Agile Lifecycles for Research-driven CSE Software

Roscoe A. Bartlett, Oak Ridge National Laboratory
Michael A. Heroux & James M. Willenbring, Sandia National Laboratories

10/08/2013

Introduction

Development of production-quality software from a research-driven computational science and engineering (CSE) project is challenging. CSE software products tend to be long-lived and multi-component. Ideally they should have reusable components and rely on external components developed by other expert groups in order to ensure state-of-the art capabilities. However, this ideal is seldom achieved. Commercial software can become unavailable and software from other research organizations can be unreliable and poorly supported. Many issues must be considered by a research-driven software project: research productivity and credibility, reuse and upgrades, maintenance, support, shared development, continued research with mature software, balancing backward compatibility and change and more.

While a great deal of work has been done in the general area of software lifecycle models, Lean/Agile lifecycle models seem particularly attractive for most CSE projects [7, 6, 3]. There seems to be little work attempting to define software lifecycle models for research-driven CSE software.

The primary purpose of this short paper is to argue for research into the broad adoption of a modern Lean/Agile-consistent software lifecycle model and framework that take into account the particular needs of the CSE community for both research and production projects. It is based on the proposed TriBITS Lifecycle Model [2]. One previous attempt to create a viable lifecycle model for CSE can be seen in the Trilinos Lifecycle Model [8] which provided for transitions of CSE software from research, to production, to maintenance (and later death). Ideas from that earlier work were carried into the development of the proposed TriBITS Lifecycle Model.

An Agile Lifecycle Model

The primary goal of the proposed lifecycle model is the development of *Self-Sustaining Software* which is defined to have the following attributes:

- **Open-source**: The software has a sufficiently loose open-source license allowing the source code to be arbitrarily modified and used and reused in a variety of contexts.
- **Core domain distillation document**: The software is accompanied with a short focused high-level document describing the purpose of the software and its core domain model [4].
- **Exceptionally well tested**: The current functionality of the software and its behavior is rigorously defined and protected with strong automated unit and verification tests.
- **Clean structure and code**: The internal code structure and interfaces are clean and consistent.
- **Minimal, controlled internal and external dependencies**: The software has well structured internal dependencies, and minimal external upstream software dependencies that are carefully managed.
- **Properties apply recursively to upstream software**: All of the external upstream software dependencies are also themselves self-sustaining software.
- **All properties are preserved under maintenance**: All maintenance of the software preserves above properties (by applying Agile/Emergent Design, Continuous Refactoring, and other good Lean/Agile software development practices).

Software with the above properties poses minimal risks to downstream customer CSE projects. To the extent that a piece of software is not consistent with any of the above properties is poses a risk to downstream customer projects. The motivation for these properties and other issues are discussed in detail in [2].

While the goal of the proposed Lean/Agile lifecycle model is to produce self-sustaining software, other properties of the software are also important and the process by which software is first created in a

research project and is later matured has to be considered. Therefore, the proposed TriBITS Lifecycle Model defines several different maturity levels for CSE software:

1. **Exploratory (EP)**: Primary purpose is to explore alternative approaches and prototypes.
2. **Research Stable (RS):** Developed from the very beginning in a Lean/Agile consistent manner. Strong unit and verification tests (i.e. proof of correctness) are written as the code/algorithms are being developed (near 100% line coverage). Has a very clean design and code base maintained through Agile practices of emergent design and constant refactoring [1]. Generally does not have higher-quality documentation, user input checking and feedback, space/time performance, portability, or acceptance testing. Would tend to provide for some regulated backward compatibility but might not. Is appropriate to be used only by "expert" users. Provides a strong foundation for creating production-quality software and should be the first phase for software that will likely become a product.
3. **Production Growth (PG):** Includes all the good qualities of Research Stable code. Provides increasingly improved checking of user input errors and better error reporting. Has increasingly better formal documentation as well as better examples and tutorial materials. Maintains clean structure through constant refactoring of the code and user interfaces to make more consistent and easier to maintain. Maintains increasingly better regulated backward compatibility with fewer incompatible changes with new releases. Has increasingly better portability and space/time performance characteristics. Has expanding usage in more customer codes.
4. **Production Maintenance (PM):** Includes the good qualities of Production Growth code. Primary development includes mostly bug fixes and performance tweaks. Maintains rigorous backward compatibility with typically no deprecated features or breaks in backward compatibility. Could be maintained by parts of the user community if necessary (i.e. as "self-sustaining software").

The transition between the RS, PG, and PM phases is meant to be smooth and without risk. Existing software is grandfathering in using the Legacy Software Change Algorithm [2, 5]. There are many other details and considerations related to the definition and proposed implementation of this lifecycle model that cannot be discussed here for lack of space (see [2]).

Summary and Research Opportunities

We propose the adoption of a Lean/Agile lifecycle model for research-driven CSE software. The proposed model, if widely adopted, could dramatically improve the productivity and impact of CSE research and applications by providing a wide range of compatible high-quality advanced software capabilities from a wide range of areas. The primary research questions for the proposed Agile lifecycle model relate to how well it will work in practice, what level of training will be needed to get is used effectively, and finding ways to measure the impact using various local and global measures. There is an ongoing attempt to implement this lifecycle model in the Trilinos project[1]. A broader effort would include applying and adapting this model to other projects as well.

References

[1] S.L. Bain. *Emergent design: the evolutionary nature of professional software development.* Net Objectives, 2008.

[2] R. A. Bartlett, Michael A. Heroux, and James M. Willenbring. Overview of the tribits lifecycle model: A lean/agile software lifecycle model for research-based computational science and engineering software. *e-science*, 2012 IEEE 8th International Conference on E-Science:1–8, 2012.

[3] K. Beck. *Extreme Programming (Second Edition).* Addison Wesley, 2005.

[4] E. Evans. *Domain-Driven Design.* Addison Wesley, 2004.

[5] M. Feathers. *Working Effectively with Legacy Code.* Addison Wesley, 2005.

[6] R. Martin. *Agile Software Development (Principles, Patterns, and Practices).* Prentice Hall, 2003.

[7] M. Poppendieck and T. Poppendieck. *Implementing Lean Software Development.* Addison Wesley, 2007.

[8] James M. Willenbring, Michael A. Heroux, and Robert T. Heaphy. The Trilinos software lifecycle model. In *ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops*, page 186, Washington, DC, USA, 2007. IEEE Computer Society.

[1] http://trac.trilinos.org/wiki/TribitsDevelopmentPractices