Code Management Framework for the UFS WM and Applications

Architectural elements of the UFS Weather Model (WM) and applications are extensively involved with various modeling components and external software libraries: atmosphere, ocean, sea ice, wave, aerosol, and coupling infrastructure, etc. EPIC is mandated to create a common code base and modeling infrastructure of the UFS system for both research and operational forecasts. Confluence codebase statistics link shows the current status of the code bases of the UFS Weather Model and Short-Range Weather (SRW) Application (App). Source code decomposition and analysis shows the number of Git repositories and file types to build the UFS WM and SRW App.

In the UFS Agile code management (CM) processes, software integrity and quality of incremental code changes are continuously tested to maintain the required baseline test cases. The UFS WM and SRW Git repositories evolve with the contribution from the community developers adopting the CM practices and Continuous Integration (CI) and Continuous Deployment (CD) tools in the community-based development and operations (DevOps) processes. Various CI tools, such as Jenkins, Git workflow CI, pyGithub, and Docker containers, are currently utilized in daily UFS CM practices and DevOps stages such as continuous development, continuous integration, continuous testing, continuous deployment, and continuous monitoring. The main goal of the UFS WM and SRW CM framework is to ensure a common code base infrastructure to facilitate and deliver new source code features, fixes, and updates in close alignment with the UFS operational and research community stakeholder requirements. A diagram of general code development and commit procedure is attached in Appendix 1. The EPIC CM framework will focus on ensuring the key priority action items summarized below,

- Coordinate code integration and porting to supported platforms of the UFS WM and Apps consistent with subcomponents.
- Ensure the code integration schedules to follow the development priorities of the UFS WM, Apps, and subcomponents.
- Communicate status to subcomponent and application code management changes in the UFS-related meetings.
- Review and track the UFS WM pull requests (PRs) daily, identify issues and conflicts, work with developers to ensure PRs are addressed.
- Evaluate and coordinate new software/libraries/input-data in the UFS WM, discuss with developers and subject-matter experts.
- Coordinate and support the UFS WM testing for software releases including system, dependent libraries and subcomponents.
- Coordinate subcomponent repository updates and ensure subcomponents are regularly synched with authoritative repositories: every two weeks.
- Ensure the UFS WM testing framework reflects latest development and code changes across the UFS applications: every two weeks.
- Develop plans to improve the UFS WM testing framework to run tests efficiently and ensure code quality prior to every UFS WM and application release.
- Coordinate the UFS WM and subcomponent code changes with workflow and downstream applications.
- Provide and archive meeting minutes and update within 48h following every code management meeting.
- Support developers on synching and merging Git repository branches, run the required UFS WM tests (also known as RT and ORT, regression and operational requirement tests) on the supported Tier-1 platforms, monitor and resolve testing, debugging performance issues.
- Maintain the UFS WM RT, CI, and ORT tests, input data and baselines on NOAA high performance computing resources.

In the following sections, we describe the UFS CM practices, guidelines, and community coordination schedules to capture the listed priorities of the EPIC CM framework in detail.

UFS CM Practices and Guidelines

UFS authoritative (or official) Git repositories are read-only for users. Branches are protected and code changes can only be made by merging a PR, not by pushing changes directly to the repository. All the UFS application development branches (usually named “develop”) point to subcomponents’ official development branches at different revisions. The subcomponent branch should not be a temporary branch in a user’s or developer group’s personal fork. Personally installed software including libraries, directories, or code are not allowed in the UFS application code. Simple and clear commit messages are required to issue a PR.

Forking:

- For new development, users will create a fork from the authoritative repository, and PRs will be made from a branch in the developers’ forks.
- When a fork is created, all the branches in the parent repository are copied automatically. It is suggested that users keep only the develop branch and other branches that they are interested in. They can ignore the remaining branches or delete them entirely.
- It is suggested that users create their own feature branch from the development branch (e.g., develop or master branch) in the authoritative (or official) repository. Users can add an upstream remote to the authoritative repository, check out upstream develop/master branch, and then create a new feature branch from the upstream develop/master branch.

Branching:

- Users should create a feature branch in their fork for development work. The develop and feature branches in a user’s fork should be synched with authoritative repositories periodically.
- When development work is done, users must sync their feature branch with the latest develop/master branch in the authoritative repository, run RTs, and make a PR to the authoritative (or official) repository.
- When the code changes are merged into an authoritative repository, it is recommended that users delete the feature branch in their personal fork.
- When new development work is needed, users will create a new feature branch, continue to sync their development with an authoritative repository, and follow PR procedures to merge the changes into the authoritative repository.

Tagging/Versioning:

- Only code managers can create production branches or public release branches in an authoritative repository.
- For a production branch, the suggested branch naming convention is production/app.vxx, where xx refers to the operational implementation version number (e.g. GFS.v16). Annotated tags will be created with names app-vxx.yy.zz (where yy is a feature upgrade and zz is bugfix upgrade).
- For a public release branch, the suggested branch naming convention is release/public-vxx.yy.zz. Annotated tags will be created with names: app-vxx.yy.zz.
- For the develop branch, benchmark tags can be created as benchmark/app-vxx.yy.zz
- Code managers are encouraged to delete the old production and public release branches. Annotated tags will remain when the production/public release branches are deleted.

- Tags will only be created for special features or milestones, not for every commit on the develop branch.

Pull Requests (PRs):

- An issue needs to be created and associated with the PR. The issue should provide detailed information on all the features in detail.
- Before making a PR, the user's feature branch should be synched with the corresponding branch in the authoritative repository. All the conflicts must be resolved. The code must also pass RTs on at least one supported platform.
- When users submit a PR for a code commit, the PR creators will add reviewers or—if they lack privileges to add reviewers—indicate in the PR message which reviewers should be added. Then, code managers can add reviewers.
- At least one reviewer with write permission needs to approve the PR before the PR can be merged. If needed, developers can ask code manager (s) to run the RTs on the required platforms that users don't have access to.
- When the PR is reviewed and no further changes are required, the PR can be put in the Commit Queue.
- For PRs that involve several repositories: All PRs need to be cross-referenced in the description of each repository PR, and a note needs to be placed in each PR describing dependencies. During the code commit, all PRs need to be ready to commit. This requires that all the related commit branches are synched with the top of their respective repositories, and no other commits will be added to each repository until the current commit is finished (repository freeze for the commit). If any of the repositories are updated, the code manager of the updated repository needs to notify all the related repository code managers to redo the RT. Before the PR can be merged in each individual repository, the top application RTs need to be finished, and RT information needs to be posted in all the related repositories.
- PRs should be small. It is recommended that developers break large features into several smaller PRs, if possible.
- Developers should add a title that clearly describes what the PR does.
- The description should include detailed information about what was changed, why it was changed, and how it was changed.

Code Commit:

- The code changes will be reviewed and approved by at least two code managers.
- The CI tests and GitHub Actions tests must pass.
- RTs must pass on all supported platforms, and all RT log files need to be updated. The exception is when a certain platform is under maintenance. At such times, the RT will be skipped on that platform. When the platform becomes available, the following PR will be tested and verified. If issues come up with the platform, PRs will be suspended until the issue is resolved.
- Code managers will discuss the current PRs in UFS and subcomponent repositories and a commit queue will be determined and put on the UFS wiki page.

Commit Procedure:

- Assuming that proper testing has been completed, code managers must also check the following list for any PR created in an application or sub-component repository:
    - The developer followed PR template instructions, and provided required information.
    - The code is merged to the top of develop/master branch.
    - Add labels such as "documentation," "bug," "enhancement," "baseline change," "no baseline change," "New Input Data Req'd," or "Input data change"

Requirements for Adding a PR to the Commit Queue:

- When a PR requester (or code manager) sets the "ready for review" label, reviewers are assigned.
- Reviewers review and approve code changes.

Steps to Merge the PR Listed at the Top of the Commit Queue:

- Developers merge file changes to match the development branch and coordinate with code managers to trigger CI Git labels (e.g., run-ci and/or jenkins-ci).
- Monitor the results of CI runs and ask reviewers' final comments to start approval procedures.
- Start RTs on Tier-1 platforms.
- If all RTs pass, the PR can be merged with final approvals from two code managers.
- If RT cases fail, some simple fixes can be added. Reviewers must approve the fixes. The CI and RT run steps must be repeated.
- If more time is required to fix issues found at commit time, the PR will be removed from the Commit Queue. It will be added to the top of the Commit Queue when the issue is fixed.

CM Daily PR Merging Steps:

- Assign reviewers, check review status, and check test status.
- If input data needs to be added, copy the data to the RT input data directory.
- Check whether the PR and subcomponent PRs are approved; reassign reviewers if needed.
- Make sure to run CI tests after the code review is done.
- If a new baseline is required, decide on a baseline directory name and communicate with PR owners and CM groups when new baselines are created on certain HPC platforms.
- Validate RT results. Coordinate with PR owners and reviewers to confirm baseline creation and RT runs on supported platforms.
- Merge the PR.

General Guidance on Code Changes:

- All new features should be implemented as options so that there is no impact to current UFS applications. It is suggested to add a regression test to demonstrate how to use the new feature.
- Bug fixes will not be implemented as options. They may change results. Developers need to make sure that all the UFS applications will work with bug fixes.

UFS WM RT Application Cases and Options on Tier-1 Platforms

The software integrity of code changes is maintained with the RTs running the bit-by-bit comparison against scientifically validated baseline cases on Tier-1 platforms. Code managers support baseline requirements for the selected UFS applications targeted for operational implementations, including the global weather forecast, subseasonal to seasonal (S2S) forecasts, hurricane forecast, regional rapid refresh forecast, and ocean analysis. At this time, there are 150 regression tests running on the bi-weekly commit schedules. The application list, physics options, and build configurations are shown in the following table.

UFS WM Applications	CCPP Suites	Build options
S2SWA	FV3_GFS_v17_coupled_p8, FV3_GFS_cpld_rasmgshocnsstnoahmp_ugwp	-DMPI=ON -DMOM6SOLO=ON
S2SW	FV3_GFS_v17_coupled_p8	-DMPI=ON -DMOM6SOLO=ON
S2SWA	FV3_GFS_v17_coupled_p8, FV3_GFS_cpld_rasmgshocnsstnoahmp_ugwp	-DDEBUG=ON -DMPI=ON -DMOM6SOLO=ON
S2SW	FV3_GFS_v17_coupled_p8	-DDEBUG=ON -DMPI=ON -DMOM6SOLO=ON
S2S	FV3_GFS_v17_coupled_p8_sfcocn	-DCMEPS_AOFLUX=ON -DMPI=ON -DMOM6SOLO=ON
S2S	FV3_GFS_v17_coupled_p8	-DMPI=ON -DMOM6SOLO=ON
ATM	FV3_GFS_v16, FV3_GFS_v15_thompson_mynn, FV3_GFS_v17_p8, FV3_GFS_v17_p8_rrtmgp, FV3_GFS_v15_thompson_mynn_lam3km	-D32BIT=ON -DMPI=ON
ATM	FV3_RAP, FV3_RAP_RRTMGP, FV3_RAP_sfcdiff, FV3_HRRR, FV3_HRRR_smoke, FV3_RRFS_v1beta, FV3_RRFS_v1nssl	-D32BIT=ON -DMPI=ON
ATM	FV3_GFS_v16_csawmg, FV3_GFS_v16_ugwpv1, FV3_GFS_v16_ras, FV3_GFS_v16_noahmp	-DMPI=ON
ATM	FV3_GFS_v16_fv3wam	-D32BIT=ON -DMULTI_GASES=ON -DMPI=ON
ATM	FV3_GFS_v16_fv3wam	-D32BIT=ON -DMULTI_GASES=ON -DDEBUG=ON -DMPI=ON
ATM	FV3_RAP,FV3_HRRR	-D32BIT=ON -DCCPP_32BIT=ON -DMPI=ON
ATM	FV3_RAP,FV3_HRRR	-DCCPP_32BIT=ON -DMPI=ON
ATM	FV3_RAP,FV3_HRRR	-D32BIT=ON -DCCPP_32BIT=ON -DDEBUG=ON -DMPI=ON
ATM	FV3_RAP,FV3_HRRR	-DCCPP_32BIT=ON -DDEBUG=ON -DMPI=ON
ATMW	FV3_GFS_v16	-D32BIT=ON -DMPI=ON
ATMAERO	FV3_GFS_v17_p8	-D32BIT=ON -DMPI=ON
ATMAQ	FV3_GFS_v15p2	-DMPI=ON
HAFSW-MOVING_NEST	FV3_HAFS_v0_gfdlmp_tedmf, FV3_HAFS_v0_gfdlmp_tedmf_nonsst, FV3_HAFS_v0_thompson_tedmf_gfdlsf	-D32BIT=ON -DMPI=ON
HAFS-ALL	FV3_HAFS_v0_gfdlmp_tedmf, FV3_HAFS_v0_gfdlmp_tedmf_nonsst	-D32BIT=ON -DMPI=ON
NG-GODAS		-DMPI=ON -DMOM6SOLO=ON
NG-GODAS		-DDEBUG=ON -DMPI=ON -DMOM6SOLO=ON

UFS Community CM Meetings

EPIC supports several UFS community code management meetings:

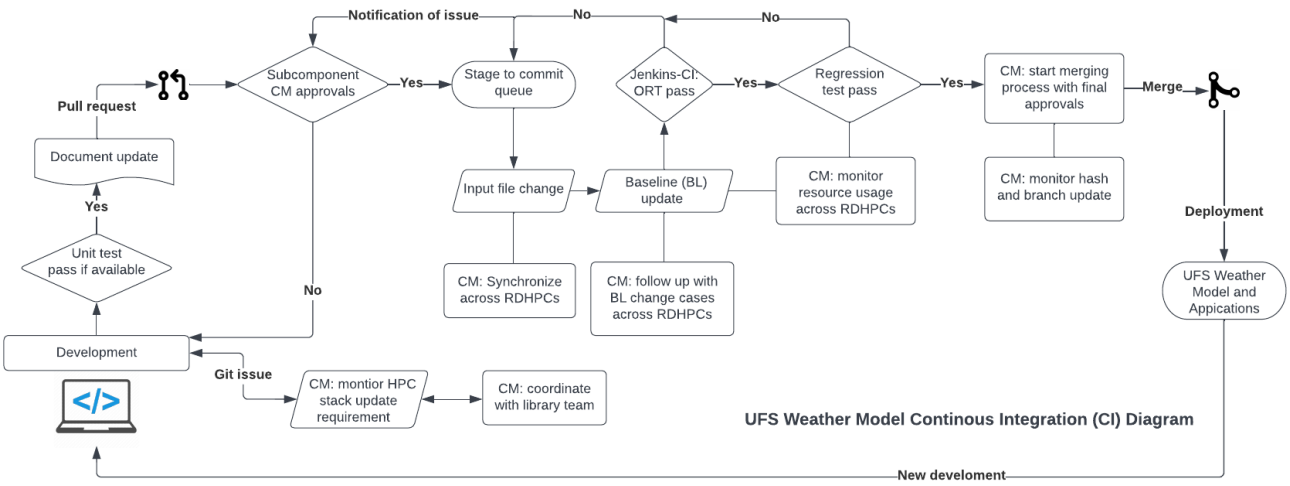
- UFS Applications and Components Coordination Meeting: Attendees at this bi-weekly Friday meeting discuss the application of CM strategies across UFS applications and components, coordinate code changes that may impact UFS applications, communicate and schedule commits to satisfy priorities and timelines, and coordinate code deliverables for UFS public release(s).
    - Meeting Link: https://meet.google.com/hwh-dnqr-vwd,
    - Commit Queue Link: https://github.com/ufs-community/ufs-weather-model/wiki/Commit-Queue
- UFS SRW App CM Meeting: Attendees discuss upcoming changes and concerns related to the SRW App, as well as recent PRs.
    - Meeting Link: https://meet.google.com/hmb-oqqf-sai
- UFS EMC/EPIC CM Daily Tag-up: This meeting is a daily discussion to identify the technical issues and schedule changes in the UFS WM Commit Queue.
    - Meeting Link: https://meet.google.com/xpf-auey-tcr
- EPIC Internal Weekly CM Tag-up: Attendees discuss on-going UFS-WM and SRW PRs.
    - Meeting Notes: https://confluence-epic.woc.noaa.gov/display/EPIC/EPIC+UFS+Code+Management

Progress and Accomplishments: FY22

- UFS WM CM Transition from EMC to EPIC: Led daily and bi-weekly UFS application and component coordination meetings
- Merged ~108 PRs in approximately four months (April 2022 - August 2022)
- Maintained UFS WM baselines on NOAA Tier-1 Platforms (Hera, Orion, Gaea, Jet, Cheyenne)
- Software stack EPIC transition and version updates: Intel-2021/2022 upgrade, hpc-stack updates including FMS-2022-01 and FMS-2022-03, ESMF-8.3.0b09, MAPL-2.22, etc.
- UFS WM Component Updates: GOCART, MOM6, CICE6, CCPPv6, UPP in-line post, support for UFS-WM prototype 8 (P8) tag
- CM support for UFS WM release/public-v3.0
- CM support for UFS SRW release/public-v2

Appendix 1

Figure A1-1. UFS WM continuous integration (CI) diagram


Notification of issue
No
No
Pull request
Subcomponent CM approvals
Yes
Stage to commit queue
Jenkins-CI: ORT pass
Yes
Regression test pass
Yes
CM: start merging process with final approvals
Merge
Document update
Yes
Input file change
Baseline (BL) update
CM: monitor resource usage across RDHPCs
CM: monitor hash and branch update
Deployment
Unit test pass if available
No
CM: Synchronize across RDHPCs
CM: follow up with BL change cases across RDHPCs
UFS Weather Model and Appications
Development
Git issue
CM: montior HPC stack update requirement
CM: coordinate with library team
UFS Weather Model Continous Integration (CI) Diagram
New develoment