Software Solutions Symposium 2017

March 20–23, 2017

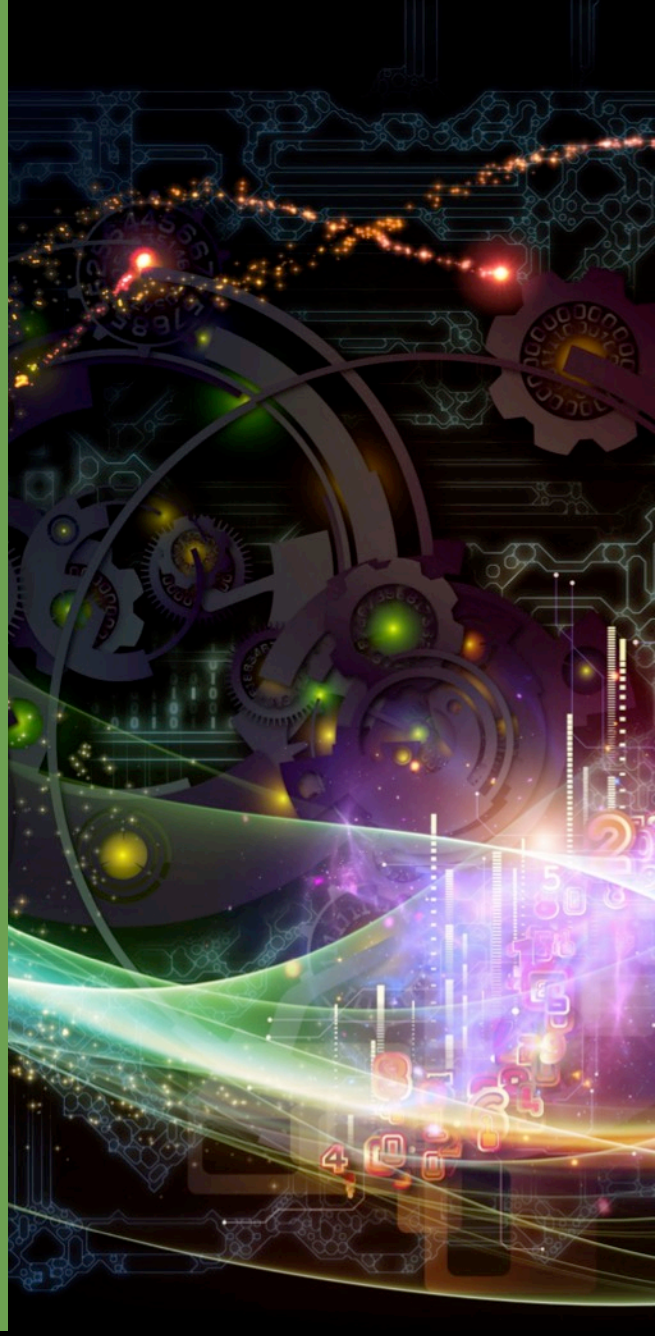
So Much Money for So Little Capability: The Reality of Sustaining DoD Software Systems

David Schneider, PM AFV (TACOM)

Fred Schenker, SEI

Grady Campbell, SEI

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0004467

Introduction

The SEI is working with Product Manager (PdM) Bradley on modernizing its software architecture.

Twenty year history, twenty year future.

In between, 10 years of programmed obsolescence: Future Combat Systems (FCS), Ground Combat Vehicle (GCV).

Opportunities have been identified in re-architecting the software to anticipate and accommodate future change.

While pursuing re-architecting opportunities, **other concerns were identified** that were out of scope for the architecture effort.

There are feasible actions that would address these concerns and make sustainment more cost-effective.

This Presentation

Contents

- Sustainment concerns for DoD acquisition
- Objectives for improved sustainment
- Specific concerns and actions for improving sustainability related to each of the following:
  - acquisition
  - systems engineering
  - software environment
  - software requirements
  - software architecture
  - component design and implementation
  - verification, validation, and certification
  - software delivery and operational use

Goal

To highlight the importance of integrating software sustainability as a DoD acquisition concern and to provide high-level guidance for doing so.

Sustainability –

The ease with which (software) capabilities can be evolved to continue to satisfy customer needs as those needs and operational context change.

Sustainability as a Concern for DoD Acquisition

Sustainment is an issue for DoD weapon systems:

- Long-term sustainment of software is needed to support evolving war-fighter needs and technology.
- Acquisition offices make short-sighted decisions that prioritize initial capabilities to limit schedule and budget over-runs, and they fail to identify and account for likely changes.
- Sustainment costs increase significantly when potential changes are not identified and planned for.
- Sustainment costs end up far exceeding the cost of initial development.

Mitigation difficulties

Instead of addressing the root causes of these issues, the community develops "better" cost models (to include the cost inefficiency).

The items identified in this briefing are not surprising. What *is* surprising?

We continue to ignore them, leading to much higher sustainment costs.

Why Software Sustainability is a Concern

Software is built to meet the perceived needs of a customer or market.

…and then what happens?

Understanding improves

Uncertainties and imperfect understanding of actual needs necessitate software changes as understanding improves.

User needs change over time

Most useful software is in use for many years, if not decades.

- Changes in customer/market needs and enabling technologies compel software changes to avoid obsolescence.
- Software is built using computational technologies that evolve and change; failure to keep up fosters obsolescence.

Multiple versions are needed

Software may exist in multiple versions that need to be kept consistent to avoid duplication of effort and defects as changes occur.

Objectives for Improved Software Sustainability

Expose potential changes

Share knowledge about potential future changes as a basis for building software that will be easier to modify.

Keep capabilities aligned with needs

Keep software capabilities aligned with changing customer/market needs to avoid technical debt and obsolescence.

Maintain architectural and structural integrity

Ensure architectural coherence and structural integrity are maintained as software is changed.

Maintain similarities

Maintain similarities among software versions as changes occur to minimize redundant efforts.

Keep documentation current

Keep documentation current as a reliable expression of expected software behavior, structure, and rationale.

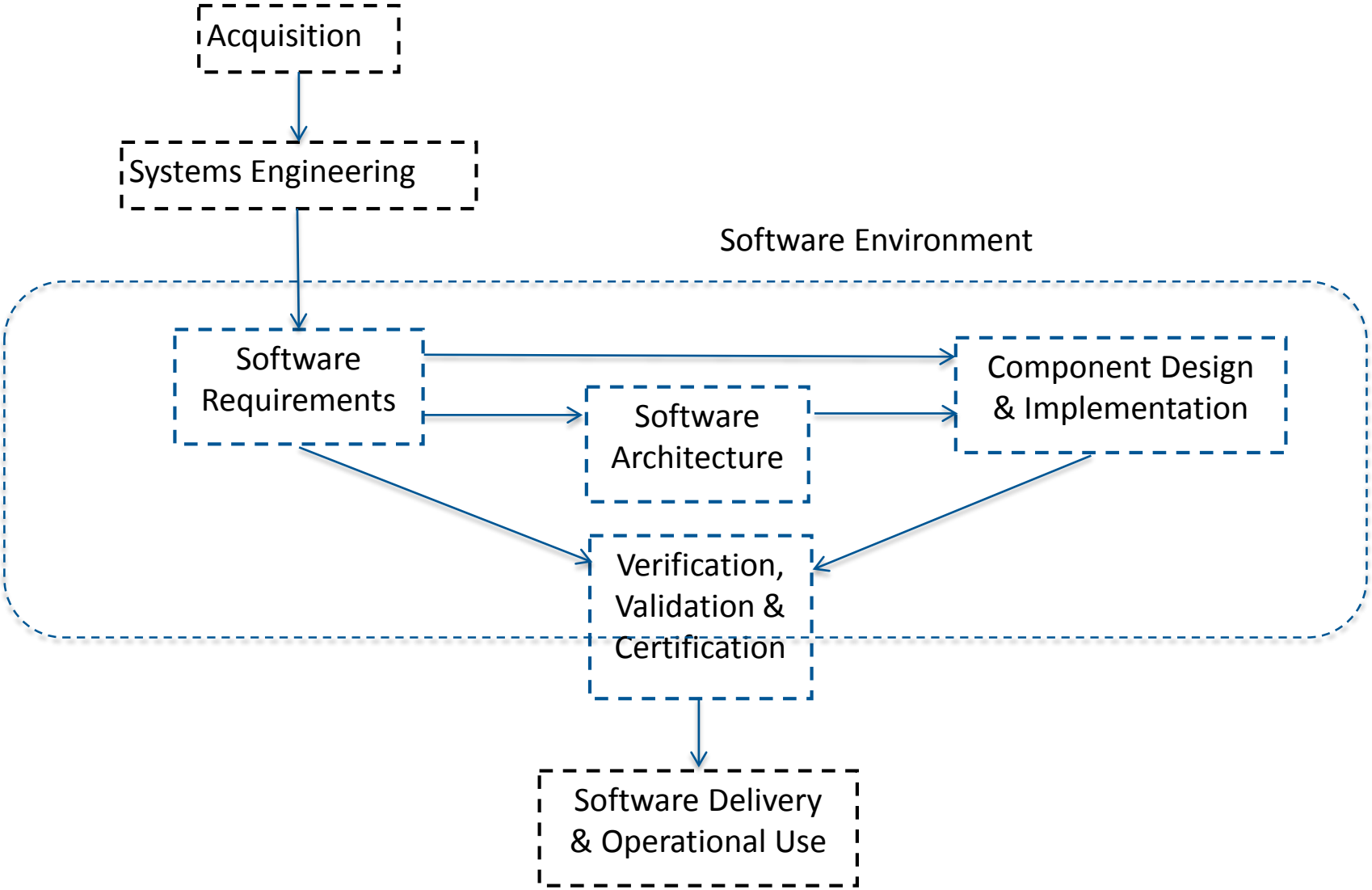
Maintain development infrastructure

Maintain development infrastructure to gain improvements and avoid the risk of non-support and more difficult transition later.

Expedited evaluation efforts

Institute methods for expedited evaluation of iteratively-evolved software, enabling faster deployment of updated capabilities.

Specific Sustainability Concerns



Acquisition Concerns

- Narrow focus on objectives of current near-term effort, discounting impacts on sustainment
- Insufficient regard for opportunities for commonality across alternate versions of a single platform and other related or similar efforts
- Ineffective coordination and communication among interdependent programs (e.g., about responsibilities and software qualities)

Acquisition –

The activities involved in obtaining the capabilities needed to support the operations of a customer enterprise.

Actions to Improve Sustainability

Institute comprehensive (PEO-level) cross-program software planning/coordination and commonality tradeoff discipline

Standardize software practices and technical data across all related programs

- Establish a reference software architecture and conformant software repository for use on all programs
- Institute common software lifecycle technical data standards (form and content) across all suppliers

System Engineering Concerns

- Changes in system requirements and architecture frequently translate into software changes.
- Systems engineering assumptions can prematurely preclude alternative software solutions.
- Preferred software solutions can conflict with systems engineering assumptions.
- Insufficient information about potential changes in needs and technology inhibits the ability to build changeable software.
- Failure to accurately specify all behavioral properties of other system components and interfaces increases cost and time to build dependent software.

System engineering –

The activities involved in establishing the requirements, architecture, and acceptance criteria for a system that provides needed capabilities to a customer enterprise.

Actions to Improve Sustainability

- Evaluate software implications when making system tradeoffs.
- Fully define a system context for needed software capabilities that identifies the following:
  - assumptions and decisions that are unlikely to change over the useful life of the product
  - areas of requirements or design uncertainty (aspects that may change with additional experience or information)
  - areas of likely future changes in customer needs and technology
  - how tradeoff decisions may change if customer needs, technology, or other circumstances change
- Define in the system architecture the behavior that software is expected to exhibit.

Software Environment Concerns

- Failing to maintain environment compatibility with current industry practices (computing hardware, OS, language, libraries, methods, tools)
- Using obsolete or incompatible COTS products/versions
- Using hardware emulation capabilities that do not conform to actual hardware behavior
- Inability to build and operate alternate (e.g., legacy or variant) versions of the software within a single environment
- Failing to provide configurability to test all supported product versions, equipment configurations, and scenarios of use

Software environment –

The hardware/ software infrastructure in which software is developed, sustained, and evaluated as to expected behavior.

Actions to Improve Sustainability

- Establish and sustain a standard configurable environment for all development and evaluation efforts.
- Support all activities of the software lifecycle within the environment, including documentation, reviews, and fully automated testing.
- Maintain compatibility across all installations.
  - Plan and budget for regular updates to all tools and computing equipment.
  - Coordinate environment updates to avoid having to convert between installations.
- Anticipate and budget for building and sustaining standard hardware device emulation and environment simulation capabilities.
- Support software configurable for observability during test with a mix of emulated and actual hardware in a live and/or simulated environment.
- Support a standard suite of version-configurable test materials, supporting automated regression and change-driven testing.

Software Requirements Concerns

- Requirements that are poorly organized, inconsistent, incomplete, informal, verbose, vague, or ambiguous provide an unsound basis for evaluating the resulting solution.
- A solution based on requirements that describe non-observable behavior cannot be properly verified.
- Requirements that prescribe excessively precise quality factor limits and other engineering decisions inhibit the potential for change.
- Requirements that fail to distinguish fixed versus potentially changeable needs results in improperly constrained solutions.
- Documentation (and training materials) not kept current with the as-built solution becomes unreliable and loses its usefulness.

Software requirements –

The activity that defines the expected as-built behavior of the software, constituting its criteria for acceptance.

Actions to Improve Sustainability

- Establish a standardized form for defining requirements as a coherent expression of expected software behavior.
  - Define externally observable behavior, external dependencies, and quality criteria.
  - Document all assumptions, alternatives, and rationale for future reference when changes are needed.
- Describe the change context in the requirements themselves as the basis for designing and implementing a sustainable solution:
  - how and under what conditions assumptions or alternatives could change
  - areas of insufficient knowledge or uncertainty that are yet to be resolved
  - aspects that are likely to change as customer needs or technology evolve

Software Architecture Concerns

- Understanding how to make changes to a solution is costly without understanding its as-built structure, the purpose of its elements, and the dependencies among them.
- The as-built structure of software is unstable when it is not expressed in a well-considered, shared specification.
- Failing to maintain the architectural coherence and integrity of a solution as changes are made increases the cost and risk of future changes.
- Making unforeseen changes to software has unpredictable cost and risks.
- Software should be designed to make likely changes easier. Without such anticipation, we unnecessarily increase the cost and risk of change in general.

Software architecture –

The activity that defines the structure and composition of the software implementation as a set of interdependent components.

Actions to Improve Sustainability

- Institute use of a sound architectural method that separates concerns for static, dynamic, and physical structure.
- Devise a reference architecture that is well-structured and receptive to future changes that were identified as being likely in the requirements.
- Define and analyze change scenarios to fix exposed architectural impediments to projected software changes.
- Define each software release as a disciplined customization of the reference architecture.
- Revise the reference architecture as the software-change projection evolves, based on evolving customer needs and technology.

Component Design and Implementation Concerns

Component Design and Implementation

- Components implemented without reference to an architecture that clearly allocates responsibilities may omit or redundantly include needed capabilities.
- A component lacking adequate definition will be difficult to build and use properly:
  - the responsibilities for implementing observable behavior or services that other components need to use must be precisely defined
  - assumptions, constraints, potential changes, references, and rationale information that current and future developers need to know to build and test it must be well documented
- A component is sustainment-negative when changes in its implementation do not change its interface but require changing the implementation of client components.
- Code that is not readable and properly documented regarding meaning and rationale will be difficult to safely change.
- Inadequate provision and use of libraries of commonly needed functionality will lead to differing, redundant implementations.

Component design and implementation –

The activities involved in defining the behavior and services of the components that comprise the software implementation.

Actions to Improve Sustainability

Component Design and Implementation

- Create a design and implementation for each component that will be easy to adjust to accommodate projected future changes in requirements.
- Institute sound design and implementation practices, enforced by directed peer reviews that reduce costly test-fix efforts.
  - Focus on areas of complexity, ambiguity, likely future change, and developer uncertainty.
  - Adhere to prescribed coding conventions for readability, understanding, and ease of change.
  - Ensure appropriate commentary sufficient to explain intent, alternatives/tradeoffs, rationale, and potential changes.
  - Ensure updates to requirements, architecture/design, test materials, and user documentation to reflect implementation efforts.
  - Perform root cause analyses to ensure future avoidance or discovery of defects missed in past reviews.
- Support provision, sustainment, and appropriate use of standard coding patterns and reusable implementations by developers.

Verification, Validation, and Certification Concerns

Verification, Validation, and Certification

- A focus on detailed features, versus systemic behavior (observable functions and qualities), that emphasizes superficial characteristics over effective fit to customer operations
- A focus on discovering individual defects without analysis of root causes that fails to deter recurrence of similar defects
- An insufficient focus on software quality interdependencies, leading to software that is functional but less effective for users
- Failure to discover inconsistencies with software in supporting documentation/training/test materials
- Failure to require hardware suppliers to adhere to fully specified interfaces, resulting in delays and error-prone changes to software
- An inability to perform evaluation of multi-version (including hardware variant) or dynamically configurable software, resulting in redundant effort to separately evaluate each version

Verification, Validation, and Certification –

The activities involved in determining the conformance of as-built software to specified and actual expectations.

Actions to Improve Sustainability

Verification, Validation, and Certification

- Organize to rapidly evaluate iteratively released interim versions of software capability.
- Create and sustain configurable materials supporting software evaluations (verification, validation, and certification).
- Systematically reuse test scenarios/scripts and data, configurable to requirements differences among software versions.
- Expedite software evaluation using a simulated environment with software-emulated devices exhibiting specified interfaces and quality factors versus waiting for actual hardware.
- Coordinate evaluation efforts so that later efforts can be viewed as regression-based extensions of earlier efforts.
- Explore options for streamlining certification efforts when previously certified software has been revised within an understood limited scope of effect.

Software Delivery and Operational Use Concerns

Software Delivery and Operational Use

- Phased deployment of updated software across hosts that requires excessive delay to full deployment, resulting in operational inconsistencies and suboptimum capabilities among hosts
- Tying software deliveries unnecessarily to hardware device deliveries, delaying improvements in deployed software
- Restricting software deliveries to base facilities and physical media, limiting options for rapid updates
- Software pre-configured for specific hardware, fostering proliferation of multiple versions requiring redundant maintenance
- Enterprises lacking effective means for developers to get feedback on software effectiveness and defects from user operations

Software Delivery and Operational Use –

The activities involved in deploying and using operational software (specifically, on a fleet of dispersed host platforms).

Actions to Improve Sustainability

Software Delivery and Operational Use

- Explore autonomously defined software elements that can be independently built, evaluated, and installed.
- Explore means to securely deploy software updates remotely en masse (independent of hardware upgrades).
- Explore dynamic reconfiguration of software capabilities based on installation/failure/removal of peripheral hardware devices.
- Explore a means to update non-critical software components without a need for full-scale software/system evaluations.
- Verify that identified *operational concerns* correspond to known potential requirements changes.
- Explore interoperability issues with operating forces that are using different software releases having differing capabilities.

Conclusions

There are many aspects to effective sustainability; we have tried to provide awareness and recommendations to integrate sustainability into software development.

Status quo drivers for upgrade projects are short term incentives and focus on new or upgraded capability / functionality for the warfighter.

This focus ignores funding what needs to be done to improve the sustainability of the software to meet changing warfighter needs.

In the context of ever growing costs for software sustainment, the status quo must change.

Contact Information

Presenter Contacts

Fred Schenker
Senior Member of the Technical Staff, SEI
ars@sei.cmu.edu

Grady Campbell
Senior Engineer, SEI
ghc@sei.cmu.edu

David Schneider
Software Branch Chief
PM Armored Fighting Vehicles
david.s.schneider10.civ@mail.mil

U.S. Mail

Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Customer Relations

Email: info@sei.cmu.edu
Phone: +1 412-268-5800
Fax: +1 412-268-6257

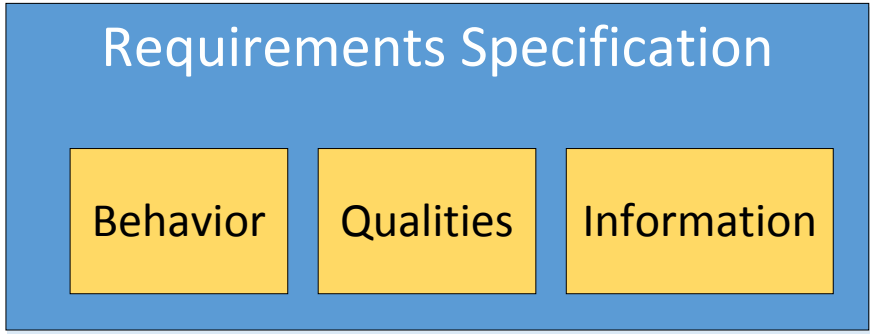
Web

www.sei.cmu.edu

References

1. Campbell, G., *Software-intensive Systems Producibility: A Vision and Roadmap (v0.1)* (CMU/SEI-2007-TN-017), CMU Software Engineering Institute, Dec 2007.
2. Cloutier, R., et al., "The Concept of Reference Architectures," *Systems Engineering* 13 (1), 2010,14-27.
3. Campbell, G., "The Illusion of Certainty," *7th Annual Acquisition Research Symposium*, Naval Postgraduate School, May 2010, 257-264.
4. Campbell, G., Levinson, H., Librizzi, R., *An Acquisition Perspective on Product Evaluation* (CMU/SEI-2011-TN-007), CMU Software Engineering Institute, Oct 2011.
5. Becker, C., et al., "Requirements: The Key to Sustainability," *IEEE Software* 33 (1), Jan/Feb 2016, 56-65.
6. Heroux, A. M., Allen, G. *Computational Science and Engineering Software Sustainability and Productivity (CSESSP) Challenges Workshop Report*. Networking and Information Technology Research and Development (NITRD) Program, Sep 2016.
7. Boehm, B., *The Criticality of Systems maintainability and the Need for Software-Intensive System (SIS) System Maintainability Readiness Levels.* PSM Users' Group Keynote, Feb 24, 2016.

Requirements – Lethality Example



Lethality Example (Behavior)
- Maintain knowledge of the operational environment
- Engage targets
- Evaluate lethality status

Lethality Example (Qualities)
- Safety
- Reliability
- Performance

Requirements Statements
- Context (mode, state, input)
- Action (expected behavior, constraints)

Lethality Example (Information)
- Natural environment (trees, terrain, mountains, weather)
- Operational environment (enemies, friendlies, non-combatants)
- Operational State (equipment, ammunition)
- Operational Profile (tasking, intelligence)