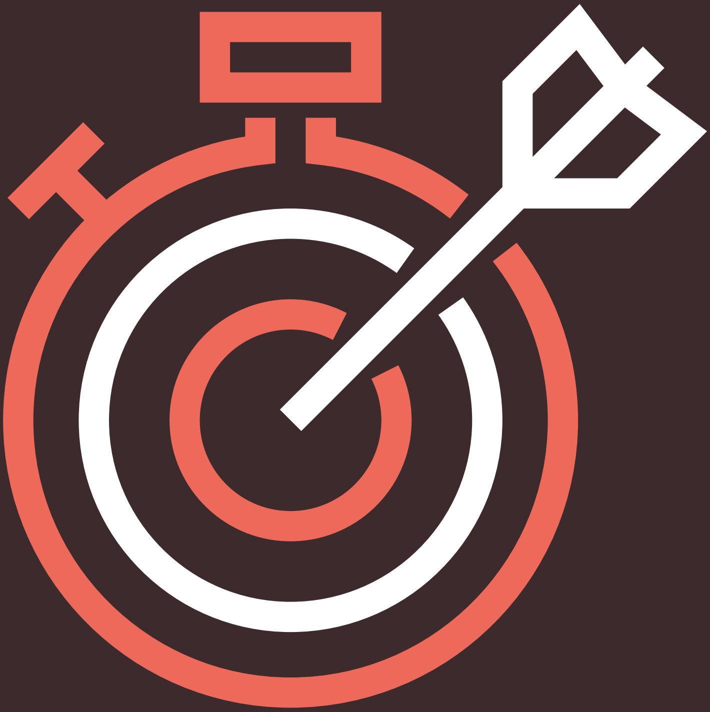
The Lean-Agile Dilemma

Product Management Inside a Chunky Corporate

Katie Tamblin

Apress®



The Lean-Agile Dilemma

Product Management Inside a Chunky Corporate

Katie Tamblin

Apress®

The Lean-Agile Dilemma: Product Management Inside a Chunky Corporate

Katie Tamblin
Welwyn, UK

ISBN-13 (pbk): 979-8-8688-0320-8 ISBN-13 (electronic): 979-8-8688-0321-5
https://doi.org/10.1007/979-8-8688-0321-5

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Shivangi Ramachandran
Development Editor: James Markham
Project Manager: Jessica Vakili

Table of Contents

About the Author

Katie Tamblin started her career as an entry-level analyst and made it all the way to the board room, working as Chief Product Officer and now serving as a Non-Executive Director to data and tech businesses and an Advisor to private equity firms. She applies the learnings amassed over a 25 year career to help readers recognize a chunky corporate for what it is and to navigate its unique qualities in order to drive efficiency and success.

Introduction

As a product professional, I have spent hundreds of hours writing business cases for new products. I've spent even more time evaluating business cases related to software product development. In retrospect, it seems like a lot of wasted energy because, looking back on it, I can see now what was invisible at the time. Business leaders have very specific conditions under which they will approve a business case or enhancement request. At large, mature tech-enabled businesses, these criteria differ wildly from what the textbooks will tell you. Making sound product decisions and executing a technology transformation efficiently depend on a shared understanding of the objectives of your leadership team—objectives that stretch well beyond the boundaries of what a product can do.

The Lean-Agile Dilemma is a compilation of lessons I learned over a 25-year career in data and product management. It is designed to help product managers, engineers, and business leaders work more effectively. I am not speaking from the perspective of a management consultant or academic, but from the experience of an entry-level employee who climbed up the corporate ladder, rung by rung.

I started out as an analyst at Delta Air Lines after finishing university in 2000. After pursuing a master's degree, I was hired at an economic forecasting company as an economist. Over the coming years, across multiple companies, I was promoted to Team Manager, Product Manager, Product Director, Head of Product and Pricing, Chief Product Officer, and Board Member. I managed the supply chain product portfolio for a $5 billion in annual revenue information behemoth called IHS-Markit (now part of S&P Global). I have run product, marketing, technology, and data science teams for private equity-backed technology businesses. I advise a number of private equity houses with combined asset investments of over $220 billion.

I am not the CEO of a software unicorn. You've probably never heard of most of the companies for which I've worked. But I have watched, at every level within an organization, how colleagues at everyday technology-enabled businesses misunderstand, misinterpret, and misapply the things their business leaders want them to do. Senior leaders think their people understand the goals of the business, but that does not mean colleagues know how to align their daily activities to delivering business goals. It is extremely difficult to see from the top where things are going wrong. It is much easier to understand how projects get off track, and how to get them back on track, when you are down in the weeds, making daily decisions that impact the business. I write from that perspective – in the thick of it, rather than the view from the top.

Throughout my career, the more experience I gained working on large technology transformations, the more I felt Lean-Agile wasn't a good fit in my working environment. I was taught (repeatedly) to follow the Lean-Agile method for product and software development. Simultaneously, it was impressed upon me, by the actions of senior leadership, that product and software innovation would not be given room in the corporate budget. I felt like a kid stuck in a custody battle, with one parent espousing one method and another parent regularly undermining it. And, why, I asked myself, was this feeling of push-and-pull constant across multiple companies and roles? Well, it's because it comes from a mismatch between popular development principles and the priorities of an investor-backed business. Let me explain.

The Lean Startup: How Constant Innovation Creates Radically Successful Businesses is a phenomenal book whose success created a population of product managers that want to innovate intelligently. Its publication in 2011 marked the beginning of a paradigm shift in Agile culture. It speaks directly to product managers and product owners, who go on to enthusiastically apply Lean-Agile principles wherever they work. When they join the workforces of typical mature organizations, however, they are met with a stark reality check. The bigger the company, the more they struggle to apply Lean-Agile principles.

Agility is not a natural attribute for a group of hundreds or thousands of individuals working alongside each other. Building software and managing data effectively depends on efficient collaboration. However, when you have more people on a project, that gets harder. Lean-Agile principles are meant for small teams innovating radically to identify and capture new markets. The clue is right there in the title: *Constant Innovation*. Investor-owned businesses don't want constant innovation. They want predictable performance. Innovation is risky and unpredictable. Bigger, more established businesses need a different toolkit – one built for what I call “chunky corporates,” not lean startups.

This book is part catharsis – I wrote it to process and understand why things go wrong when they go wrong. The how-to portions of this work are derived from many years of screwing up. By learning the hard way how *not* to, and then working out by reflection, trial and error, what might be a better approach, I have found solutions to common challenges. My aim is to help others avoid common mistakes. Over the following chapters, I will outline in detail how software projects get off track. We find a common theme in the misapplication of Lean-Agile principles.

To start, I explain how chunky corporates differ from startups and why that makes Lean-Agile principles a poor fit. Even the giants of Meta, the company formerly known as Twitter, Microsoft, and Google shed thousands of workers in the early 2020s as their investor/owners demanded margin growth. They, too, have become chunky corporates. I'll explain how individual players in the corporate organization impact business performance. We follow the story of Blake, a well-meaning but ultimately naive CEO of a fictional company called Acme Tech (see Figure 1 for a simplified organization chart).

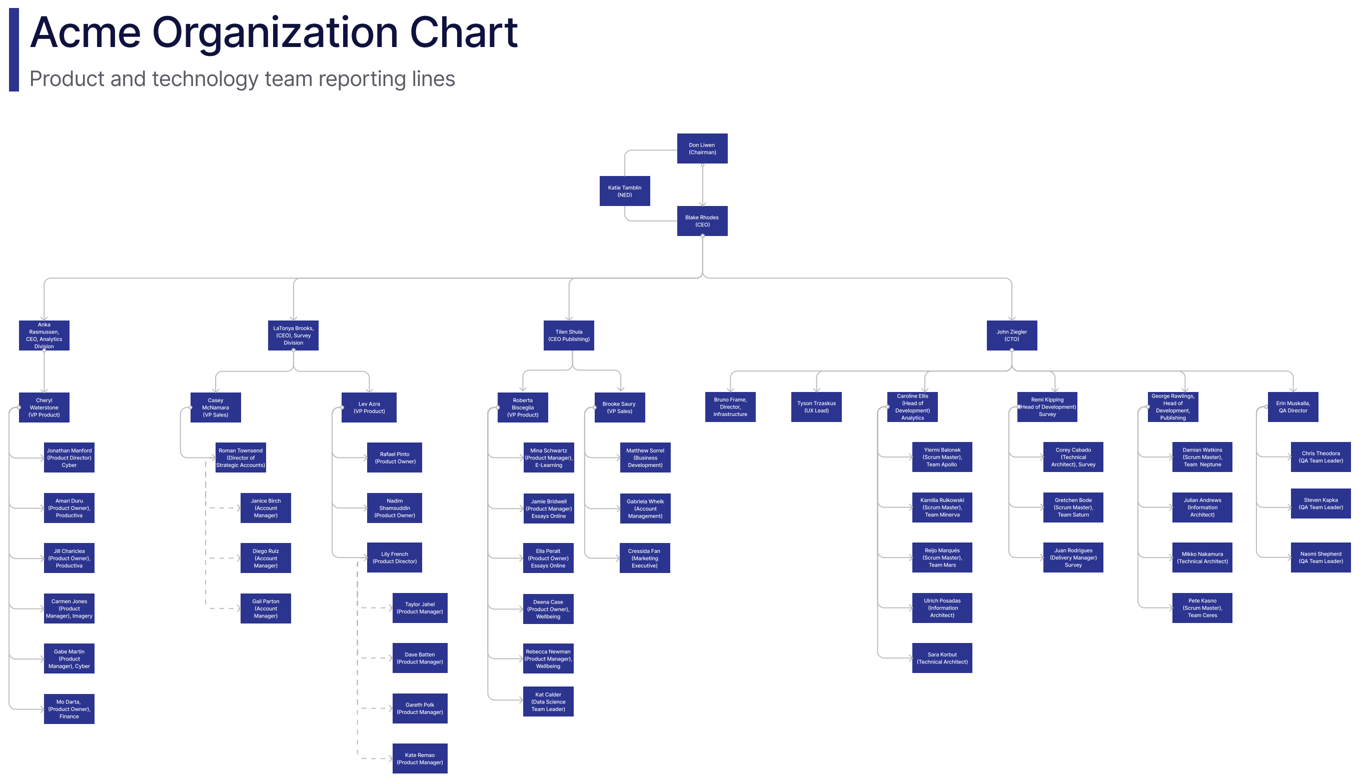

Acme Organization Chart
Product and technology team reporting lines
Don Liwen (Chairman)
Katie Tamblin (NED)
Blake Rhodes (CEO)
Anka Rasmussen, CEO, Analytics Division
LaTonya Brooks, (CEO), Survey Division
Tilen Shula (CEO Publishing)
John Ziegler (CTO)
Cheryl Waterstone (VP Product)
Jonathan Manford (Product Director) Cyber
Amari Duru (Product Owner), Productiva
Jill Chariclea (Product Owner), Productiva
Carmen Jones (Product Manager), Imagery
Gabe Martin (Product Manager), Cyber
Mo Darta, (Product Owner), Finance
Casey McNamara (VP Sales)
Roman Townsend (Director of Strategic Accounts)
Janice Birch (Account Manager)
Diego Ruiz (Account Manager)
Gail Parton (Account Manager)
Lev Azra (VP Product)
Rafael Pinto (Product Owner)
Nadim Shamsuddin (Product Owner)
Lily French (Product Director)
Taylor Jahel (Product Manager)
Dave Batten (Product Manager)
Gareth Polk (Product Manager)
Kate Remao (Product Manager)
Roberta Bisceglia (VP Product)
Mina Schwartz (Product Manager), E-Learning
Jamie Bridwell (Product Manager) Essays Online
Ella Peralt (Product Owner) Essays Online
Deena Case (Product Owner), Wellbeing
Rebecca Newman (Product Manager), Wellbeing
Kat Calder (Data Science Team Leader)
Brooke Saury (VP Sales)
Matthew Sorrel (Business Development)
Gabriela Whelk (Account Management)
Cressida Fan (Marketing Executive)
Bruno Frame, Director, Infrastructure
Tyson Trzaskus (UX Lead)
Caroline Ellis (Head of Development) Analytics
Yiermi Balonek (Scrum Master), Team Apollo
Kamilla Rulkowski (Scrum Master), Team Minerva
Reijo Marqués (Scrum Master), Team Mars
Ulrich Posadas (Information Architect)
Sara Korbut (Technical Architect)
Remi Kipping (Head of Development) Survey
Corey Cabado (Technical Architect), Survey
Gretchen Bode (Scrum Master), Team Saturn
Juan Rodrigues (Delivery Manager) Survey
George Rawlings, Head of Development, Publishing
Damian Watkins (Scrum Master), Team Neptune
Julian Andrews (Information Architect)
Mikko Nakamura (Technical Architect)
Pete Kasno (Scrum Master), Team Ceres
Erin Muskalla, QA Director
Chris Theodora (QA Team Leader)
Steven Kapka (QA Team Leader)
Naomi Shepherd (QA Team Leader)

Figure 1. *Acme organization chart*

As Blake and Acme struggle to maintain business performance, I help Blake understand how embedded and complex his challenges are. Effective software development in large organizations depends on efficient communication across large groups operating simultaneously. That is incredibly difficult to achieve. In the thousands of decisions product managers, product owners, data scientists, architects, and engineers make on a daily basis, they determine the future prospects of Acme.

Building software is the easy part. Managing expectations and coordinating activities across a large group of people is hard. I demonstrate how individual motivations, character traits, and skills impact product decisions. When Acme's product decisions are not aligned to the goals of the organization, products undermine business performance, rather than supporting it. This is a very real challenge that nearly all mature software businesses will face.

We see how easily business performance can falter and how difficult it is to get back on track. Throughout the book, Blake and I navigate common issues that erupt when various divisions of his company encounter challenges, like rebuilding aging software, fixing data issues, and product fragmentation. I help Blake to steer Acme back to where it should be: on the path of predictable financial growth sought by its investors. I outline a new software development methodology that takes the best of Lean-Agile but adapts it to the constraints chunky corporates face.

The mistakes, inefficiencies, and failures described in the book are based on real-life experience. The mistakes are my own, and I bear full responsibility for them. They were either errors in my own judgment or in the judgment of a team in which I was an active participant. None of my anecdotes are meant as an indictment of the decisions we made at the time. Most of us are doing the best we can with the resources we have. More than anything, I hope all my battle scars earned on the corporate front lines can help others to start from a stronger foundation. Managing transformational technology projects in a large mature business is inevitable. Delivering them efficiently – to a high standard, on budget, and on time – can be your competitive edge in a fast-moving market. Not doing so could be your downfall.

That being said, as I prepare to share my reflections with the world, I feel I should make a caveat, lest the Internet trolls start doling out judgment of my hypocrisies (of which I have plenty). Early in my career, I sought promotions as validation that I was good at my job. I wanted to win, to succeed, to get to the top because I am goal-oriented and competitive. However, I learned along my journey that getting to the top of the corporate ladder is not the endgame.

There is always another competition, another level of play higher than the current one. The mountain you are climbing has no summit. If you find yourself on a ledge that feels like the top, it is only a matter of time before you find another ledge above you. A bigger role at the same company or the same role at a bigger company functions like a carrot hanging just out of reach, motivating you (as it would a donkey) to just keep walking.

Corporate cultures that use blind ambition to motivate staff are breeding grounds for corporate politics. Politics are a catalyst for inefficiency and an inhibitor to growth. Genuinely, the biggest time-waster in most of my roles was managing political agents working on agendas not aligned to the core aims of the business. Growth is great. Winning is fantastic. But balance is more important in the long run. I love going to work, but I love coming home more. That has kept me sane throughout the weird and not-so-wonderful chapters of several funky projects.

As I matured in my career, I found better ways to pursue sustainable business performance. Letting down personal defenses helps us achieve effective collaboration, which sucks the oxygen out of political agents. We need to change our mindset from zero-sum competition to efficient, collaborative systems for chunky corporates to deliver predictable performance. Every chunky corporate is a system, and if there isn't the right balance in the work system, workers will find the pace of growth unsatisfying and unsustainable. Our common goal is to create work environments that deliver predictable financial performance without sacrificing the health or sanity of our people.

Lean startups might be sexy, but they are scary as well (being a little bit scary is probably what makes them attractive, which is a separate book altogether). However, constant change is not a strong foundation for the mental health of employees. Over time, people want positive and stable environments in which they can work successfully and predictably. Mature, predictable businesses need to see those qualities as desirable. They should stop chasing the lean dream. Chunky corporates have a lot going for them: when managed well, they have the potential to be safe havens in a relentlessly unstable world.

CHAPTER 1

The Luxury of a Lean Startup

Imagine yourself in a board-style meeting room, across the table from Blake, the CEO of a successful company, Acme Tech. It's 2 p.m. on a Wednesday afternoon, and Blake wears a pasted-on smile. It's forced. He knows it. You know it. You both pretend all is well. To your right is the Chairman of the Board of Directors for Acme, Don, who was appointed by Acme's private equity (PE) owners. Scattered around the table are business leaders and Board members. The conversation starts off congenially, but the team is not relaxed. Once the pleasantries have completed, Blake hands the presentation over to his Chief Financial Officer (CFO), Luke.

Luke looks uncomfortable. Sales are behind forecast, and costs are up. Luke has a handful of reasons why, logically, this is just a blip. "Sales growth will return next quarter. I am confident," he says, but his body language suggests what we all suspect: the business has lost momentum. It is the same feeling you get when you watch your favorite sports team start to lose a game. You can't explain why; they just aren't playing as a team. They aren't executing on the things they have been trained to deliver. But how can you shift momentum when you don't know the root cause of the downturn in performance? How can Blake get his corporate team back on track?

Luke completes his presentation. The mood has shifted. What started as a congenial conversation closes as a stiff and slightly awkward exchange of generic sentences we all say before parting. The frost in the air reveals a shared concern. Don and the private equity associates leave the room. Blake rubs his temples, visibly agitated.

"We are losing customers," he says, "I'm sure it will turn around. We have really good people. We are customer-driven, and I have been working hard to get the team to be *lean* and *agile*."

I don't believe him, though, and neither does the Board. The momentum has shifted. Startup competitors with more modern products are taking market share. Acme is no longer winning. Blake embarks on a fact-finding mission to understand what is behind the change in financial performance. He puts pressure on his managers to deliver results and get leaner. Despite his emphasis on innovation and customer responsiveness, the behavior he observes at his organization is not *lean* or *agile*. It is quite the opposite. Blake is not leading a lean startup. Blake is the CEO of a chunky corporate: a large, mature organization struggling to apply lean principles effectively. If Blake is going to be successful in turning his business around, he needs a new corporate identity and a new method for unlocking productivity: one that is better matched to the unique properties that define a chunky corporate.

Note Lean startup refers to a product development methodology appropriate for new companies or new products. The lean startup method advocates developing products that consumers have demonstrated they will use, proving market existence as the product is launched. The term was coined by Eric Ries, in his incredibly successful and influential book, *The Lean Startup: How Constant Innovation Creates Radically Successful Businesses*.

In contrast to a chunky corporate, a lean startup doesn't have a legacy to deliver. It doesn't have hundreds of existing customers that demand an ever-expanding list of products and features. It isn't moving customers from a previous generation platform to a new one. A lean startup has a blank sheet of paper on which to craft a beautiful portrait of a product suite. It may sound counterintuitive, but lean startups have the luxury of time. Most lean startups are not beholden to risk-averse investors. [Most lean startups aren't beholden to investors at all. Only a small portion are lucky enough to raise venture capital (VC) funding, and VC investors are not risk-averse.] Lean startups are not, in the early days, answering to existing customers on a daily basis who remind them of unfixed bugs, features they asked for that haven't been delivered, or additional items they would like to see added to the platform.

The luxury of a lean startup is in its constraints: anonymity, lack of revenue, lack of existing customers, lack of third-party investor/owners, and lack of a predefined product roadmap. A lean startup has the luxury of freedom but the constraint of resources. This keeps teams small, and the roadmap agile. In that context, the principles espoused in Ries's seminal work, *The Lean Startup*, are a fantastic guide for how to develop a new product and ensure there is a market ready for the product when it is built. Applying Ries's principles can, indeed, be the difference between success and failure of a startup.

Attractive startups, like those operating in hot markets – cyber security, blockchain, or supply chain transparency, for example – can be valued at one thousand times their annual revenue and change the world. Most, however, fail to make a dent in the universe or even survive the first few years. It is safe to say the probability of achieving a high valuation on a startup is small. This wide array of possibilities facing any new business, from wild success to abject failure, is what makes a startup glamorous. A startup is the Danny Zuko of the business world: unpredictable and exciting, but also the one your parents warned you about. We should respect those businesses that survive the early years; achieving success at a startup is a hard-fought road.

As a result of these origins, the culture of a successful startup is miles away from the typical culture prevalent at a mature organization with more history. The differences between the two have important implications for how they can unlock sustainable growth. The aim of this analysis is to provide a useful compare-and-contrast between immature and mature organizations in the software industry, enabling all organizations to capture the best of both. An immature organization should aim to be lean. A mature organization must first understand when it is not.

When chunky corporates apply lean principles to product development, it creates tension and frustration rather than radical success. The constraints that chunky corporates face are very different from those that startups face. Before deciding how to solve its product challenges, a business must first be aligned on what it is and what its goals are. A number of chunky corporate constraints result from answering to external investors or shareholders. The cadence of the budgeting and reporting process associated with investor ownership puts a level of scrutiny on revenue coming in the door and costs going out the door that changes the relationship between the organization and its customers. It changes the organization's values. The longer those processes have been in place, the more transformational the change in values.

Lean-Agile vs. Waterfall Software Development

The Agile software development methodology relies on the collaborative effort of self-organizing and cross-functional teams to deliver and release software frequently to end users. The methodology was popularized by the Manifesto for Agile Software Development[1] and calls for

[1] Beck, Kent, James Grenning, Robert C Martin, Mike Beedle, Jim Highsmith, Steve Mellor, Arie van Bennekum, et al. "Agile Software Development." Manifesto for Agile Software Development, 2001. http://agilemanifesto.org/.

adaptive planning, evolutionary development, early delivery, continual improvement, and flexible responses to changes in requirements, capacity, and understanding of the problems to be solved.

Waterfall is a software development methodology introduced in the 1970s that became the dominant method practiced in the 1980s and 1990s. In the Waterfall method of software development, all required features for a platform are defined in advance. Large projects are outlined, mapped out, and then built. Waterfall is the Big Bang of software development. Spend a year (or more) developing a product, release it to the market, and hope your customers like it as much as you do. Developers often work for months without colleagues or customers seeing what they have done. When the work is done according to the definition, the product is shared internally for quality assurance (QA) testing, and shortly thereafter, it will be shared with customers. I say *shortly thereafter* because project leaders assume developers understand the requirements perfectly and the only thing to do in the quality assurance process is to fix bugs, not adjust the underlying functionality. This creates a really funny tension in the post-build world in which teams will argue, often at length, over whether feedback from users constitutes a bug, a design gap, or a new feature. Let me assure you, those conversations are as soul-destroying as they sound.

The Agile methodology came along to disrupt that way of working (and rightly so!). Agile replaced Waterfall as the dominant methodology in the early 2000s. It stresses collaborative iteration that is more responsive to change than Waterfall. Agile popularized the concept of organizing teams in scrums to maximize collaboration and visualizing workflows via Kanban methods. It is highly effective at driving efficiency in the software development, or build, process. By engaging users earlier in the process, you increase visibility, adaptability, and accountability. Lean principles developed alongside the Agile development methodology as it matured.

The Lean Startup introduced the concept of a Minimum Viable Product (MVP), in which you develop as little as you possibly can before putting your software product in front of customers. Lean principles help development teams efficiently identify product-market fit. Alongside it, Agile governs the structure and processes of teams as they develop and release code. Lean and Agile are so tightly connected in most software working environments these days; many practitioners of Lean-Agile principles couldn't say with confidence which bits are "Lean" and which bits are "Agile." I will refer to Lean-Agile as the application of Lean principles in the context of Agile software delivery.

Through collaboration and iteration, Lean-Agile limits the inefficiencies associated with misunderstanding the requirements. By "misunderstanding the requirements," I am describing the inevitable situation in which the software engineering team doesn't build exactly what was intended. Whether you are working on Waterfall or Lean-Agile projects, the hardest thing for people to see is what *isn't* there. I know it sounds simple, but the impossible task placed upon product managers and product owners is to identify what you've missed before you hand requirements over to a software engineering team and ask them to build a feature. You only see what you missed when the software is in front of you. At that moment you realize you can't complete your task because a button is missing or a piece of data that you need isn't available. What was built doesn't meet the business need if users can't complete tasks.

By giving users a chance to test early and often, Lean-Agile is more suited to addressing our human proclivity to miss stuff and prevents more software being built on top of an already inadequate feature. This makes the build more efficient because if you build more code on top of a feature that is missing something, you have to add that something back in multiple places (the original feature and anywhere else that depends on that feature). If you catch it early, you aren't left scrambling trying to adjust the rest of the code base to account for this new element. Let me explain.

Imagine a blank sheet of paper. On that sheet of paper, you draw a room. It is a usual square box. It has a door and a window. You show it to your colleague, Nadim, who says, "That is a nice room; it could use some furniture." So, you add some furniture: a sofa and some chairs. The furniture you add suggests it is a living room. You show it to a lovely family, the Joneses, who are planning on building a house. They say, "We could really do with a kitchen." So, you draw a kitchen. This process continues until you have drawn the whole house. In the end, the Joneses want a four-bedroom house with a kitchen, living room, playroom, and home office. You have drawn the whole thing in collaboration with them. Then you give that to a construction company, and they build it. Then the Joneses move in.

Three weeks into the move, the Joneses realize they really should have put a power outlet on the south wall of the living room, not the north wall, because they want to put the television on the south wall, but there is nowhere to plug it in. Then they realize they should have added another room because they really want a home gym. They also wish they had put in one of those integrated vacuum systems, but it is too late now. Within a year, the Joneses are altering the layout of the house, making electrical adjustments, and changing all sorts of things that seemed like a good idea on paper. With a house, you don't really know what you want until you live in it. It is the same with software. The above is a Waterfall method: you draw what you want – thinking you've thought of everything – and then hand it over to be built. Then as soon as you start using it, you find all sorts of other things you wish you had.

If you were to build a house in the Lean-Agile method, the Joneses would move in as soon as the living room was built but before it was decorated. Then they would try out all the locations of where the television might go, and the build team would wire it in appropriately. Then they would add the integrated vacuum system, build a kitchen, then the bedrooms, the home office, the playroom, and the home gym. The Joneses would be constantly feeding to the build team what they want and testing how they will use it before the build is considered complete. And the build team wouldn't move on to another project until the Joneses are happy and settled in their new home.

A lean startup can focus on building the house one room at a time and getting that room right before moving on to the next one. If the Joneses never had a house before, they would likely be willing to go on this journey. A living room on its own still seems like a pretty great offering if your previous experience was living in a tent on a field. But imagine if the Joneses already have a four-bedroom house and you are proposing to move them into a new house. In that case, they aren't likely to move in to a living room on its own. A living room isn't a viable alternative to a family living in a four-bedroom house. Similarly, a customer using a really sophisticated software platform is unlikely to accept a massively slimmed down MVP as a reasonable alternative. This is a serious constraint for companies that need to replatform their product stack. Users are unwilling to try out an MVP when they already have a more robust solution in place.

What Is Replatforming?

Replatforming is the exercise all mature software businesses will face at some point: having to rebuild the technology that supports their products. In contrast to building a new product with previously non-existing features, *replatforming* refers to the re-building, re-factoring, and/or wholesale replacement of a set of software applications that already exist. When your "new" software build must support existing customers, you are beholden to rebuild old functionality first before you can layer in truly new features. Let's look at the two dominant software methodologies in the context of replatforming.

I worked on a Waterfall project many years ago. The goal was to take a set of spreadsheet calculations and replace them with a web platform. I met with the development team several times before the platform build began. Six months later, I was shown a web platform that was about to be released to our customers. I didn't love it, but it did the job. We released it to our customers. Most were happy. It was definitely smoother and better looking than the spreadsheet they had before. But here's the thing: people interact differently with a software platform than they do with a spreadsheet, so we learned a lot from watching our customers use the new product. They had lots of feedback on what could be better (as did I), but that didn't matter much because by the time it was released, the development team had already moved on to another Waterfall project.

Until real people are using software, you cannot optimize it. It would require an unreasonable amount of abstract thought to anticipate the ways in which a customer will interact with a piece of software until she is actually doing so. The inefficiency of Waterfall is in the assumption that you know what you want to build. The reality is that you never know all of the details up front. Even if you did know, by the time you built it, they will have changed. Lean-Agile processes give you the ability to test the core of the platform and validate your assumptions early. It gives you the ability to pivot if you find that the way people interact with the software is different from how you thought they would. This works really well when you are building a new product: something customers have never seen before. If it is new, customers have no preconceived notions about how it should work, or what it should do. A new product or a new market is the most appropriate place for Lean-Agile development methodology.

A replatforming exercise, though, is inherently half-Waterfall in nature because a version of the software already exists. You already know the range of features you need to deliver. You cannot go to market with an MVP that has fewer features than your customers are using today. Given how large existing feature sets are at chunky corporates, it means that the concept of an MVP doesn't really work for replatforming. That puts replatforming in conflict with the ethos of Lean-Agile development. In one Lean-Agile replatforming project, we compiled a list of the features currently available across all of our existing platforms. There were over a thousand product features in total. We would need to replace these for our customers to migrate them from the old platforms to a new one. This would enable us to decommission the old platforms and make our technology stack more efficient.

The list of features, though, represented over five years' worth of work for twelve development teams, or more than a hundred engineers working concurrently. The sheer amount of functionality the company had built up over its operational history was astounding. Five years of development work to re-build what already existed left us standing still in the market.

We had little or no capacity to add new features or products and test them. We were handcuffed to a roadmap predefined by the customer upgrade schedule. We called it an "Agile" project, but it was more akin to Waterfall development than Agile.

One hundred percent of development capacity was spent on rebuilding existing features, and we did not have the freedom to respond to change. The only value-add open to us was to build features in a new way that made them more modern and better able to stand the test of time. You can reimagine the old stuff, but it still has to achieve the business outcome of the previous product. This, to a large degree, acts like a ball and chain to your product roadmap. When you work at a chunky corporate, eighty percent to ninety-five percent of the time you are focused on making better what the company already built. The longer your company has been operating and building software, the more you'll have to rebuild or replace. Equally, the more fragmented your existing product stack, and organization, the longer it will take and the harder it will be.

Product Fragmentation

Fragmentation refers to the level to which a product stack has, or, more precisely, doesn't have, a strong technical backbone of consistency built around a common value proposition. In a fragmented product stack, there are different software products with overlapping features and random product features that don't fit neatly into the technology stack or product value proposition. High levels of product fragmentation require replatforming. As products fragment, their product stacks diverge. Divergence leads to replication, which is inefficient to maintain.
A fragmented product stack is like a disorganized closet. There are lots of little boxes with random contents without any overarching logic behind which contents are in which box. Like the closet, a fragmented product