The Common Notion of Technical Debt—Measuring and Reconciling Drag

Thanks to Ward Cunningham's metaphor of 'technical debt,' we now have a phenomenal descriptor for the resistance that software development teams face when reconciling their existing architecture after it falls out of alignment with the problem we're solving.[1]

However, as a metaphor, it's just that. It explains and helps us visualize this resistance but doesn't provide us with a way to measure or account for when we 'owe' too much.

Suppose we could somehow measure how much work was necessary and remains to bring yesterday's architecture in line with today's understanding of the problem. In that case, we could quantifiably measure our technical debt for any given project's sprint.

So, let's do what Agile teams do, and find a way.

Introducing: *Drag.*

What is 'Technical Debt'?

If you're unfamiliar, the term 'technical debt' stems from Agile software development. It was first coined by Ward Cunningham when he needed a way to frame why a non-technical stakeholder on a project should care about the hidden architectural attributes of a software system.

In order to quickly deliver the finished software product, his team had to begin with an incomplete understanding of the problem they were solving. Along the way, they would readjust both their understanding of the problem and the architecture of the solution they devised. However, if those became too misaligned, they'd have to revise the latter (i.e., by 'refactoring'), or the solution would become increasingly difficult to build.

Teams can somewhat prolong this debt situation, but the more you put off refactoring your architecture, the further misaligned everything becomes. This eventually makes it hard to add new features or adjust to new understandings of the problem because your architecture is too different from what you're trying to solve.

A given architecture's inherent resistance to change is what Ward means by 'debt' in his technical debt metaphor—and it's a fundamental characteristic of all Agile methods.

(It's a feature, not a bug.)

---

[1] Ward Cunningham. *Debt Metaphor.* https://www.youtube.com/watch?v=pqeJFYwnkjE

‘Recipe Debt’?

Somewhat similar to Cunningham’s metaphor—suppose you order a cake from a bakery, one that you found on their menu that *looks* perfect. But you want it to taste like cheesecake instead of devil’s food.

(Maybe you’re just not into devil’s food?)

The baker needs to go back to square one to bake that cake. Maybe they try to use the devil’s food recipe anyway, but the more they reference it, the harder it will become to pull a cheesecake-in-disguise out of the oven.

Common Misunderstandings About Technical Debt

Unfortunately—and ironically—the way technical debt is sometimes used doesn’t always align with its true definition. Too often, people frame technical debt as resulting from poorly written code or too hastily constructed architecture.

It’s not either of those at all.

To start, it should be noted that software development in the 70s and 80s was *slow*. Enough so that delivered products ran the risk of becoming irrelevant or not accounting for the initial problem’s changes upon completion.

The Agile methodology’s emphasis on quick deployments addresses this issue. However, to be successful, it *requires* an incomplete understanding of the initial problem and architectural readjustments based on what you learn and the feedback gathered along the way.

Technical Debt is *Not* Poorly Written Code

Technical debt is about architecture and a natural occurrence of Agile methodologies, not poorly written code. If your code has high cyclomatic complexity, poorly named variables, magic numbers, and other basic errors, then it’s poorly written.

It’s simply bad code construction, not debt.

And having bad code is always avoidable because a skilled developer knows the rules of good code construction and puts them into practice on every project and sprint.

Technical Debt is *Not* Bad Architecture

The situation that technical debt describes is very specific: the problem and the solution's architecture are misaligned, and the development team will face increased resistance if they continue attempting to force them to work in harmony.

Your architecture might be flawless, and so too your current understanding of the problem. But there's a disconnect between them due to your attempts at solving previous understandings.

Perfectly designed architecture can be leaden down with debt if it doesn't align with the problem it's solving.

Technical Debt is *Not* a Static Property

If you owe $200, that amount likely has nothing to do with how you intend to use your money in the future. In contrast, technical debt *only* makes sense in the context of continually changing systems.

If your understanding of the problem remains fixed, then it's not possible for technical debt to exist, let alone accrue. So we can't look at a software system and quantify how much technical debt it has without trying to change it.

Trying to Measure Technical Debt and Make it Visible—From Story Points to Drag

As apt as Ward Cunningham's metaphor is, it doesn't help us during planning stages, daily scrum meetings, or when reporting to project stakeholders beyond explaining that the challenge exists, as there's no measurement associated with it.

We know it exists and must be accounted for—but we can't definitively say *how much*.

As a *dynamic* property, technical debt begins accruing when your understanding of the problem changes and you start realigning your architecture. Again, it describes the resistance you face when adapting a software system designed to do X into attempting a substantially different Y.

To measure that resistance, we can turn to a fundamental aspect of one Agile methodology: scrum's story points.

Story Points

As a relative measure, 'story points' estimate the required effort a team must dedicate to completing a 'project backlog'—or an item on the list of features or changes required.

It's a projection of how difficult a specific task is, relative to other development efforts.

Crucially, story points are intentionally relative and not assigned any other valuations to keep teams operating in harmony. They remove potential politics or competitions of who can complete the most story points in the shortest time.

Despite their relativity, story points still apply here to technical debt.

If your architecture closely matches the new problems you want to solve, then you can add features or 'pay' any existing technical debt with fewer story points. If there's a greater mismatch, it takes more. We could use a scalar story point measurement of how much technical debt exists in the system. However, this would require lots of risky estimation upfront, and it is out of line with agile principles of only estimating enough to accomplish the sprint goals.

Instead, the measurement should be vector-based, and I've started calling it 'drag.'

Defining 'Drag'

As a representation of how much your current architecture impedes the team's forward progress, 'Drag' quantifies technical debt.

It's simply the ratio of story points that add value compared to those wasted on previous sprints' architecture building efforts and tracked over time, sprint-by-sprint.

Estimating Drag

To estimate your team's current drag—on a sprint-by-sprint basis—you need to know how to allocate story points to technical debt. It reveals itself through story points in ways that include:

- **Architectural adaptation bugs** – Found in *unplanned* work, these include any unexpected behaviors arising from realigning the current system architecture to the problem you're solving (e.g., object-relational impedance mismatch).

- **Unexpectedly high estimates for "easy" features** – Found in *planned* work if project backlog items that seem simple start consistently being allotted more story points than you'd otherwise expect. Put simply, your current architecture is well-aligned and your team faces minimal technical debt if 'it's easy to do easy things.' More specifically:

    - Can you easily implement features that involve the same type of transformation as another you've already built? For example, if you have a feature that lets you

make a button red, it should be simple and require minimal story points to add a feature that makes it or another button blue.

    - Can you easily represent different entities as belonging to the same category according to stakeholders' and users' minds? For example, if you have software that can represent cats as belonging to the category ‘animals,’ then it should be simple and require minimal story points to do so for dogs as well.

In a tangible sense, you can ‘see’ technical debt accruing in planned work by how shocked your stakeholders are when you provide unexpectedly high estimates for a seemingly easy project sprint (and if you’d normally agree with them).

If these types of features should be completed easily (in theory) and with minimal story points, but they are causing major problems for your team, that’s a clear-cut sign of technical debt starting to weigh your team down

More Ways to Evaluate and Measure Drag

Sprint post-mortems are another opportunity to measure drag and evaluate current levels of technical debt. They’re a perfect time, as you can take an opportunity to reevaluate your architecture before risking that discovery mid-sprint.

Take a look at each feature or issue your team encountered and ask:

- Was this bug due to architectural adaptation?
- Was this story ‘too hard’ to add?

If so—and your architecture was the culprit—mark those story points as contributing to technical debt.

Tracking a project’s drag this way will inform you whether it’s time to address technical debt by refactoring. If you see drag start to go up sprint-by-sprint, then you need to consider refactoring. If it’s holding steady or (better yet) declining, you might be able to wait.

Drag should also be used to inform stakeholders of current project impediments. In this way, you can communicate just how much technical debt there is to manage—which will help them make investment decisions and set expectations about the invisible architectural properties of our system.

Crucial Considerations and Caveats with Drag and Technical Debt

When applying the concept of drag to technical debt, keep in mind that:

- Technical debt only occurs and makes sense regarding changing system contexts; significant roadmap changes negate any previous drag metrics and accrued debt.

- Categorizing issues as technical debt is ultimately still subjective, so you shouldn't treat it as a precise quantity. Instead, it should be thought of in the same way as story points, a subject measure of how 'hard' something is, and used comparatively against previous sprints to see if drag grows or shrinks over time.

- You shouldn't play 'what if' games, try to split story points, or think about how easy the issue would be under a hypothetical architecture to determine how many points were 'wasted.' Allocate the whole backlog item—you're measuring your team's drag *today*—not what it would be under different circumstances.

If you're struggling to measure your team's technical debt and drag, or are seeking how to begin reducing it by reconciling your project's architecture, [CLIENT REDACTED] is ready to help.

Managing Technical Debt and Drag

Our consultancy services combine Agile development expertise and management skills that prioritize putting people first. We can also help you find the right team leaders or members through our all-in-one staffing, recruitment, and staff augmentation solution that matches firms with top developer talent worldwide.

Contact us today to discuss your project or how we can help expand your team.

Sources:

Ward Cunningham. *Debt Metaphor.* https://www.youtube.com/watch?v=pqeJFYwnkjE

Generalizing the Concept of Drag—Organizational Drag

In one of our most recent articles, we examined what 'technical debt' is, the impact it has, and how we can measure it through the concept of 'drag."' However, exclusively focusing on technical debt as a source limits the applicability and utility of determining drag.

More generally, drag should be thought of as a measurement for any active resistance that impedes your productivity. It's an impediment that must be reconciled.

Expanding our understanding of drag is crucial because there are—at a minimum—three types that can plague individuals, teams, or an entire organization.

Let's take a closer look at comparing each of them—and why 'organizational drag' is the worst of the three.

The Three Main Sources of Drag

When first introducing the concept of drag, we equated it to a relative way of measuring the amount of technical debt that an Agile software development team suffers from and must reconcile—both currently and as a ratio determined over time. More precisely, drag from technical debt is measured by comparing the story points from project sprints that add value against those spent on realigning the project's architecture.

However, there are at least three types of drag to contend with. It can come from:

- Technical debt
- 'Cruft' (i.e., bad code construction)
- The organization itself

To recap our previous article on this topic, 'technical debt' comes from Ward Cunningham's metaphoric attempt to explain to non-technical audiences the challenge that occurs when a team's understanding of the problem they're trying to solve changes. It's dynamic and only occurs when that understanding changes.

Specifically, 'Debt' describes the misalignment between that new understanding and the architecture the team has built for the project so far. It comes from utilizing and adapting existing architecture, with the 'payment owed' being increasingly difficult adaptations and feature additions as your development sprints continue.

It's a challenge faced by virtually every Agile developer simply because of how the methodology works. Technical debt should be considered an unavoidable consequence—the trade-off of delivering functioning software on shorter timelines rather than year(s)-long waits.

The danger is less that technical debt exists at all and more when it's allowed to accumulate to paralytic levels.

But what about these other two sources of drag?

Cruft and Drag

Put simply, 'cruft' is bad code construction—the thing many people mistakenly regard as technical debt. Examples of cruft include:

- High cyclomatic complexity
- Poorly named variables
- Magic numbers
- Erroneous outputs

Cruft results in drag not necessarily because it's difficult to adapt or add new features and functionality. Instead, it's because the through- and outputs the team is working with during development are erroneous (or are far more likely to be) before any new work is even conceptualized.

In effect, cruft is no different than trying to code with your keyboard missing keys. No matter your future diligence, the project is simply broken or suffering from major obstacles in some way. Yes, workarounds are possible—even Ernest Vincent Wright managed to pen *Gadsby* without using any 'e's—but the burden is wholly unnecessary, and wrestling with it is, frankly, *dumb*.

The drag here comes from allocating valuable team resources and bandwidth toward identifying and fixing the issues cruft causes or determining those cumbersome workarounds.

And, ultimately, cruft and bad code construction are always preventable with reliable and knowledgeable development professionals who are committed to quality work.

What Does Cruft Look Like (from the User's Perspective)?

Unlike technical debt that virtually disappears upon a project's conclusion, cruft is apparent even after delivery. Users will regularly encounter cruft-based challenges when operating the software, which will harm your reputation and future business.

For future projects and implementations, clients will simply look elsewhere.

As described by Tim Ottinger, an Agile methodology thought leader, the user experience of wrestling with crufty software can be summarized by:

> "If you are really careful, it will always work.
> If you make a mistake, it may fail or just seem not to work.
> It may do wrong things.
>
> Now you CAN do your job using this software. It's hard to bring new people up to speed and they are always making mistakes with it, but a skillful and careful operator can use it successfully most of the time."

As stated above, just because you *can* doesn't mean doing so isn't *dumb*.

Organizational Drag

Organizational drag can come from numerous sources, but it's indicative and a result of problematic company structures, processes, and culture.

Two of the easiest ways to conceptualize organizational drag are the consequences of poor onboarding (e.g., insufficient acclimation and training, not continuously occurring) and a lack of development support personnel—whether in quality or quantity.

When onboarding doesn't set up your new personnel for success, the ramifications begin creeping throughout your team and the broader organization. Flawed training and acclimation periods lead to flawed expectations on the part of both management and employees—whether overly challenging or too vague.

And, together, they create flawed environments.

Your employees will lack both organizational support and personal investment, which will be reflected in their work. New hires will eventually learn the lay of the land and begin mimicking their coworkers' disillusioned and lax attitudes.

Organizational Drag at a Managerial Level

Incompetent managers are a self-explanatory problem, but if you promote from within this environment or continue the trend of poor onboarding for development support personnel (e.g., scrum masters), the same effects will occur at a much more damaging level.

Competency no longer matters because the organization itself is actively stacked against everyone. Even the most dedicated professional—regardless of their role or position in the company's hierarchy—has a breaking point where they realize their effort just isn't worth it.

Have you ever wanted to throw your keyboard after hearing "that's just the way it is around here" one too many times? Experiences like that are tangible examples of toxic organizational drag, directly resulting from environment and culture.

In addition, hiring too few development support personnel will overtax them and further exacerbate the situation. It inadvertently incentivizes corner-cutting practices during every sprint and reduces the essential positive interactions that managers and other support personnel leverage to cultivate a thriving and invested team—degrading your onboarding processes even further.

Is IT the Worst Field for Contending with Organizational Drag?

There's a reason why IT professionals are known for experiencing excessive burnout rates: organizational drag isn't just common, it's closer to omnipresent.

When turnover rates are averaging 13.2%, it's more than just the widely reported talent gap at play.[2] Yes, there aren't enough skilled technologists out there, but too many organizations are grinding their people down and driving them away. But, before they depart, increased cruft and technical debt happen as a natural consequence.

Why is Organizational Drag the Worst Kind?

Both cruft and (excessive) technical debt negatively impact an Agile team, but they can be thought of as fairly self-contained to a given project.

Cruft will generally impact how well the developed software performs its intended function, but the consequences are unlikely to extend beyond a clunky, laggy, and error-ridden deliverable. It

---

[2] LinkedIn. *Industries with the Highest (and Lowest) Turnover Rates.* https://www.linkedin.com/business/talent/blog/talent-strategy/industries-with-the-highest-turnover-rates

may reflect poorly on your team or organization, but with every new project, cruft is effectively reset by the blank slate of unwritten code.

Similarly, an accumulation of technical debt can—in theory—always be reconciled mid-project by throwing more effort (i.e., story points) at it, and, upon delivery, that debt effectively disappears.

Misalignment between your team's understanding of the problem you're trying to solve and your architecture built thus far can always be addressed, with the consequences mostly constrained to longer deliverable timelines and ballooning budgets. Stakeholders or clients will certainly not appreciate being notified of either addition, but the problem can always be remedied.

And, excessive technical debt is a bit of a Catch-22 either way, as allowing it to accumulate and contending with misaligned architecture leads to the same result as remedying it regardless.

While continuous cruft and excessive technical debt may significantly affect a company's reputation and long-term success, there are natural resets that an Agile team can count on and leverage as opportunities to recoup, recover, and prevent their recurrence.

In essence, that's a major reason why teams perform sprint reviews.

And it's a reason why unqualified, too few, or 'too-dragged' development support personnel leading those reviews cause an exponential proliferation of the problem. At best, the causes of cruft and excessive technical debt are never resolved. At worse, they compound.

In contrast to these other two sources, organizational drag continuously persists and substantially impacts every project or hire—creating and exacerbating a higher likelihood of cruft and excessive technical debt.

There's no reset of organizational drag without a considerable and conscientious effort to address it.

How Can We Address Organizational Drag?

The easiest ways to rectify organizational drag return to the examples of onboarding and development support personnel mentioned above. Conscientiously overhauling either will make monumental strides towards reducing organizational drag and its cascading impacts.

For an excellent example, consider how Paragus—an IT services firm in Western Massachusettes—addressed its turnover challenge. No matter how many perks they provided, nothing seemed to change.

The sole thing that worked?

Dealing with the problem from a structural perspective and cultivating an environment that operated as the antithesis of organizational drag:[3]

- Expectations and a supportive culture were firmly established from the initial interviews on—from the perks of their role like employee stock ownership plans (ESOP) to 'career ladders' explicitly outlining performance goals.

- Training and career advancement opportunities were clearly communicated.

- Teams were restructured and documentation was improved to insulate from the effects of turnover—with four-person teams managing each client instead of one individual, eliminating overlap and helping to speed up onboarding when someone new joins.

- They created a 'Wall of Fame' for employees who left on good terms and celebrated their 'graduation' with a party and acknowledgment.

And Paragus' growth definitively demonstrates the resulting impact. Per Delcie D. Bean IV, Paragus' CEO:[4]

> "These efforts have been helping us continue to scale up, despite the challenges of today's labor market. We've grown to 50 employees, and our company has hit $7 million in revenue, up from about 40 employees and $4.5 million in revenue in 2016."

Tackling Organizational Drag with [REDACTED]

We founded [REDACTED] Consulting largely in reaction to the challenges of and those caused by organizational drag. It's rife throughout the IT industry, and we want to actionably address it, creating healthier workplace environments for both the company and individual's benefit.

---

[3] CNBC. *A small tech company tried it all to stop employee turnover. Only one thing worked.* https://www.cnbc.com/2019/12/03/a-tech-firm-tried-it-all-to-stop-turnover-only-one-thing-worked.html

[4] CNBC. *A small tech company tried it all to stop employee turnover. Only one thing worked.* https://www.cnbc.com/2019/12/03/a-tech-firm-tried-it-all-to-stop-turnover-only-one-thing-worked.html

When organizational drag is minimized, *everyone wins*.

Between our [REDACTED] services, we can debug the issues from both sides—helping organizations both reinvent their onboarding processes and work environments as well as find and hire top talent who will appreciate the effort and reward them in kind.

If any of the three main sources of drag are affecting your organization and teams—especially organizational drag—reach out and talk to us about a solution.

Sources:

LinkedIn. *Industries with the Highest (and Lowest) Turnover Rates.* https://www.linkedin.com/business/talent/blog/talent-strategy/industries-with-the-highest-turnover-rates

CNBC. *A small tech company tried it all to stop employee turnover. Only one thing worked.* https://www.cnbc.com/2019/12/03/a-tech-firm-tried-it-all-to-stop-turnover-only-one-thing-worked.html

Preventing and Fixing Organizational Drag

This article is the third in a series on "drag." For more background information, please read Part 1 (The Common Notion of Technical Debt) and Part 2 (Generalizing the Concept of Drag).

In our previous examinations of "drag," we've asserted that organizational drag is the worst of the three kinds as it indicates problematic structures, processes, and culture.

What is organizational drag? It's caused by poor onboarding and a lack of sufficient development support personnel, becoming a systemic challenge that gradually spreads. This causes technical debt and cruft (the other two types of drag) with increasing frequency as drag compounds and hinders your possible achievements.

So, with organizational drag capable of debilitating businesses and teams to such an extent, how do you prevent or reverse it?

Complex challenges sometimes have solutions that are simple on the face but equally complex in their application, and such is the case with organizational drag. Its antidote is a multifaceted and continuous effort spread across strategic planning, organization building, and team management.

Ultimately, the solution to organizational drag can be described as intentive and continuous onboarding, which includes the transformations necessary to implement it.

Step 1—Strategic Planning

Most successful endeavors begin with a plan that keeps them focused and on-target, chasing what is commonly referred to as the 'North Star Metric.' Without a plan, you can't effectively implement an organization-wide strategy; without an organization-wide strategy, you risk uncoordinated decision-making that creates conflicting management and individual efforts.

This fosters the disharmony that allows organizational drag to fester.

At the company-wide level, there are numerous ways to approach strategic planning, but they commonly come down to either business goals (e.g., revenue levels, market share) or solving your audience's specific challenges. At the individual level, every employee must make decisions and take action with the strategic plan in mind—so it also needs to resonate with them for maximum staying power.

Develop a Five-Year Technology Strategy

Developing and documenting a five-year plan is the first step toward adopting a holistic strategy that guides your organization and prevents an accumulation of drag.

Adhering to a five-year timeline is not exactly a hard rule to be followed. Still, your plan must similarly balance being short enough that the goals feel tangible and attainable yet forward-looking enough to account for evolutions and pivots following the challenges you'll encounter en route.

For technology organizations, this plan needs to comprise product development, delivery, and support alongside the internal technology resources and infrastructure those ongoing efforts require.

Consistency Between Your Strategy and Mission Statement

You can think of your five-year plan as a roadmap, with the direction of travel influenced by your mission statement.

Although your mission statement likely comes before your plan, for simplicity's sake, the former should act as a summation of the latter. Alternatively, if your strategic plan hones in on achieving a specific subset of your 'North Star Metric' or mission statement, it should do so in a complementary fashion.

Unfortunately, the technology industry's oft-stated mission of 'making the world a better place' is—at best—only 50% defined. It's too nebulous for your personnel to understand and apply it with any consistency, particularly when every individual will have their own views regarding what constitutes 'a better place.'

And vague mission statements like the above often lead to:[5]

- Employee dissatisfaction, lacking commitment, and cynicism—which go hand in hand with organizational drag

- Poor production and financial performance (as a result)

[5] Sage Journals. *Left in the Dust: Employee Constructions of Mission and Vision Ownership.* https://journals.sagepub.com/doi/10.1177/2329488415604457

Two studies examining these types of mission statements found that they're likely to foment antipathy, with one going so far as to deem them 'rhetorical pyrotechnics—pretty to look at perhaps, but of little structural consequence.'[6]

Instead, you and every member of your organization must be able to articulate *how* and *why* you'll achieve your mission if you were to ask them individually. That's what your strategic plan is, and it's how you give your mission statement true impact.

Be Open to Strategic Changes and Revisions that Still Pursue Your North Star

Over five years, you'll most likely encounter challenges or discoveries that require revising your strategic plan. Embrace the revision process; consider how you can reformulate your mission and plan to remain aligned with your 'North Star Metric' despite adapting to new circumstances.

There a numerous reasons you might need to do so:

- Changing technology landscapes
- Audience shifts
- A better understanding of the problem you're solving

Organizations that refuse to change quickly find themselves contending with drag. Similar to what we described in our technical debt article, it's attempting to solve a challenge with suboptimal architecture. Only, this time, it applies to your organization rather than software development.

Articulate Your Strategy to New Hires

Everyone likes to talk about buy-in, but have you taken the time to articulate your strategic plan to employees? Especially during the interview and hiring stages?

People care about what they'll be building and why. They need to understand what they're buying into.

Early interactions are the best opportunity to communicate your strategic plan in a manner that leaves a resonant first impression. Doing so defines how everyone regards your organization moving forward. Articulating your mission and strategic plan to achieve it develops a shared

---

[6] Taylor and Francis Group. *Mission Statements: A Thematic Analysis of Rhetoric across Institutional Type.* https://www.tandfonline.com/doi/abs/10.1080/00221546.2006.11778934?journalCode=uhej20

understanding of them within concrete terms, which employees will recurringly refer back to throughout their day-to-day and long after their hiring.

You can't expect to accomplish organization-wide strategic consistency without communicating it clearly.

Step 2—Organization Building

Organization building is where you start putting this strategic plan into practice. As mentioned above, you'll need to account for product development, delivery, and support. But most importantly, you'll need to prioritize your people as they're the ones who comprise the organization.

Frictionless On and Offboarding

To reiterate, organizational drag results from poor onboarding and insufficient development support personnel. Therefore, overhauling your onboarding process allows you to more quickly bring in and nurture top talent that will be directly assigned to or oversee projects.

Reducing barriers to organizational transformation starts with developing on and offboarding processes that allow you to easily build teams assembled for success. The easier it is to bring in people who will help transform your organization, the more and faster that transformation will occur.

And the imperative word there is 'teams.' Your new hires need to feel like they belong and are joining an environment that will respect their capabilities while providing opportunities for further improvement. But to accomplish this for numerous employees, you'll need to develop a comprehensive program to ensure every crucial onboarding step takes place.

The Art of Onboarding

The art of onboarding involves specific stages that must occur alongside the clear communication of your company culture, mission, expectations, and performance metrics. They're fundamental to the construction of any team, and although they were first developed in 1965, they remain just as pertinent today:[7]

- **Forming** – Introductions and first impressions significantly affect team dynamics—for both new and existing members—which requires some give and take on all sides. New members need to understand the team's dynamics, and existing members need to be

[7] Bruce Tuckman. *Development Sequence in Small Groups*. https://doi.org/10.1037/h0022100

helpful and accommodating. There must be room to evolve following the addition without losing sight of goals or what's worked in the past.

- **Storming** – Even if the 'forming' stage goes off without a hitch, there will eventually come a time when inner conflicts arise, and the team must work through them. This stage will put the success of "forming" to the test, but teams that work through these challenges in a healthy manner will be primed for success when they emerge on the other side.

- **Norming** – Following the 'storming' phase, team dynamics should be more established and cohesive. Members know their roles and how to interact with each other more positively and beneficially, continuing to move forward in pursuit of harmonious collective and individual goals.

- **Performing** – After enough time has passed in the 'norming' phase, team members will have developed enough trust in each other to enable full flexibility in how every person performs. Of course, accountability still exists with a 'performing' team, but everyone can confidently pursue their assignments while knowing they can depend on others to do the same.

Synergy Across Your Org. Chart, Product, and Strategy

The holistic approach to organization building requires an alignment across your organizational chart, product, and strategy. In simpler terms:

- The product you're building and the challenges it'll solve need to reflect your strategic plan and mission statement.

- To develop this product, you'll need the right people. This means they:
    - Have the appropriate expertise for their role
    - Are invested in your mission and strategic plan for achieving it
    - Are properly supported

These three elements of organization building—your strategy, product, and organization chart—need to be synergized. If one falls out of alignment, so too will the others—resulting in a poor product, stagnant organization, and disillusioned employees. The most dangerous is having the wrong personnel and a fractured application of your organization chart, because the right people will transform an organization and get it back on track.

Step 3—Ongoing Team Management

Effective, ongoing team management is synonymous with continuous onboarding. Truly, onboarding never ends, as there are always role adaptations, new projects, and—ideally—internal promotions.

One of the most significant mistakes too many organizations make is assuming onboarding is restricted to the first few days, weeks, or months of a new hire's employment.

This is where you'll need to cultivate and then rely on quality development support personnel to apply your strategic plan and mission statement through their team leadership efforts.

Growth Plan for Every Person

No matter how impactful your initial onboarding process is and how much your mission and strategic plan resonate with each employee, they will gradually start to lose their motivational capabilities if not reinforced over time.

The most effective way to strengthen them is simply by supporting your employees in their professional development and paths toward growth.

You'll need to adopt and adhere to personal and professional development structures like:

- Regular check-ins (e.g., monthly, quarterly)
- Designated resources and time allocations for training (conducted internally or externally)
- Clearly communicated opportunities for internal advancement

Again, these are conscientious efforts, and they require your leadership's initiative to create and enact development plans. Just because you're willing to offer this support doesn't mean your employees know that or feel comfortable approaching their managers to ask about it.

Furthermore, managers need defined roles and boundaries that preserve enough bandwidth for them to carry out these initiatives. If production remains too demanding of their time, odds are they'll place a lesser priority on employee development. That might help a product reach its estimated launch timeline more easily, but it damages your organization's timeline with a vengeance.

Treat High Turnover as a Crisis

High turnover is one of the most dangerous circumstances a tech company can face and is the leading predictor of organizational drag. If your employees frequently leave for new opportunities, competitive offers aren't the only motivation hastening their departure.

Perhaps they're burnt out, see no development paths forward for themselves, or are simply disillusioned with your organization's direction—all of which will also impact any new people you try to onboard.

Regardless of the talent you might be able to add, your organization will still contend with a loss of knowledge that impacts the quality of your onboarding, products, and service delivery. New hires will be trained and supported by employees less experienced with your products, processes, and culture. This will create a vicious cycle that becomes more and more difficult to escape.

If you find yourself facing high turnover, there are deeper structural problems your organization needs to confront.

Transform Your Organization with [REDACTED]

Although our solution to organizational drag is presented here in steps, none of them ever truly finish. Instead, they're merely organized based on the initial foundation that subsequent efforts require.

But to fix a challenge as complex as organizational drag, it has to start somewhere.

And that's where [REDACTED]'s service makes its earliest impact. Our experts will partner with your organization to first determine and begin implementing a strategic plan that will impact the organization-building and team-management efforts that follow. And, because these latter stages are so reliant on the people in each role, that's where we prioritize our efforts.

Because people comprise organizations, and people build and use technology. Therefore, organizations and technology should both revolve around people.

To learn more about remedying organizational drag, reach out to us today.

Sources:

Sage Journals. *Left in the Dust: Employee Constructions of Mission and Vision Ownership.* https://journals.sagepub.com/doi/10.1177/2329488415604457

Taylor and Francis Group. *Mission Statements: A Thematic Analysis of Rhetoric across Institutional Type.* https://www.tandfonline.com/doi/abs/10.1080/00221546.2006.11778934?journalCode=uhej20

Bruce Tuckman. *Development Sequence in Small Groups*. https://doi.org/10.1037/h0022100