Rocks, Pebbles, Sand: How to implement in practice

by Jason Cohen on July 17, 2022

This complete work-prioritization framework builds on the simplistic "Rocks, Pebbles, Sand" analogy, adding the details you need in the real world.

You know the geology-in-a-jar lesson from Stephen Covey: Schedule big things first, otherwise you run out of time:

If you do little things first, there's no time for big things.

If do big things first, then you can fill in the smaller things.

A common mistake is to think this applies only to the *size* of the work. That is, "Rocks" means "stuff that takes a few quarters," "Pebbles" means "a few sprints," and "Sand" means "less than a sprint."

This misses the most important point of work-ordering: It's about maximizing *impact* by not allowing the easy or urgent things to crowd out the strategic things that take years to unfold but are more important than everything else combined. A thousand "quick wins" do not create durable advantages or fulfill a long-term vision.

Another mistake is to think that the previous paragraph is the end of the story. "Schedule revenue-growth stuff, then maintenance updates, got it." No. Each type of work requires different prioritization frameworks, has different goals, and hide different traps that make you unwittingly ineffective.

If you pretend these differences don't exist, your team will be working hard and delivering lots of code-commits —the appearance of "productivity"—but they'll feel like they're not making progress fast enough, competition will start catching up, and they'll (correctly) complain that they can't see how their work is connected to the strategy.

The good news is: It does not take additional time to do it right. This is an instance of "smarter, not harder." You just need the right frameworks.

Three mindsets

A tabular summary is trite, but it's a handy reference:

	Rocks	**Pebbles**	**Sand**
Effort	≥3 Months	1-4 Sprints	≤1 Sprint
Maximize	Impact	ROI	Throughput
Outlook	Long-term	Short-term	Immediate
Scope	Strategic	Tactical	*any*
How Decide	Deliberate	Analytical	Intuitive
Role of Exec	Decider	Observer	*none*
Role of PM	Driver	Decider	Decider *(but engage devs)*
Beware	insufficient impact	over-estimat-ing ROI	over-thinking / over-planning

Now we'll justify and explain how to use this to maximum effect, each stone in turn.

Rocks maximize Impact

"No, I'm working from home today."

Duration: 3-12 months

Rocks take the most time; let's call it 3-6 months. Long projects are not only expensive, they're also most likely to over-run, because they're the most complex, contain the most unknowns, and have the most dependencies. So it's really 3-12 months.

> **Hofstadter's Law**
> It always takes longer than you expect, even when you take Hofstadter's Law into account.

Well-oiled agile teams will point out they can hit earlier deadlines by adjusting scope and pushing less-urgent items past the deadline. That's wonderful for learning and customer-delight. But the leftover work still needs to be done, even if rearranged, so this doesn't change the magnitude of the effort required to achieve the full effect of the idea. And anyway, there are good reasons why we're consistently incapable of estimating big projects.

A team might only do a single Rock in a year—certainly no more than three—especially because other work also needs to get done. If you can do only one big thing this year, that thing had better be extraordinary.

Maximize impact

A Rock must deliver dramatic, measurable impact, not merely "incremental improvement." It must be *strategic*, meaning that it must attack the most important challenges you face, materially advancing the company down its unique path for winning its corner of the market, leveraging existing advantages to reduce risk and to forge a path that others cannot easily follow, and build new durable advantages. This is where teams most often fall short: Not delivering enough impact to justify their investment of time.

Sadly, big projects not only over-run on time, but also often under-deliver on impact[1]. These sorts of predictions are famously inaccurate. So it's even more imperative that we demand an enormous impact: That way, if we under-achieve, it was still worth the time.

[1] Fortunately, on occasion they can over-deliver by an order of magnitude or two; this is always the *post hoc* story of a successful company, the founders shaking their heads saying they never believed it would be *this* successful.

Crucially, and perhaps controversially: **Do *not* maximize ROI**[2]. Your primary job is to execute your strategy to the fullest, spending the most-possible time on the most-impactful thing. If an idea is less impactful, yet also quicker to achieve, that is *not* the right choice. When things go worse than planned, that "less impact" turns into "incremental impact," and you cannot spend half a year on something so trivial.

[2] ROI is "Return on Investment," computed as a measure of impact divided by a measure of effort, resulting in a measure of efficiency, i.e. "value per sprint." (Whatever "value" means.)

Beware: "Maximizing impact" is harder than you think

The most common problem is executing Rocks that aren't impactful enough. The Rock claims to "make a difference," but not *enough* difference, and after a few years, it feels like "we're not moving fast enough" or "why isn't revenue higher" even though the work from engineering is high-quality and stories are duly delivered every sprint.

Hofstadter's Law applies not only to the time-estimate, but to the impact-estimate, and thus to the height of the bar that need to set: It must be higher than you think, even when you take Hofstadter's Law into account. Overshooting is the antidote to Hofstadter's Law.

Even if your ideas aren't good enough, you will be tempted to select the most impactful idea on the list and just do it. This is a mistake. It's such a common mistake, it is a cliché: "Good" is the enemy of "Great." You have to face the truth: Your biggest problem is a lack of a truly great idea, and you must solve *that* rather than embarking on a long, misguided journey. The team can do Pebbles and Sand in the meantime, thus staying productive, while also helping ideate and validate better ideas.

Deliberative decision-making process

Because you're committing so much of the team's life[3], and because you have to be so confident that the Rock is strategic and impactful, you need to spend time on this decision up-front. This is not an "agile" decision, it's a strategic one. Once the direction is set, the big picture is clear, the mountain you want to climb is identified, *then* it is ideal to be "agile" in how you climb it. Execution details are never certain; backtracking is necessary. But if you're climbing up the wrong mountain in the first place, being "agile" doesn't help; the result is a team self-managing themselves into a mediocre, unfulfilling result. Although most decisions should be fast, sometimes they should be slow; Rocks should be slow, or at least deliberate.

[3] Agilists argue that if you find yourself part-way through a failing project, you can just abort, because "that's agile." That's true, and that's smarter than plodding forward in a sunk-cost refusal to face reality, but on a human level it is demoralizing and ruins trust to abruptly cancel a project a team has been laboring on for months, "because we're agile." Canceling is necessary, but not free, so you should act *as if* a Rock is a one-way door.

Use this framework to select the most impactful idea: Binstack: Making a maximal multi-dimensional decision. This process enshrines "impact" as the highest priority, allows other dimensions to participate but neither confuse nor dominate the decision, and produces a pithy, clear explanation of the decision at the end.

If you're not coming up with good-enough ideas in the first place, try these prompts. But also consider whether the real problem is that your strategy is too vague.

Execs decide, but ideally PMs are in command

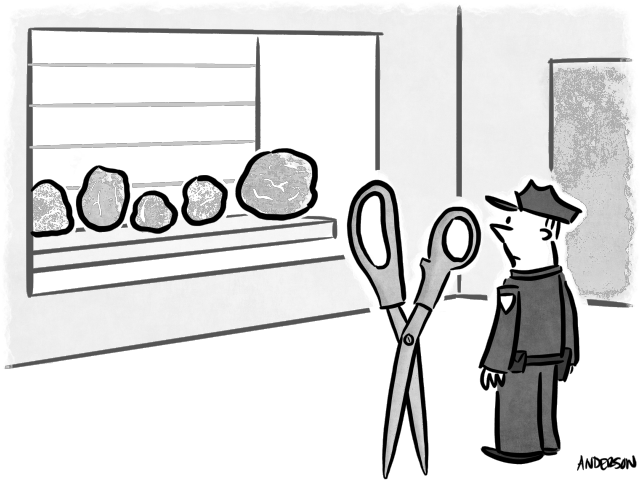
Rocks materially advance the strategy, and executives[4] ultimately own the strategy. So, ultimately the final decision of what Rock to execute rests with the executive. In practice, however, the PM should be in command of the strategy and the ideas, driving the discussion and the decision. Ideally[5] the PM is actually in command, and the executive is happy to play the role of coach and Devil's Advocate during the decision process, and to sign off on a well-thought-out proposal.

[4] At a smaller company, this means whoever has the executive role, whoever is running all of Product, or R&D, or the CTO, or the CEO.

[5] We won't cover workplace dynamics in this article, so suffice to say that this is what "healthy" looks like, and the further reality is from ideal, the more one or both parties needs to change.

Sand maximizes Throughput

Because Pebbles are the Goldilocks of work-items, it's instructive to leap over them and solve for the smallest items, establishing bookends around the middle-child.



Duration: ≤1 sprint

Sand are items that don't need to be broken down. They're short, and typically "just need to be done" without discussion or ado. They can't take longer than a sprint; sometimes they take less than an hour.

There are a million little things that are individually unmeasurable but that add up to a significant impact. Great user interfaces require a hundred tweaks to attain greatness. High-quality error-handling requires esoteric corner-case unit-tests. High performance is often the result of innumerable optimizations. High-quality code means fixing myriad little bugs, some of which customers never experienced. Continuous, low-risk tech-debt reduction requires a hundred small refactorings. Security patches and library upgrades are mandatory maintenance. Documentation tweaks are helpful, low-risk, and should be done continuously. Great writing in general requires myriad tweaks, not grand organizational "pivots;" most edits to this document change a single sentence.

In bulk, Sand is mandatory for wonderful, high-quality software. When you use a piece of software and say, "Wow, this is really well-done," that's a result of Sand. While each grain is typically impossible to "measure," ignoring them means we will never attain greatness, even with the best strategy.

Maximize throughput

Sand has a material impact only when executed *en masse*, as a sort of opposite to "death by a thousand cuts." Therefore the goal is to maximize how many of them we fit into the interstitial spaces between the Rocks and Pebbles. The measure of success is throughput: How many we complete per sprint.

We cannot—and should not—try to measure or prioritize "impact." It's too small to measure[6]. It's enough to agree that fixing ten bugs this sprint is a great accomplishment, and that our customers and support techs will thank us for it.

[6] It is conceivable that two days' work has a measurable impact on some metric. When you happen across one of these four-leaf clovers, you obviously should do it, regardless of whether you consider it Sand (because it's fast) or a Pebble (because it has impact). The point is to grant yourself the grace for Sand to *not* have a measurable impact.

Also, high-throughput is fulfilling and energizing for teams. It just feels good to cross lots of things off a list. Do not discount the importance of the *feeling* of productivity and usefulness.

Beware: Administrative overhead destroys throughput

Exactly because each grain of Sand takes little time, management processes will dramatically bloat the total time from conception to prioritization to completion. Administration is the biggest impediment to throughput, therefore PM's must ruthlessly fight against the natural urge to debate and arrange and carefully assess and estimate the work, instead of just going about the business of completing the work.

If you're using the same processes to prioritize and define Sand as to define Pebbles or Rocks, the process is wrong. Add up the time you debate the merits, craft templated user stories, fill out the fields in JIRA, vote and prioritize into this sprint then re-prioritize into the next and the next, and assign and balance work across people. Don't forget to multiply meeting time by the number of people in the meeting. This can easily occupy more time than it takes to complete the item in the first place.

Sand will often originate from engineering; notice how many of the examples above are internal requirements for great software development rather than customer-driven requests. In these cases, it's often a waste of time to perform the usual formalities like user-story-writing, because the engineers already know what to do, and why. You can observe this waste in senseless force-feeding of uninformative "proper product language" and template-filling, e.g.

Type:	Task
Priority:	Moderate
Source:	Engineering
Estimate:	1h
Must-do by:	2022-03-15
Story:	As a software developer I want to upgrade npm package `minimist` from version 1.2.2 to 1.2.3 so that I don't have a security vulnerability.
Linked Goal:	Adhere to corporate security policies of applying security patches within thirty days of a known vulnerability of "low" severity and "low" risk.
Risks:	Might break unit tests, might require refactoring around new APIs or behaviors, might delay other work scheduled in the sprint
Testing Considerations:	Normal unit tests
Acceptance Criteria:	Nothing changes after the upgrade

It can get much worse, all for something that could just be a one-liner to "upgrade `minimist` to at least `v1.2.3` because of a security patch," and in practice the work will usually be done in a few minutes, changing one line in `package.json` and re-running the unit-tests.

Prioritize with intuition and desire, not math and metrics

Because Sand is largely un-measurable, complex prioritization systems won't produce meaningful results. Fortunately, exactly because they don't take long to implement, it's typically not important when they're done, and indeed many never will get done because we always have more ideas than time.

Therefore, this is a great opportunity to engage people's emotions, and go with things people *want* to do. Since delivered-value is near-nil, why not choose things people have energy for? This increases happiness, morale, and often quality. Because people naturally work harder and better on things they want to do, you get "productivity for free," which in turn increases throughput, which is the primary goal.

Self-managed teams schedule their own Sand

In keeping with the rule to minimize administrative overhead, teams should schedule Sand themselves, not debate prioritization with executives.

If the management is constantly engaged in Sand-scheduling, something is amiss and needs to be corrected, and it's always the manager's fault. Perhaps the team is perfectly capable of scheduling Sand; in that case, the manager is micro-managing, which is the manager's fault. Perhaps the team really doesn't understand the customer, the market, the technology, or the work, and therefore truly is incapable of scheduling Sand; in that case, the manager has hired incorrectly or not created an environment where the team can understand these things, which is the manager's fault.

Little requests and suggestions are normal; indeed, these will come from all over the organization. But if you're debating with spreadsheets and virtual-sticky-note-boards, it's gone too far.

Pebbles maximize ROI

Pebbles are not a "balance" between Rocks and Sand; they are their own creatures. Their timeframe is definitionally constrained between that of Sand and Rocks; the more important difference is that while Rocks are strategic, with a view towards winning over the next few years, Pebbles are tactical wins that have an impact in the next few months, attacking the challenges you're facing right now, or a great feature idea you can surprise customers with sooner than they expect.

"We'll start with some small pebbles."

Duration: 1-4 sprints

Pebbles take multiple stories and possibly sprints. Unlike Sand, they do need to have a measurable impact; you can't spend a month or two of the team's time and have nothing objective to show for it.

It is difficult to craft great Pebbles, because impactful things have a tendency to explode in effort. Anything longer than four sprints invokes the Hofstadter problem of time and impact, and therefore must be analyzed and prioritized as a Rock. If a Pebble starts expanding, you have two choices:

1. Reduce the scope of the idea so it can be achieved in a smaller timeframe, or
2. Move the item into the "Rocks" category, where it will be prioritized appropriately.

In both cases, you often discover that the impact is no longer big enough[7]. Either this means the team needs to get more creative in how to deliver more impact with less effort, or maybe this idea simply isn't a good-enough use of your time.

[7] For (1), reducing scope might also reduce impact, in number of customers affected or in the magnitude of the effect. For (2) the impact might have been great for a one-month project, but too small when compared to other large projects.

Pebbles maximize ROI

If Rocks maximize strategic impact over the long-term, Pebbles maximize immediate impact in the short-term. Said another way: They are the "most effective use of the team's precious time."

Pebbles maximize ROI: A measure of value, divided by a measure of effort, resulting in a metric of efficiency. Modest value won in a short time, or more value over more time, are both great uses of time. What you cannot do is deliver little value but still take a long time.

Beware the surprisingly high impact of estimation error on ROI

The Hofstadter problem is magnified with ROI calculations, so you have to be especially careful, especially with classic frameworks like rubrics.

For example, consider a task that ends up producing 20% less impact and ended up taking 50% more time than expected—a common real-world result:

	Estimated	**Actual**
Impact	60	48
Effort	4	6
ROI:	**15**	**8**

This item has *half* the ROI than we originally thought. This is an immense magnitude of error, swamping the signal you thought you computed. Many other items' ROI will fall within this range of error, telling us that the noise from the error exceeds the signal from analysis. Which means the rubric is useless.

To avoid this issue, use this framework for performing an ROI analysis.

Product Managers decide Pebbles

You could argue that the Product Manager's most important job is to make decisions exactly like this one: Which Pebble will we tackle next?

Ultimately it's a judgement call: A synthesis of what customers need most, what competitors are doing, what is consistent with the strategy, what is best for company metrics, and what is best for customer delight. Input from many directions is appreciated, but one mind needs to make the call. That should be the mind closest to the customers, to the product, to the competitors, and to the market, not a drive-by decision from a manager, and not a hostage negotiation with engineers who would rather rewrite a whole module from scratch.

That said, committing a few months of time to anything is a big decision, which means it should have a sensible justification, and some objective measure of impact so we can see whether this activity is having the desired effect. With "self-managed teams" comes not only the freedom to decide and act, but the responsibility to own the results.

A simple sprint-planning system

How can you put all this together in practice, in real sprint-planning?

Schedule things in this order, skipping one if there is insufficient capacity to make significant progress on it given the other items in the list, or if high-quality stories aren't ready-to-work:

1. Time-critical items, regardless of size. *(Examples: security patches, bugs actively impacting customers, critical work for a launch or other event with an externally-imposed, immovable date)*
2. One or more stories from the current Rock.
3. One or more stories from the current Pebble.
4. Sand.

Life is never as simple as that, so here's how to manage the common issues:

"Time-Critical" takes up so much time, we can't make progress on Rocks and Pebbles

You have a meta-problem: Your team doesn't have enough time to be effective; solving *this* is now your top priority. This problem is even more important than your Rock, because in this condition you won't actually complete any Rock. There are myriad causes, and maybe multiple simultaneously: Is it a problem of individual productivity, of the team owning too many things, of architectural dependencies, of the problem-domain requiring more people, of lacking specialized skill sets, of greater fortitude of saying "no" to certain requests, or what? You must diagnose and cure the disease. Schedule sprint time to work on the solution.

Starving the Rock

Just one story per sprint will cause too much context-switching, and take too much calendar-time. If you're constantly starving your most strategic item, this is an impediment that the PM needs to address with the team. You're probably falling prey to the Eisenhower Matrix fallacy of working on things that are urgent, rather than things that are important. Maybe you need to pause your Pebble for a while?

More than one Pebble

If you're truly going to execute on all four sections, you don't have time for two Pebbles at once. Plus, the context-switching is worse for morale and for productivity. The exception is when a Pebble is essentially complete, or will be blocked for at least a week, and therefore you truly do have the time to work on something else.

Always the Rock, never the Pebble

Because Sand is definitionally small, a few of those items always fit. But, a significant story for a Pebble may not. What if Pebble stories *never* fit? Consider that this might not actually be a problem, if the Rock is so valuable. Declaring the Rock "the theme of the next four months" might be exactly what the team needs for focus and maximum impact. It could be that once the Rock is "done" (in the sense of "first com-

plete version"), you can then tackle a Pebble and then another, just doing incremental (small) updates to the Rock as you continue to learn and evolve its result, rather than immediately tackling an entirely new Rock.

Zero Sand

Since it's last on the list, and often aren't even full stories, it's easy to just never do Sand, but this will result in a poor product and unhappy engineers (as much of their internal work falls under this category). Consider toning down stories from other areas, maybe pausing the Rock or Pebble for one sprint, or even having a sprint devoted only to Sand, as a fun way to get a ton accomplished in a short amount of time and to break up the monotony of the sprint-planning cycle.

Starting a new Rock or Pebble too soon, rather than creating more quality and value from the ones that just "completed"

With long lists of genuinely terrific Rocks and Pebbles, it's tempting to start a new one as soon as the current one is complete. But software rarely works that way. Between learning how customers actually use (or don't use) things in the field, completing small items that were originally deferred (so that we shipped sooner and started learning sooner and started selling the feature sooner), and both incremental and significant follow-on functionality, often you need to keep the Rock and Pebble around longer than it first seemed, or at least not start a new one quite yet. This is realistic for great software and a healthy, sustainable pace of work, and this is another reason why our "time estimates" on both Rocks and Pebbles are subject to Hofstadter's Law, and further justifies our draconian admonitions about identifying and prioritizing that work.

Trying to "balance" every sprint

It's not important that every sprint is perfectly balanced between all types of work. It *is* important that we're balanced over a period several months, otherwise something important is getting starved. Indeed, it's often wise to build imbalanced sprints intentionally, because that means greater focus, less context-switching, and therefore getting more quality work done.

Of course all this is easy to say, but hard to execute. Still, when we write it down as simply as possible, and try to honor it, we'll make better decisions sprint by sprint, which in turn creates the most impact year by year.

It's tricky to navigate, but this framework will help you spend everyone's time more wisely.