A Grounded Theory of the Role of Coordination in Software Security Patch Management

Nesara Dissanayake
University of Adelaide
Adelaide, Australia
nesara.madugodasdissanayakege@adelaide.edu.au

Asangi Jayatilaka
University of Adelaide
Adelaide, Australia
asangi.jayatilaka@adelaide.edu.au

Mansooreh Zahedi
University of Adelaide
Adelaide, Australia
mansooreh.zahedi@adelaide.edu.au

Muhammad Ali Babar
University of Adelaide
Adelaide, Australia
ali.babar@adelaide.edu.au

ABSTRACT

Several disastrous security attacks can be attributed to delays in patching software vulnerabilities. While researchers and practitioners have paid significant attention to automate vulnerabilities identification and patch development activities of software security patch management, there has been relatively little effort dedicated to gain an in-depth understanding of the socio-technical aspects, e.g., coordination of interdependent activities of the patching process and patching decisions, that may cause delays in applying security patches. We report on a Grounded Theory study of the role of coordination in security patch management. The reported theory consists of four inter-related dimensions, i.e., causes, breakdowns, constraints, and mechanisms. The theory explains the causes that define the need for coordination among interdependent software/hardware components and multiple stakeholders' decisions, the constraints that can negatively impact coordination, the breakdowns in coordination, and the potential corrective measures. This study provides potentially useful insights for researchers and practitioners who can carefully consider the needs of and devise suitable solutions for supporting the coordination of interdependencies involved in security patch management.

CCS CONCEPTS

• **Software and its engineering** → **System administration**; • **Security and privacy** → *Software security engineering*; *Vulnerability management*; Social aspects of security and privacy.

KEYWORDS

software security patch management, grounded theory, socio-technical factors, coordination

1 INTRODUCTION

Timely development and application of security patches to the identified vulnerabilities are considered critically important to avoid potentially successful security attacks [45]. Delays in patching security vulnerabilities can cause significant data losses, for example, the Equifax case [21, 35], or even human death [15, 43]. Increased awareness about the potentially catastrophic consequences of delaying patching is leading to increased efforts aimed at improving technical and socio-technical aspects of software security patch management, hereafter called security patch management, a process that consists of identifying, acquiring, installing, and verifying security patches [45]. These activities entail several socio-technical aspects that underpin some of the critical decision points that make security patch management a complex and challenging undertaking [27, 32, 49]. Further, it needs the coordination of the efforts and decisions of multiple stakeholders with conflicting interests and several interdependencies. It has been reported that a majority of the delays in the security patch management process emerge from socio-technical aspects such as coordination [13, 32, 38, 49].

While it is widely understood that appropriate coordination support is needed for timely decisions and actions by the involved stakeholders, there is not much empirically known about the coordination aspects that may cause delays in applying patches. That means researchers and practitioners may not find much guidance in gaining a good understanding of the role of coordination in security patch management to answer some critically important questions: What is the role of coordination in security patch management? How does the coordination aspect cause delays in security patch management? What can be done for addressing the causes that may delay security patch management? These questions motivated us to carry out a grounded theory study of the role of coordination in security patch management.

Nesara Dissanayake, Mansooreh Zahedi, Asangi Jayatilaka, and Muhammad Ali Babar

In this paper, we present the first, to the best of our knowledge, Grounded Theory study exploring the role of coordination in security patch management. It is based on observations of 51 patch meetings over a period of 9 months, which involved 21 industry practitioners from two organisations in the mission-critical healthcare domain. We explain how coordination impacts the software security patch management process in four inter-related dimensions: causes, constraints, breakdowns, and mechanisms. Grounded in the evidence from industrial practices of the patch application decisions, the theory aspires to enhance the state-of-the-art understanding of researchers and practitioners in several ways: (a) the theory highlights the importance of gaining a deep understanding of the interdependencies before applying security patches and how improved support in coordination can help reduce the delays in security patch management; (b) it structures the knowledge about the unexplored phenomenon of security patching in the mission-critical healthcare domain; (c) provides a theoretical model to shape future Software Engineering (SE) research to address the practical concerns in security patching; (d) practitioners can leverage the understanding to reduce patching delays, and (e) the theory can also be useful to practitioners as guidance to enhance the confidence in patching decisions.

2 BACKGROUND AND MOTIVATION

Software Security Patch Management is "*a multifaceted process of identifying, acquiring, installing, and verifying software security patches for products and systems*" [31, 45, 49]. Although there has been extensive research [8, 27] on improving automation support in the security patch management process, we noticed a scarcity of empirical studies investigating the socio-technical aspects of security patch management. Existing empirical studies on socio-technical aspects of security patch management have primarily focused on studying system administrators [8, 12, 32, 49], the patch management process and related challenges [32, 49], and patch information retrieval behaviors and approaches of system administrators [28, 49].

We found several studies (e.g., [13, 27, 32, 37, 39]) reporting coordination and collaboration challenges in the patch management process. However, they lack a comprehensive investigation of what causes the coordination needs and related challenges, its effects on the security patch management process, and the impact on delays of patch installation. Some studies [22, 38, 40, 49] have described in high-level the dependencies between multiple stakeholders, such as vendors and organisations. Similarly, Nappa et al. [37] reported that the coordination challenges concerning vendor dependencies arise from a lack of synchronized patch releases from different vendors because of shared vulnerabilities in the software code. Their analysis was based on a large data set of deployed patches in client-side vulnerabilities. Similarly, quantitative models and frameworks presented by a few other studies [5, 6, 11] focused on optimizing patch management by synchronizing the organisation's patch cycle with the vendor's patch release cycle to reduce patching costs and risks. As such, the reported dependencies with software vendors raise important concerns about the need for an in-depth understanding of the role and impact of such dependencies on security patch management.

However, coordination has been studied extensively across various dimensions in the related domains such as software development over the last decades [2–4, 10, 29, 48]. The literature defines coordination as *the management of interdependencies* [9, 33, 34] and describes different types of coordination as explicit coordination and implicit coordination. Similarly, Crowston [9] provided a categorization of the types of dependencies based on the context such as task, knowledge, resource, and technical dependencies. Furthermore, the previous work [2] about the coordination challenges in software development processes has demonstrated that ineffective coordination of dependencies represents a major cause of project failure, justifying the need for effective coordination to manage various interdependencies. Correspondingly, a comprehensive understanding of the role of coordination in patch application presents a critical research gap, which is fulfilled by this study.

3 RESEARCH METHOD

We used Grounded Theory (GT) [18, 20] for data collection, analysis, theory development, and reporting. GT enables the systematic generation of theory from data, relating to social interactions and behaviour in real-world settings [17, 20]. The choice of GT as our research method was based on two reasons:

(a) The aim of our research, to understand the socio-technical aspects of security patch management in practice suited well with GT as it allows the investigation of people and interactions in a real-world phenomenon [17].

(b) GT is considered most relevant to research areas that have not been deeply explored before [25], and research on the socio-technical aspects of security patch management is limited in the literature.

We followed the Glaserian version of GT [18] since it offers more flexibility to uncover the underlying concerns from the emergent data rather than limiting the research angle with a defined research hypothesis upfront. Following the guidelines, we started with an *"area of interest"* - Socio-technical concerns in Security Patch Management. The guidelines by Stol et al. [47] were followed for reporting the GT findings.

3.1 Data Collection

We observed 51 patch meetings between two organisations (Alpha and Beta) in Australia, attended by 21 key stakeholders from 8 teams. These stakeholders represented diverse roles centered on decision-making, planning, and executing security patching. The longitudinal study was conducted from March 2020 - January 2021. The meetings were held every fortnight lasting approximately an hour and a half. Due to COVID-19, the meetings were held online through Microsoft Teams. The patch meetings focused on reporting security patch management status, tracking vulnerability remediation progress, discussing issues, planning patch cycles, and decision-making. Alpha is an Australian state government health services agency and Beta is an American multinational corporation that provides **security patch management** service to Alpha. While the most important and critical OS security patching was being outsourced to Beta, Alpha's in-house developed applications and other third-party applications were patched by Alpha teams and the respective third-party vendors. Table 1 presents the investigated

teams' demographics. Abiding by the human ethics guidelines, the details of the organisations and teams have been kept confidential.

We held discussions immediately after the meetings with one of Alpha's security team members to clarify any doubts that emerged during the observations and gather additional information. We also gathered data by analysing artefacts such as meeting minutes and patch mailing thread to supplement our understanding of the process, practices, and used terminology.

The first author attended all 51 meetings that were held over the course of 9 months and conducted all 11 post-meeting discussions. All the meetings and discussions in Table 2 were audio-recorded with permission and shared with all researchers. We adopted the protocol proposed by Spradley [46] to guide the data collection during the meetings (see Appendix A). The first author briefed other authors about the key aspects of the fortnightly meetings and the post-meeting discussions regularly. The data collection and analysis were performed in iterative and intertwined stages throughout. We continued with the data collection until the data analysis confirmed *theoretical saturation*. The last few observations (M46-M51) provided more examples and evidence for the emerged findings during the analysis, but no new concepts, categories, or insights emerged. All authors mutually agreed that this was a clear indication of the *theoretical saturation* and any additional data collection would not add value to the findings.

3.2 Data Analysis

We followed Glaser's data analysis procedure starting from **Open coding** through **Selective coding** to **Theoretical coding** [18, 47, 50]. The data analysis was led by the first author supported by other researchers who took the role of the validators at each stage throughout the iterative and intertwined rounds of data collection and analysis. All data including transcripts, observation and discussion notes, other artefacts (meeting minutes and patch mailing thread notes), codes, and memos were saved in the NVivo data analysis tool and shared with all co-authors. The second and third authors cross-validated all the emergent codes, concepts, categories, and core categories. Any conflicts in the coding and coding procedures were resolved in weekly detailed discussions between all authors throughout the analysis phase involving several rounds of revisions. Additionally, the emerged findings were further cross-checked with one of the senior members of Alpha's security team.

Open coding started with thoroughly reviewing the transcripts and recording *key points* containing summarized phrases [16]. It was further summarized into *codes* of three-five words each, and any specific *properties* of the code were captured in brackets, as shown in the example below.

> **Transcript:** *"The patching is delayed because this .NET security vulnerability was reported in August after patching happened for the month. But, we also have another problem as this is a different version of .NET from what is standing across the fleet. This is .NET core, not .NET version 4.801."*
> **Key Point:** *Need to identify and match Framework version dependencies*
> **Code:** *Software application inter-dependencies (version)*

Applying constant comparison on the codes that emerged within each observation, between different observations, and post-meeting discussions, we grouped them to a higher level of abstraction, i.e., *concepts* [18, 20]. Similarly, continuously comparing concepts produced *categories*, a third-level of abstraction, and from categories generated *core categories* [18] at the end of the first round of coding. The core category represents the main problem or concern *(core)* in the studied phenomenon, which presents the research question [17, 26]. Correspondingly, three potential *core categories* emerged - Legacy software systems, Role of Coordination, and Role of Patch Meetings. The % split of codes between the three core categories was 18%, 51.3%, and 30.7% respectively. We selected *"Role of Coordination in Software Security Patch Management"* as the core category because it met all the criteria defined by Glaser [17] for selecting a category as the core. For example, the selected core category was central to other categories and frequently occurred in the data; meaningfully related to both other categories easily and took the longest to saturate. Our decision of focusing on the "role of coordination" from the initial general focus on the "role of the socio-technical aspects in security patch management" was informed by Glaser's Grounded Theory guidelines [17].

After establishing the core category, we continued **Selective coding** [17] limited to only those codes that were related to the selected core category. For example, Figure 1 illustrates the emergence of the category *Technical dependencies* that relate to the *role of coordination* in the second round of coding - Selective coding. We continued to selectively code until no new insights or aspects emerged for each category, which indicated *theoretical saturation* [18, 20].

As the final step of the analysis, we applied **Theoretical coding** [17–19, 47] to establish conceptual relationships between categories, resulting in the development of a theory. We used *memos* (*memo sorting*) [17, 18] to guide us to uncover the links between the categories when developing the theory. At this point, we consulted the literature, particularly Glaser's *'theoretical coding families'* to find if any existing theoretical structure fits our current findings to visualize the theory. Glaser argues using a coding family to present the theory would help increase the completeness and relevance of the emerging theory [17, 19]. As the findings (i.e., the causes, constraints, breakdowns, and mechanisms) emerged from the data, we found that the theoretical coding model, i.e., *Dimension family* is the best fit to visualize the relationships between the categories. The *Dimension family* is a theoretical structure that enables the findings to be presented as dimensions or elements of a phenomenon [17, 19], in this case, dimensions of the role of coordination in software security patch management. Thus, the theory of the role of coordination in security patch management, depicted in Figure 2, is described using: (a) **Causes:** socio-technical dependencies that define the need for coordination; (b) **Constraints:** factors that hinder coordination; (c) **Breakdowns:** scenarios of patching failures resulting from ineffective coordination of the causes and constraints; and (d) **Mechanisms:** strategies devised for supporting the coordination in security patch management.
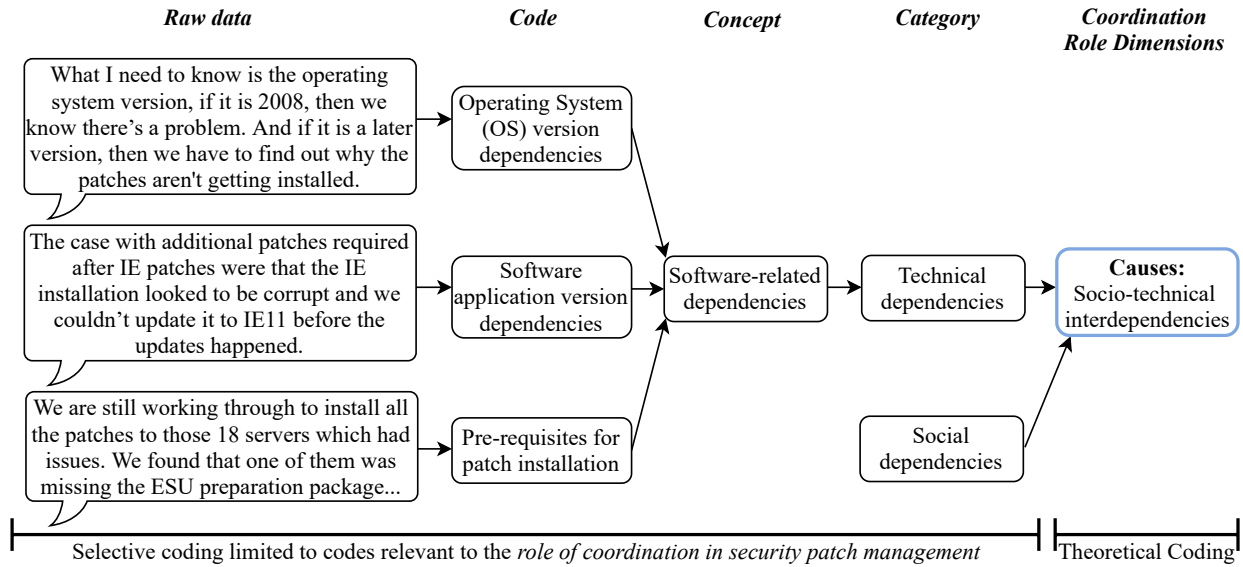
4 FINDINGS

In this section, we describe the theory of the role of coordination in security patch management using four inter-related dimensions:

Nesara Dissanayake, Mansooreh Zahedi, Asangi Jayatilaka, and Muhammad Ali Babar

Table 1: The investigated teams' demographics

Organisation	Team	Domain	Team size [*]	Distribution [+]	Roles
Alpha	T1	Electronic Medical Records (EMR)	5	Co-located	EMR Application Owner, Server Engineer, System Administrator, Server Manager
	T2	Digital Health - Windows	3	Co-located	Server Engineer, System Administrator, Windows Application Specialist
	T3	Digital Health - Non-Windows	2	Co-located	Unix Specialist, Server Engineer, System Administrator
	T4	Security	1	Co-located	Security Advisor
	T5	Change Management	1	Co-located	Change Manager
	T6	Clinical and Pathology Services	1	Co-located	Pathology Server Engineer
Beta	T1	Server (Technical)	7	Distributed	Server Engineer, Senior Server Engineer, Unix Engineer, Server Manager, Client Delivery Manager
	T2	Finance and Audit (Non-technical)	1	Distributed	Accounts Manager

[*] The team size refers to the number of team participants in the patch meeting. [+] *Distributed* refers to two locations within the state in Australia.

Figure 1: The emergence of category *Socio-technical interdependencies* from the underlying codes and concepts.

Table 2: Summary of the data collection

No. of meetings	Duration	No. of discussions	No. of hours	
			Meetings [*]	Discussions [+]
51	9 months	11	30 hours	7 hours

[*] The average time of a patch meeting=30 minutes.

[+] The average time of a post-patch meeting discussion=30-45 minutes.

Causes, Breakdowns, Mechanisms, and Constraints, providing evidence with grounded quotes from the underlying data. For ease of reference, we used unique identifiers to refer to participants, for example, *AT4, M10-dis* refers to a participant from Alpha's Security team in the 10th post-meeting discussion, and *BT1, M4* refers to a participant from Beta's Server team in the 4th meeting.

4.1 Causes - Socio-Technical Dependencies

We identified several socio-technical interdependencies inherent in the security patch management process that define the need for coordination. We recorded discussions about several scenarios exemplifying the potentially disastrous implications as a result of failing to identify dependencies before applying a patch. Through the data analysis, we have placed the observed interdependencies in two categories: *Technical* and *Social Dependencies*.

4.1.1 Technical Dependencies. The interdependencies between the software and the associated hardware and firmware give rise to technical dependencies, arising as a result of dependencies in software code. For example, a security vulnerability present in multiple functions in the source code may include dependencies among the functions. As a result, a security patch developed to fix the identified vulnerability also inherits these function-level dependencies
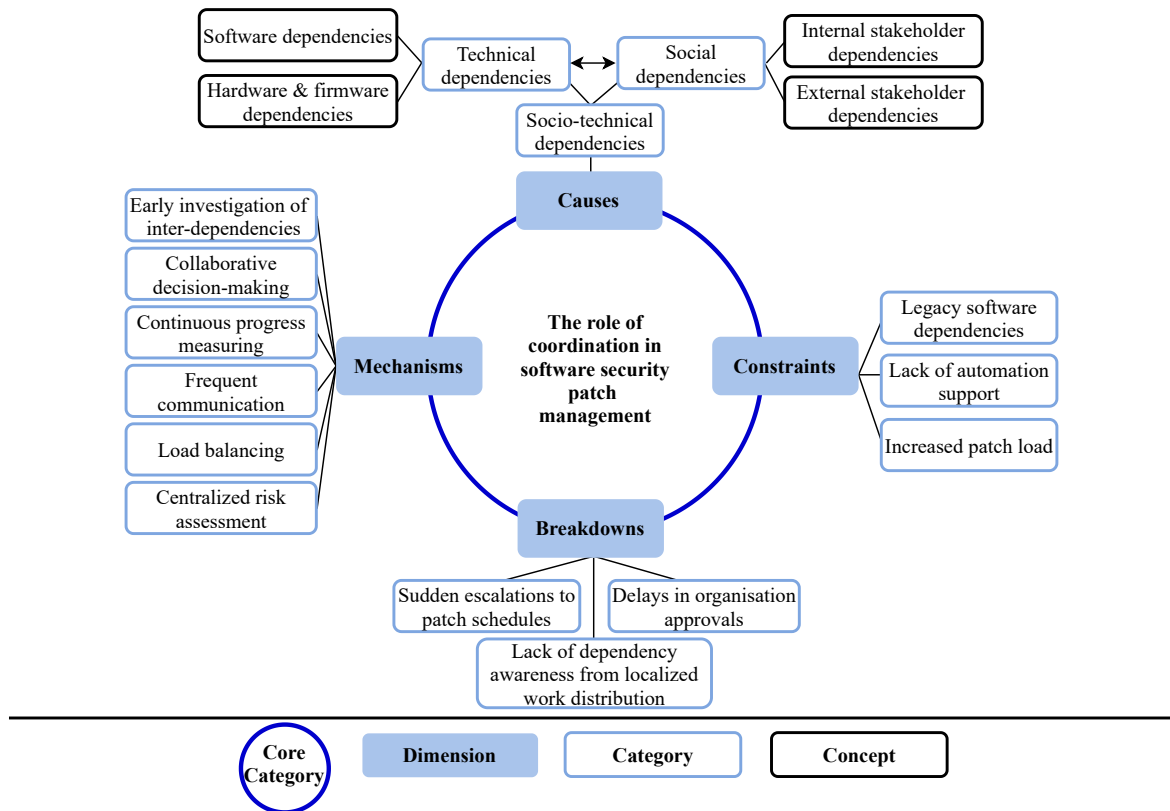
Figure 2: The theory of the role of coordination in software security patch management.

creating complexities that must be carefully handled during the patch management process.

In terms of **software-related patch dependencies,** several factors such as operating system (OS) dependencies, software application dependencies, and pre-requisites for patch installation fostered the conditions for coordinating the dependencies. Patch management in large software systems involves managing multiple software components (or services) with different OS versions. The existence of the interdependencies between OS versions and other software applications built on top of a particular OS like web browsers may create additional tasks for practitioners as all of them need to be synchronized.

"We have about 15-16 versions of Windows 10. So, before patching we need to see which version is running on which server? What is the build number? Are we running the latest? It's a lot!" - AT4, M14-dis

Additionally, some security patches contained interdependencies with legacy OS. It presented a much more arduous task to the studied practitioners since some critical emergency medical services were running on Alpha's legacy systems that were not supported by large vendors, e.g., Microsoft. In such cases, the participants often felt forced to delay patch installation as the available solutions like decommissioning or upgrading legacy systems presented high risks of operation interruptions. Consequently, this

practice of delayed patching of security vulnerabilities in systems would significantly increase the risk of exposure to attacks.

Similarly, interdependencies between software applications, platforms, and tools presented another major category of software-related dependencies. This is because the build dependencies between the software application and patch sources require the versions to be in sync before attempting patch installation [14]. As the size of an organisation grows, managing these dependencies appeared difficult with a large number of diverse applications installed. For example, Alpha's software applications ranged from general applications, e.g., Java, .NET to specific medical applications, e.g., Electronic Medical Record Software. As such, Beta teams spent most of their time detecting the existing interdependencies such as version incompatibilities between various software applications.

"There's an old HP tools version and a new version, and the vulnerabilities are coming up on the scan as with the new version. But the issue is because the old version is still there which we should have got rid of earlier." - BT1, M14

The circular dependency represented a more complex semantic dependency in security patch management. An example scenario was when software B is dependent on software A, and software A uses software B to function (B ↔ A). In such cases applying a security patch to A led to service unavailability of B as a result of rebooting A to make the security patch take effect. In particular, effectively coordinating circular dependencies was crucial in the

healthcare domain as A and B could represent critical medical services like emergency life support or surgery equipment.

"And there could be like A needs B to run, and vice versa but when we accidentally took B offline that day, A didn't work. That was when we all got goosebumps." - AT4, M10-dis

On the other hand, some security patches required pre-requisites to be established before installation for the patch to take effect. In most cases, the pre-requisites comprised registry changes and preparation package installation. We identify that it is resulting from the patches that do not contain source code modifications, as explained by Li and Paxson [31]. To investigate the pre-requisites of the security patches retrieved each month, Beta allocated a specific timeframe before patch testing and discussed with Alpha during the patch meetings how and when they would handle the identified pre-requisites. Coordinating the pre-requisites was often a manual task as it involved decisions about suitable configurations based on the organisation's needs and the other associated software dependencies. Failing to configure the pre-requisites led to errors that would halt a patch installation. However, we found that the teams became more receptive to detecting pre-requisites-related dependencies with the continuous early investigation approach employed.

"The patches listed here needed a preparation package installed before the patching window and then the reboot would have applied the patch. We'll do that just before the current patching window and then patching should proceed as normal without errors." - BT1, M11

Besides the most common software-related dependencies, some security patches also contained dependencies with the associated hardware and firmware giving rise to **hardware and firmware-related dependencies**. For example, in one instance, practitioners were unable to patch the security vulnerabilities found in virtual machine (VM) software as the VM-related firmware was not up to date. So, they had to regularly keep track of the existing dependencies and update the supporting hardware and firmware accordingly before attempting patch installation.

"Some patches need a certain type of hardware to be at a certain level. There was a 2008 security patch which we couldn't install until we updated the firmware or the utilities." - AT4, M10-dis

4.1.2 Social Dependencies. Social dependencies that stem from interdependencies between stakeholders is another major category of dependencies integral to security patch management. Security patching in large organisations is challenging due to the increased complexity stemming from a high number of stakeholders. Therefore, effectively coordinating the dependencies between multiple stakeholders is important for successful patch management. Our data analysis led to the emergence of two sub-categories of social dependencies: Internal stakeholder dependencies and External stakeholder dependencies as illustrated in Figure 3.

Internal Stakeholder Dependencies - The two organization's stakeholders worked together to achieve a common goal of securing the state's healthcare system by timely installation of security patches. Hence, the internal stakeholder dependencies, in this context, relate to the dependencies stemming from the interactions between stakeholders across Alpha and Beta. We identified

two layers of dependencies namely *team-level dependencies* and *organisation-level dependencies.*

The current context displays a *multiteam system* (MTS) structure [36] as multiple interdependent teams within each organization collaborated towards a collective goal. The interdependent team interactions gave rise to *team-level dependencies*. Since each organisational team had assigned responsibilities, coordination between and within teams remained pivotal to achieving the goals. In most cases, the inter-team tasks contained dependencies that required management of the cross-team interconnections. For example, Alpha teams T1, T2, and T3, often depended on T5 to approve security patch schedules before assigning them to Beta to be executed. Lack of awareness of the roles and responsibilities complicated the coordination of team dependencies causing delays of several weeks in patching known vulnerabilities.

"We are still waiting for an email from [P1] approving the manual patching process. (BT1)
Well, I'm not sure whether it should come from [P1] or some other guy. I will confirm it with [P2] and get to back you." - AT2, M8

Similar to the inter-team dependencies, *organisation-level dependencies* also created several challenges for security patch management. The large scale and heterogeneous nature of Alpha created additional complexities to coordinating organisation-level dependencies that often resulted in delays in patch testing and deployment. For example, every security patch required approval from Alpha's Change Management (T5) before they could be installed at the customer's sites (i.e., hospitals). Given the critical nature of the task, it was important to effectively coordinate it and leave sufficient time for the customers' agreements. It was needed to reduce the time of service interruptions and manage technical dependencies.

"This morning [Alpha's] change manager said that the change hasn't been approved yet. So, we had to suddenly change plans just minutes before the [scheduled] patch window." - BT1, M10

External Stakeholder Dependencies - The involvement of external stakeholders such as customers, end-users, and vendors is integral to security patch management. Effectively coordinating the collaborative relationships and the dependencies with external stakeholders is vital to a sound patch management process. In this context, the external stakeholders consisted of *customers* (e.g., hospitals) and *vendors* (e.g., Microsoft). The end-users dependencies consisted of hospital patients and staff, hence, these were not directly linked to Alpha and Beta. However, ineffective coordination of customer dependencies negatively impacted end-users. For example, uninformed operation interruptions to medical equipment resulted in inconveniences to patients and medical staff. In contrast to internal stakeholder dependencies, managing external stakeholder dependencies presented a much more difficult challenge to practitioners. The main reason was the lack of a shared understanding of the importance of security patching and the visibility of the existing process interdependencies.

Vendor-Related Dependencies refer to the dependencies that are created due to the need of installing security patches received from vendors. Management of vendor dependencies became difficult with the presence of shared vulnerabilities and associated technical dependencies in software applications. This demanded coordination
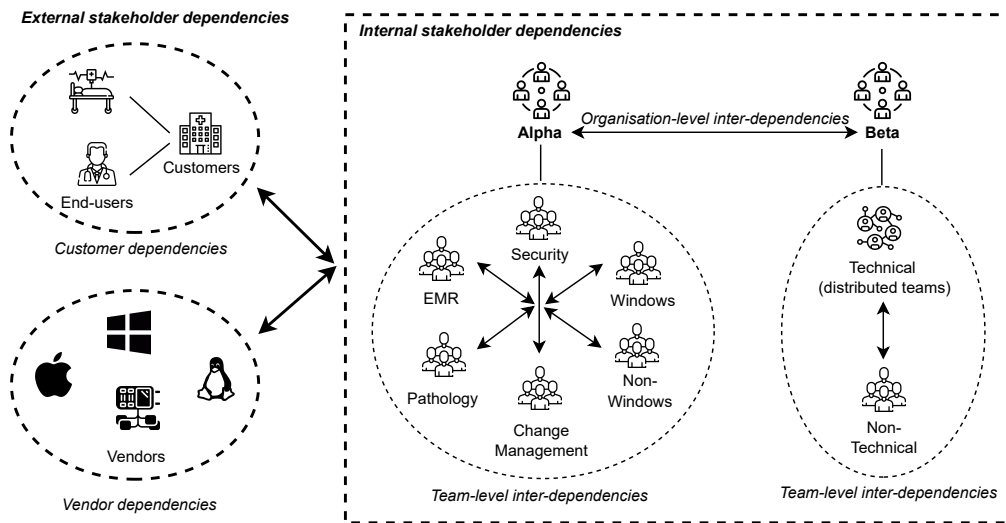
Figure 3: The social dependencies present in the studied context.

of patch releases from the vendors of different software applications. Additionally, some of Alpha's third-party applications were patched by external vendors, for example, the medical application providers, as per the agreement at the point of purchase. Thus, it required synchronising each vendor's patching cycles to avoid the unavailability of the interdependent systems.

"Regarding the recent concern from Security (T4) on [S1 server] patching is one month behind, can we confirm this with [third-party vendor's] requirements? Because this vendor's patching cycle is always one month behind, every month they release the patches for the last month." - BT1, M17

Furthermore, missing, faulty, or exempted security patches and unknown errors during patch installation increased the need for coordinating the vendor dependencies during patch testing, deployment, and post-deployment patch verification. Patch exemption was when selected security patches were excluded from installation due to legitimate reasons approved by Alpha's Security team. We observed several scenarios of patching delays due to a lack of coordination of software dependencies with exempted security patches that were managed by external vendors.

Managing *customer-related dependencies* with hospitals presented a challenge to Alpha and Beta teams, particularly when negotiating the patch schedules. Reaching a consensus on a patch installation time was essential to minimize any potential impact of service disruptions from reboots. However, an interesting observation was that in a majority of cases, practitioners spent most of the time trying to communicate to customers on the need to patch systems, as opposed to agreeing on the patch installation time. It was due to the lack of understanding of the need to apply security patches and the inability to accept the high risks of service downtime.

"A lot of customers don't always understand the worth of security patching, they just want to use the server, and keep asking; "why do you want to reboot it every then and there, or why you got to update it? It's working and leave it alone!" - AT4, M4-dis

4.2 Constraints

This section presents the constraints that hindered the coordination. When the constraints affected the socio-technical dependencies, they caused coordination breakdown. Hence, it is important to devise suitable approaches and tools for identifying and managing the potential impact of constraints.

4.2.1 Legacy Software-Related Dependencies. Legacy systems pose a security threat to organisational ICT infrastructures, particularly in mission-critical domains like healthcare. This is because most of the critical services that run on legacy systems remain unsupported by vendors leaving them vulnerable to security attacks. Alpha had several legacy systems like 2008 servers that were no longer receiving security patches from Microsoft. Furthermore, the dependencies with these legacy systems produced wider implications for security patching. It resulted in the practitioners being unable to patch until the dependent legacy systems were upgraded to the current version or offered extended support from vendors. However, upgrading legacy systems required a critical evaluation of the impact on other important services. In some cases, Alpha had acquired extended support for critical legacy systems through negotiations with the vendors. However, it presented practitioners with additional challenges of having to perform manual configurations to install security patches as the current configuration settings did not work, and some installed patches getting rolled back.

"There are twenty six 2008 servers still waiting to be patched this month. But there are some servers that we need to have a look at why the patches aren't applying. Even though we have installed all the required preparation packages, they keep rolling back." - BT1, M12

4.2.2 Lack of Automation Support. Lack of suitable tool support presented a major constraint for the coordination of the dependencies. One of the key constraints was the inability to oversee technical dependencies across all systems inventory. Inability to identify the existing software dependencies from the current tools resulted in practitioners spending hours trying to find the current

software versions in the event of errors during a patch installation. Additionally, the lack of automation support to investigate the patch pre-requisites as mentioned in Section 4.1.1, caused delays and sometimes errors in installing patches due to missing out on some registry changes.

On the other hand, the limitations on the features of the available support tools presented constraints in detecting specific dependencies such as legacy dependencies and their contextual categorization. This could be because most of the tools available to the studied teams focused on function-level patching assuming that vulnerable code resides within only one function [14, 31]. For example, Alpha's decisions were largely based on the vulnerability scan reports. However, the existing scanning tool was unable to filter the unpatched vulnerabilities resulting from legacy dependencies and exempted patches, which constrained accurate decision-making.

"These vulnerability numbers will go down by as much as half since [scanning tool] captures 2008 servers' vulnerabilities as well. I don't know whether we can do exceptions through the tool, like flag things that are legacy, to make the numbers reflect what we see." - BT1, M12

4.2.3 Increased Patch Load. As the organisation size grew, the number and diversity of systems also expanded that resulted in increased complexity in the patching. Hence, the practitioners often faced difficulties in keeping up with the patch release rate. Accumulated patch load due to the previous patch exclusions added to the challenges as they had to patch previously excluded patches in the following month. Correspondingly, an increased patch load led to more socio-technical dependencies creating additional constraints on coordination. Overall, it led to an increased risk of exposure to attacks as installing security patches was often delayed.

"There's just too much to check! We're dealing with 1500 servers, we don't have time to look at each patch for every server like, "Yeah, this one is right, this isn't. Which servers have interdependencies that can't be patched at the same time? Which server has which version of this software, that software..." - AT4, M14-dis

4.3 Breakdowns

In this subsection, we report the scenarios that exemplify the breakdowns resulting in the security patch management process from ineffective coordination of the socio-technical dependencies and the related constraints.

4.3.1 Sudden Escalations to Patch Schedules. Security patch installation within the allocated patch window is critical in a mission-critical domain like healthcare to avoid unexpected service disruptions. Alpha provided specific patch windows to Beta, usually 4-hours, to install the security patches on production servers (i.e., operating in hospitals) after agreeing with the customers. To adhere to the specified patch window while installing patches, Beta teams were required to plan well ahead to establish each patch's pre-requisites, identifying interdependencies, testing the patches, and obtaining management approval. As such, miscoordination of these conditions would usually lead to unexpected escalations to the planned patch schedules. Given the critical nature of the healthcare operations, security patch installations going beyond the scheduled patch window resulted in devastating consequences

such as life-threatening risk to critical patients from additional service unavailability even for a couple of minutes.

"We realized there's a need to do a sudden change of configuration for [ISP] servers at the time of patching, so our team had to escalate immediately to switch to manual patching because some servers needed Windows approved patches to be rolled out." - BT1, M12

4.3.2 Delays in the Organisation Approvals. All security patches needed to be approved by Alpha's change manager a month before a patch installation to allow Beta teams to prepare in advance for installing a patch. However, we observed some delays in approvals from the Change Management team resulting in abrupt changes to patch installation plans such as shorter patch windows. Such unforeseen changes invoked changes to the dynamics of socio-technical dependencies resulting in breakdowns in the process. For example, having to install the emergency security patches in shorter patch windows warranted re-testing of patches to confirm that patch installation can fit into the shorter patch window, and obtaining reapproval from customers to avoid unexpected service interruptions.

"We were ready for the patch deployment. But this morning [Alpha's] change manager said that the change hasn't been approved yet. So, if it isn't approved prior to scheduled start time, we will have to reschedule it." - BT1, M10

4.3.3 Lack of Dependency Awareness from Localized Work Distribution. Alpha and Beta teams had localized work distribution settings within their team structures. Alpha teams were located in the same office and Beta teams were distributed across different offices in the same city. This structure led to the creation of a lack of dependency awareness of the task assignments and progression between teams. We observed this during status reporting in the meetings as some members were unaware of the tasks progressing in the teams. Their lack of understanding resulted in added challenges of coordinating the inter-team dependencies that inhibited measuring the progression of security patch management tasks.

"I believe [P1 from AT2] is working on this issue at the moment. Do you know when he's likely to get that done? - (BT1)
Not sure, but I would be hoping next week. I would contact him and let you know." - AT2, M13

4.4 Mechanisms

This subsection presents a collection of strategies that have emerged from analyzing the data; the studied teams practiced these mechanisms to manage the dependencies while mediating the constraints.

4.4.1 Early Investigation of Interdependencies. Alpha and Beta teams used patch meetings to discuss the findings from the investigations of the technical dependencies. Since the participants with diverse technical backgrounds and expertise attended the meetings, this configuration enabled knowledge sharing to collaboratively identify dependencies upfront. Failing to identify the dependencies resulted in scenarios of installed patches not working as intended, patches refusing to get installed resulting in rollbacks, and patches going beyond the allocated patch window during installation. Early identification of dependencies helped practitioners to coordinate the task dependencies among teams, and make timely decisions to

address the usual problems such as raising support cases to vendors seeking expert advice and finding workaround solutions.

"From troubleshooting why the last two weeks' patches hadn't worked properly, we can realize that each patch needed to be rebooted at the beginning. Since we failed to do so, the current reboot may have applied the last month's patches." - AT4, M14

4.4.2 Collaborative Decision-Making.
Accurate and timely decision-making is pivotal throughout the patching process. The studied practitioners used patch meetings as a platform to collaboratively decide about vulnerability risk assessment and prioritization, and approval of the patch decisions and schedules. Collaborative decision-making helped the teams to maintain dependency awareness about the decisions and to plan the associated tasks with minimum impact of dependencies. For example, the decisions about patch exemptions involved a collective assessment of the requests. Thus, the awareness of patch exemptions helped the teams to plan their other patch schedules with limited dependencies to the exempted patches, and keep track of the exempted patches in the month and organise the to-do patch list in the following month including them.

"The last item on the agenda is about the [s1] servers that are marked as excluded from patching. We need to decide if they are being exempt from our patching list this month or not because we haven't got the official confirmation whether we're doing the patching or if the [third-party] vendor is doing it again?" - BT1, M13

Other examples of collaborative decision-making involved selecting the optimum patch configurations based on organisational needs, and managing legacy software dependencies. In most cases, the decisions for legacy software dependencies revolved around the need for decommissioning or rebuilding legacy systems, when and how to do it, and how to patch them following the rebuilds. Decisions about the patch schedules for the approved patches helped the teams to coordinate the planning upfront and identify the need for *out-of-band (OOB) patching*. OOB patching refers to the need to allocate an additional patch window when some security patches require more installation time than the allocated patch window due to compound dependencies involved.

4.4.3 Continuous Measuring of Progression.
Alpha teams measured the continual progress of vulnerability remediation through Beta's status reports in meetings and regular vulnerability scan reports. When the scans indicated an increase in the number of vulnerabilities present in the systems, the matters were discussed extensively to remedy the situation. The continuous measurement of the progression enabled the identification of the outliers such as missing patches resulting in the investigation of the causes and coordinating the associated stakeholder dependencies with the third-party vendors.

*"What is the status of internal [s1] server security vulnerabilities? - (AT4)
We're getting our regular scans to measure that. That one is progressing quickly. We will share the report next week" - BT1, M9*

4.4.4 Frequent Communication.
Frequent communication appeared to be essential for effective coordination of dependencies. It helped to erase boundaries between roles, teams, and organisations, and increase cohesion and trust between stakeholders. Teams used various communication mediums such as bi-weekly patch meetings,

email, and Skype. Additionally, the studied practitioners held separate meetings to discuss critical and urgent matters that emerged in between patch meetings or when patch meeting discussions were dragged beyond the allocated time. Patch meetings were the most preferred communication medium as the teams felt more comfortable with direct communication. Communication during patch meetings facilitated collaboration, knowledge sharing, and information exchange about technical and socio-technical matters affecting the patching process, for example, upcoming patch schedules, changes to patching plans such as out-of-band patch schedules, and vendors' patch release information. Regular patch meetings benefited the teams in numerous ways such as allowing visibility into task progression and assignments, staying proactive to potential issues about critical security vulnerabilities, and effectively coordinating security patch management activities.

"[Security Advisor shares the vulnerability remediation progress report on screen] We were averaging 75 high-risk vulnerabilities per server back in 2016 when I joined. As you can see now, we're down to 5 per server. Given the mix of environments we are dealing with, this is amazing. You can see that the frequent patch meetings making a big difference!" - AT4, M10

4.4.5 Load Balancing.
An interesting strategy employed by the studied teams to coordinate patch schedules was load balancing. It was used to balance the patch load in servers at any given patch installation time. Balancing the server load helped reduce service interruptions. Patching dozens of servers at the same time significantly increased the risk of system failure as all the servers go offline at the same time during reboots. Load balancing, on the other hand, helped to run the critical medical services concurrently on another server while the desired server(s) is being rebooted. However, the presence of technical dependencies created difficulties in load balancing. In particular, for instances with one-to-one dependencies such as $(A \rightarrow B)$, the practitioners had to rigorously analyse the interdependencies before planning the load on servers to avoid unexpected system downtime.

"Before we started with the load balancing, we patched 50 servers one night, and just two the next night. So, I suggested we plan to load balance. But there's a lot to manage, especially when we have systems like, system A is redundant to B, and oops! we accidentally took both of them down at the same time to patch." - AT4, M10-dis

4.4.6 Centralized Vulnerability Risk Assessment.
Regularly performing vulnerability risk assessment and prioritization was necessary as it could potentially differ from that of the vendor's assessment based on the organisation's environment. It aided practitioners to plan well in advance to promptly respond to critical security vulnerabilities. To regularly monitor security vulnerabilities, the teams had devised a centralized role in Alpha's Security team (Security Advisor) responsible for scanning and categorizing the vulnerabilities based on teams' ownership. Having a centralized structure helped maintain consistency in vulnerability risk assessments across teams as well as reduce delays in vulnerability assessment and prioritization decisions. Additionally, frequent comparisons with the previous scans assisted with evaluating the vulnerability remediation performance.

"[Security Advisor] gets the global rating of a vulnerability risk and re-assess it to see if it's critical to us and how it can be exploited.

Nesara Dissanayake, Mansooreh Zahedi, Asangi Jayatilaka, and Muhammad Ali Babar

For medium to low risks, we patch in the next cycle, but if it's critical or we're under attack, we'll patch within 48 hours." - AT4, M10-dis

5 DISCUSSION

In this section, we discuss our theory comparing it with the prior related work and elaborate on the broader implications of our theory for practitioners and researchers. Finally, we reflect upon the potential threats to the validity of this study and discuss how we have mitigated them.

Comparing to Related Work: Following Glaser's advice [20, 47], we compare our theory with the existing literature. The prior related work on this topic [13, 27, 32, 37, 39] primarily focuses on the coordination and collaboration challenges but does not provide in-depth details of the causes or the potential strategies to address the coordination challenges leading to delays in applying security patches. A few studies [22, 38, 40] investigate the social dependencies concerning the involvement of multiple internal and external stakeholders. Another set of studies [5, 6, 11, 37] exclusively focus on vendor dependencies that might arise from shared vulnerabilities in software code. In particular, their main focus remains on optimizing patch management by obtaining an equilibrium of an organisation's patch cycle with a vendor's patch-release cycle to minimise cost. An important observation is the absence of *theories* that focus on the socio-technical aspects concerning security patch management in contrast to quantitative *models* and *frameworks* [41]. This is an important point as the theories provide *"basic concepts and underlying mechanisms, which constitute an important counterpart to the knowledge of passing trends"* [23]. In contrast, our theory derived from the gathered data differs from these existing works in several ways as it:
• explains the role of coordination in security patch management as incorporating a multi-dimensional nature across four inter-related dimensions in contrast to focusing on one type such as vendor dependencies reported in [5, 6, 11, 37],
• explains the socio-technical dependencies that create the need for coordination going beyond reporting just the challenges with lack of coordination [22, 38, 40],
• explains the constraints that hinder coordination and shows scenarios of breakdowns resulting from ineffective coordination,
• suggests strategies that can be used for effective coordination,
• offers a comprehensive overview of the impact of coordination in security patching in the mission-critical domain like healthcare,
• presents a theoretical model for future research, and,
• provides guidance to practitioners to overcome patching delays, and increase confidence in their decisions.

Implications for Practitioners: The reported theory can be used to gain an in-depth understanding of the significance and the impact of the role of coordination in security patch management. Practitioners can use this understanding to realize their roles and responsibilities in ensuring coordination effectiveness across different dimensions of the role of coordination. Moreover, practitioners can use the theory as a guide to identify the related dependencies and how they might affect their security patching process. Further, we have observed that adopting these coordination mechanisms

has resulted in a reduction of the unpatched security vulnerabilities in Alpha systems. Hence, our findings may also be useful in exploring the suggested mechanisms in their organisational setting. Additionally, practitioners can benefit from the early detection of constraints and breakdowns to avoid failures.

Implications for Researchers: Given a Grounded Theory study is considered to produce a "mid-ranged" theory based on the contexts studied [18], other researchers can carry out an extension through future research including a more detailed analysis of the present dimensions, new dimensions discovered, or different contexts. Context-specific research investigating how the role of coordination is impacted by contextual factors can result in useful models of coordination in patch management [7, 42]. For example, organisation-level dependencies may not directly apply to small organisations where security patch management is usually handled by one team. Future studies can also investigate the effectiveness of the coordination mechanisms and the context in which they should be employed. The impact of organisational policies is often cited as one of the dominant socio-technical challenges in security patch management [13, 32, 38, 40, 49, 51]. Similarly, future studies can explore how organisational culture affects the role of coordination. Another possibility is to employ the findings in large-scale surveys to evaluate the theory and identify variations in different organisation settings such as in DevOps processes.

While this study is based on data collected from software security patch management, the findings can be directly beneficial to the software development research. This is because patch application is inherently dependent on patch development. An important point to note from our theory is the need to consider the *socio-technical* aspect intrinsic to patch management when developing patches (e.g., future work similar to Li et al. [32]). We show that early identification of the dependencies is the key to avoid patching delays and failures, but lack of automation support presents a key constraint as previously mentioned for timely identification of the dependencies. Therefore, there is a need for research about how Artificial Intelligence (AI) support can be employed in dependency detection in patch development and management. The findings also highlight the important need for further R&D for advanced patch management tools. For example, scanning tools can be enhanced to customize the software dependencies such as excluding exempted patches to provide real-time feedback that assists practitioners with accurate decision-making. Furthermore, our theory provides an in-depth understanding of how the role of coordination impacts the mission-critical domain, particularly, healthcare. An understanding of the causes of unexpected service interruptions can help researchers to devise strategies to avoid such downtime. While the research into dynamic software updating [24] (DSU) attempting to address this issue is progressing, our results can be useful information for future research that investigates the effectiveness of the developed strategy in mission-critical contexts.

Threats to Validity: A Grounded Theory study does not affirm generalization as the theory formulation is pertinent to the studied context [17, 20]. The context of this study is limited to the cases studied in security patch management in the domain of healthcare.

Nevertheless, we believe that our theory can be recreated in other contexts and modified.

In terms of data representativeness, our data collection is limited to the observations of patch meetings, post-meeting discussions, and analysis of meeting minutes and patch mailing thread as described in Section 3.1. We acknowledge that more data sources such as interviews and surveys can be incorporated in future studies to increase the scope of the analysis and verifiability of the theory [44].

While employing Grounded Theory procedures permits the data analysis to be grounded in collected data, there is a threat of subjectivity of the data analysis referred to as the "uncodifiable step" [1, 30]. To alleviate this threat, we regularly held internal discussions on the emergent findings throughout the study as described in Section 3.2. In addition, the findings were further cross-checked with a senior member of Alpha's security team to ensure we have accurately interpreted the theory from the observed practices.

The verifiability of a grounded theory can be deduced from the robustness of the research method, and evidence of theory formulation from its application [18]. To confirm verifiability, we have described our application of the Glaserian version of GT in detail (Section 3) and included quotations from the underlying data in the findings (Section 4). These details provide evidence of how our theory meets the GT evaluation criteria: the generated categories *fit* the underlying data (see Figure 1); the theory can *work* as it explains the main concerns of the participants in patch meetings; the theory has *relevance* to the domain of software security patch management; and the theory is open to *modification* based on future studies in different contexts [17, 47].

6 CONCLUSION

We present the Theory of the Role of Coordination in Software Security Patch Management. The developed theory explains the effects of coordination in the patch management process across four inter-related dimensions namely *causes, breakdowns, constraints,* and *mechanisms.* Our theory is based on a longitudinal Grounded Theory study of 51 patch meeting observations involving 21 industry practitioners in two organisations in the healthcare domain over a duration of 9 months. We provide the grounded evidence that the role of coordination represents a core concern, contrasting with a perception among the SE community that automation and tooling alone can be sufficient to achieve success in patching and highlight the need to have a delicate balance between the socio-technical concerns such as coordination and automation to reduce delays, which is often unrecognised in the existing literature.

Overall, besides providing a holistic understanding of the role of coordination in security patch management that is based on empirical evidence and grounded in practice, our study is the first attempt to investigate in-depth the socio-technical aspects of security patch management in the mission-critical healthcare domain. The theory provides important insights for practitioners to avoid patching delays and failures and enhance confidence in their decisions, and for researchers to shape their work on patch development to address the practical concerns in patch application. The findings can also be used for developing the next generation of AI-enabled tools for supporting the patch management process.

A   OBSERVATION PROTOCOL

Table 3: Observation protocol

Topic	Question
Participants	What are the roles and other details of the participants? • Number of attendees • Role (manager, system administrator, etc.) • Affiliation (Alpha or Beta) • Team (Security, Windows, Server, etc.) Is someone acting as a facilitator? • Who? • How is he/she facilitating the meeting?
Communication	What is the communication channel? How does the communication happen? • Directed / indirect questions? • Active participation in communication? • Any roles that are most active in communication?
Activities	What are the various discussions and activities? • Topics discussed • Challenges discussed • Activities (demonstrations etc.)
Objects	What resources/media are used? • Presentation slides, excel sheets, tools, etc.
Collaboration	How do the participants interact and corporate with each other?
Events	Are there any particular events or anything unanticipated?
Time	When does the meeting start? What is the sequence of events? When does the meeting end?
Goals	What are the participants trying to accomplish?
Feelings	How is the atmosphere?
Closing	How is the meeting ended? Is there a post-meeting planned? Is there anything discussed about the next meeting?

REFERENCES

[1] Sebastian Baltes and Stephan Diehl. 2018. Towards a Theory of Software Development Expertise. In *26th ACM Joint European Software Engineering Conference*

and Symposium on the Foundations of Software Engineering (ESEC/FSE'18). ACM, 187–200. https://doi.org/10.1145/3236024.3236061

[2] Saskia Bick, Kai Spohrer, Rashina Hoda, Alexander Scheerer, and Armin Heinzl. 2017. Coordination Challenges in Large-Scale Software Development: A Case Study of Planning Misalignment in Hybrid Settings. *IEEE Transactions on Software Engineering* 44, 10 (2017), 932–950. https://doi.org/10.1109/TSE.2017.2730870

[3] Marcelo Cataldo and James D. Herbsleb. 2012. Coordination Breakdowns and Their Impact on Development Productivity and Software Failures. *IEEE Transactions on Software Engineering* 39, 3 (2012), 343–360. https://doi.org/10.1109/TSE.2012.32

[4] Marcelo Cataldo, Patrick A. Wagstrom, James D. Herbsleb, and Kathleen M. Carley. 2006. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. In *Proceedings of the 2006 20th anniversary Conference on Computer Supported Cooperative Work (CSCW)*. ACM, 353–362. https://doi.org/10.1145/1180875.1180929

[5] Huseyin Cavusoglu, Hasan Cavusoglu, and Jun Zhang. 2006. Economics of Security Patch Management. In *WEIS*. Citeseer, 1–10.

[6] Hasan Cavusoglu, Huseyin Cavusoglu, and Jun Zhang. 2008. Security Patch Management: Share the Burden or Share the Damage? *Management Science* 54, 4 (2008), 657–670. https://doi.org/10.1287/mnsc.1070.0794

[7] Zhi Cheng, Angelika Dimoka, and Paul A Pavlou. 2016. Context may be King, but generalizability is the Emperor! *Journal of Information Technology* 31, 3 (2016), 257–264. https://doi.org/10.1057/s41265-016-0005-7

[8] Olivier Crameri, Nikola Knezevic, Dejan Kostic, Ricardo Bianchini, and Willy Zwaenepoel. 2007. Staged Deployment in Mirage, an Integrated Software Upgrade Testing and Distribution System. *ACM SIGOPS Operating Systems Review* 41, 6 (2007), 221–236. https://doi.org/10.1145/1323293.1294283

[9] Kevin Crowston. 2003. A taxonomy of organizational dependencies and coordination mechanisms. *Organizing Business Knowledge: The MIT Process Handbook* (2003), 85–108.

[10] Andrew H. Van de Ven, Andre L. Delbecq, and Richard Koenig Jr. 1976. Determinants of Coordination Modes within Organizations. *American Sociological Review* (1976), 322–338.

[11] Debabrata Dey, Atanu Lahiri, and Guoying Zhang. 2015. Optimal Policies for Security Patch Management. *INFORMS Journal on Computing* 27, 3 (2015), 462–477. https://doi.org/10.1287/ijoc.2014.0638

[12] Constanze Dietrich, Katharina Krombholz, Kevin Borgolte, and Tobias Fiebig. 2018. Investigating System Operators' Perspective on Security Misconfigurations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 1272–1289. https://doi.org/10.1145/3243734.3243794

[13] Nesara Dissanayake, Asangi Jayatilaka, Mansooreh Zahedi, and M Ali Babar. 2020. Software Security Patch Management–A Systematic Literature Review of Challenges, Approaches, Tools and Practices. arXiv:2012.00544

[14] Ruian Duan, Ashish Bijlani, Yang Ji, Omar Alrawi, Yiyuan Xiong, Moses Ike, Brendan Saltaformaggio, and Wenke Lee. 2019. Automating Patching of Vulnerable Open-Source Software Versions in Application Binaries. In *NDSS*. https://dx.doi.org/10.14722/ndss.2019.23126

[15] Melissa Eddy and Nicole Perlroth. 2020. *Cyber Attack Suspected in German Woman's Death.* Retrieved February 5, 2021 from https://www.nytimes.com/2020/09/18/world/europe/cyber-attack-germany-ransomware-death.html?smid=tw-share

[16] Svetla Georgieva and George Allan. 2008. Best Practices in Project Management Through a Grounded Theory Lens. *Electronic Journal of Business Research Methods* 6, 1 (2008).

[17] Barney G. Glaser. 1978. *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory.* Sociology Press, Mill Valley, CA.

[18] Barney G. Glaser. 1992. *Basics of Grounded Theory Analysis: Emergence vs Forcing.* Sociology Press, Mill Valley, CA.

[19] Barney G. Glaser. 2005. *The Grounded Theory Perspective III: Theoretical Coding.* Sociology Press, Mill Valley, CA.

[20] Barney G. Glaser and Anselmo L. Strauss. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Aldine Transaction, Chicago.

[21] Dan Goodin. 2017. *Failure to patch two-month-old bug led to massive Equifax breach.* Retrieved February 25, 2021 from https://arstechnica.com/information-technology/2017/09/massive-equifax-breach-caused-by-failure-to-patch-two-month-old-bug/

[22] Tanja Hanauer, Wolfgang Hommel, Stefan Metzger, and Daniela Pöhn. 2018. A Process Framework for Stakeholder-specific Visualization of Security Metrics. In *Proceedings of the 13th International Conference on Availability, Reliability and Security.* 1–10. https://doi.org/10.1145/3230833.3232855

[23] Jo E. Hannay, Dag IK Sjøberg, and Tore Dybå. 2007. A Systematic Review of Theory Use in Software Engineering Experiments. *IEEE Transactions on Software Engineering* 33, 2 (2007), 87–107. https://doi.org/10.1109/TSE.2007.12

[24] Michael Hicks and Scott Nettles. 2005. Dynamic Software Updating. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 27, 6 (2005), 1049–1096. https://doi.org/10.1145/381694.378798

[25] Rashina Hoda, James Noble, and Stuart Marshall. 2011. The impact of inadequate customer collaboration on self-organizing Agile teams. *Information and Software Technology* 53, 5 (2011), 521–534. https://doi.org/10.1016/j.infsof.2010.10.009

[26] Rashina Hoda, James Noble, and Stuart Marshall. 2012. Developing a grounded theory to explain the practices of self-organizing Agile teams. *Empirical Software Engineering* 17, 6 (2012), 609–639. https://doi.org/10.1007/s10664-011-9161-0

[27] Hai Huang, Salman Baset, Chunqiang Tang, Ashu Gupta, KN Madhu Sudhan, Fazal Feroze, Rajesh Garg, and Sumithra Ravichandran. 2012. Patch Management Automation for Enterprise Cloud. In *IEEE Network Operations and Management Symposium.* IEEE, 691–705. https://doi.org/10.1109/NOMS.2012.6211988

[28] Adam Jenkins, Pieris Kalligeros, Kami Vaniea, and Maria K. Wolters. 2020. "Anyone Else Seeing this Error?": Community, System Administrators, and Patch Information. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 105–119. https://doi.org/10.1109/EuroSP48549.2020.00015

[29] Robert E. Kraut and Lynn A. Streeter. 1995. Coordination in software development. *Commun. ACM* 38, 3 (1995), 69–82.

[30] Ann Langley. 1999. Strategies for theorizing from process data. *Academy of Management review* 24, 4 (1999), 691–710. https://doi.org/10.5465/amr.1999.2553248

[31] Frank Li and Vern Paxson. 2017. A Large-Scale Empirical Study of Security Patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2201–2215. https://doi.org/10.1145/3133956.3134072

[32] Frank Li, Lisa Rogers, Arunesh Mathur, Nathan Malkin, and Marshini Chetty. 2019. Keepers of the Machines: Examining How System Administrators Manage Software Updates. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. USENIX Association, 273–288.

[33] Thomas W. Malone and Kevin Crowston. 1990. What is coordination theory and how can it help design cooperative work systems?. In *Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work (CSCW)*. ACM, 357–370. https://doi.org/10.1145/99332.99367

[34] Thomas W. Malone and Kevin Crowston. 1991. Toward an interdisciplinary theory of coordination. *Organizing Business Knowledge: The MIT Process Handbook* (1991).

[35] Lee Mathews. 2017. *Equifax Data Breach Impacts 143 Million Americans.* Retrieved February 25, 2021 from https://www.forbes.com/sites/leemathews/2017/09/07/equifax-data-breach-impacts-143-million-americans/#1abcaed0356f

[36] John E. Mathieu, Michelle A. Marks, and Stephen J. Zaccaro. 2001. Multiteam systems. *Handbook of Industrial, Work and Organizational Psychology* 2 (2001), 289–313.

[37] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. 2015. The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 692–708. https://doi.org/10.1109/SP.2015.48

[38] Felicia M. Nicastro. 2003. Security Patch Management. *Inf. Secur. J. A Glob. Perspect.* 12, 5 (2003), 5–18. https://doi.org/10.1201/1086/43808.12.5.20031101/78486.2

[39] Shaya Potter and Jason Nieh. 2005. Reducing Downtime Due to System Maintenance and Upgrades. In *Proceedings of the 19th USENIX Systems Administration Conference.* IEEE, 6–6.

[40] Michal Procházka, Daniel Kouril, Romain Wartel, Christos Kanellopoulos, and Christos Triantafyllidis. 2011. A Race for Security: Identifying Vulnerabilities on 50 000 Hosts Faster than Attackers. In *Proceedings of Science (PoS). International Symposium on Grid and Clouds.*

[41] Paul Ralph. 2019. Toward Methodological Guidelines for Process Theories and Taxonomies in Software Engineering. *IEEE Transactions on Software Engineering* 45, 7 (2019), 712–735. https://doi.org/10.1109/TSE.2018.2796554

[42] Pilar Rodriguez, Cathy Urquhart, and Emilia Mendes. 2020. A Theory of Value for Value-based Feature Selection in Software Engineering. *IEEE Transactions on Software Engineering* (2020). https://doi.org/10.1109/TSE.2020.2989666

[43] Accenture Security. 2020. *2020 Cyber Threatscape Report.* Retrieved February 9, 2021 from https://www.accenture.com/_acnmedia/PDF-136/Accenture-2020-Cyber-Threatscape-Full-Report.pdf

[44] Jefferson Silva, Igor Wiese, Daniel M. German, Christoph Treude, Marco Aurélio Gerosa, and Igor Steinmacher. 2020. A Theory of the Engagement in Open Source Projects via Summer of Code Programs. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'20)*. ACM, 421–431. https://doi.org/10.1145/3368089.3409724

[45] Murugiah Souppaya and Karen Scarfone. 2013. Guide to Enterprise Patch Management Technologies. *NIST Special Publication* 800 (2013), 40. http://dx.doi.org/10.6028/NIST.SP.800-40r3

[46] James P. Spradley. 1980. *Participant Observation* (1st ed.). Rinehart and Winston, Austin, TX.

[47] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. ACM, 120–131. https://doi.org/10.1145/2884781.2884833

[48] Diane E. Strode, Sid L. Huff, Beverley Hope, and Sebastian Link. 2012. Coordination in co-located agile software development projects. *Journal of Systems and*

Software 85, 6 (2012), 1222–1238. https://doi.org/10.1016/j.jss.2012.02.017

[49] Christian Tiefenau, Maximilian Häring, Katharina Krombholz, and Emanuel von Zezschwitz. 2020. Security, Availability, and Multiple Information Sources: Exploring Update Behavior of System Administrators. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. USENIX Association, 239–258.

[50] Cathy Urquhart. 2013. *Grounded Theory for Qualitative Research: A Practical Guide*. Sage.

[51] Bo Yang, Naga Ayachitula, Sai Zeng, and Rajeev Puri. 2011. SLA-driven Applicability Analysis for Patch Management. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, 438–445. https://doi.org/10.1109/INM.2011.5990544