The Economics of Software Quality and Engineering Excellence:

Overcoming Executive Short-Termism to Build Product Dynasties

The Crisis of Software Value Realization in the Modern Enterprise

The global software development ecosystem is currently experiencing a profound and systemic crisis of value realization. Despite decades of rapid advancement in computational power, sophisticated programming paradigms, and the proliferation of project management methodologies, the failure rate of software projects remains staggeringly high. Industry analyses consistently demonstrate that the majority of new software products fail to deliver their anticipated return on investment (ROI) and broader business value. Research indicates that 70% of organizations experience at least one project failure each financial year, and between 65% to 80% of IT projects are classified as outright failures by executive leadership.[1]

The definition of failure within this context is not strictly limited to catastrophic, headline-generating system collapses or unrecoverable technical crashes. Rather, failure is often far more subtle and insidious, eroding capital over extended periods. A software product may technically launch and enter production, but if it runs significantly over budget, misses its critical market delivery deadlines, lacks end-user adoption, or fails to meet the strategic business objectives outlined in its foundational charter, it constitutes a failure in the eyes of stakeholders and the market.[1]

The Standish Group's seminal CHAOS report provides a highly granular breakdown of these outcomes, revealing a persistent inability across the industry to execute software initiatives reliably and profitably. Over the past two and a half decades, the statistics regarding project success have remained stubbornly bleak, highlighting a structural flaw in how software is commissioned, funded, and built.

Project Outcome Classification	Percentage of Total Projects	Defining Characteristics of the Outcome
Successfully Completed	31%	Projects delivered strictly on time, within the allocated budget, and meeting all expected

		functional and business requirements.[1]
Challenged Projects	50% - 52%	Projects that eventually launched but severely exceeded their budgets, missed delivery deadlines, or failed to deliver the promised functionality.[1]
Outright Failures	19%	Projects that were completely abandoned or scrapped prior to launch, resulting in total sunk costs with zero value realization.[1]

The financial implications of this pervasive execution crisis are colossal, creating macroeconomic drags on productivity and capital efficiency. Globally, the cost of failed digital transformation efforts and software initiatives is estimated at an astonishing $2.3 trillion annually.[4] Within the United States alone, a 2020 report from the Consortium for IT Software Quality (CISQ) found that the total cost of unsuccessful development projects is approximately $260 billion.[5] Furthermore, the total cost of operational failures stemming from poor software quality is estimated at $1.56 trillion.[5] For individual enterprises, the stakes are existential; approximately 17% of large IT projects perform so poorly that they pose a direct threat to the survival of the commissioning company.[2]

The primary drivers of these failures are rarely rooted in a fundamental lack of technical capability among engineering teams or a deficit of computational resources. Instead, these systemic failures are overwhelmingly driven by human factors, organizational resistance to change, and, most critically, a pervasive misalignment between executive financial mandates and the empirical realities of software engineering best practices.[4] When financial oversight is decoupled from engineering reality, the resulting products inevitably suffer from compromised architectures, inadequate usability, and unmanageable technical debt.

Executive Short-Termism and the Budget Paradox

A significant portion of the blame for persistent software and system failures rests squarely on the shoulders of financially driven executives and directors who prioritize short-term quarterly earnings over long-term structural viability.[6] The prevailing corporate culture of "short-termism"—heavily pressured by institutional investors, private equity models, and Wall Street analysts—incentivizes executives to artificially inflate immediate financial results at the

direct expense of long-term performance, innovation, and product quality.[7]

This dynamic creates a highly destructive budget paradox in software engineering. Consider a standardized scenario where empirical data, historical velocity, and rigorous architectural planning dictate that a nominal budget of $100,000 is required to properly develop a new product or a significant product feature. Financially driven executives, seeking to minimize capital expenditures to boost short-term profit margins, will frequently exert immense pressure to reduce this budget to $80,000.[9] They operate under the flawed assumption that cutting corners, reducing the headcount of specialized roles, and compressing timelines will forcefully increase operational efficiency and yield "more short term profit."

However, software economics definitively dictate that achieving a robust, scalable, and highly adopted product—a "product dynasty" that serves as a self-sustaining engine of continuous business value and wealth generation—actually requires going the "extra mile." For real, sustained market success and maximized net profit, the optimal investment is often $120,000.[9] This extra 20% effort, allocated specifically to deep usability research, rigorous software design, and comprehensive engineering architecture cycles, yields a product that is exponentially superior, effectively delivering a 10x return on the incremental investment.[13]

By enforcing a constrained $80,000 budget, executives systematically eliminate the critical discovery, design, testing, and refactoring phases. The resulting product is structurally deficient, difficult for the end-user to navigate, and fundamentally incapable of capturing the market share or internal adoption required to generate a positive ROI.[11] The mentality of cutting as many corners as possible and executing with the absolute minimum number of people is completely self-defeating. It destroys a massive amount of human effort, burning out highly capable professionals on product teams whose potential is ruined by arbitrary executive demands.[17]

The macroeconomic and microeconomic consequences of corporate short-termism have been systematically measured and proven to be detrimental. The McKinsey Corporate Horizon Index, which meticulously tracks the performance of companies managed for long-term value creation versus those managed for short-term earnings, reveals a stark contrast in economic outcomes.

Key Performance Metric (Evaluated 2001-2014)	**Long-Term Focused Companies**	**Short-Term Focused Companies**
Average Revenue Growth	47% higher than baseline	Baseline performance

Average Earnings Growth	36% higher than baseline	Baseline performance
Economic Profit Growth	81% higher than baseline	Baseline performance
Research & Development (R&D) Spending Growth	8.5% annualized growth rate	3.7% annualized growth rate
Total Job Creation	Added 12,000 more jobs on average	Baseline performance

Long-term companies demonstrate significantly less revenue volatility and exhibit far greater resilience during economic downturns, continuing to invest in critical research and development even when short-term firms slash budgets to preserve immediate profit margins.[7] The artificial accounting moves and arbitrary budget discounts used to meet quarterly consensus estimates directly sabotage the rigorous engineering cycles required to build high-quality software.[7] In the context of software development, this short-termism manifests as the deliberate underfunding of vital non-functional requirements, usability testing, and architectural planning, directly guaranteeing the long-term failure of the product.

Minimum Viable Usability and the Adoption Crisis

When executive short-termism forces strict budget constraints, the very first disciplines to be abandoned are typically user experience (UX) design, product discovery, and extensive usability research. This systemic underinvestment leads directly to the creation of products that lack "Minimum Viable Usability" (MVU), resulting in a catastrophic lack of end-user adoption and overwhelmingly poor user experiences.[20]

For the past decade, the technology industry has heavily relied on the concept of the Minimum Viable Product (MVP) to validate market fit. However, driven by restricted budgets, the MVP paradigm frequently over-indexes on bare-bones backend functionality while entirely ignoring the human-computer interaction layer.[21] MVU redefines this approach by establishing the absolute minimum level of usability, intuitiveness, and workflow alignment required for users to successfully achieve their essential goals without severe friction or cognitive overload.[20] If a software product fails to cross this critical MVU threshold, no amount of sophisticated backend functionality, complex algorithms, or aggressive marketing will save it from market rejection or

internal workforce rebellion.[20]

Usability testing is not merely a superficial mechanism for finding interface bugs or ensuring brand colors are correct; it is a foundational engineering practice that ensures the software's flow matches the target users' actual needs, daily habits, and established mental models.[22] When engineering teams are denied the resources to walk the "extra mile" and conduct iterative usability tests, they are forced to build products based on internal assumptions and executive biases rather than empirical user data.[23] True usability research involves extensive contextual inquiry, where designers shadow end-users to observe how they complete tasks in their actual, high-pressure work environments, specifically looking for manual workarounds that indicate software deficiencies.[24]

The financial return on investing in this 20% extra effort for usability is exceptionally high and consistently measurable. Industry analyses, such as those conducted by Forrester, indicate that every $1 invested in robust UX yields a return of $100.[25] By investing the necessary budget to conduct guerrilla usability testing, leverage platforms like Mechanical Turk for scalable feedback, or run comprehensive beta testing with actual users, product teams can iterate toward MVU before hard-coding rigid architectural paths.[26]

When MVU is ignored to save upfront development costs—dropping the budget from the necessary $120,000 to the demanded $80,000—the backend financial penalties are severe and multi-faceted. Poorly designed software generates an immense volume of support tickets. The operational cost of maintaining a large customer support apparatus to assist confused users quickly eclipses the initial, localized savings derived from skipping UX research.[4] In enterprise settings, introducing software with low usability actively disrupts existing operations, severely impacting employee productivity. Industry data shows that up to 30% of planned enterprise software functionality is never used because the interface is simply too complex for the workforce to adopt, rendering the development of those features a complete waste of capital.[28] Ultimately, in consumer-facing products, a lack of MVU leads to immediate customer churn, ensuring the product fails to gain the end-user acceptance demanded by the underlying business model.[1]

The Anatomy and Impact of Technical Debt

Beyond the critical failure of usability, the most destructive and enduring consequence of underfunded, rushed software development is the massive accumulation of technical debt. Coined by software engineer Ward Cunningham to explain software engineering trade-offs to non-technical stakeholders, technical debt is a powerful metaphor representing the future cost of rework incurred by choosing an easy, expedient, short-term implementation over a more robust, long-term architectural solution.[30]

When budgets are forcefully restricted to the $80,000 threshold and delivery timelines are artificially compressed by executive mandates, developers are forced into a corner. To deliver

anything at all, they must take severe technical shortcuts. They bypass comprehensive automated testing, duplicate code logic rather than building reusable components, ignore established coding standards, hardcode variables, and leave highly complex systems entirely undocumented.[32] Much like financial debt, technical debt accrues compound "interest" over time. The longer these workarounds and sub-optimal design choices remain active in the codebase, the more difficult, dangerous, and exponentially expensive it becomes to implement any future changes or feature additions.[34]

The burden of technical debt is a massive, often invisible drain on corporate resources, severely depressing software ROI and stalling digital transformation initiatives.[36] Analysis of enterprise IT environments burdened by high technical debt reveals a landscape defined by staggering quantitative losses and diverted capital.

Metric of Impact	Quantitative Consequence of Technical Debt
Developer Productivity Loss	Developers spend up to 42% of their total working time (approximately 13.5 hours per week) solely dealing with technical debt, bad code, and maintenance rather than building new value.[37]
Global Financial Opportunity Cost	The opportunity cost of lost developer productivity and delayed time-to-market due to technical debt is estimated at nearly $85 billion worldwide annually.[37]
IT Budget Diversion	Technical debt maintenance consumes between 10% to 20% of IT budgets specifically intended for the development of new products, and can consume up to 40% of the total IT department spend.[40]
Enterprise Valuation Drag	Chief Information Officers estimate that technical debt accounts for 20% to 40% of the financial value of their entire technology estate before depreciation, acting as a massive off-balance-sheet liability.[43]
Service Delivery Deceleration	Organizations struggling with unchecked technical debt experience up to 50% slower

	service delivery, critically delaying product releases and eroding competitive advantage.[38]

Technical debt is not a monolithic concept; it manifests in several distinct categories, each systematically compromising the core pillars of software engineering: performance, scalability, reliability, and security.

Architecture debt emerges when high-level structural decisions are rushed to meet tight budgets. This results in outdated frameworks, monolithic architectures, and tightly coupled components that jeopardize the system's ability to scale.[33] As the business attempts to grow or pivot, these rigid legacy systems lack the flexibility to adapt. The immense complexity and fragility of the codebase impede the integration of new application programming interfaces (APIs) and modern cloud environments, creating severe operational bottlenecks.[32] Because the codebase is deeply entangled and unmanageable, every single new feature deployment risks breaking existing, seemingly unrelated functionality, drastically reducing the velocity of innovation.[45]

Infrastructure and process debt compound these architectural issues. When deployment processes are outdated and Continuous Integration/Continuous Deployment (CI/CD) pipelines lack efficiency, automation is hindered.[32] Without proper infrastructure planning, teams face massive roadblocks in updating dependencies or ensuring that cloud environments remain cost-effective.[32] This lack of automated testing and quality assurance, driven by time constraints, guarantees that the resulting software is inherently unstable. This leads directly to frequent system failures, data corruption, and catastrophic application downtime.[46] High technical debt is strongly correlated with severe IT outages that disrupt business continuity, erode customer trust, and lead to severe developer burnout as engineers spend their nights and weekends fighting fires rather than innovating.[17]

Perhaps most critically, security debt is a rapidly growing threat stemming directly from underfunded engineering cycles. When teams are forced to cut corners on encryption standards, authentication protocols, or routine vulnerability patching to meet executive deadlines, the software is left dangerously exposed to cyber threats.[32] According to Veracode's State of Software Security Report, a staggering 42% of active applications possess significant security debt, with 11% harboring critical, highly exploitable flaws that have remained unfixed for over a year due to resource constraints.[49] This exposes the organization to massive financial and reputational risks, regulatory compliance violations, and catastrophic data breaches, completely destroying any short-term profit gained by the initial $20,000 budget reduction.[1]

The Economics of Software Quality

The prevailing executive mentality heavily relies on the assumption that enforcing tight budget

caps and demanding rapid delivery will inherently yield cost savings and higher profit margins. However, the rigorous economics of software quality completely invalidate this deeply flawed premise. Comprehensive research conducted by industry expert Capers Jones, detailed in *The Economics of Software Quality*, provides overwhelming empirical evidence that high-quality engineering translates directly into strongly positive ROI and a drastically reduced Total Cost of Ownership (TCO).[50]

Jones's analysis of over 13,000 software projects across diverse industries demonstrates that finding and fixing bugs is the single most expensive and time-consuming activity in the entire software development lifecycle.[13] The cost of defect remediation operates on an exponential curve based on when the defect is discovered. A bug or architectural flaw that costs $100 to fix during the initial planning and design phases can easily escalate to a cost of $10,000 if left to fester as technical debt and discovered by a user in a live production environment.[42]

A central, irrefutable axiom of software economics is that "quality leads and productivity follows".[13] Executive efforts to artificially inflate developer productivity by stripping out quality assurance (QA) processes, bypassing code reviews, and eliminating defect prevention protocols are mathematically counterproductive. Organizations that invest the extra 20% effort in rigorous, mathematical test case design, early defect prevention, and pre-test defect removal consistently achieve a 20% to 30% overall improvement in software productivity and quality.[13]

Jones's empirical data reveals that high-quality levels are invariably associated with shorter-than-average development schedules and lower-than-average development costs over the complete lifecycle of the product.[53] By investing the additional 20% budget upfront—allocating the full $120,000 rather than restricting it to $80,000—engineering teams are empowered to execute robust structural and functional quality measures. This crucial upfront investment effectively eliminates the "technical mortgage" that would otherwise quietly drain the IT budget for years post-release through endless maintenance cycles.[50] Quality Assurance is not an operational overhead to be minimized; it is an essential form of risk management, system insurance, and value protection. Defect prevention unequivocally delivers a vastly superior ROI than late-stage defect detection and emergency patching.[54]

Cultivating a Software Product Dynasty

When organizations successfully overcome the trap of financial short-termism and fundamentally commit to engineering excellence, rigorous architecture, and deep usability research, they possess the capability to build a true "product dynasty." A product dynasty is not merely a functional piece of software; it is a comprehensively designed solution that achieves a highly valuable, intensely loyal customer base, serving as a reliable, long-term engine of business value, market dominance, and sustained wealth generation over multiple decades.[55]

Building a product dynasty requires a fundamental paradigm shift: prioritizing long-term brand

affinity, unshakeable system reliability, and extreme user satisfaction over immediate, quarter-by-quarter cost minimization. Companies like Dyson, while operating primarily in the hardware space, perfectly exemplify this approach by accepting thousands of prototype failures and investing heavily in both design and usability, ultimately carving out an unassailable premium market position where they avoid price wars entirely.[56] In the software domain, a dynasty is achieved when a product seamlessly transitions from a mere transactional utility into a deeply embedded, indispensable platform that users champion.

The concept of a "10x ROI" or an exponential return framework applies directly to this methodology. An investment that specifically targets the "extra mile" in architecture, security, and user experience will generate returns that dwarf the initial expenditure.[57] For example, allocating a 20% increase in initial design and engineering investment can lead to a 100% improvement in buyer satisfaction, drastically reducing customer acquisition costs (CAC), minimizing support overhead, and driving massive organic growth through user advocacy.[59]

Conversely, attempting to artificially engineer short-term profit by cutting the budget to $80,000, outsourcing to the lowest bidder without oversight, or underfunding the core architecture guarantees a codebase so fragile and convoluted that it actively repels top engineering talent.[60] This severely hampers future hiring, permanently stifles development velocity, and ultimately necessitates a complete, highly expensive system rewrite, destroying the very profit the executives initially sought to protect.

Operationalizing Excellence: Kaizen and the Toyota Production System

To structurally prevent the accumulation of crippling technical debt, mandate Minimum Viable Usability, and enforce the creation of high-value software, organizations must adopt and rigorously enforce proven process frameworks. A major source of software failure is that financially driven executives and directors often completely lack an understanding of established industrial methodologies, such as the Toyota Production System (TPS) and Kaizen, which have been proven to drive excellence and can be directly transposed into software engineering best practices.[61]

The Toyota Production System is a comprehensive, holistic management philosophy meticulously designed to thoroughly eliminate waste (muda), reduce overburden (muri), and stabilize unevenness (mura), all while shortening lead times and delivering exceptionally high quality.[61] TPS relies on two primary pillars, both of which have profound applications in modern software development:

The first pillar is **Just-in-Time (JIT)** production, which dictates making only what is needed, exactly when it is needed, and in the exact amount needed.[61] In software engineering, JIT perfectly aligns with Agile methodologies that minimize massive upfront coding phases, focus

instead on the immediate delivery of functional, tested value, and severely reduce the massive waste associated with over-engineering features that users do not actually want or need.[61]

The second pillar is **Jidoka**, often translated as "automation with a human touch." Jidoka is based on the critical concept of stopping the production line immediately the moment an abnormality or defect is detected, preventing bad products from moving downstream and compounding errors.[61] In a software context, Jidoka manifests as robust Continuous Integration/Continuous Deployment (CI/CD) pipelines equipped with extensive, automated testing suites. When a code commit fails a quality check, security scan, or integration test, the deployment pipeline halts automatically. This forces developers to address the root cause of the defect immediately, rather than pushing flawed code into a production environment where it instantly becomes expensive technical debt.[63]

Deeply embedded within the DNA of TPS is the philosophy of **Kaizen**—the practice of continuous, incremental improvement for the better.[64] Kaizen is not a one-time initiative; it represents a cultural mindset of relentless self-reflection (hansei) and collaborative problem-solving. In software engineering, Kaizen is enacted through regular, blameless sprint retrospectives, iterative usability testing based on actual user feedback, and the continuous, scheduled refactoring of code to maintain structural health.[66]

Instead of disruptive, top-down financial mandates that arbitrarily gut project budgets, Kaizen empowers cross-functional teams—comprising developers, UX designers, and IT operations—to identify process inefficiencies and incrementally improve the codebase on a daily basis.[63] By treating software development as an ongoing, highly skilled craft rather than a finite, transactional expense to be aggressively minimized, companies can establish a resilient, digital-native engineering setup. This environment continuously adapts to market changes without fracturing under the immense weight of accumulated technical debt, ultimately fulfilling the vision of engineering excellence.[61] A culture that supports Kaizen fosters a sense of psychological safety and "team eliteness," where engineers feel empowered to take the extra 10% or 20% effort to perfect a feature, knowing that quality is genuinely valued by leadership.[68]

Strategic Alignment Through Kelly OKRs in Agile

Even with robust frameworks like TPS and Kaizen firmly in place at the team level, a severe communication and goal-alignment gap often persists between the C-suite and the engineering floor. Financially driven executives speak the language of ROI, quarterly profit margins, EBITDA, and strict budget caps, while engineers speak the language of technical debt remediation, code refactoring, MVU, and story points. To successfully bridge this divide and align the $120,000 engineering reality with executive expectations, progressive organizations are increasingly leveraging Objectives and Key Results (OKRs), particularly as adapted for Agile environments by thought leaders like Allan Kelly.

In his comprehensive methodology, *Succeeding with OKRs in Agile*, Allan Kelly outlines exactly

how OKRs can reawaken the inherent ambition and drive within Agile teams by prioritizing overarching business purpose and long-term strategy over the mindless execution of backlogs.[70] Traditional Agile teams, especially those operating under strict financial constraints, frequently fall into the trap of becoming "feature factories." They endlessly burn down backlog tickets to show velocity, without ever pausing to question if the features being shipped actually deliver tangible business value or achieve Minimum Viable Usability.[71]

The OKR framework powerfully shifts the paradigm from measuring output (lines of code written, features shipped within an $80,000 budget) to measuring true outcomes (user behavior changes, market penetration, system stability).

- **Objectives** are established as broad, highly ambitious, qualitative goals tied directly to the corporate strategy (e.g., "Establish a dominant product dynasty in the financial technology sector by delivering the most reliable platform").
- **Key Results** are the specific, quantitative, measurable constraints and success criteria that define the actual achievement of that objective (e.g., "Reduce user onboarding time by 40% through enhanced UX," or "Decrease production bug rate and system downtime by 60%").[70]

Kelly's Agile OKR framework effectively neutralizes executive short-termism by fundamentally redefining what constitutes project success. Success is no longer measured by the flawed metric of whether a project was completed for $80,000 instead of the required $120,000. Instead, success is exclusively measured by whether the Key Results—such as user adoption rates, system stability, and sustained, multi-year ROI—were successfully achieved.[72]

Furthermore, OKRs provide a highly visible mechanism for engineering teams to formalize the management of technical debt and usability research. By elevating "debt reduction" and "usability validation" to the level of strategic Key Results, these critical engineering practices become visible to the C-suite as primary drivers of business value, rather than being dismissed as hidden, unnecessary engineering complaints. McKinsey notes that leading, highly mature technology companies explicitly target roughly 15% of their total IT budgets specifically for proactive technical debt remediation, treating it as an unavoidable, necessary cost of maintaining long-term innovation and agility.[73] OKRs facilitate this critical alignment by turning managers, financial directors, and executives into true partners focused on long-term, wealth-generating outcomes, rather than adversaries focused solely on arbitrary, self-defeating cost-cutting.[70]

Conclusion

The staggeringly high failure rate of software products across the global enterprise landscape is not a technological inevitability; it is an economic, cultural, and managerial failure. The systemic, relentless drive to maximize short-term profits by enforcing austere budgets—demanding $80,000 solutions for realities that empirically require a $120,000

investment—results in catastrophic, cascading long-term consequences. This financially driven short-termism directly causes the severe erosion of Minimum Viable Usability and vastly accelerates the accumulation of crippling technical debt, rendering the resulting products unadoptable by users, unscalable by the business, and highly vulnerable to security breaches.

To break this vicious cycle and successfully build highly profitable, multi-generational product dynasties, corporate leadership must fundamentally realign its financial perspectives with the established, data-driven economics of software quality. Industry data unequivocally demonstrates that investing the extra 20% effort in upfront architectural rigor, deep contextual usability research, and automated, preventative quality assurance generates exponential returns. This targeted investment drastically lowers the total cost of ownership over the product's lifecycle, eliminating the massive hidden costs of maintenance, support, and eventual system replacement.

By deeply integrating industrial excellence frameworks like the Toyota Production System and Kaizen, organizations can empower their teams to build quality into the product inherently. Furthermore, aligning executive strategy with engineering reality through outcome-focused Agile OKRs ensures that both the board and the developers are driving toward the same vision of long-term value. Ultimately, treating software engineering as a strategic, value-generating craft rather than an operational expense to be aggressively minimized is the sole reliable pathway to sustained market dominance, operational resilience, and superior return on investment in the digital age.

Study after study: failures of design and innovation:

- Frost & Sullivan reported (i) that only one in 300 new products significantly impacts a company's growth and (ii) that only 1% of new products recoup their product development costs.
- The Corporate Strategy Board reported that over the past four decades, of the 172 companies that spent time in the Fortune 50, only 5% sustained a growth rate greater than the growth rate of the gross domestic product.
- PricewaterhouseCoopers reported that only 11% of all venture investments get to any capital liquidity.
- R.G. Cooper reported that new products succeed 25% of the time.
- The Product Development Management Association (PDMA) claims that new products succeed 59% of the time.

The 12 studies that cited innovation success rates they cited :

Source	Rate
Frost & Sullivan, "Growth Process Toolkit: New Product Development," 2008.	0.3%
Frost & Sullivan, "Growth Process Toolkit: New Product Development," 2008.	1%
Andrew Campbell and Robert Park, 'Stop Kissing Frogs," *Harvard Business Review*, July-August 2004.	1%
Dr. John Sviokla, "The Calculus of Commerce," Diamond Cluster International, Inc. 2004.	3%
Corporate Strategy Board, "Stall Points," 1998. Cited in Clayton Christensen and Michael Raynor, "The Innovator's Solution," page 5, *Harvard Business School Press*, 2003.	5%
Andrew Campbell and Robert Park, 'Stop Kissing Frogs," *Harvard Business Review*, July-August 2004.	10%
Kevin J. Clancy and Randy L. Stone, "Don't Blame the Metrics," *Harvard Business Review*, June 2005.	10%
Corporate Strategy Board, "Overcoming Stall Pints," 2006.	10%

Jakob Nielsen: We have achieved only 2% of the UXD we need.

Jakob Nielsen, Ph.D., is a User Advocate and principal of the Nielsen Norman Group which he co-founded with Dr. Donald A. Norman (former VP of research at Apple Computer). Before starting NNG in 1998 he was a Sun Microsystems Distinguished Engineer.

Member-only story

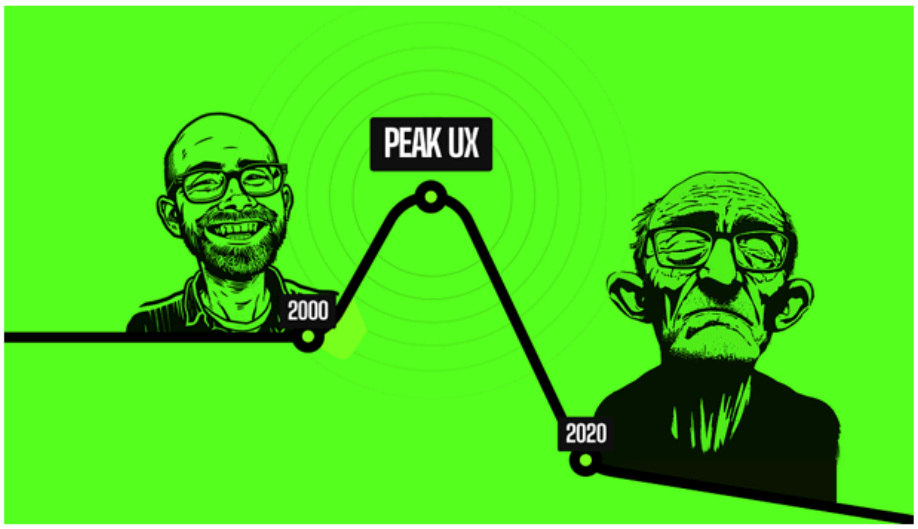
Design lost its seat at the table

Why and what can we do?

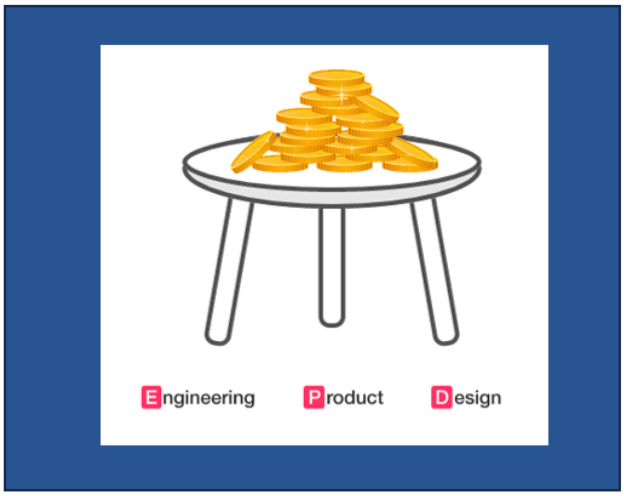
Michal Malewicz · Follow
5 min read · 6 days ago

We used to be problem solvers that entire room of stakeholders listened to with awe. Now we're figma drag & droppers. Instead of awe we only get:

Slap our design system on it and shut up!



Three-legged stool design culture replaced by finance short-termism



"No way you can consumer-research the Mac"

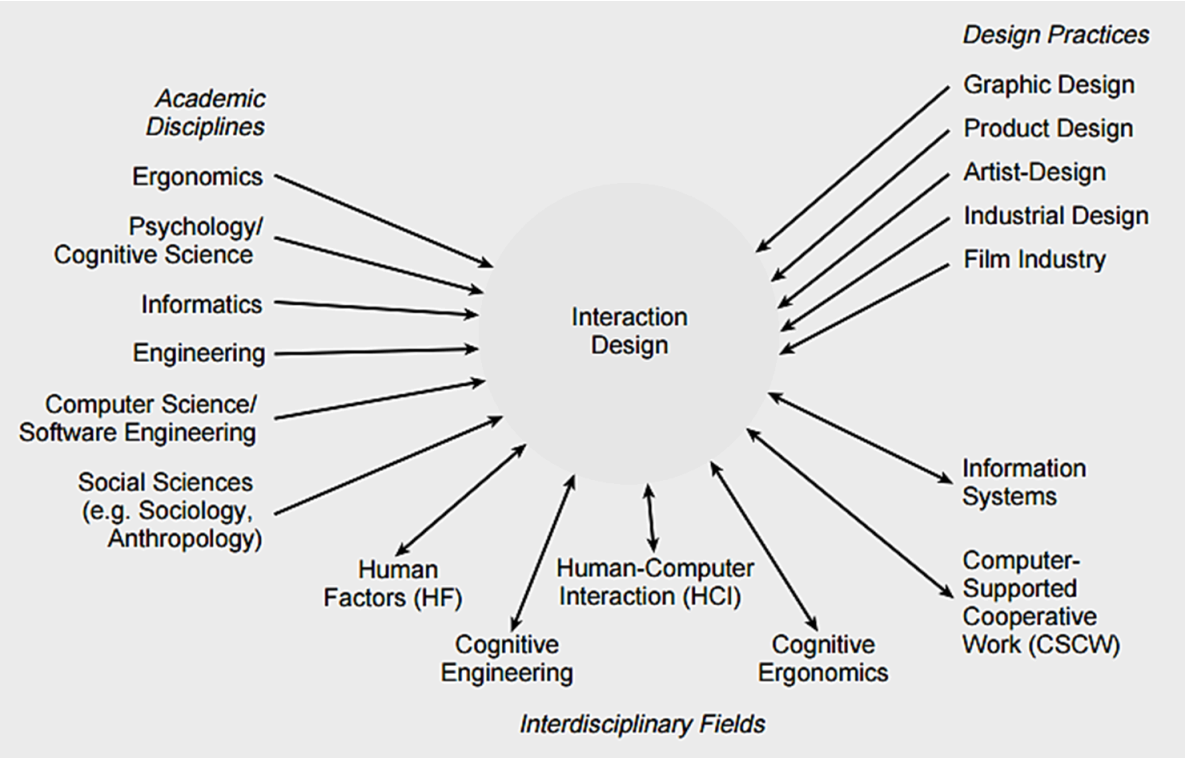
The Verge

There are two big changes to pick apart. First, there are two people replacing Ive, not one. And second: they report to the COO, not directly to Tim Cook. That is precisely the opposite of how Steve Jobs had set up Jony Ive at Apple. Here's how Jobs himself described Ive's role:

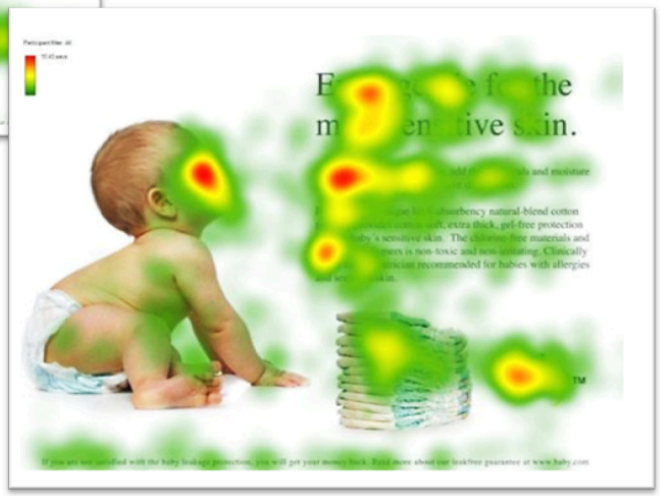
> He's not just a designer. That's why he works directly for me. He has more operational power than anyone else at Apple except me. There's no one who can tell him what to do, or to butt out. That's the way I set it up.

Hard Truths about Product Design

- There's no simple formula or recipe
- It's not love at first sight
- It's not what you would want for yourself
- It's not commonsense or intuitive
- You have to love being wrong
- Small things make a big difference
- You can’t do design w/o numerous disciplines
- Any single discipline can ruin the design
- False **velocity** ruins design culture

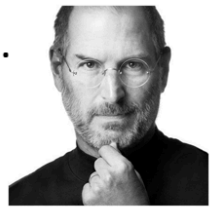


Small, subtle, counter-intuitive changes... big impact



Jared Spool: +$300M / year
Ecom revenue gain from a minor button design change

Both equally important, not equally difficult.



Visual Design

Interaction Design

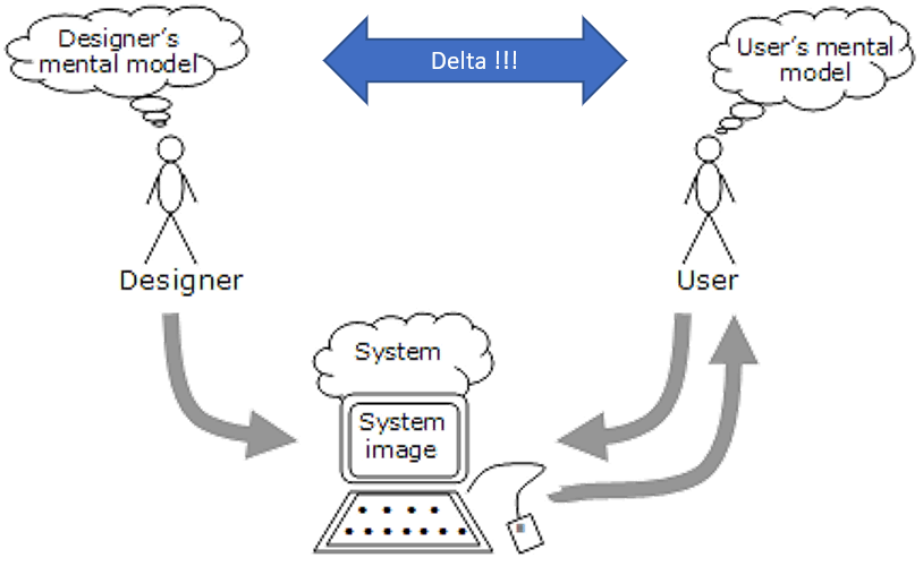
"Don't let the noise of others' opinions drown out your own inner voice."

"You can't just ask customers what they want and then try to give that to them. By the time you get it built, they'll want something new."

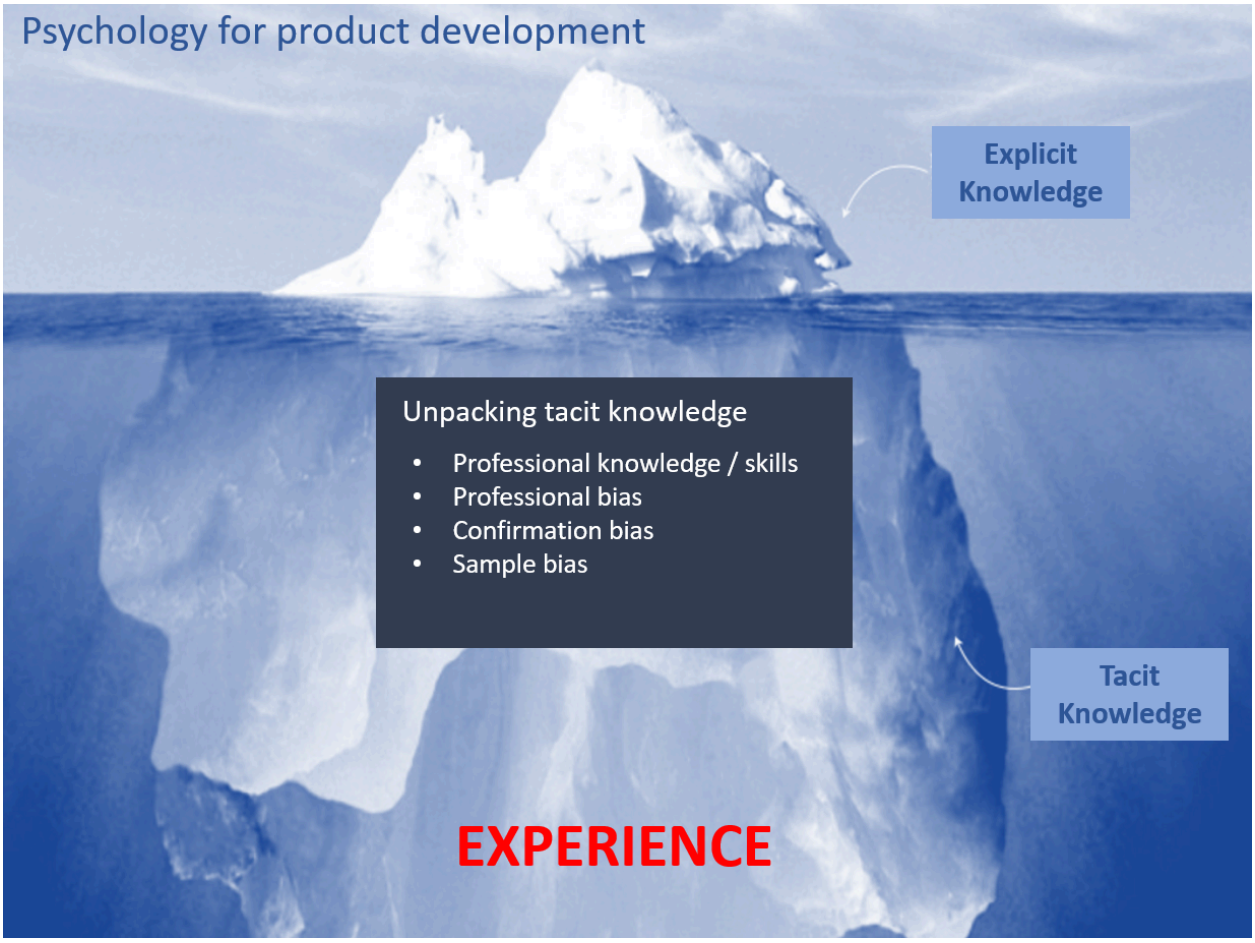
"People think focus means saying yes to the thing you've got to focus on. It means saying no to the hundred other good ideas that there are. You have to pick carefully."

Most people make the mistake of thinking design is what it looks like. People think it's this veneer—that the designers are handed this box and told, "Make it look good!" That's not what we think design is. It's not just what it looks like and feels like. Design is how it works.[6]

Don Norman Mental Model



Psychology for product development



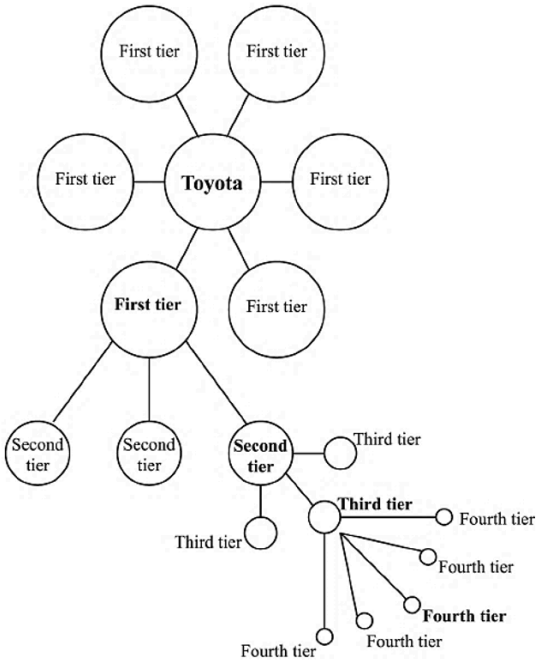
Toyota Process is about extended networks of trusted expertise

TPS is a learning system.
The biggest thing most people overlook about TPS is that it is a system designed for learning. This focus on learning and continuous improvement occurs at multiple levels.

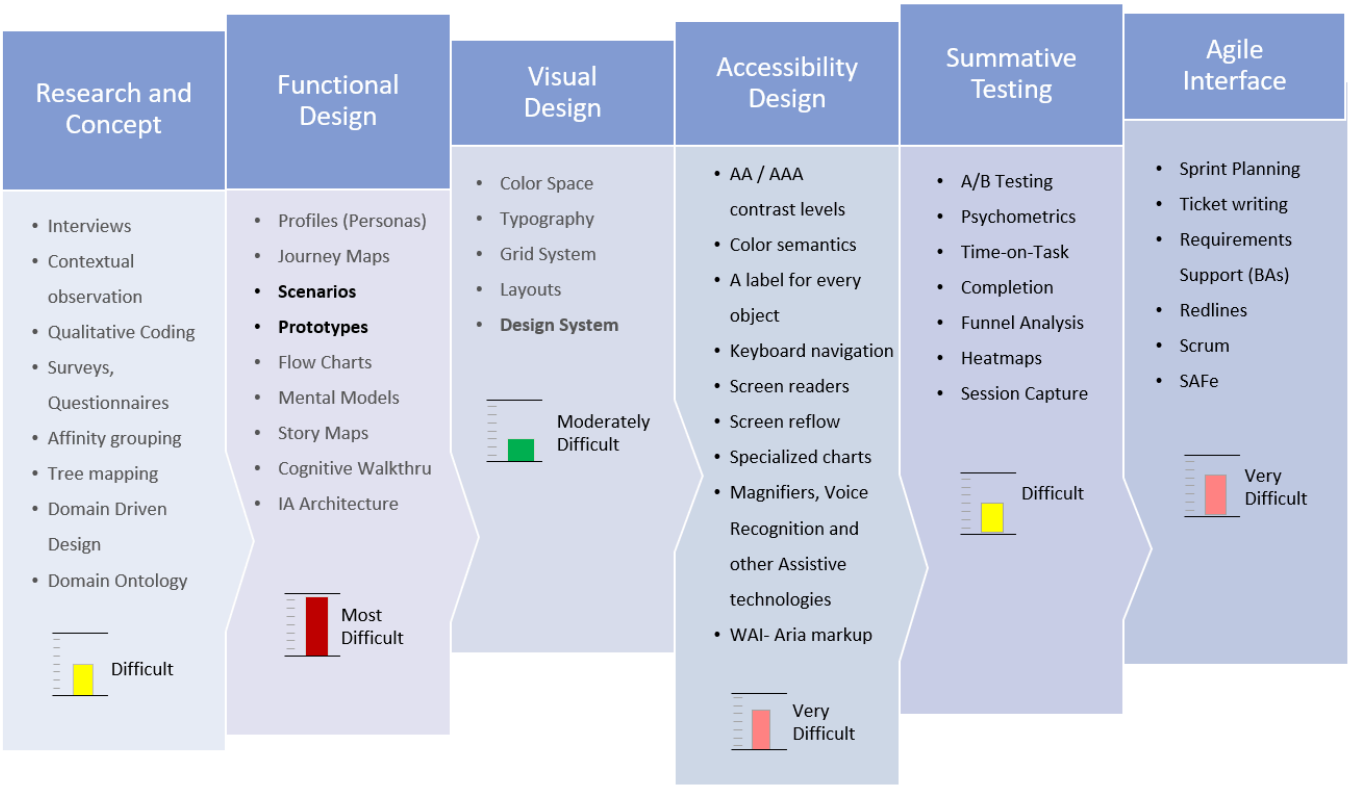
Trust and supplier support are core principles of TPS. This means relationships should be built around long-term learning partnerships and should not be transactional in nature.

When a suppler encounters problems, Toyota offers to help them address them, they are involved in continual co-design.

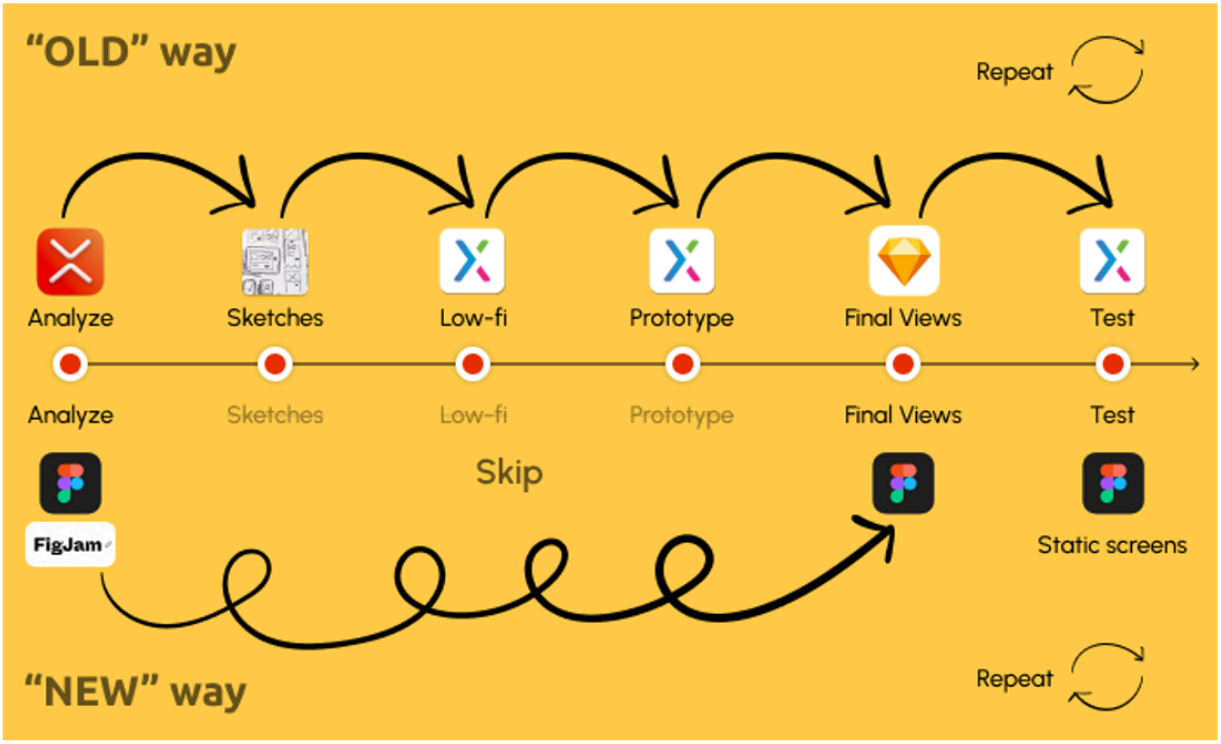
https://hbr.org/2022/11/what-really-makes-toyotas-production-system-resilient



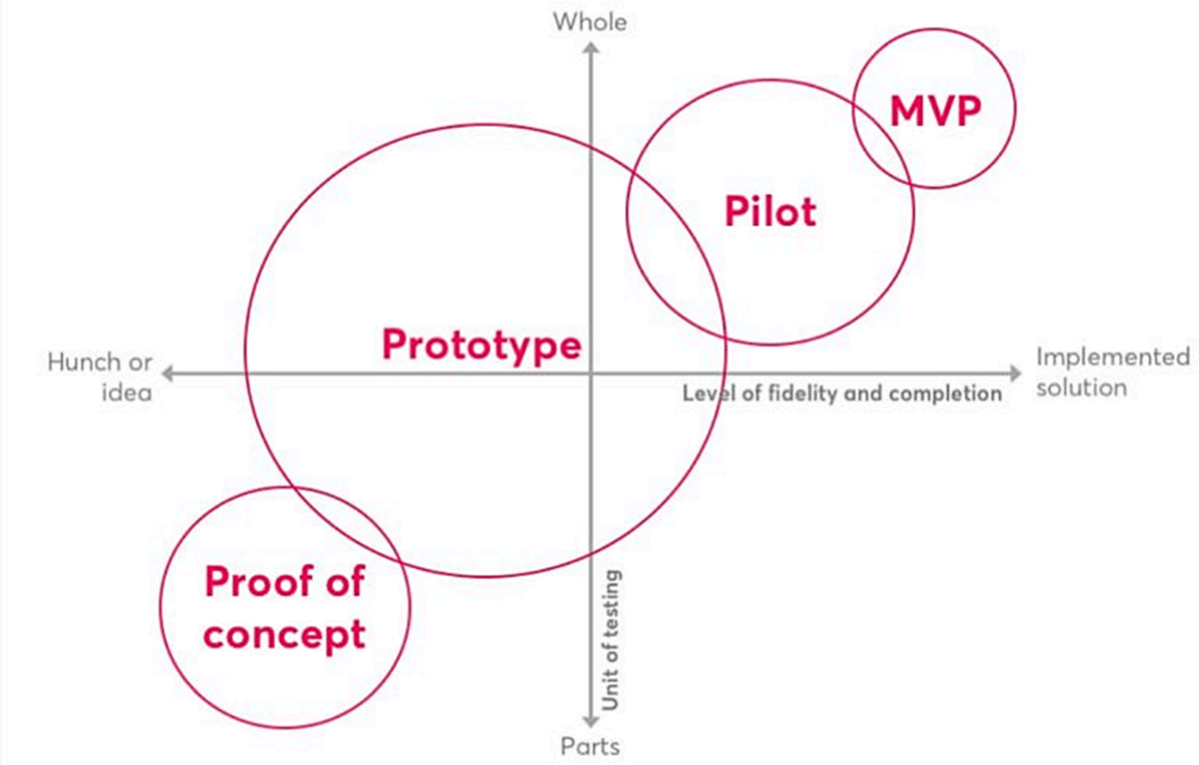
Source: Ford, D. (2002), *The Business Marketing Course: Managing in Complex Networks*, Wiley & Sons, Chichester, p. 31

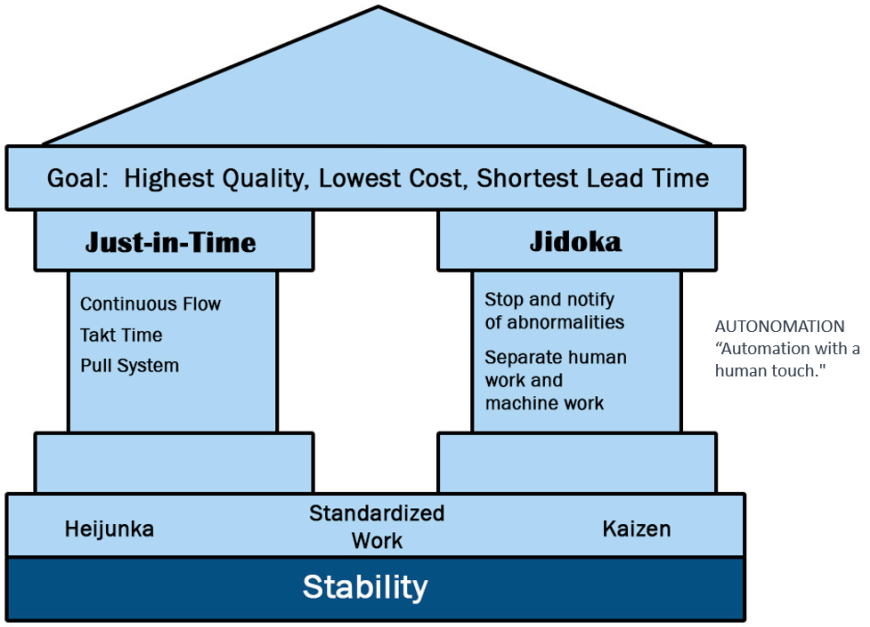


The Figma Syndrome



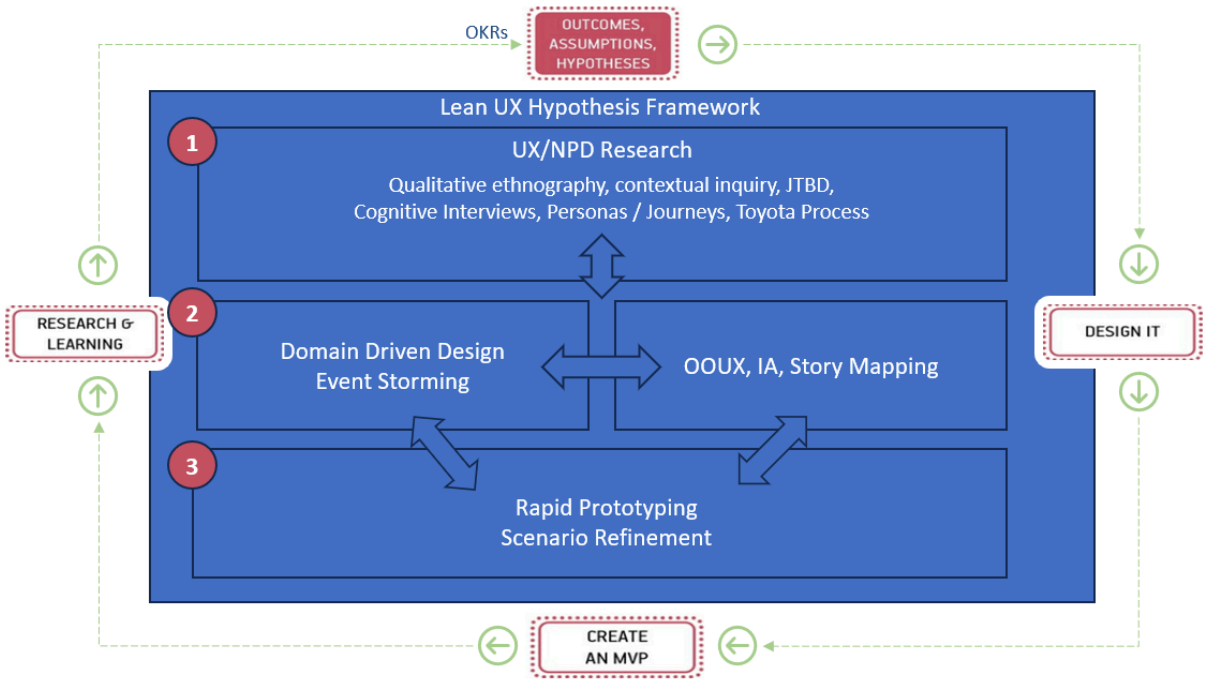
For complex apps with branching workflows the "OLD" way is better !





Value Stream Mapping (VSM): visualize and analyze the flow of materials and information required to bring a product from order to delivery. It is designed to help organizations identify waste, reduce process cycle times, and implement process improvement. (Muda, Kaizen, Mieruka)

Integrated Agile / UX / Business Design Methodology



A design system methodology synthesis by Steve King: JTBD, LeanUX, OOUX and Agile DDD/USM, etc. (Ulwick, Norman, Nielsen, Lewis, Cooper, Sauro, Albert, Tullis, Evans, Young, et. al.)

Works cited

1. Software Project Failures: Why 70% Miss the Mark - Callibrity, accessed March 29, 2026, https://www.callibrity.com/articles/why-software-projects-miss-the-mark
2. Two and a Half Decades of Project Failure - Sourcing Innovation, accessed March 29, 2026, https://sourcinginnovation.com/wordpress/2024/10/25/two-and-a-half-decades-of-project-failure/
3. Chaos Report — why this study about IT project management is so unique, accessed March 29, 2026, https://thestory.is/en/journal/chaos-report/
4. Digital Transformation Failure Rate 2025 - Why 70% of Projects Still Fail, accessed March 29, 2026, https://blog.meltingspot.io/why-digital-transformation-projects-fail/
5. IT Project Failure Rates: Facts and Reasons | Faeth Executive Coaching, accessed March 29, 2026, https://faethcoaching.com/it-project-failure-rates-facts-and-reasons/
6. Management [PDF] [7lj1rd0srdk0] - VDOC.PUB, accessed March 29, 2026, https://vdoc.pub/documents/management-7lj1rd0srdk0
7. The case against corporate short termism | McKinsey, accessed March 29, 2026, https://www.mckinsey.com/mgi/media-center/the-case-against-corporate-short-termism
8. Short-termism in business: causes, mechanisms and consequences - EY, accessed March 29, 2026, https://www.ey.com/content/dam/ey-unified-site/ey-com/en-pl/services/economic-analysis-team/documents/ey-short-termism_raport.pdf
9. 2024FY - Capital Budget Analysis - ZA00* - Miscellaneous Grant Programs - Maryland, accessed March 29, 2026, https://mgaleg.maryland.gov/pubs/budgetfiscal/2024fy-budget-docs-capital-ZA00-Misc

ellaneous-Grant-Programs.pdf
10. Jobs With an Organizational Leadership Degree: Career Paths & Salary Potential - LAPU, accessed March 29, 2026, https://www.lapu.edu/post/jobs-with-an-organizational-leadership-degree-career-paths-salary-potential
11. The Hidden Costs of "Cheap" Software Development - Medium, accessed March 29, 2026, https://medium.com/@kodekx-solutions/the-hidden-costs-of-cheap-software-development-ff11a78c1785
12. How Much Does Software Development Cost: 2026 Breakdown - Cleveroad, accessed March 29, 2026, https://www.cleveroad.com/blog/software-development-cost/
13. Using Hexawise is One of the Highest ROI Software Development ..., accessed March 29, 2026, https://hexawise.com/posts/using-hexawise-is-one-of-the-highest-roi-software-development-practices-in-the-world-according-to-recent-report
14. How to Ensure a Positive ERP Experience | 10X ERP Blog, accessed March 29, 2026, https://10xerp.com/blog/positive-erp-experience
15. What return on investment can I expect from a business mentor? - Matt Brookfield, accessed March 29, 2026, https://mattbrookfield.co.uk/what-return-on-investment-can-i-expect-from-a-business-mentor/
16. Software Development Cost Breakdown In 2026 - Tech JDI, accessed March 29, 2026, https://techjdi.com/blog/software-development-cost-breakdown
17. The CTO's Balancing Act: Technical Debt vs Innovation in the Fast-Paced Technology Landscape - NStarX, accessed March 29, 2026, https://nstarxinc.com/blog/the-ctos-balancing-act-technical-debt-vs-innovation-in-the-fast-paced-technology-landscape/
18. What explains the dramatic shift in dev culture from the relaxed wlb-focused 2010s to what we have today? : r/ExperiencedDevs - Reddit, accessed March 29, 2026, https://www.reddit.com/r/ExperiencedDevs/comments/1s5nkcb/what_explains_the_dramatic_shift_in_dev_culture/
19. MEASURING THE ECONOMIC IMPACT OF SHORT-TERMISM - McKinsey, accessed March 29, 2026, https://www.mckinsey.com/~/media/mckinsey/featured%20insights/long%20term%20capitalism/where%20companies%20with%20a%20long%20term%20view%20outperform%20their%20peers/mgi-measuring-the-economic-impact-of-short-termism.ashx
20. Does User-Driven Design Still Need User-Centered Design? - UX Tigers, accessed March 29, 2026, https://www.uxtigers.com/post/user-driven-design
21. Understanding Tradeoffs: Decentralization vs. Usability | by Tanner Philp | HackerNoon.com, accessed March 29, 2026, https://medium.com/hackernoon/understanding-tradeoffs-decentralization-vs-usability-c537ba612a5e
22. Challenging “Best Practices” for CX/UX Qualitative Sample Sizes | by Debbie Levitt, accessed March 29, 2026, https://rbefored.com/best-practices-for-cx-ux-qualitative-sample-sizes-442a6a21d371
23. Product Discovery in the Multitrack of Madness | by Lucio Santana | The Startup | Medium, accessed March 29, 2026, https://medium.com/swlh/product-discovery-in-the-multitrack-of-madness-cfe8632f1962
24. Enterprise UX Design: The Complete Guide to Scaling Complex Products - MindInventory, accessed March 29, 2026, https://www.mindinventory.com/blog/enterprise-ux-design/

25. User experience: ROI and methods to measure your investment in UX, accessed March 29, 2026, https://www.mjvinnovation.com/blog/roi-in-ux/
26. Minimum viable usability testing - Appcues, accessed March 29, 2026, https://www.appcues.com/blog/usability-testing-methods
27. ROI of UX Design: Creating The Business Case For Design Investment - Pencil & Paper, accessed March 29, 2026, https://www.pencilandpaper.io/articles/roi-ux
28. Why 70% of ERP Implementation Efforts Will Fail by 2027, accessed March 29, 2026, https://www.erpabsolute.com/blog/why-erp-implementation-efforts-will-fail/
29. Why Most AI Projects Fail: The Hidden Truths Behind the 70–90% Failure Rate - Blog, accessed March 29, 2026, https://blog.predictap.com/why-most-ai-projects-fail-the-hidden-truths-behind-the-70-90-failure-rate
30. Managing the Consequences of Technical Debt: 5 Stories from the Field, accessed March 29, 2026, https://www.sei.cmu.edu/blog/managing-the-consequences-of-technical-debt-5-stories-from-the-field/
31. What is Technical Debt? Causes, Types & Definition Guide - Sonar, accessed March 29, 2026, https://www.sonarsource.com/resources/library/technical-debt/
32. What is Technical Debt? | IBM, accessed March 29, 2026, https://www.ibm.com/think/topics/technical-debt
33. Reducing Technical Debt: Strategies to Build Scalable, Future-Proof Systems, accessed March 29, 2026, https://www.scalecomputing.com/resources/reducing-technical-debt-strategies-to-build-scalable-future-proof-systems
34. What is technical debt and how to manage it effectively - Zendesk, accessed March 29, 2026, https://www.zendesk.com/blog/technical-debt/
35. The importance of managing technical debt to boost ROI - Vaultinum, accessed March 29, 2026, https://vaultinum.com/blog/technical-debt-and-roi
36. The Hidden Cost of Technical Debt in Software - Scio Consulting, accessed March 29, 2026, https://sciodev.com/blog/the-hidden-cost-of-technical-debt/
37. Opportunity cost of technical debt | TinyMCE White Paper, accessed March 29, 2026, https://www.tiny.cloud/technical-debt-whitepaper/
38. The Real Price of Technical Debt | Article - BlueOptima, accessed March 29, 2026, https://www.blueoptima.com/post/the-real-price-of-technical-debt
39. The Hidden Cost Of Technical Debt In Your IT Environment, accessed March 29, 2026, https://www.zazz.io/blog/cost-of-technical-debt
40. Breaking technical debt's vicious cycle to modernize your business - McKinsey, accessed March 29, 2026, https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/breaking-technical-debts-vicious-cycle-to-modernize-your-business
41. Reduce and Manage Technical Debt - Gartner, accessed March 29, 2026, https://www.gartner.com/en/infrastructure-and-it-operations-leaders/topics/technical-debt
42. Technical debt and its impact on IT budgets - SIG - Software Improvement Group, accessed March 29, 2026, https://www.softwareimprovementgroup.com/blog/technical-debt-and-it-budgets/
43. Tech debt: Reclaiming tech equity | McKinsey, accessed March 29, 2026, https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/tech-debt-reclaiming-tech-equity
44. Technical Debt: Understanding and Approaching It with Axivion Tools - Qt, accessed

March 29, 2026, https://www.qt.io/quality-assurance/axivion/technical-debt
45. The Price of Technical Debt to Enterprises | Quod Orbis, accessed March 29, 2026, https://www.quodorbis.com/the-price-of-technical-debt-to-enterprises/
46. What is Technical Debt? - Harness, accessed March 29, 2026, https://www.harness.io/harness-devops-academy/what-is-technical-debt
47. Transforming Technical Debt into Business Value - Axelerant, accessed March 29, 2026, https://www.axelerant.com/blog/transforming-technical-debt
48. Examples of Technical Debt's Cybersecurity Impact - DTIC, accessed March 29, 2026, https://apps.dtic.mil/sti/trecms/pdf/AD1144728.pdf
49. Security Debt: A Growing Threat to Application Security | Veracode, accessed March 29, 2026, https://www.veracode.com/blog/security-debt-a-growing-threat-to-application-security/
50. The Economics of Software Quality - Pearsoncmg.com, accessed March 29, 2026, https://ptgmedia.pearsoncmg.com/images/9780132582209/samplepages/0132582201.pdf
51. The Economics of Software Quality by Capers Jones, Olivier Bonsignour | eBook, accessed March 29, 2026, https://www.barnesandnoble.com/w/the-economics-of-software-quality-capers-jones/1117355513
52. SOFTWARE QUALITY IN 2011: A SURVEY OF THE STATE OF THE ART - sqgne.org, accessed March 29, 2026, https://sqgne.org/presentations/2011-12/Jones-Sep-2011.pdf
53. Validation testing in software development: What it is, how it works & key examples - Qubika, accessed March 29, 2026, https://qubika.com/blog/validation-testing/
54. The Economics of Software Quality by Capers Jones - The Tester Library, accessed March 29, 2026, https://thetesterlibrary.com/foundation-agile-tester/the-economics-of-software-quality-by-capers-jones/
55. Why These are the Top 10 Manufacturing ERP Systems for 2026, accessed March 29, 2026, https://www.top10erp.org/blog/manufacturing-erp
56. Strategy Study: How Dyson's Innovation Became Its Key To Success, accessed March 29, 2026, https://www.cascade.app/studies/dyson-strategy-study
57. Secrets of The 7 Figure Coach | PDF | Entrepreneurship | Mobile App - Scribd, accessed March 29, 2026, https://www.scribd.com/document/343265608/secrets-of-the-7-figure-coach
58. Jack delosa tim morris, accessed March 29, 2026, https://7432724.fs1.hubspotusercontent-na1.net/hubfs/7432724/Elevate%20Book/Elevate%20Book%20-%20Full%20Digital%20Version%20(PDF).pdf
59. The hidden drag, quantified: Technical debt's penalty on value and growth - Deloitte, accessed March 29, 2026, https://www.deloitte.com/us/en/insights/topics/technology-management/technical-debt-impact.html
60. Why don't more startups position their software development team in foreign countries to lower costs? - Reddit, accessed March 29, 2026, https://www.reddit.com/r/startups/comments/vs7k9l/why_dont_more_startups_position_their_software/
61. Toyota Production System | Vision & Philosophy | Company | Toyota Motor Corporation Official Global Website, accessed March 29, 2026, https://global.toyota/en/company/vision-and-philosophy/production-system/
62. The Toyota Way in Services: The Case of Lean Product Development. - Conformable Decoders, accessed March 29, 2026, https://conformabledecoders.media.mit.edu/courses/2023/decoders2.0/Papers/David%20

Sadat/The%20Toyota%20Way%20in%20Services_The%20Case%20of%20Lean%20Product%20Development.pdf
63. Integrating toyota production system for sustainability and competitive advantage in medical device software design - OAE Publishing Inc., accessed March 29, 2026, https://www.oaepublish.com/articles/gmo.2024.051401
64. What is kaizen and how does Toyota use it? - Toyota UK Magazine, accessed March 29, 2026, https://mag.toyota.co.uk/kaizen-toyota-production-system/
65. Kaizen: The Toyota Way to Continuous Improvement - Businessmap, accessed March 29, 2026, https://businessmap.io/lean-management/improvement/what-is-kaizen
66. Software, Toyota, Agile. How Did Toyota's Industrial Leadership... | by Goren Acay - Medium, accessed March 29, 2026, https://medium.com/@acaygoren/software-toyota-agile-23b331d274b6
67. Engineering Excellence | Roland Berger, accessed March 29, 2026, https://www.rolandberger.com/en/Expertise/Solutions/Engineering-Excellence.html
68. Rethinking Productivity in Software Engineering - OAPEN Library, accessed March 29, 2026, https://library.oapen.org/bitstream/id/6efc52ab-8e07-4c19-9190-28892c67eec2/1007322.pdf
69. (PDF) The Mythical 10x Programmer - ResearchGate, accessed March 29, 2026, https://www.researchgate.net/publication/332927042_The_Mythical_10x_Programmer
70. Succeeding with OKRs in Agile – Allan Kelly, accessed March 29, 2026, https://www.allankelly.net/books/succeeding-with-okrs-in-agile/
71. Succeeding with OKRs in Agile - Leanpub, accessed March 29, 2026, https://leanpub.com/agileokrs
72. 3 Questions to ask when adding OKRs to Agile | by Allan Kelly - Medium, accessed March 29, 2026, https://medium.com/@allankellynet/3-questions-to-ask-when-adding-okrs-to-agile-8e70b94e0e4e
73. Build your tech and balance your debt - Accenture, accessed March 29, 2026, https://www.accenture.com/content/dam/accenture/final/accenture-com/document-3/Accenture-Build-Your-Tech-and-Manage-Your-Debt-2024.pdf