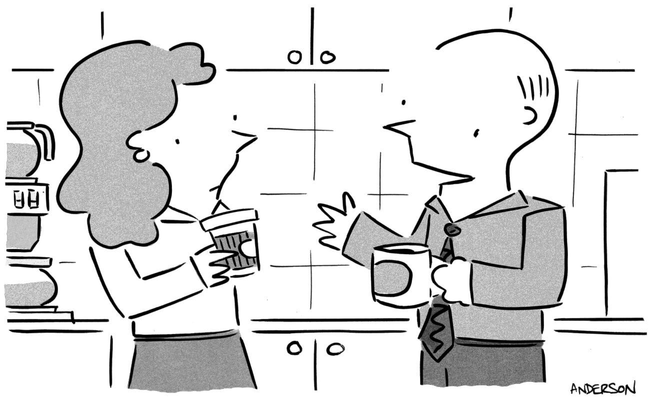
JIT selection from independent streams: An alternative to the “big backlog” of work

by Jason Cohen on August 23, 2022

The vaunted “single-threaded, ordered list” confuses “prioritization” with “work-planning,” and forces comparisons of the un-comparable. Here’s the solution.



“Things have gotten a lot easier since I moved everything from my to-do list to my it-is-what-it-is list.”

Scrum teaches us that the “single-threaded, ordered list of work” is the correct way to prioritize the work of the future. Innumerable articles—including one from this very site—tell us that a single, stack-ranked list is the simplest way to clarify priority.

I find, however, that this is unsatisfying at best, and fails to achieve sensible prioritization at worst. It also doesn’t solve the political challenges of prioritization with stakeholders.

The following system is simple, and has worked for me.

Prioritization ≠ Work Planning

Prioritization means: *What is most important?*

Critically, it does **not** mean: *What will we work on next?*

Product Managers typically conflate these, trying to do both with a single process, and a single backlog. A traditional backlog efficiently encodes the answer to the second question, but isn’t the right tool for answering the first question.

You might say, “But we should work on whatever is most important, right?” By unpacking the subtle answer, we’ll arrive at a better way to prioritize, and a better way to plan work.

Why “the big backlog of work” isn’t good for prioritization

Prioritization

If prioritization answers “What is most important,” it begs another question: What do we mean by “important?”

- Important to calm our largest customer who is threatening to leave
- Important to catch up to our biggest competitor on a specific feature
- Important to unleash the sales and marketing teams onto a new market segment
- Important to satisfy our security policy
- Important to decrease the largest cause of support tickets
- Important to mitigate the largest cause of cancellations
- Important to refactor code so that engineers are happier and more productive
- Important to increase profitability
- Important to achieve our long-term strategy, even if there’s no next-quarter revenue

Sorting a “Big List” of epics doesn’t seem like the right way to tackle this complexity.

The first problem is that you’re sorting things that are incomparable. Every bullet point above names something undeniably “important” but also belies a different motivation, with different impact, measured differently, so on what basis, with what metric, would you order The Big List?

The second problem is that the resulting decision is unsatisfying to stakeholders. Sales wants a few specific competitive features, and doesn’t have the context to understand how those are interleaved with requests from support or why the engineers need an entire month to “clean up a mess, which by the way, they made themselves?” The Big List doesn’t yield satisfactory explanations.

Work-planning

The third problem is that The Big List conflates the idea of “prioritization”—what is most important—with “work-planning”—which is the order that specific work will be executed. While there’s a correlation of course, often work-planning does not match our ideal prioritization because…

1. Hard deadlines—often externally-imposed—requiring work even if something else is nominally “more important.” (e.g. zero-day security patch is not on our list of prioritized strategic work, but must be scheduled immediately. Or: Have to get specific functionality shipped for a live customer event).
2. We might not currently have capacity on the teams who are capable of working on a high-priority thing.
3. We might not currently have the skillsets to execute a high-priority thing.
4. Dependencies—i.e. A is very important; B is not. But A cannot be done unless B is done. Therefore, we must work on B first, even though A is more important.
5. Unknowns that have to be investigated before work can commence, or even be planned. (e.g. spikes or proofs-of-concept)
6. The amount of work, or amount of risk, of executing a high-priority thing might mean we shouldn’t tackle it yet.
7. Getting success on some smaller things, while working on longer things, is better for throughput, better for morale, and is more agile.
8. Blockers—dependencies on other work, maybe even other teams, before we can make progress on a high-priority thing.
9. Starving—sometimes we’ve put off low-priority things for so long, they effectively become higher-priority.
10. Holidays and other seasonal things (e.g. kicking off a huge new initiative on December 15th is a bad idea, even if it’s strategic).

The solution is to prioritize items in separate, actually-comparable streams, and to keep that prioritization effort cleanly separate from work-planning.

Separate prioritization streams

“*Don’t cross the streams!*"
“Why?”
“It… would be bad.”
—*Egon Spengler, Ghostbusters*

The stakeholders

Folks in Sales may not know which items are most important for Support, but they sure as heck have an opinion on what would be best for Sales! It’s fun to sit down with sales reps and sales leaders and ask them to generate and then stack-rank[1] the things they believe would be most impactful. Tell them they don’t have to worry about other departments or other requirements—this time is just for them, and this list is just for them.

[1] Wait, that means a single, prioritized list! Yes. Because this is a single group, and we’re asking them a single question from the bullet-list above (e.g. “what would be best for increasing sales, whether because of increased win-rate or increased average transaction size”), we don’t have the problem of comparing unlike things, and so the “single list” is a good forcing-function to generate a clear prioritization.

The ideas will pour out. Some ideas come from the last sale someone lost; that might be insightful or might be noise[2]. Some ideas are critical causes of lost sales; bonus points if you can tie them to specific lost-sales; that’s motivating for Engineering as much as for Product. Some ideas are just a feeling—“I *know* I could sell the piss out of X.” Don’t dismiss those out of hand in your quest to be data-driven; multi-billion-dollar startups have been built atop experience-informed gut-instinct.

[2] One way to separate the short-term emotional reaction from the important trend is to observe what happens in subsequent prioritization sessions. When you see the group dramatically shifting the priority of items quarter-to-quarter, that indicates the group doesn’t have a consistent sense of what would be most impactful. Surely some of the ideas are truly great; *this* team just isn’t sure which ones those are. Sounds familiar—Product Managers struggle with this challenge all the time!

By forcing that group to stack-rank their ideas, they’ll argue amongst themselves over which are most impactful. This creates empathy for what product managers have to do all the time—struggle with prioritization in the face of multiple goals and imperfect data. It also means they can’t make everything “Priority #1,” which in turn sets up a healthier future discussion: When, in two months, they complain that “you still haven’t made feature X,” you can point out that “X” was number seven on their list, so they themselves felt that it wasn’t as important; you were just listening to their input. (Of course, everyone is allowed to change their mind; it’s useful to run this process multiple times per year.)

You’re not promising to do everything they want, in the order they want. That’s work-planning, and also this is just one list; you’re going to do this with other groups too. And this list doesn’t include the most important voice of all—the voice of the customer. You *are* promising that you will take this list seriously, because you genuinely value their input. That’s more than most Sales teams get out of Product Managers.

Now repeat for other groups.

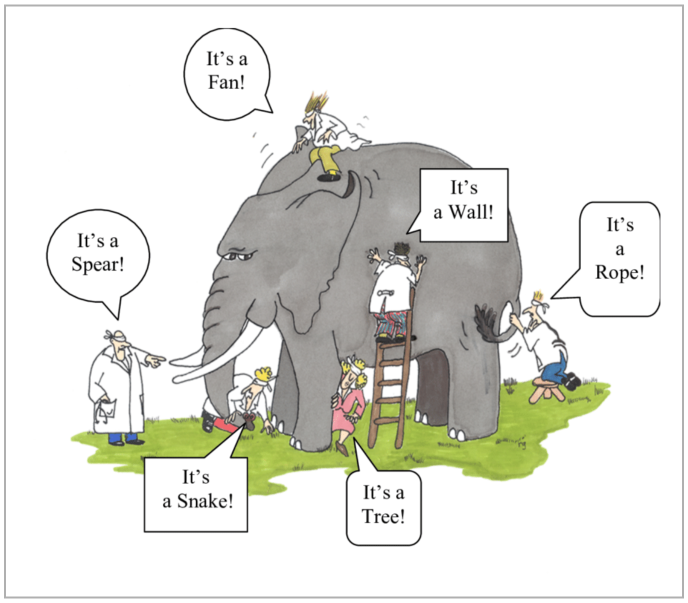
Sales Engineering has a view unique from Sales, more technical and more in tune with technical customers, often the users rather than the buyers.

Support often knows the product better than anyone, and with the right tools and processes, has better data about which topics are causing the most problems or confusion. Remember that this isn’t only about reducing work and costs for the Support team—although that’s already a great result. It’s about creating a better customer experience, because the customer who never needs to contact Support to get their work done, is a happier, more successful, more loyal customer[3].

[3] There are fascinating data supporting the claim that “needing Support less” is correlated with “customer loyalty.“

Engineering also needs their own list. There’s always things to refactor, libraries to replace, spikes for new technology to try, replacement of a continuous-integration system, test automation we neglected before but desperately need now, systems that were adequate last year, but now we’ve scaled out of, and so on. Their stuff is about operational excellence and a happy, fewer-bugs, higher-productivity environment, not about growth or the customer. Therefore, it’s easy for these things to never be prioritized, which in the long run creates resentment and grinds development velocity to a halt. But also, engineers—like everyone else—have more ideas than time, so they should be forced to stack-rank.

So… which of these lists are correct? There is truth in all, but none contain the complete picture, like the 2,500-year-old Buddhist allegory about the The Blind Men and The Elephant:



Source: J. Himmelfarb (artist G. Renee Guzlas)

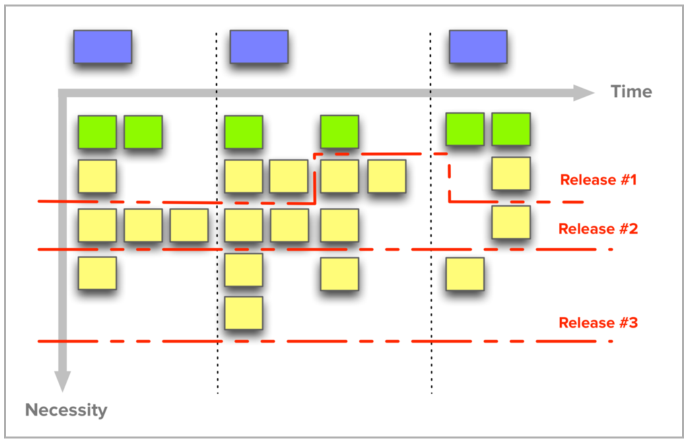
It’s the job of a Product Manager to consider all of these inputs and deduce the elephant. Keeping the inputs separate and stack-ranked helps the PM with this daunting endeavor.

The customer

These are internal groups; what about the customer, the star of the show! This is typically the purview of Product Managers and Designers and UX Researchers, as opposed to internal stakeholders. For simple products, perhaps there’s only one prioritized list representing our best idea of what’s important to the customer, but products with non-overlapping areas of functionality or that serve multiple personas might have one list per persona or functional space. So for example you might have separate lists for the needs of small versus large companies, or first-time users versus power-users.

Another common technique is to have a list per step in the customer journey, reflecting the fact that it’s important for trial users to find success quickly, and also important for paying customers to continuously see value, and also important that the highest-revenue power-users to stay and grow. As with stakeholders, the motivation for the prioritizing items within each category are not comparable.

There is plenty of prior art on this topic. This is essentially what is happening with Story Mapping or Opportunity Solution Trees, where the customer-journey steps or the main opportunities each contain a list of potential work-items. Indeed, you could argue that story-mapping is just exactly the idea expounded here, extending the concept of “independent lists” from only customer-focused topics, to topics from Sales, Support, and Engineering.



Alternate types of list

If you don’t like separating lists by function or by customer journey, here are other systems that some people like to use:

Kano

You could also create lists using the Kano Model. Imagine one list each for Delight, Performance, and Must-Be. In general you don’t want to starve any one of these categories, but it’s difficult to prioritize items between categories. For example, it’s easy to claim that everything in “Must-Be” has to be implemented first; after all it’s called “Must-Be” for a reason! There’s no point in bringing on new customers who won’t be successful. That sounds reasonable, but it’s simply not true that products should supply only basic requirements; indeed it’s often the (unexpectedly) Delightful items that cause a customer to stay *despite* failings. Having separate lists, none of which should be completely starved, helps to combat this problem.

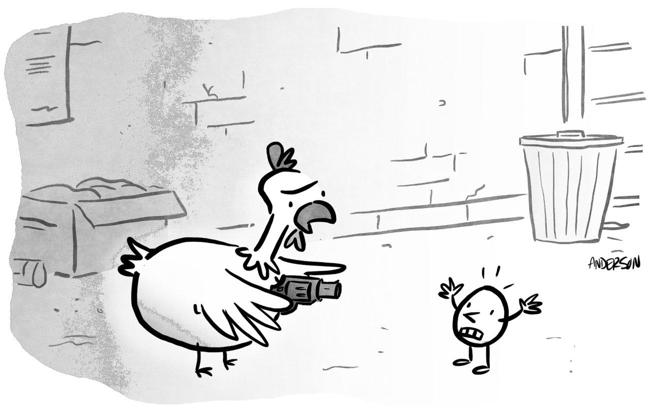
The “Four Lists”
From Jyoti Bansal, summarized by Adam Thornhill:

1. **Customer engagement.** Tuning into user feedback and desires.
2. **Sales intelligence.** Learning from the field to stay abreast of the competition.
3. **Technical debt.** The debt engineers must address to prevent performance issues.
4. **Vision for the future.** Where the product should head, expanding the addressable market.

In all cases, all of the lists are important, in the sense that you cannot ignore any one of them forever. But you don’t have capacity to move all the needles at once, and the items within each list are different-in-kind, and should be prioritized separately.

Crossing the streams for work-planning

Eventually, human beings need to do work. It makes sense to select work from these various streams and—yes—place the work in an ordered backlog. When you’re actually doing work, you want a clear answer to “what’s next?” without additional meetings and debate.



“Maybe we’ll never know who was first.
But I can tell you who’s gonna be *last!*”

By waiting until the last second to compare incomparable things, prioritization remains organized and clear. By waiting until the last second to shift your mindset from “most important” to “planning work,” you maximize clarity in the process of selecting and scheduling work.

In software, when you wait until the last possible moment to do something, it’s called JIT (“Just In Time”), hence the titular name of this technique.

You still have the unenviable job of selecting which items from which streams are going to happen next (or “soon,” in the case of a roadmap). But that was always going to be the case—there’s always more work than time, so there’s always a selection process. This technique keeps that process more organized than it would have been, and it allows you to grapple with work-planning puzzles separately from prioritization, which at least compartmentalizes the challenge. Divide-and-conquer is a good way to overcome complexity.

> *“We’re obviously in no danger of arriving at consensus.”*
> *—Warren Buffet*

You’re selecting from the top ideas at the company, not from the 1000 ideas everyone collectively has. It’s a lot easier to pick the best from 12 instead of from 1000, and in some sense you can’t go wrong—on average all 12 are probably decent ideas. And stakeholders will be happy.

This last point is under-appreciated. This process is more explainable after-the-fact. Suppose it turns out Sales’s first item is impossible to tackle right now, but you can knock out their second item because, by the way, it’s also Support’s number four and fits nicely into a multi-month customer journey improvement project you have. That’s a great message to send back. Or, a message like: We knocked out two of your top items recently; we’ve been starving the engineering list so we’re going to hit a few of those in the next few months, then resurface.

This process also helps keep yourself honest. You can easily report on:

- How much of each kind of work are we doing?
- What is the age of the top three items on each list? (i.e. Are we starving something?)
- How much of this sprint is addressing someone’s “top 3” thing?
- How much work is related to internal projects versus customer-facing?

It also leads to better ideas:

- Are there trends that everyone is seeing across all functions in the business, that therefore should inform our strategy?
- Can we think of new product ideas that hit multiple top-three items simultaneously, and thus are especially impactful?

Don’t forget to schedule rocks, then pebbles, then sand. That’s an even more primary principle for work-planning. Separate prioritization streams help identify which rocks and pebbles should be scheduled in the first place.

“*When we differ, Charlie usually ends the conversation by saying: “Warren, think it over and you’ll agree with me because you’re smart and I’m right.”*

—Warren Buffet on Charlie Munger