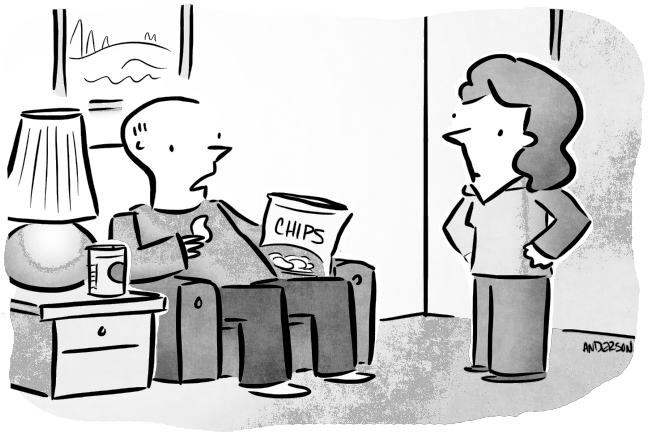
The fundamental forces of scale

by Jason Cohen on September 24, 2023

These forces make larger companies slower and more difficult to execute, but also more effective when harnessed and leveraged.



"I'm not letting myself go, I'm scaling."

Idealistic founders believe they will break the mold when they start scaling. They will *not* turn into a "typical big company."

By which they mean: No stupid rules that assume employees are dumb or evil, with everything taking ten times longer than it should, with wall-to-wall meetings, and hiring "average" or "normal" people.

That is, preserving the positive characteristics of tiny organizations while avoiding the common problems of large ones, by adapting the startup's existing processes, that have served them so well thus far. We're merely doing it with more people now, and still figuring it out as we go along, exactly as we always have.

Why do they never succeed? Why is it impossible for all of these intelligent, well-meaning founders to run a 500-person organization like a 50-person one? What are these unavoidable forces of scale?

Are they really unavoidable?

The two fundamental challenges of scale

After understanding these two primary drivers, we'll see how they explain the challenges that arise in every corner of the business.

Rare things become common

At scale, rare things become common, as fully explored in that linked companion article.

Rare things are difficult to predict, and typically difficult to prevent. Before scale, they are, well, *rare*, so this doesn't matter; they can be handled manually. But when they start happening daily, new processes become necessary.

For instance, with 1,000 servers, a server failure every three years translates to one failure per day. This frequency impacts not just DevOps but also customer support and social media teams. Customers are impacted and complaining every day, even if each individual event is rare, impossible to predict, and impossible to prevent. Automation helps, but doesn't solve the problem.

This is inescapable math crops up all over the business.

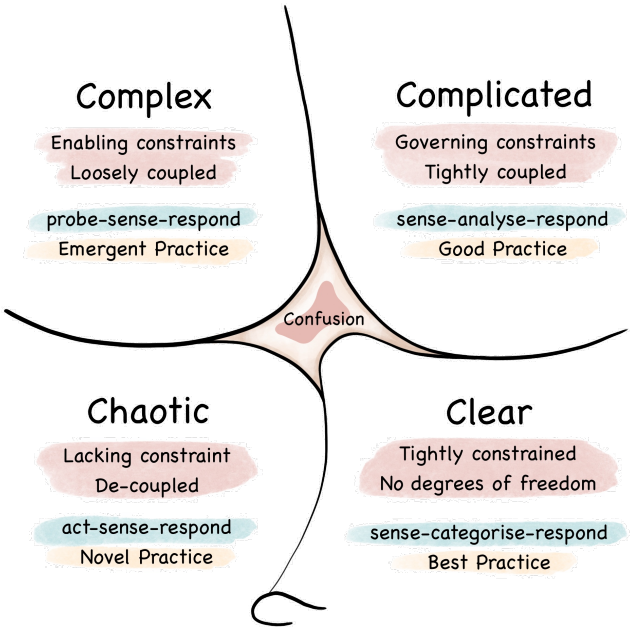
Complexity beyond human comprehension

Human minds have limited computational abilities. A sufficiently complex system cannot be fully comprehended at all.

"Complicated" systems can be comprehended. These types of systems are intricate, but they exhibit properties which allow people to tame the beast. Components of complicated systems can be separated, built and tested on their own, then assembled; thus, divide-and-conquer is a useful technique. Challenges are difficult but are

solvable, especially if you have seen them before; thus, "experts" and "specialists" can solve the puzzles. Repeatable processes can monitor the system.

"Complex" systems, in contrast, have components which affect each other, often cyclically. Therefore, while it's still useful to build and test components separately, there's just as much complexity in their interaction. Indeed, most of the difficulty lies in those interstitial spaces. The human brain is composed of neuron "components" which are relatively well-understood; it's in the complex, multi-directional, cyclical interactions that everything interesting—and inscrutable—unfolds. Large-scale effects like "consciousness" completely evade our understanding, as it emerges from complexity we have yet to comprehend.



This terminology comes from the Cynefin Framework.

As a company scales, there are both kinds of systems, but the complex ones are fundamentally difficult, and never stop being difficult. Their difficulty grows non-linearly as the company grows.

And even our unthinkably complex human brains, cannot comprehend the system.

With the two fundamental principles in hand, we can detail specific challenges of scale.

The human cost of robust teams

A "team of one" is the fastest, most efficient team, if you measure by "output per person." There's no communication, no meetings, and decisions can be made instantly. Small companies operate this way by necessity, and it works! It's a big reason why they move quickly, and "punch above their weight" as is often said.

But, an illness takes the velocity of the product or the quality of customer service from heroic to zero. With a small team, if one person leaves, you've just lost six months to hiring and getting-up-to-speed on that project. Or twelve months or a complete rewrite because there wasn't any processes and documentation in place… because it was just one person, who didn't need that stuff, because after all we're moving so quickly! And not communicating!

Or it's fatal because that was a co-founder. "Founder trouble" is a leading cause of startup death[1].

[1] Although data also show that companies with only one founder are more likely to fail. So which is better? To me, it comes down to temperament—some people really want to be the only one at the top, and some want to share the burdens with others.

"I dunno, 'first mate' has been done to death. How about this -- '*Co*-pirate?'"

So **a team of one is fast, but brittle**. When you're small, this is a good trade-off, because speed is critical for combating the things that are constantly about to kill the company—lack of customers, lack of market attention, lack of core features, all the things. When you're large, with 15-25% annual employee turnover, not to mention illness, vacation, and family, people will be leaving (temporarily or permanently) every day—rare things becoming common. Maintaining the same attitude as when you were tiny is disorganized and irresponsible, and the company would fail to function.

So, at scale, no project can have fewer than, say, three people dedicated to it, plus management and possibly some form of Product or Project Management. But that team of 4-5 will not be 4x-5x more productive than the one-person team; **per-person productivity goes down in exchange for robustness and continuity**.

On the other hand, while the small company loses 9 months to the loss of a key employee, or even implodes, the big company is the steady turtle that adds thousands of customer per month like clockwork and wins the race.

The high cost of robust systems

You might think that software systems wouldn't suffer the same brittleness as human systems, because we have things like automation and DevOps. Sadly, robust software is similarly much more costly and complicated.

Consider the case of serving a website (don't worry, you don't need to be an software engineer to follow this example!). Suppose there's a network connection to the Internet, a server that runs the web site's software, and a database that holds the web site's content. We have a single server, and it works most of the time. Let's say it works 99.9% of the time; that's sounds pretty good!

Well, it's not that good, actually. Failing for 0.1% of a year means it has failed for more than 500 minutes—surely an unacceptable amount of down-time! This happens because a rare thing (0.1% failure) became common (at the scale of about half a million minutes in a year).

What if we had two servers instead? That way, when one fails, the other is still available. They'd *both* have to fail *simultaneously* for the website to be offline, which would happen `0.1% × 0.1% = 0.0001%` of the time, which would be only 30 seconds per year, so that really *is* great[2].

OK, that's twice as expensive, because we now have two servers, but it's worth it for being more robust[3]. But wait, that's not enough, because the traffic that comes in from the Internet needs to be distributed across the two servers, and it needs to have some intelligence to route traffic to the healthy one, in case one is failing. That's called a "Load Balancer," and all cloud infrastructure providers have them, and they cost more money. So that's another component to manage, and more expense.

[2] Sadly this is not really how the world works; typically failures are correlated or cascade. We'll keep things simple for sake of the example, but this complication proves the point yet again.

[3] You might think we can save money by buying cheaper servers, because each needs only one half the capacity of the original. But no, because in the failure mode, a single server still needs to serve all the traffic, so each needs to be powerful enough to work on its own.

And wait, there's also that database. The two servers need to share a common database of content, so the database needs to be moved off to its own server. And what if *that* server fails? Then we have downtime again. So we also need two of *those* for continuity.

Now we're up to four servers plus a Load Balancer, and more than 4x the original cost. You can buy robustness, but it's far more expensive than you might think. We haven't even gotten to the fact that managing multiple servers is more difficult, or that there can be failures in the communications between servers, or failures in the Load Balancer which takes down the whole system[4]—all decreasing our robustness, requiring even more effort to deliver the end result.

[4] Yes, there's solutions to this too—more redundancy at the network layer, often with a different vendor, but again, all more cost and complication!

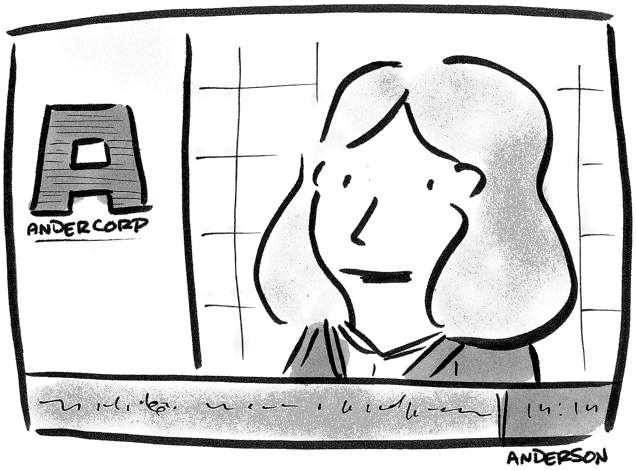
Robustness always carries a cost. Whether it's running four servers instead of one, or a team of four instead of one, you trade local inefficiency for whole-company continuity.

Predictability (that you actually need)

When you're small there's no need to predict when a feature will ship. Marketing isn't scheduling a launch and Recruiting isn't timing the start-dates of the next 50 hires in customer service and sales. This means you can—and should!—optimize myopically for speed-to-market.

Small companies correctly brag about their speed as an advantage, but it's easy to see how a larger company has a different—and massive—advantage of execution. Sure, when our company WP Engine launches a new product, the marketing department needs predictability for the launch date, but that's because it's a highly-skilled, well-funded, coordinate group of teams that explode with press, events, campaigns, social media, and newsletters, grabbing more attention in a single week than a smaller company might scrape together in a year. There are also global Sales and Support teams, so we're immediately selling to 200,000 existing customers as well as thousands of new customers per month, which means we'll add more additional revenue each month than a small company will earn over a whole year.

But all this requires predictability. We didn't line up that press and have those sales materials and train hundreds of support reps and ensure that code-quality is high enough to scale on day one, without predictability. Predictability requires going slower. Predictability requires estimation (takes time), coordination (takes time), planning (takes time), documentation (takes time), and adjusting the plan across all teams when life inevitably unfolds differently from the prediction (takes time).



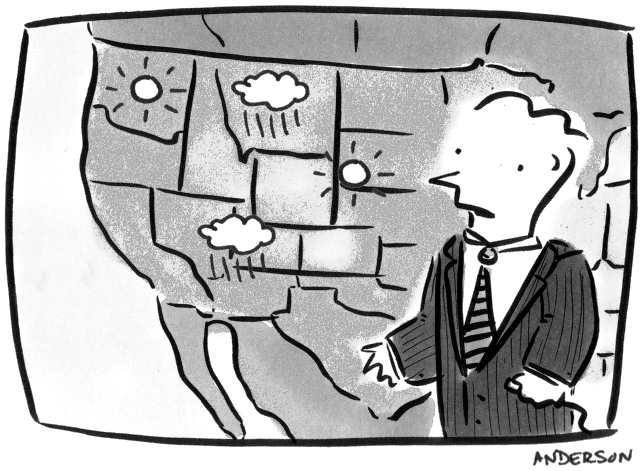
"AnderCorp surprised investors expecting the worst today by performing completely miserably."

When you're hiring lots of people, you need predictability. Consider the timeline of adding a technical support team member. First, Recruiting is casting about for potential candidates. Then scheduling and performing interviews. Then waiting for them to quit their job and take a week off. Then new-employee-orientation. Then classroom training. Then paired up with senior folks on the

floor as they ramp up their skills and comfort. Therefore the time between deciding to grow the support team, and having new people up-to-speed, is four to six months.

Therefore, we have to predict the demand for technical support four to six months in advance, because we need to be hiring for that *right now*. If we under-estimate, our support folks get overwhelmed with too much work; their quality of life suffers, and service to each customer suffers. If we over-estimate, we have too many people, which is a cost penalty. Of course the latter is a better failure mode than the former, but both are sub-optimal, and the solution is predictability.

"The future is inherently unpredictable," insists the small company, spurred on by Lean and Agile mindsets, and the truth. Indeed, blue-sky invention and execution are unpredictable. But this is also a self-fulfilling prophecy; to insist the future is unpredictable is to ignore the work that could make it more predictable, which makes it in fact unpredictable *to that person*.



"But, to be fair, there's a fifty-percent chance of just about anything."

Small companies don't have the data, customers, institutional knowledge, expertise, and often the personal experience and skillset to predict the future, so they are usually correct in saying it's impossible. But is it impossible in principle, or is it impossible for *them*? At scale, it becomes required. Not because Wall Street demands it, nor because investors demand it, nor any other causal derogatory excuse made by unpredictable organizations, but because it's critical for healthy scaling.

The Materiality Threshold

If Google launches a new product that generates $10,000,000/year in revenue, is that good? No, it's a failure. They could have taken the tens of millions of dollars that the product cost to develop, used that to make their existing operation just 0.01% more effective, and made far more money.

At more than $100B/year in revenue[5], Google can only consider products which have the potential to generate $1B/year in revenue as an absolute floor, with the potential to grow to $10B/year if things go better than expected. Things like YouTube, Cloud, and self-driving cars.

[5] *Editor's note:* This was originally written in 2017; in 2023 Google's revenue is nearly $300B, which reinforces the original point; they could not have achieved that growth if they worked on projects that generated revenues of "merely" $100M.

This principle is called the "Materiality Threshold," i.e. the minimum contribution a project must deliver, for the project to materially affect the business.

With a small business, the materiality threshold is near $0. A new feature that helps you land just a few new customers this month is worth doing. A marketing campaign that adds two sign-ups/week is a success. Almost anything you do, counts. That's nice—it feels good to be moving forward.

The financial success of the larger company dictates a non-trivial materiality threshold. This is difficult. Even a modest-sized company will need millions in revenue from new products, maybe tens of millions in the optimistic case. Very few products can generate that sort of revenue, whether invented by nimble, innovative startups or stately mature companies. As proof, consider that the

vast majority of startups never reach a $10M/year run-rate, even with decent products and extraordinarily dedicated and capable teams.

Yet, it's the job of a Product Manager at that mid-sized company to invent, discover, design, implement, and nurture exactly those products—something that most entrepreneurs will never succeed at. Tough job!

Recruiting

Employee #2 will join a startup for the experience. Even with a significant salary cut, and even if the company fails—the most likely outcome—it's worth it for the stories, the influence, the potential, the thrill, the control, the camaraderie, the cocktail-party-talk.

Employee #200 won't join for those reasons. Employee #200 will have a different risk-profile regarding their life and career. Employee #200 will be interested in different sorts of problems to solve, like the ones listed in this article instead of the ones where you're trying to understand why 7 people bought the software but the next 3 didn't. Employee #200 will not work for a pay-cut.

"So, where do you see yourself in ten minutes?"

Small companies view this as an advantage, and certainly it's advantageous to recruit amazing people at sub-market rates. But there hundreds of employees at WP Engine today who are much more skilled in their area of expertise than I've ever met at a small startup, including my own. Why?

One reason is that people with a lot of experience are often in a different phase of life, where family and other demands means they want a predictable, larger paycheck for a predictable, well-defined job. When you combine their advanced skills with the raw materials of a scaled company (i.e. customers, brand, funding, teams that can execute larger ideas), they can generate huge value while still tucking their kids into bed at night.

Another reason is that, after developing that expertise, they find it's enjoyable to apply their skills within a larger environment. For example, there are advanced marketing techniques that would never make sense with a smaller company, that are fascinating, challenging, and impactful to the top line at a larger company. There are talented people who love that challenge and would hate going "back to Kindergarten" as Jeff Bezos famously quipped, scratching out an AdWords campaign with a $2000/mo budget or assembling the rudiments of SEO or just trying to get a single marketing channel to work or being called a "growth hacker" because they finagled a one-time bump in traffic.

This has implications on compensation, how you find that talent, and why that person wants to work at your company instead of the one down the block who can pay a little bit more. Therefore, it's critical to have a mission that is genuinely important, have meaningful and interesting work to do, connect everyone's work to something bigger than any of us. These matter even more at scale, because it's the reason why talent will join and stay.

"I don't get it! We've got a mission statement, a credo, *and* a mantra!"

Communication

With four people in a company, any information that everyone needs to know can be told to just three other people. Everyone *can* know everything. If there's a 5% chance of significant misunderstanding, that doesn't happen often.

With four *hundred* people, it's never true that a piece of information can be reliably communicated, in a short period of time. A 5% chance of misunderstanding means twenty people are confused. That's assuming they even read the communication, or listened during the entire presentation over Zoom. What's the chance of *that*? Don't ask.

"Slack" is not the answer. "Email" is not the answer. The answer is: **Repeating simple messages.**

"Repetition" is the answer to "I didn't see it." Repetition in different formats, at different times, by many leaders. Asymptotically achieving 100%, although also creating collateral damage for the people who really do read and listen to everything, who are tired of hearing the same thing, and wondering why they're being punished for actually paying attention.

"Simple" is the answer to "I didn't understand / remember it." Just as with simplicity in strategy, you have to accept that people don't read, people don't remember, people have other things on their minds, people don't understand language as well as you wish, and you didn't write as clearly as you hoped.

But doesn't this mean you cannot communicate anything complex to 1000 people? Yes, that's what it means. Scale is hard.

Technology & Infrastructure

Managing 10,000 virtual servers in the Cloud Era sounds easy. Automate everything, then any process that works for 100 servers, will work for 10,000 servers just by doing the same thing repeatedly—exactly the thing computers are excellent at.

It never works like that. Reddit took 18 months to get "number of likes" to work at scale. StackOverflow took 4 years to get everything converted to HTTPS. Wired did that conversion in a "mere" 18 months. Everything is hard at scale.

What are the patterns in those stories?

One is another application of "rare things are common." Rare things are hard to predict and can be hard to prevent. Often they're hard to even identify and sometimes impossible to reproduce. This is fundamentally difficult.

Another is continuity or compatibility with existing technology. New companies get to start from scratch, but at-scale companies must transform. New companies like to make fun of large companies for how hard it is to transform, neglecting that the cause of the difficulty might also be generating $100,000,000 in revenue this year[6].

[6] Instead of "legacy code" we call it the "revenue code."

Another is bottlenecking. All hardware and software systems have bottlenecks. At small scale, you don't run into any bottlenecks, or at least the ones you do can be solved with simple techniques like increasing capacity. Eventually something difficult breaks and you have to rearchitect the stack to solve it. Even something simple like converting HTTP links to HTTPS or updating "number of likes" in real-time, becomes a monumental architectural challenge.

Another is Hickam's Dictum: Problems in complex systems generally have more than one root cause. Consider the case of a software upgrade causing problems for seven customers, in a way that wasn't detected. What is the "root cause" of this problem?

- The problem is we didn't detect the failure; had we done so, we could have reversed the upgrade immediately.
- The problem is we didn't test the thing that failed; had we had more tests, the problem would never have reached customers.

- The problem is we didn't document the code that broke; had the code been clearer, the human wouldn't have coded the error.
- The problem is we didn't review the code properly; had the code been reviewed with a proper checklist, we would have discovered the bad code before it was deployed.

As with Five Whys, there are always deeper reasons, as well as different reasons. Different from Five Whys, there isn't just one that is the "root" cause; instead there are a lot of things that interact with a lot of things. This doesn't mean it's hopeless, it just means it's complex to analyze, complex to decide what to do, and that Occam's Razor doesn't apply.

All this slows down development and adds investment. There will be entire teams who focus on infrastructure, scaling, deploys, cost-management, development processes, and so forth, none of which are directly visible to or driven by the customer, but which are necessary to manage the complexities of scale.

Risk-mitigation

For a small company, the most likely cause of death is suicide. Usually it's starvation—can't get enough customers (distribution) to pay enough money for long enough (product), in the cascade of things-that-must-be-true for product/market fit. But also things like founders splitting up, not getting enough traction to self-fund or to secure the next round of financing, having to go back to a day job, and so on.

At scale, the risks are different. There is very low risk that WP Engine will not sign up thousands of new customers this month. Other risks, however, are not only possible, but likely. Addressing those risks head-on, is required for a healthy and sustainable business that can last for many years.

Take the risk of business continuity during a disaster scenario. What if an entire Amazon datacenter became disabled for a week? How quickly could we get all those customers back up and running? Would that be true even though thousands of other businesses are also trying to spin up servers in other Amazon data centers at the same time? Could we communicate all this with our customers quickly and simply, so that our support team isn't overwhelmed by repeating the same message to tens of thousands of justifiably-angry customers?

"It's important to remember that correlation does not imply causation. Besides, we all know it was Brian."

Risk-mitigation can even result in growth. Serious customers want to work with vendors who understand and mitigate risk; this maturity becomes a selling point. That's why enterprise suppliers (like WP Engine) are constantly flouting their compliance with SOC 2 and ISO 27001 and all the others. Small companies make fun of those things as being unnecessary at best or a false sense of security at worst, but while they're busy making that point, the larger companies are busy signing three-year multi-million dollar deals.

Early on, you do not need a disaster-recovery plan. That won't be the thing that will kill the business, and your customers will understand if a young business is subject to that sort of risk. Later on, this becomes critical, and worth investing in.

Large companies are not just small companies × 100

These forces cause larger companies to be fundamentally different than small ones. It isn't a bad thing or a good thing. It's a different thing.

Some idealistic founders believe the root cause of scaling issues is the "command-and-control" organizational structure. But none of the examples above make reference to any organizational structure. It's universal. This is why Holacracy and Teal Organizations do not solve these problems. It could be a fantastic idea to experiment with organizational structure, but the fundamental forces above will not be eliminated with a recombination of roles and power structures.

Scaling is hard, the road is foggy and bendy, it lasts for years, the set of people you need might be different, and no one emerges unscathed. So, it is not a sign of disaster if you have difficulty wrestling with these forces. Everyone does.

Disaster is when a company is scaling, but the leaders don't appreciate these forces, don't work constantly to morph the organization accordingly, don't bring in experienced talent, decide they can figure it all out as they go along without help. Rather, it should mean new people, new roles, new values, new processes, new recruiting, new stories, new constraints, new opportunities.

Too many founders and leaders want to believe:

> What got us here is what's important and unique about us, and thus we should preserve *all* of it. Other companies fail because they "act like big companies," but we'll avoid all that because we're smarter than they were. As evidence of our acuity, just look at our success thus far. We will continue to succeed in the future as we have in the past.

But they're wrong.

There should be a few values that are kept constant, that's true. Otherwise none of it means anything. But the details must change.

Many founders and leaders can't make the shift. This always hurts the company, and sometimes kills the company. The world is full of those horror stories. It's sad, because it's an avoidable waste of opportunity and sometimes hundreds of person-years of effort.

Don't become one of those cautionary tales.
