RISE8

SOFTWARE FACTORY 2.0

A New Operating Model for DoW Mission Software

AUTHORED BY:

Bryon Kroger, Founder & CEO
Carlo Viray, Director of Growth
Rise8

RISE8, INC.| 601 S HARBOUR ISLAND BLVD., STE 109 | TAMPA, FL 33602

Executive Summary

Software is reshaping both markets and warfare. For the U.S. military, the cost of disruption is measured in lives and national security. The Department of War (DoW) must become the disruptor, not the disrupted. But the current model isn't working: 94% of federal IT projects are behind schedule or over budget, and 40% never deliver anything at all.

Software is now our warfighting edge. We must fundamentally change how we deliver it.

Software Factory 2.0 (SWF 2.0) redefines how DoW delivers mission software by treating it as a warfighting capability, not a cost center.

Software Factory defined: A software factory is the combination of people, process, and technology that creates the conditions for continuously delivering valuable software that end users love. It draws from Lean manufacturing, modern DevOps practices, and systems thinking to maximize flow and minimize waste.

What went wrong with SWF 1.0: Despite early successes at Kessel Run and Kobayashi Maru's Section 31, Software Factory 1.0 failed because DoW systematically undermined it through mission dilution, leadership turnover, burdensome technology mandates, DIY platform distractions, and bureaucratic reversion to traditional acquisitions, hierarchy, and decision-making.

What SWF 2.0 aims to fix: SWF 2.0 isn't agile reform, it's institutional re-architecture. By institutionalizing founder-mode continuity, focusing on mission value streams, and enforcing "prod or it didn't happen" accountability to mitigate operational risks, SWF 2.0 prevents the organizational decay that plagued its predecessor.

The three core imperatives: SWF 2.0 unifies requirements, resourcing, and acquisition into a single outcome-oriented agile model aligned with policy, including recent updates. This new model breaks down the traditional silos and optimizes for learning fastest, not predicting perfectly.

The systemic failures of SWF 1.0 must serve as institutional lessons learned. SWF 2.0 offers a blueprint for building the digital force of the future by aligning every enterprise function around mission value streams and enforcing outcome-based accountability to continuously deliver impactful software warfighters love.

SWF 2.0 enables our forces to be ready to **fight tonight**.

Post-Mortem: What Went Wrong with SWF 1.0

SWF 1.0 proved that DoW can deliver secure, high-impact software at mission speed when insulated from bureaucracy. Both Kessel Run and Section 31 achieved mission impact on timelines previously considered impossible.

Case Study Highlights

U.S. Air Force Kessel Run (2017-2019 peak)

Kessel Run was the first DoW software factory, focused on Air Operations Center mission software.

- Invented cATO and became the first DoW organization to achieve Continuous Delivery for warfighting operations on SIPRNet
- Deployed 5 applications from concept to warfighting operations in an average of 124 days
- Reduced target development timelines by 85% (38 mins to 6 mins)

U.S. Space Force Section 31 (2019-2021 peak)

Section 31 was USSF's first software factory, built to deliver capabilities to the Combined Space Operations Center.

- Deployed 8 applications to operations in an average of 64 days
- Reduced conjunction analysis from 3 hours to 15 minutes

Institutional Causes of Decline

But early wins didn't survive institutional gravity. The same repeatable, avoidable patterns undermined both software factories:

- **Mission dilution** – Teams were pulled into unrelated missions (e.g., F-35 Autonomic Logistics Information System (ALIS), Pentagon-mandated technology migrations, and providing enterprise services)
- **Leadership turnover** – Entire executive teams rotated every 2-3 years
- **Team turnover** – Embedded government personnel rotated every 1-2 years
- **DIY platform distractions** – Wasted 5+ years and $100M+ building inferior alternatives to working Commercial Off-The-Shelf (COTS)
- **Acquisitions reversion** – Transitioned from outcomes-based to traditional staff augmentation contracts
- **Political interference** – Decisions optimized for budget survival over mission need
- **Poor fit contracts** – Using Acquisition and Assistance Services (A&AS) contractors who lacked modern software delivery expertise

Lessons Learned

Every failure points to a corresponding lesson for SWF 2.0:

- **One mission per organization** – High-performing teams fail when split across missions
- **Embrace founder-mode continuity** – Keep founding teams for 8-10 years to retain mission context
- **Limit & focus organic development** – Until the service fully invests in organic developers, utilize contractors on outcomes-oriented contracts, and focus organic development on innovation at the edge, ideally utilizing opinionated COTS platforms
- **Rent > Buy > Build** – Use COTS platforms; focus on mission capabilities
- **Optimize the full value stream** – Delivery, acquisition, and operations must function as one system
- **Mission-first decision-making** – Measure mission performance, not IT metrics
- **Prioritize mission results** – Track mission effectiveness metrics (lagging indicator) and milestones for progress tracking via outcomes-oriented roadmaps (leading indicators)
- **Minimize operational risk** – "Prod or it didn't happen" accountability early and often
- **Restructure vendor selection** – Purpose-build outcomes-based contracts and avoid staff augmentation or commoditized labor

The failure of SWF 1.0 was institutional. Future success depends on redesigning the system.

Introducing Software Factory 2.0

Software Factory 2.0 is an outcomes-driven operating model built to continuously deliver software that drives mission impact. Its core principle is to optimize for learning fastest, not predicting perfectly. In a rapidly evolving threat environment, the ability to sense and respond through continuous software delivery matters more than any static suite of pre-approved products.

The intent is not agile reform, but rather institutional rearchitecture around learning, decentralization, and measurable mission impact. In an enterprise historically driven by outputs, e.g., requirements documents, budget obligations, and contract awards, the metrics must change. For SWF 2.0, success is measured by:

- **Continuous Delivery:** The ability to get changes of all types into production safely, quickly, and sustainably. Measured by DevOps Research and Assessments (DORA): Deployment Frequency, Change Lead Time, Change Fail Percentage, Failed Deployment Recovery Time, and Reliability
- **Valuable Software:** Measured by mission impact (in production) via mission metrics that matter
- **User Experience:** Measured by user outcomes (in production) and Net Promoter Score

If it's not in production, it didn't happen, and if it's not improving the mission, it's not working. The only way to buy down risk is in production: the environment where software is put into operation for its intended use by intended users. Anything else is risk.

But achieving success by these metrics requires changes across the systems that control what gets built, how it's funded, and how it's delivered. SWF 2.0 addresses this through three core imperatives: requirements, resourcing, and acquisition.

The Three Imperatives

To scale software delivery across the DoW, Software Factory 2.0 requires changes not only in how software is built, but how the enterprise scopes, funds, and acquires it. Senior leaders have three levers to pull to enable continuous delivery and measurable mission impact.

A New Requirements Imperative

The current DoW requirements process rewards prediction over adaptation. Multi-year forecasts lock teams into premature solutions and make course correction costly and slow.

SWF 2.0 reframes requirements as a continuous process of identifying mission constraints and removing them.

- Replace multi-year forecasted requirements with real-time discovery of operational friction points
- Use Value Stream Mapping (VSM) and Domain Driven Design (DDD) to define mission-aligned problem spaces, e.g., bounded contexts, where teams can operate independently, and perform user journey mapping to deeply understand the human-machine-process interactions. Requirements for outcomes-based contracts must be derived from this to be effective.
- Align software delivery teams to mission value streams, not systems or platforms, and empower them to deliver autonomously within a clearly defined mission domain, aimed at a mission impact metric
- Build the organization around mission command for software; delegate authority through clear intent and constraints, not detailed instructions

Instead of asking teams to predict future needs, this model asks them to find what's slowing the mission down and fix it. The requirement is not a list of features, but measurable outcomes in production—changes in human behavior that produce mission results.

In practice: Instead of forecasting a five-year 'targeting system' requirement, identify the bottleneck ('target approval takes 3 hours'), set a target ('reduce target approval to 30 minutes'), and empower a team to solve it.

At Kessel Run, this approach reduced target development time from 38 minutes to 6 minutes in under six months.

A New Resourcing Imperative

Rigid color-of-money rules and multi-year budgeting cycles constrain teams more than code. Software doesn't fail from a lack of funding; it fails because of how funding is allocated.

Software Factory 2.0 replaces project-based obligations with capacity-based funding and outcome-driven reallocation, funding teams based on results, not predictions.

- Fund durable team capacity, not individual requirements
- Adopt the Three Horizons model to balance investment across certainty and discovery:
  - 75% Horizon 1: Mature capabilities already delivering in production
  - 15% Horizon 2: Promising prototypes ready to scale
  - 10% Horizon 3: High-upside, disruptive growth options
- Use the venture capital model of quarterly growth boards to reallocate funding based on what's working and shut down what isn't

This approach lowers risk by preventing overcommitting to ideas that haven't proven value, and creates space for the next breakthrough.

In practice: Replace large, multi-year obligations with small, high-leverage teams funded through capacity-based models. Instead of obligating $50M for a five-year 'space-domain awareness capability,' fund three, 8-person teams at $7M/year. Use quarterly reviews tied to mission impact to determine whether to expand, sustain, or reallocate funding.

A New Acquisitions Imperative

Most software acquisitions still follow a "water-scrum-fall" model: agile in the middle, bureaucracy at the ends. Traditional contract structures—monolithic, rigid, and compliance-oriented—are fundamentally mismatched to software delivery.

Software Factory 2.0 calls for outcome-oriented acquisitions that align delivery to mission capability and enable rapid, flexible response to the changing operational environments of our users. This requires:

- Replacing traditional acquisitions with contracts that:
  - clearly define and incrementally measure gains in mission effectiveness for key mission metrics
  - enable flexible execution, e.g. contract options, within a defined mission area
- Using tools like Other Transaction Authorities (OTAs), FAR Parts 12/13, and SBIR Phase III to support rapid procurement
- Structuring contracts around the Mission Value Streams and their DDD-defined bounded contexts, not organizational charts, and derive requirements and deliverables from them (i.e. the deliverable is the evidence of improvement to the primary mission metric).
- Favoring delivery teams, not staff augmentation, to build ownership, continuity, and product maturity
- Favoring building and delivering on COTS to the maximum extent practical so delivery teams can focus on mission value instead of undifferentiated heavy lifting

The goal is to scale impact and increase mission effectiveness through adaptive, incremental, capability-focused delivery.

In practice: Establish contracts with options tied to advancing mission metrics (outcomes) across adjacent mission areas. As teams demonstrate performance against defined mission metrics in the first few areas, award follow-on options to expand scope. Structure teams to operate autonomously within defined boundaries, integrating via documented APIs, not contractor coordination. Replace components that fail independently, without disrupting the system.

Mission-Oriented Delivery Framework

SWF 2.0 integrates mission operations, user-centered design, and software delivery through decentralized, autonomous teams aligned to mission value streams with defined impact metrics. The result is a ”prod or it didn’t happen” approach, where all work is measured by what reaches operations and delivers mission impact.

This framework decentralizes control across three layers: decision-making, organizational structure, and technical architecture. Leaders push decisions to the edge, empower small teams to own outcomes, and replace centralized legacy systems with distributed services. Using DDD, teams operate autonomously within bounded contexts, enabling flexible integration and minimizing coordination overhead.

Core principle: Small, empowered, cross-functional teams own their products end-to-end, from user research to production operations. Government leads the mission value stream strategy while contractors lead planning and delivery against defined metrics. Together, they operate as one team aligned to the same outcomes-oriented roadmap with traceability down to team-level execution backlogs.

Two team types form the delivery backbone:

- Stream-aligned teams build mission capabilities within bounded contexts
- Platform teams provide the infrastructure and services that enable continuous delivery

Two core organizational units lead mission-oriented delivery:

- Value stream managers, aligned to operational missions, identify mission constraints via value stream mapping and define problems worth solving (this could be a CDD mission).
- Software Delivery Organizations (SWF 2.0) rapidly build and deploy solutions to address the constraints. Though acquired by traditional means, SDOs should be OPCON to mission commanders and their value stream managers, ensuring direct alignment to operational need.

This structure accelerates delivery by collapsing the divide between operations, acquisition, and software development, enabling faster learning, tighter feedback loops, and sustained agility at scale.

Reclaiming the Software Factory

Software is now a warfighting imperative. Software Factory 1.0 proved that continuous delivery of valuable, secure software that users love is possible inside the DoW. Its failures were not of vision, but of execution, undermined by institutional inertia, siloed systems, and misaligned incentives.

Software Factory 2.0 provides the blueprint to get it right. It replaces process reform with enterprise rearchitecture focused on three imperatives:

- Reframe requirements around bounded constraints and mission-critical outcomes
- Align resourcing to team capacity and demonstrated results
- Modernize acquisitions to prioritize continuous delivery and improved mission effectiveness

When paired with the right organizational mechanisms, this model works. Mission-oriented teams, outcomes-based accountability, and decentralized control enable faster learning and delivery across the entire value stream.

The blueprint is clear. The time is now.
Think big. Start small. Scale fast.

Contact:
For questions or to learn more about how Rise8 enables continuous delivery of mission-impacting software, contact **growth@rise8.us**