION and REFINING); the fifth column shows the total times information was captured (167). From the sixth to the ninth columns other actions performed with the tools are shown (REORGANIZING, SETTING METADATA, SETTING UP, TRACING), their respective subtotal is shown in column tenth (65). The last column shows the total times tools were used (information captured plus other actions), which sums up a total of 232 times. Each cell shows the times a given tool and artifact was used in a particular way. Results are organized per tool and artifact used. Various examples of new information being added (ADDING INFORMATION) were presented in Section 5.7. In that section, I also mentioned that Jira was often used during TICKET TRIAGE sessions. Figure 14 in particular is an example of a piece of information being refined (participants updated the ticket's title as a result of re-defining their scope). In addition to ADDING NEW INFORMATION and REFINING existing information, during TICKET TRIAGE sessions, participants also SET

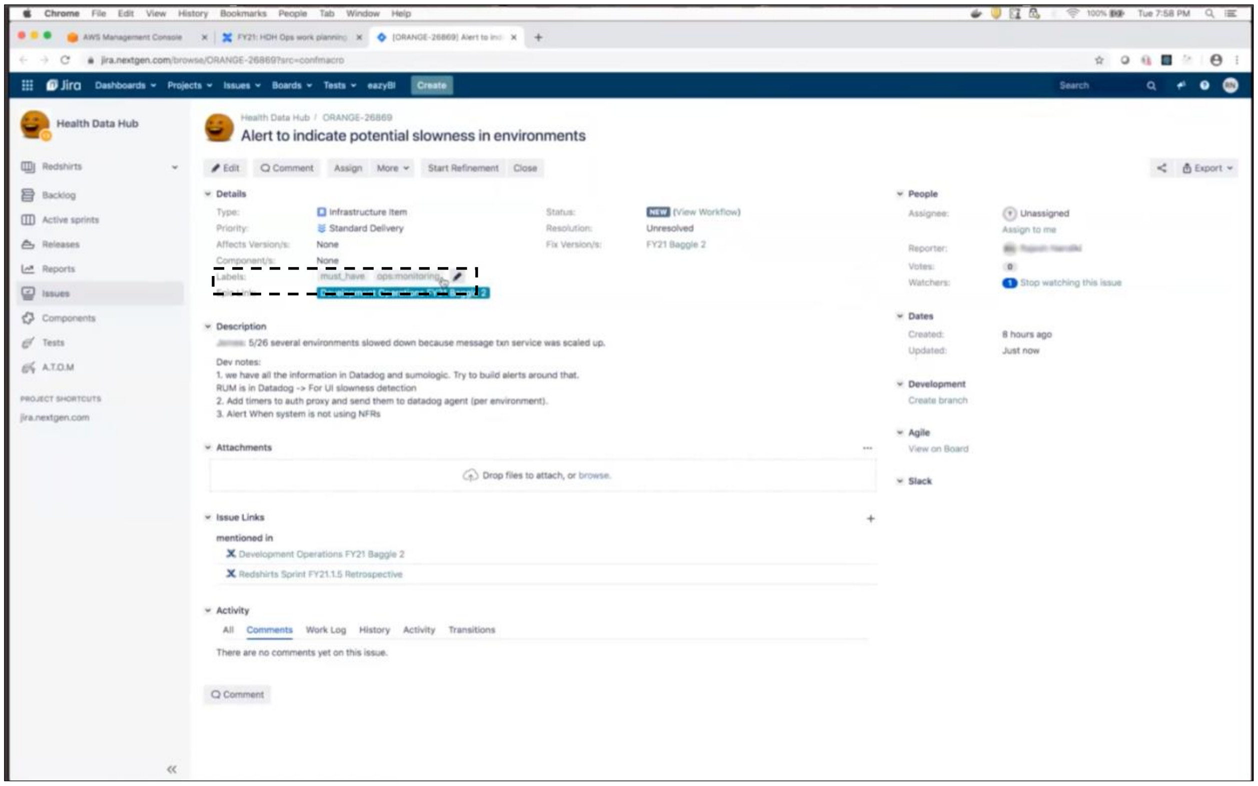
Table 30. The actions that the participants did with the tools.

Tool	Artifact	ADDING NEW INFORMATION	REFINING	Times tools were used to capture meaningful content	REORGANIZING	SETTING METADATA	SETTING UP	TRACING	Times tools were used in other ways	Total times a tool was used
CONFLUENCE	WIKI PAGE	48	7	55	7		7		14	69
	ASK AN ARCHITECT WIKI PAGE	36	4	40	1	5	2	2	10	50
	MINUTES OF THE MEETING WIKI PAGE	32	8	40	5		5	4	14	54
	PLAYBOOK WIKI PAGE		1	1			1		1	2
JIRA	ISSUE	15	4	19	2	10	4		16	35
	TICKET	11		11	3	4		3	10	21
VSCODE	TEXT NOTE	1		1					0	1
	Total	143	24	167	18	19	19	9	65	232

METADATA. For instance, they add labels to the tickets to classify them into specific buckets. Figure 18 shows an example of an issue the participants were about to classify. Note that the issue only has one label (the dashed box indicates). Figure 19 shows the label that the participants added to classify it. Note that the issue was classified into the bucket *"ops_monitoring"*. The dashed box in the picture indicates the field in which the label was added.

Metadata was also set in Confluence while capturing information in "Ask an Architect". Participants often changed the status of a question to "done" once they concluded its respective discussion. Figure 20 shows that the status of the question was "new" before closing the discussion (signaled by an arrow). Figure 21 shows how the status changed from "new" to "done" (signaled by an arrow).

Figure 18. An example in which the participants "set metadata" in Jira.


Chrome File Edit View History Bookmarks People Tab Window Help
Tue 7:58 PM
AWS Management Console
FY21: HDH Ops work planning
[ORANGE-26869] Alert to ind
jira.nextgen.com/browse/ORANGE-26869?src=confmacro
Jira Dashboards Projects Issues Boards Tests eazyBI Create
Search
Health Data Hub
Redshirts
Backlog
Active sprints
Releases
Reports
Issues
Components
Tests
A.T.O.M
PROJECT SHORTCUTS
jira.nextgen.com
Edit
Comment
Assign
More
Start Refinement
Close
Export

Health Data Hub / ORANGE-26869

Alert to indicate potential slowness in environments

Details

Type: Infrastructure Item
Priority: Standard Delivery
Affects Version/s: None
Component/s: None
Labels: must_have ops:monitoring
Status: NEW (View Workflow)
Resolution: Unresolved
Fix Version/s: FY21 Baggle 2

Description

[redacted] 5/26 several environments slowed down because message txn service was scaled up.

Dev notes:
1. we have all the information in Datadog and sumologic. Try to build alerts around that.
RUM is in Datadog -> For UI slowness detection
2. Add timers to auth proxy and send them to datadog agent (per environment).
3. Alert When system is not using NFRs

Attachments

Drop files to attach, or browse.

Issue Links

mentioned in
Development Operations FY21 Baggle 2
Redshirts Sprint FY21.1.5 Retrospective

Activity

All Comments Work Log History Activity Transitions

There are no comments yet on this issue.

Comment

People

Assignee: Unassigned
Assign to me
Reporter: [redacted]
Votes: 0
Watchers: 1 Stop watching this issue

Dates

Created: 8 hours ago
Updated: Just now

Development

Create branch

Agile

View on Board

Slack

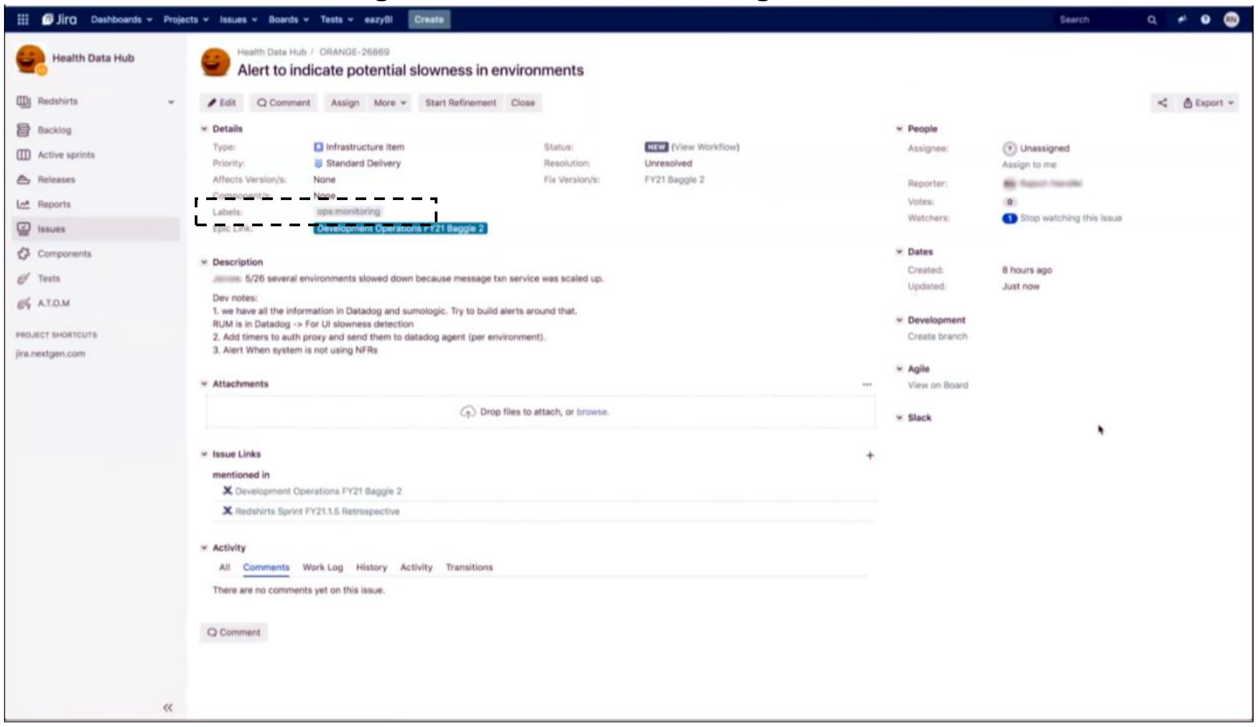


Figure 19. An issue before "setting metadata".

Health Data Hub / ORANGE-26869

Alert to indicate potential slowness in environments

Details

Type: Infrastructure Item
Priority: Standard Delivery
Affects Version/s: None
Components: None
Labels: ops:monitoring
Epic Link: Development Operations FY21 Baggie 2

Status: NEW (View Workflow)
Resolution: Unresolved
Fix Version/s: FY21 Baggie 2

Description

[redacted]: 5/26 several environments slowed down because message txn service was scaled up.

Dev notes:
1. we have all the information in Datadog and sumologic. Try to build alerts around that.
RUM is in Datadog -> For UI slowness detection
2. Add timers to auth proxy and send them to datadog agent (per environment).
3. Alert When system is not using NFRs

Attachments

Drop files to attach, or browse.

Issue Links

mentioned in
Development Operations FY21 Baggie 2
Redshirts Sprint FY21.1.5 Retrospective

Activity

All Comments Work Log History Activity Transitions

There are no comments yet on this issue.

Comment

People

Assignee: Unassigned
Assign to me
Reporter: [redacted]
Votes: 0
Watchers: 1 Stop watching this issue

Dates

Created: 8 hours ago
Updated: Just now

Development

Create branch

Agile

View on Board

Slack

In preparation to capture information before starting a discussion, participants used tools to SET UP artifacts in which information was going to be captured. For instance, they often made a copy of an empty wiki page with the "minutes of the meeting" template, which they slightly modified by adding empty bullets or fixing a table structure to capture

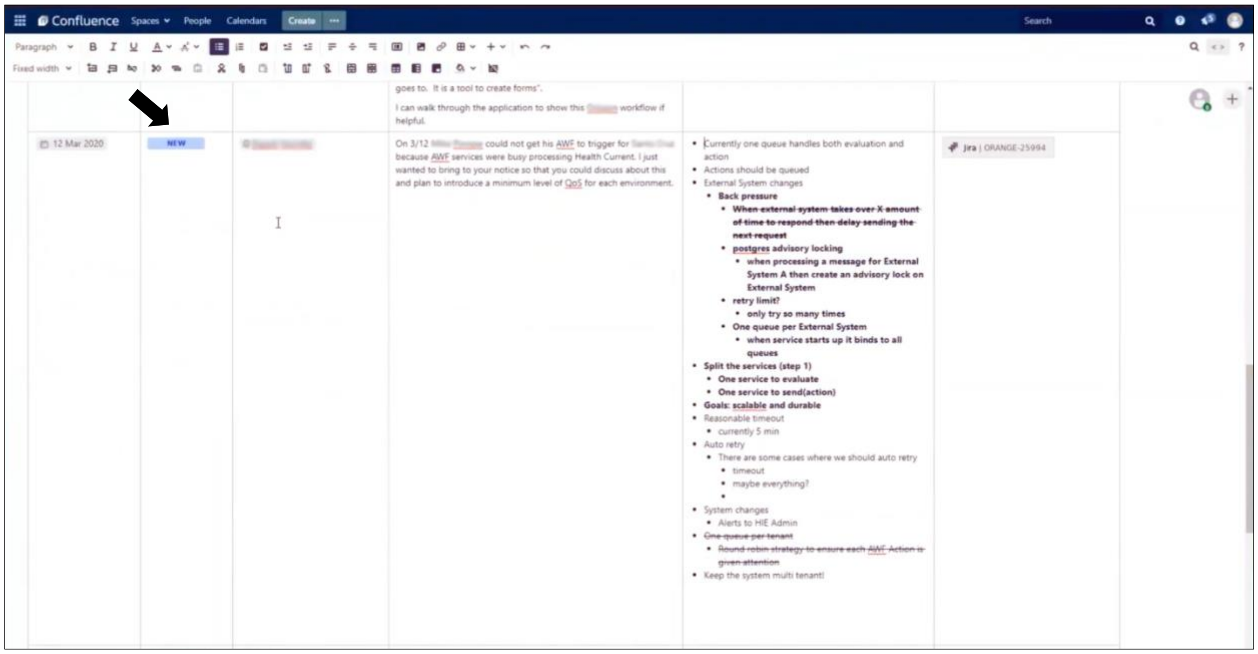


Figure 20. A question in Confluence which status is "new".

goes to. It is a tool to create forms".

I can walk through the application to show this [redacted] workflow if helpful.

12 Mar 2020

NEW

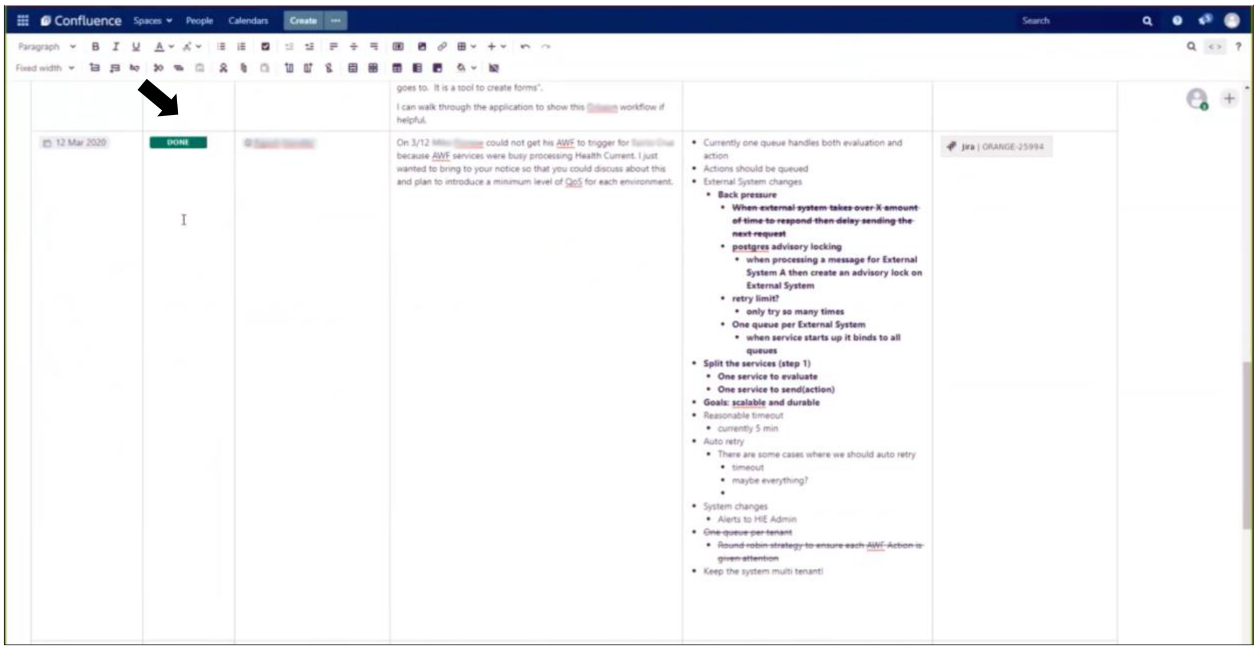
[redacted]

On 3/12 [redacted] could not get his AWF to trigger for [redacted] because AWF services were busy processing Health Current. I just wanted to bring to your notice so that you could discuss about this and plan to introduce a minimum level of QoS for each environment.

- Currently one queue handles both evaluation and action
- Actions should be queued
- External System changes
    - **Back pressure**
        - ~~**When external system takes over X amount of time to respond then delay sending the next request**~~
        - **postgres advisory locking**
            - **when processing a message for External System A then create an advisory lock on External System**
        - **retry limit?**
            - **only try so many times**
        - **One queue per External System**
            - **when service starts up it binds to all queues**
- **Split the services (step 1)**
    - **One service to evaluate**
    - **One service to send(action)**
- **Goals: scalable and durable**
- Reasonable timeout
    - currently 5 min
- Auto retry
    - There are some cases where we should auto retry
        - timeout
        - maybe everything?
        -
- System changes
    - Alerts to HIE Admin
- ~~One queue per tenant~~
    - ~~Round robin strategy to ensure each AWF Action is given attention~~
- Keep the system multi tenant!

Jira | ORANGE-25994

Figure 21. A question in Confluence which status was set to "done".



goes to. It is a tool to create forms".

I can walk through the application to show this [redacted] workflow if helpful.

12 Mar 2020 | DONE | [redacted]

On 3/12 [redacted] could not get his AWF to trigger for [redacted] because AWF services were busy processing Health Current. I just wanted to bring to your notice so that you could discuss about this and plan to introduce a minimum level of QoS for each environment.

- Currently one queue handles both evaluation and action
- Actions should be queued
- External System changes
  - **Back pressure**
    - ~~**When external system takes over X amount of time to respond then delay sending the next request**~~
    - **postgres advisory locking**
      - **when processing a message for External System A then create an advisory lock on External System**
    - **retry limit?**
      - **only try so many times**
    - **One queue per External System**
      - **when service starts up it binds to all queues**
- **Split the services (step 1)**
  - **One service to evaluate**
  - **One service to send(action)**
- **Goals: scalable and durable**
- Reasonable timeout
  - currently 5 min
- Auto retry
  - There are some cases where we should auto retry
    - timeout
    - maybe everything?
    -
- System changes
  - Alerts to HIE Admin
- ~~One queue per tenant~~
  - ~~Round robin strategy to ensure each AWF Action is given attention~~
- Keep the system multi tenant!

Jira | ORANGE-25994

information. Participants also create new Jira tickets during the meeting before starting various discussions. In Section 5.7, Figure 16 shows the creation of a new Jira ticket.

Participants also used tools to REORGANIZE information. For instance, they copied information from one section of an artifact to another section or reorganized the rows of tables. New content was never captured during these interactions with the tools.

Finally, an interesting category is TRACING. A few times, we observed participants creating links between artifacts in Confluence and Jira. Figure 15 in Section 5.7 shows an example of how a button (indicated by the right arrow) linking to a Jira ticket that had been created previously was added to "Ask an Architect" during the discussion of T3.

Observation 33: Even though capturing information is the main action performed with tools, other smaller tasks are also required during the meetings.

Overall, information was captured as discussion outflow nearly 71% of the times a tool was used. However, close to 30% of the times a tool was used, other actions were performed. Participants dedicated part of the meeting's time to get these other actions done. Given that their complexity is most of the time small (e.g., creating an empty ticket or wiki page, changing the status of a question in "Ask an Architect"), these tasks could potentially be carried out by semi-automatic tools (e.g., a bot to change a question in "Ask an Architect" as "done", a bot to close a ticket in Jira, a bot to make a copy of the "minutes of the meeting" template). I elaborate more on this idea in Section 5.10.

5.9 The participants that capture discussion outflow

In this section, I present the results of the analysis conducted to answer RQ 4.4 and RQ 4.5. For RQ 4.4: What participants drive the tools to capture discussion outflow? Table 31 shows the participants who used tools to capture discussion outflow. The first and second columns show each participant's role (as defined in Chapter 4, Table 8) and identifier (as defined in Chapter 3, Table 1), respectively. From the third to the fifth column all the tools used to capture outflow are listed. The last column presents the total number of times a participant used a tool. Results appear in descending order based on the number of times a tool was used by a given person (last column).

Only half the participants who attended the meetings took the role of notetaker at least once. P2 (one of the architects) is who played this role most often. This result is not surprising given that P2 typically moderates the meetings. However, it is interesting to

Table 31. Participants that used the tools to capture meaningful information.

Role	Participant	CONFLUENCE	JIRA	VSCODE	Total
SOFTWARE ARCHITECT	PARTICIPANT2	109	11	1	121
PRODUCT OWNER	PARTICIPANT1	15	0		15
INFRA ENGINEER	PARTICIPANT12	0	13		13
DEVELOPER	PARTICIPANT7	11	0		11
PRODUCT OWNER	PARTICIPANT5	0	6		6
SOFTWARE ARCHITECT	PARTICIPANT3	1	0		1
Total	Total	136	30	1	167

observe that other participants documented as well. Sometimes participants assumed the role during the whole meeting, such as P1 during the second meeting and P12 during the ninth meeting. Other times participants took on the role of notetaker more incidentally, while the discussion of a certain topic(s) lasted. As one example, when P5 requested to discuss a topic, she shared her screen with a document to take notes, and acted as notetaker only while the discussion lasted.

Observation 34: Even though the team has a designated notetaker during the meetings, other participants also take on this role when so needed.

On the one hand, having certain roles defined during the meetings (e.g., meeting moderator, notetaker) settles a meeting structure and dynamic that has a positive effect during meetings conducted on a regular basis, such as the ones studied here. On the other hand, having only one participant capturing information may raise the concern of capturing this individual's voice only rather than the voice of the team. In the case of these meetings, I lean more towards the former because most of the time, a participant (tool driver) kept the screen shared with the tool in which information was captured, which seems to be an established practice in the team. Therefore, the rest of the team members could have objected if wrong information had been captured. They could also have requested the tool driver to capture something they thought may be important. Moreover, regardless of P2 playing the official role of notetaker, other participants acted as notetakers as well on several occasions.

Participants also used tools to perform other kinds of actions. Table 32 shows the other actions that each participant performed from column sixth through ninth. P2 again was the one who performed other actions more often, and the only one who linked artifacts

Table 32. Participants who used the tools to capture discussion outflow and to perform other actions.

Role	Participant	ADDING NEW INFORMATION	REFINING	Times information was captured	REORGANIZING	SETTING METADATA	SETTING UP	TRACEABILITY	Times other actions were done	Total
SOFTWARE ARCHITECT	PARTICIPANT2	105	16	121	10	9	13	9	41	162
INFRA ENGINEER	PARTICIPANT12	10	3	13	2	9	2		13	26
PRODUCT OWNER	PARTICIPANT1	13	2	15	7		3		10	25
DEVELOPER	PARTICIPANT7	9	2	11			1		1	12
PRODUCT OWNER	PARTICIPANT5	5	1	6						6
SOFTWARE ARCHITECT	PARTICIPANT3	1		1						1
Total		143	24	167	19	18	19	9	65	232

(TRACING). Note how often P12 SET METADATA is equal to how often P2 did. Given that P12 ran the TICKET TRIAGE session, this result is not unexpected.

I also analyzed the ways in which tool drivers were prompted to capture information. Sometimes tool drivers captured information from their own volition, without anybody requesting them to do so (UNPROMPTED). Other times, however, an explicit request was indeed made (PROMPTED). There were also occasions in which the tool driver themselves asked the other participants what to capture (REQUESTED BY TOOL DRIVER).

Towards answering RQ 4.5: How are tool drivers prompted to capture discussion outflow? I counted the times each participant was prompted to capture information in one of the three aforementioned ways Table 33 presents the results of this analysis. The first and second columns show each participant's role (as defined in Chapter 4, Table 8) and identifier (as defined in Chapter 3, Table 1) respectively. From the third to the fifth column

Table 33. The ways tool drivers were prompted to capture discussion outflow.

Role	Participant	PROMPTED	REQUESTED BY TOOL DRIVER	UNPROMPTED	Total
SOFTWARE ARCHITECT	PARTICIPANT2	2		119	121
PRODUCT OWNER	PARTICIPANT1		4	11	15
INFRA ENGINEER	PARTICIPANT12	10		3	13
DEVELOPER	PARTICIPANT7			11	11
PRODUCT OWNER	PARTICIPANT5		1	5	6
SOFTWARE ARCHITECT	PARTICIPANT3			1	1
Total		12	5	150	167

the ways in which discussion outflow was captured are listed, and the last column shows the total number of times participants captured discussion outflow. These results appear in descending order based on the last column.

Note that with few exceptions, most discussion outflow was captured of the own volition of the tool drivers (UNPROMPTED). Only P12 was asked to capture information (PROMPTED) a considerable number of times. Interestingly, the product owners were the only ones who asked the rest of the team what information to capture (REQUESTED BY TOOL DRIVER).

Observation 35: Tool drivers capture discussion outflow mostly because they decided to do so.

This result highlights how critical it is to have a participant taking notes during meetings. While engaged in discussion, participants barely remind themselves or the tool driver to capture the outflow of the discussion. Without a person taking notes during these meetings, a lot of important information may perhaps not be captured.

5.10 Implications for research, practice and tool support

The results of the previous sections give rise to several observations concerning the current state of information capture in RSSMDMs. These results establish a baseline for future research, and have important implications for the design of tools and practices surrounding meetings of this kind. In this chapter, I discuss these implications by anchoring them in the observations and research questions introduced in previous sections.

5.10.1 Implications for research

The central contribution of this chapter is describing the kinds of information developers capture during the meetings. I observed participants capturing 16 different kinds of information from different domains. *Most of the information captured was DESIGN INFORMATION (RQ 3.1).* Information to improve the team's internal practice (TEAM PROCESS) and information related to PROJECT MANAGEMENT activities were captured as well, but not that often.

Thus far, a plethora of work about DESIGN INFORMATION states that certain kinds of information such as decisions [18], alternatives [199], constraints [201], and rationale [21] are essential to capture. However, few examples exist of what capturing such information looks like in practice. My work contributes to this body of knowledge with a detailed study of what kinds of information are captured in RSSMDMs. For example, the team captured alternatives (IDEA/ALTERNATIVE) as a list of key ideas on a wiki page. Main ideas were often captured as a parent bullet point, and dependent ideas as sub-bullets of a main idea. As a second example, I did not observe participants explicitly saying design decisions should be captured. However, the action of creating a ticket during the meeting and adding a HIGH-LEVEL DESCRIPTION of the work to be done implies that the team decided to make an update to the codebase. Some of the tickets created were often a step of an IMPLEMENTATION PLAN. The participants sometimes linked the tickets created to the artifact in which the overall plan had been captured. On a few occasions, I also observed the team capturing RATIONALE, sometimes as part of a ticket to explain what developing that ticket would afford to the end-users and other times as part of the description of an implementation plan (IMPLEMENTATION ROADMAP) in which a companion piece of RATIONALE explaining why a given step was necessary was included.

Studying the information that was captured in these meetings also revealed kinds of information that previous literature does not mention. For example, the participants captured the background of the design problems discussed (PROBLEM BACKGROUND/CODE STATE), which included narratives of the problem from the user's perspective as well as low-level descriptions of the current state of the code base. Moreover, the scope of the solutions proposed (IMPLEMENTATION GOAL/SCOPE) and feedback provided over design documents (THINGS TO KEEP IN MIND) were also captured.

The resulting kinds of design information captured that I observed also serves as a baseline to establish future research directions. For instance, future research may center on studying the connection between the kinds of prior information shared and the discussion outflow captured. *While participants shared 36 different kinds of prior information during their discussions, only 16 different kinds of discussion outflow were captured (RQ 3)*. Through an informal comparison of the kinds of information shared and the kinds of information captured I observed that some categories of discussion outflow integrate certain kinds of prior information. In this work, I did not systematically study whether prior information shared was captured as discussion outflow. However, the informal comparison performed led me to conclude that this is an important future research direction to follow. Establishing the connections between the information shared and the information captured has important implications for the design of tools that allow us to reuse in future meetings the discussion outflow captured as prior information in past meetings.

Another interesting future research direction relates to information loss. It would be worthwhile to investigate whether the participants believe that some of what was discussed during the *27 topics in which discussion outflow was not captured (RQ 3.2)* should have been captured. By exploring these 27 topics, I observed that for *many of these* *discussions outflow to capture was not generated (RQ 3.2)*. During various discussions outflow seemed to be relevant to only one participant. For other discussions work was in an early stage and capturing information seemed to be premature. Moreover, for discussions in which the objective was to classify tickets (TRIAGING) participants set metadata such as labels, but they did not capture any kind of information. I did observe six discussions in which design information was provided by some team members, and surprisingly it was not captured. These discussions raise the concern of important information not being captured. The exploration of the 27 topics provided an idea of how severe information loss was during these meetings. However, confirming this interpretation through other methods is important to formulate further conclusions about information loss in RSSMDMs.

Other future research directions stem from observing who captures information and how they do it. In these meetings, *someone was taking notes almost all the time, though this person was not always the same (RQ 4.4).* While having a designated notetaker during meetings is a well-known practice [213], [214], whether having the same person doing it or rotating the activity among the participants has not yet been studied. During these meetings, which were all multiple-topic oriented, more than one participant played the role of notetaker. Given that the nature of the topics discussed was diverse (e.g., ARCHITECTURAL RE-DESIGN, SUPPORT CASE, TESTING, COORDINATION AND PLANNING), it could be that having participants with certain backgrounds taking notes during certain discussions results convenient for the team. For example, while capturing the IMPLEMENTATION ROADMAP of an upcoming project, the management skills of product owners made them the best candidates to organize discussion outflow as a plan. As a second example, during post-mortem discussions that involved documenting internal procedures, the experience of the architects was key in articulating clear instructions about how to handle the issue that was discussed. Future studies in this regard may center on comparing the discussion outflow captured by teams with a single notetaker, versus teams with multiple notetakers. Moreover, it should be studied whether the background of participants has an impact on what information is captured.

Another interesting research direction in relation to notetaking is comparing different meeting notetaking approaches to understand which one works better and under which circumstances. During these meetings, notetakers typically shared their screen so that the rest of the team could see what they were capturing as discussion outflow. This approach to taking notes seems to mitigate the concern of only capturing the voice of a particular individual (the notetaker) rather than the voice of the team. Between having a participant taking individual notes (not showing to the others what is captured) and having everybody adding information to a shared document (i.e., a Google Docs document), this is a middle ground approach to capture collective notes that relies on a central point of entry because there is a single notetaker. Another research direction in this regard is investigating how participants feel being observed while taking notes. It has been observed before that not all people feel comfortable being observed [215]. During these meetings, discussion outflow was not captured during 27 topics. While in many cases good reasons seem to exist, as it was previously discussed, it would be worthwhile investigating if a contributing factor was that the notetakers felt uncomfortable being observed while writing.

In relation to who decides what to capture, it was interesting to observe that *most information was captured because the notetakers themselves decided to do it (RQ 4.5)*. Rather than capturing what others requested, or asking the team "*Hey, what should I capture?*", most of the time, notetakers captured information of their own volition. The rest of the team always had the opportunity to request information to be captured or corrected (which on a few occasions did happen). However, most of the time the notetakers decided when and what to capture, and this was implicitly accepted by the other participants. On the one hand, this could mean notetakers were assertive in selecting what to capture. On the other hand, it could mean that the rest of the team was not paying attention, or did not feel comfortable giving instructions to the notetaker. Further research is needed to systematically confirm how to interpret these results to be able to provide more general advice to practitioners.

Finally, other future directions emerge from the limitations of the study itself. I only analyzed the information that was captured during the meetings. However, is this information kept as is? Or do participants at some point refine it? And if so, who does it? Is it the whole team? The key internal stakeholders? Or does it vary on a case-by-case? These are questions that require further investigation by perhaps interviewing the team members or observing how information flows out of the meetings as well.

5.10.2 Implications for practice

The observations distilled also have important implications for practitioners in relation to the way they capture information during RSSMDMs. Some of the practices that I observed relate to who takes notes during the meetings, and what prompts them to do so.

Other practices relate to information loss, and the organization of the discussion outflow that is captured.

In these meetings, s*omeone was almost always taking notes, though this person was not always the same (RQ 4.4)*. During these meetings, this approach seems to work quite well for the team. First, not only one participant has the responsibility of capturing outflow. Thus, if this participant could not attend the meeting, others could take on this role. Second, given the diverse nature of the topics discussed (e.g., ARCHITECTURAL RE-DESIGN, SUPPORT CASE, TESTING, COORDINATION AND PLANNING), it seems convenient having participants with certain backgrounds taking notes during certain discussions. As it was previously explained in Section 5.10.1, some roles may be best suited to capture certain kinds of information. Other software development teams may find it useful to try out, first always having a notetaker during meetings, and second alternating this role among the meeting participants when so appropriate to see the effect it has on the discussion outflow that is captured.

Another interesting practice that I observed was that notetakers typically shared their screen so that the rest of the team could see what they were capturing as discussion outflow. If everybody is watching the information that is captured, participants always have the possibility to object when wrong information is captured or the possibility to request something relevant from the discussion (that was overlooked by the notetaker) to be captured. Thur, even though only one person is capturing information, the notes are collective, and everybody has the right to equally contribute.

Regarding information loss, I observed that *it is unlikely that the team captures information for impromptu discussion topics (RQ 3.3)* (i.e., SUPPORT CASES). This is an important factor to be aware of when the discussion of unplanned topics is requested. While capturing discussion outflow may not be always needed, information about important topics could indeed be lost. Could the key internal stakeholders somehow mitigate the risk of losing important information? Could tools detect when unplanned topics are discussed, and somehow remind the participants of it? As of now, there are no recommended practices nor tools that support participants in controlling the risk of losing information under this scenario. Therefore, this could also be a future research direction to explore. Meanwhile, meeting participants should be vigilant across all topics discussed and intentionally and explicitly decide for each topic what they should capture and who should do so.

Concerning how discussion outflow was organized, I observed good and bad practices. On some occasions, participants did separate information into different artifacts (e.g., TEAM PROCESS information was captured in Confluence, DESIGN INFORMATION was captured in Jira) during the same discussion. However, there were also occasions for which everything was captured in the same place (i.e., the minutes of the meeting). The organization of information has important implications for reusing the information captured in future meetings. The more organized information is, the easier it usually is to reuse it. In that context, the team using templates is a helpful practice. It would be worthwhile exploring whether having a template for each kind of discussion, rather than some templates plus a generic "minutes of the meeting" could improve quantity and quality of the information captured. As one example, for DEVISING A SOLUTION, a template with different sections to capture the problem's background (PROBLEM/BACKGROUND), the justification of decisions (RATIONALE), feedback (THINKS TO KEEP IN MIND), and more, might be a useful start.

5.10.3 Implications for tool support

This study leads to a future research agenda in terms of what information should be captured, where, and how. It also leads to potential ideas for tools that improve support for information sharing and capture in RSSMDMs. In the next paragraphs, I describe part of this future agenda, as well as ideas for tool prototypes that might be useful during RSSMDMs.

One interesting research question to explore is how can novel tools assist developers in creating information continuity across meetings to thereby improve the team's effectiveness and efficiency over time? In previous work [216], for instance, I proposed the design of a tool to capture Important Design Bits (IDBs), a piece of information that was spoken and is captured as a relatively short snippet of audio. Something desired and not achieved in that previous work was an understanding of which short information snippets would be a good candidate to keep for future use as IDBs. In section 5.10.1, I mentioned that establishing connections between the information that was shared and the information that was captured may have important implications to identify information that the team may reuse in future meetings. In the example provided in that section, a CODE FACT shared by one of the architects was captured as part of the problem's description. This is precisely a short piece of information to keep as an IDB.

In a similar vein, I also mentioned that I witnessed decisions made through silent agreements in which artifacts (e.g., tickets) were created, and brief HIGH LEVEL DESCRIPTIONS of the work to do were captured ("*In order to include the durability and* *stability of the AWF system we should split up the service into two pieces: evaluation service and action service*”). These brief HIGH LEVEL DESCRIPTIONS of what they decided to do could be important pieces of information to share during future discussions as well. Therefore, these are good IDB candidates too.

Another aspect to which my work contributes is highlighting the type of tools to which the research efforts should be directed. While the literature has historically reported digital whiteboards are often used for design work (e.g., [13], [85], [86]), the participants in these meetings did not use them. Instead, *a knowledge repository and sometimes an issue tracking system were the tools the participants used most often to capture discussion outflow (RQ 4.2).* Even though a broad body of knowledge exists about design knowledge repositories, and several prototypes of this kind were proposed (e.g., ADkwik [17], PAKME [165], and CADDMS [183]), their use during meetings has never been studied to date. An interesting research direction in this regard would be finding better ways to populate knowledge repositories. During these meetings, all information was captured by typing it. This activity dependeds on one person only, the notetaker. Moreover, much of the information captured quoted or phrased something somebody said closely. Therefore, another interesting research direction in this regard is exploring whether IDBs captured during an ongoing conversation could be used to populate knowledge repositories. In [217] and [216], I proposed a tool of this kind to be used during traditional design meetings at the whiteboard, and in [218] I presented a prototype to capture voice notes retroactively (after something has been said) during software design meetings. RSSMDMs provide an almost ideal scenario in which to explore IDBs collection to populate knowledge repositories so that the burden on the notetaker is reduced during the meeting.

Finally, another novel direction in terms of tool support builds upon observing other actions (i.e., not capturing information) performed with the tools. Participants linked artifacts across different tools (i.e., the reference to a ticket in Jira was established in a wiki page in Confluence) or spent some time setting up templates to capture information (e.g., creating a copy of a wiki template, creating tickets). Given that there is an emerging community of researchers investigating the use of bots in software engineering activities [219], a possible research direction in this regard might be exploring the use of bots to handle some of these repetitive tasks (e.g., changing the status of a question in Confluence, changing the status of a ticket in Jira). Could bots assist developers in handling some of the repetitive tasks needed in preparation to capture discussion outflow? Some potential ideas in this regard: a bot to mark a question in “Ask an Architect” as “done”, a bot to close a Jira ticket, a bot to make a copy of the “minutes of the meeting” at the beginning of each meeting or discussion, and more.

6 Threats to Validity

Despite the numbers presented in the tables, this work is much more qualitative than quantitative in nature, since I analyzed text and videos rather than performing a controlled experiment. Gasson [220] suggests that, rather than the validity criteria used in controlled experiments (i.e., objectivity, reliability, internal validity, external validity) to address typical threats to validity (i.e., representativeness of findings, reproducibility of findings, the rigor of method, and generalizability of findings) mentioned in the literature [221], an alternative set of criteria can be applied for studies like this one that are interpretive in nature. These criteria are: confirmability, dependability and auditability, internal consistency, and transferability [220].

Confirmability refers to the results depending on the study conditions and study rather than the researchers. We took a multistage approach to developing and refining the coding scheme, first defining a set of guiding principles to delineate what we mean by prior information and discussion outflow, together with associated guidelines for what text or video should be coded. For example, the two researchers who performed the coding set guidelines to code. In the case of prior information, small parts of relevant text as opposed to sentences or complete utterances from a single speaker, and in the case of discussion outflow, we consider as outflow all the times the participants capture information in a tool (e.g., adding new information, refining information). These guidelines were followed when performing the multi-pass coding, refining, recoding, and reviewing process outlined in Sections 4.1 and 5.1, respectively. A third person beyond the two who performed the primary coding played a particularly important role from the perspective of confirmability, as they took an independent look each time the analysis of a new meeting was completed.

That said, in the case of prior information, there was significant skill involved in interpreting a technical discussion, as well as in deciding which pieces of information were new versus a reformulation of information presented earlier. I do not feel that this is a skill only possessed by the two researchers that performed the coding, but should the study be repeated, the coders would need technical knowledge to follow the discussion in the meeting transcripts.

Dependability and auditability refer to the consistency and stability of the analysis process. The two researchers who performed the coding for prior information and discussion outflow respectively did this by consistently applying the coding schemes developed to analyze all ten meetings. For each coding scheme, the same two authors performed all coding and a third author reviewed the results each time. If new codes arose, prior meetings were re-examined to ensure that any changes in the coding scheme were applied consistently across.

Internal consistency looks at the credibility and consistency of the research findings. I note that a final analysis step involved examining all information per category to look for internal consistency within each category. I also note that the researchers who performed the coding are technically strong and have a reasonable understanding of the kinds of problems being discussed. Another strategy that I did not employ to this stage is to perform a member checking.

Finally, transferability refers to the ability to apply what is discovered to other contexts. The study to date only includes ten meetings from a single team from a single company. I take it as a positive sign that the coding scheme is domain neutral and that the resulting categories that the analysis revealed map very well onto typical software engineering concepts. That said, without further study of RSSMDMs from other teams and other companies, I will not know how transferable the findings are. The research methodology, however, is certainly transferrable and can be followed by other researchers. Moreover, these meetings have specific characteristics, their goal is to address software maintenance design, participants were distributed, and various topics were addressed per meeting. Different results may be obtained if a similar study is performed in other kinds of design meetings (e.g., early design meetings, architectural meetings) with different characteristics.

7 Conclusions

This dissertation contributes a foundational understanding of how information flows in and out of Regularly Scheduled Software Maintenance Design Meetings (RSSMDMs). RSSMDMs are recurring meetings during which the primary product leads of a software development team consider emerging issues and new directions for an already deployed and functioning software system. What motivates this work is building a baseline for future research about this kind of meeting, uncovering practices that may improve the way RSSMDMs are conducted, and inspiring the development of better tools to support participants with information capture and information sharing during RSSMDMs.

To build this understanding, I performed a single case study of ten RSSMDMs, all from a major healthcare software company. The study answers four overarching research questions:

RQ 1. What prior information do participants share in RSSMDMs in the course of design deliberation?

RQ 2. What tools do participants use in support of sharing prior information in RSSMDMs?

RQ 3. What is the discussion outflow of the topics addressed in RSSMDMs?

RQ 4. What tools do participants use to capture discussion outflow in RSSMDM?

Secondary questions were also posed and answered to complement the primary research questions.

The ten meetings were analyzed through a thematic analysis that centered on identifying what information the participants of these meetings shared, what information they captured, and what tools they used to do so.

7.1.1 Major findings

I performed a detailed analysis of the meetings and what transpired in them. Through the analysis, I made a rich set of observations, in Chapter 4 and 5. Here, I summarize:

- *Many different types of information are shared, with the range much greater than what traditionally has been considered as important to capture for future use.* This work identified 36 different kinds of information shared to discuss maintenance design. In terms of technical information, the kinds of information shared range from information about the system as deployed (e.g., RUN-TIME INFORMATION, DEPLOYMENT MANAGEMENT, DEPLOYMENT FACT) to descriptions of the codebase (e.g., CODE FACT, ARCHITECTURAL FACT, DEVELOPMENT PROGRESS) and its testing (i.e., TESTING FACT) to assessments about the system as deployed (i.e., DEPLOYMENT QUALITY ASSESSMENT), the state of the codebase (CODE QUALITY ASSESSMENT, ARCHITECTURAL ASSESSMENT), its testing (i.e., TESTING QUALITY ASSESSMENT) and its documentation (i.e., DOCUMENTATION QUALITY ASSESSMENT). The participants also shared information about issues with the software (e.g., ISSUE, ISSUE DETAIL) and new functionality to integrate (FUNCTIONALITY REQUEST). Other kinds of information shared were information about the customers (e.g., CUSTOMER CONTEX, CUSTOMER COST), information about internal practices and processes of the team (i.e., TEAM PROCESS, TEAM HOUSEKEEPING), and more.
- *Much of the information shared is fleeting, concerning the current state of the deployed software and the current state of its code base.* These kinds of

information (e.g., RUN-TIME INFORMATION, DEVELOPMENT PROGRESS, CODE FACT, DEPLOYMENT MANAGEMENT, DEPLOYMENT FACT, ISSUE DETAIL) typically offer insight into what is happening with the system at the client site or into the current state of the code and the progress in its development. This is not surprising, given that the meetings center on maintenance design [2], so design is always relative to what is already developed.

- *Information sharing is frequent, with new information shared on average once or twice a minute.* Previous research observed that software design is a knowledge intensive activity [24]. The observation that information sharing in RSSMDMs happens on average one or twice a minute strongly emphasizes this finding and provides an idea of just how important prior information appears to be in RSSMDMs. Again, perhaps this is not a surprise, given that this is maintenance design [2] and different people are typically present each time a topic is discussed. Thus, information sharing is important for all the participants to be on the same page and reach a common understanding of the work that needs to be done. That said, with a few exceptions (e.g., recollections shared, information requested), information sharing seems to be much less intentional (information is shared voluntarily). Typically, in a conversational and natural way, the prior information shared is often the companion of an idea that is being proposed, the background of an issue that is being explained, or the history behind a good practice established years ago.
- *The diversity and frequency of information captured is much less than information shared.* While participants shared 36 different kinds of prior information during

their discussions, only 16 different kinds of discussion outflow were captured. Moreover, out of 45 topics that were discussed across the ten meetings, participants shared information in 42, and only captured information in 18. This means information was shared in 93% of the discussions and captured only in 40% of them. On the one hand, this might indicate that capture is still a major hurdle. On the other hand, by exploring the 27 topics for which information was not captured, I found that capturing discussion outflow was not imperative for every topic discussed (e.g., discussions were mundane, actions were performed during the meeting rather than captured to be done later). For only six topics (out of 27) I observed valuable information that was not captured but that probably should have been captured.

- *The majority of information captured was design information.* Even though information about internal processes and management activities was captured, most of the information captured covered different aspects of design work. A key contribution that my work offers, then, is empirical evidence of what some of the types of design information historically recommended to be captured (e.g., alternatives, rationale) look like in practice. Moreover, I also found that participants capture other types of design information. For example, they captured design feedback (i.e., THINGS TO KEEP IN MIND) and the IMPLEMENTATION ROADMAP of features they decided to build.

In addition, the analysis led to a few observations about how the team shares and captures information were also obtained.

- *A significant amount of information was shared by a limited number of participants.* It is important that software development teams consciously engage in knowledge sharing and knowledge documentation, so that more than a few people know important knowledge about the system and the team's internal practices. Instead, knowledge should be spread among several team members. While it is not necessarily harmful that few people accumulate large amounts of knowledge, it is potentially harmful when only those people know it.
- *It is unlikely that the team captures information for impromptu discussion topics.* Topics for which valuable design information (i.e., design feedback) was provided and not captured, were often not part of the original meeting agenda. Participants requested to discuss them during the meeting, the discussion took place, the participant made either mental or their own (not visible to the team) notes, and the discussion concluded, with the participant then engaging sometime after in their own work as discussed.
- *There is typically a designated notetaker during each topic, but this person is not always the same for all the topics discussed.* While having a designated notetaker has been reported as a recommended practice before [213], [214], key characteristics about who that should be and how notetaking is performed in RSSMDMs, have not been studied to date. One participant was the notetaker on most occasions. However, this participant also handed off this activity to others depending on the kind of work being discussed, or the participants themselves requested to share their screen to take notes. Moreover, the notetakers (whoever they were) often shared their screen to show the information that was being captured to the rest of

the team. Therefore, even with only one person in control of the notes, the rest of the team could request information to be captured, or to be updated by the notetaker (which they did on a few occasions).

- *Traditional design tools such as diagramming and sketching tools are not used in support of the meetings, displaced by the use of Confluence (a wiki-style knowledge repository) and Jira (an issue tracker).* In terms of information capture, only these two tools were used. In terms of information sharing, however, most information was shared from memory. On the few occasions participants used tools to show prior information, Confluence and Jira were used most often, though information was also obtained from other tools (e.g., Slack, AWS, Selenium). Fleeting information in particular was typically obtained from deployment and monitoring tools such as AWS.

7.1.2 Contributions

RSSMDMs are essential to the success of medium to larger-scale software systems. The meetings are a forum for key technical stakeholders to address the typically continuous stream of design issues that tend to arise from the system being in use, whether it concerns problems in the field, new functionality to be added, or preventive changes to avoid future problems. This dissertation contributes a first study of these kinds of meetings, especially focusing on information sharing and information capture practices in the design conversations among meeting participants.

While software development meetings have been studied before (e.g., agile meetings [101], early design meetings [25], [92]), most studies did not analyze meetings from the perspective centered on the role of information. Moreover those that did focused on activities other than RSSMDMs (e.g., information needs in co-located software development teams [22], information needs in programming [23], [157]). My study contributes empirical evidence about what certain kinds of design information historically investigated (e.g., alternatives, rationale) look like in practice and reveals many other kinds of information that are shared and captured to convey design.

At a high level, the dissertation makes, the following two contributions:

C1. A rich set of 14 observations about information sharing practices and tool use in RSSMDMs. Understanding what kinds of prior information participants share to discuss design (e.g., architecture, deployment, testing), who shares it (e.g., product owners, software architects, developers), how it is shared (e.g., by request, voluntarily), and what tools are used to do so has important implications for future tool development and meeting practices in RSSMDMs.

C2. A rich set of 15 observations about information capture practices and tool use in RSSMDMs. Understanding what kinds of outflow participants capture, whether they capture outflow throughout the discussion or at specific moments, identifying when discussion outflow is not captured in tools, who captures it (e.g., product owners, software architects, developers), how participants capture it (e.g., prompted, unprompted, requested by the tool driver), and what tools they use to do so has important implications for future tool development and meeting practices in RSSMDMs.

While this study speaks to several research areas, it also exhibits certain limitations. First, it is a single exploratory case study. Thus, generic theory cannot be generated. Second, it is an interpretative study limited to analyzing transcripts and videos of the meetings. What the participants did that was not heard or seen I could not study. Beyond, I only studied a small number of meetings, in which the topics discussed were at a particular stage of development. By studying a longer timeframe, one may obtain different results (e.g., more topics could be rediscussed from meeting to meeting).

Even considering these and other limitations, the findings obtained provide insights into current practices that other software development teams may find useful to apply during RSSMDMs, or perhaps even during other kinds of meetings. In the context of RSSMDMs, the most relevant suggestions are the following:

- Encourage knowledge sharing among the team members and plan ahead what to do if members who hold much knowledge leave the team at some point.
- Always have a designated notetaker during the meetings and perhaps consider the topic to be discussed and the background this person has to select who will be the notetaker per topic (e.g., have software architects take notes while PERFORMING A POST-MORTEM, have product owners take notes while writing THE IMPLEMENTATION ROADMAP of a given project).
- Be vigilant in capturing information by explicitly deciding what should be captured for each topic. This practice should be applied for all topics without exception, even if a topic is spontaneously brought to the conversation. A question: "*Is there something to capture here?*" should always be explicitly asked.
- Define a set of good practices to follow while taking notes (e.g., notetakers should share their screen so that the rest of the team can observe what information is being captured as discussion outflow; if possible, organize discussion outflow into different artifacts such as wikis with a specific purpose or tickets; when more than

one artifact is used to capture design information, create links between the various artifacts used when so appropriate and possible.

7.1.3 Future work

This work also establishes a baseline for future research into RSSMDMs and offers suggestions for the development of improved tools to support participants with information sharing and information capture in these meetings. An outline of these ideas is provided below:

- Investigate whether the presence or absence of information needed impacts the quality of the solutions designed during the meetings. When information requested is not provided at the time the participants need it, the effects of it not being shared are unknown at this time. On the one hand, it may mean that the answers were not really needed. On the other hand, the absence of answers may lead to later problems. Moreover, it could also happen that answers are deferred to the team doing more detailed design once it has been considered by the key internal stakeholders and assigned to a participant. A study to explore these possibilities would help the community to measure the impact that missing information has on the quality of the design work done during RSSMDMs.
- Regarding design rationale, a direction to explore is to investigate how often rationale information is needed for other types of design work. While during RSSMDMs, rationale was not shared that often, a different distribution might be observed for other design settings (e.g., early design, architectural design, re-design to address technical debt). Overall, having a better idea of the design scenarios in

which rationale is needed would help the research community to devise new approaches to capturing and sharing design rationale, and practitioners to identify when it is important to capture it.

- Investigate whether a significant amount of information shared by a few participants represents a high risk for teams. In this regard, we must investigate whether software development teams prepare for such occasions (i.e., key team member leaving the company). Moreover, if they do prepare, we also need to investigate what actions are taken so that such strategies can be applied by others. An ethnographic study of teams in which a key team member leaves could provide valuable observations for practitioners on how to prepare for such scenarios.
- In terms of tool support for information sharing, it would be worthwhile to investigate three directions:
    - We must investigate why tools are not used that often to share information. A possible reason might be that tools do not live up to their potential to assist developers with information sharing during meetings. Another possible reason could be that users find it difficult to search information in the tools during the meetings. Exploring these and other possibilities will provide key insights into how to improve tools to share information during RSSMDMs.
    - Investigate how relevant it is for the meeting participants to share recollections of what external people said during the meetings. Even though recollections may not be shared very often, when it did happen in the RSSMDMs I studied, the information shared seemed to be critical to move the design process forward. Performing a study exclusively focused on this

    aspect would shed some light on the importance of recollections for RSSMDMs.
    - Explore mechanisms to obtain information from multiple sources. Initial ideas in this regard stem from the tools that were most often used. For instance, could we build a mechanism able to find and share information from sources such as Confluence, Jira and Slack channels during the meetings? And most importantly, could sharing information in this way make design deliberations more accurate in some way? In terms of tool prototyping, this is definitely an important research direction to pursue.
- Investigate what factors inhibit participants to not capturing important information. In my study, I observed that important information was shared and not captured for topics brought up impromptu. May this be an inhibitor to not capturing important information? Perhaps in the flow or articulating questions and answers for which participants were not prepared, they usually forget to capture design outflow. Moreover, I also observed that for many of these topics the developers were the notetakers. Is there something particular about their skills that prevents them from capturing outflow? Perhaps they did capture something, but not through the tool shared on screen. Figuring out what may cause the lack of capture in these scenarios may help practitioners to mitigate the risk of losing important information, because they would be aware of when they should actually capture.
- In terms of notetaking practices, three research directions stand out:
    - Studying the effects of having a single notetaker during the whole meeting versus having multiple notetakers (that would be chosen depending on the

topic to be discussed) to understand which approach is better and under which circumstances. Given that the nature of the topics discussed was diverse in these meetings (e.g., ARCHITECTURAL RE-DESIGN, SUPPORT CASE, TESTING, COORDINATION AND PLANNING), having participants with certain backgrounds taking notes during certain discussions might have been beneficial for the team (e.g., product owners captured implementation roadmaps, software architects documented internal procedures during post-mortem discussions). Future studies in this regard may center on comparing the discussion outflow captured by teams with a single notetaker, versus teams with multiple notetakers for different meeting scenarios (e.g., RSSMDMs, early-field design meetings, architectural debt meetings). This study would clarify what approach is better and for which kinds of discussions.

- Investigate whether the background of participants has an impact on what information is captured. Is project management information better organized when a project owner captures it? Is design information more specific and detailed when a software architect or a developer captures it? Finding answers to these questions would shed some light on how to select a notetaker for a given discussion.
- Investigate how participants feel being observed while taking collaborative notes. Privacy is an important angle of many work activities. Personality (e.g., extroverted, introverted) and skills (e.g., writing skills, being with the expectation of sharing the native speaker for the language in which notes are

taken), as well as other factors, play an important role in whether one feels comfortable performing certain activities in front of others. Towards rotating the role of notetakers and with the expectation of sharing the screen, it is also important to investigate whether participants feel comfortable doing so.

- In terms of tool support for information capture, it is important to explore other ways to easily capture information in RSSMDMs so that the workload of notetakers is reduced. For instance, IDBs [216] captured by the participants could be used to populate a knowledge repository such as Confluence. This could bring two important benefits to a team. First, it could reduce notetaking itself because less information would have to be captured. Second, it could make information capture more agile. Rather than capturing complete ideas, small updates to the IDBs captured could be made in order to shape the final documentation. Moreover, it is also important to explore the use of bots for some of the activities performed in preparation for capturing discussion outflow. For instance, bots could assist developers in handling some of the repetitive tasks needed in preparation to capture discussion outflow (e.g., a bot to mark a question in "Ask an Architect" as "done", a bot to close a Jira ticket, a bot to make a copy of the "minutes of the meeting" at the beginning of each meeting or discussion). As one example, participants usually have to find the template of the "minutes of the meeting", create a copy, assign it a name, and make some basic edits such as updating the date. All this work could be done by a bot as per the request of the notetaker via a simple command (i.e., "*set up minutes of the meeting"*). Even though most of these tasks are not complex, they do take time and participants tend to perform them for almost every topic discussed during a meeting.

- Investigate how to create information continuity across meetings. Critical steps to explore in this regard are:
    - Establishing connections between the information that is shared and the information that is captured. For example, for the meetings I have analyzed, I could go through each segment in which information was captured, to analyze whether something in the content includes some of the segments coded as prior information in the previous discussion (within the scope of each topic). This analysis would provide a clear idea of how often prior information shared is captured.
    - Investigate whether better tools could help to create such continuity. How can novel tools assist developers in creating information continuity across RSSMDMs to thereby improve the team's effectiveness and efficiency over time? My previous work with IDBs [216] could be a starting point in this regard. Building an interface to bring back IDBs during a meeting (e.g., meeting assistant, chatbot), and evaluating its performance by conducting design studios with professional software developers such as the ones conducted in [92], would provide insights into designing an effective interface for this task.

REFERENCES

[1] R. N. Taylor and A. van der Hoek, "Software Design and Architecture The once and future focus of software engineering," in *Future of Software Engineering (FOSE '07)*, May 2007, pp. 226–243. doi: 10.1109/FOSE.2007.21.

[2] P. Grisham, H. Iida, and D. Perry, "Improving design intent research for software maintenance," 2009. [Online]. Available: http://users.ece.utexas.edu/~perry/work/papers/0712-PG-atgse1.pdf

[3] A. Baker and A. Van Der Hoek, "Understanding maintenance design: how developers work at the whiteboard to design solutions for software maintenance," University of California, Irvine, Institute for Software Research Technical Report, 2010. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.187.8446&rep=rep1&type=pdf

[4] V. Stray, D. I. K. Sjøberg, and T. Dybå, "The daily stand-up meeting: A grounded theory study," *Journal of Systems and Software*, vol. 114, pp. 101–124, 2016.

[5] P. Lous, P. Tell, C. B. Michelsen, Y. Dittrich, M. Kuhrmann, and A. Ebdrup, "Virtual by Design: How a Work Environment can Support Agile Distributed Software Development," in *2018 IEEE/ACM 13th International Conference on Global Software Engineering (ICGSE)*, May 2018, pp. 97–106.

[6] P. Kruchten, R. Nord, and I. Ozkaya, *Managing Technical Debt: Reducing Friction in Software Development*. Addison-Wesley Professional, 2019.

[7] D. Falessi, P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt at the crossroads of research and practice: report on the fifth international workshop on managing technical debt," *SIGSOFT Softw. Eng. Notes*, vol. 39, no. 2, pp. 31–33, Mar. 2014, doi: 10.1145/2579281.2579311.

[8] S. Rastkar and G. C. Murphy, "Why did this code change?," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 1193–1196.

[9] D. Falessi, G. Cantone, R. Kazman, and P. Kruchten, "Decision-making techniques for software architecture design: A comparative survey," *ACM Comput. Surv.*, vol. 43, no. 4, p. 33:1-33:28, Oct. 2011, doi: 10.1145/1978802.1978812.

[10] G. Fischer *et al.*, "Seeding, evolutionary growth and reseeding: The incremental development of collaborative design environments," *Coordination theory and collaboration technology*, vol. 447, p. 472, 2001.

[11] N. Mangano, T. D. LaToza, M. Petre, and A. van der Hoek, "Supporting informal design with interactive whiteboards," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, Apr. 2014, pp. 331–340. doi: 10.1145/2556288.2557411.

[12] G. Bortis, "DesignMinders: Preserving and Sharing Informal Software Design Knowledge," p. 8.

[13] B. Vesin, R. Jolak, and M. R. V. Chaudron, "OctoUML: An Environment for Exploratory and Collaborative Software Design," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, May 2017, pp. 7–10. doi: 10.1109/ICSE-C.2017.19.

[14] V. Clerc, E. de Vries, and P. Lago, "Using wikis to support architectural knowledge management in global software development," in *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, New York, NY, USA, May 2010, pp. 37–43. doi: 10.1145/1833335.1833341.

[15] B. Decker, E. Ras, J. Rech, P. Jaubert, and M. Rieth, "Wiki-Based stakeholder participation in requirements engineering," *IEEE Software*, vol. 24, no. 2, pp. 28–35, 2007, doi: 10.1109/MS.2007.60.

[16] R. Farenhorst, P. Lago, and H. Van Vliet, "Eagle: Effective tool support for sharing architectural knowledge," *International Journal of Cooperative Information Systems*, vol. 16, no. 3–4, pp. 413–437, 2007, doi: 10.1142/s0218843007001706.

[17] N. Schuster, O. Zimmermann, and C. Pautasso, "ADkwik: Web 2.0 collaboration system for architectural decision engineering," 2007, pp. 255–260.

[18] H. van Vliet and A. Tang, "Decision making in software architecture," *Journal of Systems and Software*, vol. 117, pp. 638–644, Jul. 2016, doi: 10.1016/j.jss.2016.01.017.

[19] M. Nygard, "Documenting Architecture Decisions," 2011. https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions (accessed Jun. 10, 2022).

[20] D. G. Bobrow and I. P. Goldstein, "Representing design alternatives," in *Proceedings of the 1980 AISB Conference on Artificial Intelligence*, NLD, Jul. 1980, pp. 25–35.

[21] T. P. Moran and J. M. Carroll, *Design Rationale: Concepts, Techniques, and Use*. CRC Press, 2020.

[22] A. J. Ko, R. DeLine, and G. Venolia, "Information Needs in Collocated Software Development Teams," in *29th International Conference on Software Engineering (ICSE'07)*, Minneapolis, MN, USA, May 2007, pp. 344–353. doi: 10.1109/ICSE.2007.45.

[23] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: improving cooperation between developers and users," in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, New York, NY, USA, Feb. 2010, pp. 301–310. doi: 10.1145/1718918.1718973.

[24] P. N. Robillard, "The role of knowledge in software development," *Commun. ACM*, vol. 42, no. 1, pp. 87–92, Jan. 1999, doi: 10.1145/291469.291476.

[25] G. M. Olson, J. S. Olson, M. R. Carter, and M. Storrosten, "Small Group Design Meetings: An Analysis of Collaboration," *Human–Computer Interaction*, vol. 7, no. 4, pp. 347–374, 1992.

[26] G. M. Olson and J. S. Olson, "Distance Matters," *Human–Computer Interaction*, vol. 15, no. 2–3, pp. 139–178, Sep. 2000, doi: 10.1207/S15327051HCI1523_4.

[27] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting Empirical Methods for Software Engineering Research," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. London: Springer, 2008, pp. 285–311. doi: 10.1007/978-1-84800-044-5_11.

[28] C. Wohlin and A. Aurum, "Towards a decision-making structure for selecting a research design in empirical software engineering," *Empir Software Eng*, vol. 20, no. 6, pp. 1427–1455, Dec. 2015, doi: 10.1007/s10664-014-9319-7.

[29] G. Guest, K. M. MacQueen, and E. E. Namey, *Applied Thematic Analysis*. SAGE Publications, 2011.

[30] D. S. Cruzes and T. Dyba, "Recommended Steps for Thematic Synthesis in Software Engineering," in *2011 International Symposium on Empirical Software Engineering and Measurement*, Sep. 2011, pp. 275–284. doi: 10.1109/ESEM.2011.36.

[31] "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990*, pp. 1–55, Dec. 1990, doi: 10.1109/IEEESTD.1990.101064.

[32] E. B. Swanson, "The dimensions of maintenance," in *Proceedings of the 2nd international conference on Software engineering*, 1976, pp. 492–497.

[33] B. A. Kitchenham *et al.*, "Towards an ontology of software maintenance," *Journal of Software Maintenance: Research and Practice*, vol. 11, no. 6, pp. 365–389, 1999, doi: 10.1002/(SICI)1096-908X(199911/12)11:6<365::AID-SMR200>3.0.CO;2-W.

[34] S. Mamone, "The IEEE standard for software maintenance," *SIGSOFT Softw. Eng. Notes*, vol. 19, no. 1, pp. 75–76, Jan. 1994, doi: 10.1145/181610.181623.

[35] D.-R. Harjani and J.-P. Queille, "A process model for the maintenance of large space systems software," Jan. 1992, pp. 127–136. doi: 10.1109/ICSM.1992.242550.

[36] N. Chapin, J. E. Hale, K. Md. Khan, J. F. Ramil, and W.-G. Tan, "Types of software evolution and software maintenance," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 13, no. 1, pp. 3–30, 2001.

[37] "ISO/IEC/IEEE 14764:2022(en), Software engineering — Software life cycle processes — Maintenance." https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:14764:ed-3:v1:en (accessed Mar. 23, 2022).

[38] M. O'Keeffe and M. Ó. Cinnéide, "Search-based refactoring: an empirical study," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 5, pp. 345–364, 2008, doi: 10.1002/smr.378.

[39] T. D. Hendrix, J. H. Cross, S. Maghsoodloo, and M. L. McKinney, "Do visualizations improve program comprehensibility? experiments with control structure diagrams for Java," in *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, New York, NY, USA, Mar. 2000, pp. 382–386. doi: 10.1145/330908.331890.

[40] L. Layman, M. Diep, M. Nagappan, J. Singer, R. Deline, and G. Venolia, "Debugging Revisited: Toward Understanding the Debugging Needs of Contemporary Software Developers," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Oct. 2013, pp. 383–392. doi: 10.1109/ESEM.2013.43.

[41] A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni, "Identifying the characteristics of vulnerable code changes: an empirical study," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2014, pp. 257–268. doi: 10.1145/2635868.2635880.

[42] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, "How do software engineers understand code changes? an exploratory study in industry," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, New York, NY, USA, Nov. 2012, pp. 1–11. doi: 10.1145/2393596.2393656.

[43] R. Purushothaman and D. E. Perry, "Toward understanding the rhetoric of small source code changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 511–526, Jun. 2005, doi: 10.1109/TSE.2005.74.

[44] J. Sillito, G. C. Murphy, and K. De Volder, "Asking and Answering Questions during a Programming Change Task," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 434–451, Jul. 2008, doi: 10.1109/TSE.2008.26.

[45] B. De Alwis, "Supporting conceptual queries over integrated sources of program information," 2008, doi: 10.14288/1.0051181.

[46] M. Dias, A. Bacchelli, G. Gousios, D. Cassou, and S. Ducasse, "Untangling fine-grained code changes," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 341–350.

[47] T. D. LaToza and B. A. Myers, "Visualizing call graphs," in *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Sep. 2011, pp. 117–124. doi: 10.1109/VLHCC.2011.6070388.

[48] Y. Yoon, B. A. Myers, and S. Koo, "Visualization of fine-grained code change history," in *2013 IEEE Symposium on Visual Languages and Human Centric Computing*, 2013, pp. 119–126.

[49] L. Voinea, A. Telea, and J. J. van Wijk, "CVSscan: visualization of code evolution," in *Proceedings of the 2005 ACM symposium on Software visualization*, New York, NY, USA, May 2005, pp. 47–56. doi: 10.1145/1056018.1056025.

[50] A. J. Ko and B. A. Myers, "Source-level debugging with the whyline," in *Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, New York, NY, USA, 2008, pp. 69–72. Accessed: Mar. 14, 2022. [Online]. Available: https://doi.org/10.1145/1370114.1370132

[51] B. de Alwis and G. C. Murphy, "Answering conceptual queries with Ferret," in *Proceedings of the 30th international conference on Software engineering*, New York, NY, USA, 2008, pp. 21–30. Accessed: Mar. 14, 2022. [Online]. Available: https://doi.org/10.1145/1368088.1368092

[52] A. J. Ko and B. A. Myers, "Designing the whyline: a debugging interface for asking questions about program behavior," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2004, pp. 151–158. Accessed: Mar. 14, 2022. [Online]. Available: https://doi.org/10.1145/985692.985712

[53] G. Rasool and N. Asif, "Software architecture recovery," *International Journal of Computer, Information, and Systems Science, and Engineering*, vol. 1, no. 3, pp. 206–211, 2007.

[54] C.-H. Lung, "Software architecture recovery and restructuring through clustering techniques," in *Proceedings of the third international workshop on Software architecture*, 1998, pp. 101–104.

[55] M. Lungu, M. Lanza, and O. Nierstrasz, "Evolutionary and collaborative software architecture recovery with Softwarenaut," *Science of Computer Programming*, vol. 79, pp. 204–223, 2014.

[56] M. Schmitt Laser, N. Medvidovic, D. M. Le, and J. Garcia, "ARCADE: an extensible workbench for architecture recovery, change, and decay evaluation," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, New York, NY, USA, 2020, pp. 1546–1550. Accessed: Mar. 14, 2022. [Online]. Available: https://doi.org/10.1145/3368089.3417941

[57] E. Kouroshfar, M. Mirakhorli, H. Bagheri, L. Xiao, S. Malek, and Y. Cai, "A Study on the Role of Software Architecture in the Evolution and Quality of Software," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 246–257.

[58] G. Mark, "Extreme collaboration," *Commun. ACM*, vol. 45, no. 6, pp. 89–93, 2002.

[59] S. Teasley, L. Covi, M. S. Krishnan, and J. S. Olson, "How does radical collocation help a team succeed?," in *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, New York, NY, USA, Dec. 2000, pp. 339–346. doi: 10.1145/358916.359005.

[60] L. M. Covi, J. S. Olson, E. Rocco, W. J. Miller, and P. Allie, "A Room of Your Own: What Do We Learn about Support of Teamwork from Assessing Teams in Dedicated Project Rooms?," in *Cooperative Buildings: Integrating Information, Organization, and Architecture*, Berlin, Heidelberg, 1998, pp. 53–65. doi: 10.1007/3-540-69706-3_7.

[61] M. Eccles, J. Smith, M. Tanner, J. Van Belle, and S. van der Watt, “The impact of collocation on the effectiveness of agile is development teams,” *Communications of the IBIMA*, vol. 2010, pp. 1–11, 2010.
[62] B. Sengupta, S. Chandra, and V. Sinha, “A research agenda for distributed software development,” in *Proceedings of the 28th international conference on Software engineering*, New York, NY, USA, 2006, pp. 731–740. doi: 10.1145/1134285.1134402.
[63] K. Ketler and J. Walstrom, “The outsourcing decision,” *International Journal of Information Management*, vol. 13, no. 6, pp. 449–459, Dec. 1993, doi: 10.1016/0268-4012(93)90061-8.
[64] S. Komi-Sirviö and M. Tihinen, “Lessons learned by participants of distributed software development,” *Knowledge and Process Management*, vol. 12, no. 2, pp. 108–122, 2005, doi: 10.1002/kpm.225.
[65] D. Šmite, “A Case Study: Coordination Practices in Global Software Development,” in *Product Focused Software Process Improvement*, Berlin, Heidelberg, 2005, pp. 234–244. doi: 10.1007/11497455_20.
[66] J. A. Espinosa, S. A. Slaughter, R. E. Kraut, and J. D. Herbsleb, “Team Knowledge and Coordination in Geographically Distributed Software Development,” *Journal of Management Information Systems*, vol. 24, no. 1, pp. 135–169, Jul. 2007, doi: 10.2753/MIS0742-1222240104.
[67] R. E. Kraut and L. A. Streeter, “Coordination in software development,” *Communications of the ACM*, vol. 38, no. 3, pp. 69–82, 1995.
[68] D. E. Perry, N. A. Staudenmayer, and L. G. Votta, “People, organizations, and process improvement,” *IEEE Software*, vol. 11, no. 4, pp. 36–45, Jul. 1994, doi: 10.1109/52.300082.
[69] J. Noll, S. Beecham, and I. Richardson, “Global software development and collaboration: barriers and solutions,” *ACM Inroads*, vol. 1, no. 3, pp. 66–78, Sep. 2011, doi: 10.1145/1835428.1835445.
[70] F. P. B. Jr, *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Pearson Education, 1995.
[71] D. E. Perry and A. L. Wolf, “Foundations for the study of software architecture,” *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, pp. 40–52, Oct. 1992, doi: 10.1145/141874.141884.
[72] M. A. Jackson, *Problem Frames: Analysing and Structuring Software Development Problems*. Harlow: Addison-Wesley, 2000.
[73] E. Gamma, R. Johnson, R. Helm, R. E. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Deutschland GmbH, 1995.
[74] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, “Let’s Go to the Whiteboard: How and Why Software Developers Use Drawings,” in *SIGCHI Conference on Human Factors in Computing Systems*, 2007, pp. 557–566. doi: 10.1145/1240624.1240714.
[75] K. Yatani, E. Chung, C. Jensen, and K. N. Truong, “Understanding how and why open source contributors use diagrams in the development of Ubuntu,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, Apr. 2009, pp. 995–1004. doi: 10.1145/1518701.1518853.
[76] C. Zannier, M. Chiasson, and F. Maurer, “A model of design decision making based on empirical results of interviews with software designers,” *Information and Software Technology*, vol. 49, no. 6, pp. 637–653, Jun. 2007, doi: 10.1016/j.infsof.2007.02.010.
[77] J. D. Herbsleb, H. Klein, G. M. Olson, H. Brunner, J. S. Olson, and J. Harding, “Object-Oriented Analysis and Design in Software Project Teams,” *Human–Computer Interaction*, vol. 10, no. 2–3, pp. 249–292, Jun. 1995, doi: 10.1080/07370024.1995.9667219.
[78] N. Mangano, T. D. LaToza, M. Petre, and A. van der Hoek, “How Software Designers Interact with Sketches at the Whiteboard,” *IEEE Transactions on Software Engineering*, vol. 41, no. 2, pp. 135–156, Feb. 2015, doi: 10.1109/TSE.2014.2362924.
[79] C. Zannier and F. Maurer, “Comparing Decision Making in Agile and Non-agile Software Organizations,” in *Agile Processes in Software Engineering and Extreme Programming*, Berlin, Heidelberg, 2007, pp. 1–8. doi: 10.1007/978-3-540-73101-6_1.
[80] U. Dekel and J. D. Herbsleb, “Notation and representation in collaborative object-oriented design: an observational study,” *SIGPLAN Not.*, vol. 42, no. 10, pp. 261–280, 2007, doi: 10.1145/1297105.1297047.
[81] W. Heijstek, T. Kühne, and M. R. V. Chaudron, “Experimental Analysis of Textual and Graphical Representations for Software Architecture Design,” in *2011 International Symposium on Empirical Software Engineering and Measurement*, Sep. 2011, pp. 167–176. doi: 10.1109/ESEM.2011.25.
[82] J. A. Landay and B. A. Myers, “Interactive sketching for the early stages of user interface design,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1995, pp. 43–50.

[83] C. H. Damm, K. M. Hansen, and M. Thomsen, "Tool support for cooperative object-oriented design: gesture based modelling on an electronic whiteboard," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, New York, NY, USA, Apr. 2000, pp. 518–525. doi: 10.1145/332040.332488.

[84] J. Lin, M. W. Newman, J. I. Hong, and J. A. Landay, "DENIM: finding a tighter fit between tools and practice for Web site design," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, New York, NY, USA, Apr. 2000, pp. 510–517. doi: 10.1145/332040.332486.

[85] N. Mangano, A. Baker, and A. van der Hoek, "Calico: a prototype sketching tool for modeling in early design," in *Proceedings of the 2008 international workshop on Models in software engineering*, New York, NY, USA, May 2008, pp. 63–68. doi: 10.1145/1370731.1370747.

[86] D. Wüest, N. Seyff, and M. Glinz, "FlexiSketch: A Mobile Sketching Tool for Software Modeling," in *Mobile Computing, Applications, and Services*, Berlin, Heidelberg, 2013, pp. 225–244. doi: 10.1007/978-3-642-36632-1_13.

[87] "Studying Professional Software Design," 2010. https://www.ics.uci.edu/design-workshop/ (accessed Apr. 11, 2022).

[88] N. Cross, "A brief history of the Design Thinking Research Symposium series," *Design Studies*, vol. 57, pp. 160–164, 2018.

[89] H. Christiaans and R. A. Almendra, "Accessing Decision-Making in Software Design," *Design Studies*, vol. 31, no. 6, Art. no. 6, 2010, doi: 10.1016/j.destud.2010.09.005.

[90] B. Matthews, "Designing Assumptions," in *Software Designers in Action: A Human-Centric Look at Design Work*, M. Petre and A. van der Hoek, Eds. Chapman and Hall/CRC, 2013, pp. 249–266.

[91] R. K. E. Bellamy, B. E. John E., and M. Desmond, "Concern Development in Software Design Discussions," in *Software Designers in Action: A Human-Centric Look at Design Work*, H. Ossher, M. Petre, and A. van der Hoek, Eds. Chapman and Hall/CRC, 2013, pp. 249–266.

[92] A. Baker and A. van der Hoek, "Ideas, subjects, and cycles as lenses for understanding the software design process," *Design Studies*, vol. 31, no. 6, pp. 590–613, Nov. 2010, doi: 10.1016/j.destud.2010.09.008.

[93] A. Tang, A. Aleti, J. Burge, and H. van Vliet, "What makes software design effective?," *Design Studies*, vol. 31, no. 6, pp. 614–640, Nov. 2010, doi: 10.1016/j.destud.2010.09.004.

[94] S. Baltes and S. Diehl, "Sketches and diagrams in practice," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, Nov. 2014, pp. 530–541. doi: 10.1145/2635868.2635891.

[95] S. Baltes, P. Schmitz, and S. Diehl, "Linking sketches and diagrams to source code artifacts," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, Nov. 2014, pp. 743–746. doi: 10.1145/2635868.2661672.

[96] S. Baltes, F. Hollerich, and S. Diehl, "Round-Trip Sketches: Supporting the Lifecycle of Software Development Sketches from Analog to Digital and Back," in *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, Sep. 2017, pp. 94–98. doi: 10.1109/VISSOFT.2017.24.

[97] D. Wüest, N. Seyff, and M. Glinz, "FLEXISKETCH TEAM: Collaborative Sketching and Notation Creation on the Fly," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, May 2015, vol. 2, pp. 685–688. doi: 10.1109/ICSE.2015.223.

[98] D. Wüest, N. Seyff, and M. Glinz, "Sketching and notation creation with FlexiSketch Team: Evaluating a new means for collaborative requirements elicitation," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, Aug. 2015, pp. 186–195. doi: 10.1109/RE.2015.7320421.

[99] S. Grapenthin, M. Book, V. Gruhn, C. Schneider, and K. Völker, "Reducing complexity using an interaction room: an experience report," in *Proceedings of the 31st ACM international conference on Design of communication*, New York, NY, USA, Sep. 2013, pp. 71–76. doi: 10.1145/2507065.2507087.

[100] M. Book, M. Riedel, H. Neukirchen, and M. Götz, "Facilitating collaboration in high-performance computing projects with an interaction room," in *Proceedings of the 4th ACM SIGPLAN International Workshop on Software Engineering for Parallel Systems*, New York, NY, USA, Oct. 2017, pp. 46–47. doi: 10.1145/3141865.3142467.

[101] J. Dalton, "Daily Stand-Up," in *Great Big Agile: An OS for Agile Leaders*, J. Dalton, Ed. Berkeley, CA: Apress, 2019, pp. 155–157. doi: 10.1007/978-1-4842-4206-3_24.

[102] S. Grapenthin, S. Poggel, M. Book, and V. Gruhn, "Facilitating Task Breakdown in Sprint Planning Meeting 2 with an Interaction Room: An Experience Report," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, Aug. 2014, pp. 1–8. doi: 10.1109/SEAA.2014.71.

[103] Y. Y. Ng, J. Skrodzki, and M. Wawryk, "Playing the Sprint Retrospective: A Replication Study," in *Advances in Agile and User-Centred Software Engineering*, Cham, 2020, pp. 133–141. doi: 10.1007/978-3-030-37534-8_7.

[104] V. G. Stray, N. B. Moe, and A. Aurum, "Investigating Daily Team Meetings in Agile Software Projects," in *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, Sep. 2012, pp. 274–281. doi: 10.1109/SEAA.2012.16.

[105] V. G. Stray, Y. Lindsjørn, and D. I. K. Sjøberg, "Obstacles to Efficient Daily Meetings in Agile Development Projects: A Case Study," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Oct. 2013, pp. 95–102. doi: 10.1109/ESEM.2013.30.

[106] V. Stray, "An empirical investigation of the daily stand-up meeting in Agile software development projects," *J. Syst. Softw. o*, 2014.

[107] J. M. Atlee and M. W. Godfrey, "Studying Professional Software Designers and their Use of Abstraction," 2010.

[108] V. Stray, N. B. Moe, and G. R. Bergersen, "Are Daily Stand-up Meetings Valuable? A Survey of Developers in Software Teams," in *Agile Processes in Software Engineering and Extreme Programming*, 2017, pp. 274–281.

[109] K. Niemantsverdriet and T. Erickson, "Recurring Meetings: An Experiential Account of Repeating Meetings in a Large Organization," *Proc. ACM Hum.-Comput. Interact.*, vol. 1, no. CSCW, p. 84:1-84:17, Dec. 2017, doi: 10.1145/3134719.

[110] J. M. Barrero, N. Bloom, and S. J. Davis, "Why working from home will stick," National Bureau of Economic Research, 2021.

[111] K. Parker, J. Menasce Horowitz, and R. Minkin, "How Coronavirus Has Changed the Way Americans Work," *Pew Research Center*, 2020. https://www.pewresearch.org/social-trends/2020/12/09/how-the-coronavirus-outbreak-has-and-hasnt-changed-the-way-americans-work/ (accessed Mar. 15, 2022).

[112] S. Rintel, P. Wong, A. Sarkar, and A. Sellen, "Methodology and Participation for 2020 Diary Study of Microsoft Employees Experiences in Remote Meetings During COVID-19." [Online]. Available: https://www.microsoft.com/en-us/research/uploads/prod/2020/10/2020-10-FOW-SIM1-RemoteMeetingsDuringCOVID19-MethodologyAndParticipation.pdf

[113] A. H. Anderson, R. McEwan, J. Bal, and J. Carletta, "Virtual team meetings: An analysis of communication and context," *Computers in Human Behavior*, vol. 23, no. 5, pp. 2558–2580, 2007.

[114] B. Saatçi, K. Akyüz, S. Rintel, and C. N. Klokmose, "(Re)Configuring Hybrid Meetings: Moving from User-Centered Design to Meeting-Centered Design," *Comput Supported Coop Work*, vol. 29, no. 6, pp. 769–794, 2020.

[115] "Socially Intelligent Meetings," *Microsoft Research*. https://www.microsoft.com/en-us/research/project/socially-intelligent-meetings/ (accessed Jan. 19, 2022).

[116] N. Prenner, J. Klünder, and K. Schneider, "Making meeting success measurable by participants' feedback," in *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*, New York, NY, USA, 2018, pp. 25–31. doi: 10.1145/3194932.3194933.

[117] A. Sarkar *et al.*, "The promise and peril of parallel chat in video meetings for work," in *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA, May 2021, pp. 1–8. doi: 10.1145/3411763.3451793.

[118] N. Yankelovich, J. McGinn, M. Wessler, J. Kaplan, J. Provino, and H. Fox, "Private communications in public meetings," in *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, Apr. 2005, pp. 1873–1876. doi: 10.1145/1056808.1057044.

[119] S. Whittaker, P. Hyland, and M. Wiley, "Filochat: Handwritten notes provide access to recorded conversations," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1994, pp. 271–277.

[120] B. N. Schilit, L. D. Wilcox, and N. "Nick" Sawhney, "Merging the benefits of paper notebooks with the power of computers in dynomite," in *CHI '97 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, Mar. 1997, pp. 22–23. doi: 10.1145/1120212.1120228.

[121] L. Stifelman, B. Arons, and C. Schmandt, “The audio notebook: paper and pen interaction with structured speech,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, Mar. 2001, pp. 182–189. doi: 10.1145/365024.365096.

[122] R. Capilla, R. Jolak, M. R. V. Chaudron, and C. Carrillo, “Design Decisions by Voice: The Next Step of Software Architecture Knowledge Management,” in *Human-Centered Software Engineering*, Cham, 2020, pp. 166–177. doi: 10.1007/978-3-030-64266-2_10.

[123] A. M. Soria, “KNOCAP: capturing and delivering important design bits in whiteboard design meetings,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, New York, NY, USA: Association for Computing Machinery, 2020, pp. 194–197. Accessed: Apr. 26, 2022. [Online]. Available: https://doi.org/10.1145/3377812.3381397

[124] P. Chiu, J. Boreczky, A. Girgensohn, and D. Kimber, “LiteMinutes: an Internet-based system for multimedia meeting minutes,” in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 140–149.

[125] A. Waibel *et al.*, “SMaRT: the Smart Meeting Room Task at ISL,” in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP ’03).*, Apr. 2003, vol. 4, p. IV–752. doi: 10.1109/ICASSP.2003.1202752.

[126] M. Haller *et al.*, “The NiCE Discussion Room: Integrating Paper and Digital Media to Support Co-Located Group Meetings,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, Apr. 2010, pp. 609–618. doi: 10.1145/1753326.1753418.

[127] Microsoft, “OneNote Digital Note Taking App | Microsoft 365.” https://www.microsoft.com/en-us/microsoft-365/onenote/digital-note-taking-app (accessed Dec. 09, 2021).

[128] “Online Notepad | Best Note-Taking App | Evernote.” https://evernote.com/ (accessed Aug. 13, 2022).

[129] “Notes on the App Store.” https://apps.apple.com/us/app/notes/id1110145109 (accessed Aug. 13, 2022).

[130] “Google Keep.” https://keep.google.com/u/0/ (accessed Aug. 13, 2022).

[131] “Notion – One workspace. Every team.” https://www.notion.so/product?utm_source=google&utm_campaign=2075789710&utm_medium=80211061601&utm_content=372709093345&utm_term=notion&targetid=kwd-312974742&gclid=Cj0KCQjwl92XBhC7ARIsAHLl9anpiuLOcy8c2QYGPRk56Co-4k9V7DmiqPEyiuGFGI1edeBuSgEUIwIaAv6HEALw_wcB (accessed Aug. 13, 2022).

[132] “Obsidian.” https://obsidian.md/ (accessed Aug. 13, 2022).

[133] Google, “Google Docs.” https://docs.google.com/ (accessed Dec. 09, 2021).

[134] “Microsoft Office is part of Microsoft 365.” https://www.microsoft.com/en-us/microsoft-365/microsoft-office (accessed Aug. 13, 2022).

[135] “Digital Whiteboard | MURAL,” 2022. https://mural.co (accessed Mar. 15, 2022).

[136] “Miro,” 2022. https://miro.com/ (accessed Jan. 14, 2022).

[137] “Jamboard,” *Wikipedia*. Jan. 09, 2022. Accessed: Jan. 14, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Jamboard&oldid=1064728847

[138] C. C. Dudding, “Digital Videoconferencing: Applications Across the Disciplines,” *Communication Disorders Quarterly*, vol. 30, no. 3, pp. 178–182, May 2009, doi: 10.1177/1525740108327449.

[139] D. M. Fetterman, “Research news and Comment: Videoconferencing On-line: Enhancing Communication Over the Internet,” *Educational Researcher*, vol. 25, no. 4, pp. 23–27, 1996.

[140] M. S. Ackerman, B. Starr, D. Hindus, and S. D. Mainwaring, “Hanging on the ‘wire’: a field study of an audio-only media space,” *ACM Trans. Comput.-Hum. Interact.*, vol. 4, no. 1, pp. 39–66, Mar. 1997, doi: 10.1145/244754.244756.

[141] H. Richter, “Understanding Meeting Capture and Access,” in *CHI ’02 Extended Abstracts on Human Factors in Computing Systems*, 2002, pp. 558–559. doi: 10.1145/506443.506480.

[142] R. Cutler *et al.*, “Distributed meetings: a meeting capture and broadcasting system,” in *Proceedings of the tenth ACM international conference on Multimedia*, New York, NY, USA, Dec. 2002, pp. 503–512. doi: 10.1145/641007.641112.

[143] S. R. Ahuja, J. R. Ensor, and D. N. Horn, “The rapport multimedia conferencing system,” *SIGOIS Bull.*, vol. 9, no. 2–3, pp. 1–8, Apr. 1988, doi: 10.1145/966861.45411.

[144] W. Geyer, H. Richter, L. Fuchs, T. Frauenhofer, S. Daijavad, and S. Poltrock, “A Team Collaboration Space Supporting Capture and Access of Virtual Meetings,” in *International ACM SIGGROUP Conference on Supporting Group Work*, 2001, pp. 188–196. doi: 10.1145/500286.500315.

[145] H. Richter, G. D. Abowd, W. Geyer, L. Fuchs, S. Daijavad, and S. Poltrock, "Integrating Meeting Capture within a Collaborative Team Environment," in *Ubicomp 2001: Ubiquitous Computing*, Berlin, Heidelberg, 2001, pp. 123–138. doi: 10.1007/3-540-45427-6_11.

[146] N. Yankelovich, W. Walker, P. Roberts, M. Wessler, J. Kaplan, and J. Provino, "Meeting central: making distributed meetings more effective," in *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, New York, NY, USA, Nov. 2004, pp. 419–428. doi: 10.1145/1031607.1031678.

[147] Zoom, "Video Conferencing, Cloud Phone, Webinars, Chat, Virtual Events," Dec. 09, 2021. https://zoom.us/ (accessed Dec. 08, 2021).

[148] "Google Meet." https://meet.google.com/ (accessed Aug. 15, 2022).

[149] Cisco, "Video Conferencing, Cloud Calling & Screen Sharing | Webex," Dec. 09, 2021. https://www.webex.com/ (accessed Dec. 08, 2021).

[150] Atlassian, "Jira | Issue & Project Tracking Software," *Atlassian*. https://www.atlassian.com/software/jira (accessed Aug. 15, 2022).

[151] "Zenhub - Productivity Management for Software Teams." https://www.zenhub.com/ (accessed Aug. 15, 2022).

[152] bugzilla.org, "Home :: Bugzilla." https://www.bugzilla.org/ (accessed Dec. 09, 2021).

[153] Atlassian, "Confluence - Team Collaboration Software," 2022. https://www.atlassian.com/software/confluence (accessed Dec. 09, 2021).

[154] G. Tur *et al.*, "The CALO Meeting Assistant System," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 6, pp. 1601–1611, Aug. 2010, doi: 10.1109/TASL.2009.2038810.

[155] S. Squartini and A. Esposito, "CO-WORKER: Toward Real-Time and Context-Aware Systems for Human Collaborative Knowledge Building," *Cogn Comput*, vol. 4, no. 2, pp. 157–171, Jun. 2012, doi: 10.1007/s12559-012-9136-5.

[156] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, New York, NY, USA, Nov. 2006, pp. 23–34. doi: 10.1145/1181775.1181779.

[157] T. Fritz and G. C. Murphy, "Using information fragments to answer the questions developers ask," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, May 2010, vol. 1, pp. 175–184. doi: 10.1145/1806799.1806828.

[158] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web? (NIER track)," in *Proceedings of the 33rd International Conference on Software Engineering*, New York, NY, USA, May 2011, pp. 804–807. doi: 10.1145/1985793.1985907.

[159] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing, "What do developers search for on the web?," *Empir Software Eng*, vol. 22, no. 6, pp. 3149–3185, Dec. 2017, doi: 10.1007/s10664-017-9514-4.

[160] J. Burge and D. Brown, "SEURAT," in *2008 ACM/IEEE 30th International Conference on Software Engineering*, May 2008, pp. 835–838. doi: 10.1145/1368088.1368215.

[161] A. Girgensohn and F. M. Shipman, "Supporting knowledge acquisition by end users: tools and representations," in *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing: technological challenges of the 1990's*, 1992, pp. 340–348.

[162] S. Vasanthapriyan, J. Tian, and J. Xiang, "A Survey on Knowledge Management in Software Engineering," in *Reliability and Security - Companion 2015 IEEE International Conference on Software Quality*, Aug. 2015, pp. 237–244. doi: 10.1109/QRS-C.2015.48.

[163] J. Ward and A. Aurum, "Knowledge management in software engineering - describing the process," in *2004 Australian Software Engineering Conference. Proceedings.*, Apr. 2004, pp. 137–146. doi: 10.1109/ASWEC.2004.1290466.

[164] A. Aurum, R. Jeffery, C. Wohlin, and H. Meliha, Eds., *Managing Software Engineering Knowledge*. Springer-Verlag, 2003.

[165] M. A. Babar, X. Wang, and I. Gorton, "PAKME: A Tool for Capturing and Using Architecture Design Knowledge," in *Pakistan Section Multitopic Conference*, 2005, pp. 1–6. doi: 10.1109/INMIC.2005.334419.

[166] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, "10 years of software architecture knowledge management: Practice and future," *Journal of Systems and Software*, vol. 116, pp. 191–205, Jun. 2016, doi: 10.1016/j.jss.2015.08.054.

[167] A. Jansen, P. Avgeriou, and J. S. van der Ven, "Enriching Software Architecture Documentation," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1232–1248, 2009, doi: 10.1016/j.jss.2009.04.052.
[168] F. Sousa, M. Aparicio, and C. J. Costa, "Organizational Wiki as a Knowledge Management Tool," in *28th ACM International Conference on Design of Communication*, 2010, pp. 33–39. doi: 10.1145/1878450.1878457.
[169] R. M. Akscyn, D. L. McCracken, and E. A. Yoder, "KMS: a Distributed Hypermedia System for Managing Knowledge in Organizations," *Communications of the ACM*, vol. 31, no. 7, pp. 820–835, 1988, doi: 10.1145/48511.48513.
[170] G. Fischer, R. McCall, and A. Morch, "JANUS: Integrating Hypertext with a Knowledge-based Design Environment," in *Second Annual ACM Conference on Hypertext*, 1989, pp. 105–117. doi: 10.1145/74224.74233.
[171] "What are lightweight Architecture Decision Records? | Packt Hub." https://hub.packtpub.com/what-are-lightweight-architecture-decision-records/ (accessed Jun. 10, 2022).
[172] T. Deshpande, "A Simple but Powerful Tool to Record Your Architectural Decisions," *Better Programming*. https://betterprogramming.pub/here-is-a-simple-yet-powerful-tool-to-record-your-architectural-decisions-5fb31367a7da (accessed Dec. 12, 2021).
[173] M. Polanyi, "The Tacit Dimension," in *Knowledge in Organizations*, Routledge, 1997.
[174] I. Nonaka and D. J. Teece, *Managing Industrial Knowledge: Creation, Transfer and Utilization*. SAGE, 2001.
[175] I. Rus, M. Lindvall, and S. Sinha, "Knowledge management in software engineering," *IEEE software*, vol. 19, no. 3, pp. 26–38, 2002.
[176] H. van Vliet, "Software Architecture Knowledge Management," in *19th Australian Conference on Software Engineering (aswec 2008)*, Mar. 2008, pp. 24–31. doi: 10.1109/ASWEC.2008.4483186.
[177] P. Kruchten, P. Lago, and H. van Vliet, "Building Up and Reasoning About Architectural Knowledge," in *Quality of Software Architectures*, Berlin, Heidelberg, 2006, pp. 43–58. doi: 10.1007/11921998_8.
[178] V. Clerc, P. Lago, and H. van Vliet, "The Architect's Mindset," in *Software Architectures, Components, and Applications*, Berlin, Heidelberg, 2007, pp. 231–249. doi: 10.1007/978-3-540-77619-2_14.
[179] U. van Heesch and P. Avgeriou, "Mature Architecting - A Survey about the Reasoning Process of Professional Architects," in *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, Jun. 2011, pp. 260–269. doi: 10.1109/WICSA.2011.42.
[180] A. Jansen, J. Bosch, and P. Avgeriou, "Documenting after the fact: Recovering architectural design decisions," *Journal of Systems and Software*, vol. 81, no. 4, pp. 536–557, Apr. 2008, doi: 10.1016/j.jss.2007.08.025.
[181] H. Yan, D. Garlan, B. Schmerl, J. Aldrich, and R. Kazman, "DiscoTect: a system for discovering architectures from running systems," in *Proceedings. 26th International Conference on Software Engineering*, May 2004, pp. 470–479. doi: 10.1109/ICSE.2004.1317469.
[182] M. Pinzger, M. Fischer, H. Gall, and M. Jazayeri, "Revealer: a lexical pattern matcher for architecture recovery," in *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.*, Nov. 2002, pp. 170–178. doi: 10.1109/WCRE.2002.1173075.
[183] L. Chen, M. A. Babar, and H. Liang, "Model-Centered Customizable Architectural Design Decisions Management," in *2010 21st Australian Software Engineering Conference*, Apr. 2010, pp. 23–32. doi: 10.1109/ASWEC.2010.31.
[184] A. Rüping, *Agile documentation: a pattern guide to producing lightweight documents for software projects*. John Wiley & Sons, 2005.
[185] S. Voigt, D. Hüttemann, and A. Gohr, "sprintDoc: Concept for an agile documentation tool," in *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*, Jun. 2016, pp. 1–6. doi: 10.1109/CISTI.2016.7521550.
[186] "GitHub: Where the world builds software," *GitHub*. https://github.com/ (accessed Aug. 15, 2022).
[187] J. Lee, "Design Rationale Systems: Understanding the Issues," *IEEE EXPERT*, p. 8, 1997.
[188] W. Kunz and H. W. Rittel, "Issues as elements of information systems (Working Paper 131)," *Center for Planning and Development Research, Berkeley, USA*, 1970.
[189] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge, "How do developers discuss rationale?," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Mar. 2018, pp. 357–369. doi: 10.1109/SANER.2018.8330223.

[190] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrik, *Rationale-Based Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. doi: 10.1007/978-3-540-77583-6.

[191] R. McCall, "Using Argumentative, Semantic Grammar for Capture of Design Rationale," in *International Conference on Design Computing and Cognition*, 2018, pp. 519–535. doi: 10.1007/978-3-030-05363-5_28.

[192] D. and R. Noble, "Issue-Based Information Systems for Design," 1988. Accessed: Oct. 14, 2021. [Online]. Available: http://papers.cumincad.org/cgi-bin/works/BrowseTreefield=seriesorder=AZ/Show?ca71

[193] S. Buckingham Shum, A. M. Selvin, M. Sierhuis, J. Conklin, C. B. Haley, and B. Nuseibeh, "Hypermedia Support for Argumentation-Based Rationale: 15 Tears on from gIBIS and QOC," in *Rationale Management in Software Engineering*, A. Dutoit, R. McCall, I. Mistrik, and B. Paech, Eds. Berlin: Springer-Verlag, 2006, pp. 111–132.

[194] M. Bhat, K. Shumaiev, and F. Matthes, "Towards a Framework for Managing Architectural Design Decisions," in *11th European Conference on Software Architecture: Companion Proceedings*, 2017, pp. 48–51. doi: 10.1145/3129790.3129799.

[195] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. Ali Babar, "A comparative study of architecture knowledge management tools," *Journal of Systems and Software*, vol. 83, no. 3, pp. 352–370, 2010.

[196] F. Gilson, S. Annand, and J. Steel, "Recording Software Design Decisions on the Fly.," in *Joint Proceedings of SEED & NLPaSE*, 2020, pp. 53–66.

[197] A. M. Soria, A. van der Hoek, and J. Burge, "Recurring Distributed Software Maintenance Meetings: Toward an Initial Understanding," in *2022 IEEE/ACM 15th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, May 2022, pp. 21–25. doi: 10.1145/3528579.3529179.

[198] U. Kuckartz and S. Rädiker, *Analyzing Qualitative Data with MAXQDA: Text, Audio, and Video*. Springer, 2019.

[199] L. Xu, S. A. Hendrickson, E. Hettwer, H. Ziv, A. van der Hoek, and D. J. Richardson, "Towards supporting the architecture design process through evaluation of design alternatives," in *Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, New York, NY, USA, 2006, pp. 81–87. Accessed: Oct. 15, 2021. [Online]. Available: https://doi.org/10.1145/1147249.1147260

[200] A. Tang, J. Han, and R. Vasa, "Software Architecture Design Reasoning: A Case for Improved Methodology Support," *IEEE Software*, vol. 26, no. 2, pp. 43–49, 2009.

[201] A. Tang and H. van Vliet, "Modeling constraints improves software architecture design reasoning," in *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture*, Sep. 2009, pp. 253–256. doi: 10.1109/WICSA.2009.5290813.

[202] G. Post and A. Kagan, "OO-CASE tools: an evaluation of Rose," *Information and Software Technology*, vol. 42, no. 6, pp. 383–388, Apr. 2000, doi: 10.1016/S0950-5849(99)00099-3.

[203] H. M. Harmain and R. Gaizauskas, "CM-Builder: an automated NL-based CASE tool," in *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*, Sep. 2000, pp. 45–53. doi: 10.1109/ASE.2000.873649.

[204] R. Acerbis, A. Bongio, M. Brambilla, and S. Butti, "WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications," in *Web Engineering*, Berlin, Heidelberg, 2007, pp. 501–505. doi: 10.1007/978-3-540-73597-7_44.

[205] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: state of the art, current trends and challenges," *Multimed Tools Appl*, Jul. 2022, doi: 10.1007/s11042-022-13428-4.

[206] L. Mich, "NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA," *Natural language engineering*, vol. 2, no. 2, pp. 161–187, 1996.

[207] L.-C. Ungureanu and T. Hartmann, "Analysing frequent natural language expressions from design conversations," *Design Studies*, vol. 72, p. 100987, Jan. 2021, doi: 10.1016/j.destud.2020.100987.

[208] B. Huber, S. Shieber, and K. Z. Gajos, "Automatically Analyzing Brainstorming Language Behavior with Meeter," *Proc. ACM Hum.-Comput. Interact.*, vol. 3, no. CSCW, p. 30:1-30:17, Nov. 2019, doi: 10.1145/3359132.

[209] S. Sheshadri, "Identifying Action related Dialogue Acts in Meetings: Research into identifying Action Items, Decisions, and Ideas from multi-party meeting transcripts." 2019.

[210] M. Majthoub, M. H. Qutqut, and Y. Odeh, "Software Re-engineering: An Overview," in *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, Jul. 2018, pp. 266–270. doi: 10.1109/CSIT.2018.8486173.

[211] S. K. Chang, *Handbook of Software Engineering and Knowledge Engineering*. World Scientific, 2001.

[212] J. Janák, "Issue tracking systems," *Brno, spring*, p. 17, 2009.

[213] L. A. LeBlanc and M. R. Nosik, "Planning and Leading Effective Meetings," *Behav Analysis Practice*, vol. 12, no. 3, pp. 696–708, Sep. 2019, doi: 10.1007/s40617-019-00330-z.

[214] T. Shellenbarger and J. Chicca, "More meaningful meetings," p. 3.

[215] A. Adams and M. A. Sasse, *Privacy Issues in Ubiquitous Multimedia Enviroments: Wake Sleeping Dogs, or Let Them Lie?* IOS Press, 1999.

[216] A. M. Soria and A. Van Der Hoek, "The Design of a Study Concerning the Capture of Important Design Bits at the Whiteboard," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, Oct. 2021, pp. 390–399. doi: 10.1109/MODELS-C53483.2021.00062.

[217] A. Meza Soria and A. van der Hoek, "Toward Collecting and Delivering Knowledge for Software Design at the Whiteboard," in *2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, May 2018, pp. 108–109.

[218] A. Meza Soria and A. van der Hoek, "Collecting Design Knowledge through Voice Notes," in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, May 2019, pp. 33–36. doi: 10.1109/CHASE.2019.00015.

[219] C. Lebeuf, M.-A. Storey, and A. Zagalsky, "Software Bots," *IEEE Software*, vol. 35, no. 1, pp. 18–23, Jan. 2018, doi: 10.1109/MS.2017.4541027.

[220] S. Gasson, "Rigor in Grounded Theory Research: An Interpretive Perspective on Generating Theory from Qualitative Field Studies," *The Handbook of Information Systems Research*, 2004.

[221] D. I. K. Sjoeberg *et al.*, "A survey of controlled experiments in software engineering," *IEEE Transactions on Software Engineering*, vol. 31, no. 9, pp. 733–753, Sep. 2005, doi: 10.1109/TSE.2005.97.