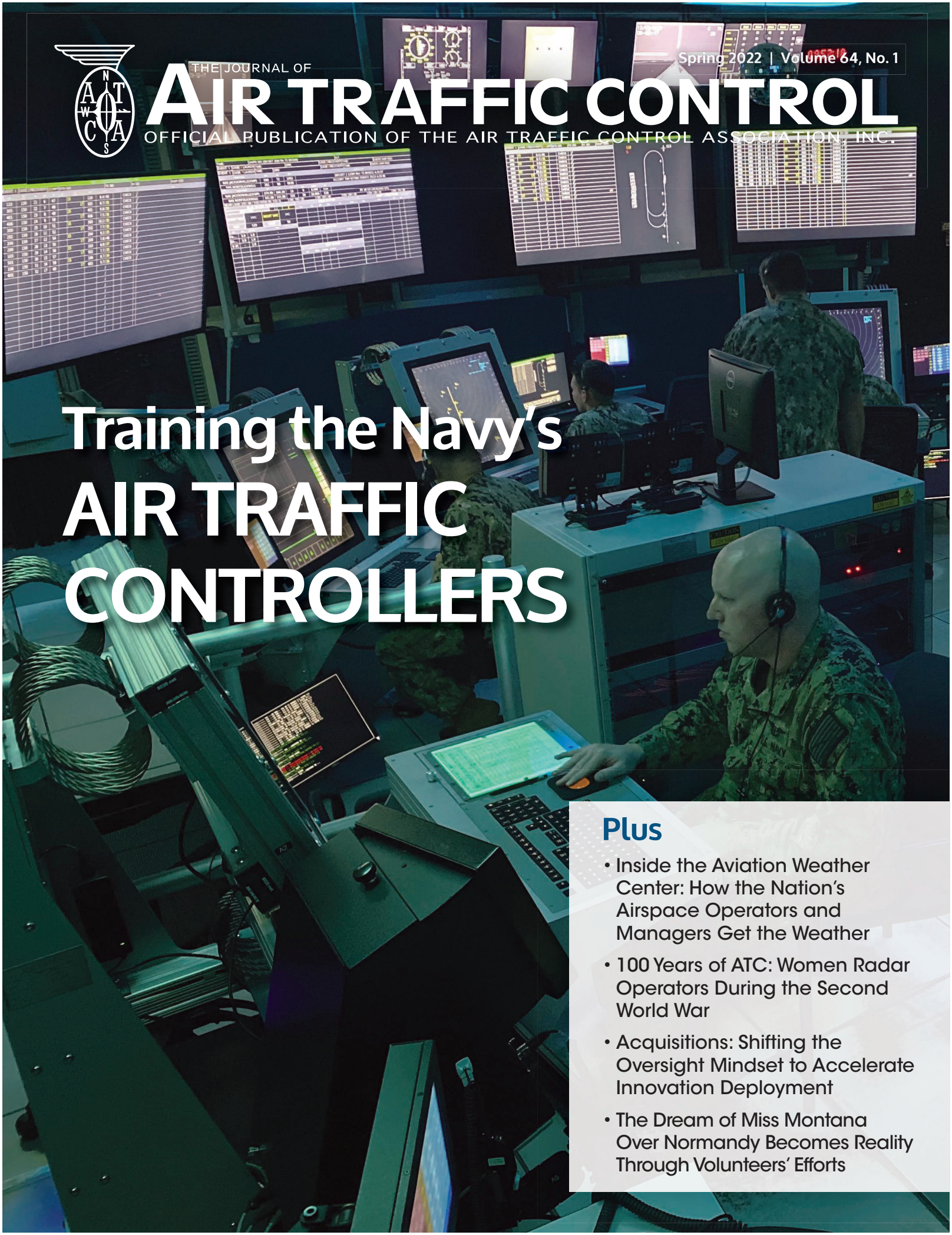
THE JOURNAL OF
AIR TRAFFIC CONTROL
OFFICIAL PUBLICATION OF THE AIR TRAFFIC CONTROL ASSOCIATION, INC.
Spring 2022 | Volume 64, No. 1



Training the Navy's AIR TRAFFIC CONTROLLERS

Plus

- Inside the Aviation Weather Center: How the Nation's Airspace Operators and Managers Get the Weather
- 100 Years of ATC: Women Radar Operators During the Second World War
- Acquisitions: Shifting the Oversight Mindset to Accelerate Innovation Deployment
- The Dream of Miss Montana Over Normandy Becomes Reality Through Volunteers' Efforts

CONTENTS

ARTICLES

DEPARTMENTS

THE JOURNAL OF AIR TRAFFIC CONTROL
OFFICIAL PUBLICATION OF THE AIR TRAFFIC CONTROL ASSOCIATION, INC.

Spring 2022 | Vol. 64, No. 1

Published for:

Air Traffic Control Association
225 Reinekers Lane, Ste. 400
Alexandria, VA 22314
Phone: 703-299-2430
info@atca.org
www.atca.org

Published by:

LESTER

140 Broadway, 46th Floor
New York, NY 10005
Toll-free phone: 866-953-2189
www.lesterpublications.com

President, Jeff Lester
Publisher, Jill Harris
Sales & Art Director, Myles O'Reilly

EDITORIAL

Managing Editor & Social Media Coordinator, Lindsay Risto

DESIGN & LAYOUT

Senior Graphic Designer, John Lyttle
Graphic Designer, Daniel Costa
Advertising Coordinator, Alina Cho
Online Media Designer, Mark Aquino

ADVERTISING

Sr. Account Executive, Quinn Bogusky
Sr. Account Executive, Louise Peterson
Account Executive, Scott Reid
Account Executive, Charmage de Veer

DISTRIBUTION

Administrative Assistant, Maryanne Li

Cover photo courtesy of the US Navy

ATCA members and subscribers have access to the online edition of *The Journal of Air Traffic Control*. Visit **www.lesterfiles.com/pubs/ATCA** User name and password: **ATCAPubs** (case sensitive).

Lean Acquisitions

Shifting the Oversight Mindset to Accelerate Innovation Deployment

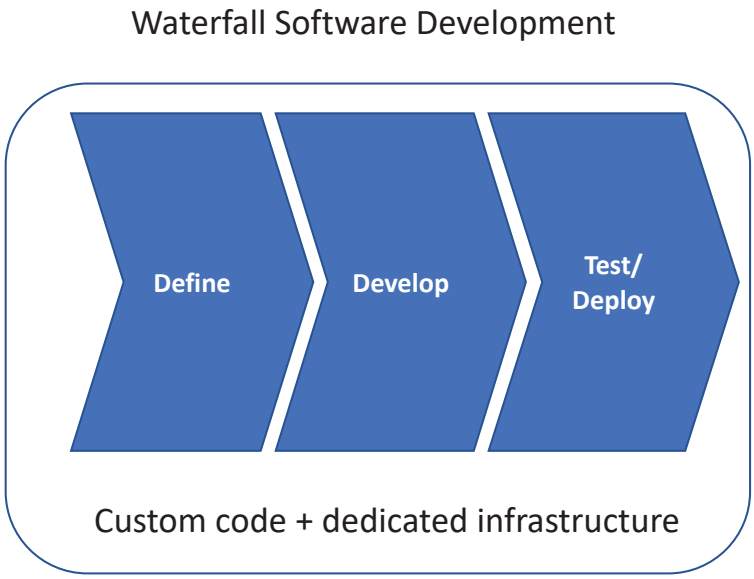
By Liviu Nedelescu, CEO, Avansys Solutions

From Free Flight in the 1970s to the National Airspace Evaluator of the early 2000's Joint Planning and Development Office days, the aviation community has always had an eye on the future. Implementing bold concepts has historically relied on large scale development, which in turn brought with it the tangible risk of big failures. The community has consequently been reluctant to undertake paradigm shift innovation. The safety-critical nature of the business further cemented the strong preference for incremental transformation.

The shift in how technology is delivered is poised to disrupt the industry's premises for incremental transformation fundamentally. The "as-a-service" model instantiated by cloud solutions and agile practices is growing to encompass more and more of what used to be custom development. As important, risk thinking is shifting from avoidance at all costs to factoring small failures as a normal component of effective outcomes. All these trends are converging to drive new opportunities for implementing bold innovation faster. To leverage them fully, the very core of government, its governance systems, need to evolve.

This article discusses the growing disconnect between the latest commercial development practices and a governance model over reliant on planning. A primer on the de facto government governance model

kamawatth / 123rf



is overviewed, followed by a discussion of the evolution of commercial development practices. The implications for governance systems are subsequently considered.

The de Facto Government Governance Model – a Brief History

The Second World War brought large-scale military development to the forefront. The system development lifecycle was established at that time, giving rise to systems engineering. The engineering mindset demanded that development of a system could only proceed after careful up-front analysis. The result was the institutionalizing of a planning mindset which has been a key tenant of government ever since.

Acquisition management was aligned to the engineering model. Cost and scope analysis were added to the engineering analysis phase, allowing for the planning of budgets. Compliance tracking against baselined cost and scope soon followed, ensuring full lifecycle accountability to the US taxpayer.

This mindset and supporting processes have permeated the entire US government, becoming the de facto way it has done business for the past 70 years. With the aid of this model, the aviation community has been a direct contributor to globalization by delivering affordable, safe commercial flight. The successful implementation of the NAS can be directly traced to the systems engineering model and the accompanying governance practices.

However successful, this paradigm has capped transformation to an incremental pace. Even as NAS automation is periodically refreshed and updated, the underlying operation has changed little over time. Incremental improvement of existing operations was reliably favored over big ideas such as Free Flight.

Challenges to the Engineering World View

Challenges to the classic engineering model's heavy focus on up-front planning were apparent soon after the Second World War. By the late 1960s, the cybernetics school of thought challenged the planning paradigm by calling out the impossibility to formulate a set of rules objectively by which complex systems function. It was proposed that the very act of observing such a system changes its behavior. The stock market was one such example – to understand it, one must participate, which in turn changes the initial investment premise.

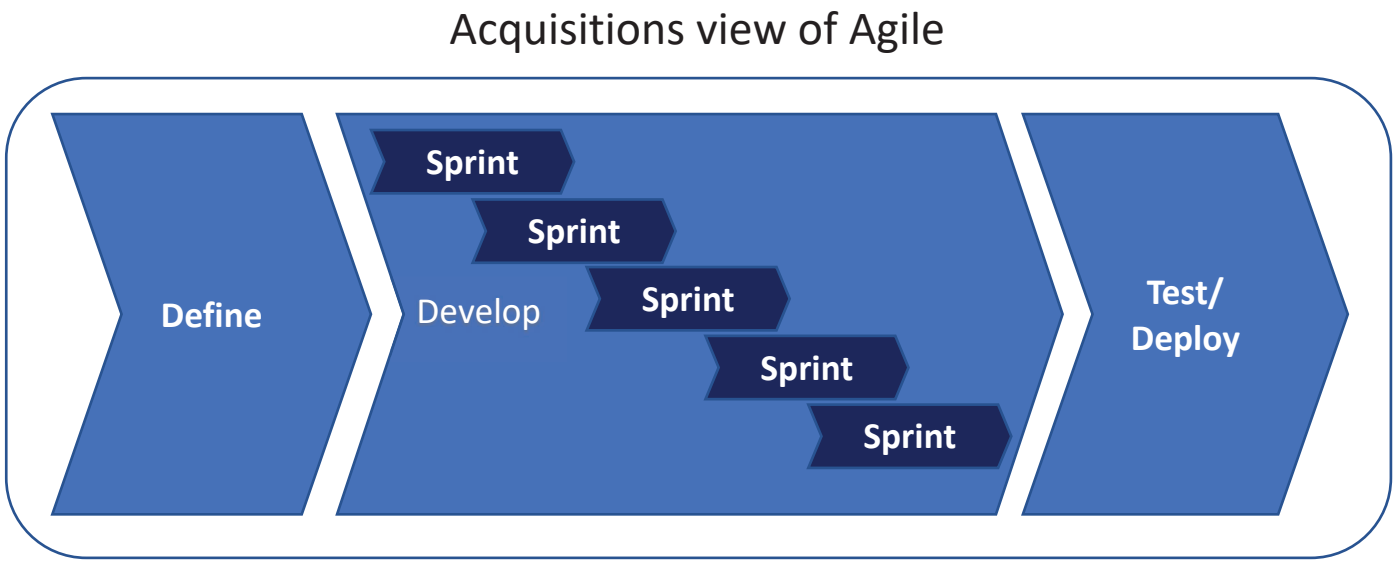
Herbert Simon, the 1978 Nobel Prize Laureate in Economics, further cemented the limits of decision-making based on objective plans with his theory of bounded rationality. He then established the school of design thinking, which led to the creation of the user-centered design practice. Simon and visionaries like him posited long before the advent of agile that understanding user needs is a messy and subjective trial-and-error process which is best undertaken iteratively.

The Birth of Waterfall Development

While intriguing, these rather abstract notions didn't have a real impact on the world of government contracting. If anything, "trial-and-error" and iterative exploration would have sounded more like a recipe for disaster in the engineering-dominated post-Second World War mindset.

However, by the 1980s software was fast overtaking hardware development. Software development was initially modeled to conform to the systems engineering lifecycle (i.e., define, develop, test/deploy) giving birth to what is now known as the "the waterfall method." Dedicated supporting infrastructure would have to be purchased for each new undertaking, and all resulting code was custom. Each system ended up being unique and there was little opportunity for reuse. Developing systems this way was certainly an expensive enterprise and costs proliferated through the maintenance phases.

Waterfall development did conform very well to a governance mindset heavily reliant on up-front planning. Investment analysis could be performed against a pre-defined



scope, which was all but guaranteed not to incur further changes during development. Contract performance and program control metrics could be defined ahead of time and program performance tracked against baselined cost and scope parameters. This was both straightforward and convenient.

Enter Agile

Waterfall development would have remained a best practice today had it not been for one nagging problem: no matter the careful up-front engineering analysis, projects still failed. The industry learned through massive cost overruns and scope creep that software is very different than hardware. Abstraction introduced subjectivity to the process of eliciting requirements. Long, protracted analysis before development was undertaken simply resulted in stale requirements. Lengthy analysis and planning cycles couldn't keep up with the pace of commercial innovation, resulting in designs predicated on obsolete technology.

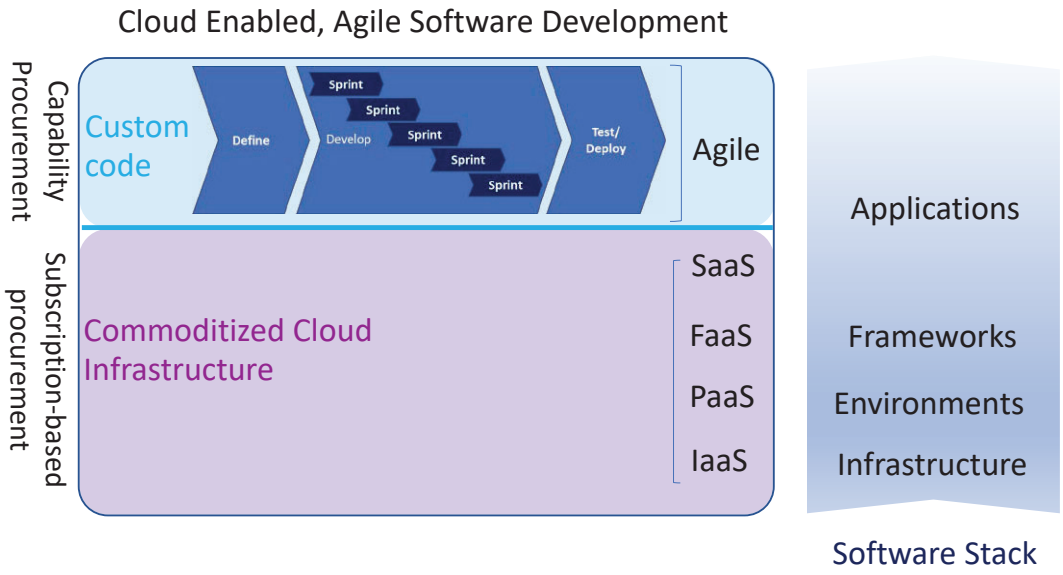
At over $2.6 billion, the FAA's Advanced Automation System (AAS) provided one of the last and most visible failures associated with waterfall development. With a $1.6 billion write-off, it was labeled by Robert N. Britcher, author of the *The Limits of Software: People, Projects, and Perspective*[1] as possibly the "the greatest failure in the history of organized work."

Only a few years after the AAS was shut down, Steve Zaidman, associate administrator for acquisitions for the FAA stated, "The benefits of AAS came at the very end, and since the project never reached the intended finish line, most of them never materialized. In contrast, today the FAA is approaching the problem of systems development with a series of smaller projects that are supposed to deliver continual, incremental benefits."[2] By the late 1990s waterfall was all but dead. Agile was here to stay and the aviation community clearly understood its use.

While agile development became the norm for developing software, the governance model remained largely unchanged.

The engineering mindset demanded that development of a system could only proceed after careful up-front analysis.

[1] Britcher, R., *The Limits of Software: People, Projects, and Perspective.*
[2] Edward Cone, The Ugly History of Tool Development at the FAA, https://www.baselinemag.com/c/a/Projects-Processes/The-Ugly-History-of-Tool-Development-at-the-FAA.



Those in charge of governance regarded agile as a coding methodology rather than a trial-and-error mindset. From a governance standpoint, agile simply meant that planning would be done based on sprints. The up-front definition efforts and back-end testing and deployment remained in place and largely unaffected. Furthermore, all the governance methods associated with waterfall still applied. Investment analysis could be performed against a fixed scope that would be delivered in smaller sequential releases to conform to agile. Progress could continue to be tracked against baselined scope, cost, and schedule values using simple interpolation techniques reminiscent of earned value management.

The Early Days of the Cloud

The fact that a large part of what previously was custom development and dedicated information technology infrastructure was being replaced by virtualized cloud services and commoditized applications were at first conveniently ignored for governance purposes. Even though more of the information technology stack could be procured as subscriptions to virtualized services (i.e., Infrastructure-as-a-Service [IaaS], Platforms-as-a-Service [PaaS], Software-as-a-Service [SaaS], Framework-as-a-Service [FaaS]), governance processes continued to focus on determining the up-front scope and cost of an envisioned capability as if it were fully custom developed. The level of effort that went into front-end governance pursuits never diminished even as the need for planning accuracy was effectively offset by the ability to make subsequent adjustments as needed, afforded by the on-demand business model.

Leading Edge Commercial Trends

Steve Jobs introduced the single technology that fundamentally transformed the world: the smart phone. While this innovation might seem radical to us today, it was brought about by the integration of existing technologies such as iPod interfaces and touchscreens. This convergence turned out to be the paradigm shift innovation. Similarly, agile and cloud technologies are coming together to drive something that is larger than both: subsuming custom software development as part of the "as-a-service" model.

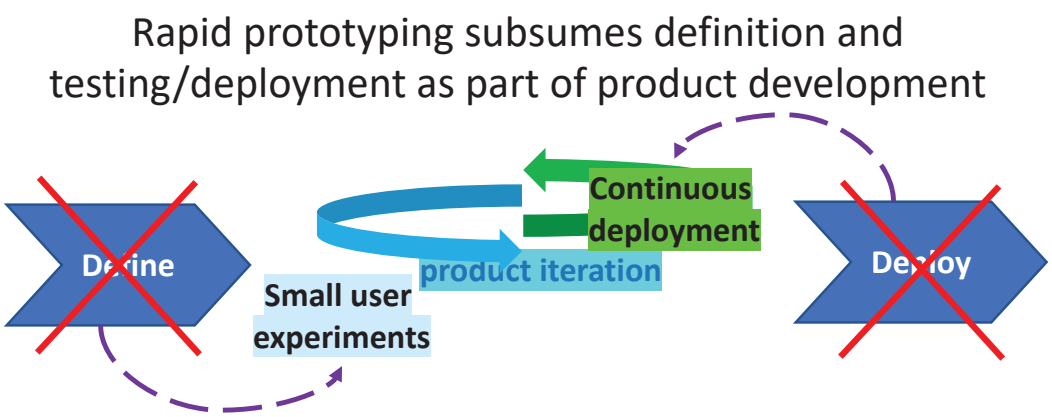
A key innovation that capitalized on all the advancements of cloud and agile is rapid prototyping, and it points to the eventual addition of custom development to the "as-a-service" stack. The implications for planning-based governance systems are profound. Before discussing them, it is important to highlight the uniqueness of this methodology, itself part of a larger family of "adaptive development approaches."

Rapid Prototyping – a Self-Contained, Self-Correcting Methodology

Rapid prototyping is to cloud and agile what the iPhone was to the iPod interface, touchscreens, and the Global Positioning System (GPS). It converges cloud technologies and agile methodologies to transcend both. The net effect is that pre-development definition and post-development test and deployment phases of a traditional product development lifecycle are subsumed in development itself.

As compared to stand-alone agile, rapid prototyping is no longer limited to software coding. It is rather a full product development method. This means that rapid prototyping seamlessly embeds governance elements for defining what is to be built and validating that the final product meets its intended purpose.

User needs definition, testing, security accreditation, and deployment are all performed continuously "inside" the small product increments that are delivered iteratively directly to a production environment. Ongoing user interviews drive product experiments whose validity are subsequently verified with the users. The short timespan between the definition of experiments and their instantiation in product functionality leave little opportunity for misunderstandings. Efficiencies are further achieved by cloud automation platforms that fully integrate what used to be stand-alone layers of the software stack (IaaS, PaaS, FaaS, SaaS)



with DevSecOps and advanced architecture models such as Kubernetes. All this enables highly automated pipelines for delivery of production-ready software.

From a technology perspective, rapid prototyping and supporting cloud technologies can be said to be "self-contained;" that is, they contain all the necessary tools for the shortest path from code to working product. No stand-alone testing or deployment activities need to be performed outside of or alongside development. Testing, deployment of code to production environments, and security accreditation are all handled as part of the development process itself. Furthermore, all these are evolved incrementally to match the iterative nature of development, rendering related up-front planning efforts less meaningful.

From a governance perspective, rapid prototyping can be regarded as "self-correcting." As the user feedback is considered continuously throughout the development effort, the risk for arriving at a product that doesn't meet user needs is minimized. Scope creep on the other hand is capped to the size of the small increments by which the product is iteratively delivered. Timespans for closing the loops are extremely small, rendering the possibility of going on tangents correspondingly small. All this hints at a significant reduction of the need for planning-based governance systems, as well as stand-alone oversight as the product is being developed.

Governance Implications – Doing More with Less

Stepping back to the big picture, rapid prototyping adds the procurement of custom code to the "as-a-service" stack. With the addition of Development-as-a-Service (DaaS), the full stack can be procured as a service. More importantly, the fully integrated service stack is self-contained technologically, while the development methodology is self-correcting. This reduces both the up-front planning typically required for governance and the ongoing need for oversight.

From an investment perspective, the cost of a rapid prototyping project is simply the sum of subscriptions to the various layers of the stack that may be needed (DaaS, IaaS, FaaS, SaaS, etc.) In this scenario, investment analysis becomes a very simple and transparent activity. The dependencies on planning-based governance as applied to the definition of scope (i.e., requirements), test planning, program controls measured against baselined scope, and cost are also significantly diminished.

In short, cloud and rapid prototyping reduce the dependency on analysis and planning, which represent a significant overhead to development. Through the application of a self-correcting methodology, such as rapid prototyping in tandem with a self-contained technology stack, one can produce more with less overhead. To achieve this outcome, a necessary reduction in planning-based governance is required, along with a change in the mindset.

Evolving the Existing Governance Mindset and Practices

Planning overhead represents a significant chunk of the overall effort to deploy new capabilities. The commoditization of custom software development and the virtualization of supporting infrastructure dramatically reduces the need for such overhead, providing a huge opportunity to deliver more capability faster, and with leaner investment. The shortening of time between concept and implementation can also significantly reduce the risk of engaging in bolder innovation initiatives.

An outstanding issue is whether effective oversight can be maintained with leaner governance practices. To address this legitimate concern, it is important to highlight a few recurring governance questions that underlie the planning mindset:

- Scope governance: What will be built?
- Investment governance: What should the budget be for a certain development?
- Affordability governance: Can the user self-contain their needs?
- Testing governance: Has the correct product been built?
- Program governance: How is progress measured?
- Risk governance: How can risk be capped to an acceptable level?

It is important to understand that most of these questions largely stem from a waterfall mindset. Accurate up-front planning is critical in such an environment since waterfall affords little change opportunity once a plan is in place. Consequently, all possible care must be invested into predicting as accurately as possible, ahead of time as much as possible, about what is to be developed. By contrast, not only do modern commercial practices allow for ongoing adjustments, they encourage it.

The paradigm shift contained in methodologies of the adaptive development family centers on the mechanisms for self-containment and self-correction introduced above. As highlighted, these are emergent properties of the convergence of a multitude of technologies and principles, traceable to even earlier than Simon's 1970s ideas. These mechanisms are key to addressing the otherwise reasonable questions about the inherent risk for leaning out legacy governance practices. Each such question is discussed from the perspective of modern development practices below.

Scope Governance

In a rapid prototyping environment, self-correction is achieved by keeping close to the users throughout the lifecycle. In development, rapid prototyping defines small chunks of scope iteratively, and validates that the correct outcomes are achieved with every small functional increment that is delivered. Therefore, the need to define the scope up-front is minimized. All that is needed to kick-off this self-correcting mechanism is a general framing of the problem space which provides the general boundaries of the future product. Once development gets underway, users and developers incrementally discover the functionality that makes up the problem space. Scope emerges as development proceeds.

Investment Governance

With the addition of DaaS to the development stack, investment governance can accelerate what is already a pivot towards the procurement of services (rather than capabilities). Just as the cloud offers scalable on-demand computing resources, DaaS provides on-demand custom development capacity. Smaller or larger development capacity can be procured by adjusting the size of the self-contained development team that comes with all the needed "as-a-service" technology tools as a package. Given any starting point, a DaaS team using rapid prototyping techniques can quickly iterate with the user in the loop to frame a particular problem space. Scope then emerges as the problem space is incrementally explored and defined based on user discovery cycles. As explained, the definition of scope is contained in the development process itself, rendering the up-front costing of an envisioned capability less meaningful.

Affordability Governance

In lieu of an up-front definition of scope against which a cost can be estimated, classic affordability notions need to be refactored. Assuming a significant pivot to procurement of on-demand services to include development capacity, the affordability question is more meaningful if regarded as a subscription cash-flow problem. A subscription cash flow can be determined for the initial phases of a rapid prototyping project. Subsequently, when enough is known about the impending scope, a more traditional affordability analysis centered on the cost of a capability can be performed. At the very least, in-depth affordability analysis should be delayed until enough is known about an envisioned product. Product development start doesn't need to wait for all this analysis up-front, making a faster path to product outcomes possible. Financial oversight is also implicit in the

gajus / 123rf

Waterfall development did conform very well to a governance mindset heavily reliant on up-front planning.

continuous interaction with the end users. The sophisticated user interview process that includes "user experiments" with minimally viable product principles from agile ensure that what is built is viable and useful. Furthermore, all coded functionality is production ready at any point in time. This means that even in the event of an unexpected stop work order, the product will be in a working state, albeit partially covering the full problem space.

Test Governance

Techniques such as test-driven-development (TDD) provide the means to test functionality continually as it is incrementally built. User stories embed test cases before they get coded. These describe expected outcomes for what a user should experience based on the behavior of the compiled code. The code is not pushed to deployment unless such outcomes are achieved. From this perspective, deployed code has already been tested by the time a user interacts with the product. There are other similar techniques that enable the testing paradigm to match the iterative nature of development and keep pace with continuous deployment. Other top-down testing techniques complement TDD by focusing on closing the loop on whether larger chunks of functionality operationally conform to the expectations set in prior user experiments. Once again, the dependency on heavy up-front planning of detailed test procedures is diminished.

Program Governance

Peter Drucker's astute distinction between doing things right and doing the right things has had implications far outside the world of management science and leadership. It has helped highlight the fact that product effectiveness is far more difficult to validate than process efficiency, while also being more valuable.

Russell Ackoff acknowledged this difficulty with his typical dose of sarcasm, "Managers who don't know how to measure what they want, settle for wanting what they can measure." It is relatively straightforward to claim accountability by checking a series of pre-determined boxes to verify compliance against a baselined plan (i.e., doing things right). It is far more difficult to

decide whether an undertaking is achieving its intended outcome (i.e., doing the right things). Modern commercial practices have pivoted the focus on measuring user satisfaction with a product.

Program governance has traditionally been ensured through metrics and measurements tracking to baselined plan parameters such as scope, schedule, and cost. Entire practices and professionals have been stood up to cater to things being done right as the basis for the very human need to remain in control. Doing the right things, on the other hand, is assessed by outcomes, and outcomes need to happen so they can be experienced. However, there is a leap of faith period while an outcome is pursued that no leader wants simply to assume. To mitigate this problem, leading indicators of outcome success can be defined. Methodologies such as rapid prototyping work with user experiments that embed falsifiable hypotheses and product assumptions that might be invalidated. As these are confirmed, the path to a certain outcome can be reinforced along the way, increasing confidence in its achievement.

Risk Governance

Since no plan survives reality, course corrections are a necessary part of any undertaking. Traditional risk management aims to keep course corrections within a manageable size by predicting needed adjustments with enough lead time, so their implementation is not disruptive to the plan. The unintended byproduct of a planning-based risk paradigm is long decisional timespans, which in turn allow risks to grow correspondingly large. Risk magnitude becomes directly correlated to the level of effort involved in making course corrections. The possibility of systemic risks is itself the biggest risk concealed in the mechanics of the method itself under the false sense of certitude conveyed by sound planning.

By contrast, the mechanism of fast iteration that anchors the family of agile development practices effectively caps risk magnitude through much shorter and more frequent course corrections. The risk mindset shifts from preventing large failures at all costs through planning, to factoring small failures as a normal component of rapid "trial-and-error" learning cycles. Allowing for the possibility of tiny failures from which one recovers quickly provides for far more effective resiliency against unforeseen disruptors. Methodologies such as rapid prototyping can be said to embed much more adaptive capacity, which is the best overall risk mitigator. Risk planning can and should still be performed. However, it should be done with less focus on getting all the underlying parameters correct. When sufficient adaptive capacity is built into the development method, the degree and granularity of risk planning can itself be relaxed.

Agile practices ushered in incremental development as a risk mitigator for the failure of large-scale projects. Next generation practices no longer limit this mindset to development itself. By adding self-contained and self-correcting mechanisms as part of software development, the need for standalone oversight is minimized.

The on-demand paradigm delivered by the cloud has already streamlined the acquisition of information technology infrastructure. As-a-service procurement is now expanding to include more of the software stack, notably custom applications. Up-front planning for capability development can accordingly be relaxed.

These and other meta-trends are converging to enable faster delivery of transformative innovation. However, industry is fast approaching an inflection point for force fitting commercial innovation practices to legacy governance systems. Either the governance mindset evolves, or the benefits of leading-edge commercial innovation will be hard to spot. As an analogy, using a race car for towing is, of course, possible, but it defeats its purpose.

Charts courtesy of Liviu Nedelescu

Liviu Nedelescu is the CEO of Avansys Solutions, LLC, where he consults government customers on reframing governance practices for next generation technology delivery models. Nedelescu's 20-year aviation track record straddles solution implementation and thought leadership. Innovation highlights include patented air traffic simulators, research roadmaps for autonomous flight, and framing the long-term evolution of the NAS. Nedelescu also advises on product development for commercial clients.