

To Deploy, or Not to Deploy, That is the Question

A qualitative study of the decision-making
experiences of engineers deploying software
changes in production

Jessica DeVita | LUND UNIVERSITY



**To Deploy, or Not to Deploy,
That is the Question**

**A qualitative study of the decision-making experiences of
engineers who deploy software changes in production**

Jessica DeVita

Lund 2023

**Under the supervision of:
Dr Tove Frykmer**

To Deploy, or Not to Deploy, That is the Question

A qualitative study of the decision-making experiences of engineers who deploy software changes in production

Jessica DeVita

Number of pages: 108

Illustrations: 6

Keywords

Automation, Common ground, Complexity, Coordination, COVID19 pandemic, Deployment decision, Incident response, Mental model, Naturalistic decision-making, Phenomenology, Resilience, Risk, Software engineering, SRE, Strategy, Temporal reasoning, Uncertainty, Web operations

Abstract

The web services that millions of people count on require large-scale computer resources (servers, nodes, networking) and an intricate set of interdependent software services. Under pressure to continually improve these services, code and configuration changes are deployed hundreds or thousands of times every day. Unforeseen issues and vulnerabilities during the deployment process can lead to costly incidents. As society increasingly relies on web services across industries, it is imperative to better understand how engineers make decisions when deploying code and configuration changes to create safer deployment mechanisms that increase confidence for engineers. There is a lack of research on the lived experiences of engineers who deploy changes to software or configuration in production environments. In this qualitative study, 15 participants were interviewed to better understand their lived experiences deploying code and configuration changes. Phenomenological explication guided the data analysis. The findings of this study indicated that perceptions of uncertainty and risk, temporal reasoning, and relationships influenced engineers' deployment decisions. Participants also offered feedback on what they would change to make their deployment decision-making safer. Organizations should reduce the risk and uncertainty engineers face when making deployment decisions, recognize how temporal reasoning influences decision-making, and cultivate environments in which relationships are recognized as influencing engineers' confidence in decision-making.

© Copyright: Division of Risk Management and Societal Safety, Faculty of Engineering
Lund University, Lund 2023

Avdelningen för Riskhantering och samhällssäkerhet, Lunds tekniska högskola, Lunds universitet, Lund 2023.

Riskhantering och samhällssäkerhet
Lunds tekniska högskola
Lunds universitet
Box 118
221 00 Lund

<http://www.risk.lth.se>

Telefon: 046 - 222 73 60

Division of Risk Management and Societal Safety
Faculty of Engineering
Lund University
P.O. Box 118
SE-221 00 Lund
Sweden

<http://www.risk.lth.se>

Telephone: +46 46 222 73 60

List of Tables and Figures

Tables

Table 1 <i>Literature Review Topics and Authors</i>	6
Table 2 <i>Fitts List</i>	10
Table 3 <i>Participant Demographics</i>	18
Table 4 <i>Semi-structured Interview Questions</i>	19
Table 5 <i>Data Analysis Process</i>	21
Table 6 <i>Themes and Clusters</i>	25
Table 7 <i>Participant Deployment Context</i>	26
Table 8 <i>Theme 1 Clusters and Examples</i>	28
Table 9 <i>Theme 2 Clusters and Examples</i>	40
Table 10 <i>Theme 3 Cluster and Examples</i>	42
Table 11 <i>Theme 4 Desired Changes and Examples</i>	45
Table 12 <i>Research Questions</i>	60

Figures

Figure 1 <i>HABA-MABA (Humans are better at... Machines are better at...)</i>	9
Figure 2 <i>System of Interest: Deployment Decision-Making</i>	17
Figure 3 <i>Tools Used by Participants</i>	27
Figure 4 <i>Klein et al.'s Recognition-Primed Decision Model</i>	48
Figure 5 <i>Above the Line/Below the Line Framework</i>	49
Figure 6 <i>Example of a Pull Request with Side by Side "diff"</i>	56

Contents

Research Questions	4
Structure of the Study	5
Literature Review	6
Decision Theories	7
Classical decision theory	7
Heuristics and biases	7
Skills, rules, and knowledge	8
Naturalistic decision-making	8
Coordination: Common ground and joint activity	9
Functional allocation: Ironies of automation	9
Literature Related to Decision-making	11
Complexity and mental models	11
Temporal reasoning and mental models	11
Time pressure	12
Risk and uncertainty	13
Strategies for decision-making in adverse conditions	13
Software deployment research	14
Methodology	17
Participants	17
Demographics	18
Interviews	18
Data Analysis/Explicitation	19
Research Ethics	21
Limitations	23
Validity	24
Generalizability	24
Findings	25
Theme 1: Deployment Decision-Making is Risky	27
Cluster 1: Risk mitigation strategies	28
Cluster 2: The deployment experience	30
Cluster 3: Surprise! Things are broken now	33
Cluster 4: Managing the outcome of deployment decisions	39
Theme 2: Temporal reasoning influences deployment decision-making	40
Cluster 5: Time as a pressure	40
Cluster 6: Time as cause	41
Cluster 7: Duration of the deployment experience	41
Theme 3: Relationships influence deployment decision-making	42

Cluster 8: Relationships with self.....	42
Cluster 9: Relationships with family and friends.....	43
Cluster 10: Relationships with co-workers	44
Cluster 11: Relationships with management	44
Theme 4: What would engineers change?.....	45
Testing.....	45
Discussion	47
Decision-making theories, mental models, and temporal reasoning.....	47
Mental models and temporal reasoning.....	48
Common ground and coordination in joint activity	50
Common ground breakdown: Automation surprises.....	51
Ironies of automation.....	52
Automation introduces new risks	53
Deployment decisions are risky	53
General description of the deployment experience.....	54
COVID-19 Pandemic.....	54
Holiday code freezes	55
Planned changes	55
Emergency changes	56
Strategies	56
Issues and Implications.....	61
References.....	65
Appendices	73
Appendix A: Recruitment	73
Appendix B: Recruitment Flyer.....	74
Appendix C: Screening Materials	75
Appendix D: Participant Consent.....	76
Appendix E: Participant Narratives	77

Glossary

<i>Term</i>	<i>Category</i>	<i>Definition</i>
Ansible	Automation	Configures and manages systems, deploys applications, orchestrates workflows
API	Software interface	Defines how different components or systems communicate and exchange data
AWS	Cloud provider	Cloud services, such as compute, storage, database, analytics, networking, and more
Azure	Cloud provider	Cloud services, such as compute, storage, database, analytics, networking
Azure DevOps	Software development platform	Provides version control, agile planning, testing, CI/CD
Cache	Data storage	Stores frequently used data or files for faster access
Canary	Testing strategy	Releases new features to a subset of users before rolling out to everyone
CI/CD	Software development practice	Automates the process of continuous integration and continuous delivery or deployment
CI/CD pipeline	Software delivery workflow	Defines the stages and steps for building, testing, and deploying software continuously
CosmosDB	Database service	Provides a globally distributed, multi-model database for Azure
DataDog	Monitoring	Integrates with various tools and services to provide visibility into the infrastructure stack
Database migration	Database operation	Transfers data from one database system to another
Deployment	Pushing a software change to production	Process involving the release of software, installation, and activation. It also involves updates, reconfiguration, redeploying, and removal.
Docker	Container platform	Enables building, running, and sharing applications using containers
GCP	Cloud provider	Provides a range of cloud services, such as compute, storage, database, analytics, and networking
GitHub	Version control	Hosts Git repositories and provides collaboration features
Git commit	Version control operation	Records changes to a repository or branch
Jira	Work Tracking System	Tool for tracking software development work
Kubernetes	Infrastructure platform	Open-source system for automating the deployment and management of containerized applications
Postmortem	Post-incident process	In the tech industry, used to describe documents and meetings to discuss the incident/outage failure mode
Pull request (PR)	Propose change to code	A proposal to merge your changes to the code into the production branch
Secure Socket Layer (SSL)	Security infrastructure	Encrypts the link between a web server and a browser
Slack	Messaging/Chat	Collaborative messaging/chat application

Acknowledgments

To my beautiful family; Jason, Maxwell, Mason, and Matteo I love you and appreciate you so much!

This thesis is dedicated to Dr. Richard Cook and his wife Karen Hansen Cook. Dr. Cook's teachings, writings, and mentorship on safety, culture, management, and incident analysis taught me *how to see things*. Now, I cannot *unsee* the things, and for that I am grateful. The "root cause" is John Allspaw. In 2015, he sent me a PDF of Common Ground and Coordination in Joint Activity (Klein, et al., 2005) and later on, encouraged me to apply to the Lund program.

What a privilege it has been to learn from Dr. Anthony Smoker, Dr. Johan Bergstrom, Dr. Tove Frykmer, Dr. David Woods, Dr. Laura Maguire, Dr. James Nyce, Dr. Roel van Winsen and all the guest lecturers. Special thanks to Dr. Steven Shorrock, Jennelle Crothers, Chad Todd, Pirmin Schuermann, Dr. Lorin Hochstein, Nick Stenning, William Thurston, Jennifer Davis, J. Sawyer, Rick Carlson, Pamela Sieg, Aimee, Sebastian, and Sharon.

Finally, this research would not be possible without the participants who took time out of their busy schedules to share their experiences. It is my sincere hope that I have told your stories faithfully so that the technology industry better understands the challenges of deployment decision-making and can learn meaningful lessons from you.

With gratitude,
Jessica DeVita

A single decision can have dramatic effects that propagate rapidly and widely. (Rasmussen, 1997, p. 186)

In the time it takes the average person to watch a favorite movie on a Friday night, multiple changes to the software code and hardware configurations delivering that experience have been deployed without interruption to the movie-viewing experience (Bhartiya, 2018). Operation of digital services require large-scale computing resources (servers, nodes, networking) and a complex set of interdependent software services, each configured in specialized ways. Since people expect these services to be operational 24 hours a day, 7 days a week, and 365 days a year, engineers face enormous pressure to keep the services running while continually adding features and improving existing functionality. As a result, engineers are making decisions to deploy code and configure changes hundreds or thousands of times every day (Dijkstra, 2017). Allspaw and Cook (2018) explained the complexity of engineers' work:

The modern "system" is a constantly changing melange of hardware and software embedded in a variable world. Together, the hyperdistribution, fluctuant composition, constantly varying workload, and continuous modification of modern technology assemblies comprises a unique challenge to those who design, maintain, diagnose, and repair them. (p. 1)

While many of these deployments are uneventful and proceed without issues, sometimes they can trigger costly incidents and outages (Ballinger, 2020; Nolan, 2020). Unforeseen permissions issues, network latency, sudden increase in demands, and security vulnerabilities may manifest in production (the live system). The pressure on engineers is exacerbated by complex test and release automation that makes the system harder to troubleshoot when things go wrong. Ideally, changes can be quickly reverted, but sometimes changes introduce latent bugs that go unnoticed. New failure modes appear that make deployment decisions even more consequential and the outcome difficult or impossible to predict. The following three incidents demonstrate a variety of failures that were triggered by deployment decisions.

1. On May 12, 2020, a configuration change at Slack (a communications and chat platform) caused an increased load that enabled just the right conditions for triggering a long-standing performance bug (Nolan, 2020, para. 2). The problem was identified, engineers rolled back the change and thought they fixed the problem, but 8 hours later, a complete outage occurred.

2. On December 14th, 2020, Google experienced a global outage that lasted for nearly an hour, preventing everyone who used Google from logging in. From their status page, a description of the failure mode was provided:

Google Cloud Platform and Google Workspace experienced a global outage affecting all services which require Google account authentication for a duration of 50 min. The root cause was an issue in our automated quota management system, which reduced the capacity for Google's central identity management system, causing it to return errors globally. As a result, we couldn't verify that users' requests were authenticated and served. (Google, 2020, para. 7)

3. On May 12, 2021, a software deployment at Fastly introduced a bug that did not manifest immediately (Fastly's content delivery network [CDN] technology makes sure websites and apps load quickly on your device, no matter where you are in the world). On June 8, a Fastly customer made a change to their own Fastly CDN configuration, which created the precise conditions to trigger the bug that had been introduced earlier. Service was interrupted for *many* large-scale customers who also depended on Fastly. During the outage, citizens in the United Kingdom could not access government services and were "unable to renew passports, apply for tax allowances, or obtain driving licenses during the outage" (Warren, 2021, para. 4). The incident was described by Fastly on their blog

On May 12, we began a software deployment that introduced a bug that could be triggered by a specific customer configuration under specific circumstances. Early June 8, a customer pushed a valid configuration change that included the specific circumstances that triggered the bug, which caused 85% of our network to return errors. (Rockwell, 2021, para. 3)

This incident highlighted the extent to which CDNs are relied on by web services and the surprise that a configuration change made by one customer could affect a majority of Fastly customers.

These outages indicate that every change carries variable potential for triggering an incident or an outage. Woods (2017) noted that for engineers working on these systems,

the technical problem almost always requires specific actions, and these actions entail exposure to risk. Entering console commands, restarting services, making changes to, and deploying code, altering network settings, and the myriad other changes that may be required to address the outage all present some risk. (p. 21)

Often in the aftermath of incidents or outages, the deployment decisions that preceded the event are brought under a microscope, and the cause is usually attributed to human error (Blackwell, 2016; Hanna, 2013). In my experience working in the domain of software engineering and web operations, incident analysis efforts focus on measuring time to respond, time to repair, time between failures, and other “shallow metrics” (Allspaw & Cook, 2018). The experience of the engineer is rarely explored except to assign blame and declare (counterfactually) what they should have done.

As society increasingly relies on software to support safety critical digital services such as banking, healthcare, financial markets, air travel, and technical infrastructure for utilities such as water, power, gas, and wastewater treatment, it is imperative to understand how engineers make deployment decisions. This topic represents a new context for safety science (Allspaw, 2015), as

there remains a significant divide between academic research and practical application in the tech industry (Shorrock, 2019, p. 225). Several researchers have applied safety theories from studies in domains like patient safety (Cook, 1999), aviation (Rierson, 2001), and space exploration (Vaughan, 1996) to understand how engineers respond to incidents (Grayson, 2018), manage the costs of coordination during incident response (Maguire, 2020), and use heuristics to resolve outages in software and web operations (Allspaw, 2018). Other literature has described types of deployment decisions (Knodel & Naab, 2014) and proposed models for software change activities (Borg et al., 2016) and software release engineering processes (Adams & McIntosh, 2016; Knodel & Naab, 2014). However, these studies do not offer any explanations or descriptions of the decision-makers' lived experience. To the best of my knowledge, no study has focused specifically on the deployment decision-making experiences of engineers. Considering the criticality of keeping critical digital services operational, as well as the gap in the literature, I sought to understand how engineers experience decisions to deploy changes given that even a single character change can lead to severe outages that may affect human life.

Research Questions

I am interested in the lived experiences of engineers who regularly make deployment decisions and make changes to production. Therefore, the following research questions guided the study:

1. What are the deployment decision-making experiences of engineers in the field of software and web operations making changes to technical systems?
2. How do they experience the decision to deploy those changes?
 - a. How were engineers' experiences difficult or frustrating?
 - b. How was time pressure a factor in engineers' deployment decision-making experiences?
 - c. How did engineers derive confidence in the decision to deploy?

- d. How did engineers' previous work experiences factor into their decisions to deploy the change?

A better understanding of engineers' lived experience of deploying a change could guide the design of deployment automation tooling that is safer to operate.

Structure of the Study

The first chapter of the thesis introduces the topic and describes the research question. The second chapter offers an overview of literature including brief descriptions of several decision-making theories and research relevant to the current study. The third chapter details the qualitative methodology employed in this study. Chapter four offers the findings of the study, which were drawn from the data analysis and organized into four themes and 15 clusters that described the lived experiences of how engineers decide to deploy code and configuration changes. The fifth chapter offers a discussion of the findings, situated within the literature. Lastly, chapter six describes the implications of the findings and suggestions for future research.

Literature Review

This literature review will briefly explore several theories of decision-making: Classical decision theory; heuristics and biases; the skills, rules, and knowledge framework; and naturalistic decision-making. Each of these theories offers a unique perspective on how decisions are made. This review then examines how coordination, automation, complexity, and risk influence decision-making in a variety of domains including software deployment (see Table 1).

Table 1

Literature Review Topics and Authors

Theories and Research Topics	Authors
Classical Decision Theory	Leveson, (2009), Lipshitz (1993), Orasanu & Connolly (1993)
Heuristics and Biases	Slovic et al. (2005), Slovic et al. (2007), Tversky & Kahneman (1974)
Skills, Rules, and Knowledge	Rasmussen (1983)
Naturalistic Decision-Making	Klein (1993)
Common Ground and Joint Activity	Klein et al. (2005)
Functional Allocation	Bainbridge (1983), Fitts (1951)
Complexity and Mental Models	Lim & Klein (2006), Rasmussen (1983), Shorrock (2022), Sougne et al. (1993), Woods (2017)
Temporal Reasoning and Mental Models	Block (1990), De Keyser (1990), Sougne et al. (1993), Suddendorf et al. (2018)
Time Pressure	Khoo & Mosier (2008), Ordóñez et al. (2015), Rochlin et al. (1987), Starbuck & Farjoun (2009)
Risk and Uncertainty	Kabir et al. (2021), Slovic et al. (2005), Woods (2017)
Decision-Making in Adversity	Morel et al. (2008)
Research From Software Industry	Adams & McIntosh (2016), Allspaw (2015), Cook (2020), Grayson (2018), Kerzazi & Adams (2016), Maguire (2020), Schermann et al. (2016), Woods (1993)

Decision Theories

Classical decision theory

Classical decision theory (CDT) treats decisions as choices (Orasanu & Connolly, 1993) and optimal decisions are viewed as distinct processes that can be studied as isolated phenomena (Leveson, 2011). However, CDT fails to accurately describe decision-making in complex, human-operated automated systems, according to Lipschitz (1993). He argued that goals may shift in dynamic environments, and problems may be ill-structured and uncertain, leading to decisions being made based on incomplete and rapidly changing information. Lipschitz advocated for comprehensive, contextualized, and dynamic models in decision-making rather than linear, oversimplified prescriptions of the process.

Heuristics and biases

When considering the probability of outcomes, Tversky and Kahneman (1974) argued that people tend to avoid the complex mental tasks of assessing probabilities, instead relying on a set of heuristics to make judgments in uncertain situations: “(a) representativeness: where people judge the probability that an object or event belongs to class or process B; (b) availability: where people assess the frequency or plausibility of an event; and (c) adjustment from an anchor and making predictions from an initial value” (p. 1131). The heuristics lens can be effective for making decisions in uncertain situations, but it often leads to systematic errors.

People make decisions in risky situations intuitively, and feelings of risk are used as a “mental shortcut,” also known as the affect heuristic, which is faster than “weighing the pros and cons . . . or retrieving relevant examples from memory” (Slovic et al., 2005, p. S36). An inverse relation between perceived risk and perceived benefit has been linked to the strength of positive or negative affect associated with the activity. People “base their judgments of an activity or a technology not only on what they think but how they feel about it—favorable = risk low, unfavorable = high risk/low benefit” (Slovic et al., 2005, p. S36). Slovic et al. (2007) explained:

In this sense, the affect heuristic enables us to be rational actors in many important situations. But not in all situations. It works beautifully when our experience enables us to anticipate accurately how we will like the consequences of our decisions. It fails miserably when the consequences turn out to be much different in character than we anticipated. (p. 1350)

Skills, rules, and knowledge

Rasmussen (1983) created the skills, rules, and knowledge (SRK) model, which better addressed the complexity of decision-making by acknowledging different types of decision-making. According to the SRK framework, there are three types of decision-making: skills-based, rules-based, and knowledge-based. Skills-based decisions are grounded on the individual's abilities, are habitual, and are made without conscious attention. Rules-based decisions are founded on guidelines and procedures. This type of decision-making is more explicit than skill-based decision-making. Knowledge-based decisions are made when there is a thorough understanding of the problem and decisions are made consciously in an unfamiliar situation.

Naturalistic decision-making

According to the theory of naturalistic decision-making, people tend to evaluate their situation and receive situational feedback, rather than choosing from several options (Klein, 1993). Klein's (1993) recognition-primed decision (RPD) model states that people use situation assessment to generate a plausible course of action and use mental simulation to evaluate that course of action (p. 138). Klein (1993) further explained that experienced decision-makers can usually identify the course of action quickly, without the need for a long deliberation.

An understanding of decision-making theories provides a useful framework for the study. I also reviewed studies that have examined decision-making in adverse conditions and complex situations, as well as the roles of teamwork, risk and uncertainty, and time pressure in other types of decision-making experiences. The following is a summary of the studies I found.

Coordination: Common ground and joint activity

Research from Klein et al. (2005) on common ground and coordination described how people work together to achieve a goal:

Key aspects of common ground include: 1) The types of knowledge, beliefs and assumptions that are important for joint activity, including knowledge of roles and functions, standard routines, and so forth; 2) Mechanisms for carrying out the grounding process: to prepare, monitor and sustain, catch and repair breakdowns; 3) The Basic Compact committing the parties in a joint activity to continually inspect and adjust common ground. (p. 37)

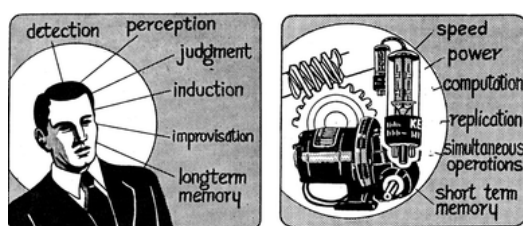
Common ground can be extended to interactions with machine agents. The requirements are helpful in describing the problems that people have working with automation. Engineers often work in teams to make deployment decisions, and the code change itself often contains contributions from several engineers. The change is then pushed to production using automated deployment tools.

Functional allocation: Ironies of automation

Theorists have also examined how decision-making has been influenced by an increasing reliance on automation. Initially, theories of functional allocation suggested that certain tasks would be better suited for humans to perform, while others should be done by a machine (see Figure 1). Fitts (1951) proposed a set of 11 statements known as “Fitt’s List” (see Table 2) that allocated tasks according to the abilities of humans and machines.

Figure 1

HABA-MABA (Humans are better at... Machines are better at...)



Note: (Fitts et al., 1951, pp. 7–8).

Table 2

Fitts List

Humans appear to surpass present-day machines in respect to the following:	Present-day machines appear to surpass humans in respect to the following:
1. Ability to detect a small amount of visual or acoustic energy	1. Ability to respond quickly to control signals and to apply great force smoothly and precisely
2. Ability to perceive patterns of light or sound	2. Ability to perform repetitive, routine tasks
3. Ability to improvise and use flexible procedures	3. Ability to store information briefly and then to erase it completely
4. Ability to store very large amounts of information for long periods and to recall relevant facts at the appropriate time	4. Ability to reason deductively, including computational ability
5. Ability to reason inductively	5. Ability to handle highly complex operations, i.e. to do many different things at once.
6. Ability to exercise judgment	

Note: (Fitts et al., 1951, p. 10)

Bainbridge (1983) criticized functional allocation and argued for a more thoughtful approach that considers the strengths and limitations of humans and machines since the automation of industrial processes increases problems for the human operators. She asserted that automation does not eliminate the need for skilled operators. Rather, operators are left with much more complicated problems when the automation fails. When a manual takeover is required to repair or analyze an automated system, the operator must be highly skilled. Operators of automatic processes make decisions based on adequate knowledge of processes, previous decision experiences, and feedback on their previous decisions, according to Bainbridge. However, over time, operators who monitor automated systems lose the daily experiences of problem-solving that are so crucial to decision-making, as “efficient retrieval of knowledge from long-term memory depends on frequency of use” (Bainbridge, 1983, p. 775).

Literature Related to Decision-making

Complexity and mental models

Technical systems have become increasingly complex and more difficult to manage in the last decade, according to Shorrock (2022), as new technology and automation are mixed with legacy system elements. This complexity makes it difficult for engineers “to maintain accurate mental models even of their own technical systems, let alone how they may interact with other systems” (Shorrock, 2022, para. 9). Consistent with Shorrock (2022), al (2017) argued that as systems become more complex, any single person’s mental model or understanding will be reduced.

According to Rasmussen (1983), mental models are a “causal structure to fit the specific task” (p. 261). Similarly, Sougne et al. (1993) defined a mental model as “a structure analogous to the world which allows the possibility of testing its veracity” (p. 320). According to Woods (2017), “when a technical system surprises us, it is most often because our mental models of that system are flawed” (p. 12). Lim and Klein (2006) found that accurate team mental models were developed by organizing knowledge of team tasks, equipment, roles, goals, and abilities and were positively related to team performance. The authors recommended that future research should examine the mechanisms by which team mental models affect performance and explore the potential moderating effects of other variables, such as team size and team diversity. The research of Shorrock (2022), Woods (2017), and Lim and Klein (2006) provide a compelling narrative illustrating the need for more research on how complexity and mental models affect the work of engineers.

Temporal reasoning and mental models

In another definition, Sougné et al. (1993) asserted that mental models are based on perception, memory, and knowledge of the world, which allows us to simulate the possible outcomes and draw inferences from those simulations. When deciding to deploy a change, are engineers imagining how different scenarios might play out in a kind of mental time travel or episodic foresight as Suddendorf et al. (2018) claimed? Do people have temporal strategies to anticipate

events and decide when to act or intervene? The search for answers (or better questions) drew me to research from De Keyser (1990a) who argued that technological development and market pressure led to situations that took on temporal characteristics that are increasingly difficult for the operator to control, resulting in errors that appear as the product of a mismatch between the characteristics of the situation and the operator's resources (p. 121). Although the research questions centered on the lived experience of engineers making decisions, research on mental models and temporal reasoning offers insight into another dimension of the problem that drives the research question.

Citing Block's (1990) aspects of psychological time, Sougne et al. (1993) argued that people come to conclusions by inferring cause and effect about sequential events, the duration of events, as well as conceptions of the past, present, or future (p. 313). Sougne et al. (1993) used the work of anesthesiologists to explain how various mental models and temporal aspects of past, present, and future were employed by anesthesiologists who had to balance temporal, clock time, and hospital schedules to achieve a goal.

Time pressure

Because engineers are often under extreme pressure to make decisions quickly, a brief examination of the effect of time pressure on decision-making is warranted in the literature review. Khoo and Mosier (2008) studied the decision-making processes of pilots who were confronted with conflicting information and found that pilots who experienced time pressure missed important diagnostic cues because they accelerated and filtered their information search.

The combination of severe time pressure and the need to make a critical decision may lead systems to "fail spectacularly at some point, with attendant human and social costs of great severity" (Rochlin et al., 1987, p. 1). In the Columbia space shuttle disaster, time pressure was identified as a key contributing factor where "ongoing time pressure and associated time stress, as well as the time-urgent culture that ultimately emerged, sowed the seeds of disaster" (Starbuck & Farjoun, 2009, p. 7). According to Ordóñez et al. (2015), people make decisions under time

pressure by either switching to a simpler strategy, keeping the same strategy if the cognitive effort to switch would be high, or relying on old habits.

Risk and uncertainty

According to Slovic et al. (2005), studies have indicated that combining intuitive and logical thinking when analyzing risk leads to the most effective decision-making outcomes. Uncertainty about a situation can increase the perception of risk. Some precautions that people take to handle uncertainty include assuming that the future is like the past; following rules, norms, and conventions to eliminate some of the worst potential outcomes; adding in buffers and redundancies; accepting trial and error; undertaking routine maintenance; engaging regulatory institutions; and considering alternatives (Kabir et al., 2021).

Woods (2017) also investigated the role of risk and uncertainty in how decisions are made. He asserted that when uncertainty is accompanied by escalating consequences, it can lead to a pressure cooker situation and argued that uncertainty is closely linked to surprise (p. 16). In Woods's study, participants reported that the risk of taking action was judged much greater when the cause of the disruption was unknown or speculative (p. 22).

Strategies for decision-making in adverse conditions

Engineers often make deployment decisions in challenging situations, for example responding to incidents that require deploying a change to stabilize the system, but how do they know the change is safe? Morel et al. (2008) researched decision-making in sea-fishing skippers and identified resilience as the ability to recognize and make safe decisions in adverse conditions. The authors noted three strategies that participants used to make decisions:

1. Imagine the situation before it happens
2. Adapt to the situation to find a solution
3. Manage the outcome.

Although Morel et al. (2008) focused on sea-fishing skippers and found them to be independent decision-makers, the strategies they identified are applicable to this study. In my experience, engineers in software and web operations may roll back a software release, which is an example of Morel et al.'s (2008) strategy of adapting to the situation.

Software deployment research

In recent years, there has been increased interest in applying safety science to the field of software and web operations. Three studies by Allspaw (2015), Grayson (2018), and Maguire (2020) all used process tracing to investigate the cognitive work of engineers tasked with resolving incidents and outages. Process tracing uses multiple data sources such as logs and chat records, in addition to verbal reports about what the person experienced and how they interpreted those experiences (Woods, 1993).

Allspaw (2015) used process tracing in his case study of a business-critical internet service outage to identify the heuristics used by engineers to resolve the outage. He found that engineers used three diagnostic heuristics: (a) initially they looked for a correlation with recent software changes, (b) if none were found, they expanded their search for other possible factors, and (c) they focused on the easiest diagnosis that aligned with similar symptoms from past incidents. Additionally, the study found that when engineers had to make a change to resolve the issue, they prioritized peer review over automated testing.

Grayson (2018) used process tracing methods to study how engineers handle unexpected events. She identified occurrences of abnormalities that might overwhelm the system's ability to withstand continuous demand, and how human and machine agents worked together to resolve the problem. She found that it was difficult for the responders to assess problems and take corrective action due to unseen processes where automation was actively affecting the system. It was exciting to see Grayson's use of the Above the Line/Below the Line (ABL) framework (Cook, 2020) in the study. She suggested that the framework would be useful in creating decision support tools to make detecting and resolving incidents take less time.

Maguire (2020) studied the costs of coordination in large-scale distributed systems using process tracing as a methodology to understand how software engineers coped with outages. She identified that the strategies used by engineers were flexible and depended on being able to predict and change the approach based on what others were doing in real time. For example, if an incident commander is making it hard to get things done, people will find ways to get around the high cost of coordinating with just a single person.

I scoured existing literature on safety science in the field of software and web operations to understand what is currently known about roll backs as well as to establish the existence of a gap in the literature. In my experience, the need to revert a software or configuration change is often done under time pressure during an outage. The degrees of reversibility in software systems may also impact decision-making by engineers. Software deployment systems are often designed with some degree of reversibility built in so that a change can be quickly reverted if a problem arises. However, interactions with other changes might mean that reverting the change is not possible (Adams & McIntosh, 2016; Schermann et al., 2016). This specific challenge of unpredictable roll backs as a result of deployment decision-making, and the need to revert to the previous version of the software is not well represented in existing decision-making theories nor in the literature. It is unclear how having the ability to roll back influences the decision-making process, especially if reversibility is overestimated. There is a lot of complexity in roll backs: the decision to roll back, the awareness of the need to roll back, familiarity with the tools of rolling back, specific consequences of rolling back or rolling forward, and whether the functionality is even available.

The studies that do exist about software deployment decision-making are not always built on sound research methodologies. Kerzazi and Adams (2016) studied the factors that led to botched releases of web-based software. They only examined the releases from one company, and they described the role of a release engineer in the deployment process, but not all teams have a release engineer. The authors built explanatory models without citing any research to validate their

claims. They found that the size of the change, the instability of dependencies, the past performance of the developer making the change, and the higher number of people working on a particular area of a code base led to more botched releases (Kerzazi & Adams, 2016).

Generalizing their findings, they employed blameful language: “Release engineers often get burned by unreliable developers” (Kerzazi & Adams, 2016, p. 578). They invented out of whole cloth a measure of trustworthiness as “delivering solid code that will not backfire in production” (Kerzazi & Adams, 2016, p. 581). Defining trust as “the number of times a team has broken production before (lemons),” such code is “more likely to trigger a new incident” (Kerzazi & Adams, 2016, p. 578). The measure of trustworthiness that Kerzazi and Adams (2016) developed was based on a subjective statement from one release manager who said that developers are given a “fresh slate of Karma . . . reduced with each major error” (p. 578) and that code from such developers cannot be deployed because they “lack enough trust by the release manager” (p. 578). The methodological weaknesses in this study provide further justification for the need for additional research on the role of roll backs in deployment decision-making.

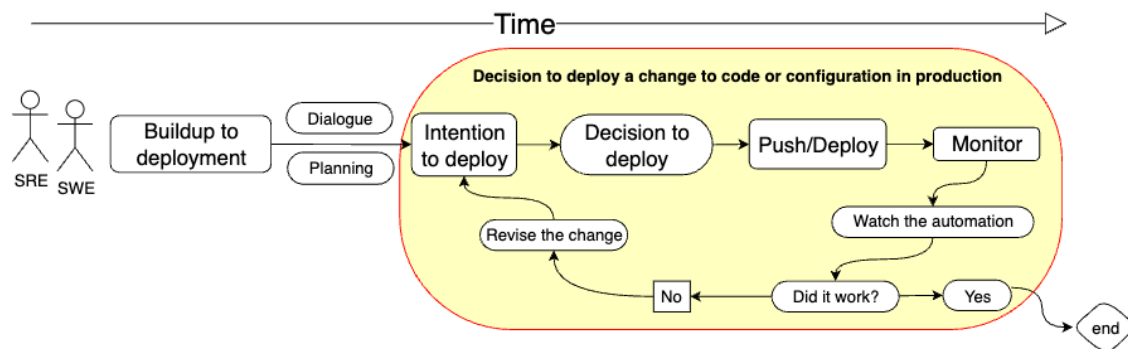
This brief literature review examined several decision-making theories that influence the decision-making process. Research has indicated that factors such as complexity, time pressure, risk, and uncertainty have been shown to influence decision-making; however, there remains a lack of research in the fields of software engineering, web operations, and critical digital services on how engineers experience deployment decision-making. The next chapter describes the methodology employed in this study, including how participants were selected, how the data was collected through interviews, and how I analyzed the data to arrive at themes. Discussion and implications are offered in the remaining two chapters.

Methodology

This chapter details the qualitative methodology used to investigate the research questions that guided the study. I was inspired by phenomenology because I was interested in the lived experiences of engineers whose deployment decisions are often judged harshly if the outcome is poor. The data analysis process is also described in this chapter. For this study, I interviewed 15 participants from a variety of companies to understand their lived experiences deploying code and configuration changes. I carefully analyzed participants' accounts to describe the essential components of the experience of deploying a change to software or configuration in production (live systems). Figure 2 describes a simplified deployment decision process in which SRE stands for site reliability engineer and SWE stands for software engineer.

Figure 2

System of Interest: Deployment Decision-Making



Participants

Participants for this study were recruited on Twitter and LinkedIn via public posts (see Appendix A: Recruitment). The sample size was based on the goal of interviewing enough participants to understand their experiences within the scope of available time for research, the availability of participants (convenience sampling), and the participants having experience with the phenomena being studied (criterion sampling) (Creswell, 2006). A total of 18 engineers responded to the request for participation, three withdrew before scheduling, and 15 people met the participation criteria (see Appendices B and C), completed the consent document (see Appendix D), and agreed to be interviewed.

Demographics

Participants were from different countries including the United States, Germany, Norway, The Netherlands, Portugal, and Sweden; were at least 18 years of age; had recent deployment decision-making experience; and were available to be interviewed in the time allotted for the study. Table 3 provides more information about the demographics and participant roles.

Table 3

Participant Demographics

Attribute	Description
Gender	3 women, 12 men
Industry	DevOps, Cloud, Internet of Things (IOT), entertainment, travel, retail, telecommunications, IT services/consulting
Role	Software engineer (9), SRE (3), Operations engineer (1), Director of SRE (1), Engineering manager (1)
Education (8 of 15 responded to this question)	<ul style="list-style-type: none">• CS Bachelors degree• Bachelors in Electronics• 1 year accelerated degree• BS Computer Information Systems, self-taught on systems/networks, etc. on the job.• Self-taught, a few courses at university• M. Sc. Eng. + autodidact• Unrelated university degree• University degree (MSc)

Interviews

I conducted the interviews in English via Zoom video conference between December 2020 and February 2021. The interviews were between 60 and 90 minutes. I used a semi-structured interview format inspired by Klein et al., (1989) critical decision method that explores how experts make decisions by asking about the knowledge, cues, and strategies they used (Klein et al., 1989). Participants were asked to describe their experience with a recent deployment of code or configuration into a production environment. The questions were designed to prompt participants to describe what they experienced and how they experienced it. These questions are

documented in Table 4. After the participants shared their experiences, I asked follow-up questions.

Table 4
Semi-structured Interview Questions

Cue type	Question
Experience	Describe the deployment decision event or situation you experienced
Experience	Describe the outcome of the deployment.
Influences	What aspects of your experience were difficult or frustrating?
Cues	What were you seeing, hearing...?
Knowledge	What information did you use to make a decision?
Analogues	Were you reminded of any previous experience?
Goals	What were your goals?
Options	Were there any other options available to you?
Basis	How was the option selected? What options were rejected?
Experience	What prior training or experience was necessary?
Aiding	Hypothetical: What information would have helped?
Time Pressure	How much time pressure did you experience?

Note: Adapted from Klein et al., 1989, p. 466.

Data Analysis/Explicitation

I was inspired by phenomenological explicitation methods and thematic analysis for the data analysis process (Blaxter et al., 2006; Braun et al., 2015; Creswell, 2006; Hycner, 1985; Saldana, 2015). Hycner (1985) warns that the use of the term data analysis implies a disassembling of the data, whereas explicitation permits the researcher to investigate various elements of the topic and avoid losing the whole phenomenon (p. 300). The explicitation method requires five steps, according to Blaxter et al. (2006): bracketing, identifying meaning units, clustering meaning units into themes, summarizing interviews, verifying accuracy with participants, and crafting general descriptions.

According to Creswell (2007), bracketing is a “process of setting aside one’s beliefs, feelings, and perceptions to be more open and faithful to the phenomenon” (p. 269). Transcripts from the interviews were generated and corrected word-for-word using the transcription service

Descript.com. In qualitative research, the researcher must privilege their participants’ experiences

and use their words. To interpret the data, I divided each transcript into distinct statements, described by Hycner (1985) as general meaning units, without discarding anything. I further simplified the text into condensed meaning units, from which I identified relevant meaning units as significant statements that offered insight into the research question (Creswell, 2006; Saldana, 2015). From the 15 verbatim participant transcripts, 364 significant statements were identified. The significant statements were imported into Google Sheets, after which I used In Vivo coding to produce descriptive labels for each statement. Because I am a novice researcher, I sent the coding from one participant to my mentor, Richard Cook,¹ for advice. His response was that “unlike most interviews, your data seems to be heavily dependent on sequences and temporal reasoning about cause and effect, either about the past or what will happen in the future” (R. Cook, personal communication, May 21, 2021). He suggested the following coding structure, which I followed:

Statement describing some sequential process, e.g., x happened because of y that preceded it

Statement explicitly referring to duration, e.g., it takes so long for the CI/CD pipeline to complete that...

Statement about a distant past decision, e.g., they decided a long time ago to use kafka as the message broker

Statement about the nature of time as a pressure, esp. as it forces a sacrifice decision, e.g., we decided to restart everything to be ready to open tomorrow

¹ <https://www.adaptivecapacitylabs.com/blog/2022/09/12/richard-cook-a-life-in-many-acts>

Statement that does not refer to time, e.g., a description that is treated as true over time such as we always have trouble getting the right people to review the code.

Identify explicit call outs to uncertainty or risk

I then sorted the data in Google Sheets by descriptive label to group the significant statements into clusters of related meaning. I drew upon thematic analysis to recognize, examine, and make sense of patterns of significance (referred to as “themes”) in the qualitative data (Braun et al., 2015). Table 5 provides a sample of the data analysis process I followed.

Table 5
Data Analysis Process

Condensed Meaning Unit	Descriptive Label	Category	Temporal Reasoning	Uncertainty/Risk
Staging does not have the same database as production	Different database version	Test vs. Prod	Past decision	Risk
Should have taken a couple hours, it took a full day	Took all day	Mental model	Duration	n/a
Hard when everybody is working from home. Don't know if the expert is available. In the office, it's easy to say "hey, can you look at this?" but it all gets lost in work from home context.	Work from home	Communication		Uncertainty
In a rush to get my work complete so I could take time off over the holidays.	In a rush	Holidays	Time as a pressure	n/a

Research Ethics

After reviewing the Swedish Ethics Review Authority website (2018) and the Lund University (2022) research ethics website, I did not find a requirement to obtain ethical permission for my

research. However, observing and adhering to ethical principles is still imperative in any research study involving human subjects. Standard ethical principles in research included informed consent, voluntary participation, anonymity, confidentiality, and minimal risk (Barrow et al., 2022).

All participants received a participation request letter that specified the intent and procedures of the study to ensure informed consent. In this letter, participants were also notified of their right to withdraw from the study's procedures at any time, without giving a reason, and without repercussions or impact on their employment status. The participants were assigned a pseudonym and informed that their identity and personal information would be kept anonymous. Consent to participate was recorded via DocuSign. The study had a minimal risk for participants as there was no concern for discomfort or harm as part of participation.

Before conducting the data collection, I endeavored to bracket my preconceptions and experience with the phenomena as suggested by Creswell (2007), who also advocated for clarifying professional connections of the researcher to the research topic. Because of my work analyzing incidents as well as my experiences with software developers and site reliability engineers, I am familiar with the field of software engineering, although I have not worked directly in those roles. My familiarity with the field made it important for me to bracket my personal biases through journaling my personal assumptions and biases in the form of notes and in-text coding. Through thinking about my own experiences and background as well as my assumptions regarding the participants, I sought to avoid interference with the research. During the interviews, when I realized my biases, I acknowledged to the participants that I had a bias and then consciously refocused the interview back to their experience. In most cases, the participants' employers were unknown to me. However, I did become aware that two of the participants in the study were employed by the same company I was working for at the time of the interviews, but I had not interacted with them in any capacity prior to the interviews. I took steps to ensure that their identities and participation in the study would be kept confidential and

that their decision to participate or not would in no way affect their employment. I became aware that one participant worked in a company I had previously worked for several years before the interview, although we did not know one another or interact during that time or subsequently. One participant was a member of a community “Learning From Incidents in Software,” of which I am an administrator, but we had not interacted to my knowledge.

I had several assumptions about what I thought the participants in this study may be experiencing. Due to the holiday time period around Thanksgiving, Christmas, and New Year’s Day, my first assumption was that the participants would be facing unique constraints and challenges in deploying changes that might limit or even prevent deployments during this time. My second assumption was that the participants may not have adequate safety mechanisms and tooling to observe how their changes behave in production.

Limitations

Because data is subjectively evaluated by the researcher, it can be difficult to establish the reliability, validity, and generalizability of the findings, especially when the sample sizes are small. Offredy and Vickers (2010) argued that data relies on participants being able to communicate their experiences, which requires them to be both interested and articulate (p. 103). They noted that participants may have challenges with expressing themselves, including language barriers, age, and embarrassment in describing personal experiences.

Qualitative research on experiences depends on participant memories, which can be unreliable. As Kordes (2009) argued, at any moment, direct living experiencing can at best be observed, but hardly researched; therefore, the research of experience is the research of memories of past experiences (p. 68). Descriptions of any aspect of a participant’s life will always remain more complex than even the most thorough explicitation can reveal (Hultgren, 1990, p. 18).

Validity

Quality refers to the thoroughness and relevance of the data as it relates to informing the research question, whereas validity describes the trustworthiness and accuracy of the data, such as how accurately and truthfully it reflects participants' lived experiences (Polkinghorne, 1989). Guided by Polkinghorne's (1989) five questions for confirming the quality and validity of data (p. 57), I listened to audio recordings of the transcripts multiple times as a review process to verify that my explication accurately reflected what the participants communicated. Since I relied on explication to guide my data analysis, I provided each participant with a copy of the transcript and their participant narrative, which were revised upon request to ensure the accuracy of the data.

Generalizability

The generalizability of the findings requires consideration of the context, participants, and their experiences. This study was limited by the number of participants' organizational contexts, which makes it difficult to generalize the results. This chapter details the methodology used in the study, and the following chapter reports the findings.

Findings

This chapter details the findings of the study, in which I sought to understand how software engineers decide to deploy code and configuration changes in production environments. The aim of the analysis was to answer the following research questions and sub-questions:

1. What are the deployment decision-making experiences of engineers in the field of software and web operations making changes to technical systems?
2. How do they experience the decision to deploy those changes?
 - a. How were engineers' experiences difficult or frustrating?
 - b. How was time pressure a factor in engineers' deployment decision-making experiences?
 - c. How did engineers derive confidence in the decision to deploy?
 - d. How did engineers' previous work experiences factor into their decisions to deploy the change?

Within the time allotted for this study, I chose to focus on the four themes and 15 related sub-themes, here called clusters, that were most salient for answering the research questions (see Table 6):

Table 6
Themes and Clusters

Theme	Cluster
Deployment Decision-Making is Risky	Risk Mitigation Strategies The Deployment Experience Surprise! Things Are Broken Now Managing the Outcome of Deployment Decisions
Temporal Reasoning Influences Deployment Decision-Making	Time as a Pressure Time as Cause Duration of the Deployment Experience
Relationships Influence Deployment Decision-Making	Relationships With Self Relationships With Family and Friends Relationships With Co-Workers Relationships With Management
What Would Engineers Change?	Testing Feature Flags Post-deployment Validation Improved Metrics

Changes to code and configuration were deployed for various systems. Two participants chose not to deploy a change. One participant deployed a change in canary (a type of testing environment) but intended to delay the final stage of deployment until after the holidays (P7). Four participants described deploying a change in response to an unplanned incident or outage. More than half of the participants deployed a planned change such as a new feature, an upgrade, a database schema change, or a change to the configuration of SSL certificates. Five participants said that their planned change resulted in an incident or outage, and two participants described how their planned change resulted in a near miss. In Table 7, each participant’s deployment context is described along with the reasons for making their decision.

Table 7
Participant Deployment Context

P#	System state	Type of change	Deploy	Reason for decision
P1	Stable	Reduce startup time	No	“Instinct and intuition”
P2	Stable	Debug infrequent problem	Yes	Low risk, debug infrequent problem
P3	Unstable	Rollback	Yes	Emergency
P4	Unstable	Fix missing column	Yes	Feature missing, hotfix required
P5	Unstable	Fix payment processor	Yes	Critical payment gateway down
P6	Stable	New feature	Yes	Deliver feature before code freeze
P7	Stable	Change SSL configuration	No	Deploy SSL change to Canary
P8	Stable	Load test	Yes	Talked to people involved, run tests
P9	Unstable	Emergency change to avoid data loss	Yes	Emergency incident, data loss risk
P10	Stable	New database feature	Yes	Tests passed
P11	Unstable	Emergency fix, CosmosDB out of space	Yes	Made a backup before proceeding
P12	Unstable	Change failover configuration	Yes	Required change to config, pairing/peer review
P13	Unstable	Upgrade failed, restore database	Yes	Test passed
P14	Stable	Planned change to database	Yes	Worked in staging, a “simple change”
P15	Unstable	Emergency fix SSL cert issue	Yes	Old cert revoked, new cert had to be issued

As shown in Figure 3, participants reported using various tools to deploy including Ansible, SSH, Concourse, Spinnaker, GitHub, Kubernetes, Azure DevOps, and internally developed continuous integration/continuous delivery (CI/CD) tools. Monitoring tools used by participants included DataDog, Prometheus, and Grafana. Participants reported deploying to targets hosted by Azure, AWS, and Google Cloud.

Figure 3
Tools Used by Participants



Theme 1: Deployment Decision-Making is Risky

All participants reported feeling concerned about risks associated with their decisions to deploy a change and worried about making mistakes. Code freezes, which are intended to reduce risk, can be interpreted differently by different teams, who may continue to prepare changes to code or configuration without deploying. Where deployments were highly automated, participants reported being able to deploy changes very quickly. However, automation brought significant challenges, failures, and cascading issues. Risk and uncertainty were especially profound for those working in critical digital services where deployment decisions had the potential to harm human life: “Your work affects people’s lives. There are consequences. I think I’ve killed far more people working in software than I did working in medicine” (P7). I identified four clusters of meaning within this theme: Risk mitigation strategies, The deployment experience, Surprise! Things are broken now, and, finally, Managing the outcome of the deployment (see Table 8).

Table 8*Theme 1 Clusters and Examples*

Cluster	Examples
Risk mitigation strategies	“Be really surgical . . . because things might go wrong.” (P5)
The deployment experience	“We have an automated canary timer. If no one pushes the stop button and the canaries don’t blow up catastrophically, the canary cluster automatically promotes the deploy 30 minutes.” (P10)
Surprise! Things are broken now	“We thought errors would have been caught by the test environment but it didn’t. My colleague looked at the recent updates to the library and noticed that the connection was bad. We had messages in our logs as well.” (P3)
Managing the outcome of deployment decisions	“It is a very nerve-racking and fraught experience to be asked to come to a meeting with the directors and explain what happened and why your product broke.” (P8)

Cluster 1: Risk mitigation strategies

Companies often enact holiday code freezes, which limit or block deployments, in an effort to reduce risk of problems since there are usually fewer people available to respond. Participants sought to mitigate risks and employed various strategies to prevent their own or others’ errors. Testing, documentation, backups, checklists, runbooks, discussing changes before deploying, and coordinating deployment actions with colleagues, all contributed to their confidence (or lack thereof) in their decision to deploy.

Be careful. Many participants spoke about the need to be careful, “really surgical . . . because things might go wrong” (P5). One engineer described how they “spent a lot of time reviewing today’s planned deployments” and relied on “phased rollouts” (P7), whereas another mentioned strict change management procedures for “big changes” (P10).

Testing. Testing is a strategy to mitigate the risk of deployment decisions and is an essential aspect of how confident an engineer is about proceeding with a deployment, as P4 explained:

Things that give me the most confidence happen earliest in the process, like peer review where someone is actively looking at [the change]. . . . There’s a perverse incentive that the more you test, the longer the tests take to run . . . there’s a

quantifiable cost to building confidence through testing. Computers can run tests, but inevitably problems are found by humans, so I'm trying to figure out the balance.

Automated vs. manual testing. Some participants expressed a preference for automated tests over manual testing. P6 said that automated tests are “part of the value I bring to the team.” On the contrary, several engineers felt they could not trust automated tests because of uncertainty about how their change would interact with other changes.

Test environments. Test environments are another strategy for building confidence before deploying to production. There are often multiple pre-production environments that are progressively more like production. If tests pass in these environments, it should increase confidence in the deployment, as P4 explained:

We are testing in production. QA is not production, staging is not production, production is production. Confidence you can build on the way to production is only as good as the fidelity of your non-production environment and how close to parity they are, both in construction and in behavior.

Participants made inferences about the various test environments and noted the problems they encountered when trying to use them. P6 shared an experience of how challenging it can be to make the test environments behave like production:

I relied on integration tests to prove the correctness of what I was pushing out and had not really done rigorous testing with a copy of the real data locally. . . . Honestly, it's a laziness thing, sometimes it can be difficult to replicate things sufficiently to manually test it. The easier thing to do is to push out to a test environment where all the infrastructure is there . . . and hotfix repeatedly until you get something good. Configurations and interdependencies between applications can be complex . . . It's onerous, *especially* if it starts getting into the

point of configuring security certificates locally. . . . If I can't make two systems locally talk to each other because they're trying to use SSL and I haven't gotten the certificate set up, I'm going to give up and just go to a deployed environment where all that stuff is already handled for me.

Using test environments during a code freeze. One engineer felt that they could “continue to deploy to their test and staging environments” but warned about a “side effect of a lot of things in a staging environment waiting to go out. These can compound the risk of the deployments” (P6).

Alerts. Alerts are a risk mitigation strategy for engineers. Alerts are designed in advance for particular failure modes that automatically create a “ticket” with a severity level pre-assigned.

Believing the change is low risk. Participants proceeded to deploy when sufficient mitigation strategies were in place, believing the change was low risk. P4, P14, and P2 enacted changes that they thought were low risk. P14 described “a simple change,” whereas P2 thought, “What could possibly go wrong? It's fascinating because I thought it couldn't fail.”

Cluster 2: The deployment experience

With strategies in place to mitigate risk and uncertainty, participants made the decision to deploy or not to deploy. Most participants described having the capability to delay the final deployment to production by first deploying to test environments. These environments are meant to resemble production; staging (like production but standalone), integration (like production but meets other changes), and canary (change is in production, but only sees a small fraction of production traffic). P10 explained, “We have an automated canary timer. If no one pushes the stop button and the canaries don't blow up catastrophically, the canary cluster automatically promotes the deploy 30 minutes.”

Documenting the change. Participants documented their changes in “git commits” (P12), a “Jira ticket” (P14), and other systems of record. Even if not mandated by the company,

participants often documented their own actions in the system by writing up descriptions of change, problem, scope, and impact (P10).

Announcing the change. Participants described telling co-workers of their intention to proceed with a change:

I tell them what component I'm changing, just adding a record to the database. I announce as I start, and after each region is complete (so people can test it), "sanity check" (P14)

Automated deployments: It's complicated. Changes to the code occur constantly in all parts of the code base and may depend on each other for proper functionality. The risk of these changes interacting in surprising ways or being deployed in the wrong order was a significant concern for engineers. Some deployment pipelines are highly automated, requiring no human interaction whereas others are highly automated but require human decision-makers to approve each step, and the person making the decision at each step may not be the person who initially made the decision to deploy. Other deployments are very manual or only partially automated. One engineer recalled the experience of deploying a "custom script that ran the automation of everyone else's deployment pipelines and then I had to monitor all of it to make sure the fix went out." (P3). Automating the deployments can be useful in some contexts, as P2 explained:

The deployment is very automated. If we change a tag on the container image, it's deployed to production within minutes. We have a git-ops model and it's the first really nice thing that's happened on the deployment front for many years. I'm very happy with that system, we've been using it for about a year.

However, automating the deployments can also create new risks for engineers. One engineer described how deployment automation can go wrong:

Sometimes it's just a risk management thing. I could do two builds of each project, release them both into integration simultaneously, and see what happens

if both of them are deploying at the same time, but even if that's successful, I don't trust [it]. You're describing a giant Rube Goldberg of a race condition. Even if it was successful in integration, staging, and canary and everything looked fine, I can't promise that when [it] goes across an entire production fleet that I won't get the timing wrong. . . . I don't think it is worth the effort to manage this with automation. . . . It feels like a technical answer to a human problem. I've seen folks [using] Jenkins [automated deployment tool] try to put locks in where when one job is running, it locks a given set of other jobs so that those jobs can't run until the end. That's cute, but how often does it blow up in your face? How often does one of them fail to release the lock? Now everybody's stuck. Now you got a different problem. Congratulations! Welcome to your new issue! If the change is significant enough to be worried about, there should be a human shepherding it. The whole point of organizing people should be to make sure that the two people who are about to poke the same system know each other's names.

Just talk to each other, it's not that hard. (P7)

Seeing the Change. Most participants reported the importance of being able to observe their changes as they proceeded through the deployment process and their use of various visual displays and dashboards to understand the state of systems. P14 reported, “I babysit the AWS [Amazon Web Services] console, watch containers go down and come up.”

Many participants reported the difficulties they faced if these visual displays were unavailable, inaccurate, or misleading. Some tools used by participants offered no way to validate that a particular change was actually applied. One engineer recounted how they used a configuration management tool to configure the database failover settings but could not see any confirmation, so if they “second-guessed themselves, they just had to do it again” (P12). This lack of visibility was particularly difficult for one engineer, who had to approve dozens of A/B tests (two versions

are deployed so performance can be measured and compared) daily but noted that these were “dangerous because they couldn’t see what was changing” (P8).

Cluster 3: Surprise! Things are broken now.

Participants described working closely with colleagues to respond to deployment failures by evaluating alerts and severity levels, deciding what actions to take, evaluating the effects of the actions, providing ongoing documentation of actions taken, and regularly communicating progress to users.

Surprised by alert behavior. Participants reported relying on alerts but were often surprised by how they failed to perform as expected due to their absence, going unnoticed, or conflict between the design of the alert and the automatic remediation for the alert itself. P11 described this conflict:

There was an alert for a rising exception count, but it was set in a very confusing manner. We had protections in place, if something is exceeding 40 exceptions, notify someone. But we also had a system which was saying, if you're excepting, try after 30 seconds. So, two competing forces acting in different directions.

These are good practices to backoff, but it didn't help in this context.

Surprised by code linting tool behavior. P5 recounted supervising a junior engineer who was fixing old code. They were using an IDE [integrated development environment] called VS Code, which checks the syntax for correctness, but VS Code did not detect the problem:

VSCode tells you “Okay, it’s ESX, it's fine, go ahead.” But obviously, Grunt doesn't know anything about it, so it complains. We changed that back to the early 2010s style of JavaScript and re-ran the process. (P5)

Surprised by code freeze failure. P7 recalled an experience in which their mental model of the complexity of time was represented by SSL certificates. P7, who had a mental model of time about code freezes during the holidays, believed that their change would stay in staging during

the code freeze. P7 was in the process of renewing a certificate but believed that they had more time to renew it. However, they were shocked to realize that a change took effect despite their efforts to wait until after the holidays:

It's right before Christmas. Stopping. We're done. We're going to let this bake in canary for two weeks. Because why not? There's no harm. We're not deploying during that time. We'll come back in January, and assuming canary looks good, we'll resume. (P7)

Pandemic challenges. One participant described the stress of responding to an incident while balancing childcare responsibilities and working from home during the pandemic:

It's hard when everybody is working from home. . . . When we're in the office, it's easy to say "Hey, can you look at this?" But it all gets lost in the work-from-home context I was on [a work] call, in the parking lot of the school. In non-pandemic times, I would have had the whole day to just worry about work. (P3)

Surprised by past configuration decision. For example, one engineer recalled how a misunderstanding of documentation on database partitioning led to an incident almost 3 years later when the database exceeded the partition limit: "It was a system limitation, probably very deep in the documentation, nobody bothered to read it" (P11).

Try to figure out what is wrong. Participants described their efforts to understand the nature of the failure: "We thought errors would have been caught by the test environment but it didn't. My colleague looked at the recent updates to the library and noticed that the connection was bad. We had messages in our logs as well" (P3).

Assign a severity level. If the engineer is not alerted and just notices a problem, they may assign the severity level. They also might change the preassigned severity level to match the present circumstances. P9 explained, "If we're not sure about what's going on, we raise what we call a

WTF, which is like a SEV1 [severity level 1], something that has a high chance of customer impact.”

Communicate progress. Participants described the tension and communication challenges that can occur when responding to an incident:

In the heat of the moment, we’re not very good at telling everyone that something is wrong. . . . I feel like we get too deep in the technical “let’s fix this problem” and we don’t look back into the people aspect or let them know, “Hey your service might have failed around a certain time.” It’s not great. (P3)

Participants also described the expectation of regular public communication with external customers during an incident. P11 recalled:

I wondered what we were going to explain to our customers. Because if I write a message, I'm going to say that CosmosDB failed but the lead said, “That's not what they want to hear.” I knew if I go into handling communications that I cannot focus on fixing the system I would write more technical stuff than necessary. . . . They’re not going to understand: “Your system will be eventually consistent.” I told [the lead] what happened and to manage communications . . . because I have bigger fish to fry. If [the lead] handles communications, I’ll be able to fix the technical problem. If [they] want me to do both, I’m probably going to mess it up.

The role of documentation in incident response. P9 described working closely with a colleague on a critical incident during the holidays, the role of documentation, and the uncertainty of how each command might affect the system:

When we realized what the problem was, and we were looking at a number of different ways to possibly fix it, which none of us were entirely sure about. All of it started to sound quite fraught with potential pitfalls. The more of these steps

that we thought that we'd have to do, the less excited we were... The whole service will be down while it happens. So that was a late night... it involved a lot more point-by-point communications with my counterpart. For example, "I'm going to try this, do you agree this is a good thing? What are we going to type in?". . . . There's that moment right before, a sense of "I *think* this is going to do what I want, but I'm not 100% sure because I've never done this particular thing in production." The documentation says it should work; Amazon support pages say it should work. So, we tried it, and the command had an error, but it still seemed to work. So, who do I trust? You don't really know what's going to happen until you've done it.

Can we roll back? After a deployment failure, one action that can be taken is to roll back the change by downgrading to a previously known good version. Several participants stated that the ability to roll back a deployment was a critically important capability in their ability to remediate issues. A roll back on one part of the system might trigger a roll back on other dependent services as well. One engineer said that once they understood the problem, rolling back is the first action they take:

The first thing is to just roll it back to get it to a serviceable state... the roll back part was automatic for us because we have the proper safety practices in place.

The other option was rolling forward but our pipeline takes a long time and has a lot of gates that are validated only in test. (P3)

On the other hand, rolling back can sometimes make the problem worse:

There can be cases where rolling back doesn't work. With a distributed system, you never know. If it fails on some machines, it can get into a weird state. For a graph database, I have to be very careful about not rolling back because it can result in data inconsistencies. (P8)

One engineer described their belief about the functions and capabilities of the roll back tooling:

It had also not rolled back the addition of that column, which I believe it's supposed to happen automatically, but obviously, that's a misunderstanding on my part. (P6)

Just roll forward. Sometimes engineers deploy a new change to address an outage or incident:

“We could have rolled back, but in this case, it was easy so it was just a fix forward instead of a roll back. We made the change, committed it, and 17 minutes later it was fixed” (P2).

P2’s deployment experience captures the essence of deploying a planned change, discovering a problem with the deployed change as they just happened to check on it during a larger, unrelated incident, receiving the alert in a place where it might not be noticed, all because of a previous decision about the automatically assigned severity level if containers are restarting.

Since we had the signaling storm, I wanted to see how my component was performing. Although it was to the side of the problem, it would still be affected.

I looked at the graphs and noticed it wasn’t performing as well as I expected.

Then I noticed the alert in Slack. Low severity alerts go to Slack, but if it’s user impacting it goes to PagerDuty. A container restarting could be bad but if end users are not impacted it will go to Slack. We added this a long time ago when we wanted to get the heads up if things were failing or beginning to deteriorate. In the beginning it was great, now it’s becoming a hassle. (P2)

Buying time. Another engineer described how a particular setting was used to buy time to resolve a serious problem that could have resulted in significant data loss: “This time to live (TTL) thing is only possible because of the retention and the logs, otherwise this data would have vanished” (P11).

Automation surprise. P8 described a complex situation where the automated deployment ensures the versions of code and config are the same, noting that if a manual deployment is

necessary, care must be taken to pull the correct configuration. Manual intervention will still be needed from time to time, so they created a “quick script” to correct any issues with versions of code and config on the servers.

Anybody can make a commit to the code or to the config, so in the time between when you commit your code, the config can change . . . when you promote from canary to production, that config is locked but if one fails and you try to do a manual deploy, you'll actually pull from head rather than the very specific locked to config version . . . [you end up with] config versions that differ across the product. It makes it hard to debug... We typically keep a couple of different versions on the servers so in an emergency when the roll back functionality didn't work properly, I wrote a quick script that logged into all the servers that ran our code, stopped the product, changed the SYM link, then started the product again, to put it back to the previous version.

Going down a rabbit hole. When engineers intervene after an automation surprise, they may become confused about the state of the system, and go down “rabbit holes” (P5, P6). P6 described their experience deploying a change during the holidays:

I copied the staging database and I was looking at the column with those timestamp values in it and expected to be able to refresh it periodically and see new ones jump to the top of the queue... I know there is a situation where it corrects for the databases in UTC [Universal Time Coordinated] and obviously, we're in PT [Pacific Time], which always throws me because I used to live in UTC. So, I was *chasing my tail* [emphasis added] for a while here, thinking that there was a problem with time zones in the database. I spent a lot of time investigating and I thought that was the problem and it turns out that wasn't really what was going on. And I'm really annoyed with myself now There

were a number of steps because I just kept messing things up. There's a whole sequence of pushes I did in order to fix things. (P6)

Cluster 4: Managing the outcome of deployment decisions

After an incident or outage, management and engineers hold postmortems to discuss the incident. Postmortem meetings often lead to decisions about action items (repairs) that are designed to prevent the same problems from happening again in the future. P8 offered a rich description of the experience of postmortems:

The postmortem is actually run by a centralized team and those follow a very strict path and that stuff is fed back to senior management. There are trainings, plans, documents, and forms you're supposed to follow that are meant to create some kind of useful consistency . . . but it really depends on who's running it and who shows up at the meeting on how beneficial these meetings can be. I think John Allspaw said, "The mechanism for understanding whether not your post-mortem works is how interested people are at being there."

Being blamed. When deployment decisions result in incidents and outages, some participants described being blamed for the outcomes. "What's the meantime to innocence? How quickly can you show that it's not a problem with your system?" asked P9. Describing a blameful culture during post-incident review meetings, P8 said

So, if I'm responsible, if I put in a change or I had something that went wrong, and I was responsible to fix it, then I have to go and defend my . . . not defend, but explain what happened. It is a very nerve-wracking and fraught experience to be asked to come to a meeting with the directors and explain what happened and why your product broke.

Blaming themselves. Participants also blamed themselves when their decisions led to incidents. Participants described making "honest mistakes" and that they must avoid repeating mistakes

that impacted customers. Several participants identified themselves as the cause of an incident, as P12 did when describing their experiences in a post-incident review:

We are supposed to follow a blameless process, but like a lot of the time people self-blame. . . . You can't really shut it down that much because frankly there are very causal events. I'm not the only one who in the PIR [post-incident review] can't really let go of [it]. I know it was because of what I did.

Theme 2: Temporal reasoning influences deployment decision-making

Temporal reasoning, or reasoning impacted by the perception of time in terms of the past, present, and future, was frequently referenced as a factor impacting how participants exercised decision-making. Three clusters of meaning were identified within this theme: time as a pressure, time as cause, and duration of the deployment experience (see Table 9).

Table 9

Theme 2 Clusters and Examples

Cluster	Examples
Time as a pressure	“I tried to get ahold of people but being six time zones away, they weren't online yet. It took 30 minutes or so.” (P15)
Time as cause	“I didn't directly connect that what I had done to try to fix the page was what had caused the outage because of a specific symptom I was seeing.” (P12)
Duration of the deployment experience	“The deployment was supposed to take approximately two hours, but instead took an entire day.” (P14)

Cluster 5: Time as a pressure

Participants described experiencing time pressures from management, from customers, and from the engineers themselves, which they felt contributed to the risk associated with their deployment decision. Some were working to deliver a feature that was promised by a certain date. Many participants were constrained by a “code freeze” where, during certain times of the

year, “we can’t make significant production changes that aren’t trivial or that fix something. I didn’t want to be the person who breaks the environment and then goes on vacation” (P6).

Responding to incidents during the holidays was a stressful experience for participants. One engineer described it as “a third level of panic WTF?!!!! Nothing is going through for anyone in the entire [system]!” (P9). Engineers wondered if their changes would trigger new and different problems. Many participants reported feeling a great deal of stress in these situations, particularly at night when they were tired. Although they needed “moral support” (P13), engineers did not want to wake up co-workers in different time zones.

Cluster 6: Time as cause

Past decisions informed current decision-making experiences for participants, who credited previous decisions for preventing problems from occurring and kept the problems that did occur from being worse. Most participants described how outcomes of past decisions directly shaped what they believed would happen when faced with a current deployment decision as well as future decisions. Participants reported feeling that they got “lucky” because of those decisions (P1, P11, P12) and that it “could have been worse” (P4, P11, P12).

Cluster 7: Duration of the deployment experience

Many participants spoke about duration during their deployment decision-making experience. Duration was explained in a variety of ways: predicting the time planned for tasks, estimating the time required to diagnose or determine a problem, and the total time taken to resolve a problem. One engineer said, “The deployment was supposed to take approximately 2 hours, but instead took an entire day.” P11 described how the “problem was fixed in 3 to 4 hours after we decided this is the approach we're going to take.” The need to contact team members in other time zones affected the duration of the decision-making experience: “I tried to get a hold of people but being six time zones away, they weren’t online yet. It took 30 minutes or so.” (P15)

Theme 3: Relationships influence deployment decision-making

Participants described their relationships with themselves, with family and friends, with co-workers, and with management, which are outlined in Table 10.

Table 10
Theme 3 Cluster and Examples

Cluster	Verbatim
Relationships with self	“I do what I need to do for myself, to make sure I’m doing what’s right for me and my workflow.” (P12)
Relationships with family and friends	“I wasn’t supposed to be working because I was at home with my daughter who was sick, so I was feeling a bit stressed because she was getting very restless.” (P15)
Relationships with co-workers	“We make decisions together. We talk a great deal now, but it might not be as often in the future.” (P1)
Relationships with management	“It is a very nerve-wracking and fraught experience to be asked to come to a meeting with the directors and explain what happened and why your product broke.” (P8)

Cluster 8: Relationships with self

Strategies for concentration. Five participants shared their strategies for being able to concentrate on tasks and what they did to help themselves (P1, P5, P6, P11, P12). P6 recounted balancing feature work over tidying other technical debt: “If I tried to do these things in parallel, I would go down a rabbit hole and ignore the feature work I should be doing” (P6). Another engineer described a process of “drawing and sketching, writing down things, and doodling. It helps me think” (P5).

Lack of sleep. Several participants had difficulty caring for themselves during challenging and long-running incidents and felt that they could not really take any breaks. P13 talked about the impact of a lack of sleep on their effectiveness at work as “not operating on all cylinders. It’s no different than having three or four drinks.” The same engineer worried about the impact of sleep loss on their readiness to respond to another incident, which “could happen in the middle of the night when you’re already tired and a little delirious. It’s a form of intoxication in my book” (P13). P12 described how being woken up several times during the night was a direct cause of

taking down production during their on-call shift, noting that the first time they were directly responsible for an outage that they had been paged 2 hours before they usually woke up:

The first time I was directly responsible for a production outage, I got paged at 7:00 AM. I usually wake up around 9:00 AM. I had gotten paged at least twice before that one. I didn't directly connect that what I had done to try to fix the page was what had caused the outage because of a specific symptom I was seeing. Now I'm very aware that it's more likely to be whatever the most recent change is, even if the symptoms wouldn't seem to suggest it.

P12 didn't want to wake up their co-worker and felt they should "just try one more thing, but it's hard to be self-aware in the middle of the night when things are broken, we're stressed and tired." P13 reflected: "You don't just have the stress of the company on your shoulders. You've got the stress of paying attention to what you're doing and the stress of having to do this late at night."

Cluster 9: Relationships with family and friends

Balancing work and relationships with family was discussed by many participants, especially within the context of the COVID-19 pandemic and working from home. For instance, one participant described the conflict between work and personal demands when taking care of a sick child. Often participants had to respond to an incident when they were in the middle of dropping off or picking up their children at school: "The issue started in Slack, but then we went to a Google hangout, but I was picking up my child at school. In non-pandemic times, I would have had the whole day to just worry about work" (P3). Another participant wanted to support a friend recovering from a medical procedure but had to leave suddenly when they were alerted to a critical incident: "I had to go make sure that a friend of mine has food and hasn't overloaded on painkillers. Then I started getting texts on my phone" (P7).

Cluster 10: Relationships with co-workers

Participants made deployment decisions with co-workers (see Table 11). All participants described how their relationships with co-workers influenced their decision-making experiences. One participant shared an experience of how their relationships with themselves and with their co-worker were surprisingly strengthened after a difficult incident.

It was really a good feeling because you never know how these things are going to go. It's easier to pass blame than it is to provide confidence and assistance to someone else, especially when gender and things like that are considered. It gave a lot of credence to what I was saying that I probably wouldn't have had on my own, being a queer woman in a primarily white male dad environment. (P13)

Table 11

Relationships With Co-workers

Description	Verbatim
Participants shared how working with others positively influenced their decision-making.	“We produced documents with state transitions and conditions. It was a reasonably thorough description of what the behavior should be.” (P6) “I was glad to be there for someone else and have some empathy and see them through it.” (P13)
Participants described how they interacted with others when they were working on a common goal.	“It’s valuable to sync up and determine whether the things I’m working on have a dependency on other things.” (P7) “We’re going back and forth, I’m trying to explain exactly what is going to happen and then he is verifying it.” (P9) “We often talk on Slack or a Google meet call as we make decisions together.” (P1)

Cluster 11: Relationships with management

Participants noted how relationships with management influenced their decision-making experiences. Describing a difficult database migration, P12 said, “Upper management was not straightforward with us. We compromised our technical integrity because of management pressure, [we] compromised our standards for ourselves because we were told we had to.” P12 also noted that in the early days of the pandemic, there was pressure “specifically from the CEO

for people to go into the office” and that hiring would be frozen for “any group that doesn't have people showing up to the office in sufficient numbers.”

Theme 4: What would engineers change?

After the interviews were conducted, I realized that it would be helpful to industry readers to hear participant recommendations what they would change. Participants were asked via a follow-up email: If you could wave a magic wand, what would you change about your current environment that would help you feel more confident or safe in your day-to-day deployment decisions? (E.g., tools, infrastructure, practices, and/or relationships with co-workers or management). Eight of the 15 participants chose to answer this question (see Table 11).

Table 12

Theme 4 Desired Changes and Examples

Desired Change	Verbatim
Modernization	“For this legacy project, there are three critical things that are required: 1 - Migrate the infrastructure to an infrastructure-as-code tool (e.g. terraform, cloudformation) 2 - Automate the deployment pipeline for the different environments 3 - Improve test coverage (we only have meaningful test coverage on areas where we intervened directly)”
Testing	“Thorough and automated testing, parity between prod/non-prod environments, and the time to implement all of these things.” (P4) “I return to this space every few years and am a bit surprised that this still is so hard to get right.” (P15)
Feature Flags	“Feature flag user groups or percentages of users and be able to easily compare metrics for this to the rest (like a control group). [it’s] a good approach that allows us to deploy with greater confidence.” (P2)
Post-deployment Validation	“Make it easier to assess health after a deployment. The primary product is reasonably monitored, but the ancillary services / deployments are more difficult to validate after a deploy has gone out.” (P7)
Improved Metrics	“From a previous job, I have very good experience with integration tests that exercise the subject component along with the graph of dependencies (other components, databases etc), using only public APIs. I.e. no "direct to database" fixtures, no mocking. When deploying it would be very useful to be able to feature flag user groups or percentages of users, and be able to easily compare metrics for this to the rest (like a control group). While we have very good metrics (by far the best I've ever worked with), it can still be extremely difficult to determine what causes the problem, or even whether there's a problem. E.g. the number of customers may drop because a system two steps

Desired Change	Verbatim
	removed from us is unhappy with a policy we set, and we have no visibility, except turning on trace logging on the intermediate system – which we cannot do because it has hundreds of thousands of users. Not sure what to change, but this is a frequent problem.” (P2)
Retire/operate independently	“If I could wave a magic wand I’d just get enough money to retire instead... More relevant, I think that getting spun off to operate without the company that bought the company that bought the company making technical decisions without having ever done work on the product would help a lot.”

Testing

In addition to faster pre-deployments, participants overwhelmingly spoke about needing better testing, especially front-end code. Seeking improved confidence in deployments, one engineer wanted more feature flags (software that is deployed but not “switched on”). Engineers also expressed a desire for better ways to assess health after a deployment. Another engineer wished for improved metrics that could identify problems with deployments more effectively.

Inspired by phenomenological explication, this chapter offered the findings of the study, organized into three themes that contained 15 clusters, with a fourth theme describing what engineers would change about the deployment experience. The next chapter engages in discussion with the findings, providing context for the themes and clusters identified in this chapter and exploring how they relate to other research on the topic of deployment decision-making.

Discussion

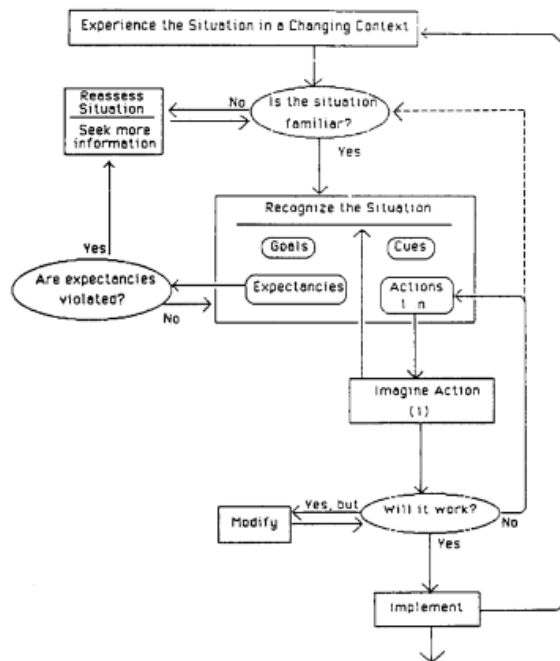
The research question that guided this study was: What are the lived experiences of engineers who deploy changes to software or configuration in production environments and how do they experience the decision to deploy those changes? Six months into the global COVID-19 pandemic, I interviewed 15 engineers about their decisions to deploy changes during the holidays between October and December 2020. Within the time allotted for the study, I chose to focus on the three most important factors that influenced the deployment decision-making experiences of engineers: risk, temporal reasoning, and relationships. I also asked participants “If they had a magic wand, what would you change?” This chapter begins with a discussion of the decision-making theories that guided the interpretation of the findings. Then I discuss the themes in context with the literature. Finally, I offer a general description of the deployment decision-making experiences as reported by participants.

Decision-making theories, mental models, and temporal reasoning

The findings of this study support the use of a naturalistic decision-making model (Klein, 1993) to interpret real-world decision-making experiences in the field of software and web operations, especially safety-critical digital services (Allspaw, 2015; Shorrock, 2022). I used Klein et al.’s (1989) critical decision method to formulate my interview questions, which generated the data. These interview prompts are used for “modeling tasks in naturalistic environments characterized by high pressure, taking into account goals, expectations, and cues” (Klein et al., 1989, p. 462). Klein’s (1993) theory of recognition-primed decision-making (RPD) emerged from studies on decision-making strategies in high-stress occupations such as firefighting. In RPD, experts imagine the action they will take and if it is likely to work, they take the action (see Figure 4). Imagining an action is part of creating a mental model.

Figure 4

Klein et al.'s Recognition-Primed Decision Model



Note: From Klein et al. (1989), p. 464. © 1989, IEEE. Reprinted with permission.

Mental models and temporal reasoning

Mental models help engineers “predict what the effect of typing a character or clicking a mouse will be” (Woods, 2017, p. 12). How do engineers decide if it is the right time to deploy a change? Participants in this study described a “need to form a clear mental picture” (P6) before deploying a change. Participant descriptions of their decision-making experiences consistently included time (time of the year, time to deploy, and duration of events), often referring to their past experiences in previous jobs where they “had things crash and burn” (P9). The findings support previous research by Sougne et al. (1993) and De Keyser (1990a), who contended that people rely on temporal reasoning (e.g., cause and effect, the duration of events), as well as conceptions of the past, present, or future, to make decisions.

Relying on Klein’s (1993) RPD, recent research on mental models in the domain of software and web operations find that mental models also include a mental model of *other* people’s mental models (Woods, 2017). As cited by Woods (2017), the “Above the Line, Below the Line” (ABL)

framework from Cook (2020) paints a vivid picture of the cognitive work of engineers' decision-making (p. 45). Figure 5 illustrates the “above the line” work of “observing, inferring, anticipating, planning, troubleshooting, diagnosing, correcting, modifying and reacting to what is happening. . . . These representations allow the people to do their work” (Woods, 2017, p. 12). Engineers' mental models are what create the systems “below the line,” using their keyboard and mouse to “interact [with] representations on their computer screens” (p. 12).

Figure 5
Above the Line/Below the Line Framework

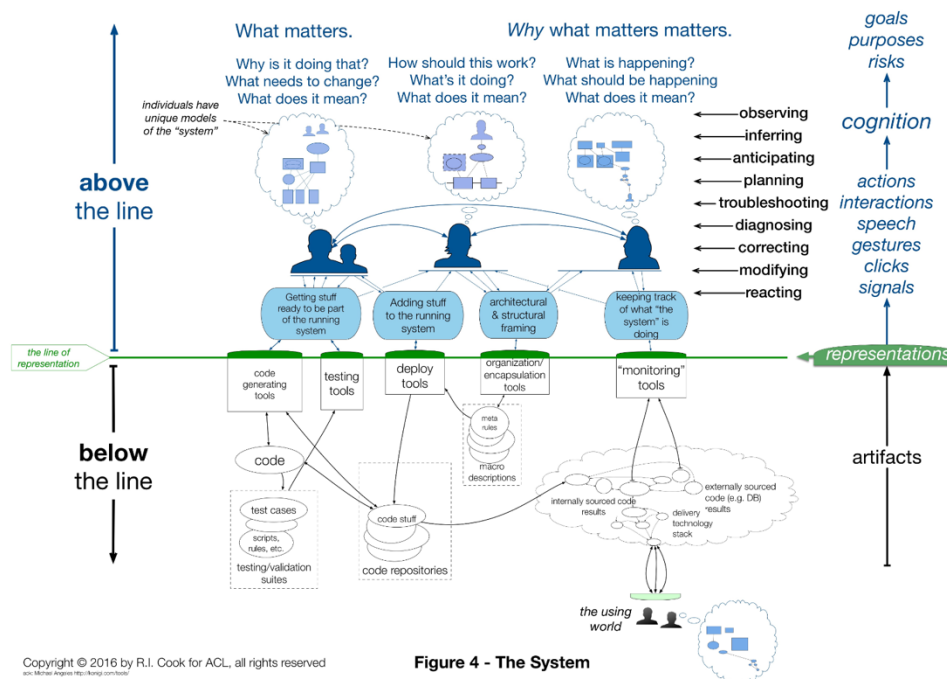


Figure 4 - The System

Note: From Woods (2017, p. 11), Copyright © 2016 by R.I. Cook for ACL.

Reprinted with permission.

Mental models are both a necessary and yet problematic aspect in deployment decision-making. Mental models can “quickly become stale” (Woods, 2017, p. 12) because the system will have already changed in ways not yet visible to the engineer. Shorrock (2022) also discussed complexity and mental models as part of the five challenges faced by engineers working with critical technical systems: dealing with change in production pressure, coping with complexity, planning and coordination, maintaining expertise, and learning from experience. He noted the

difficulty of working in complex systems and argued that “engineers can struggle with the number, scale, and speed of changes, sometimes occurring simultaneously in major software releases” (para. 5). Consistent with Woods (2017), Woods and Cook (1999), and Shorrock (2022), participants in this study described how complexity, coordination, and stale mental models contributed to incidents and outages as they made decisions based on inaccurate knowledge.

Common ground and coordination in joint activity

To support their decision-making processes, engineers coordinate their actions with the actions of other engineers, taking into account the activity happening in the technical infrastructure (networking, storage, memory, databases, etc.). Consistent with research from Klein et al. (2005) on common ground and joint activity, participants in this study described working with each other and with automation to deploy changes into production.

Decisions to deploy are inherently social. As discussed in theme three, relationships were mentioned by all participants in their experiences of deployment decision-making. Aligning with Pettersen et al. (2010), who argued that “human actions in socio-technical systems take place within the context of a social system” (p. 182), participants often described actions and decisions using the word “we.” The participants frequently mentioned the teamwork and camaraderie they felt when working together in their descriptions of their deployment decision-making experiences. In this study, cooperative relationships positively influenced participants’ decision-making experiences, and, in contrast, they described blameful relationships with management. These findings provide support for Lim and Klein (2006), who found that close teamwork improved team performance.

Relationships were particularly important during an incident response. Some participants described feeling they could call on their co-workers when they needed help and that they could trust their co-workers during the heat of an incident. When responding to an incident, the pressure to communicate effectively is crucial. During an incident, engineers must “translate”

and coordinate their actions with their colleagues. Engineers were often expected to explain the situation to customers, but reducing the technical problem into something palatable and understandable was challenging. Shorrock (2022) asserted that “without effective communication, there can be assumptions and misunderstandings about understanding who is doing what, when, why, how, with whom and for whom” (para. 10).

Through the model of a conversation between people, Klein et al. (2005) illustrated how a simple conversation is not actually so simple. Klein et al. (2005) used (1996, p. 152) joint action ladder to illustrate “the successive inferences that we make during a conversational interaction” (p. 15). “Climb[ing] the joint action ladder” (Klein et al., 2005, p. 15) requires that people notice the signal being sent, understand it and then communicate the action they will take as a result.

I believe engineers have a similar “conversation” with the technical infrastructure when they decide to deploy a change; however, as discussed in the literature review, machines cannot communicate in a way that is sufficient to establish and maintain common ground. In a recent study, Maguire (2020) extended Klein et al.’s (2005) theory to explore how “tooling designed to aid coordination incurs a cognitive cost for practitioners” (p. i). Maguire (2020) argued that “expertise smooths out the rough edges of poorly designed work practices or technologies” (p. 160).

Common ground breakdown: Automation surprises

Automation surprises are a type of fundamental common ground breakdown (Klein et al., 2005). These surprises are confusing for engineers. Under time pressure, engineers may try many potential solutions. As they search for solutions, one of the problems they may encounter is cognitive fixation or a failure to revise. Participants described going down “rabbit holes” during their deployment decision-making experiences. In a personal communication, Cook explained how to recognize cognitive fixation in software and web operations engineers:

In the early stages, the diagnosis matches the situation in enough respects that it is taken seriously. But as the situation evolves the cues and clues that could suggest alternatives do not redirect attention away from the fixation. This is particularly common when there are multiple claims on attention, increasing consequences, or cognitive “noise” in the environment [e.g., think many dashboards]. . . the requirements to characterize a practitioner as being fixated are:

1. The practitioner must have a fund of knowledge and experience that would allow for a “correct” diagnosis;
2. The practitioner must hold on to the incorrect diagnosis *in spite* of multiple opportunities to revise it;
3. The correct diagnosis must be recognized during the incident.

A key behavioral feature of “cognitive tunnel vision” episodes in the wild is that the fixated practitioners themselves immediately and conclusively recognize that they have entertained the wrong diagnosis for some time and that there were multiple opportunities to “get” the right diagnosis. (R. Cook, personal communication, July 9, 2021)

Ironies of automation

Some participants talked about the problems that automation introduced, which aligned with Bainbridge’s (1983) ironies of automation. P7 noted challenges with automated tooling and felt strongly that “there should be a human shepherding it.” In automated systems, the engineer’s role changes from operator to one of supervising the automation (Bainbridge, 1983). The need to monitor automated systems reveals a contradiction:

The automatic control system has been put in because it can do the job better than the operator, but yet the operator is being asked to monitor that it is

working effectively . . . if human judgement and intuitive reasoning are not adequate in this context, then which of the [automation] decisions is to be accepted? The human monitor has been given an impossible task. (Bainbridge, 1983, p. 776)

Automation introduces new risks

To detect problems early, deployment automation tools are configured to control the pace of the deployment, mandating waiting periods before the change can proceed to production. For some participants in this study, the promotion of a change through a deployment pipeline was gated by human review, and in other cases, the promotion was gated through automated testing criteria.

In agreement with De Keyser's (1990a, 1999b) research on temporal reasoning, participants in the current study made inferences about the progress of the deployment by considering each step in the pipeline. Controlling the pace of the deployment pipeline is designed to mitigate risk, but automation also creates new problems for engineers. Engineers come to rely on and trust certain automations (Moray et al., 1994), like the automated gating process, but they may be surprised when planned changes roll out sooner than expected. Consistent with Bainbridge (1983), Allspaw (2015), and Maguire (2020), the findings from this study support the conclusion that human expertise cannot be supplanted by technology.

Deployment decisions are risky

Participants highlighted how stale mental models, automated systems, and time pressures all contributed to the feeling that deployment decisions were risky. Although common ground with co-workers can assist in the decision-making process, the challenges of automation are compounded by the complexity of systems, which can contribute to common ground breakdowns and automation surprises. Allspaw and Cook (2018) noted that making changes in complex systems is "often stormy and dangerous" (2018, p. 439). Changes in these systems are often being made by many different people at different times, with different technologies, and

with increasingly automated deployment tools (Shorrock, 2022). Some parts of the system may have been deployed years ago; other parts may have been deployed recently (Shorrock, 2022).

Overall, the study findings indicate that deployment decisions are fraught with uncertainty and risk. All the participants made decisions under risky and uncertain conditions and worried about the potential consequences of their decisions on their co-workers, on customers, and even on human life. While it may be surprising to think that deploying a change could lead to such harm, one engineer described the potentially serious consequences of deployment decisions:

With the scale of the systems I work on now, if I make a mistake and take down the wrong cluster, there's going to be an amplification factor and might take out a life safety system such as a hospital. . . . I think I've killed far more people working in software than I did working in medicine. (P7)

General description of the deployment experience

COVID-19 Pandemic

Participants described deployment decision-making experiences during the COVID-19 pandemic and the holiday time period, both of which were presented. Although written over 30 years ago, De Keyser's assertion that the operator can no longer observe the actions of the team because of geographical constraints (p. 572) is helpful in explaining the impact of COVID-19 on the participants in this study as they recalled how their work was made much more difficult by suddenly having to work at home. The findings support research by Kaur et al. (2022) that day-to-day communication with co-workers became more difficult during the pandemic. Kaur et al. (2022) noted that "it can be difficult to establish common ground—and develop implicit coordination with colleagues when working and coordinating remotely" (p. 18). The COVID-19 pandemic introduced a new barrier for participants who already faced challenges in establishing and maintaining common ground and coordination (Klein et al., 2005).

Holiday code freezes

Besides the challenges of dealing with new working situations during the pandemic, participants were also making decisions during the holidays. Many engineers take time off during holidays, so organizations often pause deployments and freeze changes to mitigate risks since there are fewer people on call to respond. On one hand, a code freeze is a risk mitigation strategy, but code freezes can also increase risks. Engineers often create changes to code during a freeze, but these changes are not deployed, so, when the code freeze lifts, multiple changes go through all at once. Furthermore, employee interpretations of a code freeze might be different than the official stance of a company. This is an example of Shorrock's work-as-imagined (WAI) versus work-as-done (WAD): "How people *think* that work is done and how work is *actually* done are two different things" (Shorrock, 2016, para. 1). Companies enact code freezes and communicate to employees that changes should not be deployed except in the case of emergency. Companies believe that new changes are not happening; however, the reality is that engineers generally continue to work but do not push changes through to production.

Ironically, *not* deploying during a code freeze is a change, because the system is not being exercised. When the code freeze lifts, the changes that have been queued up get released all at once, potentially leading to complex failures that will be more difficult to troubleshoot than if they were released in smaller increments. Within the context of COVID-19 and holiday code freezes, participants described two types of deployment decision-making experiences: planned changes and changes in response to an incident or outage.

Planned changes

In general, when describing their experiences deploying planned changes into production environments, participants discussed the reasons for the change and considered potential solutions with their peers. After reviewing the code and running tests, participants generally created "pull requests" that show the differences between the existing code and the proposed

change, which then were deployed to pre-production test environments prior to final deployment to production (see Figure 6).

Figure 6
Example of a Pull Request with Side by Side “diff”

```
pypy/objspace/std/stringobject.py
space.is_w(space.type(w_s), space.w_unic
return w_s
return _str_join_many_items(space, w_self, list_
361
def _str_join_many_items(space, w_self, list_w, size
self = w_self._value
reslen = len(self) * (size - 1)
for i in range(size):
w_s = list_w[i]
if not space.isinstance_w(w_s, space.w_str):
if space.isinstance_w(w_s, space.w_unic
# we need to rebuild w_list here, be
# w_list might be an iterable which
w_list = space.newlist(list_w)
w_u = space.call_function(space.w_un
return space.call_method(w_u, "join"
raise operationerrfmt(
space.w_TypeError,
"sequence item %d: expected string,
357 357 space.is_w(space.type(w_s), space.w_unic
358 358 return w_s
359 359
360 360 return _str_join_many_items(space, w_self, list_
361 361
362 362 from pypy.rlib.jit import JitDriver
363 363
364 364 one = JitDriver(greens = [], reds = ['size', 'reslen
365 365 two = JitDriver(greens = [], reds = ['i', 'list_w',
366 366
367 367 def _str_join_compute_reslen(space, self, list_w, sl
368 368 reslen = len(self) * (size - 1)
369 369 for i in range(size):
370 370 one.jit_merge_point(size = size, reslen = re
371 371 self = self, list_w = li
372 372 w_s = list_w[i]
373 373 if not space.isinstance_w(w_s, space.w_str):
374 374 if space.isinstance_w(w_s, space.w_unic
375 375 return -1
376 376 raise operationerrfmt(
```

Note: Image credit: <https://bitbucket.org/blog/pull-requests-with-side-by-side-diffs>

Emergency changes

In contrast, deployment decision-making while responding to an incident or outage was generally more stressful for participants as “the passage of time and escalating consequences create enormous pressure to gain resolution quickly” (Woods, 2017, p. 21). Participants in this study first looked for recent changes to the system, researched any error messages, and moved to more synchronous conversations (e.g., Zoom or a phone call) with co-workers to troubleshoot. With a better understanding of the problem, they took actions that they believed would restore normal functioning such as rolling back the change.

Strategies

Morel et al.’s (2008) temporal strategies of imagining the change, adapting to the situation, and managing the outcomes were very helpful in explaining the findings of this study. Participants imagined how their changes would behave before and after deployment. Participants described using strategies to mitigate risks that enabled them to proceed with deployment decisions, adapting when incidents and outages occur, and managing outcomes. In the field of software and web operations, engineers and designers imagine the different failure modes when deciding to

deploy changes. In the event of an incident or outage, the engineer adapts and responds to the incident in order to figure out what went wrong. After the incident has been stabilized, there is typically some form of post-incident analysis.

Imagining the situation. As discussed at the beginning of this chapter, people create mental models of the situation before making a decision. For engineers, part of forming a mental model is reducing and oversimplifying it to cope with the complexity of the system. Woods and Cook (1999) suggested that oversimplifications are a coping mechanism for decision-making under demanding conditions and that using simplified models of otherwise complex systems is necessary to make decisions. When engineers feel confident in their mental model of the system, they take action to deploy the change. They then look for confirmation that the change is having the intended effect. If all goes well, they move on with other tasks. As discussed in the introduction to this thesis, deployments sometimes lead to incidents or outages.

Adapting to the situation. Like Morel et al. (2008), De Keyser highlighted the necessity of adjusting a plan to adapt to a complex situation. When responding to an incident, participants recalled trying to understand what had gone wrong. P11 said: “The first thing for me is: What changed?” This was consistent with Allspaw’s (2015) findings that engineers check whether a change could be the cause of an incident, and if not, they look for other possible causes that remind them of past incidents with similar characteristics.

To adapt, engineers often try to roll back the change or make new changes to stabilize the system. The ability to roll back is a critically important capability as well as having the knowledge to determine whether rolling back is the appropriate course of action. Rolling back is highly dependent on capabilities being built in advance, regularly practicing rolling back to understand how roll backs work and in what conditions they do not work, but there are no guarantees that roll backs work. There is a gap in the literature on roll backs. The findings in this study begin to address this gap.

One of the most interesting findings on adapting was from P12. Although their team had agreed on not requiring pull requests (a line-by-line comparison of a proposed change against the existing code or configuration), P12 chose to do them anyway as a form of self-care. Unknown to P12 at the time, their change contained a typo which silently invalidated critical backup configurations. P12 said that because they used a pull request as a strategy, the typo was detected. Luckily, they did not have an incident that required restoring from backup, as no backups existed.

Sacrifice decisions. Another way engineers adapt to the situation is to make sacrifice decisions. Participants reported that certain sacrifices had to be made in order to buy more time to make further decisions. In a sacrifice decision, engineers accept reduced functionality and often have to forego normal testing to quickly deploy the change into production. According to Woods (2017), sacrifice decisions usually occur in high-pressure situations. Allspaw and Cook (2018) described how engineers often deal with situations where there are “only bad alternatives” (p. 445).

Managing outcomes. The third strategy identified by Morel et al. (2008) was managing decision outcomes. According to Woods (2017), “Postmortems are private or semi-private events intended to identify and capture important factors and features associated with anomalies” (p. 24). Several participants in the study mentioned using postmortems to evaluate and reflect on incidents or outages. One participant described having a “project manager who manages the outcome of the actions that are taken, that stuff is fed back to senior management.”

Whether deploying a planned change or responding to an incident, participants in this study had to make decisions in risky and uncertain situations based on mental models that were often incorrect due to the complexity and automation of the systems in which they worked. Table 13 shows how the findings informed the research questions. This study supports research on temporal decision-making (De Keyser, 1990b), automation (Bainbridge, 1983), and mental models (Johnson-Laird, 1983). Participants described decision-making strategies that aligned

with Morel et al.'s (2008) research on temporal strategies. Participants also highlighted the role of relationships in their decision-making experiences (Lim & Klein, 2006, p. 404). The findings of this study add to the existing literature on naturalistic models of decision-making (Klein, 1993) and contribute to the safety science literature in the domain of software and web operations. Participants noted that their deployment decisions can have serious and wide-ranging impacts. Deployment systems do not provide confidence to the operator that the change they intend to make are safe; there is a need for better tooling. The final chapter of this study discusses how organizations can implement changes to make deployment decision-making safer and offers suggestions for future research.

Table 13*Research Questions*

Research Question	Answer	Quote
What are the deployment decision-making experiences of engineers?	Planned or unplanned, sometimes led to incidents or outages, every deployment has some level of automation, emergency fixes, sacrifice decisions, had strategies to manage outcomes when things went wrong, supervising/mentoring co-worker, all participants suggested ways to improve the experience	<p>"We make decisions together." (P1)</p> <p>"Hey, does this look good to you?" (P1)</p> <p>"Actually happy about this outage because they learned a lot of interesting stuff." (P2)</p>
How do they experience the decision to deploy those changes?	Risky and uncertain, incomplete mental models, stressful, temporal reasoning, work from home, awareness of possibility of harm to human life, self-care, blameful management	<p>"Always chasing the last problem and never catching the one that's coming." (P6)</p> <p>"Postmortems are a nerve-wracking and fraught experience, the meetings are filled with corporate politics." (P10)</p>
How were engineers' experiences difficult or frustrating?	Confusion about state of automation, sufficiently replicating production conditions, code freezes, coordination, being unable to observe the change/progress, alerts behaving differently than intended, working from home, COVID-19 pandemic, triaging, when failures were not caught by test environments, balancing communication versus fixing,	<p>"One fragile little instance that we couldn't break because we would then lose data which we really try not to do." (P9)</p> <p>"It was unfortunate because I was just on my phone, in the parking lot of the school. In non-pandemic situations you would have had the whole day to just worry about the work." (P3)</p>
How was time pressure a factor in engineers' deployment decision-making experiences?	Holidays, code freezes, severity of problem, from management and customers, time of day, time zones	<p>"Chasing my tail." (P6)</p>
How did engineers derive confidence in the decision to deploy?	Testing, co-worker support, pairing, risk mitigation strategies, being able to deploy changes quickly in an emergency, checklists, runbooks, discussing changes with co-workers, coordination, believing the change is low-risk	<p>"Git-ops deployment tooling enables us to change a tag on a container... the change rolls out within minutes." (P2)</p> <p>"Pipelines, peer review, unit tests and QA give me confidence... but there's a quantifiable cost to building confidence through testing." (P4)</p>
How did engineers' previous work experiences factor into their decisions to deploy the change?	All participants mentioned previous experiences, previous experience was similar enough that it influenced the decision to deploy or not deploy a change, previous experience also influenced decision to roll back and knowledge of how to roll back, previous experiences of others were also referenced, previous experiences caused people to be more careful	<p>"Reminded of a previous experience of trying to shift tasks from startup and instead bake them into the image but it completely blew up." (P1)</p> <p>"A lot of the reluctance to page other people comes from my previous job." (P12)</p>

Issues and Implications

The aim of this study was to collect descriptions of the lived experiences of engineers deploying changes that could be used to guide the design of deployment automation tooling that is safer to operate. Engineers in this study made deployment decisions by *talking* to their co-workers, *interpreting* the state of systems, *coordinating* their deployments, and *monitoring* progress.

Participants sought information from systems, from documentation, from others, and from their own previous experiences, then carried that information into a model of the world (Budd, 2012, p. 78). While an engineer brings with them previous experiences, the tools and conditions of their work environment determine the outcome of the decision to deploy. Historical events, power structures, and hierarchy “enable and set the stage for all human action” (Dekker & Nyce, 2014, p. 47).

Time introduced a tension between speed and safety for participants deploying changes in production. Changes were rolled out slowly, but when there was a problem with the deployment, the expectation was to deploy as fast as possible. Rolling out a change at a slow pace introduces a different risk of temporally separating the potential bad outcome from the person who has the most context for fixing it. The downside of deploying slowly is that the consequences of the deployment decision may occur long after the original change was made. Therefore, the person who made the change and has the most knowledge about the change may not be there to help fix any problems that arise. As Lim & Klein (2006) recommended, organizations should consider interventions that support team mental model similarity and accuracy to support team performance, especially where many different engineers are contributing to the codebase.

Organizations should seek to understand how past and future experiences influence present decisions so they can make investments in tooling that support temporal reasoning, “for the effects of an action can be totally different, if performed too early or too late” (De Keyser, 1990b, p. 574).

A key finding from this study was: *There's no place like production*. Testing might pass in multiple pre-production environments, but tests do not ensure success because “production is a unique place that cannot be fully replicated” (P4). Participants were asked a follow-up question via survey after the interviews: If you could wave a magic wand, what would you change about your current environment that would help you feel more confident or safe in your day-to-day deployment decisions? (E.g., tools, infrastructure, practices, and/or relationships with co-workers or management). Participants overwhelmingly spoke about needing better testing capabilities to improve confidence in deployments. Participants sought advanced capabilities for feature flags, better ways to assess health after a deployment, and improved metrics that could identify problems with deployments more effectively. The findings suggest a need to reduce the risk and uncertainty engineers face—especially in contexts in which production and test environments are vastly different—and a need to improve testing capabilities.

Deployment decisions cannot be understood independently from the social systems, rituals, and organizational structures in which they occur (Pettersen et al., 2010, p. 182) Deciding to deploy a change is not an isolated, individual experience, but is rather a very social experience. Studies of high-hazard industries that carry the potential for catastrophic accidents, such as nuclear power plants and aircraft carriers, require strong social systems and high levels of psychological safety to ensure safe operations (Carroll, 1998; Edmondson & Lei, 2014; Weick & Roberts, 1993). I think of psychological safety in the workplace as the freedom to speak up about risks and not be punished or humiliated for making mistakes; however, organizations that espouse valuing psychological safety should also understand that it is only an “explanatory construct—a set of intangible interpersonal beliefs and predictions—*rather* [emphasis added] than a managerial lever or action. There are actions leaders can take to build psychological safety . . . but it cannot be mandated or altered directly” (Edmondson, 2002, p. 26). I would encourage organizations to recognize that decision-making is not an individual, solitary act. Confidence in deployment decisions is enhanced where there are cooperative relationships.

The software engineering industry has not achieved the blameless culture that we aspire to. As outlined in the discussion section, despite all the tools for automating deployments, automation introduces new problems, and decision-making is still risky. All changes have the potential to break production. Engineers want to be confident in their deployment decisions, but if they are blamed for something out of their control, they may be hesitant to deploy. Despite decades of research on the psychological relationship between humans and machines, and the allocation of tasks between humans and machines, the view of humans as deficient operators in complex systems remains prevalent in software and web operations. Organizations should improve their ability to learn from incidents and implement forward-looking models of accountability (Dekker, 2016) to avoid hindsight bias or blame; far from human error being the cause of these incidents and outages, the present study provides convincing evidence for the argument that automation has yet to be a “team player” (Klein et al., 2004; Woods et al., 1997). Human expertise is *still* required to respond to automation failures (Bainbridge, 1983). As Woods remarked in an interview with Hobson (2019), “People are the hidden source of resilience that make these systems work much more often than we would expect, given how brittle they are” (para 8).

Considering the complexity of deployment environments, and the diverse factors that participants experienced as influencing decision-making, additional research on deployment decision-making would be beneficial in safety science and software engineering. In particular, it would be fruitful to conduct further research on why engineers decide *not* to deploy, with a larger sample size. Future research avenues could include an observational approach as decisions to deploy are being made. Process tracing studies in software engineering have provided valuable insights into understanding coordination in software engineering operations (Allspaw, 2015; Grayson, 2018; Maguire, 2020) and could provide a robust complement to lived experience research on deployment decision-making generated through phenomenology methods. Further phenomenological research on decision-making could also include the descriptive experience

sampling (DES) method, where engineers are provided with a device that emits a gentle acoustic signal at random moments (Kordes, 2009). Then, the participant “tries to ‘freeze’ his/her experience immediately prior to the beep—by describing it as accurately as possible into a handy notebook” (Kordes, 2009, p. 68).

Revisiting the existential questions posed by Shorrock (2022):

What will be the unintended consequence of a software update on collaborating technical systems? How can we detect problems with code in a software release when there are no obvious consequences until specific operational conditions occur? How can one know in which release a bug was introduced? Should we roll back to a previous software release (which may itself contain bug fixes), or try to find and fix the bug we are presented with now? (para 10)

I hope that the study provides partial answers to these questions.

References

- Adams, B., & McIntosh, S. (2016). Modern release engineering in a nutshell: Why researchers should care. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 5, 78–90. <https://doi.org/10.1109/SANER.2016.108>
- Allspaw, J. (2015). *Trade-offs under pressure: Heuristics and observations of teams resolving internet service outages* [Master's thesis, Lund University]. <http://lup.lub.lu.se/student-papers/record/8084520>
- Allspaw, J., & Cook, R. (2018). SRE cognitive work. In D. Blank-Edelman (Ed.), *Seeking SRE* (pp. 445–462). O'Reilly Media.
- Bainbridge, L. (1983). Ironies of automation. *Automatica: The Journal of IFAC, the International Federation of Automatic Control*, 19(6), 775–779. [https://doi.org/10.1016/0005-1098\(83\)90046-8](https://doi.org/10.1016/0005-1098(83)90046-8)
- Ballinger, K. (2020, May 22). *April service disruptions analysis*. Github. <https://github.blog/2020-05-22-april-service-disruptions-analysis/>
- Barrow, J. M., Brannan, G. D., & Khandhar, P. B. (2022). Research ethics. In *23StatPearls [Internet]*. StatPearls Publishing. <https://www.ncbi.nlm.nih.gov/books/NBK459281/>
- Bhartiya, S. (2018, January 5). *How Netflix built Spinnaker, a high-velocity continuous delivery platform*. The New Stack. <https://thenewstack.io/netflix-built-spinnaker-high-velocity-continuous-delivery-platform/>
- Blackwell, E. (2016, February 9). *Telstra struck by mass phone, internet outage*. HuffPost. https://www.huffingtonpost.com.au/2016/02/08/telstra-outage-australia_n_9191146.html
- Blaxter, L., Hughes, C., & Tight, M. (2006). *How to research* (3rd ed.). Open University Press.
- Block, R. A. (1990). Models of psychological time. In R. A. Block (Ed.), *Cognitive models of psychological time* (Vol. 283, pp. 1–35). Lawrence Erlbaum Associates.
- Borg, M., de la Vara, J. L., & Wnuk, K. (2016). Practitioners' perspectives on change impact analysis for safety-critical software - A preliminary analysis. *5th International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems (SASSUR 2016), September 20-23, 2016, Trondheim, Norway, 9923*, 346–358. https://doi.org/10.1007/978-3-319-45480-1_28

- Braun, V., Clarke, V., & Terry, G. (2015). Thematic Analysis. In *Qualitative Research in Clinical and Health Psychology* (pp. 95–113). Macmillan Education UK. https://doi.org/10.1007/978-1-137-29105-9_7
- Budd, J. M. (2012). Phenomenological critical realism: A practical method for LIS. *Journal of Education for Library and Information Science*, 53(1), 69–80. <http://www.jstor.org/stable/23249097>
- Carroll, J. S. (1998). Organizational learning activities in high-hazard industries: The logics underlying self-analysis. *Journal of Management Studies*, 35(6), 699–717. <https://doi.org/10.1111/1467-6486.00116>
- Clark, H. H. (1996). Joint actions. In *Using Language* (pp. 59–91). Cambridge University Press. <https://doi.org/10.1017/cbo9780511620539.004>
- Cook, R. I. (1999). Two years before the mast: learning how to learn about patient safety. *Enhancing Patient Safety and Reducing Errors in Health Care*, 8–10.
- Cook, R. I. (2020). Above the line, below the line. *Communications of the ACM*, 63(3), 43–46. <https://doi.org/10.1145/3379510>
- Creswell, J. (2006). *Qualitative inquiry and research design: Choosing among five approaches* (2nd ed.). SAGE Publications.
- Creswell, J. (2007). *Qualitative Inquiry and Research Design*. Sage.
- De Keyser, V. (1990a). Temporal decision making in complex environments. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 327(1241), 569–576. <https://doi.org/10.1098/rstb.1990.0099>
- De Keyser, V. (1990b). Temporal decision making in complex environments. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 327(1241), 569–576. <https://doi.org/10.1098/rstb.1990.0099>
- Dekker, S. (2016). *Just culture: Balancing safety and accountability*. CRC Press. <https://doi.org/10.4324/9781315251271>
- Dekker, S., & Nyce, J. M. (2014). There is safety in power, or power in safety. *Safety Science*, 67, 44–49. <https://doi.org/10.1016/j.ssci.2013.10.013>

- Dijkstra, B. (2017, August 21). *5 effective and powerful ways to test like tech giants*. TechBeacon.
<https://techbeacon.com/app-dev-testing/5-effective-powerful-ways-test-tech-giants>
- Edmondson, A. C. (2002). *Managing the risk of learning: Psychological safety in work teams*. Division of Research, Harvard Business School.
<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.1943&rep=rep1&type=pdf>
- Edmondson, A. C., & Lei, Z. (2014). Psychological safety: The history, renaissance, and future of an interpersonal construct. *Annual Review of Organizational Psychology and Organizational Behavior*, 1(1), 23–43. <https://doi.org/10.1146/annurev-orgpsych-031413-091305>
- Google. (2020). *Google Cloud Infrastructure Components Incident #20013*. Google Cloud Status Dashboard. <https://status.cloud.google.com/incident/zall/20013#20013004>
- Grayson, M. R. (2018). *Approaching overload: Diagnosis and response to anomalies in complex and automated production software systems* [The Ohio State University]. <http://dx.doi.org/>
- Hanna, J. (2013, February 8). *Manufacturer blames Super Bowl outage on incorrect setting*. CNN.
<https://www.cnn.com/2013/02/08/us/superdome-power-outage/index.html>
- Hobson, J. (2019, June 18). *Argentina blackout and massive Target glitch raise question: Why do systems fail?* [Radio broadcast interview transcript]. WBUR.
<https://www.wbur.org/hereandnow/2019/06/18/why-do-systems-fail>
- Hultgren, F. H. (1990). Researching Lived Experience: Human Science for an Action Sensitive Pedagogy by Max van Manen. In *Phenomenology Pedagogy* (pp. 361–366).
<https://doi.org/10.29173/pandp15124>
- Hycner, R. H. (1985). Some guidelines for the phenomenological analysis of interview data. *Human Studies*, 8(3), 279–303. <https://doi.org/10.1007/bf00142995>
- Johnson-Laird, P. N. (1983). *Mental models: Towards a cognitive science of language, inference, and consciousness*. Harvard University Press.
- Kabir, H. M. D., Khosravi, A., Mondal, S. K., Rahman, M., Nahavandi, S., & Buyya, R. (2021). Uncertainty-aware decisions in cloud computing: Foundations and future directions. *ACM Computing Surveys*, 54(4), 1–30. <https://doi.org/10.1145/3447583>

- Kaur, M., Parkin, S., Janssen, M., & Fiebig, T. (2022). "I needed to solve their overwhelmness": How System Administration Work was Affected by COVID-19. *Proc. ACM Hum.-Comput. Interact.*, 6(CSCW2), 1–30. <https://doi.org/10.1145/3555115>
- Kerzazi, N., & Adams, B. (2016). Botched Releases: Do We Need to Roll Back? Empirical Study on a Commercial Web App. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 1, 574–583. <https://doi.org/10.1109/SANER.2016.114>
- Khoo, Y.-L., & Mosier, K. (2008). The Impact of Time Pressure and Experience on Information Search and Decision-Making Processes. *Journal of Cognitive Engineering and Decision Making*, 2(4), 275–294. <https://doi.org/10.1518/155534308X377784>
- Klein. (1993). A Recognition-primed decision (RPD) model of rapid decision making. In G. Klein, J. Orasanu, R. Calderwood, & C. Zsombok (Eds.), *Decision making in action: Models and methods* (pp. 138–147). Ablex Publishing Corporation.
- Klein, G., Calderwood, R., & MacGregor, D. G. (1989). Critical decision method for eliciting knowledge. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(3), 462–472. <https://doi.org/10.1109/21.31053>
- Klein, G., Feltovich, P. J., Bradshaw, J. M., & Woods, D. D. (2005). *Common ground and coordination in joint activity*. Academia.edu. https://www.academia.edu/download/31764257/Common_Ground_Single.pdf
- Klein, G., Woods, D. D., Bradshaw, J. M., Hoffman, R. R., & Feltovich, P. J. (2004). Ten challenges for making automation a "team player" in joint human-agent activity. *IEEE Intelligent Systems*, 19(06), 91–95. <https://doi.org/10.1109/mis.2004.74>
- Knodel, J., & Naab, M. (2014, February). *Mitigating the Risk of software change in practice: Retrospective on more than 50 architecture evaluations in industry (Keynote paper)*. 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), Antwerp, Belgium. <https://doi.org/10.1109/CSMR-WCRE.2014.6747171>
- Kordes, U. (2009). The phenomenology of decision making. *Interdisciplinary Description of Complex Systems*, 7(2), 65–77. <http://dx.doi.org/>

- Leveson, N. G. (2011). Applying systems thinking to analyze and learn from events. *Safety Science*, 49(1), 55–64. <https://doi.org/10.1016/j.ssci.2009.12.021>
- Lim, B.-C., & Klein, K. J. (2006). Team mental models and team performance: A field study of the effects of team mental model similarity and accuracy. *Journal of Organizational Behavior*, 27(4), 403–418. <https://doi.org/10.1002/job.387>
- Lipshitz, R. (1993). Converging themes in the study of decision making in realistic settings. In G. Klein, J. Orasanu, R. Calderwood, & C. E. Zsombok (Eds.), *Decision making in Action: Models and methods* (pp. 103–137). Ablex Publishing Corporation.
<http://www.academia.edu/download/62235127/5f06ae853c5ce3f7da5e86bb9f59c88b50b320200229-84777-abct9q.pdf#page=115>
- Lund University. (2022). *Ethical review*. <https://www.staff.lu.se/research-and-education/research-support/support-research-process/research-ethics-and-animal-testing-ethics/ethical-review>
- Maguire, L. M. D. (2020). *Controlling the Costs of Coordination in Large-scale Distributed Software Systems* [The Ohio State University].
https://etd.ohiolink.edu/apexprod/rws_etd/send_file/send?accession=osu1593661547087969&camp;disposition=inline
- Moray, N., Hiskes, D., Lee, J., & Muir, B. M. (1994). Trust and human intervention in automated systems. In *Expertise and technology: cognition & human-computer cooperation* (pp. 183–194). L. Erlbaum Associates Inc. <https://dl.acm.org/citation.cfm?id=214713>
- Mörel, G., Amalberti, R., & Chauvin, C. (2008). Articulating the differences between safety and resilience: the decision-making process of professional sea-fishing skippers. *Human Factors*, 50(1), 1–16. <https://doi.org/10.1518/001872008X250683>
- Nolan, L. (2020, June). *A terrible, horrible, no-good, very bad day at Slack*. Slack.
<https://slack.engineering/a-terrible-horrible-no-good-very-bad-day-at-slack-dfe05b485f82>
- Orasanu, J., & Connolly, T. (1993). The reinvention of decision making. In G. Klein, J. Orasanu, R. Calderwood, & C. E. Zsombok (Eds.), *Decision making in action: Models and methods* (Vol. 1, pp. 3–20). Ablex Publishing.

<http://www.academia.edu/download/62235127/5f06ae853c5ce3f7da5e86bb9f59c88b50b320200229-84777-abct9q.pdf#page=15>

- Ordóñez, L., Benson, L., III, & Pittarello, A. (2015). Time-pressure perception and decision making. In G. W. Gideon Keren (Ed.), *The Wiley Blackwell handbook of judgment and decision making* (pp. 517–542). John Wiley & Sons. <https://doi.org/10.1002/9781118468333.ch18>
- Pettersen, K. A., McDonald, N., & Engen, O. A. (2010). Rethinking the role of social theory in socio-technical analysis: A critical realist approach to aircraft maintenance. *Cognition, Technology & Work*, 12(3), 181–191. <https://doi.org/10.1007/s10111-009-0133-8>
- Polkinghorne, D. E. (1989). Phenomenological research methods. In R. S. Valle & S. Halling (Eds.), *Existential-phenomenological perspectives in psychology: Exploring the breadth of human experience* (pp. 41–60). Springer US. https://doi.org/10.1007/978-1-4615-6989-3_3
- Rasmussen, J. (1983). Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3), 257–266. <https://doi.org/10.1109/tsmc.1983.6313160>
- Rierson, L. K. (2001). Changing safety-critical software. *IEEE Aerospace and Electronic Systems Magazine*, 16(6), 25–30. <https://doi.org/10.1109/62.931137>
- Rochlin, G. I., La Porte, T. R., & Roberts, K. H. (1987). The self-designing high-reliability organization: Aircraft carrier flight operations at sea. *Naval War College Review*, 40(4), 76–92. <http://www.jstor.org/stable/44637690>
- Rockwell, N. (2021, June 8). *Summary of June 8 outage*. Fastly. <https://www.fastly.com/blog/summary-of-june-8-outage>
- Saldana, J. M. (2015). *The coding manual for qualitative researchers* (3rd ed.). SAGE Publications.
- Schermann, G., Cito, J., Leitner, P., & Gall, H. C. (2016). Towards quality gates in continuous delivery and deployment. *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, 1–4. <https://doi.org/10.1109/icpc.2016.7503737>
- Shorrock. (2016). The varieties of human work. *Humanistic Systems*. <https://humanisticsystems.com/2016/12/05/the-varieties-of-human-work/>

- Shorrock. (2022, February 9). *We need to talk about engineering*. Humanistic Systems.
<https://humanisticsystems.com/2022/02/09/we-need-to-talk-about-engineering/>
- Shorrock, S. (2019). Safety research and safety practice: Islands in a common sea. In J.-C. Le Coze (Ed.), *Safety science research* (pp. 223–245). CRC Press. <https://doi.org/10.4324/9781351190237>
- Slovic, P., Finucane, M. L., Peters, E., & MacGregor, D. G. (2007). The affect heuristic. *European Journal of Operational Research*, 177(3), 1333–1352. <https://doi.org/10.1016/j.ejor.2005.04.006>
- Slovic, P., Peters, E., Finucane, M. L., & Macgregor, D. G. (2005). Affect, risk, and decision making. *Health Psychology: Official Journal of the Division of Health Psychology, American Psychological Association*, 24(4 Suppl.), S35-40. <https://doi.org/10.1037/0278-6133.24.4.S35>
- Sougné, J., Nyssen, A.-S., & De Keyser, V. (1993). Temporal reasoning and reasoning theories - A case study in anaesthesiology. *Psychologica Belgica*, 33(2). <https://doi.org/10.5334/pb.856>
- Starbuck, W., & Farjoun, M. (2009). *Organization at the limit: Lessons from the Columbia disaster*. John Wiley & Sons. <https://doi.org/10.2189/asqu.51.2.293>
- Suddendorf, T., Bulley, A., & Miloyan, B. (2018). Propection and natural selection. *Current Opinion in Behavioral Sciences*, 24, 26–31. <https://doi.org/10.1016/j.cobeha.2018.01.019>
- Swedish Ethical Review Authority. (2018, November 28). *Vad säger lagen?*
<https://etikprovningsmyndigheten.se/for-forskare/vad-sager-lagen/>
- Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. *Science*, 185(4157), 1124–1131. <https://doi.org/10.1126/science.185.4157.1124>
- Vaughan, D. (1996). *The Challenger Launch Decision: Risky Technology, Culture, and Deviance at NASA*. University of Chicago Press. <https://play.google.com/store/books/details?id=6f6LrdOXO6wC>
- Warren, T. (2021, June 8). *Huge web outage takes Reddit, Twitch, and other big sites offline for an hour*.
<https://www.theverge.com/2021/6/8/22523953/twitch-reddit-down-fastly-outage-issues>
- Weick, K. E., & Roberts, K. H. (1993). Collective mind in organizations: Heedful interrelating on flight decks. *Administrative Science Quarterly*, 38(3), 357–381. <https://doi.org/10.2307/2393372>
- Woods, D. D. (1993). Process-tracing methods for the study of cognition outside of the experimental psychology laboratory. In G. Klein, J. Orasanu, R. Calderwood, & C. E. Zsombok (Eds.),

Decision making in action: Models and methods (pp. 228–251). Ablex Publishing Corporation.

<https://onlinelibrary.wiley.com/doi/10.1002/bdm.3960080307>

Woods, D. D. (2017). *STELLA: Report from the SNAFUcatchers workshop on coping with complexity*. The Ohio State University. <https://snafucatchers.github.io/>

Woods, D. D., & Cook, R. I. (1999). Perspectives on human error: Hindsight bias and local rationality.

In F. Durso (Ed.), *Handbook of applied cognitive psychology* (pp. 141–171). Wiley.

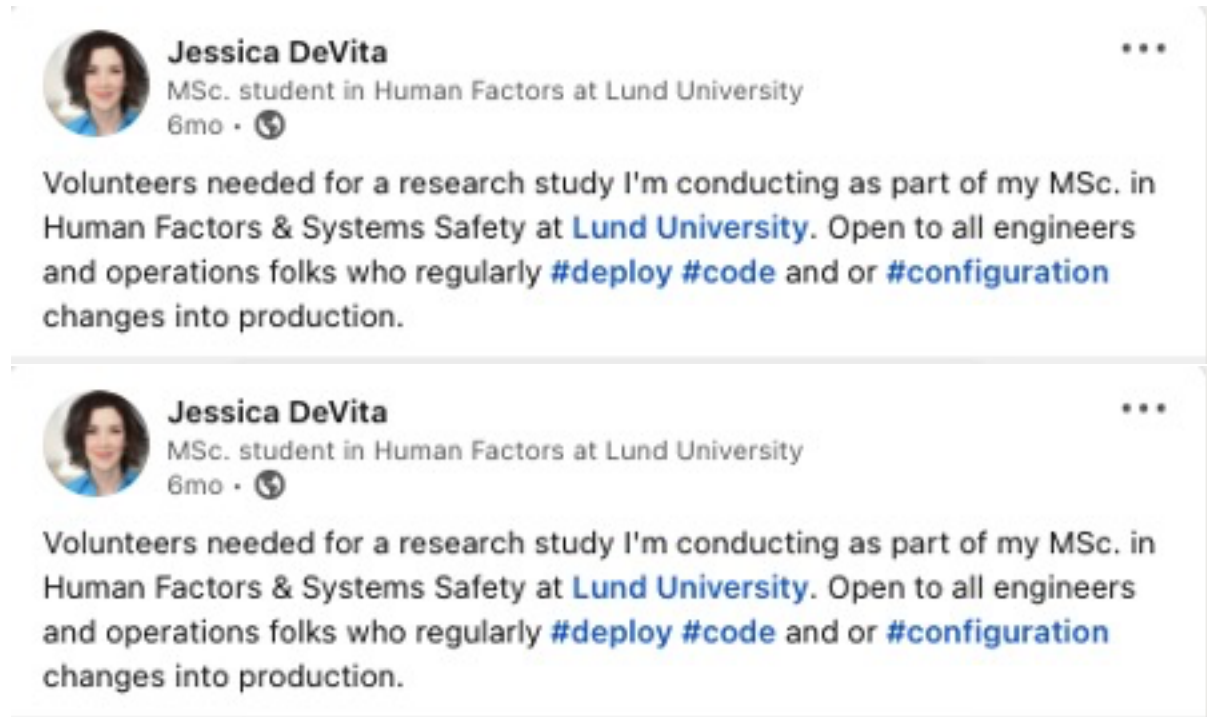
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.474.3161>

Woods, D. D., Flanagan, J., Huang, T., Jones, P., Kasif, S., & Others. (1997). Human-centered software agents: Lessons from clumsy automation. *J. Flanagan, T. Huang, P. Jones, & S. Kasif, S. (Eds.), Human Centered Systems: Information, Interactivity, and Intelligence*, 288–293.

Appendices

Appendix A: Recruitment

<https://www.linkedin.com/posts/activity-6745995450819710976-rVNN?>



The image shows two identical screenshots of a LinkedIn post. Each post features a profile picture of Jessica DeVita, her name, and her title 'MSc. student in Human Factors at Lund University'. The post text reads: 'Volunteers needed for a research study I'm conducting as part of my MSc. in Human Factors & Systems Safety at Lund University. Open to all engineers and operations folks who regularly #deploy #code and or #configuration changes into production.' The posts are separated by a horizontal line.

Jessica DeVita MSc. student in Human Factors at Lund University
6mo • 🌐

Volunteers needed for a research study I'm conducting as part of my MSc. in Human Factors & Systems Safety at **Lund University**. Open to all engineers and operations folks who regularly **#deploy #code** and or **#configuration** changes into production.

Jessica DeVita MSc. student in Human Factors at Lund University
6mo • 🌐

Volunteers needed for a research study I'm conducting as part of my MSc. in Human Factors & Systems Safety at **Lund University**. Open to all engineers and operations folks who regularly **#deploy #code** and or **#configuration** changes into production.

<https://twitter.com/UberGeekGirl/status/1340189417504272388>

Appendix B: Recruitment Flyer

Volunteers Wanted for Research Study on Deployment Decision Making

A study of the experiences of engineers and operators making decisions to deploy software and/or configuration changes.

This is an invitation to developers, engineers, or operations staff who deploy changes to software and/or configuration in production environments.

You will be asked to participate in an interview (approximately 60 minutes) and a follow-up interview (approximately 30 minutes) if needed. The topic of both interviews is your experiences with deciding to deploy software and/or configuration changes in production environments. The interviews will be conducted over video conference and will be recorded.

Confidentiality

No one will know you are participating other than the researcher. Reporting of the results will not include any identifying information. All personal identifiers will be removed by the researcher, and all audio/video recordings will be destroyed immediately after the study.

In order to participate, you must meet the following criteria:

1. Must be at least 18 years of age
2. Must identify as a person who regularly deploys software and/or configuration changes in production environments.

If you are interested in participating in this study or would like more information, please initial here ___ to proceed to the screening questions. If you wish to confirm your participation, please review and sign the Participant Consent form. Contact the researcher if you would like more information.

Appendix C: Screening Materials

Participants must:

1. be at least 18 years of age
2. identify as a person who deploys software and/or configuration changes in production environments.
3. [Click here](#) to request to use a translator if you do not speak English but would like to participate.

Criteria for Qualifying

1. Regularly makes decisions to deploy software and/or configuration changes in production environments.

Please proceed to the Consent letter ____initial here

Appendix D: Participant Consent

Title: How do engineers decide to deploy software and configuration changes in production?

This study is being performed as partial fulfillment of the requirements for the master's degree in Human Factors and Systems Safety at Lund University.

Dear Participant,

You are being asked to participate in a research study that seeks to understand the lived experiences of developers, engineers, and/or operations staff who deploy software or configuration changes in production. You will be asked to participate in an interview (approximately 60 minutes) and a follow-up interview (approximately 30 minutes). The interviews will be conducted over video conference and will be recorded.

Risks and compensation:

Involvement in this study will not impose any risks to your safety and well-being. This study is optional, and participants will not be monetarily compensated. Participation will require no monetary cost to you.

Confidentiality:

Participant identifying information will remain anonymous throughout this study and within any subsequent reports originating from this study, as well as this study's publication. Only the researcher will be aware of participants names, and participants names will be coded with pseudonyms to protect privacy. The video recording will be transcribed and all identifying information about you will be removed. The recordings will be deleted after transcription and analysis.

Right to withdraw:

This study is voluntary. You are free to accept or refuse the invitation, and you are also free to leave the study or interview (and not complete the study) at any time, without offering an explanation. Should you choose to participate in the study, your participation will remain strictly confidential. Your participation or lack thereof in the study will in no way impact your employment position.

Summary of results:

A summary of the results of this research will be provided to you, at no cost, upon request.

Voluntary consent:

I have read the above statements and understand what is being requested of me. I also understand that my participation is voluntary, I am free to withdraw my consent at any time, for any reason. On these terms, certify that I am willing to participate in this research project. I understand that if I have any questions about my participation, I may contact the researcher.

Appendix E: Participant Narratives

P1 is an SRE who recently joined the company, and was pairing with his co-worker, "in constant communication" and talking through things with a "shared screen" on a change they were planning to make. They were working on an optimization to try to reduce the startup time for a particular service. P1 described how they begin their day and approach their work:

So what I do for almost anything significant I am involved in, I keep a text file for the entirety of the project anything that I might learn, anything that's open, people I have to talk to, and tasks I have to do. This also serves as an entry point for each day because I can simply review what I wrote down - like notes, resources, and open tasks that I have left. It gives me a single view on the entirety of the project so far and aggregates the open projects into a more comprehensive list. This is something that I do to work with the way my brain works, which is to say that I'm terrible at retaining information, unless I write it down. So I don't have to worry about what am I forgetting, what am I to remember? It has the incredibly handy side effect of being able to give to new people instantly.

They tested the proposed solution and it appeared to work as planned. As they were preparing to deploy to production, P1 felt "apprehensive" because he was reminded of a previous experience of trying to shift tasks from startup and instead bake them into the image but it "completely blew up". He pointed out "specific anxieties " from that experience and, so they reached out to a dev who works with the service day in and day out to review their plan "Hey, does this look good to you?". The dev asked, "doesn't this kill the workaround for the broken caches?". P1 was surprised, "Whoa, hold on, what workaround?" It turns out that there is an occasional problem in the service that results in caches being constructed incorrectly and it can result in a complete outage for the particular region where the service is deployed. Because the service is a "forked copy of an open-source project", it isn't touched very frequently and is now 3 years old. While it does occasionally get bug fixes, it is essentially unmaintained and no longer

comparable to the project it was forked from. If someone "tinkers" with it, they may unintentionally trigger the problem with the build caches. P1 felt that it was important to know where the state for a given service is kept and know who actually runs the service to understand "everything that can go wrong". In general, reviewing the logs for a service is a practice P1 recommends because he believes they offer a lot in terms of being able to debug issues, but had they deployed and it resulted in an outage, the logs would show a problem but it wouldn't be obvious that the cache construction was the problem. P1 didn't feel any particular sense of time pressure and would have relied on the "capability to roll back" if the change had rolled out. P1 felt they had "got lucky" and decided not to deploy until they could find a solution.

Participant 2

P2 is a SE and the original developer on a system that is running in Kubernetes. Now a tech lead, but still a primary developer, he was working with a junior engineer to debug a "very infrequent problem", so they added a TCPdump to the pod in an effort to get more information about the problem. P2 felt fortunate in that the system has "not really ever failed a deploy", but this change resulted in a problem that they detected a few days later during an unrelated outage. P2 wanted to see how the component was performing, and "looked at the graphs" and noticed it was not performing as expected "oh shit, this has not been working well!" He saw that the component had "restarted 400 times in 20 minutes ", and while several low severity alerts and alarms had fired, they were obscured by the overall volume of low severity alerts across all systems as they quickly filled the alert Slack channel.

P2 was surprised that the system was still functioning in spite of the issues It was "really cool that we were still able to process data". It turned out that the TCPdump process was missing a parameter about how much memory it could consume while it was bounded on the container level. The increased load due to the outage caused the process to consume more than its allowance, and thus it got terminated. P2 discovered the missing parameter "a capital B for

buffer size" in the documentation. It was "pretty obvious what was happening" and so the fix was "fairly straightforward". P2 was able to make the change and 17 minutes later, the fix rolled out to production. Since the outage was beginning to stabilize, P2 did not feel a lot of time pressure "it wasn't a terribly stressful situation".

P2 noted that they have a "git-ops" model in their deployment tooling which enables them to change a tag on a container and the change is rolled out within minutes. For P2, this was the "first really nice thing that's happened on the deployment front in years. P2 reported that he was "actually happy about this outage because they learned a lot of interesting stuff".

Participant 3

P3 is a SE who works on a service that is foundational to a large number of critical services. There are several dozen additional pipelines managed by individual teams that utilize this service. P3 describes that they have a very automated pipeline and when they release a new artifact, the downstream services will pick up the new artifact on their own schedule. About a week before the holiday code freeze, P3 was picking up his child from school when he was alerted about an outage in production, "it was unfortunate because I was just on my phone, in the parking lot of the school. In non-pandemic situations you would have had the whole day to just worry about the work". With everyone working from home, P3 noted that when they were in the office, it was easy to say, "hey can you look at this?", but now it was difficult to know if someone was available.

P3 then posted a message in Slack to coordinate the response with his co-workers. P3 described that "after reviewing the logs, it was "pretty obvious" that the service wasn't talking to the endpoint". For P3 the first step is to roll back to the previous known good configuration. P3 noted that this was only possible because of previous decisions to put "proper ops and safety practices" in place. The priority for P3 was to "just get the system back online". P3 discovered that the issue was a misconfigured endpoint coordinate in one of the libraries and that it had

taken down two critical services, necessitating an "emergency fix". His concern was that even though they had an override in place, that there was a possibility that a downstream service would get built with the bad library and cause an outage down the line.

P3 was particularly worried because they thought errors would be caught by their test environment but it didn't work in this case because the misconfiguration only existed in the production environment. The fix would require them to not only release a new base image which takes 2-3 hours, but additionally, they'd need to trigger all of the downstream pipelines to immediately pick up the new build. They had some scripts to do this but hadn't used them in a while because prior to this incident, the service had worked very well for the last year and a half" so they needed to write some custom scripting to enable the process.

P3 felt that in the "heat of the moment, we're not very good at telling people when there is a problem". Because the service P3 works on is so unique, the centralized teams who might be coordinating the incident response were not able to be particularly helpful, "it becomes troublesome because every time that you need help for something, you've got to go through and explain your entire deployment process and it's not well understood."

Participant 4

P4 is a Director of SRE, working on deploying a change in a monolithic application. He discovered that a feature was missing from a deployment one morning and determined that a hot fix would need to go out by 9:00 AM since it takes "about an hour or so to build and run tests". He felt the change was a specific "pinpoint fix without broad-based side effects" and so it presented as low risk. He described the necessary change to another engineer who built it, and then P4 reviewed the pull request and deployed it into the pipeline where a codified set of tests run. Once it was in production, he confirmed the change was working as expected.

For P4, it is important to build consistency in the pipelines otherwise they can "break in unique ways". Using the same pipelines and a combination of peer review, unit tests and QA give him

confidence in the deployment. P4 is always thinking about the risks of deployment and what he can do to manage the risks. Since P4 is working in a monolithic application, he is aware that his "change has the potential to break everything". For this type of application, there is a "quantifiable cost to building confidence through testing", and the more automated testing done, the longer it takes for the deployment to proceed. For P4, there is a constant balancing of the need to test and the need to get code out to production quickly, noting that "computers can run tests but problems are found by humans". P4 described that even with all of the confidence derived from the tests, that "QA is not production, staging is not production, production is production" and that there will always be emergent behavior that may arise only in production. For P4, "production is a unique place that you cannot fully replicate".

Participant 5

P5 has been a SE for about 10 years and is pairing with a junior engineer to deploy a change to an "old school" application that is over 10 years old. The application has experienced a failure in a critical finance function and they had about a month to develop a plan to replace it. P5 felt some pressure to ensure the change was delivered to the staging environment before the end of the year, along with the pressures of the holidays. He wanted to take time off as "it's been a really hard year". The application was running in AWS and they had a staging environment but didn't "have the usual CI/CD system that would help deliver features automatically" and so it was a very manual deployment, just "SSH in and deploy". Before proceeding, as a precaution P5 created an AMI "just in case I do something very wrong". On Friday, December 18th, they began their deployment but it didn't go as expected. Because the service was "antiquated" it "failed miserably". P5 prepared a checklist because, without a pipeline, he knows there are "quite a few things that can go wrong". Through a process of elimination, P5 determined that they had inadvertently introduced an invalid syntax into the project and so they changed it back to an earlier style of JavaScript that was compatible. The junior engineer was working on the code in an IDE that may warn of syntax errors, but because it was technically valid JavaScript, there was

no warning. Once they corrected this, the deployment was able to proceed without further difficulties.

While it wasn't possible in this situation, P5 feels it's important to deliver small changes "steadily over time" in an automated way instead of "big bang releases" because if it fails, it's "chaos all over the place". P5 advises his teams to be more consistent in delivering software in this manner so that they wouldn't be "dependent on one person who knows the exact shell script to run". For P5, this approach "de-risks the process". P5 described that in contrast, this situation was difficult because it was "spaghetti code, all intertwined" and that they "can't risk this thing dying". For P5, it's important that infrastructure should be treated like "cattle, not pets" meaning being able to lose a particular instance because you can simply spin up another instance to replace it.

Participant 6

P6 is a SE and was working on a feature for their deployment automation that had become quite important to deliver before the end of the year. He was working in tandem with a co-worker to write a thorough description of the desired behavior. P6 described needing to build a "clear mental picture" to go from understanding the experience from the users' perspective to understanding how he would implement it from a technical perspective. Because they have a "git-ops" style workflow, the automation looks for new commits to the main branch, builds an artifact, and then begins certain tests that allow it to automatically be promoted to the next environment. P6 committed the change and it rolled out to the test environment and came up healthy so it proceeded automatically to the staging environment. They have automated testing to detect if schema changes are correct, but changes that involve real data may not be caught when there's a "peculiar condition that only exists in production." P6's change was to an empty table in test, so the migration was successful but since the database in staging has data, the migration didn't work. P6 is a strong proponent of automated tests, noting that in this case the unit tests "looked great" so he didn't think it would be necessary to copy down the real data

locally. P6 explained that it can be difficult to fully replicate conditions in the infrastructure locally because "the configurations and interdependencies between applications can be complex and so it's easier to push to the test environment and hot fix there."

P6 discovered another issue and made a "really straightforward change" and deployed it to test and staging, but unknown to him at the time there was a typo so the change only partially applied, and when it failed it had not rolled back entirely, leaving the column that had been added. P6 thought he needed to delete a row but the row had actually never been written which led to him deleting the "wrong row". P6 corrected the problems and asked a co-worker to review his pull request, but then another error occurred "I was chasing my tail for a while, thinking that there was a problem with time zones in the database".

In the past, P6 had experienced a problem where the database would attempt to correct for the time zone when it was already correct and shouldn't be adjusted. Thinking he was seeing "wonky values", P6 noted that it "sent me down a rabbit hole that turned out to be nothing". After multiple days of troubleshooting and working to resolve the problems, it was finally working and ready to be promoted, but when P6 was prompted in Slack "do you want to push this out to production?", he made the decision to delay the deployment noting "there's no reason to give myself a headache if there was yet another thing wrong. So I decided to just call it a day."

Because the feature was so important, he wanted to ensure he'd resolved all of the problems as he "didn't want to be the person who breaks the environment and then goes on vacation."

P6's experience made him wonder if there was a way to "automate something that would catch conditions that are dependent on data existing in the production tables." It seemed to P6 that they were "always chasing the last problem and never catch the one that's coming."

P6 felt a fair amount of time pressure due to the holidays approaching, kids being out of school, and wanting to take some time off. Making things more challenging, P6 was having a lot of repairs done at home which made them have to "climb in and out of windows to get to the

bedrooms and bathrooms". They didn't want to stay in a hotel due to COVID-19 risks so it was "...not exactly the ideal working environment with lots of noise and varnish fumes in the house."

Participant 7

P7 is a SE working in a large scale cloud system. Previously in the medical field, P7 worries that he's killed far more people working in software than he ever did in the medical field. With the scale of the systems he works on now, if he makes a mistake and takes down the wrong cluster, there's going to be an amplification factor and might take out a life safety system such as a hospital.

P7 joins a daily meeting where they discuss any planned deployments for that day. "Whenever two things are being released, they may conflict in subtle or not so subtle ways and blow up in your face" so he syncs up with that person to determine:

Do I have a dependency? If the answer is yes, we discuss who's going first. Even if the changes have been tested in integration environments, when done in a certain order of operations, it will fail. Knowing that two things interact, it's just a risk management thing, if two things are deploying at the same time it's a giant Rube Goldberg of a race condition. I can't promise that the timing won't go wrong.

For P7, it's not worth it to manage it through automation because "it feels like a technical answer to a human problem." He recalls situations, where people have "set up Jenkins locks that prevent other jobs from running until the job, is complete, but that many times one of them fails to release the lock and now everybody is stuck. If a change is important, a human should be shepherding it."

The deployment experience for P7 was to a certificate as part of a phased rollout to move from static certs to an automatic rotation mechanism right before Christmas. The process prompted him to upload a certificate and offered the choice to use one that is stored locally on his laptop or

to select one that would be synchronized with the key management system that permits auto-rotation.

P7 generated the certificate request and since they are all versioned with a static SHA, he was able to utilize the same name thus “avoiding retooling a bunch of things.” He noticed that there was no indication as to whether the existing certificate was static or synced, so he checked in with his co-worker who said "Oh yeah, I just uploaded the cert. The key management synchronization wasn't working for me, so I just didn't do it". P7 proceeded under the impression that he was generating a certificate that would utilize the synchronization process and that it was replacing a static certificate. For the phased rollout, P7 pushed the change to canary and imported the certificate from the key management service and it rotated to the new certificate. Because the thumbprints changed to the values he expected, P7 thought it had picked up the new certificate from the correct location. The “metrics all looked good and there were no increased logging errors” So he thought “mission accomplished!”

At this point, P7 decided that because of the holiday code freeze and that the feature wasn't due until the end of January, he would not proceed further and to “let this bake in canary for two weeks.” He expected to come back in January and if everything looked good then he would resume. Later that day, P7 went to visit a friend that recently had surgery. During the visit, P7 received pings on his phone from a co-worker who said, “I just saw this error message about a bad certificate, I thought you didn't ship to that region?” P7 thought “that's not good, I hadn't deployed to that region yet, it should still be using a static cert.” At this point, P7 realized that it turned out every region was actually configured to use the key management service and no one knew this. He recalled again that there were zero visual indicators about whether it was synced or not. P7 asked his co-worker to start rotating certs to all regions, and he started driving home. He arrived home and his co-worker was working from the top of the list so P7 started from the Ws and said, "all right, dude, we'll meet in the middle". P7 realized that the cert he had requested earlier was not signed by an external certificate authority and so he and his co-worker

had to get a certificate that was properly trusted. P7 thought the non-trusted cert “should have been harmless. It’s a time bomb but it should have gone off in canary, giving us time to use a known good certificate when we were ready for the rollout.”

For P7, this was a surprising experience that “bit [him] pretty good” and was one of those “cascading failures where while we’re investigating it, another region tips over on you.” Also frustrating to P7 was that it was a service they wanted to deprecate as it was masking problems that had supposedly been fixed. When P7 attempted to bypass the service, problems ensued and he thought “oh no, you guys didn’t fix your shit, you didn’t fix anything!” and then turned the service back on.

Participant 8

P8 is a SRE who works with large-scale infrastructure and databases. P8 has been an SRE for a long time and was a systems administrator before that. He knows the ins-and-outs of the deployment tooling at work and knows exactly how breaks happen. Certain configuration changes can be “dangerous” in P8s experience. He explains that “code and config live in separate repos and use different products, ...although we are migrating in between which is an interesting transition [long pause] ...” For a significant migration happening between two critical components of code and config, P8s use of the word “interesting”, the intonation, pauses, and the subtle sarcasm in the air gave the impression that the transition isn’t going as well, and he is imagining a clear failure event in the future.

P8s expertise with the systems allowed him to write a quick script that logs in to every server to revert a config gone wrong - an emergency workaround necessary when automatic rollback mechanisms are not available or working as expected. The phrase “I wrote a quick script” to get out of the bind suggests expertise and experience in the manual actions to be done to put the system in the desired system state.

P8 described that “today, config changes go through a pipeline, but there’s a new system that a/b test configs, bypassing the pipeline. “ P8 has used the system just once or twice. This new system rarely fails:

“999 times out of 1000, nothing goes wrong. But sometimes the system goes pretty wrong and debugging is difficult. While these a/b tests are technically considered under the category of a deployment, “it doesn’t go through the pipeline, it’s just a config switch. These a/b test pull requests are opaque on what actions will be taken on the system. Tasked with approving thirty (30) or more of these a day, the changes are tiny, but unintelligible: n,1000 or B1206. P8 thought, “what is an n,1000? How am I supposed to know what this is actually doing?” and contacted the developers for answers.

P8s experience with this new method for testing configs brought up a war story of config testing gone wrong where an unbounded experiment suddenly going from 20 calls to suddenly calling the entire database. I was surprised to hear P8 say it was “just a config switch.” On one hand, P8 says that it’s essentially on / off button, but when it flips on, A) P8 has to decide to approve or deny it, and B) “it’s a hard process that can be dangerous, we can’t actually see what’s changing.”

P8 described a multitude of ways he warns the system can break. Exhibiting a depth of technical knowledge with an example of a past issue that brought them that knowledge. P8s expertise is not only in technical skills, but also in his knowledge of organizational history, describing his employers strict postmortem process dynamics. Postmortems are a “nerve wracking and fraught experience, and the meetings are filled with corporate politics.” P8 wonders “if it’s in-fighting or the idea of “be an owner.”” This seems to imply that there is a right set of behaviors and actions to exhibit a good model of service ownership.

A centralized team runs postmortems for incidents of medium severity and a Project or Product Manager (PM) is assigned to track action items and provide reports to management, and upper

management. The hierarchy of order is strictly maintained. PMs designed the entire postmortem process and tightly control its outputs. There are an explicit set of questions in the postmortem, including: “How long did it take?” usually followed quickly by “Why did it take so long?”

P8s rich description of coming to a meeting with directors after an incident is eye-opening: “We are the very deep back end of a system; everyone is dependent on us, so if something happens, everyone says You need to explain what happened because it hurt us.”

Participant 9

Participant 9 is a senior SE and was on call during the holidays where he described dealing with two incidents – “oddities that happen around the holidays.” P9 received a page about a traffic spike on Christmas evening. They described working with a colleague and feeling surprised and puzzled by the sudden increase in traffic, as they were not expecting it. “What is going on? Is this a DDoS? We were trying to figure out why this was happening.” They and their counterpart worked to try to identify the source of the traffic and determine if there was a problem with their platform, feeling a sense of responsibility to find a solution. “One of my colleagues who worked for Amazon has a famous phrase he likes to use: “What's the meantime to innocence? How quickly can you show that it's not a problem with your system?”

P9 described feeling frustrated by the lack of information, and the difficulty in identifying the source of the issue. Learning about this incident revealed a failure mode they didn't know about previously:

It highlighted some technical limitations of how that particular component was implemented. It shouldn't have been possible for us to lose these messages, but in this particular failure mode when we started to send a message out, it was marked as done, but if it didn't complete it was backed up on an internal queue. That was the only place that lived at that point. There was more tension in that because we have this resource that we cannot destroy. We want our infrastructure

to allow us to restart it, move load somewhere else, etc. But this was one fragile little instance that we couldn't break because we would then lose data which we really try not to do.

Participant 10

P10 discussed their experience with deploying a change into production at their new job. In the process of decoupling a large PHP monolith into smaller services and migrating to the cloud, P10 mentioned that the organization had challenges with observability and understanding what was happening in production. In order to gain more insight into the behavior of the system, P10 proposed a change to SQL to help track where queries came from and commented that “my previous job had observability gaps, my new job there's observability mesas, and it's all gaps with these occasional mesas of visibility.” P10 also mentioned that because it was their first change to the codebase, they paired with a staff engineer to implement the change. First they write up descriptions of the change, describing the problem, scope, and impact of any change they want to make, using “git commits or JIRA tickets to document the change. There is a post-deploy tool that gathers those in one place to see info for all commits.” P10 described the system as a shared monolith where they batch up deployments, and wait for a time where there are no new commits, then deployment starts.

“You don't know exactly when your deploy is going to be ready to go. We wrote the code, we tested it, I committed it. Then I just waited, I had to keep refreshing the page. There's a slack channel to know when the deploy is going but I felt really annoyed and uncomfortable. The not knowing, the waiting, the uncertainty was really frustrating. In the past I've done deployments where I knew exactly when it was going out. I was very agitated and frustrated. Finally the deployment started, it went to the canary server. I used DataDog to look at

impact to CPU and I/O. Wondered if it looked normal, but I didn't know what normal looked like, so I relied on eyes and expertise of my staff engineer."

P10 said that they "felt confident this was going to be a simple, easy deploy because I'd done this before in other systems." After pushing their change, P10 notified the build team who "agreed to push the button" and deploy started. I asked P10 about how they monitor or watch the progress of the deployment and he explained that he did a " cursory check, watched it for five minutes and assumed it was good. I felt more comfortable walking away because I'd communicated (written up) about the change and it didn't seem that risky." When I asked P10 about how their deployment pipeline worked, he described having an "automated canary timer. If no one pushes the stop button and the canaries don't blow up catastrophically, the Canary cluster automatically promotes the deploy 30 minutes later."

Participant 11

P11 mentioned that they were the only person on the operations team at the time and that they had returned from vacation. As part of his normal daily routine, he logs into Grafana and sees a bunch of requests being rejected in CosmosDB. A specific call has been failing for 2-3 days. P11 is thinking and asking himself: "Is it really a problem, is something actually wrong or is it a transient issue with Cosmos?" P12 confirmed the problem was real because "nobody was on call to look at it and no one else noticed it". P11 considered the possibility that there had been a recent deployment or change to the system that might have caused the issue but found that there had been no changes made in the previous week. P11 wonders why this could happen. He thinks to himself, "this is not a 429 error which would indicate a scalability problem; it's a retry later situation." The specific error was 403 "not allowed to add any more records into the database." The error was foreign to P11; he'd never seen it before. So he asked clarifying questions, almost as if he was talking to the system itself: What are you partitioning? It doesn't make sense to partition this data. P11 described the SE's mental model: Cosmos DB has 2 config options: 1)

with partition schema. 2) without partition schema. SE thought that without partition schema (2) you could store infinite records. Because P11 didn't know it had a partition, he thought it was unpartitioned and thus couldn't exceed a limit. He didn't remember anyone talking about partitioning, so he thinks, 'no, it's probably not partitioned, so what is wrong?'

P11 thinks about the age of the system (three years), and recalls that “originally, Cosmos DB could be configured without schema and partitions--ie. single partitions where you could store up to 20 gigs. It was a system limitation buried deep in the documentation that no one bothered to read.” At this point P11 has 2 options: Given that they've been losing data for 2 days, P11 realizes this is not transactional data, which is not a critical path but still important. For P11, it was 9:00 AM, but it was 5:30 AM for his co-workers. He could call them but worried because of how early it was for them, and he knew that fixing it would require re-partitioning, and it's a big change to insert a partition scheme in the middle of the incident. P11 is at a crossroads, he cannot decide to introduce this partition on his own, because a lot would have to be done, so he rules out doing this in the moment. His options to further troubleshoot: 1) Calling others and 2) Check if they were losing data. P11 realizes they have 7 days of retention, so he hypothesizes that the event hub has 7 days of retention. He feels they “got lucky because 7 days of data is still there.” P11 believes they would be able to repopulate the data from the logs to “string enough information together.” P11 described realizing how serious the problem was and how bad it could have been if they didn't have the retention policy: "Even if I partitioned right now, if I do not have the old data, it's gone, it's gone! Unless I have this retention, it's just all gone. I could have had ten options at this moment, none of them would have mattered. None of them would have solved this, the data would have vanished."

P11 identifies that the problem is occurring in a specific table, and since other tables are functioning well, P11 could rule out “noisy neighbors”. To remediate the problem, they had to make a sacrifice decision to reduce the database size forcefully by dropping older records that

were not as frequently accessed. Because they had a backup, P11 felt confident they could get the records back later after the size issue was properly resolved.

P11 is now determined to communicate to the customers. P11 wanted to call his manager in Europe, but he can't call because of the time difference. Instead P11 contacts product management, and explains that if they focused on communication, he could work on fixing the problem. "Obviously I was more serious, but I was joking as I thought "do you want me to sit here and write a message or do you want me to get back to fixing it?". I said "if you want me to do both, I'll probably mess it up. I would write more technical stuff than necessary for what the customer understands. They're not going to understand 'Your system will be eventually consistent.'" P11 explains to the product manager that data isn't there now, they won't see it, but that it is not lost, and they will see it eventually. P11 characterizes this as expectation management.

Two of the senior engineers were on vacation, P11 reports feeling they are "short staffed". This comment is important, because P11 needs these people to aid / lead in decision making on the next step. He doesn't want to decide on his own and felt that SEs would be able to make the decision confidently. "Because of COVID, nobody is actually on vacation" so he felt they "got lucky" in that aspect as he was able to reach the SEs, and then it was "all hands on deck".

Participant 12

P12 experienced a problem with communication and documentation when deploying a change to their production databases.

We had tried right before Halloween to actually do this change. But it failed and we had to revert because we couldn't handle the load. So last year, our big focus was how do we test load accurately? And the answer was by actually testing it in production because we didn't have the budget to completely replicate all of the traffic to an additional environment.

They were in the process of changing from one database to another and had a complicated process for configuring app servers to specific database servers. P12 and another person were supposed to follow this process and update a document every time they did so, but the other person did not update the document.

Because we were trying to do things fast, we had a team working agreement that we were just going to edit straight to master instead of doing a pull request.

Which I didn't like, but I was overruled. Even though all of us have administrator privileges to merge a pull request without approval, we were just pushing. We have another team member who likes directly writing to master specifically only for documentation changes. He just wants to make what he considers trivial changes directly to master instead of opening a pull request and then merging.

This led to confusion and P12 having to redo the process, as they were not sure which steps had been completed. P12 described what happened after that, and how a process they use to help themselves ended up being the reason a serious error was caught:

That's when one of the database guys, actually noticed that we had a typo in the automation that we were doing this with that that would completely invalidate all of our settings for failovers. What we have been doing for almost a month at that point was pointing the failover at something that didnt exist. So the only reason he found it was because I had submitted a pull request instead of committing directly to master like we had agreed to do. So it could have not happened so easily, but he saw it. I have ADHD and it's really way easier for me to keep track of things. For me, I'm doing something that's going to work with my workflow. With a pull request, I only have to be careful once. On branches with the expectation of squash merging, I still try to write commit messages that are going

to help me later. If something happens that distracts me and I have to come back and I look at the commit messages to see what I had been doing.

P12 described how being on-call impacted their sleep:

I also tend to wake up a lot for pages that weren't real. It doesn't happen to most people, but there's at least one other person on my team who also, while on call, will dream that they got a page, that they acknowledged it. If you acknowledge the page, it won't keep paging you even if the problem [still exists]. So if you acknowledge the page, you go back to sleep, there will be a problem. And I'll check my phone, look at the app and it'll be like, "no, you didn't get paged" and I'll go back to sleep. But once in awhile, it's a bigger deal. Sometimes I will knock my phone off the nightstand, it'll fall under the bed or something and bounce, and I'll have to like actually get out of bed and stand up. If I get a page that takes longer, more like 45 minutes, maybe even an hour, those are the hardest ones to fall back asleep from... I do think I get worse at handling pages as I've handled more of them in the single night. I noticed a lot of my silliest mistakes happen when I've already handled something else before and it's just been quite a night already. The first time I was directly responsible for production outage. It was 7:00 AM when I got paged and I usually wake up around nine or so, and I had gotten paged several times that night. I didn't directly connect that what I had done to try to fix the page is what had caused the outage. I thought that a different problem was responsible because of a specific symptom I was seeing. But from that, I'm now very aware that it's more likely to be whatever the most recent change is, even if the symptoms wouldn't seem to suggest it.

P12 described how they experience post-incident reviews:

after every incident like this, we have a post event or retrospective where we go over everything that, if it happened again, we would like to do differently next time. We are supposed to follow a blameless process, but like a lot of the time people self-blame. In these in, you can't really shut it down that much because frankly there are very causal events. I'm not the only one who in the PIR can't really let go of. I know it was because of what I did. But we do try to make as many changes as we think will help to our processes, to our documentation, to our automation, to our monitoring and alerting, to our working agreements, whatever is necessary because we always want to not have another outage.

Participant 13

P13 describes their experience deploying a change to a production environment for a proprietary software product that few people want to deal with “you inherit these things and they're hard to get rid of, they're kinda vampires, you open the front door and they get in and you can't ask them to leave.” P12 said that management sees value in the tools but engineers avoid working on them due to the “sheer risk” involved.

Risk is the big thing that I always think about when I'm doing a deploy, especially to a service like this. Because not only do you not want to break anything you want to, you depend on any sort of backups or snapshots or database dumps that you might have being an integral part to it. And since it's not very well automated or automated at all in this case, it's a very manual process, which is prone to human error. So you don't just have the stress of the company on your shoulders. You've got the stress of paying attention to what you're doing, and you're put into the stress of having to do this late at night.

They mention feeling uncertain about the process, as the product is not widely used and information about it is hard to come by,

you always go into it with this very uncertain feeling of how long your night's going to go... even though allegedly it is clustering, it doesn't actually do what it says it does in the box. So you really can't gauge how long it's going to take what the behavior will be like during it coming back up. Is it going to re index the database is going to do some big change? And it gives you certainty into the probable steps that you'll need to complete, but it doesn't really give you any confidence past those steps into what you're actually gonna be deploying it into... there's yet another level of risk, which is if this goes wrong, are we going to be able to knowledgeably recover from it if we've not truly tested it? And this seems to be very difficult to communicate to people above you. I tried desperately to, but I think this is why a lot of engineers get very afraid of these things and for good reason, especially if they can't go on Google and find much of anything,

P13 described how the process worked

“I always do find value in doing a test and conventional deploy and upgrade. And if it doesn't go well, reverting back, cause it gives me a chance to test those functionalities before we actually get there. But that becomes an arduous and manual task potentially fraught with human error... The checklist is written down in that we document the procedures of how to do the thing, but we don't necessarily document how to prepare ourselves or how to deal with the social aspects of it. If it goes wrong, what teams do you need to talk to? What is the actual urgency and level of response that is required for this? What is the actual value here? Because the default value is everything must work as predicted all the time. If you don't assign a value to that or understand what the actual impact is

as an engineer, you're just going to have to go into it, possibly half asleep, cause you're on call and you got woke up the night before, or maybe it's a very stressful day or maybe something happened in your personal life or you ate something bad for dinner. I've definitely done my share of deploys when I'm not feeling well.

P13 described the stress they experienced as a result of having to do the work at night and the risk that something could go wrong:

When night comes it gets a little stressful because, you're tired, all the other coworkers and anyone who can provide you technical or moral support is probably not online. You really don't want to wake them up, especially if they're on the East coast and you're on the West coast because it's extra middle of the night for them. We have a change window that starts about midnight for this product. So it can go all night which is awful. It is supposed to be back up by, 9:00 AM Eastern time. So you have a limited window to get this done. So once again, another reason that people really don't want to be deploying this, or even adding datasets to it themselves, because of this risk, even if they're just importing a raw data set into it, there's things that can happen to unbalance the cluster and cause it to go into a spiral of unknown conditions.

After doing the upgrade, P12 was “facing the screen of all these services starting up and they look like an old fashioned mainframe from like a seventies movie where you've got all these little green dots.” When the problem occurred, P12 realized they needed help.

“I'm going to have to wake up somebody that either knows more than me and or someone who has access to the backups. Hopefully there are backups. So basically I paged the person I had inherited this monster from. He had changed teams and I had to wake the poor guy up.”

After the incident, P13 described the post-mortem experience:

I invited the former owner of this project and he agreed with me. He supported me, which was really a good feeling. You never know how these things are gonna go. It's easier to pass blame than it is to provide confidence and assistance and honest opinion to someone else - especially when gender and things like that are considered. It gave a lot of credence to what I was saying that I probably wouldn't have had on my own - being queer and being female in a primarily white male dad environment that exists in the Midwest, certainly all stereotypes apply.

Participant 14

P14 described an experience of deploying a change to a production environment. This change involved updating the backend database and rolling out new container versions in different regions. However, when P14 ran the necessary database query, it did not work as expected and caused issues with particular values in the database. P14 had to “involve the chief architect and the product architect to troubleshoot the issue and fix it.” This process “took a full day, rather than the expected couple of hours.” P14 mentioned that they had to go through all the tables that the change would affect and make edits to them. Though the retrospective, P14 asked about the lack of integration environment:

Why are we running production changes for the first time? It's prone to errors and you should have a way to test these changes. And that sort of set up, like “Why is our staging environment, the testing environment?” Instead, we went on to build an integration environment where the developers who are working on different micro services, meet there rather than staging. Make staging a replica of production.

P14 spoke about the difficulty of self-care during incident response: “Not a whole lot of self-care scenario because when you're the point person doing the change? It's just me. I'm like, “Hey, I need to get this right, fix what's happening, and not think of anything else.”

Participant 15

P15 is an engineering manager responsible for managing and investigating an issue with the SSL certificate for a customer-facing e-commerce website with a high volume of traffic. The issue occurred on December 15th and was first noticed when the search engine on the website did not return results in a particular country. P15 was the first responder and initially thought it was a front-end bug, but later discovered that the existing SSL certificate had been revoked. P15 struggled to get in touch with team members in another country to resolve the issue, and it was not fully resolved until around lunchtime. P15 mentioned that they were not officially on call and had not felt prepared for certificate issues but had a lot of experience with critical incidents. P15 also mentioned feeling stressed because they were at home with a sick child and were worried about making the situation worse.