Software Best Practices at Crustal Dynamics Data Information System (CDDIS): Steps to Consider

Justine Woo, Sigma Space Corporation
Sandra Blevins, Science Systems and Applications, Inc.
Patrick Michael, NASA Goddard Space Flight Center
Carey Noll, NASA Goddard Space Flight Center
Rebecca Limbacher, Science Systems and Applications, Inc.

Introduction

Geodesy and the earth sciences have a high level of reliance on software that varies in quality. Although scientific methods and the technologies used in these fields have evolved, efforts devoted to the design, development, and documentation of innovative software has lagged. The resulting imbalance from our reliance on software without long-term sustainability goals leads maintainability issues and discontinuities in software knowledge and mechanics. This creates an issue fundamental to science where ideally, experiments, methods, and results are reproducible. CDDIS has been working to address these issues and, in doing so, have implemented software engineering best practices while incrementally transitioning to a new software system. The system increasingly depends on automation and quality control measures making software sustainability and error catching essential to the entire process. Best practices in software engineering are quite extensive. In an effort to promote these practices, CDDIS has broken them down into four core items that persons across all levels of expertise in coding can implement: regression testing, repositories, documentation, and error collection (for automated systems). These best practices are the basis of traceable, sustainable, and ensuring software.

Regression Testing

Regression tests verify changes made to the codebase, including enhancements and bug fixes, to ensure the code continues to perform as expected without adverse changes to the output. Most modern programming languages have packages that support this. In addition, there are different levels of testing that you can apply based on the interconnectivity of your software: unit, integration, system, and acceptance.

To determine the level of regression testing required for implementation consider balancing the following: the magnitude to which the software must be scaled and the time and resources of the personnel working on the code base. Generally, the CDDIS runs full regression tests on its processing software; the code base is relatively extensive and very interconnected, but the CDDIS has the time and resources to dedicate to this. In addition, full regression testing saves time and resources by preventing errors before the software version is released. In some instances, full regression tests are not necessary and unit tests may suffice for standalone pieces of software. One of the core reasons to use unit testing in these instances is the limited amount of time needed for development, especially in the starting phase. Utilizing an organized set of unit tests and designing the code for flexibility and scalability (utilizing object-oriented programming (OOP), etc.) at the starting phase provides a solid baseline; from there, tests may be improved and refined as time allows. However, once software is pushed into production, the level of regression testing needs to match the scale of the software to prevent unintended effects that would prove more time consuming to fix.

Repositories
Repositories archive software, support version control, and allow for easy retrieval to download onto another system. There are multiple repository tools available for use including Git (which is used at the CDDIS) and Apache Subversion. No matter the tool used, the benefits gained from utilizing a repository include:

1. Version control - removes the need for software developers to keep multiple copies of software applications, which often leads confusion during collaboration
2. Older versions of software are easily accessible to correct for an error or for testing purposes
3. Changes to the software can be easily viewed for discrepancy identification
4. Software can be safely stored offsite in case of a hardware failure

On teams, branching and merging allows multiple people to work on the code simultaneously and to track the changes made by collaborators. Being able to track the changes made by other users is critical to understand why issues or hinderances may be occurring and, with source control mistakes can be easily rolled back. The CDDIS utilizes Git via BitBucket which connects to Jira, a project planning and bug tracking tool that provides additional team support.

The primary barrier to the use of a repository is that it's a change in routine and some of its capabilities, such as merging, add in additional changes to procedures. Training may help support overcoming some of these barriers as developing them into habits is the sustainable end goal. A last resort for some teams has been to automate this task. For example, managers have implemented software to detect and automatically push changes to a repository at timed intervals. While allowing for the newest version of the software to be saved (in case of a hardware failure, file corruption, etc.) the commit message is left blank or the code changes may be incomplete. It then becomes cumbersome to go through multiple commits to find an error, rather than having a message detailing the changes made, and rolling back all the code becomes impossible. Depending on the system, these benefits may outweigh the drawbacks and certain goals can be realized with this solution.

Documentation
The purpose of documentation is to communicate how the system and/or software works as a whole and to provide standard operating procedures for operations. Clearly written, detailed, and well-organized documentation provides a knowledge base and reference for the team that increases the longevity of the software.

Documentation can be done in a variety of ways based on the type of software being built and its extensiveness. Ideally software documentation should include instructions on how to compile and build the code, the version of the compiler, the operating system (OS), and the exact commands used to run the program.

Documentation is can be provided, to an extent, by external tools. For example, keeping software updates documented helps developers refresh themselves on items built in the past while informing other team members of edits; at the CDDIS, Git check-in messages fulfill this need. Within the code itself, several documentation generators may be utilized to translate comments into documentation, such as a Doxygen and JavaDoc.

The CDDIS utilizes:

1. Wiki: for general documentation including the standard operating procedure (SOP), roles and responsibilities for the different scientific techniques we support (GNSS, SLR, DORIS, VLBI), a layout of the tasks and information on the software supporting those tasks, and sections on how to utilize external tools such as the Git repositories.
2. Comments in the codebase: information includes the purpose of the software and explanation of methods
3. Git: check-in messages for documenting changes made overtime

Creating documentation can be time consuming but setting minimum documentation requirements can help set boundaries that make it less so and ensures the information communicated is valuable. Otherwise the documentation may not effectively communicate important and necessary information to developers and personnel. To ensure that the documentation is useful to end users, personnel at the CDDIS conduct informal code reviews alongside the documentation and a walkthrough of the SOP is given. This allows team members to suggest or make additions to the documentation based on their understanding of what was presented.

Error Collection

Error collection is critical for automated systems: it ensures small errors do not escalate into unmanageable ones, measures quality of the work being performed, and tracks and identifies errors during processing, enabling errors to be quickly corrected.

The CDDIS has two tables to log the errors found within our system:

1. Errors from data providers – records collected allow us to contact individual groups with details specific to their data files so they can provide corrections
2. Errors from the system – warnings and crashes are logged so that corrective measures may be taken

At the CDDIS, errors are primarily caught during processing and therefore exposed immediately. Secondary error checks are performed at timed intervals to check the performance and load on the overall system.

The primary issue faced with enforcing automated error catching is that it blatantly reveals how a system is performing. There are concerns about repercussions from transparency. It's important that management keeps an open mind about the fact that, when error catching software is first implemented, a substantial amount of errors may be shown and that it will take time to address them. - I don't understand what is being communicated in this section in between ** - Sandra

Conclusion

The benefits of these software best practices are known in the Earth science and Space Geodesy communities, yet various barriers are keeping them from being implemented. Utilization of helpful tools, such as those discussed in this paper, may help with this endeavor. The level of effort needed to ensure teams follow these best practices is highly dependent on individual team

members. Encouragement, practice, and staying consistent with examples given by teammates and management may help keep individuals motivated to keep up best practices, especially while working remotely or independently. Deeper level training designed to enhance knowledge of software best practices and their implementation on the systems that teams support and accountability will lead to a stronger individual understanding and whole team commitment to the best practices established for their system.