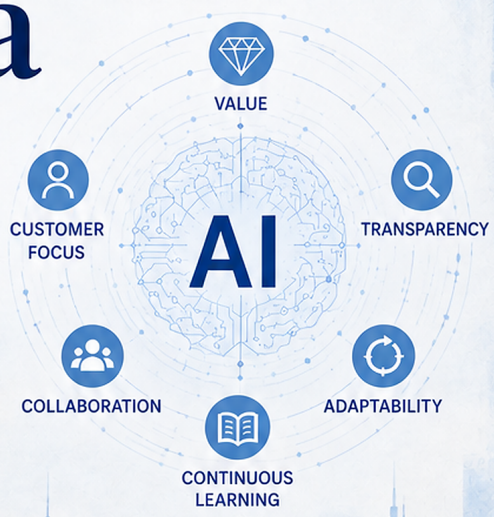
AI GENERATED
EU AI ACT
TRANSPARENCY

Product Development in AI Era

BUILDING ADAPTIVE HARDWARE PRODUCTS IN THE AGE OF AI TRANSFORMATION

Maarit Laanti, Ph.D
WikiAgile



MobAI

MOBILE | EMBEDDED | ON-DEVICE AI

EXPLORE > EXPERIMENT > DELIVER > LEARN > EVOLVE

AGILE PRINCIPLES. MODERN TECHNOLOGIES. INTELLIGENT FUTURE.

Product Development in AI Era

Building Adaptive Hardware Products in the Age of AI Transformation

Front Matter + Part I preview

Maarit Laanti, Ph.D.
WikiAgile

A Note on the Use of AI

Portions of this book were developed with the assistance of generative artificial intelligence tools, used for research support, drafting assistance, and editorial refinement. The cover image was generated using AI image-generation tools. All AI-assisted content — text and image — has undergone substantive human review, fact-checking, and editing by the author, who holds full editorial responsibility for the final text and cover.

This disclosure is provided in the interest of transparency, in the spirit of Regulation (EU) 2024/1689 (the EU Artificial Intelligence Act).

Table of Contents

Foreword

[Foreword text to be supplied.]

Preface

Why This Book, Why Now

I have been thinking about writing this book for a long time — ever since I began collaborating with Joe Justice in 2021, when we started co-creating MobAI® and CadenceFree®. But it was the spring of 2026, standing on stage at LPPDE Europe in Helsinki, that finally made up my mind. The audience — engineers, product leaders, transformation coaches from across Europe — leaned forward. The energy in the room was unmistakable. After the keynote, the conversations confirmed what I had been sensing for years: the field was ready, the need was urgent, and the book had to be written.

This is that book.

Three Forces Converging at Once

We are living through a technological transition of the kind that economic historian Carlota Perez has described as a great surge — a moment when a cluster of radical innovations reshapes not just industries but the physical fabric of society itself.[1] The steam engine gave us the railway and reorganized geography. The internal combustion engine gave us the automobile and reorganized the city. Los Angeles was designed around the car. Helsinki around the horse-drawn carriage. Stockholm's old town is for pedestrians. When autonomous vehicles become mainstream, we will no longer need to dedicate a third of our urban land to parking. That space will return to people. The question is: what kind of future do we want to build with it?

This is not a distant thought experiment. It is unfolding right now — driven by three forces converging simultaneously.

The first force is artificial intelligence. GenAI and large language models have already transformed software development beyond recognition. But AI is not only a software phenomenon. AI is now a hardware designer. It runs simulations, generates test scenarios, accelerates regulatory documentation, and drives robotics at a speed and scale that was unimaginable five years ago. When you look at what Tesla and SpaceX have built — extreme vision combined with extremely fast iteration cycles — the contrast with traditional product development is not incremental. It is the difference between night and day.

The second force is agile hardware development. The insight that agile thinking is not limited to software has been gaining ground for two decades. I have witnessed it firsthand: at Nokia, where we led what was, at the time, the largest agile transformation in the world; at Wärtsilä, where we moved from milestone-driven to genuinely iterative hardware development; and in healthcare AI,

[1]Perez, C. (2002). Technological Revolutions and Financial Capital: The Dynamics of Bubbles and Golden Ages. Cheltenham, UK: Edward Elgar Publishing. ISBN: 9781843763314.

where the stakes of getting it right are measured in human lives.[2345] The old way of dividing organizations into hardware departments and software departments, verifying quality only at the end of a long design cycle, is over. The question is no longer whether hardware development can be agile. The question is how fast your organization can make the shift.

The third force is regulatory and planetary. The European Union's Circular Economy Action Plan, the Corporate Sustainability Reporting Directive, and the EU AI Act are not optional add-ons at the margins of business strategy. They are reshaping what it means to build and sell a physical product.[6] Up to 80% of a product's environmental impact is determined at the design phase. The Netherlands has already committed to reducing resource consumption by 50% by 2030. Companies that understand this early are repositioning — from selling products to providing products as a service, retaining ownership and responsibility across the entire lifecycle. Research by Sitra, Finland's innovation fund, shows that circular economy business models can deliver up to seven times the revenue of traditional models, while dramatically increasing resilience.[7] This is not a cost. It is a competitive advantage.

These three forces — AI, agile hardware, and the regulatory and sustainability wave — are not arriving one at a time. They are arriving together. Organizations that treat them as separate initiatives will be overwhelmed. Organizations that understand them as a single convergence will be the ones that lead.

A Note on Hope

I want to say something directly, especially to the younger engineers and product leaders reading this.

Do not surrender to the scale of the challenges we face — biodiversity loss, climate change, resource depletion. In the 1980s, people said that China's industrialization was impossible, because with the technology of the day, it would have been. But China did not try to replicate the old technology. It

---

[2]Laanti, M. (2024). Accelerating Product Development with Agile Practices in Hardware Design. WikiAgile Whitepaper, January 2024. Available at wikiagile.com.

[3]Laanti, M. (2024). Business Improvement with Agile Hardware Development. WikiAgile Whitepaper, September 2024. Available at wikiagile.com.

[4]Laanti, M., & Justice, J. (2025). Cadence Free™ Product Development using MobAI™ as a Core — Superfast and Truly Agile. WikiAgile Whitepaper, March 2025. Available at wikiagile.com.

[5]Laanti, M. (2016). Piloting Lean-Agile Hardware Development. ResearchGate, May 2016. DOI: 10.13140/RG.2.1.5016.2805.

[6]European Commission (2020). A New Circular Economy Action Plan: For a Cleaner and More Competitive Europe. COM/2020/98 final. Brussels: European Commission.

[7]Sitra (2022). Sustainable Growth with Circular Economy Business Models — Playbook for Businesses. Helsinki: Finnish Innovation Fund Sitra.

leapfrogged into new ones.[8] The future belongs to entirely new kinds of products and solutions — and this is precisely what is most exciting about companies like Tesla and SpaceX: they combine relentless vision with rapid iteration, and that combination produces innovation at a speed that once seemed impossible.

Wärtsilä's latest engine, which generates electricity directly from hydrogen, is an example of what visionary thinking makes possible. AI is not our enemy in this effort. AI is one of the most powerful tools we have ever had for solving the problems we have created. The best future is a sustainable one — where companies offer products as services, maximize resource efficiency, and take responsibility for the full lifecycle of what they build. That future is achievable. This book is about how to build the organizational capability to get there.

Who This Book Is For

This book is written for leaders of established product development organizations — engineering leaders, product managers, and transformation coaches who are responsible for how a large, complex organization builds physical products. It is for the people in traditional companies, including the suppliers now delivering to companies like Tesla, who know that transformation is necessary but are wrestling with how to do it at scale.

It is not a manual for an individual engineer seeking techniques to improve their own workflow. It is not a startup playbook — though startups may find ideas here worth borrowing. It is not a theoretical framework for academic analysis. And it is decidedly not a recipe for a quick agile fix.

This book is for organizations with something built — and therefore something to lose, and something enormous to gain.

How to Read This Book

The book is organized in three parts. Part I establishes the strategic imperative — why hardware organizations must change now, and what the AI era fundamentally changes about the nature of competition. Part II is the AI transformation itself: iteration and modularity as the foundation, the new realities of Hardware- and Software-in-the-Loop testing, what AI can already do, the human role in AI-native development, and the feedback-loop thinking that ties the whole operating model together. Part III is where it leads — the MobAI® and Cadence-Free® approaches we have developed

[8]The concept of technological leapfrogging — bypassing intermediate stages of development to adopt advanced technologies directly — has been widely documented in China's development trajectory. See: Fu, X., & Gong, Y. (2011). Indigenous and Foreign Innovation Efforts and Drivers of Technological Upgrading: Evidence from China. World Development, 39(7), 1213–1225; also Mathews, J.A. (2002). Competitive Advantages of the Latecomer Firm: A Resource-Based Account of Industrial Catch-Up Strategies. Asia Pacific Journal of Management, 19(4), 467–488.

at WikiAgile, and the practical work of piloting, scaling, and leading this transformation inside a real organization.

You can read this book cover to cover. You can also navigate by role: if you are a senior leader focused on strategy, start with Part I and the closing chapters of Part III on piloting, scaling, and leadership. If you are an engineering leader or product manager working on day-to-day development practice, Part II will be most immediately actionable. If you are a transformation coach, you will find the whole arc — from strategic imperative through the transformation itself to where it leads — most useful when taken as a whole.

You can also treat the book as a transformation roadmap: each chapter ends with questions for reflection and action — prompts to take the ideas off the page and into your organization.

The transformation that hardware product development must now undergo is the same transformation that software development went through when agile methods emerged — and then again when cloud and continuous delivery arrived. Those transitions took years and were painful and difficult. They were also, for the organizations that navigated them successfully, the source of extraordinary competitive advantage.

The next transition is here. Let's build it together.

Maarit Laanti, Ph.D.
SAFe Fellow · WikiAgile
Helsinki, 2026

About the Author

Maarit Laanti, Ph.D., has spent more than two decades helping large organizations build the conditions in which truly great products become possible. She spent twenty years at Nokia — most of them as a hands-on product developer — at an extraordinary time: a young, driven company with a genuine mission to connect the world. It was there that her foundational work in large-scale agility took shape.

Between 2000 and 2013, she made three contributions that shaped the field. She invented the Boulders, Rocks, and Pebbles fractal information model for scaling agile backlogs. She is also behind the Lean-Agile Financial Planning model that became the basis for SAFe Lean Budgets.[9] And in 2009, she co-designed and co-delivered the S30/S40 Learning Journey with Dean Leffingwell — published as Leading SAFe. Her doctoral research at the University of Oulu, completed in 2013, formalized these insights and has since been cited over 800 times, placing it among the top 10% most-referenced works in the field.[10]

In 2013, she co-founded Nitor Delta, which brought SAFe to the Nordic countries and trained over 10,000 people — more than 3,000 of whom were personally trained by Maarit. Over twelve years of independent consulting across software, ICT, marine technology, financial services, and manufacturing, she developed her own Agile Hardware training programs, which she continues to evolve and deliver internationally.

In 2021, she began collaborating with Joe Justice to develop MobAI® and CadenceFree® — practices for AI-era product development now deployed in real organizations. She is a SAFe Fellow (2020), founder of WikiAgile (2023), and author of three whitepapers on agile hardware and cadence-free development.[11] In 2024, she co-authored Yritysketteryyden käsikirja (The Enterprise Agility Handbook).[12] Tivi has named her among Finland's 100 most influential IT figures five years running, and she is included on the LIA100 — Lean In Agile's recognition of 100 women making significant contributions to lean and agile practices across the globe.

---

[9]Sirkiä, R., & Laanti, M. (2013). Lean-Agile Financial Planning with SAFe. Original Whitepaper, Scaled Agile Framework. Also published as: Sirkiä, R., & Laanti, M. (2015). Adaptive Finance & Control: Combining Lean, Agile and Beyond Budgeting. HICSS 2015.

[10]Laanti, M. (2013). Agile Methods in Large-Scale Software Development Organizations. Dissertation. University of Oulu. ISBN: 978-952-62-0034-7.

[11]Laanti, M. (2024). Accelerating Product Development with Agile Practices in Hardware Design; Business Improvement with Agile Hardware Development. WikiAgile Whitepapers. Laanti, M., & Justice, J. (2025). Cadence Free™ Product Development using MobAI™ as a Core. WikiAgile Whitepaper. Available at wikiagile.com.

[12]Laanti, M., & Backman, N. (2024). Yritysketteryyden käsikirja (The Enterprise Agility Handbook). Finland.

She lives in Helsinki, Finland, and sails the Finnish Archipelago with her Swan 36.

Connect: maarit@wikiagile.com · linkedin.com/in/mlaanti · wikiagile.com

PART I

The Strategic Imperative — Why Hardware Must Change Now

Chapter 1. The Dawn of a New Era

What if the way your organization develops products is already obsolete — and the companies proving it have been doing so in plain sight for over a decade?

That question is not rhetorical. It is the uncomfortable reality that leaders in automotive, industrial, medical, defense, and consumer hardware are waking up to. A new model of product development has emerged — not in a research lab or a consulting report, but in the factories and launch pads of companies that decided to operate differently. The foundations of agile hardware development have been built over more than a decade of real-world practice. What has changed is the stakes: artificial intelligence has arrived as a force multiplier that rewards organizations with fast iteration cycles and punishes those without them. Agile hardware is no longer a competitive advantage. It is the prerequisite for surviving the AI era. The results are measurable, the gap is widening, and the window for established organizations to respond is not unlimited.

This chapter tells the story of how that new era arrived — and why it is impossible to ignore.

A Problem I Saw Coming — in 2010

The story, for me, begins not with Tesla but with Nokia.

In 2010, Nokia's software organization was already running at scale with iterative, agile development. We had established cadences, backlogs, and cross-functional collaboration that made software move fast. And then we hit a wall. A two-week delay in a hardware prototype did not produce a two-week delay in the software schedule. It produced a ten-week delay — because every software team that depended on that prototype had to stop, replan, and restart. The feedback loop between hardware and software was so tightly coupled that a small hardware slip cascaded into a much larger software crisis.

Antti Vasara, then Senior Vice President of Nokia Mobile Phones, drew the obvious conclusion: if we wanted the whole system to move faster, we needed hardware development to become agile too.

We assembled a team of around twenty engineers — all with deep hardware backgrounds across different disciplines — and spent roughly nine months workshopping what agile hardware development could even mean. The motivation was not abstract. We were trying to rescue product development competitiveness: hardware engineering in China and India was becoming dramatically cheaper, and the only way to stay relevant was to become faster and smarter, not just less expensive. I brought in Preston Smith, one of the earliest and most rigorous thinkers on flexible hardware development,[13] to run a workshop with the team. Smith's core argument was compelling:

[13]Smith, P.G. (2007). Flexible Product Development: Building Agility for Changing Markets. Jossey-Bass. ISBN: 9780787995843. Revised edition (2018): Smith, P.G., Doiron, L., & Farnbach, J.S. Flexible Product Development: Agile Hardware Development to Liberate Innovation. ISBN: 9780578401973.

flexibility in hardware development does not just make life easier for engineers — it delivers better return on investment. A product that reaches market six months earlier captures six additional months of revenue at full margin. The hardware engineers were skeptical at first. Some were quietly hoping to prove the idea wrong. Results came — but changing how people work is never as simple as presenting a good argument. I underestimated the need for sustained coaching and change management. That lesson shaped everything I have done in the field since. When Nokia Mobile Phones ultimately wound down its operations in Copenhagen, the hardware expertise that had grown through this work did not disappear — it dispersed into the Danish hardware industry, where it continued to take root.

I later wrote about this pilot in a research paper.[14] The central insight it confirmed was this: the problems caused by waterfall processes are the same in hardware as in software — fixed plans that resist change, feedback that arrives too late, decisions made on assumptions that were never tested. But in hardware, those problems are harder to solve because there are fewer degrees of freedom. You cannot define iteration boundaries arbitrarily. They must align with the natural points in development where feedback is actually available — where a prototype can be tested, where a module can be evaluated, where a design decision can be validated against reality.

Vision Against the Odds — From WIKISPEED to Tesla and SpaceX

Nokia faced what seemed like an impossible challenge: make a large, established hardware organization move faster in a world where costs were rising, and competition was accelerating. Tesla and SpaceX faced challenges that looked even more impossible — and thrived despite them.

When SpaceX set out to build reusable rockets, the established space industry considered it a fantasy. Rockets were expendable by design — the economics of recovery seemed prohibitive. On December 21, 2015, a Falcon 9 delivered eleven satellites into orbit, and its first stage then returned and landed vertically at Cape Canaveral. Musk called it 'a fundamental step change compared to any other rocket that's ever flown.'[15] When SpaceX listed on the Nasdaq in June 2026 — in what became the largest IPO in history, raising $75 billion at a valuation of approximately $2 trillion — Musk said at the opening bell: “I gave SpaceX a less than 10 percent chance of succeeding at all.” The rocket company that was supposed to fail now launches more rockets annually than all other providers combined. The operating principle behind this — and behind Tesla — is the relentless removal of everything unnecessary. Not just from the product, but from the process: eliminate every component, every step, every requirement that does not directly serve the goal. Remove until you

[14]Laanti, M. (2016). Piloting Lean-Agile Hardware Development. Proceedings of XP2016. DOI: 10.13140/RG.2.1.5016.2805.

[15]SpaceX landed the first stage of its Falcon 9 rocket vertically on December 21, 2015, at Cape Canaveral, Florida — the first time an orbital-class rocket booster had been returned intact after delivering a payload to orbit. Source: Space.com (2015); Wikipedia, Falcon 9 reusable launch system development program.

have to add something back. That discipline, applied consistently and at speed, is what produces the results that traditional organizations describe as impossible.

Joe Justice had been building something that made the agile hardware world pay attention — and the timeline matters, because it is sometimes misunderstood. Joe founded WIKISPEED in 2006, before Tesla had produced a single mass-market vehicle. Starting with a team of volunteers assembled through social networking — no automotive funding, no traditional expertise, no factory — he applied the principles of Scrum and object-oriented design directly to physical hardware. The result was the SGT01: an ultralight car achieving over 100 miles per gallon, designed with a fully modular architecture of eight snap-in/snap-out modules built around standardized interfaces. The engine could be swapped in roughly the same time it takes to change a tire. The car was road-legal and safety-certified in both the United States and Germany, achieving a five-star crash-equivalency rating on front, side, and rear impact. The entire first prototype was developed in 12 weeks — a world record. Joe's teams set four world records in total.

In 2010, WIKISPEED entered the Progressive Automotive X Prize — a $10 million international competition for ultra-efficient vehicles — and beat Tesla in that contest. The result was not just a trophy. It was a proof of concept visible enough that Tesla took notice: Joe began consulting for Tesla that same year, collaborating on the early development and deployment of their agile program. He was not learning from Tesla. He was bringing agile hardware thinking to Tesla. This distinction matters. Joe Justice is not someone who reverse-engineered Elon Musk's methods and packaged them for sale. He is a pioneer who helped introduce agile thinking into the company in the first place — and who subsequently documented and taught those methods more systematically than anyone else. Throughout the 2010s, he consulted for Toyota, BMW, Volkswagen Group, Nissan, and all three of the world's largest defense contractors, applying the WIKISPEED methodology globally. Paolo Sammicheli's book on Scrum for hardware, published in 2018 with a foreword by Joe,[16] documents WIKISPEED as the founding proof of concept for the entire field — not as a case study of Tesla, but as the original demonstration that agile hardware development works at scale.

I invited Joe to keynote at Scan Agile, a conference our team was organizing. Then COVID arrived. The conference was cancelled. When the pandemic shut down global travel, the demand for in-person training and keynote speaking evaporated overnight. Joe found himself without the international work that had sustained his career.

I suggested he would go back to Tesla — this time not as a consultant but as an employee. In 2020, from Tesla's Fremont headquarters, Joe operated Agile@Tesla, working physically on the factory floor alongside robots and assembly teams, building thousands of Tesla parts while simultaneously running the agile programme. Musk has consistently said he does not want a monopoly on these methods and does not patent Tesla's ways of working, because he believes the world needs to learn

[16]Sammicheli, P. (2018). Scrum for Hardware. Foreword by Joe Justice. ISBN: 9781983373312. Justice, J. (2013). Joe Justice Tells the Story of Team WIKISPEED. Sunbourg.

from them. When Joe later published his book[17] and began teaching the JoeDX methodology — distilling the operating model of the Musk companies into twelve reinforcing practices — Musk retweeted his talk, amplifying the message that everyone in a company is a worker and contributor, with no two-class system of managers and employees. Joe is now a millionaire many times over — a very different situation from that COVID spring. But more importantly, the model he documented has become a reference point for every hardware organization trying to understand what is now actually possible.

I have been fortunate to take Joe's course — he taught me as a friend, at no charge — and we have had the opportunity to work with clients together. What he brings from inside Tesla's operating model is unlike anything available from the outside.

What Tesla and the Musk Companies Actually Did — and Why It Matters

It is tempting to dismiss Tesla as a special case. A startup, unburdened by legacy processes. A founder willing to ignore industry norms. A product — the electric vehicle — that happened to be well-timed. All of this is partly true. But it misses the deeper point.

Walter Isaacson, who shadowed Musk for two years across Tesla and SpaceX, describes him not primarily as an inventor but as a perfecter — a specialist in what Musk calls 'building the machine that makes the machine': honing products and manufacturing systems down to their essentials and producing them quickly at drastically reduced cost.[18] Musk's stated time horizon is a thousand years. When you think in centuries rather than quarters, the decisions you make about organizational structure, process, and technology look entirely different. The Musk companies — Tesla, SpaceX, Neuralink, The Boring Company — share a common operating logic: start from a bold, long-range vision; remove everything from the process that does not serve it; iterate at maximum speed; use AI not as an add-on but as a pervasive part of how work gets done, from design and simulation to production management and performance tracking. The Justice Board — named by Joe Justice — is one visible expression of this: a real-time performance dashboard that replaces traditional management layers by allowing small teams of engineers to track and improve key metrics autonomously. AI is not a tool at the margin of Tesla's operations. It runs through the entire system.

A note on Musk as a person: not everyone admires his political positions or his public behavior, and those concerns are legitimate. His role in DOGE and his broader political interventions are genuinely controversial. This book is not an endorsement of any of that. What this book observes is his operating model — because understanding it is strategically important regardless of one's views on the man. The same was true at the beginning of the agile transformation in software: the question was not whether you admired Amazon or Apple, but whether you could afford to ignore what they

---

[17]Justice, J., & Klein, P. (2021). Scrum Master: The Agile Training Seminar for Business Performance. ISBN: 9798704954057. Available at: leanpub.com/scrummasterEN.

[18]Isaacson, W. (2023). Elon Musk. Simon & Schuster. ISBN: 9781982181284.

were doing. Not every company became Amazon. But the companies that refused to adapt paid a price for that refusal.

> ***By comparison:*** *When General Motors applied agile methods to the Hummer EV, they developed the vehicle in approximately two years — compared to the industry-standard five. The key enablers were Hardware-in-the-Loop (HiL) and Software-in-the-Loop (SiL) virtual testing, a modular platform architecture, and cross-functional mission-oriented squads. A GM engineer described it: virtual testing allowed them to drastically reduce the number of 3D-printed prototype parts, which can cost 10 to 20 times more than a mass-produced equivalent. GM is not a startup. It is one of the world's most traditional automotive manufacturers. If GM can cut development time from five years to two, the excuse that 'this only works for Tesla' no longer holds.*

Why Every Hardware Industry Is Next

The automotive industry is the most visible example, but the pattern is spreading to every sector where physical products are developed.

In defense, the shift is already advancing — and unusually transparent. The development of military drones in Ukraine has compressed design-to-deployment cycles from years to weeks, iterating in direct response to real-world operational feedback. This is agile hardware development under the most extreme conditions imaginable. The lesson is not that all organizations should operate under wartime urgency. The lesson is that when the conditions demand it, fast hardware iteration is possible. The physics never prevented it. It is a matter of the mindset.

In medical devices, the economic case is stark. Development costs are already enormous, and even a modest reduction in cycle time yields a disproportionate improvement in return on investment — more market time at full margin, earlier recovery of development costs, and faster response to clinical feedback. COVID demonstrated something else: when urgency is real, and resources are aligned, development timelines that seemed fixed are not. Vaccines that would normally take a decade to develop and validate were developed and validated in under a year. The lesson is not that normal standards should be abandoned. The lesson is that the assumption of fixed, immovable timelines deserves serious challenge.

The pattern repeats across industries: marine technology, consumer electronics, industrial manufacturing, robotics. The organizations that learned to iterate fast gained ground; those that did not lost it.

The Manufacturing-R&D Gap: Industry 4.0 Without the Development to Match

There is a paradox at the heart of most hardware organizations today. Manufacturing has transformed. Industry 4.0 and Industry 5.0 have delivered flexible, automated, data-rich production

environments that can accommodate variation and respond to change at a speed unimaginable twenty years ago. The factory floor is ready to iterate.

R&D is not.

Product development processes in most established hardware organizations still operate on assumptions from ten, fifteen, or twenty years ago: long planning horizons, milestone-gated phases, sequential handoffs between engineering disciplines, and a fundamental orientation toward avoiding change rather than incorporating it. The manufacturing system has evolved. The development system that feeds it has not kept pace. The result is a bottleneck that sits not on the factory floor but in the R&D process — and no amount of investment in flexible manufacturing can compensate for a development process that delivers designs too slowly and too rigidly to take advantage of it.

This is the gap that agile hardware development exists to close.

AI as Amplifier — Not Substitute

Artificial intelligence is now reshaping what is possible in hardware development. AI can run simulations in hours that would previously require weeks of physical testing. It can generate design variants, flag failure modes, accelerate regulatory documentation, optimize manufacturing processes, and enable teams to iterate at a speed that was previously impossible. The change is real and accelerating.[19]

But AI does not replace the need for an agile operating model. It amplifies whatever model it is applied to. An AI tool applied to a waterfall process makes the waterfall faster — and therefore more efficient at producing the wrong output with greater speed and confidence. The same tool, applied to an agile operating model, enables a step change in learning speed. The organizations that will benefit most from AI are not necessarily those with the most advanced AI tools. They are the ones with the most adaptive ways of working — organizations designed to form hypotheses, test them against reality, and incorporate what they discover into the next iteration.[20]

This is why agile hardware is not a nice-to-have in the AI era. It is the operating condition that determines whether AI investment produces a competitive advantage or merely accelerates existing dysfunction.

---

[19]GM Hummer EV development: CNN Business (2020). 'GM is making the Hummer EV in record time. Here's how.' The vehicle was developed in approximately two years versus the industry-standard five years, using Hardware-in-the-Loop (HiL) and Software-in-the-Loop (SiL) virtual testing, modular platform design, and cross-functional mission-oriented squads.

[20]Laanti, M. (2024). Accelerating Product Development with Agile Practices in Hardware Design. WikiAgile Whitepaper. Available at wikiagile.com.

Reflection Questions

- How long is your current hardware development cycle — from concept to customer? How does that compare to your fastest-moving competitor?
- Where in your process does a hardware delay amplify downstream — into software, testing, supply chain, or market timing?
- How flexible is your manufacturing system compared to your R&D process? Where is the real bottleneck?
- If AI amplifies whatever operating model you have — what does that mean for your organization right now?

Chapter 2. Architecture Is Strategy — Why How You Build Determines How You Scale

Figure 2.1. A hardware-first roadmap pattern still common across product companies today: each tier — high, mid, low — developed as its own parallel track, software following hardware, with release milestones marked at the end of each combination.

Chapter 1 closed on a warning: AI does not replace your operating model; it amplifies it. A fast, adaptive organization becomes faster and more adaptive. A slow, siloed one becomes a slow, siloed organization moving at greater speed toward the wrong outcome. That warning raises an obvious question. What determines whether an organization is fast and adaptive in the first place? The answer is not culture, and it is not process maturity, at least not directly. It is architecture. How a product is built — how tightly its parts depend on one another — determines how much coordination the organization needs to build it, and how much coordination it needs determines how the organization can scale. Get the architecture wrong, and no amount of agile ceremony will make the organization fast. Get it right, and speed becomes structural rather than aspirational.

Three Eras, Three Measures of Success

It helps to see the shift in one frame. Three operating eras can be compared across three dimensions: how each one measures success, the rhythm at which work happens, and where the bottleneck sits.

	Waterfall	**Agile**	**AI Era**
Success measure	Output	Outcome	Impact
Work rhythm	Linear	Iterative	Conversational
Bottleneck	Delivery	Capacity	Leadership

Waterfall-era organizations measured success by output: did we finish what we planned to build? Agile organizations shifted the measure to outcome: did the customer get value from what we built? The AI era shifts it again, to impact: is the value that AI helps us generate actually being steered toward something that matters — the business, the people who depend on it, the wider system it sits inside? Impact is not a replacement for outcome; it is outcome held to a higher, more deliberate standard. And the mechanism that gets an organization from outcome to impact is the same mechanism this book keeps returning to: the speed at which the organization learns. An organization that can form a hypothesis, test it, and fold what it learns into the next iteration is best positioned to steer AI-generated value in a direction worth going. Learning speed is how impact gets made, not a separate virtue alongside it.

Where the Bottleneck Moves

The bottleneck row in that table is worth sitting with, because most engineering leaders have lived through the shift it describes without necessarily naming it.

In the waterfall era, the bottleneck was delivery. The deadline was the finish line everyone ran toward, and the closer the project came to it, the more effort was thrown at it. Anyone who has worked in traditional product development remembers the pattern: long nights in the weeks before a milestone, because the project was “almost done,” and “almost done” meant everyone stayed until it was actually done. The constraint was simply getting the thing finished.

Agile development changed what the constraint was. Once teams began working in increments that were tested and potentially shippable at every step, a different realization set in: not everything wanted can actually be built. Content has to be chosen. Someone has to decide what is genuinely important and what can wait. The bottleneck moved from finishing the work to deciding which work is worth finishing — in other words, from delivery to capacity, because capacity is what forces the choice.

AI changes the constraint again, and in a way that raises the stakes rather than lowering them. AI accelerates doing so substantially, allowing projects once buried — shelved years ago because they were too expensive or too labor-intensive to justify — to be credibly reopened. That sounds like good news, and in one sense it is. But it means the question a large organization must answer becomes sharper, not softer: what is actually important? Where does the value genuinely lie? When almost anything becomes possible, leadership — clarity of vision, quality of decisions, and ethics at the speed of AI — is what stands between an organization doing what it can and an organization doing what it should.

Architecture as the Hidden Variable

This is where architecture enters the picture, because it determines how much of that leadership burden any given part of the organization has to carry. Consider two ends of a spectrum. One end looks like the classic scaled-agile pattern: teams aligned around a shared plan, synchronized cadences, coordinated releases — an approach built to create alignment across a large, interdependent system. The other end looks like the way Tesla has been described operating: no imposed alignment at all, just clear goals handed to teams who are trusted to self-manage their way there.

Neither end is right by default. The real question an organization has to answer is how much alignment it actually needs, and how much scaling up — or scaling down into independent, self-contained units — it can afford. And that question is answered by architecture, not by org-chart preference. A tightly coupled system, where one module cannot change without touching several others, forces coordination whether anyone wants it or not. A genuinely modular system, where interfaces are clean and stable, allows independent teams to move without waiting for each other. Architecture, in other words, is the variable that decides which model of scaling is even available to an organization. That is why it belongs in a chapter of its own, ahead of value streams, ahead of

iteration cadence, ahead of any specific framework: everything else in Part II is downstream of this choice.

When Architecture Forces Scale: The Nokia Mobile Phones Case

Nokia Mobile Phones offers a clear illustration of what happens when architecture leaves no choice. For years, software in a given phone line was effectively a single product tied to a single processor platform. Every feature, every driver, every fix for one model lived in the same tightly bound codebase as every other feature for that model. There was no meaningful way to split the work into independent streams because the architecture itself did not allow it: change one part of the system, and the ripple would reach the rest. The organizational response was, necessarily, to scale up coordination — large, synchronized planning events, shared backlogs, visible cross-team dependencies, the kind of big-room planning that later became formalized in scaled agile practice. This was not a preference for heavyweight process. It was what the architecture demanded.

Modularity as a Deliberate Choice: The Spotify Case

Spotify's engineering organization offers the counter-case, and it is worth being precise about what actually changed there. Spotify's desktop client began as a monolith, which was manageable while the product was simple. As the company added features and multiplied product teams, the monolithic structure began to limit both delivery speed and team autonomy: new code could only reach production once every team's changes were stable and every interaction between teams had been tested.[21]

Spotify's response was to define standards for a modular, API-enabled architecture. The effect was what its engineers came to call a “limited blast radius”: one squad's experiment could no longer take down the whole system, so squads could release independently, experiment freely, and learn faster — without waiting on a central release train.[22]

The lesson is not that Spotify made the right architectural choice and Nokia Mobile Phones made the wrong one. The lesson is that, in both cases, architecture determined the organizational model available to them. Nokia's coupled architecture required scaling up alignment. Spotify's modular architecture allowed scaling down into independent, autonomous units. Neither organization chose its coordination model directly — they inherited it from a prior decision about how tightly their systems were built.

---

[21]Kniberg, H. (2014). Spotify Engineering Culture, Part 1. Spotify Labs / Spotify Engineering. Published March 27, 2014.

[22]Kniberg, H. (2014). Spotify Engineering Culture, Part 2. Spotify Labs / Spotify Engineering. Published September 20, 2014. See also Baiyere, A., Ross, J.W., & Sebastian, I.M. (2017). Designing for Digital — Lessons from Spotify. MIT CISR.

The Optimization Problem: How Modular Is Modular Enough

None of this should be read as an argument that maximum modularity is always the goal. It is not. Every interface between modules has a cost: coordination overhead, integration risk, sometimes a performance penalty paid at the boundary. Every shared, tightly coupled component also has a cost: it blocks independent progress and concentrates risk. Architecture is not a binary choice between monolith and modularity — it is an optimization problem. Some things genuinely benefit from being decoupled. Others genuinely benefit from staying together. The skill is in telling which is which, for a given product, at a given point in its life.

Tesla is a useful reference point here precisely because its modularity is deliberate rather than extreme, built on an operating model of small, cross-functional teams working directly against clear goals rather than a heavyweight coordination structure.[23]

The car seat is a good example: seat-heating control lives in software that can operate across different hardware configurations, but the software also has to recognize gracefully when a given seat variant lacks the hardware to support the feature at all. That distinction — what to make vertical and what to make horizontal — follows a clear pattern. Functionality that evolves quickly and depends on hardware and software working together, such as camera image quality, benefits from being built as a vertical slice, so the entire capability can be iterated as a single, holistic piece. Components that are relatively stable and can be commoditized, such as a standard engine, benefit from being treated as a platform — a stable horizontal layer — while differentiation happens in the software running on top of it. A battery, by contrast, is currently under heavy innovation, which argues for treating it as a separable, swappable module in its own right. None of this is dogma about modularity; it is a series of specific calls about where coupling helps and where it hurts. That is what makes architecture a strategy rather than technical housekeeping: the decision about how modular to be, made deliberately and differently for each part of the system, determines the coordination the organization will need and the scaling model that follows from it.

Nokia's Later Chapter: Decoupling Through a Middle Layer

Scaled agility rarely stays static. Organizations that first scale up coordination to cope with a coupled architecture tend, over time, to invest in loosening that coupling — because the coordination cost becomes visible and expensive enough to justify the redesign. Nokia's own later evolution followed this path: introducing a middleware layer between hardware and software decoupled their development cycles, which in turn allowed each side to iterate on its own schedule rather than waiting for the other. The direction of travel is the same one Spotify took with its API-enabled architecture: scaled coordination is often a first response to a coupled system, and architectural

[23]Justice, J., & Klein, P. (2021). Scrum Master: The Agile Training Seminar for Business Performance. ISBN: 9798704954057.

investment — not more process — is what eventually allows the organization to scale back down into faster, more independent units.

The Missing Piece: Test Automation and Industrial DevOps

Decoupling the architecture is necessary, but it is not sufficient on its own — and Nokia's own history offers a cautionary case in the other direction. At one point, its S60 software organization tried the opposite experiment: running roughly 100 agile projects in parallel on the same software base. The results were poor. The problem was not the teams or agile as a method; it was that the integration and test infrastructure had not been built to absorb that much simultaneous change. Without automated, continuous integration capable of validating a hundred concurrent streams, parallel agile work simply produced parallel integration chaos.[24]

This is the piece many embedded and hardware-heavy organizations still miss. Consumer software companies took a decade to build the continuous integration and automated test pipelines that make Spotify-style modularity actually pay off; most embedded device manufacturers have not made the equivalent investment. Modularizing the architecture without modernizing test and integration capability simply moves the bottleneck — from tightly coupled code to an overwhelmed test cycle. The practice that closes this gap for cyber-physical systems by combining software, hardware, and firmware into a single continuously verified pipeline is now documented as Industrial DevOps.[25]

The implication for an architecture decision is direct: how modular a system can safely become is bounded by how fast the organization can integrate and test the pieces back together. Architecture and test capability have to scale together, or the coordination the architecture was supposed to remove simply reappears downstream, in the test cycle, later and more expensively.

The Cost of Getting the Roadmap Wrong

The price of skipping that architectural investment shows up most visibly in the product roadmap — the pattern shown in Figure 2.1. A common pattern in hardware organizations is to develop each product tier — high, mid, and low — as its own parallel project, with hardware and software for each tier built, tested, and released as a bundle. The logic looks reasonable on paper. In practice, any delay in one tier's hardware cascades directly into a delay in that tier's software, and because every tier is a separate project, the organization ends up carrying the complexity of several near-duplicate development efforts at once. The burden becomes most visible in testing, where every combination

---

[24]Laanti, M. (2013). Agile Methods in Large-Scale Software Development Organizations. Dissertation. University of Oulu. ISBN: 978-952-62-0034-7.

[25]Johnson, S., & Yeman, R. (2023). Industrial DevOps: Build Better Systems Faster. Portland, OR: IT Revolution. ISBN: 9781950508792.

of hardware and software variant across every tier has to be validated separately — exactly the integration load that overwhelmed Nokia's hundred parallel projects.

The alternative is to treat hardware as a platform and let software drive differentiation: software features are specified, built, and released independently of hardware releases, and the hardware is designed from the outset to support multiple software versions. This is the same principle at work in the Tesla seat-heating example, generalized into a roadmap discipline — decouple what does not need to move together, and let each side release on its own cadence, backed by the test automation to verify it safely.

AI as the Architect's New Collaborator

If architecture is now a strategic decision rather than a background technical one, it matters that AI is already changing how that decision gets made. Three uses stand out.

The first is optimizing an existing design. Generative design tools can take a current component and search a vast space of geometric variations against a fixed set of constraints. WHILL, a Japanese mobility-scooter manufacturer, used Autodesk's generative design tools to redesign a battery case starting from its existing structure, reducing its weight by roughly 40 percent while meeting the same load requirements.[26]

The second is generating alternative architectures outright. Rather than starting with one design and refining it, an architect can describe requirements and constraints and ask for several distinct high-level architectures in return — each optimized for a different priority, such as performance, scalability, or cost — to compare before committing to one direction.[27]

The third is simulation ahead of commitment. DESCO Corporation, also in Japan, used generative design to redesign an engine control unit, improving engine performance and reducing vehicle weight in the process — work that went on to win an iF Design Award. What generative, simulation-driven design changes is the architect's role itself: less a matter of manually drawing a shape, more a matter of defining the objectives, constraints, and performance criteria and letting a computational process explore the space those constraints allow.[28]

None of this changes the underlying argument of this chapter — it sharpens it. AI does not decide how modular a system should be. It gives architects a faster way to explore the consequences of that decision, which means the organizations that have already learned to make good architectural calls

---

[26]Formlabs (2024). Generative Design 101. Formlabs Blog. WHILL case study using Autodesk Fusion 360 generative design.

[27]AWS Prescriptive Guidance. Generative AI use cases for architecture and design. Amazon Web Services documentation.

[28]DESCO Corporation ECU redesign case, cited in: Generative Design for Performance Enhancement, Weight Reduction — A Review. arXiv:2007.14138.

— and that have the test automation to safely verify what AI proposes — will get the most out of it. AI amplifies whatever operating model — and whatever architecture — it is applied to.

Why Agile Is the Prerequisite — Not the Destination

Put together, these threads point to a single conclusion. Architecture determines coordination need. The need for coordination determines which scaling model — tightly synchronized or genuinely autonomous — is available to an organization. And getting that architectural decision right is not a one-time event: it is an ongoing optimization, revisited as products, teams, and technology evolve, and bounded throughout by how fast the organization can integrate and test what it builds. Agile practices — value streams, modular design, short iteration cycles — are how an organization acts on a good architectural decision once it has been made. They are the prerequisite for speed, not the source of it. The source is architecture itself: the deliberate, strategic choice of what to couple and what to set free.

Reflection Questions

- Where in your product does a change in one component force a change somewhere else it should not need to touch?
- If you mapped your own roadmap the way Figure 2.1 maps a hardware-first tier structure, how many parallel variants would you find — and how much of your testing effort goes into covering the combinations between them?
- Which parts of your product are evolving fast enough to deserve a vertical, holistic slice, and which are stable enough to be treated as a shared platform?
- If you modularized your architecture tomorrow, does your test and integration pipeline have the automation to keep up — or would the bottleneck simply move downstream?
- If AI could generate three alternative architectures for your next product tomorrow, would your organization know how to judge which one is right — or would that decision still be missing an owner?

Chapter 3. Good Business — Why Sustainable Hardware Is More Profitable

Chapter 2 argued that architecture determines what an organization can build efficiently — that the coupling or modularity of a product's design decides how fast and how cheaply it can be built, tested, and changed. This chapter asks the question architecture alone cannot answer: efficiently building what, for whom, and against which competitors? Good architecture is a precondition for good strategy. It is not the same thing. This chapter is about strategy — where the profit actually comes from, who wins when digitalization rewrites an entire industry, and why the regulatory and sustainability wave named in the Preface turns out to be one instance of a larger competitive pattern rather than a constraint bolted on from outside it.

Blue Ocean, Solved Differently

Every product strategy is, at bottom, a bet about where the profit actually is. W. Chan Kim and Renée Mauborgne's Blue Ocean Strategy made the distinction explicit: a red ocean is a market already crowded with competitors fighting over the same customers on the same terms, where every gain comes at a rival's expense, and margins bleed out accordingly. A blue ocean is uncontested market space, created by making the existing basis of competition irrelevant rather than beating it on its own terms — and blue oceans consistently deliver the best profitability, precisely because nobody is yet fighting over them.[29]

Sustainable, resource-efficient products, discussed later in this chapter, are a blue ocean of exactly this kind. The temptation is to treat sustainability as a cost — a compliance tax paid to satisfy regulators. The more useful frame is the opposite one: a genuinely sustainable solution is usually also a genuinely better one, because it solves the underlying problem in a new way that happens to use fewer resources, last longer, and cost less to run. Waste, in a mature industry, is rarely a deliberate design choice; it is simply what survives when nobody has yet been forced to ask whether there is a cheaper, cleverer way to deliver the same value. Regulation forces that question. The organizations that answer it well do not merely comply — they find a blue ocean the rest of their industry has not yet noticed.

This is, at bottom, what a product business is for: solving a real human problem in a new and useful way — useful very often meaning cheaper, because a solution nobody can afford is not much of a solution. Agile hardware development, combined with software-first digitalization, is what makes finding that new way practical at the speed a competitive market now demands. Design Thinking is the discipline that maps which problems are actually worth solving, and sketches the range of ways they might be solved, before committing engineering resources to any one of them.

---

[29]Kim, W. Chan, and Renée Mauborgne. Blue Ocean Strategy: How to Create Uncontested Market Space and Make the Competition Irrelevant. Harvard Business School Press, 2005.

Mapping the problem once, before the build begins, is not enough on its own, because in a service-based, retention-priced business — the model examined later in this chapter — the work is never really finished at launch. This is where a third stage of discovery becomes necessary, one that operates not before delivery but during it: continuously validating, in real use, whether the chosen solution is still the right one as customers, competitors, and constraints keep moving. This third diamond of discovery, following the two diamonds of problem framing and solution design that precede it, is what makes an ongoing, service-based business model actually work.[30] Retention — whether the customer is still there next month — is the honest signal this ongoing diamond is built to read. A product team that stops learning at launch is, in effect, betting that nothing about the problem or the customer will change again. In a market moving at the pace described throughout this book, that is a bet worth almost never taking.

Figure 3.1. The three diamonds of product discovery — problem framing, solution design, and continuous delivery validation — and where each one's learning actually happens.

From Ownership to Access

Digitalization has a well-worn pattern for turning owned products into accessed services, and the pattern has already run its course once, in an industry with no obvious connection to hardware. Music once meant a physical object: a vinyl record, then a cassette, then a CD, each one purchased, owned, and shelved. Digital compression turned the music into a file — the MP3 — that could be copied and shared for free, breaking the old business model before it could build a new one. What actually stabilized the industry afterward was not ownership of files but access to a catalog: streaming, priced as a subscription, profitable because the marginal cost of one more stream is close to zero and because a good recommendation engine keeps the subscriber paying every month rather than once. Nobody under thirty today thinks of a music collection as something to own.

Chapter 1 noted, in passing, that when autonomous vehicles become mainstream, cities will no longer need to dedicate a third of their urban land to parking, because a car that can drive itself does not need to sit, parked, waiting for its owner to need it again. That observation understates the size of the shift coming. If a self-driving car is available on demand, reliably and cheaply, in about the time it currently takes to open a ride-hailing app, the case for owning one privately weakens in almost exactly the way the case for owning a CD weakened once a phone could stream any song, instantly, for a monthly fee. A car sitting in a garage twenty-three hours a day is a worse capital allocation than a shelf of CDs nobody plays anymore; it is simply harder to notice, because the car is bigger and the depreciation is slower. Car manufacturers who see this coming are already repositioning as transportation-service providers rather than vehicle sellers — retaining ownership, running fleets, competing on uptime and ride quality rather than on badge and horsepower.[31]

---

[30]Laanti, Maarit. “PM & PO — The Hunt for the Third Diamond.” WikiAgile presentation, 2026.

[31]See also the Level 5 autonomous-driving business model discussion in the author’s keynote “Accelerating the Digital Revolution: Agile Hardware and Sustainability,” XP 2024 conference.

Owning a car may, within a generation, be a choice people make the way some still choose to buy vinyl: for enthusiasts, for a specific kind of pleasure, not because there is no better way to get the ride.

If the ownership-to-access shift is coming for cars the way it already came for music, the more urgent question for anyone leading product strategy is not whether the shift happens but who wins it. History suggests three mechanisms decide that, and all three reward speed over scale.

Cost Innovation

The first mechanism is cost innovation: entering a market not with a cheaper, worse version of the incumbent's product, but with genuinely new technology that delivers the same or better performance at a substantially lower price. Tesla's own playbook is a repeatable illustration. When Tesla launched the third generation of its Solar Roof, the price of the same system dropped by roughly 40 percent compared with the previous generation, driven by faster installation and simplified manufacturing rather than by cutting corners on the product itself.[32] Chinese manufacturers have industrialized the same logic at national scale: Ming Zeng and Peter Williamson's Dragons at Your Door documents how Chinese firms have repeatedly entered high-value categories — engineering, design, even R&D-intensive products — not by competing on price alone but by using a fundamentally lower cost base to offer specialty capability once reserved for premium players.[33] The lesson for an incumbent is uncomfortable: cost innovation is not a race to the bottom. It is a race to redefine what the entry price for real capability should be, and whoever runs that race first sets the new floor everyone else has to compete against.

Growth Hacking Through Rapid Experimentation

The second mechanism is growth engineered into the product itself rather than bought with a marketing budget. GoPro's rise is the clean hardware example. Rather than buying advertising to describe what its cameras could do, GoPro built a system — branded hashtags, an annual awards program with real prize money, prominent reposting of the best footage across its own channels — that turned every customer who strapped a camera to a helmet into an unpaid demonstration of the product in use. By the time the market matured, GoPro held 97 percent of dollar share and 87 percent of unit share of the U.S. action camera category, built substantially on content its own customers created for free.[34] The Design Thinking discipline introduced earlier in this chapter is what makes this repeatable rather than lucky: running fast cycles to frame which content or referral loop

---

[32]Fred Lambert, “Tesla Solar Roof V3 real quote shows price dropped by 40%,” Electrek, October 27, 2019.

[33]Zeng, Ming, and Peter J. Williamson. Dragons at Your Door: How Chinese Cost Innovation Is Disrupting Global Competition. Harvard Business School Press, 2007.

[34]GoPro, Inc., fourth-quarter 2018 financial results: 97 percent dollar share and 87 percent unit share of the U.S. action camera category.

might spread, testing a cheap version of it, and discarding what does not work is simply the problem-and-solution diamonds run against an audience rather than a product feature.

Network Effects Through Recommendation

The third mechanism is the one that ultimately decides which platform in a given category becomes structurally hard to leave. A recommendation algorithm gets better as more people use it, because it has more listening, viewing, or usage data to learn from — which means the product with the most users today has a compounding advantage in becoming the best product tomorrow. Spotify's Discover Weekly playlist illustrates the mechanism directly: users whose listening is more diverse — precisely the behavior the recommendation algorithm is built to encourage — churn at rates ten to twenty percentage points lower than users who do not engage with it.[35] This is the same mechanism, in software form, that will decide which robotaxi network wins a given city: the network with more rides accumulates more data on traffic, demand, and route efficiency, which improves its service, attracts more riders, and generates more data still. Whoever accumulates that loop first becomes difficult to dislodge afterward — not because of brand loyalty, but because the product genuinely gets better the more people use it.

Put together, these three mechanisms explain why fast iteration cycles are not a nice-to-have in this environment; they are close to the entire competitive question. Cost innovation rewards whoever redefines the price floor first. Growth hacking rewards whoever runs the most experiments against a real audience. Network effects reward whoever accumulates usage data first and never lets go of the lead. All three are races, and in a race, the organization that can complete a learning cycle in weeks beats the one still working in years — regardless of which one started with the better underlying technology.

A Question Twenty Years in the Making

The blue ocean introduced at the start of this chapter is not a new discovery, either — which raises an uncomfortable question of its own. In 2005, at the International Conference on Agile Manufacturing in Ohio, one of the standing questions on the table was the circular economy: how do you build products, and the systems that make them, so that materials keep circulating instead of ending in a landfill?[36] That is worth sitting with. Twenty years before circular economy became a line item in EU legislation, it was already a live question inside the agile manufacturing community — one that assumed manufacturing had to become flexible, responsive, and adaptable to survive.

Twenty years is a long time for a question to sit on the table without a good answer. The uncomfortable truth is that despite two decades of conferences, white papers, and good intentions,

---

[35]Spotify listener-diversity and churn analysis: users with more diverse listening, driven by algorithmic recommendation such as Discover Weekly, churn at rates 10–20 percentage points lower than less-diverse listeners.

[36]International Conference on Agile Manufacturing (ICAM), Ohio, 2005.

the global transition has barely started. As of the most recent measurement, only 8.6 percent of materials used in the global economy are cycled back into use, and the circularity rate in the EU sat at 11.7 percent in 2021 — up a mere 0.2 percentage points since 2017.[37] Talking about circularity has never been the bottleneck. Doing it, at scale, inside real product organizations, has been.

This chapter is about why that is finally changing — not primarily because attitudes have shifted, but because three things are now converging that were not converging in 2005: binding regulation, a business case that has become impossible to ignore, and — the part this book is built around — an operating model, agile hardware, finally capable of building products that can actually meet these constraints.

Two Worldviews, One Structure

It is worth naming the deeper pattern before getting to the regulation itself, because it explains why this chapter belongs in a book about agile hardware rather than in a separate sustainability appendix.

There are, broadly, two ways to see the relationship between a company and the material world it draws on. One treats time as linear — a beginning and an end — and treats the natural world as a resource to be used and disposed of. Its default posture toward the product life cycle is "use and dispose," and its default posture toward development is the same: plan once, build once, ship once, done. The other treats time as cyclical — seasons, materials, and products all returning rather than terminating — and treats the natural world as something with value beyond what can be extracted from it. Its default posture is "use and reuse," toward materials and toward products alike.[38]

The pattern should look familiar. It is the same distinction this book has been drawing between waterfall and agile all along. A linear worldview produces linear processes. A cyclical worldview produces iterative ones. Sustainability, in other words, is not a bolt-on constraint sitting outside the agile hardware conversation — it is the same underlying structure, applied to materials instead of to software releases. An organization that has already learned to think in cycles rather than straight lines has a head start on a problem that, at its core, is asking for exactly that shift.

The Regulatory Wave

None of this would carry much urgency without regulation attached to it. The EU's Circular Economy Action Plan, adopted in 2020, requires circular and eco-design principles across essentially all product categories sold in the EU.[39] The Corporate Sustainability Reporting Directive, in force since

---

[37]European Commission, Directorate-General for Environment, "Circular economy – New tool for measuring progress," Publications Office of the European Union, 2023.

[38]Woollacott, Marjorie, Anne Shumway-Cook, and Natasha Tassell-Matamua. "Worldviews and environmental ethics: Contributions of brain processing networks." EXPLORE (2023).

[39]European Commission, New Circular Economy Action Plan, 2020.

2023, requires companies above a certain size to report on sustainability with the same rigor previously reserved for financial statements.[40] The Netherlands has gone further still, committing to cut its consumption of primary abiotic resources — minerals, metals, fossil carbon — by 50 percent by 2030.[41] The single figure worth underlining for anyone leading product development: up to 80 percent of a product's total environmental impact is locked in at the design phase, before a single unit is manufactured.[42] Environmental performance, in other words, is not something operations fixes after the fact. It is an architectural decision, made at the same table where the decisions in Chapter 2 get made — and, per the blue ocean argument above, a decision about which uncontested market space to enter.

When Regulation Meets Resistance: The Nokia Lead Story

Regulation of this kind rarely arrives to a warm welcome, and it is worth being honest about that rather than pretending compliance is simple once the law is passed. I remember, from my years at Nokia, when the EU restriction on lead in electronics manufacturing first landed. The initial reaction inside the organization was resistance — lead-based solder was a known, reliable, well-understood process, and the alternative meant requalifying manufacturing lines, retesting reliability, and absorbing real short-term cost and risk. It was a genuine challenge, not a paperwork exercise.

And it was solved. The industry requalified its processes, lead-free manufacturing became the default, and the products kept shipping. I raise this not to minimize how disruptive regulatory change can feel from inside an organization living through it, but to make the opposite point: the pattern of “this will be too disruptive” followed by “we adapted and it became normal” is not new. It is close to universal. Organizations bracing for CSRD reporting, or eco-design requirements, or a digital product passport mandate, are living through the early, resistant phase of a cycle that has played out before and has a known ending.

Sustainability Is Good Business, Not Just Compliance

The business case reinforces the blue-ocean argument made at the start of this chapter: sustainability rewards whoever moves first, not merely whoever complies. Sitra, Finland's innovation fund, has found that circular economy thinking can increase the life-cycle value of a product by up to 75 percent, and that circular economy business models can generate up to seven times the revenue of traditional, linear models.[43] The same research found that companies already invested in circular practices weathered the COVID-19 shock better than the rest of the market — sustainability, in that light, functions as resilience, not just as ethics.

---

[40]European Commission, Corporate Sustainability Reporting Directive.

[41]Government of the Netherlands, “Circular Dutch economy by 2050.”

[42]European Commission, Circular Economy Action Plan impact assessment: share of environmental impact determined at the design phase (up to 80%).

[43]Sitra, “Sustainable Growth with Circular Economy Business Models,” 2022.

Two examples make this concrete. 3Step IT, a Finnish company founded in 1997, built its entire business on circulating IT hardware rather than selling and disposing of it — leasing, refurbishing, and redeploying equipment across its lifecycle. By 2022, that model was generating €418.6 million in revenue on 2.4 million managed assets.[44] BlueMovement, launched out of Bosch's decade-long enterprise agile transformation, took the same logic to home appliances: instead of selling a washing machine, it rents one, with Bosch retaining ownership, maintenance responsibility, and the incentive to build something that lasts.[45] Neither company is selling sustainability as a virtue. Both are selling a service, priced on retained ownership and reliability, that is dramatically more resource-efficient than the model it replaced. That is the point: done right, sustainability is not a cost center bolted onto the business model. It is the business model.

From Selling Products to Selling Outcomes

This is where the regulatory wave, the business case, and agile hardware's own logic converge most directly, because the shift Sitra describes — from selling a physical unit to retaining ownership and selling an outcome — has a structural precondition: it only works if the product can be maintained, monitored, and upgraded after it leaves the factory. That precondition is software.

Consider the trajectory of Wärtsilä's flexible fuel engines, work I have been directly involved in. A flexible fuel engine is not a single, frozen design shipped once and left to run unchanged for its operating life; it is a platform that can be reconfigured to run on different fuel blends as availability and regulation shift, extending its useful life well beyond what a single-fuel design would allow. The same direction of travel shows up, more radically, in the engines now being developed to generate electricity directly from hydrogen — technology I have not been personally involved in building, but which points at the same underlying pattern already discussed earlier in this book: the more a product's core value is defined and adjusted in software rather than cast permanently into hardware, the longer and more sustainably it can be kept in service.

SpaceX's Falcon 9 makes the same case from a different industry entirely. The first successful reuse of a Falcon 9 first stage occurred on 30 March 2017 — and the modular design that made reuse possible was not incidental to SpaceX's survival, it was a condition of it.[46] A reusable rocket is, among other things, a circular-economy product: a very expensive asset engineered from the outset to return, be inspected, and fly again rather than being discarded after a single use. It is worth connecting this back to Chapter 2's central claim: architecture determines what is possible. A tightly coupled, disposable-by-design product architecture cannot become circular no matter how much regulation demands it. A modular one, engineered for inspection, maintenance, and reuse, can.

---

[44]3Step IT company figures, financial year 2022: revenue €418.6 million, profit €25 million, 2.4 million managed assets; company founded in Finland in 1997.

[45]Bosch enterprise agile transformation, 2012–2021; BlueMovement appliance-as-a-service launched from the Netherlands as an outgrowth of that transformation.

[46]First successful reuse of a SpaceX Falcon 9 first stage: 30 March 2017.

None of this is free of friction. The EU's push toward digital product passports — machine-readable records of a product's materials, origin, and repair history, traveling with it across its lifecycle — is, for most manufacturers, a genuinely hard problem.[47] I have not been directly involved in building one, but the pattern is clear from every organization wrestling with the requirement: a digital product passport cannot be maintained with a spreadsheet and a paper trail. It requires the product to carry software, to report on itself, to be addressable and updatable after it ships. Which means the organizations that will meet this requirement comfortably are, again, the ones that already made the software-first architectural choice described in Chapter 2 — not because they were planning for digital product passports specifically, but because software-first is the same underlying capability that circular economy compliance turns out to require. Consumerism's original sin, in this light, is structural rather than moral: a business model built on selling disposable, single-purpose units gives the organization every incentive to make products that fail on schedule. A service model, where the company keeps the product working, gives it the opposite incentive. Software is what makes that second model operable at scale.

Speed Changes the Calculus

None of the three competitive mechanisms described earlier in this chapter — cost innovation, growth hacking, network effects — happens on the old development timeline, and neither does keeping up with regulation. The old model — years-long design cycles, frozen specifications, compliance treated as a late-stage gate before shipment — cannot keep pace with regulation and material science that are both moving quarter by quarter. Elon Musk put the underlying logic bluntly in 2018: “Pace of innovation is all that matters in the long run.”[48] The point is not speed for its own sake. It is that an organization's rate of learning determines how quickly it can find a compliant, resource-efficient, genuinely better solution — and regulation is now moving fast enough that a slow rate of learning is itself a compliance risk.

My friend Joe Justice — with whom I have spent countless hours comparing notes on exactly this question — has long argued that the same short-iteration discipline that lets Tesla and SpaceX ship hardware faster is what lets them absorb a materials change, a supplier shift, or a new regulatory requirement without it becoming a multi-year re-architecture. The sustainability wave is, from this angle, simply one more fast-moving requirement among many. Organizations that have already built the muscle to iterate in weeks rather than years are the ones that will absorb it without drama. Organizations still working in multi-year design cycles will experience every new regulation as a crisis, because their operating rhythm was never built to metabolize change of any kind quickly — sustainability-related or otherwise.

---

[47]European Commission, Circular Economy Action Plan, digital product passport provisions.

[48]Elon Musk (@elonmusk), X (formerly Twitter), September 26, 2018: “Pace of innovation is all that matters in the long run.”

The Convergence

Good strategy, in the end, keeps asking the same question this chapter opened with: where is the uncontested space, and who gets there first? Sustainability is one blue ocean, forced open by regulation and planetary limits that leave no real alternative. The shift from ownership to access is another, forced open by digitalization doing to physical products what it already did to recorded music. Cost innovation, growth hacking, and network effects are not separate strategies competing with sustainability for attention — they are the mechanisms by which an organization actually wins whichever blue ocean it has found, sustainable or otherwise. What ties all of it together is the same argument this book has been making since Chapter 2: architecture determines what can be built efficiently, but strategy determines what is worth building at all, and speed determines who gets there while the ocean is still blue. Regulation is forcing hardware organizations to ask the question. The organizations that answer it fastest, not merely correctly, are the ones that will own the next decade of their industry. Answering it, day to day, requires an operating charter — which is where this book turns next.

Reflection Questions

- If you mapped your industry's current competition onto the red-ocean/blue-ocean distinction, which of your product lines are still fighting over the same customers on the same terms — and where might a blue ocean be hiding in plain sight?
- Which of the three competitive mechanisms — cost innovation, growth hacking, or network effects — is your current product strategy actually built around, and is that the one your market will reward?
- If up to 80 percent of your product's environmental impact is set at the design phase, who in your organization currently owns that decision — and do they know it?
- Where in your organization's history has it resisted a regulatory change that, in hindsight, became simply the new normal? What made the resistance eventually give way?
- If you shifted from selling units to retaining ownership and selling an outcome, what would have to be true of your product's software for that model to actually work?
- If retention, not first sale, were the metric your team was measured on, what would you build differently?

Chapter 4. The Agile Product Charter for the User

Chapter 3 argued that winning any of the three competitive races described there — cost innovation, growth hacking, network effects — rewards whoever completes a learning cycle fastest. This chapter asks the practical question underneath that claim: why does a short iteration actually produce a better product than a long one? The answer is not a slogan. It is a specific, well-

documented mechanism called validated learning, and it is the foundation for everything else in this chapter — including the operating charter this book has been building toward since Chapter 3.

The Threat of the Long Design Loop

Every hardware project depends on its slowest module. If nine of ten subsystems can be redesigned in a week but the tenth takes six months, the whole project moves at the pace of the tenth — and worse, early design choices in that slow subsystem cannot be verified and corrected in time for launch, because there is no time left for another loop once the problem surfaces. A system built from modules with fast design loops is, in effect, several loops ahead of one built from modules with slow ones by the time both reach the same calendar date. The practical risk is not abstract: parts designed against assumptions that turned out to be wrong simply do not fit together, and there is no time left to fix it.

Barry Boehm's cone of uncertainty describes the same problem from a different angle: uncertainty about a project's eventual cost, schedule, and design is at its widest at the very start, and narrows only as real information — from prototypes, tests, and use — accumulates.[49] A project that commits to a single design early is committing before the cone has narrowed, betting that its first guess falls inside a range that, at that point, is still enormous.

The Design Paradox

Design theorist David Ullman named the uncomfortable structure underneath this problem the design paradox: at the very beginning of a project, almost nothing is known about the true design problem, yet design freedom is at its maximum. Every decision made from that point on narrows the freedom left for the decisions after it — and most of a product's lifetime cost is committed during exactly this early phase, well before anyone has learned enough to make those decisions well. Knowledge about the real problem, meanwhile, keeps arriving throughout the process — it just arrives too late to inform the choices that most needed it.

Ullman's proposed resolution is the paradigm shift this chapter is built around: obtain more knowledge earlier, specifically in order to keep design freedom open for longer, and postpone the commitment of cost until that knowledge exists.[50] Plotted against each other, the curves make the argument visually: design freedom collapses early if knowledge arrives late, while committed cost rises early on the same schedule regardless of whether the knowledge needed to justify it has actually arrived. Pulling the knowledge curve earlier is the only lever available to change the shape of the other two.

---

[49]Aroonvatanaporn, Pongtip, Chatchai Sinthop, and Barry Boehm. “Reducing estimation uncertainty with continuous assessment: tracking the ‘cone of uncertainty.’” 2010.

[50]Ullman, David. The Mechanical Design Process. McGraw-Hill, 1992.

Figure 4.1. The design paradox — design freedom, committed cost, and accumulated knowledge plotted across a project's timeline, from concept through release. Adapted from Ullman, D., The Mechanical Design Process, McGraw-Hill Inc. New York, 1992

Learn Then Design, Not Design Then Test

Toyota's classic response to this paradox, documented as set-based concurrent engineering, is to deliberately postpone commitment. In developing its exhaust systems, Toyota deliberately gave suppliers loose requirements and asked them to build a range of variants rather than a single specification. Early prototypes were used not to prove a chosen design correct but to explore the limits of what was possible — mapping trade-off curves between competing parameters, and the failure limits of each — before any single solution was picked. The set of options was narrowed only once enough was known to narrow it well, which avoided the unplanned redesign loops that come from committing to a design built on incomplete knowledge.[51]

Dantar Oosterwal documented the same underlying shift at Harley-Davidson, where the organization moved from a design-then-test model — pick a design, build it, then discover through testing whether it was right — to a learn-then-design model, where the purpose of an early prototype is explicitly to learn, not to prove. Oosterwal's own summary of the shift has become something of a maxim: prototypes are for learning, not for proving.[52] Harley-Davidson's projects had repeatedly failed for the same reason Ullman's paradox predicts: designs selected in early phases could not go through enough test-and-redesign loops in the time remaining before launch. Product development, in the model that replaced it, is organized around what needs to be learned next — not around which tasks are needed to exit the next gate.

From Verification Mindset to Learning Mindset

None of this works if failure is treated as something to be avoided rather than as information. A verification mindset treats a test as a pass/fail checkpoint: the system is expected to work, and a failure is a negative event to be minimized. That mindset is not wrong on its own terms, but it quietly restricts how fast an organization can learn, because a team that is punished for failing an early test will stop running the tests that are most likely to fail — which are exactly the tests that would have taught it the most.

This does not come naturally. A cluster of well-documented cognitive biases pushes people toward the verification mindset by default: confirmation bias leads teams to seek information that supports what they already believe rather than what challenges it; loss aversion makes the fear of a visible failure outweigh the value of what would be learned from it; hindsight bias makes past failures look,

---

[51]Ward, Allen, Jeffrey K. Liker, John J. Cristiano, and Durward K. Sobek. “The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster.” Sloan Management Review, 1995.

[52]Oosterwal, Dantar P. The Lean Machine: How Harley-Davidson Drove Top-Line Growth and Profitability with Revolutionary Lean Product Development. AMACOM, 2010.

in retrospect, as though they should obviously have been avoided, which discourages the trial-and-error that produced the insight in the first place; status quo bias favors the current approach simply because it is current; and overconfidence bias leads teams to skip experiments they wrongly assume they do not need. Thomas Edison's own description of his failed attempts at the light bulb filament makes the alternative stance explicit: “I have not failed. I've just found 10,000 ways that won't work.”

Embracing failure as a deliberate strategy, not merely tolerating it, is what separates organizations that learn fast from organizations that merely try to avoid embarrassment. Asked directly about the widely publicized failure of the Fire Phone, Amazon's Jeff Bezos did not distance the company from it: “If you think that's a big failure, we're working on much bigger failures right now, and I am not kidding. Some of them are going to make the Fire Phone look like a tiny little blip.”[53] The point was not that failure does not matter. It was that an organization unwilling to risk a failure at Fire Phone's scale has already capped how much it will ever learn.

Validated Learning and the Four Risks

Put together, the design paradox and the case against the verification mindset point toward a specific practice rather than a general attitude: validated learning — exploring the limits of a solution space before committing to one point within it, and treating each iteration as an experiment designed to close a specific, named gap in knowledge rather than simply as a step toward a predetermined answer.

That knowledge gap is rarely one thing. Product management research typically separates it into four distinct risks, and a mature validated-learning practice tests each one deliberately rather than assuming that solving one solves them all.[54] Value risk asks whether a customer will actually buy the solution — market fit, cost, quality. Usability risk asks whether the user can learn, use, and perceive its value at all. Viability risk asks whether the solution works for the business itself — can it be sold, serviced, and funded. Feasibility risk asks whether the organization actually knows how to build and scale it — whether the knowledge gaps, the development time, and the process itself allow it.

This is also, ultimately, what makes agile development agile rather than merely fast: not speed for its own sake, but a tight loop of reflecting on what was just learned, adapting the plan in light of it, and learning again — adaptive control applied to a design problem instead of to a mechanical system. Learning speed is not a side effect of short iterations. It is the entire reason they exist.

Key Decisions and Maturity Milestones

---

[53]Jeff Bezos, quoted in “Amazon's Jeff Bezos on the Fire Phone: ‘We're working on much bigger failures right now,’” GeekWire, 2016.

[54]Cagan, Marty. INSPIRED: How to Create Tech Products Customers Love. 2nd ed., Wiley, 2017.

A validated-learning mindset still needs a rhythm and a decision structure, or it dissolves into activity without accountability. A field pilot run inside a Fortune 500 electrical engineering organization — the first hardware program in that company run this way, completed in record time and with high quality — worked out what that structure actually needs to look like.[55]

The pilot's starting data point was the same one Oosterwal found at Harley-Davidson, cited earlier in this chapter: five years of project milestone data showed no correlation at all with project success. What did correlate, once the team started tracking it, was iteration length — specifically, whether a given part could iterate its way to a mature design in the time available. Milestones measured whether a calendar date had been hit. They said nothing about whether anything had actually been learned by that date.

The pilot's key structural move was to stop combining two different things that traditional hardware milestones had always combined: checking the quality of the work, and making the business decision about what to do next. Separating them produced two different kinds of checkpoints, run on two different rhythms; see Figure 4.2.

Figure 4.2. Key Decision and Synchronization Points. Key Decisions are when major investment decisions are made; maturity is checked beforehand in Synchronization Points.

Maturity milestones — called Build Maturity Reviews in the pilot — are learning checkpoints, not approval gates. A Build Maturity Review occurs when a new hardware prototype is ready to be tested and measured, and it tells the team exactly one thing: how many errors exist in this iteration and how much rework the design still needs. A genuinely new hardware design typically needs at least three such iterations before it reaches maturity — not because three is a magic number, but because that is how long it took, empirically, for the cone of uncertainty to narrow enough to trust the design. Every few weeks, peer-to-peer cadence reviews checked whatever was ready against the same quality criteria the old heavy milestones had used, but run as a review among engineers who had done similar work, not as a report to a manager several levels removed from the technology. The purpose was explicitly not reporting. It was learning: each cadence review was, in the pilot's own language, a learning cycle.

Key Decision Points are a different kind of event entirely, and the pilot kept them deliberately separate: investment decisions, a program ramp-up, a contract signature — business commitments that require the information the maturity milestones have accumulated, but that are not themselves technical reviews and do not gate technical execution. Synchronization points, held when cross-functional integration actually made sense rather than on a fixed calendar, fed both cadence output and rolling-plan updates into these decisions, so that by the time a Key Decision Point arrived, it was informed by real, current maturity data rather than by a date on a Gantt chart.

---

[55]Laanti, Maarit. “Piloting Lean-Agile Hardware Development.” Research, May 2016. DOI: 10.13140/RG.2.1.5016.2805.

The distinction matters because it resolves a tension every hardware organization eventually runs into: formal, expensive decisions genuinely need formal approval, and nobody credibly argues otherwise. What breaks under the old model is treating every review as if it were also a formal decision, which makes the review too heavy to run often and turns every technical checkpoint into a piece of political theater. Separate the two, and the learning can happen as often as the design paradox demands — inside cadence and synchronization — while the business decisions that genuinely need weight and ceremony happen only when there is actually something worth deciding.

The Charter, Grounded in Learning

Validated learning is the mechanism. The Agile Product Charter is the set of operating commitments that make practicing it consistently actually possible for a hardware organization. First published in 2016 and co-authored by a group of practitioners that includes Joe Justice, the charter reads:[56]

"We embrace agile methods as the engine driving innovative solutions and collaboration to amplify economic, ecological, and social benefits across our planet. Through this work we have come to value:"

- *Cross-functional team collaboration over specialization, process, and tools.*
- *Modularity over tightly-coupled solutions.*
- *Continuous customer collaboration over inflexible contracts.*
- *Useful continuous delivery over a single comprehensive delivery.*
- *Extending development through manufacturing over fixing problems in the field.*
- *Useful continuous documentation over comprehensive documentation.*

"That is, while there is value in the items on the right, we value the items on the left more."

What follows takes each of these six commitments in turn — briefly, because each deserves, and has received, treatment well beyond what one chapter can offer.[57] Read against the mechanism established above, each commitment is really the same instruction applied to a different seam in the organization: shorten the loop, and make sure what comes back through it is real information rather than a comforting confirmation.

Cross-Functional Team Collaboration Over Specialization, Process, and Tools

[56]Agile Product Charter, first published September 28, 2016, agileproductcharter.org. Co-authored by a group of practitioners including Joe Justice.

[57]Hajou, A., & Voropaeva, M. Engineering Agility: A Pragmatic Approach to Adapting Agile Practices for Hardware Product Development.

Traditional hardware organizations are built around discipline: mechanical engineering, electrical engineering, firmware, software, manufacturing engineering, each its own department, each handing its output to the next in line. Horizontal, process-based slicing of this kind — breaking work into phases or specialties rather than into end-to-end capabilities — has a specific, well-documented cost: early phases provide almost no information about the true state of the product, because nothing is actually validated until the phases are integrated at the end. Problems are found late, exactly when the design paradox says they are most expensive to fix.

Vertical slicing is the alternative: build a capability that includes a slice of as many phases and disciplines as possible in one pass, even though this requires more upfront investigation to define. The payoff is direct: more frequent system-level evaluations, meaning more learning cycles per month; more frequent releases that are actually valuable rather than merely complete; closer cooperation between the disciplines that have to solve the integration problem together instead of at a distance; and a sharp reduction in the risk of sunk cost, because a bad direction is discovered in weeks rather than after months of specialty work have already been poured into it.

This is exactly where the traditional boundary between hardware and software work is dissolving. When a hardware capability is developed simulation-first — validated computationally before a single physical prototype exists — the team that can actually complete one vertical slice through the design paradox has to include simulation, software, and hardware expertise together from the start, because the loop being shortened is a loop between a physical assumption and a computational test of it. I have seen this pattern repeat across more than one organization: the moment hardware development stops being sequential — model, then build, then test — and starts being simulated in the same loop as the software controlling it, the old departmental boundary between “hardware people” and “software people” stops describing how the actual work happens. The team is no longer organized around what kind of engineer someone is. It is organized around which loop they are trying to shorten together.

Robin Yeman, a Lockheed Martin Fellow, has described the results of over a decade of cross-functional collaboration applied to some of the most demanding hardware in existence — satellites and submarines — where teams spanning software, firmware, and hardware compressed delivery timelines that had previously been measured in years.[58] A different case makes the same point from an unglamorous but telling angle: procurement. In the development of an electric bus powertrain, Scania replaced a traditional phase-gate handover from construction to procurement — in which procurement's feedback arrived only after a design was already fixed, forcing exactly the kind of late redesign loop this chapter has been describing — with ten-week rolling-wave planning loops that invited procurement into the design conversation directly. Procurement stopped receiving fixed specifications and started learning the actual context and constraints behind them, which let its

[58]Robin Yeman (Lockheed Martin Fellow), “Speed of Delivery,” Agile for Hardware Conference, Amsterdam, 2019.

feedback reach construction early enough to act on, rather than late enough to force a redesign.[59] If cross-functional collaboration works at the level of complexity that defense and aerospace hardware demands, and reaches as far into the organization as procurement, the claim that a given team's own hardware is “too specialized” for the same approach becomes difficult to sustain.

Modularity Over Tightly-Coupled Solutions

Chapter 2 covered this principle at length as an architectural and strategic choice, so it needs only a brief restatement here as an operating value: teams should be structured to mirror the module boundaries of the product, not the other way around. The design-loop argument earlier in this chapter gives modularity an additional justification beyond the architectural one — a module with a clean interface can run its own short design loop and narrow its own cone of uncertainty independently, without waiting for the rest of the system's slowest module to catch up. A team whose work is entangled with three other teams' work inherits their slow loops as its own, no matter how fast its own part of the problem could otherwise be learned.

Continuous Customer Collaboration Over Inflexible Contracts

Hardware development has long run on the logic of the fixed-scope contract: requirements are gathered, frozen, signed off, and built to specification, with the customer reappearing only at acceptance testing. The problem is not the discipline of writing things down. The problem is that the customer's understanding of what they need is rarely complete when the contract is signed, and a rigid contract has no mechanism to absorb what is learned afterward — the same design-paradox problem examined earlier in this chapter, applied to the customer relationship rather than to a single component. Scania's rolling-wave planning with its own procurement function, described above, is the same fix applied one layer further out: replace a single frozen handoff with a running conversation, so what gets learned along the way can still change what gets built.

Continuous customer collaboration replaces the single sign-off with exactly that kind of running conversation: early prototypes shown before they are polished, pilot units placed with real users before full production commits, and — as Chapter 3 argued — retention treated as the honest measure of whether the collaboration is actually working.

Useful Continuous Delivery Over a Single Comprehensive Delivery

The traditional hardware release is a single, comprehensive event: one big-bang delivery, frozen at the moment of shipment. The charter's alternative is to deliver smaller increments more often. For cyber-physical systems, this requires a pipeline that can verify software, firmware, and hardware changes together, continuously, rather than in one final integration push — the discipline now

[59]Case example from the author's Leading Agile Hardware course material: Scania e-propulsion bus development, rolling-wave procurement planning in ten-week loops.

documented as Industrial DevOps.[60] It is no coincidence that Robin Yeman, the source of the cross-functional example earlier in this chapter, is also a co-author of the book that defined the practice: continuous delivery and cross-functional collaboration reinforce each other. A team that cannot integrate its own changes continuously cannot deliver continuously, no matter how cross-functional its membership.

Extending Development Through Manufacturing Over Fixing Problems in the Field

The charter's fifth commitment is easy to misread, because it sounds at first like an argument for supporting products after they ship. It is closer to the opposite: a preference for catching and closing problems through an extended, rigorous manufacturing process, rather than allowing them to surface only once the product is already in a customer's hands. A problem found on the production line is expensive. The same problem found in the field — after shipping, after a customer has already built the product into their own operations — is usually far more expensive, and often reputationally costly in a way a production-line defect never becomes.

This is the design-loop argument again, applied to the boundary between development and manufacturing rather than to the boundary between disciplines. Toyota's set-based approach to its exhaust-system suppliers, described earlier in this chapter, treated manufacturing variation itself as a source of learning, not merely as an execution step to get through once a design was finalized. Extending development's rigor — its test loops, its set-based exploration of trade-offs, its willingness to look for failure rather than merely confirm success — through the ramp into full manufacturing is what the charter is actually asking for. None of this eliminates the need to learn from the field once a product genuinely is out in the world; a fleet that reports data back and can receive corrections remotely is a valuable safety net for whatever the manufacturing process does not catch. But it is a safety net, not the plan. The charter's ordering is deliberate: extend development through manufacturing first, and treat fixing problems in the field as the fallback, not the primary loop.

Useful Continuous Documentation Over Comprehensive Documentation

The charter's final commitment is easy to treat as an afterthought, but it follows directly from everything above it. Comprehensive documentation, written once and frozen at the moment of a big-bang delivery, is accurate on the day it is written and increasingly wrong every day after, because nothing about a continuously delivered product stays still long enough to describe comprehensively. The same Industrial DevOps pipeline that makes continuous delivery possible is what keeps documentation useful rather than comprehensive: documentation generated from, or continuously validated against, the same system it describes, updated in the same loop as the product itself, rather than produced separately and left to drift. A hardware organization that has solved

[60]Johnson, S., & Yeman, R. (2023). Industrial DevOps: Build Better Systems Faster. IT Revolution Press.

continuous delivery has usually already solved this one as a side effect; an organization that has not is the one still shipping a manual describing a version of the product that no longer exists.

The Charter as a Bridge

Put together, these six commitments are not arbitrary preferences. Each one is the design-paradox argument, applied to a different seam in the organization: cross-functional teams shorten the loop between disciplines; modularity shortens the loop between one module's learning and the rest of the system's; continuous customer collaboration shortens the loop between what the customer actually needs and what the organization believes they need; continuous delivery and documentation keep that shortened loop from silently reverting to a big-bang default; and extending development through manufacturing shortens the loop between a design assumption and the moment it is tested against physical reality — before, rather than after, a customer discovers the gap for them. None of these commitments is achievable through intention alone. Cross-functional teams need the value stream structures described in the next chapter; continuous delivery needs the iteration cadences and test automation covered later in this part; modularity needs the architectural discipline Chapter 2 already laid out. Validated learning is the mechanism. This charter is the value system built to practice it, consistently, at the pace the market described in Chapter 3 now demands. Part II of this book is the machinery that makes living up to it possible in practice.

Reflection Questions

- Which module or subsystem in your current product has the longest design loop — and how many of your other teams are, in effect, waiting on it without realizing it?
- Where in your last major project was cost already committed before your team had learned enough to justify it?
- Is your organization's default posture toward a failed test closer to a verification mindset or a learning mindset — and which cognitive bias described in this chapter would your own team recognize first in itself?
- Of the four risks — value, usability, viability, feasibility — which one does your current validation process actually test, and which ones are you assuming away?
- Where does your organization still slice work horizontally, by phase or specialty, when a vertical slice would surface the same problem weeks earlier?
- Extending development through manufacturing, or fixing problems in the field — which one does your product's current cost structure actually reward?
- Does your organization separate technical quality review from business investment decisions — or do your milestones still combine both, making every review a de facto go/no-go moment?