1

And Why Do I Care?

Technology. It's everywhere. In our homes, in our hands, running our corporations, healthcare and banking systems, militaries, and the very apparatus of government itself. Simply put, "*Our civilization runs on software*."[1] We are therefore very fortunate that there are a great many organizations and millions of individuals across the globe who work tirelessly to create products and services that do actually serve us well. Sometimes they do this against the odds, battling with aging or "legacy" systems to create the new features that consumers desire but which the original designers did not envisage. Yet, despite these great efforts on the part of so many, software-based technologies are increasingly failing us, with cybercrime, global system outages, and economic competitiveness becoming the norm rather than the exception. These issues stem from ever-greater consequences and implications for us all. However, you may well think, "*Well . . . it's not my problem to solve, so why should I care about it?*" After all, it's not an issue on the scale of, say, solving world hunger, right? Wrong. In financial terms, it's much, *much* bigger than the cost of solving world hunger, according to UN figures. Let's delve into this a bit.

As we shall learn, there is a lot of aging software that currently runs many of the systems that modern society relies upon. So much so that the aforementioned quote by Bjarne Stroustrup, creator of the widely used C++ language, which has an extensive and enduring presence across critical software systems worldwide, could reasonably be

modified to state that "*Our civilization runs on aging software*." Not all aging software is "bad" per se, but running, updating, or improving older or "legacy" technologies can come at a significant cost, both financial and otherwise. Modernization of these technologies, sometimes expressed in terms of legacy modernization or digital transformation, is an imperative for most governments and corporations around the developed world, at least, since they have built up the largest estates of such legacy technologies and therefore face the biggest transformation challenges. We have chosen to use the term Technology-Led Transformation (TLT) throughout this book to describe change programs where digital technology plays a significant role—whether modernizing legacy systems or digitizing manual processes. This doesn't mean such initiatives must be led by the technology function; rather, TLT refers to any transformation where technology is central to the envisaged change. However, as multiple studies conducted across many countries and industries have shown, between 70% and 88% of all transformation efforts fail to deliver on their original goals. Digital transformation initiatives of any kind have the lowest success rates of all, as evidenced by the Bain survey involving 1,000 companies around the world to gauge their level of digital readiness. They found that only 5% of those undertaking some form of digital transformation initiative had achieved or exceeded the expectations they set for themselves, compared to a success rate of 12% for conventional transformations (Figure 1.1).[2]

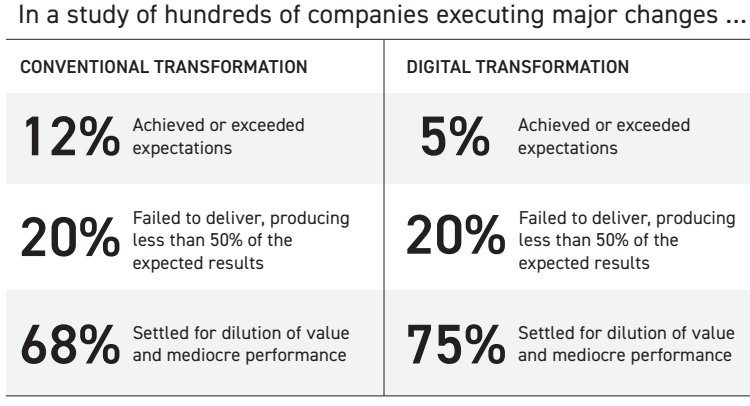

Figure 1.1 Conventional vs Digital Transformation.

Source: Bain risk history survey 2017 (n = 403); Digital 360 Barometer survey 2017 (n = 1012)

A full 75% of these companies settled for dilution of value and mediocre performance, and 20% failed outright.

The range of outcomes recorded by other consultancies, including BCG and McKinsey, split broadly by total successes, partial results, and out-and-out failures, does vary somewhat, but the de facto success rates of between 5% and 30% are poor rates of return on stakeholder resources, regardless of how the overall program is viewed.

Table 1.1 summarizes the data published by three of the world's top management consulting firms. As you can see, the track record is not good. Furthermore, these figures relate to the types of companies and governments that provide us with the bulk of the products and services we rely on to live, thrive, or survive.

Table 1.1 Reported Success and Failure Rates: Sample Bain & Company, BCG, and McKinsey & Company.

Source	What Is Measured	Achieved or Succeeded	Partial Results	Failed
Bain & Company (2017)	Success of digital transformation initiatives	5% achieved or exceeded expectations	75% settled for dilution of value and mediocre performance	20% failed to deliver, producing less than 50% of the expected results
BCG (2020)	Success of digital transformation projects	30% met or exceeded their targets and resulted in sustainable change	44% created some value but did not meet their targets and resulted in only limited long-term change	26% created limited value (less than 50% of the target), producing no sustainable change

(*continued*)

Table 1.1 (Continued)

Source	What Is Measured	Achieved or Succeeded	Partial Results	Failed
McKinsey & Company (2018)	Success of overall digital transformation	16% have successfully improved performance and also equipped them to sustain changes in the long term	7% performance improved but improvements were not sustained	77% performance did not improve*

* Implied, but not stated

Source: https://www.einst4ine.eu/digital-transformation-success-and-failure-part-i-insights-from-industry-and-grey-literature/

1.1 Time, Please

IT systems are composed of multiple interconnected technological components that must function together as a whole. These components typically have interdependencies, whereby component A is reliant upon components B, C, and D to function well before it can function as designed. A failure in one component can lead to cascading issues across the whole system. This fact introduces complexity and increases the risk of delays and errors during the build, implementation, testing, and integration stages of a project. This problem is particularly acute when components are tightly coupled, as tends to be the case with legacy systems. In contrast, more modern, typically "decoupled" systems mitigate such risks by isolating failures. This interdependence plays a key role in explaining why many IT projects experience significant time and cost overruns. One study of 106 software projects conducted within a single organization found that, on average, projects took 200% longer than estimated, with the worst

case running nearly 700% over time.[3] The data indicated that large overruns are more frequent than the company anticipated. Other studies support this conclusion, with one analysis of 72 projects across 23 organizations identifying cost overruns of up to 525% that were much more heavily skewed toward significant overruns than had been expected. The U.S. Department of Defense reports that for the fiscal year ending in 2020, IT project spending was $37 billion, and only 35% of the projects were within budget.[4] A study by McKinsey and the BT Centre for Major Programme Management at the University of Oxford reports that, on average, large IT projects run 45% over budget, with some projects overrunning cost—presumably time budgets—by as much as 400%.[5]

One of the most comprehensive studies in this space analyzed 5,392 IT projects worth a combined $56.5 billion, measured in terms of 2015 USD.[6] It found that IT project cost overruns follow a power-law distribution, meaning that while most projects experience small overruns, a few projects incur extremely large overruns, disproportionately driving up average costs and making such extreme events more common than expected under normal distributions. The findings challenge the implicit managerial assumption that overruns follow a normal distribution. This, in turn, can dangerously underestimate the likelihood and cost of large-scale failures. The study conclusively demonstrates that interdependencies among technological components cause these extreme overruns. The key takeaway is that IT projects are far riskier than traditionally assumed, and accurate risk assessment must account for the possibility of rare but devastating overruns. So if leaders assume a normal distribution of IT project cost overruns, they may be unwittingly exposing their organizations to extreme risk by severely underestimating the probability of significant cost overruns.

According to Gartner, the global technology research and advisory firm, of the $5 trillion spent globally on IT in 2024, the largest proportion of that, at $1.55 trillion, was spent on technology services.[7] This aligns closely with the data from Statista of $1.51 trillion (£1.18 trillion) for 2025.[8] Taking a conservative position, we will work with these lower figures. As a point of reference, by far the largest component of Global IT expenditure is accounted for by the US with 36.4%.

The UK accounts for 7.5% of the total and the expenditure per capita is similar in both countries at around $1,642 per person per year; this, however, is nearly nine times the global average. So, countries such as the UK, US, and other developed nations have the most to gain by getting this right.

In a typical year, the proportion of IT expenditures allocated by organizations to transformation activities—sometimes referred to as Change The Business (CTB) initiatives, which per the Gartner definition includes both "Grow" and "Transform" activities but excludes "Run" costs—can be as low as 30% and as high as 80% during a major technology transformation, if only at the departmental or business unit level. However, we will work with the lower end of this range.

Estimates for the level of wasted expenditure in the area of TLT range from 50% to 80%. Again, we will work with the lower end of that estimate.[9] This implies a *conservative* estimate for wasted IT-related transformation expenditure is at least $226 billion every year, and it might actually be as much as three times this figure, depending upon the assumptions used.

Furthermore, on average, organizations can expect to spend up to 70% of their change budget on people factors that lie outside of pure technology and process-related CTB activities.[10] This implies a figure of around $3 of additional expenditure for every $1 of direct technology-related investment. Combined, these amount to a wastage level associated with TLT initiatives of over $900 billion every year, at the low end. At the higher end, the estimate is over $2.4 trillion, which is greater than the projected GDP for all but the top eight economies of the world in 2025.[11] Yes, we really could be solving world hunger instead.

There are approximately 8.2 billion people on Earth, and 828 million of them, or 1 in 10, are hungry, according to figures produced by the UN World Food Programme (UNWFP). Of those, 50 million are on the brink of famine and desperately need our help. According to a statement made by the UNWFP's Executive Director, David Beasley, in 2021, it would take an estimated $40 billion each year to end world hunger by 2030, so nearly $400 billion in total over 10 years.[12] For comparison, the figure quoted by Ceres2030, a coalition funded by the German government and the Bill and Melinda Gates Foundation in

2020, was closer to $330 billion (£253 billion) over 10 years, or $33 billion per year.

Taking the higher figure of $40 billion each year [see table in appendix entitled "*Calculating the True Cost of Wasted Technology-Led Investment*"], just one year of wasted technology-led transformation (TLT) expenditure, when combined with the ancillary non-technology costs described above, could fund world hunger solutions for over 22 years! Yes, folks, you read that correctly; 22 years of funding solutions to world hunger gets "wasted" each and every year, by corporations and governments around the world in their efforts to create TLT. Or putting it another way, using our lower figure, just 4% of that annual wastage could theoretically fund the UN World Food Programme's annual hunger-eradication program. Alternatively, just one year of that wasted TLT expenditure could fund 12 years of the $74 billion required each year to fund free, basic, universal healthcare in poor countries.[13]

Furthermore, this wastage does not take account of the opportunity cost of that expenditure. By opportunity cost, we mean the value of the next best alternative that you give up when you make one choice in favor of another. Put simply, it's what you miss out on by investing this much in TLT instead of something else that could provide a greater return on that enormous level of investment. If this sum were to be invested in initiatives that are only half as wasteful as the current TLT programs, organizations could free up at least $1 trillion a year for other initiatives. The lower figure $900 billion of waste is one-third of the $2.53 trillion that the world spent in 2024 on Research & Development (R&D) across *every* industry sector.[14] So, halving that waste could add nearly 20% to global R&D investment into life-enhancing capabilities, such as improved gene therapies, renewable energy solutions, sustainable agriculture, quantum computing, and education development technologies. Remember, these are all very conservative figures, and the true level of IT-related wastage could be as much as three times the aforementioned figure. Such enormous financial wastage has potentially massive consequences for humanity as a whole; millions of lives could be saved, billions of lives could be improved, and both corporations and government could do much more with less. But wait, there's more. Much more.

1.2 The Technical Debt Crises

Most of us have heard the term "debt crisis." A debt crisis happens when a country, a business, or sometimes an individual has borrowed so much money that it can't repay its debts on time or perhaps at all. It usually triggers serious financial and economic problems that can range from currency devaluation, inflation, or the imposition of austerity measures through to recession or even economic depression. Debt crises are generally "no bueno" for anyone involved. However, you may not be aware of the equivalent situation in the world of software engineering, which is termed "Technical Debt," or TD. This is a software development metaphor that refers to the cost of choosing a quick or easy technical approach or solution now instead of a better, more time-consuming one, which will in due course require "repayment" in the form of refactoring, fixing, or extra work later on.

TD is one of those invisible issues that organizations either realize that they have a problem with—and so can plan to do something about it—or they don't know, and that's worse. It generally arises because it's faster, cheaper, and easier to put things off until a future date, rather than dealing with them now, but refactoring the relevant code. *Refactoring* is the term used in software when restructuring of existing code is undertaken without changing its external behavior. The goal is to improve the code's internal structure, making it cleaner, more readable, more maintainable, and often more efficient while ensuring it still performs the same functions.

Refactoring is like paying interest, whereby you make small, consistent, and continuous improvements to prevent TD from growing in quantity or seriousness. Simply put, TD is the extra work you'll have to do in the future because you took shortcuts today. It is analogous to the idea of borrowing today to spend tomorrow and end up paying for that choice for many years to come. TD is often created as a byproduct to meet a deadline, perhaps by hardcoding something instead of building a flexible, reusable code module. That could be termed "deliberate" technical debt; it works for now, but when requirements change later, it's a pain to update, and that pain is your technical debt coming due. "Accidental" technical debt, on the other hand, is debt that accumulates through knowledge gaps, poor practices, or lack of

documentation. This type typically carries higher long-term costs. There are signs when TD starts to become an issue in an organization. These include, but are not limited to, the following:

- New features take significantly longer than expected to deliver.
- Changes made to one area create bugs in unrelated parts of the system.
- Developers have to avoid touching certain parts of the code for fear of breaking things.
- Onboarding new team members is becoming painful.
- "Quick fixes" snowball into major issues.
- Test coverage, by which we mean the footprint of software tests, is poor or outdated.

Essentially, when the cost of change rises and momentum slows, technical debt may be to blame. However, TD is a natural byproduct of any iterative, evolutionary process like software development, and it's not inherently a bad thing. In fact, it can be a strategic trade-off and a part of the cost of doing business when teams want to deliver features to get early and frequent user feedback before investing further in that area of the system. The real problem arises when technical debt is left unaddressed over time, allowing it to accumulate unchecked. Like financial debt, it can grow to a point where the "interest"—the extra effort needed to work around it—becomes overwhelming. Eventually, you may find yourself spending more time managing the consequences of the debt than delivering new value. At that point, repayment can feel impossible, and the situation escalates into a full-blown crisis with systems failing either partially or completely.

1.3 How Big Is the Debt?

This debt is staggeringly large in actual fact. If we broaden the definition of TD to include the accumulated cost of maintaining and reworking outdated or suboptimal technology implementations, which are themselves a symptom of that debt, and poor software quality in general, which may be a cause of TD or a consequence of it, or both, the annual cost to organizations of all types and sizes is around $2.41

trillion according to The Consortium for Information & Software Quality™ (CISQ™). Its 2022 report states that ". . . finding and fixing bugs is the largest single expense component in a software development lifecycle."[15] Furthermore, the accumulated software technical debt is estimated to be $1.52 trillion in the US alone. Industry analysis suggests that global TD has roughly doubled over the past decade, growing by about $6 trillion between 2012 and 2023.[16] This in turn implies a cumulative TD of approximately $12 trillion total by 2023. This enormous "principal" of technical debt acts as a drag on innovation and efficiency, like barnacles and weeds on the bottom of a sailboat. Eventually, such a boat simply cannot sail fast in any direction and ends up being buffeted by winds and tides. Organizations are paying a hefty "interest" on this debt. For example, many companies spend at least 40% of IT budgets just to maintain legacy systems and debt instead of adding new value.[17] It is worth noting that a proportion of the TLT investment that we outlined above is currently being undertaken by organizations specifically to reduce the level of TD within their estates, with varying degrees of success. A Total Economic Forum study by Forrester showed that retiring old legacy systems could reduce combined hardware and operational running costs by 65%, so the rewards for getting it right can be significant.[18]

TD can encompass changes in debt incurred during the building of new systems, using modern approaches and technologies in addition to the debt associated with legacy technologies, many of which have been in existence for decades. There's no formal definition of "legacy system," but it's commonly understood to mean a critical system that is out of date in some way. It may be unable to support future business operations; the vendors that supplied the application, operating system, or hardware may no longer be in business or support their products; the system architecture may be fragile or complex and therefore unsuitable for upgrades or fixes; or the finer details of how the system works are no longer understood. In effect, the entirety of any legacy system, which the owners would like to be able to make rapid and frequent changes to but cannot due to one or more of those reasons, could be considered to be TD. So the estimates for global TD levels include, but are not limited to, legacy IT estates, and so the terms are sometimes used interchangeably.

Developers waste approximately 33% of their time dealing with technical debt rather than building new features, according to the Stripe Developer Coefficient Study, which surveyed over 1,000 developers and more than 1,000 C-level executives.[19] Organizations with high technical debt deliver new features 25–50% slower than competitors with well-maintained codebases, and high-debt codebases typically experience 2–3 times more production bugs than well-maintained systems.[20] Taking all of these factors into account, Oliver Wyman estimates that the proportion of IT spend that is consumed by TD is around 18%. Given that this applies to the combined total annual IT services expenditure and includes the non-IT staff involved in TLT, the projected net impact of TD each and every year is an additional $1 trillion or enough to fund world hunger programs for an *additional* 25 years, with every year that passes.

Table 1.2 summarizes the technical debt burden by domain with rough order-of-magnitude estimates.

Table 1.2 Estimated Global Technical Debt by Domain in USD Which Approximates to the "Principal," Together with Associated Annual "Interest" Costs and One-Time Remediation Costs. Industries are Examples with Especially High Debt in Each Domain.

Domain	Size of Tech Debt (USD)	Annual Wasted Cost (USD/yr)	Estimated Remediation Cost (USD)	Key Industries with High Debt
Backend (core apps, legacy engines)	~$5–6 **Trillion**	~$1.0–1.2 **Trillion**	~$5 **Trillion+**	Finance, Government, Telecom, Retail
Data Infrastructure (databases, pipelines)	~$2–3 **Trillion**	~$0.5–0.6 **Trillion**	~$2–3 **Trillion**	Finance, Healthcare, Tech

(*continued*)

Table 1.2 (Continued)

Domain	Size of Tech Debt (USD)	Annual Wasted Cost (USD/yr)	Estimated Remediation Cost (USD)	Key Industries with High Debt
DevOps & Infra (CI/CD, cloud, IT ops)	~$1.5 **Trillion**	~$0.3 **Trillion**	~$1.5 **Trillion**	Government, Transportation, Manufacturing
Security (vulnerabilities, legacy security tech)	~$1 **Trillion**	~$0.3 **Trillion**	~$1 **Trillion**	Healthcare, Finance, Government
Frontend (UI frameworks, clients)	~$1 **Trillion**	~$0.1–0.2 **Trillion**	~$1 **Trillion**	Retail, Banking, Public Sector

1.4 What Does This Mean for Me?

If you are the CTO, CIO, or CFO of a large organization, this should mean a lot to you. But if, like most of us, you are simply a taxpayer or a customer of an organization that relies upon products or services that in turn rely upon systems with extensive TD to provide those services, you may be in for some unexpected and unwanted surprises in the years ahead, if indeed you need to wait that long. In his landmark study *The Shock of the Old: Technology and Global History Since 1900* [Oxford University Press, 2007], British historian David Edgerton claimed that although maintenance and repair are central to our relationship with technology, they are "*matters we would rather not think about*." As a result, technology maintenance "*has lived in a twilight world, hardly visible in the formal accounts societies make of themselves*." Indeed, the very invisibility of legacy IT is a kind of testament to how successful these systems are. Except, of course, when they're not.

Worse still, while the financial cost is absurdly high, the human costs can be even higher. In just in the last few years, faulty air traffic control systems in the United States and across Europe have caused significant flight delays and cancellations at peak times, while public sector purchases of software systems that are supposed to improve our lives often result in staggeringly expensive failure. Airplanes running poorly tested software have killed hundreds of people, and our personal data has been compromised or stolen by nefarious foreign governments and independent hackers alike with total impunity. Perhaps the worst example of a TLT program that sadly ticks all the wrong boxes is the UK's Post Office's Horizon accounting system debacle. This led to more than 900 Post Office employees being prosecuted for stealing money they didn't in fact take. Even more tragically, the harassment of the accused resulted in some of those individuals taking their own lives.[21] Yet not a single person has been called to account for these failings, and the many potential lessons that could be derived from this sorry affair have largely been lost in the media noise surrounding the case.

The way in which good technology gets designed and delivered by organizations should not be viewed as a form of black magic or an impenetrable black box or just "someone else's problem." It is a result of good people who are willing to take responsibility for making better choices that benefit the many, not the few, and then who put these decisions into action, supported by robust governance frameworks that are built to serve all relevant stakeholders, not just a privileged few. It can be understood. The associated processes can be constantly refined and improved. Technology teams can improve their productivity and the quality of their outputs by adopting better methods and behaviors. In this way, we can collectively navigate toward creating better outcomes for all concerned. This book explains what those methods and behaviors are, how to adopt them, and the benefits of doing so.

If we are to bring about the necessary change, it pays to improve society's understanding of how things work: specifically, of how the leadership and the respective organizations operate and why, to facilitate a better understanding of how to make things better in the future. This understanding-led approach may not bring about revolution-on-the-streets type change, as was attempted during the storming of the

US Capitol on January 6, 2021. However, there is great value in the gradual and persistent evolution toward a better future for all of us. We are talking about the type of change that increasing levels of education brings about in a developing society, year-after-year as people become more literate, more numerate, and more aware of their rights. Ideally, each successive generation feels more able to tackle the existing bureaucratic system and to bring about the changes that they seek. This book aims to be a single voice among many—a point of reference to guide action, and, with hope, a north star to navigate through uncertainty.

1.5 What's in It for Me?

Let's first qualify what we mean by technology, since there is a lot of it in different forms in any modern society. In this book we are referring primarily to the creation or modernization or reengineering of software and data technologies that run on generic, rather than purpose-made, computer hardware, and which can be designed to deliver new capabilities, products, or services to users. The programmable nature of the modern software and generic hardware combination makes it truly pervasive, by which we mean that it has become the mechanism for controlling almost anything mechanical or electrical, from Tesla cars, to John Deere tractors, to TVs and spaceships.

This future reality was brilliantly articulated in August 2011, when Marc Andreessen, the founder of the graphical browser, wrote a now-famous essay in *The Wall Street Journal* entitled "Why Software Is Eating the World." He argued that software was reshaping industries by automating processes, boosting efficiency, and creating new business models. Companies like Netflix, Amazon, and Google exemplified this trend, disrupting traditional industries and leading the charge in digital transformation. Today, software touches every aspect of our lives, from finance and healthcare to social interactions and national security. With the recent rise of GenAI (Generative Artificial Intelligence), we are entering yet another phase of rapid technological change, reshaping how we interact with people and machines, which will likely lead to the destruction of some industries and the creation of new ones on a scale and at a speed that will be truly breathtaking.

This is precisely why we need to get more things right more often than we currently do, and we need to do so quickly.

GenAI may well be part of the answer regarding how we achieve this, as we shall explore in Section 4. However, it will inevitably also be part of the problem. Criminals and nation-states are already leveraging these technologies to industrialize the creation of fake everything: to spoof identities, hack passwords, socially engineer nefarious outcomes, and generally frustrate the attempts of good people to do good, protect their assets, and their privacy, too. To illustrate this further, one of the first known instances of GenAI deepfake technology being used to impersonate corporate executives in a live video conference was the targeting of Arup, a British engineering and design consultancy. In January 2024, a finance employee at Arup's Hong Kong branch received a phishing email purportedly from the company's UK-based Chief Financial Officer (CFO), requesting participation in a confidential transaction. The employee duly joined a videoconference featuring what appeared to be the CFO and other senior staff members. In reality, these were AI-generated deepfake representations that convincingly mimicked the voices and appearances of actual Arup executives. Trusting the authenticity of the meeting, the employee proceeded to make 15 separate transfers totaling HK$200 million to five different bank accounts over the course of a week. This resulted in the direct financial loss of around $25 million of Arup company funds.

Herein lies the problem; so much of the software and associated working practices that we rely upon today were never built to handle any level of change, let alone *this* level of change. It was built to do whatever the original designers had in mind at the time and no more. It has become legacy software. And yet, change has been literally forced upon such legacy systems to progress, as change has been forced upon them to advance the ambitions of the respective organizations. From banking to aircraft booking, from in-home music controls to the cars we drive, most modern vehicles, craft, devices, and services rely partially or totally on software. And many of those software systems were designed decades ago, when the pace of change was much, much slower, and the scope for delivering competitive advantage primarily through a software technology advantage was relatively rare.

We can divide the problem into a number of technical areas, starting with the "front end," by which we typically mean the way information is presented and interacted with on the screen by users. This in turn is connected to the "back end," which encompasses the core business applications, servers, and enterprise architecture that handle critical transactions, and more recently, the data infrastructure that includes the pipelines, databases, and analytics systems that handle an organization's data. Finally, there is the DevOps and infrastructure technical debt, which refers to the shortcuts and legacy setups in IT environments and the related software deployment processes. DevOps is a blend of the words "Development" and "Operations" and refers to a set of practices, tools, and a cultural mindset that aim to bridge the gap between software development and IT operations. With DevOps, the goal is to enable teams to deliver software faster, more reliably, and to achieve higher-quality outcomes.

However, as Accenture research confirms, "enterprise applications," by which they are primarily referring to the backend of enterprise systems, are the number-one source of technical debt for companies.[22] Consequently, the backend technical debt is by far the largest and most costly category of all, often running on legacy platforms, including mainframes, which utilize defunct computer languages such as COBOL, in addition to outdated versions of more current languages. Such systems often date back many decades; COBOL was first introduced in 1959! As one report notes, there are over 220 billion lines of COBOL still in use, powering approximately 70% of global business transactions.[23] These entrenched systems are difficult to replace, leading organizations to continually patch and extend them, thereby accumulating massive additional TD. Based on the available figures, we estimate backend TD on a global scale to be in the region of $5–6 trillion in potential remediation costs, making up as much as 50% of the total global TD. Globally, this is a multi-trillion-dollar issue.

Most importantly, these issues of ongoing waste and inefficiencies caused by aging frontend interfaces, monolithic backend systems, brittle data infrastructure, manual operational processes, and outdated security postures affect every one of us, to a greater or lesser extent.

It might affect the ability of the company you work for to pay higher wages or to invest in future opportunities that could create new career opportunities for you and your colleagues. It could affect the competitiveness of whole industries and economies. And it certainly affects the percentage of the tax that you pay to your government that is turned into useful services for you and your fellow citizens, because we are talking about sums of money that are material proportions of annual government expenditures worldwide. By way of example, a 2019 US Government Accountability Office (GAO) report found that the US federal government spends more than $100 billion on IT and cyber-related investments annually, of which about 80% is spent on the operations and maintenance of legacy systems.[24] The study identified 10 critical federal information technology (IT) legacy systems, across 10 agencies, that were most in need of modernization. The agencies included the Departments of Defense, Homeland Security, the Interior, the Treasury, the Office of Personnel Management, the Small Business Administration, and the Social Security Administration.

The systems ranged in age from about 8 to 51 years old, with many built in COBOL and several containing known security vulnerabilities or running on unsupported hardware and software. Collectively, these 10 systems cost about $337 million annually to operate and maintain. Only seven agencies had modernization plans, and only the Departments of the Interior's and Defense's modernization plans included all of the basic elements identified in best practices, such as milestones, a description of the work required, together with a plan of the sort necessary to complete the modernization of the legacy system. All of these 10 systems are core to the respective missions of each department; however, these are not isolated examples or handpicked cases made to prove a point. Over 60% of businesses still struggle with integrating legacy tools and platforms with new applications, and 57% report that they lack the business agility necessary to respond to emerging business challenges and opportunities, in part due to these legacy and TD issues.

The fact is that most of these systems were never expected by their original designers to undergo much change during their

lifespan, and their anticipated lifespan was probably much shorter than their actual lifespan turned out to be. However, as Heraclitus, a Greek philosopher who lived around 500 BCE said, "The only constant is change," so these legacy systems have often been tinkered with repeatedly by different people, over different periods, have been made to do things that their original architects could never have envisaged they would be expected to do, and as a result of all this unexpected tinkering, the current caretakers are now hampered by insufficient documentation relating to whatever changes were made in prior years and why those changes were made. Twenty or thirty or even fifty years later, making further substantial change without breaking something major has become next to impossible. As a result, attempts to do so often result in catastrophic outcomes, which in turn lead to terrible experiences by the ultimate beneficiaries of those systems, namely you and me.

Well-designed, modern software is typically built in the form of discrete, interconnected modules, which emphasize maintainability, adaptability, and scalability, supporting rapid product innovation. This is often in stark contrast to many older architectures that are often described as being "monolithic" and "legacy" in nature, and because of this have often accumulated a very high degree of TD. When we say that an architecture is "monolithic," we usually mean that it has a single codebase whereby all aspects of the system are built into a single, often very, very, large lump of code. This encompasses the front end, or the way things look onscreen, the backend, including the business logic that determines the outcomes of processes like calculating the balance of your bank account or the status of your delayed flight, and finally the data that feeds these results, which is usually stored in a database of some sort. So, when we talk about "legacy" systems, we are typically referring to such monoliths, built using outdated software languages and "deterministic" design approaches that assumed everything the system would ever be asked to do could be written into the design documents before coding commenced and would never undergo significant change from that point forward. Combined, this contributes to and may constitute a significant proportion of the TD we described above.

1.6 Not so SWIFT

As Heraclitus correctly foretold, the only constant is change, and usually at an ever-increasing rate. Yet all of this TD acts like a brake on change, on innovation, on security, on improving efficiencies, and on improving the lives and well-being of people all over the globe, simply because they were never built for change. This single fact affects individuals, global corporations, local governments, national governments, and sometimes even intercountry, globally relied-upon systems, some of which come under the heading of Critical Infrastructure. It can be critical to a locale, such as the US Air Traffic Control system, which contributed to the collision between an Army helicopter and an American Airlines regional jet preparing to land at Ronald Reagan Washington National Airport, killing 67 people. Or a larger geographic area, such as the power grid covering the Iberian Peninsula, affecting mainland Portugal and peninsular Spain, where on Monday, 28 April 2025, electric power was interrupted for about 10 hours. Or it could be critical to national security, as would be the case with nuclear power plants. Or critical to the effective running of a large proportion of the world's commercial enterprises, as is the case with SWIFT.

SWIFT stands for Society for Worldwide Interbank Financial Telecommunication and is used by over 11,000 banks and other financial institutions in 200+ countries to exchange payment orders, securities transactions such as bonds or stocks, foreign exchange instructions, and trade finance messages. SWIFT has come into the public eye of late because it has been used by the United States and European Union to impose economic sanctions on countries such as Iran and Russia.

It was founded in 1973, and the messaging network was first launched in 1977, the same year that George Lucas released the first *Star Wars* movie. Thankfully, SWIFT has continued to service its clients, in this galaxy at least, for nearly half a century. However, SWIFT was designed during what we might now term as "gentler times," when the term "cybercrime" would have been analogous to science fiction. Consequently, while the network itself has proven to be relatively secure to date between the endpoints, by which we mean the

connection from client to client, the original designers could not envisage how organizations would use and eventually abuse the system. Take the 2016 heist against the central bank of Bangladesh Bank, sometimes referred to as the Lazarus Heist because of the Lazarus hacking group connected to North Korea, which is believed to have perpetrated the crime. The hackers took control of the Bank's own SWIFT connection via malware and attempted to steal $951 million of bank funds using valid SWIFT credentials after infiltrating the bank's network.

Another weakness of SWIFT is that internal actors with high privileges can initiate or approve illicit transfers, and weak internal controls at client organizations can further increase this risk. However, in the instance of the Lazarus Heist, part of the problem was the lack of real-time verification across this legacy system, such that messages sent by one institution to initiate a payment by another institution are acted on based on trust. Once processed, it's hard to reverse fraudulent transactions quickly. Furthermore, the older or legacy infrastructure of some banks on the network makes it harder to implement modern cybersecurity practices consistently.

Despite all of these weaknesses, SWIFT has done a remarkably good job over the years, but sometimes in spite of itself. The actual sum stolen by the Lazarus Heist was, in fact, just $81 million, of which $15 million was subsequently recovered. How, you may ask, was such a massive crime thwarted in part, but not in whole? Well, it all came down to a simple—but for the hackers—a very costly spelling mistake. The fraudulent SWIFT transfer instructions for £20 million to a bank in Sri Lanka were intended for a fake non-governmental organization (NGO) called Shalika Foundation. The payment instruction included the recipient's address as Jupiter Street in Manilla, misspelled by the hackers as Jupitar Street. This in turn triggered a sanctions alert within the intermediary bank, which was Deutsche Bank in this case, because it resembled the name of a sanctioned vessel the *MV Jupiter*, which was very fortunately spotted by the keen eye of one particular individual, albeit for completely the wrong reasons since there was absolutely no connection between the sanctioned vessel and the street in Manilla. However, this red flag halted this particular payment and contributed to the non-payment of most of the other 34 fraudulent SWIFT

messages sent as part of the same heist. Some payments were stopped for different reasons, such as the incorrect formatting of the messages sent and suspicion by the Federal Reserve Bank of New York relating to unverified account details. However, the headline here is that the payments messaging network that literally runs much of the modern world almost allowed nearly $1 billion to be fraudulently stolen by a hacking group, due in no small part to the design and technology of that network, but for the keen eye of one individual in one of the specific banks involved in the payments chain, rather than because of the technological wizardry built into the payments network itself. If this were an isolated example, one might be forgiven for letting it pass; however, it was not. In 2015 Ecuador's Banco del Austro lost $12 million to hackers who compromised the bank's SWIFT terminals and in 2017 hackers once again used malware to steal £6 million from the Russian Central Bank among other institutions. To borrow from the Talking Heads 1981 song "Once in a Lifetime," you may ask, "Well, how did we get here?"

1.7 Where Did It All Go Wrong?

There are many reasons over many decades that have contributed to the current situation, some of which we shall touch upon in this book. However, it probably doesn't help that beyond learning how to use basic office productivity tools such as MS Word or MS Excel today, or WordStar and Lotus 123 in the 1990s, many of the senior leaders who oversaw the selection, operation, and use of such enterprise-grade technologies were never formally educated in computer science, data science, or even the natural sciences themselves, making it very hard for top-level management to ascertain what "good" might look like at that time, let alone 40-plus years into the future. Even today, among the staff of the UK's public sector, fewer than 10% have a Science, Technology, Engineering, or Math (STEM) degree. By contrast, the equivalent figure in the US is 16%.[25] In the US private sector, less than 4% of workers have a STEM degree out of a total of 152.6 million workers as of 2021.[26,27]

Statistically, by far the majority of leaders in private and public sector organizations around the world have almost no formal

training in and, by inference, limited understanding of data science and computing technologies at all. Most will have learned what they know on the job. So, they may not know how they work, how best to deploy them, and how to change them. In a technical domain, there is a big disparity in knowledge between those who have a broad or "managerial" or even conversational level of understanding about a specific, technical topic and those who have the required depth and breadth of knowledge necessary to make optimal decisions and achieve the ideal set of outcomes for their organization. Several of the case studies we cover in this book serve to highlight the gulf that can lie between these two constituents and the often catastrophic consequences that can result from this gap between those who know and those who do not. It could be said that it's not so much the computer that says "No!" as it is the people involved in the key decisions who say "No!" to the right things and "Yes!" to the wrong things.

1.8 Change 101

With so much money at stake and numerous change initiatives continuing to fail, drastic and substantial action is required to change the course of future initiatives. This book aims to start that ball rolling by highlighting how and why technology is failing us and how we can do some very practical things to improve the situation. It is written for those who use or rely on technology, as well as for those who build that technology or manage the teams that design and build it, whether within private or public-sector organizations. The change we are seeking to bring about is for leaders and practitioners of change to feel more empowered to ask the right questions and to be in a position to adopt a more software-centric mindset when considering future initiatives. This software-centric mindset would encompass a better understanding of the high-level processes, language, and, most importantly, the behaviors associated with successful software development and change. Additionally, it would help greatly to have some familiarity with the potential pitfalls—and related examples—that may be encountered along the journey and some pragmatic,

easy-to-understand ways of minimizing the undesirable while creating beneficial outcomes for all involved. We will expand on the importance of adopting a more software-centric mindset later in the book.

Having outlined the scope and scale of the problem and its implications for us all here in Section 1, in Section 2 we will define the 14 Behaviors that have been proven over more than two decades of application to reduce the chances of failure and to significantly increase the likelihood of creating more successful outcomes that can benefit us all. Section 3 is a step-by-step survival guide to setting up and running technology programs of any size. Building and implementing technology is, paradoxically, often a very human-intensive and creative endeavor, where the path to achieving one's goals may not be known at the start of the journey. An adventure if you will. So, it's interesting to note that both the 14 Behaviors and the Survival Guide have also been successfully applied to creative endeavors and adventures of the non-technology variety, from music production to home renovations, since they provide structure and form to any creative process. If by the end of this book you feel better informed and more empowered to ask "why" did a technology-related problem arise and "how" can we do better next time, then the revolutionary ball has started rolling, and we have achieved at least some of our goals.

Ultimately, failing to modernize technology affects every aspect of society, from the family left stranded in a Spanish airport at the end of their holiday, to a business that struggles to tap into foreign markets for its new product because it's built on old technology, to industry-wide challenges of the European EV car manufacturers facing down competition from better, cheaper, software-enabled Chinese brands, all of which affect national competitiveness and productivity. As Western nations stagnate economically, the inability to update critical systems and develop new products and services supported by modern software practices will become an increasingly significant barrier to future progress. Addressing this challenge is not just an IT issue; it is an economic imperative. This book will explore these issues in depth, providing practical guidance on how to build resilient, scalable, and adaptable

software that can support future innovation. The lessons shared here come from decades of experience working with and helping the many client organizations that have successfully navigated this complex relationship between software, technology strategy, and business success. We cannot offer you the promise of better government or better products and services, but we can hopefully improve the way governments and product or service providers spend your money.