AdaCore WHITE PAPER

Disruptive Technology for Military-Grade Software

SPARK is a software development technology (programming language and verification toolset) specifically designed for engineering ultra-low-defect-level applications, for example where safety and/or security are key requirements. SPARK Pro is the commercial-grade offering of the SPARK technology developed by AdaCore, Altran and Inria.

SPARK has an extensive industrial track record. Since its inception in the late 1980s it has been used worldwide in a range of industrial applications such as civil and military avionics, air traffic management/control, railway signaling, cryptographic software, cross-domain solutions, medical devices and automotive.

The SPARK language is a large subset of Ada 2012. SPARK includes as much of Ada as is possible/practical to analyze formally, while eliminating sources of undefined and implementation-dependent behavior. SPARK includes Ada's program-structure support (packages, generics, child libraries), most data types, safe pointers, contract-based programming (subprogram pre- and postconditions, scalar ranges, type/subtype predicates), Object-Oriented Programming, and the Ravenscar subset of tasking features.

Since its inception at the DoD in the late 1970s, the Ada programming language has been used successfully in hundreds of military projects in the US and the rest of the world. Ada is the programming language that helped build the mainstay of the Air Force or Navy, as it is used on combat aircraft programs such as the F-15 Eagle, the F-16 Fighting Falcon, the F-18 Hornet and the F-22 Raptor. It is also used in many other allied programs such as the Eurofighter Typhoon or the JAS 39 Gripen. Beyond combat aircraft, it is only used on many other military programs such as the C-130J and C-17 airlifters, the AH-64 Apache helicopter and many more.[1]

This is a direct result of the choice of General Design Criteria originally set for the Ada language: Generality, Reliability, Maintainability, Efficiency, Simplicity, Implementability, Machine Independence and Complete Definition.[2] While some of these criteria are shared with other programming languages today, Ada's advantages are particularly visible in the areas of Reliability and Maintainability, which can be most succinctly described as follows (from Ada's Technical Requirements document):

Reliability

The language shall be designed to avoid error-prone features and to maximize automatic detection of programming errors.

Maintainability

The language should promote ease of program maintenance. It should emphasize program readability (i.e., clarity, understandability, and modifiability of programs).

In a nutshell, Ada is a classical stack-based general-purpose language. Unlike languages such asJava, it does not require garbage collection, a virtual machine or an extensive runtime. Additionally, Ada is not tied to any specific development methodology. It offers:

- simple syntax designed for human readability;
- structured control statements;
- flexible data composition facilities;
- strong type checking;
- traditional features for code modularization ("subprograms");
- standard support for "programming in the large" and module reuse (packages, Object-Oriented Programming, child libraries, generic templates);

- a mechanism for detecting and responding to exceptional run-time conditions (“exception handling”);
- high-level concurrency support (“tasking”) along with a deterministic subset (the Ravenscar profile) appropriate in applications that need to meet high-assurance certification requirements; and
- the possibility to restrict the features of the language through “restrictions” for use in constrained environments.

The success of Ada in flagship military programs is a direct result of Ada maintaining a strong focus on reliability and maintainability throughout its evolution and successive standards (1983, 1995, 2005, 2012 and soon 2022), which has not been matched by other programming languages. This focus is critically relevant to the development of software in military systems today. With increased digitization of all functions, both the complexity and importance of the software grow, which can lead to catastrophic failures (such as the Aegis missile cruiser USS Yorktown losing control of its propulsion system in 1997 after a division-by-zero error in its Remote Data Base Manager software[3]). These concerns are heightened by the combination of an increase in cyber warfare and the increasing reliance of weapon systems on software. The former increases the likelihood that a vulnerability in critical software will be discovered and exploited by enemies: operation Orchard[4], which demonstrated in 2007 how enemy air defenses could be suppressed by cyber means, was probably the first of many more to come. The latter will probably increase by one order of magnitude as systems of systems, such as the Next Generation Air Dominance[5] or the British Tempest[6], are developed and fielded.

The Future of Software in Combat Systems

The properties of software that contribute to its unique and growing importance to military systems are summarized in this quote from the “Critical Code: Software Producibility for Defense” study by the US National Research Council in 2010:

> Software is uniquely unbounded and flexible, having relatively few intrinsic limits on the degree to which it can be scaled in complexity and capability. Software is an abstract and purely synthetic medium that, for the most part, lacks fundamental physical limits and natural constraints. For example, unlike physical hardware, software can be delivered and up-graded electronically and remotely, greatly facilitating rapid adaptation to changes in adversary threats, mission priorities, technology, and other aspects of the operating environment. The principal constraint is the human intellectual capacity to understand systems, to build tools to manage them, and to provide assurance—all at ever-greater levels of complexity.

The role of software has been restated in the “Software Acquisition and Practices (SWAP)” study[7] by the US Defense Innovation Board in 2019, which emphasizes the importance of reliability and maintainability.

The DoD requires in its “Defense Innovation Board Metrics for Software Development” study that the number of bugs caught in testing vs field use be greater than 75% for custom software running on commodity hardware (in data centers or in the field); and greater than 90% for custom software running on custom hardware (e.g., embedded software).

The DoD further strongly recommends in its “Defense Innovation Board Do’s and Don’ts for Software” study that military systems “use modern software languages and operating systems”, explaining that

> Modern programming languages and software development environments have been optimized to help eliminate bugs and security vulnerabilities that were often left to programmers to avoid (an almost impossible endeavor).

We can argue that the long-lived Ada, which was the first programming language to exhibit these increased properties of reliability, is a prime example of such a modern language.

The same study also recommends unequivocally that all source code be made available for auditability and maintenance, with two of the "Ten Commandments of Software". For auditability, the study says:

> Commandment #6. Every purpose-built DoD software system should include source code as a deliverable.

The study elaborates on this commandment, saying that:

> Providing source code will also allow the DoD to perform detailed (and automated) evaluation of software correctness, security, and performance.

For maintenance, the study says:

> Commandment #7. Every DoD system that includes software should have a local team of DoD software experts who are able to modify or extend the software through source code or API access.

With the availability of many online tutorials[8], it has never been easier to solve the often mentioned shortage of Ada programmers by training them in-house. Moreover, the focus of Ada on maintainability will be much appreciable to those DoD teams that are tasked with "modify[ing] or extend[ing] the software through source code" developed by a DoD contractor.

A related noteworthy effort to reduce development costs, integration costs, and time-to-field for avionics capabilities is the The Open Group's Future Airborne Capability Environment (FACETM) Consortium.[9] FACE defines a standard software architecture and associated data model and leverages common open standards to lower the cost of reusing airborne software components across multiple programs and platforms. Among the languages that are called out in the FACE Technical Standard – C, C++, Ada and Java – Ada promotes high assurance coupled with code portability the best. AdaCore is a Principal Member of FACE and helps FACE software suppliers meet assurance requirements for reliability, safety, and security while realizing the portability and reusability benefits that come from FACE conformance.

AdaCore and the Future Airborne Capability Environment (FACE)

The FACE™ approach is a government-industry initiative for reducing defense system life cycle costs through portable and reusable software components. It consists of a technical approach — a software standard based on well-defined common interfaces — and a business strategy for encouraging the development and deployment of FACE conformant products.

AdaCore is committed to the success of the FACE approach, and both the Ada language and the company's product offerings directly support the initiative's objectives. Ada is unique in its support for portable, reliable and efficient code. The language was designed for programming critical real-time embedded systems and has a long and successful track record in military / aerospace projects and in other high-assurance domains where safety and/or security are required.

For more information on how AdaCore's product offerings contribute to the FACE approach, please visit https://www.adacore.com/industries/defense/face

SPARK Use Cases in Military Systems

Two major design goals of SPARK are (1) the provision of an unambiguous and formal semantics, which therefore permits (2) the soundness of static verification: i.e., the absence of false negatives. If the SPARK tools report that a program does not have a specific vulnerability, such as a buffer overflow, then that conclusion can be trusted with mathematical certainty. Soundness builds confidence in the tools, provides evidence-based assurance, completely removes many classes of dangerous defects, and significantly simplifies subsequent verification effort (e.g., testing), owing to less rework, and in some cases can eliminate these activities entirely.

In a context where source code is provided by a contractor, SPARK analysis allows a thorough audit of the source code, by proving that the guarantees claimed by the contractor are indeed achieved in the codebase. A major guarantee that can be thoroughly audited is the absence of run-time errors (such as buffer overrun or integer overflow), a well-known source of security vulnerabilities in software written in C/C++. The system integrator can also verify claims made by the contractor by using SPARK to prove that contractor-provided contracts on subprograms are met by the implementation. Moreover, the contractor and system integrator can trace these code-level contracts to software requirements, adding mathematical assurance that the software satisfies its requirements. In a configuration where SPARK code is provided by a subcontractor to the prime contractor, SPARK analysis guarantees can be checked both at the prime contractor level and at the customer level.

Note that all Ada 2012 capability sets defined in FACE allow contract-based programming, hence SPARK formal verification.

SPARK is used today by defense actors across the world to enhance the reliability of their most critical software assets:

- QinetiQ[10] uses SPARK to modernise the development environment for its Trials Control System (TCS), a command and control system designed specifically for the training, test and evaluation of military equipment.
- secunet[11] uses SPARK to develop cross-domain solutions for civilian and military usage.

SPARK is used outside the defense sector by actors in other critical industries to provide the necessary safety and security guarantees that they are expecting of their software:

- NVIDIA[12] uses SPARK to get strong guarantees that their most critical firmware codebases are free of whole classes of bugs and vulnerabilities.
- Hillrom[13] uses SPARK for their ECG algorithm development to demonstrate absence of run-time errors, leading to fewer problems at integration time and during maintenance.
    - JTEKT[14] uses SPARK to prove critical safety properties of their safety-critical power steering system software.

Towards Sufficient Evidence for Military Systems

In 2007, the US National Research Council published the influential report “Software for Dependable Systems - Sufficient Evidence?”[15] recommending to “Make the most of effective software development technologies and formal methods” in order to reduce the cost and difficulty of producing dependable software, and to “Include security considerations in the dependability case” because security vulnerabilities can undermine the case made for dependability properties.

In a context of increasing cyber security threats with the development of cyber warfare capabilities by many actors, an effective protection requires increasing the speed at which new software capabilities are developed and deployed, while maintaining a high level of protection of the software produced against attacks. This requires more automation than is possible with traditional verification methods based on tests and reviews. The guarantees provided by automatic verification as demonstrated in SPARK will be critically useful in this context.

---

1. https://www2.seas.gwu.edu/~mfeldman/ada-project-summary.html#Military_Applications_ https://www.adacore.com/company/our-customers
2. http://iment.com/maida/computer/steel/steelman.htm
3. https://en.wikipedia.org/wiki/USS_Yorktown_(CG-48)
4. https://en.wikipedia.org/wiki/Operation_Outside_the_Box
5. https://fas.org/sgp/crs/weapons/IF11659.pdf
6. https://en.wikipedia.org/wiki/BAE_Systems_Tempest
7. https://innovation.defense.gov/software/
8. https://learn.adacore.com/
9. https://www.opengroup.org/face
10. https://www.adacore.com/press/qinetiq-selects-mentorship-service-spark
11. https://www.adacore.com/press/spark-pro-secunet
12. https://blogs.nvidia.com/blog/2019/02/05/adacore-secure-autonomous-driving/
13. https://www.adacore.com/uploads/techPapers/Welch-Allyn-Whitepaper.pdf
14. https://www.adacore.com/press/jtekt-spark-pro-automotive
15. https://www.nap.edu/catalog/11923/software-for-dependable-systems-sufficient-evidence