Cloud CRE
Production Maturity Assessment

Monitoring and Metrics

Establishing desired service behavior, measuring how the service is actually behaving, and correcting discrepancies. Example metrics: response latency, error or unanswered query rate, peak utilization of resources.

SLO Definition and Measurement

Having a published SLO that represents user requirements of the service. Evaluating and reporting against the published SLOs.

- Does your service / application have a well defined SLO?
  A Service Level Objective (SLO) is a target for a measurement of service behavior, e.g. that the latency for requests to the service is under 500ms for 95% of queries.

- Does the SLO reflect the experience of customers?
  I.e., does meeting the SLO imply an acceptable customer experience? And conversely, does missing the SLO imply an unacceptable customer experience?
- Is the SLO published to users?
  This isn't common, but sometimes if you have a single big consumer of your service then you may publish your SLO so they can plan around your service's performance.
- Do you have well-defined measurements for the SLO?
  - Is the measurement process documented?
  - Is the measurement process automated?
    I.e., do you get reports or graphs for the SLO automatically, or do you have to manually run a command or fill in a spreadsheet to see how well you are meeting the SLO?
- Do you have a process for changing / refining the SLO?
  For instance, if you're consistently failing to meet your SLO but your users don't seem to be complaining, how do you decide whether to change the SLO target level?

Maturity Levels

1 - Unmanaged

Service has no SLO or SLO does not represent user requirements.

2 - Managed

Service has defined SLOs. Service has measurement of SLO but it is ad hoc or undocumented.

3 - Defined

Service has defined SLOs. Measurement is a documented process with clear owner.

4 - Measured

SLO is published to users. SLO is measured automatically. SLO represents user requirements.

5 - Continuous Improvement

Ongoing evaluation and refinement of SLO with users.

Dashboards & Visualization

Clear presentation of data to support service management, decision making, and behavior modification.

- Do you collect data about your service?
  - Which collection methods do you use?
    - Logs processing

        - White box Monitoring
          Inspecting internal system state
        - Black box Monitoring
          Synthetic transactions/probes that emulate real user requests
        - Metrics collection from client devices
        - Other
            - Can you briefly describe your collection method(s)?
    - Is the collection automated?
- Does your service have dashboards?
  A dashboard is typically a web page displaying graphs or other representation of critical monitoring information, allowing you to see at a glance how well your service is performing.
- Do the dashboards contain key service metrics?
  E.g.: QPS, latency, capacity, error budget exhaustion
- Do the dashboards contain key business metrics?
  E.g.: visits, users, views, shares
- Are the dashboards part of a centralized dashboard location for the business?
- Do dashboards share a UI / UX across applications / services / teams?
- Does the business have tools for ad hoc data exploration?
  E.g.: drill-down dashboards, splunk / bigquery, data warehousing

Maturity Levels

1 - Unmanaged

No Dashboards. Data gathering is ad hoc, inconsistent, and undocumented.

2 - Managed

Dashboards may exist for key metrics, but only in static form (no customization). Dashboards not centrally located, or dashboard use is not standardized. Dashboard ownership is unclear.

3 - Defined

Dashboards exist and support common technical use cases. Dashboard ownership is clear. Tools to support ad hoc queries exist but are somewhat cumbersome (using them requires training or hours of experimentation to produce a result.)

4 - Measured

Dashboards support both technical and business use cases. Ad hoc data exploration tools are customizable and support common use cases without training (e.g. by following a recipe) or through intuitive interfaces.

5 - Continuous Improvement

Dashboards are standardized across the business units.

User Focus

Collection of data that accurately reflects user experiences with the product, and use of that data to maintain service quality. Distinguishing between synthetic metrics and measurement of critical user journeys; e.g., if the server is “up” but users still can’t use the product, that metric doesn’t give insight into user experience.

- Does the service collect any data that is not exported from the server?
  Typically, probes or critical user journey scripts
- Does probe coverage cover complex user journeys?
  E.g., for e-commerce: homepage => search for product => product listing page => product details page => "Add to Cart" => checkout
- Do you offer SLOs on specific user journeys / flows?
- Do you have an established response to regressions in the user-journey-focused metrics?
  E.g., whenever the measured time for users to complete an account creation workflow increases to above 5 minutes, the established response is to put a hold on feature rollouts and prioritize remedial work.
  - What is the response?
    - Rollback a release
    - Page someone
    - Develop a tactical fix
    - Other
      - Can you briefly describe your response method?
- Do you evaluate the status of user journey SLOs in order to drive service improvements?

Maturity Levels

1 - Unmanaged

No data is collected about user experience or only available data is server-side metrics (e.g. service response latency.)

2 - Managed

Probes exist to test specific user endpoints (e.g. site homepage, login) but probe coverage of complex user journeys / flows is missing. Often probes are single-step (fetch this page) as opposed to multi-step journeys.

3 - Defined

Complex user journeys are measured (either with complex probing or client-side reporting of live traffic). Service improvements are reactive, not proactive.

4 - Measured

SLOs explicitly cover specific user journeys. Not simply "is server up" but the product is up sufficiently to allow users to perform specific product functions. SLO violations of user journeys result in release rollbacks.

5 - Continuous Improvement

Service latency & availability continuously evaluated and used to drive service improvements.

Capacity Planning

Projecting future demand and ensuring that a service has enough capacity in appropriate locations to satisfy that demand.

Business Metric Forecasting

Predicting the growth of a service's key business metrics. Examples of business metrics include user counts, sales figures, product adoption levels, etc. Forecasts need to be accurate and long-term to meaningfully guide capacity projections.

- Do you have key business metrics for your application?
  E.g., number of users, pictures, transactions, etc.
- Do you have historical trends for the key business metrics?
  - What is the retention time, in months?
- Do you forecast growth of key business metrics into the future?
  - How far in the future do you forecast, in months?
- Do you measure forecast accuracy by comparing the forecast to the observed values over the measurement period?
  - What is the observed percent error for the 6-month forecast? (0-100+)
- Does your forecast support launches or other inorganic events that may cause rapid changes in business metrics?

Maturity Levels

1 - Unmanaged

No units defined, or units defined but no historical trends.

2 - Managed

Defined business units, historical trend for units, but limited ability for forecasting these units. Forecasts are either inaccurate or short-term (<=1 quarter)

3 - Defined

Have solid historical trend, have a forecast for 4-6 quarters. Have a non-trivial forecast model.

4 - Measured

Have measured the accuracy of past predictions. Using 6-8 quarter forecasts.

5 - Continuous Improvement

Defined & measured & accurate forecast, know how to forecast accurately & improving record of accuracy; accuracy of actual demand within 5% of 6-month forecast.

Supply Metric Modeling

Calculating capacity requirements for a set of business metrics through an empirical translation model. A very basic example model: *To serve 1000 users takes 1 VM. Forecasts say users grow 10% per quarter, so VMs must also grow 10% per quarter.*

- Do you have a model to convert from business metrics into logical capacity metrics (VMs, pods, shards, etc.)?
  A simple example of a model might be, "For every 1000 users we need an additonal VM-large-1."
  - Is the model documented?
- Do you validate any aspects of your capacity model?
  For the example model above: How do we know that 1000 users can fit on a single VM-large-1? Do we have a load test?
  - Is the validation automated?
- Does the model account for variations in resource size or cost?
  E.g., differences across locations in the CPU cost efficiency or the available VM size.
- Do the ongoing measurements of actual demand feed back into revisions of the capacity model?

Maturity Levels

1 - Unmanaged

No model exists for changing business metrics into logical supply. Service is overprovisioned (buy and hope model.) No significant time spent on capacity planning.

2 - Managed

A simplified rule of thumb/guideline is in use for your service, e.g. 1000 users / VM.

3 - Defined

Has a load test, knows the likely scaling points. e.g. Load tests provide data that concludes 1200 users really fit on a VM. Validated rule of thumb with empirical evidence. Model verification may be ad hoc (unautomated.)

4 - Measured

Have a documented service model that is regularly verified (e.g. by a loadtest during release qualification.) Model is simplistic with gaps in dimensionality (e.g. covers CPU well, but not bandwidth.) Model does not account for difference in resources cost / size between locations. For example, CPU family, resource cost, VM sizes, etc.

5 - Continuous Improvement

Have a well defined model to translate units of demand into bundle of supply units using feedback from the live system. Have agreed development plans to improve critical dimensions. Understand and account for in model difference between platforms/regions/network. Understand non-linearity if present.

Acquiring Capacity

Provisioning additional resources for a service: knowing what resources are needed when and where, understanding the constraints that must be satisfied, and having processes for fulfilling those needs.

- How many total hours does the team spend on resource management each month?
- Are the processes for acquiring more resources well-defined?
    - Are the processes documented?
    - Are the processes automated?
- Are resource requests automatically generated by the forecasting process?
- Do you know what constraints determine the location of your service?
  E.g., do your servers have to be in a specific Cloud zone or region? If so, why? Do your serving, processing, and data storage have to be colocated, or in the same zone or region, and why? Do you have data regionalization requirements?
    - Are the constraints documented?
- Do you monitor the adherence to those service constraints?

Maturity Levels

1 - Unmanaged

Resources are deployed by hand in an ad hoc fashion. Relationships between system components are undocumented. No idea if resources deployed are sufficient.

2 - Managed

Rules of thumb exist for service constraints (database and web near each other) but constraints are not measured (unverified). Defined process for acquiring resources; but process is manual.

3 - Defined

Acquiring resources is mostly automated. When to acquire resources is a manual process.

4 - Measured

System constraints are modelled, but lack evaluation. Service churn over time can cause constraints in the model to drift from reality. When to acquire resources is driven by forecasting.

5 - Continuous Improvement

Models are continuously evaluated and maintained even against service churn. Where to acquire capacity is driven by models, when to acquire capacity is driven by forecasting.

Capacity Utilization

Monitoring the service's use of its resources, understanding the close and sometimes complex relationship between capacity and utilization, setting meaningful utilization targets, and achieving those targets.

- Have you defined any utilization metrics for your service?
  E.g., how much of your total resources are in use right now?
- Do you continuously measure your utilization?
- Do you have a recognized utilization target?
  I.e., do you have a specific threshold for your utilzation metric(s) that is documented/recognized as the required minimum, such that failing to meet that target would trigger some response?
- Does the utilization target drive service improvements?
- Do you review your utilization target and your service's conformance to that target on a regular basis?
  E.g., a quarterly review.

- Do you understand what the resource bottlenecks are for your service?
  For the example above (1000 users / VM): If we put 1200 users on a VM, what resource would be exhausted first? (CPU, RAM, threads, Disk, etc.)
  - Can you briefly describe them?
- Do you test new releases for utilization regressions?
- Do significant utilization regressions block releases?

Maturity Levels

1 - Unmanaged

No system utilization metrics are defined.

2 - Managed

Utilization metric and target defined, measurement is ad hoc (non-continuous). Quarterly / yearly.

3 - Defined

Utilization is measured continuously and reviewed on a regular basis (monthly / quarterly).

4 - Measured

Utilization is an indicator of release health (releases with large regressions in utilization are caught and block release.)

5 - Continuous Improvement

Utilization metrics are continuously evaluated and are used to drive service improvements (e.g. to keep service costs sustainable.)

Change Management

Altering the behavior of a service while preserving desired service behavior. Examples: canarying, 1% experiments, rolling upgrades, quick-fail rollbacks, quarterly error budgets

Release Process

Flag, data and binary change processes.

Disambiguation of terms

Terminology around releases can vary in different contexts, so here are some definitions we'll work with for the purpose of this assessment.

- ***Service version:*** *A bundle of code, binaries, and/or configuration that encapsulates a well-defined service state.*
- ***Release candidate:*** *A new service version for deployment to production.*
- ***Release:*** *the preparation and deployment to production of a new release candidate.*
- ***Release candidate preparation:*** *The creation of a viable release candidate. This covers a range of possible activities, including compilation and all testing, staging, or other validations that must occur before any contact with production.*
- ***Deployment:*** *The process of transitioning to a new service version. E.g., pushing a binary to its production locations and restarting the associated processes.*
- ***Incremental deployment:*** *Structured deployment to multiple production locations as a series of consecutive logical divisions. E.g., deployment to several distinct sites, one site at a time. Or, deployment to 1000 VMs as a series of 100 VM subsets.*
- ***Canary:*** *A strategic deployment to a limited subset of production to test the viability of the version, with well-defined health checks that must be satisfied before a wider deployment can proceed.*

- Do you have a regular release cadence?
  I.e., do you have a set frequency for executing your release process?
  - Which of these is most similar to your release cadence?
    - Push-on-green
    - Daily
    - Weekly
    - Biweekly
    - Monthly
- How many engineer-hours are required for a typical release?
  How many hours of human time, from the start of release candidate preparation to the end of deployment monitoring?
- What percentage of releases are successful?
  A successful release is generally one that reaches full deployment and doesn't need to be rolled back or patched.
- Is the release success rate commensurate to your business requirements?
  E.g., are releases reliable enough that they don't impede development or undermine confidence in feature deliveries?
- Do you have a process for rolling back a deployment?
- Do you practice incremental deployment?
  E.g., giving a new release to 1% of users, then 10%, then 50%... (See the definitions above.)
- Do you have well-defined failure detection checks for releases?
  E.g., metrics which humans or robots evaluate to see if the release seems to be broken.
- Do you use a canary (or series of canaries) to vet each deployment?
  I.e. a small amount of real requests get sent to a new deployment to see whether it handles them correctly, before it gets all the requests. (See the definitions above.)
- Is any part of the release process automated?

    - Which parts?
        - Release candidate preparation
        - Deployment
        - Failure detection
        - Deployment rollback
        - Canarying
- Is any part of the release process documented?
  I.e., if you told a new team member to perform a release, would they be able to find and follow instructions for any part of that process?
    - Which parts?
        - Release candidate preparation
        - Deployment
        - Failure detection
        - Deployment rollback
        - Canarying

Maturity Levels

1 - Unmanaged

Releases have no regular cadence. Release process is manual and undocumented.

2 - Managed

Release process is documented. Regular releases are attempted. Process for mitigating bad release exists.

3 - Defined

Customer has automated continuous integration / continuous delivery pipeline. Rollouts are predicatable and success rate meets business needs. Rollout process is designed to allow mitigation of adverse impact within parameters that have been pre-agreed with the business.

4 - Measured

Completely automated release process, including automated testing, canary process, and automated rollback. Rollouts meet business needs. Capability to rollback without significant error budget spend is demonstrated.

5 - Continuous Improvement

Standard Q/A, and canary process. Rollouts are speedy, predictable and require minimal oversight. Process fully mapped and published. Automated release verification and testing, integrated with monitoring. Robust rollback and exception procedure. Push and release frequency matches business needs. Release process matches the product's need.

Design & Launch

Architecting a service to be successful, via early engagement, design review, best practices, etc.

- Does the team have a review process for significant code changes?
  E.g., design reviews or launch reviews.
  - Are the reviews self-service?
  - Do the reviews have specific approvers?
- Does the team have best practices for major changes?
  E.g., if you know you're shipping a feature that adds a new component to your system, is there agreement about how to set up, monitor, deploy and operate the component in a standard way?
  - Are they documented?
- Are the best practices enforced by code and/or automation?
- Can a launch be blocked for not following the best practices?
- Does the team have a mandatory process for risky launches?

Maturity Levels

1 - Unmanaged

The team has no design or review process for new code or new services.

2 - Managed

The team has a lightweight review process for introducing new components or services. The team has a set of best practices, but they are ad hoc and poorly documented.

3 - Defined

The team has a mandatory process for risky launches. The process involves applying documented best practices for launches and may include design / launch reviews.

4 - Measured

Launch reviews are self-service for most launches (e.g. via short survey that can trigger review for edge cases.) Best practices are applied via shared code modules and tuning automation. The team has a documented process for significant service changes and these designs have designated reviewers / approvers.

5 - Continuous Improvement

Best practices are enforced by automation. Exemptions to best practices are allowed by request. Everything in 1-4.

Change Process Automation

Automating manual work associated with service operations

- Do you have change process automation?
  I.e., when you need to perform production operations, such as restarting a set of jobs or changing a set of VMs' configuration, do you have scripts or other tools or services to help you make the change more safely and easily?
- Does your automation use "Infrastructure as code"?
  As opposed to procedural (imperative) automation.
- Does your automation support "diffing" or "dry-run" capabilities so operators can see the effect of the operation?
- Is your automation idempotent?
- Is your automation resilient?
  I.e., it's rarely broken by service, policy, or infrastructure changes.
- Are your routine processes documented?
  E.g. if you told a new team member to go upgrade the version of Linux on all your VMs, would they be able to find and follow instructions on how to do this?
- What percentage of your routine tasks are automated?
- Do you have a process to turn up or turn down your service?*E.g., to deploy in a new GCP region.*
  - Is the process documented?
  - Is the process automated?
  - How many engineer-hours does it require?

Maturity Levels

1 - Unmanaged

The team spends most time on operations or has an "ops team" who spends significant time on operations. Tasks are underdocumented (e.g. specific individuals know processes, but they are not written down.)

2 - Managed

Manual tasks are documented. High-burden tasks are partially automated and / or there is an understanding of how much operations time is spent.

3 - Defined

Many high-burden tasks are automated. Team spends less than half of time on operations. Automation is not flexible and is vulnerable to changes in policy or infrastructure.

4 - Measured

All high burden tasks are automated. Team spends < 20% of time on operational work.

5 - Continuous Improvement

All automatable tasks are automated, team focuses on higher level automation and scalability. In-place automation is quickly adaptable to service or policy change.

Emergency Response

Noticing and responding effectively to service failures in order to preserve the service's conformance to SLO. Examples: on-call rotations, primary/secondary/escalation, playbooks, wheel of misfortune, alert review.

Organize Oncall Rotation

Responding to alerts

- Do you have an incident management protocol for major incidents?
  E.g., designating an incident manager, initiating cross-company communications, defining other roles to manage the crisis.
- Does your service have playbooks for specific situations?
  A playbook is documentation that describes how to respond to a particular kind of failure, e.g. failing over your service from one region to another.
  - Do the majority of playbook entries provide clear and effective response actions?
    I.e., actions that will mitigate the problem, and aid diagnosis or resolution of the cause.
- Does your service have an escalation process that is routinely followed?
  I.e., if you know you can't fix a problem, is there a process by which you can find and contact someone who can?
  - Is the escalation process documented?
- Can first responders resolve 90% of incidents without help?
  E.g., are the incidents resolved without escalating to subject matter experts / co-workers?

Oncall rotation structure

Procedures and expectations around oncall responsibilities for a clearly identifiable trained team of engineers, sustainably responding to incidents quickly enough to defend the service's SLO.

- Does your service have an oncall / pager rotation?

- Does your pager rotation have a documented response time?
  A response time is "time-to-keyboard" not "time-to-resolution".
    - What is the response time, in minutes?
- Do your oncall team members perform handoffs on shift changes?
- Does your team have a process for adding new responders to the oncall rotation?
  E.g., training process, mentorship, shadowing the main responder.
    - Is the training material documented?
    - Does your team have an "oncall pairing" scheme, where an experienced responder is paired with a new responder for training purposes?
- Does your team perform "wheel of misfortune" or other oncall training exercises?
  Wheel of misfortune: *production disaster role-playing, for training and to review and maintain documentation.*

Maturity Levels

1 - Unmanaged

Ad hoc support: best effort, daytime only, no actual rotation, manual alerting, no defined escalation process.

2 - Managed

Documented rotation and response time. Automated alerts integrated with monitoring. Training is ad hoc.

3 - Defined

Training process exists (wheel of misfortune, shadow oncall, etc.) Rotation is fully staffed but some alerts require escalation to senior team members to resolve.

4 - Measured

Measurement of MTTR and MTTD for incidents. Handoff between rotations. Most incidents require minimal escalation (can be handled by oncall alone).

5 - Continuous Improvement

Weekly review of incidents and refinement of strategy, handoffs, communication between shifts, majority of problems resolved without escalation, review value and size of rotation, evaluating scope. Incident response protocol established.

Alert Analysis

Review of alerts received in practice, coverage of existing systems, what are the practices and processes for managing alerts, the number of actionable alerts, ability to characterize events by cause, location, and conditions.

- Are your alerts automated, based on monitoring data?
- Does your team get a consistent volume of alerts?
  "Yes" indicates that most oncall shifts have a comparable number of alerts.
- Does your team have a process for maintaining sublinear alert growth?
  I.e., as your service grows, whether through organic usage growth or the addition of new components/flags/features/etc., do you take deliberate steps to prevent a proportional increase in the alert volume?
- Does your team have automated alert suppression rules or dependencies to reduce the number of alerts?
  E.g., when a load-balanced server has been drained, alerts for that server might be automatically suppressed; or, finer-grained alerts such as specific latency thresholds might have a dependency on coarser-grained "unreachable" alerts being silent.
- Does your team routinely ignore or manually suppress any alerts, because they are noisy, spammy, spurious, or otherwise non-actionable?
  E.g., do you have an alert that has been firing every few days for months, and is routinely resolved with no action taken? It's OK to have spammy alerts in the short term, as long as you're fixing them.
- Does your team spend significant time outside of an oncall shift working on incidents that occurred during their shift?
- Does your team have a defined overload process for periods when alerts fire at unsustainable levels?
- Does your service experience major outages that aren't initially detected by your alerts?
  I.e., outages where no alerts fired, or your alerts fired only after the incident had been discovered through another source.
- Does your team collect data on alert statistics (causes, actions taken, alert resolutions) to drive improvements?
- Do you have a recurring review meeting where alerts or incidents are reviewed for pattern matching, occurrence rate, playbook improvements, etc?

Maturity Levels

1 - Unmanaged

Pager is overloaded, alerts are ignored, alerts silenced for a long time, no playbook, volume increasing or unpredictable, incidents occur with no alerts (incidents detected manually).

2 - Managed

Incident rate unsustainable (oncall members report burnout). Bogus alerts are silenced. Playbooks exist, but often provide little guidance on required actions. Many alerts are 'informational' and have no explicit action. Alert volume unpredictable.

3 - Defined

Most alerts have an associated playbook with clear human action. Alert volume does not vary significantly between shifts / weeks. Alerts are at sustainable level (measured by oncall team) and a process is in place for when alert volume crosses overload thresholds (e.g. developers stop developing and work on reliability.)

4 - Measured

Useful playbook entry for most alerts. Almost all alerts require thoughtful human reaction. Top causes of alerts are routinely analyzed and acted upon. Alert suppression actively used to eliminate duplicate pages and pages already alerting another team. Process for sustaining sub-linear alert volume in the face of service growth.

5 - Continuous Improvement

Identification of patterns of alerts, periodic review of failure rates, prune alerts, review of essential service failure modes, all alerts require thoughtful human intervention, alert playbook provides appropriate entry point for debugging.

Postmortems

Policy of writing postmortems, with a given format and expectations for action items and followup. Practice of root causing issues and using the findings to drive service reliability improvements.

- Does your team have a postmortem process?
  Postmortems might also be known as incident reviews, retrospectives.
- Is the postmortem process invoked only for large / major incidents?
- Does your postmortem process produce action items for your team?
    - Does your team have a process to prioritize and complete postmortem action items?
- Do the majority of incidents receive a thorough root cause analysis, with a clearly identified result?
- Does your service have a process for prioritizing the repair or mitigation of identified root causes?
- Which of these, if any, are measured as part of your postmortem process?
    - Time to Detection
      The elapsed time for the incident from onset to detection.
    - Time to Repair
      The elapsed time for the incident from onset to resolution.
    - None of these
- Does your business have a process to share postmortems between organizations / teams?

- Does your business collect postmortem metadata (such as root cause categorization, MTTR, MTTD)?
  MTTD: *Mean time to detection. The average elapsed time for incident discovery, from onset to detection.*
  MTTR: *Mean time to repair. The average elapsed time for incidents from onset to resolution.*
  - Does your business have a process for using this data to identify problem areas?
    E.g., is the data categorized and/or aggregated to identify patterns of failure, or higher-risk service aspects, to direct investment in risk mitigation?
- Does your team or your business have any ownership or review cycle for the postmortem process?
  I.e., does anyone standardize the format and periodically evaluate its effectiveness and value?

Maturity Levels

1 - Unmanaged

No followup or recognition of systemic error, no root cause, outages contained but not analyzed, the same incident keeps happening, long term trends are not recognized.

2 - Managed

Defined postmortem process with action items. Followup on action items is poor (only P0 items addressed.) Root cause analysis (RCA) insufficient (not enough Whys). Similar incidents recur due to failure of action item followup or poor RCA.

3 - Defined

Postmortem process is applied to all major incidents (w/action items). Action items at P0 level are prioritized. RCA is extensive and correctly attributes root cause in the majority of incidents.

4 - Measured

Postmortems come with annotated metadata to facilitate analysis. Postmortems are shared widely amongst affected teams to learn from mistakes.

5 - Continuous Improvement

All action items closed in a timely manner, reviewed by involved teams and others for learning purposes, identify problem areas, process in place to make sure action items are completed, standardized format, review of the postmortem process to see if it provides value.