UNCLASSIFIED

DEPARTMENT OF DEFENSE
DEFENSE SCIENCE BOARD

DESIGN AND ACQUISITION OF SOFTWARE FOR DEFENSE SYSTEMS

Co-Chairs:
The Honorable William LaPlante
Dr. Robert Wisnieff

Findings and Recommendations

UNCLASSIFIED

Terms of Reference

- The DoD and defense industrial base need to capitalize on the opportunities provided by commercial sector improvements in software development techniques and practices. Therefore, the Task Force should:
  - Examine the current state of DoD software acquisition and recommend actions for DoD and its suppliers
  - Consider development, test, and evaluation of learning systems
  - Contrast and compare DoD and commercial software development and determine what commercial software development capabilities the military systems should embrace
  - Identify impediments in DoD requirements, contracting, and program management and how they might be removed
  - Determine if “agile” software techniques are being used effectively and identify impediments
  - Determine if the commercial concept of a minimum viable product should be adopted by DoD
  - Determine best management approaches to achieve rapid and effective software upgrades, including an analysis of modular, open architecture
  - Look at lessons learned from recent software challenges (OCX, F-35)
  - Provide recommendations to ensure rapid adoption of cognitive capabilities as they mature

Task Force

- Chairs:
  - Dr. Bill LaPlante
  - Dr. Bob Wisnieff
- Executive Secretary:
  - Mr. Jim Thompson
- Advisors:
  - Mr. Joe Heil, Navy
  - Ms. Cindy Schurr, Air Force
- Task Force Members:
  - Dr. Victoria Coleman
  - Mr. Chris Lynch
  - Dr. John Manferdelli
  - Dr. Joe Markowitz
  - Mr. Bob Nesbit
  - Dr. Paul Nielsen
  - Mr. Mark Russell
  - Dr. Fred Schneider
  - Mr. Lou Von Thaer
  - Mr. Alfonso Velosa

Briefings

- OCX GPS Program Office
- Joint Fighter F-35 Program Office
- CAPE (cost estimation)
- Defense Procurement and Acquisition Policy (incentives)
- Pierre Chao (software sustainment)
- Dick Ginman (Intellectual Property)
- USAF Rapid Capabilities Office (open architecture)
- USAF Expeditionary Combat Support System (ECSS)
- U.S. Army RDECOM (FACE program)
- House Armed Services Committee
- AT&L/C3, Cyber, and Business Systems (C3CB)
- Intel Software Day
  - OUSD/AT&L/SSI, ODNI/SRA, NRO, NGA, NSA
- San Jose Fieldtrip
  - Google, IBM, Facebook, Kaggle, Brave Software, Qualcomm
- Defense Digital Services (DDS)
- 18F
- Code for America
- Carnegie Mellon University (deep/machine learning)
- Raytheon
- Lockheed Martin
- Boeing
- SpaceX

Importance of Software in Defense Systems

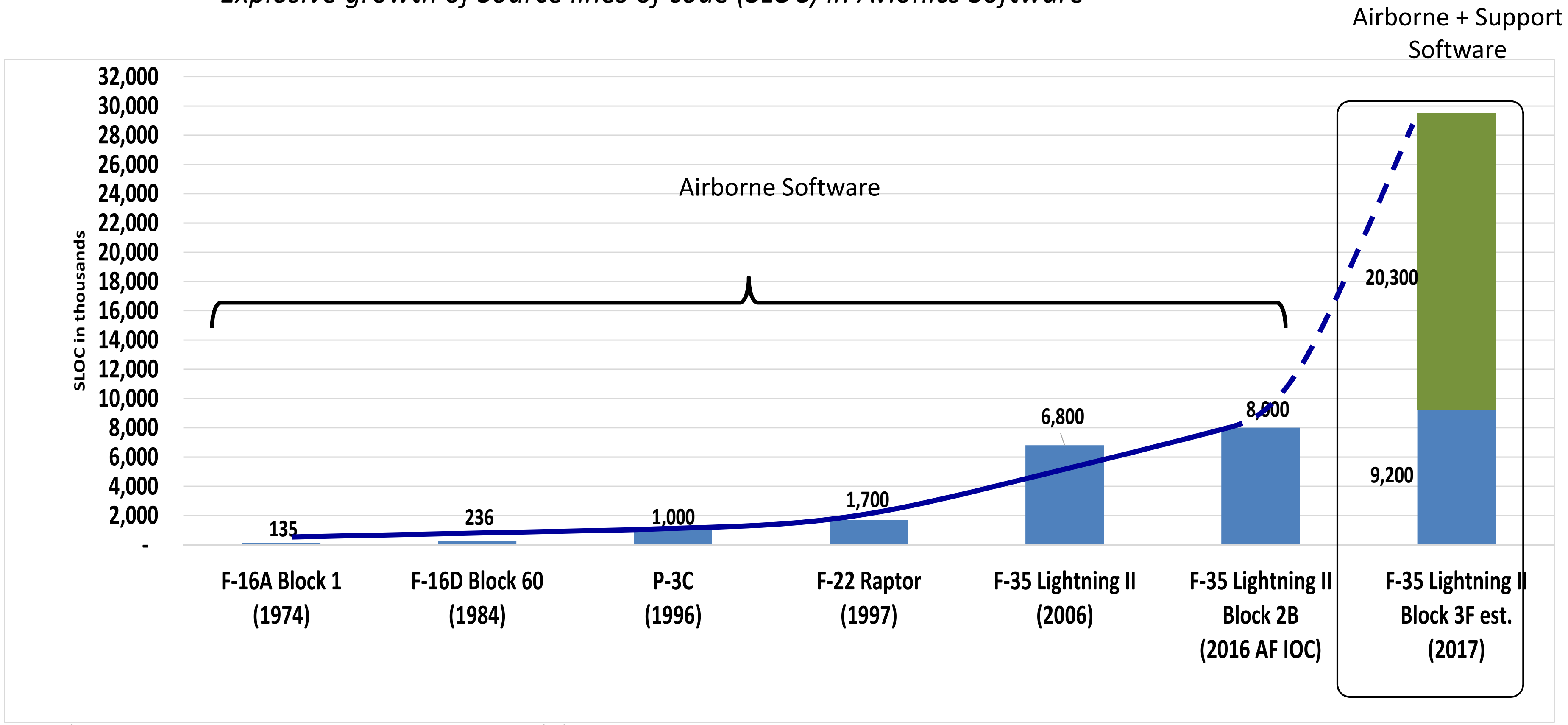
- Software is a crucial and growing part of weapon systems/national security mission
  - "The DoD is experiencing an explosive increase in its demand for software-implemented features in weapon systems…in the meantime, defense software productivity and industrial base capacity have not been growing as quickly."
    –Institute for Defense Analyses, 2017

- Software never dies. It will require DoD to update continuously and indefinitely

DoD Software Growth

- DoD Software complexity and size rapidly growing

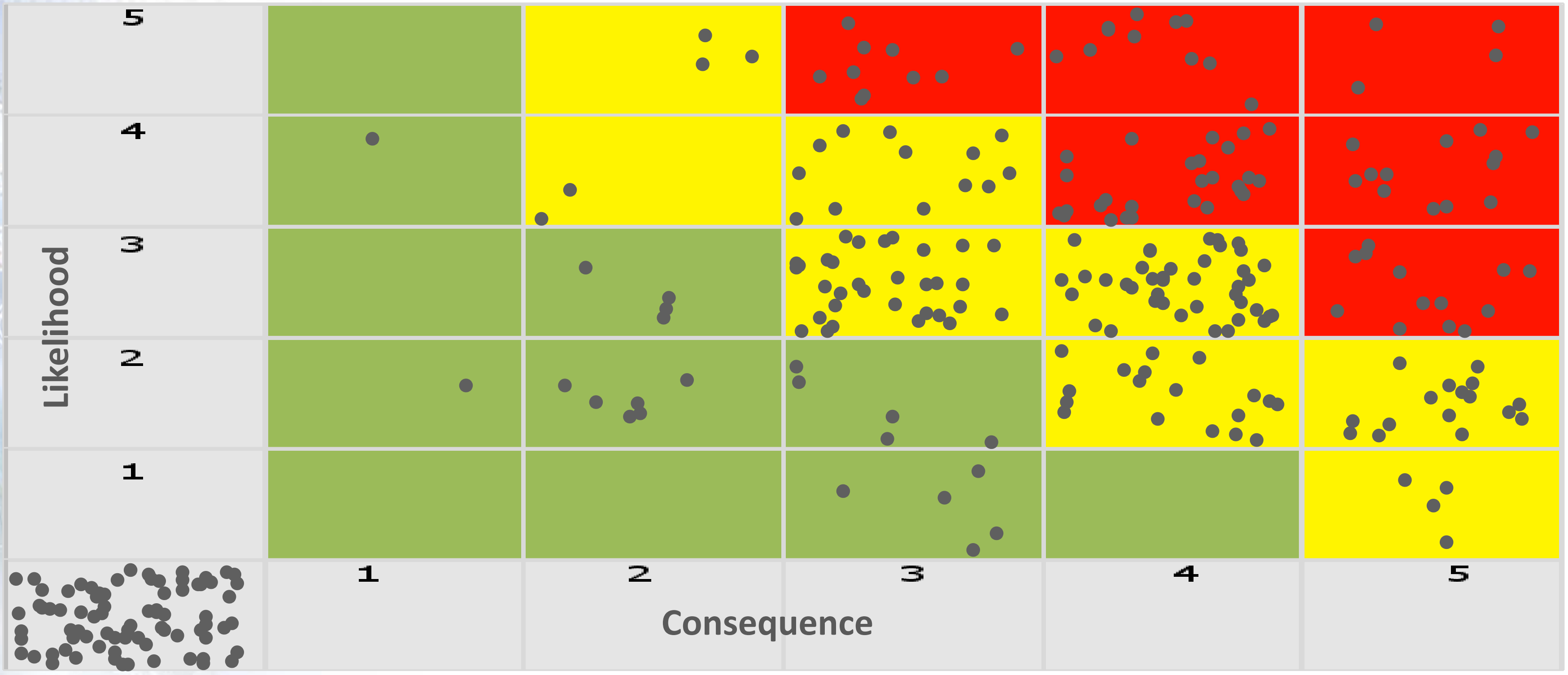
Explosive growth of Source lines of code (SLOC) in Avionics Software



Note: SLOC for F-35 Block 2B, 3F and Support SW, KC-46 Source: 2017 DASD (SE)
SLOC for F-16 and F-22 are at first operation flight Source: "Software- The Brains Behind US Defense Systems", AT Kearney, "A historical compilation of software metrics with applicability to NASA's Orion spacecraft flight software sizing", Judas, Paul A, and Prokop, Lorraine E., Innovations in Systems and Software Engineering: A NASA Journal, DOI 10.1007/s11334-011-0142-7, 2011 NASA

Software Risk Assessed by DoD Program Offices

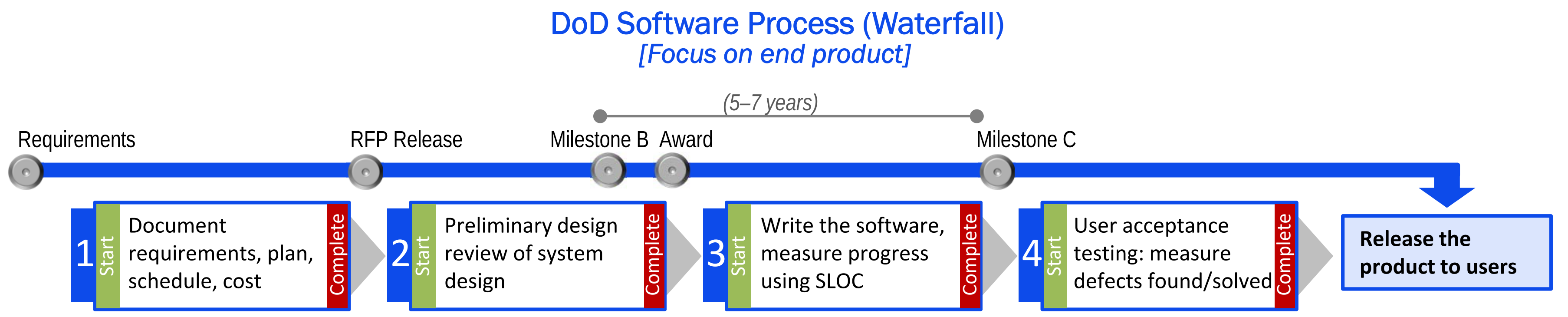
FY14 - FY16

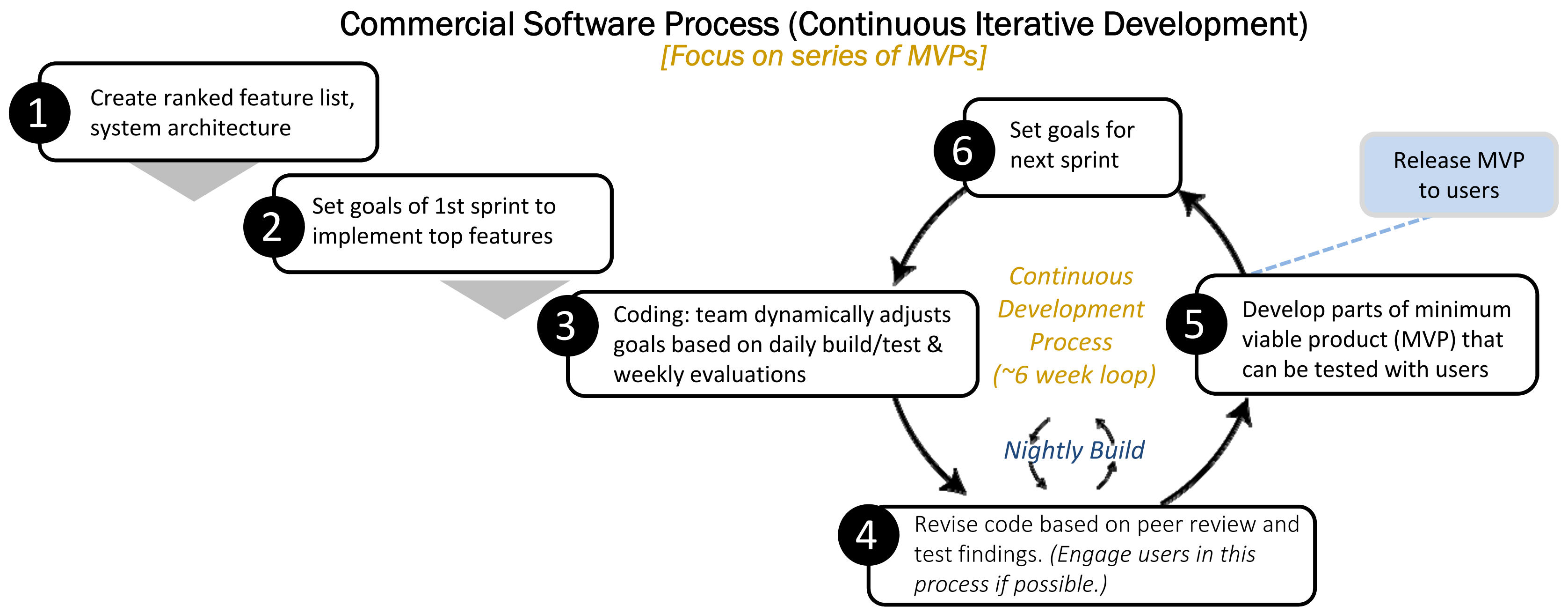


Software not in top program risks

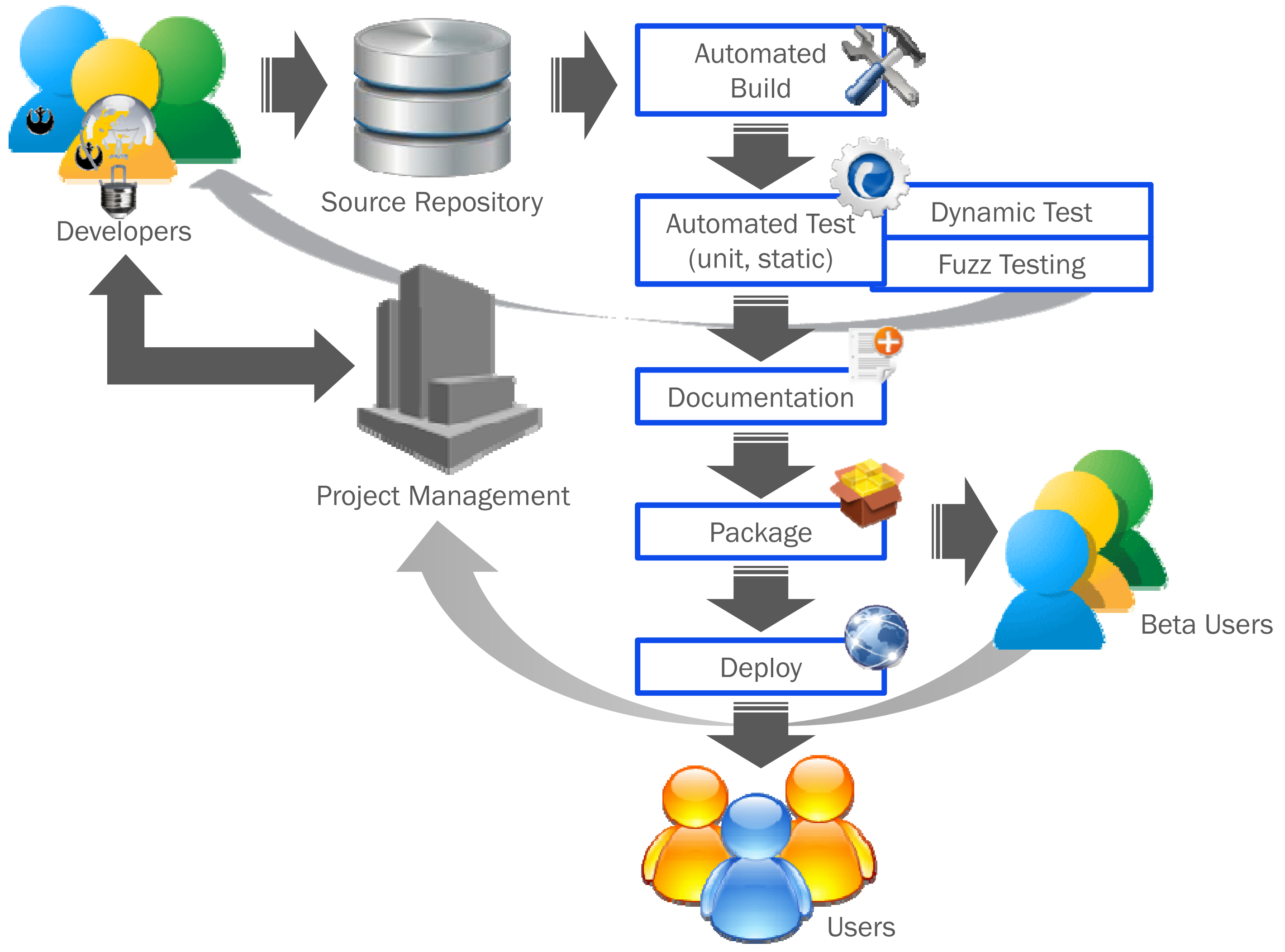
Software assessed among most frequent and most critical challenges, driving program risk on ~ 60% of acquisition programs

DoD vs. Commercial Software Process

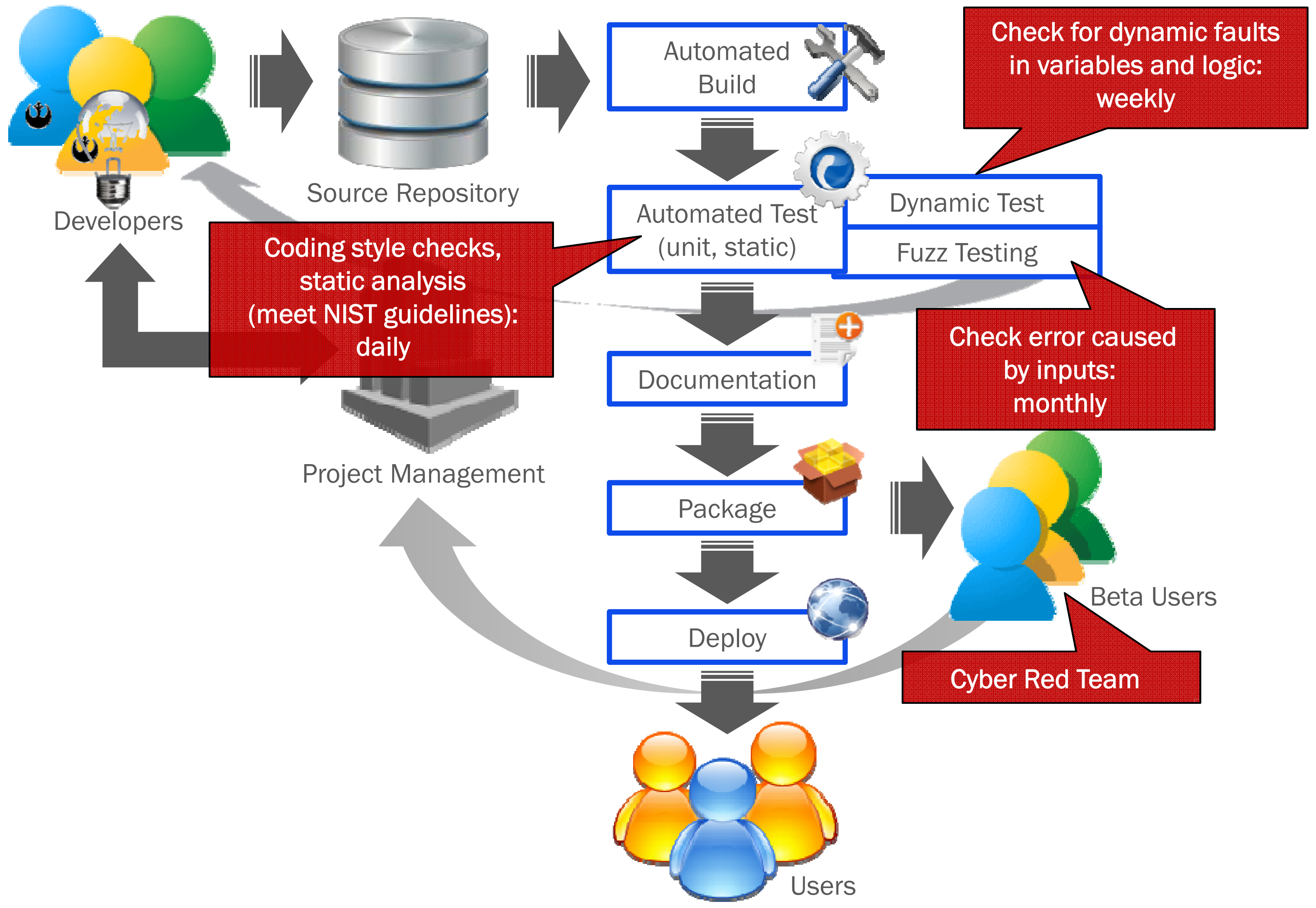

DoD Software Process (Waterfall)
[Focus on end product]
(5–7 years)
Requirements
RFP Release
Milestone B
Award
Milestone C
1 Start Document requirements, plan, schedule, cost Complete
2 Start Preliminary design review of system design Complete
3 Start Write the software, measure progress using SLOC Complete
4 Start User acceptance testing: measure defects found/solved Complete
Release the product to users


Commercial Software Process (Continuous Iterative Development)
[Focus on series of MVPs]
1 Create ranked feature list, system architecture
2 Set goals of 1st sprint to implement top features
3 Coding: team dynamically adjusts goals based on daily build/test & weekly evaluations
4 Revise code based on peer review and test findings. (Engage users in this process if possible.)
5 Develop parts of minimum viable product (MVP) that can be tested with users
6 Set goals for next sprint
Release MVP to users
Continuous Development Process (~6 week loop)
Nightly Build

Software Factory



Addressing Cyber


Developers
Source Repository
Automated Build
Automated Test (unit, static)
Dynamic Test
Fuzz Testing
Documentation
Package
Deploy
Users
Beta Users
Project Management
Check for dynamic faults in variables and logic: weekly
Coding style checks, static analysis (meet NIST guidelines): daily
Check error caused by inputs: monthly
Cyber Red Team

Importance of Architecture

- DoD systems are complex
  - This means software architecture is difficult to communicate effectively
  - This can lead to incomplete specification that interferes with coherent implementation
  - This makes it difficult to do efficient parallel development.
- Special emphasis must be made early in a program to develop a clear and complete and easily communicated software architecture that can be used by large implementation teams

Introduction

- Commercial development best practices (as done in Silicon Valley) allow software production rapidly — and continuously — and can adjust more efficiently
  - New tools and techniques being utilized (automation at scale)
  - Computing power has increased and cost has fallen
  - Static, dynamic, and fuzz testing techniques have allowed substantial, automatic software testing
  - Open source appears prevalent and growing
  - Continuous — in development and testing (billions of hours of usage of its software every day)
- It is the Task Force's assessment that DoD is significantly behind the commercial sector (though bright spots exist)
- DoD can leverage the development best practices to its advantage, including on its weapons systems. This will enable DoD to move from a capabilities-based to a threat-based acquisition — increasing speed to respond
  - Adversaries are increasingly able to present us with capabilities we have not anticipated
- However, defense contractor base is not at the same state of adoption of commercial development best practices
  - DoD needs to change internal practices and encourage/incentivize practices in contractor base
  - DoD develops software and associated contracting based on detailed systems requirements/ specifications (this approach was heavily utilized 20 years ago — systems engineering flowdown)

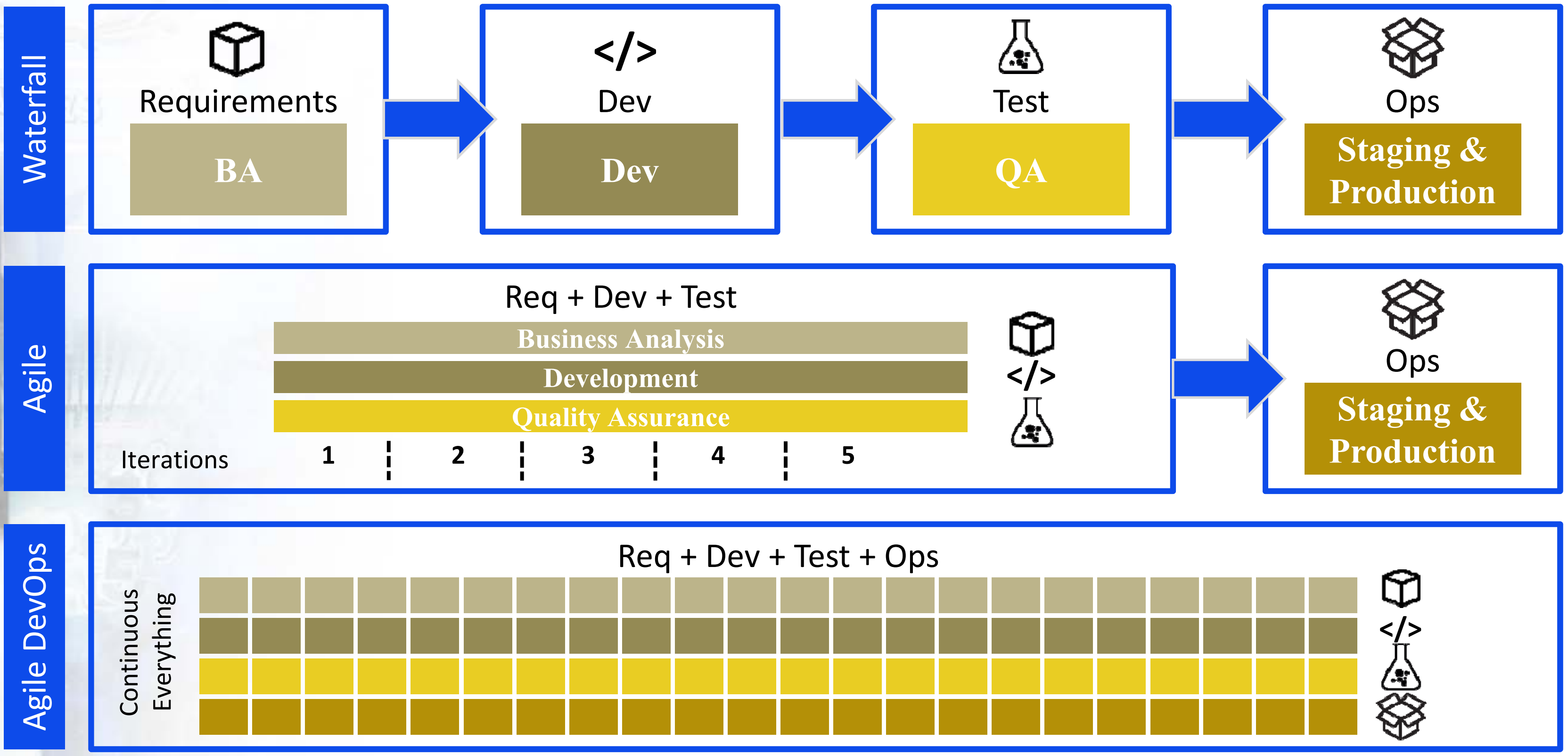
FINDINGS

Continuous Iterative Development For DoD

Silicon Valley Baedeker: Theories of Software Development

How did we get here?

Shift from Waterfall to Agile, from Silos to Collaboration



Source: Hewlett Packard Enterprise, FedInsider, Intel

Iterative Development: Agile, Spins and Spirals

- Iterative development is the ineluctable process imposed by use of a product — especially a software product — which reveals a shortcoming or suggests a new improvement
- What distinguishes traditional iterative development from newer software design and development is the velocity and granularity of the iterations
- In venerable software production methodology (waterfall development) the iterations are commonly at the end-product level after field deployment and use
- Newer constructs — agile/spiral/spin — are able to uncover and deal with flaws and opportunities sufficiently early in the process, efficiently leading more robust product delivered to the field

Harvard Business Review: Embracing Agile

The Right Conditions for Agile

Conditions	Favorable	Unfavorable
Market environment	▪ Customer preferences and solution options change frequently.	▪ Market conditions are stable and predictable.
Customer Involvement	▪ Close collaboration and rapid feedback are feasible. ▪ Customers know better what they want as the process progresses.	▪ Requirements are clear at the outset and will remain stable. ▪ Customers are unavailable for constant collaboration.
Innovation Type	▪ Problems are complex, solutions are unknown, and the scope isn't clearly defined. ▪ Product specifications may change. ▪ Creative breakthroughs and time to market are important. ▪ Cross-functional collaboration is vital.	▪ Similar work has been done before, and innovators believe the solutions are clear. ▪ Detailed specifications and work plans can be forecast with confidence and should be adhered to. ▪ Problems can be solved sequentially in functional silos.
Modularity of Work	▪ Incremental developments have value, and customers can use them. ▪ Work can be broken into parts and conducted in rapid, iterative cycles. ▪ Late changes are manageable.	▪ Customers cannot start testing parts of the product until everything is complete. ▪ Late changes are expensive or impossible.
Impact of Interim Mistakes	▪ They provide valuable learning.	▪ They may be catastrophic.

SOURCE: BAIN & COMPANY

Harvard Business Review: Embracing Agile

The Right Conditions for Agile

Conditions	Favorable	Unfavorable
Market environment	▪ Customer preferences and solution options change frequently.	▪ Market conditions are stable and predictable.
Customer Involvement	▪ Close collaboration and rapid feedback are	▪ Requirements are clear at the outset and
Innovation Type	unknown, and the scope isn't clearly defined. ▪ Product specificat ▪ Creative breakthr are important. ▪ Cross-functional	▪ Detailed specifications and work plans can be forecast with confidence and should be red to. ems can be solved sequentially in unctional silos.
Modularity of Work	▪ Incremental deve customers can us ▪ Work can be bro conducted in rapid, iterative cycles. ▪ Late changes are manageable.	omers cannot start testing parts of the uct until everything is complete. changes are expensive or impossible.
Impact of Interim Mistakes	▪ They provide valuable learning.	▪ They may be catastrophic.

Platform Mission software, EW, Communications, Radar, Launch systems

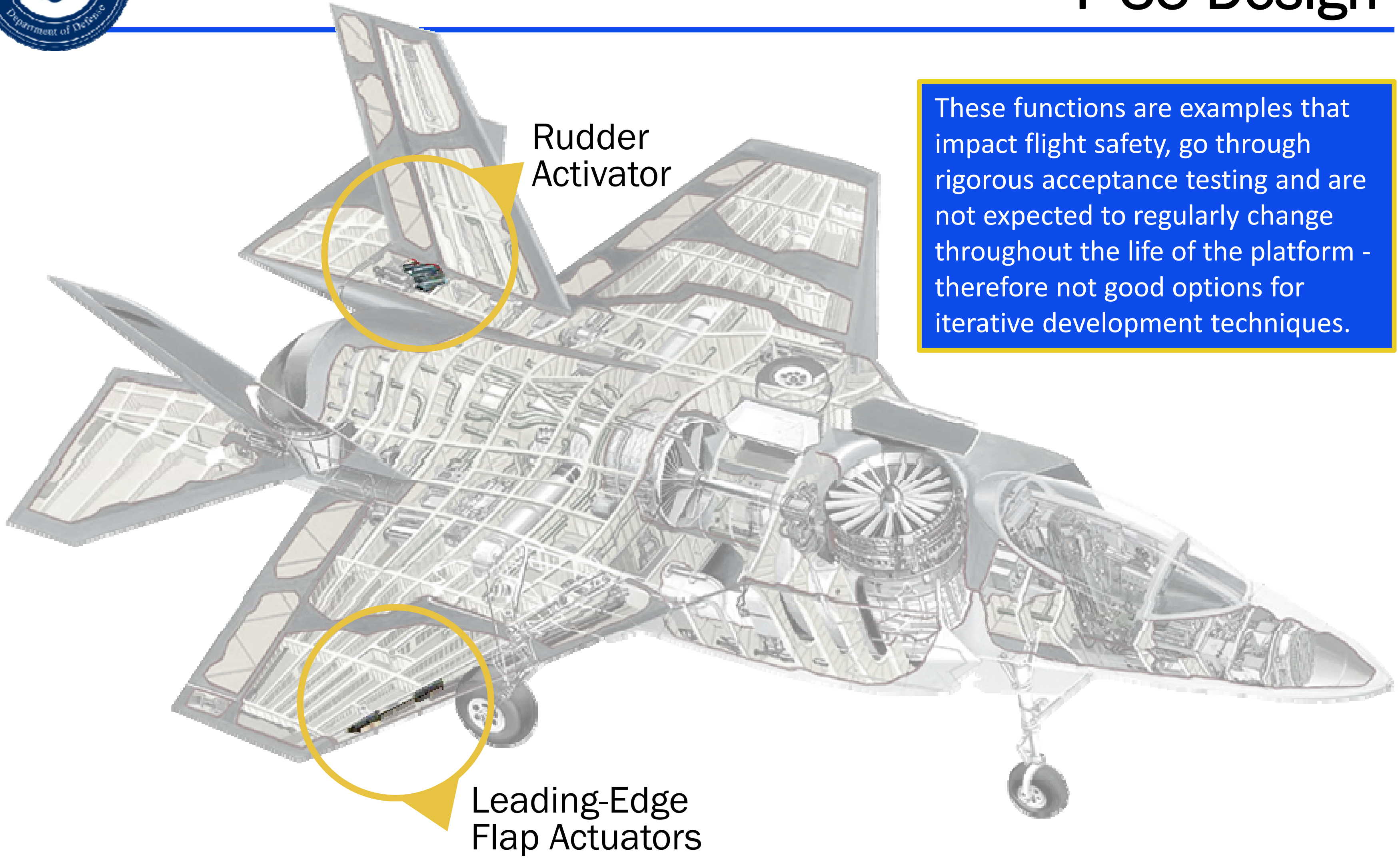
Digital Engine Control Systems, Low Level Mission Critical Flight Control Systems, Legacy systems at end of lifecycle

Ground control systems, Command and Control

Enterprise Logistics Support Systems

SOURCE: BAIN & COMPANY

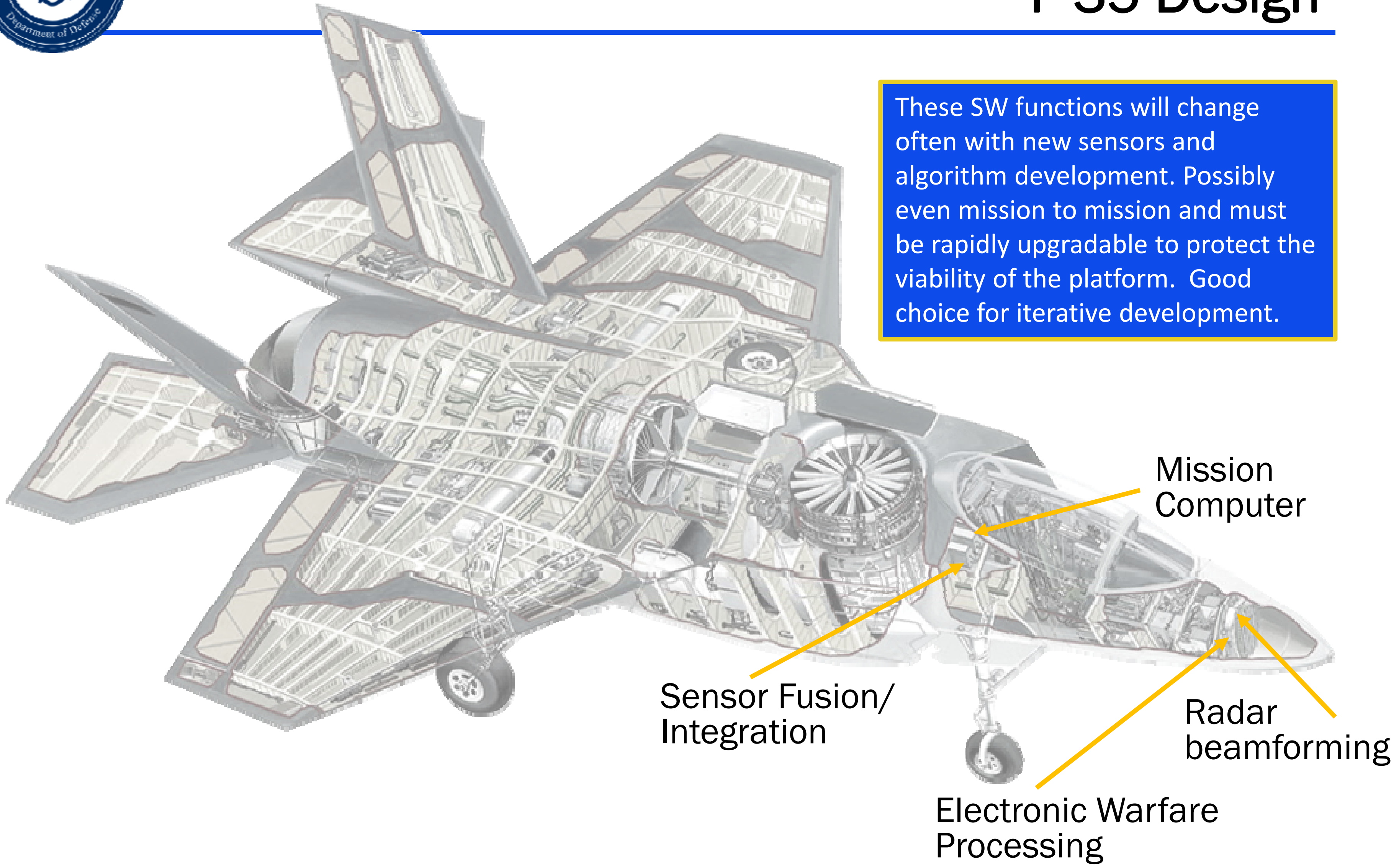
F-35 Design



These functions are examples that impact flight safety, go through rigorous acceptance testing and are not expected to regularly change throughout the life of the platform - therefore not good options for iterative development techniques.

F-35 Design

These SW functions will change often with new sensors and algorithm development. Possibly even mission to mission and must be rapidly upgradable to protect the viability of the platform. Good choice for iterative development.



Summary of the Case For/Against Agile

- Agile / Continuous Iterative Development makes sense "in theory" for the software found in many of the weapons systems DoD builds
- Published empirical data is incomplete (therefore: not convincing)
  - We have **no direct confirmation / refutation** for whether benefits will be achieved by DoD contractors transitioning to Agile / Continuous Iterative Development for weapons systems.
  - Widespread adoption of these approaches by industry suggests benefits are being seen in that setting.
  - No reported transitions back to "waterfall" development approaches.

Empirical data and strong industry movement to agile development across all domains strongly motivates DoD to move to agile development.

Agile Expectations vs. Experiences

- Two meta-studies do survey literature that gives empirical comparisons.
  - Tore Dyba and Torgeir Dingsoyr. Empirical studies of agile software development: A systematic review. Information and Software Technology 2008.
  - David F Rico. What is the ROI of Agile vs Traditional Methods. 2009 https://davidfrico.com/rico08g.pdf
- Meta-survey of 36 empirical studies prior to 2005 [Dyba and Dingsoyr].
  - Four studies give empirical data for productivity comparison of agile and traditional ("waterfall") development. Most focus on XP ("extreme programming") form of agile.

Study	Traditional Prod	Agile Prod	Productivity Gain
S7	3 LOC/hr	13.1 LOC/hr	337%
S10	3.8 LOC/hr	5.4 LOC/hr	42%
S14	300 LOC/month	440 LOC/month	46%
S32	157 LOC/engr	88 LOC/engr	-44%

S7 involved 15 teams used 4 different approaches. Greatest difference shown.

Note: Agile team delivered far more code, but the same functionality as traditional.

S14 agile team had more experience with languages and management

S32 is a study concerning student programmers.

Published Empirical Comparisons (2)

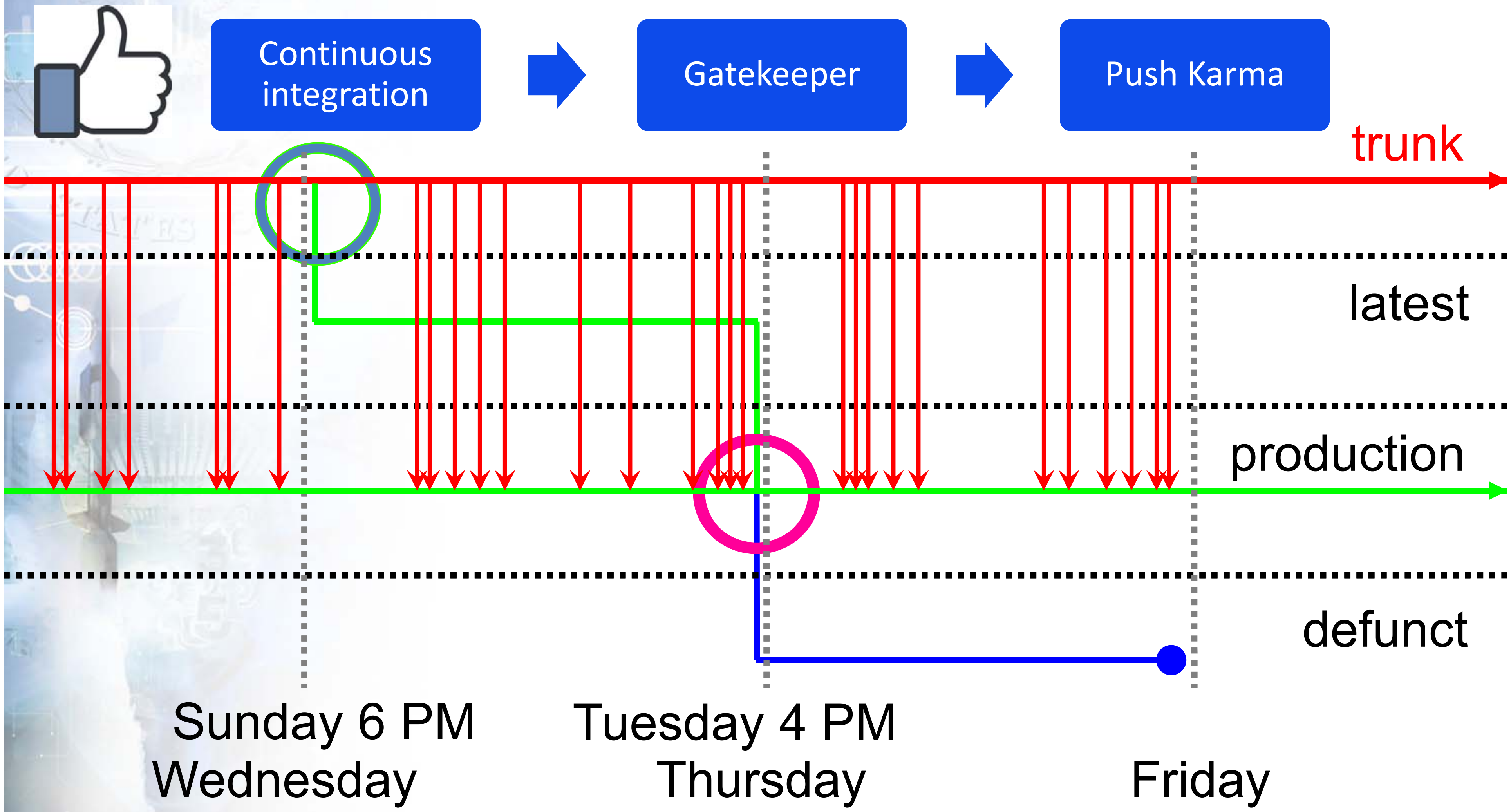
- Meta-survey of 29 studies that contained ROI data (of 300 articles analyzed).* Rico 2009
  - On average, studies of Agile Methods reported
    - 29% lower cost
    - 91% better schedule
    - 50% better quality
    - 400% better satisfaction

*Rico 2008

FINDINGS

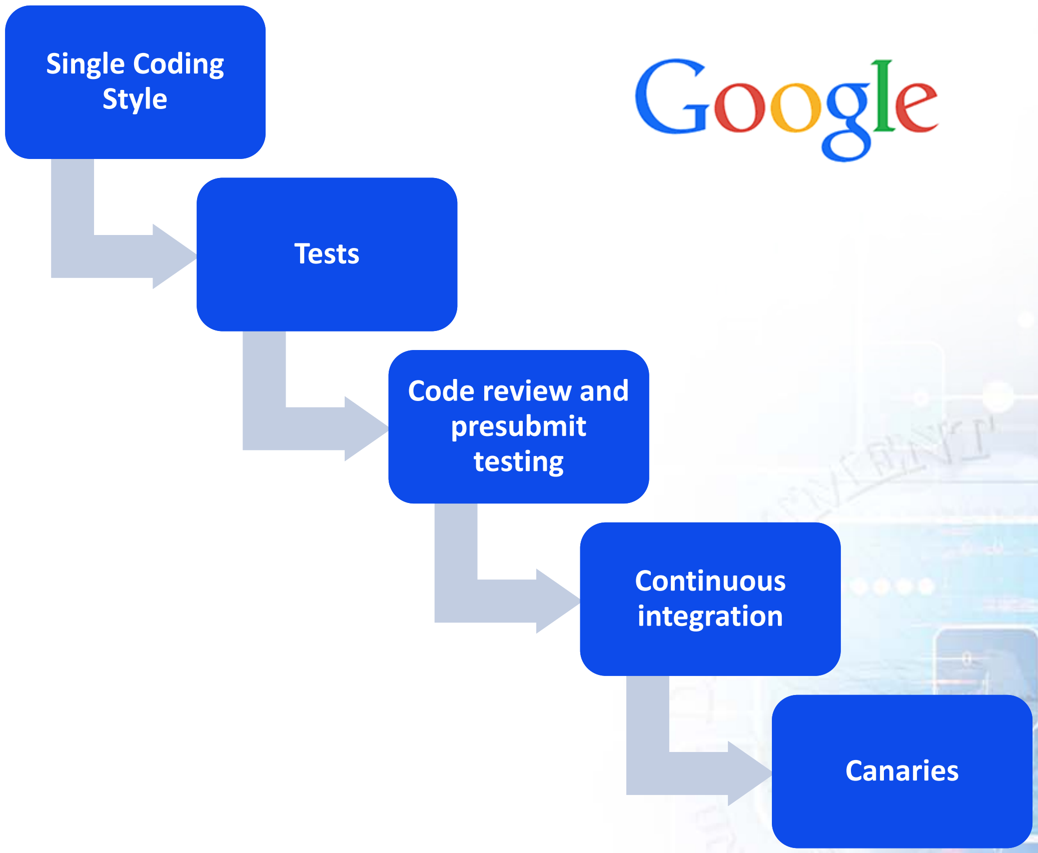
Commercial, DoD and its partners: Case Studies

Facebook Software Process Elements



Google Software Process

- More than 30K developers in 40+ offices
- 13K projects under active development
- 30K submissions per day (1 every 3 seconds)
- 20+ code changes per min with 90+ bursts
- 50% of code changes monthly
- 150M+ test cases per day
- Continuous integration for all teams



State of Play – Defense Prime Contractors*

- The majority of software developed by the major defense prime contractors follows a traditional waterfall process
- All are familiar with iterative development, some in more depth than others. Most have used it on selected small programs or portions of larger DoD programs in the past
- Some are quite eager to pursue iterative development of software as their primary methodology, understand they are "15 years behind best commercial practice," and would welcome closer and more frequent participation with users in plant. But they claim they are not able to do so because their DoD contracts are written requiring documentation, progress reviews and incentives based on a waterfall model
- Others see iterative development as something useful for web apps but not appropriate for most defense systems. They do not seem inclined to change their current approach to developing software
- And still others are already trying to adopt portions of the iterative process into their developments when it does not conflict with their contract language. And they claim to have realized cost and schedule benefits in doing so. We even saw cases of using iterative processes on large scale fixed price development programs whose requirements have been unchanged for seven plus years (an example being the KC-46A Tanker)

*Note – discussions were held with top level company executives

Iterative Development for National Security Mission: SpaceX

- Appears to be an “existence proof” that modern DevOps commercial practices can be used effectively for rapidly changing systems that are mission critical for national security – Air Force Space Launch
- Moving toward space launch every two weeks, with matching software updates for mission critical flight and ground systems
  - Only third party software they use is from Linux, everything else developed organically
  - Flight systems mission software changes 5-10% per mission
  - Space launch certification (including FFRDC independent oversight) occurs within two week window; maturity of process and scalability still uncertain
- Acquisition model: Government competes launch as a service. When SpaceX wins award, they then have freedom to develop hardware and software organically as they see fit – BUT must remain launch certified
  - SpaceX has been using iterative software development for ~seven years
  - Requirements changes come from both customer (e.g. specifics to each launch mission) and from themselves (e.g. improvements to capability)

NSA has successfully moved to Agile …with limitations

- NSA has successfully moved to an agile, iterative model for much of their software development over the past five years
  - Have built tools and in-house expertise that allows defense contractors to contribute and bring mission experience
  - But NSA essentially owns the software factory and buys software development by the hour from the contractors
- Using modern commercial tools, combined with NSA approved encryption and security measures, teams of multiple contractors at multiple locations can collaborate simultaneously
- The model has been quite successful but does have some limitations
  - Typically used for systems with stable hardware processing environments only
  - NSA defines and manages the development process. While contractors apply specific local expertise and write most of the code, the NSA tightly manages the process and metrics. This requires the customer to have highly trained and capable program managers that are experts in the Agile process
  - Since NSA manages the process and buys software development by the hour, contractors do not develop intellectual property and therefore do not have a strong business case to make big investments to advance the relevant technologies. The government customers are sometimes disappointed that industry isn't investing more in these areas
  - NSA is intimately involved in the daily development of the software by the contractor

NRO Best Practice – Database of Historic Cost Actuals for Software Development – Waterfall or Agile

- Cost estimation at the start of software intensive DoD programs is difficult; most independent cost estimates (the "ICE" – performed by the CAPE or Service Cost Estimators) use outdated SLOC-based cost models
- CAPE and Service cost estimators historic cost data appears sparse – SLOC based assumptions are then compared to historical "comparables" – with mixed results in matching program actuals
- NRO Best Practice – Established contractual relationship with all their major primes to provide internal cost data to the NRO – years of data available to inform cost estimation of new programs

FINDINGS

Acquisition Strategies and Contracting Approaches

Misalignment

- Classic Acquisition Metrics: Cost, Schedule, Performance
- Classic Phases of Acquisition: Development, Production, Sustainment – *yet modern software is in continuous development!*
  - Average ACAT I Development Program: Development Schedules for five years (Milestone B to C), initial development actually takes ~seven years, follow on capability every two years
  - Colors of money and phases are not well-aligned with how software developed today
  - Closest DoD analogy: P3I, smaller ACAT programs, life-extension, routine sustainment, SOCOM MFP-11
- GAO's annual report to Congress simply compares total cost and schedule (all $ type) per program values to previous year, BUT will also compare changes from original estimate years before:
  - Usually Washington Post front page
  - *"Over the past year, the total acquisition cost for the 79 programs in the 2015 portfolio decreased by $2.5 billion and the average schedule delay in achieving initial capability increased by 2.4 months. When assessed against first full estimates, total costs have increased by $469 billion, over 48 percent, most of which occurred over 5 years ago. The average delay in delivering initial capabilities has increased to almost 30 months." -GAO*

The defense acquisition process, how money is requested, appropriated, and measured is misaligned with modern software development.

Defense Acquisition could use Continuous Iterative Development in many types of programs

- **Ongoing major development programs – small scale (hybrid model): (e.g., KC-46A fixed price development)** It may be done at small scale as long as end-product remains unchanged (i.e., meets spec of contract)
  - Typically overall technical specifications derive from the requirements — a perfected statement of the need expressed by the beneficial user; Precise specifications are typically enshrined in the contract which is awarded at beginning of development (Milestone B) in traditional defense acquisition programs
- **Ongoing major development programs – large scale (e.g., F-35):** In the course of incremental developments, designer and/or customer may find original specs too ambitious or otherwise undesirable due to technology change/warfighter need
  - If incrementally build/test and expose users to the interim products, alternatives may suggest themselves, and thus agile opportunities emerge
  - However, to change a requirement in an ongoing program, the law requires there be a "Configuration Steering Board," a formal process called which may require highest level approval. Lengthy staffing and approval process – the opposite of Agile.
- **New Programs:** Clean slate opportunity to do iterative development from the beginning
  - An alternative acquisition approach could be compete software development as a service where source selection is "Best Value" for mission success –Consider multiple vendors and consider it a service
- **Legacy Programs (development is complete):**
  - Even legacy programs can improve (e.g., Tomahawk)

Example of Legacy Program Moving to Iterative Development: Tomahawk

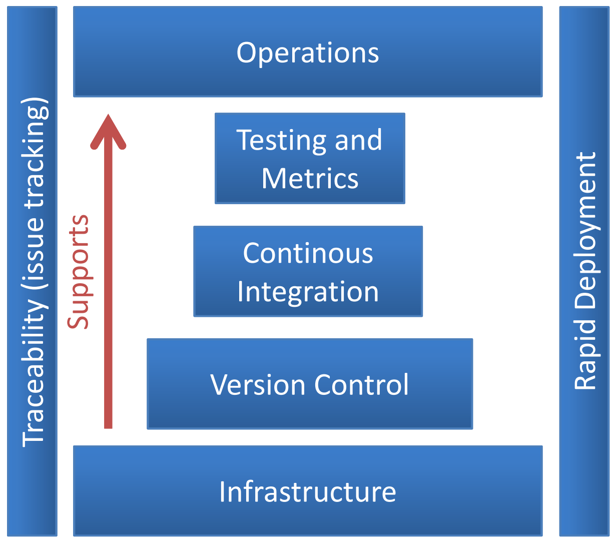
- Tomahawk currently executing a streamlined, hybrid-Agile approach with good results
  - The development approach for Tomahawk add-on is still waterfall
- Conducting two-week long sprints over a defined period of time (i.e., the waterfall spiral time) with the goal of discovering defects earlier (not shortening the time to completion)
- Benefits:
  - Shorter sprints allow for periodic deliveries for early integration and testing, as well as cyber scans
- This approach will be implemented in full in the next baseline (TTWCS v5.6.1)

RECOMMENDATIONS

Recommendation 0: Software Factory

- Key evaluation criterion in source selection process should be the efficacy of the offeror's software factory
  - The USD(R&E) should task DDS, USAF Life Cycle Management Center (LCMC), Software Engineering Institute, and NAVAIR to establish a common list of source selector criteria for evaluating the software factory for use throughout the Department (see next slide for suggested draft criteria)
  - Make demonstration of proof of software factory, judged based on offeror meeting ***at least*** a pass-fail criteria, to be minimally technically acceptable in the proposal
  - Criteria should be reviewed and updated every five years
- DoD has limited agile development expertise. Focusing this expertise during the source selection uses this limited talent in the most efficient way

Software Factory Source Selection Criteria Suggestions



- Configuration management software (e.g., Puppet, Chef, Ansible)
- Continuous Integration (build and test) Systems (e.g., Travis CI for hosted service, Jenkins for open source application)
- Scripts and code used to release software (e.g., Python scripts)
- Servers, network or other infrastructure that support release tools
- Software and tools to support developer self-service operations (NewRelic for application performance over time, diagnostic tools, monitoring)
- External test frameworks (e.g., Jersey Test Framework, Testplant/Eggplant)
- External operational monitoring and log mining tools (e.g., Splunk, Elasticsearch + Logstash + Kibana (ELK) Stack)
- Source code repositories (e.g., Github for hosted service, GitLab for open source application)
- Issue tracking systems (e.g., JIRA, Trello, GitHub)
- Container driven tools (e.g., Docker, Elastic Container Service (Amazon Web Services (AWS)), Kubernetes)
- Requirements management (e.g., Doors, Blueprint)
- Infrastructure and cloud providers (e.g., AWS, Rackspace, Azure, RedHat OpenShift, Pivotal Cloud Foundry
- IDEs integrated DevOps process

Recommendation 1: Continuous Iterative Development

The Department and its Defense Industrial Base partners need to adopt continuous iterative development best practices (continuing through sustainment) for software

- SAE, with PEO, PM, and Joint Staff/J8, should identify Minimum Viable Product (MVP) approaches and delegate acquisition authority to PM (cascade) providing motivation to do MVP and work with the users
  - Deliver a series of viable products (starting with MVP) followed by successive "Next Viable Products" (NVPs)
  - Establish MVP and the equivalent of a product manager for each program in its formal acquisition strategy – get warfighter to adopt IOC as MVP
  - Engage Congress to change statutes to transition CSB's to support rapid iterative approach (FY2009 NDAA, Section 814)
- DAE/SAE or Milestone Decision Authority (PEO or PM) should require all programs entering Milestone B to implement these iterative processes for ACAT I, II, and III programs. Goal is not to be overly prescriptive; details should be tailored to each program.
- SAEs should identify best practices and decide how to best incorporate into regular program reviews (e.g., DABs, IPRs, Service Review Boards, etc.)
  - Waivers done only by exception

Recommendation 2: Risk reduction and metrics for new programs

For all new programs, the following best practices should be implemented in formal program acquisition strategies:

- MDA (DAE/SAE/PEO/PM) should allow multiple vendors to begin work. Downselect after at least one vendor has proven they can do the work. As feasible, retain several vendors through development. Do so as a risk reduction practice.
- MDA with CAPE, USD(R&E), Service Cost Estimators, etc. should modernize cost/schedule. Evolve from pure SLOC approach to comps; adopt NRO approach of contracting with Defense Industrial Base for work breakdown schedule data (staff, cost, productivity, etc.).
- MDA should require the PM to build a program-appropriate framework for status estimation, metrics examples include*:
  - **Sprint burndown:** Tracks the completion of work throughout the sprint
  - **Epic & release burndown:** Tracks the progress of development over a larger body of work than a sprint
  - **Velocity:** The average amount of work a team completes during a sprint.
  - **Control chart:** Focus on the cycle time of individual issues–the total time from "in progress" to "done”
  - **Cumulative flow diagram:** Shows whether the flow of work across the team is consistent, visually points out shortages and bottlenecks

  ** Such metrics should also be used by the Department, GAO, and Congress. For more information on agile contracting approaches and metrics – see Digital Services techFAR*
- There may be short term costs in transitioning to iterative development (software factory, training) however our expectation is over the longer term commercial practice has demonstrated that net costs are reduced.

Recommendation 3: Current and legacy programs in development, production, and sustainment

For ongoing development programs:

- USD(AT&L) should task PMs, with their PEOs, for current programs to plan transition to a software factory and continuous iterative development.
  - Defense Prime contractors transition execution to a hybrid model, within the constraints of their current contract.
  - Defense Prime contractors incorporate continuous iterative development into long term sustainment plan
- USD(AT&L) should task SAEs to provide quarterly status update to USD(AT&L) on transition plan for programs, per ACAT category

For legacy Programs (development is complete):

- USD(AT&L) should task PMs, with their PEOs, to do business case for whether to transition program

Sharing Best Practices:

- USD(AT&L) should task PMs of programs that have transitioned successfully to brief lessons learned across the Services

Recommendation 4: Workforce (part 1/2)

The government does not have modern software development expertise in its program offices and broader functional acquisition workforce – this requires Congressional engagement and significant investment immediately

- Service acquisition commands (USAF LCMC, NAVAIR, NAVSEA, Army Materiel Command) need to develop competency — acquire/access a small cadre of software system architects with deep understanding of iterative development
  - Use this cadre <u>early</u> in acquisition process in formulating acquisition strategy, developing source selection criteria, and in evaluation
  - Goal is to ensure software development expertise is established as core to the program and to ensure mission is done in smaller pieces with functionality at each step

Recommendation 4: Workforce (part 2/2)

- Beyond development of coders and developers, there is a need for software-informed PMs, sustainers and software acquisition specialists
  - Service Acquisition Career Managers should develop a training curriculum to create/train this cadre
  - SAE and PEOs should ensure program managers of software-intensive programs are knowledgeable about software and with software acquisition training
  - USD(AT&L)/ASD(R&E) direct DAU to establish curricula addressing modern software practices – leverage expertise from FFRDC community (e.g., CMU SEI)
- Defense Primes must build internal competencies in modern software methodologies
  - CEOs should brief USD(AT&L) quarterly to demonstrate progress
- Working with the Congress, career functional Integrated Product Team (IPT) lead immediately establish a special software acquisition workforce fund modeled after DAWDF whose purpose is to hire and train a cadre of software acquisition experts across the Services; Objective is 500+/year starting in FY18
- PMs create Agile IPT with associated training; Service Chiefs delegate role of Product Manager to these IPTs

Recommendation 5: Software is Immortal: Software Sustainment (part 1/2)

- RFPs should specify the basic elements of the software framework supporting the software factory including code and document repositories, test infrastructure (e.g., gtest), software tools (e.g., fuzz testing, performance test harnesses), check-in notes, code provenance, and reference and working documents informing development, test and deployment.
- Availability, cost, compatibility and licensing restrictions of such framework elements to the government and its contractors will be a selection criteria for contract award.
- At RFP, proposers may designate pre-existing components not developed under the proposal but used or delivered as part of the project; however, limitations related to use or access to underlying design information (including components designed using the “software factory” approach) may also be a selection criteria.

Recommendation 5: Software is Immortal: Software Sustainment (part 2/2)

- Except for such pre-existing components, all documentation, test files, coding, API, design documents, results of fault, performance tests conducted using the framework, tools developed during the development as well as the software factory framework shall be (pick one)
  - 1. delivered to the government at each production milestone
  - 2. escrowed and delivered at such times specified by the government (e.g., end of production, contract reward).
- Selection preference shall be granted based on the ability of the government to reconstitute the software framework and rebuild binaries, rerun test, procedures and tools against delivered software and documentation.
- These requirements shall flow down to subcontractors and suppliers subject to reasonable restrictions affecting use, duplication and disclosure of material not originally created as part of the development agreement.

Recommendation 6: IVV for Machine Learning

- Machine learning is an increasingly important component of a broad range of defense systems (including autonomous systems) and will further complicate the challenges of software acquisition. With machine learning, code may write itself.
- The Department must focus and invest to build a better posture in this critical technology.
  - DARPA and the DoD Labs should establish research and experimentation programs around the practical use of machine learning in defense systems with **Independent Verification & Validation (IVV) and cybersecurity** being the primary focus
    - establish a machine learning/autonomy data repository and exchange along the lines of the CERT to collect and share necessary data from and for the deployment of machine learning/autonomy
    - create and promulgate a methodology and best practices for the construction, validation & deployment of machine learning systems including architectures and test harnesses.

BACKUP

Software Factories — the Next Step

- Maintaining a well-equipped software factory demonstrably benefits the efficient and effective design and production of software.
  - DoD should ensure that prospective offerors understand and take advantage of those benefits, emphasized by contractor-contract selection criteria
- Much of the benefit derives from a well-stocked software repository.
  - Developers normally have access to their local, often proprietary code base, as well as open-source software
  - DoD has available to it a considerable accrual of source repositories (and could negotiate more aggressively for new acquisitions)
  - DoD, then, could provision a "global" repository and provide controlled access to that source-base to its contractors on a per-contract basis

Requirements Preparation of the Agile Development Battlefield

- Agile software development methodology may work better in cases where we are unsure of exactly what we want/need
- This rigid certainty describes most DoD acquisition requirements as espoused (by to the JROC, e.g.) …
  - But, in actuality, not all requirements are so precisely known, and
  - Nearly all requirements bend to the wind of cost, schedule and technical reality
- Modern, desirable agile development methods would be better served by some latitude in the original requirement. For example:
  - A "trade-space" or "performance envelope" that describes several requirements that are in tension with one another
  - The respective/comparative "value propositions" of individual requirements more granular than just indicating "KPP" (a one-bit descriptor of value and unnecessarily rigid)

Modern Software Development in New Programs

Key Event – RFP Release to Begin Development

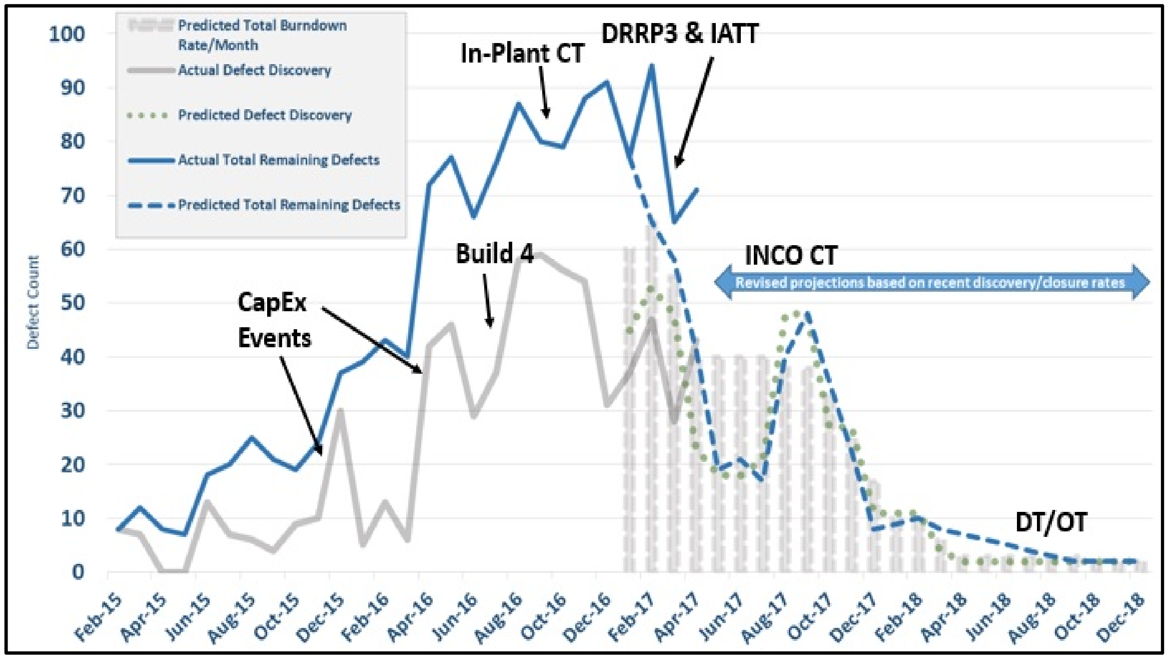
- RFP Development and Source Selection Criteria:
  - Incorporate demonstrated modern software factory maturity as part of minimum technical acceptability threshold - “ready for development evaluation”; Becomes pass/fail of proposal
  - Demonstrated software development acumen during risk reduction phase prior to Milestone B
  - Consider software development contracted as a service!
- Require initial operating capability (IOC) date and criteria to align with agreed upon Minimally Viable Product (MVP) (however, note F-35B experience)
- Align Configuration Steering Board tempo with each software development sprint – handle at lowest organization level possible between requirements and acquisition communities

Examples of Ongoing Development Iterative development: Space Fence Software Quality

	Phase Defects Injected/Originated					
	Reqts	Design	CUT	DIT/ SWIT	System Intgn	System FQT
Defects Found	57	343	1730	22	44	0
% Found In-Phase	32%	83%	86%	82%	100%	
Cumulative Defects Found In-Phase				84%		

Defects Contained

Efficient/Manageable Defect Burndown



Advantages of Iterative Development*

1. Because of frequent interaction with the user, the developers are more likely to create what the user really needs
2. Because each iteration is a short fixed duration, the cost of an iteration is predictable – the acquirer can then more readily understand the approximate cost of each feature
3. Iterative development allows progress to be tracked in terms of actual user/system functionality completed, instead of measuring technical artifacts such as lines of code produced, or adherence to a predefined plan, or delivery of documentation
4. It eliminates chronically false assumptions which can endanger a program
   - That we can accurately predict how long it will take to develop the software
   - That we know all the user's requirements up front
   - That those requirements will not change during development
5. With iterative development there is very little chance of getting to the end of the program and finding out in test that the system does not function as intended, requiring significant rework or cancellation of the program

*Adapted from "The Business Value of Agile Development," Microsoft

Agile Expectations vs. Experiences

- Technical literature contains anecdotal reports about specific "agile" approaches but offers few rigorous empirical studies.
  - No empirical study is widely cited or considered authoritative.
  - No studies found for weapons systems (e.g., real-time control, high-end security threat)
- What could be basis for comparison by empirical study:
  - **Direct**: Quality of system (bugs or vulnerabilities), size of system (LOC), productivity (LOC/hr), development effort (elapsed time, labor hours)
  - **Indirect**: Number of companies transitioning to agile / continuous iterative development. (e.g., North American and European Enterprise Software Services Survey, Business Technographics Ed., 2005: "*14% of companies are using agile methods and 49% are aware and interested in adopting them*")
- Two meta-studies do survey literature that gives empirical comparisons.
  - Tore Dyba and Torgeir Dingsoyr. Empirical studies of agile software development: A systematic review. Information and Software Technology 2008.
  - David F Rico. What is the ROI of Agile vs Traditional Methods. https://davidfrico.com/rico08g.pdf

Published Empirical Comparisons (1)

- Meta-survey of 36 empirical studies prior to 2005 [Dyba and Dingsoyr].
  - Four studies give empirical data for productivity comparison of agile and traditional ("waterfall") development. Most focus on XP ("extreme programming") form of agile.

Study	Traditional Prod	Agile Prod	Productivity Gain
S7	3 LOC/hr	13.1 LOC/hr	337%
S10	3.8 LOC/hr	5.4 LOC/hr	42%
S14	300 LOC/month	440 LOC/month	46%
S32	157 LOC/engr	88 LOC/engr	-44%

  - S7 involved 15 teams used 4 different approaches. Greatest difference shown.
    - Note: Agile team delivered far more code, but the same functionality as traditional.
  - S14 agile team had more experience with languages and management
  - S32 is a study concerning student programmers.