

Recovering Sustainment Control of Legacy Embedded Software

Preserving application value while reducing platform and supplier dependence

Paul Kruchko | Dynamis Consulting & Controls, LLC

September 2026 | Version 0.2

A program can possess useful, fielded application software and still lack the practical ability to maintain it. The missing capability may be a reproducible build, a supported operating environment, a board support package, diagnostic data, development tools, or the rights needed to use and share those items with a receiving team. An application source repository alone does not establish independent sustainment.

The engineering opportunity is to preserve the application behavior worth retaining while addressing the dependencies that prevent affordable maintenance and change. For Army and other defense programs, this can support government sustainment, competitive support arrangements, or a more balanced relationship with the incumbent supplier. It requires a program-specific technical and economic assessment. Replacing proprietary infrastructure is valuable when the resulting capability justifies the transition cost and risk.

This paper describes an approach to that assessment and transition. Its central recommendation is to establish a maintainable baseline in stages, with evidence that the receiving team can build, diagnose, modify, test, and deploy it. Existing qualification evidence should be retained where it remains applicable, with the effects of each change evaluated explicitly.

The cost extends beyond initial development

Software sustainment includes correcting defects, adapting to changing environments, improving capability, and maintaining the product over time. In a 2019 discussion, Carnegie Mellon's Software Engineering Institute summarized published maintenance and sustainment cost estimates ranging from 40 to 80 percent of total software lifecycle cost, with a median near 70 percent. [1] That historical range provides context for taking sustainment seriously. It is not a validated cost allocation for every Army program, nor an estimate of savings available from a migration.

Accurate program data matters. GAO's 2019 review of weapon-system software sustainment identified gaps in the collection and reporting of those costs. [2] A transition business case should therefore begin with the program's actual support work, expenditures, and constraints. Generic percentages cannot determine whether rebuilding a platform, acquiring additional rights, or continuing the current arrangement is the least costly option.

Dependence becomes expensive when routine work requires access the customer cannot readily obtain. A defect may require an unavailable tool version. A software change may be blocked by an incomplete build environment. Diagnostic records may be intelligible only through one supplier's tooling. A processor

replacement may force changes across otherwise useful application code because hardware details have spread throughout it. These are different problems, with different remedies.

Identify the dependency before choosing the remedy

The first task is to establish what the program has, what it may use, and what it can demonstrate. Inventory the application, operating software, drivers, startup code, linker configuration, programmable-logic interfaces, diagnostic definitions, calibration data, build tools, and test assets. Record versions and their relationship to the fielded configuration. Identify missing items without assuming that every gap requires new development.

Technical access and permission are separate requirements. A readable file may still have restrictions on modification or disclosure to a replacement provider. Conversely, permission to use an interface is of limited practical value if its specification is incomplete. The program's acquisition and legal personnel should establish the applicable rights and license conditions alongside the engineering inventory. This is a review of the actual agreements and deliverables, not an assumption that possession of application code confers rights to every supporting component.

GAO's 2025 review of sustainment data rights found that selected programs experienced dependence on a single supplier because of rights shortfalls. Officials reported possible cost increases and longer repair timeframes. [3] The report supports treating rights and usable technical data as part of sustainment planning.

The following comparison helps keep the response proportionate to the constraint.

Observed constraint	Candidate response	Evidence needed before committing
Build environment incomplete	Recover artifacts and document a controlled build	Receiving team reproduces the defined build or explains approved differences
Needed data or rights unavailable	Obtain deliverables or negotiate suitable access and permissions	Program confirms the required use and disclosure are authorized
Diagnostic tools or formats create dependence	Document interfaces and data; introduce compatible alternatives where feasible	Authorized alternative tool interprets representative data and exercises agreed services correctly
Unsupported platform software blocks maintenance	Evaluate support, adaptation, or targeted replacement	Representative application behavior, timing, and fault response meet acceptance criteria
Hardware dependence is spread through the application	Isolate selected dependencies where the benefit supports the change	Change scope and regression evidence show that intended behavior is retained

A phased roadmap for sustainment recovery

The roadmap below generalizes the progression used in Dynamis's sustainment proposals for the M109A7 Paladin's Vehicle Control and Distribution Module (VCDM), developed with PM SPHS and GVSC-SEC: establish readiness, recover an executable baseline, address wider integration, and demonstrate the result on the intended hardware. Each phase produces usable engineering work and evidence for the next decision. Documentation, configuration control, and knowledge transfer begin with the first phase and continue throughout.

Phase	Engineering work and deliverables	Completion criterion
0. Assess and prepare	Inventory source, tools, hardware, interfaces, rights, and available evidence. Restore development access. Define the pilot, risks, scope, and acceptance approach.	The customer and engineering team agree on an executable work package, available assets, and the evidence needed to judge it.
1. Recover a runnable baseline	Stabilize startup, board support, and operating software as needed. Adapt interfaces for selected application modules. Deliver controlled builds, a lightweight validation harness, representative cases, simulation setup, and design and interface notes.	The agreed configuration builds reproducibly and runs selected application behavior with recorded results and explicit limitations.
2. Integrate the wider system	Address required processor, network, programmable-logic, boot, diagnostic, and peripheral interfaces. Deliver integrated builds, interface definitions, regression results, and an updated change-impact assessment.	Representative end-to-end and fault scenarios satisfy agreed integration criteria; remaining qualification and hardware gaps are understood.
3. Verify on target and transfer capability	Exercise the intended hardware configuration, complete the agreed assurance work, and deliver a tagged baseline with build, debug, test, and maintenance guidance. Conduct a receiving-team sustainment exercise.	The agreed target evidence is accepted and the receiving team demonstrates the defined maintenance tasks. Release decisions follow the program's approval process.

The first agreement need not purchase the entire roadmap. A constrained engagement can establish the runnable baseline and recommend a justified next phase. That is a meaningful result when the starting point cannot reliably build or execute. The follow-on assessment should identify remaining dependencies, feasible options, estimated effort, and the evidence still needed before wider use.

Representative validation must have a stated boundary. A harness exercising selected modules, or a network simulation demonstrating message flow, does not establish complete application coverage or production qualification. Broader integration, comprehensive verification, qualification, and fielding support require explicitly agreed work packages. Phase completion is judged against the agreed scope.

Estimate duration and staffing after reviewing the baseline, hardware access, tool availability, and receiving team's capacity. The phases describe a sequence of engineering decisions; their lengths are not universal. Some work can overlap, and discoveries may require revisiting an earlier boundary. Each review should support a concrete choice to proceed, narrow the next increment, resolve a dependency, or stop an uneconomic transition.

Preserve the application and change deliberately

Useful application software represents more than implementation effort. It can embody requirements interpretation, calibration knowledge, fault handling, operational experience, and an accumulated verification record. Rewriting it wholesale can put those investments at risk. Retaining it is also not an automatic decision: known defects, missing requirements, an unsuitable architecture, or poor recoverability may justify a broader redesign.

Begin by establishing the present baseline and the intended sustainment use. Confirm the relationship among source, executable images, configuration, and deployed hardware. Recover requirements and design information where available; distinguish approved requirements from behavior observed in a running system. Existing behavior is evidence to investigate, not an instruction to preserve every historical defect.

A bounded pilot can resolve the highest uncertainties before the program funds a full transition. Select a representative function with meaningful inputs, outputs, timing, diagnostics, and failure behavior. Demonstrate a build and controlled execution using the proposed support environment. Where hardware evidence is needed, exercise the actual target or an appropriately justified test configuration. Record deviations and their causes.

Then make changes in increments that leave their effects understandable. An adaptation layer may contain operating-system or peripheral dependencies without changing the control strategy. A board support package or driver may need reconstruction. A diagnostic interface may need documentation and a compatible implementation. The work should follow the dependency assessment; a new operating system is not the answer to every maintenance problem.

Kernel architecture is one tradeoff among others. A monolithic kernel is not inherently unsuitable, and a microkernel or open-source alternative is not inherently a lower-cost solution. Timing behavior, available drivers, resource use, isolation needs, license conditions, support arrangements, and verification effort all affect the decision. Evaluate candidates against the system's requirements and intended service life.

Open interfaces can improve the range of viable support options. The Department of Defense's February 2025 MOSA guide treats modular architecture, interfaces, intellectual property, and sustainment planning as connected concerns. [4] For an existing system, applying those principles may mean opening a few important boundaries first, rather than undertaking a complete architectural conversion.

Diagnostics need both communication and meaning. A documented service interface is insufficient if the receiving team lacks the definitions needed to interpret identifiers, units, scaling, fault records, or configuration data. ASAM's ODX standard supplies a vendor-independent model for describing ECU diagnostic information. [5] A standard format can support interchange, but the program must still verify the delivered dataset, supported functions, tool compatibility, and permitted uses. Standards adoption alone does not prove independence.

Reuse evidence without assuming qualification transfers

Previously qualified application software is valuable, but changing its execution environment can change the behavior of the integrated system. An operating-system replacement can affect scheduling and synchronization. A driver or peripheral change can alter sample age, update timing, or fault detection. Compiler and numeric changes can affect results even when the application source appears unchanged.

Edward Lee's analysis of cyber-physical systems explains why computational correctness alone does not establish correct physical behavior: timing properties omitted from a software abstraction may still determine whether the system succeeds. [6] For sustainment, this means a successful build or host regression run cannot by itself establish that a replacement platform preserves the relevant behavior.

Plan verification from the change-impact assessment. Identify the requirements, interfaces, assumptions, and assurance claims affected by the proposed change. Determine which existing analyses and test results remain applicable, which must be repeated, and which new evidence is required. Agree on the acceptance approach with the responsible program authorities. The extent of requalification depends on the baseline, the change, and the applicable assurance obligations; it should not be promised away in a sales claim.

Compare relevant behavior under normal operation and faults. Examine startup and shutdown, reset and recovery, scheduling, resource limits, communication loss, diagnostic behavior, and interactions with the physical equipment as appropriate to the system. Test results need an identified hardware and software configuration, reproducible procedures, and an account of limitations. A host test, simulation, and target measurement answer different questions.

Security and access controls belong in the transition design. The receiving organization needs a documented, authorized path for building and deploying approved releases, including the required signing and configuration-control arrangements. Those responsibilities should be assigned and exercised during the handoff.

Make the savings case measurable

Compare credible alternatives over the same service-life horizon. The current arrangement may remain the best choice. Other alternatives may include buying missing deliverables, negotiating a support or licensing change, replacing selected platform components, moving to a new hardware baseline, or redesigning a subsystem. Each option should account for transition effort, qualification and regression work, tools, training, continuing engineering support, and the receiving organization's capacity.

Expected savings arise only when the future costs and operational consequences avoided justify those investments. Use low, central, and high estimates where costs or defect recurrence are uncertain. Show the effects of a shorter service life, a delayed transition, or higher verification effort. Apply the program's cost-estimating conventions consistently, including discounting when appropriate. Do not count the same avoided expenditure twice or present an unpriced risk reduction as a realized dollar saving.

Cost is not the only outcome. Track how long a representative change takes, how much of the build and diagnostic environment the receiving team can operate independently, and whether another qualified provider can perform agreed maintenance tasks. Assess readiness or support-time benefits where the program has a defensible way to measure them. Establish the baseline before claiming improvement.

VCDM sustainment planning

Through Dynamis, Paul Kruchko worked with the Army's Project Manager Self-Propelled Howitzer Systems (PM SPHS) and Ground Vehicle Systems Center Software Engineering Center (GVSC-SEC) to shape a phased scope for software sustainment and internal capability development for the M109A7 Paladin's Vehicle Control and Distribution Module (VCDM).

The VCDM controls vehicle chassis components, including ramps and lights, and distributes low-voltage 28 V power throughout vehicle subsystems. The sustainment engagement addressed the gap between possessing legacy application software and having the platform components, tools, and engineering knowledge needed to maintain it.

The revised six-month proposals focused on a runnable processor baseline, board support and operating-system integration, a compatibility layer for selected application modules, a lightweight C validation harness, representative test cases, and a simulated network demonstration. Design and interface notes accompanied a written assessment of the integration work still needed. The documentation appendix emphasized a practical package that government engineers could use to reproduce the environment, validate the baseline, and continue the work. Broader system integration and formal qualification remained separately scoped activities.

This was a customer engagement to define the technical approach and follow-on scope. The contracting path for executing that proposed scope was not finalized. It illustrates how Dynamis approaches the problem at its outset: make the missing capability explicit, identify the engineering work needed to recover it, and define how the customer would demonstrate that the handoff works.

Deliver a capability the receiving team can operate

The practical acceptance event is a sustainment exercise performed by the receiving organization. Its engineers obtain the controlled source and authorized tools, build the defined baseline, investigate a representative fault, make a bounded change, run the required checks, and deploy to the agreed test configuration. The originating team observes and resolves gaps in the handoff. The evidence should show what the receiving team can do without undocumented intervention.

The delivered package should include the agreed source and configuration items, build instructions, dependency and licensing records, interface and diagnostic definitions, requirements and design information, test procedures and results, known limitations, and an identified maintenance process. The exact contents follow the program's needs and rights. Completeness is demonstrated through use, rather than inferred from the number of files delivered.

Dynamis Consulting & Controls approaches this work as embedded-systems engineering: recovering usable baselines, assessing operating-system and hardware dependencies, implementing targeted platform changes, and developing the documentation and verification evidence needed for sustainment. An initial assessment should give the program a concrete account of what can be retained, what blocks independent maintenance, and what a justified transition would require.

The customer should be able to choose continued supplier support because it offers value, while retaining a credible alternative when its needs change. A successful engagement leaves the application useful, the dependencies understood, and the next maintenance action within the receiving team's demonstrated capability.

References

1. Robert Ferguson, "A 5-Step Process for Release Planning," Carnegie Mellon University Software Engineering Institute, April 22, 2019. [SEI article](#).
2. U.S. Government Accountability Office, *Weapon System Sustainment: DOD Needs to Better Capture and Report Software Sustainment Costs*, GAO-19-173, February 25, 2019. [Report and findings](#).
3. U.S. Government Accountability Office, *Weapon System Sustainment: DOD Can Improve Planning and Management of Data Rights*, GAO-25-107468, reissued September 29, 2025. [Report and findings](#).
4. Office of the Under Secretary of Defense for Research and Engineering, *Implementing a Modular Open Systems Approach in Department of Defense Programs*, February 2025. [Official guide](#). See the technical and business considerations and Appendix C on intellectual property and interface data.
5. ASAM, "ASAM MCD-2 D," also known as Open Diagnostic Data Exchange or ODX. [Official description](#), accessed September 29, 2026.
6. Edward A. Lee, *Cyber Physical Systems: Design Challenges*, UCB/EECS-2008-8, University of California, Berkeley, January 23, 2008. [Technical report](#), especially §2 on timing and abstraction.