NPS-AM-24-193

ACQUISITION RESEARCH PROGRAM
SPONSORED REPORT SERIES

Software Improvement Options for the H-1 Program

June 2024

Maj Joshua M. Westlund, USMC

Thesis Advisors: Jeffrey R. Dunlap, Lecturer
Dr. Robert F. Mortlock, Professor

Department of Defense Management

Naval Postgraduate School

Prepared for the Naval Postgraduate School, Monterey, CA 93943

ACQUISITION RESEARCH PROGRAM
DEPARTMENT OF DEFENSE MANAGEMENT
NAVAL POSTGRADUATE SCHOOL

The research presented in this report was supported by the Acquisition Research Program of the Department of Defense Management at the Naval Postgraduate School.

To request defense acquisition research, to become a research sponsor, or to print additional copies of reports, please contact the Acquisition Research Program (ARP) via email, arp@nps.edu or at 831-656-3793.

Acquisition Research Program
Department of defense management
Naval Postgraduate School

ABSTRACT

The AH-1Z and UH-1Y helicopters' software has long been called "spaghetti code" by those in the program office and developmental test squadron. For the first 15 years of the current models' service, years would go by between software updates due to the time required to code and test the software. Recent years have seen an improvement in software delivery timelines, but errors, rework, and delays still occur. A major factor in this issue is the software architecture: it is a large, unstructured monolith. Two types of upgrade options, modular monolith and microservices, are analyzed to determine a suitable alternative to the current software. The modular monolith architecture proves to be the most suitable based on its lower cost, higher performance, and faster delivery capabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

ABOUT THE AUTHOR

Maj Joshua "Chucky" Westlund graduated from the U.S. Naval Academy in 2011 with a degree in Oceanography. After The Basic School and earning his wings of gold at flight school, Maj Westlund served as a UH-1Y pilot with HMLA-167 in New River, NC, and HMLAT-303 at Camp Pendleton, CA. He is currently serving as the Assistant Program Manager for Systems Engineering at PMA-276 at Patuxent River, MD. He is married and has one child.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

A huge thanks to my wife, Meghan, for all the support during this program. I would not have started this academic journey without her nudging me to apply, and I certainly wouldn't have finished without her support.

Many thanks to all those that helped me find the pubs and other data: Jen, Trent, John, Ben, Dennis, Beka, Jeremy, "Rizzo," "Hodor," "Beaker," "Buzz," and "Chief."

Thanks to my advisors, other professors, and classmates for an interesting and challenging two years.

THIS PAGE INTENTIONALLY LEFT BLANK

NPS-AM-24-193

ACQUISITION RESEARCH PROGRAM
SPONSORED REPORT SERIES

Software Improvement Options for the H-1 Program

June 2024

Maj Joshua M. Westlund, USMC

Thesis Advisors: Jeffrey R. Dunlap, Lecturer
Dr. Robert F. Mortlock, Professor

Department of Defense Management

Naval Postgraduate School

Prepared for the Naval Postgraduate School, Monterey, CA 93943

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AAF	Adaptive Acquisition Framework
CI/CD	continuous integration, continuous delivery
DIB	Defense Innovation Board
DOD	Department of Defense
DODI	Department of Defense Instruction
DON	Department of the Navy
FY	Fiscal Year
HMLA	Light Attack Helicopter Squadron
MAGTF	Marine Air-Ground Task Force
MOSA	modular open systems approach
NAVAIR	Naval Air Systems Command
NDAA	National Defense Authorization Act
OUSD (A&S)	Office of the Under Secretary of Defense for Acquisition and Sustainment
PMA	Program Management, Air
PMA-276	Light Attack Helicopter Program
PMI	Project Management Institute
USMC	United States Marine Corps

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The AH-1Z *Viper* and UH-1Y *Venom*, collectively referred to as H-1s, are multi-mission helicopter platforms for the United States Marine Corps (USMC) that make up Light Attack Helicopter Squadrons (HMLA; U.S. Navy, 2021). The mission computer software in H-1 aircraft is considered by many personnel in the Light Attack Helicopter Program Office (PMA-276) and users (including the author) as slow to update, time-consuming and expensive to test, and is often error-prone upon deployment (Department of the Navy [DON], 2022a). A primary source of these issues lies in the software architecture (Tran & Schneider, 2024). The complex nature of the software code means that programmers have more difficulty bringing in new code without breaking existing proficiency (Tran & Schneider, 2024). Because of the ease with which errors can work into the software, everything the mission computer interacts with requires extensive testing, extending an already lengthy timeline of getting capability upgrades to the H-1 fleet.

For over a decade, industry best practices have been to avoid software code and architecture similar to that found in the H-1, preferring to have groupings of software—be they in a monolith or distributed architecture—to accomplish individual tasks that communicate with one another in a controlled manner (Richards, 2022). Instead, the software in the H-1 mission computer is what Foote and Yoder (1997) would call "a big ball of mud" (p. 1); that is to say, the software code has grown large and tangled, making it hard to work with (Tran & Schneider, 2024). PMA-276 urgently needs options to improve the H-1 software to decrease cost, speed up development and testing, and ultimately deploy more valuable features to the fleet.

A. RESEARCH QUESTIONS

The primary question of this research is, What is the best option for upgrading the H-1 mission computer software to better align with current software practices, decrease cost and time to test, and increase capability and speed to the fleet? The primary question covers the type of architecture to use and how to transition from the current mission computer architectural pattern to a new architecture.

Secondary questions for research include:

- How long might it take to upgrade the software?
- What is a fair estimate of the cost for a contractor to conduct the upgrade?
- Would hiring an outside software architect and/or consultant to oversee the transition benefit the program office?
- How might the program office maximize utility to the fleet while minimizing cost during the transition?

B. METHODOLOGY

This capstone uses qualitative analysis to determine which software architecture and transition methods are most suitable for use in the H-1. The literature review presents various types of architecture as a baseline, and the analysis chapter delves into the H-1's software to determine the best fit. As an infinite number of software configuration options are available, this study presents and compares common types in use.

Various attributes are used to assess and analyze software and software architecture. The following is a list of the primary attributes used for this research and their definitions within the context of software engineering:

- **agility:** a broad term for the ease of responding to or implementing change; it is comprised of more measurable metrics such as testability and deployability (Ford et al., 2022).
- **cost:** relative expense of developing or implementing change to a system.
- **deployability:** the ease with which a system can be delivered and installed; it can also be referred to as implementation (*PC Magazine*, n.d.a).
- **extensibility:** "Capable of being expanded or customized" (*PC Magazine*, n.d.b).
- **fault tolerance:** the ability for the system to continue to operate if an error is encountered (Richards, 2022). A system with low fault tolerance will fail completely if an error is encountered; a system with high tolerance may continue to conduct other processes if a fault is encountered (Richards, 2022).
- **performance:** the speed of a system, measured in the time required to conduct a task (Shanthi, 2018).
- **reliability:** how likely a system will conduct the functions that it is meant to or has been asked to perform (*PC Magazine*, n.d.c). Measures of this include available time, the probability of failure on demand (how often a system will fail when asked to do a function), rate of occurrence of failure,

and mean time to failure (average time between failure events; *PC Magazine*, n.d.c).

- **security:** how safe the system is from attack (*PC Magazine*, n.d.d). The security of a computer system incorporates who or what has access, authenticating access attempts, encrypting data, protection from malware or spyware, and data backup and recovery (*PC Magazine*, n.d.d).
- **simplicity:** relative ease to initially create and deploy a system.
- **testability:** how simple it is to test a system and how complete such a test is (Ford et al., 2022).

The scope of this research will not look to prove or test the technical accuracy of the references' assessments of the various software architectures, but rather use what is found in the literature review as a baseline to analyze the H-1 software and recommend an upgrade strategy. Finally, the analysis is based on the current mission computer used in the aircraft. The mission computers have recently been upgraded and any will likely be in use for several years (DON, 2022a). The analysis is based on this hardware constraint.

C. LIMITATIONS AND SCOPE

In researching this topic, much of the published information is unspecific, as companies wish to limit views into their proprietary software. The architectures mentioned in most articles and books are either large and meant for multiple servers or for consumer goods such as smartphones, not computers in an aircraft. The focus of this capstone research project is on a relatively small computer in an aircraft vice a network of servers across the globe to connect business and their customers. It bears closer resemblance to the needs of a smartphone but also requires other considerations for safety, inputs, and testing. While the principles of software architecture are the same regardless of the size of the system, there may be different challenges not mentioned in most research for the niche application of aircraft software. The literature generally uses theoretical ideas, best practices, and generalized terms. As there is no one-size-fits-all approach, the best software for the H-1 program may ultimately differ from the ideal envisioned by the current literature. Further, as software is ever-evolving, options may expand in the coming years and necessitate further changes.

The scope of this capstone limits the prospective choices to the most common and viable options. As previously mentioned, there are infinite options for coding software;

thus, limiting the potential choices was deemed necessary. Future readers should ensure that new methodologies are explored for their own programs, as software engineering is a rapidly evolving field.

This capstone research is meant to provide a qualitative and programmatic perspective on software architecture decisions. Some technical details and background are presented to help readers understand software architecture, though not at a level of detail sufficient to make one an expert in how to structure software. Recommendations are based on the qualitative information found in current research and the analysis of the H-1 software.

D. ORGANIZATION OF PROJECT

This capstone contains the following chapters:

- **Chapter II:** Software Development Literature Review
- **Chapter III:** Aircraft and Software Background
- **Chapter IV:** Analysis
- **Chapter V:** Summary and Recommendations

Chapter II summarizes the software acquisition pathway and software architecture pattern theory, and gives examples of some patterns and their strengths and weaknesses.

Chapter III covers information on the AH-1Z and UH-1Y aircraft, their history, mission, and historical software challenges.

Chapter IV presents the data and analysis. The H-1 software and procurement process is qualitatively analyzed using variables such as agility, testability, and reliability.

Chapter V summarizes the data presented, conclusions, and recommendations for the H-1 program and follow-on research.

II. SOFTWARE DEVELOPMENT LITERATURE REVIEW

This chapter introduces software architecture fundamentals. An overview of two major types of architecture and some specific styles are presented, including the strengths and weaknesses of the styles. The overviews are not meant to delve deeply into the technical details, rather to give a sufficient understanding of software architecture to help analyze the current H-1 system and potential future courses of action.

A. THE SOFTWARE ACQUISITION PATHWAY

As part of the Fiscal Year (FY) 2018 National Defense Authorization Act (NDAA), the Defense Innovation Board (DIB) was tasked with investigating ways to improve software development and acquisition (Defense Innovation Board [DIB], n.d.). One of the DIB's recommendations was to create a software acquisition pathway (DIB, 2019). In January 2020, Department of Defense Instruction (DODI) 5000.02, *Operation of the Adaptive Acquisition Framework* (AAF), introduced the software acquisition pathway (Office of the Under Secretary of Defense for Acquisition and Sustainment [OUSD(A&S)], 2022). Figure 1 shows the AAF. This new pathway "establishe[d] policy, assign[ed] responsibilities, and prescribe[d] procedures for the establishment of software acquisition pathways to provide for the efficient and effective acquisition, development, integration, and timely delivery of secure software" (OUSD(A&S), 2020, cover page).

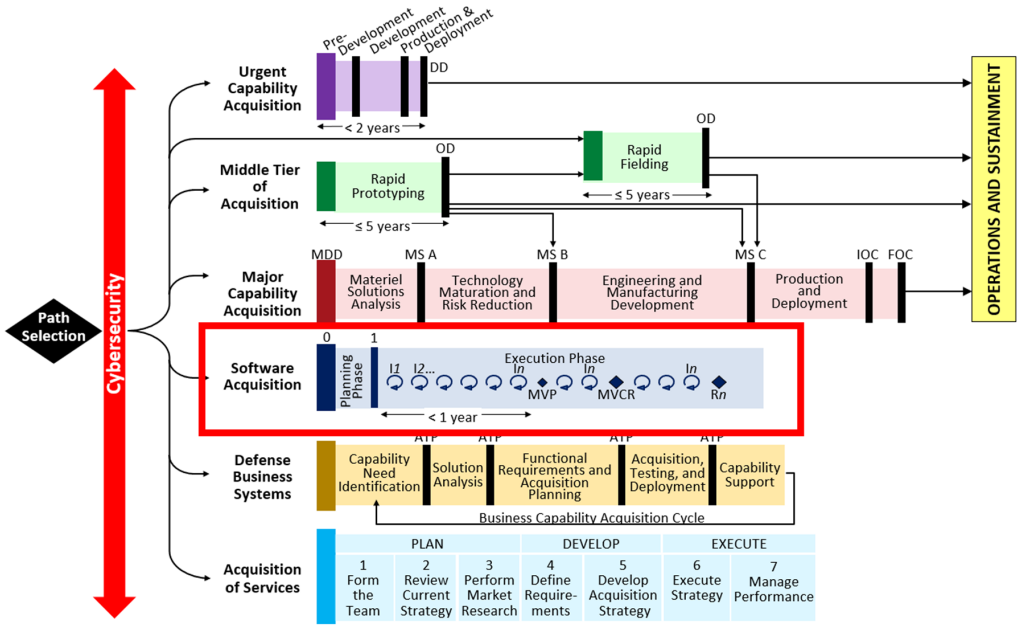

Figure 1. Adaptive Acquisition Framework.
Adapted from OUSD(A&S, 2022).

DODI 5000.87, *Operation of the Software Acquisition Pathway*, breaks the new software acquisition pathway into two phases: planning and execution (OUSD[A&S], 2020). In the planning phase, the instruction mandates that programs using the software acquisition pathway must include end users often throughout the life cycle of the project, test and evaluate the software continuously, and deliver a product on an annual basis, with the first delivery due 1 year after beginning a software acquisition pathway program. It also recommends using architecture patterns that enable a modular open system approach (MOSA) and utilizing flexible and modular contracts to maximize agility and responsiveness. In the execution phase, DODI 5000.87 recommends using modern software development practices such as continuous integration and continuous delivery (CI/CD), automated testing, and frequent user feedback (OUSD[A&S], 2020). The instruction allows programs using another type of acquisition pathway to also use the software acquisition pathway for embedded software as long as the acquisition strategies are aligned and integrated into one another: schedules for testing, evaluation, and fleet release should be coordinated to minimize cost and time and maximize capability to the warfighter.

As stated, DODI 5000.87 mandates an iterative approach to software acquisition (OUSD[A&S], 2020). This shift in methodology is one of the biggest differences between the other pathways of the AAF and the software path. With the other pathways, especially the Major Capability Acquisition pathway used for the development of major development acquisition programs with extensive hardware, the primary historical method of acquisition is the predictive method, also known as traditional or waterfall (Project Management Institute [PMI], 2021). In the predictive method, requirements are established during the planning phase, and then during the execution phase, the contractor works on the project with minimal (and highly vetted) changes (PMI, 2021). With an adaptive, iterative, or Agile strategy, the goal is to enable the development team to rapidly change in response to user needs (PMI, 2021). In the *Operation of the Software Acquisition Pathway*, programs are mandated away from the waterfall method to an iterative methodology (OUSD[A&S], 2020). Figure 2 shows a simplified version of the software pathway; note the emphasis on iteration loops and consistent delivery of software to the end user. Figure 3 shows a comparison between the waterfall and Agile methods.

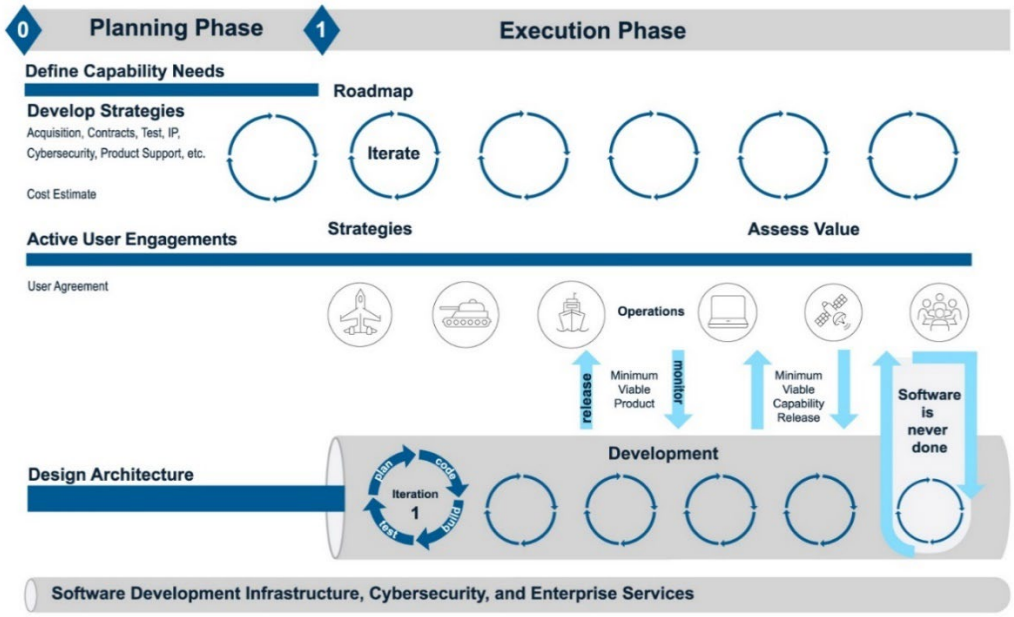

Figure 2. Software Acquisition Pathway. Source: OUSD(A&S, 2020).

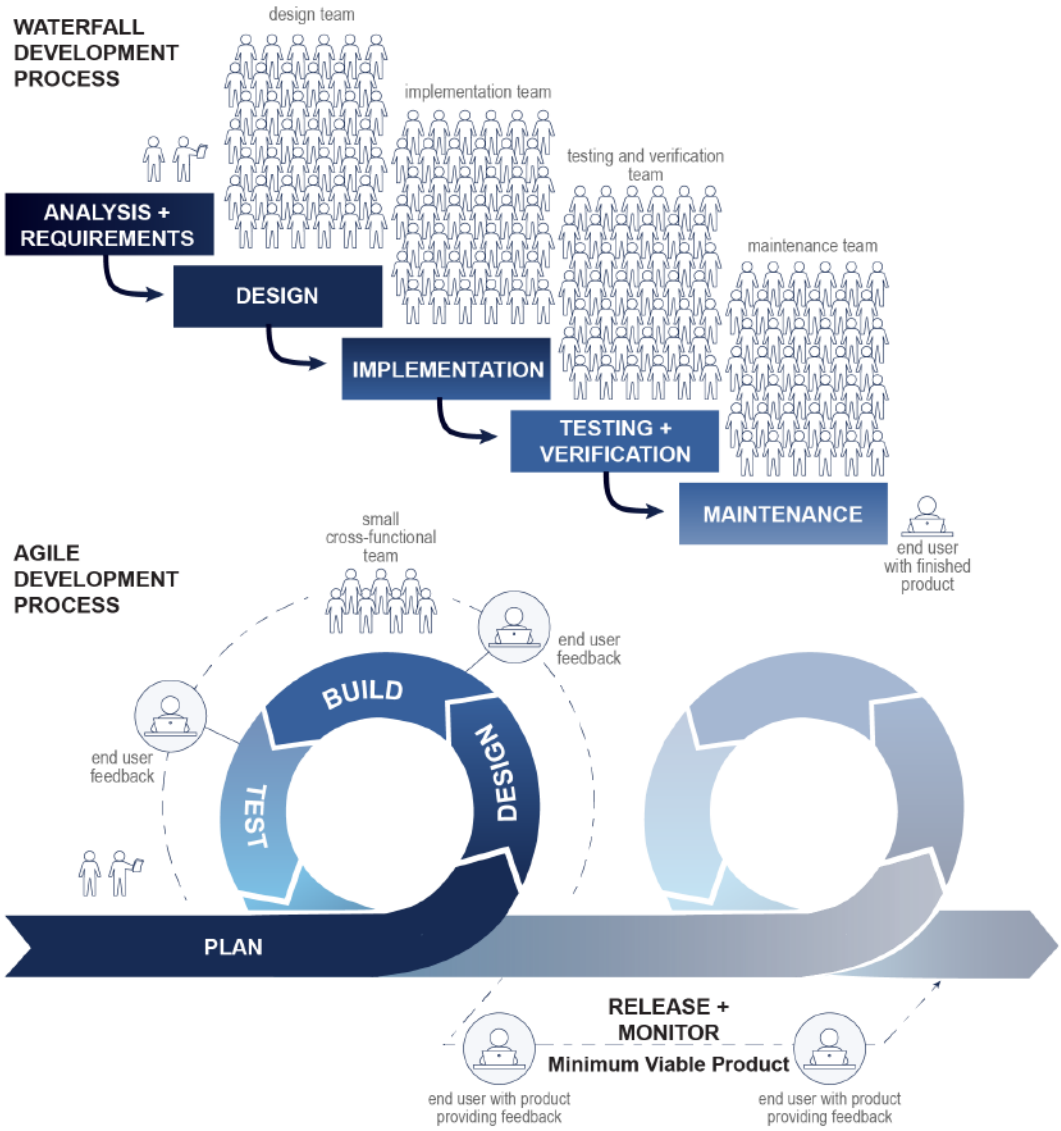

Figure 3. Comparison of Waterfall and Agile Development Processes. Source: Government Accountability Office (GAO, 2023).

B. SOFTWARE ARCHITECTURE

When building anything, it is usually important to have a plan. For the software engineer, this plan is software architecture. Software architecture can be understood as the "blueprint" of a system and its subsystems (Dhaduk, 2020). Just as a structural architect may use different styles to design a building, such as Gothic, mid-century modern, or Victorian architecture, a software engineer also has choices in how to structure computer software. These choices in software architecture can significantly

impact the software's characteristics, such as performance, scalability, testability, and maintainability (Milić & Makajić-Nikolić, 2022; Richards, 2022).

The software blueprint can be broken down into several nesting layers: architectural style, architectural pattern, and design pattern (Richards, 2022). As Figure 4 shows, several design patterns may be used within any architectural pattern, and multiple architectural patterns may be used within an architectural style. In the literature, these names are often used interchangeably. This study focuses on architectural style; all use of "software architecture" refers to the style unless otherwise noted.

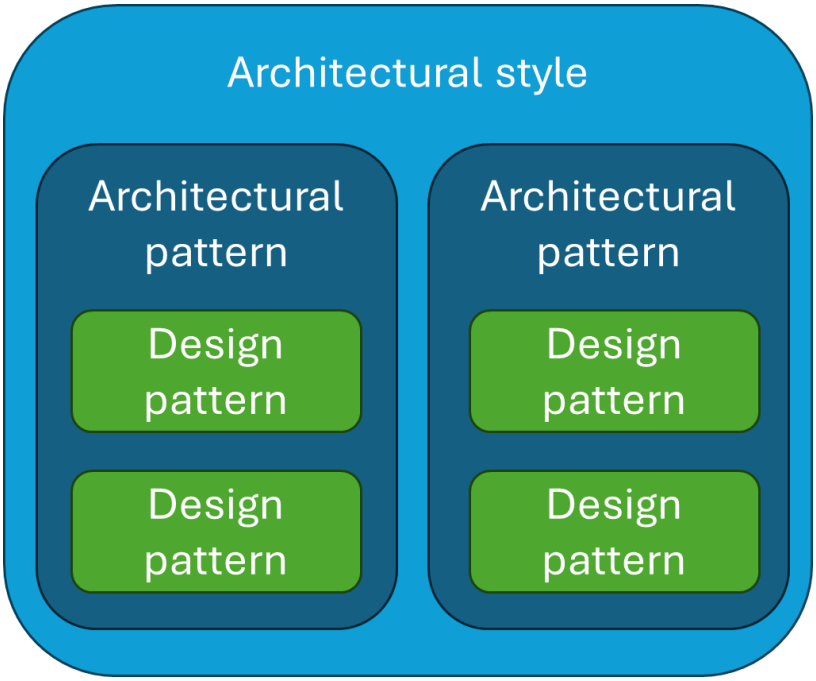

Figure 4. Software Design Relationships. Adapted from Richards (2022).

There are two main categories of software architectural styles: monolith and distributed (Richards, 2022). Monoliths are single repositories of code that conduct all operations required of the software (Semaphore, 2022)—see Figure 5 for a visual representation (Semaphore, 2022). In contrast, distributed styles are made up of multiple "boxes" of code(Richards, 2022)—see Figure 6.

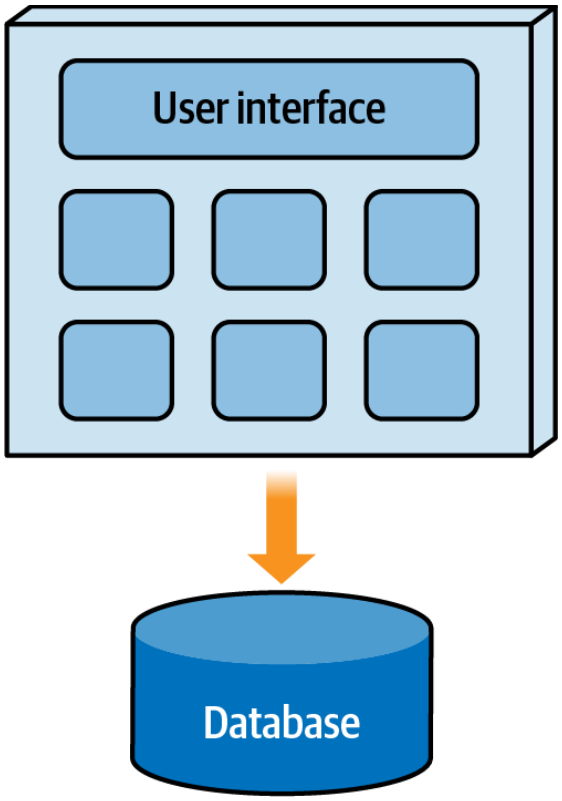

Figure 5. Monolithic Architectural Style Example. Source: Richards (2022).

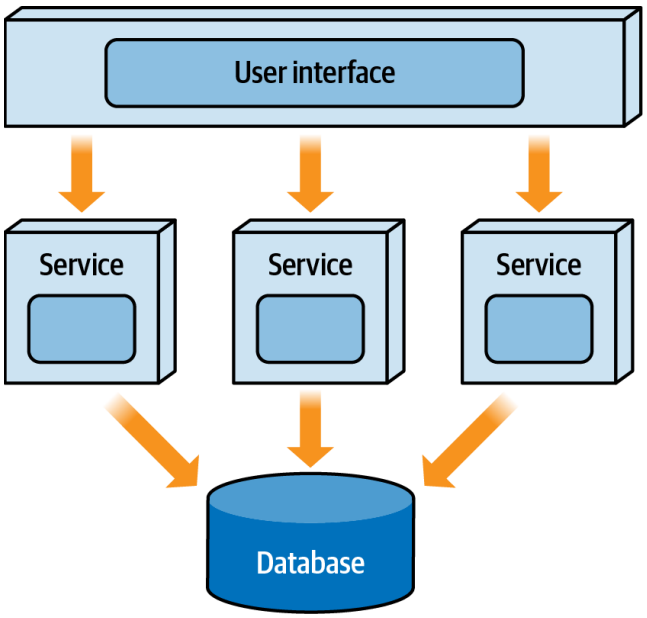

Figure 6. Distributed Architectural Style Example. Source: Richards (2022).

C. MONOLITHS

A monolith's strength lies in its simplicity (Richards, 2022). As Richards noted, they are generally easier and less expensive to create and can be delivered to the end user faster than other architectural styles. Issues may arise as the monolith grows. If not carefully handled, a monolith can become what can be called spaghetti code or a "big ball of mud" (Foote & Yoder, 1997, p. 2). As Foote and Yoder detailed, this growth can happen haphazardly or due to hastily initiated fixes to the code. What may have started out as a simple, well-structured code base—or not, as it may have never been neatly ordered—can become fouled with circular logic and repeated information (Foote & Yoder, 1997). An unstructured, large repository of code can make it difficult to maintain and update, as any changes in one portion of the code may have unintended consequences elsewhere (Su & Li, 2024). Some examples of monoliths include layered, microkernel, and modular monoliths (Richards, 2022).

Layered architecture, sometimes called n-tier, is one of the most common software architectures and has been around for many years (Richards, 2022). Normally, a layered architecture has four layers, though the number of layers can change based on the use case (Richards, 2022). The layers are usually partitioned by technical domain, such as presentation logic, business logic, and persistence logic and each has a specific job (Richards, 2022). When a user requests information, the request goes to the first layer; if the information is there, it is sent back to the user for display (Richards, 2022). If not, the request is sent down a layer; if the information is there, it is sent back up to the previous layer and then to the user for display (Richards, 2022). This can happen all the way down through as many layers as the system contains (Richards, 2022).

Layered architecture is usually relatively easy to build, quick to deploy, and inexpensive (Richards, 2022). Drawbacks of this type of architecture include difficulty in updating the code, low fault tolerance, and, as with all monoliths, a need to redeploy the entire application anytime anything is changed within the code (Richards, 2022). As with all monoliths, any code change to one portion means the entire application must be tested. While it is generally a simple task to test the layered architecture, it can be a laborious and time-consuming process.

The modular monolith has become popular relatively recently, though the monolith and modular architecture are both much older (Su & Li, 2024). It can be a bridge between a monolith and microservices architectures, combining the monolith's simplicity with the strengths of microservices (Su & Li, 2024). While still deploying as a monolith, the modular monolith seeks to break up the code within an application into smaller parts for work (Su & Li, 2024). The parts are separated within the code and communicate via strict interfaces (Semaphore, 2022). Multiple studies, books, articles, and coder blogs recommend updating a monolith to a modular monolith prior to updating further to microservices; once the architecture becomes a modular monolith, further change to microservices may be unnecessary (Fernandez, 2023; Semaphore, 2022; Su & Li, 2024). Modular monoliths maintain the relative ease of development of a standard monolith while being easier to maintain and update (Smith, 2024). Drawbacks of the modular monolith include a requirement to use only one coding language and deploying as one unit vice the "plug-and-play" ability of microservices (Gupta, 2020).

D. DISTRIBUTED ARCHITECTURE

Distributed architectures are made up of multiple units, often called services, to accomplish their tasks (Richards, 2022). Although they may be more difficult and expensive to develop initially, they are more easily updated and scaled and have greater fault tolerance (Richards, 2022). One of the most common distributed architectures is microservices. In a microservices architecture, the software is broken down into individual services that communicate via strict instructions (application programming interface [API]; Semaphore, 2022). As with all distributed architectures, microservices' strengths are increased agility, deployability, reliability, and scalability, among others (Merson, 2015). Some of the weaknesses of microservices include high cost, difficult testability, increased security needs, and high memory use (Merson, 2015).

E. SUMMARY

The relatively new software acquisition pathway gives programs the ability to approach software acquisition in a modern way (OUSD[A&S], 2020). Using Agile methodologies, software teams and contractors can work with end users to "plan,

develop, build, test, release, deliver, deploy, operate, and monitor" (OUSD[A&S], 2020) software to the fleet faster, more efficiently, and with more return on investment. The H-1 software acquisition program will be analyzed against DODI 5000.87's standard.

The two main types of software architecture are monolith and distributed (Richards, 2022). Each has its own strengths and weaknesses, with monoliths generally being fast, cheaper, and easier to create, but they are prone to becoming overly complex and difficult to update (Richards, 2022). Distributed architecture, exemplified by microservices, is usually easier to update and faster to test, while also more difficult and expensive to start (Richards, 2022). A modular monolith combines many of the strengths of each type (Su & Li, 2024). Chapter IV presents a comparison between a modular monolith and microservices as candidates for use in the H-1.

THIS PAGE INTENTIONALLY LEFT BLANK

III. AIRCRAFT AND SOFTWARE BACKGROUND

The AH-1Z *Viper* and UH-1Y *Venom*, colloquially known as the Cobra and Huey and known collectively as H-1s, are multi-mission helicopter platforms for the United States Marine Corps (USMC) (U.S. Navy, 2021). As the aircraft that make up Marine Light Attack Helicopter Squadrons (HMLAs), their mission statement is to support "the Marine Air-Ground Task Force (MAGTF) commander by providing offensive air support, utility support, armed escort, and airborne supporting arms coordination; day or night; under all weather conditions; during expeditionary, joint, or combined operations" (Marine Light Attack Helicopter Squadron 267, Unit History section, first paragraph, n.d.).

A. AIRCRAFT HISTORY AND MISSON

The history of the Cobra and Huey goes back to the 1950s when the first version of what became the Huey flew at the Bell Aircraft facility in Fort Worth, TX (Fardink, 2016). The Huey became famous during the Vietnam War, known for its iconic looks (see Figure 7), millions of flight hours, and the signature "whop-whop-whop" sound of its two main rotor blades, not to mention its inclusion in Vietnam War movies since that time (Vietnam Veterans Memorial Fund, n.d.). The Cobra was developed from the baseline of the Huey, and the two aircraft have been upgraded multiple times over the following decades (U.S. Navy, 2021). Figure 8 shows an early Cobra gunship. Figures 9 and 10 showcase the differences between the upgraded aircraft and their predecessors.

Figure 7. A Vietnam-Era UH-1E. Source: Collings Foundation (n.d.).

Figure 8. A Vietnam-Era AH-1G. Source: U.S. Army (n.d.).

Figure 9. A UH-1Y Firing a Rocket. Source: Bell Textron Inc. (n.d.b).

Figure 10. An AH-1Z in Flight. Source: Bell Textron Inc. (n.d.a).

The *Viper* and *Venom* versions of the storied H-1 platform have been purpose-built to have 85% parts commonality, including a glass cockpit, tail section, rotor blades, and engines (Bell Textron Inc., n.d.b). One of the parts common to both aircraft is the mission computer. The mission computers' software controls the pilots' displays, keyboards, and other selection keys. Pilots use the software outputs to monitor critical flight systems, set up weapons and communications channels, and input navigation data, among other tasks (Bell Textron Inc., 2004). The mission computer software is updated regularly, usually in concurrence with hardware upgrades that necessitate software changes. The software has also been updated to increase capability within existing processes and fix errors and glitches. Table 1 summarizes releases, major capability updates, and errors, glitches, and other difficulties with each software deployment.

Table 1. Software Version History..

Mon/Year	Version name	Major Capability Updates	Errors/Bugs/Capabilities Not in Use
May 2006	SCS 3.3	-Baseline -Systems Display -Flight Information	-Tactical Data Communications (TDC) page inoperative
May 2008	SCS 4.0	-Minor updates	-TDC page inoperative
Sep 2012	SCS 5.3	-Improved flight page -Navigation display improvement	-TDC page inoperative
Mar 2013	SCS 6.0	-Warnings update -Systems display update -Waypoint Library addition -User waypoint count increase	-TDC page inoperative
Dec 2015	SCS 7.1	-TDC page partially operative (requires hardware update for full capability) -Targeting update -Ground proximity warning system (GPWS) -Identification, friend or foe improvement	-GPWS does not account for rising terrain -Full motion video page inoperative -Default conditions not as expected -7.1 and older model of sensor incompatibility -Targeting delay -Subsystem memory drop on shutdown -Airspeed/altitude error
Oct 2018	SCS 7.1.1 & SCS 8.1.1	-Airspeed/altitude error update -8.1.1 for new mission computer hardware, no other appreciable difference	-Subsystem memory drop on shutdown
Jun 2021	20.1.5Q2	-Targeting improvements -Survivability equipment display changes -Hover aid graphic	-Hover graphic errors -Subsystem memory drop on shutdown
Mar 2022	21.1.5Q2	-Fuel flow calculator -Flight display improvements -Stick shaker over-g cueing -TDC page capability improvement -Navigation improvements -Software to support new weapon	-Subsystem memory drop on shutdown -Targeting distance error -Fuel flow calculator errors/difficult user interface
Apr 2023	22.1.3Q	-Update to support communications hardware -Emergency procedures display on screen	-Subsystem memory drop on shutdown -Targeting distance error

Adapted from Department of the Navy ([DON], 2013, 2014, 2015a, 2015b, 2019a, 2019b, 2022a, 2022b, 2022c, 2023a, 2023b); Naval Air Systems Command ([NAVAIR], 2006, 2008, 2009, 2012); L. Simpson (former PMA-276 product developer, interview with author, May 21, 2024)

Like the previous versions of the H-1, the AH-1Z and UH-1Y are capable of accomplishing the required missions; however, HMLA pilots consistently ask for more capabilities from the program office and for inefficiencies and bugs to be fixed (PMA-276, 2024). As aircraft and pilots of the HMLA could find themselves doing any number of missions during a single sortie, the mission computer software must be robust and

capable of tasks such as navigation, sensor display, and weapons setup. On today's modern battlefield, pilots expect the aircraft to automate as many mundane tasks as possible to allow more cognitive focus on flying, decision-making, and flight leadership. To do so, the software needs to be capable of quick, efficient, and inexpensive updates.

B. SOFTWARE HISTORY AND ISSUES

In a world where speed to the fleet is paramount and budgets are tight, the H-1 mission computer software architecture is one of many roadblocks keeping PMA-276 from delivering needed capability fast enough to stay relevant (Tran & Schneider, 2024). The H-1 software has long been called "spaghetti code" by those in the program office and developmental test squadron, HX-21 (J. Hurst, email to author, October 14, 2022). One definition of *spaghetti code* is "program code written without a coherent structure" (*PC Magazine*, n.d.e). In more technical terms, the software may be described as having an unstructured monolith architecture (Belcher, 2020). Figure 11 provides a visual example of spaghetti code.

While a monolith is generally easier to begin coding, it may eventually make it harder for developers and coders to update the software and add new features (Richards, 2022). When changes are made, they often lead to bugs in other areas of the code because of the interconnectivity and/or circular logic within the software (Belcher, 2020). Due to the error-prone nature and lack of containment of potential errors, testers must check the entirety of the software for faults, increasing the time and cost of the test. When faults are found, the software often needs immediate fixes before deployment, adding time and expense.

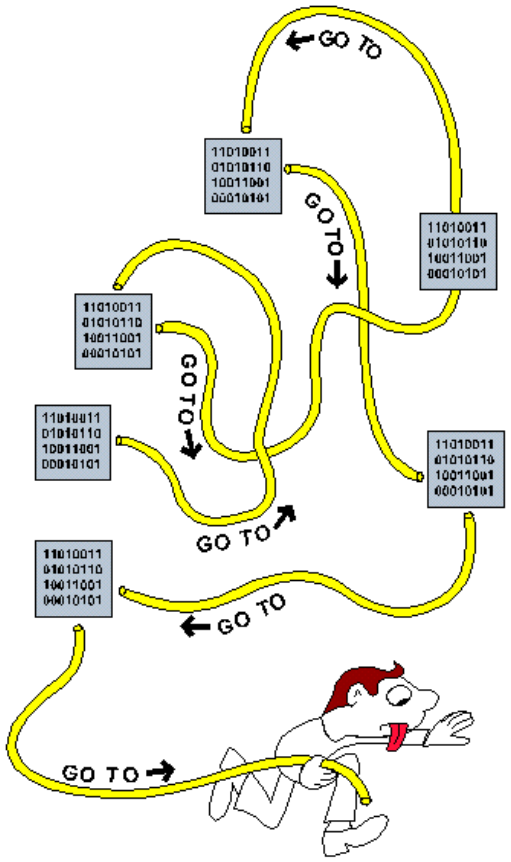

Figure 11. Simplified Visualization of Spaghetti Code. Source: *PC Magazine* (n.d.e).

Instead of military aircraft software that includes all the features desired for missions, pilots have increasingly turned to tablets to provide needed situation awareness and processes (Robinson, 2017). In the author's experience, most are military-procured tablets, while others are personally procured hardware with civil aviation applications. Regardless of the source, tablet capabilities, as well as personal use smartphones and other software-reliant hardware, set the standard for how pilots expect their aircraft software to behave and the interval at which it is updated. Aircraft software acquisition has been slower than commercial tablet and smartphone upgrades.

Since the beginning of the *Viper* and *Venom* program, software has been viewed as slow to update (J. Tran, email to author, May 7, 2024), especially by the H-1 fleet (including by the author during time spent in an HMLA). Deliveries of software to the PMA from the contractor sped up with a new contract for the last three software releases (J. Tran, email to author, May 21, 2024). Under this new contract, the PMA receives a software

delivery every 10 weeks for reference and testing, and the fleet would ideally receive a software update release once per year (J. Tran, email to author, May 21, 2024).

By searching various editions of the publication that details the H-1 software, the intervals between software release were determined: H-1s could go several years without software updates in the first 15 years of the program (DON, 2013, 2014, 2015a, 2015b, 2019a, 2019b, 2022a, 2022b2022c, 2023a, 2023b; NAVAIR, 2006, 2008, 2009, 2012). Even then, updates did not bring all the capabilities pilots desired, nor was it free of glitches (DON, 2022b). In recent years, the software has been updated on a 12 month cycle (ideally), though this is still slow when compared to common software standards, such as Apple's iOS (Casserly, 2024; PMA-276, 2024). Furthermore, the issues with bugs and errors persist. In late 2023, software was scheduled to be tested at HX-21 (T. Trepanier, email to author, October 11, 2023). However, it was delayed due to a critical error in the code, necessitating that the prime software contractor readdress and fix said software, all at the taxpayers' expense and to the detriment of the fleet user (T. Trepanier, email to author, October 11, 2024).

All the delays in delivering software to the fleet potentially means that warfighters cannot keep up with the threat landscape and improved weapons. In addition to providing less capability for the warfighter, the complexity of the current software makes it more likely to be compromised by a sophisticated enemy with cyber-attack capabilities, such as Russia or China. In order to better defend against such cyberattacks, the H-1 program office needs the ability to quickly and efficiently update software in weeks rather than months or years.

C. SUMMARY

The H-1 is a storied aircraft with a long history of mission accomplishment. The newest versions are even more capable than their predecessors, though that does not mean the aircraft is without its drawbacks. The mission computer software architecture contributes to the delayed delivery of updates by increasing difficulty of writing new code (Tran & Schneider, 2024). The analysis chapter explores the best option for a new software architecture to increase speed of delivery to the fleet.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ANALYSIS

The H-1 software architecture is analyzed primarily from a programmatic lens as opposed to technical. Where applicable, a user lens is also used. General characteristics of the current software architecture and their effect on the acquisition process and user are employed and compared with a modular monolith and microservices. Since an architectural style may be implemented in many ways with varying degrees of quality, the analysis of potential future upgrades is based on an ideal implementation of each architectural style. Where relevant, the current software architecture is compared with the possible upgrade choices. In some cases, it is left out of the following paragraphs, such as startup cost and simplicity, where the current software means doing nothing and thus no cost nor difficulty to start. The final stipulation for this analysis is working within the constraints of the current hardware due to recent upgrade and financial limitations (DON, 2022a).

A. AGILITY

When viewed from a user perspective, the agility of the current H-1 software is low based on the time between updates. In comparing versions of each aircraft's Naval Aviation Technical Information Product (NATIP) from various years, the software was found to have been updated nine times since 2006 (DON, 2013, 2014, 2015a, 2015b, 2019a, 2019b, 2022a, 2022b, 2022c, 2023a, 2023b; NAVAIR, 2006, 2008, 2009, 2012). The average time between software updates was 26 months, with a range of 6 months to over 4 years (DON, 2013, 2014, 2015a, 2015b, 2019a, 2019b, 2022a, 2022b, 2022c, 2023a, 2023b; NAVAIR, 2006, 2008, 2009, 2012). Compared with the release schedule of new iOS builds, the H-1 software is slow and inconsistent (Casserly, 2024). Apple released major updates (such as from iOS 4 to iOS 5) to its software 16 times between 2007 and 2023 (Casserly, 2024); this count does not include minor updates to each version of iOS. For smaller upgrades, Apple deployed 107 updates to its iOS versions between January 2020 and May 2024 (Apple, 2024); the H-1 program had no such minor updates. Apple's major updates occurred once a year, with the longest time between

updates being approximately 1 year and 4 months (Casserly, 2024); the longest time between the smaller updates from 2020 to 2024 was 65 days (Apple, 2024).

The example of Apple shows that even monoliths, of which iOS is one, can be updated quickly. It could also be argued that the update cycle of such common software as found in smartphones sets a standard that personnel look to achieve regardless of use case. While Apple and other software creators operate under different rules, needs, and requirements than government aircraft acquisitions, the speed at which smartphones are upgraded gives the program office a high benchmark to strive for; in this regard, it has so far been lacking.

The H-1 software's lack of agility may stem from multiple sources, such as rework due to its complex nature, lack of funding, and the time required to test. Multiple factors likely play a part in each long stretch between software upgrades to the H-1 fleet. The ambiguous nature of "agility" relegates it to a lower tier of importance when deciding on a new architecture to upgrade to. That said, both a modular monolith and microservices architectures would likely improve the agility of the H-1 software by reducing the complexity of the code, thus reducing time to code, rework required, and testing hours required.

B. COST

In the current U.S. government landscape of tight—and often delayed—budgets and continuing resolutions, cost is arguably the most important factor in upgrading the H-1 software. Major factors of this cost are paying the contractor to code, funding any upgrades (or not), and testing. For this analysis, the base contract with the prime contractor is ignored as it is already in place, and upgrades can be worked within its current boundaries.

Each new software version requires extensive test to ensure it meets requirements and other portions of the code have not been broken, adding to its expense. As part of the contract, the software contactor conducts testing on the software (J. Tran, email to author, May 16, 2024). Once complete, the software is delivered to PMA-276 for further testing (W. Cosgrove, email to author, April 9, 2024). Until recently, all testing at PMA-276 had

to be completed in the aircraft (J. Hurst, email to author, May 20, 2024). The time estimated to test the software is 4 hours of ground time and 2 hours of flight time per functional area tested (W. Cosgrove, email to author, April 9, 2024). The cost of each flight hour is approximately $34,000, and each ground hour is $1,100 (W. Cosgrove, email to author, April 9, 2024).

Since the H-1 software has a monolithic architecture, the entire software needs to be tested. Especially important for test are the updated portions and safety of flight related items (J. Hurst, email to author, May 17, 2024). Upgrading to a modular monolith or microservices architecture could save money by removing some burden of test (Richards, 2022). If the H-1 mission computer software architecture were upgraded, the PMA could save time, effort, and money on test efforts and reinvest back into further updates or maintenance.

For the decision matrix, cost is broken into the initial cost of creating a new architecture and long term maintenance costs. A modular monolith is likely a less expensive route to take than microservices or any distributed architecture (Su & Li, 2024). Microservices can be exponentially more expensive to build than a modular monolith due to its initially complex nature (Su & Li, 2024). In a budget-constrained environment, cost becomes one of the determining factors on options available to the H-1 program, making the startup cost the most important factor for the decision matrix.

In the long term, microservices and a modular monolith would have similar maintenance and new feature costs (Su & Li, 2024). Each divides the code base into smaller parts to help manage writing new code (Su & Li, 2024). The current software scores worst of the three due to longer timelines to add capabilities and consistent rework required when unforeseen errors occur. Long term maintenance costs are in the second tier of importance for the decision matrix as cost is likely to continue to be an issue for the program office but not as difficult to realize or justify as the large, up-front cost required of a architecture overhaul.

C. DEPLOYABILITY

The deployability of the current system is high: procedures are in place that make updating the software in the aircraft relatively simple with common tools in the squadrons. An upgrade to either a modular monolith or microservices is unlikely to affect the ability of the program to deliver mission computer software and the squadrons to load the software into the aircraft. One way that microservices would be an improvement over a modular monolith is the ability to deploy portions of the software (Richards, 2022). With a monolith, the entirety of the software is deployed all at once, whereas with microservices, small portions can be deployed individually (Richards, 2022). For the mission computer, this feature is assessed to be a low priority and is excluded from the decision matrix.

D. EXTENSIBILITY

The current H-1 mission computer software has the ability to upgrade but with difficulty (Tran & Schneider, 2024), giving it a low extensibility score. Normally, microservices is more extensible than a monolith (Richards, 2022). However, with the advent of the modular monolith architecture, the monolith becomes more extensible and able to grow (Su & Li, 20224). Without more technical details, determining whether a microservices architecture would make the H-1 software significantly more extensible than a modular monolith is nearly impossible. Both a modular monolith and microservices would improve the extensibility of the mission computer software.

E. FAULT TOLERANCE AND RELIABILITY

Generally speaking, a microservices architecture is more fault tolerant than a monolithic architecture (Richards, 2022). However, according to Su and Li (2024), a modular monolithic architecture attains the same levels of fault tolerance as microservices. When compared to the rest of the aircraft, the current mission computer and software are highly reliable, as less than 0.1% of aircraft discrepancies were written against the mission computer (R. French, email to author, May 20, 2024). For the decision matrix, fault tolerance is given a weight of only one due to the minimal improvement between the current software architecture and any future upgrades.

Reliability is excluded from the decision matrix given the low rate of failure for the current system and any upgrades would likely provide minimal improvements.

F. PERFORMANCE

The current system speed is sufficient for its use. Monoliths are usually faster than microservices, as the communications between services required in a microservices architecture take time (Richards, 2022). While this is normally counted in milliseconds (Richards, 2022), this is important when responding to threats, and every moment matters. A modular monolith then has the advantage over microservices for performance.

Microservices is also likely difficult to impossible to implement on the current hardware in the H-1. While possible to containerize for microservices on a desktop or similarly sized machine using systems such as Docker (Docker, n.d.), microservices are typically used on the cloud meaning access to large numbers of servers and high computing power (IBM, n.d.). This could realize in the inability to put microservices into the H-1 mission computer or it being prohibitively slow.

Performance is included in the second tier of importance for the decision matrix. It is important that the system run quickly to display and announce important safety of flight information to the pilots. It is not included in the top tier due to the current system operating at a sufficient speed and the overall importance of startup cost on working on an upgrade.

G. SECURITY

The details of the H-1 mission computer's security are above the classification level of this report. From a general standpoint, security is dependent on many factors to prevent malicious actors from gaining access to the system (*PC Magazine*, n.d.d). Security should be part of software development from inception through deployment regardless of architectural style (OUSD[A&S], 2020). Additionally, Su and Li (2024) stated that modular monoliths attain the security abilities of microservices and both are an improvement on a standard monolith. While security is vitally important to any computer system, it is given minimal weight for the decision matrix as it may be possible to have high security if well coded regardless of architectural style.

H. SIMPLICITY

The current architecture is simple yet complex: it can be likened to a ball of yarn or mud (Tran & Schneider, 2024). The current system may have started as a simple code base, but it has grown more complex with age.

Simplicity is broken into two parts for the decision matrix: initial build and future upgrade as both would be important to the program office for deciding a path for the software.

A modular monolith is generally a simpler upgrade path than microservices (Su & Li, 2024). Because microservices are complex and difficult to start, some coding professionals recommend upgrading to a modular monolith prior to microservices (Belcher, 2020; Gupta, 2020; Richards, 2022; Su & Li, 2024). Time to implement is a major factor of simplicity, and modular monolith would deploy to the fleet in less time than microservices. As noted in Chapter III, microservices can be difficult to create, while a monolith can be much easier, which is why many firms still use monoliths despite their potential drawbacks (Richards, 2022). As with cost, the difficulty in the initial creation of a microservices architectural style potentially makes it beyond reach of the H-1 program; a modular monolith architecture is less likely to have this problem (Richards, 2022; Su & Li, 2024). The simplicity of initial upgrade essentially becomes the main measure of schedule for the decision matrix.

In the long term, both a modular monolith would improve the ability of the contractor to upgrade the software. As discussed in Chapter III, the current unstructured nature of the monolith leads to difficult coding, rework, and errors deploying to the fleet. Both a modular monolith and microservices remove much of this difficulty (Su & Li, 2024).

I. TESTABILITY

Normally, a microservices architecture is more easily and quickly tested than a monolith (Richards, 2022). However, according to Su and Li (2024), the modular monolith is capable of similar levels of testability as a microservices architecture. Either upgrade option would increase testability over the current software. Without further

technical details that would be decided upon during the upgrade planning and coding, determining any difference between the two types is difficult, but either would improve testability over the current system.

J. ANALYSIS SUMMARY

Table 2 shows a summary of the qualitative analysis for each measurement characteristic for each software architecture. Factors are given a score of high, medium, or low to help compare and rank the three architectures. For all factors except cost, high is good; for the cost factors, high is a poor score. The qualitative analysis is then used in the decision matrix to assign a rank, one through three, with one being the best. Since the three architectures receive the same score for reliability and deployability, those two factors are excluded from the decision matrix.

Table 2. Qualitative analysis summary of software architectures

	Current Architecture	Modular Monolith	Microservices
Agility	Low	High	High
Cost - initial	Low	Medium	High
Cost - O&S	High	Low	Low
Deployability	High	High	High
Extensibility	Low	High	High
Fault tolerance	Low	High	High
Performance	High	High	Low
Reliability	High	High	High
Security	Medium	High	High
Simplicity - Initial Build	High	Medium	Low
Simplicity - O&S	Low	High	High
Testability	Low	High	High

K. DECISION MATRIX

The decision matrix is a simple ranking based on the previous analysis for the unweighted row. For instance, the current architecture is the least expensive to build, followed by the modular monolith, and finally microservices. Therefore, the current

architecture receives one point, modular monolith two, and microservices receives three points for the unweighted row. In the case of a tie between architectures, the average score for remaining places are averaged; if two architectures are vying for places one and two but tie, each is given a score of 1.5. The points are then multiplied by the criteria weight to get a weighted ranking. Both rows are added up to get a total score for both unweighted and weighted. For this decision matrix, overall lower scores are better.

Table 3 shows the decision matrix as an unweighted ranking of the current architecture, modular monolith, and microservices. Table 4 shows the weighted and unweighted ranking.

Table 3. H-1 Software Architecture Decision Matrix (unweighted scores only)

Decision Matrix													
		Cost			Simplicity				Fault			Scores	
	Criteria →	Initial	O&S	Performance	Initial Build	O&S	Agility	Extensibility	Tolerance	Security	Testability	(Lower is Better)	
Options ↓	Criteria Weight	3	2	2	1	1	1	2	1	2	1	Unweighted	Weighted
Current architecture	Unweighted Ranking	1	3	1.5	1	3	3	3	3	3	3	24.5	
	Weighted Ranking												
Modular Monolith	Unweighted Ranking	2	1.5	1.5	2	1.5	1.5	1.5	1.5	1.5	1.5	16	
	Weighted Ranking												
Microservices	Unweighted Ranking	3	1.5	3	3	1.5	1.5	1.5	1.5	1.5	1.5	19.5	
	Weighted Ranking												

In Table 3, the points spread for the unweighted scores shows the modular monolith as the best choice, followed closely by microservices. The current monolithic architecture trails well behind with the most points.

Table 4. H-1 Software Architecture Decision Matrix with weighted ranking

Decision Matrix													
		Cost			Simplicity				Fault			Scores	
	Criteria →	Initial	O&S	Performance	Initial Build	O&S	Agility	Extensibility	Tolerance	Security	Testability	(Lower is Better)	
Options ↓	Criteria Weight	3	2	2	1	1	1	2	1	2	1	Unweighted	Weighted
Current architecture	Unweighted Ranking	1	3	1.5	1	3	3	3	3	3	3	24.5	
	Weighted Ranking	3	6	3	1	3	3	6	3	6	3		37
Modular Monolith	Unweighted Ranking	2	1.5	1.5	2	1.5	1.5	1.5	1.5	1.5	1.5	16	
	Weighted Ranking	6	3	3	2	1.5	1.5	3	1.5	3	1.5		26
Microservices	Unweighted Ranking	3	1.5	3	3	1.5	1.5	1.5	1.5	1.5	1.5	19.5	
	Weighted Ranking	9	3	6	3	1.5	1.5	3	1.5	3	1.5		33

Adding the weighting increases the gap between the architecture options. The modular monolith remains as the best choice with the lowest score. Microservices remains in second place, though the gap between first and second place has doubled. The gap between microservices and the current architecture has narrowed.

L. SENSITIVITY ANALYSIS

Mathematically, microservices is incapable of achieving the lowest score, weighted or unweighted. Microservices can tie with the modular monolith for lowest score if startup cost, performance, and initial build simplicity are ignored completely. However, the Table 5 shows one possible set of weights to achieve a tie between the modular monolith and microservices. Changes to the weights from the original decision matrix are shown in red.

Table 5. Decision matrix with microservices as tied for best score

		Cost			Simplicity				Fault			Scores	
	Criteria →	Initial	O&S	Performance	Initial Build	O&S	Agility	Extensibility	Tolerance	Security	Testability	(Lower is Better)	
Options ↓	Criteria Weight	0	3	0	0	2	2	2	1	2	1	Unweighted	Weighted
Current architecture	Unweighted Ranking		3			3	3	3	3	3	3	21	
	Weighted Ranking		9			6	6	6	3	6	3		39
Modular Monolith	Unweighted Ranking		1.5			1.5	1.5	1.5	1.5	1.5	1.5	10.5	
	Weighted Ranking		4.5			3	3	3	1.5	3	1.5		19.5
Microservices	Unweighted Ranking		1.5			1.5	1.5	1.5	1.5	1.5	1.5	10.5	
	Weighted Ranking		4.5			3	3	3	1.5	3	1.5		19.5

If the criteria weights are changed to six for startup cost and initial build simplicity and one for O&S cost, extensibility, and security, the current architecture achieves the best weighted score. Table 6 shows weights that result in the current architecture achieving the best score.

Table 6. Decision matrix with current architecture as best score

Decision Matrix													
		Cost			Simplicity				Fault			Scores	
	Criteria →	Initial	O&S	Performance	Initial Build	O&S	Agility	Extensibility	Tolerance	Security	Testability	(Lower is Better)	
Options ↓	Criteria Weight	6	1	2	6	1	1	1	1	1	1	Unweighted	Weighted
Current architecture	Unweighted Ranking	1	3	1.5	1	3	3	3	3	3	3	24.5	
	Weighted Ranking	6	3	3	6	3	3	3	3	3	3		36
Modular Monolith	Unweighted Ranking	2	1.5	1.5	2	1.5	1.5	1.5	1.5	1.5	1.5	16	
	Weighted Ranking	12	1.5	3	12	1.5	1.5	1.5	1.5	1.5	1.5		37.5
Microservices	Unweighted Ranking	3	1.5	3	3	1.5	1.5	1.5	1.5	1.5	1.5	19.5	
	Weighted Ranking	18	1.5	6	18	1.5	1.5	1.5	1.5	1.5	1.5		52.5

M. CONCLUSION

Based on the assessed needs of the program office and H-1 fleet, the modular monolith is the preferred choice among the three potential software architectures. Many software architectures styles exist and any upgrade to a well-structured software architecture would help tremendously in most areas of H-1 software acquisition. The modular monolith is capable of doing so for the least expense and the fastest timeline while combining the strengths of various architecture types (Su & Li, 2024). A

microservices architecture would likely be much more expensive and harder and slower to implement. Any advantages of a microservices architecture are unlikely to outweigh the expense and time to implement.

The advantages of a modular monolith over the current architecture are the long term cost savings and ease and speed to upgrade. The initial cost of upgrading may be recouped after several years from cost savings found in coding time (less money required to pay contractor to upgrade the system in the future) and test costs. The modular monolith is also less likely to deploy with errors due to it being easier to upgrade and test. Doing nothing with the software is likely to cost the program office more money in the long run and mean that needed capability is delivered to the fleet slower and with more errors.

The sensitivity analysis shows that microservices would not be a great fit for the H-1 mission computer regardless of priority factors. The best weighting microservices can achieve compared to the other two types is a tie if some factors are ignored. In that case, it would make most sense to then look at those factors and choose another architecture. The sensitivity analysis also shows that staying with the current architecture is most appealing when initial cost and build simplicity are significantly more important than all the other factors. For most other weight selections, the modular monolith remains the best choice.

V. SUMMARY AND RECOMMENDATIONS

The major drawback to the current H-1 software is the difficulty of updating: the tightly coupled code creates difficulty any time updates are attempted (Tran & Schneider, 2024). Developers must carefully change or add code and then check and test the entire software ecosystem for unforeseen consequences, such as a now-inoperative primary system display (T. Trepanier, email to author, October 11, 2024). Both the contractor and developmental test squadron conduct these checks to ensure errors are not present (W. Cosgrove, email to author, April 9, 2024; J. Hurst, email to author, May 17, 2024; J. Tran, email to author, May 16, 2024). Until recently, all testing had to be completed on the aircraft, which increased the cost of said tests (W. Cosgrove, email to author, April 9, 2024). If the software was partitioned, only the changed parts would need testing, saving time and money. If the number of hours required to test were halved, the program would save approximately $40,000 for each new software build. Upgrading the mission computer software should be a high priority for the program office to see long term cost savings for adding features and deploying them faster to the warfighter.

A. RESEARCH CONCLUSIONS

The primary research question stated in Chapter I is what is the best option for upgrading the H-1 mission computer software to better align with current software practices, decrease cost and time to test, and increase capability and speed to the fleet? Assuming that the H-1 program office, the H-1 fleet, and Headquarters Marine Corps weigh cost, schedule, and performance as this study does, a modular monolith upgrade is the most appropriate architecture for the H-1 mission computer. Of the two upgrade options addressed in this study, it has the lowest cost, fastest build time, and highest performance. In the long term, it will likely save the program office time and money in delivering needed capability to the fleet.

The secondary questions (How long might it take to upgrade the software? What is a fair estimate of the cost for a contractor to conduct the upgrade? Would hiring an outside software architect and/or consultant to oversee the transition benefit the program office? How might the program office maximize utility to the fleet while minimizing cost

during the transition?) are more technical or specific in nature than this study seeks to answer and are good candidates for future research.

B. H-1 PATH FORWARD RECOMMENDATION AND FOLLOW-ON RESEARCH

Based on the analysis, it is recommended that PMA-276 work toward a modular monolithic architecture for the mission computer. Any improvement of the current code base or new architecture would help long term, but a modular monolith would be the best option for cost, performance, and schedule reasons. As software developers find new ways to structure and code, it is also recommended to stay abreast of industry standards and incorporate them as able.

The exact means of breaking down the current unstructured monolith into a modular monolith will need to be explored. As there are infinite ways to arrive at the same user interface, the coders and software architects will need to smartly untangle the software, modularize, and put it back together in the way they most see fit under the supervision of the program office. The program office needs to set clear and specific requirements for the contractor to meet in order to best serve H-1 pilots and aircrew.

Cyber-security should remain top of mind for H-1 software upgrades as required of all DOD software programs (OUSD[A&S], 2020). The program office and the contractor need to ensure cyber-security capabilities are as robust and current as possible. As threats from hostile actors develop more sophisticated attack capabilities, H-1s need to be agile and resilient to prevent cyberattacks.

C. RECOMMENDATIONS FOR FUTURE RESEARCH AND PLATFORMS

Regardless of platform or size of software, the Department of Defense should more forcefully incentivize and assist programs to update legacy architectures to modularized software to the maximum extent practical. Especially for those platforms that experience problems similar to the H-1 mission computer, a modular monolith or microservices architecture would help program offices be more responsive to the needs of the users, minimize rework, and save time and money on testing efforts.

During research, multiple helicopter platforms were found to have similar issues as H-1, such as difficulty updating software and a requirement to test the entire system when changes are made (D. Backlund, email to author, April 15, 2024; M. Cecchini, email to author, April 4, 2024; D. Feddersen, email to author, April 5, 2024; T. Roberts, Mission Systems Lead, interview with the author, April 9, 2024). For future systems, modularity must be incorporated into systems. The commercial space provides a potential idea for how to accomplish this: operating software with applications. With an overall backbone, like iOS, new applications could be developed, tested, and deployed quickly in response to new threats or capabilities.

Another method to modularize is to separate the computers entirely from one another and minimize the communications between them. The more separate the computers and the less they need to communicate, the more cyber-secure each can be. Additionally, development can be faster for less cost, and testing becomes easier. For instance, if a future aircraft has one computer for managing and running the flight safety and management systems (such as engines, electrical power generators, and transmissions) and one to run mission systems (such a communications, sensors, weapons, and threat detection and countermeasures), they can be developed, tested, and deployed separately. Additionally, the flight management system can be developed along with the aircraft in a more traditional or waterfall method while the mission systems use an Agile approach throughout the life of the aircraft (Dunlap, 2024). With entirely separate systems, the flight-critical portion can be even more cyber-secure than the mission systems, ensuring that even if the mission computer is compromised, the aircraft can safely remain flying.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Academic Accelerator. (n.d.). *Computer performance*. Retrieved November 28, 2023, from https://academic-accelerator.com/encyclopedia/computer-performance

Apple Inc. (2024, May 13). *Apple security releases*. https://support.apple.com/en-us/HT201222

Belcher, M. (2020, September 8). Breaking down the monolith. *Codurance.* https://www.codurance.com/publications/2020/09/08/breaking-down-the-monolith

Bell Textron Inc. (n.d.a). [AH-1Z *Viper* flying over water]. Retrieved March 21, 2024, from https://www.bellflight.com/products/bell-ah-1z

Bell Textron Inc. (n.d.b). [UH-1Y *Venom* firing a rocket]. Retrieved March 21, 2024, https://www.bellflight.com/products/bell-uh-1y

Bell Textron Inc. (2004). *UH-1Y pocket guide* [Fact sheet]. https://www.aviatorsdatabase.com/wp-content/uploads/2013/07/Bell-UH1Y.pdf

Casserly, M. (2024, February 24). *iOS versions: every version of iOS from the oldest to the newest*. Macworld. https://www.macworld.com/article/1659017/ios-versions-list.html

Collings Foundation. (n.d.). [Bell UH-1E Iroquois]. Retrieved March 21, 2024, from https://www.collingsfoundation.org/aircrafts/bell-uh-1e-huey/

Cybersecurity and Infrastructure Security Agency. (2021, February 1). *What is cybersecurity?* https://www.cisa.gov/news-events/news/what-cybersecurity

Defense Innovation Board. (2019, May 3). *Software is never done: refactoring the acquisition code for competitive advantage*. https://media.defense.gov/2019/May/01/2002126691/-1/-1/0/SWAP%20FLYER.PDF

Defense Innovation Board. (n.d.) *Software Acquisition and Practices (SWAP) study*. Retrieved March 21, 2024, from https://innovation.defense.gov/software/

Department of Defense. (2021, October 19). *DevSecOps playbook*. https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOps%20Playbook_DOD-CIO_20211019.pdf

Department of the Navy. (2013, March 20). *UH-1Y Naval aviation technical information product* (NTRP 3-22.4-UH1Y).

Department of the Navy. (2014, January 7). *AH-1Z Naval aviation technical information product* (NTRP 3-22.4-AH1Z).

Department of the Navy. (2015a, December 15). *AH-1Z Naval aviation technical information product* (NTRP 3-22.4-AH1Z).

Department of the Navy. (2015b, December 15). *UH-1Y Naval aviation technical information product* (NTRP 3-22.4-UH1Y).

Department of the Navy. (2019a, July 1). *AH-1Z Naval aviation technical information product* (NTRP 3-22.4-AH1Z).

Department of the Navy. (2019b, November 1). *UH-1Y Naval aviation technical information product* (NTRP 3-22.4-UH1Y).

Department of the Navy. (2022a, March 22). *UH-1Y Naval aviation technical information product* (NTRP 3-22.4-UH1Y).

Department of the Navy. (2022b, November 1). *AH-1Z Naval aviation technical information product* (NTRP 3-22.4-AH1Z).

Department of the Navy. (2022c, November 1). *UH-1Y Naval aviation technical information product* (NTRP 3-22.4-UH1Y).

Department of the Navy. (2023a, April 1). *AH-1Z Naval aviation technical information product* (NTRP 3-22.4-AH1Z).

Department of the Navy. (2023b, May 1). *UH-1Y Naval aviation technical information product* (NTRP 3-22.4-UH1Y).

Dhaduk, H. (2020, July 4). *10 software architecture patterns you must know about*. Simform. https://www.simform.com/blog/software-architecture-patterns/

Docker. (n.d.). *Docker desktop*. Docker. Retrieved 24 May, 2024, from https://www.docker.com/products/docker-desktop/

Dunlap, J. (2024). *Innovation in software acquisition: The good, bad, and ugly* [Unpublished manuscript]

Fardink, P. J. (2016, September–October). Huey turns 60: A retrospective review of the UH-1's remarkable military service. *VERTIFLITE, 62*(5), 50–52. https://vtol.org/files/dmfile/50-52HueybyFardinkSO162.pdf

Fernandez, T. (2023, February 14). 12 ways to improve your monolith before transitioning to microservices. *Semaphore*. https://semaphoreci.com/blog/monolith-microservices

Foote, B., & Yoder, J. (1997, September). *Big ball of mud* [Paper presentation]. Fourth Conference on Patterns Languages of Programs, Monticello, IL, United States. https://www.cin.ufpe.br/~sugarloafplop/mud.pdf

Ford, N., Richards, M., Sadalage, P., & Dehghani, Z. (2022). *Software architecture: The hard parts*. O'Reilly. https://learning.oreilly.com/library/view/software-architecture-the/9781492086888/copyright-page01.html

Gupta, P. (2020 October 29). U*nderstanding the modular monolith and its ideal use cases*. TechTarget. https://www.techtarget.com/searchapparchitecture/tip/Understanding-the-modular-monolith-and-its-ideal-use-cases

IBM. (n.d.). *What are microservices?*. IBM. Retrieved May 24, 2024, from https://libguides.nps.edu/citation/apa#website

Light Attack Helicopter Program. (2024). *Capability defect package tracker backlog* [Unpublished raw data].

Marine Light Attack Helicopter Squadron 267. (n.d.). *Unit history.* Marine Light Attack Helicopter Squadron 267. Retrieved April 4, 2024, from https://www.3rdmaw.marines.mil/Units/MAG-39/HMLA-267/

Merson, P. (2015 November 5). Microservices beyond the hype: What you gain and what you lose. *Software Engineering Institute Blog*. https://insights.sei.cmu.edu/blog/microservices-beyond-the-hype-what-you-gain-and-what-you-lose/

Milić, M., & Makajić-Nikolić, D. (2022). Development of a quality-based model for software architecture optimization: A case study of monolith and microservice architectures. *Symmetry*, *14*(9), 1824–1860. https://doi.org/10.3390/sym14091824

Naval Air Systems Command. (2006, May 1). *Preliminary NATOPS flight manual Navy model UH-1Y helicopter* (NAVAIR 01-110HCG-1)

Naval Air Systems Command. (2008, May 1). *Preliminary NATOPS flight manual Navy model UH-1Y helicopter* (NAVAIR 01-110HCG-1)

Naval Air Systems Command. (2009, April 15). *NATOPS flight manual Navy model UH-1Y helicopter* (NAVAIR 01-110HCG-1)

Naval Air Systems Command. (2012, September 1). *NATOPS flight manual Navy model UH-1Y helicopter* (NAVAIR 01-110HCG-1)

Oakley, S. (2023). *Defense software acquisitions: Changes to requirements, oversight, and tools needed for weapons programs* (GAO-23-105867). Government Accountability Office. https://cle.nps.edu/access/lessonbuilder/item/335588/group/305d9ba8-2eaf-4d7c-b903-f9faa216cd70/Week%206/Defense%20Software%20Acquisition.pdf

Office of the Under Secretary of Defense for Acquisition and Sustainment. (2020, October 2). *Operation of the software acquisition pathway* (DOD Instruction 5000.87). Department of Defense. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF?ver=virAfQj4v_LgN1JxpB_dpA%3D%3D

Office of the Under Secretary of Defense for Acquisition and Sustainment. (2022, June 8). *Operation of the adaptive acquisition framework* (DOD INSTRUCTION 5000.02). Department of Defense. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500002p.pdf?ver=2020-01-23-144114-093

PC Magazine. (n.d.a). Deployment. Retrieved April 13, 2024, from https://www.pcmag.com/encyclopedia/term/deployment

PC Magazine. (n.d.b). Extensible. Retrieved April 13, 2024, from https://www.pcmag.com/encyclopedia/term/extensible

PC Magazine. (n.d.c). Reliability. Retrieved April 13, 2024, from https://www.pcmag.com/encyclopedia/term/reliability

PC Magazine. (n.d.d). Security. Retrieved April 13, 2024, from https://www.pcmag.com/encyclopedia/term/security

PC Magazine. (n.d.e). Spaghetti code. Retrieved November 28, 2023, from https://www.pcmag.com/encyclopedia/term/spaghetti-code

PMA-276 Software and Avionics Test. (2024, April 10). *Software and Avionics Test (SWAT)* [Presentation]. PMA-276 Rollups, Lexington Park, MD, United States.

Project Management Institute. (2021). *The standard for project management and a guide to the project management body of knowledge* (7th ed.) [PDF version]. https://www.pmi.org/pmbok-guide-standards/foundational/pmbok#download

Richards, M. (2022, August). *Software architecture patterns, 2nd edition*. O'Reilly. https://learning.oreilly.com/library/view/software-architecture-patterns/9781098134280/

Robinson, H. (2017, March 29). *Let's get digital: 2/6 Marines train with MAGTAB at WTI*. Defense Visual Information Distribution Service. https://www.dvidshub.net/news/229462/lets-get-digital-2-6-marines-train-with-magtab-wti

Semaphore. (2022). *Transitioning from monolith to microservices handbook: Converting monoliths to the microservice architecture* [PDF version]. https://semaphoreci.com/wp-content/uploads/2022/09/Monolith_to_Microservices_Handbook-1.pdf

Shanthi, R. (2018). *Computer architecture*. https://www.cs.umd.edu/~meesh/411/CA-online/index.html

Smith, S. (2024 February 21). *Introducing modular monoliths: The Goldilocks architecture*. Ardalis. https://ardalis.com/introducing-modular-monoliths-goldilocks-architecture/

Su, R., & Li, X. (2024, January 22). *Modular monolith: Is this the trend in software architecture?* ArXiv. https://arxiv.org/pdf/2401.11867.pdf

Tran, J., & Schneider, B. (2024, February). *Capability accelerator: Partitioned architecture* [Presentation]. H-1 Fleet Sync, Oceanside, CA, United States.

U.S. Army. (n.d.). [AH-1G flying low]. Retrieved May 27, 2024, from https://www.thisdayinaviation.com/tag/bell-ah-1g-cobra/

U.S. Navy. (2021, October 22). *AH-1Z Viper and UH-1Y Venom helicopters*. https://www.navy.mil/Resources/Fact-Files/Display-FactFiles/Article/2160217/ah-1z-viper-and-uh-1y-venom-helicopters/

Vietnam Veterans Memorial Fund. (n.d.). *Helicopters*. Retrieved May27, 2024, from https://www.vvmf.org/topics/Helicopters/#:~:text=The%20variety%20of%20roles%20bred,Monument%20in%20Arlington%20National%20Cemetery

Acquisition Research Program
Naval Postgraduate School
555 Dyer Road, Ingersoll Hall
Monterey, CA 93943

www.acquisitionresearch.net