Postmortem Action Items

Plan the Work and Work the Plan

JOHN LUNNEY, SUE LUEDER, AND BETSY BEYER

John Lunney is a Senior Site Reliability Engineer at Google Zürich. His team manages G Suite, productivity apps for Enterprise customers. He holds a degree in computational linguistics from Trinity College in Dublin, Ireland. Before Google, he worked on several lexicography projects for the Irish language. lunney@google.com

Sue Lueder is a Site Reliability Program Manager in Google's Mountain View office. She's part of the team responsible for disaster testing and readiness, incident management processes and tools, and incident analysis. Before Google, Sue worked as a Systems Engineer in the wireless and smart energy industries. She has an MS in organization development from Pepperdine University and a BS in physics from UCSD. slueder@google.com

Betsy Beyer is a Technical Writer for Google Site Reliability Engineering in NYC. She has previously provided documentation for Google Data Center and Hardware Operations teams. Before moving to New York, Betsy was a lecturer in technical writing at Stanford University. She holds degrees from Stanford and Tulane. bbeyer@google.com

In the 2016 O'Reilly book *Site Reliability Engineering*, Google described our culture of blameless postmortems and recommended that operationally focused teams and organizations institute a similar culture of postmortems in their approach to production incidents. A postmortem is a written record of an 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. The chapter "Postmortem Culture: Learning from Failure" describes criteria for deciding when to conduct postmortems, some best practices around postmortems, and advice on how to cultivate a postmortem culture based upon the experience we've gained over the years.

We write postmortems to ensure we achieve a few primary goals:

- We understand all contributing root causes.
- The incident is documented for future reference and pattern discovery.
- We enact effective preventive actions to reduce the likelihood and/or impact (i.e., duration and/or scope) of recurrence.

We refer to the preventive actions identified during root cause analysis as *postmortem action items*, which in aggregate form the *postmortem action item plan*.

This article addresses the challenges in designing an appropriate action item plan and then executing that plan. We discuss best practices for developing high-quality action items (AIs) for a postmortem, plus methods of ensuring these AIs actually get implemented. If the AIs are not closed out, you are implicitly agreeing that it is acceptable to suffer the exact same outage again. Furthermore, if you are successful as a service, the outage will be larger the next time around.

It's worth noting that Google teams are by no means perfect at formulating and executing postmortem action items. We still have a lot to learn in this challenging area and are sharing our approach to give a starting point for discussion throughout the industry.

Action Item Best Practices

Successful AIs require careful thought at both ends of their life cycle: formulation and follow-through. The following sections detail best practices we've cultivated as we continually refine our methods.

Enacting AIs

Classifying Action Items for Full Coverage

We classify action items by category (Investigate, Mitigate, Repair, Detect, Prevent) to make sure that the action item plan covers both very short-term and longer-term fixes. Making sure to consider AIs for each category can inspire simple but effective changes, particularly around detection (as early detection is often the best way to reduce time to resolution).

When an outage has multiple contributing causes, you need a multi-dimensional action item plan that will address each root cause and all systems that contributed to the outage.

At a minimum, your postmortem must include AIs to **Mitigate** and **Prevent** future incidents, but it should also include all other relevant categories listed below. Note that many teams initiate incident investigation and mitigation (bullets one and two) before conducting the postmortem.

- **Investigate** this incident: what happened to cause this incident and why? Determining the root causes is your ultimate goal.
  Examples: logs analysis, diagramming the request path, reviewing heapdumps
- **Mitigate** this incident: what immediate actions can we take to resolve and manage this specific event?
  Examples: rolling back, cherry-picking, pushing configs, communicating with affected users
- **Repair** damage from this incident: how can we resolve immediate or collateral damage from this incident?
  Examples: restoring data, fixing machines, removing traffic re-routes
- **Detect** future incidents: how can we decrease the time to accurately detect a similar failure?
  Examples: monitoring, alerting, plausibility checks on input/output
- **Mitigate** future incidents: how can we decrease the severity and/or duration of future incidents like this? how can we reduce the percent of users affected by this class of failure the next time it happens?
  Examples: graceful degradation; dropping non-critical results; failing open; augmenting current practices with dashboards, playbooks, incident management protocols, and/or war rooms
- **Prevent** future incidents: how can we prevent a recurrence of this sort of failure?
  Examples: stability improvements in the code base, more thorough unit tests, input validation and robustness to error conditions, provisioning changes-

When filing issues or bugs for these action items, make sure to use the appropriate classification (bug vs. feature request). Although this differentiation may seem subjective, in our view, a **bug** is a deviation from required behavior, while a **feature request** is new required behavior. Typically, you should use the type your team tracks most strictly (see the later section "Prioritizing Action Items " for more details).

Wording Action Items

The right wording for an AI can make the difference between easy completion and indefinite delay due to infeasibility and/or procrastination. A well-crafted AI should manifest the following properties:

- **Actionable**: Phrase each AI as a sentence starting with a verb. The action should result in a useful outcome, not a process. For example, "Enumerate the list of critical dependencies" is a good AI, while "Investigate dependencies" is not.
- **Specific**: Define each AI's scope as narrowly as possible, making clear what is and what is not included in the work.
- **Bounded**: Word each AI to indicate how to tell when it is finished, as opposed to leaving the AI open-ended or ongoing.

Table 1 provides examples of poorly worded vs. well-crafted AIs.

Poorly Worded	Better
Investigate monitoring for this scenario.	(Actionable) Add alerting for all cases where this service returns >1% errors.
Fix the issue that caused the outage.	(Specific) Handle invalid postal code in user address form input safely.
Make sure engineer checks that database schema can be parsed before updating.	(Bounded) Add automated presubmit check for schema changes.

Table 1: Examples of action items

We recommend implementing automated fixes when possible, as opposed to prevention/mitigation that requires ongoing manual intervention.

Consider grouping AIs either by theme or by team. In addition to providing a clear organizational or responsibility-focused structure, this categorization may also help you spot an unbalanced AI plan (see "Unbalanced Action Item Plans").

After the post-incident dust settles, don't be afraid to update a poorly worded AI to make it more tractable.

Prioritizing Action Items

It's crucial to properly prioritize action items because the priority guides future attention each AI will receive. At Google, we use the following priority levels, based on estimated risk:

- **P0**: *High risk of unmitigated recurrence of the incident if this AI is not resolved.* Resolving this AI will directly address a root cause. Resolution will either completely prevent such incidents from recurring or greatly reduce their impact to a negligible level.
- **P1**: *Medium risk of unmitigated recurrence of the incident if this AI is not resolved.* Resolving this AI will directly address the root cause. Resolution will either significantly mitigate the impact of a recurrence or have a high chance of preventing a recurrence.

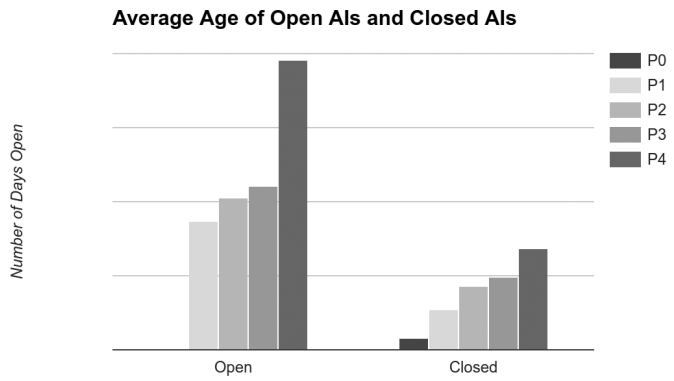

Figure 1: Time to close out action items

- **P2**: *Low risk of unmitigated recurrence of the incident if this AI is not resolved.* Resolving this action item will only superficially mitigate a recurrence of this issue or will address only peripheral contributing conditions.
- **P3**: *Trivial risk of unmitigated recurrence of a similar incident if this AI is not resolved.*

We require every postmortem stemming from a user-visible event to have at least one P0 or P1 action item. If the outage was bad enough to disrupt users, it's important enough to require high priority follow-up work to avoid or mitigate recurrence.

Figure 1 shows that on average, high priority actions are closed more quickly than low priority AIs. However, when it comes to AIs that are still open, priority doesn't significantly influence their age—on average, outstanding P1 AIs have been open almost as long as outstanding P3 AIs. We use this data to implement initiatives to bring more attention to open actions from postmortems.

Following Up on AIs

Postmortem Reviews

Many teams at Google that participate in incident response conduct postmortem review sessions. These reviews are helpful in bringing key parties together to ensure that the postmortem is complete and that the action item plan covers required categories and avoids anti-patterns. Most postmortem reviews have the following general format:

- **Walkthrough of incident timeline, impact, and root cause**: Include clarifications and address open discussion threads.
- **Review of lessons learned**: Discuss updates, additions, and mappings to action items.
- **Review of action items**: Review the checklist (see the Appendix) to make sure AIs have owners, wordings are clear, priorities make sense, and that no category (Investigate, Mitigate, Repair, Detect, Prevent) is missing.

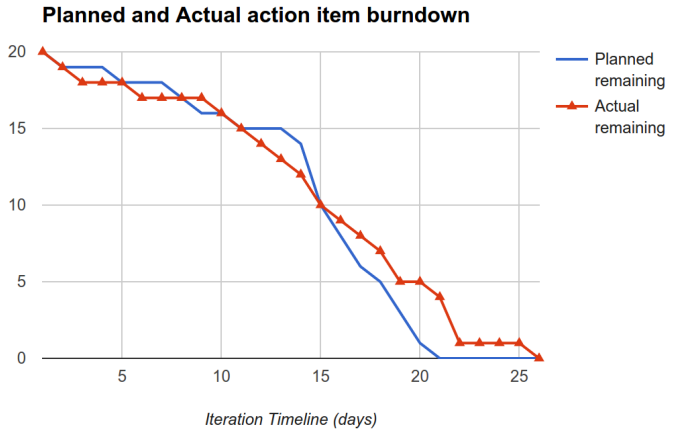

Figure 2: AI burndown for a single postmortem

These reviews should happen soon after an incident so that the parties involved remember what happened. You can hold reviews on a small scale with just the impacted team(s), or on a large scale with many parties and observers.

Action Item Closure Tracking

Encourage action item owners to close AIs that you'll never have time to address—don't keep them around forever. If an AI is obsolete or infeasible, it just distracts you from the AIs that still need work.

It's a good idea to provide periodic visibility into team progress towards reducing the technical debt identified in postmortems. Consider adding postmortem action item burndown progress (that is, AI follow-through) to your regular service or team reporting. For example, you might build postmortem AI reports in your bug/issue-tracking system and track these issues against a closure-time objective, following up with outliers.

In many cases, an action item requires considerable effort and must fit in with work that's already scheduled. Keeping an eye on how long it takes to close out action items on average helps us identify where slow action item closure leads to additional risk to reliability.

We actively monitor bug burndown over time. There are multiple ways to visualize this data. Figure 2 shows how we might track burndown for a single postmortem. In this example, the team planned out an action item completion schedule for all 20 actions to be completed over 21 days. They monitored progress until the final action item was complete on the 25th day.

Figure 3 shows how we might track AIs across any part of the organization by measuring the number of postmortem AIs created vs. closed by day. The widening distance between the two lines indicates accumulating technical debt over time, a pattern that you should seek to avoid.

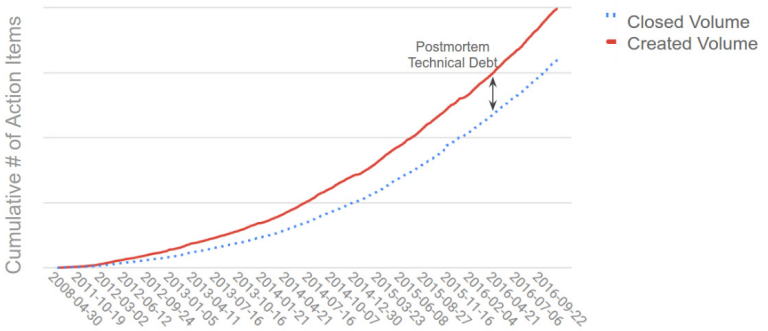

Figure 3: Postmortem AIs created vs. closed, by day

Executive Focus

You can further shine light on postmortem AI follow-up through close attention from senior leaders in your organization. We regularly review postmortems with VPs and Directors to ensure that high priority postmortems and action items receive the attention they deserve.

Action Item Anti-Patterns

In reviewing the thousands of postmortems we've conducted at Google over the years, we've identified a number of common deficiencies when it comes to both constructing and handling action items. The most common shortcoming is lack of follow-up (and many of our best practices aim to mitigate this problem).

The following section presents several other anti-patterns, which relate to how we structure or enact postmortem action items. Our experience shows that if either of these steps goes wrong, no amount of follow-up will help because vague or misleading AIs can't be completed.

Structuring AIs

Unbalanced Action Item Plan

If your postmortem action item plan contains only long-term, unrealistic, or infeasible actions, it's likely that you won't resolve any AIs before the next outage hits. On the other hand, a plan that only includes tactical items and never explores better ways to architect a more robust system is a missed opportunity to increase reliability.

Mitigation: Create a balanced action item plan that includes both:

- Near-term fixes to prevent/mitigate a similar outage
- Strategic improvements to the design of implicated systems to increase reliability

Strike a healthy balance between local/incremental/Band-Aid solutions and loftier long-term improvements. Covering all the categories in "Classifying action items" helps in this effort.

Tossing Work "Over the Wall"

An action item plan often requires work from partner teams in other parts of the organization. Don't draft and file action items against other teams without some discussion with the team that owns the component. Without this discussion, you're essentially throwing work over the wall and hoping that it gets done.

Mitigation: Include partner teams in the postmortem drafting process. Make sure each owner or team is satisfied with their assigned action items before publishing the postmortem. You might want to discuss the action item plan in person or via videoconference and ask for commitment to a resolution time frame. If conflict arises, "Executive focus" (see the best practice) may help with escalation.

Focusing on Elimination (at the Cost of Mitigation)

It can be tempting to design an action item plan that will eliminate the chance of the incident from ever happening again. Of course, you should take those actions when appropriate, but you should also spend time evaluating how to reduce the duration and impact of the incident—especially if such a fix will take effect sooner than a potential "elimination fix."

Mitigation: Take a look at how an incident unfolds and consider writing detection and mitigation action items that address the following:

- Could we have detected the incident sooner?
- Could we have triaged the impact sooner, leading to a more appropriate incident response?
- Could we have understood the root cause sooner, leading to faster rollback?
- Could the rollback have proceeded faster or more smoothly?
- Could we have scaled back the initial faulty rollout, thereby impacting a smaller percentage of users?

Thinking Only of the Current Incident (Missing Patterns)

One of our colleagues appropriated Mark Twain to observe, "We rarely repeat incidents, but we sometimes have incidents that rhyme." If we only consider a given incident in isolation, we may overfit a specific solution to the incident at hand. We also might create duplicate actions by missing information about improvements that are underway as part of another postmortem action item plan. Even worse, we might miss an opportunity to kill two risks with one stone.

Mitigation: Review postmortems for similar incidents and their accompanying action items. You might identify an opportunity to add resources to an action item that's not getting the attention it deserves, or an opportunity to collaborate on a new action item that would help in both types of incidents.

Enacting AIs

Lack of Ownership

The surest way for a postmortem author to ensure that an action item never gets completed is to leave it without an owner.

Mitigation: Always assign an owner for every action item as it is enacted, even if that owner's primary task is to find the best person for the job (e.g., the Tech Lead for the team responsible for that product or software). Your issue tracker is an appropriate place to assign ownership.

Overly Specific Monitoring Changes

With the benefit of hindsight, it's easy to say we should have monitored an XYZ-specific signal, which would have alerted us to the problem before it became a huge incident. However, this strategy only helps if that very specific failure mode recurs in the exact same way (which is frequently not the case).

Mitigation: Make the effort count: look for ways to improve monitoring for a whole class of issues. Preferably, these improvements should target user-focused symptoms rather than internal metrics. Otherwise, you risk tying alerts to implementation details.

Fixing Symptoms (Not Root Causes)

Root cause analysis (RCA) is one of the most crucial parts of a postmortem. The outcome of this analysis should drive the construction of the action item plan. Shallow RCA limits action items to impermanent fixes or surface patches to problems.

Mitigation: A thorough RCA is key to defining action items that will prevent or mitigate future incidents of this nature. Use the five-whys RCA (or another methodology [2]) to help determine which contributing causes in the chain your AIs should target.

Blaming Humans (Missing System Fixes)

It's very rarely productive to think of humans as the ending "why" in a root cause chain. During an emergency, people are typically doing the best they can under intense pressure and when faced with ambiguous data. As a result, what looks like an obvious point in the cold light of day can be quite non-obvious in the heat of the moment.

Attributing blame to a specific person or group doesn't improve your system or spur development of systematic defenses. The next time there is an emergency, the hapless on-duty person will be faced with a similarly difficult problem to solve in real time. If you trust your engineers to make the best decision given available information, it's more helpful to consider an error to be a failure of the entire system, as opposed to the fault of one or more humans.

Mitigation: Rather than finger-pointing, it's much more helpful to think about:

- How we can give people better information to make decisions?
- How we can make our environment, systems, tools, and processes more immune to human fallibility?

When you feel tempted to use human error as a root cause, use a critical eye to avoid one-off fixes. A useful stance is to believe, "The system should not have been able to fail this way." Ask yourself the following questions:

- How likely is the next person to cause the same problem? Could a new hire or sleepy SRE at 4 a.m. have made this mistake? Why did the system let them?
- Was information flawed, misleading, or poorly presented? Can we fix that misinformation (preferably through the use of automation)?
- Could software have prevented/mitigated this error? Can we automate this activity so it doesn't require human intervention?

Fixes Late in the Software Life Cycle (Missing Earlier Chances)

It can be tempting to stop a badly behaving system from impacting users by implementing a check or safeguard at the last step before changes enter production. For example, you might implement additional checks right before a config file is pushed to production but fail to consider adding configuration file coding standards, automated testing, improved training, or making sure there are fewer ways to break configuration files in the first place. The fact that bugs are much more expensive to fix late in the software life cycle is well understood in the industry [3].

Mitigation: When reviewing the postmortem timeline and lessons learned, look for ways to address the root cause (and possibly, the event trigger) as early as possible. The fix might be the same (e.g., input validation) but applied to the first system as opposed to the last one.

Conclusion

Years of conducting postmortems at Google have taught us that there's no one-size-fits-all approach to conducting this exercise successfully. However, this accumulation of experience—what we've done right, what we've done wrong, and how we've iterated to improve—has led to a certain amount of insight, which we hope can benefit other companies and organizations. We believe that it's very important to both construct high quality postmortem action items and follow up on them in a timely and comprehensive manner. Only by completing these AIs can we hope to avoid recurrence of costly and time-consuming production incidents.

The checklist appended to this article is a good starting point if you're new to conducting postmortems, or perhaps a useful honing tool for veterans of this process. As we continue to refine our approach to this imperfect science, we hope to learn equally valuable lessons from others in the field.

References

[1] B. Beyer, C. Jones, J. Petoff, and N. Murphy, eds., *Site Reliability Engineering* (O'Reilly Media, 2016).

[2] More formal methodologies exist for those looking for more rigor. For example, Ishikawa fishbone diagrams, 8Ds, fault tree analysis, and failure mode and effects analysis (FMEA). See https://en.wikipedia.org/wiki/Root_cause_analysis for more ideas.

[3] J. Leon, "The True Cost of a Software Bug: Part One," Celerity blog, Feb 28, 2015: http://blog.celerity.com/the-true-cost-of-a-software-bug.

Checklist

Structuring

- ☐ Each lesson learned is addressed with at least one AI.
- ☐ AI plan is balanced between near-term fixes and strategic design improvements.
- ☐ AIs address both prevention and decreasing resolution time.
- ☐ "Rhyming" incidents and their action plans have been reviewed.
- ☐ No work is tossed "over the wall": all involved teams are committed to relevant AIs.

Enacting

- ☐ AIs cover the two most critical categories (Mitigate, Prevent) + all other relevant categories (Investigate, Repair, Detect).
- ☐ AIs are worded to be actionable, specific, and bounded.
- ☐ AIs are prioritized, with at least one P0 or P1 to avoid or mitigate recurrence.
- ☐ All AIs have an owner.
- ☐ AIs aren't overly specific (for example, could you monitor something more general?).
- ☐ Problem is caught as early as possible in the software life cycle.
- ☐ AI plan addresses a root problem (as opposed to just patching symptoms).
- ☐ AIs don't blame humans (focus instead on automatic system detection).

Follow-up

- ☐ AI plan is shared with your team, stakeholders, and those involved in the incident.
- ☐ AIs are appropriately filed and tagged/tracked to appear in your reporting system.
- ☐ AI plan was reviewed with an executive or group of leads for visibility.
- ☐ Postmortem and AI plan were reviewed/approved per team policy.