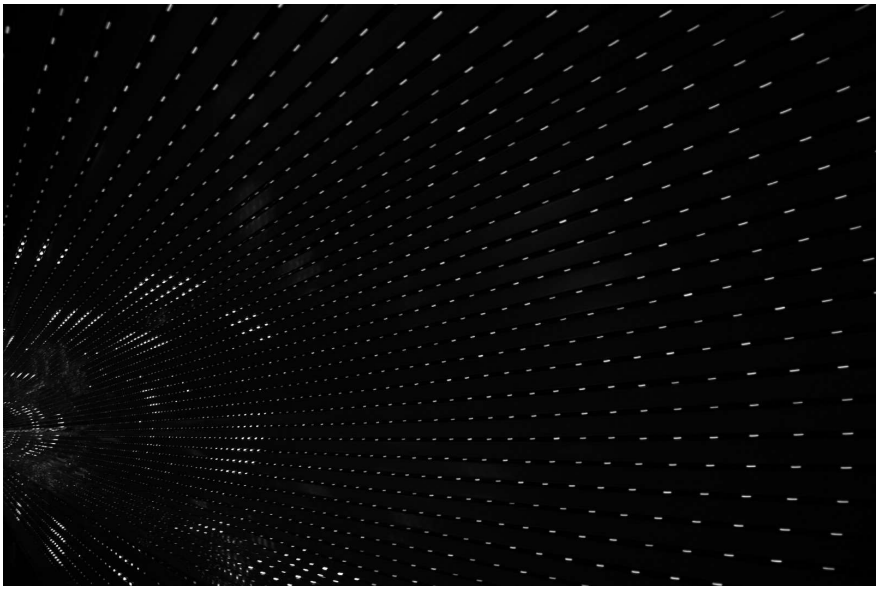
Mar 27 · 28 min read

The Effective Tech Lead is a 100x Engineer



Hyperbolic click-bait title aside, I wish to convince you that the effective Tech Lead is a force-multiplier and a unicorn-cousin of the famed 10x engineer, except the Tech Lead's power lies in the *transference* of that potent developer juju to each member of their team. The fortunate engineers who reside under the tutelage of these Tech Leads find their own powers magnified 10-fold and enveloped in a support-system unlike anything they've ever experienced.

And, I wish to convince you that this magical 10x aura need not come at the cost of one's well-being other than a few gray hairs on the Tech Lead's head. The Lead disposes of all wasteful "work" and vastly accelerates productivity for the work that *matters*, and returns to their team the option to *enjoy work life harmony*. Such is the boon for those paired with the effective Tech Lead.

> *The fortunate engineers who reside under the tutelage of these Tech Leads find their own powers magnified 10-fold*

I'm an Engineering Manager at Webflow. We believe our team's welfare directly translates into our user's welfare, so the more we spend on our team, the more our users benefit. We believe in providing meaningful work. We believe in lengthy, rich employment tenures—we are in this

for the long-haul. We believe that work should happily coexist *with the human experience*—think of it as a symbiotic, rather than parasitic, relationship.

The Tech Lead aids in the realization of these beliefs. So, I wrote a Tech Lead Guide to support our leads in this challenging pursuit, and have provided a near-verbatim replica for you here. We hope that you too can also enjoy the the meaningful pursuit of passion and success in a world that so often labels such a combination as lofty and absurd.

Please, enjoy our Tech Lead guide (Markdown with TOC version here).

• • •

The Webflow Tech Lead Guide

Welcome

Hello! 👋 Welcome to this document. Happy to have you.

If you've accepted a Tech Lead role, congratulations, you've demonstrated exceptional technical skills and a knack for leadership—rare qualities, indeed! Or, if you're curious about what the Tech Lead role offers and want to decide if it's right for you, you've come to the right place.

Assuming the responsibilities of a Tech Lead is a staggering leap from the average engineer's daily routine. This guide is here to quickly help you understand the

1. Definition of a successful Tech Lead
2. Expectations when moving from a purely technical role to one that is a blend of management and technical expertise, and how to manage those expectations
3. Strategies to mitigate common challenges
4. Stance Webflow takes on "management" *(hint: it's about service, not dictatorship)*

What's a Tech Lead, anyway?

By its most basic definition, a Tech Lead is "solely accountable for a project's success, spending 30% of their time writing code, and the other 70% managing a project". The *Lead* is no misnomer—your purpose is to navigate a team of talented engineers through choppy waters, direct them, steer them away from hazards, clear obstacles, and to keep them well fed with meaningful work.

This is easier said than done.

There are myriad skills a Tech Lead must possess and cultivate, but the most important are *sincere empathy*, *crystal clear communication*, and *technical excellence*. These skills are equally weighted. The Tech Lead is a "hybrid" role with one foot in management and the other in engineering, and acts as a liaison between project expectations and development tasks. A project's success is on the Tech Lead's shoulders, and it is on Webflow's shoulders to ensure they are excessively supplied with the support required to succeed.

Why management is different at Webflow.

Management has gotten a bad rap at most companies. It is often associated with treating employees as "cogs" and it conjures images of dictators with sun-eclipsing egos. This is not how Webflow operates. We view each team member as a *human being* first, and a talented contributor second. Humans need relationships built on compassion and cooperation. It is the Tech Lead's job to foster such an environment, and such environments are engendered through an attitude of *service*.

The Tech Lead's job is *not* to micromanage, but to be a service leader, which is to say they are there to *support* their team, to *serve* them as though they worked *for* them (not the other way around). They might be accountable for a project's success, but it is the collaborative effort *with* their team that brings a project to fruition.

Here are some hints to help approaching how best to serve a team:

1. Be direct with project needs. Do not fear to challenge your team as long as you care deeply about their welfare.
2. When successes occur, lavish your team with praise and give them credit for everything—without them success is impossible.

Why might I want to be a Tech Lead?

You may *not* want to be a Tech Lead, and that's just fine. Webflow seeks to provide many different opportunities for engineers to advance their career, including Individual Contributor tracks that offer similar significance to advanced management roles. The Tech Lead is under more pressure than the average engineer, and it is challenging to balance the demands of managing a team and contributing code, especially when first entering the Lead role (this is completely normal, by the way).

That said, management life can be extraordinarily rewarding. You will have input into decisions much higher up on the food chain. Your impact on Webflow's user base multiplies. You will develop clout that will reflect in your performance reviews, and subsequently, provide more opportunities for career growth. The role is often seen as a stepping stone to the title of "Senior" Engineer, as well as a prerequisite for an Engineering Manager position. You will mentor and help other engineers grow. Some find these added challenges exciting and help push *them* to new limits.

How can I become a tech lead?

Just ask! Yes, it's that easy. In your one-on-ones, express to your manager that you are interested in becoming a Tech Lead. It's your manager's duty to design a path to new roles, and, depending on your current experience, might include assigning you as a Tech Lead on your next project—and if not, then to provide you opportunities to develop the skills needed to become a Tech Lead.

What's expected of me when managing a team?

The Tech Lead's job consists of these responsibilities (in no particular order):

1. To work closely with a Product Manager to set reasonable expectations around deadlines, and to be *clear* when projects are going *off-track* (See: Help! We are behind schedule!)
2. To break up projects into digestible tasks, to tie those tasks to iterative deliverables, and to keep track of those deliverables
3. To provide ample uninterrupted work time for their team so they may frequently enter the flow state, and to act as their team's guardian against any potential blockers and distractions

4. To ensure your team is sufficiently supplied with work at all times so that no one "spins their wheels"
5. To perform diligent code reviews, first-pass QA, and to contribute code where possible
6. To be *available* to team members as they execute their tasks. (Windows of blocked time for heads down work is expected, but windows of team availability are expected, too)
7. To occasionally work with other departments

Working with Other Departments

Product Management aligns user expectations with product features. Marketing makes those features known to the world. Support ensures Webflow makes good on those promised features. Each is critical to Webflow's continued success and growth. Engineering is at the crux of these departments and the Tech Lead acts as the liaison between them.

The Tech Lead is responsible for communicating their project's status to other departments in two forms:

1. A weekly status meeting with their team in which a dedicated Product Manager or Support Liaison* may also participate. (See: Meetings) This meeting is mandatory regardless of Product Manager or Support Liaison participation.
2. A weekly "All Hands" report for the entire company to see. (See: How do I provide status reports to All Hands and Lattice?)

Some projects might not warrant a Product Manager or Support Liaison, and in these cases, the Tech Lead will express their team's status and needs to their Engineering Manager. On occasion, Marketing may also ask the Tech Lead when they should begin campaigning for a feature.

** The Stabilization Team (See: What are the different types of teams a Tech Lead can manage?) will work closely with a Support Liaison to focus on fixing bugs with the greatest user impact.*

Tracking Tasks

A great Tech Lead knows how to break a project into meaningful and easily digestible tasks (digestible means about three days scope). This gives their team members a holistic view of a project as well as a finish line, and allows the Tech Lead to assign tasks to team members each

week. Breaking a project down into small tasks is a time-consuming process, and is often an ongoing effort, but is critical in providing team members with a sense of progress. It also allows the Tech Lead to create waypoints toward unknowns, and to keep those unknowns contained to small time windows (See: Task toward unknowns)

Master Tracking Issue (MTI)

At the onset of a project, or at the onset of a project's continuing milestones, the Tech Lead must take time to thoroughly review the project's specifications and do their best to break down the specification into trackable tasks with a scope of 1–5 days of work (outside Code Review / QA), and an optimal timeline of 3 days. These tasks should then be grouped into Milestones. Each Milestone is a *deliverable* with a deadline date. (See: Milestones)

> Pro Tip: Consider enlisting your team to help you break down Milestones into tasks. This is sometimes the *only* option if you've got a team member with domain knowledge you don't possess. Delegate where it makes sense, but be sure to *review* all tasks and to *validate* their scope and/or assumptions.

Webflow's current practice is to create GitHub issues for every task that are then tracked in a "Master Tracking Issue". The MTI should receive a `[Master Tracking Issue]` label in the issue's title as well as in GitHub's label section.

The MTI is a centralized and clearly outlined view of GitHub issues that lists Milestones, their projected delivery date (See: Milestones), and their related tasks in a list that

1. Can be easily assigned to your team members who will then be responsible for opening a PR to close the issue
2. Displays the task's GitHub issue number *and* the PR that will close the issue, as well as a title for the issue. This is usually best accomplished in a tabular format.
3. Provides the estimated finish date for each milestone, and the status of each issue toward those milestones (See: Displaying Progress in the MTI)

Tasks

Each issue (or 1–5 day task) must clearly point to the portion of the specification the issue addresses *and* to the concerned areas of

Webflow's codebase (if they exist). We've found it is best for each task to

1. Clearly point to the original specification the issue addresses, with any *visual* content that will help an engineer complete the task, including screenshots/screencasts from the specification or from Webflow itself
2. List a best guess of TODOs to help the engineer build a mental model around the problems they must solve

Below is a task template. This should be located in a GitHub issue and should receive the same title that is tracked in the MTI.

> Master Tracking Issue: #00000 (Place the Github link here)
>
> **Objective**
>
> List the goal of the tasks here. It does not need to be long, and can take the form of a user story, e.g. "As a user, I would like to X, so that I can X", or "As a user, I would like to be able to right-click and delete an item, so that I don't have to move my mouse all the way up to the top of the screen."
>
> **Tech Spec**
>
> <Insert screenshots/wireframe/visual content of finished feature>
>
> *Clearly* outline the expectations for the tasks here. Place them in the form of TODOs. For example:
>
> - [ ] Include a "Delete" option in the right-click menu for item
> - [ ] Wire the "Delete" option to the DELETE_TEM system event
>   - [ ] Write unit test for delete operation
>   - [ ] User may *not* delete item if multiple items are selected
>
> Also add condition material, if needed:
>
> - [ ] When the user is logged into a free account, disallow deletion
>
> **Design Artifacts**
>
> Provide a list of design artifacts on which the above tech spec is based. This could be an external link to an artifact the Design or UX team provided. Include authors names so that the task owner can reach out.
>
> **Notes**
>
> Any clarifying content unrelated to the above items (Or, just a word of encouragement, like "You're doing great!")

This is tough

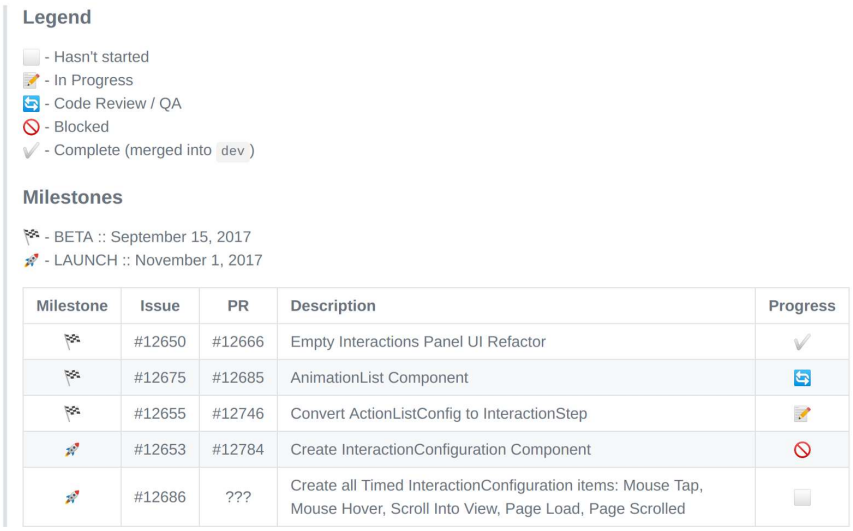
Creating the Master Tracking Issue will feel like it's taking too much time and will make you question whether or not you are performing the most effective work. Trust us: it *is* critical, and the clearer the MTI, the higher likelihood of a project's success. Depending on the size of the project, it could take upwards of a week or more. 😱 It's fine. Plan for it. Make it happen. Your team will thank you. It is crucial to helping your team feel a sense of meaningful progress (See: How do I keep my team motivated?).

> Pro Tip: It can be helpful to keep a document open beside the spec and to write down a list of tasks before beginning the MTI. When you've got a solid brain dump of tasks, open an issue, write a basic description and

> highlight the specification area, and *then* go into the codebase to find where to point the issue to.

Displaying Progress in the MTI

You can think of the MTI as a dashboard that displays the progress of every issue associated with a milestone. This, in turn, shows the status of *entire* milestones, and subsequently, the *entire* deliverable. For instance, here's an example of how an MTI might progress:



Legend

⬜ - Hasn't started
📝 - In Progress
🔄 - Code Review / QA
🚫 - Blocked
✔️ - Complete (merged into `dev`)

Milestones

🏁 - BETA :: September 15, 2017
🚀 - LAUNCH :: November 1, 2017

Milestone	Issue	PR	Description	Progress
🏁	#12650	#12666	Empty Interactions Panel UI Refactor	✔️
🏁	#12675	#12685	AnimationList Component	🔄
🏁	#12655	#12746	Convert ActionListConfig to InteractionStep	📝
🚀	#12653	#12784	Create InteractionConfiguration Component	🚫
🚀	#12686	???	Create all Timed InteractionConfiguration items: Mouse Tap, Mouse Hover, Scroll Into View, Page Load, Page Scrolled	⬜

The above gives a PM (or, anyone concerned) a quick way to gauge the progress of a project. For instance, one can see the BETA milestone is about 75% complete, and since tasks are broken into roughly 1–5 day increments, it is easy to tell if a milestone is going `off-track` (See: Help! We are behind schedule!).

It is up to the Tech Lead to maintain the status of the above MTI, though they may wish to delegate updating the status of each line item to the team member responsible for completing that issue. The important elements to display for each task are

- Its Milestone and date
- Its Issue
- Its Pull Request
- A short description
- Its Progress

> Pro Tip: If a single MTI grows too long and too unwieldy, it's fine to split them into separate MTIs.

Milestones (a.k.a. Deliverables)

The Tech Lead must keep their Product Manager (or Engineering Manager if no Product Manager is assigned) updated on how well they are tracking against Milestones, as well as provide weekly All Hands updates (See: How do I provide status reports to All Hands and Lattice?). These Milestones and their respective tasks are determined by the Tech Lead and confirmed by a Product Manager, Engineering Manager, or otherwise.

A "Milestone" is

- A *major* deliverable, usually with a six-week timeline (See: How long should projects take?)
- Responsible for driving a series of tasks/issues, and is complete when *all* tasks/issues have been pushed to production
- Named according to the type of deliverable, e.g. Phase, Launch, Version (See: Types of Milestones)
- Assigned a deadline date

The planning structure for a large project should only ever consist of two levels: Milestone -> Tasks. The Milestones themselves will be under the purview of a Feature, such as Rich Content Editor or Interactions 2.0, which may take months (or years) to complete. Milestones are "chunks" of continuously delivered work, and are usually accomplished sequentially. It is rare to have a team work on Milestones in parallel unless they are highly interrelated, though some overlap is expected when moving from one Milestone to another.

> Pro Tip: Be incredibly wary of scope increases. Scope creep is real. *Always* use a Milestone's date as the affected factor when scope changes, and clearly communicate the new scope's impact.

For more info on Milestone timing, See: How long should projects take?

Types of Milestones

Milestones are *major* deliverables and are *functionally* the same to each other, though they can be *semantically* separated into Phases, Launches, and Versions. It's important not to dwell too much on these

differences, but it can be helpful to name them accordingly for Product Managers and Marketing.

Term	Definition
Phase	Anything Marketing doesn't need to let users know about. These are nuts and bolts type milestones that don't introduce any major experiential changes to users. Phases take on the name of their goal, e.g. "IX2 Flux Integration", or "Storybook Components for Interactions 2.0".
Launch	Anything Marketing *needs* to know about so they can drum up the eyeballs. This includes alphas, betas, and official launches, and will require many weeks of lead time to prep marketing materials.
Version	This is another version of a launch for a feature *that has already launched*. So, for IX2, after the initial launch, we labeled the subsequent launches IX2.0.1, and so on.

Task toward unknowns

Milestones deadlines are hard to estimate, but Webflow asks that the Tech Lead do their best to place a *realistic* date on them. This constraint might seem limiting at first, but we treat deadlines more as focal points (with mitigation strategies) than immovable *dead*-lines (See: Help! We are behind schedule!).

Rather than rely on a Milestone's hazy, fog-covered finish line, it's much better to "task toward unknowns". Our features tend to forge new industry territory, the likes of which the JavaScript world has never seen, so it's often impossible to have a crystal ball view of upcoming work. Some of it will be clear, sure, but there will invariably be a portion of a specification that causes the best Tech Lead to scratch her head and say "Um, I have no idea how long this will take." Clear the haze. Shorten the forecast by breaking down the unknown into small tasks designed to uncover the unknown as soon as possible.

Be adamant when prioritizing your tasks. Pivot when more information arises. Let your PM know on which of these tasks your team is currently working. Stacking these unknowns is how *actual* Milestone deadlines are discovered.

> Pro Tip: Sometimes new tasks arise from uncovering unknowns that weren't outlined in the original MTI. It's fine to include new tasks if they are absolutely necessary to complete the Milestone. Be sure to inform your Manager if they alter the Milestone's deadline.

Meetings

The Tech Lead should organize one ~30-minute project meeting per week, preferably at the week's start and early in the day, whose agenda looks like the following:

1. Perform a Mini-retrospective that asks:
    i. What went well last week?
    ii. What didn't go so well last week?
    iii. How can we improve what didn't go well?
2. Ask each team member:
    i. What's the current status of your task?
    ii. Are you blocked?
    iii. How can I help unblock you? [**Tech Lead**]
3. Assign new tasks to each team member
4. Communicate the project's status to the Product Manager
5. Answer any questions and engage in light and witty banter

Limit team-wide meetings to this one weekly event. Hopping on a Slack call or a code pairing session should not be considered a “meeting” and should be employed liberally where needed.

Weekly Status

Every engineer is asked to report their `on-track` / `off-track` status each day to #status-frontend or #status-backend accordingly, and it is on the Tech Lead to confirm those daily (a Slack "reaction" 👍 is always nice). This holds each engineer accountable to their weekly tasks and it allows the Tech Lead to step in if a task goes wildly `off-track` or beyond 5 days.

> Pro Tip: Help your team members to focus on *one to three* concurrent tasks at a time. Any more than that is difficult to track, so offer to help reduce or combine their tasks and figure out what’s causing the fragmentation.

Agile? Project Managers?

You may be wondering, “Where’s the methodology behind this way of managing projects?”. It might resemble Agile, with its two-week forecasts and weekly “Scrum”-like meetings, but it lacks burn-down charts and Scrum Masters. While we love the agile philosophy, aim to move quickly, and pivot where possible, Webflow does not subscribe to a specific methodology. This is what works for us right now, and we are always open to reevaluating it as we go. 👍

What are the different types of teams a Tech Lead can manage?

Webflow arranges its talented engineers into *Action* and *Permanent* teams for which a single Tech Lead will be responsible.

Team	Description
Action	Assemble around a feature (or prototype) and disband on its completion.
Permanent	Assemble around a domain problem and continually work on it without ever disbanding, e.g. the Performance and Stabilization teams. Tech Leads and Team Members can rotate through these teams.

What size team should I expect?

Team sizes vary (they can even be a league of one), but the general rule is a team will include *three* members, including the Tech Lead. It is relatively easy to manage relationships with two individuals engaged in solving the same problems, but once someone is asked to manage a third, or fourth, or fifth relationship, the permutations of communication potentials grow drastically. This isn't isolated to the Tech Lead's relationship, but also to how the members of the team communicate with each other. Larger teams *can* work, but the rule of three seems to be a good starting point.

This isn't to say a *team* must have only *three* members. An Action Team might contain seven members, including a Tech Lead who can divide the team into two groups (of three) and focus each group on parallel tasks *within* the feature's overall scope. It is then up to the Tech Lead to create a single Team Lead for each group and hold them accountable for their group's work. Bear in mind that each group should be focused on *feature* efficiency and collaborate on solving problems *with* each other so as to reduce the blocking latency commonly encountered when parallelizing individual resources.

The aforementioned team structures can be comprised of Back-End *and* Front-End engineers. Webflow wants to blur the lines between these engineering disciplines, as well as non-engineering disciplines, e.g. designers. Forming cross-discipline teams is the end-game for feature efficiency; whether or not you pursue it is up to you and the demands of your project.

Team Efficiency and Organization

There are two ways of designing a team. One of "Feature" efficiency, which favors groups that collaborate on solving closely related problems *together*, and another of "Resource" efficiency, which favors individuals working on wholly unrelated tasks that run in parallel. Both have their strengths, but we ask that the Tech Lead optimize for *feature* efficiency where possible. See Flow vs. Resource Efficiency for more information. [We've replaced "Flow" with "Feature" in this article as it's easy to conflate Flow with the "Flow State"]

> Pro Tip: Parallelization requires well-defined scope. If you are leading a project that is iterating on design specs *while* iterative development occurs, it is best to only optimize for *feature* efficiency.

Help! We are behind schedule!

It's cool. Really. Go grab some coffee, or get some sun, and return to your desk when your inner self reflects the same glossy sheen as a calm pond (See: How can I stay "centered"?).

Pretty much every project encounters some unknown that threatens its delivery date. Instead of desperately trying to avoid this, try to *expect* this. You need to build it into your estimates (See: How can I make better estimates?). Recognize this as absolutely normal, and take comfort in the solidarity that all Tech Leads experience it. This is what separates the *good* from the *great*.

We equate missing deadlines with heart wrenching guilt. This is a morale killer. Morale is your team's most precious resource. Instead, it's best to think of "delays" as "deferred progress", and to pitch it as such. Webflow understands Software Development is tough, so we've got some tricks up our sleeves to help you frame missing a deadline as *progress*.

A Word on Project Management

Before we dive into our *Rework / Defer / Abandon* deadline model, there are two key project management concepts that will help you understand *why* we follow it.

First, it is important to emphasize the need to *tie deliverables to fixed dates*. Progress is hard to measure without a visible target. We must measure progress toward something, even if that something is just a guess. Progress is the lifeblood of motivation.

Second, there are four levers you can pull to help get a project back `on-track`. They are as follows

Lever	Description
Time	When the deliverable is launched
Quality	The craftsmanship put into the deliverable
Resources	The number of participants contributing to the deliverable
Scope	The breadth of what the deliverable is and does

These four levers can change as a project evolves. They are the tools effective Project Managers reason with. That said, Webflow produces the highest possible quality product and will not sacrifice Quality for Time, Resources, or Scope, so we only have those three levers available to us, which we will expand on in the next section.

> Pro Tip: The Tech Lead role is often an engineer's first foray into trying to meet the bottom-line needs of a business. Their decisions must be framed in the question: "How does this keep the company healthy?" If you've little or no business acumen, have a look at Josh Kaufman's The Personal MBA. It's a fantastic crash-course in modern business practices and will help you make better decisions when considering Webflow's needs and the needs of your team.

Rework / Defer / Abandon (Mitigation Strategies for Deferred Progress)

You have three options when confronted with a threatened deadline that should be discussed with your Product Manager. Here they are in sorted by order of consideration:

- Rework the deliverable
- Defer the deadline
- Abandon the project

Rework

Rework consists of asking two questions:

1. Can we add resources to the project to meet the deadline?
2. Can we change the scope of the deliverable to meet the deadline?

Questioning your resources and scope should be the first tool when evaluating how to mitigate a missed deadline. Ask first if more resources can help the situation, though this is usually *not the case* unless the project was initially understaffed to begin with. Adding late-stage resources can even push the deadline out farther! So, your next tool is to reduce scope.

> Pro Tip: Reducing scope is often the #1 choice when trying to hit a deadline while still providing business value. The likelihood a project requires more resources to hit a deadline is probably in the 10% range. Reduce scope 90% of the time.

Reducing scope is usually feasible. As passionate software developers, we tend to bite off more than we can chew. This is your opportunity to use a fork and knife to slice up the deliverable into bite-sized pieces with more realistic expectations, and for you to communicate those expectations to other key stakeholders.

Defer

If scope cannot be reduced, and adding resources isn't an option, the next *best* option is to *push the deadline out*. Yes, you heard it right. It's to *move* the deadline. "What's the point in deadlines, then, if they can just be moved all willy-nilly?" Well, we do our best to avoid moving deadlines, but sometimes it happens, and that's totally okay. Too much is at stake when we attempt to hit an unrealistic deadline, and among them are team burnout, poor product quality, reduced morale, and more.

The important idea here is *not to lose sight of a delivery date*. That's all that matters. Projects will fall into limbo when a missed deadline stays (ahem) dead and the project careens toward the unknown. This is *worse* than moving the deadline, so move it!

Abandon

The final and rarest option is to abandon the project altogether. Consider this if you (or another stakeholder) discover the deliverable will negatively impact the company. Scrap it! Focus on *efficient* work, not *productive* work.

An optional fourth strategy: Watch it closely

There is a fourth option, too, when the threat of a missed deadline is no more than a subtle twang in your gut, and that is to *watch it closely*. Pay special attention when your intuition whispers something's off. It's important to get ahead of the problem, and this should be the moment where you preemptively strike. Make your manager aware of it.

> Pro Tip: The key to making your and everyone else's life easier is to master the art of *managing expectations*. It is wise to under-promise and over-deliver as long as you remain candid and honest. Always state what is true. Announce worries about missing deadlines or losing a key resource. Announce wins about finishing work earlier than expected. Be as truthful as you are skeptical about unknowns.

How can I make better estimates?

At the time of this writing, no person has discovered a magic eight-ball estimation method for predicting software development timelines. Some might try to sell you snake-oil and tell you otherwise, and some might say it's downright impossible. It's best to accept that software estimation is rarely accurate and work from there. This is at the core of the Agile Philosophy: iterate and discover, then deliver and improve. It's an art of discovery, not an art of delivery. Webflow follows an iterative process (See: What's expected of me when managing a team?) as outlined in other sections, so estimation is important, but not as important as uncovering unknowns. That said, here are some tactics to help estimate tasks:

Pad estimates for the unexpected

Development rarely unfolds as planned. Instead of *precise* estimates, give your best guesstimate for a given task and multiply it times *four*, *especially* if that task involves uncovering an unknown. That might sound crazy—and sometimes it is; experience helps Tech Leads refine that equation—but it's a good starting point that leaves room for dastardly unknowns.

Add up tasks toward unknowns

Once you've created your Master Tracking Issue (See: What's expected of me when managing a team?), you can get a sense of how long the project might take. Be sure to identify which tasks are associated with *discovery* (finding unknowns), and which have more concrete definitions. Once you've completed all the discovery tasks, you will have a *much* better sense of the deadline's accuracy.

The 80/20 Rule

It is easy to overlook time-consuming nuances that slow the final 20% of a project. When you view your project holistically, break it up using the 80/20 rule, and consider that the final 20% of a project might account for *another* 80% of the overall timeline. There are a number of reasons for this, but the final 20% is often filled with polishing the deliverable, and complex features require polish for *every* feature and edge case, which compounds near the project's end.

What does this mean for you? Just treat the 80% point in your project as the halfway marker. That will align expectations against the added effort nuance prescribes.

Never forget QA

When you estimate deadlines, set a date for *code completion* so that QA can have time to discover any bugs or UX issues. Your estimates must consider this extra phase, and to consider QA's current workload.

Cooldown: Bug fixes after delivery

On delivery, plan to leave some time to fix any immediate bugs before starting new milestones. The amount of time can vary based on the deliverable's complexity, and a week is usually a good window. This is an opportunity to give your team some downtime before leaping into the next set of tasks, and it gives you a chance to tighten up the next milestone's MTI.

How long should projects take?

While the scope of a feature might require months and months of work, its versioned *milestones* should aim for six-week timelines, including QA, so each milestone is *code complete* around four weeks. This allows Marketing to evaluate a *proven*set of features and put them in their pocket, so to speak, and queue them for announcement based on market trends. Breaking a large feature into six-week timelines can appear challenging at first, but we ask this for a few important reasons:

1. It is much easier to reason about smaller scope and timelines
2. It allows projects to pivot if its business value somehow proves meager
3. It allows groups of three to move faster

A six-month project's *major* Milestones may then look like this:

1. Alpha Launch (6 weeks)
2. Beta Launch (6 weeks)
3. Feature Launch v1.0 (6 weeks)
4. Feature Launch v1.0.1 (6 weeks) 🏁

Should I branch off of feature branches or not?

Not.*

Do not branch off of `feature-branches`. Tech Leads should aim to have their team commit their `feature-branches` directly to `dev`

rather than to another `feature-branch` that is kept up-to-date with `dev`. Long-lived `feature-branches` often introduce code dependencies and other programming patterns that require cherry-picking and other *hard-to-keep-in-sync-with-other-branches* issues. Instead, the Tech Lead should place their project behind a *Feature Flag* and continually merge it with `dev`.

To summarize, Webflow has two *main* branches:

1. `dev`
2. `master`

And a `feature-branch`

1. May branch from: `dev`
2. Must merge back into: `dev`

Feature flags

We encourage all of our engineers to push code every day (if possible), and to prevent a new feature from stepping on the toes of our users, we suggest Tech Leads place those new features behind a "Feature Flag" that can be toggled with the ShortcutHelper.

> *Okay, there *might* be a case for a long-lived branch to which other branches commit. And by "might", we mean maybe 1% of the time where we must refactor a critical, widely-used portion of our infrastructure. So, basically never. 😊 Should the need for such a branch arise, please inform the *entire* team, your product manager, and your engineering manager of your intent. You may be surprised about how the work could be organized into smaller, continually merged branches.

How can I stay "centered"?

Staying "centered" means you take care of yourself first and foremost and find a "happy" place from which to approach solving problems. Life is about performing as much meaningful work as it is about performing meaningful *human activities*. This means you will need to take a break from your daily tasks and engage in activities that keep you fresh and focused. Does reading a book help you? Does binge-watching some Netflix? Does exercise? Fresh air? Find a routine that keeps you on point in work *and* in life, and don't be afraid to express those needs to

your manager, and never fear to make time for them, even if it feels like it's cutting into your productivity.

If you aren't centered, your team won't be centered. Lead by example.

How do I stay organized?

New Tech Leads feel overwhelmed, and if they don't, then they probably aren't performing some part of their job. 😅 (Okay, fine, some of us may be able to take the role in stride, but it's uncomfortable for most). The key to mitigating the dreaded stress of *too much*, is to learn the art of time management. This can take shape in many ways, and it boils down to your own preferences. If you've never picked up a book on time management, we recommend starting with David Allen's Getting Things Done. It's a great first step to learning how to transfer the cacophony of noise in your head elsewhere. If his method doesn't work for you, seek to find another and share it when you do.

How much should I expect to code?

This depends on the project, but a good estimate is that you will code 30% of the time (if not fewer), *review* code 30% of the time (if not more), and serve your team with your remaining time.

Code Reviews

Since you are ultimately responsible for the quality of the deliverable, you will want to review and sign off on every PR. This can be incredibly time consuming on larger teams, so it's good to encourage your team to review *each other's* code. That said, expect to perform *a lot* of code reviews, and look at them as an opportunity to mentor junior team members, and with senior team members, to keep you on top of your skills.

How do I provide status reports to All Hands and Lattice?

Every Thursday at 11am PST (as of this writing), Webflow holds an "All Hands" meeting where the management team relays the status of all of Webflow's ongoing projects as well as large company goals and initiatives. It is the Tech Lead's responsibility to provide a progress update for their project to the Webflow Project Tracker Google document *prior* to this meeting. This document is shared in the #all-hands channel in Slack. A template for the updates is located at the end

of the Google document. Please follow it accordingly. The items in the template are

1. TDLR, or a brief blurb on the project's state of affairs.
2. MILESTONE ON-TRACK/OFF-TRACK, where you provide the track updates for each active milestone, their percent progess, and the percent change from the previous week (these are guesstimates). Also list out the next two weeks of tasks the team will work on and their expected delivery dates.
3. KEY DECISIONS, where you mention any big key decisions that lead to timeline changes, scope changes, and anything that relates to support/marketing, or change in resources.
4. RISKS, UNKNOWNS, AND BLOCKERS, where you mention any risks, unknowns, or blockers that appeared since the last week.

Lattice

Webflow uses Lattice to help track higher level company goals. In addition to your weekly All Hands updates, we will ask that you also update any Lattice goals that are assigned to you. If you do not have an account, reach out to your Engineering Manager for help.

How do I keep my team motivated?

Engendering a sense of progress, and giving sufficient room for creative problem solving without dictating *how*, motivates humans more than money, or any carrot and stick. We are intrinsically motivated creatures with simple heuristics: If you place realistic goals in front of us, the tools to do it, and a sense of purpose for why we should, we will move mountains.

Science has given us some key insights into what motivates humans. Many of the concepts in this document are built on top of those insights, so you've already been employing tactics to keep your team motivated! That said, here are some of the underlying mechanics of our process.

Autonomy, Mastery, and Purpose

Daniel Pink, in his book Drive, dispelled the myth that humans are extrinsically motivated, or that is to say motivated by *external* factors such as money or nicer offices, job titles, etc. Instead, he found that we are motivated by *internal* (or intrinsic) factors, such as a being given a sense of belonging, opportunities to grow skills, and to do so on our

own terms. These three intrinsic factors can be boiled down to Autonomy, Mastery, and Purpose, and are excellent starting points for dissecting the basics of motivation.

Part of providing these key motivators falls on Webflow's shoulders, but a clever Tech Lead can use them to great effect, too. So, every week ask yourself these questions:

1. Am I giving my team enough room to solve problems on their own terms? Am I dishing out commands when I should be providing direction and intent? [Autonomy]
2. Am I placing my team members on the right tasks that can help them grow? [Mastery]
3. Am I aligning *why* we are building this feature with *how* Webflow wants to help the world? [Purpose]

Provide Feedback

Kim Scott, a Harvard grad that served as an executive at Google and Apple, sums up how to best manage the relationships and expectations with each individual on your team in her book Radical Candor. It turns out we *shouldn't* water down how we feel and what we say to each other, but instead we should frame tough discussions in a personal and caring way. The basic premise of this axiom is to "Care personally, Challenge Directly", which means you must *empathize* with your team and demonstrate to them that you care about their welfare, but still provide them critical feedback (that might hurt).

By providing critical feedback early and often, *and* by demonstrating how much you care for people, you will sidestep catastrophic challenges later down the road. Also, this doesn't just apply to *negative* feedback, but also *positive* feedback, too. Both are crucial. Consider picking up her book for more information.

> Pro Tip: The way in which we *frame* feedback can make all the difference to how well it is received. Instead of attacking personal flaws, highlight the *behavior* that lead to the feedback. Consider using the Situation, Behavior, Impact model for such framing. It works like this: Bring up the situation where the behavior occurred, highlight the behavior, then mention the impact, e.g. "During today's meeting, you interrupted Brian multiple times, and made Brian feel like he couldn't speak up until the meeting's end where he presented the winning idea. This made the meeting longer than it needed to be." Here's a great guide if you'd like to learn more.

Flow

It's important to stress the need for each of your team members to have ample opportunity to enter Flow. This, in and of itself, is enough to keep most people happy at work *and* in life. It's such a critical factor in motivation and in work *efficiency* that we've listed it here as a reminder.

I have an under-performing member on my team. What do I do?

Have you heard the old adage, "There are no bad employees, only bad managers?" Well, it's mostly true. Webflow hires talented engineers, so before you jump to any conclusions about what's wrong with an under performer, make sure you are servicing your team 100% (See: How do I keep my team motivated?).

Each team member must be sufficiently motivated through ample opportunity for producing meaningful progress, autonomously, with room for mastery, and with a sense of purpose. Providing continual feedback is also an essential ingredient.

You also must consider a team member's *inner work life.* It's okay to ask, "How are things? Everything all right outside of work?" You *should* ask these questions often, but remember not to pry. Give your team members room to discuss personal issues while remembering they are *personal*.

If you've done your best to foster the right environment for your team member to do their best work, and they *still* aren't meeting your expectations, have a chat with your manager about what to do next.

How can I avoid burning my team out?

If a team can't meet a deadline, it's a *management* problem, and not the team's problem. This means that, somewhere along the way, the project didn't go as planned and a course wasn't corrected. So, *Rule Number 1* to avoid burnout is "Manage the project and expectations well" (See: Help! We are behind schedule!).

Rule number 2: Never ask more of your team than you would ask of yourself (and you mustn't ask yourself to work nights and weekends). Other organizations might ask their teams to pull longer hours when the going gets rough. This is a laser-focused bullet train to attrition and long-term inefficiencies. Webflow cares deeply about its team, not only

professionally, but personally, so we must do our best to *manage our time well*.

Crunch Time

Oh, crunch time, you've haunted the best teams, and you are oh so hard to avoid.

As a Tech Lead, you will invariably run up against a deadline that's *just* within reach and may require slightly more effort to push it out in the last stretch. By *slightly*, we mean your team might need to put in a few more hours over their 40-hour work week. Yes, that's right. Our version of "crunch" isn't crazy hours that bleed into the evening or weekend. It's just a *few*. When people operate at their peak performance, where they engage in the flow state 2–4 hours a day, *they are incapable* of more work without drastic consequences. They should already be operating at peak efficiency, and asking more of them has severe diminishing returns and a detrimental impact to them personally, *and* to Webflow as a company.

Crunch time is real. Crunch time can be a symptom of poor management. We must do our best to limit these hyperactive periods to one or two times a year.

Recommended Books

Flow

Deep Work

Drive

Leaders Eat Last

The Manager's Path

The Progress Principle

Getting Things Done

Getting To Yes

Radical Candor

Search Inside Yourself

Now Discover Your Strengths

The Personal MBA