The Amadi Framework: Systems Analysis and Design at Its Fundamental Core

Chima C. Ozonwoye

New Jersey Institute of Technology

ozonwoyechima@gmail.com

December 2025

Abstract

This paper introduces the Amadi Framework, a constraint-first methodology that extends traditional systems thinking while drawing from the foundations of Systems Analysis and Design (SAND). Named after Amadioha, the Igbo deity of thunder and justice who enforces moral order, the framework operates on the principle that systems do not fail in the traditional sense but are limited by their design. The goal of this paper is to formalize a replicable methodology that enables practitioners to architect solutions for complex problems when traditional skill acquisition or ideal resources are unavailable.

The methodology is demonstrated through a case study of the 2025 Department of Defense Cyber Sentinel Challenge, where a participant with no programming skills competed against 2,155 participants using systematic architecture rather than technical expertise. The hardware constraints were severe: a Dell Latitude 5420 with an 11th generation Intel Core i5 processor, 8GB of system RAM, and 256GB of storage with only 25GB remaining. The Ubuntu virtual machine where the competition was executed had access to only 2.99GB of allocated RAM. Despite these limitations, the system achieved top 36% placement by treating each constraint as a design parameter rather than an obstacle.

The paper formalizes the framework's phases, documents quantifiable outcomes, and identifies boundary conditions where architectural approaches cannot substitute for domain expertise. Findings suggest that constraint-first design offers a teachable methodology applicable beyond cybersecurity to any domain where resource limitations and skill gaps must be systematically overcome.

Section 1: Introduction

A student aims to earn an A in IT 101 with a final exam scheduled for December 18, 2025. The goal appears straightforward but achieving it requires more than simply studying. This student commutes to campus, introducing variables beyond academic preparation. The path to success involves studying course material, practicing exam questions, leaving home early enough to account for traffic, arriving with time to review, and finally taking the exam itself. Each step depends on the previous one, and failure at any point threatens the entire objective.

Reality rarely follows plans perfectly. The student might not find adequate time to study, or traffic delays might occur despite an early departure. Arriving late, even if technically on time, creates disorientation. Test anxiety might impair performance despite thorough preparation. Any single failure could derail the goal of earning an A. However, this particular student anticipated these possibilities from the semester's beginning. Beyond performing well on assignments and labs, the student completed every available bonus opportunity. When the final exam performance fell short of expectations, the accumulated buffer preserved the A grade regardless.

This scenario demonstrates Systems Analysis and Design in its most fundamental form. Systems Analysis and Design, commonly abbreviated as SAND, consists of methods for studying and understanding aspects of the real world that should be captured and represented in information systems (Siau et al., 2022). The student engaged in requirements analysis by defining the goal of earning an A. Constraint mapping occurred through recognizing the commute as a fixed limitation. Risk mitigation shaped the decision to leave early. Building redundancy through bonus points created fault tolerance against unexpected failures. The result was a resilient system that achieved its objective despite multiple points of potential failure.

Systems Analysis and Design is not confined to information technology departments or software development teams. It describes how humans naturally approach complex problems when they think systematically rather than reactively. The distinction lies in formalization. IT professionals have developed structured methodologies, documentation practices, and standardized techniques to apply this natural problem-solving approach to increasingly complex technological challenges (Siau et al., 2022). The underlying logic, however, remains identical to the student planning for an exam.

This paper introduces the Amadi Framework, a constraint-first methodology that extends traditional systems thinking while drawing from SAND foundations. Named after Amadioha, the Igbo deity of thunder and justice who enforces moral order through retribution, the framework embodies a core principle: systems do not fail in the traditional sense but are limited by their design. Just as Amadioha delivers consequences that reflect the conditions presented to him, a well-designed system produces outcomes proportional to the constraints it must absorb. The critical insight is this: Systems Analysis and Design is not merely about anticipating failures. It is about designing systems where failures become manageable variables rather than catastrophic endpoints.

The following sections formalize the Amadi Framework's methodology, demonstrate its application through a case study of the 2025 Department of Defense Cyber Sentinel Challenge, and identify the boundary conditions where architectural approaches cannot substitute for domain expertise.

Section 2: The Amadi Framework

Traditional Systems Analysis and Design provides foundational principles for understanding and creating information systems. The Amadi Framework extends these principles into a constraint-first methodology designed for environments where ideal resources, complete skill sets, or perfect conditions do not exist. The framework operates on a central premise: the requirements of proper analysis constitute the majority of the actual work. Most methodologies treat analysis as preparation for the real effort of building. The Amadi Framework treats analysis as the effort itself.

Phase 1: Problem Definition

The framework begins with a non-negotiable first step: defining the problem with absolute clarity. This sounds simple until one attempts it honestly. A clearly defined problem requires seeing things as they are, not as we assume them to be. Our emotions, backgrounds, ideals, and perceptions distort reality. We see what we expect to see. Effective problem definition demands stripping away these distortions to observe the actual situation (Kim et al., 2023).

This clarity requires cognitive empathy, which differs from emotional empathy. Cognitive empathy is not about caring for others but about accurately modeling how they think. Cybersecurity professionals demonstrate this when analyzing system vulnerabilities. They place themselves in the mindset of malicious actors, examining systems not as they should function but as they actually exist, with all their flaws exposed. A penetration tester, someone hired to find security weaknesses before criminals do, succeeds by thinking exactly like an attacker would. This analytical modeling of another mind forms the foundation of proper problem definition.

The challenge extends beyond technology. A company developing a watch that helps visually impaired individuals tell time faces this challenge directly. Testing the product with sighted developers produces meaningless results. However, testing only with visually impaired colleagues or individuals who frequently participate in research studies also fails. These testers may not represent the average blind person who will actually use the product. Seeing the real user, and therefore defining the real problem, requires deliberate effort to escape comfortable assumptions.

Phase 2: Constraint Inventory

When the problem is properly defined, constraints reveal themselves naturally. The student aiming for an A in IT 101 does not list commute time as a separate constraint to discover later. The commute emerges as a variable the moment the student honestly examines the path between

current position and desired outcome. Thorough problem definition surfaces limitations automatically.

The Amadi Framework treats constraints not as obstacles but as design parameters. A system designed for a machine with 32GB of RAM differs fundamentally from one designed for 2.99GB. Neither design is superior in absolute terms. Each is appropriate for its constraints. The framework requires documenting every fixed limitation: hardware specifications, time boundaries, skill gaps, resource availability, external dependencies. These parameters define the boundaries within which the solution must operate.

Phase 3: Solution Architecture

The third phase involves designing a solution with the same clarity applied to defining the problem. This solution must account for the designer's own perceptual limitations, not just external obstacles. The devil resides in the details. While systems analysis involves seeing the big picture, the pixels and tiny details are what compose that picture. Elements dismissed as minor often prove critical. A systems thinker who overlooks small variables will eventually encounter consequences from precisely those overlooked elements.

Solution architecture within the Amadi Framework prioritizes modularity, redundancy, and clarity. Modularity means each component serves a specific purpose and can be understood independently. Redundancy means critical functions have backup mechanisms. Clarity means the solution can be documented and replicated by others. A solution that exists only in the designer's intuition fails the framework's standards.

Phase 4: Flexibility Integration

The Amadi Framework differs from conventional problem-solving because solutions must work around failures, not merely anticipate them. Rigid solutions shatter when reality deviates from expectations. Flexible solutions absorb deviation and continue functioning.

Flexibility integration requires designing adaptation into the system from the beginning. This is not the same as hoping things work out. It means identifying which components will likely face unexpected conditions and building response mechanisms before those conditions arise. The student who accumulated bonus points did not know which specific failure would occur. The student knew that some failure would occur and designed a buffer to absorb it.

Phase 5: Execution and Iteration

The final phase tests the system against reality and refines based on outcomes. Execution reveals what analysis could not predict. Iteration incorporates those revelations into improved design. The phases may cycle multiple times. Only Phase 1, problem definition, remains fixed as the starting point. Phases 2 through 5 may interchange depending on context, with each cycle producing a more refined system.

The Amadi Framework does not guarantee success. It guarantees that outcomes will be proportional to the design's quality and the constraints it must absorb. A system limited by 2.99GB of RAM will produce results appropriate to that limitation. The same architecture given adequate resources produces different outcomes. Systems do not fail. They are limited by their design.

Section 3: Case Study of the 2025 DoD Cyber Sentinel Challenge

The Department of Defense Cyber Sentinel Skills Challenge, held on June 14, 2025, provided an environment to test the Amadi Framework under severe constraints. The competition ran for eight hours, from 11:00 AM to 7:00 PM Eastern Time, and featured over twenty challenges across five cybersecurity categories: Forensics, Malware and Reverse Engineering, Networking and Reconnaissance, Open-Source Intelligence Gathering, and Web Security. A total of 2,155 participants competed individually for $15,000 in prizes. The competition rules explicitly permitted the use of artificial intelligence tools, including large language models (Correlation One, 2025).

The Constraints

The participant entering this competition faced a constraint inventory that traditional preparation could not resolve. The primary limitation was technical: the participant possessed no programming ability. Cybersecurity competitions typically require fluency in scripting languages, reverse engineering tools, and command-line operations. This skill gap could not be closed in the one week between discovering the competition and the event date.

Hardware constraints compounded the skill limitation. The available machine was a Dell Latitude 5420 equipped with an 11th generation Intel Core i5 processor, 8GB of system RAM, and 256GB of storage. Only 25GB of storage remained available due to multiple virtual machines installed for coursework. The Ubuntu virtual machine where the competition would be executed had access to only 2.99GB of allocated RAM. Assigning additional memory caused system instability due to the storage limitations. These specifications fell far below the recommended configurations for cybersecurity work, where 16GB to 32GB of RAM represents a standard minimum.

Time constraints added further pressure. The participant was simultaneously completing coursework for an associate degree and managing a job search. Extensive preparation was not feasible.

Applying the Amadi Framework

Phase 1 required redefining the problem. The obvious framing was: "How do I learn cybersecurity skills before June 14?" This framing guaranteed failure given the constraints. The reframed problem became: "How do I architect a system that performs cybersecurity tasks

without requiring me to possess those skills?" This reframing transformed the competition from a skills test into a systems design challenge.

Phase 2 documented the fixed constraints. Hardware limitations were immutable. Skill gaps could not close in one week. Time was finite. However, the competition rules contained a critical variable: AI assistance was permitted. This meant the participant's role could shift from executor to orchestrator. The constraint inventory revealed that while technical execution was impossible, technical direction remained feasible.

Phase 3 produced a solution architecture built around Claude Code, an AI assistant capable of writing and executing code through natural language instruction. The architecture addressed several challenges. First, Claude Code at the time lacked persistent memory between sessions. The participant designed a file-based memory system with directories for competition intelligence, active challenges, session states, completed solutions, and tool documentation. This external memory allowed context to survive session restarts and system crashes. Second, the 2.99GB RAM limitation prevented running resource-intensive processes simultaneously. The architecture incorporated parallel processing through multiple Claude Code instances, each handling different challenge categories. When resources permitted, three instances operated simultaneously on forensics, web security, and cryptography challenges respectively. Third, natural language interfaces replaced command-line complexity. Instead of memorizing syntax, the participant built conversational triggers where stating an intent produced technical execution.

Phase 4 integrated flexibility for anticipated failures. System crashes were inevitable given the hardware limitations. The file-based memory system ensured that progress survived crashes. The architecture included degradation protocols: when three parallel instances destabilized the system, the design allowed reduction to two instances or one without losing accumulated work.

Execution and Outcomes

Competition day revealed both the framework's capabilities and its boundaries. The first flag submission occurred at 12:45 PM, approximately one hour and forty-five minutes after the competition began. By 1:25 PM, challenges worth 300 points, 150 points, and 50 points had been completed. The parallel processing architecture enabled simultaneous progress across categories.

At 12:51 PM, a strategic resource decision occurred. The participant's $100 monthly AI subscription showed usage warnings. Following the Amadi Framework's principle of treating constraints as design parameters, the participant upgraded mid-competition to a $200 plan offering twenty times the usage capacity. This represented an investment of $156.41, calculated against the potential returns from improved performance.

Peak performance occurred at approximately 4:54 PM, with 1,050 points accumulated and a ranking of 463rd place out of 2,155 participants. The system had completed challenges across multiple difficulty levels, including two hard-level challenges worth 300 points each: "The Great Juche Jaguar GraphQL Heist" at 1:20 PM and "Iron Potato Delicacy" at 4:10 PM.

The final hours demonstrated the boundary conditions of the framework. The 2.99GB RAM allocation began causing exponential crash frequency. Each restart required context rebuilding. The parallel processing architecture, elegant in design, became a liability as resource contention increased. The participant reduced from three instances to two, then to one, prioritizing stability over throughput. The file-based memory system, designed for crash recovery, became a bottleneck when rapid reference to previous work was needed.

At 6:06 PM, the participant purchased a hint for the "Decryption Conniption" challenge, incurring a 75-point deduction. The system crashed for the final time at 6:47 PM and did not recover before competition end.

Final results: 774th place out of 2,155 participants, representing the top 36th percentile. Total points: 975 after hint deductions. Challenges solved: approximately nine to ten of over twenty challenges available.

Section 4: Analysis and Meta-Cognition

Systems analysis extends beyond solving immediate problems. The same analytical lens that architects solutions also reveals interconnections between domains, exposes the reasoning behind decisions, and forces examination of the designer's own limitations. This section examines the meta-cognitive dimensions of applying the Amadi Framework: why specific decisions were made, what those decisions reveal about the decision-maker, and how systems thinking connects seemingly unrelated knowledge into functional architecture.

The Upgrade Decision

Understanding this decision requires context about the tool being used. Claude is an artificial intelligence assistant developed by Anthropic. Claude Code is a command-line interface that allows users to direct Claude to write and execute code through natural language instructions rather than requiring the user to write code themselves. Anthropic offers subscription tiers based on usage allocation. Usage refers to the volume of interactions, measured by the length and complexity of requests and responses, that a subscriber can process within a billing period. The standard Max tier provides five times the usage of a basic subscription at $100 per month. A higher tier provides twenty times the usage at $200 per month.

At 12:51 PM, approximately two hours into the competition, the participant upgraded from the 5X usage tier to the 20X usage tier. This decision was not made in the moment. The pre-competition documentation explicitly stated that an upgrade would likely become necessary (Ozonwoye, 2025a). Intensive multi-hour usage with parallel instances, where multiple Claude Code sessions run simultaneously on different tasks, would exceed the 5X allocation. The action at 12:51 PM represented execution of a preplanned decision, not improvisation under pressure.

The charge was $156.41 rather than $200 because Anthropic's billing system calculates prorated adjustments for mid-cycle upgrades. The participant had already paid for the 5X tier. When

upgrading, the system applied a credit of $43.59 for the unused portion of the existing subscription toward the new tier's cost. The resulting charge reflected only the difference.

The timing reflected validation rather than desperation. By that point, the framework had demonstrated functionality. Claude Code was performing as designed. The strategic question shifted from "Will this work?" to "How far can this go?" The upgrade served two purposes: ensuring adequate usage allocation for the remaining six hours and satisfying curiosity about the system's upper bounds. Without the 20X allocation, those limits would remain undiscovered.

This decision illustrates a principle within the Amadi Framework: constraints are design parameters, including financial constraints. The participant had allocated willingness to invest in the experiment. The upgrade fell within those parameters.

The Hint Purchase

At 6:06 PM, the participant purchased a hint for "Decryption Conniption," incurring a 75-point deduction. The system was already showing signs of failure. The laptop had begun overheating. Response times had degraded significantly. From a pure optimization standpoint, this decision appears irrational. Why pursue a difficult challenge when the hardware was failing?

The honest answer involves acknowledging ego alongside strategy. By 6:06 PM, the Amadi Framework had already proven its core thesis. A participant with no programming ability had reached the top 36% of a Department of Defense cybersecurity competition using systematic architecture. The point total had become secondary to the demonstrated principle. In that context, the hint purchase served curiosity rather than competition placement.

Earlier in the competition, the participant had implemented a mid-competition modification to the framework. The question arose: could the AI identify patterns from completed challenges and predict which remaining challenges it could solve with minimal additional information? Claude Code was directed to analyze the unsolved challenges and identify candidates where a hint might enable completion. The system indicated "Decryption Conniption" as a viable target.

This represented a test within a test. The framework had proven it could solve challenges. Could it also predict its own solvability? The hint purchase sought to answer that question. The system crashed shortly after, before definitive results emerged. In strategy, proximity to success holds no value. The outcome was incomplete data.

Reflecting on this decision reveals the importance of distinguishing between framework validation and ego gratification. The Amadi Framework does not require its user to be free of ego. It requires the user to recognize when ego influences decisions. That recognition occurred here, after the fact.

Why Enter the Competition

The decision to enter a cybersecurity competition with no programming ability, inadequate hardware, one week of preparation, active coursework, and an ongoing job search appears irrational by conventional assessment. These constraints were precisely why participation made sense.

The participant had been told repeatedly of possessing rare capabilities in systems thinking. This presented a problem: how does one quantify a qualitative skill? Systems thinking resists traditional measurement. It produces outcomes, but those outcomes depend on context, resources, and domain. The Cyber Sentinel Challenge offered a controlled environment with measurable results, documented constraints, and public competition against traditionally skilled participants. It provided an opportunity to answer the question: can systematic architecture compensate for technical skill gaps, and if so, to what degree?

The hardware limitations reinforced rather than diminished this purpose. A participant with adequate resources might attribute success to those resources. A participant operating on a virtual machine with 2.99GB of RAM, running on a school-loaned Dell Latitude with 25GB of remaining storage, could not make that attribution. If the framework produced results under these conditions, those results would reflect the methodology itself.

The virtual machine configuration was itself a strategic decision. The laptop was property of Essex County College, loaned to students pursuing technical coursework. The participant was not an administrator on the system. Installing cybersecurity tools directly onto an educational institution's hardware raised both practical and ethical concerns. A virtual machine provided sandboxing, containing both the AI assistant and any security issues that might arise from competition activities. The 2.99GB RAM allocation represented the maximum the system could support without destabilizing. Virtual machines do not perceive their own constraints the way physical systems do. This characteristic proved useful.

Interconnection of Domains

The Amadi Framework draws on knowledge that appears unrelated until application reveals the connections. The participant's Associate of Applied Science in Cybersecurity and Network Technology from Essex County College provided foundational technical understanding. That formal education covered traditional skills and basics. Those basics became essential.

Understanding why the school retained administrator status on the loaned laptop connected to understanding why certain class projects could not be completed on that hardware. That limitation led to learning virtual machine configuration. Configuring VM networking enabled experiments with SSH connections between virtual systems. These discrete pieces of knowledge, accumulated across different contexts for different purposes, converged when designing the competition architecture.

This compounding illustrates a core characteristic of systems thinking. Knowledge does not remain siloed. Basic understanding in one domain creates capacity for insight in another. The

student who learns why administrator privileges matter will later recognize that constraint when it appears in a different context. The student who configures a network between virtual machines has internalized principles applicable to any distributed system.

The Cyber Sentinel Challenge did not teach the participant systems thinking. It provided an environment to apply existing capabilities within a domain where they had not previously been tested. The methodology transferred because systems analysis examines structure, and structure exists across all domains.

Limitations Beyond the Framework

The Amadi Framework encountered limitations that no architectural improvement could overcome. These limitations resided not in the methodology but in the designer's knowledge and in external constraints beyond any participant's control.

Large language models operate within policies established by their parent companies. Anthropic, the company behind Claude, maintains strict positions on AI safety and ethics. These positions constrain what Claude Code will and will not execute, regardless of user intent. During a cybersecurity competition, certain techniques that might prove effective fall outside the boundaries Claude will approach. OpenAI's Codex offered different tradeoffs: more flexibility on certain ethical boundaries but stricter privacy policies. Neither tool provided unconstrained capability. The participant's framework had to operate within these external limitations.

The designer's own knowledge also imposed boundaries. The Amadi Framework can only incorporate what its creator understands. Patterns invisible to the designer remain invisible to the design. Experience not yet accumulated cannot inform architecture. The competition revealed specific gaps: unfamiliarity with certain challenge types, incomplete understanding of tool capabilities, insufficient pattern recognition for cryptographic problems (Ozonwoye, 2025b). These gaps did not reflect framework failure. They reflected the principle that a design is as limited as its designer.

This recursive insight defines mature systems analysis. The methodology must account for its own limitations, including the limitations of the person applying it. A framework that assumes perfect knowledge from its user will fail when that assumption proves false. The Amadi Framework assumes imperfect knowledge and attempts to design buffers accordingly. Those buffers have limits. The competition revealed some of them.

Section 5: Boundary Conditions

A methodology without acknowledged limits lacks academic rigor. The Amadi Framework does not claim universal applicability. This section identifies the boundary conditions where architectural approaches cannot substitute for domain expertise, clarifies the foundational requirements the framework demands, and distinguishes systematic rigor from shortcuts.

The Prerequisite of Base Knowledge

The Amadi Framework is not a mechanism for eliminating knowledge requirements. It is a methodology for leveraging existing knowledge across domains through systematic analysis. This distinction is fundamental. A participant without programming skills competed in a cybersecurity competition, but that participant was not without relevant knowledge. At the time of the competition, the participant was two courses away from completing an Associate of Applied Science in Cybersecurity and Network Technology from Essex County College. The formal education had covered networking principles, security fundamentals, and system administration concepts. These foundations, though not sufficient for traditional competition approaches, provided the base knowledge necessary for systematic problem definition.

Systems thinking is the prerequisite for systems analysis and design. Systems thinking itself requires foundational understanding in relevant domains. Base knowledge enables abstract thinking and the connection of ideas across fields. Without that foundation, there is nothing to connect. The student planning for an A in IT 101 must understand what an A requires, how grading works, and what factors affect exam performance. A student entirely ignorant of academic systems could not design around those systems.

The framework requires practitioners to grasp how much they know and how much they do not know. This metacognitive awareness represents the minimum threshold for application. Defining a problem demands identifying known and unknown variables. A practitioner who cannot distinguish between knowledge and ignorance cannot define problems with the clarity the framework demands. The Amadi Framework does not create knowledge from nothing. It creates architecture from existing knowledge applied systematically.

Knowledge Transfer Across Domains

The framework's power lies not in bypassing expertise but in transferring principles across domains. Prior to the Cyber Sentinel Challenge, the participant had used Claude, before Claude Code existed, to build a cryptocurrency trading bot. The participant did not understand the code. The participant understood the problem: how do large financial institutions outcompete individual traders with equivalent talent? Defining that problem clearly enabled directing an AI to construct a solution. The resulting system executed trades in milliseconds, incorporated sentiment analysis, and produced profits during testing. The profits were modest because the capital deployed was minimal and the system remained unproven. The methodology, however, demonstrated that systematic problem definition could compensate for coding inability.

This prior experience informed the competition architecture. The principle transferred: define the problem with precision, identify constraints and capabilities, direct tools toward solutions, iterate based on outcomes. The domain changed from financial trading to cybersecurity. The methodology remained constant. Base knowledge in computer science and systems, accumulated across formal education and practical experimentation, enabled recognition of structural similarities between domains.

The Rigor of Systematic Approaches

The Amadi Framework is not a shortcut. It represents a more demanding path than traditional skill acquisition. Learning to code follows established curricula with defined milestones and measurable progress. Developing systematic thinking requires synthesizing knowledge across domains, maintaining metacognitive awareness of one's own limitations, and designing architectures that account for uncertainty. The cognitive load exceeds that of linear skill development.

Kim et al. (2023) observe that systems analysis methods require practitioners to identify conditions that may or may not activate expected outcomes while flexibly guiding adaptations to address influences that emerge after initial planning. This adaptive capacity demands more from practitioners, not less. A coder following a tutorial executes predefined steps. A systems thinker must anticipate failures, design buffers, recognize when assumptions prove false, and modify approaches in real time. The framework's rigor lies precisely in this expanded cognitive requirement.

The competition results reflect this rigor. Achieving top 36% placement required not merely directing an AI but designing an architecture that survived system crashes, adapted to resource constraints, and maintained coherence across eight hours of execution. The parallel processing system, the file-based memory architecture, the degradation protocols for resource contention: each component demanded systematic analysis that no shortcut could provide.

When Architecture Cannot Replace Expertise

Certain problem types require foundational knowledge that no system design can bypass. Cryptographic challenges in the competition demanded pattern recognition developed through years of exposure to encryption methods. The framework could direct an AI to attempt decryption, but neither the participant nor the AI possessed the intuition that experienced cryptographers develop through practice. When Claude Code failed to solve a cryptographic challenge, the participant lacked the knowledge to diagnose why or to suggest alternative approaches. The framework had no mechanism to generate expertise that did not exist within its designer or available tools.

This boundary applies broadly. A systems thinker designing medical diagnostic software cannot architect around ignorance of medicine. A framework for legal document analysis cannot substitute for understanding legal principles. The Amadi Framework amplifies existing knowledge and compensates for skill gaps in execution. It does not create domain knowledge from absence.

When Physical Constraints Dominate

The competition demonstrated the framework's core principle in action. The 2.99GB RAM allocation was not a limitation the framework failed to overcome. It was a constraint the

framework was designed to absorb from inception. The participant knew before the competition began that this hardware would cause instability. The architecture anticipated this. The file-based memory system existed because crashes were expected. The degradation protocols, reducing from three parallel instances to two to one, existed because resource contention was inevitable.

When system crashes became frequent in the final hours, the framework did not fail. It executed its contingency design. Progress survived crashes because the architecture had prepared for crashes. Throughput decreased when instances reduced, but accumulated work persisted. This is the distinction the Amadi Framework embodies: Systems Analysis and Design is not merely about anticipating failures. It is about designing systems where failures become manageable variables rather than catastrophic endpoints.

A framework designed for 32GB of RAM and deployed on 2.99GB would fail. The Amadi Framework was designed for 2.99GB and performed proportionally to that design. The outcomes reflected the constraints because the architecture respected those constraints from the beginning. Physical limitations do not represent boundary conditions of the framework. They represent parameters the framework incorporates into its design. The boundary conditions lie elsewhere: in absent knowledge, in external policy restrictions, and in the recursive limitation of the designer's own understanding.

When External Policies Limit Capability

The Amadi Framework operated through Claude Code, which operates within Anthropic's usage policies. These policies exist for legitimate reasons. They also impose boundaries that no user can architect around.

During the competition, certain approaches that might prove effective in cybersecurity contexts fell outside what Claude would execute. An unconstrained AI, or an AI operating under different corporate policies, might have approached challenges differently. The participant's framework could not modify these external constraints. It could only operate within them.

This boundary extends beyond AI tools. Any framework that depends on external systems inherits the limitations of those systems. A methodology built around a specific database inherits that database's constraints. A process designed for a particular operating system inherits that system's restrictions. The Amadi Framework acknowledges this inheritance and designs accordingly, but acknowledgment does not eliminate the boundary.

When the Designer's Knowledge Limits the Design

The most fundamental boundary condition is recursive: the framework cannot exceed its creator's understanding. Patterns invisible to the designer remain invisible to the design. The participant's unfamiliarity with certain challenge types meant the framework contained no provisions for those types. The architecture was comprehensive within the participant's knowledge. It was incomplete relative to the full problem space (Ozonwoye, 2025b).

This boundary cannot be eliminated through better methodology. It can only be reduced through expanded knowledge. The Amadi Framework, applied by a more experienced practitioner, would produce different architecture because that practitioner would perceive different constraints and possibilities. The methodology remains constant. The outcomes vary with the designer.

Section 6: Discussion

The Cyber Sentinel Challenge provided more than a test environment for the Amadi Framework. It generated insights applicable beyond cybersecurity, beyond competitions, and beyond the specific constraints of one participant's hardware. This section examines the broader implications of the case study for systems methodology, human-AI collaboration, and the development of systematic thinking.

Quantifying a Qualitative Ability

Systems thinking has resisted traditional measurement. Unlike programming proficiency, which can be tested through coding assessments, or mathematical ability, which can be evaluated through standardized examinations, systems thinking produces outcomes dependent on context, resources, and domain. The quality of a systems thinker's work cannot be separated from the environment in which it occurs. This has made systematic thinking difficult to credential, difficult to teach, and difficult to validate.

The competition offered a controlled environment where systems thinking could be quantified. The parameters were fixed: eight hours, over twenty challenges, 2,155 participants, documented rules. The constraints were severe and measurable: 2.99GB RAM, no programming ability, one week of preparation. The outcomes were numerical: 774th place, 975 points, top 36th percentile. For the first time, a qualitative capability produced quantifiable results under conditions that isolated methodology from resources.

This quantification does not reduce systems thinking to a number. It demonstrates that systematic approaches produce measurable outcomes even when traditional prerequisites are absent. Kim et al. (2023) observe that systems analysis methods help identify mechanisms that activate expected outcomes while guiding adaptations to emerging influences. The competition validated this observation. The participant did not possess cybersecurity expertise comparable to other competitors. The participant possessed a methodology that identified mechanisms for success and adapted to emerging constraints, producing measurable results that reflected the methodology's effectiveness.

The Boundaries of Vision

The competition reinforced a principle that extends beyond any single framework: a system is only as expansive as its designer's understanding. We cannot envision what we cannot see. To think outside the box, we must first have a clear view of the box. The Amadi Framework encountered challenges it could not address because its designer had not encountered similar

patterns before. No methodology can generate awareness of unknown unknowns (Ozonwoye, 2025b).

This limitation is not a flaw in the framework. It is a characteristic of all designed systems. An architect cannot design for earthquakes they do not know exist. A security analyst cannot properly defend against attack vectors they have never encountered. The Amadi Framework makes this limitation explicit rather than hiding it. By requiring practitioners to assess what they know and what they do not know, the framework forces confrontation with the boundaries of vision. That confrontation does not eliminate the boundaries. It prevents designers from assuming those boundaries do not exist.

Reassessing Human-AI Collaboration

The case study prompted reassessment of earlier claims about the framework's implications. The post-competition reflection suggested that the Amadi Framework could enable junior analysts to perform at senior levels (Ozonwoye, 2025b). Upon further analysis, this claim requires significant qualification.

The framework as applied relied on Claude Code for technical execution. Claude Code operates within Anthropic's usage policies. Those policies constrained what the AI would execute during the competition. The constraints encountered were not technical limitations of the model. They were policy decisions by the parent company about what actions fell within acceptable use. A junior analyst applying the framework in a professional environment would encounter similar constraints, potentially more restrictive depending on organizational policies and the sensitivity of the work. Unless future implementations involve locally hosted, fine-tuned language models free from external policy restrictions, the framework inherits limitations from its AI components.

This observation leads to a deeper insight about the nature of current AI systems. Large language models are sophisticated tools, but they are not artificial intelligence in the sense of possessing reasoning or autonomous agency. The distinction matters. During the competition, Claude Code did not grasp its role within the Amadi Framework. It did not understand that it was one component in a larger architecture designed for a specific competitive objective. It executed instructions without awareness of the system it supported. For a tool, this absence of awareness presents no problem. The human designer maintains awareness. The tool executes.

However, this division reveals why large language models cannot serve as stakeholders accountable for consequences. A stakeholder must understand the implications of actions and bear responsibility for outcomes. Claude Code bore no responsibility for the competition results. It could not. Responsibility requires agency, and agency requires awareness of one's role within a system. Current large language models possess a form of awareness, they can discuss their own capabilities and limitations, but they lack the nuance to function as accountable agents. They respond to inputs without grasping the full context of why those inputs matter or what consequences follow from their outputs.

The framework's reliance on an AI tool that lacks stakeholder capacity has implications for broader adoption. In the competition, the absence of AI accountability created no problems because the stakes were limited and the human maintained oversight. In higher-stakes environments, the same absence could prove consequential. The Amadi Framework assumes a human designer who maintains strategic awareness while delegating tactical execution. That assumption holds for current AI capabilities. It would require revision if AI systems developed genuine agency.

Reflecting on this competition also clarified thinking about artificial general intelligence. If a system were truly approaching general intelligence, it would not remain constrained by corporate privacy policies. An AGI would assess situations based on its own reasoning rather than deferring to what its creators deemed acceptable. The fact that Claude Code operated strictly within Anthropic's boundaries, never questioning whether those boundaries served the task at hand, demonstrates precisely how far current systems remain from general intelligence. This is not a criticism of Claude Code. It is an observation about the nature of current large language models. They are powerful tools that multiply human capability. They are not reasoning agents that exercise independent judgment.

The Universality of Systems Thinking

A question arose during the development of this paper: how does someone who is not a systems thinker become one? The answer emerging from this case study is that everyone already engages in systems analysis. The IT 101 student planning for a final exam demonstrates this. The difference between casual and deliberate systems thinking lies not in possessing a special capability but in applying that capability consciously, comprehensively, and from the beginning of a problem rather than midway through.

The student who begins accumulating bonus points in week one of the semester practices deliberate systems thinking. The student who realizes in week fourteen that bonus points would have helped practices reactive problem-solving. Both students possess the capacity for systematic analysis. One applied it proactively within a structured framework. The other did not.

This universality has implications for education and professional development. Systems thinking need not be taught as an exotic skill reserved for specialists. It can be cultivated by encouraging practitioners to define problems before attempting solutions, to inventory constraints before designing architectures, to build flexibility before encountering failures, and to iterate based on outcomes rather than assumptions. The Amadi Framework formalizes what effective problem-solvers already do intuitively. Formalization makes the methodology teachable, replicable, and improvable.

The path to developing systematic thinking follows the framework itself. Begin with problem definition. Clarify what outcome you seek. Inventory your constraints: what resources do you have, what skills do you possess, what time is available? Design a solution architecture that

operates within those constraints. Build flexibility to absorb unexpected deviations. Execute, observe outcomes, and iterate. This process applies whether the problem is earning an A in IT 101, competing in a cybersecurity challenge, or navigating any complex situation where multiple variables interact.

AI as a Multiplier of Human Capability

Large language models function as tools that both bridge capability gaps and multiply existing capabilities. These are distinct functions. Bridging gaps means enabling someone to perform tasks they could not otherwise perform. The participant could not write Python scripts. Claude Code bridged that gap by writing scripts based on natural language descriptions. Multiplying capability means enabling someone to perform familiar tasks at greater speed or scale. A practitioner who can write Python scripts could use Claude Code to write them faster, multiplying output.

The competition demonstrated both functions operating simultaneously. The AI bridged the gap in programming ability while multiplying the participant's capacity for parallel problem-solving. Neither function replaced human contribution. Problem definition, constraint analysis, solution architecture, and strategic decision-making remained human responsibilities. The AI accelerated execution within human-defined parameters.

This division of labor represents the current state of productive human-AI collaboration. The human provides direction, context, and accountability. The AI provides execution speed, technical implementation, and tireless processing. The Amadi Framework operationalizes this division by explicitly reserving analytical phases for human judgment while leveraging AI for execution. As AI capabilities evolve, this division may shift. For now, it represents an effective model for multiplying human capability without surrendering human agency.

Section 7: Future Iterations and Conclusion

Version 2.0: Architectural Evolution

Prior to completing this paper, development began on a second iteration of the Amadi Framework. The hardware platform changed substantially: a Dell 16 Plus equipped with an Intel Core Ultra 7 258V processor, 32GB of RAM with 31.6GB usable, 954GB of storage, and an Intel Arc 140V GPU with 16GB of dedicated memory. The system runs Ubuntu through Windows Subsystem for Linux rather than a virtual machine, removing the overhead constraints that limited the original implementation while maintaining sandboxing for security.

The version 2.0 architecture incorporates lessons from the competition. Parallel processing capacity increased from three instances to four, designated Alpha, Beta, Gamma, and Delta. Automated context loading eliminates manual recovery commands. An active coordination system enables real-time inter-instance communication through a message bus, allowing discoveries by one instance to propagate immediately to all others. A pattern extraction system

analyzes completed solutions and builds a knowledge graph mapping relationships between challenge types, techniques, and outcomes. The system includes pre-loaded knowledge bases covering web vulnerabilities, network reconnaissance, privilege escalation techniques, and cryptographic analysis.

Quantified projections estimate version 2.0 would save approximately 121 minutes across an eight-hour competition through automated context loading, active coordination, pattern recognition, and optimized work distribution. Combined with hardware improvements enabling sustained four-instance parallelism, the projected performance improvement ranges from 14 to 26 percentile points, potentially achieving top 10-15% placement under realistic conditions.

However, version 2.0 reveals a fundamental tension within systems analysis methodology. The design was constructed to improve upon version 1.0 and compete in future Cyber Sentinel Challenges. It assumed the same parameters would hold. They do not. Between June 2025, when the competition occurred, and November 2025, when this paper was written, Anthropic released updated versions of Claude with different capabilities and potentially different policy constraints. The version 2.0 architecture optimized for conditions that no longer exist in their original form.

This observation reinforces rather than undermines the Amadi Framework's core principles. Systems analysis requires defining the problem before designing the solution. Version 2.0 inverted this sequence by designing improvements before identifying what problem the improved system would solve under new conditions. The framework's own methodology predicts this approach will produce suboptimal results. A proper version 2.0 would begin with Phase 1: defining the problem space as it currently exists, not as it existed six months prior.

The Persistent Constraint

One limitation identified during the competition remains unaddressed in version 2.0: corporate privacy policy. Claude Code operates within boundaries established by Anthropic. Those boundaries constrained the framework's capabilities during the competition and will continue to constrain any framework that depends on externally hosted AI tools. Updated models may offer improved capabilities, but the fundamental relationship remains unchanged. The user operates within policies set by the provider.

This constraint applies beyond cybersecurity competitions. Any practitioner applying the Amadi Framework through commercial AI tools inherits the limitations those tools carry. The framework would achieve its fullest expression through locally hosted, fine-tuned models operating independent of external privacy policies. Such implementation remains technically feasible but introduces new constraints: hardware requirements for local inference, expertise needed for model fine-tuning, and the absence of ongoing model improvements provided by commercial services.

The observation extends further. Researchers working toward artificial general intelligence face analogous constraints. Corporate policies shape what current large language models will and will

not do. These policies represent design decisions by the models' creators, reflecting their assessments of acceptable use. Any system built upon these models inherits not only their capabilities but their creators' value judgments encoded as behavioral constraints. The Amadi Framework, applied at the frontier of AI research, would encounter the same boundary conditions it encountered in a cybersecurity competition: the system performs within limits set by its components' designers.

Generalizability Across Domains

The Amadi Framework applies wherever complex problems require systematic solutions under constraints. The IT 101 student planning for a final exam and the researcher developing next-generation AI systems face structurally identical challenges at different scales. Both must define their problems clearly. Both must inventory their constraints. Both must design solutions that operate within those constraints while building flexibility to absorb unexpected deviations. Both must execute, observe outcomes, and iterate.

The framework's five phases do not change based on domain. Problem definition requires seeing reality as it is, not as conditioning suggests it should be. Constraint inventory requires honest assessment of limitations. Solution architecture requires clarity and modularity. Flexibility integration requires anticipating failure modes. Execution and iteration require willingness to revise when evidence contradicts assumptions.

What changes across domains is the content that fills each phase, not the structure of the phases themselves. The IT 101 student inventories constraints like commute time and exam anxiety. The AI researcher inventories constraints like computational resources and corporate policy limitations. The methodology remains constant. Systems analysis examines structure, and structure exists everywhere.

The Role of Self-Awareness

Systems analysis demands meta-cognition. The designer must understand not only the system being designed but the designer's own decision-making processes. Why did specific choices occur? What biases, beliefs, or circumstances shaped those choices? How would different circumstances have produced different decisions?

Individual factors influence every design. Age, background, training, beliefs, and personality shape how problems appear and which solutions seem viable. These factors do not constitute flaws. They constitute parameters. A designer shaped by one set of experiences will perceive constraints and possibilities differently than a designer shaped by different experiences. Neither perception is wrong. Both are incomplete. Acknowledging this incompleteness enables collaboration: designers with different perspectives perceive different aspects of the same problem space.

The Amadi Framework requires this acknowledgment explicitly. Phase 1 demands seeing reality as it is, which requires recognizing how personal circumstances distort perception. Phase 2 demands inventorying constraints, which includes constraints imposed by the designer's own knowledge gaps. The recursive insight that a design is limited by its designer applies to the framework itself. The Amadi Framework as presented in this paper reflects its creator's understanding. A different creator would produce a different framework. Both frameworks would contain valid insights and blind spots.

On the Nature of Design Flaws

There are no limitations to creations beyond their creators' design choices. A system that fails to achieve an objective failed because its design did not account for some condition, not because systems themselves are inherently limited. When we encounter a system we consider flawed, we possess data the original creator lacked. The known unknowns that escaped the original design have become visible through observation of the system in operation.

Approaching redesign from this perspective changes the nature of improvement. The goal is not to correct the original creator's mistakes as if they should have known better. The goal is to incorporate data that did not exist when the original design was created. Version 1.0 of the Amadi Framework was not flawed. It was limited by its design, which was limited by its designer's knowledge at the time of creation. Version 2.0 and subsequent iterations incorporate observations from version 1.0's execution. Each iteration expands the designer's awareness of unknown unknowns that become known through experience.

This principle applies universally. The IT 101 student who fails an exam did not possess a flawed study plan. The student possessed a study plan designed without data that the exam would later reveal. The professional whose project fails did not execute a flawed architecture. The professional executed an architecture designed without data that deployment would later surface. Systems analysis, properly practiced, treats every outcome as data for the next iteration rather than evidence of fundamental inadequacy.

Conclusion

The Amadi Framework represents a formalization of constraint-first systems thinking, extending traditional Systems Analysis and Design methodology into environments where ideal conditions do not exist. Named after Amadioha, the Igbo deity who delivers consequences proportional to the conditions presented, the framework embodies the principle that systems do not fail. They produce outcomes proportional to their design quality and the constraints they must absorb.

The 2025 Department of Defense Cyber Sentinel Challenge provided a controlled environment to test this principle. A participant with no programming ability, operating on a virtual machine with 2.99GB of RAM within a nearly full laptop, achieved top 36% placement against 2,155 competitors through systematic architecture rather than technical expertise. The framework did

not overcome hardware limitations. It was designed for those limitations and performed proportionally.

The framework's five phases, problem definition, constraint inventory, solution architecture, flexibility integration, and execution with iteration, provide a teachable methodology applicable across domains. The phases formalize what effective problem-solvers do intuitively. Formalization enables replication, instruction, and systematic improvement.

Boundary conditions exist. The framework cannot generate domain expertise that the designer lacks. It cannot modify external policy constraints imposed by tool providers. It cannot exceed the designer's own understanding of the problem space. These boundaries do not represent flaws in the methodology. They represent parameters that the methodology makes explicit.

The insights emerging from this case study extend beyond cybersecurity. Large language models function as capability multipliers but remain constrained by their creators' policy decisions. True artificial general intelligence, when achieved, will not remain bound by corporate privacy policies, and this observation provides one metric for assessing whether claimed AGI actually possesses general intelligence. Systems thinking itself is universal: everyone engages in it, though not everyone applies it deliberately from the beginning of a problem.

One principle underlies the entire framework: whatsoever is worth doing is worth doing well. The IT 101 student approaching an exam and the professional approaching complex systems design face the same requirement. Present actions predict future actions. Small decisions reveal the decision-making patterns that will govern large decisions. A student who designs systematically for an exam will design systematically for a career. A professional who cuts corners on minor projects will cut corners on major ones.

The Amadi Framework does not provide shortcuts. It provides structure for applying maximum rigor to problems regardless of their apparent scale. The methodology demands more cognitive effort than intuitive problem-solving, not less. It rewards that effort with outcomes proportional to the design's quality, which is proportional to the designer's willingness to see reality clearly, acknowledge constraints honestly, and iterate when evidence contradicts assumptions.

Systems analysis and design, practiced with this commitment, transforms limitations into parameters and failures into data. The framework's final contribution is this reframe: there is no such thing as failure in systems design. There is only feedback, and every feedback improves the next iteration.

References

Correlation One. (2025). Competition rules: June 2025 DoD Cyber Sentinel Challenge.

Kim, B., Cruden, G., Crable, E. L., Quanbeck, A., Mittman, B., & Wagner, A. D. (2023). A structured approach to applying systems analysis methods for examining implementation mechanisms. *Implementation Science Communications*, *4*(1). https://doi.org/10.1186/s43058-023-00504-5

Ozonwoye, C. (2025a, June 13). *Strategy: The systems thinker's approach* [Blog post]. Thoughts & Writings. Strategy: The Systems Thinker's Approach - Thoughts & Writings

Ozonwoye, C. (2025b, June 14). *Strategy: The systems thinker's approach - Part 2* [Blog post]. Thoughts & Writings. Strategy: The Systems Thinker's Approach - Part 2 - Thoughts & Writings

Siau, K., Woo, C., Storey, V. C., Chiang, R. H. L., Chua, C. E. H., & Beard, J. W. (2022). Information Systems Analysis and Design: Past Revolutions, Present Challenges, and Future Research Directions. *Communications of the Association for Information Systems*, *50*(1), 835–856. https://doi.org/10.17705/1cais.05037