Aerospace, Programming Languages, and Information Technology Co-Development: Software-Defined Aerospace and Autonomy

Brendan Sechter

2026-07-23

This eleventh article of the twelve-part series covers the shift from mechanical and hydraulic aerospace systems to software-mediated systems across the 1980s through 2020s. The engineering pattern that produced this shift is that once computing capability reached capability thresholds, aerospace systems that had previously been implemented mechanically or hydraulically could be reimplemented in software with considerable improvements in weight, cost, flexibility, and function. Fly-by-wire flight control replaced mechanical linkages. Glass cockpit displays replaced electromechanical instruments. Software-defined radios replaced fixed-function communication equipment. Unmanned aerial vehicles operated software-mediated flight functions that had previously required onboard human pilots. Autonomous mission planning replaced dedicated flight-planning personnel for mission classes. Together these transitions transformed aerospace platforms into what the industry now calls software-defined systems, in which the mission capability is determined considerably by the software configuration rather than by the mechanical hardware.

The framing established in A237 identified computing capability as the enabling side of the co-development coupling from roughly 1970 onward. This article treats the programmatic manifestations of that enabling capability in aerospace platforms. The software engineering practices treated in A244 and A245 supplied the engineering methodology that made software-mediated aerospace systems achievable at the reliability targets that safety-critical operation requires. The real-time operating system technology treated in A241 supplied the computing substrate on which the software-mediated systems run.

Fly-by-Wire Flight Control

Fly-by-wire replaces the mechanical cables, pushrods, and hydraulic servos that traditionally connected pilot controls to aircraft control surfaces with electrical signals and computer-mediated actuation. The pilot's stick and rudder pedal displacements are read by electrical position sensors, transmitted to a flight-control computer that computes the appropriate control-surface deflection commands, and delivered to electrohydraulic or electric actuators that move the control surfaces. The engineering value of fly-by-wire is that the control law that maps pilot input to control surface output can be arbitrarily complex, permitting flight envelope protection, gust alleviation, structural mode damping, and other functions that mechanical linkages cannot economically provide.

The first production fly-by-wire aircraft was the Concorde supersonic transport, first flight 1969 and operational from 1976, which used analog electronic flight-control for pitch and roll augmentation while retaining mechanical linkages for the primary control paths. The first production aircraft with full digital fly-by-wire was the General Dynamics F-16 Fighting Falcon, first flight 1974 and operational from 1978, which was designed as relaxed-static-

stability aircraft that required continuous computer-mediated pitch control to remain flyable. Static stability of an aircraft is characterized by the static margin, treated as standard material in Etkin 1996 previously cited in A241, defined as the fraction of mean aerodynamic chord separating the aerodynamic center from the center of gravity,

$$K_n = \frac{x_{\text{ac}} - x_{\text{cg}}}{\bar{c}}$$

with positive K_n indicating a stable aircraft that returns to trim after a disturbance and negative K_n indicating an unstable aircraft that diverges from trim without active control. The F-16 was designed with static margin K_n approximately negative 5 percent, which produces significantly better performance than a stable aircraft of the same size and weight but requires the fly-by-wire computer to be always operational, providing one of the engineering rationales for the fault-tolerant computer architectures treated in A244.

The first commercial aircraft with full digital fly-by-wire was the Airbus A320, first flight 1987 and operational from 1988. The A320 established the commercial-aviation fly-by-wire pattern that subsequent Airbus and Boeing aircraft extended. The control law that fly-by-wire implements can be characterized by its transfer function from stick input to aircraft angular rate response. For a pitch-rate command system with time constant τ and gain K, the closed-loop response is

$$G(s) = \frac{K}{\tau s + 1}$$

where s is the Laplace transform variable and G is the transfer function relating stick displacement to pitch rate. The choice of K and τ across the flight envelope, with gain scheduling that adjusts the parameters based on air density and airspeed, determines the handling qualities that pilots experience. The tuning of these parameters is one of the engineering activities that consumes significant flight-test time on modern aircraft development programs.

The subsequent development of fly-by-wire across the 1990s and 2000s produced the control-law sophistication that characterizes contemporary aircraft. Envelope protection prevents the aircraft from exceeding structural or aerodynamic limits even when the pilot commands would otherwise cause exceedance. Gust alleviation actively damps atmospheric-turbulence responses to reduce structural fatigue and passenger discomfort. Autothrottle integration coordinates engine thrust with control-surface deflections to maintain speed. Each of these functions is straightforward to implement in software once the sensor and actuator infrastructure is in place, and each provides substantial value that mechanical control systems cannot economically provide.

Glass Cockpit Displays

Glass cockpit displays replace the electromechanical instruments that traditionally occupied aircraft cockpit panels with software-driven electronic displays, initially cathode-ray tubes and later liquid-crystal displays. The Electronic Flight Instrument System hereafter EFIS combines primary flight display, navigation display, engine indication, and system-status information on a small number of large-format displays whose content is generated by graphics-processing computers under software control. The value of glass cockpit displays is that the display content can be tailored dynamically to the current flight phase and pilot workload, that display real estate is not permanently consumed by rarely-used instruments, and that the display generation is far lighter than the electromechanical instrumentation it replaces.

The Boeing 757 and 767 first flown in 1982 were the first commercial transport aircraft with marked glass cockpit installations, with electronic primary flight displays and engine indication supplementing traditional electromechanical instruments for backup. The Airbus A320 in 1987 extended the glass cockpit to essentially all cockpit displays, with electromechanical backup limited to standby instruments for minimal-function operation in case of complete display-system failure. Subsequent commercial and military aircraft adopted glass cockpits nearly universally through the 1990s and 2000s, with the engineering trade-offs among display size, brightness, resolution, and refresh rate stabilizing at the values that liquid-crystal display technology could deliver at aerospace-appropriate reliability. The comprehensive reference on digital avionics including glass cockpit displays and their integration with other avionics subsystems is Spitzer 2006, the standard multi-volume handbook for the discipline.

The display-refresh rate required for aerospace cockpit applications is determined by the highest-bandwidth dynamic content the display must present. Attitude indicators must respond to aircraft angular rate changes at rates matching pilot perceptual bandwidth, giving display update rates of approximately

$$f_{\text{display}} \gtrsim 30 \text{ Hz}$$

with modern liquid-crystal displays typically operating at 60 Hertz or higher, well above the perceptual threshold. Latency from sensor measurement to display presentation must also be bounded to preserve the pilot's tight-loop control response. The total sensor-to-display latency budget

$$T_{\text{sensor-display}} = T_{\text{sensor}} + T_{\text{bus}} + T_{\text{compute}} + T_{\text{graphics}} + T_{\text{display}}$$

typically must fit within approximately 50 to 100 milliseconds to avoid pilot-perceptible lag, with each contributing subsystem consuming approximately 10 to 30 milliseconds under realistic engineering budgets. The engineering discipline of low-latency graphics generation for aerospace displays became an extensive specialty within the broader aerospace software community and produced standards including ARINC 661 for cockpit-display definitions and several user-interface description standards adapted for aerospace-specific reliability, safety, and qualification requirements.

Software-Defined Radios

Software-defined radios replace fixed-function communication equipment with programmable digital signal-processing platforms whose modulation, protocol, and frequency-band selection is determined by software configuration rather than by hardware component selection. The foundational paper on the software-defined radio architecture concept is Mitola 1995 in IEEE Communications Magazine, which introduced the engineering framework that subsequent commercial and military software-defined radios adopted. The traditional radio architecture used analog filters, mixers, and demodulators to a particular communication protocol, with sizable rework required to support additional protocols or frequency bands. A software-defined radio uses wideband analog-to-digital and digital-to-analog conversion connected to programmable digital signal processors that implement the modulation and protocol functions in software, permitting a single hardware unit to support multiple communication modes through software configuration changes.

The value of software-defined radios in aerospace applications is that a single hardware unit can support the wide variety of communication protocols that aerospace platforms must interoperate with, including civil-aviation voice on very-high-frequency amplitude modulation, controller-pilot data link on very-high-frequency data broadcast, satellite communications on multiple frequency bands, military tactical data link on Link 16 and its successors, and

increasingly wideband broadband communications for high-throughput mission data. The engineering value of consolidating these functions into a single programmable platform is appreciable in weight, cost, and future-proofing terms, at some cost in the engineering effort required to make the software-defined implementation as reliable as the dedicated-hardware alternatives.

The Nyquist-Shannon sampling theorem cited in A241 bounds the analog-to-digital conversion rate that a software-defined radio must support to preserve the full information content of the radio-frequency signal. For a signal of bandwidth B centered at radio frequency f_c, direct sampling requires

$$f_{\text{sample}} \geq 2 f_c$$

which for typical aerospace radio-frequency bands from 30 megahertz to 30 gigahertz gives sample rates from 60 megasamples per second to 60 gigasamples per second, at the high end substantially exceeding contemporary analog-to-digital converter capability. Direct-conversion architectures address this by downconverting the radio-frequency signal to baseband before sampling, requiring only $2B$ rather than $2f_c$, at some cost in the engineering complexity of the analog downconversion stage.

Unmanned Aerial Vehicles

Unmanned aerial vehicles are the case in which software substitution for onboard human function reaches its logical extreme. The fixed-wing UAV design and operational engineering treated in the earlier corpus fixed-wing UAV series established the engineering vocabulary that unmanned aircraft engineering uses. The historical treatment of unmanned aviation from early proposals through operational deployment is Newcome 2004, the standard reference on the history of the field. This article treats the software-mediation aspect that distinguishes UAV operations from crewed aircraft operations.

The engineering challenge that UAVs present is that the real-time judgment functions traditionally performed by an onboard pilot must be either delegated to a remote pilot connected through a data link or implemented autonomously by onboard software. Remote piloting introduces the latency and reliability constraints of the data-link path. The command-and-control link margin for a remote-piloted UAV must satisfy

$$M_{\text{link}} = P_{\text{tx}} + G_{\text{tx}} + G_{\text{rx}} - L_{\text{path}} - P_{\text{rx,min}} \geq M_{\text{margin}}$$

for transmit power P_{tx} in decibels relative to one milliwatt, transmit and receive antenna gains G_{tx} and G_{rx}, free-space path loss L_{path}, minimum receiver sensitivity $P_{\text{rx,min}}$, and required margin M_{margin} typically 10 to 20 decibels to accommodate weather attenuation and other losses. Autonomous operation introduces the reliability constraints of the onboard software. Both approaches are used in contemporary UAV operations depending on the mission profile, with remote piloting dominating for high-value long-endurance surveillance and strike aircraft including the General Atomics MQ-1 Predator and MQ-9 Reaper and the Northrop Grumman RQ-4 Global Hawk, and autonomous operation dominating for consumer-recreational quadcopters and for defense applications including loitering munitions and swarm-capable systems.

The autonomy level required for an UAV mission profile can be characterized by a taxonomy in the range from purely tele-operated through mission-level autonomy to full autonomous decision-making. Each successive level requires more sophisticated onboard software and produces different safety and reliability trade-offs. The balance point across the autonomy spectrum for a mission depends on the reliability of the data link, the acceptable failure mode

in case of data-link loss, the latency requirements of the mission tasks, and the regulatory constraints of the airspace being operated in.

Autonomous Mission Planning

Autonomous mission planning is the software function that generates an executable mission plan from higher-level mission objectives and constraints. Traditional mission planning is performed by dedicated flight-planning personnel who consult charts, weather forecasts, threat maps, and mission requirements to produce a flight plan that the aircraft crew then executes. Autonomous mission planning delegates all or part of this function to onboard or ground-based software that can generate plans faster and can adapt them in real time to changing conditions.

The algorithmic techniques underlying autonomous mission planning include shortest-path graph algorithms including Dijkstra and A-star, sampling-based motion planning including rapidly-exploring random trees hereafter RRT and probabilistic roadmaps, and mixed-integer linear programming for higher-level assignment and sequencing problems. Each of these techniques has a primary source that established its foundational form. Dijkstra 1959 introduced the shortest-path algorithm that bears his name in Numerische Mathematik. Hart Nilsson and Raphael 1968 introduced the A-star heuristic search algorithm in IEEE Transactions on Systems Science and Cybernetics. LaValle 1998 introduced the rapidly-exploring random tree in a Computer Science Department technical report at Iowa State University that later became the standard reference for the technique. The computational complexity of these algorithms varies widely, with shortest-path algorithms scaling polynomially in the map size and mixed-integer programming scaling exponentially in the number of assignment decisions. For a map with N waypoints, the worst-case complexity of Dijkstra's shortest-path algorithm is

$$T_{\text{Dijkstra}}(N) = O(N^2)$$

with the more efficient A-star variant, when implemented with a Fibonacci-heap priority queue and equipped with an admissible heuristic, achieving worst-case complexity bounded above by

$$T_{\text{A-star}}(N, E) = O(E + N \log N)$$

for a graph with N nodes and E edges, with the practical performance approaching linear in the path length when the heuristic closely estimates the true remaining cost. RRT motion planning is probabilistically complete, meaning that the probability of finding a feasible path in a configuration space of n samples approaches unity as

$$P_{\text{RRT}}(n) \to 1 \text{ as } n \to \infty$$

with convergence rate depending on the structure of the configuration space and the extension heuristic used. More sophisticated combinatorial optimization approaches for multi-vehicle assignment problems reach exponential worst-case complexity that requires heuristic pruning to remain tractable within onboard computational budgets.

The engineering value of autonomous mission planning for aerospace applications is that it permits the mission-plan generation and update to occur at rates matching the operational tempo of dynamic missions, which manual planning cannot economically match. The engineering cost is the considerable software development and verification effort required

to build autonomous planners to the reliability standards that safety-critical operation requires. Autonomous planning for military applications has significant deployment in the 2020s. Autonomous planning for civil-aviation applications remains limited by the regulatory constraints that require human-in-the-loop confirmation of the plans before execution.

The Software-Mediated Aerospace Platform

The cumulative effect of the software substitutions treated in the preceding sections is that contemporary aerospace platforms are markedly software-mediated systems. The fraction of aerospace platform function that is implemented in software rather than in dedicated mechanical or electrical hardware has grown from essentially zero in the 1950s to substantial fractions by the 2020s. For the F-35 Lightning II mentioned in A237 the software fraction of mission capability is variously estimated at 80 to 90 percent, meaning that most of what the aircraft can do is determined by software configuration rather than by hardware capability alone.

The growth trajectory of software fraction of aerospace platform function across the operational history of aviation follows approximately a logistic S-curve

$$f_{\mathrm{sw}}(t) \approx \frac{f_{\mathrm{sat}}}{1 + e^{-r(t - t_{\mathrm{mid}})}}$$

with saturation level f_{sat} approaching but not reaching unity because certain aerospace functions including primary structure, engine thermodynamics, and aerodynamic surfaces necessarily remain physical rather than software artifacts, midpoint year t_{mid} approximately 1995 to 2005 depending on the platform category, and growth-rate parameter r of order 0.05 to 0.10 per year over the 1950 to 2050 window. The engineering, economic, and regulatory implications of software-mediated aerospace platforms remain a subject of active discussion in the industry and the contemporary snapshot article treats them at greater length.

Framework Application to Software-Defined Aerospace

The six-axis framework introduced in A237 applies to software-defined aerospace with axis weightings reflecting the character of the software-substitution transition.

The first axis is numerical computation demand. Software-defined aerospace increases the computation demand of aerospace platforms because the software must implement functions that hardware previously implemented at essentially zero computational cost. Contemporary combat aircraft flight-control computers execute of order 10^9 instructions per second across multiple redundant channels, and mission systems computers execute of order 10^{11} to 10^{12} instructions per second.

The second axis is real-time control. Software-defined aerospace is intrinsically real-time. Fly-by-wire control loops close at rates of 25 to 100 Hertz depending on the control axis. Display update loops close at rates of 30 to 60 Hertz. Software-defined radio processing loops close at sample-rate-derived rates. Each real-time loop must meet its deadlines under all admissible operating conditions.

The third axis is reliability and verification. Software-defined aerospace has driven marked expansion of the safety-critical software engineering discipline treated in A245. The reliability targets for fly-by-wire, glass cockpit, and other software-mediated safety-critical functions are the numerical targets that the discipline was developed to meet, and the verification effort that these targets require has become one of the largest cost drivers on modern aircraft development programs.

The fourth axis is networking and distribution. Software-defined aerospace platforms are internally distributed real-time systems with multiple avionics computers connected by data buses, and increasingly externally distributed through communication links to ground stations, satellites, and other aircraft. The network architectures involved draw on both the aerospace-specific data bus tradition and the broader networking tradition treated in A243.

The fifth axis is software engineering as a discipline. Software-defined aerospace is the application area that produced the modern safety-critical software engineering discipline. The programming languages, development methodologies, verification techniques, and organizational structures that the discipline employs were developed appreciably for aerospace applications and continue to evolve under aerospace-driven requirements.

The sixth axis is semiconductor economics and dual-use. Software-defined aerospace platforms use general-purpose computing hardware adapted for aerospace-specific reliability, radiation, and environmental requirements. The dual-use pattern is that mainstream commercial computing supplies the base capability that aerospace-specific suppliers then adapt at premium prices for the aerospace market, essentially the reverse of the earlier Silicon Valley pattern treated in A246 in which aerospace defense procurement subsidized the manufacturing base that later commercial markets consumed.

Conclusion

Software-defined aerospace and autonomy constitute the late-stage manifestation of the aerospace-computing coupling in which computing capability enables aerospace forms that were previously impossible. Fly-by-wire, glass cockpit displays, software-defined radios, unmanned aerial vehicles, and autonomous mission planning are each technical instances of the general pattern of software substitution for mechanical and hydraulic aerospace systems. The engineering discipline that safety-critical software required for these applications drove extensive development of the discipline itself, and the commercial computing capability that made the software implementations economically feasible reflected the broader trajectory of the computing industry that Silicon Valley and its equivalents produced. Contemporary aerospace platforms are greatly software-mediated systems whose mission capability is determined by software configuration rather than by hardware capability alone, and the trajectory of increasing software mediation continues into the developments that the contemporary snapshot article treats.

The next article in the series takes the contemporary snapshot as of 2026, applies the series framework to the current state of the aerospace-computing co-development pair, treats current forcing pressures including machine learning integration and autonomous swarming, extrapolates forward across the 2026 to 2050 window, and treats competing extrapolation strategies as illustrative alternatives.

References

Books

- Etkin 1996
- Newcome 2004
- Spitzer 2006

Reference

- ARINC 661
- Airbus A320 Fly-by-Wire
- ANSI/JAUS Autonomy Levels

Related Posts

- A237 Framing and the Co-Development Mechanism
- A241 Aerospace Simulation and Real-Time Systems
- A243 ARPANET and Networking Origins
- A244 Space Shuttle Software as Engineering Landmark
- A245 Safety-Critical Software
- A246 Silicon Valley from Defense Contracting

Research

- Dijkstra 1959
- Hart Nilsson and Raphael 1968
- LaValle 1998
- Mitola 1995