# Component-Based Development of CFD Software FUN3D

Xinyu Zhang
*Analytical Mechanics Associates, Inc*
*Hampton, VA 23666*
William T. Jones, Stephen L. Wood, and Michael A. Park
*NASA Langley Research Center*
*Hampton, VA 23681, USA*

# Summary

- FUN3D is taking the component-based development approach to meet the challenge of the increasing software complexity.
- This approach improves code reusability and supports fast development.
- In this paper, the workflow and CI is presented. The integration of the components is mainly automated.
- Currently, the FUN3D component still serves as the driver in the system. It is being refactored and componentized.
- This is an on-going work, and the approach will be continuously improved.

# Introduction

- FUN3D
  - Is a suite of Computational Fluid Dynamics simulation and design tools
  - Has undergone continuous development at the NASA Langley Research Center since 1980s
  - Has supported many leading edge research projects
- New requirements
  - The complexity of the code has increased significantly
  - The interaction of simulations from different disciplines is needed
  - Numerous solvers, mesh partitioners, and communication schemes are desired
- Foundation of this approach
  - Study of software design principles

    O'Connell, M. D., et al., "Application of the Dependency Inversion Principle to Multidisciplinary Software Development," AIAA Paper 2018-3856, 2018.
  - Study of software interface design

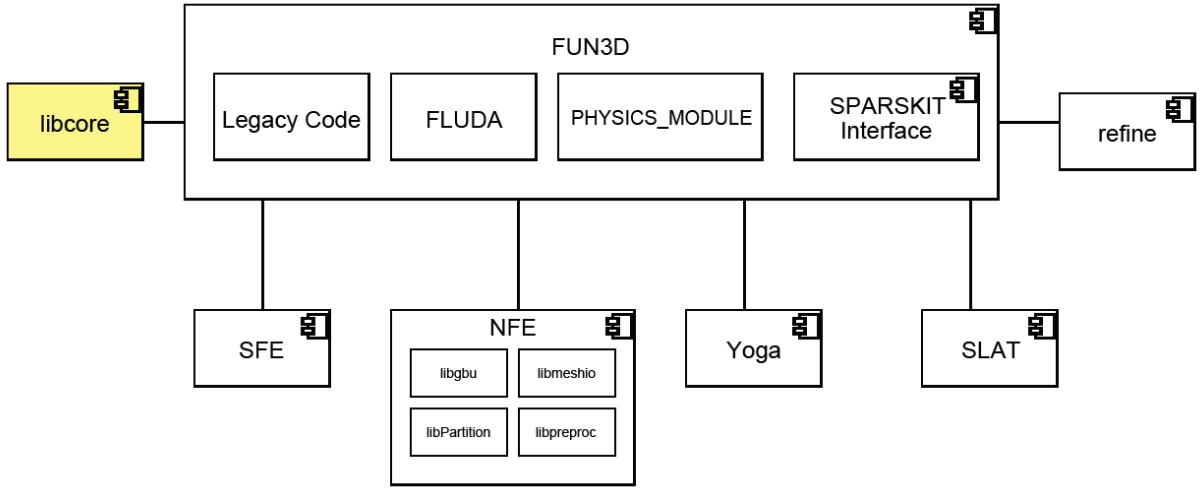
    Jones, W. T., Wood, S. L., Jacobson, K. E., and Anderson, W. K., "Interoperable Application Programming Interfaces for Computer Aided Engineering Applications," AIAA Paper 2021-1364, 2021.
- Transition to component-based development

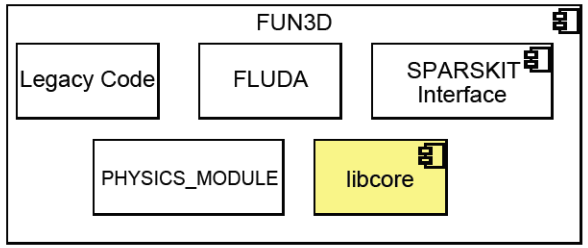
# Component-Based Development

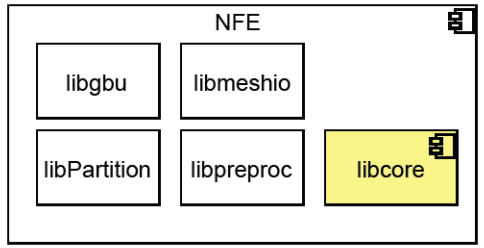
- Approach
  - Building systems with reusable software units -- components
  - Main objectives: increase productivity, save costs, and improve quality
- Component

  A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

  Szyperski, C., Gruntz, D., and Murer., S, *Component Software: Beyond Object-Oriented Programming*, edited by C. Szyperski, Component Software Series, Addison-Wesley, 2002

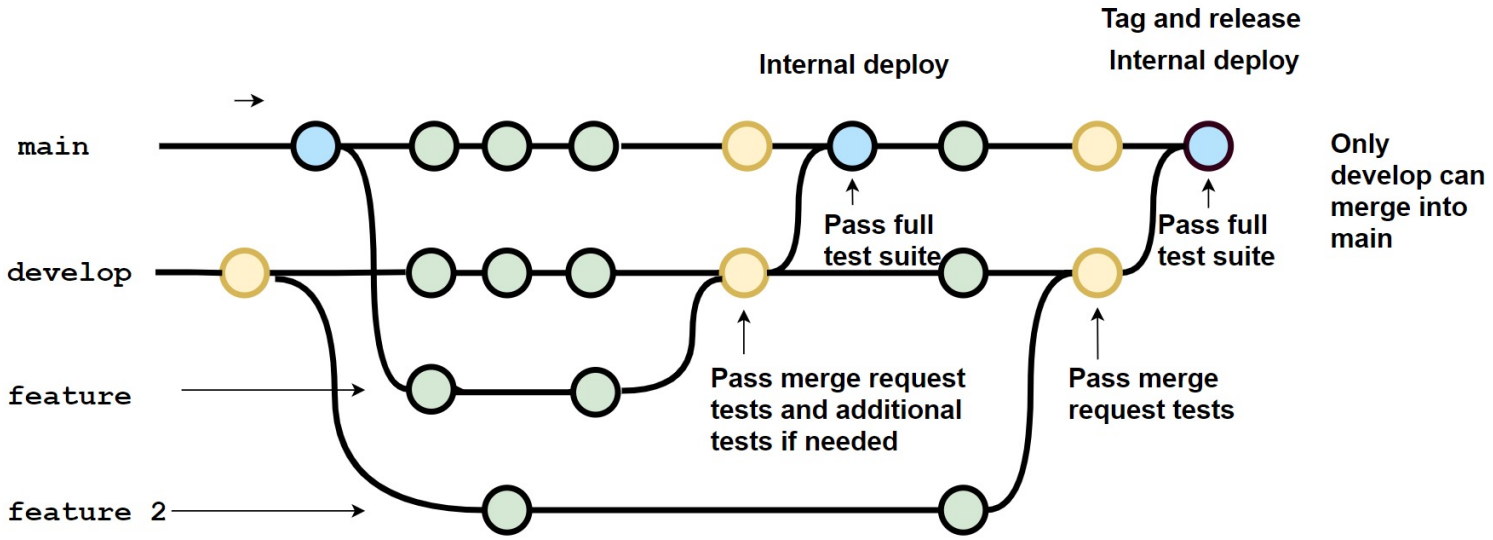


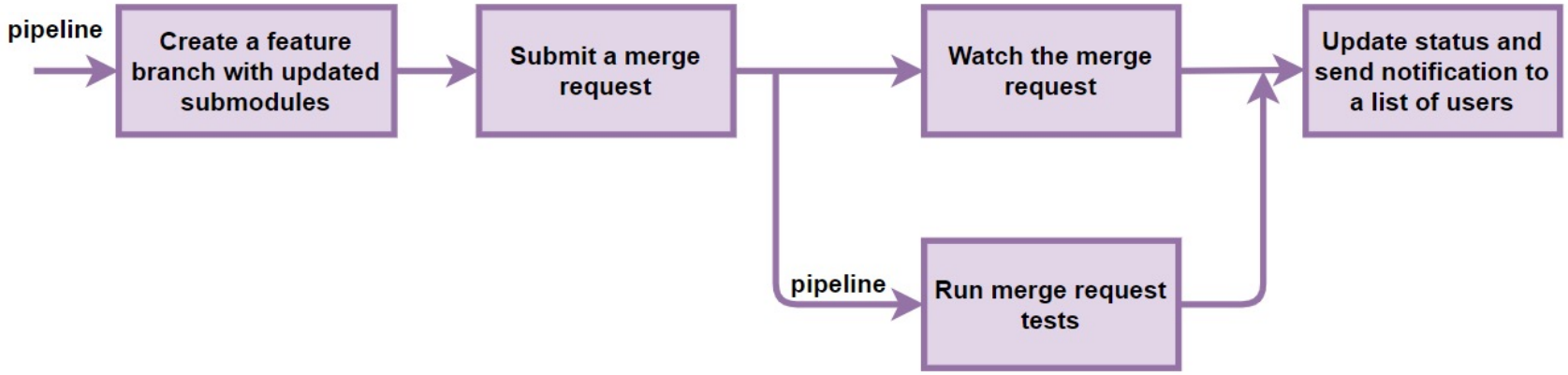
# Version Control

- Git is used for version control.
  - Each component or system is a Git repository.
- Components are assembled into a system through submodules.
  - Submodule points to a specific SHA (commit identifier).
  - SHAs are advanced once the tests have passed.
  - Only SHAs from a protected branch (usually the main branch) are allowed to be merged.
- FUN3D component Git branches



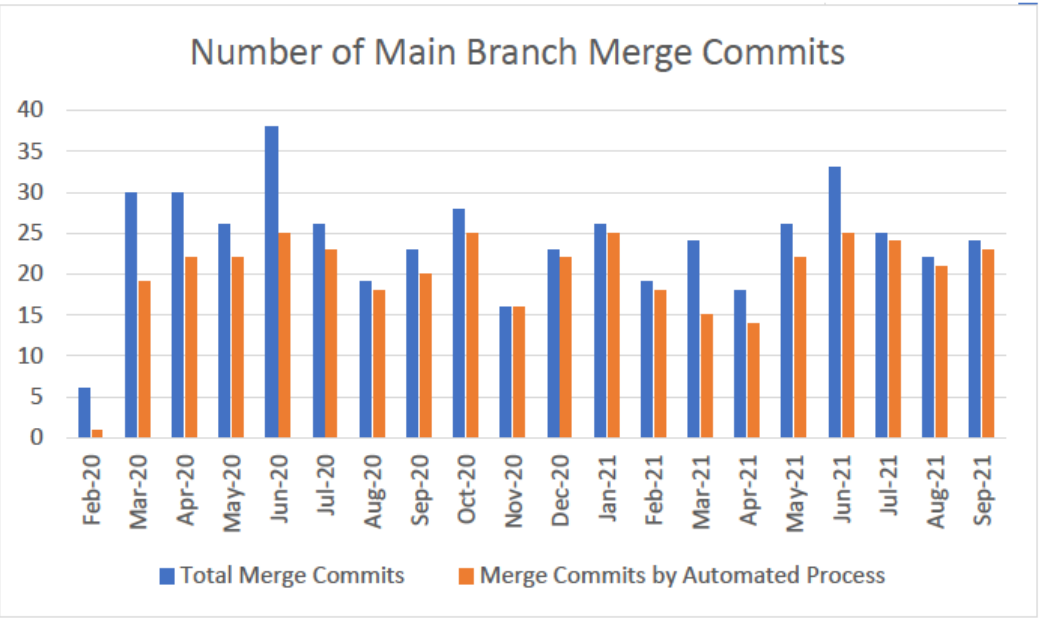
# Continuous Integration

- Continuous integration with Gitlab-CI
- FUN3D component CI use cases
  - Submit a merge request and run the set of merge request tests
  - Submit a merge request, run the set of merge request tests and some additional tests, or the full test suite
  - Run tests on a feature branch without opening a merge request
- Automated integration use cases
  - A component's main branch update triggers the downstream system update
  - A component triggers the tests in the downstream system, but does not update the system
  - The system updates its components



# Continuous Integration

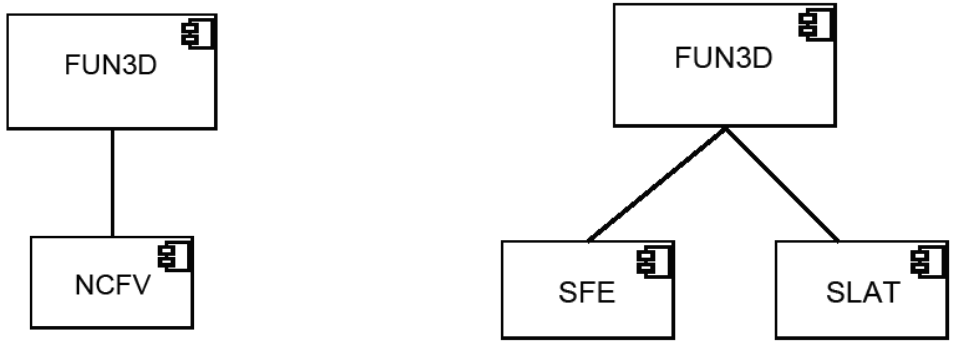
- Manual integration is needed for coupled component updates.
- Most merges are through the automated process.




FUN3D INTG Merge Commits.

# Component and Integration Testing

- Each component takes the responsibility of testing its own functionalities.
  - Unit, regression, performance, and acceptance tests are carried out at the component level and run intensively.
  - Integration tests are carried out at the system level and run less frequently.
- Some components are difficult to test without other components.
  - Use the deployed binaries from other components.
  - Create a system with limited components or with alternative components.



# Practices and Lessons Learned

- It is important to have well-defined interfaces. The update of interfaces adheres to the open-closed principle (open to extension but closed to modification).
- Automate as many procedures as possible.
- Lessons have been learned in maintaining the tests.
- Disadvantages of the approach
  - Takes extra efforts to keep track of contents related to the components, such as test cases, paperwork.
  - Developers need time to get familiar with the new approach.
  - When extending the interfaces, an agreement needs to be made among developers.

# Acknowledgments


- The authors would like to thank the FUN3D development team for the support.
- This work is carried out at NASA Langley Research Center Computational AeroSciences Branch. Xinyu Zhang is supported by NASA LaRC TEAMS3 contract.