REQUIREMENTS

Make room for opportunities in program requirements

Problem: Requirements are fixed ahead of market research and experimentation, leading to overly complex contracts, neglect of new tech insertion, and fielding of obsolescent systems

Recommendation: Focus on statements of outcomes to increase flexibility in solution design and allow for requirements iteration over time

Separate the Big "R" from the Little "r":

- Separate strategic program-level requirements from tactical requirements in the product backlog
- Connect user stories in each iteration to higher-level capability statements
- Pick top capabilities for next increment and share with stakeholders

It's All About the Art of Writing Requirements:

- List statements of outcomes limited to one sentence descriptions
- Avoid prescriptive language
- Keep few non-tradeable requirements, validated with users
- Specific metrics should start modest and converge on objectives

Stakeholder Communication:

- Formalize a series of cross functional teams at multiple levels
- Refresh existing requirements into outcome-focused mission threads
- Keep an open channel with stakeholders (requirements officers, testers, users, joint interests, etc.) to iterate when advantageous
- Use collaborative tools to decentralize coordination
- Identify who is the (one) champion and who are the (few) veto points
- Avoid requirements by consensus (gold-plating)—if you get a lot of pushback you're doing it right!
- Bring industry into the process early

Success: Requirements stay aligned with technical progress and user feedback by keeping an open channel with stakeholders

Resources: DAU Adaptive Acquisition Framework website; MITRE's Accelerate Requirements

Context & Motivation

The need for adopting commercial technologies to national security use cases has never been stronger. Back in 1960, DoD funded one-third of global R&D. Today, the figure is just 3 percent. This is reflected in the fact that eight of the DoD's top ten modernization priorities are dominated by commercial technologies including cyber, quantum, space, biotech, microelectronics, networking, AI/ML, and autonomy. DoD continues to lead in hypersonics and directed energy, but more than ever government must adapt or fall behind.

Programs can struggle adapting to change. For example, F-35 pilots regularly flew with tablets in their lap forcing the Air Force to start testing integration in 2021. The problem is pervasive. In 2018, the GAO reported that the average age of major defense acquisition programs was over 14 years old. Most of these programs got started before Amazon offered cloud services or Apple introduced the smartphone.

The commercial sector is now releasing new technologies inside the "OODA loop" of govern-

ment decision-making. Acquisition officials must have the tools to take advantage of new opportunities created by the larger technology ecosystem and experiment with them to form new operational concepts within a program.

A major barrier to technology insertion is the linear requirements-pull model. In the model, intelligence defines enemy threats, operators identify capability gaps, programmers resource it, and the acquisition community at the end of this chain executes the baseline plan. It's not surprising that many important military innovations do not owe their origins to requirements, including jet engines, nuclear propulsion, ballistic missiles, and more recently the internet, GPS, night vision, lasers, stealth, and UAVs.

The answer, however, lies not in a technology-push model. An interplay between the R&D and user communities has long been known to have the most advantages.

The first play seeks to focus Big "R" requirements for the program on statements of outcomes. Certainly there is nothing novel here, but past efforts have failed for a couple reasons. First, there was little definition for how specific features were to be prioritized by the product owner in execution. Second, there wasn't a collaborative process for iterating on requirements when needed. Users, testers, and requirements officers should be included in deciding which features make it to the top of the queue for the next increment of work.

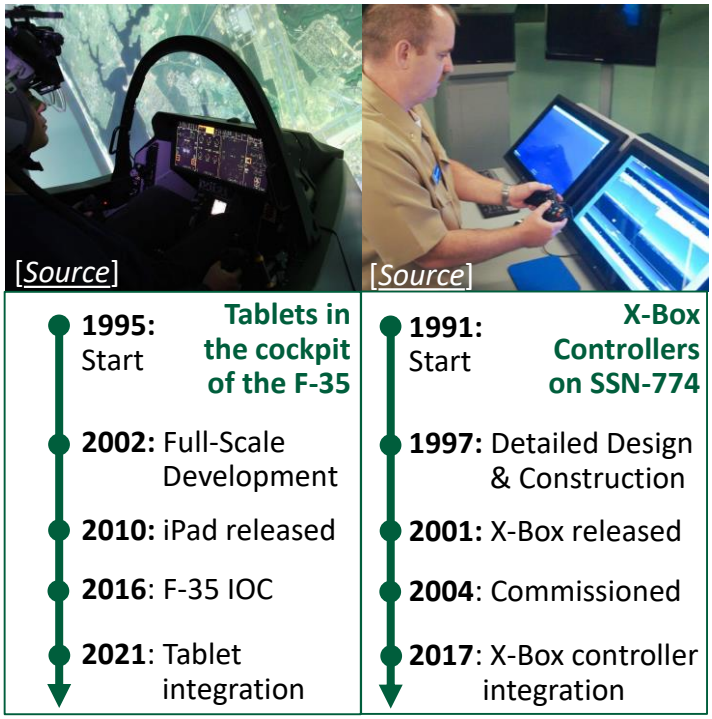


Figure 3. Examples of Lagged Integration of Commercial Technology

Formalize two-tiered requirements sets. Kessel Run has a requirements model that distinguishes strategic level requirements and user centered design. The strategic Big "R" requirements are found in a Capabilities Development Document, Capability Needs Statement, or similar. These high-level outcomes decompose into tactical Little "r" requirements found on a product backlog that get iterated on. This basic construct works not just for DoD Software or Middle Tier pathways, but all federal acquisition programs. Requirements for existing programs can be refactored with stakeholders. Figure 4 on the next page illustrates a process for iterating on Big "R" and Little "r" requirements.

Writing good requirements. The Army IVAS program for augmented reality headsets had requirements for shock-proof, waterproof, and ruggedness, but did not consider the need for soldiers to brace a rifle against their cheek, or army crawl on rough terrain. Luckily, through iterative development, they caught the issues and closed them. Of course, the Army could have predicted all of these features and made them into a long specification list. The preferred alternative is to write general statements of outcomes like "Does not impede the soldier's regular combat actions."

Prioritization is a team effort. Survivability, reliability, and lethality are all great, but if you maximize all of them then you will definitely lose mobility and range. Maybe you won't have enough money to then go after networking. Even with just a few non-tradeable requirements, the solution space may still be described by "unobtainium." A partial set of requirements may be better than asking for everything. Alternatives should be discussed with users to reach agreement on what comes first in terms of tactical-level features to be developed in the next increment.

Use collaborative tools. The translation between Big "R" and Little "r" requirements cannot be confined to the acquisition community. It must involve users throughout development rather than going up and down hierarchical chains. Directly communicate with counterparts in the combatant commands, materiel commands, requirements offices, and other organizations. Clearly define a regular cadence of interactions. Provide access to shared folders of important information and status. Use enterprise tools like Office 365 or Slack to communicate horizontally with stakeholders. There's no replacement, however, for side-by-side testing with users.

Create a map of the stakeholders. This collaborative program process requires formalizing the stakeholders and their role in the process. Major program decisions may involve numerous functional and leadership positions. It may seem each one of them is a potential veto point and thus the program must accommodate all of their additional requirements. These demands must be resisted if program officials and users alike do not prioritize them. Document these choices and be prepared to defend them to the decision authority that actually represents a veto point.

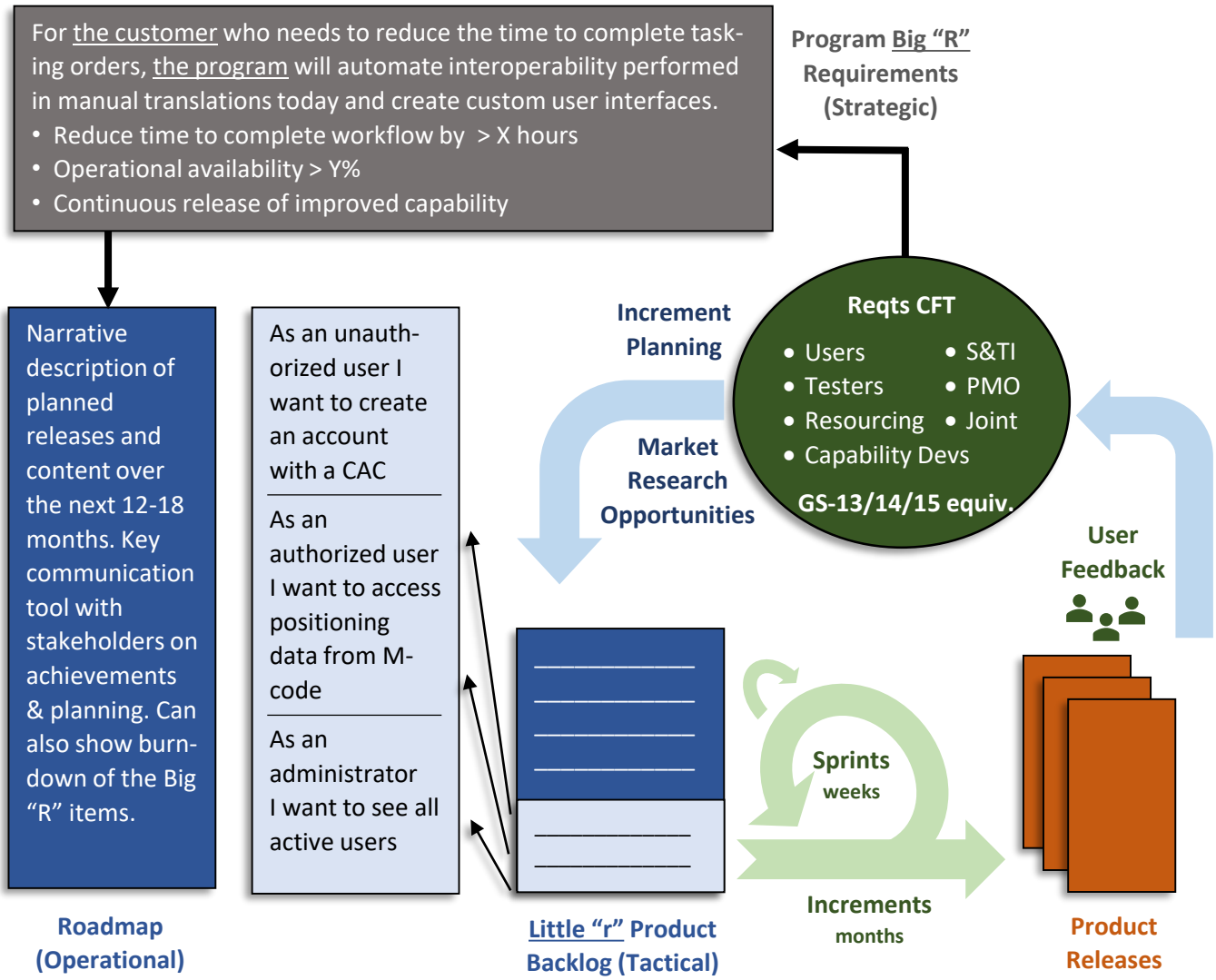


Figure 4. Illustrative Process for Iterating on Big "R" and Little "r" Requirements

Capability Over Time Curve. For traditional acquisitions, capabilities are not ready to be tested and released until they meet a laundry list of requirements, even if the last few percent of capability take half the time and cost of the entire program. Key performance parameters should not be considered static program objectives to be met all at once or not at all. A better path is to incrementally release capabilities and track progress on an evolving set of metrics. Program officials should chart a similar course with the test community by tracking capability over time curves. For example, an autonomous system might track improvements in way-point navigation or the accuracy of targets detected by computer vision. An IT system might track user adoption rates and workflow completion times. Metrics should be tailored to the program's operational context and expected to drift over time. Avoid over-concentrating on particular metrics at the expense of the program's intangibles. Goodhart's Law states that "When a metric becomes a target, it ceases to be a good measure." That's because organizations can sometimes optimize an objective regardless of the consequences.

> **"It's a sin when we refuse to break things into increments because we take options away from the commander."**
>
> Requirements Officer

Wrap Up. The acquisition process is devised with linearity in mind. Fully formed requirements are expected to pull technology along in predictable ways. When uncertainty rears its head, observers often blame requirements creep or gold-plating. In too many acquisition programs, success is measured by the minimization of disruption—by execution to plan. Yet the desire to prematurely freeze requirements before technical trades can be made incentivizes program officials to ignore new information. The move to iterative requirements reflects a process of learning and error correction. As famed philosopher of science Karl Popper observed, "Error correction is the most important method in technology and learning in general." The trial-and-error method is scientific precisely because it creates a reflexive interaction between requirements-pull and technology-push approaches, which corresponds to higher-level concepts of deduction and induction. Fortunately, even traditional acquisition programs are open to iteration. This can be accomplished by using outcome-oriented statements, formalizing a two-tiered set of requirements, creating a repeatable process of stakeholder communication, and tracking progress to capability over time curves.