IT UNIVERSITY OF COPENHAGEN

What happens in Triage?

An Empirical Study of Bug Triage in Software Product Evolution?

Marjahan Begum
Yvonne Dittrich

IT University Technical Report Series TR-2024-210

ISBN 978-87-7949-008-3 December 2024

Copyrigth © 2024 Marjahan Begum
Yvonne Dittrich

IT University of Copenhagen
All rights reserved.

Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.

ISBN 978-87-7949-008-3

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK – 2300 Copenhagen S
Denmark

Telephone: +45 72 18 50 00
Telefax: +45 72 18 50 01
Web: www.itu.dk

What happens in Triage?
An Empirical Study of Bug Triage in Software Product Evolution

Marjahan Begum
School of Computer Science
University of Nottingham, UK
marjahan.begum@nottingham.ac.uk

Yvonne Dittrich
Department of Computer Science)
IT University of Copenhagen
Copenhagen, Denmark ydi@itu.dk

***Abstract*—This paper investigates and evaluates the role of bug Triage in software evolution and maintenance. Traditionally, Triage decision-making has been based on bug reports. Decision-making concerns whether a bug is to be fixed and, if so, when and by whom. Research in this area focuses on automation of some aspects of bug fixing, enhancing information on bugs reports, and most significantly, automating Triage through Machine Learning (ML) techniques. Our paper is based on an ethnographic study of a software team, and includes Triage and Stand-up meeting observation, analysis of bug report documents, study of the development environment and ad-hoc meetings. The framework of Distributed Cognition for Teamwork served as a theoretical lens for this study. Based on the analysis 33 complex bugs, the paper argues that Triage was used as a major information hub for discussing a wide range of information (e.g. about organisational processes and development of the software), allowing knowledge development that is valuable in software evolution beyond bug fixing alone.**

***Index Terms*—software engineering, empirical studies, collaborative and social computing, DiCoT**

I. Introduction

Fixing bugs is an important part of software evolution and maintenance. However, it is also considered cumbersome and work-intensive. In many cases, work on bugs begins with Triage, where software engineers and domain experts jointly decide whether and how a reported bug should be fixed, its priority, and to which engineer or domain expert it should be assigned. Triage is a decision-making process that requires significant organisational resources.

This article argues that Triage is not only about deciding how, when and who is best-qualified to fix a bug. Our analysis of what actually happens in Triage finds that it yields much richer information for software evolution. Therefore, it complements other research on bug resolution, including ML approaches [1]–[4], by focusing on the research question, What types of knowledge are generated in the Triage of complex bugs?

The analysis suggests that Triage is a major information hub one of the principle concept of information flow in Distributed Cognition for Teamwork (DiCoT) [5]. The information discussed in Triage has a wide range of implications, including for organisational processes and future development beyond deciding whether, when, how, or by whom a bug should be fixed. Through the discussions, the participants share knowledge and maintain their understanding of why the software was developed in a particular way (past), its current usage and problems (present), and how it should be developed and changed to meet customer needs (future). In other words, Triage raises awareness and yields knowledge that is important for present and future development. This value is hidden from researchers if the focus of the research is on reducing time spent. Machine learning (ML) algorithms may be used to support the Triage (e.g. categorising complex bugs for Triage, whereas simple bugs are automatically assigned to engineers).

This paper presents a seven-month ethnographic study that focused on software maintenance at the Warehouse and Transportation division of some enterprise resource planning (ERP) software. Data were collected through offline and online observation of Triage and Stand-up meetings, interviews with engineers, and by analysing the information in the related software repository. The results were discussed with members of the Triage team. Distributed cognition [6] provided the framework for the analysis.

The importance of understanding social aspects of software engineering has been thoroughly argued, for example, by Storey and her colleagues [7]. They show that, in 2017, on average, only five articles published by the ICSE and EMSE communities addressed descriptive field studies. This study provide descriptive insights into the way Triage operates, thus extending descriptive and qualitative research with professional software engineers in their everyday work environment.

This paper is structured as follows. Section II provides background of the study. Section III explains the paper's theoretical underpinning. Section IV presents the research method with details of the case, field work and data-collection, and data analysis. It also discusses the trustworthiness of the data and research triangulation, member checking, prolonged engagement, audit trails and research ethics. Section **??** describes the bug's journey through the lens of DiCoT. Section VI presents the analysis of DiCoT's information-flow

in Triage. Section VII discusses the findings. Section VIII offers conclusions and directions for future research.

II. Background

Fixing bugs has long been a subject of research in software maintenance and evolution. This section contextualises our investigation by presenting three relevant types pf research into fixing bugs: enhancing quality of bugs reports; using ML approaches for automation; coordination and collaboration.

Bug reports are important electronic artefacts for resolving bugs. They contain basic descriptions of the bug, related bugs, associated work items and sometimes code snippets, among other information. One of the most important items of information is the 'Steps to Reproduce' (S2R). S2R gives the runtime behaviour of a bug and the problems faced from the end-users perspectives. Good bug reports support decision-making in Triage. As bug reports include vast amounts of information, it is important to identify the information most relevant to fixing bugs. Bettenburg et al [8] investigated the quality of bug reports with quantitative and qualitative surveys of 466 open-source developers. They found that S2Rs, stack traces, and test cases provided the most helpful information. This information is also very difficult to provide in a bug report. They used these insights to build a ML model to predict the quality of 289 bug reports with an accuracy of between 31% and 48%, and this provided feedback to the bug reporter. Chaparro et al. [9] explored the quality of S2R using language analysis, matching sentences with a neural sequencing labelling model in the steps leading to the application interactions. A low match was attributed to a S2R having multiple or no matching screen elements, or requiring additional steps that were not documented in the report. The technique gives feedback on the clarity of the information and on any missing steps.

In recent years, most of the research on bug Triage has concerned the use of ML techniques to assign bugs to developers to fix [1], [10]–[14] and to prioritise bugs [12] to reduce time in Triage. Various types of information were used for the ML models. Naguib et (2013) et al [15] profiled the user of a bug tracking system based on engineers' activities, to see whether they are assigning and/or resolving bugs, and combined this with engineers' expertise in prediction. Xuan et al. proposed a text-classification approach to automate Triage through data-reduction techniques [16]. Condensing the bug report to only relevant information improves the bug report from a ML perspective, as it does not have to incorporate irrelevant variables. Jeong et al [11] combined ML with bug tossing graphs, which detail a graph from a bug to the fixer. Most of the ML approaches used supervised learning, and the accuracy of the predictions ranged from 30% [14] to 98% [4] in some cases. Anvik et. al [4] corroborated their findings with human triagers.

ML driven research assumes that all relevant information on bugs may be made explicit with the help of the project repository, and that ML algorithms can identify the required information, and provide recommendations that support bug fixing [17]. This assumption is challenged by Aranda et al. [3]. They demonstrate that information is distributed among various parts of the repository, and of course resides with individual developers. Co-workers are the most frequent source of information, and if they are unavailable, progress is slow [18]. They concluded that the distributed nature of the information should be taken into consideration when assessing the quality of a bug report or discussing a bug in Triage. Making Triage decisions based solely on a bug report may be counterproductive, when some key information is not explicitly available. Carstensen et al. researched the coordination of the work surrounding bug fixing, and identified the bug report as a central coordinating mechanism [19], [20]. The patterns surrounding bug fix and the behind-the-scenes automated information were later explored by Aranda et al. [3]. They categorised coordination activities, related communication, bug databases, coding, reviewing code, and meetings such as Triage. They suggest that Triage and code review were perceived as the most essential coordination patterns, and are strongly dependent on the social, organisational and technical knowledge of individual developers [18]. Another coordination activity that has been researched is bug reassignment in Open Source Software [21]: the main reasons for reassignment are identification of the root cause, ambiguous ownership, a poor quality bug report, difficulty in determining a proper fix or balancing a workload between developers.

The research discussed in this section confirms that information in the bug report needs to be relevant. Several articles propose the use of automation to enhance information and to sift out unnecessary elements. However, few articles explore the dynamics of the collaboration surrounding bug fixing and explore Triage as an important coordination activity for resolving bugs.

III. Theoretical Underpinnings

Distributed Cognition (DCog) and Distributed Cognition for Teamwork (DiCoT) theory and framework [5], [6], [22] provide a lens for analysing our field work. DCog advocates an understanding of how an entire environment, its physical artefacts and information, are used to coordinate and execute tasks that are regarded as involving cognition. Cognitive processes are often distributed among the members of a social group, and require coordination of the internal cognitive activity and the external (physical, material or environmental) structure. Similarly, cognitive processes may be cognitive processes may be distributed over time in such a way that the products of earlier events may effect the nature of later events [6].

Grounded on DCog, DiCoT [5] was developed to explore the understanding of the information flow during team communication and collaboration in the context of software engineering and other socio-technical domains [5], [23]. DiCoT was developed around three main themes or groups of concepts: physical layout, artefact and information flow.

A. *Physical layout*

Physical layout includes equipment, artefacts and people working in the environment, and how their placement affects

what team members can access, hear and see. An example of where the Scrum board is located and its role and function in Agile development context [24].

Artefacts are enablers, mediators and/or are used simply to monitor progress [5]. Therefore, they have specific roles depending on the context of the respective task. As our observations and part of the work of interacting with the software repository were mainly virtual, the virtual artefact concept is more relevant to this study. Virtual artefacts may include and reference other artefacts [23]. Information resides in artefacts and flows among them.

Information flow is another core theme that helps to understand how teams cooperate to execute cognitive tasks. As the information flow theme turned out the main theme in the analysis, the individual elements are discussed in detail:

Information movement describes how information is moved between subsystems of complex systems [25]. Information moves for functional reasons and results in different representation and physical transformation of artefacts. The mechanism can be physical movement of the artefacts or change of the representation of the information (text, graphical, verbal, facial expression, telephone, electronics, alarms).

Information transformation occurs when the representation changes from one form to another. The representation is to aid reasoning and problem solving. An example is the process of filtering where information is gathered, structured and sifted so that it would aid problem solving for the next agent in the pipeline. Flor and Hutchins [26] show how software engineers reused previous code, shared goals and plans, negotiated various approaches to problem-solving, solved problems in a shared space, and collaboratively searched through large alternative solution spaces.

Information hubs are team meetings where different information channels meet and information is processed and results in information transformation. Such information hubs have a high communication bandwidth [5]. An example are Sprint planning meetings where decisions are made concerning what software features will be developed. They have a high communication bandwidth because information from various sources (channels) is brought together for discussion. Without such meetings, coordination concerning, and collaboration on the work item would be compromised.

Buffering refers to the storage of information from various sources or channels for a period of time, until decisions are made or there is further information transformation and movement.

DCog and DiCoT have been used to study software engineering [23], [24], [26] due to the complex nature of software engineering, and the nature of interactions between people and artefacts. They provide a lens for analysing interactions as a whole or part of bug fixing process. Sharp et al.'s study of virtual and geographically dispersed settings [23], they explore the role of key virtual artefacts (e.g. OneNote, Microsoft Visual Studio Team Foundation Server (TFS), Skydrive). For example, OneNote was used for scaffolding when it was adopted as a way to outline behaviour of the software.

To summarise, DCog and DiCot are powerful tools for analysing and understanding complex collaboration. In line with Sharp et al. [23], we use the DCog underpinning and DiCoT's terminology to analyse collaboration around the bug fixing process at a software product development organisation.

IV. Research Method

The purpose of this study was to better understand the bug handling process and the information it requires by investigating what information is brought to and generated in Triage and how Triage is done, from the members' perspective by focusing on the rationale behind various activities. Accordingly, this was designed as an ethnographic study [27]. The fieldwork was carried out by the first author. The second author supported the fieldwork through regular debriefing and participated in the data analysis detailed below. The field work began in February 2020 on-site, and then moved online during the COVID-19 lock-down. To understand the complex interaction between people and virtual artefacts, the information flow from DiCoT as described above was used as an analytical lens, as described in section IV-C

A. Case Description

This study was carried out in Microsoft with a team that was responsible for the Warehouse and Transportation module of Microsoft Dynamics Enterprise Resource Planning (ERP). This software is highly customisable and may be configured to customers' specific needs. The team adapted a variation of agile software development model, with sprints, retrospectives and backlogs that comprised of work items, including bugs. Bug fixes were prioritised, and often resolved within a few days.

The development of the ERP system is distributed over several countries: USA, China, Denmark and Ukraine. The study focused on a team that develops and maintains the Warehouse and Transportation modules of the system. The team consists of four to five software engineers (SEs), two to three senior software engineers (SSEs) and three program managers (PMs). They were employed by the main company. There were also five to six SE employed by the subcontractor, based in a different country. Employees in the subcontractor's company are treated in the same way as the main team members. SEs are directly involved in fixing bugs, whereas SSEs have advisory roles, as they have extensive experience of the code base. SSEs have over five years of experience, whereas PMs are very knowledgeable about the product from a business perspectives. PMs act as an interface between design that involves important domain knowledge centred around use cases, and development. This team develops new features as well maintenance of the software which includes fixing bugs. In the context of bug Triage and fixing, the team collaborates with support engineers, and the teams responsible for other modules of the software that has dependencies with Warehouse and Transportation modules as necessary.

The main collaboration and coordination environment used by Microsoft is Azure DeveOps, which is integrated into the

engineers' development environments, Code Flow (for code review) and Version Control Systems.

Bug reports are stored as work items in databases. Bug reports consist of a unique number, title and description, S2R (Steps to Reproduce), indicators of severity and priority, and links to related bugs and/or deliverables. The priority to the customer. Severity indicates how often the scenario is used. S2R is step-by-step instruction on how to reproduce the bug in the local environment with reference to the Demo Application, a specific version of the application, with demo data

B. Field work and data collection

In line with ethnographic research traditions, our field work began with immersion, from February 3rd to March 23rd, 2020. This began with a meeting with the Senior Architect in charge of the focal team. This meeting focused on understanding the team structure and the organisation's general software engineering process. The Stand-up and Triage meetings were the key meetings related to bug fixing. At 09:00 stand-up meetings began, followed by Triage meetings. Stand-up meetings averaged 10 minutes, whereas Triage meetings lasted 10 to 30 minutes. There were 18 onsite observations between February 10th and March 23rd which provided access to the daily working of the teams, to understand the context to build trust which proved vital when the researcher asked for permission to record Triage and Stand-up meetings, when development work moved online during the Covid 19 lockdown.

TABLE I
FIELDWORK

Method	Date(s)	Documentation
Meeting, Senior Architect	Feb 3, 2020	Field notes, whiteb. photo
Observation, 18 stand-up meetings and 18 Triages	Feb 10 – Mar 23, 2020	Field notes
Virt. observ., 12 stand-up meetings and 12 Triages	Mar 24 – Apr 8, 2020	Field notes, transcripts
Interview, Senior Engineer	Apr 30, 2020	Transcript
Interview, Senior Architect	June 26, 2020	Notes from recordings
Clarification interview, Senior Engineer	July 20, 2020	Transcript
Desk research of development environment	until Aug 2020	Field notes

Detailed observation notes were taken. As the discussions in these meetings focused on individual work items and bugs, where possible, bug numbers/work items (unique numbers) were taken and the gist of the conversation was noted without any set protocols. Ad hoc conversations with a core SSE in the team provided an understanding of relevant parts of the development environment.

Studying software engineering in a large organisation is a challenge. The DiCoT framework helped us focus on identifying where in the processes there was evidence of collaboration. The observation showed that during the Triage meetings there were important exchanges that went far beyond decisions regarding which bug should be fixed, and by whom. The researcher decided to focus on Triage as it was one of the core collaborative activities for fixing bugs. Following onsite observations, the field notes were reviewed to identify where any clarification was needed from the SEE, the bug database was accessed to follow specific bugs and the information shared was studied.

Following this phase, the characteristic of the field work changed, due to the pandemic: software development and maintenance activities moved online and used Microsoft Teams for meetings, and the research followed accordingly. From March 24th to April 8th, Stand-up and Triage meetings were recorded and transcribed, in order to capture their detailed references to various sources of information. This comprised 12 hours of Triage and Stand-up meeting observation (see Table I). Researchers supplemented meeting observation with the study of the information about the bugs discussed in the meetings. This desk research amounted to about 125 hours, distributed over the entire period.

When the principal researcher discovered that the observation did not yield any new knowledge about bug resolution, the observation was terminated and the focus shifted to analysing the data. During this analysis, the researcher interacted with the main SSE daily, to clarify the meaning of the recorded data. Several formal interviews (4 hours) were conducted to clarify details about the bug handling process and to obtain an in-depth understanding of the background of some of the bugs. These interviews functioned as member checking to corroborate and develop our understanding of the information shared during meetings.

Though not used as an explicit framework for note-taking, it was soon evident that the DiCoT's information flow theme, specifically the information hub, was very relevant for the analysis of the field material.

C. Data Analysis

Data analysis began in parallel with data collection, allowing the adaptation of the fieldwork to further explore the evolving findings. During the immersion phase (February 3rd to March 23rd), the complexity and importance of the information that flowed from various artefacts and people was crucial to decision-making that concerned how bugs should be fixed. Information flow, a fundamental theme of DiCoT, became the analytical lens for studying Triage.

To analyse the information discussed in greater detail, Triage and Stand-up recordings were transcribed verbatim and contextual information was added, for example, bug numbers or the focus of the screen, where it was relevant to understanding the discussion. This contextual information helped to identify specific bugs in the bug database.

As the bug was the core unit of discussion in Triage meetings, the analysis also focused on the paths of individual bugs. All the transcript data associated with each unique bug was collated including relevant contextual information. During the 12 days of Triage and Stand-up meetings, 85 bugs were followed from when they reached the development team to their resolution. In reading the transcripts several times, it

became apparent that some bugs required repeated Triage and long discussions. We began to call these bugs 'complex bugs': a bug is complex if it requires Triage several times, and/or additional background investigations were required for Triage to reach the correct decision. Thus, we identified a subset of 33 complex bugs.

The Triage discussions of complex bugs were then subject to a thematic analysis: the transcripts were open-coded, that is, the themes were identified in a bottom-up way. These codes were discussed, categorised and consolidated through several iterations. This resulted in the identification of the four categories of information that are presented in Section VI-C. A further analysis of each of these categories showed that the information discussed had a temporal dimension: for example past decisions were referenced and markers were communicated for future development.

D. Trustworthiness

We carefully designed our research to address factors that could undermine its validity [28], [29]. Below we detail the measures taken.

1) Data and researcher triangulation: We used a variety of data sources. The initial interview and observations were supplemented with recorded and transcribed observation. The analysis was supported by the desk research in the development environment. Interviews with core members of the team triangulated and deepened the understanding of specific bugs. The first author was responsible for the fieldwork and the analysis. During the fieldwork, debriefing meetings with the second author were held. The second author joined the thematic analysis of the Triage meetings. Disagreements regarding the coding were discussed and resolved, which in some cases resulted in changes to the coding scheme.

2) Member Checking: The interviews that were held following the observation phase served to clarify open questions and as a vehicle that enabled members to check the evolving analysis. The draft paper was shared with the Senior Architect and Senior Engineer, who were the contacts for this research. They clarified and confirmed the definition of complex bugs, and the description of the bug resolution process. They also confirmed the categories of knowledge we identified.

3) Prolonged engagement: As the fieldwork took place over an extended period. Over time, the first author, in particular, gained knowledge and insight into company terminology, which reduced the risk of misunderstanding. An example of an insight that may not be directly relevant to the focus of this research, but helped to understand SE processes, is that a bug is not resolved until it goes through the code review process and passes all tests. This ensured a reasonable understanding of how bugs are resolved, and the role and the flow of information in Triage and Stand-ups.

4) Audit trail and detailed description: The fieldwork was carefully documented. The detailed descriptions that underpin the findings provided below are meant to help the reader to follow – and criticise – the grounding of the discussion and insights from the field material.

5) Research ethics: We developed an information sheet and consent form detailing which data were to be collected, how anonymity and confidentiality would be maintained and addressed issues of data security. These documents were circulated to all team members, and they were encouraged to seek clarification if required. Data and excerpts from the research have been anonymised. All the names in the presentation of the results are changed. All recordings of the meetings and engineering work on bugs and interviews were deleted by 30 August 2020. The Triage and Stand-up transcriptions and details of the bugs were retained for cross-checking.

V. A Bug's Journey from the Perspective of DiCoT

In this section we describe the journey of the bug through DiCOT lense. From a DiCoT perspective, Azure DevOps is a virtual artefact. It contains other virtual artefacts, which in turn contain relevant information. The most important virtual artefact is the bug report, which has information about a bug and S2R. The information flow theme is also very important, as information flows between people and between people and artefacts. The information hub is where people meet to make important decisions with or without their being mediated by artefacts. Broadly, information transformation and/or movement from one source to another enables collaborative activity. Figure 1 details of a bug's journey.

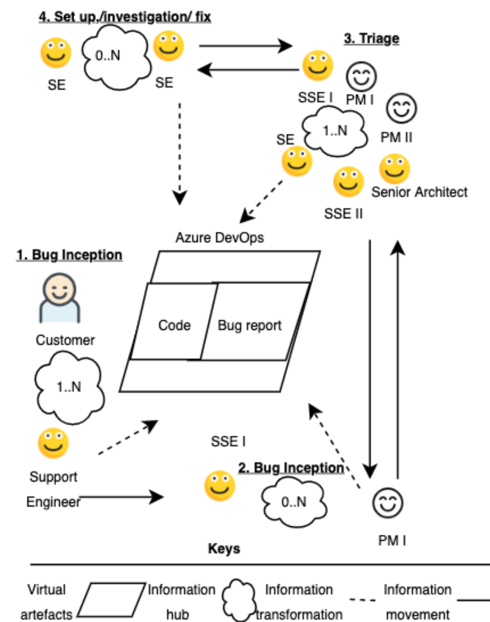

Fig. 1. Bug's Journey using DiCoT Framework

1) Bug Inception: A bug report is created as a result of consultation(s) between the customer, and a Microsoft Support Engineer or an Escalation Engineer. In some cases, the support team resolves the problems before a bug report is created. The Support Engineer's understanding of a bug involves

transforming the information into a bug report, which is then moved to the pre-triage information hub. During information transformation, the movement of information is implicit.

2) Pre-Triage: The SSE(s) and RM(s) collaboratively investigate the bugs, and prepare a bug by adding relevant information to the bug report (information transformation, dotted line to bug database), so that a Triage decision may be reached. The bugs' S2Rs are checked, and, for some bugs, new S2Rs are generated. If it is relevant to the bug report, related previous bugs and deliverables are linked. Sometimes, open issues on bugs are tagged to relevant team members. Tagged team members are then required to respond to the tagged issues before Triage begins. At this point, information is moving from pre-Triage to Triage. Pre-Triage is another information hub, because of its collaborative nature, and it is where information transformation and information movement are essential to decision-making. This is also true of Triage, which is introduced in the subsection below.

3) Triage: Once a bug report has the required information, it is discussed during a Triage meeting. Triage meetings take place daily, and average 20 minutes. A Triage team consists of PMs and SSE. The PMs are key persons, as they bring domain knowledge about how the software is supposed to behave. They act as an interface between the customer and the engineering team. During Triage, an informed decision is reached regarding whether the bug is 'to fix', 'to not fix', 'investigate' or if it is 'by design'. When the decision is 'to fix', the bug is taken up by the development team. Often, if the decision is 'by design', some documentation is updated and the customer is familiarised with how they may use the software to achieve their goal. In some cases, fixing the bug would require a fundamental design change that would need to be discussed in other contexts. The Triage is the most important information hub in a bug's journey, as it determines how to handle it. The information transformation and movement results in knowledge development. This will be explored in depth in Section VI.

4) Set-up, Investigation and Fixing : If a bug is marked as 'to fix', a SE selects the bug according to FIFO. The engineer first investigates the root cause, using the Demo application to reproduce the bug with the S2R. This stage helps to localise the bugs. The engineer reports on the progress and status (investigating, creating tests, pull request, waiting for review) of the bug at Stand-up meetings. These Stand-up meetings are held daily. Most changes are by editing the existing classes. For testing, new tests are added or old test classes are edited. These test classes are usually testing scenarios reported in the S2R. 'Scenario' is a term used by the company to describe a part of the functional requirements, and it is integral to deliverables in software development. Sometimes, manual smoke tests are done by playing around with the scenarios for the reported bugs. As soon as all formal tests are passed, a pull request is created. If further problems are encountered during the investigation phase, the engineer responsible for the bug contacts other team members to help them identify the correct strategy to fix it. This may require a number of meetings, emails and discussion. Once a pull request is created, the code review process starts in the Code Flow tool. On successful code review, the automatic testing cycles are initiated. If the tests are positive, the bug is closed.

From the DiCoT perspective, this part needs to be regarded as an information hub as well, as the complex nature of a fix is discussed between the senior engineer(s) and the engineer(s) who will fix the bug. So far, we have seen that there were four information hubs in a typical bug journey, and of those, we found Triage the central one.

VI. Analysis of Information Flow in Triage

Thirty-eight per cent of the bugs identified on our field work were complex bugs. In contrast to simple bugs, complex bugs require more than one Triage to decide how to resolve them. In this section, we first describe the history of a complex bug, then we present an analysis of the information categories referred to in the Triage meetings that concerned complex bugs whose Triage and resolution we were able to follow during field work. We use this to establish the basis for our discussion of the role of Triage.

A. The History of Bug no. 430105

Bug no. 403105 related to the movement of goods between warehouses. When goods are moved from warehouses that are not managed with the help of the ERP system, certain information is missing, which in turn affects the inventory functionality of the receiving managed warehouse. It took three Triage meetings and five Stand-ups to resolve this bug. Also, this bug was one of the few that was reactivated before being picked up by an engineer. Table II gives an overview of the events, starting with the first Triage meeting. The terms used in the table are company's own terms.

TABLE II
Timeline of Bug no. 430105

Date	Event	Result
Mar 19 2020	Opened	Opened by support engineer
Mar 20 2020	1st Triage	Investigation needed
Mar 23 2020	Resolved	S2R cannot be reproduced
Mar 24 2020	Re-activated	Re-opened by support engineer
Mar 24	Resolved	S2R cannot be reproduced
Mar 26 2020	Re-activated	Re-opened by support engineer
Mar 26 2020	2nd Triage	Fixing strategy, assigned for fixing
Mar 27 2020	Stand-Up	Ready for pick-up
Mar 30 2020	Stand-Up	SE reports on investigation
Mar 31 2020	Stand-Up	Report on further investigation, question whether to fix
Mar 31 2020	3rd Triage	Decision on work around; area declared as bug farm
Apr 1 2020	Stand-Up	SE announces pull request
Apr 2 2020	Stand-Up	Peer review prepared

During the first Triage, Bug no. 430105 was assigned the label 'Investigate', because in Pre-Triage stage, the SSE was unable to reproduce the bug based on the S2R. The SSE leading the investigation informed the SE to whom the

investigation was assigned. As a result of the investigation, the bug was marked 'resolved'. The bug was subsequently reactivated by the Support Engineer on March 24th, because the customer's end-users could not execute a business-critical task. On this same day, the SSE marked it as resolved a second time, because he was still unable to reproduce it using S2R.

After two days the bug was again re-activated by the Support Engineer. Then, during the second Triage, six days after it was first opened, the Triage leader and PM, Mathias, opened the discussion by summarising key information from the bug report and the S2R for the others. Together, Nicola, the SSE responsible for the bug, Jan and Mathias, both PMs, tried to clarify their understanding of the actual behaviour versus the intended behaviour. Mathias then further explained the use scenario. This pattern (clarifying and explaining) continued among Mathias, Jan and Nikola until the participants determined that they had reached a sufficient understanding. Finally, Mathias proposed fixing the problem by treating the goods with the missing information, as they do with purchased items. The bug was designated 'to be fixed'. At the Stand-up meeting the next day, a Friday, Nikola emphasised that bug no. 430105 needed to be addressed soon, as it took some time to reach a decision. At the Monday Stand-up meeting, Petre, a SE, informed the others that he had taken up the bug.

On Tuesday, Petre reported on the status of his investigation and on discussions with other team members and experts for specific domains: they discovered that the bug caused inconsistencies in the inventory data, and other bugs were found with similar problems. Nikola, the SSE, recommended further expert meetings, which in turn resulted in the realisation that the proposed fix did not work. The bug was returned for discussion during a third Triage meeting. During that Triage meeting, the same day, the bug was further discussed at length.

Two software engineers, Chris and Filip, and the PM, Mathias, discussed the problems the team encountered when fixing this bug. While investigating the bug fix, they discovered a serious error that caused data corruption in the functionality they wanted to use. They also emphasised that they did not have tests for the relevant functionality, which made it difficult to implement the change safely. In the discussion, they referred to the behaviour the code described, and the usage scenario that provided the requirements for the functionality. At some point, Per proposed taking the discussion to another meeting, as the technical details were not at a level of that was suited to a Triage meeting.

Filip spoke up again and explained that the problem was that the possible fix they developed with an expert on the data side of the ERP system resulted in ripple effects that affected other aspects of the functionality. They again engaged in several rounds of clarification and explanation. Filip summarised that the whole area might be problematic: *'It sounds like this is a bug farm, is kind of I guess what I am saying.'* A bug farm is an area in the code that is related to a number of bugs. Mathias confirmed that the problematic functionality was part of a problematic area – *'We have already have tracking dimension in the bug farm' – and that the redesign was already scheduled for future releases – '[...] we have plenty of deliverables in the bug farm for tracking number, so but...'*

At that point Nikola spoke up and questioned whether the proposed fix would not merely shift the problem to another part of the workflow. Nikola and Filip then directly discussed a possible temporary solution: *'So "Manual Movement" would be a temporary workaround for this case.'* Here, Mathias tried to conclude the discussion by asking about the next steps. Filip asked him to meet with Petre to agree on the details of the specific behaviour of the software for various connected scenarios, as this was a risky fix. Per agreed and stated, *'[...] this [part of the software] is definitively a bug farm that we have known about, right? [...]'*

Petre picked up the bug again on next day, and the day after reported that the pull request was ready for code review. Nicola and Petre briefly discussed who should do the code review for this fix, during the Stand-up.

B. Information movement and transformation in Triage

Applying the DiCoT framework the Triage meetings functioned as an information hub. In this section we present a rich analysis of the nature of *information transformation and information movement* [5] in Triage.

The participants brought their specific domain knowledge to the meeting, in order to make an informed choice about whether and how to fix a bug. In the discussion of Bug no. 430105, the most prevalent information sources were the bug report and the source code from the software engineering domain, and the use scenarios from the PM (*information-movement*). The latter two kinds of information were discussed in a tightly integrated fashion: The changes to the code were presented with respect to their effect on the use scenarios, and vice versa. This was done to the extent that sometimes it was difficult for us, as researchers, to distinguish whether they were talking about scenarios or the source code. Similarly, the results of meetings outside of Triage with other SE was shared (*information movement*). When reporting on the results of these meetings, the participants summarised the relevant information and highlighted the conclusions that were relevant to the Triage discussion. Similar sifting occurred when a longer discussion was summarised to prepare a decision (*information transformation*). Similar *sifting* occurred when a longer discussion was summarised to prepare a decision about information transformation.

However, information was not only shared, sifted and combined; in several cases, the Triage meeting was a place where a fix or, as emerged in the discussion above, a workaround was designed. Here, new information was generated to support the later implementation process. Similarly, the SE emphasised that ripple effects were to be avoided when fixing a bug. Interestingly, such future-related information generation was not only related to the specific, active bug. For example, at some point in the discussion, Mathias stated, *'We already have a tracking dimension in the bug farm. I mean we have plenty of deliverables in the bug farm for tracking. So but... '* He used the term deliverable to indicate features already specified for

future development. The reference to deliverable was used to argue that a quick fix or workaround was adequate until the new functionality was developed. At the same time, it served as a mental note to pay attention to the issue under discussion when redesigning this aspect. Similarly, the term bug farm was used. A bug farm refers to 'something that is a rich source of bugs' [30]. By declaring some area a bug farm, or by relating an error to an existing bug farm, the participants created a mental note to pay attention to this area in future, for example, when deciding on the development in future releases.

Finally, when Mathias scheduled meetings with some of the SE, he planned specific meetings to communicate information, in this case about the intended behaviour of the software from a domain perspective.

To summarise, the Triage meeting yielded not only a decision concerning whether, when and who should fix Bug no. 430105, but information from various domains and the results of other meetings were also shared and connected, bug fixes and workarounds were developed, notes for future development were taken, and problematic areas of the code were marked for future, more systematic refactoring.

Up through this section we presented what happened in Triage for a specific complex bug. This understanding could be developed due to DiCoT. DiCoT made it possible to identify Triage as an important *information hub*. The concept information transformation and information movement [5] enabled us to investigate from whom and from where information is brought to the Triage and explore the nature of the information and knowledge developed. Therefore, we needed to examine the nature of the information brought to and created in Triage. The following subsection presents an analysis of the information of all complex bugs that were discussed in the observed Triage meetings.

C. Categories of Information Discussed in Triage

The previous section concluded that what happens during Triage goes beyond assigning bugs; information is shared and information for future development is generated. In this section the analysis digs deeper into the categories of information that were discussed during the Triage of the 33 complex bugs we analysed. We categorised the information shared in the Triage meetings as technical, domain-related, organisational processes and business-context-related. Below, we discuss each category of information we found, and provide examples.

1) Technical: This category comprises information related to the technical side of a bug and how to fix it. These kinds of information range from discussing the code, test cases, whether the fix would break previously-working functionality, data integrity and data corruption, the discussion of parts of the code as 'bug farms', and ripple effects, that is, the impact of a bug fix on other areas of the code. The discussion connected the current problem and how to fix it to both past development and to how the fix may affect future developments. The discussion of backports, that is, integrating a change to a recent release into an older version, is an example of knowledge of past development. Backports may also be discussed when a bug is found and fixed in the current release, and the fix should also be implemented in earlier releases that are still covered by the support agreement with customers. In some cases, the Triage team was unsure whether a possible fix would risk breaking other functionality. The discussion of the data integrity in the analysis of Bug 430105 in section VI-A is a good example for the discussion of the ripple effects of a bug fix. The statement below is another example from our fieldwork:

'First one here is a very technical thing around some index. At least Cole has some comments, here. It's the new area for this sorting. I suggest maybe we get Aman to look into whether he can come up with a scenario for this [, based on this code,] and not risk that we break something by making the suggested fix by the Cole. And then I guess we would like to fix it.'

A very obvious example of addressing future technical development was the connection of a bug to a 'bug farm', as also discussed in section VI-A. Similar discussions were observed in the discussions of other complex bugs. The team communicated, and by discussing a specific bug, contributed new knowledge of problematic areas in the code. In some cases, the group decided to take the bug fix into a grooming meeting, to discuss it in relation to ongoing development.

2) Domain: Under this category, sharing information about the ERP system's functionality from a user or domain perspective was addressed. This could be information regarding features, scenarios often associated with complex work flow as they are currently supported, scenarios as they should be, and how errors are seen from the end users' perspective. Also, the S2R information falls into this category. With respect to technical information, the discussion connects information about past development, a current problematic situation and how to address it, and future planned development.

Usually, when scenarios related to bugs were discussed, the question was about why the design did not take into account the particular flow the customer was implementing: *'... to release the one order with the correct address, and then I enabled the new consolidated shipment feature, and then things stopped working.'* - 436803

In the example below, the team states that the lack of information about how the software actually should function is related to past decisions:

'Then I put one on, PM X. I tried to follow up with him. [... [Describes the feature] We can't find whether you are allowed to mix dimensions, so basically variant driven [highlights the title on the screen]. [...] My problem is that I am not even able to find documentation for it, except these links I found, but none of the places it says anything about that it should only work for limited processes.' - 434161.

During Triage, some bugs were designated as being 'by design'. In these cases, a change that would meet the customer's expectations would be in conflict with the current design rationale of the whole module. At the beginning of Triage, the team often focused on ensuring that scenarios and flows were well-understood by all, and that the S2R accurately reflected the problems encountered by the user. The way a fix

affects end-user and customer experience was often discussed at length. Often, these discussions also addressed the severity of the bug. *'[...]so the thing is: it ... I think, it needs a little bit grooming: what exactly do we want to do? Because, like, every time when you click it, do you want to show this? or do you want to show this dialogue only if there are already existing transactions? Which would add like a slight performance impact, but again, then this is a setup form, maybe it's not problematic.'* - 436696

An example of the future dimension of domain knowledge is when the team discovered that fixing a bug should be addressed through a *new deliverable*.

3) Organisationl: In several Triage meetings, the team discussed how a bug related to organisational concerns. This could include identifying the expertise within the team/organisation to support fixing bugs. Or it could refer to changes to the development process. For example, in one Triage meeting, the team discussed how they could improve their testing strategy, and that they had taken up this issue in a past retrospective: *'Sounds like a typical PM forgot to test the scenario ... And this is exactly what we talked about – just sidetrack – talked about in the last sprint retrospective [...]: All PMs and maybe one engineer to do a cookbook every time you do a feature on inbound, what you need to test. And the same for outbound and all other stuff.'*- 435902

On another occasion, the team discussed their documentation processes, when, during the discussion of a bug, the team recalled a similar problem, but could not find the documentation of the older bug in the repository.

4) Business: This category refers to discussions of support contracts and specific customer relations that had an impact on the decision to fix or not fix a bug. Discussions of business information often coincide with the discussion of the possibility of backporting. If a previous release is no longer covered by a support agreement, the company is not required to continue to fix bugs. In such cases, customers would be encouraged to upgrade. Sometimes, the fix was backported even though customers had plans to upgrade to a working version, if the customer was not able to wait for the upgrade. Often, such situations also prompted discussions regarding support contracts and customer relationships.

'Customer requested this [...] now that the end of support has been extended. So I have not seen that in mail from Susan this morning. So for this [release] 10.08, I still see 'End of servicing: April the 13th'. So maybe what they think is moved. And yeah, so we still have few days to actually backport it, and they can still use that build.' - 427767.

5) Summary: Information is the building block in the information flow [5], hence it was important to analyse information types. This analysis shows that there is a wide range of different kinds of information shared and discussed in the Triage meetings. Information is related not only to the whether, how and who of a bug fix, but also to the domain knowledge necessary for fixing bugs, how the possible fix would interact with other bugs, scenarios and tasks, and to improving the process of bug fixing.

In a given Triage meeting, these categories of information were often tightly interlaced. For example, in one case (Bug no. 431122), it was impossible to reproduce the problem in the current version of the software. Various reasons were considered: the customer was still deploying an earlier version; that it was fixed in an earlier version, to which the customer had not yet upgraded yet; that the customer may have customised the software in ways that interfered with this scenario, or the error could be caused by hardware dependency. These problems could be resolved if the customer upgraded to the current version. Contractual conditions and customer relationship management also influenced the how the technical fix would be done. In other words, domain related, technical, business and organisational information were related in one and in the same discussion. This discussion also resulted in pointers for future development: problems to consider when designing new features (deliverables) and parts of the code that needed special attention. In other words, the Triage meetings generated information and developed knowledge that informed future design and developments.

VII. Discussion

The previous section presented an in-depth analysis of Triage on 33 complex bugs. In this section, we first present the role of DiCoT in the analysis and the resulting understanding of Triage, then we discuss the nature of the information generated and discussed in Triage, and finally we consider the implications of our understanding of ML research on bugs.

A. The Role of DiCoT in the Analysis

DiCoT's information flow with its associated concepts of information hub, information movement and information transformations [5] provided the conceptual framework for structuring and analysing the complex nature of observations/data of the various stages of a bug's journey see Figure 1. This led to identification and a deeper understanding of various information hubs [5], especially the Triage meeting. DiCoT's principles of information movement and information transformation kept the focus on identifying which types of information were brought into the hubs, by whom and how team members transformed them, to understand past, present and future software development. Without DiCoT, the sheer volume of information that is available in a development environment would have been difficult to navigate without losing sight of the original overarching aim: to understand how bugs are resolved. At the same time, the analysis deepened the understanding of what happens in an information hub: we understand the information hub as a socio-technical collaborative environment that requires information movement and information transformation, mainly from people. This occurs through an iterative process of explanation and clarification at team level. This is where most important decisions are made, and during this process, information is generated for a collaborative task, in this case, bug fixing.

As shown above, Triage is a goal-driven, team-coordination activity that aims to determine whether, when and how to

address a bug. Using the DiCoT framework, it functions as an information hub [5] for knowledge-building and sharing, where experienced members of the team meet to take the most important decisions about a bug: to fix or to not fix. It has a high bandwidth, given the diverse information types discussed, and in terms of its impact on changes in the source code, changes in business practices and in changes in organisational processes. Participants share a wide range of information that relates domain knowledge, technical expertise, business knowledge and organisational information. The information movement and transformation [5] generates information that has implications for future design, and in some cases even underpins changes in the development organisation. The Artefacts principle of DiCoT also has a role here [5]. Information hubs, like Triage, are mediated by artefacts, like the S2R as part of the bug report, which in turn resides in the bug database. The bug database is an artefact that coordinates resources, as after the discussion of each bug, information about it is updated, for example engineers are tagged to fix bugs. This finding resonates with previous research that finds that Triage is an important coordinating activity, which cannot be accomplished solely by being based on electronic traces in the database [3].

B. Triage is a trigger for generating information and knowledge

Though the main goal of Triage is to determine whether or not, and how to fix bugs, the discussions in Triage are not limited to this goal, and the information shared and developed in the discussions is important input for future development of the software. The nature of the information discussed and generated in Triage expands the boundary of time, and brings together several dimensions. Though the focus is on information related to the bug, the Triage discussions help to identify information the team lacks, which in turn prompts a series of information-seeking and information-clarification events. This process generates further information that is essential to understanding the past and present, and to prepare for the future evolution of the software: Triage leads to a better understanding of why the software was designed the way it was, how it should have been designed (bugs), and it also feeds into future deliverables.

Technical: Triage leads to better understanding of the code base, not only with regard to a given bug, but also to the related area, on which resolving the bug will have an impact. For example, discussing data corruption (data integrity) and 'bug farms' leads to a better understanding of fixing, beyond the current bug. Business: Triage has an impact on understanding the business setting, in terms of contractual obligations, managing customer relationships and their impact on upgrades to the latest version. Organisational: Information generated in Triage is fed back into software engineering and other organisational processes (see the 'cookbook' example) and it resulted in issues that were fed into retrospectives. Domain: Discussing information concerning domain knowledge that is about functionality and scenarios from a customer perspective.

The existing research focuses on the importance of artefacts, such as bug reports, supporting collaboration and coordination [19]. Ko et al. [18] explored what information engineers need, and why they need this during Triage. Their analysis focuses on legitimacy, difficulties and the cost of fixing a given bug. To the best of our knowledge, none of the literature has addressed the role of the Triage in generating information to support the future of design and development, of the code base and the software.

C. Role of Machine Learning(ML)

ML has proven to be very effective in categorising or classifying problems. Therefore, it is unsurprising that previous research used machine learning approaches to determine whether bugs may be assigned automatically or provide feedback on the quality of bug reports. Important information from the bug reports might be generated using ML approaches, but in this case, 'learning' resides in the machine.

Most of the ML research into automation related to bug fixing has focused on electronic traces, such as the bug report and other repository sources [1], [10]–[12], [17]. Our analysis shows how the participants in Triage meetings combine and relate a wide range of information. The specific information needed was often not anticipated. Triage yielded not only a decision concerning fixing bugs, but also strategies for fixing complex bugs, and, last but not least, information regarding the future evolution of the software.

ML could be used to complement human Triage, which is an expensive process and may waste resources, especially when it is a question of simple bugs. Perhaps ML could be used to categorise simple and complex bugs. Currently, all bugs are discussed in Triage, 62% of which are simple bugs. Simple bugs could then be handled by the developers directly. For complex bugs, Triage could be kept as an important space for decision-making, and continue to be an information hub, where learning and knowledge-development occur and informs future development and organisational processes.

VIII. Conclusion

In our analysis of Triage, we show how information concerning the source code, current and future deliverables, and organisational processes are discussed, contributing to the team's expertise, and generating information that informs the future development of the software. This is done through a series of clarifications and explanations. Thus, this study concludes that the traditional understanding of the objective of Triage should be reconsidered. The final outcome of Triage is not just the assignment of bugs; instead, the process that leads to this outcome generates important information for the future design and development of the software in question. If Triage is seen as bug assignment only, this hidden value of the Triage meetings is overlooked. There are three points that future research could take up. First, ML may be used to categorise simple and complex bugs. This would depend on data to train the ML algorithm. Also, it would be interesting to investigate whether the ML algorithm previously researched performs

better when applied to simple bugs assigned to engineers. Second, there is growing evidence of the importance of the role of awareness-building through the social structure and organisation of software engineering [4], [14], [18], [31]–[33], yet progress on this research has been sporadic. Building on our findings, future research could investigate whether similar results emerge in the Triage of other software development teams or for other types of software. Finally, though DiCot has been instrumental in our analysis it has its limitations. It addresses awareness of information in collaborative environment only in relation to physical layout of the team, and the concept of the information hub is not well-developed. Here the research method applied in the research presented here could help to further understand in detail of information exchange in different contexts. In distributed software development, the physical layout may not be as important as social and organisational structures. So far, DiCot has not defined principles surrounding social structures [5]. Theoretical work on refining DiCot principles may yield a more suitable framework for analysing this type of work.

REFERENCES

[1] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, "Automated, highly-accurate, bug assignment using machine learning and tossing graphs," *The Journal of Systems and Software*, vol. 85, pp. 2275–2292, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2012.04.053

[2] S. R. Lee, M. J. Heo, C. G. Lee, M. Kim, and G. Jeong, "Applying deep learning based automatic bug triager to industrial projects," *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, vol. Part F130154, pp. 926–931, 2017.

[3] J. Aranda and G. Venolia, *Proceedings: The Secret Life of Bugs: Going Past the Errors and Omissions in Software Repositories*, 2009.

[4] J. Anvik and G. C. Murphy, "Reducing the Effort of Bug Report Triage: Recommenders for Development-Oriented Decisions," *ACM Trans. Softw. Eng. Methodol*, vol. 20, no. 10, 2011. [Online]. Available: http://doi.acm.org/10.1145/2000791.2000794

[5] A. Blandford and D. Furniss, "DiCoT: a methodology for applying Distributed Cognition to the design of team working systems," Tech. Rep., 2005.

[6] J. Hollan, E. Hutchins, and D. Kirsh, "Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research," *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 2, pp. 174–196, jun 2000. [Online]. Available: http://dl.acm.org/doi/10.1145/353485.353487

[7] M.-A. Storey, N. A. Ernst, C. Williams, and E. Kalliamvakou, "The who, what, how of software engineering research: a socio-technical framework," *Empirical Software Engineering*, vol. 25, pp. 4097–4129, 2020. [Online]. Available: https://doi.org/10.1007/s10664-020-09858-z

[8] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, T. Zimmermann, and T. Org, "What Makes a Good Bug Report?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, 2010.

[9] O. Chaparro, C. Bernal-Cárdenas, K. Moran, A. Marcus, M. D. Penta, D. Poshyvanyk, V. Ng, J. Lu, D. Penta, and V. . Ng, "Assessing the Quality of the Steps to Reproduce in Bug Reports," 2019. [Online]. Available: https://doi.org/10.1145/3338906.3338947

[10] J. Anvik, L. Hiew, and G. C. Murphy, "Who Should Fix This Bug?" Tech. Rep., 2006. [Online]. Available: www.bugzilla.org/,

[11] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *ESEC-FSE'09 - Proceedings of the Joint 12th European Software Engineering Conference and 17th ACM SIGSOFT Symposium on the Foundations of Software Engineering*. New York, New York, USA: ACM Press, 2009, pp. 111–120. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1595696.1595715

[12] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows," in *ICSE 2010*, 2010.

[13] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information Needs in Bug Reports: Improving Cooperation Between Developers and Users," in *CSCW '10: Proceedings of the 2010 ACM conference on Computer supported cooperative work*, 2010. [Online]. Available: http://www.mozilla.org/

[14] D. Cubranic and G. C. Murphy, "Automatic bug triage using text categorization," in *16th Int. Conference on Software Engineering and Knowledge Engineering*, 2004, pp. 92–97.

[15] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," *IEEE International Working Conference on Mining Software Repositories*, pp. 22–30, 2013.

[16] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *arXiv*, vol. 27, no. 1, pp. 264–280, 2017.

[17] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth, "Hipikat: A project memory for software development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 446–465, 2005.

[18] A. J. Ko, R. Deline, and G. Venolia, "Information N eeds in Collocated Software Dev elopment Teams," Tech. Rep., 2007.

[19] P. H. Carstensen, C. Sørensen, T. Tuikka, P. H. . Carstensen, and C. . Sørensen, "Let's Talk About Bugs!" Tech. Rep. 1, 1995. [Online]. Available: http://aisel.aisnet.org/sjis/vol7/iss1/6

[20] K. Schmidt and C. Simonee, "Coordination mechanisms: Towards a conceptual foundation of cscw systems design," *Computer Supported Cooperative Work (CSCW)*, vol. 5, no. 2-3, pp. 155–200, 1996.

[21] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Not my bug! and other reasons for software bug report reassignments," *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, pp. 395–404, 2011.

[22] E. Hutchins, "Cognition, Distributed," *International Encyclopedia of the Social & Behavioral Sciences*, no. Kitcher 1990, pp. 2068–2072, 2001.

[23] H. Sharp, R. Giuffrida, and G. Melnik, "Information flow within a dispersed agile team: A distributed cognition perspective," in *Lecture Notes in Business Information Processing*, vol. 111 LNBIP. Springer Verlag, 2012, pp. 62–76.

[24] H. Sharp, H. Robinson, J. Segal, and D. Furniss, "The role of story cards and the wall in XP teams: A distributed cognition perspective," *Proceedings - AGILE Conference, 2006*, vol. 2006, pp. 65–75, 2006.

[25] D. Furniss and A. Blandford, "Understanding emergency medical dispatch in terms of distributed cognition: A case study," *Ergonomics*, vol. 49, no. 12-13, pp. 1174–1203, 2006.

[26] N. Flor and E. Hutchins, "Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance," in *EMPIRICAL STUDIES OF PROGRAMMERS: FOURTH WORKSHOP*. Ablex, Norwood, 1991, vol. 1, no. -, pp. 36–64.

[27] H. Sharp, Y. Dittrich, and C. D. Souza, "The Role of Ethnographic Studies in Empirical Software Engineering The Role of Ethnographic Studies in Empirical Software Engineering," no. c, pp. 1–25, 2016.

[28] J. W. Creswell, *A Concise Introduction to Mixed Methods Research*. Sage Publications, 2014.

[29] C. Robson and K. McCartan, *Real World Research*, 4th ed. Wiley. [Online]. Available: https://www.wiley.com/en-us/Real+World+Research%2C+4th+Edition-p-9781118745236

[30] R. Chen. (2011, Sep.) Microspeak: The bug farm. [Online]. Available: https://devblogs.microsoft.com/oldnewthing/20110920-00/?p=9603

[31] C. Treude and M.-A. Storey, "Awareness 2.0: Staying Aware of Projects, Developers and Tasks Using Dashboards and Feeds," in *ICSE 2010*, 2010.

[32] I. Omoronyia, J. Ferguson, M. Roper, and M. Wood, "Using developer activity data to enhance awareness during collaborative software development," *Computer Supported Cooperative Work*, vol. 18, no. 5-6, pp. 509–558, 2009.

[33] C. Gutwin, R. Penner, and K. Schneider, "Group awareness in distributed software development," in *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*. New York, New York, USA: ACM Press, 2004, pp. 72–81. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1031607.1031621