Post-Incident Action Items: Crossroads of Requirements Engineering and Software Evolution

Matt Pope
mattpope@byu.edu
Brigham Young University
Provo, UT, USA

Jonathan Sillito
sillito@byu.edu
Brigham Young University
Provo, UT, USA

ABSTRACT

The evolution of a socio-technical system is an iterative and collaborative process, in part driven by failures of the system. When a system failure is sufficiently severe, an organization may conduct an incident analysis to learn from the failure and choose post-incident action items to co-evolve the technical and human aspects of the system. Developing a better understanding of this failure-driven software evolution processes is the goal of our ongoing research project. As a first step, we have collected publicly published incident reports, extracted and analyzed 104 post-incident action items in those reports. The focus of our analysis has been on: (1) what motivates those actions, (2) what changes are being made to systems, (3) which parts of the systems are being changed, and (4) the goals of the changes. In this paper we report on preliminary findings from this analysis. Another aspect of this software evolution process that has not yet been studied (as far as we have been able to determine), is the relationship between incident analysis (along with the work that follows the analysis) and requirements engineering. So in this paper, we also discuss how aspects of requirements engineering may be helpful in addressing challenges or open questions related to incident analysis.

ACM Reference Format:
Matt Pope and Jonathan Sillito. 2024. Post-Incident Action Items: Crossroads of Requirements Engineering and Software Evolution. In *Workshop on Multidisciplinary, Open, and RElevant Requirements Engineering (MO2RE 2024), April 16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3643666.3648578

1 INTRODUCTION

Requirements engineering (RE) covers a wide spectrum of activities across the broader software engineering life-cycle. Taken together they deal with "identifying, specifying, modeling, analyzing, and validating the needs and constraints of a system".[1] The way these activities happen varies significantly from organization to organization; some activities are upfront while others are iterative, some are abstract while others are concrete (perhaps bridging the *what* and the *how*). But all of them feed into the design and implementation and operation work of teams as they develop and evolve systems.

In our experience, one context in which some RE activities may occur (perhaps informally) is after an *incident*, an event in which the system experiences a failure such as an outage or degraded functionality or performance. After an incident is mitigated, organizations may conduct an incident or postmortem analysis of the incident and produce "a written record of [the] incident that details its impact, the actions taken to mitigate or resolve it, the root cause(s), and the follow-up actions taken to prevent the incident from recurring." [13] The written record which is produced from the analysis is commonly referred to as a postmortem or incident report (IR) [20] and the follow-up actions to be performed are commonly referred to as preventive actions, remedial actions, or just action items.

The relationship between failure-driven software evolution process and RE has not been studied from a research perspective. And in practice, RE techniques are often not well (or explicitly) used in the context of incident analysis, though we hypothesize that incident analysis and the work that follows would benefit from such techniques. To explore these ideas further, we have performed an analysis of 104 action items, which are the output of the incident analysis process. We identified these action items by reviewing 20 publicly available incident reports, and have performed an analysis using a grounded theory-based analytic process with a focus on characterizing the incident analysis process and understanding how it drives the system's evolution. For more details on our analysis, see section 3.

In this paper, we report on our preliminary analytic results and make the following contributions. We describe key findings relating to action items: action items are motivated by diverse triggers throughout the lifecycle of incidents (Section 4.1), the actions being taken both evolve the system immediately based on the analysis of the behavior of the system during the incident and also have the potential to lead to future evolution (Section 4.2), the actions evolve multiple parts of the *socio-technical system* rather than just the software (Section 4.3), and the goals of action items tend to be narrowly focused on safety, visibility, and process maturation (Section 4.4). We present three concepts which emerged from our analysis and which highlight how incident analysis already borrows important considerations found in RE activities (Section 4.5). We also include a discussion on how the treatment of incident analysis as an RE activity could address several challenges of incident analysis we discovered and improve how we evolve systems in response to failures (Section 5).

[1] https://mo2re.github.io/

MO2RE 2024, April 16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0569-4/24/04
https://doi.org/10.1145/3643666.3648578

2 BACKGROUND AND RELATED WORK

What follows is a brief discussion of related work that we build on in a number of overlapping areas, including software evolution, requirements engineering, and learning from failure. While we do not have space to say much about each, nor about all topics that we do build on, this section should provide important background on the lenses that have informed our work.

2.1 Learning From Failure

Incident response encompasses multiple engineering activities done in response to (significant) system failures, including internal postmortem analyses. One product of these analyses is an incident report which contains explanations of how and why the failure(s) occurred, how they were mitigated, and most importantly for our research, what changes are planned to prevent future failures. These changes are often situated in lessons which are learned formally and informally. The area of research and learning process is formally known as learning from incidents (LFI) [14] and has recently earned renewed attention in the area of software systems.[2]

Organizational learning includes aspects such as: observation, lesson identified (an observation which has been analyzed and validated), lesson learned (an identified lesson which was approved by decision makers), lesson implemented (the lesson being implemented and results of the implementation are verified), and best practice (the preferred actions in a specific type of situation to achieve an objective) [1]. Incident analysis is part of the (organizational) learning process [5, 6], producing new knowledge (lessons) from incidents and helping build "an organizational memory of what happened and why" [15].

2.2 Evolution in Socio-technical Systems

Some of the learning done during incident analysis is manifested in action items [13]; that is, actions taken "to fix, remediate or 'prevent' the incident in the future" [17]. Engineers and key individuals in the organization that owns the system are responsible for selecting action items, or in other words, are responsible for driving the system's evolution. Unsurprisingly, incident-driven evolution is often unplanned and unexpected. While some organizations have created best practices for selecting action items [13] through their own experience with incidents, we have yet to discover a standard, cross organizational approach. Incident analysis itself is a complex activity, and simple principles that ideally guide the selection of action items (like "what-you-find-is-what-you-fix") may not apply universally [12, 16].

"Software evolution represents just one aspect of the evolution of socio-technical systems" [8]. Socio-technical system theory considers the interrelated nature of an organizational system of joint human and software parts: through people, infrastructure, technology, process, goals, and culture [4]. Socio-technical systems theory proposes that evolution of the socio- and technical parts that does not consider the effect on the other will have limited effectiveness [21].

2.3 Requirements Engineering

Software evolution can be examined from the perspective of requirements engineering, founded upon the premise that evolution is best managed with reference to the requirements of a system [7]. This perspective leads us to examine three elements involved in the creation and maintenance of software: the domain assumptions the software will operate in, the requirements the software must meet, and the specification or detailed plan for implementing the software.

Requirements engineering is an iterative process [19] of refinement, involving interactions between these three elements. Post incident analysis is one contributor to this iterative process, as they present opportunities to reflect on and refine incorrect assumptions (about the domain, behavior of the system, etc), identify incomplete or missing requirements (whether functional or non-functional, business, user, performance, etc), and explore problematic parts of the specification or implementation of the system.

3 METHODOLOGY

Our analytic interest is in understanding how action items evolve systems and the stakeholder concerns they represent to help us as we characterize the post-incident requirements engineering work. To this end, we have analyzed publicly available incident reports, which document various failures and the action items taken in response to the failures. In our experience, incident reports typically represent careful analysis by engineers and managers, contain important contextual information relevant to our analytic interest, making them suitable for a grounded theory-based analysis. The unit of analysis for this research are the action items, and so far we have extracted 104 of them from 20 incident reports.

3.1 Data Collection

The Verica Open Incident Database (VOID)[3] is a "community contributed collection of software-related incident reports", housing over 10,000 categorized artifacts like tweets, status page updates, conference talks, media articles, and company postmortem IRs from 590 organizations. From VOID, we obtained a list of publicly available IRs which fit our criteria, which was that a candidate report had to 1) be categorized by VOID as an IR (instead of a status update, say), (2) be about a software failure, and (3) report on to an incident no earlier than 2015. We randomly sorted the list and, as an initial set, we have selected the first 25 from that randomized list and sequentially assigned each an ID, from 1 to 25. After reading each report, we discarded five for not being incident reports (IRs 1, 11, 16, 17, 23), since they did not meet the above selection criteria despite having been categorized in VOID in that manner.

From the remaining 20 accepted IRs, we identified 104 action items which were explicitly listed in each report. By explicit, we mean that the action items were called out as actions they intend to complete and not offhanded comments. We then reviewed the reports to identify a rich set of information relevant to our analytic interest about each action item. Specifically, for each action item we have collected the *event* which is the primary motivation for why the action item was added, the *action* which is the action that will be performed on the socio-technical system, the *target* which is

[2] https://www.learningfromincidents.io/

[3] https://www.thevoid.community/

Table 1: Example action items.

ID	Event	Action	Target	Goal
6-4	Incident responders restarted database system without verifying data integrity	Add a DBA (expert) to the change committee to evaluate failures before applications are restarted	Emergency change committee and procedures	Improve decision making during incident response
9-18	Enabling feature caused failure and was rolled back	Test and re-enable feature	Streaming subsystem	Carefully reintroduce performance improvement
14-7	Cluster manager promotion caused failures	Test more scenarios through fault injection	Testing practices and tools	Find failures in unanticipated scenarios
25-11	Responders' unfamiliarity with tools delayed status updates	Change engineering training curriculum	Incident response training and procedures	Timely customer communication during incident response

the part of the system being changed by the action, and *goal* which is the purpose of the action item. We have given each extracted action item (and the rich set of additional information included with each one) an ID made up of the ID of the IR and a sequence number, such as 6-4, which is the fourth action item in IR 6. Table 1 summarizes a few examples of our extracted data.

3.2 Data Analysis

We followed a grounded theory approach for our analysis, with two researchers (the authors of this paper) independently reviewing a subset of the incident reports and identifying the action item. They then came together to discuss and reach agreement on which action items to include/exclude and how to accurately capture the four properties (event, action, target, goal). We used an open coding approach to categorize the dimensions of the actions individually, creating an initial set of codes which were refined iteratively. Finally, we performed axial coding on the grouping of codes to help tune the coding scheme by interpreting and reflecting on their meanings. During each part of this analysis, we kept in mind that the purpose of the analysis was to characterize post-incident analysis, and focused on this aspect while selecting and refining our codes.

We selected four questions to aid our characterization of incident analysis. *What motivates these action items? What is being done? What parts of the system are changing? Why are these actions being performed?* We have also identified underlying concepts found during our coding by looking at the relationships between action items in a given incident and also across all the incidents. In the following section, we report on the results of this analysis. We have made the action items and coding available publicly [4].

3.3 Limitations

Our analysis is limited to what is reported in publicly published incident reports, which limits our analysis and the claims we can make in several important ways. First, we have no visibility into the range of actions that were considered (but possibly not selected) and we have only a limited view of how actions were chosen. Second, the publicly published actions list may not include all actions (or all details about the included actions), meaning our dataset may provide an incomplete view of the ways systems are evolved. Finally, we do not currently have insight into what happens after the report is published and work on the actions begins. Despite these limitations, we have developed some important insights, however as this research continues we will incorporate additional sources of data, such as interview data.

[4]https://github.com/BYU-SE/mo2re-action-items-crossroads-of-re-and-se

4 RESULTS

The purpose of this section is to characterize the incident analysis activity from the perspective of its action item output. We are reporting our preliminary findings in two parts. In the first four subsections, we report on answers to our aforementioned questions that we found in the events, actions, targets, and goals in our data set. Second, we discuss three concepts that have emerged from our data as important concerns to stakeholders after an incident. The understanding formed in this section will allow us to explore how *formal treatment* of incident analysis as an RE activity in Section 5.

4.1 What motivates these action items?

As mentioned above, we have identified the incident *event* that motivated each of the action items we have extracted. These events can be seen as inputs to the incident analysis process and they capture details about the scenario, the way the system behaved (both what went well and what did not), the actions of the responders and the consequences of those. We have categorized the events into three top-level categories capturing the relationship of the event to the incident:

(1) a trigger that began the incident,
(2) a part of the incident, or
(3) in response to part of the incident.

This categorization demonstrates how action items are selected in response to all parts of an incident's lifecycle from its inception to its lingering effects. We note that many of the events *could have* occurred different parts of that lifecycle, though we do not consider this in our categorization.

Before the incident describes the time when the socio-technical system was operating normally and the event describes a trigger, a

perturbation to the normal operation. Maintenance has risks when a human maintainer can act on incorrect data such as incomplete health metrics (7-5) even when "following the documented procedure" (IR 7) and these actions may have long-reaching effects like incidents lasting into peak hours (7-10, 7-11). Two incidents and eight action items had deployment-related events–when a defect was missed in the pipeline and was deployed (13-1, 13-2, 13-3, 13-5) and when old defective code was reintroduced after a crash and restart (21-1, 21-3, 21-5).

Unsurprisingly the largest motivation is failures from the incident, occurring as roughly half of the events. There are a wide variety of failures, ranging from components of all types failing under load (accounting for 21 of the 52) to disconnections between components to promoting the wrong node in a cluster. We consider so-called "bad behavior" such as thundering herds (10-1) and connection leaks (7-8) as failures, since it represents a likely deviation from intended behavior. Limits, such as resource thresholds for threads (several action items in incident 25) acted as tipping points in the incidents were also the motivation for some action items.

The response to a failure, which may even while the incident is still on-going in the case of automated actions like load-shedding (2-1, 2-2, 2-3), is another motivation for action items. In our dataset, many incident reports still demonstrate the human-driven nature of incident response–and human action was often an event. Human action is often slow such as when triaging (4-2, 9-7) or when time zones complicate timely responses (5-1, 5-2) but the actions humans take can also be slow, such as manual cache deployment (9-15), restarting (9-16, 25-5), and rolling back (4-3). Communication and coordination also proved to represent some pain-points (5-4, 12-1, 14-1). Lastly, we note that visibility is an important property to a socio-technical system and several events stemmed from missing visibility: faults weren't noticed until peak traffic (4-1), operators and responders were misled because metrics were lacking (20-5), and status reporting can be unclear and incomplete (14-3, 14-4).

We argue that the inputs to this process (the events and also other inputs outside of events such as the environment the system is operating in, the requirements it must fulfill, etc) provide the basis for exploring ideas for evolution that then lead to action items. In general, we see that events captured "what went wrong" during all parts of an incident and the response to it, and there is evolution occurring to take specific action against those "wrongs".

4.2 What is being done?

Here we describe the types of actions of the action items in order to characterize the work that occurs (or at least is intended to occur) after the incident analysis process. The types of actions in our data set can be categorized by their effect on the system, and we have created three larger abstractions to capture that effect which we have named simply formative, evolutionary, and other.

(1) Formative actions are *preparative* and *prioritizing*,
(2) Evolutionary actions are *additive*, *subtractive*, *changing*, and
(3) Other actions

Formative actions are typically preparative in that they represent plans for evolutionary actions for the socio-technical system but don't evolve it. Evolutionary actions can be additive in that they add something which was not formally present, subtractive in that they remove something which was formerly present, and changing in that are they both additive and subtractive such as replacing a component. We also included an other categorization which captures changes like one-off tasks that don't fit into these other categories.

Formative actions include preparative actions largely are those which are about investigation and planning. Investigative actions may seek to first identify problems which have yet to be explored in sufficient detail (7-7, 7-8, 9-16), identify options available to engineers (21-4), evaluate or reevaluate those options (10-4), or perform a review on configurations (18-5), decisions (20-4), and algorithms (18-1). Planning actions included plans for preemptive scaling for known seasonal traffic (18-2) and plans for when to unfreeze deployments (13-2). Adjusting the priority of work, like accelerating a project (25-8), also can be considered formative in that the evolution itself is what is changing. Formative actions also include other actions like developing coordination processes (5-4) and reinforcing development methodology (2-1).

Additive evolutionary actions include adding components, documentation, features, maintenance procedures, monitoring and logging, testing practices, and staff as well as increasing the frequency of existing practices like testing. For example, adding a new subsystem to allow DNS caching (8-4) or adding fine-grained alarming (25-2). We found subtractive evolutionary actions such as deleting obsolete data (9-13), reducing the health checks that occur as an activation delay (7-2), decoupling a circular dependency (9-5). We have also included reducing friction for feature flags by making them easier to code into the systems (2-2), reducing the frequency of queries (10-3), and reducing call frequency for requests (24-3) as subtractive actions. Changing actions include replacement (of systems, tools, deployment architecture), scaling (e.g., hardware), fixing (e.g., configurations, defects), and improving (e.g., runbooks, decision making matrices, failover mechanisms).

4.3 Which parts of the system are changing?

The target of the action items refers to the part of the system being changed. Since the system being changed is really a socio-technical system, we have categorized the target into the six components of socio-technical system theory. These are, with their associated counts of action items in each component,

(1) Technology (67 action items), or the software that comprises the systems,
(2) Processes/Procedures (26 action items) used by humans on the system,
(3) People (5 action items),
(4) Infrastructure (3 action items), or the physical hardware and aspects of the system,
(5) Goals (3 action items), and
(6) Culture (0 action items).

We have divided the most common category (Technology) into three subcategories: architecture, software component, and infrastructure. Nine Technology architecture targets include changes to the way a system was deployed (e.g. how a system is partitioned or if it is distributed). 47 Technology software component targets were actions taken on software that was central to the system(s) involved in the incident such as the application, caching layer, cron

jobs, storage systems, services, and test suites at various levels such as configuration, source code, interaction mechanisms, queries, etc. Lastly, 11 Technology infrastructure targets were changes to deployment systems, networking infrastructure, verification mechanisms, OS configuration, and hardware instance types.

The Processes/Procedures component includes those obviously related to incident response such as emergency change committee and related procedures, customer communication and related procedures, documentation, escalation tooling and processes, rollback and recovery mechanisms, and training. Other targets in this category are maintenance procedures (7-10, 7-11, 19-3), testing activities and practices (9-2, 12-2, 14-7, 18-4), operational documentation like runbooks and guides (20-4), organizational and coordination processes (5-4), and even user engagement processes (2-4). Changes to development and engineering methodology (2-1) fall into this category.

The People component has organization targets, namely its staffing (5-2, 9-10), customer support (5-1), grouping (5-3), and organization (19-4). The Infrastructure component has data center targets: its architecture (9-8), its network cabling (19-1), and physical documentation like labels (19-2). The Goals component has three identical targets: the engineering roadmap (9-9, 14-4, 25-8).

In our data, we see that the incident analysis process considers designing and refining requirements, domain assumptions, specifications, and the implementations for most parts of the sociotechnical system and this contrast a naive position that only the software element of the system is considered for changes after an incident.

4.4 Why are these actions being performed?

Above we have described the kinds of changes that result from the incident analysis process our work aims to characterize. Here we describe the goals of that work. In our data, we have found that many improvements relate to:

(1) safety to tolerate or prevent a failure or event,
(2) visibility to detect and react to a failure or event, or
(3) maturing incident response and development activities.

Safety goals include human aspects such as improving decision making (6-4, 7-11), creating trust in existing safety mechanisms (13-5), and engaging experts more quickly (12-1). In software, these include improving performance, preventing specific failure scenarios, adding safety margins by providing headroom, and isolating failure effects (25-7). Several goals related to tolerating failure, namely sustained unavailability of a dependency (25-9) and loss of an entire data center (14-5).

Visibility goals covered changes in several areas of software development such as ensuring visibility into capabilities like feature flags (2-3), detecting and catching problems "before they become user facing" (4-1) or earlier in the release pipeline (21-7), and improving the accuracy of health metrics (7-5). Visibility can be used to proactively react (i.e. hopefully preventing an incident) such as earlier notification of approaching failure (9-6) and awareness of changes made in dependencies (5-3). Visibility goals also seek to change incident response - to allow engineers to more quickly identify problems (9-5, 20-5) and understand behavior and performance (9-7) during debugging for instance.

In some cases, the goal of the action items is to mature a process or activity so that it becomes less brittle, and this is often in response to a pain point exposed in the incident, and done to prevent the pain point from being felt in the future. Timeliness and communication tended to have room for improvement. Time-oriented goals considered reducing time to recover (avoiding using invalid cache data, minimizing cold start time, quicker access to documentation, speeding up incident response mechanisms), reducing time to resolve, reducing time for triage, and timely communication (with clear notifications, around the clock support, better escalation) between engineers and also to customers. Communication goals encompassed including more relevant information (7-12), additional granularity (14-3), and maintaining an open channel where previously there was none (2-4).

Lastly, there were some development goals relating to completing an existing effort more quickly (9-9, 9-10, 14-4, 25-8), fixing a problem permanently (21-3), and carefully reintroducing a feature (9-18).

4.5 Concepts

We present three concepts which emerge from the data, and from at least one of the four questions we asked in the previous subsections. We argue that these concepts are important, cross-cutting ideas that represent concerns that stakeholders including incident responders, engineers of the system, and users hold. The discussion here relates to (briefly) presenting these concepts and we will say more later about how these concepts are important to treating incident analysis process as an RE activity.

4.5.1 Similarity. We have repeatedly seen IRs include statements suggesting that the intent of doing incident response, action items, and more generally, learning from incidents, is to prevent a reoccurrence of the incident (or a part of the incident). This statement is couched in the idea of *similarity* - a future incident may be completely identical, or it may be similar in some number of ways. The analytic concept of similarity shows up in action items, frequently in events (e.g. a specific failure) and goals (e.g., preventing overload), and we have also captured it in memos we have written about the IR itself (i.e. not tied to one particular action item).

The concept of similar addresses 1) what is it similar to and 2) how something is similar. The former is most commonly an event which may be a failure (e.g., a DNS failure in 22-4), but also may be a tipping point such as a scaling limit (25-7), bugs, (21-6) or even just a negative aspect of the incident like miscommunication to customers (14-1) or unclear status reporting (14-3). Something may be similar in the impact, including the duration (7-10), blast radius, or even effort to fix. They also appear to indirectly refer to the behavior of the system or how it may react to the event (e.g., protection against a *class* of failures or scenarios in 13-1, 25-7).

4.5.2 Uncertainty. The concept of uncertainty also manifests in the events and goals of action items. Unforeseeable, unexpected, and unanticipated events may be tipping points (6-1, 9-11, 10-1), limits (10-1), and complex failures (6-1). There is uncertainty in goals related to ideas of safety and risk like providing additional headroom (9-13, 10-2, 10-3, 10-4, 10-5, 24-3, 24-4, 25-1) and being careful when reintroducing a feature after it led to an extensive

outage (9-18). Some incident reports also mention uncertainty untied to a specific action item - such as being unable to ascertain the origin of a failure (IR 3), noticing the limits of anticipation (IR 2), and addressing the immediate culprit but being aware that they are continuing to be at risk for tipping over (IR 10).

4.5.3 Prioritization. Action items have a prioritization driven by scarcity in roadmap, budget, etc. In our experience, key action items are done most quickly, and in some cases before the incident report is even published. Action items included in incident reports are typically those that have not yet been completed and range from those being done soon to those which will be placed on a backlog and may never be completed. We note the difference between prioritization of an action item (i.e., the ordering of that item relative to other work which needs to be completed), the amount of effort required to complete an action item (e.g. expansive, time-consuming actions), and the scheduling of an action item (i.e., when it will be started).

In our data, we observed four examples of prioritization. First, ordering explains how some action items will be done or take place before others (25-3, 25-4), though it may be the importance of later action items which defines the priority of preceding ones (in other words, preparatory work which by itself is inconsequential). Second, a backlog is where actions with lower priority (13-4, 13-5) may live until items with higher priority are completed. Third, action items may be reprioritizations (2-1, 14-4) e.g., when engineers "realized more clearly the value of [a feature]" (2-1). All the examples in our data are from lower priorities to higher ones, but the opposite could occur and does occur naturally when the previously high priorities are pushed back after reprioritization. Fourth, accelerating and also continuing existing work shows how the priority of an action may be increased or affirmed (7-4, 9-8, 14-3, 25-7).

5 DISCUSSION

From our perspective, incident analysis is related to requirements engineering in several ways, but this connection is not well studied, nor is it made explicit during incident analysis. A failure sheds light on how well requirements are satisfied (or more specifically a scenario is which they are likely violated), challenges domain assumptions, and provides light on the context in which the system operates. As part of this ongoing research project, we are interested in exploring this relationship further, and we are interested in exploring what benefits a requirements engineering focus can bring both to incident analysis and the work that follows that analysis. In this section, we present some of our initial ideas along these lines, by discussing four incident analysis challenges or open questions, and hypothesizing about how RE activities and principles might help in that engineering context. These four challenges are summarized in Table 2.

(1) Adding unplanned effort. The incident analysis process we have studied occurs when there is a failure and of course is not scheduled in advance. We have seen that the actions that result were not previously planned and so have implications for an organizations "roadmap", staffing, etc. Even when an action represents the re-prioritizing of existing work, this still implies a change to the roadmap. For example, accelerating efforts to move a system to multiple availability zones (9-8) is a major effort and is even called out in another action item as a change to roadmap and staffing plans to accelerate their existing efforts (9-9). We anticipate that incident analysis situated in requirements engineering could use a requirements-related risk analysis [2] to allow the changes to roadmap and staffing to be prioritized correctly amid the planned work. For example, moving the system to multiple availability zone may be judged to be a critical way manage a risk that has been unacceptable; to eliminate a determined single point of failure that had exacerbated the impact of the root cause issues in a way that contributed to the extended 73 hour outage. Determining where this work should lie (immediate, within the quarter, within the year) is challenging and requirements engineering can improve decision making around tradeoffs. We also expect that RE treatment can help with improving our understanding and handling of how unplanned work impacts roadmaps.

(2) "Similar" incidents. When considering how action items should prevent a similar incident, the similarity scale goes from a repeat of the exact incident (narrow) to a future incident which only loosely rhymes (broad). The scale may indicate, and provide an size estimate of, a gap in requirements, domain assumptions, specifications, implementation, etc. The iterative nature of software engineering means that we know software is unlikely to reach completeness and so the scale of the gap may provide an estimate of risk (e.g., of how incomplete the software is). Likewise, the similarity scale that the collective action items address closes off some portion of the total risk. In other words, similarity is a useful dimension to engineers when considering what they want to do next (after an incident) since a future roadmap is, in a way, a description of the organization's comfort with risk. It is clear that before that before the correct actions could be decided on or acted on, additional RE work would be needed. We anticipate that this work might include formalizing the similarity scale with a more complete description of potential "similar" dimensions targeted to the incident which just occurred and then judging the level of acceptable risk of those similar incidents.

(3) Inherent uncertainty. It is important to admit, from an RE perspective, that uncertainty is inherently present in most software systems and many software engineering activities (e.g. designing, modeling, prototyping, verifying, various deployment strategies, monitoring, testing) relate to managing uncertainty. From an RE perspective, there is uncertainty in (1) the extent to which requirements and assumptions are complete, (2) the extent to their satisfaction by the implementation, and (3) the extent to which there will be obstacles to their satisfaction [3]. The socio- side of the sociotechnical system also brings uncertainty. Knowledge uncertainty is uncertainty in the assessment of physical uncertainty by experts when judging e.g., how likely a failure is to occur. The work which follows an incident can be couched as ways to reduce these uncertainties and making uncertainty explicit will allow management by RE uncertainty approaches [18].

(4) Violating assumptions. As part of an incident analysis, an assumption may have been discovered to be incorrect (e.g. "we had believed [..] to be impossible", IR 21). In this case, the assumption was about the behavior of the system, and rightly so, several of the action items had the goal of evolving the system so that there was more confidence that the assumption was true (catch the problem earlier in the pipeline, fixing a defect, removing the temporary fix

Table 2: A summary of incident analysis challenges and open questions, and requirements engineering applications

Challenges and open questions	Potential application of Requirements Engineering
How to accommodate the *unplanned effort* that comes from action items	Use requirements-related risk analysis to improve the basis for making roadmap tradeoff decisions with existing work
What *"similar" incidents* should be prevented?	Use a contextualized similarity scale for the incident to determine the acceptable risk of violating requirements.
How to better manage the *inherent uncertainty* in the system?	Be explicit about the uncertainty in requirements, assumptions, specifications, and (the behavior of the) implementation of the system
How to handle unexpected *violations of assumptions* in the system?	Be explicit about assumptions in the system, perform risk analysis to anticipate the harm, and strategically eliminate unacceptable risks

from a previous patch which was part of the incident). Alternatively, engineers could have considered ways to change the assumption, perhaps by loosening or removing it (engineer the system in a way that it doesn't matter if the behavior occurs). The choice of action was a way that uncertainty, in this case about the satisfaction of an assumption, was addressed. We suspect that a solution to this problem might be RE related, such as mapping or making explicit the types of assumptions present in systems and performing risk analysis to both discover assumptions and rank the harm if they were to be violated.

The ideas just discussed are very preliminary, and we plan to further explore the relationship between incident analysis and requirements engineering. Specifically, we have already begun work to identify the relationships between action items, though we have not yet reported on these due to space constraints. We anticipate that understanding patterns will help preserve the intention [11] behind the actions and help engineers better maintain the link between requirements and source code [9, 10].

6 CONCLUSION

We have performed an analysis of action items found in publicly available software incident reports and found that the work being is evolutionary and formative in nature. The actions affect many parts of the socio-technical system, though most are concentrated on the technology (the software systems) and rarely if at all affect the culture, goals, and infrastructure. These actions are being performed in response to events which occurred in all parts of an incident's lifecycle (inception to lingering effects) and these actions are frequently taken to improve safety, visibility of the system, and maturity of engineering activities.

Our analysis has elicited important concepts in action items, and we presented and discussed these with the intent of characterizing incident analysis. We have also proposed how a formal treatment of the activity as an RE activity could address challenges and open questions, leading to a better selection of action items and corresponding evolution of the socio-technical system.

REFERENCES

[1] Dennis Andersson and P Eriksson. Inter-organisational lessons learned: perspectives and challenges. In *Proceedings of the International Emergency Management Society 2015 Annual Conference*, volume 30, 2015.

[2] Yudistira Asnar, Paolo Giorgini, and John Mylopoulos. Goal-driven risk assessment in requirements engineering. *Requirements Engineering*, 16:101–116, 2011.

[3] Antoine Cailliau and Axel Van Lamsweerde. Handling knowledge uncertainty in risk-based requirements engineering. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 106–115. IEEE, 2015.

[4] Matthew C Davis, Rose Challenger, Dharshana NW Jayewardene, and Chris W Clegg. Advancing socio-technical systems thinking: A call for bravery. *Applied ergonomics*, 45(2):171–180, 2014.

[5] Linda Drupsteen, Jop Groeneweg, and Gerard IJM Zwetsloot. Critical steps in learning from incidents: using learning potential in the process from reporting an incident to accident prevention. *International journal of occupational safety and ergonomics*, 19(1):63–77, 2013.

[6] Linda Drupsteen and Frank W Guldenmund. What is learning? a review of the safety literature to define learning from incidents, accidents and disasters. *Journal of contingencies and crisis management*, 22(2):81–96, 2014.

[7] Neil Alexander Ernst. *Software Evolution: a Requirements Engineering Approach*. University of Toronto (Canada), 2012.

[8] Massimo Felici. Structuring evolution: on the evolution of socio-technical systems. In *Structure for Dependability: Computer-based Systems from an Interdisciplinary perspective*, pages 49–73. Springer, 2006.

[9] Alicia M Grubb and Marsha Chechik. Modeling and reasoning with changing intentions: an experiment. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 164–173. IEEE, 2017.

[10] Tobias Hey. Indirect: Intent-driven requirements-to-code traceability. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 190–191. IEEE, 2019.

[11] Jacob Krüger, Yi Li, Chenguang Zhu, Marsha Chechik, Thorsten Berger, and Julia Rubin. A vision on intentions in software engineering. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 2117–2121, 2023.

[12] Jonas Lundberg, Carl Rollenhagen, and Erik Hollnagel. What you find is not always what you fix—how other aspects than causes of accidents decide recommendations for remedial actions. *Accident Analysis & Prevention*, 42(6):2132–2139, 2010.

[13] John Lunney, Sue Lueder, et al. Postmortem action items: Plan the work and work the plan. 2017.

[14] Anoush Margaryan, Allison Littlejohn, and Neville A Stanton. Research and development agenda for learning from incidents. *Safety Science*, 99:5–13, 2017.

[15] Wilem J Muhren, Gerd Van Den Eede, and Bartel Van de Walle. Organizational learning for the incident management process: Lessons from high reliability organizations. 2007.

[16] Mohammad Farhad Peerally, Susan Carr, Justin Waring, and Mary Dixon-Woods. The problem with root cause analysis. *BMJ quality & safety*, 26(5):417–422, 2017.

[17] J Paul Reed. Maps, context, and tribal knowledge: On the structure and use of post-incident analysis artifacts in software development and operations. 2018.

[18] Ahmad M Salih, Mazni Omar, and Azman Yasin. Understanding uncertainty of software requirements engineering: A systematic literature review protocol. In *Requirements Engineering for Internet of Things: 4th Asia-Pacific Symposium, APRES 2017, Melaka, Malaysia, November 9–10, 2017, Proceedings 4*, pages 164–171. Springer, 2018.

[19] Eva-Maria Schön, Jörg Thomaschewski, and María José Escalona. Agile requirements engineering: A systematic literature review. *Computer standards & interfaces*, 49:79–91, 2017.

[20] Jonathan Sillito and Esdras Kutomi. Failures and fixes: A study of software system incident response. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 185–195. IEEE, 2020.

[21] Michael Sony and Subhash Naik. Industry 4.0 integration with socio-technical systems theory: A systematic review and proposed theoretical model. *Technology in society*, 61:101248, 2020.