# Modeling Alert Quality

Moshe Zadka – https://cobordism.com

**Acknowledgement of Country**

Belmont (in San Francisco Bay Area Peninsula)
Ancestral homeland of the Ramaytush Ohlone

## 0.1 What are alerts?

Before I talk about alert *quality*, I want to make sure we are all on the same page: what are alerts? Whether good or bad, it is important to distinguish them from other things.

**What are alerts?**

Good or bad

### 0.1.1 Monitoring

There can be different models of alerts. This is a specific, but popular model. If alerts are to be about a system, then the system has to send some monitoring, or observability, data somewhere.

If we want more than a single data point to cause an alert, this data sink has to aggregate the data. This is going to be the *source* of the alerts.

**Monitoring**

System → Aggregator

### 0.1.2 Event

A system that constantly alerts is as useless as a system that always alerts. If alerts are based on data, and should not be sent all the time, we can model an alert as a kind of *event*. The definition of an event is that some query against the aggregated data returns an atypical value.

But not events are alerts!

**Event**

Aggregator query atypical value

### 0.1.3 Alert: Low priority

A *low priority* alert is when an event indicates a *problem*, but the problem is not urgent. This kind of alert usually shows up in e-mail, as a task, or in slack.

**Low priority alert**

Bad event (not urgent)

### 0.1.4 Alert: High priority

A *high-priority* alert is an event that we have decided should require an immediate action. This kind of alert will usually text, ring, or otherwise cause sounds, lights, and vibrations from a mobile device.

This is intended to draw immediate attention, potentially waking someone up. These alerts, high priority ones, are the focus of this talk.

It is those alerts whose quality we want to measure.

**High priority alert**

Break-fix needed!

Focus of this talk

## 0.2 What is alert quality?

So now that we have decided what alerts to measure, what exactly will we measure? It is useful to break the measurement into measuring three kinds of alarms:

- True alarms: Those are alarms that indicated a real problem that needed to be fixed.
- False alarms: Those are alarms that happened, despite there not being a problem, or a problem that had to be fixed immediately.
- Missing alarms: Much like the curious case of the dog in the night time, alarms that *do not* exist are just as important as those which do. A missing alarm is a problem that, in retrospect, needed to be fixed urgently, and yet no alarm was sent.

Distinguishing these alerts from each other can only be done retroactively. If you knew an alert was a false alarm, there would be no need to send it! This leads to an important aspect of alert quality: it can only be measured retroactively.

**What is alert quality made of?**

True alarms

False alarms

Missing alarms

### 0.2.1 True Alarm

What parameters would make up the quality of a true alarm? In other words, what numbers are good when they go in one direction and bad when they go in another?

Right now we are not focused on how to trade them off, let alone improving! All we want here is to capture the data.

The most important thing about a true alarm is the latency of the alert. So important it is, in fact, that it is useful to break it down:

- From the beginning of the issue to the detection
- From the detection to someone acknowleding it
- From acknowledgement to having some diagnosis (enough to fix)
- From initial diagnosis to remediation.

Though the aggregate matters, breaking it down gives better insights into where the problems are.

**True Alarm**

- Start to detect
- Detect to acknowledge
- Acknowledge to diagnosis
- Diagnosis to remediation

### 0.2.2 Missing Alarm

A missing alarm, by definition, still caused a problem that needed fixing. Since this is the measurement stage, this problem has already been fixed.

This means that the very same parameters can be measured for the missing alarm: latency to remediation, broken down by similar metrics.

Indeed, one of the measurement of a true alarm should be to improve at least one of those. The glaring one is "time to detect", but this needs not be the only one.

A true alarm could route better, improving detection to acknowledgement. It could add diagnostic information, allowing faster diagnosis. It could point to the right runbook, allowing faster remediation.

**Missing Alarm**

- Start to detect
- Detect to acknowledge
- Acknowledge to diagnosis
- Diagnosis to remediation

### 0.2.3 False Alarm

A false alarm is one, by definition, that did not have any remediation involved. This means that the latency for a false alarm is from the detection to diagnosis.

In this case, there are fewer steps. It is still worthwhile to break down the times from detection to acknowledgement, and from acknowledgement to the "all clear" diagnosis.

**False Alarm**
Detect to acknowledgement
Acknowledgement to diagnosis

### 0.2.4 Cost of alerting

A **false** alarm is one where no incident happened. In contrast, a **useless** alarm is one that indicated an incident that had already been alerted on.

This might mean a previous alert has already indicated the incident, or it might mean a human became aware of the issue in some other way. For example, users reporting issues in an application falls under this bucket.

Both kind of alerts are overhead: had their not been an alerting system at all, they would not have been sent, with no degradation to the service provided.

**Alerting costs**
False alarm
Useless alarm

### 0.2.5 Cost of not alerting

If there is only measurement of costs of alerting, than the incentive is to never alert. People have the revealed preference of building alerting systems.

The reason is because there is a cost to *not* alerting. A missing alarm can result in increased remediation latency. Measuring both the total increase, and breaking down the increases, is useful.

**Non-alerting costs**
Extra time to remediate
Broken down

### 0.2.6 Alert quality as cost

Putting those two costs together allows modeling alert quality **as** cost. The total cost of alerting, plus the total cost of not alerting, is an "anti-quality" measurement.

In order to get a quality measurement, negate it. If dealing with negative numbers is too depressing, since the best is zero, add a large constant.

The important thing about alert quality is how to improve it, so adding a constant value does not change the resulting actions. It is, sometimes, nicer to avoid having to say "OKR is to reduce alert value by 10%".

**Alert quality as value**
Cost of alerting
plus cost of not alerting
Negated
Plus a constant

## 0.3 Cost of alarm

In order to model alert quality as (anti-)cost, we need to measure cost. *Measuring* is always and forever a process of *estimation*.

In other words, measuring alerting costs means gathering the data from the incidents, and estimating a cost per alert. This can mean that sometimes it's useful to give a cost not in dollars, but in some fake currency.

This can sometimes communicate better *systemic* estimation errors. Systemic errors end up not changing the suggested actions or overall feeling, so reducing those might not be as useful.

**Breaking down alerting costs**
Data → Estimation

### 0.3.1 False alarm

Since a false alarm does not result in any degradation, any amount of people involved in it is "wasted". Because of that, totalling the total amount of person time invested in the diagnosis is important.

**False alarm**
Number of people
Time

### 0.3.2 Convenience

Alerts, false, true, or missing, can have different levels of "convenience". Think of convenience as "amount of engineer dissatisfaction" or "burn-out factor".

This convenience can depend on various aspects, and ultimately on the people involved. Is an alert on Saturday at 4pm worse than one on Tuesday at 2am?

It can also depend on what else these people are doing. Especially an alert which adds distraction or context-switching overhead to an engineer can be harmful.

**Alarm convenience**
Off business hours?
Delaying critical project?

### 0.3.3 People involved

The *number* of people involved is also an important cost metric. How many people needed to get involved? Across how many teams? Was it to find the responsible party or to get help?

**People diagnosing and remediating**
Interaction with other teams?
Finding responsible party?

### 0.3.4 Work involved

How much work was involved in remediation? By whom? *Where* was this work spent: diagnosis, test, deployment, etc. This is useful to make sure all remediation work was involved.

**Work diagnosing and remediating**
Work to diagnose
Work to test
Work to deploy

## 0.4 Cost of incident

Separate from the cost of the *alert* itself, it is important to measure the cost of the incident. An incident means that a service that someone cared about was degraded.

How much caring, how much degradation, and how long, is important. This is separate from how much *work* it took to remediate.

For example, a one-line fix that took an hour for the automated deployment to finish might have little work, but a lot of time to remediate. In contrast, a fix that took five minutes to deploy, but three people over twenty minutes to develop, takes more work, but takes less long to remediate.

**Incident cost**
Separate from work on incident

### 0.4.1 Time to detect

The time to detect is another thing that can only be determined retroactively, and sometimes, even then only approximately. This is the time from when a problem *started* to the time the alert was sent. At this point, the problem is still not clear: someone needs to ascertain it is not a false alarm, figure out what is going on, and how to fix it.

**Time to detect**
Unknown problem

### 0.4.2 Time to acknowledge

The time to acknowledge is counted from the time someone was notified until they acknowledged it. This measures the time needed to take up, if needed, and get enough devices enough access to confirm that the alert has activated.

Modern alerting systems make it easy to "acknowledge" an alert. This time can rise if on-call rotation is not optimal, or if people are not aware that they are on call.

**Time to acknowledge**
Time until confirmation of detection

### 0.4.3 Time to remediate

The time to remediate is counted from the time someone acknowledged the problem, until the time a solution is in place. This can be tricky to count, especially as partial mitigations are put in.

If there's some sense of the "partiality" of a mitigation, it can be scaled. For example, remediation that fixed 50% of the problem after 1 hour, and then fixed the problem fully in another one, is considered to have taken 1 and a half hours.

**Time to remediate**
Known problem

### 0.4.4 Cost

An incident's cost to the business can be broken down into two: immediate cost and reputational cost.

**Cost**
Immediate
Reputational

The immediate cost is the value of business lost due to the degradation. If there is a payment, or discount, because of missed SLAs, this falls under this category. If there an immediate loss of income, such as lost sales revenue or ad revenue, this is also part of the immediate cost.

Immediate costs are more straightforward to estimate. They are also, usually, not the bulk of the cost.

**Immediate cost**

SLA missed

Business missed

Reputational cost is everything else. Bad customer feedback undoing PR costs. Customers leaving, or reducing their expenditures. Finally, an incident can result in lowered future customer acqusition.

These costs are incredibly hard to estimate. They are also important to estimate, when building a business case for investing in better alerts: ones that reduce duration of degradation as well as its impact.

Problems being difficult are not a reason to avoid them. They are a reason to work harder, and learn more.

**Reputation cost**

Customer feedback

Customer continued business

New customer acquisition

### 0.4.5 Secondary incidents

Finally, often while trying to fix one problem, or diagnose it, other incidents are caused. This is normal, expected, and unfortunate.

The costs of these secondary incidents should also be measured.

**Secondary incidents cost**

Any degradation caused by remediations/mitigations

## 0.5 Balancing costs

Now that there is a sense of how to measure the current cost of alerts, it is important to see what it is, and how it tracks with other costs. This is the baseline from which to improve, so it is important to understand it. It is also important to understand what the value would be in improving the alerts.

**Balancing cost**

What would constitute "better"?

### 0.5.1 Gathering data

The first part is to gather the data. The value in breaking down the metrics is that, ideally, data on each one can be gathered separately.

Some data will not be available, or too hard to acquire. When you need to, give estimates. These estimates should be clearly marked, indicate what they are based on, and include confidence intervals.

**Gather data**

Estimate when you need to

### 0.5.2 Deciding on priorities

The next step is to consolidate the data, and deciding which parts of it are "bad". These are things that should be changed.

For example, it might be that the time to remediate is low, but time to detect is high. This means that time to detect should be decreased.

These priorities should first, and formost, be determined by business strategic needs. These strategic needs should be translated to the tactical goals: what metrics needs to be reduced, by how much, and what is the value of that.

**Priorities**

Strategy
Tactics

### 0.5.3 Tracking trailing OKRs

Gathering the data about an alert takes time. Getting a good enough sample of alerts takes time.

Any changes you make to alerts, monitoring, reliability, or on-call rotations will not have an immediate effect. Alert quality is a good indicator, but one that *lags* badly.

It is important track over the long term, but it is useful to have OKRs that are approximate, but easier to measure.

**Tracking quality**

Actual quality:Lagging indicator

### 0.5.4 Tracking immediate OKRs

There are several things that can be tracked more immediately. For example, counting false alarms is reasonably straightforward. Counting alarms on/off business hours is straightforward. Counting how many alarms had a useful runbook is straightforward.

None of those are *exactly* alert quality, but they give quick approximations that can be tracked. Over the long term, check which OKRs track quality most closely. If an OKR is not a good approximation, drop it or modify it.

**Tracking quality: immediate**

Approximate quality
Track that

### 0.5.5 Black swans

It is easy to overfit on alert quality. A "black swan" is a rare event with an outsized impact.

Even when measuring alert quality over a medium term, a once-in-a-year event is hard to account for. Those kind of alerts might have a high false alarm rate, but a true alarm can days of downtime.

There are no hard and fast rules here. Only to avoid blindly following OKRs and quality measurements, remembering that they cannot tell the future perfectly.

**Tracking quality: black swans**

Take into account wide "safety margins"

### 0.5.6 Goodhart's law

Alert quality should be carefully *not* made an instance of Goodhart's law. Tying people's compensations or performance reviews leads to endless litigation and arguing over definitions.

Since alert quality is a *lagging* indicator, usually this ends up tying comp/perf to approximations which are trivially gameable. This is a good way to shoot the alert quality down hard.

**Tracking quality: Goodhart's law**

Not a target

Feedback

## 0.6 Summary

### 0.6.1 Alert quality matters‘

Alert quality really matters. Bad alert quality leads to frustrated developers, and frustrated customers.

**Summary: Alert quality matters**

Burn out

Customer satisfaction

### 0.6.2 Alert quality take effort to track

Tracking alert quality takes real effort. Gathering the data, estimating costs, figuring out all the details is hard work.

It takes time and effort, and easy to deprioritize.

**Summary: Alert quality difficult to track**

Time and effort!

### 0.6.3 Improve and iterate

Putting in this effort is a long-term investment. It allows measuring, fixing, and measuring the improvement.

This is a process of constant improvement. The oppposite is a process of constant degradation of alert quality, as churn and legacy software build up. If there is no process to improve, eventually alerts will become completely useless.

**Summary: Alert improvement**

Measure
Fix
Iterate