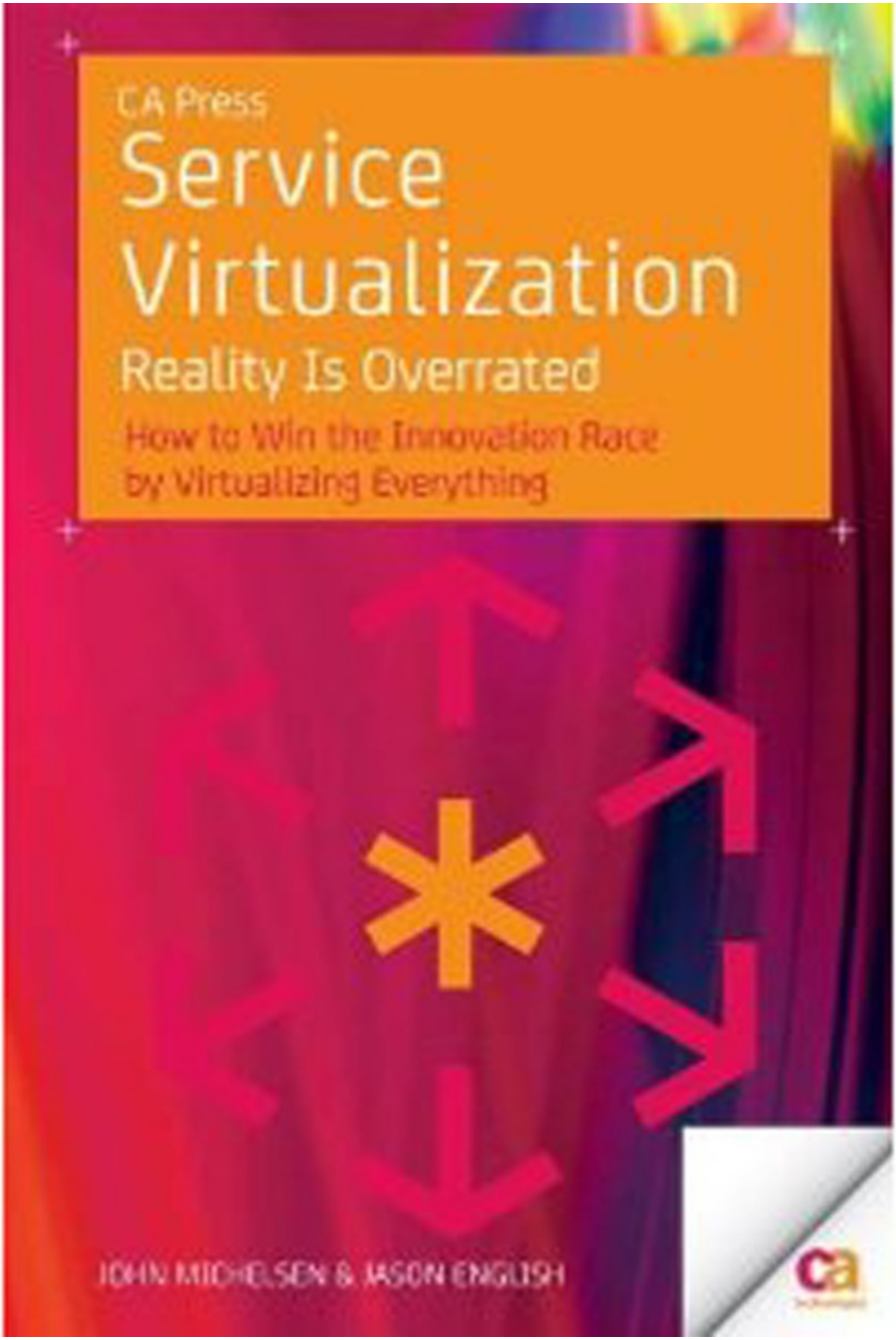

CA Press
Service Virtualization
Reality Is Overrated
How to Win the Innovation Race by Virtualizing Everything
JOHN MICHELSEN & JASON ENGLISH

SERVICE VIRTUALIZATION

REALITY IS OVERRATED

John Michelsen
Jason English

Apress®

The information in this publication could include typographical errors or technical inaccuracies, a
the authors assume no responsibility for its accuracy or completeness. The statements and opinio
expressed in this book are those of the authors and are not necessarily those of CA, Inc. ("CA"). C
may make modifications to any CA product, software program, method or procedure described in th
publication at any time without notice.

Any reference in this publication to third-party products and websites is provided for convenien
only and shall not serve as the authors' endorsement of such products or websites. Your use of suc
products, websites, any information regarding such products or any materials provided with su
products or on such websites shall be at your own risk.

FedEx® is a registered trademark of Federal Express Corporation.

To the extent permitted by applicable law, the content of this book is provided "AS IS" witho
warranty of any kind, including, without limitation, any implied warranties of merchantability, fitne
for a particular purpose, or non-infringement. In no event will the authors or CA be liable for any lo
or damage, direct or indirect, arising from or related to the use of this book, including, witho
limitation, lost profits, lost investment, business interruption, goodwill or lost data, even if express
advised in advance of the possibility of such damages. Neither the content of this book nor a
software product referenced herein serves as a substitute for your compliance with any laws (includi
but not limited to any act, statute, regulation, rule, directive, standard, policy, administrative orde
executive order, and so on (collectively, "Laws") referenced herein or otherwise. You should consu
with competent legal counsel regarding any such Laws.

ISBN-13 (pbk): 978-1-4302-4675-8

ISBN-13 (electronic): 978-1-4302-4672-5

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symb
with every occurrence of a trademarked name, logo, or image we use the names, logos, and imag
only in an editorial fashion and to the benefit of the trademark owner, with no intention
infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if th
are not identified as such, is not to be taken as an expression of opinion as to whether or not they a
subject to proprietary rights.

President and Publisher: Paul Manning
Acquisitions Editor: Robert Hutchinson
Technical Reviewer: Ruston Vickers
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editor: Rita Fernando

Copy Editor: Jennifer Sharpe
Compositor: Bytheway Publishing Services
Indexer: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6t Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please contact us by e-mail at info@apress.com, or vis www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotion use. eBook versions and licenses are also available for most titles. For more information, referen our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales. To place a order, email your request to support@apress.com.

Contents

About the Authors

John Michelsen, CTO, CA Technologies and Co-Founder, ITKO

John has lived his career helping enterprise customers push the leading edge of IT transformation to deliver on business outcomes and is a highly respected technologist who moves others to action. As the CTO of CA Technologies, John is responsible for technical leadership and innovation, as well as aligning CA's software strategy, architecture, and partner relationships to deliver customer value. With 12 patents awarded or in process and with market-leading inventions delivered in database, distributed computing, virtual/cloud management, multichannel web application portals, Service Virtualization (LISA®), and many other areas, John is a factory of innovation.

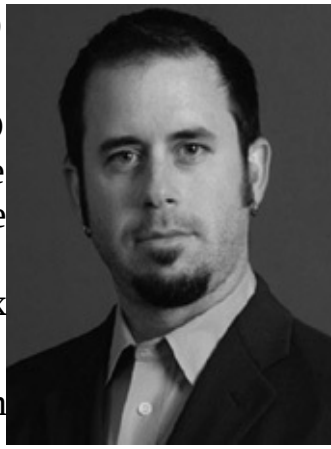
John is a frequent writer in leading trade publications and has presented at technology and busine conferences around the world. He joined CA Technologies in 2011 through its acquisition of ITKO, company he co-founded and drove successfully for 12 years. Prior to ITKO, John's broad technolog experience included leading SaaS and e-commerce transformations for global enterprises at Trilog and Agency.com. In his spare time, he enjoys vacationing with his family on islands or in th mountains, and he has also become an expert at country and Texas swing dancing.

Jason English, CA Technologies Director, Product Marketing and ITKO Historian

Jason joined ITKO (now a CA Technologies company) in 2004 as employee #3, functioning as the company's "Marketing Department" during many stretches of its existence, while handling other tasks from software UI design to sales support.

Before managing marketing for ITKO, Jason was Executive Producer of the in2action interactive consulting unit at i2 Technologies, where he was responsible for web marketing and messaging during a period of extreme growth, as well as working directly with major companies of every industry to build easy-to-use workflows for complex B2B collaboration systems. Prior to that, he served as one the first information architects, defining customer experience for Fortune 500 clients at the pioneerin interactive firm Agency.com. He has also designed, written, and scored soundtracks for international released computer games in addition to producing advertising and television commercials. H continues to compose and play music and remains an active brewer.

About the Technical Reviewer



Ruston Vickers, CA Technologies VP Research and Development and ITKO Co-Founder

As co-founder of ITKO (now a CA Technologies company) and leader of R&D efforts, Ruston has been instrumental in advancing the company's product suite from its inception. He manages all customer product deployments and is the lead developer of CA LISA's integration frameworks, which help clients gain tremendous extensibility and quality across many technologies within complex IT environments.

Prior to ITKO, Ruston designed and built solutions for world-class clients in energy, automotive, and travel industries while working at Agency.com and EDS. Early in his career, he developed some of the first highly interactive online experiences for ED General Motors, and Dr. Martens. He also worked closely with Netscape and Macromedia durin development of new technologies for dynamic and streaming content delivery. Ruston is also a aficionado of high-end guitar and sound gear, which generally gets him VIP access to all entrances. H holds a Bachelor of Science degree from Texas Tech University.

Acknowledgments

The authors would like to thank:

Our patient wives, impatient kids, Moms, Dads, and families;

The visionary customers from whom John has learned so much, for your help documenting dozens great SV examples and case studies with Jason over the past six years, especially Sven Gerjets, Lau Miller, Russ Wheaton, Jamie Williams, and many others for allowing us to recount your storie herein;

All the genius, committed technologists from ITKO's early days who made this incredible ne technology happen, especially our cofounder Ruston Vickers—you are all the best in the business;

Ken Ahrens and Rajeev Gupta for helping us kill it on a couple of the tougher sections on SV proce and performance techniques;

Shridhar Mittal, Chris Kraus, Anuj Gulati, and Luther Birdzell for valuable input and help;

Robert Humphrey for championing the cause, and Scott King, Paul Neumann, Jim Dugger, Just Vaughan-Brown, and everyone else who stepped up while Jason was off noodling on this book;

And finally, all the technologists, executives, and partners who are acting on this unique opportuni to make a game-changing new approach real for business. We hope this book inspires you to take even further.

Virtually There at FedEx®

It's a Reality Most Businesses Would Want to Run Away From

Fifteen years ago, a team at FedEx had to make absolutely, positively certain that their deliveri would be supported by a software architecture of around 200 systems. Today, the number of movin parts and services they must fit together *easily exceeds several thousand unique IT services a systems.* And that's just one key group. Millions of end-customer and partner transactions hit FedE systems from around the globe every day.

Here's what Russ Wheaton—Director IT, FedEx—had to say about their journey:

> In keeping with the customer demand and expectation of the times, our company was testing a very specific stack of software 15–18 years ago, with some of our key systems originally having been built in the '80s. The earliest goal was to certify software functions that were considered "revenue-impacting" or "customer-facing" to our business. As time progressed and the system reality came closer to business expectation, the number, type and scale of the systems falling into this category grew quite a bit.
>
> We were facing a challenge: As we continued to roll out and connect more services to provide a higher degree of flexibility and service level to our customers, many more core systems were playing in the space of customer or revenue impacting. As the number of interconnected systems rose, the complexity of the "business transaction" increased significantly. This, combined with a fast-growing company strategy realizing ever-increasing shipping transactions, forced a big think on how we would address the end-to-end certification of the core flow, without regularly adding resources or budget to make it happen. We needed to change our strategy.
>
> Around the same time, a new movement was growing called SOA (Service-Oriented Architecture) which promised to simplify the problem via a set of principles and methodologies targeting the design of discrete but interoperable services. This was great in that it gave us more reuse and faster development of common enterprise services, allowing us to design, deploy, and decouple at a much more manageable level, eliminating many of the "big-system" dependencies and the fall-forward strategy.
>
> But there was a downside that SOA introduced for the large system certification processes. When you have a lot of teams or systems depending on one another being ready, that dependency has an impact on schedules. If the services needed at a specific time in the certification process weren't all sized appropriately, or were not ready to go, or were coming together for the first time in end-to-end testing, it just didn't work. It became an exercise in heroics to bring the pieces together while staying on schedule.
>
> About seven years ago or so, we introduced interface standardization as a core architecture principle that would sit across all our development silos. We decided to standardize interface technology on both the transport and encoding. Many good things resulted from

having well-defined (even in some cases self-defining) interfaces that helped significantly with software design and delivery in a complex heterogeneous environment. We had also hoped that we were making an investment in our future thinking that someday this would facilitate a more standard, repeatable certification process for our very complex applications. Ideally we could leverage a “simulation” technology (other industries had been leveraging simulators for decades), where we could stand up analogs of our well-defined interface and simulate them for functional or performance testing purposes in such a way that dependent development teams could work independently of one another from a development schedule perspective as long as the interface or “contract” between the two was well-defined and standardized.

While that was important for our schedule, we were also highly concerned with reliability. How could we certify each of these systems independently as a baseline and take a scientific approach, so that if one piece of code changed, we could ensure in an automated fashion that the results we expected were happening? In essence, could we leverage this technology to develop a new technique to push quality further up in the development lifecycle potentially as far as code and unit test?

For a company our size, the solutions we deploy to certify our revenue-impacting or customer-facing applications have to be technology-agnostic on the back end. We continue to work with the architect community to leverage standardized technologies like SOAP, REST, EJBs and integration buses, regardless of whether they talk to back-end mainframes, internal distributed services, clouds, or even external services. We know nirvana for us is to achieve consistent technology and encoding across all core enterprise interfaces.

Twenty years ago things were simpler. Business and IT were separated by a great divide. IT enabled things like accounting, but very little of the business productivity was driven by IT. But the Internet started bringing that gap together, and now business strategies are tied at the hip and very dependent upon the IT solutions and enabling strategies.

That puts pressure on our IT systems to look more like business solutions. We need IT to drive new capabilities, enable faster turnaround times for new services, and create greater agility for the business. We have to certify faster to get to market faster. And while IT evolves, customer expectations increase, and customer tolerance for system failure drops. Over time, in some cases, it has evolved from simply irritating the customer to impacting the customer’s very business model.

We need to keep raising the bar on ourselves. If our systems are not fast, secure, and accurate, customers will do business elsewhere.

When John Michelsen was in our office a couple years ago, that was the situation I laid out for him: Ever-increasing business complexity and demand for new feature development, on time, right the first time, every time—while nobody gets any extra time, money, or resources to make it happen. We wanted to change the game to get more productivity out of the hours and people we had to meet that demand, while maintaining a sensible and rewarding work–life balance for our professionals. It is forcing us to rethink our environment.

Today, everything has gone virtual, giving us a higher degree of repeatability and predictability amongst other things. We have virtualized servers; the industry nailed that one years ago. Via Service-Oriented Architecture (SOA), commoditized services and data are commonplace in today’s enterprise computing environment. With the introduction of

Service Virtualization technologies, something we internally call “Interface Simulation,” we’re now able to stand up hundreds of interfaces and virtual back ends without requiring the complex interdependencies of the peripheral systems (to the system under test). In one example from my team, we simulate 25 back-end services representing about 200 different servers in a space we traditionally struggled with. Interestingly enough, we didn’t even test those services. We just needed them to test the higher-end dependent service, yet they took the bulk of the time to set up and administer.

Taking away the need to work with real systems has greatly simplified our process. For adoption, we had to prove that virtual services worked better than the real thing to gain trust. The first time we were able to hand a performance manager an extra week of time in his cycle, it was like giving him a sack of gold.

But acceptance of any change in mindset across an organization can be hard, especially when you are confronted with “the way it’s always worked.” So my advice to anyone considering moving to virtualized services and interfaces would be this: ***Pick a spot where you have the toughest constraints, focus on it, make it excellent*—**and people will come out of the woodwork to support it.

CHAPTER

1

Introduction

Whether you realize it or not, you are likely already in the business of making software. Servi Virtualization is not just a topic for a select few IT professionals. If you are in a company that delive services to customers over the Internet or enables sales and service teams through software, this boo is for you.

Success is there for the faking. Service Virtualization offers a transformational new way to overcom the constraints that inhibit your teams from delivering software to production so you can get yo company's products and services to market faster, better, and cheaper than your competition.

Service Virtualization Defined

Service Virtualization (SV) is the practice of capturing and simulating the behavior, data, an performance characteristics of dependent systems and deploying a Virtual Service that represents th dependent system without any constraints, thus allowing software to be developed and delivere faster, with lower costs and higher reliability. This rather concise definition of SV will be elaborate upon and refined in Chapter 5.

Service Virtualization includes a new type of technology and an accompanying methodology f "virtualizing everything" in the environment around any software-enabled or Internet-based busine service or product you are developing. Since there are very few companies in business today that d not depend on software, the competitive and economic impact of Service Virtualization will b profound and far-reaching across many industries.

You Make the Transformation Happen

We will challenge you to understand your own role in advancing the transformational practice o Service Virtualization, whether you are an IT executive, delivery manager, architect, or in almost an way involved in software development. You will gain a basic understanding of how SV technolog works to alleviate software constraints. More importantly, you will learn why, when, where, and ho SV practices should be employed for maximum success and value to your business.

Practical applications for SV that are in use by enterprises today enable you to

- deliver faster
- reduce your infrastructure footprint
- transform your performance and scale
- manage your data scenarios

These are by no means the limit of what can be done with SV, but provide valuable approaches w have seen companies use to drastically reduce new feature delivery timelines, infrastructure, and lab costs, while eliminating unexpected risks from problematic software projects.

About This Book

This book assumes a basic understanding of how Internet-based business software projects a planned, budgeted, developed and delivered. The practice of Service Virtualization has profoun potential for improving the time to market and overall competitiveness of any business strategy with technology component. Organizational buy-in is key to success here, so while the core of Servi Virtualization practitioners are software development, testing, and IT environments teams, we als focus on business-level approaches and benefits.

Disclosure: *The writers of this book are a co-founder and early employee of the software firm ITK now a CA Technologies company. In 2007, ITKO invented, patented, and released the first Servic Virtualization software on the market within their CA LISA® product suite. Now Servi Virtualization is a growing category of software with established service provider offerings an related tools on the market from several leading vendors. This book's purpose is not to make claim about CA or LISA software products, but instead focuses on the best practices for enabling Servi Virtualization practices, no matter what combination of tools you select and use in your environment*

Signposts in the Book

Look for these icons as you read—they'll highlight some useful supplemental information.

Advice: These are tips that can help you smooth adoption of Service Virtualization practices for yo business.

Alert: These warnings can help you avoid common organizational pitfalls or implementation dange often found to inhibit success.

Geek Out: Engineers will find these details quite interesting, but if you are not reading for technic purposes, you may skip over these sections.

Remember: Don't forget these points when embarking on your own implementation

Definitions: Terms of art are highlighted at first occurrence in the text with bold italics and a asterisk (***term****) and are defined in the Glossary.

CHAPTER

2

The Business Imperatives: Innovate or Die

Almost all high-level enterprise executives when we first meet will say that software development not really their core business. They'll say: "We are first and foremost driven to be leaders in th [banking-insurance-telco-utilities-retail-travel-health care] industry. Our core business is helpin customers, not developing software." (Unless, of course, they are actually working for a softwa company.)

Well, we'd like to challenge that assumption right now. Any major company with an IT deliver component already has a huge software organization teeming under its surface, with thousands developers, performance and test engineers, and support and customer representatives all attemptin to drive technology to deliver on the expectations of the market. *I can't count how many CIOs go ou of their way to tell me they have more developers on their staff than a software company the size of C Technologies, or even in some cases Microsoft!*

Your enterprise software organization is likely spread out over multiple offices, organizational silo and partners, and paid for out of multiple budgets—but it has a shared motivation: Innovate or di Your competition has the same motivation too. This **innovation race** is driven by four very re business imperatives.

Consumers Have No Mercy

The ability to innovate and deliver new business capabilities for customers via IT is no longer just perk for most companies—it is the critical factor for success. Innovative software that performs an works flawlessly for customers wherever and however they choose to interact with your company ha now become a primary competitive differentiator.

Fail to deliver, and get left behind. Today's consumer has no patience for a poorly performing app an no tolerance for unexpected errors when doing business with your company.

Take my teenage daughter as an example of how much today's and tomorrow's customers expect fro IT. I keep her equipped with a debit card for day-to-day purchases (I am also a bit of a pushover fo my girl!). One day she happens to be in line at the food court in the mall and calls me to say she nee $10 in her account for lunch right now. I say, "OK, honey, let me see if the bank can transfer the fund that fast from my phone." She says, "Well if it can't, let's SWITCH BANKS."

Scary? Yes! Consumerization of IT means "no mercy" from today's customers. Had my bank no provided immediate access to those funds through my iPhone app, she would have been telling h 1,350 Facebook friends how awful my bank is and looking for a better one. They are accustomed transacting business with your company wherever they are, whether online or on the phone, an

getting almost instant gratification. If there's a problem with your application delivery, th competition is always just a click away.

Business Demands Agile Software Delivery

Let's take a look at one of the largest banks we have worked with. Rather than advertise their lon history, strong asset base, deep professional experience, and many physical locations, they ru television ads for things like:

- A new "quick pay" web service
- A new "scan checks from my phone" app
- A better security system for catching online identity theft

None of these above are traditional defining characteristics of a financial institution—these are a new software features that had better work as advertised when delivered in a web browser smartphone! It just goes to show that the rapid design and development of software applications now a primary way companies require us to go to market and differentiate in today's consumer-drive economy.

To top it off, over the last decade most major enterprises have put a huge emphasis on cutting IT cos as much as possible, and therefore the IT budgets of the old dot-com days aren't coming back. D more with less. This is the ***new normal**** state of IT economics that we must function in. The ver same bank mentioned above is expected to take hundreds of millions of dollars out of their IT spen over the next few years. That means your business must be ready to deliver new software functionali at breakneck speed in an increasingly difficult environment, without an increasing budget expenditu to match.

Increased Change and Complexity Are Inevitable

In an attempt to make IT more agile in delivering new software features at Internet speed, mo companies have moved toward composite application development approaches. These new servic oriented methodologies espoused the idea that new software could be produced more rapidly atop existing systems when broken up into smaller functional units or "services" that were more reusab and loosely coupled.

While this approach did accelerate development on a functional unit basis at first, over time it als created a "spaghetti mess" of services architecture with highly unpredictable results: man interdependent components, developed using heterogeneous technologies, managed and owned b distributed teams, with each version changing on its own release cycle (Figure 2-1). Yesteryear unitary "app" has evolved into a composite of several piece-parts of several other applications.

What happens in this type of highly volatile environment? We must account for this agile servic development by expecting to discover more and more software errors occurring owing to th unintended consequences of change. Some IT shops are doing more break-fix in production than eve which is not a sustainable model. Others throw more and more budget at each project, sometim increasing their lab infrastructure and QA budgets by 5 or 10 times in an attempt to ensure th software will function as expected once released.

One major insurance payer we know said they routinely "planned for their unpredictability" (oh, th irony) in delivering software by automatically adding ***30 percent more hours*** to the end of eve project plan!

Business Software Cannot Sustain without Simulation

If we built commercial airplanes the same way we build software today, we would never fly!

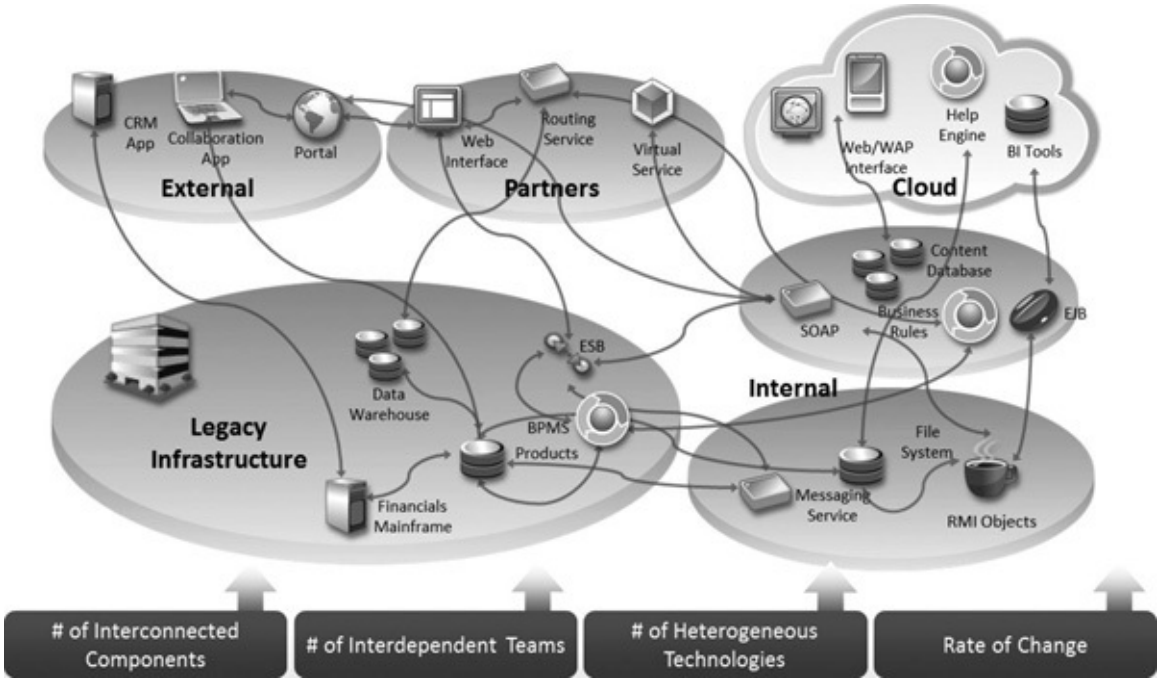


Figure 2-1. Composite application sprawl happens due to the decoupling of applications into servic based, distributed architectures, consisting of multiple interconnected heterogeneous technologies th are frequently changed and updated by multiple delivery teams.

Let's compare your business software development challenges to the manufacturing process f commercial airplanes. If we were the design team for a new wing and we performed that task as w build software today, we would demand hundreds of actual airplanes full of cargo and a pilot as ou "lab." We would then crash hundreds of planes in our horribly inefficient and desperate attempt to g our new wing design to fly.

But that is exactly what we commonly see companies do with the critical software that runs the operations today! In software, if some component fails in production, we just *send the release back development* and eat the cost of finding and fixing those errors, miss opportunities, and fail in front customers.

You can see that if airplane manufacturers took this "wait and see, then send back to dev" approac they'd go out of business long before we ever had viable commercial aircraft.

Today, aircraft design, manufacturing, and flight are the result of a process of constant simulatio testing, monitoring, and improvement. Each and every part of the plane's design is developed, the tested independently using modeling and simulation, and then tested again as part of an integrate system, while being continuously monitored in-flight for any performance issue. Real wing-desig teams know that they don't need planes to build a wing: they need modeling tools and a wind tunnel.

Simulation tools utilize feedback from this real production data to ensure correctly functionin components without waiting until the integration stage for verification. Pilots must even spend hou training in flight simulators that virtualize the airplane's behavior in real-world scenarios, without th risk.

The design of an aircraft presents engineers with an extremely complex architecture of thousands unique and highly interdependent components, systems, and sensors. It would be impossible account for all of the "what-if" scenarios of introducing a new element to a real aircraft witho

simulation.

Think about it: You can't expect to find every environmental condition that our wing will face durin a live test flight to see if it works—of course not! Instead, we must have complete control over th environment—a "wind tunnel" that simulates all the environmental conditions we need without th real plane. This allows us to fully verify the design much faster. *Simulation—it's just scienc Einstein!*

Today's software architectures now look more like a complicated aircraft design than the previou simplistic ***client/server**** or on-premise systems we once knew (Figure 2-2). It is surprising an business software runs at all given the woeful lack of robust simulation.

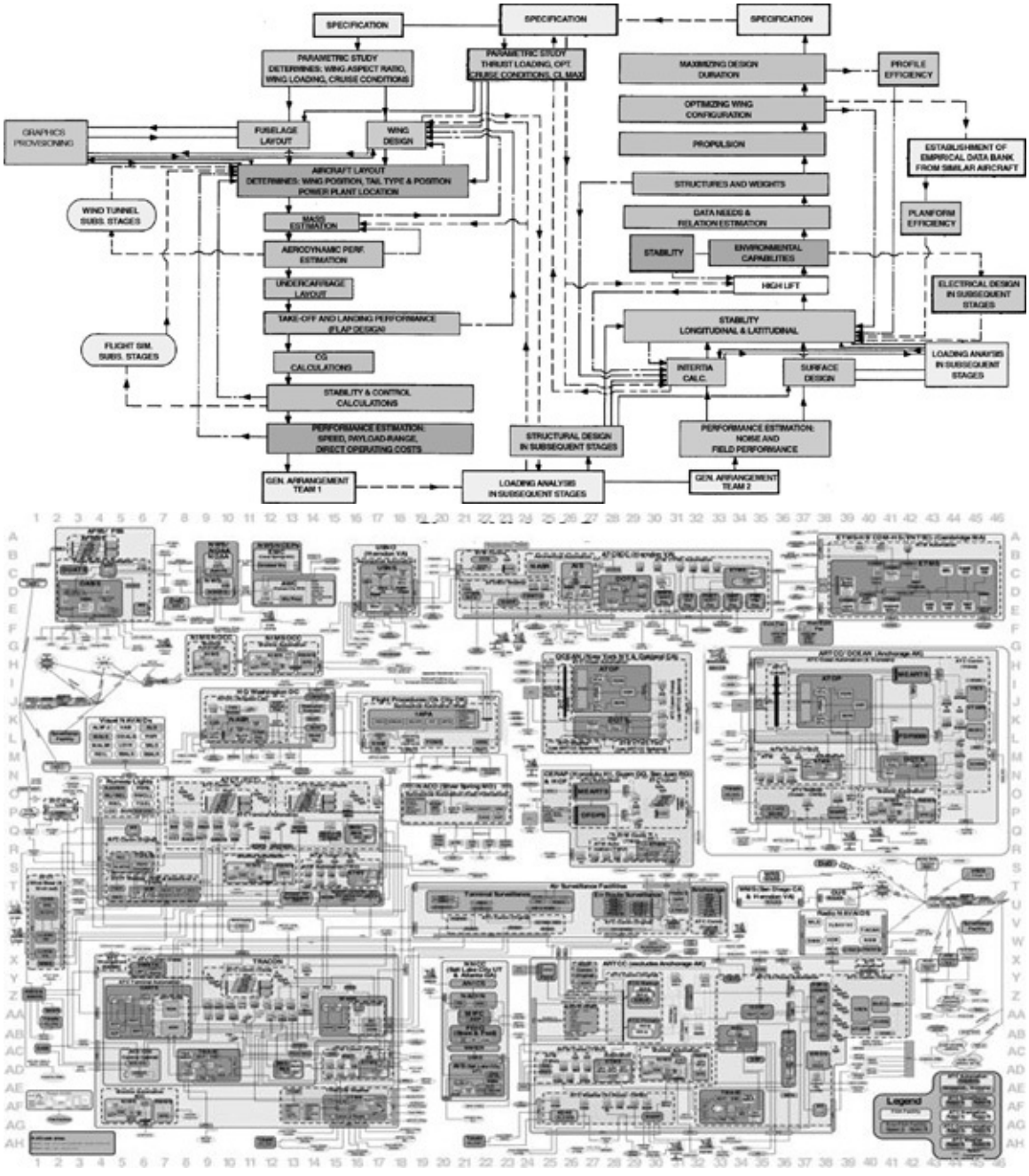


Figure 2-2. An aircraft design process (top) compared with a typical composite applicatio architecture view (bottom) (US FAA NAS public network architecture, 2008)

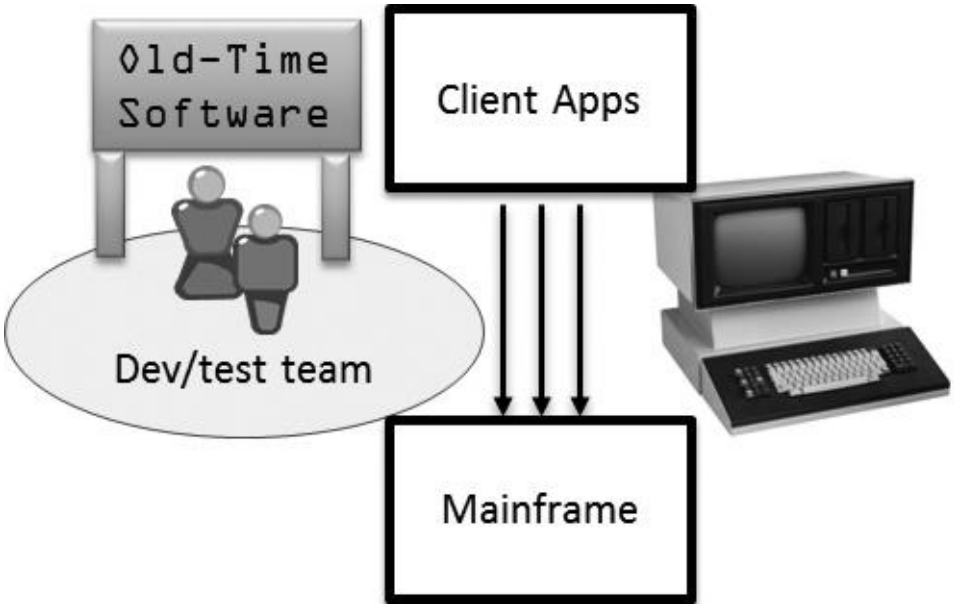
Other industries—from consumer electronics to automotive to pharmaceuticals—understand th
science of modeling and simulation for experimentation purposes in design and development. With s
much critical functionality on the line, it is now time that the business software industry sign up f
the same scientific discipline of proving out our hypotheses by simulating real-world environmen
ahead of delivery, for more predictable and safe results.

CHAPTER

3

How We Got Here

Let’s go back to the days of running a software shop twenty years ago—before many of today developers had even compiled their first line of code. Grizzled techies still talk about it like th Golden Age of software engineering, but it was such a simple world compared to today.



Software applications were essentially closed, monolithic systems. There was one client UI, th software ran on one target platform, and we usually had a year or more to get the project done. Bett yet, everyone on the project was likely co-located in the same building and could have lunch together

Yes, we had to homebrew many parts of the system to make it work. But in the early ’90s, we ha much lower expectations of interactivity or system intelligence. We had little or no interconnectivit or reliance on any external systems—everything was completely under our control. And we thought was hard back then…

Things have gotten incredibly difficult since then. We now have an enormous distributed staff an many partners, under incredible pressure to deliver new business functionality faster, with an eve increasing amount of complexity and change in IT environments.

From Monolithic to Composite Apps

It’s not like anyone wanted to make applications evolve this way. Customers demanded higher leve of service to meet their specific needs. Therefore, companies started keeping track of more custom

information, as well as offering more complex products and services that needed to be accounted f
by those core ***mainframes****.

Rather than replace these often irreplaceable systems, in the later ’80s the rise of the desktop P happened, and we naturally learned to “layer” on new technology and relational data to try ar abstract new software features in these clients, thereby working atop the slow-changing nature of co servers.

To survive, businesses needed to become more flexible and develop new software to meet the needs customers. So each evolution of our applications—from the still-often-critical mainframe client/server and n-Tier apps to today’s service-oriented composite apps (Figure 3-1)—was simply th next way to respond to ever-faster-changing customer and market demands with new software, whi carrying forward the core systems the business relied upon. We frequently describe our customer environments as “museums without plaques.”

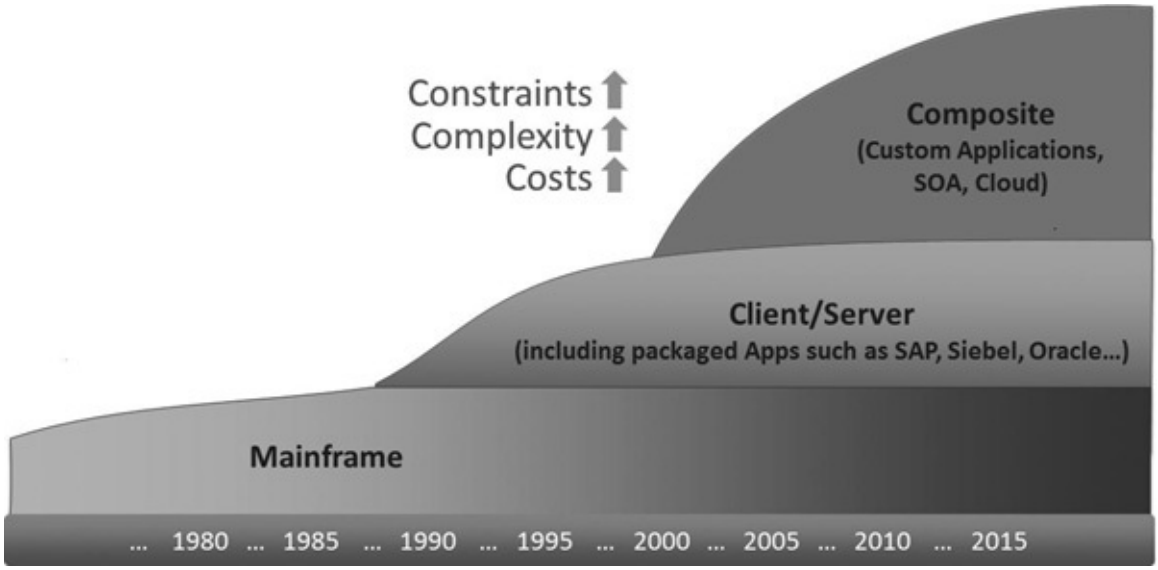


Figure 3-1. Evolution to composite apps from mainframe and client/server approaches. Note that th existing technology investments never go away.

Today’s Complex Service Environments

Let’s take a look at a simplified reference architecture for a modern composite application, which w will reference throughout this book (Figure 3-2). (If you are not a developer, don’t worry. You wi understand this!) A real enterprise software implementation will consist of many, many more boxe but the basic concept is a multi-tiered “layer cake” that looks like this:

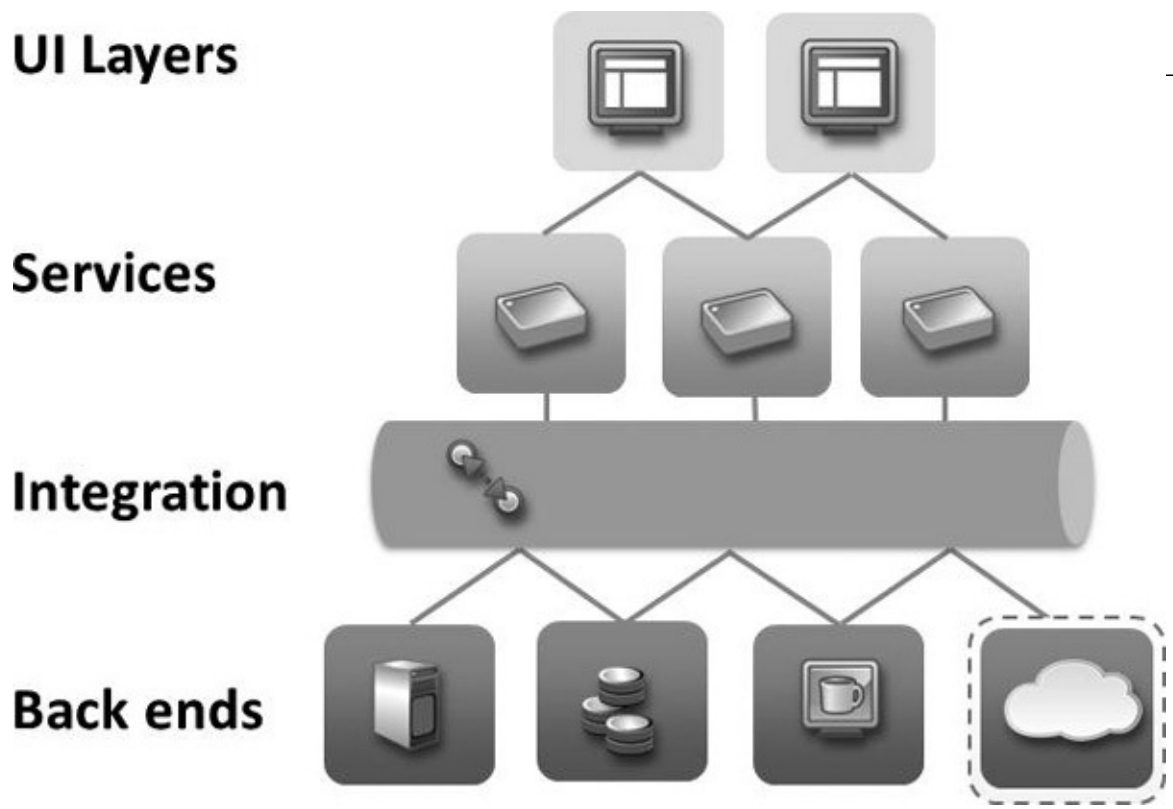


Figure 3-2. Simplified architecture diagram of a typical modern composite application with multip tiers of technology.

UI Layers: End users interface with apps through web or device UI layers that usually contain ve little business logic. These components are usually the most dynamic and variable aspects of a application, as any user experience may be customized and configured to accept a near-infini number of possible scenarios.

Services: Web and UIs running on app servers call on underlying services layers, which are modul components that contain much of the discrete business functionality that development teams a building. These services basically process requests when called and pass along data using a appropriate message protocol (SOAP, XML, etc.). While there are industry standards around thes protocols, companies inevitably have customized formats for some distributed communication.

Integration: As composite applications become more complex and new services and systems a constantly added, the orchestration of many moving parts must be addressed. To coordinate thes services to meet the needs of robust business processes, most enterprises adopted an ***Enterpri Application Integration (EAI)**** approach with an integration "backbone" or ***Enterprise Service Bu (ESB)**** system as a broker to route and queue up messages from the services layer and make calls the back-end mainframes and systems of record when needed.

Back Ends: Most requests of the top layer of the application will eventually delegate to these systen for execution. These are the core systems of record for a company such as SAP and Oracle Financial as well as legacy applications and third-party hosted applications and mainframes. Softwa development teams usually try to minimize changes to these highly utilized environments as the layers handle important live tasks and are difficult or costly to change or replicate.

As you can see, each business function supported by a given UI will have many downstream steps business logic, data, and dependencies that must respond in order to successfully execute a give business workflow.

Tech Note: While every composite app is unique, they usually share common design patterns. As yo move outward from the core systems toward the surface of an architecture, you will notice th underlying back-end systems and data sources are generally slow-changing (requiring an "act of God to make a major shift), middle-tiers are updated a little more frequently (new processes or integration added), while new features are implemented most frequently and cause the most changes at th services and application UI layers.

From Waterfall to Agile Development

Let's take a look at the evolution of software development now through a process lens. Developme teams are attempting to answer the need for speed as well—by moving away from the exhaustiv ***Waterfall**** development method of several sequential, gated steps that a team must finish and veri in order to complete a release (Figure 3-3).

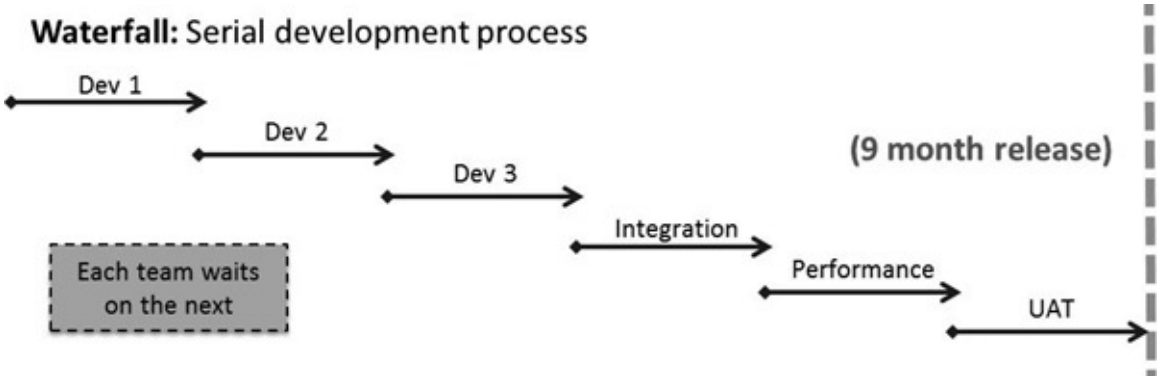


Figure 3-3. The Waterfall development process consists of several sequential development and te gates over time, building toward a long-term project release.

Instead of taking months or more to deliver a release using the Waterfall approach, in the last 15 yea we have seen a huge surge in popularity for ***Agile**** development methodologies. The Agile approac recommends smaller, independent teams to define, test, and develop small units of functionality i shorter cycles or ***scrums**** with short-term deliverables or "sprints" toward the end goal (Figure 3-4).

One cool aspect of Agile is that it promotes ***Test-Driven Development* (TDD)***—which mean developers "Test, then Code." First, a test is written. Then the developer starts coding. When th developed code passes the test, this is a proof point that the code works as intended. These unit tes are typically quite simple and test for the "happy path" of what the developer expects to deliver, b when developers unit test at a higher frequency, it increases the quality of each dev cycle. In th manner, incremental adjustments of software to meet a business requirement can be made over th course of several iterations, and the productivity (and engagement level) of development teams shou be increased.

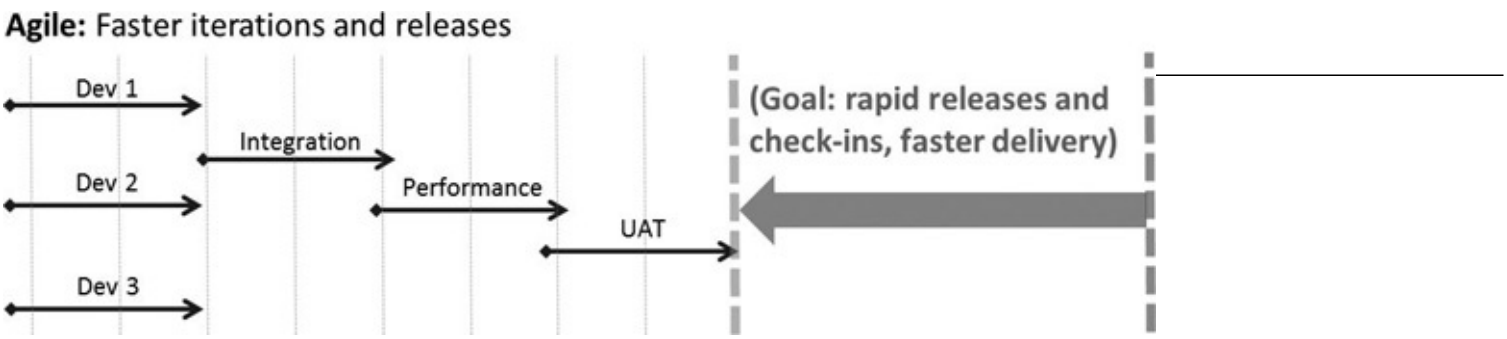


Figure 3-4. Agile development process breaks development into smaller, independent teams with th responsibility to develop smaller units of functionality in fast iterations or "scrums," with the goal faster alignment to the delivery requirement than Waterfall approaches.

Agile development also lends itself naturally to service-oriented technology approaches, as small units of functionality can be built and reused in the environment as modular, decoupled component Agile proved excellent for new, clean-slate development projects. For larger enterprises, however, often failed to deliver the expected boost in successful release speed. The combination of distribute Agile development with service-based applications, atop a raft of existing system and da dependencies, soon created a new set of challenges that caused project delays and failures, in the for of ***constraints***.*

Tip on Agile: Tons of great developer-level content has been published about Agile. We recommen starting with "The Agile Manifesto" site at http://agilemanifesto.org and following up with books b some of the leading authors in that space.

Caveat about Agile: Since most Agile authors largely focus on developer-level coding and testin activities, they tend to ignore the realities of interconnectedness and complexity inherent in enterpri IT. Therefore, at this time, Agile experts seldom acknowledge the external constraints and need fo simulation that we are talking about here.