CUTTER Business Technology Journal

Management, Innovation, Transformation

Vol. 32, No. 7, 2019

Is Software Eating the World?

Greg Smith
Guest Editor

CUTTER Business Technology Journal

As business models for creating value continue to shift, new business strategies are constantly emerging and digital innovation has become an ongoing imperative. *Cutter Business Technology Journal* delivers a comprehensive treatment of these strategies to help your organization address and capitalize on the opportunities of this digital age.

Cutter Business Technology Journal is unlike academic journals. Each monthly issue, led by an expert Guest Editor, includes five to seven substantial articles, case studies, research findings, and/or experience-based opinion pieces that provide innovative ideas and solutions to the challenges business technology professionals face right now — and prepares them for those they might face tomorrow. *Cutter Business Technology Journal* doesn't water down or delay its content with lengthy peer reviews. Written by internationally known thought leaders, academics, and practitioners — you can be certain you're getting the uncensored perspectives of global experts.

You'll benefit from strategic insight on how the latest movements in digital innovation and transformation, artificial intelligence/machine learning, Internet of Things, blockchain, analytics, and cloud, to name a few, are changing the business landscape for both new and established organizations and how cutting-edge approaches in technology leadership, enterprise agility, software engineering, and business architecture can help your organization optimize its performance and transition to these new business models.

As a subscriber, you'll also receive the *Cutter Business Technology Advisor* — a weekly bulletin featuring industry updates delivered straight to your inbox. Armed with expert insight, data, and advice, you'll be able to leverage the latest business management thinking to achieve your organization's goals.

No other journal brings together so many thought leaders or lets them speak so bluntly — bringing you frank, honest accounts of what works, what doesn't, and why. Subscribers have even referred to *Cutter Business Technology Journal* as a consultancy in print and likened each month's issue to the impassioned discussions they participate in at the end of a day at a conference!

Get the best in thought leadership and keep pace with the technologies and business models that will give you a competitive edge — subscribe to *Cutter Business Technology Journal* today!

Founding Editor: Ed Yourdon
Publisher: Karen Fine Coburn
Group Publisher: Christine Generali
Managing Editor: Cindy Swain
Copy Editors: Jennifer Flaxman, Tara Meads
Production Editor: Linda Dias
Client Services: service@cutter.com

Cutter Business Technology Journal® *is published monthly by Cutter Information LLC, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA • Tel: +1 781 648 8700 • Fax: +1 781 648 8707 • Email: cbtjeditorial@cutter.com • Website: www.cutter.com • Twitter: @cuttertweets • Facebook: Cutter Consortium. ISSN: 2475-3718 (print); 2475-3742 (online).*

Subscription rates are US $485 a year in North America, US $585 elsewhere, payable to Cutter Information LLC. Reprints, bulk purchases, past issues, and multiple subscription and site license rates are available on request.

NOT FOR DISTRIBUTION
For authorized use, contact Cutter Consortium +1 781 648 8700 or service@cutter.com.

☐ Start my print subscription to *Cutter Business Technology Journal* ($485/year; US $585 outside North America).

Name	Title	
Company Address		
City	State/Province	ZIP/Postal Code
Email (Be sure to include for weekly *Cutter Business Technology Advisor*)		

Fax to +1 781 648 8707, call +1 781 648 8700, or send email to service@cutter.com.
Mail to Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA.

Request Online License Subscription Rates

For subscription rates for online licenses, email or call: **sales@cutter.com** or **+1 781 648 8700.**

CUTTER CONSORTIUM
Access to the Experts

Opening Statement

by Greg Smith

In 2011, Marc Andreessen, developer of the Netscape browser and cofounder of the Silicon Valley venture capital firm Andreessen Horowitz, stated in an article in the *Wall Street Journal* that "software is eating the world."[1] I remember thinking at the time that this was a memorable aphorism, but while it captured the increasing importance of software, it seemed somewhat cryptic or vague. Little did I realize that, over the next 10 or so years, it would come to articulate a profound transformation of the world we live in and, especially, the enterprises we lead and operate within.

Over the last 10 years we have seen a fundamental shift, whereby organizations that have spent decades developing and perfecting their business models and core capabilities have been outcompeted by organizations that have used software to disrupt existing models or establish wholly new models. Additionally, this "softwarization" of products, services, and experiences has, in many ways, only just started — especially if we consider artificial intelligence (AI) and machine learning to be a specialized class of software.

So what does this mean for your organization?

If the current capabilities, strategies, differentiation, and/or competitive advantage that define your organization are rooted in the physical world but can be replicated within software, then the challenge is clear. If you are creating value through mastery of the physical environment and your competitor can replicate this mastery in a software environment, then the outcome will be inevitable, although the timeline might be variable.

Take the hypothetical example of two pharmaceutical companies in a race to develop a new blockbuster drug. One organization has optimized its R&D processes to have both the highest velocity and lowest cost in developing new formulations in the lab and assessing their potential efficacy. The second organization has invested in state-of-the-art simulation software, theoretically allowing it to formulate drugs and, just as importantly, eliminate noneffective options within the software environment. If the second organization can simulate and eliminate 50% of options within the software environment that previously would have had to be developed in the lab, then probability suggests there will be only one winner in the race, despite the advantage the first organization holds in physical R&D.

I recently encountered an interesting example in the industrial refrigeration market. The traditional goal of R&D within this industry has been targeted on physical engineering to optimize energy and product efficiency. However, a disruptive competitor was looking to enter the market with a way of driving efficiency based on applying fluid dynamics and optimizing airflow. The software necessary to achieve a reliable simulation of airflow needs to handle high complexity, be computationally intensive, and requires an understanding of mathematics far removed from typical engineering R&D. However, these capabilities are well established and available to aerodynamicists operating within motor racing, where the potential disruptor learned and perfected its capability. The inevitable outcome is that there is likely to be one long-term winner when physical engineering innovation is competing with complex mathematical models that can optimize over a thousand iterations in an automated software simulation.

Fortunately, though, there is a better answer than physical engineering competing with sophisticated software in a dialectic battle, and this is to combine the two domains into one physical/digital innovation capability. All organizations will need to become equally skilled in both domains if they are to become leaders in their industry, but this introduces a big challenge.

The Fundamental Challenge Facing Organizations

For the last 30 years, most large and well-established organizations have followed industry "best practice" in terms of their IT capability and platforms. They have implemented standard packaged applications, inevitably delivered through the agency of specialized system integrators and overseen by internal IT functions, whose role has been limited to strategy, procurement, and delivery management. This has led to a situation where large enterprises, with internal IT functions comprising several hundred people, might contain no specific roles focused on software creation or might even not possess an understanding of how software should be developed!

To put it simply: at the point where mastery of software is becoming critical to the success and ongoing survival of the enterprise, there is an absence of expertise and insight within the organization's decision-making forums to represent the potential that software can unleash and the inevitable disruption that will be required to seize this potential.

In some ways this dichotomy is an updated manifestation of the "Two Cultures" C.P. Snow identified in British academia in the 1950s.[2] Snow was perplexed at how the scientific and arts and humanities communities he encountered in leading universities could be so ignorant of each other's domains; an ignorance that was especially confounding given that these were some of the brightest minds, almost exclusively drawn from the same backgrounds and demography. Two non-overlapping cultures had emerged, where almost all interactions and experiences served to reinforce the division and mutual antipathy.

The two cultures we experience in 2019 within our enterprises can be categorized as those schooled in the business school curriculum of case-study strategy, financial management, and corporatism versus those schooled in mathematics, software engineering, and algorithms.

The Start of an Answer

The good news is there are various patterns and approaches that can start to bridge the two cultures and unlock the value-creating potential of softwarization.

A great place to start is to fully embrace the principles of Lean Startup, as set out by Eric Ries.[3] The culture of experimentation, rapid iteration, and a single cross-functional team, working in an accelerated and nonhierarchical way, is a great learning experience. It quickly exposes the team to both the software process and its potential and builds confidence through a "show me, don't tell me" approach.

A second way is to focus ruthlessly on those areas where bespoke software can unlock a problem or drive competitive advantage. I believe in the principle of "build for competitive advantage but buy for competitive parity." It is important, if the senior leadership is to start believing in the power of software, that that power be applied to the opportunities that will unlock substantial business value and where a real difference can be manifested.

Third, it is imperative that those who understand software find ways of communicating and evangelizing the opportunity it presents within their organization and to its leadership. This almost invariably involves patience in overcoming frustration, constant reframing, humility, and tenacity. How many of us tasked with creating understanding and enthusiasm for software within our organization can genuinely say they embrace these virtues on a daily basis?

I am reminded of a saying that changed my personal approach over a decade ago: "Nobody ever changed their mind by being proven wrong!" If the opportunity presented by software is to be fully realized in the enterprise, then it is imperative that those who understand software create the bridge to those who need to understand.

In This Issue

In our first article, Cutter Consortium Fellow Steve Andriole examines the extent of software's rule in the areas of process automation, privacy and security, enterprise software, intelligent software engineering, and converged convenience. For each area, he evaluates in what ways software's reign is good (rewarding us), bad (punishing us), or ugly (threatening us). Andriole's belief is that software's rule is inevitable and will expand. It is our decision what to do about the "kingdom of software."

In the next article, Joost Visser begins with an acceptance of software's having "eaten the world" and the need, after your organization's digital transformation, to master the evolution of software. Software evolves in the environment of the marketplace, where the forces of innovation, cost reduction, growth, regulation, and coevolution drive change. As with biological evolution, only the fittest will survive. For sustainable evolution — for organizations not to see their software eaten by the world — refactoring and commoditization are essential. After examining these internal changes, Visser discusses the essential capabilities organizations must possess in the areas of data, design, and decisions to master software evolution. He concludes with the critical questions organizations must answer to determine whether they are ready for the long haul.

Next, Sunil Mithas, Kaushik Dutta, and Cutter Consortium Senior Consultant San Murugesan intriguingly compare software to the ouroboros, the mythical serpent of the ancient world that eats its own tail and is reborn from itself. Like the ouroboros, software has cannibalized and transformed itself. In recent years, software has evolved toward autonomy. Autonomous software has the capability to change itself (as with automatic updates) and even to write itself (AI can write software code or even be the software). Software evolution and changes in software development imply that software will become ever more pervasive and affordable, that firms must master disciplined autonomy in order to follow dual strategies, and that the role of IT professionals is being redefined. The authors conclude with the steps that senior leaders and managers need to take for their organizations to transform and be reborn.

Paul Pagel next discusses the key importance of a modern software labor strategy for organizations hoping to remain competitive in today's digital and innovative world. The right team is key to crafting software systems capable of supporting innovation. Software delivery talent, however, is extremely difficult to find for a multitude of reasons. The solution, according to Pagel, is to structure software teams to deal with fragility and to thrive on change.

In our final article, Michael Papadopoulos and Olivier Pilot examine how a limited view of digital transformation impedes organizations from fully benefiting from the new, Agile ways of working. Papadopoulos and Pilot attribute this failure, fundamentally, to reliance on traditional architectural stacks where multiple teams and products rely on large, shared layers, and a change in a layer to meet the needs of one product may inadvertently break other products. To support a feature team–based organization, each team must have full end-to-end ownership of its stack, which consists of smaller, decoupled parts — microservices — that are loosely bound together. The authors advocate domain-driven design and the atomic design principle[4] as the basis for enabling reuse. A managed, messy architecture is the key to an organization structured around feature teams, which enable digital transformation.

Clearly, as this issue suggests, the rise of software represents the biggest single hurdle and opportunity to business. We hope the articles inspire you to conquer the fundamental challenges facing your organization today and help you unlock your full value-creating potential.

References

[1]Andreessen, Marc. "Why Software Is Eating the World." *The Wall Street Journal*, 20 August 2011.

[2]Snow, C.P. *The Rede Lecture*. Cambridge University Press, 1959.

[3]Reis, Eric. *The Lean Startup*. Currency, 13 September 2011.

[4]Hacq, Audrey. "Atomic Design: How to Design Systems of Components." Medium, 28 June 2017.

Greg Smith is a Senior Consultant with Cutter Consortium's Business Technology & Digital Transformation Strategie *practice. He is also Partner of Arthur D. Little (ADL), cofounder and co-leader of ADL's Digital Problem Solving practice based in London and New York, and leader within ADL's global Technology & Innovation Management practice. Mr. Smith's work specializes in the application of disruptive information technologies to solve intractable business problems in major enterprises, which one recent client described as "bringing a slice of Silicon Valley into the corporate." Over the past three years, he has been focusing on bringing patterns that are well known and accepted in FANG (Facebook, Amazon, Netflix, and Google) companies into major enterprises to positively disrupt their digital transformation initiatives. This includes addressing the human/cultural side of digital problem solving.*

During the last decade, Mr. Smith has alternated between strategic advisory and consultancy roles (ADL, Capgemini, and Atos Consulting) and hands-on technology leadership as CIO of a major, private equity–owned logistics company going through a merger in record time. This latter role allowed him to discover the joys of applying Agile principles to wholesale business transformation along with the need to be able to explain the explicit value contribution of IT, as technology funding was provided from the owners' private capital. Mr. Smith holds a BSc in biological sciences and finds that after 30 years of dormancy within his professional life, the underlying concepts of biology are becoming increasingly valuable at unlocking business problems and articulating solutions — especially where reductive, engineering-based approaches need to be replaced with whole-system, evolutionary thinking. He can be reached at gsmith@cutter.com.

IT'S INEVITABLE, RIGHT?

Is Software Good, Bad, or Ugly? Depends on Where You Sit

by Steve Andriole

Even if you have played no role in the design, development, or support of the applications that manage your personal and professional lives, you know that software rules the world. Every aspect of your life is enabled by a suite of fixed or mobile software applications that more often than not live in the cloud. It's safe to say that if you were separated or divorced from your apps, you would be unable to function. We can also define the rule of software by the integration of our personal and professional activities, which have strongly converged over the past decade in ways that often make it impossible to cleanly distinguish personal versus professional agendas.

Let's look at the extent of software's rule today in five areas and where we expect it to be in five to 10 years. A "good/bad/ugly" lens will help us assess the trajectories and determine the role that software should — and should not — play in our lives.

The Five Themes

There are many ways to understand good, bad, and ugly software and what the reign of software will deliver in the next decade or so. This article covers five themes:

1. Process automation
2. Privacy and security
3. Enterprise software
4. Intelligent software engineering
5. Converged convenience

Theme 1: Process Automation

Routine tasks — and even what appear to be the complex, deductive, inferential tasks that we associate with "knowledge" industries — will be automated by software bots of one kind or another; robotic process automation (RPA) will absolutely, positively eliminate jobs, careers, and whole professional existences. Indeed, it has been predicted that artificial intelligence (AI) (broadly defined) will eliminate 77 million jobs over the next 20 years: "By 2030, 75 million to 375 million workers (3 to 14 percent of the global workforce) will need to switch occupational categories."[1] Bloomberg has even developed a tool to help you determine if you're likely to be automated. According to Bloomberg (based in part on research conducted at the University of Oxford), "Nearly half of all US jobs may be at risk in the coming decades, with lower-paid occupations among the most vulnerable."[2] Compensation and benefits managers, auditors, accountants, credit analysts, loan officers, sales reps, truck drivers, administrative services managers, and even dental hygienists are at high risk and will most likely lose their jobs to automation. The same research suggests that (most) physicians, surgeons, (some) lawyers, financial managers, pharmacists, teachers, and computer and information systems managers are among the professions least likely to be automated.[3] The timing for all this varies. Some analysts believe significant professional displacement will easily occur by 2030, while others believe it will take longer — though not much longer.

Good, Bad, or Ugly?

If your job is in any of the at-risk categories noted above, you're likely doomed. The important question is, "How long do you have?" This, of course, is the nagging question about all disruptive technologies and the impact they will have on the jobs market. A recent VICE News/HBO special, *The Future of Work*, presented demonstrations of disruptive technologies — such as self-driving trucks, expert legal systems, financial management tools, and surgical robotics — already hard at work.[4]

While there will be some lag due to regulatory and liability requirements (especially regarding autonomous vehicles), disruptive technologies are marching quickly toward deployment. Many industries consider all this very good. Professionals in the most vulnerable fields believe it's bad. Some economists consider it all ugly since it displaces millions of professionals with no plan to relocate them into productive, well-paying careers. If ever there was an outcome dependent upon where you sit, process automation is it.

Theme 2: Privacy and Security

There is no privacy. Everyone is under surveillance. Security is so weak that foreign governments are easily able to penetrate US elections. Software enables — and, to be fair, tries to combat — all these conditions. However, the trends here are anything but good.

Let's start with security. According to the US Department of Homeland Security (DHS), threats are everywhere and growing. DHS believes that the US should "reduce threats from cybercriminals. In partnership with other law enforcement agencies, DHS must prevent cybercrime and disrupt criminals and criminal organizations who use cyberspace to carry out their illicit activities and leverage identified threat activity and trends to inform national risk management efforts."[5] The problem is enormous and growing faster than anyone can measure. Most computer scientists believe that no system is completely safe. Breaches are frequent — and frequently underreported.

How about privacy and surveillance? If you're on the grid, you're under surveillance. If you tweet, blog, or post, you're under surveillance. If you shop with credit cards, you're under surveillance. If you rideshare, you're under surveillance.

CCTV, smart TVs, Internet searches, social media, voice recognition/response systems, credit/debit cards, loyalty programs, facial recognition, image understanding, and even drones all enable surveillance. Within a few years, it will be possible for companies to profile nearly all of us from how we live our rich, full digital lives. As analytics improve, fewer and fewer digital indicators will be necessary to fully profile us. But surveillance will also empower government offices and agencies to profile individuals. Some of this will be good, such as enabling the pursuit, capture, and prosecution of criminals. But some of it will be ugly, such as what might happen when social, economic, and political enemies seek control, revenge, or worse. Make no mistake: the surveillance infrastructure is already in place and will only get wider, deeper, and stronger.

What's next? Technologies such as AI, machine learning (ML), 5G, blockchain, cryptocurrency, the Internet of Things, and wearables will make surveillance easier, faster, and complete. There's no need to implant chips into our bodies, though some are doing so, because we're immersed in digital trackers in our pockets, cars, homes, phones, TVs, appliances, thermostats, security systems, and, of course, our desktops, laptops, and tablets. We also know that leaving the digital grid is impossible. Surveillance is therefore inevitable.

Good, Bad, or Ugly?

This is an easy one — it's not good, and, at times, is very ugly. The lack of privacy due to the rise of surveillance is bad and ugly. There are aspects that make sense, such as criminal and terrorist digital surveillance. But regardless of the percentage of good versus bad (or ugly) and the convenience software enables, software used to reduce privacy and increase surveillance definitely nets ugly.

Trajectory? Much uglier: total grid dependency and technologies such as facial recognition will finalize surveillance. Cybersecurity also looks bad and ugly. As more and more activities, processes, and assets move to the cloud, it will become increasingly difficult to secure transactions, especially since general awareness of the breadth, depth, and severity of threats is ill-defined and underappreciated, and because cybersecurity funding, especially at the federal level, is incredibly inadequate. Both privacy and security are bad and headed toward ugly.

Theme 3: Enterprise Software

Who would undertake a five-year corporate software implementation project today? The failure rate for big "enterprise" software projects is downright scary. Depending on whose study you read, the enterprise resource planning (ERP) failure rate, for example, is anywhere between 50% and 75%. If even vaguely informed, executive management knows the project is likely to fail. Yet in the 1990s and early 21st century, there were still companies willing to try their hand with big software and prove they were not like the others who failed so spectacularly — until they too failed. Failure is the result of several trends and outcomes. One is control.

When a company embarks on a multiyear journey with an ERP or customer relationship management (CRM) vendor, they cede significant, if not total, process control to that vendor. ERP modules were originally designed to eliminate process chaos. Remember when "legacy" software was a barrier to scalability, not to mention how expensive it was to maintain? Moreover, a significant side effect of big software was the loss of process governance to the vendors that defined supply chain management, financial reporting, CRM, and other business processes for the companies they serviced.

The cloud also killed big software. Years ago, companies would implement huge enterprise software systems in their own data centers. The early 21st century gave us the cloud, so the pain of forever implementation was avoided. But there are also smaller, cloud-based alternatives to big software that scale, integrate, and share process control through customization tools deliberately built into the modules. Small companies can find lots of incredibly inexpensive alternatives from vendors such as Zoho and Zendesk, among others. Many of these companies will grow, as will the incredibly inexpensive, cloud-based systems that scale and integrate right along with them.

Good, Bad, or Ugly?

The movement of huge enterprise software suites to the cloud is good. The adoption of smaller, cloud-based, microservices-based applications is also good. But "good" depends on where you sit: for obvious reasons, software consultancies preferred the endless on-premise implementation of huge enterprise software applications. Big software vendors were happier before the cloud offered alternatives that organizations could adopt relatively quickly. Note that the growing capabilities of business software applications are unquestionably good. Overall? Good. The consultancies and software vendors will adapt and rearchitect their huge software suites into smaller pieces. Watch how SAP, Oracle, IBM, and Microsoft, among others, adapt to the competition from smaller, "enterprise" vendors. In fact, many of them already have, even it if means selling smaller suites to their clients.

Theme 4: Intelligent Software Engineering

We tend to think about functionality (i.e., what apps actually do) when we think about software. But where does software come from? How is it built? Will software help us develop software? Absolutely, and not just any old software: software will be designed and developed by intelligent — artificially intelligent — software. At the most basic level, smart software will automate many of the tedious steps in the software design and development process (e.g., testing). But the most significant impact will be felt in the auto-generation of code through the shadowing of human programmers and "learning" from their successes and failures — and then even deeper learning–based "programming."

So what happens to programmers when all of this automation takes hold? The timing of intelligent software design and development is difficult to estimate, though it's safe to say that within the decade much of this will be ready. Major software companies are investing heavily in intelligent software engineering, including SAP through its Leonardo Machine Learning Foundation.[6] IBM, Oracle, and Microsoft, among others, are also spending heavily in the area. Programmers will evolve to support personnel. Application development lifecycles will be compressed. Programmers will become incredibly productive.

Good, Bad, or Ugly?

The software industry continues to grow. Intelligent, automated software design and development is good. In fact, artificially intelligent–supported software development will become one of the most promising application domains of AI and ML. Programmers will adapt over time and learn to exploit the support of intelligent software design/development assistants (which will eventually become leaders). One of the red flags is the ethics of automated software design and development — so-called *ethical AI*[7] — and the value systems that enable expert and other intelligent systems to "decide" what to build. This is a larger issue for intelligent software engineering that could turn the assessment from good to bad, though probably not ugly.

Theme 5: Converged Convenience

We love streaming music and using location-based apps. We love tweeting and blogging. We love our project management tools and Microsoft Office 365. We love ridesharing. We also love Amazon and eBay. Love? Let's just say that these and so many other apps are indispensable to our personal and professional lives.

Lest we wax too poetic about the accessibility and functionality of these apps, remember that the greatest Trojan horse of the 21st century is the convenience our digital toys deliver: how easy it is to order anything we want from Amazon, how much fun it is to download music and books, and how effortlessly we can find a car, a house, and a date online. But what's the tradeoff? Every time we avail ourselves of these conveniences, we reveal a little more about who we are and what we do, which is all stored and analyzed permanently for those who want to buy some insight into what we like, what we will buy, and how they should pitch to us. It all seems innocent enough until we assess what's really happening.

That said, software enables the integration and management of our personal and professional lives. Schedules, shopping, meetings, and grocery delivery can all be managed from single portals that manage multiple applications. Smart city applications help us navigate locations, and telecommuting applications help us work from home. There are countless others that keep our lives manageable and productive.

Good, Bad, or Ugly?

The problem with converged convenience software is that it's really good, sometimes bad, and occasionally ugly. It's also inevitable because convenience is undeniable. Trajectories show more of the same: our personal and professional lives will continue to converge, and our need for software that makes these lives easier will grow. Most users will sacrifice some — perhaps a great deal of — privacy and even security if you make their lives easier. Software is a huge and growing part of this "transaction." Which presents a dilemma: Do we reduce convenience in exchange for privacy and security? Or do we sacrifice privacy and security for convenience? It's likely that convenience wins for so many personal and professional reasons. The drivers are unstoppable as are the returns on personal and professional software investments. By 2030, the personal/professional convergence will be seamless and assumed.

Good, Bad, Ugly — or Something Else?

There can be no debate: software rules the world. The five themes discussed in this article suggest how software is rewarding, punishing, and threatening us — all at the same time. There are clear winners and losers in the reign of software. In process automation, companies win by reducing costs and increasing profit, but all while realigning and reducing whole professions. In enterprise software, some consultancies and big software vendors have reluctantly adjusted to microservice architectures and cloud delivery, and some have exploited both these features of newer enterprise software with inexpensive, scalable products and services. Intelligent software engineering will generate faster code as it changes the role of the traditional software engineer. Privacy and security are the losers in the reign of software. There's too little awareness, focus, and funding, and it's already way too late in the game. Security and privacy are bad, trending to ugly. Part of the explanation is traceable to our love of convenience and the software toys we refuse to divorce even though they compromise our digital freedoms.

Above all else, we must acknowledge the inevitability of a world where software will continue to rule, and a world where the software kingdom will continue to expand. Some of this expansion will be good, some bad, and some ugly. Expansion also begins at earlier and earlier ages, with two- and three-year-old kids embarking into games and other digital toys on several mobile platforms. RPA seeks to automate as many corporate processes as possible. AI and ML will accelerate software development. Enterprise software will continue to shrink, spread, and scale. Privacy and security will yield to convenience. All of this is inevitable. The open question is, "What should we do about the kingdom of software?" Embrace it? Challenge it? Anything?

References

[1]Vlastelica, Ryan. "Automation Could Impact 375 Million Jobs by 2030, New Study Suggests." Market Watch, 4 December 2017.

[2]Whitehouse, Mark, and Mira Rojanasakul. "Find Out If Your Job Will Be Automated." Bloomberg, 7 July 2017.

[3]Whitehouse and Rojanasakul (see 2).

[4]*VICE Special Report: The Future of Work*. HBO, 2019.

[5]"US Department of Homeland Security Cybersecurity Strategy." US Department of Homeland Security, 15 May 2018.

[6]"SAP Leonardo Machine Learning Foundation." SAP, 2019.

[7]Bostrom, Nick, and Eliezer Yudkowsky. "The Ethics of Artificial Intelligence." Machine Intelligence Research Institute, 2011.

Stephen J. Andriole is a Fellow with Cutter Consortium's Business Technology & Digital Transformation Strategies and Data Analytics & Digital Technologies practices and the Thomas G. Labrecque Professor of Business Technology at Villanova University. Dr. Andriole was the Director of the Cybernetics Technology Office of the Defense Advanced Research Projects Agency (DARPA); the CTO and Senior VP of Safeguard Scientifics, Inc.; and the CTO and Senior VP for Technology Strategy at Cigna Corporation. His most recent books include Ready Technology: Fast Tracking New Business Technologies *and* The Innovator's Imperative: Emerging Technology for Digital Transformation. *He has published articles in* MIT Sloan Management Review, Communications of the ACM, IEEE IT Professional, *and* European Business Review, *among others. He can be reached at sandriole@cutter.com.*

A SUSTAINABLE DIET

The World Is Eating Your Software

by Joost Visser

Software has become the main differentiator by which organizations compete in today's economy. Through software we achieve faster delivery, higher efficiency, more accurate manufacturing, and higher customer satisfaction. To become and remain competitive, it is imperative that your business master software evolution. This does not only mean being able to create a winning application once; it means always being able to immediately adapt your applications to the unique needs of your users and your market and to continuously swap out those software components that no longer provide you with a competitive edge. If your business is not able to relentlessly keep evolving its software portfolio, you will fall behind.

Software Has Eaten the World

It has come to pass: software has eaten the world.

In 2011, Marc Andreessen used this imagery to describe his *personal vision* of what was then *still to come*: "My own theory is that we are in the middle of a dramatic and broad technological and economic shift in which software companies are poised to take over large swathes of the economy."[1] Today, this is accepted lore and is not limited to Silicon Valley:

> [T]he digital economy is worth US $11.5 trillion globally, equivalent to 15.5 percent of global GDP and that has grown two and a half times faster than global GDP over the past 15 years.[2]

The world has woken up to the increasing importance of software, and the question now for businesses everywhere is not *whether* to embark on a digital transformation of their business, but *how* to transform successfully — and as soon as possible. For new businesses, digital is not a question but a given.

Software Needs to Evolve

But digital transformation is not an end point; it is just a beginning. By going digital, your organization is only entering the game. To actually play and win, it is not sufficient to build a great digital product or service just once. Rather, it is imperative to continuously evolve your digital solutions to keep meeting the needs of your users and your market. After successful transformation, you need to master software evolution.

Starting in 1974, Professor Manny Lehman and his colleagues famously sought to capture important insights regarding software evolution by formulating a series of Laws of Software Evolution.[3] Though these laws predate the dawn of our digital economy by several decades, let's review three of them for clues they hold to help us evolve our digital assets:

- **First law: law of continuing change.** This law states that any software system in an organizational context "must be continually adapted, else it becomes progressively less satisfactory in use." In other words, software that is in actual use has the peculiar characteristic of being *indefinitely unfinished*. Once a software component has been created and first released, it becomes the subject of a stream of subsequent changes: bug fixes, updates, enhancements. Software that does not evolve at the right speed and in the right direction quickly falls out of use.

- **Second law: law of increasing complexity.** This law states that "as a [software system] is changed its complexity increases and becomes more difficult to evolve unless work is done to maintain or reduce the complexity." Thus, with each evolutionary step, the code tends to become messier and the architecture more entangled, making it harder and harder to apply further changes. Software that evolves becomes difficult to evolve. An explicit effort is needed to counteract this tendency.

- **Fifth law: law of conservation of familiarity.** This law states that "the incremental growth (growth rate trend) of [software] systems is constrained by the need to maintain familiarity." In other words, the speed by which an organization can enhance and grow its software systems is limited by its collective intellectual capacity to understand the structure and behavior of these systems. Knowledge dissipation and software growth conspire to kill software development productivity.

While these laws tell us that evolution is necessary (first law), potentially self-defeating (second law), and knowledge-bound (fifth law), they do not identify the actual *sources* of evolutionary pressure. How is it, exactly, that software "becomes progressively less satisfactory in use" as time passes?

Evolutionary Pressure

As in biology, changes in the environment fuel evolution. For business software, the environment is the marketplace. As Figure 1 illustrates, we can readily identify five types of market forces:[4]

1. **Innovation.** Businesses compete by bringing new or improved products and services to the market. Software supports the production of products and the delivery of services. Sometimes, software is an integral part of the product. Other times, software *is* the product. Business innovation drives software change.

2. **Cost reduction.** Services and products that were once innovative lose their differentiating power when competitors start offering the same for less. In markets where similar products or services compete on price, the operational costs of the software systems that support them become a critical factor. Reduction of operational costs drives software change.

3. **Growth.** A successful software business attracts new users and retains existing ones. This leads to a growth in interactions and in the volume of data processed, stored, and served. Unless storage, algorithms, and interfaces are optimized, the system performance will degrade and hurt usability. Growth drives software change.

4. **Regulation.** Governments are constantly at work to change laws and regulations, be it for the betterment of society or for propping up the financial system. Such changes in the rules require modifications not only to the governmental software systems that enforce the rules, but also to the software systems of banks, airlines, and other businesses that must comply with these rules. Laws and regulations drive software change.

5. **Coevolution.** Each software system is dependent on others. For example, a Web store depends on a payment system, a database system, an Internet browser, several operating systems, and so on. Apart from those execution-time dependencies, software systems have development-time dependencies on libraries, frameworks, and development tools. All those systems and components are likewise under evolutionary pressure. Changes in any of these induce the need for updates in the system that depends on them. Thus, changes in one system drive changes in other systems by propagating through the network of dependencies among them.

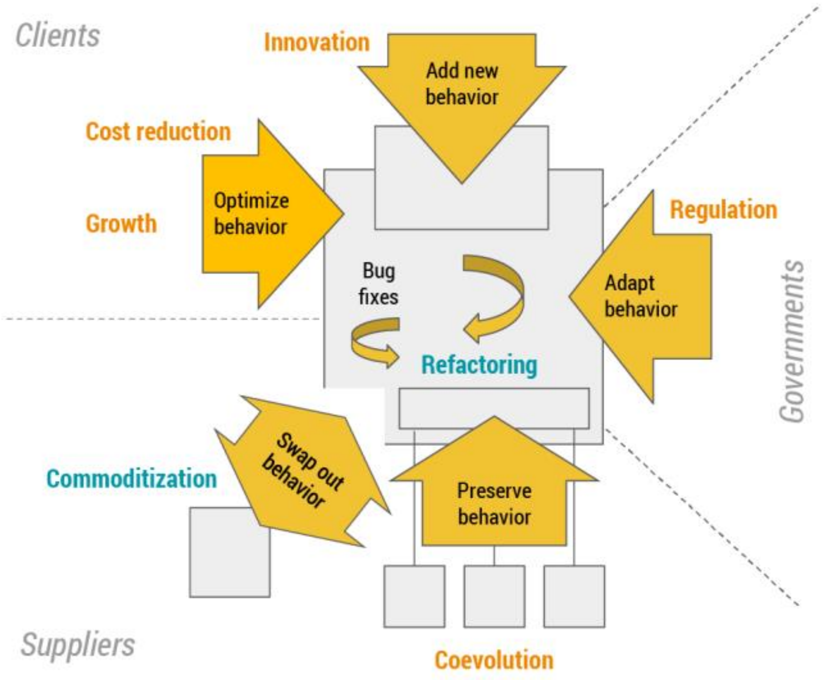

Figure 1 — The force field of software evolution.

While these environmental factors fuel software evolution directly, they also have a significant indirect effect. Each change inevitably introduces bugs. As a result, the initial change indirectly leads to the need for further changes (bug fixes) down the line.

These (direct and indirect) evolutionary pressures from the marketplace explain Lehman's first law of software evolution (continuing change).[5] But, according to laws two (increasing complexity) and five (conservation of familiarity), such evolution rapidly runs into problems if the size and complexity of the system are allowed to increase unchecked.

Unable to relentlessly keep evolving your software portfolio, your business will lose competitive power and will fall behind.

Survival of the Fittest

These problems are not imaginary. When seemingly small feature requests take ages to complete, when stability problems appear impossible to stamp out, when developers are afraid to break the system when making a change — these are the symptoms that your digital asset has evolved into a liability.

Unable to relentlessly keep evolving your software portfolio, your business will lose competitive power and will fall behind. Your team will waste its time and focus on software that does not make a competitive difference. The experience of your users will become unremarkable, and you'll find yourself making excuses rather than delivering on your promise.

Only the fittest survive in a world that is being eaten by software. Those that do not evolve at the right pace in the right direction will see their software eaten by the world.

Making Evolution Sustainable

Sustainable evolution requires two more types of change, not fueled by the marketplace, that must be initiated internally (refer back to Figure 1):

1. **Refactoring.** The complexity of the program code of a software system can be reduced through a series of small changes that improve its structure but preserve its behavior. Performing such incremental code improvements is called "refactoring."[6] While refactoring takes effort, it does not provide any direct value in terms of new functionality. For this reason, developers may find it difficult to justify their refactoring efforts to their colleagues and managers. Nonetheless, timely and judicious refactoring is generally considered a best practice to keep code complexity in check and, hence, prevent the second law of software evolution (increasing complexity) from kicking in.

2. **Commoditization.** Modern software is not built in a vacuum. Rather, a range of generally available functionality from external software libraries, frameworks, and services is used as a platform on top of which new, innovative functionality is created. However, the boundary between "innovative" and "generally available" shifts over time. Pieces of functionality that you developed in-house several years or perhaps just a few months ago may have given you a competitive edge then. But others have taken notice and developed similar components, perhaps in a more cost-effective, smarter way.[7] And recognizing the general utility of this functionality, they may have made it available as a *reusable component*. Your once-unique functionality has become a commodity, and the smart thing to do is to exchange one for the other. Doing so will relieve your team from continuing to evolve that functionality, and you can instead focus on functionality that makes you unique and competitive. While swapping homebrew against commodity may involve substantial effort, it can prevent the fifth law of software evolution (conservation of familiarity) from killing the productivity of your developers.

Since refactoring and commoditization are not sparked by external pressures, they are easily forgotten or de-prioritized in favor of other types of change. They are motivated by long-term sustainability and require a degree of foresight that may feel antithetic to our fast-paced digital age. The judicious application of refactoring and commoditization is a critical success factor, nonetheless.

Mastering Software Evolution

So what are the essential capabilities for any organization to master software evolution, both on the engineering and on the leadership level?

Mastering software evolution means that leadership, the business, and engineers need to work together to balance the various evolutionary pressure factors in such a way as to *sustainably* outpace the competition. Sustainability means bringing new functionality to the market while keeping your underlying software assets healthy and nimble.

To achieve such balance is easier said than done. It requires a form of data-driven design and decision making that balances commercial and technical considerations. Let's review what is needed in terms of data, design, and decisions.

Data

To feed your design and decision-making processes, your organization needs to look for data in at least three directions:

1. **Look around.** At any moment, your organization needs to know what is available in terms of reusable libraries, frameworks, services, tools, and technologies. And you need to have an informed opinion on which of those may be useful to you. To get this information, you can tap into external sources, such as the ThoughtWorks Technology Radar.[8] For a selected shortlist, you should consider a more experiential approach, such as a hackathon.

2. **Look inside.** Your organization needs to have an up-to-date inventory of its software assets. This is not just a list of systems, but also their interdependencies and their properties, including volume, quality, rate of change, functional focus, and technical health. To get this information, your software development infrastructure needs to include measurement instruments, such as code scanners, architecture analysis tools, and metric dashboards.

3. **Look ahead.** Your organization needs to know which market needs you will likely need to satisfy in six to 12 months. To get this information, you need to have product owners on board that curate a high-level backlog of product ideas that can be detailed out into feature descriptions. To collect these product ideas, your product owners need to interact intensively with users, business representatives, thought leaders, and (whenever possible) competitors.

These three data streams provide the basis for design and decision making regarding software evolution.

Design

Your organization's software design process must go beyond the new functionalities you want to add. Design must also deal with restructuring and eliminating existing software components:

- **Refactor.** The purpose of refactoring is to improve the design of the system to counteract complexity increases (as described above), as well as to prepare for upcoming additions and deletions. Your software teams must master the art of refactoring and must have the mandate to invest the necessary effort.

- **Enhance.** An optimally designed enhancement not only takes into account *what* should be added in terms of functionality, but also *how* and *where* in the software the changes are to be implemented, given what is known about the current state of software components, their interdependencies, and their likely commoditization trajectory. (For example, if your next feature gives you a unique competitive advantage, its implementation should not be mixed in with low-level generic functionality; it should be designed to remain confined to higher-level, product-specific components.) Your product owners, architects, and developers must optimize enhancement designs together.

- **Eliminate.** The design activity should include determining what functionality to remove (or replace by commodity components), when, and how. Functionality that is no longer desired may have been inextricably woven into the code, such that significant refactoring is needed before it can be replaced. Good software design takes future replaceability of functionality into account.

Decisions

Only a small part of software development is about editing code, while a very large part is about fast, effective, and coherent decision making at all levels of the organization. The sheer number of decisions to be taken implies that not all decisions can be made at the top. A large degree of decentralization is needed, while coherence across the organization is safeguarded by shared goals that are set centrally:

- At the engineering level, teams and individuals are empowered to make fast, informed decisions about as much as possible. These decisions are constrained by shared goals.

- At the leadership level, constraints for decentralized decision making are set in terms of goals (what and when, not how). These goals must be continuously reviewed and any changes in these goals must be communicated clearly and broadly within the organization.

Litmus Test

So what are the critical questions you should ask if you want to determine whether your software evolution capability is ready for what lies ahead? After transformation comes evolution. Is your organization ready for the long haul? Here are some critical questions to ask yourself:

- **Does everybody know and agree on which of our software components give us competitive advantage?** Are the teams assigned to these components aware of what new functionalities likely need to be added in the next six months? Are required changes to these components made diligently and without compromising their structural quality?

- **How about the components that do not give us competitive advantage?** Are they few and well isolated? Do we have a clear plan and timeline for removing them or replacing them with commodity components? Is that timeline shorter than three months?

- **Do our engineers enjoy and master the art of continuous removal of the nonessential elements of the portfolio?** Are we refactoring the codebase continuously, and in small increments, to prepare for removal of noncore components? Are we removing code at the same pace as adding code? Do we celebrate decommissioning components even when we (recently) invested blood, sweat, and tears to create them?

If your answers are affirmative, you are able to keep your software healthy, nimble, and focused on what you need to compete. If not, you will gradually lose the ability to adapt quickly to new market needs, regulations, and technological opportunities. The world moves on while your software falls behind.

Conclusion

In this article, we discussed the inevitability of software evolution, pressured by market factors, such as the need to innovate, reduce costs, grow, comply with new regulations, and coevolve with other systems. Under these pressures, software evolution risks being self-defeating, leading to increasing complexity and diminishing competitive power.

To evolve software *sustainably*, organizations must balance these external pressures with an internal drive to continuously improve the software's internal structure (refactoring) and regularly swap out functionality developed in-house for commodity components supplied by others. To achieve this balance requires decentralized design and decision-making capabilities that feed off a steady stream of data about upcoming functional needs, external technology developments, and the current condition of your internal digital assets.

In our digital economy, the capability to evolve software, fast and continuously, is key. Either your software eats the world, or the world eats your software.

References

[1]Andreessen, Mark. "Why Software Is Eating the World." *The Wall Street Journal*, 20 August 2011.

[2]Xu, William, and Adrian Cooper. "Digital Spillover — Measuring the True Impact of the Digital Economy." Huawei and Oxford Economics, 5 September 2017.

[3]Lehman, Meir M. "Laws of Software Evolution Revisited." *Proceedings of the* 5th *European Workshop on Software Technology*, Springer, 1996.

[4]Visser, Joost. "Change is the Constant." *ERCIM News*, 2 January 2012.

[5]Lehman (see 3).

[6]Fowler, Martin. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.

[7]Boulding, William, and Markus Christen. "First-Mover Disadvantage." *Harvard Business Review*, October 2001.

[8]ThoughtWorks Technology Radar (https://www.thoughtworks.com/radar).

Joost Visser has held various leadership roles for the past 12 years at the Software Improvement Group, a technology-based consultancy firm that helps organizations get and remain in control of the software their businesses rely on. He has written numerous publications on measuring and managing software quality and economics and is the author of Building Maintainable Software and Building Software Teams. *Dr. Visser is Professor of Software Science with a focus on large-scale software systems at Radboud University, the Netherlands. He can be reached at j.visser@cs.ru.nl.*

INFINITE LOOP

Software as the Ouroboros: Implications for Software Developers and Business Leaders

by Sunil Mithas, Kaushik Dutta, and San Murugesan

Without a doubt, software has become pervasive and indispensable. It is now everywhere and has impacted almost every aspect of our day-to-day activities and nearly every industry. Supported by such technologies as cloud computing, the Internet of Things (IoT), and artificial intelligence (AI), software has revolutionized the world. It continues to transform business, education, healthcare, banking, and many other key sectors, including government and politics. Indeed, the World Economic Forum's Global Agenda Council on the Future of Software & Society identified 21 examples of software-enabled changes that will strongly affect "human health, the environment, global commerce and international relations."[1] As the council's report highlights, "We are entering a time of momentous societal shifts brought on by advancements in software.... These changes will impact people around the world."

While the rise of software and its valuable influence is now common knowledge for most, what many people — even IT professionals and business executives — don't recognize is that software, particularly in the last two decades, has also transformed the software industry. In other words, software has "eaten" or auto-cannibalized[2] software, much like the ouroboros, the mythical emblematic serpent of ancient Egypt, India, and Greece, eating its own tail and being reborn from itself (see Figure 1).[3] The expectations end users have of software today are significantly different than those they had just a few years ago. Previously, users expected software to perform predefined and preprogrammed functions, such as automating business processes. Today, however, users expect software to be both smart and adaptive, changing itself (like the ouroboros). The purpose of this article is to articulate the nature of these rising expectations and examine what managers should focus on in developing newer software.

The impact of the ongoing transformation of software and the software industry is — and will continue to be — significant and widespread. Those that fail to pay attention to the next frontiers in software are putting themselves at risk. Thus, organizations and software developers alike should ready themselves for this new world of software. Understanding and adapting to the new software landscape, collaborating with major global partners as well as startups and "crowds," and being continuously innovative have become ever more important. Being prepared to navigate the new software landscape requires awareness of its ongoing changes and an understanding of their implications.

We begin with a brief outline of how software has evolved and what has changed in the software arena, particularly in the last few decades. Next, we examine how, over the years, software has transformed itself and its own development. We also discuss the implications of software evolution and offer recommendations on how business leaders can embrace and adapt to a new era in software.

Software: What Has Changed?

Many aspects of software have changed in the last few decades, particularly since the advent of personal computers in the 1980s, the World Wide Web in the

Figure 1 — The ouroboros.

1990s, and the widespread use of mobile and cloud computing in the last decade or so. Other key changes have been the massive trend toward outsourcing and offshoring in the 2000s and the widespread use of social media beginning in the 2010s. In turn, social media, along with mobile and cloud computing, created a trend toward the consumerization of IT that has challenged the software industry and IT departments.

Software companies and IT departments responded to some of these technologies and trends by making dramatic changes in software development methodologies and initiating Agile methods and design thinking approaches that create closer collaboration among software developers and customers to develop and improve software. In some cases, the need to address constant customer feedback and facilitate closer collaboration has brought software development back in-house (as part of a backsourcing initiative) and has promoted a hybrid model of computing that includes both on-premise and cloud computing.

We have also seen a parallel trend emerging toward the use of AI for software development in two primary roles: (1) AI as a tool to program software, and (2) AI as the software itself (aka Software 2.0[4]). In the first role, AI directly writes program code or indirectly helps human programmers to write program code; in the second role, AI *is* the software, and the software gets trained, eliminating the need for coding. Both roles exemplify how software is rapidly changing itself.

A key trend today is autonomous software, where software has the ability to change itself. The crudest such example is the automatic software update, where software (such as the Windows operating system or mobile device application) downloads periodic updates and replaces itself with the newer version of the software. A more futuristic scenario already underway is to use AI and the cloud to identify any potential bugs or issues and to fix those automatically without human intervention.

On the whole, software development has undergone major transformations over time. Newer developments in software often feed on themselves, rendering previous developments and approaches obsolete, just as the ouroboros metaphor suggests.

Software Evolution in Recent Decades

To understand the evolution toward autonomous software, we must realize that the most significant change in the last decade has been the availability of increased bandwidth to connect hardware across distant geographical regions, keeping in mind that hardware supports software. In 2018 alone, the average Internet speed in the US grew by 40%.[5] From 2007 to 2018, average Internet speed grew from 3.5 Mbps in 2007 to 18.5 Mbps at the end of 2017.[6] Such massive growth in Internet speed has enabled the growth of the cloud, which allows accessing of remote applications from anywhere in the world. This improved connectivity has also been a catalyst to allow applications to run in a distributed fashion. Both of these advances have enabled organizations to use the cloud and massively parallel distributed systems such as Hadoop and Spark to store and analyze large volumes of data at very low cost. Figure 2 depicts the evolution of software over the past decade, powered by technical infrastructure (network bandwidth leading to cloud).

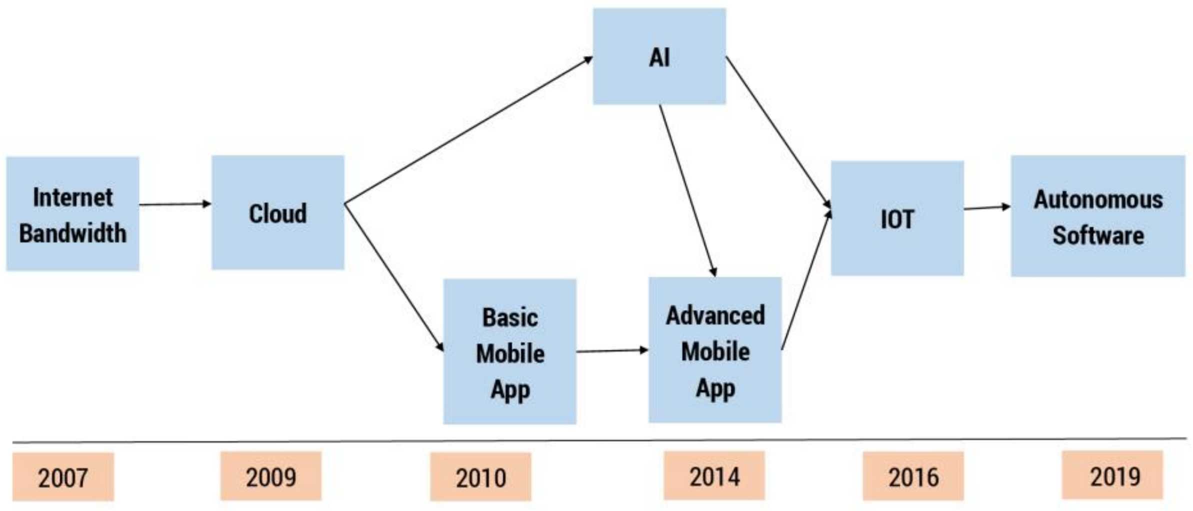

Figure 2 — Evolution of software in the last decade.

The ability of organizations to run massively parallel distributed systems in a very cost-effective way in the cloud has, in turn, given birth to the ubiquitous application of AI. Interestingly, the fundamentals of AI were developed back in the 1960s,[7] but hardware limitations restricted its growth and use. Easy and fast hardware connectivity together with a distributed software platform on top has made the application of AI a reality in today's world. Even small and medium-sized organizations, as well as startups, are applying AI to solve new problems.

Although Moore's law as we knew it no longer holds, that has not held up the growth of software due to the shift from a centralized system to a globally distributed system and the distributed nature of today's applications.[8] *Today's managers need to think of software that can run anywhere, can be accessed from anywhere, and can be scaled limitlessly.*

Increased bandwidth has played a major role in opening access to computing devices to the masses. Though mass access to computing devices is commonly credited to mobile devices such as smartphones, iPads, and tablets, one needs to remember that netbooks[9] and Palm devices[10] were around in the late 1990s. It is the improved bandwidth and the cloud, however, that have made smartphones and other mobile devices vastly attractive to the masses. These devices provide virtually everyone with unfettered access to information and computational power, in contrast to an earlier period when consumers needed a bulky and costly computing device to access applications and information that were otherwise out of reach. Mobile device–based apps, along with cloud-based computation, have given most consumers access to AI and other complex applications. *Managers need to think about how these capabilities can be used to reach the mass of consumers.*

Consider the current popularity of the IoT. Easy connectivity allows IoT-based software applications to reach consumers in ways that were only science fiction in the 1960s. Amazon's Alexa can order groceries for you because it knows what is in the refrigerator. The software in a Tesla car can report problems and fix the relevant software per instructions downloaded from the cloud. Software is now embedded in every device consumers use. This software not only enables device connectivity but also monitors the health of the device and autocorrects itself. The software in these devices can now even predict failure before the failure occurs. Think about a scenario where your air-conditioning unit tells you to find a service technician to address a few issues before it fails.

AI and the cloud are playing an important role in making possible self-maintained, auto-corrected software that can identify defects and take action. Such software requires massive data collection, data processing, and application of AI. Ubiquitous access to the cloud and enormous bandwidth availability from edge devices in the IoT make this scenario possible. Examples include Tesla cars and smart home devices, such as washing machines, refrigerators, and air-conditioning units.

New generations of self-managed software have created new expectations on the part of consumers. Consumers are no longer willing to wait for a service technician or to take a product (such as a car) to a service center. Consumers want problems with products and services to be taken care of without disruption and as efficiently as possible. One vivid example is Tesla's recent brake problem, which traditionally would have required the car to be taken to a service center; instead, Tesla fixed the problem through a remote software update.[11]

In today's world, managers shouldn't just develop software. *Managers need to integrate software with other devices and platforms, such as IoT devices and smart devices, and make that software self-manageable (i.e., autonomous).* Without adopting self-manageable, self-evolving, self-maintaining, autonomous software, enterprises cannot thrive in today's new world.

Today's managers need to think of software that can run anywhere, can be accessed from anywhere, and can be scaled limitlessly.

Implications of Changes in Software Development

The implications of the changes discussed above are enormous. In this section, we discuss three of those implications: (1) more pervasive and affordable software; (2) dual strategies; and (3) the redefined role of IT professionals.

1. More Pervasive and Affordable Software

First, software has become more pervasive and affordable. Moreover, software development and deployment — and the software business in general — have become more democratic, as evidenced by the ability of individual software developers to create apps, fix bugs, or make improvements (e.g., the open source movement).

Software no longer means the dominance of big software companies that have the infrastructure to distribute their software. Furthermore, software development tools have evolved to be more AI- and cloud-based. Examples include GitLab, Ansible, Packer, Nagios, Puppet, and ELK. Further still, the evolution of intelligent software has made software development easy and possible even for small software companies with fewer resources. Such small companies and startups can now develop and distribute software at a scale that no one could imagine 10 years back.

Software developers need to be prepared for a world in which AI will increasingly perform lower-level programming tasks.

2. Dual Strategies

Second, firms are having to innovate with high quality and high velocity at the same time, as illustrated by Apple, Google, and Amazon, to meet or create customer demand. These firms often follow "dual strategies," in contrast to conventional "either-or" strategies, such as either efficiency or innovation, or either exploitation of current resources or exploring and embracing new opportunities. However, executing dual strategies is not easy, and successful execution requires a new approach called "disciplined autonomy."[12] Disciplined autonomy is defined as the extent to which an organization adopts work templates or standards while providing sufficient autonomy to employees and developers.

One way of thinking about disciplined autonomy in IT projects is to realize that the traditional focus of IT project management has been on discipline, evident in waterfall-like approaches and process maturity frameworks such as CMMI. In contrast, newer approaches, like Agile and Scrum, allow individuals and teams greater autonomy to respond to the volatility of business environments and changing customer needs. Such disciplined autonomy techniques are particularly valuable in uncertain environments. The value of software-based strategies lies in enabling managers to pursue disciplined autonomy.

At the level of platforms, Alphabet, Amazon, Facebook, and Apple appear to demonstrate disciplined autonomy in their platform strategy when they allow third-party developers to provide complementary solutions. In such cases, platforms leverage outside innovation by granting considerable autonomy to third parties while encouraging desirable behaviors through governance and APIs. This notion of disciplined autonomy also applies to conglomerates with loosely connected firms or business units. For example, Google was reorganized as a subsidiary of Alphabet to provide it autonomy within the overall organizing logic that Alphabet provides. Conglomerates like GE and the Tata Group have followed a similar approach to grant business units sufficient autonomy in their respective businesses while leveraging potential synergies. Other approaches to create disciplined autonomy include Humana's "Palo Alto culture" in Kentucky;[13] use of autonomous squads arranged in circles or subcircles at Zappos;[14] and squads, chapters, tribes, and guilds at Spotify.[15]

3. Redefined Role of IT Professionals

The third implication of changes in software development approaches is the redefined role of and demand for IT professionals.[16] Increasingly, the use of AI for software development raises fears about job losses for programmers, a fear that is not totally unfounded. Software developers need to be prepared for a world in which AI will increasingly perform lower-level programming tasks. That does not mean that all software jobs will disappear, as Nobel Laureate and father of AI Herbert Simon feared back in the 1960s.[17] Indeed, AI can create new jobs or change the nature of activities that a software developer performs.[18] For example, AI assistance can help human programmers avoid coding errors by acting as a pair-programming partner. Even if AI were to completely replace the software code in relatively stable tasks or situations, we would likely still need conventional programming in more dynamic or creative environments to delight customers or serve their latent needs. In such situations, software developers will design and develop the architecture that brings together AI modules to solve a problem. They will also focus on data governance and activities requiring judgment and creativity and will address ethical questions relating to bias and discrimination. Moving ahead, developers will write software with the assistance of AI and the cloud and, increasingly, the software they are writing will be designed to be autonomous.

Role of Senior Leaders and Managers

So what do changes and developments in the software industry mean for senior leaders and managers? In this section, we discuss several steps they need to take to avoid the curse of ignoring the need for transformation that afflicted incumbents like Borders and Blockbuster.

First, *all managers must develop a vision that embraces software-driven strategies, recognizing that, increasingly, it is software that powers their business processes and provides competitive advantage*. Following on from that, managers need to articulate their software strategy as part of their business strategy by questioning and abandoning conventional strategy concepts based on the logic of "tradeoffs" in favor of newer ways of thinking based on the logic of "tradeons," because software can allow firms to pursue seemingly paradoxical objectives such as revenue growth and cost reduction at the same time.[19]

Second, *managers should continuously transform their organizations by scanning for and intelligently deploying new technologies*. They should avoid handicapping themselves by making imprudent use of outsourcing when what is being outsourced involves skills critical for the organization's future. Moreover, they should use configurational logic,[20] which supports multidimensionality in thinking about strategies and governance processes, because it is not just one lever that provides competitive advantage, it is simultaneously pulling multiple levers that allows organizations to occupy profitable and sustainable niches.[21]

Third, *managers should pay attention to governance processes to ensure successful deployment of their strategies and should become involved in the careful consideration of IT decision rights (i.e., who decides what), the structure and role of the IT department, how much to spend on IT, and how to deliver IT services internally and externally.* Furthermore, they should think of their governance system as a platform for integrating strategic initiatives, similar to an operating system, which allows a variety of applications to be built on a common platform.[22]

Fourth, *managers must become involved in executing IT projects*, which requires (1) being aware of technology evolution; (2) making informed decisions regarding technology upgrades; and (3) helping to adopt, diffuse, and exploit IT systems.

Fifth, *managers need to adopt, where needed, software that can run anywhere, can be accessed from anywhere, and can be scaled limitlessly*. Managers need to use newer technologies, such as AI and the cloud, to achieve these requirements.

Sixth, *managers need to determine how to reach the mass of end users with AI and cloud-based software leveraging mobile devices*. Managers must integrate that software with other devices (e.g., smart devices and IoT devices) by means of AI, the cloud, and a high-bandwidth network and make the software self-manageable (i.e., autonomous) so that it can adapt itself to a given context and repair itself.

Managers should continuously transform their organizations by scanning for and intelligently deploying new technologies.

Finally, *managers must realize that no one technology or software by itself provides a competitive advantage*. Managers must empower their organization to ask critical questions related to newer technologies and their business relevance amid changing customer tastes that leads to creating a data-driven, decision-making culture that will foster organizational survival.

Conclusion

Ongoing changes in software development approaches and continuing advances in AI will bring significant transformation to the IT profession and the work of software developers. To avoid becoming obsolete, software developers must stay abreast of new technology developments to keep their skill sets current and relevant. Senior leaders need to be aware of software trends and their implications and be innovative in effectively embracing both human intelligence and AI to solve business and societal problems and to leverage the new opportunities that software advances bring.

References

[1]Global Agenda Council on the Future of Software & Society. "Deep Shift: 21 Ways Software Will Transform Global Society." World Economic Forum, November 2015.

[2]Mehra, Rohan. "Autocannibalism Is When You Eat Bits of Your Own Body." BBC, 13 December 2016.

[3]Bekhrad, Joobin. "The Ancient Symbol That Spanned Millennia." BBC, 4 December 2017.

[4]Karpathy, Andrej. "Software 2.0." Medium, 11 November 2017.

[5]Molla, Rani. "US Internet Speeds Rose Nearly 40 Percent This Year." Vox, 12 December 2018.

[6]Holst, Arne. "Average Internet Connection Speed in the United States from 2007 to 2017 (in Mbps), by Quarter." Statista, 13 August 2018.

[7]Solomonoff, R.J. "Some Recent Work in Artificial Intelligence." *Proceedings of the IEEE*, Vol. 54, No. 12, 1966.

[8]Simonite, Tom. "Moore's Law Is Dead. Now What?" *MIT Technology Review*, 13 May 2016.

[9]Finnegan, Matthew. "Slide 7: Laptops, Tablets, and Smart-watches: The Evolution of Mobile Computing." *Computerworld*, 16 February 2015.

[10]Finnegan, Matthew. "Slide 8: Laptops, Tablets, and Smart-watches: The Evolution of Mobile Computing." *Computerworld*, 16 February 2015.

[11]Marshall, Aarian. "Tesla's Quick Fix for Its Braking System Came from the Ether." *Wired*, 30 May 2018.

[12]Mithas, Sunil, Thomas Kude, and Sorel Reisman. "Digitization and Disciplined Autonomy." *IT Professional*, Vol. 19, September/October 2017.

[13]Loftus, Tom. "Can You Put a Little Palo Alto Into an Insurer in Louisville?" *The Wall Street Journal*, 28 April 2015.

[14]Berman, Dennis. "Tony Hsieh Tells How Zappos Runs Without Bosses." *The Wall Street Journal*, 26 October 2015.

[15]Rigby, Darrell K., Jeff Sutherland, and Hirotaka Takeuchi. "Embracing Agile." *Harvard Business Review*, May 2016.

[16]Mithas, Sunil, Thomas Kude, and Jonathan Whitaker. "Artificial Intelligence and IT Professionals." *IT Professional*, Vol. 20, No. 5, September/October 2018.

[17]Simon, Herbert A. "The Corporation: Will It Be Managed by Machines?" In *Management and Corporations 1985*, edited by Melvin Anshen and George Leland Bach. Praeger, 1975.

[18]Mithas, Kude, and Whitaker (see 16).

[19]Mithas, Sunil, and Roland T. Rust. "How Information Technology Strategy and Investments Influence Firm Performance: Conjecture and Empirical Evidence." *MIS Quarterly*, Vol. 40, No. 1, March 2016.

[20]Configurational logic relies on the notions of conjunctural causality (as opposed to the effect of just one focal variable), equifinality (multiple pathways to achieve an outcome), and asymmetric causality (if the presence of a factor is necessary for success, that by itself does not mean that absence of that factor will necessarily lead to failure). These notions are sharply distinct from dominant conventional notions that rely on singular focus on one variable, unifinal outcomes (only one way to achieve an outcome), and symmetric causality.

[21]Park, YoungKi, and Sunil Mithas. "Organized Complexity of Digital Business Strategy: A Configurational Perspective." *MIS Quarterly*, forthcoming, 2019.

[22]Mithas, Sunil, and F. Warren McFarlan. "What Is Digital Intelligence?" *IT Professional*, Vol. 19, July-August 2017.

Sunil Mithas is a Professor in the Department of Information Systems and Decision Sciences, Muma College of Business, University of South Florida. Previously, he taught at the Robert H. Smith School of Business, University of Maryland, and has held visiting positions at UNSW Business School, Australia; University of Mannheim, Germany; and University of California, Davis. Dr. Mithas is among the top IS scholars in the world, and his interdisciplinary work has appeared in premier business journals. He has worked on research or consulting engagements with various organizations, including A.T. Kearney, Ernst & Young, Johnson & Johnson, the US Social Security Administration, and the Tata Group, and is a frequent speaker at industry conferences for senior leaders. Dr. Mithas is Senior Editor of MIS Quarterly *and* Production and Operations Management; *Department Editor of* Management Business Review; *and serves on, or has served on, the editorial boards of* Information Systems Research *and* Journal of Management Information Systems. *His papers have won best-paper awards and have been featured in various practice-oriented publications and websites, such as* MIT Sloan Management Review, CIO, *and Bloomberg. Dr. Mithas earned his PhD from the Ross School of Business, University of Michigan, and an engineering degree from IIT Roorkee, India. He can be reached at smithasusf@gmail.com.*

Kaushik Dutta is a Professor, Department Chair, and Muma Fellow in the Department of Information Systems and Decision Sciences, Muma College of Business, University of South Florida. He has 22 years' professional and research experience in the field of enterprise IT infrastructure, data analytics, and big data systems. Dr. Dutta's current interest is in the area of mobile advertisement, healthcare, and the application of blockchain in enterprise applications. He has published 35 journal articles and 64 peer-reviewed conference publications and holds patents in the areas of IT infrastructure, caching, and cloud security. Dr. Dutta has received about US $2 million in funding from public and private organizations for research, student projects, and university IT infrastructure. He has served as a reviewer of many IEEE, ACM, and INFORMS journals and conferences. Previously, Dr. Dutta was Associate Professor at National University of Singapore and Florida International University, and he was CTO of Mobilewalla, a Madrona-funded company that developed a big data–based mobile data platform. He can be reached at duttak@usf.edu.

San Murugesan is a Senior Consultant with Cutter Consortium's Data Analytics & Digital Technologies practice, Director of BRITE Professional Services, and an Adjunct Professor in the School of Computing and Mathematics, Western Sydney University, Australia. He is Editor-in-Chief Emeritus of IEEE's IT Professional. *Dr. Murugesan has four decades of experience in both industry and academia, and his expertise and interests include AI, the Internet of Everything, cloud computing, green computing, and IT applications. He offers certificate training programs on key emerging topics and keynotes. Dr. Murugesan is coeditor of a few books, including* Encyclopedia of Cloud Computing *and* Harnessing Green IT: Principles and Practices. *He is a member of the COMPSAC Standing Committee and a fellow of the Australian Computer Society. Dr. Murugesan held various senior positions at Southern Cross University, Australia; Western Sydney University, Australia; the Indian Space Research Organization; and also served as Senior Research Fellow of the US National Research Council at the NASA Ames Research Center. He can be reached at smurugesan@cutter.com.*

Transformation Starts with the Team

by Paul Pagel

As software continues to work its way into every corner of our daily lives, no industry will be spared. While many new companies get their start every year as digital-first organizations, many predigital companies that helped forge the world as we know it are facing the more challenging task of adapting to this changing tide. Industries such as construction and logistics are tackling not only the pressure to adapt to digitally driven ways of working, but also to continue innovating. And software systems are not just the new baseline; they're uncovering insights and creating opportunities to expand your business's capabilities and to offer new services you might never have predicted.

These innovations don't come to businesses magically, though. They require hard work and discipline to craft software systems capable of supporting them. Like most new capabilities, the execution matters as much as — or more than — the strategy. Building the correct software execution muscle starts with building the right team. Transitioning from an IT labor strategy to a modern software labor strategy is key.

Why Is It So Important to Start with Labor Strategy?

These Employees Will Manage Your Biggest Cost Center

Technology is becoming the largest cost center for more and more companies, and these costs are still growing. A survey last year of 500 US executives from privately held companies discovered that 57% of mid-market businesses were spending more on tech than they did the year before, while a third of respondents were spending more than 5% of their annual revenues on technology.[1] Another survey that collected responses from nearly 4,000 CIOs and technology leaders across 84 countries found similar trends: 86% of respondents expected their IT budgets to increase or stay the same, with 47% expecting to increase headcount on their IT teams.[2]

These investments are not purely to support innovation. While newer initiatives, such as protecting against cybersecurity threats and launching new digital initiatives, account for a large share of the projected US $4 trillion in IT spending worldwide, a CNBC report identified enterprise software as the fastest-growing area of tech investment.[3] Companies that don't consider themselves tech hubs are being forced to keep up by building modern infrastructure to allow for rolling upgrades, cloud migration, and integration of new insights into a system not designed to support constant changes.

Like most new capabilities, the execution matters as much as or more than the strategy.

None of this work is simple, and companies need to invest in experts to ensure they transition their software systems to successfully take advantage of the new technology. The biggest cost and risk in these continual transitions are associated with the team and the team members' expertise. And none of these costs is likely to decrease any time soon.

A Different Way of Working

IT investments cause a ripple effect that goes beyond a company's balance sheets and into its everyday operations. As mentioned earlier, software execution matters as much as software strategy, and high-quality software execution requires a new way of working that is foreign to most traditional companies.

Traditional businesses have typically grown successful through hierarchical processes. Specific tasks, goals, and priorities are handed down from executives to managers and then delegated to teams. Higher levels of authority in the business in many ways dictate the teams' daily work.

Today's modern, high-quality software cannot be built this way. Software teams must be highly collaborative, with frequent feedback loops of multilateral communication. While executives and managers can help set a vision, it is the software team of product owners, designers, and developers who will make the thousands of decisions that will determine the end results. Consequently, the feedback loops and specialties will develop inside the team rather than going through the hierarchy.

You need to build a team of complementary skill sets and give that staff the time and space to build expertise to handle unique problems and constraints.

Transaction Costs Are Very High

Building teams capable of executing at this level is not easy. The market has a shortage of experienced, highly skilled software developers, and even when you do manage to hire the best and brightest developers and designers, they will still require a significant investment in onboarding and training to get them comfortable in your unique domain, ecosystem, and team workflows. The transaction costs in recruiting, training, and retaining this talent start high and will grow with volatility.

This underscores the importance of building your software team the right way, from the ground up. If you invest in hiring the right people to work on the right problems, the culture created will decrease the transaction costs involved in software talent.

Why Is Software Delivery Talent So Difficult to Find?

High-Complexity Activity

Computer science is fractal in nature. At each layer of a system's stack, and at each level of abstraction, there is a dizzying amount of complexity that is easy to get lost in. Your team will need to understand complexity at each of these layers well enough to understand the tradeoffs of different design decisions and how they might impact larger decisions such as scope and flexibility as your system matures.

Trying to manage your labor within this ecosystem is equally challenging. It's often difficult to know how long a task will take until the developer has begun building the feature and you can see how the codebase responds. Your leadership team will need to account for this ambiguity by growing comfortable with more uncertainty than you're used to when building budgets, forecasts, and project plans.

Generalist and Specialist Skill Sets Requirement

It's unrealistic to think any one developer can understand all the complexity described above. You need to build a team of complementary skill sets and give that staff the time and space to build expertise to handle the unique problems and constraints across your system.

Your team will need to feature two different kinds of expertise: breadth and depth. You will need developers with a detailed understanding of the holistic system: where are the important interaction points, what are the key dependencies, and how is the system shaped?

You will also need team members with expertise in some of the specialties in the system; for example, operations, front-end development, or data engineering. The specialist expertise required may change depending on the nature of the problems you need to solve, but it is critical to build the team with the right skill sets and to ensure they are fully utilized.

Both types of experts are essential to your team and to your ability to understand and plan for the complexities within your system in a responsible way. But they cannot exist in a silo. These two types of experts are most valuable when they interact with each other. This adds a layer of complexity to team management decisions, as there cannot be a full division of labor across your system. The different team members focusing on different problems need to constantly learn from each other and collaborate if they are to achieve a high-quality solution optimized for your system.

What makes this arrangement even more challenging is that neither of these types of expertise is static. Every six months, thought leaders introduce new architectural concepts that can improve your system, and new tools will show up to support new languages. Becoming an effective expert in even one specific component requires actively looking outside of your company's code for new ways to constantly improve the system.

No Formal Path to Trained Expert

With so much fluctuation within an expertise, it should not be a surprise that there is no standard path to becoming an expert. There is no straight line from novice to expert. Every developer's journey to mastery goes through fits and starts, leaps and setbacks, and will only result in true mastery if every step is built on a solid foundation. The immaturity of the industry means there is no consistent institution or curriculum to ensure the foundation exists.

While everyone walks his or her own path, the industry has failed to establish an agreed-upon way to measure progress, and there is no standard way to gauge competency in a software discipline. Many in the industry have adapted by placing an emphasis on length of experience. If you scroll through any of the thousands of job postings looking for software developers, you will see teams looking for developers with some specified amount of experience — 5, 10, even 15 years' professional experience using technologies that have barely been around that long.

This metric, however, is woefully inadequate for judging someone's actual capabilities. Simply delivering software is an insufficient metric, and it becomes misleading in predictable ways. Software development 10 years ago looked a lot different than developing today's modern software systems. Every year we discover new ways of architecting our code, gain new tools to build on new platforms, and discover new best practices to apply to these. As a result, the experience of developing a particular product 10 years ago is often less relevant than the lessons you would have gained from developing a similar product just 10 months ago.

Throughout my career, I have seen dozens of software crafters with only a few years' experience mentor others who — though they have been industry professionals for decades — had never been taught to follow the more foundational principles and practices that lead to higher-quality software. While the experienced developer might have a better recall of idiomatic nuances, the inexperienced one may be the developer who understands how to apply best practices and will drive your team to success.

Another popular approach to training developers for success has been the influx of coding bootcamps and similar programs. These programs are good crash courses for developers to quickly get up to speed on the skills they need to contribute to a particular software stack, but the expediency of this training is often chasing after trailing indicators of what the industry wants.

One lesson we can take away from these programs is that there's no shortcut to mastery. It requires an ongoing investment in continual learning at every stage of a developer's career.

Fast-Changing Skill Sets

These problems are endemic to software because the industry itself has immature and changing best practices. Software is still a relatively new discipline, and the frequency of change in tools and technologies has made it difficult to cement established best practices the way other disciplines have, much less have those practices become commonplace around the world.

Without strong guardrails around the ways we work, teams are often tempted to take shortcuts to deliver more quickly and please their stakeholders. This short-sighted strategy has led to enormous waste that riddles the industry.

Every corner cut at the beginning of a project has the potential to add exponential complexity later. Several months down the road, as you're collecting live user feedback and fine-tuning your business model, what might seem like trivial software changes will end up requiring massive refactorings or rewrites because so many layers of complexity will have been built on top of a decision your team never considered might need to change.

After being involved in helping hundreds of projects either get unstuck after years of stagnancy or trying to anticipate changes to a greenfield app, I have seen how following disciplined practices and processes makes all the difference.

What Do You Do About It?

Software teams need to be structured to deal with fragility. Fragility is a shortcoming that's easy to understand; when you try to add new features or modify the codebase in some way, it crumbles like a house of cards, with unintended side effects causing bugs throughout the system. The goal is to build talented teams that don't fall into this trap. Organizations must create a system that is not just sturdy and resilient to change, but actually thrives on change. A high-quality software system will respond to a failure by exposing new and stronger ways of designing and architecting it.

Investing in a network of skilled managers and mentors ensures that all team members have someone supporting their functional and career development and never feel that they are on their own.

To incorporate a software delivery capability, an organization should embrace this mentality throughout its entire operation. From the executive team on down to junior workers, your team should thrive on adopting new technology, practices, and processes. There are three primary ways companies can do this effectively, as we explore below.

1. Become a Training Organization

As technology becomes more chaotic, your business can approach this challenge as an opportunity rather than as a threat. Investing in your team's ongoing education will inspire team members to dive deeper and learn more about how to improve operations. While this training will not have an obvious or immediate impact on your bottom line or productivity, it will provide the support needed for your team to discover the most stable solutions, allowing for higher profitability and sustainability in the long run.

There are many different ways to implement this kind of training. Companies can offer budgets for attending conferences and taking advantage of other educational venues and materials and hold regular "lunch and learn" workshops to introduce new tools and concepts. However, it is important to move the professional development budget past these traditional ideas into creating a learning culture that has maximum impact. For example, companies can adopt weekly "10% time" reserved for professional development and learning; they can develop apprenticeship programs or dedicated mentoring programs. The important distinction is that sufficient space and investment are devoted to training for it to become part of both the work and the company culture, rather than an afterthought or yearly endeavor. Training should be modeled from the top down, with leaders visibly participating in growing their own skills and investing in learning as a core value.

There is a distinct competitive advantage in being able to grow your own talent through training and experience. Having a combination of expert practitioners (who can push your technology forward) and teachers (who can educate your team on how to keep up with any changes) can go a long way toward keeping your team engaged and productive for long careers. Expert teachers and mentors will also allow you to hire smart people from a wider range of backgrounds and train them to be engineers. This provides not just a competitive advantage in hiring but also helps you train new mentors and teachers and create a virtuous feedback cycle of mentorship that will guarantee a baseline of quality across your team as you scale your business.

One of the key factors in this strategy is making sure you're hiring for the right traits. Rather than using the application and interview process to check off past accomplishments and experiences, you should look for a curiosity and growth mindset, with the capacity to learn whatever your potential new hire doesn't already know. This type of hiring process is an imperfect science, as it requires diving quite a bit deeper than the stuff highlighted on someone's CV. However, if you can ensure the people you are hoping to train actually want to grow and learn, they are much more likely to remain engaged and successful employees throughout their careers.

2. Invest in Keeping Retention High

Complex workers are intrinsically motivated through autonomy, pursuit of mastery, and clear connection to the challenges they are solving. One of the best ways to keep retention high is to encourage and feed team curiosity. Your teams will stay engaged and invested if you give them interesting problems to work on and the support and opportunity to solve them.

It is important to consider carefully how you provide this support. Investing in a network of skilled managers and mentors ensures that all team members have someone supporting their functional and career development and never feel that they are on their own. Developers stuck on a problem need to have outlets where they can ask for and receive help.

These support systems should lend themselves naturally to a culture of feedback essential to making your team feel invested and secure in their role at your company. If team members are struggling, they should have support built into their regular work cycles. If they are performing extremely well, they should be recognized. It's important to celebrate successes and regularly remind your team that their work is appreciated.

It also helps to pay your team competitively and offer inclusive and flexible working policies. Many tech companies invest in flashy perks — like free lunches and onsite ping pong. There is nothing inherently wrong with these perks, but they should not come at the expense of policies that provide for a healthy work-life balance. When your company is flexible to different working arrangements and embraces policies that encourage outside interests, you're helping to establish an environment that is sustainable for long careers. If you provide support to employees at all phases of life, you're not only encouraging employees to stay but also helping attract people with diverse and complementary skill sets and background experiences.

3. Mix in Specialists from the Outside

While it might follow logically that a team fully dedicated to your software and your business's unique problems is a positive, teams can struggle when they fall too far into their own work silo. If your teams are completely absorbed in your software system, they aren't paying attention to developments in the community that could provide a simpler solution or even transform your employees' capabilities by adopting new strategies and tools.

Whether you bring in specialists from the outside to do one-off training presentations, to consult on the state of your software system, or to offer a more prolonged residency program, gaining insight into new ways of working and thinking through problems is invaluable. Such insights will not only stimulate your team but also ensure your organization is keeping up with modern standards and practices. It is important to break down the walls of the organization to ensure improvements are flowing in.

Conclusion

Yes, software is eating the world, but a successful strategy for adapting to this new reality will still revolve around attracting and growing the talented people who can drive innovation and manage your business's new technological foundation.

Investing in a holistic onboarding program and ensuring your team has the skills and resources to continue growing within their roles will go a long way toward making sure your business keeps pace with today's rapidly changing tools, processes, business priorities, and more. A human-centered approach will also protect your business from being swallowed up by today's constant change and will maintain more authentic and lasting engagements with both employees and your system's users.

References

[1]Bujno, Maureen, and Chris Jackson. "1 in 3 Mid-Sized Firms Lack IT Governance and Could Face Digital Disruption." *The Business Journals*, 8 December 2018.

[2]Pratt, Mary K. "How to Build the Next Generation of IT Leaders." *CIO*, 6 August 2018.

[3]Rosenbaum, Eric. "Tech Spending Will Near $4 Trillion This Year. Here's Where All That Money Is Going and Why." CNBC, 8 April 2019.

Paul Pagel is the founder of 8th Light and a leading voice in the software community. Under his direction as CEO, 8th Light has doubled in size twice and currently employs more than 150 software professionals across five offices. Mr. Pagel has also overseen the evolution of 8th Light's apprenticeship program, transitioned the company to employee ownership, and set strategy to bring a set of services to market. He earned an executive master's of business degree from Northwestern University's Kellogg School of Management and a bachelor of science degree in computer science from DePaul University. He can be reached at paul@8thlight.com.

The Evolving Role of Architecture in Digital Transformation

by Michael Papadopoulos and Olivier Pilot

The mantra of "disrupt or be disrupted" has created an increasing demand for organizations to be more adaptable and responsive than their competition. Over the last five to 10 years, the idea of "digital transformation" has led to an increased prevalence of Agile ways of working in today's enterprises. Agile promises to make teams more adaptable and responsive and to reduce products' time to value. As organizations "go Agile" to reap the benefits of these new ways of working, they restructure themselves to have end-to-end feature teams. Such feature teams (e.g., Large-Scale Scrum [LeSS] or Spotify squads) theoretically are capable of taking responsibility for the design, implementation, and evolution of end-to-end, customer-centric features in digital products or platforms.

Today's best architecture is essentially minimalist, messy, and inconsistent.

However, after trying this new organizational structure for a few months or years, organizations eventually stumble upon challenges and realize that the end-to-end feature teams aren't really working for them. Though this approach to digital transformation usually succeeds in positively addressing a number of mindset issues, a fundamental problem arises when organizations try to make feature teams work with traditional architecture (or architectural patterns). Organizations quickly realize that fundamental limitations in their technology assets — systems, infrastructure, and tooling — prevent them from reaping some of the most valuable benefits of the new, Agile ways of working. The weakest link is to be found somewhere else.

Digital transformation has hit a wall. The need for reinventing how we think about and approach architecture is becoming ever more prevalent, especially if an organization is to truly become Agile.

The Need for a Managed "Messy" Architecture

It is important to understand that software development has changed significantly in the last few years. Software is no longer built totally from the ground up without any dependency on some form of existing software solutions, nor does software come purely from all-encompassing, off-the-shelf packages configured to one's needs, spanning multiple business or technical domains. Software is now essentially built by "smart stitching" together already existing pieces of legacy code, new code, open source software and frameworks, and software-as-a-service (SaaS)/platform-as-a-service (PaaS) solutions, while leveraging core infrastructure-as-a-service (IaaS) components that also significantly affect functionality.

Thus, today's best architecture is essentially minimalist, messy, and inconsistent. Architecture should no longer aim to provide robust, detailed frameworks for mandated solutions and componentry. Rather, architecture should focus on strong general design principles and ensure that it provides guardrails for security, scalability, availability, elasticity, and maintainability (all the typical "-ity" nonfunctional requirements). Moreover, architecture needs to enable solutions that allow for rapid evolution of product and platform features, as well as experimentation with new technologies, by ensuring that all integrations are open as well as API- and event-driven. Architecture should also ensure that all technology decisions support safe, error-free, low-latency, continuous delivery to facilitate a rapid pace of change in a trusted manner.

Fundamentally, we need to recognize that time to value is the most important decision factor in today's world. Optimizing time to value through the ability to sustain a rapid pace of quality software delivery is what architecture's contribution to digital disruption needs to be all about. As long as the solution's reliability is assured, we should no longer care about duplication, solution "inconsistency," or anything that conflicts with the more traditional approaches of the past that have emphasized consistency and reuse (see sidebar, "What Use Is Reuse?"). In today's context, it is increasingly as relevant to design for (unexpected) change as to design the change itself.

Feature Team Limitations in Agile Transformations

Feature teams working in a traditional architecture cannot meet the agility expectations that Agile transformation promises. With a traditional architectural view, "feature teams" aren't actually feature teams. They focus on the top layers of the architecture, the user experience layer, and, in the best case, the API layer. They rely on large, shared medium- to high-complexity layers underneath, which are difficult to safely and quickly adapt and change. Not only does this mean that feature teams cannot truly own their "stack," it also means that the traditional promise of reusability and consistency is typically not kept, because only a limited separation of concerns by business domain exists in these shared layers.

Conway's law famously says that "*organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations.*"[1] When an organization decides to structure the teams that look after its digital products and platforms into feature teams, the technology stack must closely follow that structure. Most organizations, however, do not structure their technology stacks in this way, and the challenges that we observe today when trying to make changes at speed are usually attributable to that misalignment.

In the traditional architecture stack shown on the left side of Figure 1, products A, B, C, D are all dependent on a few stacks, and there is high reuse and high consistency. The right side of the figure shows that changing one of the layers to meet the needs of product A may introduce changes that break B, C, and/or D. This is why Band-Aid solutions (e.g., increased

What Use Is Reuse?

Consistency and reuse were a key requirement in even the near past mainly due to the operational burden of running multiple solutions — you needed people to manage the infrastructure, the databases, and the networks. However, with the advent of more automation, DevOps, and technological breakthroughs such as serverless moving us into low-impact operations, more solutions can be supported without the need for ever-larger operational teams.

Creating reusable software isn't free, however. To support *planned* reuse, teams need to expend more effort creating the software — more documentation, more planning, more tests. This leads to an increased time to value.

With one of the key benefits of reuse fading away and the key challenge to creating reusable software still in play, should we really continue to strive for it?

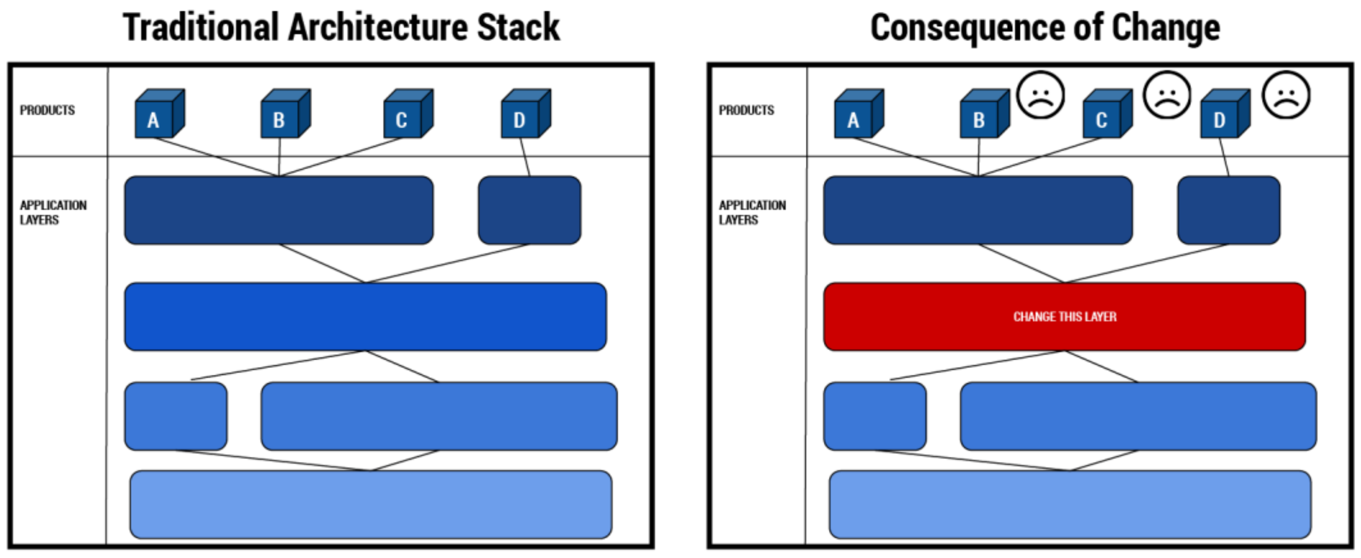

Figure 1 — The challenges of making changes in a traditional architecture.

dependency management, big room–planning events, Scrum of Scrums–style meetings, and giant dependency boards) are in vogue in some companies. Not only do dependencies introduce significant challenges regarding time to value, but they also usually introduce hidden costs, since nobody factors in the cost of additional project planning time and of time wasted in meetings devoted to ensuring that nothing gets broken.

Products should effectively be microservices – smaller, decoupled parts that are loosely bound together to create a whole application or product.

Thus, we need an architecture that is capable of supporting a feature team–based organization at the deepest level. To achieve that type of architecture, products should effectively be microservices — smaller, decoupled parts that are loosely bound together to create a whole application or product. Importantly, it is acceptable to have duplication in these microservices.

When each team has full end-to-end ownership of the stack, as shown in Figure 2, changes in one stack do not affect any other stack. There is no need for increased dependency management, big room–planning events, Scrum of Scrums–style meetings, and/or giant dependency boards. This simplifies management, allows for rapid change, effectively reduces time to value (the most significant effect), and eventually decreases costs (through reduced management overhead).

A significant percentage of reuse is still possible, however, if you use domain-driven design and the atomic design principle (i.e., everything begins with the smallest element of the interface: the atom).[2] We would argue that this is the correct approach to reuse as opposed to the monolithic layers frequently observed today. You can reuse and recombine your "atoms" to create entirely new elements (modules). These atoms on their own are not useful; in fact, they are not even deployed on their own; rather, they are shared building blocks. If you design your elements correctly, your atoms can become "shared libraries" across the duplicated components, as shown in Figure 3.

A level of orchestration still needs to happen on top of the atom, but that is where the specialization and decoupling from other services come in. Your atoms essentially become your underlying stack: your identity and access management service, your API gateway, your load balancer technology, and so on. Essentially, this way of thinking also helps you move toward "function as a service" and SaaS, enabling the rapid implementation and evolution of the digital products and platforms from which you decide to derive a competitive advantage.

For example, unless your business is in identity and access management, you probably should not spend time and effort creating an identity and access management stack; instead, you should just integrate an existing solution. The same criterion applies for all such technical component types (e.g., API gateway, database stores, data streaming, analytical engines).

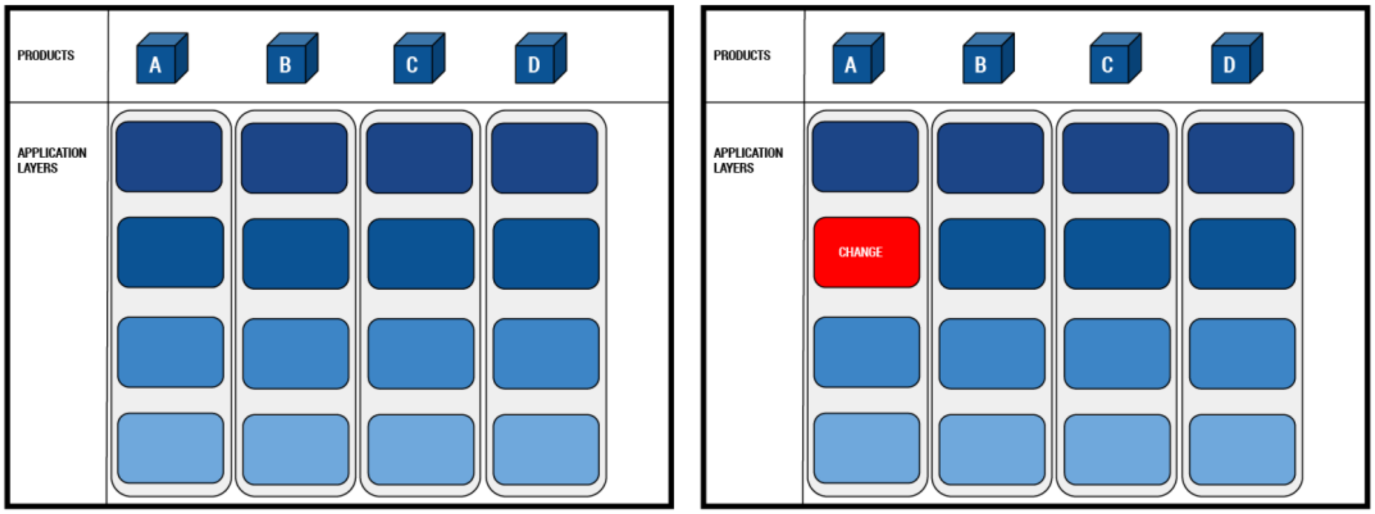

Figure 2 — End-to-end ownership of the technology stack by features teams.

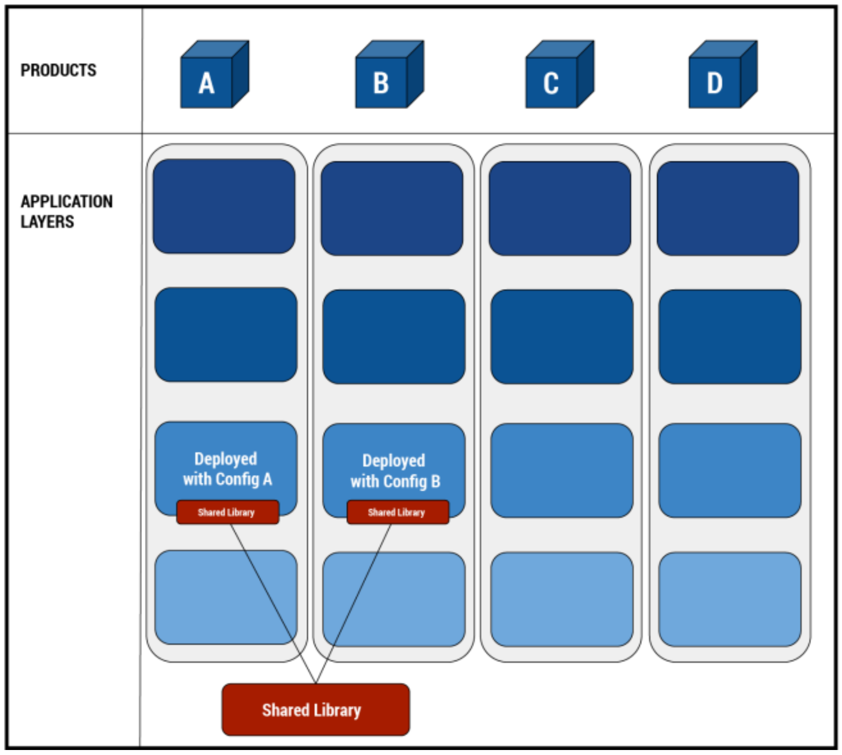

Figure 3 — Using the atomic principle to allow for reuse in a managed "messy architecture."

Pay Attention to Architecture for Successful, Sustainable Digital Transformation

Digital transformation has focused on evolving organizational structures and processes to move closer to what digitally native organizations have defined as end-to-end feature teams. These evolutions are clearly going in the right direction for any organization wishing to fulfill the promises of digital transformation. But they are not sufficient.

Once the top layers of an existing technology architecture have been transformed to support more Agile digital product and platform implementation, much work still needs to be done at a deeper level to align architecture and feature teams. To achieve this, one needs to accept that the speed to market promised by end-to-end feature teams can be achieved only if such teams have maximum autonomy regarding the technology stack on which they build. In a traditional architecture, this is usually difficult since the technology stack is organized in large, shared layers with the — often missed — objective of enabling maximum reuse of technology components. Such traditional architectures make the lives of feature teams difficult at best, since changes to a feature can easily break another team's feature.

Beyond Agile and Scaled Agile, architecture needs to be on the priority list of any organization wishing to go further in deriving competitive advantage from technology. Architecture's focus should change from simply driving consistency and reusability to allowing a safely managed, messy, occasionally inconsistent architecture that focuses on minimizing time to value. Indeed, modern architecture needs to enforce strong principles for the design and safe delivery of software at pace, while reusability is managed at the atomic, rather than the component or service, level. This modern architecture, we believe, is a key area that will differentiate the winners and losers of the next wave of digital transformation.

References

[1]"Conway's law." Wikipedia.

[2]Hacq, Audrey. "Atomic Design: How to Design Systems of Components." UX Collective, 28 June 2017.

Michael Papadopoulos is a Senior Consultant with Cutter Consortium's Business & Enterprise Architecture practice and Chief Architect of Arthur D. Little's UK Digital practice. He is passionate about designing the right solutions using smart-stitching approaches, even when elegance and architectural purity are overshadowed by practicality. Mr. Papadopoulos leads the scaling of multidisciplinary organizations by focusing on continuous improvement, establishing quality standards, and following solid software engineering practices. He mentors team members, leaders, and managers along the way. Mr. Papadopoulos is a strong advocate of the DevOps culture and Agile principles and has demonstrated experience in solving problems in challenging global environments. Coming from a development background, he remains highly technical, with hands-on involvement in code review, design, architecture, and operations. Mr. Papadopoulos has 15 years' experience in technology and digital consulting and has worked in a variety of sectors, including telecom, gaming, energy, and media. He can be reached at mpapadopoulos@cutter.com.

Olivier Pilot is a Senior Consultant with Cutter Consortium's Business & Enterprise Architecture practice and a Principal Architect with Arthur D. Little's UK Digital practice. He has broad experience across a range of projects involving enterprise architecture. His focus areas include digital strategy, Agile digital solution delivery, design and architecture, C-level advisory, and information systems roadmaps. Mr. Pilot's recent sample engagements include production of a technical and industrial strategy for a large European mobility provider to enter the "mobility-as-a-service" space; design and delivery of a real-time voucher decision and distribution engine for a FTSE 100 gaming and entertainment firm; delivery of digital concept store prototypes, demonstrating facial recognition, proximity marketing, and smart vending use cases to the board of a global Fortune *100 FMCG firm; working as design authority for the Agile delivery of a real-time monitoring and alerting solution to detect and assign issues in 40+ digital products of an FTSE 100 online education company; and serving as lead architect for a large international airline in the Agile delivery of an innovative, real-time, situational-awareness dashboard on top of siloed systems to enable coordinated decision making in disruption. Previously, Mr. Pilot worked at Atos Consulting and Cap Gemini. He holds a master's of engineering degree in IT from Ecole Centrale de Lyon. He can be reached at opilot@cutter.com.*

CUTTER Business Technology Journal

About Cutter Consortium