Software modernization for the UFS

A position paper

Hendrik L. Tolman

Senior Advisor for Advanced Modeling Systems at the National Weather Service
Hendrik.Tolman@NOAA.gov

July 2025

DOI: 10.25923/gfbx-pk53

This page is intentionally left blank

Table of Contents

This page is intentionally left blank

1. Introduction

This report is a position paper on the need for software modernization for the Unified Forecast system (UFS). It is written by the author in his role as Senior Advisor for Advanced Modeling Systems, and is intended both as an inventory of ongoing activities and to encourage the development of more formal projects associated with this topic. It is the result of a set of interviews with key partners inside and outside of NOAA, and as such is informational. It should not be construed as a fully vetted approach or strategy.

Furthermore, it focuses on traditional physics-based numerical models, not on more recent data-driven AI/ML models. Even as data-driven models as well as AI/ML post- and pre-processing tools and code emulators are accelerating toward operational use, it is expected that the core physics-based models will remain relevant in the foreseeable future. Note that AI/ML tools for modernization and optimization of the traditional models are relevant for the present assessment.

The general NOAA UFS strategy for modeling seeks to focus NOAA modeling on a small set of community models that is sufficiently focussed to create critical mass for its support while at the same time is sufficiently diverse to broadly cover NOAA's mission. Moreover, it seeks to unify software approaches for research and operations, and to build a broad modeling community with partners including but not limited to government, academia, and the private sector.

NOAA first looked into a community modeling approach with a Unified Modeling Working Group under the NOAA Research Council[1] (Link et al. 2017a, b). NOAA first formalized a UFS approach in a vision document and a roadmap for NOAA operations published in Tolman and Cortinas (2020a,b)[2], which included a formal sign-off of the Assistant Administrators of most NOAA Line Offices. Since then, the UFS approach has been documented and defined in a set of four BAMS papers. Jacobs (2021) addresses the power of community modeling. Uccellini et al. (2022) presents NOAA's support of community modeling through the Earth Prediction Innovation Center (EPIC). Alves et al. (2023) presents examples of accelerated transition of research to operations in a UFS approach. Tolman (2025) presents lessons learned for community modeling. Moreover, the UFS approach is supported in the LEGEND Act (15 U.S.C. § 8512a), in the first ever NOAA Modeling Strategy (Morgan et al, 2024), in a NOAA Administrative Order 201-118 entitled "*Software Governance and Public Release Policy*", and in the SHARE IT act (Public Law No: 118-187).

In the UFS community some initial choices were made to couple component models to each other with wrappers around these models. This enabled coupled modeling while simultaneously limiting the impact of the UFS approach on existing community-based component models. In the selected approach model coupling is executed at the Application Programming Interface (API) level. This implies that community models are "wrapped" using the Earth Systems Modeling Framework approach (ESMF, Collins et al. 2005) augmented with an interoperability layer

[1] Now known as the NOAA Science Council.
[2] These documents were developed in 2016-2018. Sign-off and publication were delayed to assure full compatibility with the burgeoning EPIC effort.

defined by the National Operational Prediction Capability (NUOPC)[3]. This approach enables communication between UFS component models, but is generally not intrusive to the component models. This has left governance of the individual component models mostly untouched, with minimal impact on existing communities.

The UFS has reached a level of maturity where many applications are now supported at least partially by EPIC, where the production suite at NOAA is shrinking in terms of major applications (23% reduction by summer of 2024), and where we are now developing basic capabilities as well as targeted operational implementations with much larger teams. Moreover, environmental component model coupling has reached a level of maturity that virtually all UFS releases consider coupled modeling approaches. In the context of developing more advanced modeling techniques, there is a need for upgrading the component models to allow for more integrated model coupling. Such an approach is aligned with a general need to update component codes, many of which have seen decades of use and represent older coding techniques and best practices. Software modernization is also becoming urgent today as previously established code architectures are not generally optimal for new hardware architectures and exascale computing. Note that these issues are not unique to the UFS (e.g., Govett et al. 2024).

Discussing the need for software modernization is not new. NOAA has had a Software Engineering for Novel Architecture (SENA) program for decades. For instance, the original development of WAVEWATCH III ® (henceforth denoted as WW3) in the mid 1990s was fully funded out of this program. This program, however, has mostly been focusing on “one-off” projects like building the latter model. Only recently has NOAA through SENA focused more systematically on software modernization, in particular for Exascale computing (Govett et al., 2024). NOAA code optimization in general has been reactionary for several decades, where the codes are optimized for the “hardware of the day”, mostly focusing on operational applications. Moving into the GPU era, there have been some pilot projects world-wide (see more detailed discussions below) but few if any systematic approaches. With that, NOAA should not do this alone, but focus on the broader, evolving UFS partnership. This position paper is intended to encourage and accelerate this process. It is not intended to provide a comprehensive strategy at this time, but instead outlines a path to such a strategy.

This position paper starts with this introduction, followed by a general framing of the issues in Section 2. Section 3 documents the sampling / interview strategy used here, pointing to outcomes of interviews as gathered in Appendix A. A summary and suggested action items are gathered in Section 4.

[3] https://earthsystemmodeling.org/nuopc

2. Framing issues

Operational and research modeling has been part of NOAA's mission for well over half a century. Due to the long history, it is unavoidable that at least some of the modeling approaches are outdated, and could benefit from software modernization. This need for software modernization is driven by two top level considerations.

The first top-level driver applies to the entire UFS community and focuses on the ongoing change in hardware architecture. This change mostly revolves around the move from CPU to GPU processor architecture, and the more general move to exascale computing. From a human perspective this change is interesting as it comes at the end of over two decades of slowly changing hardware design. The previous major transition to new hardware occurred nearly 25 years ago as the first massively parallel computers started to replace the previous dominant High Performance Computing (HPC) vector computers. This architecture change required a re-factoring and re-write of many codes. The generation of programmers that did this work is now mostly retired, so that there is limited hands-on experience with systematic code refactoring at this level and for this purpose.

The second top level driver focuses more on NOAA and considers the transition of NOAA to a UFS approach to modeling as outlined in the Introduction. Within the UFS, there are two main reasons for software modernization. The first is the need for a community model to be highly portable over a wide range of hardware architectures. Portability becomes a much bigger issue with the recent increase in diversity in hardware as mentioned above. Portability in the newer environment becomes more difficult as it appears that different hardware architectures imply different optimal data layouts. From my personal experience with the WW3 wind wave model (e.g., Tolman 2025), it is both interesting and annoying to see that optimization for GPUs drives us back to data structures we initially used in the 1990s, but that were abandoned in the stable parallel compute architecture era. Another reason for software modernization in the UFS is the need for a next generation coupling approach. The UFS is outgrowing its initial API level coupling approach as described in the Introduction, and in general requires component models to become more object oriented so that parts of one model can be called independently in a coupled model to allow for more closely coupled systems.

The discussion about software modernization has been going on for at least a decade. An incomplete list of some of the topics and related questions driving this discussion is presented below (the order of topics does not imply ranking or importance).

- **Open source and open science**: Since the first internal NOAA assessments of the need for open source (code is available) and open science (we take code from many others), the open UFS approach is accepted in NOAA, formalized in law, in a NOAA administrative Order, and in the NOAA modeling strategy (see Introduction). These choices are foundational and well established, and are presently not a subject for discussion.

- **Fault tolerance**: Exascale computing initially moved to hyper-parallel (CPU) systems, where failure of a single CPU or parallel process tentatively results in a failure of the entire model. Fault tolerance seeks to immunize massively parallel codes from such failure. Between making hardware more fault-tolerant at the operating system level, and by tentatively relying more on a smaller number of GPUs rather than on monolithic CPU-only machines to achieve exascale capabilities, discussions more recently have moved away from focusing on fault tolerance.

- **Load balancing**: Traditionally, load balancing principles are built into the code stack, with environmental settings optimized for specific applications on specific hardware. More advanced techniques consider dynamic load balancing as part of the base code stack, both for code efficiency and to increase fault tolerance. An example of such an approach can be found in the Uinta software of the University of Utah.

- **Optimization**: Code optimization at the software stack level has historically been performed for the hardware available to its users. Due to the slow development and relative homogeneity of hardware architecture over the last two decades, performance optimization has been fairly portable between different computers. The diversity introduced by the CPU - GPU and hybrid computer architectures has recently reduced performance portability. In particular, different computer architectures appear to favor different data layouts. This raises the question if we should maintain different codes for different processors, or if we should move toward automated custom code generation for each computer, based on a higher level description of the problem (i.e., using Domain Specific Languages, DSLs).

- **Coupling**: The API-level approach to coupling of component models in a more integrated environmental approach as presently used in the UFS allowed for effective development of component models and coupling approaches at the same time. At the present level of maturity of the UFS this coupling approach now starts to seriously impede load balancing and code optimization. Closer coupling where parts of the component models can be called outside of a high-level API framework requires a more object-oriented approach to the architecture of component models.
    - Note that a functional plug-compatibility is needed here rather than a formal object oriented approach. This will require a deliberate focus on data structures used in the code.
    - Note that even in a closely coupled UFS environment it will remain essential for rapid development and for “downstream” uses of models to still be able to compile the software stack of component models into a stand-alone application with relevant input from data rather than from coupled models.
    - Note that the term coupling is also used in terms of general software engineering, in particular in the context of coupling and cohesion in software design[4]. For the

[4]See, for instance, https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/

UFS, these aspects of coupling have a large impact on the cost of testing code during development in an open-science workflow.

- **Code quality**: working in a community environment requires high-quality code following coding standards, with documentation, and with a well-developed and supported workflow to contribute code. The latter implies (amongst others) a well developed development workflow with automated unit / regression testing approach. Note that some members of the modeling community fear that contributions of a large community to a code could result in poor general code quality. Many in the community (including the present author) have experienced the opposite; well-managed community development with its many eyes on the code and with porting to many compilers and hardware architectures tends to result in high-quality code with less (hidden) bugs.
    - **Code Development Workflow:** Code development approaches are important for modern software development. For the UFS, this is presently either left at the component model team level, or managed by EPIC.
    - **Security Considerations:** Software security has become a significant concern in modern software development. As the UFS codes are distributed as source codes rather than executables, security risks have generally not been considered to be significant, and can be mostly avoided by a requirement that all UFS code is provided as source code, and that executable contributions are not accepted. More recently, however, memory leakage of compiled codes has been identified as a potential security issue[5].

- **Knowledge and expertise**: 25 years ago environmental (weather) modeling was one of the core applications for HPC, and the physical scientists became the first software engineers on these systems. Nowadays, models are more complex and the field of HPC software engineering is more complex, which has resulted in a need for both specialized software engineers and specialized physical scientists to tackle software as a team. In such a team approach it is important to maintain thorough scientific understanding of the codes while rapidly adapting codes to diverse and varied computing platforms and modern coding practices.

- **Knowledge, expertise and optimization**: combining the two bullets above, it is interesting to observe that in the late 1990s optimized codes could perform up to 50% of the nominal performance of the processor, measured as Floating Point Operations relative to a standard LINPACK benchmark test. Using this metric present operational codes at for instance NOAA and ECWMF now operate at only 3-4% of peak performance of the hardware. Data movement on processors now limits the amount of compute cycles that can be performed. Architectures moved away from enabling typical environmental problems (less work on more data) to image processing and AI (more work on less data).

[5] See, e.g., https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF

    - A structural approach to mitigate this historical drop in efficiency requires a custom processor design for UFS-like models. With environmental modeling now representing only a minor part of the processor market, it is not realistic to expect that this is an economically viable approach for the UFS.
    - Conversely, the small efficiency of present codes presents an opportunity that previously did not exist. Increasing the performance from 3-4% to 6-8% effectively corresponds to a monetary value equivalent to the value of the entire HPC budget. This implies that a major investment in code optimization is now much more justifiable than it would have been several decades ago.

- **Moving away from Fortran**: Many of the UFS component models are still written in Fortran. Historically, Fortran has been used as it resulted in the most efficient HPC codes. As this no longer seems to be the case, and as the pool of Fortran programmers as well as compilers seems to shrink, code modernization should address a potential transition to more modern languages like Python and C++. Our present experience with hybrid-languages codes in the UFS indicates that language changes can be done incrementally on a module-by-module basis. Note that there are many AI/ML tools to help convert code to other languages.

3. Sampling ongoing activities

As mentioned in the Introduction this position paper uses a sampling of opinions and activities related to, or relevant for, UFS code modernization. The sampling is achieved by informal interviews / discussions with relevant (potential) partners for the UFS, with an attempt to broadly cover the intended UFS community. The coverage of the community in this section is intended to be representative, not complete. Hence, whether or not a specific group has been interviewed here is not a statement of their importance in the UFS. The order of the interviews in the subsections below represents the time line of the interviews, some of which were scheduled, and some of which were meetings of opportunity. These interviews loosely covered all topics outlined in Section 2, and summaries of the interviews are gathered in Appendix A.

The round of interviews and discussions associated with this report started with ongoing discussions with the UK Met Office specifically focused on the need to modernize and optimize the WW3 code, and to assure that associated work done by the UK Met Office would find its way into the formal code base. This discussion naturally moved toward a tag-up with EMC and EPIC on the status of software modernization at NOAA, and several meetings of opportunity with industry partners at the AGU and AMS annual meetings in December 2024 and January 2025. The assessment of the state of development then shifted to invited talks at the UFS Steering Committee meetings of both the DOE E3SM team and GFDL on their software modernization efforts. Within NOAA this left the SENA program and NCEP Central Operations (NCO) as obvious targets for interviews. Finally, interviews were conducted with NOAA's National Water Center (NWC) considering their recent systematic approach to modernizing the National Water Model (NWM). The discussions documented in the Appendix are augmented with outcomes of previous discussions with the NCAR CESM and MMM groups, and with considerations from previous discussions about alternative programming languages.

This page is intentionally left blank

4. Moving forward

The next step in this assessment was to identify common issues and action items by mapping the opinions and experiences documented in Appendix A to the general issues identified in Section 2. This is done here in two steps; Section 4.1 presents the above mapping, and Section 4.2 presents suggested action items. Note that as this is a position paper by the author, the action items do not represent formal decisions or strategies adopted by NOAA.

4.1. Summary of discussions

The experiences and opinions summarized in Appendix A have been summarized here as follows.

- **Open Source and Open Science:** There was no disagreement with the statement that open science is here to stay. Implementing efficient open-science workflows does remain a work in progress in scientific programming.

- **AI/ML**: Even though AI/ML as a replacement of traditional physics-based numerical modeling was placed outside the scope of the present assessment, modernization of the traditional software will benefit greatly for AI/ML tools developed specifically for this purpose (See Appendix A.3).

- **Fault Tolerance:** This is presently not considered a main issue with coding on supercomputers, with the exception of a need for modern codes to produce clear failure messages.

- **Load balancing**: There are two main issues associated with load balancing; one is the need for code restructuring associated with general code optimization, and the other is the need for re-assessing coupling approaches for component models used in the UFS. Both topics will be addressed in the following two bullets.

- **Optimization**: The interviews and discussions identified three focal areas of optimization; the first is general code optimization, the second is the selection of the programming language and the third is the use of utilities in general and Domain Specific Languages (DSLs) in particular.

  Data structures play a crucial role in optimizing code performance, particularly in complex modeling systems like the UFS. Different hardware architectures, such as CPU versus GPU, often require specific data layouts for efficient processing. Older codes tend to use global data structures that are no longer efficient both due to the growing size of compute domains, and due to the changing requirements for efficiency on evolving hardware architectures. Systematically re-designing a data structure in an established model represents a major coding effort, which could justify other major code changes such as a transition to a new programming language.

Switching from older languages like Fortran to languages such as Python and C++ can improve code efficiency, particularly in advanced modeling systems. The latter languages often provide better support for new hardware architectures like GPUs, leading to more efficient code. Moreover, these languages offer access to extensive libraries and tools that can streamline development and enhance performance. Note that the benefit from a change in language is likely to be much smaller than the benefit from going to more suitable data structures, irrespective of the language. Conversely, the need for a full code rewrite to drastically change data structures could well be considered as an opportunity to move away from Fortran at the same time.

Experiments of various groups with changing to more suitable data structures and at the same time changing languages show speed-ups of codes on CPUs in the factor 2-5 range, and for GPUs in the factor 10-30 range, with the understanding that these numbers are relative to Fortran codes that have not been recently optimized for the newer hardware. Note that other languages like Julia and Rust have been mentioned, but have not been used yet in component models relevant for the UFS. Note, furthermore, that Python and C++ are used in operations, but Julia and Rust are not yet.

Various tools and Domain Specific Languages (DSLs) are emerging to aid in software modernization, particularly in the context of high-performance computing and scientific modeling. As with languages, there is no one preferred set of tools. GT4Py is used by various organizations with Python. DSLs used with C++ are Kokos (DOE), Psyclone (UK Met Office), DaCe + NDSL (GFDL) as documented in Appendix A. Earthkit of ECMWF contains both tools and DSL approaches. Complementary to these approaches, tools like GitHub Copilot and Codeium leverage AI for code completion and understanding, while Moderne automates large-scale refactoring and modernization.

Note that the deployment of such tools on NCO's FISMA[6]-high security computers introduce unique challenges. Because the UFS is intended to support both research and operations, it is important to choose tools and DSLs that are both sufficiently cutting-edge to advance research, while being sufficiently mature to implement on a FISMA-high operational computer.

- **Coupling of component models**: Coupling modernization for the UFS is a crucial step towards enhancing its efficiency. The initial ESMF/NUOPC API-level coupling approach, while effective for initial development, now presents limitations for load balancing and associated code optimization. Note that this issue is related to the issues with global versus localized data structures discussed above.

  As the UFS matures, a shift towards more integrated coupling of component models is necessary, enabling component models to interact more directly. This entails moving away from global API-level coupling and adopting a more functional, object-oriented approach. Technologies like the Flexible Modeling System (FMS), used within GFDL's component models, and the Basic Modeling Interface (BMI), employed by the National Water Model offer examples of closer and more localized component model coupling.

[6] Federal Information Security Management Act

- **Code quality**: There appears to be consensus among the interviewed groups that CI/CD (Continuous Integration/Continuous Deployment) and DevOps approaches have a profound positive impact on open source code quality. With the UFS being an ecosystem of many communities, the principles of such approaches rather than a dictate of associated workflows is appropriate for the UFS.

  A key element of such approaches is automating testing as part of the development workflow, which in turn is strongly associated with the level of coupling and coherence of code from a general software perspective (see discussion of coupling in Section 2). Ideally, associated test suites are (automatically) applied at a granular level of the development. The UFS contributors observed that many codes have been "retrofitted" with functional regression testing. Such testing is becoming prohibitively expensive. There appears to be consensus that code management at the UFS needs to transition rapidly to the much cheaper unit testing (associated with general decoupled coding practices) as the foundation of the development workflow, whereas there may be value in retaining more traditional regression tests for occasional testing, and to compare model options at a foundational physical level.

  Software security is crucial for protecting software and data from unauthorized access and vulnerabilities. It involves secure design, development, and maintenance throughout the software lifecycle. Traditionally, UFS type software has been considered "safe" as source codes rather than executables are shared. Note that this view appears to shift as executable codes generated by the UFS may have security vulnerabilities associated with memory inconsistencies. In the interviews the latter was mentioned as a benefit of using the Rust programming language, which is designed around memory integrity.

- **Knowledge and expertise**: There is limited consensus on the knowledge and expertise being an issue in particular in the context of using Fortran. Some organizations see this as a systemic issue, others see this as a training issue. Concerns about the diminishing support of the community in particular for upkeep of Fortran compilers is a rising concern that could become a Continuity of Operations (COOP) issue (see next bullet). In a more general sense several organizations are concerned about single points of expertise / failure in particular in the contexts of Artificial Intelligence and Machine learning becoming an integral part of environmental modeling.

- **Moving away from Fortran**: Moving away from Fortran is starting to happen, both due to a reduced performance of its compilers, and due to a potential of losing high-quality compilers with a shrinking user group for this language. Presently the two predominant choices of alternative languages are Python using GT4Py, and C++. For both languages Domain Specific Languages (DSL) have been used for optimization across hardware solutions. More modern languages like Rust or Julia[7] or Rust[8] may be suitable for a systematic modernization. Julia has seen some recent high-performance applications in

[7] https://julialang.org, first released in 2012, version 1.0 released in 2018.
[8] https://www.rust-lang.org, sponsored since 2009 by Mozilla, version 1.0 released in 2015.

environmental modeling (e.g., Silvestri et al. 2024, Wagner et al. 2025). Presently we are not aware of mature Rust applications in this field, but Rust seems to have tools and features making it potentially very suitable for environmental modeling. For the UFS it is important to address the feasibility of using these languages including DSLs in an operational FISMA-high security environment.

4.2. Action Items

The present assessment highlights the absence of a single, dominant approach to software modernization within the UFS community. Moreover, given the diversity of components and contributors within the UFS ecosystem, a one-size-fits-all approach to software modernization is not only impractical but potentially detrimental, as it risks stifling innovation and failing to address the specific needs of individual component models and tools. Finally, the diversity of programming languages and modernization approaches within the component models and tools within the UFS is currently not a significant hindrance and effectively fosters innovation. All these observations necessitate a strategic approach focused on high-level guidance rather than rigid mandates. Moreover, actions should prioritize facilitating localized efforts by the responsible "owners" of UFS components.

With this, the following action items and recommendations have been distilled from the present assessment.

1) The UFS community should not provide a detailed software modernization strategy, but instead should start tracking modernization efforts in the UFS, preferably as part of the UFS inventory[9] that was started in 2024 and that is intended to be updated annually. A key benefit of this action will be that it identifies which parts of the UFS need additional attention with respect to software modernization. This inventory can be used by those who provide resources to the UFS development, operations, and maintenance to prioritize their efforts and funding.

2) Within the UFS concept, responsibilities for code management and hence modernization rest with the “owners” of the code, i.e. with the main contributors to component models, tools etc. Considering the experiences documented in this report, the owners are strongly encouraged to

    a) implement decoupled general software approaches with unit testing as a foundation in the community open-science workflow used for the specific element of the UFS,

    b) start or accelerate strategically moving away from Fortran,

    c) while assuring that other languages and DSLs, middleware and tools approaches are enabling both cutting edge research and are sufficiently mature for use on

[9] https://ufs.epic.noaa.gov/wp-content/uploads/2024/08/UIFCW_2024_poster.pdf

FISMA-high (operational) HPC systems. The latter will benefit from early and recurring interactions with NCO for NOAA operations.

Note that as the UFS is mostly a coalition of the willing, the UFS governance leadership can only encourage the owners of parts of the UFS to follow these recommendations.

3) An integral part of the code modernization needs to be the transition to the next generation coupling approach for component models. To start this process the UFS needs to task a Working Group to develop a draft plan, including

    a) the selection of a more closely coupled methodology that is rooted in a community open-source approach,

    b) an assessment if this will become the new approach, or an alternative approach next to the present ESMF / NUOPC approach,

    c) and an assessment of the feasibility of formal collaborations, as we had in the NWS- OAR - NCAR Memorandum of Agreement on coupling infrastructure that has supported the development of the present coupling capabilities in the UFS.

    It has been noted that *reducing* the close coupling from a general software architecture perspective and *increasing* close coupling from a model component coupling perspective may be conflicting development directions and therefore will require additional discussion and assessment.

This page is intentionally left blank

5. References

Alves, J., H. Tolman, A. Roland, A. Abdolali, F. Ardhuin, G. Mann, A. Chawla, and J. Smith, 2023: NOAA's Great Lakes Wave Prediction System: A Successful Framework for Accelerating the Transition of Innovations to Operations. *BAMS*, **104**, E837–E850.

Ben-Nun, T, J. de Fine Licht, A. N. Ziogas, T. Schneider, and T. Hoefler, 2019: Stateful dataflow multigraphs: a data-centric model for performance portability on heterogeneous architectures. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Article No.: 81, 1-14.[10]

Cikanek, H, P. Burke, D. Snowden, E. Clark, B. Gross, M. Ji, I. Stajner, H. Tolman, C. Alexander, S. Benjamin., J. Mahoney, W. Anderson, E. Burger, D. Carlis, J. Cortinas, S. Gopalakrishnan, T. Hamill, J. Whitaker, R. Webb, L. Wicker, and W. Lapenta, 2019: *A Vision Paper for the Earth Prediction Innovation Center (EPIC) Version 5.0.* NOAA Internal report.[11]

Collins, N., G. Theurich, C. DeLuca, M. Suarez, A. Trayanov,V. Balaji, P. Li , W. Yang, C. Hill, and A. da Silva, 2005: Design and Implementation of Components in the Earth System Modeling Framework, *International Journal of High Performance Computing Applications*, **19**, 341-350.

Dahm, J, E. Davis, F. Deconinck, O. Elbert, R. George, J. McGibbon, T. Wicky, E. Wu, C. Kung, T. Ben-Nun, L. Harris, L. Groner, and O. Fuhrer, 2022: Pace v0.1: A Python-based Performance-Portable Implementation of the FV3 Dynamical Core. *EGUsphere* 2022-943.[12]

Dahm, J, E. Davis, F. Deconinck, O. Elbert, R. George, J. McGibbon, T. Wicky, E. Wu, C. Kung, T. Ben-Nun, L. Harris, L. Groner, and O. Fuhrer, 2023: Pace v0.2: a Python-based performance-portable atmospheric model. *Geophys. Model Development*, **16**, 2719-2736.[13]

ECMWF, 2024: *Introducing Earthkit*. European Centre for Medium Range Weather Forecasting Newsletter, **179**.[14]

Govett, M., B. Bah, P. Bauer, D. Berod, V. Bouchet, S. Corti, C. Davis, Y. Duan, T. Graham, Y. Honda, A. Hines, M. Jean, J. Ishida, B. Lawrence, J. Li, J. Luterbacher, C. Muroi, K. Rowe, M. Schultz, M. Visbeck and K. Williams, 2024: Exascale Computing and Data Handling: Challenges and Opportunities for Weather and Climate Prediction, *BAMS*, **105**, E2385–E2404.

Hell, M, B. Fox-Kemper, and B. Chaperon, 2023: *Particle-in-CelL for Efficient Swell PiCLES; Towards coupling with Earth System Models*, CESM OMWG presentation.[15]

Hutton, E.W., M.D. Piper, and G.E. Tucker, 2020: The Basic Model Interface 2.0: A standard interface for coupling numerical models in the geosciences. *Journal of Open Source Software*, **5**(51), p.2317.

Jacobs, N. A., 2021: Open Innovation and the Case for Community Model Development. *BAMS*, **102**, E2002–E2011.

[10] https://doi.org/10.1145/3295500.3356173
[11] https://epic.noaa.gov/wp-content/uploads/2022/09/EPIC_Vision_Paper_noaa_29972_DS1.pdf
[12] https://doi.org/10.5194/egusphere-2022-943
[13] https://doi.org/10.5194/gmd-16-2719-2023
[14] https://www.ecmwf.int/en/newsletter/179/computing/introducing-earthkit
[15] https://www.cesm.ucar.edu/sites/default/files/2023-03/2023-cesm-omwg-m.hell_.pdf

Link, J. S., H. L. Tolman, E. Bayler, C. Holt, C. W. Brown, P. B. Burke, J. C. Carman, S. L. Cross, J. P. Dunne, D. W. Lipton, A. Mariotti, R. D. Methot, E. P. Myers, T. L. Schneider, M. Grasso, and K. Robinson, 2017a: *High-level NOAA unified modeling overview,* NOAA report.[16]

Link, J., S., H. L. Tolman and K. Robinson, 2017b: NOAA's Unified Modeling Strategy, *Nature*, **549**, 458-458.

Met Office, 2018: *The Next Generation Modelling Systems Programme*. Online publication.[17]

Met Office, 2021: *The Next-Generation Modelling System reaches a new milestone*. Online publication.[18]

Met Office, 2023. *LFRic - a modelling system fit for future computers*. Online publication.[19]

Morgan, M., D. Koch, B. Gross Brian, D.Carlis, P. Burke, H. Tolman, J. Vogt, J. Mahoney, J.Kazila, J. Dunne, C. Kongragunta, J. Carmen , M. Huang, K. Garrett, J. Pica, Y.-J. Kim, C.Stawit, H.Townsen, D. Barrie, Y. Jung, S. Gopalakrishnan, H.-S. Kim, C. Zhang, M.Poti, A.Mariotti, M. Supernaw, M. Brooks, M. Srinivasan and R. Dempsey, 2024: NOAA Modeling Strategy : strategic plan 2024-2033. Publication, United States. National Oceanic and Atmospheric Administration.[20]

Ogden, F., Avant, B., Bartel, R., Blodgett, D., Clark, E., Coon, E., Cosgrove, B., Cui, S., Kindl da Cunha, L., Farthing, M. and Flowers, T., 2021: The Next Generation Water Resources Modeling Framework: Open Source, Standards Based, Community Accessible, Model Interoperability for Large Scale Water Prediction. In AGU Fall Meeting Abstracts (Vol. 2021, pp. H43D-01).[21]

Paredes, E. G., L. Groner, S. Ubbiali, H. Vogt, A. Madonna, K. Mariotti, F. Cruz, L. Benedici, M. Bianco, J. VandeVondele, and T. C. Schulthess, 2023: GT4Py: High Performance Stencils for Weather and Climate Applications using Python. arXiv preprint arXiv:2311.08322.[22]

Silvestri, S., G. L. Wagner, C. Hill, M. R. Ardakani, J. Blaschke, J.-M. Campin, V. Churavy, N. C. Constantinou, A. Edelman, J. Marshall, A. Ramadhan, A. Souza, and R. Ferrari, 2024: Oceananigans.jl: A Julia library that achieves breakthrough resolution, memory and energy efficiency in global ocean simulations, arXiv [physics.ao-ph] 2309.06662.[23]

Tolman, H. L., 2025a: What makes a successful community model for research and operations? Lessons learned from WAVEWATCH III®. *BAMS*, EOR.[24]

Tolman, H. L., and J. Cortinas, 2020a: A strategic vision for NOAA's physical environmental modeling enterprise. Report, NOAA. 9 pp.[25]

Tolman, H. L., and J. Cortinas, 2020b: 2017-2018 roadmap for the production suite at NCEP. Report, NOAA. 35 pp. + Appendices.[26]

[16] https://doi.org/10.7289/V5GB2248
[17] https://www.metoffice.gov.uk/research/approach/modelling-systems/next-gen-modelling-systems
[18] https://www.metoffice.gov.uk/research/news/2021/gungho-and-lfric-10th-anniversary
[19] https://www.metoffice.gov.uk/research/approach/modelling-systems/lfric
[20] https://doi.org/10.25923/gggz-jb43
[21] https://ui.adsabs.harvard.edu/abs/2021AGUFM.H43D..01O/abstract
[22] https://arxiv.org/pdf/2311.08322
[23] https://arxiv.org/abs/2309.06662
[24] https://doi.org/10.1175/BAMS-D-24-0223.1
[25] https://doi.org/10.25923/0rac-9017
[26] https://doi.org/10.25923/kr4a-zy63

Trott, C. R., D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D Sunderland, B. Turcksin, and J. Wilke, 2022: Kokkos 3: Programming Model Extensions for the Exascale Era, *IEEE Transactions on Parallel and Distributed Systems*, **33(4),** 805-817.[27]

Ubbiali, S., L. Bonaventura, C. Kühnlein, and T. C. Schulthess, 2025: Exploring a high-level programming model for the NWP domain using ECMWF microphysics schemes. *Geoscientific Model Development*, ***18*(2)**, 529–559.[28]

Uccellini, L. W., R. W. Spinrad, D. M. Koch, C. N. McLean, and W. M. Lapenta, 2022: EPIC as a Catalyst for NOAA's Future Earth Prediction System. *BAMS*, **103**, E2246–E2264.

Wagner, G. L., S. Silvestri, N. C. Constantinou, A. Ramadhan, J.-M. Campin, C. Hill, T. Chor, J. Strong-Wright, X. K. Lee, F. Poulin, A. Souza,K. J. Burns, J. Marshall, and R. Ferrari, 2025: High-level, high-resolution ocean modeling at all scales with Oceananigans, arXiv [physics.ao-ph], 2502.14148.[29]

WPO, 2019: *The Earth Prediction Innovation Center (EPIC) Community Workshop Report.* NOAA Internal report.[30]

[27] https://doi.org/10.1109/TPDS.2021.3097283
[28] https://gmd.copernicus.org/articles/18/529/2025/
[29] https://arxiv.org/abs/2502.14148
[30] https://wpo.noaa.gov/wp-content/uploads/2022/08/FINAL-EPIC-Community-Workshop-Report-Novemeber-2019.pdf

This page is intentionally left blank

Appendices

This page is intentionally left blank

A. Data Mining

Documentation of the results of the informal interview processes described in Section 3 are gathered here. Note that the documentation identifies the individuals that were included in this process. Note that the references presented in the Appendix are gathered with the core References in the main body of this report.

A.1. UK Met Office

On December 3 2024 I had an open discussion with Andrew Saulter, Christopher Bunney, Juan Castillo and Christofer Stokes. Whereas the initial request for a discussion came from the UK Met Office regarding software modernization specifically for WW3 [31], a significant part of the discussion centered on software modernization in general. Key elements of code modernization at the UK Met Office are:

- **Next Generation Modelling Systems Programme**: this programme is designed to reformulate and redesign the Met offices weather and climate research and operational production systems, to allow them to fully exploit future super computer systems (Met Office, 2018
- **GungHo Project:** The GungHo project highlighted the need for automatic code generation to develop new dynamical cores suitable for next-generation supercomputers (Met Office, 2021).
- **LFRic and PSyclone:** The LFRic project, designed for future supercomputers, utilizes PSyclone to automate the generation of parallel code. This allows the Met Office to effectively leverage the power of advanced computing systems (Met Office, 2023).

Our discussion went from the above general approaches used by the UK Met Office to a discussion on some of the topics identified in Section 2 with a focus on Chris Bunney's code optimization work in WW3, with the following outcomes:

- The UK Met Office does not consider fault tolerance something that needs to be addressed inside the codes. This does imply that the UK Met Office intends to make an effort for clear failure messages to be generated by the codes.
- A key element of optimizing WW3 at the UK Met Office codes is the selection of the data layout, in particular the layout of the loop structures, and the use of derived data types. Using "tiling" in data structures appears to result in code that is efficient across hardware architectures.
- The UK Met Office does most of its fundamental science using Python, although many incoming scientists tend to have used MatLab for basic science. Most engineering aspects of coding are done using C++ (similar to NOAA's use of C++ in many

[31] This discussion was driven by the practical consideration that the UK Met Office had invested a significant effort in optimizing WW3 for use on GPUs, but that the main developer (Chris Bunney) was moving to other responsibilities at the UK Met Office.

infrastructure codes). The LFRic codes are still written in Fortran. With this, operational codes at the UK Met Office are multi-lingual with respect to coding languages. The UK Met Office expects their scientists and software engineers to be adaptable.

- The UK Met Office tends to hire relatively new graduates as software engineers and often loses them. However, they presently have enough experienced scientists and software engineers to maintain some continuity of knowledge (including Fortran proficiency). An identified risk is AI, which will require further diversification of coding skills - and the thinner they spread themselves the more it carries risks of single points of expertise.
- As is outlined in the first set of bullets in this section, the UK Met Office has been experimenting with automating code generation. In the discussion it became clear that the the Met Office and NOAA share concerns with respect to code robustness and reliability when using automatically generated codes in operations, although this is likely the way they will go with LFRic.

Circling back to the starting point of the discussions with the UK Met Office, they expressed their keenness of continuing discussions about WW3 code modernisation. They did note that the pressures on making the code GPU compatible have reduced a bit recently, but that they would like to keep some momentum on what has been done so far and what will be needed to make the code more futureproofed for different architectures.

A.2. Environmental Modeling Center (EMC)

On December 16 2024 I had an open discussion with Jacob Carley and Ed Hartnett from the Environmental Modeling Center (EMC) . We loosely followed the potential issues described in the previous section with the following outcomes:

- At EMC, regression testing is becoming prohibitively expensive in code management. With that EMC is moving toward unit testing as a code QC technique that has much less overhead, in particular for the shared libraries in the production suite (nceplibs). Ed expressed this as a need to decouple the code base to efficiently move towards unit testing. Note that in this context, coupling refers to software coupling rather than model coupling, see the “coupling” topic in Section 2. After our discussion, EMC formally committed to transition to a unit testing approach.
- EMC voiced the intent to go to a “devops” approach to development where the top of the core repository is deployable code. The top of the code repository would tentatively be used in all development, whereas tags are used in operations.
- For EMC costs of doing development in containers is acceptable as it saves development and Human Resource costs with portability in particular during development. This may become more important in the future when we expect some operational work to move to the cloud.
- Acknowledging that a unified workflow is still not well implemented in the UFS, EMC noted that the Unified Workflow (UW) tools look promising. Since the interview, the EPIC

team working on the UW tools has provided presentations both at EMC and at one of the UFS open meetings.

- EMC noticed that a large part of the code is still written in Fortran, but that UFS applications have long been multi-lingual, in particular with other languages being prevalent in UFS coupling tools and in MET. EMC sees no direct need or urgency in moving away from Fortran altogether.
- Note that EMC presently has no access to GPUs in operations. This is partially a catch 22, as lack of operational codes able to fully use GPU power is likely feeding into choices made in NOAA's HPC contracting.

A.3. Involvement with Industry

NOAA's engagement with industry partners generally occurs at professional meetings of organizations like the American Meteorological Society (AMS), the American Geophysical Union (AGU), the International Association of Emergency Managers (IAEM), and the National Weather Association (NWA). Such interactions fosters knowledge and information exchange, and may lead to collaboration on weather and climate research, technology, and service delivery.

NOAA's engagement with industry includes scientific and engineering support, facilitated by programs like ProTech[32], which provides NOAA with access to a wide range of professional, scientific, and technical services. This encompasses critical areas such as systems engineering, IT infrastructure management, and engineering support for complex observing systems like satellites and radar networks. Critical for software modernization, this also includes direct support for the development and enhancement of the Unified Forecast System (UFS) through support services contracts and through the Earth Prediction Innovation Center (EPIC, see below).

At the December 2024 AGU and January 2025 AMS conferences, I had discussions with representatives from Booz Allen and AWS as part of the above described process of continuous engagement. Part of the discussions relevant for the present assessment focused on the utilization of AI tools providing solutions to automate code analysis, refactoring, and testing was a key focus, with the aim of accelerating the modernization of complex software systems like the Unified Forecast System (UFS).

The assessment below provides a short sampling of AI tools relevant for software modernization. The list is intended to illustrate the large number of tools available, is not intended to be complete, and is expected to change rapidly. Note that the selection of specific AI tools for implementation may be influenced by existing contractual agreements and partnerships.

- **Key Challenges and AI Applications:**
  - **Legacy Fortran Code:** AI tools can assist in translating and modernizing this code. AI can also help in parallelizing Fortran code for HPC.
  - **Optimization for HPC:** AI can optimize code for HPC architectures. Machine learning can create surrogate parts of models or models.

[32] https://www.noaa.gov/acquisition-grants/protech

    - **Data Assimilation:** AI can improve data assimilation techniques.
    - **Code Refactoring and Optimization:** There are AI-powered code analysis tools. Generative AI can be used to automate refactoring.
- **Examples of Tools and Their Companies:**
    - **GitHub Copilot (Microsoft):** AI-powered code completion. Helps understand and modify legacy code.
    - **Codeium (Ex-Tabnine):** AI-powered code completion and chat functionalities. Generates and modifies numerical modeling code.
    - **Moderne (Moderne, Inc.):** Automates large-scale code refactoring and modernization. Migrates legacy codebases to modern architectures.
    - **Google Colab:** A free cloud-based Jupyter notebook environment, allowing for collaborative coding and data analysis.
    - **Tools utilizing transformer models**: Many companies are creating tools that use transformer models, that can analyze large code bases, and assist in language translation. Many large cloud providers, like Amazon(AWS), Google(Google Cloud), and Microsoft(Azure) are creating AI code assistance tools that utilize transformer models.

Considering NOAA's and my interest in the topic, here is a sampling of AI-Powered Unit Testing Tools and Techniques:

- **AI-Driven Test Case Generation:**
    - AI, particularly machine learning, can analyze code and automatically generate test cases that cover various code paths and edge cases.
    - This helps to ensure thorough testing and reduces the manual effort involved in creating test suites.
- **Property-Based Testing with AI Assistance:**
    - Property-based testing involves defining properties that the code should satisfy, and AI can help to generate test data that verifies these properties.
    - This can be particularly useful for testing complex numerical algorithms.
- **Mutation Testing with AI Support:**
    - Mutation testing involves introducing small changes (mutations) to the code and checking if the test suite detects these changes.
    - AI can help to automate the generation of mutations and analyze the effectiveness of the test suite.
- **Static Analysis with AI Enhancement:**
    - Static analysis tools analyze code without executing it, identifying potential errors and vulnerabilities.
    - AI can enhance static analysis by learning patterns of errors and suggesting more effective checks.
- **Test Code Generation from Natural Language:**
    - Some emerging tools are using Natural Language Processing (NLP) to generate unit tests from natural language descriptions of the code's functionality. This can

help to bridge the gap between requirements and test implementation.

Associated with these general testing topics are a small sample of available tools, subject to the same context and limitations as mentioned with the list of more general tools mentioned above.

Tools and Technologies:

- **Diffblue Cover:** This tool uses AI to automatically write unit tests for Java or Python code, aiming to increase test coverage and reduce the time spent on manual test creation.
- **Parasoft C/C++test:** automates the creation and execution of unit tests, helping to ensure that individual code components function correctly.
- **Tools that utilize Large Language Models(LLM's)**: Many companies are working on tools that will utilize LLM's to create unit tests based off of code analysis, and natural language prompts. This is an emerging field, and will likely improve rapidly.

A.4. Earth Prediction Innovation Center (EPIC)

The Earth Prediction Innovation Center (EPIC)[33] represents a unique initiative, designed to accelerate advancements in NOAA's operational weather and seasonal forecast systems, particularly in direct support of the Unified Forecast System (UFS). Established in accordance with the Weather Research and Forecasting Innovation Act (WRFIA) of 2017 (Public Law 115-25), and further defined by the National Integrated Drought Information System Reauthorization Act of 2018 (Public Law 115-423), EPIC serves as a hub for collaborative partnerships between government, academia, and industry (e.g., Uccellini et al. 2022).

EPIC's present operations were initially defined by two workshops held in 2019. An internal NOAA workshop in March 2019 laid the groundwork, defining EPIC's scope and exploring its role in enhancing model development, and culminating in a vision paper (Cikanek et al. 2019). This was followed by the broader EPIC Community Workshop in August 2019[34], which engaged the "weather enterprise" – including academia, public, and private sectors – in the planning, development, and strategy for EPIC (WPO, 209). These two meetings still form the foundation EPIC as outlined on the EPIC web page[35].

EPIC now focuses on the following seven investment areas:

- External engagement and community.
- Software engineering.
- Software infrastructure.
- User support services.
- Cloud-based high performance computing.
- Scientific innovation.
- Management and planning.

[33] https://epic.noaa.gov/about-epic
[34] https://wpo.noaa.gov/an-overview-the-earth-prediction-innovation-center-epic-community-workshop
[35] https://wpo.noaa.gov/epic

With this, EPIC supports the “infrastructure” of community modeling, including modernizing collaboration techniques through providing user support and supporting a CI-CD pipeline. Actual software modernization is limited to infrastructure software including containerization of applications and workflow approaches, but generally does not focus on the component models in coupled UFS applications.

A.5. CICE sponsors meeting and UFS presentation on DoE E3SM modernization efforts

At the December 16, 2024 CICE consortium sponsors meeting, code modernization was briefly discussed, with a focus on moving to Python and C++ for performance, as well as due to concerns with being able to get programmers capable of, and/or interested in, programming in Fortran.

This discussion led to an invitation for Elizabeth Hunke to provide a presentation on software modernization at DOE in general and for CICE in particular at a UFS Steering Committee open meeting. The CICE Consortium: New model developments and plans presentation was given on March 7, 2025. It focussed mostly on the CICE model from a physical perspective, but also dealt specifically with code modernization efforts at the Department of Energy (DOE). The latter part of the presentation focussed on more mature efforts of code optimization of the OMEGA ocean model and SCREAM atmosphere model, and more on implications for similar ongoing efforts for CICE[36], all within the E3SM model[37]. With that the following relevant observations and conclusions were taken from the presentation

- **Code Refactoring:** Refactoring existing code for improved memory access patterns and computational performance is essential for optimal code efficiency.
- **Hardware Evolution:** Adapting to hardware evolution. For older hardware, it has been best practice to focus on floating-point operations. For modern hardware, optimization needs to focus on bandwidth limitations to memory.
- **Vectorization:** Vectorizing the EVP (elastic-viscous-plastic dynamics) kernel in CICE and in other parts of the code is essential for better performance.
- **C++ Rewriting:** DOE is rewriting Icepack/Thermodynamics in CICE in C++ for faster performance, particularly on GPUs.
- **Kokkos Implementation:** DOE uses Domain Specific Languages (DSL) and Kokkos (Trott et al., 2022) for automated low-level code generation for CPUs or GPUs in the OMEGA ocean model. This experience is expected to inform a similar modernization for CICE.
- **Technical coding issues**: The presentation identifies successful technical approaches such as (i) enabling the entire EVP calculation to be put onto a single node or GPU, (ii) Saving minimal information needed to reproduce / recalculate all quantities, (iii) using point-based domain decomposition, and (iv) keeping communications local.

[36] Note that presently CICE is part of the UFS whereas OMEGA and SCREAM are no.
[37] https://eesm.science.energy.gov/

- **Achieved performance improvement**: computations were accelerated by a factor 2-5 for CPUs and 30-35 for GPUs (see presentation for details).

A.6. Geophysical Fluid Dynamics Laboratory (GFDL) UFS presentation

Back to back with the UFS Steering Committee presentation of DOE software modernization efforts described in the previous section, Rusty Benson of the Geophysical Fluid Dynamics Laboratory (GFDL) gave the presentation Pace and PyFV3: Performance-portable computing using GT4py on March 7, 2025.

The presentation emphasizes the need for models to run efficiently on both CPU and GPU-based supercomputers, especially with the rise of GPU-dominant exascale machines. GFDL aims to achieve this by developing "Pace," a Python port of the SHIELD weather model designed for performance-portable computing using GT4Py. The following bullets summarize the presentation

- **PyFV3, PySHIELD and Pace:** PyFV3 (Dahm et al., 2022) is the Python port of the FV3[38] dynamical core. PySHIELD (Dahm et a., 2023) is the GT4Py port of SHIELD's[39] physics parameterizations. Pace (Dahm et al., 2023) is built from PyFV3 and PySHIELD.
- **Technology used**:
    - **Python:** Python is highlighted as an attractive language for model development due to its accessibility, extensive libraries, and integration with AI/ML tools.
    - **GT4Py**: a Domain Specific Language (DSL) for weather modeling (e.g., Paredes et al., 2023).
    - **DaCe[40]**: a separate compilation framework for data flow optimization, with key technologies enabling performance portability (Ben-Nun et al. 2019).
    - **NDSL:** The NOAA/NASA Domain Specific Language Middleware simplifies model development by providing abstractions for common computational patterns and easing the use of GT4Py and DaCe.
    - **FMS[41]**: Flexible Modeling System coupling tools used in the GFDL modeling ecosystem.
    - **PyFMS:** Python interface to the FMS, allowing Pace to integrate into the GFDL ecosystem.
- **Scientific Targets:** Initial applications of Pace include large eddy simulations for cloud studies and high-resolution global studies of tropical cyclones.
- **Future Plans:** The goal is to achieve science-readiness of Pace, enhance its components, and bring it fully into the GFDL modeling system. Collaborations with NASA, ETH Zurich, and NOAA's Global Systems Laboratory are ongoing.

[38] https://www.gfdl.noaa.gov/fv3
[39] https://www.gfdl.noaa.gov/shield
[40] https://github.com/spcl/dace
[41] https://www.gfdl.noaa.gov/modeling-systems-group-fms

Note that FMS is used inside component models of the UFS, but not as a general coupling tool in the UFS. It does represent a technology that allows for closer coupling than the present coupling in the UFS at the API level.

A.7. European Centre for Medium Range Weather Forecasting (ECMWF)

NOAA and The European Centre for Medium-Range Weather Forecasts (ECMWF) have had several informal discussions on software modernization for roughly a decade. For instance, the topic was discussed in the hallways at the 2016 Annual Seminar of ECMWF, during a personal visit to ECMWF in June 2022, and during a ECWMF workshop on wound waves and coupling in 2024. The notes of these meetings and an internet search using Gemini resulted in the following bullets.

- **Focus on Scalability and Performance:** ECMWF deals with massive datasets and computationally intensive models, requiring continuous optimization for high-performance computing (HPC) environments. With ECMWF historically focusing on a single (coupled) model on a single target hardware architecture, they have been able to focus on this arguably better than any other weather and climate prediction center.
- **GT4Py:** ECMWF is involved in projects using GT4Py (e.g., Paredes et al., 2023), which allows for the encoding of stencil operators in a hardware-agnostic way. This enables more concise and maintainable scientific applications. GT4Py then translates these abstract definitions into high-performance, low-level code. This is a very important part of their code modernization strategy.
- **Languages**: GT4Py is being used in projects that are re-writing parts of the Integrated Forecasting System (IFS) into python (e.g., Ubbiali et al., 2025).
- **Earthkit:** ECMWF has developed Earthkit (ECMWF, 2024), a Python framework designed to simplify data access, processing, and visualization. Earthkit promotes code reusability and componentization, which are key aspects of modern software practices.This system is designed to allow for greater interoperability between different parts of their software systems.
- **General Modernization Trends:** ECMWF, like many other forecast centers, is moving towards more modular and flexible code structures. This includes adopting modern software development practices and leveraging advancements in supercomputing. They are also starting to move away from a focus on in-house model development by increasing the use of external software, and of open development.

The topics gatherered in Section 2, and the above bullets were shared with Any Brown from ECMWF who confirmed on May 12 2025 the contents of the above bullets and provided some additional references. He also shared that ECMWF has recently had more discussions about moving away from Fortran, but that these discussions have not been shared with the general public yet.

A.8. Software Engineering for Novel Architectures (SENA)

On March 26, 2025 I had a discussion with Frank Indiviglio about the NOAA Software Engineering for Novel Architectures (SENA) program. This program has existed for several decades, and as a fun fact, fully funded the original development of WAVEWATCH III from 1993 through ca. 1998. Due to its limited funding, SENA has traditionally focused on specific projects regarding high-payoff optimization on the "hardware of the day" used at NOAA. More recently, SENA has been focussing on more systematic approaches dealing with exascale computing and the transition of CPU to GPU hardware. Examples of the latter are the exascale data and computing studies of Govett et al. (2024), and efforts to port the the GFDL models to new computer architectures by converting the code to Python (e.g., Dahm et al, 2022, 2023). With this, SENA efforts inform the present study though targeted software innovation projects, and should be considered without NOAA as a resource for UFS software modernization.

Frank Indiviglio did point out that the need to move away from Fortran is not just driven by the availability of programmers fluent in Fortran. He noted that the number of viable Fortran compilers becomes smaller and smaller, and that the present Fortran compilers may no longer result in the most efficient codes on HPC. If the availability of Fortran compilers erodes even more, this may become a serious Continuity of Operations (COOP) issue for operational environmental modeling.

A.9. NCEP Central Operations (NCO)

I had a discussion on 5/9/2025 with David Michaud, the director of NCEP Central Operations (NCO). NCO is responsible for the operational Production Suite of the NWS and broader NOAA, and as such represents the operational target of many UFS applications. The discussion followed the topics outlined in Section 2 with the following outcomes (including a review and additional information provided by Steven Earle).

- **Open source and open science**: For many years, NCO has been leaning more into the use of open-source software. A good example of this is the support of NCO for developers to use Python and some of its tool sets, and incorporating the language and its tools in operations. Adopting the UFS approach to community-developed operational applications is a natural extension of this move to more open-source software.
    - This has led to the need for maintaining more and more tools and libraries. This requires additional efforts by NCO, but this is still manageable.
    - Note that for operationalizing community-developed applications NCO requires a commitment of support for the software, which for community models is usually provided by EMC or other NOAA Line Offices.
    - Adoption of community software in operations is complicated by the fact that the operational computers are considered to be high-security assets under the Federal Information Security Management Act (FISMA). This implies that there may be a delay for upgrades to community software to become available for operations, and this implies that developers should work with NCO early and

    - often to assure that packages that developers want to use can also be used in operations.
    - All these separate issues strongly favor a continuous discussion between developers and NCO, starting at early development, rather than at the "handoff to operations".
- **Fault tolerance**: With computer architectures apparently moving away from massively parallel systems with relatively slow CPUs toward hybrid systems with much more powerful processors like GPUs, and with fault tolerance being addressed more systematically at the operating system level, NCO does not see fault tolerance of its biggest model as a significant rising issue.
- **Load balancing**, **Optimization**, **Coupling**: With the operational HPC available to NCO typically used as close to capacity as is consistent with a 99.9% on time product delivery, code efficiency is of paramount importance. NCO general has some human resources for code optimization available as part of their HPC contracts. Note that by the nature of the contracts, the associated code optimization is project driven focusing on the available HPC hardware and the actual software applications in operations.
- **Code quality**: Operational implementation standards can be found at their web site[42].
- **Knowledge and expertise**: Support of the production suite operates in a three tiered structure. Tier 1 consists of 24/7 operators with responsibility for managing failure; They have basic helpdesk knowledge and can execute well documented procedures. If it's outside that scope then they contact Tier 2 support; this consists of the operations team that has a more in depth understanding of the systems, applications and their dependencies. If Tier 2 support is unable to solve the problem then Tier 3 support is brought in. This support consists of the developers / code managers of the application with the most robust knowledge of the application.
- **Moving away from Fortran**: As has already been noted in Section 2, the UFS and the NCO production suite are already inherently multi-lingual with respect to programming languages used. NCO presently does not have a major issue with finding or training programmers working with Fortran, but the potential risk of losing access to reliable Fortran compilers as identified above could become a serious COOP issue for NCO. In this context there is value in moving away from Fortran, and with the existing multi-lingual environment, this can be done incrementally until and unless we identify a hard COOP deadline.

A.10. National Water Center (NWC)

I had a discussion with Fred Ogden from the National Water Center (NWC) on 5/20/2025. The discussion focused mainly on the modernization efforts for the National Water Model (NWM) and the selected coupling approach in the new model. There was general agreement on a needed focus on community modeling, and the reduced focus on fault tolerance. The NWC is

[42] https://nws-hpc-standards.readthedocs.io/en/latest

not yet considering a systematic move away from Fortran The following paragraphs were provided by Fred Ogden.

Attempts in 2019 to improve the modularity of the WRF-Hydro code used in the National Water Model (NWM) were impeded by the monolithic nature of the code, and its origins as a research code. An audit of the code by the GSA 18F group suggested a complete refactoring using modern coding practices with an emphasis on coupling interfaces and the use of standards. OWP embarked on creation of what is called the Next Generation Water Resources Modeling Framework (NextGen Framework) (Ogden et al. 2021). A series of interagency discussions involving NOAA/USACE/ USGS/USBR/DOE identified use cases. The Basic Model Interface (BMI) model coupling standard version 2.0 (Hutton et al, 2020) was identified as suitable for constructing the National Water Model running different models and process modules in different parts of the country. The Open Geospatial Consortium, WaterML 2.0 Part 3 Hydrologic Features (HY-Features) conceptual data model was selected as suitable to describe the surface water hydrologic features of the landscape.

Hydrologic phenomena are often discontinuous in space and time and governed by highly uncertain processes and exhibit threshold behaviors. Because of this, models to predict the hydrologic response of a region to atmospheric forcing tend to rely on conceptualizations. The BMI model coupling standard provides a “thin middleware” standard for model coupling. The BMI standard consists of approximately 45 function definitions that allow an external driver program to initialize, step through time and finalize operation of a model code. It also provides a means for the driver program to access and exchange model states and parameters for models that use a variety of discretizations. For a model to comply with the BMI standard it must allow external control of its march through time. Models may retain internal timestepping for numerical stability and optimal performance, but must provide control to a driver program through BMI interface at specified intervals.

The BMI interface definition is flexible enough to support a variety of programming languages. In accordance with interagency discussions, the NextGen Framework supports codes written in C, C++, Fortran and Python. There are some challenges regarding the BMI interface with languages that do not support pointers directly, but these can be mitigated using a number of approaches.

A.11. Additional observations

The interviews and meetings reported on in this Appendix did not occur in a vacuum, but are the extension of a long ongoing effort, as its subjects have been addressed in many other meetings. The last sub-section of the Appendix documents some selected relevant outcomes from earlier and separate discussions.

- **NCAR / CESM:** The Community Earth System Model (CESM) of the National Center for Atmospheric Research (NCAR) is closely related to the UFS through its shared infrastructure that was co-developed using a formal Memorandum of Agreement as mentioned in the body of the report. With that, the UFS tends to focus on shorter

"weather" forecast time scales, whereas the CESM focuses on longer "climate" forecast time scales. With that, it could be expected that both the CESM and UFS community have similar needs and interest with respect to software modernization.

A casual assessment of CESM work presented in professional meetings suggests that the main focus of the CESM group is not on structural software optimization and modernization, but on developing new modeling techniques for component models of the CESM. An example of such work is the development of the PiCLES wind wave model that uses a "particle" description of wave fields rather than the spectral description used in WW3 (Hell et al. 2023) to reduce the dimensionality of the problem and hence create much cheaper models for climate applications. It appears to be a fair observation that CESM uses lower resolution grids for forecasts with much longer forecast horizons, and therefore is less negatively impacted by the present API level coupling techniques shared by the UFS and the CESM..

- **NCAR / MPAS:** The Mesoscale and Microscale Meteorology (MMM) Laboratory of NCAR has developed the Model for Prediction Across Scales (MPAS). The dynamical core of this model is presently being integrated in the UFS infrastructure for use for convection allowing weather forecasting. Even at the initial stages of this development, it is clear that the present coupling techniques in the UFS are inefficient for the target spatial scales and associated grid size for MAPS applications, contrary to the above described experiences with the CESM. This implies that MPAS applications in the UFS are likely to benefit significantly from more integrated coupling techniques. Initial assessments are that the MPS coupling requires "local" approaches to coupling rather than the inherent "global" (i.e. considering the entire model grid) approach used in ESMF.

- **Computer Languages in Industry**: Complimentary to discussions with representatives from industry addressed in Appendix A.3, the UFS has direct activities with commercial modeling groups. In a recent discussion with Pieter Smit from SoFar Ocean, he pointed out that the beggar players in the AI and cloud sectors of HPC are moving to Rust as a programming language. Due to the inherent support from large industry partners, and the open-source nature of Rust, this might be another language to consider for the UFS.

  Rust is a modern programming language known for its strong emphasis on safety, speed, and concurrency. It offers memory safety without sacrificing performance, as Rust achieves speeds comparable to C and C++. Additionally, Rust's ownership system and thread safety features make it excellent for building reliable and efficient concurrent https://docs.google.com/document/d/1NIbX6Qlcf3FpkM2mrmp3zdkXNOlS6IDWw-glM2Z0e-E/edit?tab=t.0applications, crucial for tasks like system programming, web development, and high-performance computing.

  Rust and C++ share similarities in their performance, suitability for systems programming, and focus on low-level control. However, they differ significantly in their approaches to memory safety and error handling. Rust's emphasis on safety and its

modern features make it an attractive choice for new projects, while C++ remains a powerful option for existing projects and performance-critical applications.

B. Revision history

The Table below identifies revisions of this document.

Version	Date	Description
1.00	7/14/2025	First version completed