O'REILLY®

Overcoming IT Complexity

Simplify Operations, Enable Innovation, and Cultivate Successful Cloud Outcomes

Early Release
Raw & Unedited
Compliments of
f5®

LEE ATCHISON
with contributions from MARK MENGER

A big "yes" to modernization. But what about complexity?

As applications become more distributed than ever, complexity continues to rise. And with 84% of organizations planning moves toward the edge, this complexity will only increase. Read these and other trends in the latest F5 State of Application Strategy Report.

Get the report ›

Overcoming IT Complexity

Simplify Operations, Enable Innovation, and Cultivate Successful Cloud Outcomes

Lee Atchison
with contributions from Mark Menger

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Overcoming IT Complexity

by Lee Atchison

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Jennifer Pollock and Gary O'Brien	**Indexer:** FILL IN INDEXER
Production Editor: Beth Kelly	**Interior Designer:** Monica Kamsvaag
Copyeditor: FILL IN COPYEDITOR	**Cover Designer:** FILL IN COVER DESIGNER
Proofreader: FILL IN PROOFREADER	**Illustrator:** Kate Dullea

March 2023: First Edition

Revision History for the Early Release

2022-07-07:	First Release
2022-08-08:	Second Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781492098492* for release details.

978-1-492-09843-0

[FILL IN]

Contents

What is the Modern IT Complexity Dilemma?

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at gobrien@oreilly.com.

The complexity of modern IT systems supporting modern applications can impact your customers, your partners, your employees, and the quality and security of your application.

Addressing this complexity dilemma is essential, but it is not easy. The IT Complexity Dilemma refers to the challenges businesses face when trying to manage and optimize their IT operations. The increasing complexity of modern IT systems has made it difficult for businesses to keep up with technology changes and make necessary updates to their infrastructure, making it difficult for them to realize their desired return on their technology investments. Business pressures have caused IT organizations to focus increasingly on creating new features and focus less on resolving ongoing problems and upgrading existing architectures. This has led to a build-up of technical debt, which can have consequences such as slowed business processes, increased IT costs, and even system failures.

There are several measures that businesses can take to mitigate the effects of the IT Complexity Dilemma, including investing in automation technologies, establishing standard operating procedures, and hiring skilled IT professionals. By taking these steps, businesses can improve their ability to manage and optimize their IT operations, minimizing the negative impacts of complexity.

However, even with these measures in place, businesses will still face challenges in managing their IT operations due to the increasing complexity of modern IT systems. By attempting to mitigate the effects of this complexity, businesses can improve their efficiency, security, and competitiveness in today's increasingly complex IT environment.

As we will discuss later in this chapter, the core to the complexity dilemma is this technical debt. In fact, technical debt and complexity go hand-in-hand. We'll learn that, beneath the surface, technical debt is much more than needed code refactorings.

Technical debt and complexity go hand-in-hand.

Instead, technical debt is a metric that describes everything that interferes with the smooth operation and customer experience of an application. In other words, technical debt is everything that makes the application complex—whether that is operational complexity, complex customer experiences, or simply complexity that makes the application difficult to enhance, expand, and improve.

But before we can talk about complexity, it's important to understand how IT organizations are structured in modern enterprises, and how the operations and development teams within the IT organization interact to create a working system.

Structure of Modern IT Organizations

IT organizations vary considerably in size and shape. A modern IT organization is typically a relatively flat, matrix-type structure. Teams of developers and operators work together closely in this organization. To keep up with the fast pace of customer demand and technological change, both the development and operations teams need to adapt quickly. Development needs to both quickly build new systems and applications and be able to repair and upgrade existing systems. Operations not only needs to deploy and manage those systems quickly but also detect and resolve problems when they occur. A close, working relationship between development and operations is critical to this speed.

However, natural separations start taking shape as applications grow in complexity and the organization grows in size. Traditional divisions between development and operations begin to formalize, and the space between the two groups grows and expands.

A flat management structure has been able in the past to assist the organization in keeping the communications channels flowing as much as possible—flowing between development, operations, security, and product leadership teams. But, as the organization grows, keeping the organization flat and responsive becomes harder and harder.

Management and organizational structures are required to keep the growing organization operational. Formalized processes help yield consistent results and plans. Yet, these same structures and processes create a natural blockage to communications flow. This blockage makes

it harder for the organization to function. Teams split, and the organizational distancing limits cooperation and communications. This limits growth.

Ironically, the biggest inhibiter to growth is, in fact, growth.

Ironically, the biggest inhibiter to growth is, in fact, growth.

The organizational structure gets more complex, and the application gets more complex.

Since an IT organization is only as good as the management that drives it, it is essential to have a strong and effective management team in place. This team is responsible for steering the organization in the right direction, setting goals and objectives, and ensuring that all aspects are running smoothly.

The management team must also adapt to changes quickly and effectively respond to new demands in the marketplace. They must work across organizational boundaries, and operate in unison with all product, development, operational, and support teams.

How the IT organization is structured varies considerably based on the nature of the business. The type of company is the biggest indicator of the structure of the IT organization. And nothing drives the organization more significantly than where and how the software development teams are organized.

ROLE OF SOFTWARE DEVELOPMENT IN IT ORGANIZATIONS

Within an IT organization, the structure and responsibility of the development organization is related to the nature of the business and the structure of the rest of the company. The importance of the development organization ranges from a limited role in managing internal processes and systems to a role of being an integral part of the core business model of the company. This means there is no one-size-fits-all organizational structure that defines a modern IT organization.

But we can generalize. For this discussion, we're going to define three types of businesses and describe the IT organization structure that tends to occur in each of these businesses:

- The SaaS Focused IT Organizations
- The Non-SaaS Software Focused IT Organizations
- The Non-software Focused IT Organizations

Let's look at each of the three types individually and their characteristics. Then we will look at how the IT and software development organizations look different inside each type of company. In the end, you will find that your circumstances lead to an amalgam of two or three of these types.

Business Type #1: The SaaS Focused IT Organizations

This is a business where the primary purpose is to create a Software as a Service (SaaS) application, or a company where the operation of customer software is the primary goal of the business. This includes business-to-business (B2B) companies providing services such as inventory management, financial planning, communications, and sales infrastructure. Examples of SaaS companies and products include Intuit QuickBooks, Slack, Shopify, MailChimp, and Salesforce.

There are also business-to-consumer (B2C) SaaS companies providing entertainment, retail shopping, and social media services. Example B2C SaaS companies include Amazon, Netflix, Facebook, and Twitter.

A highly functional SaaS organization requires a high caliber application development team, **and** a high caliber operations organization. The two teams must cooperate to succeed. DevOps is a common model in a modern organizational structure.

Take a look at the example SaaS company in Figure 1-1. The engineering and product management of a SaaS company are typically proportionately large groups that require large investments in application development. Each team owns a part of the overall SaaS application and builds, tests, deploys, operates, and maintains only the services they are responsible for. Operations teams provide a smooth operations infrastructure that supports the development teams. The focus of the enterprise, typically, is on the software development teams. These companies may have separate IT organizations to support business processes, but the application development teams are not part of that organization.

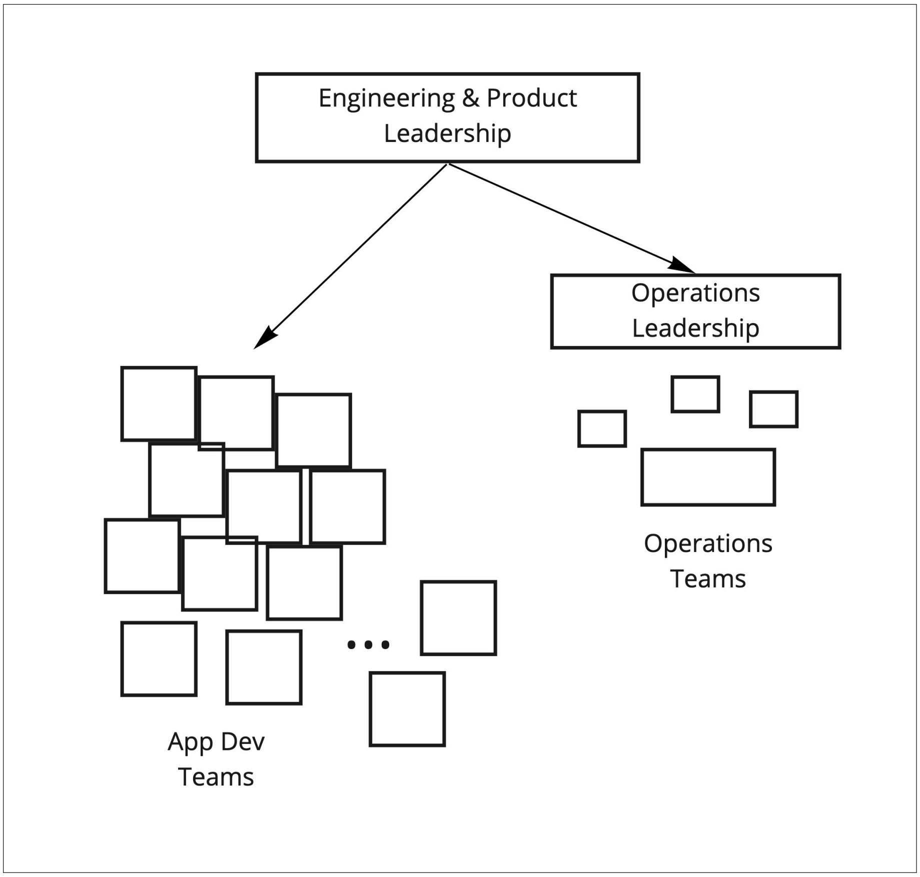

Figure 1-1. SaaS Focused Organization

In these high tech, software driven companies, the application development teams are a core component of the enterprise. Given its importance, this group reports high in the company's organizational hierarchy.

Business Type #2: The Non-SaaS Software IT Organizations

This is a business with the primary purpose of creating software that they sell to other people or companies, but they typically do not operate the software for their customers. Any software that doesn't require a significant back-end SaaS-like application to function requires the software they produce to be operated directly by the customer.

Examples of this type of software include popular software such as Microsoft Word and Adobe Illustrator. But it can also include game software such as Angry Birds, music applications, and ebook readers. It might include tools like anti-virus software, firewalls, and news aggregators.

A non-SaaS software organization has a software or engineering focused mission. These organizations have high caliber applications development teams, but they typically do not have a strong operations team. Since the software the organization sells is given to customers to run, rather than operated by the company itself, there is no need for a large operational focus.

A generic separate IT Organization provides tools and processes that support the company as a whole, including the product development teams and sales/marketing, etc. However, these teams are support teams. They support the efforts of the product development teams, but they do not contribute to the product development or operations at all.

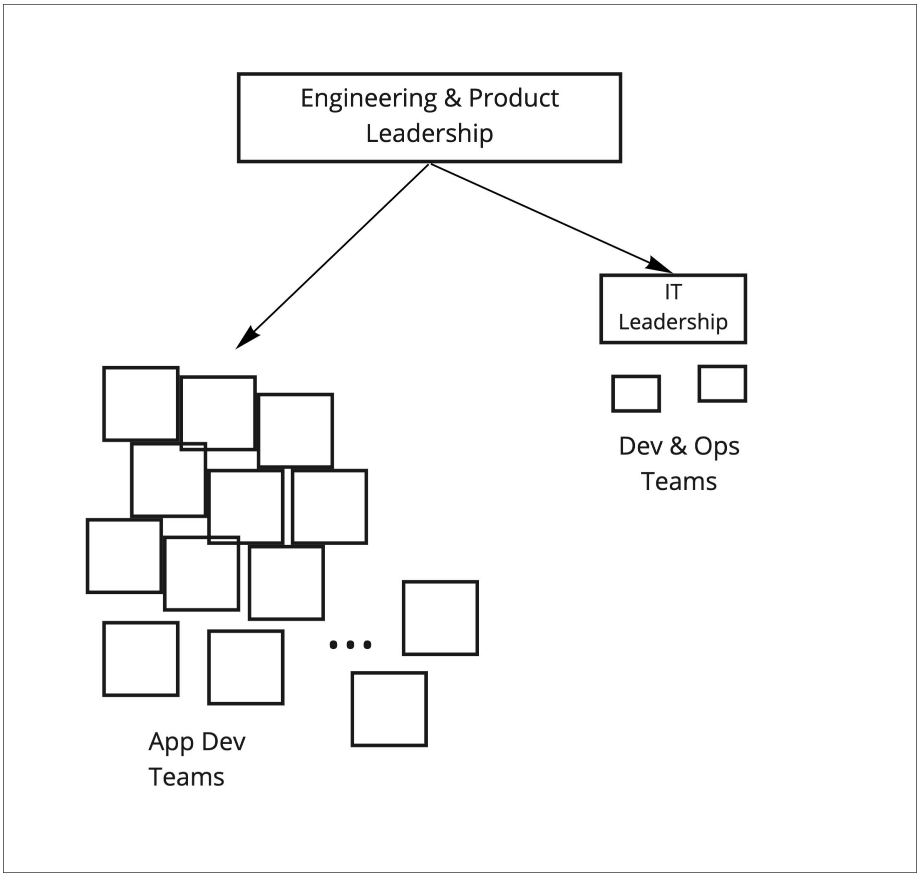

Figure 1-2. Non-SaaS Software Company

Take a look at the example in Figure 1-2. Non-SaaS software companies have the same focus on application development teams as SaaS focused organizations, but these teams are typically not DevOps teams. They are independent software development teams that produce software sold and delivered to customers. The IT organization is small and provides support to the company as a whole, including development and operations of tools for the company. The IT organization is separate and isolated from the primary, mainstream product development software teams.

Business Type #3: The Non-Software Focused IT Organizations

This is a business with a primary purpose other than producing, selling, or operating software. They likely will use both internally and perhaps customer facing software to run their business, but their primary business is not software. This is almost all non-technology focused companies, including banks, restaurants, stores, taxi services, airlines, railroads, media companies, etc.

Notice that some companies do fit multiple categories. For instance, Microsoft offers SaaS services (Office 365) and non-SaaS software (Microsoft Word and Halo). Additionally, a company such as Charles Schwab may offer investment software as a service, yet they also focus on general financial services and investments. These companies may have different divisions that appear to be separate companies, each structured differently. Or they may have a hybrid structure. Keep in mind that these categories are generalizations.

The mission of a non-software company is not technology focused. They may use software as a tool internally to manage the sales, marketing, manufacturing, or other business processes, but software is secondary to their primary business.

As such, there are no large application development teams. There is an IT organization, and that organization will have a relatively small development and operations team within the IT organization itself. Calling the organization "small" is in relation to the company itself. Only a small portion of the company's resources are invested in IT systems and personnel.

Figure 1-3 illustrates this. The company leadership has bigger things to focus on, leaving IT leadership to manage these small development and operations teams.

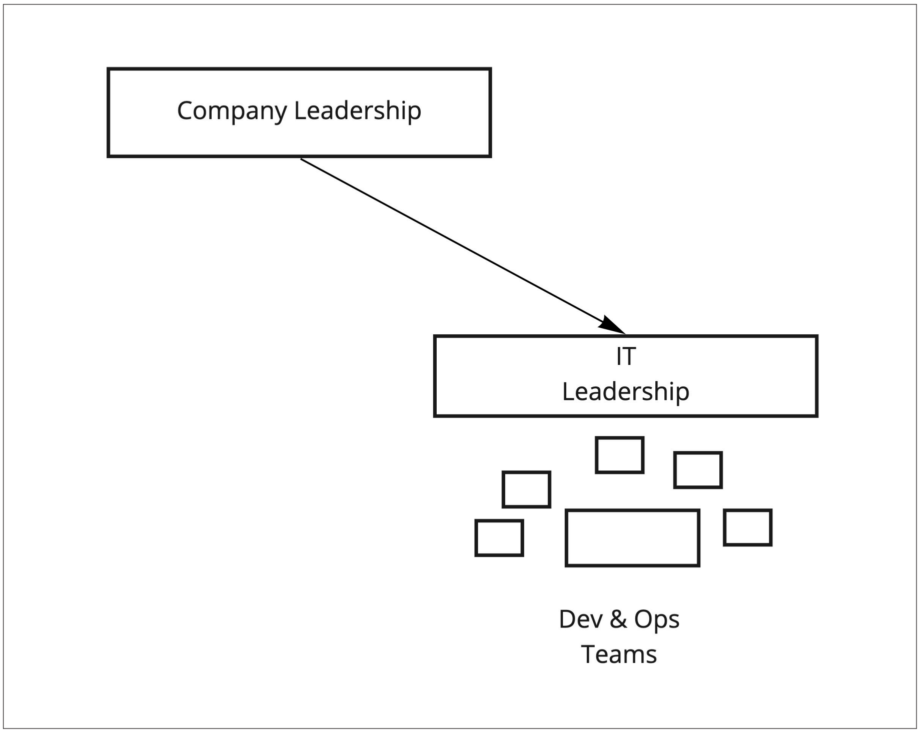

Figure 1-3. Non-Software Focused Company

For these more traditional non-software-based enterprise companies, the IT organization is purely in a supportive role. The IT organization is primarily operations focused, but may also have some development capabilities. Tooling and operational processes are the central focus.

THE STRUCTURE OF IT ORGANIZATIONS

Within an IT organization, the development teams are focused on driving the tooling and resources needed to operate the company, including building applications necessary for the company to run. The responsibilities and the processes the teams follow depend on where the company fits within the three types of organizations we just discussed.

Development – Company Focus and Talent Availability

There is a direct correlation between the amount of focus your company places on software development with the availability of high quality technical talent and software leadership available to your organization. The best software developers, architects, and software leaders tend to gravitate towards the much more lucrative opportunities in SaaS application development, and other software-centric companies.

This means that organizations where software plays only a secondary role in the company's mission find it difficult to attract and retain software talent. Often this means the organization, as a whole, suffers. Yes, there are high quality, talented developers in these organizations, but they are much harder to locate and hire, and hence tend to be less available to a typical non-development-centric organization. This tends to create less innovation and fewer creative solutions to problems in these types of organizations. Rather than state of the art software applications, the applications created in such organizations tend to be supportive applications that lack a high degree of innovation.

The caliber of your IT development teams is critically important in determining the sophistication of applications your organization can support, and your organization's ability to respond to the increase in complexity that occurs over time.

The result is, organizations where software is a secondary part of the business rather than a primary part tend to be organizations that are more sensitive to the IT complexity dilemma.

Operations — Different Focus than Development

Interestingly, a different characteristic occurs in the operations part of IT organizations. Operations organizations tend to attract high quality talent based on how central the operational aspects of the organization are to the business focus. This means SaaS companies, which are highly dependent on high quality operational organizations, tend to attract high quality operational talent, compared to organizations where the operations are secondary to the goals of the business.

The next highest talent attraction is to non-software companies that require a significant internal software infrastructure to keep the organization running smoothly. This includes organizations such as banks and other financial firms. These companies have important software infrastructures that must stay operational and attract strong operational-focused talent.

Less attractive to operational talent are companies that produce "boxed" software that is operated on customer computers and systems, rather than internal operational environments.

This makes sense. SaaS companies rely heavily on highly performant operational environments and invest heavily in these areas. This investment, and the opportunities it produces, attract the top operational engineering talent to these organizations. Organizations that are less operations focused need less investment in this area, and hence don't attract as much interest.

Traditional operations, however, is changing. Many traditional operational capabilities are handled by outsourced infrastructure, such as SaaS applications and cloud service providers. Additionally, newer tooling and capabilities automate a large portion of basic operational needs.

Tools such as *Infrastructure as Code* (IaC) and *Operations as Code* (OaC) help with this automation, and strive to make operational setup and basic operational responsibilities automated and repeatable. This improves overall operational reliability. Additionally, since Scripts and script-like descriptions drive iaC and OaC, these capabilities encourage code as documentation and knowledge sharing of the operational environments involved. Finally, since IaC and OaC generalize the operational aspects of an application into code-like capabilities, they allow the use of standardized and well understood development processes, such as revision management. Revision management allows tracking and correlating failures to changes, creating a better operating team, reducing mistakes, increasing security and traceability, and improving overall accountability.

The Role of DevOps in Modernization Enterprise

DevOps is a term in wide use in modern application organizations. It describes the collaboration between development and operations teams within an organization. The intent of DevOps is to break down the traditional barriers that exist between these two groups and encourage collaboration and cooperation, allowing them to work together more effectively and efficiently. This leads to faster problem solving, increased efficiency and responsiveness, and overall higher application quality and availability. DevOps is becoming the norm in modern IT organizations.

In a DevOps organization, individual teams own some portion of the application—both the development and testing of the component and the deployment and ongoing operation of the component. In modern applications, the components are typically application services, meaning the owning team is responsible for the development, testing, deployment, operation, and ongoing support of the services in their care.

Take a look at Figure 1-4. This shows a software company that uses DevOps principles. As such, it describes the same sort of SaaS company described back in Figure 1-1. The primary difference is that both the development aspects, and most of the operational aspects of ownership are assigned to the same team under the engineering and product leadership organization. There is still a very thin operations support organization, but the job of this organization is not to manage the operations of the application; rather, their job is more of a tools and infrastructure team. They provide tools and assistance to the product teams that own and operate their individual services.

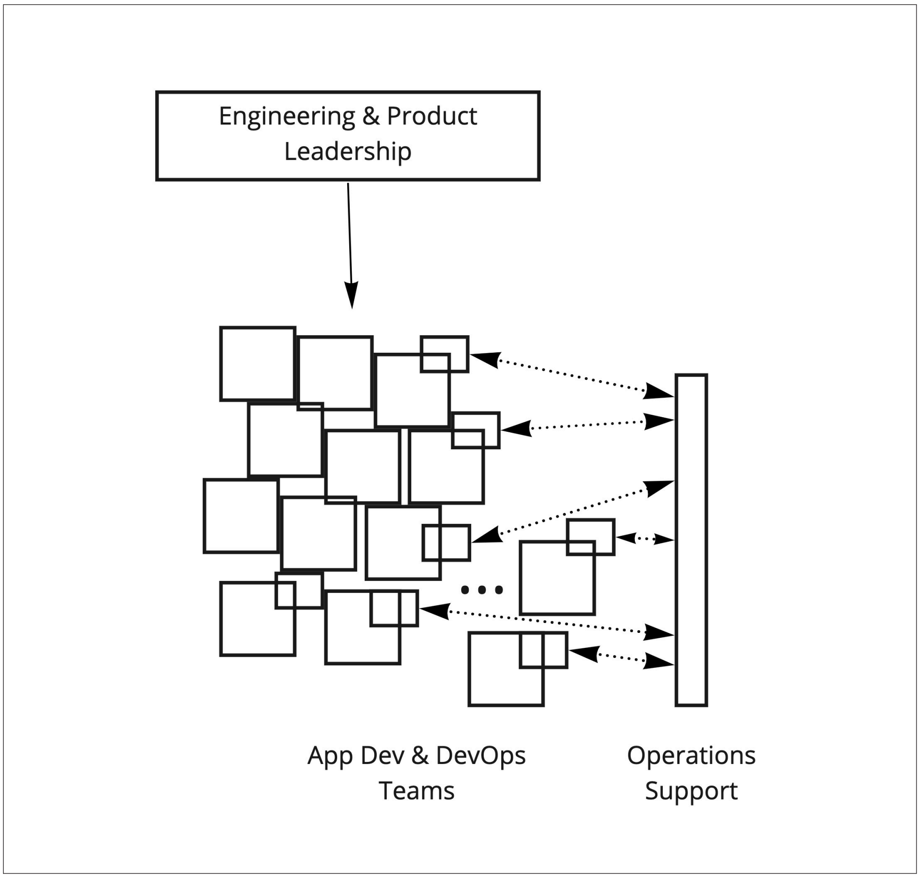

Figure 1-4. DevOps Focused Software Company

By merging the development and operational aspects of individual services into a single organization, communication barriers between the historically typical development org and operations org diminish, which improves the performance and reliability of the organization.

Now that we understand more about how modern IT organizations are structured, we can talk about the problem complexity plays in these organizations and how complexity impacts the long term viability and success of the organization.

THE ADVENT OF COMPLEXITY

It's hard to point exactly when complexity begins within a young startup IT organization, but it's usually tied to some decision that was meant to reduce time to market or cost to market, at the expense of some further work or cost later on. This starts the slow and inexorable climb in complexity. For larger, more established enterprises it's undoubtedly tied to the incorporation of technology into the established enterprise's processes. In either case, the increase in complexity is tied to the increase in technical debt. So the advent of IT complexity is driven by the collection of technical debt within the organization.

Technical Debt – The Key to Complexity

In software development, technical debt is the cost of additional rework caused by choosing an easy, limited, or sub-optimal solution now, rather than using a better approach that would take longer or cost more money.

Ward Cunningham coined the term technical debt. According to Ward:

"technical debt includes those internal things that you choose not to do now, but which impede future development if left undone."

The code development aspect of technical debt only captures a small part of it. Technical debt applies to all aspects of the modern application design, development, and long term operation of the application.

For SaaS and other cloud-centric applications, the long-term operational impact is a significant driver of technical debt. The longer an application operates, the greater the technical debt. The greater an application's technical debt, the greater the negative impact on the long-term operation of the service.

When a SaaS company decides to add a new capability to an application but takes shortcuts in the architecture to launch the capability quicker, they are adding to the technical debt of the application and the company. Unless a concerted effort is made to resolve the technical debt by completing the proper design and architecture, your application will build up this debt until it hinders your development and operational processes.

Technical debt is similar to financial debt. In moderation, it can be handled and the cost can be dealt with. But if technical debt is not repaid, it can accumulate "interest" in the form of additional technical debt. Just as in financial debt, too much interest and you can no longer afford to repay the debt. Too much technical debt makes the changes necessary to resolve (or pay back) the technical debt harder and more expensive to implement.

Let your financial debt grow too large, and you will go broke. Let your technical debt grow too large, and your application will become unsupportable and unsustainable.

How does technical debt grow? Technical debt can grow naturally and quietly during the normal product development process. Every project that contributes to a product, also contributes to its technical debt. This is illustrated with the top box in Figure 1-5. During the normal product development, work and output is added to the product, as well as some amount of debt to the stack of technical debt

Sometimes, a project is done “quick and dirty”, such as when a new feature is added without proper design in order to get it out the door quicker. In these cases, the project adds more debt. Sometimes, the project can even add more technical debt than useful capabilities. This is the example project shown in the middle box in Figure 1-5. More technical debt is added to your application than the amount of real value the project provided.

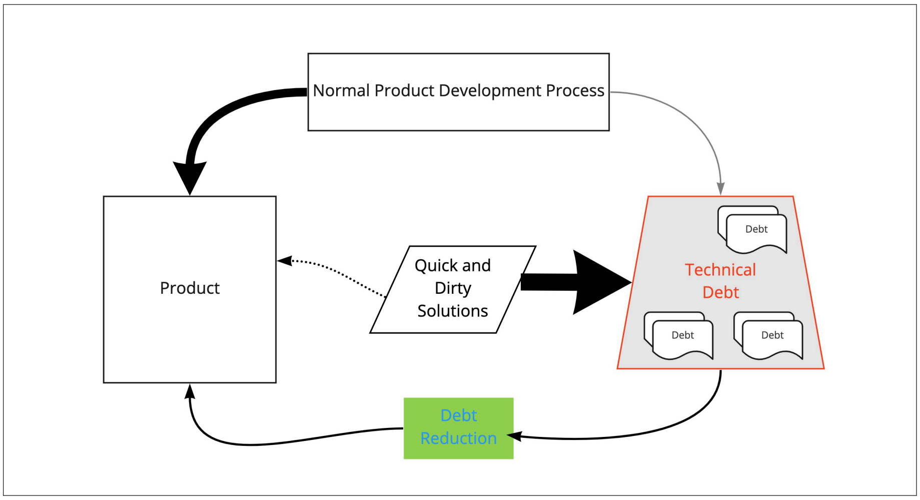

Figure 1-5. Flow of technical debt during product development

To keep technical debt from growing without bounds, some effort needs to be added to each project to reduce the technical debt. As shown in the bottom box in Figure 1-5, keeping your technical debt at a sustainable level requires constant investment in reducing the technical debt over time.

This constant flow of increasing and decreasing technical debt is one of the reasons why it can sneak up on a product. If more debt is regularly added than is reduced from the backlog, the debt will grow, yet the growth may not be noticeable. It's not until the debt has grown to a point where it starts having a negative impact on your product that you notice its size. At this point, it may be too large to deal with effectively and easily.

Each project can either increase or decrease the technical debt within a system. During a full, high quality project, the planned work often includes doing all the work necessary, along with working on reducing some amount of related technical debt. When the work is completed, the technical debt for the application is lower than it was before. This is illustrated in Figure 1-6. The **work completed** for the project is larger than the project itself, and the extra effort is towards reducing the size of the technical debt. This is a project that's dealing with technical debt in a healthy way.

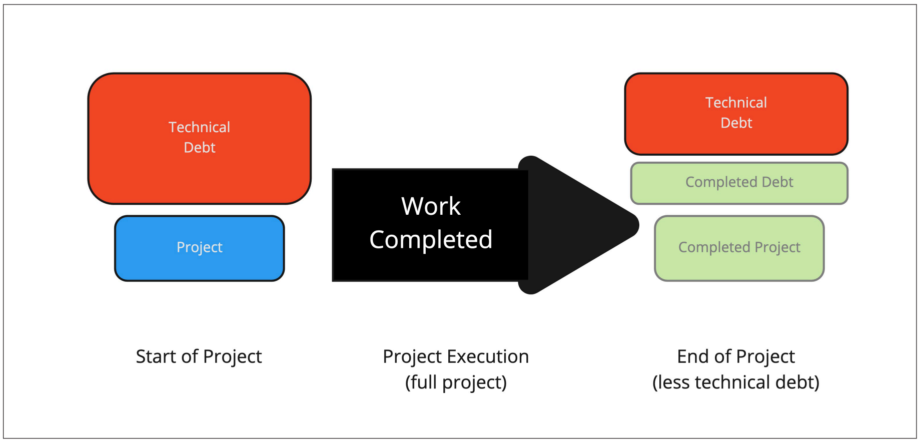

Figure 1-6. A full/complete project can plan to reduce technical debt

Unfortunately, many projects are much more quick and dirty. They are designed to only complete as much of the project as is absolutely necessary, leaving the rest of the project as work that will be completed later. In fact, a common project management philosophy involves building an MVP — Minimum Viable Product, essentially dictating that you do as little product work as possible to get a functional product out the door.

The result is work that is not completed. More often than not, this increases the overall technical debt of the application. This phenomenon is illustrated in Figure 1-7. Here, the **work completed** is only part of the total project. We have left some amount of **work not done** out of the project. This additional work which was not completed, ends up increasing the overall technical debt remaining in the project.

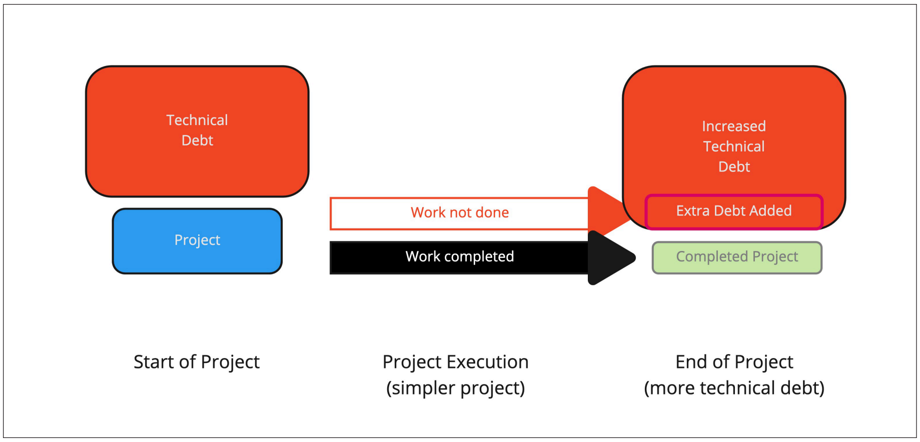

Figure 1-7. A quick & dirty project often increases technical debt

In any case, as projects are executed, the amount of technical debt can vary over time, sometimes decreasing, sometimes increasing. The more full, high quality projects that are completed, the lower the overall technical debt. The more quick and dirty projects used to implement functionality, the higher the resulting technical debt. The types of projects you execute will, over time, vary the total technical debt you have in your application.

The Negative Impact of Technical Debt Sometimes deciding to build a simpler solution now, in favor of delaying longer term implementation is advantageous (this is the Figure 1-7 situation). It allows you to get a solution out to customers earlier, which allows the company to start monetizing the change, and receive customer input on capabilities the customer likes and does not like, which can be fed into a later, more ambitious solution. This is analogous to saying that borrowing money is advantageous if you use the money to contribute to a greater cause, such as purchasing a home. Paying some interest on borrowed money is fine, as long as the money you borrowed is put to good use. So too, with technical debt, managing some technical debt is useful and appropriate as long as you give value to your product and your company. Technical debt becomes a problem when it is left unresolved—unresolved technical debt ages over time and increases in cost.

Using the financial metaphor, technical debt becomes a problem when it builds up so that the cost of servicing the debt is too great, and it impedes your ability to invest in future projects. So, too, technical debt becomes a problem when managing and servicing that debt is overwhelming compared to managing and servicing the product.

When too many quick and dirty projects rule your project plan, and projects designed to reduce debt are not staffed in your company, your debt starts to become unmanageable. If this goes on for too long, technical debt overwhelms the project, as shown in Figure 1-8.

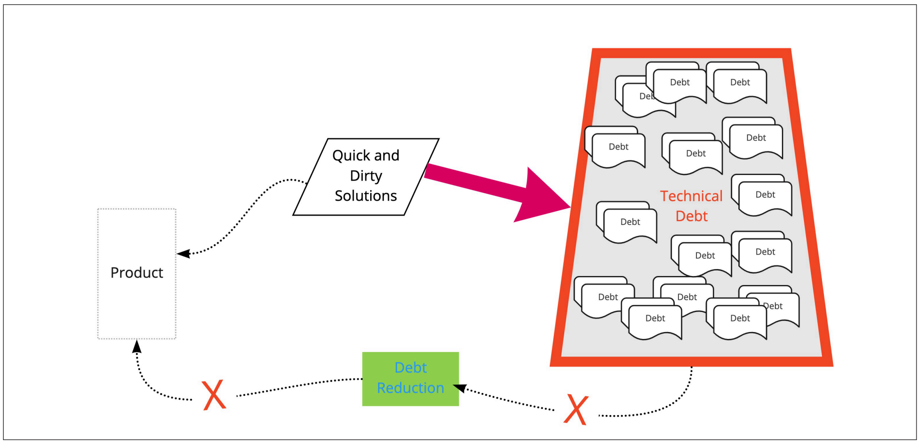

Figure 1-8. Technical debt overwhelming the project

In this scenario, servicing the debt becomes the dominant role of your team, and you spend little or no time contributing to improving the product. Your debt is too large to be effectively managed.

The organizational pain of complexity

Technical debt and complexity go hand-in-hand. So too do complexity and organizational pain. While technical debt is not a complete picture of application complexity, the growth of technical debt is tied very closely to the growth of application complexity.

As your application gets more and more complex, many things happen to it:

It becomes brittle

A complex application is subject to minor issues quickly escalating into major problems. While most applications operate in a well of a positive feedback cycle, a complex application's operational well turns into a negative feedback cycle, and minor issues quickly escalate out of control. Figure 1-9 shows a stable application tends to stay in the valley between the two hills, which builds success upon success, while the brittle application is ready to roll away off the top of the hill into failure at the smallest of nudges.

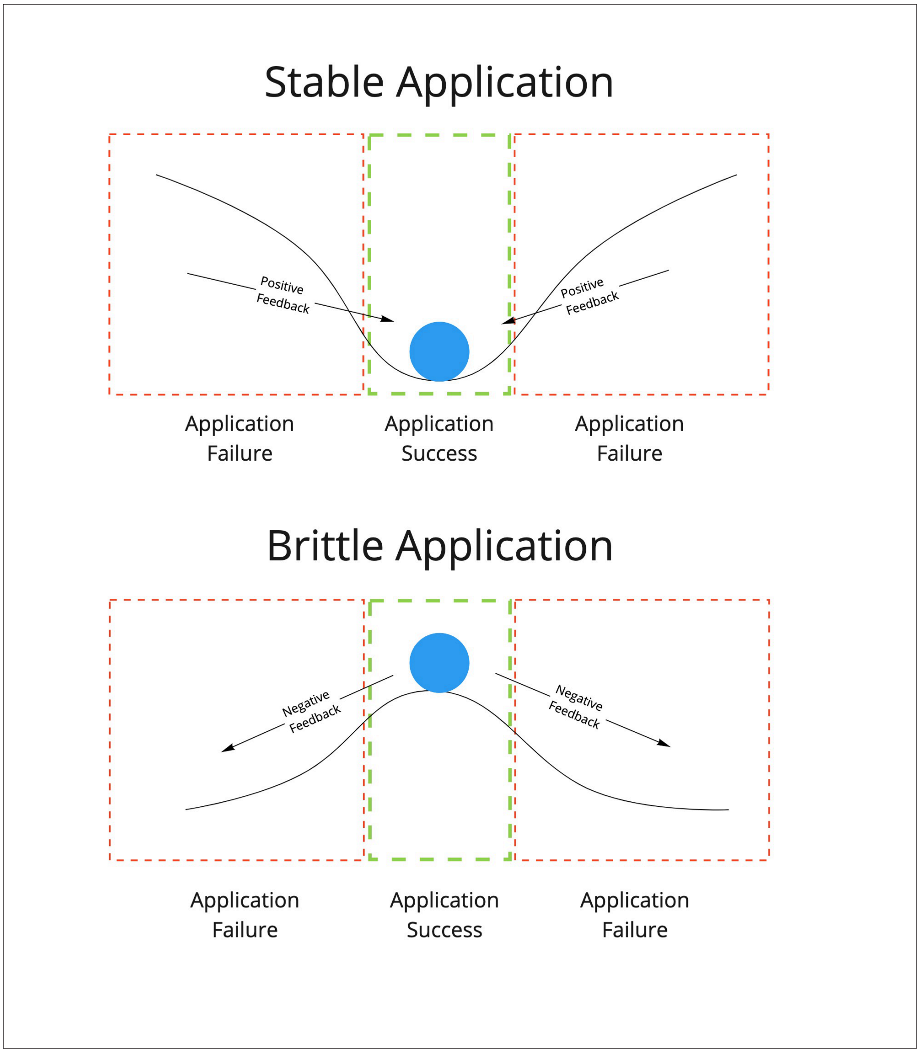

Figure 1-9. Application brittleness leads to instability and failure.

Fewer engineers have complete knowledge of the application
Only the most senior engineers, and those engineers working on the application the longest, have a broad understanding of how the complete application works. As time goes on, the knowledge of these most senior engineers becomes

diluted, less accurate, higher level, or more specialized. Broad, general purpose, but detailed knowledge of the application as a whole is no longer possible by single individuals.

The knowledge that engineers do have on the application becomes obsolete quicker

Complex applications change frequently, and engineers' knowledge about how the application works gets outdated quicker.

It gets harder to bring new engineers up to speed

Complex applications have long learning paths. This is not only because there is more to learn. The knowledge needed for new engineers to become productive is more distributed, anecdotal, and out of date.

The net result of these issues is higher organizational pain. This pain translates into poor quality changes, less motivated staff, and ultimate staff turnover. Higher turnover means greater need to train new engineers, which is harder as the pain increases. Brittleness leads to lower availability, and customer-visible issues and failures.

This is the pain of complexity.

Messy Desk Syndrome Imagine a perfectly clean desk. Now, take a sheet of paper and set it in the corner. Is your desk messy? No, not yet. Now you take fifty other sheets of paper that go together and sit them on top of the one sheet in the corner. Is your desk messy? No, not yet. Now imagine more sheets, but these don't go with the stack in the corner, they are for a different project. So you put them in different locations on the desk, just single sheets in single locations, seemingly in a location that makes sense. Now put more papers and documents and books and folders and pictures one at a time all over the desk. If you don't know where something goes, just put it in a new location. You'll figure it out later. Sooner or later, your desk is messy. In fact, it's extremely messy.

Unless you have a solid organizational plan for organizing the papers on your desk established *at the beginning*, and stick with it, sooner or later your desk will become messy, one sheet at a time.

Your desk becomes messy because you didn't have a plan from the beginning, but just "winged it" along the way. You made your desk messy simply by using and working on it.

So too, your organizational pain becomes large because you didn't have an architectural plan from the beginning. Rather, you started with no plan and adjusted and changed the plan as time went along. You "winged it", metaphorically. You have added technical debt, and hence organizational pain, simply by working and building the application.

Every action you take, little by little, builds up. Your technical debt grows a bit at a time until it becomes overwhelming.

- "Let's change our login process to allow saving login credentials in the user's browser"
- "Let's add this new feature to that menu"
- "Users would rather this feature work in three steps rather than the current four steps. Let's combine two of the steps."
- "We need to remove the per-session limit on this resource"
- "We don't have time to build this full feature now, but we can build this smaller feature, which will make many customers happier. We can do the rest later."
- "Let's release this feature this way first, and then we can collect input from customers and modify it to make it more user friendly as we get more input"

Any one of these statements can correspond to a simple set of changes that makes perfect sense at the time. It might not have any obvious impact on overall technical debt at all.

But the little changes...and the little debt...and the little impact...and the little piece of paper on the corner of your desk...adds up. And like the messy desk, each action may individually seem perfectly benign. Actions may look perfectly acceptable. But, when combined, they multiply and become overwhelming.

Icing the Cake

Many IT organizations use the expression "icing the cake". Icing the cake is when you describe the current situation as "everything is ok", whether it actually is ok or not.

The slow accrual of technical debt and organizational pain leads to this process of icing the cake. At the start, all is well and it's easy to say "everything is ok", because it is—or at least appears to be.

But as time goes on, and technical debt increases, and organizational pain increases, little by little, things aren't ok any more. But the tendency to keep saying "everything is ok" is strong. The organization keeps "icing the cake".

Ultimately, a debt-ridden application operated by a pain-ridden organization still says that "everything is ok". They fail to notice what's obvious to every outsider. The pain is real and the organization is not ok. Without you noticing it, the icing went bad.

Complexity in an IT Organization

Complexity grows in IT organizations as well. Complexity starts by growing within your application. As your application grows complex, so does the infrastructure needed to run the application. Your IT operations become more complex. Your engineering organization becomes more complex. To wrap their minds around all of this, your IT management gets more complex.

What started as a simple increase in the needs of your application, has changed into the growth of a complex IT organization.

An organization that was once agile tends to change and migrate over time. It changes into either a robust or rigid organization—one that is afraid of and rejects change in order to keep the system stable and supported, or it changes into a fragile organization—an organization where every minor change risks breaking a larger system or process, limiting the ability to adjust and grow. This is illustrated in Figure 1-10.

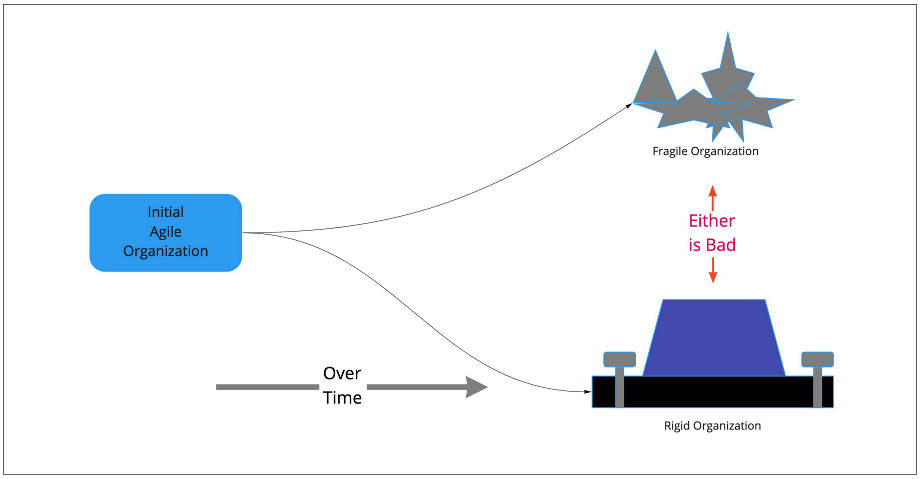

Figure 1-10. An Agile organization fails over time either by becoming rigid, or fragile.

As complexity increases, the ability of the organization to change direction quickly and respond to new demands diminishes. The organization becomes less agile.

As complexity increases, the ability of the organization to change direction quickly and respond to new demands diminishes.

Almost independent of the specific company, complexity and technical debt correspond to a lowered ability to respond to market and competitive demands.

Why is this true? There are typically two reasons why application complexity leads a company to lower agility.

First, when an organization has overly complex applications, changes to those applications are increasingly likely to cause problems. Small changes and small adjustments cause large failures and outages to occur. This makes the organization hesitate. Changes go through additional review cycles, changes get consolidated, changes that don't show clear value are discarded as too dangerous. Rather than having a "Yes, we can try that" attitude, the stock attitude of the organization changes to a much more conservative view. "No, not unless it's absolutely necessary" becomes the more likely answer.

This is the reaction to keep the application from failing. It's the company's response to keeping the application robust—the fewer changes you make, the safer the application is.

The application development process slows down considerably. This makes the application—and the company—significantly less agile than its competitors.

Second, if the organization doesn't naturally slow down, then it might continue to make changes. However, because of the application complexity, the organization tends to make changes without understanding what's involved in the changes. This risky behavior leads to dangerous changes, and these changes break things. The organization doesn't slow down and become conservative, but moves forward recklessly. The result? Application availability suffers and customer issues increase. Technical debt increases even more than it had previously. This feeds into the complexity and it creates a vicious circle. Complexity leads to brittleness which leads to failures which lead to technical debt which leads to complexity.

IT Death

So, technical debt leads to complexity, and complexity leads to organizational pain. This all ultimately leads to *IT death*.

But what does IT death look like?

IT death is what happens to an organization when the pain of complexity sends the organization into a state of ineffectualness. It cannot improve, it cannot grow, and hence it stagnates. Since competitors will continue to grow, an organization's stagnation ultimately leads to its death.

You see it in many organizations.

Xerox, long the leader in copiers for larger organizations, suffered from the inability to pivot from copiers to the personal computer. Despite the fact that Xerox PARC originally conceived the modern personal computer user interface,[1] they were unable to compete with Microsoft and Apple for the personal computer operating system. Arguably, without Xerox PARC, there would be no Apple Macintosh computer, yet Xerox's inability to pivot kept them from this innovation.

It's not just technology companies that suffer this fate. Firestone, the tire company, was facing the difficult task of modernizing its tire creation process in light of radial tire technology created by one of its competitors, Michelin. Firestone bogged down and could not update its processes to handle the new technology. Try as it might, it kept making tires that customers did not want, and their business suffered. Ultimately, Firestone was absorbed by Bridgestone. This is an example of what the Harvard Business Review[2] calls *Active Inertia*.

Many other originally highly innovative companies fall into the trap of IT death by losing their ability to innovate. Hewlett Packard, one of the founding companies of Silicon Valley—the heart of technical innovation across the world—found its lack of innovation lead to a slow death spiral.

And let's not talk about the innovation failure of Polaroid, which couldn't innovate new camera technology; or Blockbuster Video, which failed to embrace the importance of video streaming technology.

And Borders book store, which was overwhelmed by the innovation of the upstart Amazon.com.

Technical debt and complexity slow down innovation. They keep companies from staying competitive, and ultimately this results in their eventual downfall, and potentially even death.

What makes a mature IT organization

A mature IT organization is an agile organization. It can make decisions quickly and easily, stick by those decisions until organizational needs dictate a change, and implement those decisions quickly and effectively.

Why is it important for a mature IT organization to be agile? Without agility, companies fall into two traps that can bring them down:

- Lack of *competitive offerings.*
- Unsafe *security vulnerabilities.*

Let's look at each of these in turn.

Competitive Offerings Maintaining competitiveness is critical to a modern application. This is because the pace of change is accelerating. Technology is advancing and new competitors are emerging constantly. Your competitors are moving faster than ever before. If you can't keep up with your competitors, you quickly fall behind and soon become irrelevant. Keeping up means moving faster and faster, which means being able to adapt and change as the situation demands.

Customers are constantly pushing the feature, price, quality envelope with new and innovative ways of doing business. If you have a customer that is pushing you on price, quality, or feature set, you need to be able to respond to remain competitive.

New ideas are the lifeblood of a competitive company. When a new idea comes up, you need to be able to quickly adjust and adapt to enable the new idea. This requires agility.

Customers are looking for innovation when it comes to making a buying selection. Companies that appear innovative are more likely to get the customer's business. This means you need to respond to customer requests and customer needs faster. Failing to do this will not only lose you customers, but it will soon cost you credibility in the industry.

Agility is essential to maintaining a thriving business.

Security Vulnerabilities Your competitors aren't the only ones innovating. Bad actors are innovative as well.

Never before has the IT infrastructure of our valuable applications been at risk to security vulnerabilities and actions of bad actors as it is today. Bad actors are not only growing in numbers, but they are growing in sophistication as well. Bad actors are just as innovative at coming up with new ways to attack your application, as your competitors are innovative at coming up with new ways to attack your business success.

Bad actors are constantly innovating, improving their attack vectors, and exposing the vulnerabilities of our applications.

You, as a company and the owner of your application, must innovate constantly to keep your applications safe and secure. You have to constantly strive to keep one step ahead of the bad actors. This innovation requires agility.

Summary

Hence, the IT complexity dilemma. IT agility is critical to building a successful company, yet the very success itself adds technical debt and complexity, and this complexity leads to either rigidity, or fragility. In either case, ultimately, competitors will outpace the organization in innovation, and the organization dies. Long term success for a company means managing the IT complexity dilemma.

1 Xerox's Palo Alto Research Center.

2 Why Good Companies Go Bad, Harvard Business Review.

About the Author

Lee Atchison is a recognized industry thought leader in cloud computing, and the author of the best selling book *Architecting for Scale*, published by O'Reilly Media, currently in its second edition. Lee has 34 years of industry experience, including eight years at New Relic and sever years at Amazon.com and AWS, where he led the creation of the company's first software download store, created AWS Elastic Beanstalk, and managed the migration of Amazon's retail platform to a new service-based architecture. Lee has consulted with leading organizations on how to modernize their application architectures and transform their organizations at scale. Lee is an industry expert and is widely quoted in publications such as *InfoWorld*, *Diginomica*, *IT Brief*, *Programmable Web*, *CIO Review* and *DZone*. He has been a featured speaker at events across the glove from London to Sydney, Tokyo to Paris, and all over North America. LinkedIn profile: https://www.linkedin.com/in/leeatchison.