Hyper-Concurrency: The Convergence of Development, Test, and Training

Joshua D. Fields, Tim D. Mobeck, Trevor J. Rossi, Craig L. Smith, Jason J. Valestin
Collins Aerospace
Cedar Rapids, IA, Sterling, VA
joshua.fields@collins.com, timothy.mobeck@collins.com, trevor.rossi@collins.com,
craig.l.smith@collins.com, jason.valestin@collins.com

ABSTRACT

Managing the differences between training systems and the platforms they simulate is a persistent engineering and acquisition challenge. Training system deployments often trail production systems, sometimes by years, due to underlying differences between the two. This delay prevents operators from training on deployed system configurations. To move at the speed of relevance, it becomes critical to create new development and deployment strategies that minimize the gaps between training and production systems. This paper describes the concept of 'hyper-concurrency' in the development and deployment of simulation systems. Hyper-concurrency is the state of deploying working software to both fielded platforms and training systems simultaneously. Hyper-concurrency is attainable by converging development, test, and training environments. To achieve hyper-concurrency, software development architectures for platforms must use the same simulation models, virtual environments, and tools as the test and training systems. By using modern software architectures, applying lessons from ARINC-610, and leveraging advanced simulation, applications can be built for multiple targets simultaneously, tested in high-fidelity simulation environments that replicate the operational domain of the System Under Test (SUT), and deployed to training systems alongside production deployments. Integration of DevOps and Digital Engineering environments enables automation and ensures integrity of the development process. Implementing ARINC-610 functions during aircraft systems development and using simulation technology from the training industry within virtual test environments reduces both the need for costly and slow rehosting activities, as well as development cycle times by enabling rapid integration and automated test for the SUT.

ABOUT THE AUTHORS

Joshua D. Fields is a Senior Software Engineer at Collins Aerospace Simulation & Training Solutions based in Sterling, VA, where he is a member of the Common Modeling and Simulation Environments team. Joshua has a B.A. in Computer Science and Theatre from Northwestern University.

Tim D. Mobeck is a Senior Principal Engineer at Collins Aerospace Advanced Technology with 12 years of experience in Flight Management Systems and Mission Systems. Tim is based in Cedar Rapids, Iowa where he is the Program Chief of Mission Engineering. Tim has a B.S. in Aerospace Engineering from Iowa State University.

Trevor J. Rossi is an Associate Director of Engineering at Collins Aerospace Simulation & Training Solutions with 18 years of experience in Simulation & Training industry. Trevor is based in Sterling, VA, where he is the Chief Engineer for integrated simulation solutions and products. Trevor has a B.S. in Computer Science and a Master of Business Administration from the University of Richmond.

Craig L. Smith, Major, USMC (ret.) is a Senior Principal Systems Engineer at Collins Aerospace based in Cedar Rapids, IA. Craig retired from the Marines in October of 2017. On active duty, he served in a multitude of roles including; KC-130, Joint Terminal Attack Controller, Operational Test Project Officer, and Modeling and Simulation Officer. His twilight tour was as the Deputy Director of the Simulations and Exercise Support Department for the Second Marine Expeditionary Force (II MEF), where he supported II MEF and USMC efforts to train units using virtual and constructive simulations. Craig is a 2013 Master's Program graduate of the Modeling, Virtual Environments and Simulations (MOVES) curriculum at the Naval Postgraduate School.

Jason J. Valestin is an Associate Director of Systems Engineering at Collins Aerospace with 23 years of experience in Systems Engineering, Integration and Test. Jason is based in Cedar Rapids, IA, where he is the Discipline Chief for Strategic Integrated Solutions. Jason has a B.S. in Electrical Engineering from Purdue University.

Hyper-Concurrency: The Convergence of Development, Test, and Training

Joshua D. Fields, Tim D. Mobeck, Trevor J. Rossi, Craig L. Smith, Jason J. Valestin
Collins Aerospace
Cedar Rapids, IA, Sterling, VA
joshua.fields@collins.com, timothy.mobeck@collins.com, trevor.rossi@collins.com, craig.l.smith@collins.com, jason.valestin@collins.com

INTRODUCTION

According to the USAF Operational Training Infrastructure 2035 Flight Plan, training system concurrency is the "condition where the configuration and operation of [a] training system matches the configuration and functionality of the reference weapon system, to the extent necessary to provide required training" (2017). Despite decades of innovation, managing the differences between training systems and the platforms they simulate is a persistent engineering and acquisition challenge. Training system deployments continue to trail production systems, sometimes by years, due to underlying differences between the two. This delay prevents operators from training on deployed system configurations. In order to close the concurrency gap, it is critical to create new development and deployment strategies that minimize the differences between training and production systems.

The Digital Engineering (DE) revolution underway in the defense industry holds great promise for delivering war fighting capabilities at an unprecedented pace. However, rapid capability acquisitions powered by DE will compound the concurrency problem requiring training systems to keep up with newly deployed platform capabilities like never before. Additionally, the adoption of Joint All Domain Command and Control (JADC2) is increasing the need for multidomain training (Dempsey & Editors, 2020). In multi-domain training, concurrency challenges are compounded by the need not only to keep pace with rapid capability deployment, but also to maintain compatibility between training systems across domains in the virtual battlespace (Dempsey & Editors, 2020). Additionally, the complexity of 5th and 6th gen platform Operational Flight Programs (OFPs) and sensors require high-fidelity simulation environments that have their own concurrency requirements. Therefore, new concurrency paradigms are necessary to address the evolving challenges posed by rapid multidomain capability acquisition and deployment.

Literature on training system concurrency spanning decades has focused on a litany of problems and potential solutions for improving concurrency with successes along the way. Past efforts aimed at improving concurrency have concentrated primarily on data availability for simulator manufacturers, simulator design, methods for incorporation of OFPs in training simulators, device test efficiency, cost-benefit analysis, and concurrent development of training devices during the platform Engineering & Manufacturing Development (EMD) phase. A new approach is necessary; one that builds on past successes, integrates modern digital engineering and software development practices, and creates collaboration environments throughout development, test, and training phases of the platform lifecycle. Through the application of this new approach, we propose that the level of concurrency achievable is extraordinarily high, and the deltas between platform development, test, and training systems will begin to blur or become operationally meaningless. We have termed this state 'hyper-concurrency'. Historically speaking, the DoD's goal of concurrency has had limited success with occasional breakthroughs. We need to create systems that are inherently concurrent. We need an approach that accelerates the platform lifecycle. The traditional approach to concurrency is iterative and sequential with a platform update followed by test updates followed by training system updates. Rather, an emergent process is needed by which one can achieve a continuous 'hyper-concurrent' state. Hyper-concurrency is defined as the convergence of platform development, test, and training environments allowing simultaneous deployment of approved release artifacts to fielded platforms and training systems, resulting in training systems that accurately represent fielded configurations.

The following sections layout a vision for hyper-concurrency, suggest paths forward to achieve hyper-concurrency, and identify areas where inroads have been made. The focus of this paper is primarily on engineering. The authors acknowledge that procurement, contracts, program management, and organizational challenges will continue to pose barriers to achieving concurrent systems; however, they are not the main focus of this paper.

THE PROBLEM WITH CONCURRENCY

In recent decades, inroads have been made to improving concurrency; however, many challenges remain. Aircraft avionics are no longer monolithic, immutable, unchanging systems. Gone are the days of steam gauges and fixed instrumentation. Aircraft avionics have evolved, and today, they are complex software-defined systems that can be updated frequently and at times with large new feature sets. The legacy process of creating a training device (simulator) after the aircraft system is fielded is no longer practical due to the aforementioned challenges. Simulating aircraft equipment or integrating real equipment into training devices has been used as an approach to address the concurrency problem. However, as avionics have become more complex, it became impractical to continually update simulated versions in training systems. Additionally, when aircraft equipment is used in a training system it creates added expense and a supply chain issue for production due to competition for platform parts. Today, advances in rehosted avionics have obviated the need to use aircraft equipment in trainers. The current challenge is ensuring that rehosts are available at the right time from original equipment manufacturers (OEMs) and support trainer integration and training patterns.

Another approach to addressing concurrency has been to award a single weapon systems contract that includes training systems development during platform EMD. Early training system development is a key first step in solving the concurrency problem; however, developing the training system with the platform in parallel has created a new set of challenges. First and foremost, it has created a moving target for the training system development. Having a training capability ready by platform Initial Operating Capability, or sooner, is crucial to fielding a weapon system; however, schedule slips, requirements changes, and development priorities all impact the ability to field a trainer.

As platform upgrades bring more and more complexity to existing training devices, new problems have arisen. Today, concurrency is as much about integrating aircraft OFP changes as it is about ensuring that the synthetic environment is represented in sufficient fidelity to support mission training and rehearsal. For example, simulated threat environments require frequent updates to ensure concurrency with the latest national intelligence data and models. Additionally, a platform update that adds a capability such as a Tactical Targeting Network Technology (TTNT) radio to a fielded system not only requires OFP updates and radio model development, but it often requires complex environment modeling such as datalink behavioral modeling and RF modeling, which may not have existed in the test or training environment previously. This challenge could be solved by authoritative models of systems outside the boundary of the system under development that are used across stages of the engineering lifecycle. However, models developed for analysis, development, test, or training rarely are created with such considerations.

Other challenges continue to persist with only incidental progress in pockets across the industry. Inflexible and proprietary simulator designs often require costly updates to replace obsolete compute resources or operating systems. Timely access to data from OEMs remains a problem, particularly when OEMs are faced with limited resources and must prioritize system development over training system support.

Perhaps the biggest problem with concurrency is when the platform architecture does not adequately address training during system design and development. Although platform systems are typically designed for purpose, designed for cost, and designed for testability, they are rarely designed for training. This is likely due to the best overall value not being considered during procurement where requirements could be levied for compatibility with training design patterns (for example the ARINC-610 consideration in Avionics designs). Although, not a technical issue, it is worth noting that training system funding and requirements often come from different program offices with different funding sources, which results in implementation challenges. Simulation is used extensively throughout the engineering lifecycle, particularly in the development, test, and training domains. A lack of standardization, reuse, and interoperability across lifecycle stages limits the ability to create highly concurrent systems.

Platform development practices pose challenges to achieving training system concurrency. Simulations created during platform development often do not translate to the test or training environment. Simulation fidelity may be inappropriate for other purposes due to tooling, timing, and cost. These simulations may not be authoritative or may only simulate the externalities of a segment of platform development sufficient to achieve engineering goals. Likewise, synthetic environments used during development are often not sufficient to replace development activities virtually as required by DE. Systems are often created for specific development and deployment environments, which can prevent reuse across multiple contexts (development, test, and training). Legacy practices including use and development of simulation often do not support the DE paradigms of today's rapidly changing digital world.

Whether performing subsystem test, platform test, developmental test (DT), or operational test (OT), simulation is necessary to test advanced weapons systems. A lack of interoperability between simulation environments and components across disparate customers and suppliers prevents data sharing and deployment of updates to test environments. Simulations created during platform development often do not translate to the test realm. Delivery and deployment pipelines stop at organizational boundaries preventing rapid integration of system and simulation updates. Accurately representing the synthetic environment sufficient for OFPs integration into virtual operational domain poses significant challenges for test environments. Security requirements and compartmentalized systems without multi-level security environments limit interoperability and integration within and across platforms.

Bespoke training systems suffer from a lack of commonality with technological solutions used by platform developers and testers due to a variety of factors. Simulation environments created during platform development often do not translate to the training realm either due to the lack of real-time nature or incompleteness with respect to the operational domain. Simulations for testing may exist at an interface level or conversely require much higher fidelity and system in the loop capabilities that would be cost prohibitive to include in a training system. Further, platform system designs that do not consider training system needs during development will continue to require extensive rework and integration during update cycles. And finally, the lack of authoritative operational domain simulation usable across the platform lifecycle hinders the ability of a training system to integrate new platform capabilities as previously noted.

A VISION FOR THE FUTURE

Solving the problems laid out in the previous section can be accomplished, but in order to do so, there needs to be convergence among development, test, and training. Only by removing barriers to interoperability, simulation reuse, and data sharing across the digital engineering lifecycle can the concurrency challenge be laid to rest.

Hyper-concurrency is not simply a matter of process, rather it is a state to be achieved. Programs can attain this state through the application of five hyper-concurrency tenets. The five key tenets for achieving hyper-concurrency are discussed below. There is not a one-sized fits all approach to hyper-concurrency, and there are many paths, as specific implementations will differ based on the needs of a program.

Tenet #1 – Design Systems for Development, Test, and Training

To achieve reuse across development, test, and training environments, systems must be designed with the requirements for all of them in mind. If test and training requirements are not considered upon initial design, it becomes significantly more difficult to ensure reuse of applications in each subsequent environment. Hyper-concurrency can only be achieved when collaboration occurs between different teams, suppliers, and customers. The criteria are:

1. Training-specific capabilities (e.g. ARINC-610) must be added to platform requirements.
2. A shared collection of authoritative simulation models is used across all phases of the lifecycle - development, test, and training - by OEMs, suppliers, customers, and training system manufacturers.

Tenet #2 – Perform Integration Early in the Lifecycle

System integration needs to begin in the development environment. When feature creation occurs in a vacuum, it often results in features designed in isolation from the overall system. The criteria are:

1. Developers work in virtualized environments that enable early integration.
2. Feature creation is completed in vertical slices, where features are integrated and tested to produce functional units.

Tenet #3 – Embrace a Test-Oriented Process

Features are tested throughout their lifecycle. A program can identify issues earlier in the product lifecycle by testing during the creation of said features, resulting in reduced cost and schedule risk as compared to when those issues are found later in the process. The criteria are:

1. System tests are developed and run prior to the deployment of new features to production and training.
2. System tests are run alongside development rather than as a distinct testing phase.
3. Unit tests are used to ensure requirements are met.

4. Automated testing exists that validates both training and production workflows.

Tenet #4 – Create Near Ops Environments for Development, Test, and Training
A Near Ops environment is a virtualized operationally representative execution context that allows us to replicate how a system is deployed in the real world. The goal is to replicate the operational domain such that it is indistinguishable from reality from the perspective of the system under development. It is paramount that consistent, working updates are provided to training and production systems. Programs must be able to replicate issues that arise within training and production systems in development and test environments. Development and test must have environments that are nearly identical to training and production. This allows the developer/tester to replicate issues/bugs found downstream in the lifecycle process. Using Near Ops environments assures that code working in one environment will work in all. The criteria are:

1. A Near Ops environment exists where virtual integration and test can occur.
2. The Near Ops environment replicates the operational domain context of the system under development.
3. Test runs an identical software load out to both training and production instances, testing for both use cases from the start.

Tenet #5 – Deploy to Training and Production in Parallel
Features are deployed in parallel to training and production environments. Features should be developed in such a way that identical artifacts can be deployed to training and production. As much as possible, differences to how training and production systems interface with new features are kept to a minimum. The criteria are:

1. Software is developed to run in containers to allow dependencies to be built into deployment solutions, resulting in closer deployment processes between training and production.
2. Training systems reuse simulation environments created across the lifecycle resulting in reduced rehost activities.

ACHIEVING HYPER-CONCURRENCY

Achieving hyper-concurrency is a challenge. It will require a confluence of customer requirements, purposeful design decisions, and technology investments. The following sections detail the design considerations and technology investments necessary to move toward a hyper-concurrent future.

Greenfield vs. Brownfield Platform Development
The barriers to achieving a hyper-concurrent state are dependent on whether a platform update is greenfield, new development, or brownfield, legacy development. For example, a new 6^{th} generation fighter where the production and training systems are new development has more flexibility to implement the methods, practices, and designs required to achieve hyper-concurrency than a $4^{th}/5^{th}$ generation fighter that is already fielded and undergoing a platform update.

On greenfield development, program schedule is partially driven by key requirements such as certifications, or ground testing, flight testing, and operational test & evaluation (OT&E) that cannot be completely eliminated with Digital Engineering tools and processes. If the tenets discussed in the *Vision for the Future* section are considered in program design, programs will be inherently hyper-concurrent, even if the aforementioned immutable requirements are present. Additionally, when the criteria described in the tenets are applied during initial development, they will provide a solid foundation for hyper-concurrency and the opportunity to reduce the overall program schedule. It is paramount that greenfield development activities incorporate a near ops environment early in the process so that it can be used across development, test, and training.

During brownfield development, achieving hyper-concurrency is much more difficult. Transformational change will require investment in development, test, and training improvements driven by customer concurrency requirements. The challenges with existing platform modifications are worthy of a separate paper on the topic. Therefore, the remaining sections will discuss achieving hyper-concurrency goals on new platform development.

Converge Simulation Environments

The concurrency challenge cannot be met without evolving our ability to replicate operational constructs at every stage of the Digital Engineering lifecycle. We must converge simulation environments across lifecycle stages and ensure that development environments, test environments, and training environments match to the maximum extent possible. Creating a simulation environment sufficient to meet the goals of development, test, and training and to move at a sustainable pace is key to achieving inherently concurrent systems. A common, authoritative simulation environment can address challenges posed by concurrency requirements by enabling rapid development, simulation-driven virtual integration, scenario-based automated test, and concurrent deployment of new training system capabilities. A common approach to simulation will unlock value from existing unintegrated capabilities and allow developers to focus on creating solutions for the customer rather than developing complex simulation infrastructure and simulation models for a single use (development, test, or training). Efficiencies can be realized by reducing duplication of simulation efforts, breaking down silos, and increasing reuse of simulation assets across platform lifecycle stages. Application of open standards and MOSA architectures will ensure that simulations are interoperable with suppliers, customers, and end users, enabling deeper integration and collaboration. A converged simulation environment will provide the foundation for hyper-concurrency. The following sections detail the superset of capabilities and needs for such an environment.

Account for Needs Across the Platform Lifecycle

A converged simulation environment must account for the needs of each lifecycle phase. Without a shared simulation environment between development, test, and training, the ability to share components ranging from simulations to rehosted operational software applications will be reduced and will drive additional schedule, cost to the total program, and limit the ability to achieve concurrency. Figure 1 depicts a lifecycle for a program that would continuously be updated.

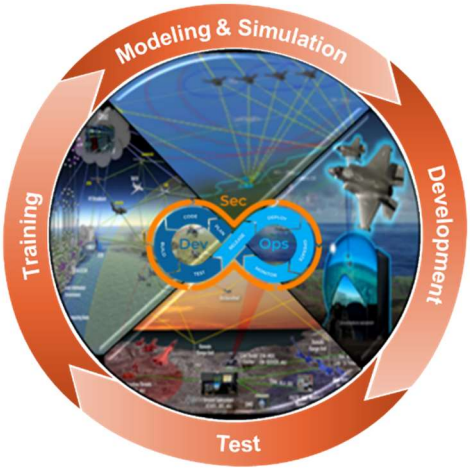

Figure 1. Modeling & Simulation Lifecycle

Development – The environment should assist in system requirement and design definition. It should also be capable of integrating with tools necessary to perform hardware or software development; along with low level or unit level testing on the component being developed. The simulation environment should support performing a System of Systems (SoS) integration using a mix of real and virtual systems. An example would be integrating multiple new radios into an avionics system. Integration should support human on the loop as well as human in the loop. The use of Live, Virtual, and Constructive (LVC) paradigms would be a key tenet of the SoS integration as well as during training.

Test – This same environment should be able to be used for test; inclusive of contractor test, government DT, and OT&E. The test environment should support individual bit manipulation for subsystems as well as high-fidelity scenario-based testing that leverages physics-based dynamic simulation of the operational domain of the system under test at sufficient fidelity to achieve a trusted test outcome with system-in-the-loop.

Training – The simulation environment should support dynamic and scenario-based test cases. Simulator functions supporting operations such as initialize, reset, freeze, snapshot, and malfunctions are essential to achieving training goals but also support test and development activities. Simulation interoperability is a prerequisite to support distributed training and integration in joint training environments. Recent trends from live training to virtual training require the synthetic training environment to achieve higher fidelity levels that will allow more of the training curricula to be performed virtually. As such, the simulation environment must be interoperable with other training systems through distributed simulation standards.

Faster Than Real Time

The simulation environment should provide real-time execution, resulting in deterministic run time with periodic scheduling and management of models and simulations. However, for developers to be able to rapidly iterate on new features, they need to be able to quickly see how their code functions within the context of the overall system. Full

system integration tests can often run for hours or even days at time, preventing the ability for efficient iteration. By supporting Faster Than Real Time (FTRT) simulation execution, developers can quickly see how their new features work within the context of the larger system, and unit and systems tests can be executed orders of magnitude faster than human in the loop testing. A FTRT mode can also support the need for an accelerated capability within training simulators.

Hardware / Software in the Loop

The simulation environment needs to easily support integration of 3rd party operational software and simulations. It should support a distributed network topology and be highly scalable for up to and including large-force n-participant cooperative simulation events. It should also easily transition between use of virtual models, simulations, and rehosted software, to include a mix of real hardware and software components in a virtual testbed. There should be native support for Hardware in the Loop (HWIL) testing, allowing system testing to occur at incremental system maturity levels. The HWIL infrastructure should incorporate discrete interfaces, analog interfaces, as well as standard digital bus protocols (such as MIL-STD-1553, ARINC 429, RS-232, etc.). Platforms with complex software integration require a Software in the Loop (SWIL) testing capability to run software application(s) on target hardware, while having the rest of the system still function in the virtualized environment. This allows for reuse of aspects across the different environments. An example would be a set of software test procedures run daily on builds in the virtualized environment, and when that virtualized software is loaded onto target hardware, the same test procedures can be executed with no modifications. The environment should extend from a purely virtualized environment, to a mixed of real hardware and/or software in the loop, to integrating external live assets for test and training. The ability to leverage virtual, SWIL, and HWIL environments across the lifecycle is key to achieving hyper-concurrency.

Virtualize Everything

According to Dr. Will Roper in *Bending the Spoon, "*Digital Engineering must achieve a measure of authoritative virtualization that replaces, automates, or truncates formerly real-world activities" (2021). An authoritative virtualization is a trusted simulation of a system that can be used as a surrogate for the real system throughout the engineering lifecycle. Authoritative virtualizations are important because they allow us to shift activities that are costly and slow into the virtual domain in order to shorten acquisition cycles and reduce sustainment costs.

Hardware Virtualization

Virtualized environments are a must for distributed development. Virtualized environments help to reduce program cost due to the reduction of dependence on physical assets. Given today's atmosphere where work-from-home initiatives are more prevalent than in the past, the dependence on virtualized environments as opposed to physical assets has dramatically increased. The virtualized environments should be built for distributed simulation and cloud deployment using containerization (e.g., Docker) and other cloud-native architectural methods which provide scalable capacity and worldwide connectivity between partner sites. This is needed to allow collaboration between companies, so they can integrate independent and jointly developed applications.

With virtualized environments of sufficient fidelity and containerized software, identical software can run in many kinds of compute, adjusting for the needs of the program phase. This includes both on-premise and cloud compute, allowing for program readiness as the defense industry pushes towards a more cloud-oriented world. Isolating software from the underlying hardware architecture allows deployment to any platform, VMs, Kubernetes clusters, cloud environments, and target hardware. This hardware virtualization methodology assures that software built once will run in any environment whether development, test, and training.

Imbue Digital Twins with Training Technology

Digital twins are authoritative virtualizations that can be used for a variety of development, test, and training use cases. A "digital twin is a digital representation of a real-world entity or system. The implementation of a digital twin is an encapsulated software object or model that mirrors a unique physical object, process, organization, person, or other abstraction" (Gartner, 2022). The convergence of development, test, and training requires that the digital twins of platforms be adaptable to all three use cases; however, this does not mean that each use case has the same fidelity requirements nor that the digital twin must replicate every single aspect of the physical twin in true fidelity. Instead, there is a completeness constraint for all simulations and digital twins. A digital twin needs only to be complete with respect to the system interfaces and of sufficient fidelity to stimulate said system. It is important to recognize the use cases of converged environments for development, test, and training and the limitations of any "digital twin solution"

from vendors. Hyper-concurrency will undoubtedly require custom digital twin solutions supporting development, test, and training. This does not imply that hyper-concurrency dictates the creation of a superset of digital twin tools, but that digital twins created during the lifecycle be interoperable across use cases to the maximum extent practicable. Test and training environments require the integration of real avionics, whether a hardware or software solution, to be effective. Therefore, it is imperative that engineering digital twins slated for use in test and training include all capabilities required for those use cases.

Virtualize the Operational Domain with Authoritative Representations

To reduce cycle times during development, trusted testing outcomes must be reached quickly, and to create effective up-to-date training, the converged simulation environment must faithfully replicate the real world. It is insufficient to virtualize the system under development without also virtually replicating the operational domain in which the system operates. Digital twins need a synthetic world in which to execute missions. The Near Ops environment used during development and test should form the basis for the synthetic training environment providing a high-fidelity authoritative virtual world in which to train. The environment should provide tunable fidelity where appropriate to achieve development, test, or training objectives. For example, the physics-based virtual RF simulation that provides accurate signal attenuation and multipath effects necessary for a test environment may not be desirable for a procedures trainer where training objectives may not require degraded communications. Creating a shared authoritative simulation environment is a challenge for industry; however, simulation environments such as the DoD's Joint Simulation Environment (JSE) are steps towards creating an authoritative representation of operational domain contexts that can be reused across development, test, and training paradigms.

Embrace Digital Engineering and DevOps to Accelerate Concurrency

Digital Engineering (DE) is "an integrated digital approach that uses authoritative sources of systems data and models as a continuum across disciplines to support lifecycle activities from concept through disposal" (DAU Online). DevOps is the combination of development methodologies, practices, and tools that increase the ability to deliver systems at high velocity. Digital Engineering and DevOps practices are essential to achieving hyper-concurrency by reducing cycle times and connecting the previously unconnected across lifecycle phases.

DevOps Key Principles

The 'Three Ways' of DevOps provide principles that can help achieve hyper-concurrency. The First Way of DevOps, "Flow/Systems Thinking" (**Error! Reference source not found.**) emphasizes the performance of the entire system rather than a single silo of work. Traditionally, development, test, and training are separate silos each individually optimized to achieve schedule, cost, and performance goals. However, thinking of platform development, test, and training as a single system in which flow must be understood, maximized, rework minimized, and optimized across the entire system can provide a profound understanding of how to achieve hyper-concurrency for a platform.

Figure 2. The First Way

Hyper-concurrency relies upon rapid iteration of new features across multiple environments simultaneously. The Second Way of DevOps, "Amplify Feedback Loops" (**Error! Reference source not found.**) stresses the need to shorten cycles and ensure that working software and systems are passed downstream. Within traditional concurrency methods, it is not unheard of that defects are found in platform systems during training device integration, even after the platform had been updated in the field. To achieve a hyper-concurrent state, lengthy feedback loops are nonstarters. By using Continuous Integration / Continuous Deployment (CI/CD) automation across traditional silos, feedback loops can be created, and full system (development, test, and training) cycle times can be reduced. In an increasingly software-driven world, the test approach becomes crucial to achieving DevOps flow and fast feedback loops. Within these pipelines, builds can be created and tested for multiple environments immediately following a code commit. If builds or any tests fail, the developer can receive an immediate notification about the issues. If these builds and tests succeed, they can be

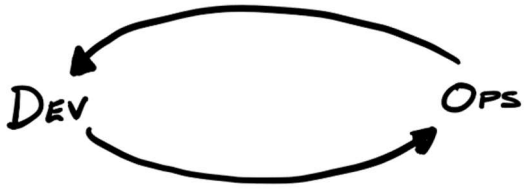

Figure 3. The Second Way

immediately merged into subsequent development or production environments, including releasing any artifacts, and then made available to test, training, and production systems. In conjunction with the use of CI/CD pipelines, Infrastructure as Code can be used to store the details of each environment's infrastructure configuration, allowing the CI/CD pipeline to spin up its own test environment using the same virtual near ops environment across all lifecycle phases.

The Third Way of DevOps, "Continuous Experimentation and Learning" (**Error! Reference source not found.**), enshrines a culture that fosters taking risks, learning from failures, and understanding that practice and repetition builds mastery. Much the same way that DoD training systems require concurrency, industry and its customers must embrace the hyper-concurrency goal and build competency in the domain, while understanding that there will be failures along the way. However, only by taking the risk and learning from failure can the reward be reached.

Figure 4. The Third Way

Model Based System Engineering (MBSE)

The International Council on Systems Engineering (INCOSE, 2007) defines MBSE as "the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases." In more practical terms, the vision for using modeling that MBSE ascribes transitions Systems Engineers away from the legacy processes of requirements capture using written/narrative text and instead leverages modeling languages (e.g., UML, SysML) to describe the system requirements. The modeling language approach combined with modern digital engineering software tools means that the system is more clearly defined and ensures better alignment between the requirements generator (customer) and the developers building the system. Modern MBSE software tools enable code stub creation which ties the implementation backwards through the requirements. These stubs are then passed along to software developers, who then create the code for the subsystem. After code is written, it can be fed back into the MBSE tool which created the stubs based on the design, improving the MBSE models, thus reducing friction between systems and software developers and completing a necessary feedback loop for attaining hyper-concurrency. MBSE as an authoritative source of truth for an entire platform inclusive of its training system can enable hyper-concurrency by managing requirements and design for all at the same time; allowing the impact of system upgrades to be modeled within modeling languages and applied across development, test, and training.

Design for Concurrency and Collaboration

A convergence of environments, tools, and practices alone is insufficient to achieve hyper-concurrency. Designing for concurrency from the earliest stages of platform development must occur. A hyper-concurrent design can be achieved by understanding the needs at program onset of the entire platform lifecycle inclusive of test and training requirements and collaboration environments between contractors.

ARINC 610C

Perhaps the "most important consideration in designing for concurrency is simulating the avionics systems such that future changes to the aircraft configuration and software can be incorporated into the simulator with minimal time and resources (Caylor, 2002)." ARINC Report 610C, *Guidance for Design of Aircraft Equipment and Software for Use in Training Devices*, is a key tool in achieving training system concurrency. ARINC Report 610C (hereafter A610C) "sets forth the general philosophy and basic design guidance for using aircraft equipment (hardware and/or software) in training devices. It specifies unique simulator functions that shall be supported by this equipment" (ARINC, 2009). Simulator functions or "trainer-isms" such as Freezes, Repositions, Snapshot, Faults, Stores Management and Speed Times N are essential to implementing effective training systems. However, getting those features integrated with a newly released OFP in a timely manner is often the crux of the concurrency challenge. A610C helps solve that problem, but over the years, it has been applied inconsistently across the avionics industry and is often not part of the initial system design, particularly within the military avionics realm. A610 must be a requirement for avionics. A comprehensive overview of A610C and how it can be used to support concurrency of training systems is available in the past IITSEC paper *Concurrency – A Moving Target.* The same functionality defined by A610C for training systems can be used to enable rapid iteration in development and test environments, helping to create the convergence of

development, test, and training that is needed for hyper-concurrency. Platform systems and simulation models that can readily initialize, reposition, and execute faster than real time during development, integration, and test can be integrated into DevOps toolchains and enable rapid execution of simulation-based unit tests and scenario-based acceptance testing. A610 should not only enable training but also power development and test environments.

System Certification

Beyond the environment used, the architecture and system design of the actual product should consider how to limit the impacts to functions / capabilities that require certification efforts. For example, as the threat environment changes and the mission capabilities required need to be changed, the system should not have to go through an entire certification effort again as this is costly and more importantly runs counter to achieving hyper-concurrency due to the time required to go through certification.

The system design should provide a separation between the safety critical and mission functionality. This would allow the safety critical portion to be certified a single time and then have a continuous authority to operate as the mission system changes constantly, as the mission system has less test/certification rigor. Safe-guards or preventative mechanisms would need to be put into place to prove that changes to the mission system could not impact the certification of the safety critical components. One design methodology to achieve separation may be a modular microservice-based architecture. Where certification is required, hyper-concurrency can only be achieved when a continuous authority to operate is attained; allowing feature development without impacting system certification. Without the continuous authority to operate, updates to the system would require new certification efforts.

Collaboration

Collaboration between the government customer, the OEM and their suppliers, and the training system provider is critical to the attainment of a hyper-concurrent state. It is important to design for collaboration when designing for hyper-concurrency. This can be accomplished in a number of ways.

A common interoperable environment is a key component for collaboration. Regardless of who designs the environment, industry or the customer, the benefits discussed in the previous sections are achievable if the tenets for hyper-concurrency are followed. A technique for collaboration that has been found to be effective is to share the same environment with a customer during integration and then provide that same environment to the producer of the training system to use as a basis for the trainer. Another way to design for collaboration is to ensure that there is schedule synchronization across all parties involved in the process. To facilitate delivery and deployment of system updates, development timelines need to complement each other such that different groups align and do not fall out-of-sync.

There are also other constraints that need to be considered, such as data rights and the ability to share Intellectual Property and proprietary data between disparate vendors. More work and coordination are required in this area to help drive to hyper-concurrency. Hyper-concurrency can be achieved when coordination and agreements are made at the beginning of a program to ensure all vendors share a common framework, as well as a common set of tools and processes.

APPLIED HYPER-CONCURRENCY

This section details progress towards hyper-concurrent systems and the convergence of development, test, and training. The following example covers a convergence of development and test with deployment to a manned-unmanned teaming (MUM-T) Human Machine Interface (HMI) simulator approximating a training device. While it does not include sustainment of a training system, it shows steps toward achieving hyper-concurrency.

The authors applied many of the hyper-concurrency tenets during the Development and Test phases of a recent Unmanned Mission System program. During the program, teams developed and integrated Minimum Viable Products (MVPs) based on given scenarios/requirements, with incremental demonstrations for customer feedback. This methodology segmented the customers' desired end-state mission into individual capabilities and allowed for rapid incremental development. During the project's 15-month duration, the team used a DevOps pipeline to rapidly stand-up three environments for development, integration, and test. A Software-In-the-Loop (SWIL) environment was created, followed by a Faster-Than-Real-Time (FTRT) environment to speed up testing. Finally, a Hardware-In-the-

Loop (HWIL) environment was used to evaluate the mission system on target hardware. These environments used a common simulation baseline and built upon each other to increase levels of fidelity in the system integration lab (SIL).

The CI/CD pipeline used commercial off the shelf (COTS) tools and automation scripts to build and deploy the software to each environment. The team unit tested the mission system software with the FTRT environment using a subset of the mission system applications with simulation plugins running on a single developer's machine. Once verified locally, modified software is moved into the SWIL/HWIL environment for scenario-based testing.

Agile and DevOps techniques enable rapid software development, integration, and testing. When the SIL was initially stood up, some CI/CD pipelines were absent: an engineer had to manually install software on each test fixture requiring at times a full day to complete. After CI/CD pipeline implementation, software changes could be implemented, tested, compiled, and deployed within minutes. On a typical day, developers built locally on the FTRT environment as often as once an hour, testing their changes with simulation-based tests prior to checking them in. Then, simulation environment software and mission system software are built and deployed to the SWIL and HWIL environments in the SIL up to five times a day. This process is only limited by the duration of the two-hour mission through the synthetic environment. CI/CD pipeline agility allows rapid response to customer inputs on overall mission design, as well as tactics and autonomous behaviors.

The use of the DevOps pipeline facilitated the creation of a virtual System Integration Lab (vSIL). The vSIL is a complete synthetic duplicate of both the mission system software and architecture, and the simulation software and architecture. Using the vSIL improved designs without purchasing physical hardware or needing real-world testing to discover problems. This provides a standardized ‘clean slate’ start-up for every run, minimizing the risk of incorrect initializations between tests. A significant benefit of the vSIL is that it deconflicts resource utilization between the simulation team and the mission system software development team, allowing both teams to develop and test in parallel.

Upon completion of the baseline effort, the team extended these processes and tools to flight test. In just three months, the team took the digital twin and integrated two COTS UAS Ground Stations that allowed for full control of both the live and virtual UAS platforms. Using the FTRT environment, the team was able to rapidly configure and test the autonomy with the new autopilot and scenario. By running thousands of FTRT scenarios in a matter of days on the ground with the integrated ground controllers, the team built confidence in the solution and went on to have three successful days of flight testing with no lost assets.

This same environment was used to power a cockpit demonstrator (Figure 5) for another team working on HMIs for unmanned systems. The demonstrator was very similar to a flight training device and required occasional updates as system development progressed. This demonstrator could then be employed in a relevant operational context to evaluate the HMI and recommend design changes. Building once and deploying to development, test, and demonstration environments provided a solid foundation for different teams to work across different aspects of a platform for a common purpose.

Figure 5. Cockpit Demonstrator

CONCLUSIONS

The digital engineering revolution is providing the tools necessary to accelerate not only platform development but training systems deployment and concurrency as well. After decades of work and progress on concurrency, the development of inherently concurrent platform training systems is within reach. Building on the practices of digital engineering and DevOps, concurrency updates can match the pace of rapid platform feature development. Hyper-concurrency is possible with the tools and technology available today. Through thoughtful, intentional design, converged simulation ecosystems, and the use of digital engineering tools, programs can become hyper-concurrent and deliver the platform updates and training systems critically needed by the warfighter.

ACKNOWLEDGEMENTS

The authors would like to thank James Merchant, who coined the term hyper-concurrency, as well as our SMEs and paper reviewers: Jim Anderson and Nick Burgart.

REFERENCES

ARINC Report 610C (2009, September 11). Guidance For Design of Aircraft Equipment and Software for Use in Training Devices. Aeronautical Radio, Inc. Annapolis, MD.

Caylor, Jim (2002). Concurrency – A Moving Target. I/ITSEC.

DAU - Glossary Content (n.d.). Retrieved June 10, 2023, from https://www.dau.edu/glossary/Pages/GlossaryContent.aspx?itemid=27345.

Dempsey, T., & Editors. (2020, July 7). Why Multi-Domain Training Requires Concurrency and Low Latency. Retrieved May 11, 2023, from https://modernbattlespace.com/2020/07/07/why-multi-domain-training-requires-concurrency-and-low-latency/.

INCOSE. (2007, September). Systems Engineering Vision 2020. INCOSE-TP-2004-004-02.

Haseltine, Eric C (1989). Trainer Concurrency – Problems and Solutions. I/ITSEC.

Gartner. (n.d.-a). Glossary: Digital twin. Retrieved June 7, 2023, from https://www.gartner.com/en/information-technology/glossary/digital-twin.

Object Management Group, Inc. (2022, August 8). What is a Digital Twin? Digital Twin Consortium. https://www.digitaltwinconsortium.org/faq

Potter, Dwight W, John C. Larson, David R. Sando (1992). Training System Challenge Lowering the Cost of Concurrency. I/ITSEC.

Schradin, R. (2023, April 26). Digital Engineering the Key to Moving at the Speed of Innovation. Retrieved May 11, 2023, from https://modernbattlespace.com/2023/04/26/digital-engineering-the-key-to-moving-at-the-speed-of-innovation/.