The ITEA Journal of Test and Evaluation 2019; 40: 95-105

Test in the Age of Agile:
Rising to the Challenge of Agile Software Development

Col Douglas "Beaker" Wickert, Ph.D.
Air Force Test and Evaluation, Headquarters US Air Force

"The best way to get a project done faster is to start sooner."
– Jim Highsmith (Agile Manifesto signatory)

Agile software development is an iterative, incremental, and evolutionary approach to managing software development that is relatively common in the commercial sector. With the objective of reducing development time and responding to changing requirements on faster cycles, military programs are increasingly adopting Agile as part of their acquisition strategy. Adapting Agile methods for military systems, which often have unique requirements, requires careful consideration and a shift in traditional test and evaluation methods. This paper describes the challenges of test and evaluation of Agile software and suggests several ways of addressing those challenges, including embedding testers with development teams. A cautionary note regarding the criticality and complexity of military systems is offered along with the importance of safe, effective, and efficient tools for risk management. We include examples and lessons from the test and evaluation of several Air Force Agile software projects.

Introduction

The single greatest loss of life for coalition forces in Desert Storm was due to a software flaw that was insufficiently tested. On February 25, 1991, an Iraqi SCUD missile hit the 475th Quartermaster barracks at Dhahran, Saudi Arabia, killing 28 and wounding another 100 soldiers.[1] Patriot missiles launched to intercept the inbound SCUD missed the intercept due to a range gate error caused by the accuracy limits of the internal clock. The "clock," an incremental counter that counted every tenth of a second since the system was turned on, only used a 24-bit register for timing. At the time of the attack, the Patriot battery had been operating continuously for more than 100 hours, saturating the 24-bit register counter. Prolonged operations was an endurance condition that had never been tested since the Patriot was designed as a mobile system and was expected to restart every couple of hours. Twenty-five years later, the risk of insufficient software testing is a challenge that the Test and Evaluation (T&E) community must meet as the Department of Defense (DoD) moves increasingly to Agile Software Development (ASD) practices.


Col Douglas P. Wickert, Ph.D.

Agile Software Development, commonly contracted to 'Agile' (and usually capitalized[2]), is an iterative and incremental approach to software development. It is typically implemented through small, collaborative, self-organizing, cross-functional teams. Modern ASD emerged out of several different product management theories in the late 1990s including Scrum, Lean, Kanban, Rapid Application Development, and Extreme Programming.[3] Agile was encapsulated in the Agile Manifesto, a set of four values and twelve principles signed by seventeen software management theorists at a retreat in Snowbird, UT, in 2001 (*Table 1*).[4] Since then, Agile as a management structure has further evolved into a wide variety of methods and practices self-described as Agile including Scrum, ScrumBan, Scrum/XP, Kanban, Scaled Agile Framework (SAFe), and Extreme Programming (XP) to name a few of the more common categories. DevOps (and DevSecOps) are more recent manifestations of Agile in which development and operations (or development, security monitoring, and operations) are combined to run concurrently with close integration and collaboration between developers and users. The values

Table 1: Agile Manifesto Values and Principles

Values of the Agile Manifesto				
value more	Individuals and interactions…	over	…processes and tools	value less
	Working software…	over	…comprehensive documentation	
	Customer collaboration…	over	…contract negotiation	
	Responding to change…	over	…following a plan	

Twelve Principles of Agile Software	
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	7. Working software is the primary measure of progress.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	9. Continuous attention to technical excellence and good design enhances agility.
4. Business people and developers must work together daily throughout the project.	10. Simplicity – the art of maximizing the amount of work not done – is essential.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.	11. The best architectures, requirements, and designs emerge from self-organizing teams.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

and principles of the Agile Manifesto are now also being widely adapted for management of non-software projects. A 2016 *Harvard Business Review* article by one of the original seventeen manifesto signatories highlights that agile management methods have spread from software development to a broad range of industries, functions, and "even into the C-suite."[5]

Since 2001, Agile has become a dominant approach to software delivery, at least in the commercial sector. With a few exceptions, the defense industry has continued to rely on a traditional, top-down "waterfall" approach—first codified by DOD-STD-2167 in 1985 but with earlier origins in the 1970s—in which requirements are cascaded down to define architecture, which is then coded, verified through testing, and validated in operational test (*Figure 1*).[6] Though now frequently maligned and much disparaged as creaking and archaic, the waterfall model is essentially system engineering applied to software. When closely examined, the real complaint against waterfall has little to do with waterfall itself and more to do with the nature of acquisition and the challenge of writing requirements in general. Agile excels in domains in which requirements are uncertain (Agile proponents claim that the requirements "emerge" through development) because flexibility and iteration permit the gradual development and definition of detailed requirements. In reality, the distinction between waterfall and ASD is not an absolute choice between mutually exclusive practices. Successful Agile projects depend on a stable development environment or functioning baseline before beginning the cycle of sprints that characterize ASD. For example, the Defense Innovation Board (DIB)[7] recommends approximately six months of development before initial fielding and the start of the "continuous delivery" phase which then runs on a recommended three-month release cycle.[8] Stable functioning baselines necessarily need some level of architecture design and systems engineering, i.e., a waterfall.[9] On the other hand, nothing in waterfall precludes incremental or iterative delivery. A decade before Agile was a buzzword in commercial software, using incremental delivery to reduce risk in defense software was not unusual: "To mitigate downstream test risks, TRW has defined an incremental test approach that satisfies TRW and Government objectives for development/integration testing and for formal requirements verification… [that] conforms to DOD-STD-2167A standards."[10] A 1987 Defense Science Board report on military software included a section discussing "rapid prototyping and

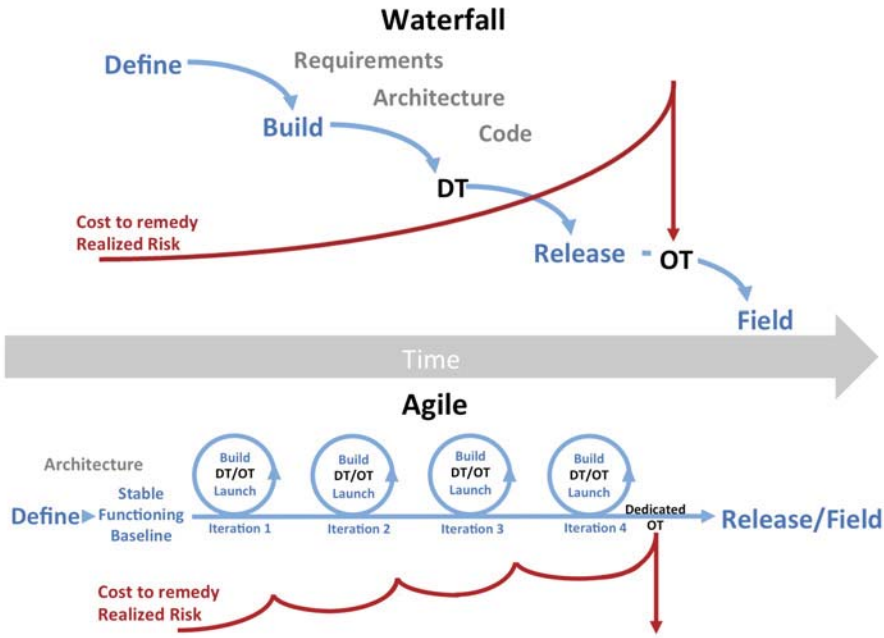

Figure 1: Comparison of Waterfall (top) and Agile (bottom)

iterative development" technology.[11] *That which is done is that which shall be done: and there is no new thing under the sun.*[12]

Although speed was not an explicit principle behind the Agile Manifesto, most commercial and government projects that adopt Agile for software development do so because they expect it to be faster.[13] A 2017 survey of the commercial software industry found that 75% of teams that had adopted Agile management methods did so in order to accelerate software delivery.[14] The DIB's recommendation to adopt Agile is based on the claim that "faster reduces risk by focusing on the critical functionality rather than over-specification and bloated requirements."[15] But Agile is not a panacea. A different 2017 IT industry report, which surveyed 300 US and UK Chief Information Officers (CIOs), found that one-third of Agile projects fail to some degree.[16] Over half of the surveyed CIOs considered Agile "discredited" and an "IT fad." The most common complaints from CIOs against Agile were uncertain timescales; inadequate planning; insufficient documentation ("Agile has been interpreted as a license to abandon the discipline of project documentation."); and ignoring non-functional, enterprise-wide requirements (security, accessibility, system resilience).[17]

The purpose of this article is not to argue for or against Agile in defense systems. Directed by the FY18 National Defense Authorization Act and encouraged by the DIB's Software Acquisition and Practices (SWAP) study, the DoD is pursuing streamlined software development. Given a record of delays, cost overruns, and poor performance in several DoD software acquisition projects, there is clearly ground for improvement. The DIB explicitly recommends the Services adopt Agile based on their finding that "at the present time, DoD's software prioritization, planning, and acquisition processes are among the worst bottlenecks for deploying capability to the field at the speed of relevance."[18,19] The Air Force has prescribed Agile Software Development as a requirement for all new programs unless waived by the Milestone Decision Authority.[20] The first two recommendations of a 2018 Defense Science Board (DSB) report were 1) for the Services to establish criteria for software factories, and 2) to adopt continuous iterative development for software.[21] The Air Force has embraced these recommendations by creating the Kessel Run and Kobayashi Maru software factories to support programs under the Digital Program Executive Office (PEO) (formerly the Battle Management PEO) and the Space Systems PEO respectively.

This article will examine tester involvement in ASD before highlighting the unique challenges of software testing in Agile frameworks. A special focus on criticality and complexity of military systems is emphasized. The Air Force T&E enterprise has addressed a number of the challenges of testing alongside continuous development by insisting on early tester involvement. The *Agile Software Development Test and Evaluation Guide*, issued by Air Force T&E in 2018, details several considerations and best practices for collaboration between

managers, developers, testers, and users.[22] For good or bad, the Air Force and DoD are pursuing Agile Software Development and the T&E community needs to adapt to the unique challenges associated with testing in Agile and continuous development frameworks.

Agile and Early Tester Involvement

At the heart of Agile Software Development, at least as applied to commercial software products, is the notion that the development team should quickly and effectively adapt the software to user desires that may continuously shift and change. This is explicit in the second principle of the Agile Manifesto (*Table 1*). To be effective, Agile methods require a close relationship with the end user. For commercial software, this is done by inviting the end user to be part of the development team. Throughout development, the end user provides continuous feedback to guide further development and improvement of the evolving software. For a variety of reasons, this is difficult to achieve in military systems. A 2019 GAO review of DoD software acquisition for space systems found that early user engagement was mostly ineffective, contributing to significant cost overruns and extended development timelines.[23]

In commercial software development, the ASD developer typically tests their own code while soliciting user feedback on whether the software is doing what the user wants it to do. Because of the critical and unique nature of many military requirements (e.g., interoperability, cybersecurity, safety, effectiveness, suitability), the developer self-test and end-user feedback approach to test is inadequate. Developmental and operational testers who are knowledgeable in the evaluation of the unique requirements for military systems are essential. If a new webmail feature crashes when it is deployed to users, users may be inconvenienced but no one is likely to die from the failure. Military system function and reliability is a different matter altogether.

The commercial practice of embedding representative end users with software development teams is not directly analogous to military systems. Analogs to the specifically trained T&E experts in military operations do not exist in the commercial sector, so the developers rely on customer feedback of early software releases. Early tester involvement is the appropriate analog to early end-user engagement from commercial ASD. However, merely embedding end users with development teams, as done with commercial ASD, is insufficient for critical military systems which usually have unique military requirements. As an example, consider an airman who works in the Combined Air and Space Operations Center (CAOC) and is responsible for planning and tracking aerial refueling in an Area of Responsibility (AOR). Although this airman may be well-qualified for this particular role in the CAOC, the tanker-planning airman is unlikely to understand the very specific requirements for cybersecurity. Likewise, a single air battle manager (ABM) embedded with a software development team would be unable to rigorously conduct the volume, load, and fuzz tests necessary for evaluating the effectiveness of new software before it reaches the CAOC. Nor is identifying the end users of highly-integrated military systems non-trivial. An Air Force senior leader made this point explicit during a recent Kessel Run program review: the "end user" is the combatant commander and the performance metric that matters above all else is combat capability, not coding speed or cycle time.[24]

Early tester involvement, a cornerstone of Air Force Instruction 99-103, Capabilities-Based Test and Evaluation, is also a key enabler for realizing rapid acquisition or ASD. The Air Force's interim policy guidance on middle-tier acquisition states that "Early tester involvement, beneficial in any acquisition strategy, is particularly essential for rapid acquisition."[25] With several Agile projects, the Air Force has adopted the approach of embedding developmental and operational testers with the software development teams. For example, developmental testers from the 45th Test Squadron and operational testers from the 605 Test and Evaluation Squadron are continuously embedded at the Pivotal software factory in downtown Boston as part of the Kessel Run project. These testers are integral parts of the development team (*Figure 2*). Because trained testers are in high demand yet low supply, embedding testers with ASD teams strains limited test personnel and presents the T&E community with the first of several practical challenges to realizing effective ASD.

Agile Challenges to Test and Evaluation

Successful Agile development requires early and continuous tester and user involvement. Because multiple programs are being developed concurrently and the supply of testers and end users is limited, the availability of military testers and end users is a particular challenge for ASD. The availability of military end users is further constrained by the fact that they have a primary job (often in remote locations) that does not involve sustained support to software development. Testers are typically grown from the end-user community and can help close some of the end user participation gap, but tester availability for sustained support is also constrained by the volume of work testers have to do.

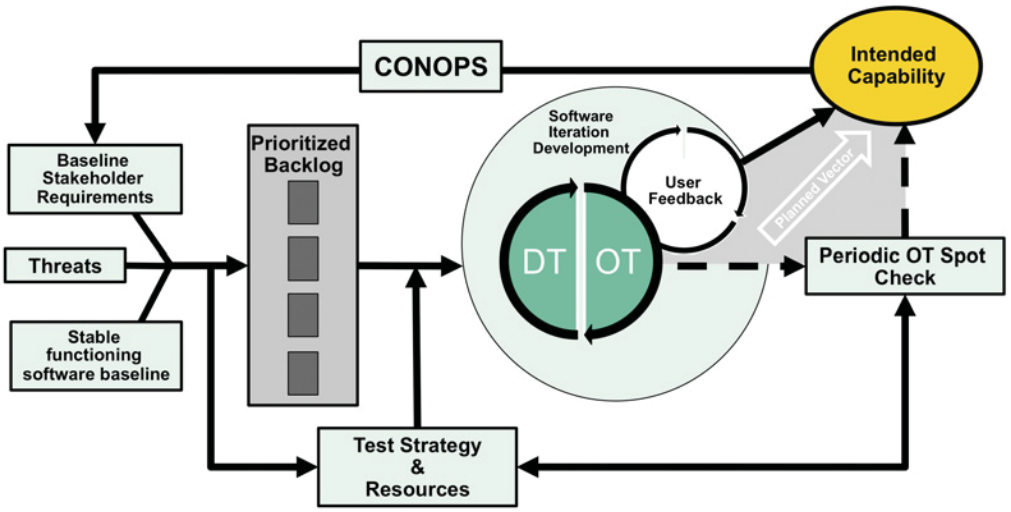

Figure 2: Embedding Testers in Agile Sprint Cycles

The second challenge of testing Agile is that not all software is the same and that the tools developed for Agile testing in the commercial sector are not immediately applicable to military systems. The spectrum of the "software zoo" runs from relatively simple applications (e.g., chat apps) to complex, highly integrated embedded and real-time systems (e.g., operational flight program (OFP) control laws) with a wide range of applications in between (search functions, document management, office automation tools, virtual reality, training devices, simulators, decision support systems, data warehousing, analytic/scientific processing, machine learning, and deep artificial intelligence). Other attributes that distinguish different types of software and systems include the level of user involvement (intense interaction or autonomous), volume of data, persistence of data, time criticality (from hard real-time to soft real-time, and from minutes to years), CPU intensiveness, processing distribution (local, location agnostic, or ubiquitous), update availability (24/7 monitoring systems without scheduled downtime), roll-back requirements, level of interoperability, number of unique interfaces, datalinks, and security. How you build, test, and deploy software depends on what software you are building.

The Agile Manifesto values working software over documentation. Although nothing in the Manifesto argues for zero documentation, in practice, developers commonly rely on user stories instead of established specifications or requirements. Extreme approaches to Agile, e.g., Extreme Programming, eschew any specification of requirements. The lack of detailed requirements and specifications presents a third challenge to testing Agile software. Testers must rely on their judgment and an understanding of the software architecture and backlog to develop test requirements informed by risk management (*Figure 3*). Without documentation, testers often have to build functional control diagrams from scratch to understand all states and flows through the software to ensure it has been completely tested. Determining requirements has never been easy: "The most difficult part of requirements gathering is not the act of recording what the users want; it is the exploratory, developmental activity of helping users figure out what they want."[26] An old software rule of thumb holds that three-quarters of software errors are requirements errors (leaving only a quarter of "bugs" as actual coding errors).[27] This has practical limitations to automated testing, which cannot catch requirement errors, a point we return to below.

A fourth challenge in the test and evaluation of Agile software is the tension between functional and "non-functional" requirements, particularly as they relate to measures of suitability. In waterfall development, program managers can effectively trade cost, schedule, and performance to address end-user needs. Since typical Agile projects establish a fixed schedule up front with set deliveries at the end of every sprint, performance is the only unconstrained parameter (resources are generally also fixed by small team sizes and scaling problems). As such, minimum viable products often do not or cannot address reliability, maintainability, interoperability, etc. Further into development as the technical debt inevitably builds in the program backlog, Agile

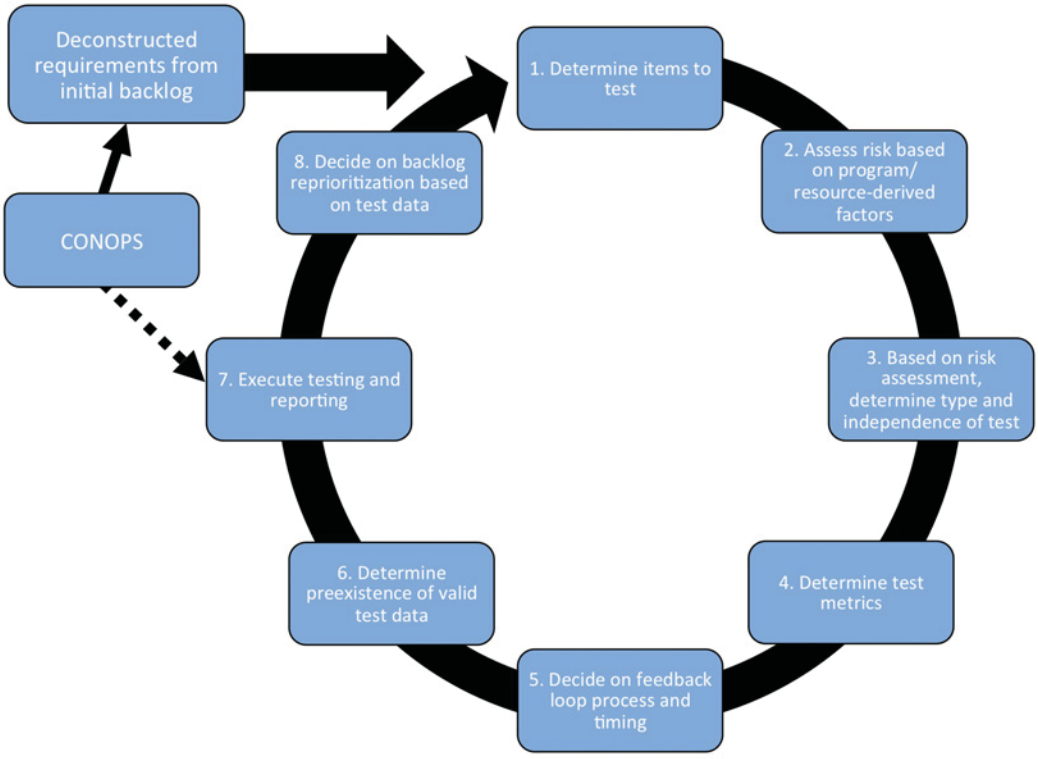

Figure 3: Software Iteration Test-Driven Events

projects tend to continue to prioritize the technical performance of functional features that the user desires over the non-functional. Operational testers can report poor performance on suitability measures, but these tend to persist in the backlog without being addressed. Some suitability measures, e.g., interoperability, can only be addressed up front during architecture design. This is discussed further with respect to system complexity considerations below.

Given these constraints and the wide range of software types, it should be obvious that all software is not appropriate for Agile development. Agile is most effective if functions or features are modular and can be delivered in increments.[28] Many military functions are not (new software for a radar warning receiver without a mission data file to go with it is useless). If individual modules or components cannot be deployed separately or do not have value to the warfighter, the software is poorly suited for incremental delivery.

There is no one-size-fits-all approach to software. The success of Agile in the commercial sector may not warrant a direct comparison to military systems. Indeed, much of the 'success' of Agile, commercial or otherwise, is purely anecdotal. The Defense Science Board admits as much:

> *There are no widely cited or authoritative empirical studies to support the thesis that Agile development practices are superior to Waterfall approaches. Even if there were such studies, they would likely be focused on commercial software and, thus, one might question whether those results would translate to the kinds of software systems that the DoD builds, which are often characterized by a real-time control requirement and a high-end security threat.*[29]

It is not expected that any of the concerns or challenges above will slow the DoD's commitment to Agile. This presents a final challenge of ASD that is more related to the choice of an appropriate acquisition strategy than to test and evaluation: applying Agile to software projects that are poorly suited to Agile. Because of their experience on previous projects, testers can often predict where programs will encounter problems. Testers are the ones to discover and report the flaws; the challenge lies in reporting deficiencies without the taint of smugness. Cautious skepticism is not cynicism or pessimism, though it can be perceived as such. To develop safe, effective, and efficient tools for meeting the challenges of testing Agile, the T&E community must see and work beyond the hype and magic thinking of Agile.

Testing Agile

Writing good software is hard. Testing software, good or bad, is even harder. Agile places a significant emphasis on automated testing. It is not uncommon to encounter Agile purists who insist that automated testing is the sole testing necessary before delivering software to the end user.[30] Many ASD approaches in the commercial sector, particularly those using DevOps, are built around the idea of automated pipelines. In a fully automated pipeline, pieces of code are unit tested using automated scripts and tools. Once a piece of code passes unit testing, it is continuously integrated into the larger ecology, delivered into production, and simultaneously deployed to users.[31] Amazon claims to release new software every 11.6 seconds.[32] If automated testing fails to detect a problem during continuous integration, the bug will be discovered in operations through user feedback. User complaints generate issues and error reports that are placed into the backlog for developers to address during a subsequent iteration and release.

The pace of DevOps in commercial software would not be possible without automated testing. Although no reasonable person expects deployment intervals for military software at rates comparable to commercial DevOps, automated test is an extremely valuable productivity tool. The DIB included automated testing in their ten commandments of software: "Automate testing of software to enable critical updates to be deployed in days to weeks, not months or years."[33] However, automated test alone is insufficient as not all tests can be automated. In the Air Force's Kessel Run project, manual testing by developmental and operational testers regularly reveals bugs that were not identified through automated testing.

ASD typically uses a test-driven development philosophy in which the automated test and code are written at the same time. Under the test-driven development framework, the code is written so that it will pass the test. If the test is not rigorous or complete, unsatisfactory, error-prone code will enter production. As noted above, automated tests cannot find errors in requirements which, as empirical data suggest, outweigh actual coding errors by a factor of three-to-one. Test-driven development can be improved through better or updated requirements (better interpretations of user stories in Agile), which in turn leads to improved tests. The test cards used in manual testing also become useful source documents for writing additional scripts for new forms of automated testing.

Software should also be subjected to performance and edge testing, much of which can be automated (*Figure 4*). Performance testing includes volume testing, load testing, and endurance testing. In volume testing, large amounts of data are used to stress the system, handling, and memory. Load testing and concurrency measure the effects and limits of multiple users employing the system simultaneously. Endurance testing, running the software for long, continuous periods of time without restarts, can reveal memory leakage, clock problems, and other issues. Edge testing includes both fuzz testing (a barrage of random inputs) and link path testing. To be complete and thorough, software should be tested in both software-in-the-loop and hardware-in-the-loop environments.

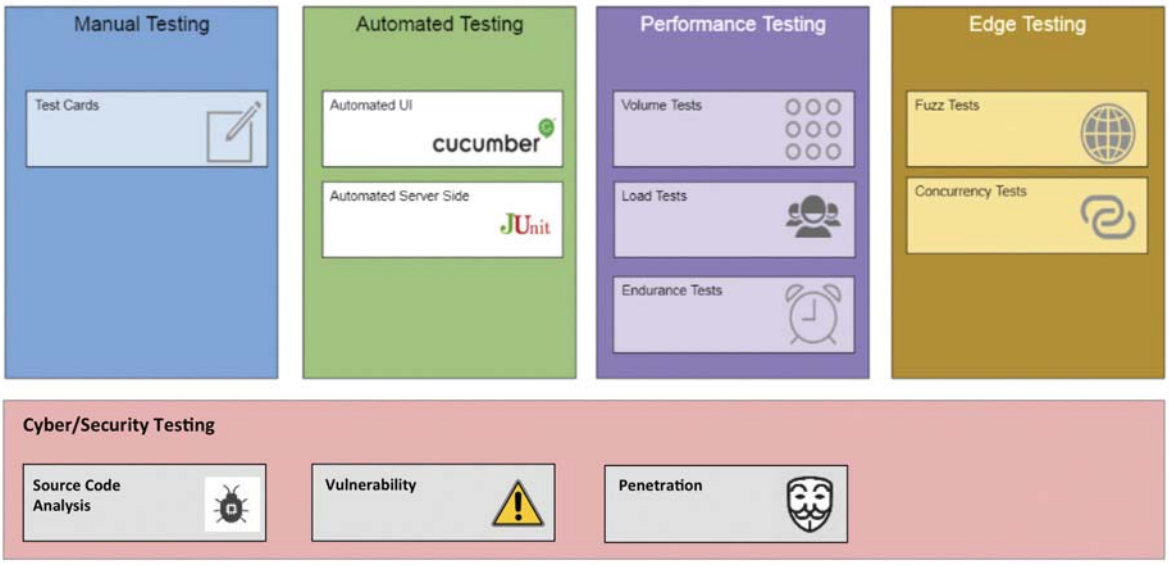

Figure 4: Types of test in Agile DevOps

With commercial software, "operational testing" can be achieved by deploying software to actual users. In the case of critical military software, testing software in actual operations is ill-advised. Military software should be tested in as operationally representative of an environment as possible. This includes hardware, bandwidth, latency, distribution, etc. Code that works fine in the developer's environment may be useless or dangerous when deployed to user systems.

Last but not least, Agile must address the full extent range of cyber and security requirements through rigorous testing. The more critical the system, the more rigorous the test criteria must be. Vulnerability testing can only be automated for known, existing vulnerabilities. Source code analysis is essential for revealing potential attack surfaces and requires specialized expertise. Penetration testing also requires specialized expertise which is hampered by the first challenge identified above, namely, a high demand for the limited number of software red teams.

The ultimate goal of testing is to ensure the delivery of effective and suitable capability to the warfighter. Test data by themself serve little value until they are evaluated with respect to how they will be employed. This requires a clear understanding of the concept of operations (CONOPS). Agile strategies often eschew defining up-front requirements in favor of letting requirements emerge during the course of iteration and user feedback on incremental capability. Regardless of strategy or level of documentation, it is absolutely essential that users clearly articulate their intended CONOPS before development starts. This is a subject we return to in the context of complexity.

System Complexity: Requirements, Architecture, and Systems Engineering

Agile is best-suited for simple systems. Complexity, a hallmark of many military systems, is the antithesis of simplicity. Though there are some military systems that are unnecessarily and gratuitously complex, many are complex by the multi-domain nature of the underlying mission. A complex system is formally defined as a network made up of a large number of nodes or components which interact with each other in unpredictable ways.[34,35] A feature of complex systems is that, due to the sheer number of interactions and possible states (i.e., the complexity) of the system, system-wide outcomes are difficult or impossible to predict from behavior or local attributes of individual nodes.[36]

This definition differs somewhat from the Agile Manifesto's perspective on simplicity. The Manifesto measures simplicity by "the amount of work not done," not by the nature of the possible system states. This matters beyond lexical comparisons. System complexity tends to continuously and inexorably increase, straining the ability to effectively conduct test and evaluation. Agile is not immune to the trends of increasing complexity. A well-known rule of software development, Wirth's law, states that "software systems grow faster in size and complexity than methods to handle complexity are invented."[37] MIT Professor Nancy Leveson, long an advocate for engineering safety into software and complex systems, offers a cogent warning against complexity:

> *We are designing systems with potential interactions among the components that cannot be thoroughly planned, understood, anticipated, or guarded against. The operation of some systems is so complex that it defies the understanding of all but a few experts, and sometimes even they have incomplete information about its potential behavior. Software is an important factor here: It has allowed us to implement more integrated, multi-loop control in systems containing large numbers of dynamically interacting components where tight coupling allows disruptions or dysfunctional interactions in one part of the system to have far-ranging rippling effects. The problem is that we are attempting to build systems that are beyond our ability to intellectually manage: Increased interactive complexity and coupling make it difficult for the designers to consider all the potential system states or for operators to handle all normal and abnormal situations and disturbances safely and effectively.*[38]

Complexity is inescapable. The best way to deal with complexity is through careful architecture design. Initial architecture decisions are software engineering decisions that are often expensive to change after the design has started. Agile purists argue that the architecture will emerge as developers iterate on the product (Principle 11, *Table 1*). Purists even oppose software system engineering approaches such as Big Design Up Front (BDUF) and Big Up Front Architecture (BUFA), which seek to establish a general architecture before starting development. Relying on organic architecture emergence may be suitable for some simple commercial applications, but it is inappropriate for military systems which are usually more complex and often comprised of multiple architectures. Agile does not eliminate the importance of systems engineering.

Although the DSB encourages reduced reliance on specification and documentation, they do insist that

architecture still matters: "While full specifications should be eschewed, emphasis must be placed early on in a project to develop clear, complete, and easily communicated principles of operation. Initial builds with alternate architectures may help to gain sufficient understanding to make an informed choice of final architecture."[39] The DSB addresses concerns of complexity by urging the architect "to define modules in a way that avoids cross-couplings, whereby changes in one module impact and require changes to other modules. DevOps requires careful architectural design to avoid unintended complications by concurrent efforts. In general, this requires carefully defining the module and subsystem interfaces; thorough testing of interfaces is mandatory."[40]

Because complex systems are inherently difficult to analyze, they are also difficult to test. Test design requires a full functional understanding of how the system is designed (architecture) as well as an understanding of how it will be used (CONOPS). Upfront architecture design and a complete understanding of the CONOPS are necessary before embarking on development. This has two significant consequences for test and evaluation. First, because system behavior cannot be reliably predicted from the behavior, a subset of a complex system, component-level testing is not a reliable indicator of overall system performance or even that the component will behave in the same manner when it is part of the larger system. Component-level testing is still important, but results should be weighed with healthy skepticism. Thus, the end-to-end, operational test in an operationally representative environment becomes particularly important for complex systems. Second, because the number of possible states of a complex system grows factorially large, the operational assessment is never really complete. Complex systems must be continuously monitored over time, something for which Agile is well-suited due to the close integration of users, developers, and testers.

Conclusions

"The current approach to software development is a leading source of risk to DoD; it takes too long, is too expensive, and exposes warfighters to unacceptable risk."[41] So begins the Defense Innovation Board report that encourages the prioritization of software development, the adoption of DevSecOps practices, and a culture that prioritizes speed as a critical metric. The Test and Evaluation community has a critical responsibility to ensure that speed is vectored in a direction that serves the needs of the warfighter. "Fail fast is an intoxicating prospect but in practice, it can blur the distinction between continuous improvement and genuine failure. How do you know when a project is actually on the road to ruin? You may be iteratively improving, one failure at a time, towards the wrong outcome."[42]

A new focus on Agile attempts to address DoD's poor track record on software. The Defense Innovation Board acknowledges that "Agile is a buzzword of software development, and so all DoD software development projects are, almost by default, now declared to be 'Agile.'"[43] With nearly every project seeking to brand itself Agile, it is difficult to separate true innovation from hype and fad. Agile purists who imagine themselves in a jihad against waterfall, denouncing independent test as needless bureaucracy, exacerbate the challenge. But such views are not part of the original spirit of the Agile Manifesto. In the words of Jim Highsmith, one of the original seventeen signatories:

> *The Agile movement is not anti-methodology, in fact many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes. We plan but recognize the limits of planning in a turbulent environment.*[44]

In his widely influential and frequently cited 1986 essay, "No Silver Bullet—Essence and Accident in Software Engineering," Fred Brooks[45] argued that "there is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement within a decade in productivity, in reliability, in simplicity."[46] The things that are hard to do will still be hard, regardless of how we approach them. Software is no different. To rapidly deliver effective capability to the warfighter, testers need to get involved with projects at the earliest possible time, regardless of how they are labeled. Debating what is or what is not Agile misses the point of the first principle of the Manifesto: "Our highest priority is to satisfy the customer [warfighter]."[47]

Even as the test community responds to new approaches for managing software development, the landscape will shift. Meeting the challenges of Agile Software Development outlined above is not the end of the challenge. Systems under test will become more complicated and increasingly software dependent. System complexity will inexorably increase and integration and interaction will grow deeper. Systems-of-systems architectures will become more pervasive. Our overall reliance on software is unavoidable. All this will couple

with new fads and ideas in management to create new and significant challenges for Test and Evaluation. If we are to avoid failures such as the Patriot glitch that killed dozens and wounded far more, the Test and Evaluation community must remain committed to its core responsibility. Test and Evaluation is the means by which we ensure the safety of the systems we create as well as characterize their performance in representative environments. The fate of the warfighter and the success of the mission depends on Test and Evaluation to perform this vital function. The new focus on Agile challenges T&E in new ways, but the Test and Evaluation community is adaptable and will doubtless respond. It is too important not to. ❑

Col DOUGLAS P. WICKERT, Ph.D., is the Policy, Programs, and Resources Division Chief for Air Force Test and Evaluation. He has commanded test organizations at the group and squadron levels and is a distinguished graduate of the US Naval Test Pilot School with more than 2000 flying hours in 40 different aircraft. Col Wickert is a distinguished graduate from the US Air Force Academy and holds advanced engineering degrees from MIT and AFIT.

Acknowledgments and Disclaimer

The views expressed are those of the author and do not reflect the official policy or position of the US Air Force, the Department of Defense, or the US Government. This material has been approved for public release and unlimited distribution.

Endnotes

[1] "Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia," Government Accountability Office Report to Chairman, Subcommittee on Investigations and Oversight, Committee on Science, Space, and Technology, GAO/IMTEC-92-26, Feb 1992.

[2] https://www.nomachetejuggling.com/2010/09/10/agile-with-a-capital-a-vs-agile-with-a-lowercase-a/.

[3] B. Boehm and R. Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison Wesley, 2004.

[4] Manifesto for Agile Software Development, 2001. http://agilemanifesto.org/.

[5] D. K. Rigby, J. Sutherland, H. Takeuchi. "Embracing Agile," *Harvard Business Review*, May 2016.

[6] DOD-STD-2167, 4 Jun 1985.

[7] The Defense Innovation Board (DIB) is a group of tech sector executives, academics, researchers, and technologists who advise the DoD on improving acquisition. It was chartered in 2016 and chaired by former Google CEO Eric Schmidt. https://innovation.defense.gov/Members/.

[8] "Chapter 0. README," Defense Innovation Board (V1.4, 11 Jan 19).

[9] "Framework for Test and Evaluation of Agile Software Development," joint AF/TE and SAF/AQ memorandum, 14 Aug 2018.

[10] M. Springman. "Incremental Software Test Approach for DoD-STD-2167A Projects," TRW Systems Engineering & Development Division, TRW-TS-90-02, Jan 1990. https://apps.dtic.mil/dtic/tr/fulltext/u2/a243023.pdf.

[11] "Military Software," Report of the Defense Science Board Task Force, Office of the Under Secretary of Defense for Acquisition, AD-A188 561M, Sep 1987.

[12] Ecclesiastes 1:9, King James Version.

[13] The Manifesto emphasizes early delivery and frequent releases, but these are not in and of themselves necessarily faster. Software delivered early is often necessarily incomplete, with additional features to be added later. Whether or not Agile is any faster than Waterfall in delivering complete capability is an open question, hotly debated by both camps with mostly anecdotal evidence. The few quantitative studies that have been done (e.g., Lee and Xia, 2010) are largely inconclusive.

[14] Version one 12th Annual State of Agile Report.

[15] "Chapter 0. README," Defense Innovation Board (V1.4, 11 Jan 19).

[16] C. Porter. "An Agile Agenda: How CIOs Can Navigate the Post-Agile Era," 6Point6 Technology Services, Apr 2017.

[17] Ibid.

[18] "Chapter 0. README," Defense Innovation Board (V1.4, 11 Jan 19).

[19] "Deliver performance at the speed of relevance" is an explicit goal in the *Summary of the 2018 National Defense Strategy of the United States of America*. The NDS also includes the mandates to "organize for innovation," and "streamline rapid, iterative approaches from development to fielding."

[20] Air Force Guidance Memorandum for Rapid Acquisition Activities, AFGM2018-63-146-01, 13 Jun 2018.

[21] "Design and Acquisition of Software for Defense Systems," Defense Science Board Final Report, Feb 2018.

[22] USAF Test & Evaluation, *Agile Software Development Test and Evaluation Guide*, 29 Oct 2018.

[23] "DOD Space Acquisitions: Including Users Early and Often in Software Development Could Benefit Programs," Government Accountability Office Report to Congressional Committees, GAO-19-136, Mar 2019.

[24] Author's notes.

[25] HQ USAF/TE Memorandum: Interim Rapid Acquisition Test Policy, 9 Jan 2019.

[26] Steve C. McConnell. *Software Project Survival Guide*, Microsoft Press, 1997.

[27] Author correspondence with MIT Professor Nancy Leveson.

[28] D. K. Rigby, J. Sutherland, H. Takeuch. "Embracing Agile," *Harvard Business Review*, May 2016.

[29] "Design and Acquisition of Software for Defense Systems," Defense Science Board Final Report, Feb 2018.

[30] Extreme Agile purists, "Agilists," feel called to proselytize at best and are engaged in a form of Agile Jihadism at worst. In the author's opinion, attitudes of "move fast and break things" are inappropriate for critical military systems. Some systems, such as Nuclear Command and Control, leave little room for "failing fast."

[31] R. Cagle, T. Rice, and M. Kristan. "DevOps for Federal Acquisition," The MITRE Corporation, 2015-2018.

[32] J. Humble. "The Case for Continuous Delivery," Thought

Works blog, 13 Feb 2014. https://www.thoughtworks.com/insights/blog/case-continuous-delivery.

[33] "Defense Innovation Board Ten Commandments of Software," Version 0.14, last modified 15 April 2018.

[34] Neil F. Johnson. *Simply Complexity: A Clear Guide to Complexity Theory*, Oneworld Publications, 2009.

[35] Martin van Steen. *Graph Theory and Complex Networks*: p3, 2010.

[36] D. J. Watts. "The 'New' Science of Networks," *Annual Review of Sociology*, 30: 243-270, 2004.

[37] Tim A. Majchrzak. *Improving Software Testing: Technical and Organizational Developments*, Springer Science & Business Media, (2012).

[38] Nancy Leveson. "A New Accident Model for Engineering Safer Systems," *Safety Science*, Vol. 42, No. 4, April 2004.

[39] "Design and Acquisition of Software for Defense Systems," Defense Science Board Final Report, Feb 2018.

[40] Ibid.

[41] "Software is Never Done: Refactoring the Acquisition System for Competitive Advantage," Defense Innovation Board, TL;DR (v1.5, 11 Jan 19). https://media.defense.gov/2019/Jan/14/2002079285/-1/1/0/TL;DR_TOC_DIB_SWAP_V1.5_2019.01.11.PDF.

[42] C. Porter. "An Agile Agenda: How CIOs Can Navigate the Post-Agile Era," 6Point6 Technology Services, Apr 2017.

[43] "DIB Guide: Detecting Agile BS," Version 0.4, last modified 3 Oct 2018.

[44] Jim Highsmith. "History: The Agile Manifesto," (2001), http://agilemanifesto.org/history.html.

[45] Fred Brooks went on to serve as Chair of the 1987 Defense Science Board Task Force on Military Software where the "No Silver Bullet" statement was repeated. It is interesting to note that the 1987 DSB report found that "in spite of the substantial technical development needed in requirements-setting, metric and measures, tools, etc., the Task Force is convinced that today's major problems with military software development are not technical problems but management problems."

[46] F. Brooks. "No Silver Bullet — Essence and Accidents of Software Engineering," *IEEE Computer*, 20 (4): 10–19.

[47] Manifesto for Agile Software Development, 2001. http://agilemanifesto.org/.