15

Ramping Experiment Exposure: Trading Off Speed, Quality, and Risk

> The real measure of success is the number of experiments that can be crowded into 24 hours
> – *Thomas A. Edison*

Why you care: *While experimentation is widely adopted to accelerate product innovation, how fast we innovate can be limited by how we experiment. To control the unknown risks associated with new feature launches, we recommend that experiments go through a ramp process, where we gradually increase traffic to new Treatments. If we don't do this in a principled way, this process can introduce inefficiency and risk, decreasing product stability as experimentation scales. Ramping effectively requires balancing three key considerations: speed, quality, and risk.*

What Is Ramping?

We often talk about running experiments with a given traffic allocation that provides enough statistical power. In practice, it is common that an experiment goes through a *ramping* process to control unknown risks associated with new feature launches (aka. controlled exposure). For example, a new feature may start by exposing the Treatment to only a small percentage of users. If the metrics look reasonable and the system scales well, then we can expose more and more users to the Treatment. We ramp the traffic until the Treatment reaches desired exposure level. One of the most known negative examples is the initial launch of Healthcare.gov. The site collapsed as it was rolled out to 100% users on day one, only to realize that it wasn't ready to handle the load. This could have been mitigated if they had rolled out the site by geographic areas or last names A–Z. Insisting on a ramping process became a key lesson for subsequent launches (Levy 2014).

How do we decide which incremental ramps we need and how long we should stay at each increment? Ramping too slowly wastes time and resources. Ramping too quickly may hurt users and risks making suboptimal decisions. While we can democratize experimentation with a fully self-served platform as described in Chapter 4, we need principles on how to ramp to guide experimenters and ideally, tooling to automate the process and enforce the principles at scale.

We primarily focus on the process of ramping *up*. Ramping *down* is typically used when we have a bad Treatment, in which case we typically shut it down to zero very quickly to limit the user impact. In addition, large enterprises usually control their own client-side updates, so they are effectively excluded from some experiments and ramping exposure.

SQR Ramping Framework

In the ramp process, how can we iterate fast while controlling risk and improving decision quality? In other words, how do we balance **speed, quality,** and **risk (SQR)**? Consider why we run controlled online experiments:

- **To measure** the impact and Return-On-Investment (ROI) of the Treatment if it launched to 100%.
- **To reduce risk** by minimizing damage and cost to users and business during an experiment (i.e., when there is a negative impact).
- **To learn** about users' reactions, ideally by segments, to identify potential bugs, and to inform future plans. This is either as part of running any standard experiments, or when running experiments designed for learning (see Chapter 5).

If the only reason to run a controlled experiment is to measure, we could run the experiment at the maximum power ramp (MPR)[1], which often means a 50% traffic allocation to the Treatment providing the highest statistical sensitivity, assuming our goal is to ramp that Treatment to 100%. This gives us the fastest and the most precise measurement. However, we may not want to start at MPR – what if something goes wrong? That is why we usually

[1] If the experiment has the entire 100% traffic with only one Treatment, the variance in the two-sample t-test is proportional to $1/q(1-q)$, where q is the treatment traffic percentage. The MPR in this case has a 50/50 traffic allocation. If there is only 20% traffic available to experiment between one Treatment and one Control, the MPR has a 10/10 split, and so on. If there are four variants splitting 100% traffic, then each variant should get 25%.

start at a small exposure with the goal to contain the impact and mitigate potential risk.

We may also need intermediate ramp stages between MPR and 100%. For example, for operational reasons we may need to wait at 75% to ensure that the new services or endpoints can scale to the increasing traffic load.

Another common example is to learn. While learning should be part of every ramp, we sometimes conduct a long-term holdout ramp, where a small fraction (e.g., 5–10%) of users do not receive the new Treatment for a period of time (e.g., two months) primarily for learning purposes. The goal is to learn whether the impact measured during MPR is sustainable in the long term. See more discussion in Chapter 23.

Four Ramp Phases

Figure 15.1 illustrates the principles and techniques for balancing speed, quality, and risk in the four ramp phases. For more discussion, see Xu et al. (2018).

For simplicity, let's assume that our goal is to ramp a single Treatment to 100%, so the MPR has 50% Treatment exposure. Putting all the pieces together, the SQR framework divides the whole ramp process into four phases, each with a primary goal.

The first phase is mainly for risk mitigation, so the SQR framework focuses on trading off speed and risk. The second phase is for precise measurement, so the focus is on trading off speed and quality. The last two phases are optional and address additional operational concerns (third phase) and long-term impact (fourth phase).

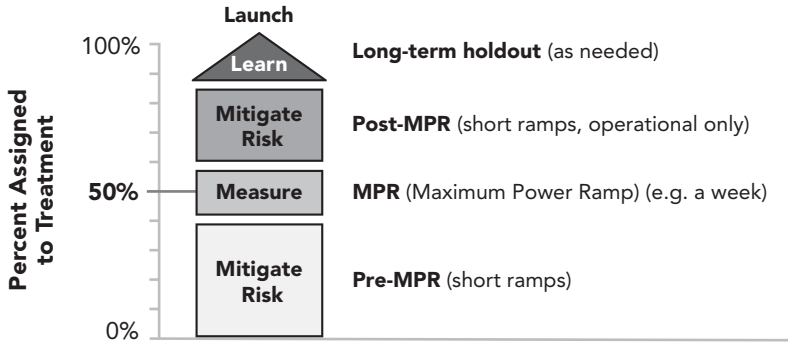

Figure 15.1 Four phases of the ramping process

Ramp Phase One: Pre-MPR

In this phase, you want to safely determine that the risk is small and ramp quickly to the MPR. You can use these methods:

1. Create "rings" of testing populations and gradually expose the Treatment to successive rings to mitigate risk. The first rings are usually to get qualitative feedbacks, as there simply isn't enough traffic to get a meaningful read on data. The next rings may have quantitative measurement but still be uncontrolled as the statistical power is low. Many bugs can be identified during the early rings. Note that measurements from the early rings can be biased as those users are likely the "insiders." Commonly used rings are:
   a) Whitelisted individuals, such as the team implementing the new feature. You can get verbatim feedback from your team members.
   b) Company employees, as they are typically more forgiving if there are bad bugs.
   c) Beta users or *insiders* who tend to be vocal and loyal, who want to see new features sooner, and who are usually willing to give feedback.
   d) Data centers to isolate interactions that can be challenging to identify, such as memory leaks (death by slow leak) or other inappropriate use of resources (e.g., heavy disk I/O) (see Chapter 22). At Bing, the common ramp-up is single-data center small traffic (e.g., 0.5–2%). When a single data center is ramped up to a decent traffic, then all data centers can ramp up.
2. Automatically dialing up traffic until it reaches the desired allocation. The desired allocation can either be a particular ring or a preset traffic allocation percentage. Even if the desired allocation is only a small percentage (e.g., 5%), taking an extra hour to reach 5% can help limit the impact of bad bugs without adding much delay.
3. Producing real-time or near-real-time measurements on key guardrail metrics. The sooner you can get a read on whether an experiment is risky, the faster you can decide to go to the next ramp phase.

Ramp Phase Two: MPR

MPR is the ramp phase dedicated to measuring the impact of the experiment. The many discussions we have throughout the book around producing trustworthy results directly apply in this phase. We want to highlight our recommendation to keep experiments at MPR for a week, and longer if novelty or primacy effects are present.

This ramp phase must be long enough to capture time-dependent factors. For example, an experiment that runs for only one day will have results biased towards heavy users. Similarly, users who visit during weekdays tend to be different from users visiting on weekends.

While we usually get smaller variance with a longer experiment, there is a diminishing return as we wait longer. In our experience, the precision gained after a week tends to be small if there are no novelty or primacy trends in the Treatment effect.

Ramp Phase Three: Post-MPR

By the time an experiment is past the MPR phase, there should be no concerns regarding end-user impact. Optimally, operational concerns should also be resolved in earlier ramps. There are some concerns about increasing traffic load to some engineering infrastructures that may warrant incremental ramps before going to 100%. These ramps should only take a day or less, usually covering peak traffic periods with close monitoring.

Ramp Phase Four: Long-Term Holdout or Replication

We have seen increasing popularity in long-term *holdouts*, also called *holdbacks*, where certain users do not get exposed to Treatment for a long time. We want to caution not to make a long-term holdout a default step in the ramping process. Besides the cost, it could also be unethical when we know there is a superior experience but deliberately delay the delivery of such experience, especially when customers are paying equally. Decide to do a long-term holdout only if it can be truly useful. Here are three scenarios where we have found a long-term holdout to be useful:

1. When the long-term Treatment effect may be different from the short-term effect (see Chapter 23). This can be because:
   a. The experiment area is known to have a novelty or primacy effect, or
   b. The short-term impact on key metrics is so large that we must ensure that the impact is sustainable for reasons, such as financial forecasting, or
   c. The short-term impact is small-to-none, but teams believe in a delayed effect (e.g., due to adoption or discoverability).
2. When an early indicator metric shows impact, but the true-north metric is a long-term metric, such as a one-month retention.
3. When there is a benefit of variance reduction for holding longer (see Chapter 22).

There is a misconception that holdout should always be conducted with a majority of the traffic in Treatment, such as 90% or 95%. While this may work well in general, for the 1c scenario discussed here where the short-term impact is already too small to be detected at MPR, we should continue the holdout at MPR if possible. The statistical sensitivity gained by running longer is usually not enough to offset the sensitivity loss by going from MPR to 90%.

In addition to holdouts at the experiment level, there are companies that have *uber* holdouts, where some portion of traffic is withheld from any feature launch over a long term (often a quarter) to measure the cumulative impact across experiments. Bing conducts a global holdout to measure the overhead of experimentation platform (Kohavi et al. 2013), where 10% of Bing users are withheld from any experiments. There can also be reverse experiments, where users are put back into Control several weeks (or months) after the Treatment launches to 100% (see Chapter 23).

When experiment results are surprising, a good rule of thumb is to replicate them. Rerun the experiment with a different set of users or with an orthogonal re-randomization. If the results remain the same, you can have a lot more confidence that the results are trustworthy. Replication is a simple yet effective way to eliminate spurious errors. Moreover, when there have been many iterations of an experiment, the results from the final iteration may be biased upwards. A replication run reduces the multiple-testing concern and provides an unbiased estimate. See Chapter 17 see more discussion.

Post Final Ramp

We have not discussed what happens after an experiment is ramped to 100%. Depending on the implementation details of the experiment (see Chapter 4), there can be different post ramp cleanup needed. If the experiment system uses the architecture that creates a code fork based on variant assignment, one should clean up the dead code path after launch. If the experiment system uses a parameter system, then cleanup would simply mean to use the new parameter value as the default. This process may be overlooked in fast-moving development process, but it is critical for keeping the production system healthy. For example, in the first case, it can be disastrous when a dead code path that is not being maintained for a while is accidentally executed, which could happen when the experiment system has an outage.