Curation and Refactoring of Legacy Software

Walton Macey
The Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
wmacey@vols.utk.edu

***Abstract*—Often it is difficult to build upon and learn from another individual's code and it is critical to have as much information as possible to build and use the program so that you can utilize or improve the software. Some of the greatest scientific minds have created software that often lays dormant, and has flawed or no metadata to assist in any future endeavors to utilize the software. Software curation involves maintaining, preserving and adding value to legacy research software. For legacy code still in active development, there are countless challenges to be faced in modifying and implementing changes especially for outdated or large scale scientific projects. Digital curation and data preservation are influenced by a desire to share with the wider research community. Necessary actions are required to promote curation and preservation of legacy programs especially in regards to national entities like the Department of Energy. We outline our approach to extending the lifetime of this software in two forms: shelved/dormant software and software that is still in active development.**

***Keywords*-Metadata; Curation; Legacy Software; Preservation; Software Productivity; ACME; ALM; Data Refactoring; ESTSC; software package**

I. Introduction

As software is increasingly a critical part necessary to achieve goals of the mission of organizations like the Department of Energy, it is necessary to identify problems of interest to, and find solutions that have an impact on the software development within DOE and that are of interest to the broader software engineering research community. A systematic measurement program used both to identify the problems and assess the impact of solutions is key to satisfying this need.

One of the critical challenges to improving DOE's software capabilities (and, therefore, accomplish its mission goals efficiently and effectively) is the general lack of understanding of the current DOE capabilities and resources that are distributed over numerous projects implemented over at least half a century. The first priority to achieve the vision described above would, therefore, be to inventory current DOE capabilities and resources.This involves both, collecting and curating software artifacts that have been developed so far evaluating the practices used to develop software in ongoing DOE projects.

We envision a two-prong approach with one direction focused on curating historic artifacts and the second direction focused on assessing representative DOE projects. Both efforts will focus on developing tools to support this measurement activity and make it possible to repeat it on a regular basis as well as identifying the needs and priorities for tools that would support the ability to upgrade the existing resources for the new-generation computing infrastructure. The tools for ongoing projects would include modularization capabilities, estimates of effort needed to implement and upgrade various functionalities, and recommendations for best practices to achieve specific goals.

In the specific case of our proposed restructuring of the data architecture for ACME, we expect sub-model implementation and modification to be much more straightforward for both scientists and engineers. To reduce risks to a project of this size undergoing active development, we have designed an iterative approach. For scaling up, we have incorporated steps to automate repetitive manual work and we rely on the existing testing infrastructure suites to verify the success of the restructuring.

II. Curation of Software Artifacts

The Energy Science and Technology Software Center (ESTSC) is the Department of Energy's centralized software management facility which licenses and distributes scientific and technical software system. The center is not a modern system. Currently, software packages are burnt to CD and then shipped to the requester. Massive improvements have been made in the form of the modern advancement of DOE Code, a new DOE software services platform and search tool for DOE-funded code which provides functionality for collaboration, archiving, and discovery of scientific and business software. This new service provides a modern method for future DOE collaborations. The remaining difficulty is utilizing the thousands of currently stored software packages some of which have been copied and distributed to the community, others which may have been untouched for years.

At the Office of Scientific and Technical Information (OSTI) in Oak Ridge there are currently over 3300 software packages managed by the ESTSC and physically stored as CD's with accompanying manuals and information stored in folders. With decades of software submissions, during which the standardization and accounting of the submissions may have varied dramatically, the metadata concerning each package is inconsistent and often incorrect. In the past, a software is copied to CD on a request by request basis, so some of the software in the center may still exist in an outdated form of physical media. For these legacy softwares, there is a clear need to identify key information concerning the quantity, type and variety of software packages stored at the center. In the past few decades, a large majority of software submissions have simply been notification that an open source version is available. The links to open source repositories may or may not still be valid. Additionally, code that was previously not open source may now be. Also, the ESTSC database contains records of over 3300 software packages, but there are significantly less physical records within the center.

III. Implementation

The initial goal was to inventory and gather as much knowledge to allow a seamless future digitization of all software packages. A Brute force approach of opening each package at a time seemed hasty and potentially near impossible. The software center has accumulated packages over decades and the method and manner of documenting and accumulating the software has lacked consistency and standardization in appropriately defining the metadata. A full audit would allow appropriate prioritization and accuracy when transitioning the entire collection into a digital form.

We decided to define a landscape and provide an overall picture to make up for inconsistencies and errors in the existing database. To perform this analysis, I first performed a random sampling of a small number of software packages to appropriately access the accuracy of the metadata to physical. Random sampling consisted of cross validating individual software with metadata in the center's database, information gathered from the physical folder of the software package and extensive on-line verification research. From our random sampling of 100 software packages we uncovered several troubling statistics which limit the ability to utilize legacy software. The random sampling revealed that over 28 percent of files were misclassified in their open source designation. Many data fields for the metadata of the software collection were incomplete or convoluted in nature. Additionally, of the 100 packages sampled 9 did not have a physical folder present within the ESTSC. Finally, roughly 30 percent of software packages controlled by the software center did not have a known Open Source classification at all.

With these known issues providing an educated understanding of the state of the ESTSCs software collection we began implementing methodologies to correct and validate the entirety of the collection. To correct flawed and empty metadata from the centers database, we designed a simple keyword extraction script[1] in python to search for specific keywords with which we could potentially populate an empty data parameter. For example, over 80 percent of software packages do not indicate which programming language is used. With these extraction scripts, we could scan the "software details" data field which usually consisted of several paragraphs of descriptive information. The script would output potential matches, which needed to be verified within the context of each software package.

[1]All code outlined in this paper as well as a visual presentation over this research can be found https://github.com/ssc-oscar/curation

IV. Full Audit Analysis

In order to deliver a final full audit of the content for every software package in the center, we had to physically inventory the collection and perform online research for each package to validate or determine its open source classification. A physical inventory consisted of cross validating the physical folders containing software packages stored within the ESTSC with the entire inventory cataloged in the center's database. This lengthly process revealed that over 1000 softwares had no folder or CD stored in the center. This could be explained if each of these 1000 were designated open source and did not require a hard copy within the center, but a little over 1/3 of the missing were classified as non-open source.

To determine the validity of each package's open source designation, we developed a programmatic solution to scour online repositories for any trace of an open source version of any software packages controlled by ESTSC.We created a custom search tool in python utilizing Google's custom search API and focusing on common source code hosting sites such as Github, Bitbucket, codebase and gitlab. The custom search program utilizes keywords from each software package's metadata to search for any potential matches according to information found in a website's header description. The custom search tool returned a list of potential urls and their associated website header description. The returned list of url information is mapped to the keyword, software package name and id number that initially triggered the potential open source match.

To eliminate the overwhelming presence of false positives, we analyzed the results to attempt to reduce our final potential open source locations. Most often, parsing libraries and bibliographies would falsely trigger a potential positive. By eliminating matches containing keywords and those ending in file types common to parsing libraries and bibliographies, we were able to narrow down results by 95 percent. For our 950 unclassified software packages, an initial run of the custom search tool resulted in 5700+ potential open source locations for these 950. After removing the false positives using the methodologies mentioned, the final results were reduced to just over 400 potential matches.

Following this approach, we were able to fully audit the software collection of the ESTSC. We have added value to legacy research software by efficiently providing an accurate and complete inventory of the state of the DOE software collection. Now that the metadata for the center's database is confirmed to be accurate and validated, OSTI engineers can move forward with a digitalization of the collection utilizing the informed understanding of the landscape of the software collection to create optimized approach for efficient digitalization of the software packages

V. Active Development Software Introduction

In large-scale Earth System simulation codes, such as the Accelerated Climate Model for Energy (ACME), complex user-derived data types (containing large number of variables) are designed to represent the interactions of atmosphere, ocean, land, ice, and biosphere to project global climate under a wide variety of conditions.

The following is our proposed approach to restructure the data architecture of a land component within the ACME project while the project is undergoing active development. The data architect for the land subsystem defines the new datatype requirements that would greatly simplify the implementation of terrestrial land submodels by converting more than fifty to just eight primary data-types. Since the code is developed with the community governance, we have to ensure that the restructuring does not interfere with the other development which, with dozens of changes occurring every day, make it impossible to work on a shared development branch. The active development also occurs on almost five hundred branches, making it extremely difficult to assess potential interactions.

To address these challenges we have designed and started an iterative procedure for implementing the data restructuring and estimating both the effort it takes to restructure and the effort would save once the restructuring is implemented.

VI. Accelerated Climate Model for Energy

The Accelerated Climate Model for Energy (ACME) has been designed to accelerate the development and application of fully coupled, state-of-the-science Earth system modeling, simulation and prediction for scientific and energy applications. With an early emphasis on improving software design and practice, development has included maintaining build, test, and performance tools for the relevant computer platforms, and providing rapid development and debugging capabilities to the team. The ACME code repository both expedites the merging and testing of the fully coupled system and supports a distributed development environment where separate features are being co-developed at different sites [1].

Inside ACME, the scientists/modelers have developed a terrestrial land model,called the ACME Land Model (ALM). ALM is a process-based model with a collection of key biogeophysical and biogeochemical functions that represent the energy-water-biogeochemical interactions between the atmosphere and the terrestrial landscape. ALM contains hundreds of energy, water and ecosystem variables which may be used or modified by many terrestrial ecosystem functions/modules. Designing data types to contain variables in a more modular fashion can facilitate scientific model development and unit testing [6].

In our previous work [3] we have described quantitative ways to evaluate such data modularity and a search algorithm that finds the optimal assignment of variables to modules. Based on the prior analysis and ongoing and future requirements of ALM development, the fundamental data types of ALM needs to be "re-modularized".

VII. Background

Modularization of software systems is a well known and widely used concept in the field of Software Engineering [2], [4]. However, the discussion on software modularity tends to revolve around modularity of the source code [4]. In this paper we are concerned with the modularity of variables and data[2] used in software where very complex data structures are shared among many specialists, each focusing only on a small subset of that data structure. A guiding principle for the new data structures has been to represent the primary functionality of the model as isolated modules connected through clearly defined interfaces.

In order to achieve this goal, it is worth considering why ALM simulations involve such complex data structures. To run very large simulations the data representing landscape surface is subdivided into a grid. That way, in each simulation step the information is exchanged only among neighbors of each grid cell, reducing the communication overhead on the supercomputers in which the simulation is being run and improving execution performance. Variables representing the state of the grid cell are collected into a single structure simplifying message passing. However, the human aspect of the scientist modeling a specific process within a cell, has not been previously considered to be as important and the proposed restructuring is aiming to address this.

VIII. Restructuring Road Map

The critical part of the restructuring involves separating the task into small pieces that could be easily and rapidly implemented and tested. We are also starting from manually implementing changes and using the experience to design automation tools that would reduce the effort spent on repetitive tasks. In particular, our initial implementation would guide us as to which parts of the procedure could benefit from automation and whether the introduction of automation would reduce the overall effort.

The transition involves converting the current data structure of ALM.v0 with necessary modularity improvements, and developing a hybrid data structure for future ALM development (such as ALM.v1). In addition, we would like to evaluate the impact of ALM data structure on model development and software productivity.

[2]We use variables and data interchangeably in this paper.

The analysis of the current data structures revealed several types of variables that may require different types of treatment. At the higher level, we can classify the variables according to scientific domains to which they pertain, according to the part of the grid cell they represent, and according to their primary purpose being simulation or validation. In contrast to the previous implementations, the proposed architecture has separate types for variables at each level of the sub-grid hierarchy [5].

Our proposed data types would intrinsically define the relevant sub-grid level. In order to satisfy individual subroutines or model capabilities, these data types can be constructed as collections of pointers to the foundational types [5]. As new variables are added during development, the data structures should ensure that these variables are introduced in a logical part of the sub-grid hierarchy and should support energy balance constraints that are used to validate the simulation. For example, "a new soil column representation or a new vegetation cohort representation can be introduced and evaluated easily against existing module as long as the between-level interface variable naming and meaning is maintained" [5].

Because ACME already has extensive automated test suites for the ongoing development of the model, we will utilize the existing testing procedure to verify that the overall model will be unchanged by our implementation of these new data structures. ACME uses an automated, regression test system for more efficient and robust development that allows the model to maintain a reliable and releasable state. We will be running the acme-developer test suite with each iteration, to instill a basic confidence that the set of changes will not break or change the model according to initial baselines.

The specific work breakdown of the data restructuring has the following steps:

- Manually convert one land datatype
- Systematically test for code integrity
  - Single case testing (compile/execution)
  - Regression testing (acme developer test suite)
  - Bit-for-Bit output comparison
- Learn from analysis of manual implementation
  - How much effort and time did the process take
  - What parts can be automated or executed more efficiently
  - How best to improve productivity

The overall objective is to finalize a new data structure for ALM.v1 by getting the code into main trunk and to summarize the lessons learned for the future restructuring.

IX. Summary

Over the past five decades massive amounts of legacy softwares have been developed at huge costs. When wishing to utilize, learn from or contribute to legacy software, a user faces many challenges linked to the outdated documentation and functionality of these software packages. Even if the desired software can be located, there is no guarantee that the software will run on the latest architecture nor that a more recent version can be attained elsewhere. With the aim to assist in managing legacy software in an effective manner, we outlined our approaches to curating a collection of legacy software packages and to refactoring one of these legacy software projects.

For our approach to curating a legacy software collection, we first uncovered potential challenges and issues through a random sampling analysis. We next corrected and validated our resulting data to deliver a complete and verified report for the software collection.

When exploring a refactoring initiative of an active legacy software development project, we initially designed an iterative approach to reduce effort spent on repetitive tasks through automation. By developing assistance tools emphasizing refactoring ease and defining an effort refactoring model, we ensure that any future refactoring attempts can be repeated and justified as an affordable improvement for the code base.

References

[1] D. Bader, W. Collins, R. Jacob, P. Jones, P. Rasch, M. Taylor, P. Thornton, and D. Williams. Acme project strategy and initial implementation plan. Technical report, Department of Energy, 2014.

[2] O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare. *Structured programming*. Academic Press Ltd., 1972.

[3] Y. Ma, T. Dey, and A. Mockus. Modularizing global variable in climate simulation software: position paper. In *Proceedings of the International Workshop on Software Engineering for Science*, pages 8–11. ACM, 2016.

[4] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[5] P. Thornton. Data structures for acme land requirements. Technical report, 2016.

[6] D. Wang, Y. Xu, P. E. Thornton, A. W. King, C. A. Steed, L. Gu, and J. Schuchart. A functional test platform for the community land model. *Environmental Modelling and Software*, 55:25–31, 2014.