OAK RIDGE
NATIONAL LABORATORY
MANAGED BY UT-BATTELLE
FOR THE DEPARTMENT OF ENERGY

Proposal: Application of Agile Software Development Process in xLPR ORNL-2012/41412

November 2012

Prepared by
Hilda B. Klasky
Paul T. Williams
B. Richard Bass

UT-BATTELLE
ORNL-27 (4-00)

ORNL-2012/41412

Computational Sciences and Engineering Division

Proposal: Application of Agile Software Development Process in xLPR ORNL-2012/4141

Hilda B. Klasky

Paul T. Williams

B. Richard Bass

Date Published: September 2012

Prepared by

OAK RIDGE NATIONAL LABORATORY

Oak Ridge, Tennessee 37831-6283

Managed by

UT-BATTELLE, LLC

for the

U.S. DEPARTMENT OF ENERGY

under contract DE-AC05-00OR22725

Contents

Introduction

Reflecting on what we have seen regarding of the xLPR team need to report progress to project management, and the current evolution of facts in which teams have started to write all sorts of code before formal approval of the long list of documents to be delivered, we feel the prevailing need to recommend that the xLPR software development process be modified for reasons outlined herein. In the following, we discuss the deficiencies of the current approach, and then present a proposed solution that addresses those deficiencies.

We want to stress that our proposal to change the software development process in the xLPR project aims to excel on being:

- **Results Oriented** by building software prototypes extremely early in the development process as a response to the early requirements of the user. A series of prototypes or a series of modifications to the first prototype will gradually lead to the final software product,
- **Increase Software Quality** by handling changes and identifying issues early during the development,
- **Focus on Customer Satisfaction** by promoting communication between the team and the final customer through the project lifespan,
- **Light-weight**: there is less are fewer number of document deliverables and approvals of these deliverables are not cumbersome.
- **Resources Saving**: are achieved by removing overhead activities, time, and tasks, team members are focused on producing results.
- **Maintain Team Morale** by using a light weight process that will keep team members engaged and will show progress in their accomplishments. Motivation is very important to increase productivity.
- **Allowing managers to interrogate collaborative tools at any given time to determine the status of each module** by using the issue tracking system JIRA based on a relational database; xLPR team leaders and managers will be able to see module status automatically in the dashboard of the project.
- **Complying** with the **ASME NQA-1-2008** (including Addenda **2009**) Quality Assurance Requirements for Nuclear Facility Applications:

- Part I
    a. Requirement 3, Paragraph 800;
    b. Requirement 11, Paragraphs 100, 200, 400, 500, and 600;
- Part II, Subpart 2.7; and
- Part II, Subpart 2.14, when applicable.

The problem: xLPR is following a *Waterfall* Software Development Process Type: the *Spiral* approach

We believe that the current xLPR SQA approach is guiding the xLPR team to follow what is generally termed a "*waterfall*" type software development model, in which each phase follows the next in sequence. The *waterfall* software development process (or model) is a sequential design process, often used in the early days of software development, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation, and Maintenance. ***The waterfall development model originated in the manufacturing and construction industries, which are highly-structured physical environments, where after-the-fact changes can be prohibitively costly, if not impossible***. Since no formal software development methodologies existed in the early days of computing (the late 1950s), this hardware-oriented model was adapted for software development and eventually formalized into a generally accepted practice by around 1970. In the software industry today, the term "*waterfall*" is typically used to describe a flawed, nonworking (although still used) model for software development practice. (See a review of the pros and cons of the waterfall model at the end of these notes in Annex 2 Waterfall Model of Code Development – Pros and Cons.)

All of the software development processes that are based on the *waterfall* model ***include the expectation of having a constant approval process;*** the latter can prove to be **very resource-consuming**. These types of development processes are by nature documentation- and plan-driven approaches, as captured in the following sequence:

SRD->approval->SDD->approval->V&V->approval->Code->approval

History has shown that projects using the *waterfall* software development processes **usually** run over budget, over time, and deliverables are partially achieved in the best-case scenarios. To back

up this statement we are including a recent research study performed at the University of Southampton, School of Electronics and Computer Science.

The reason for these deficiencies is that the team focuses on generating the documentation and getting it through the approval process. The effort to generate of the final product can be diluted and, many times, never completed. By the time part of the documentation is complete, the project resources have been exhausted. And these facts can already be observed in the development teams of xLPR, who are naturally moving to write code before completing and formal approval of their list of documents, as required by to the *Spiral* approach.

The proposed solution: follow an *Agile* Software Development Process type

To counter deficiencies with the *waterfall* type process (the *Spiral* approach, in the case of xLPR), software development groups evolved an alternative that **serves to decrease the time spent in the development cycle**. The sequential phases were mostly removed and **only those essential steps that produce the expected deliverables, i.e., dialogue, coding and testing, were retained**. This more streamlined approach is identified in the industry as the "*Agile*" software development process. Per our software development experience at ORNL, we propose moving away from the current *Spiral* approach and adopting this *Agile* process for the xLPR project. The *Agile* software development process is a very lightweight process that

- employs short iteration cycles;
- actively involves users to establish, prioritize, and verify requirements; and
- relies on tacit knowledge within a team, as opposed to documentation.

This process takes into account

- the realization that most users do not have a fully-formed idea about their needs, and
- the problem of missing and changing requirements, recognizing that most changes in requirements occur within a project's life span.

The *Agile* process suggests building prototypes extremely early in the development process as a response to the early requirements of the user. A series of prototypes or a series of modifications to the first prototype will gradually lead to the final product. *Agile* is meant to embody short

iterations, where the system is improved in each cycle. In addition, development proceeds step-by-step with the user, as insight into the user's own environment and needs is accumulated.

The sequence of steps in the *Agile* approach consists of the following:

implement->test->review->specify->redesign/refactor->re-implement.

Again, starting with implementing the system, the emphasis here is on *Agile* development. Thus, the team focus is results oriented, not process oriented.

In the following sections, we propose detail a proposal for deliverables' schedule that we believe will help alleviate some of the problems highlighted in the first xLPR QA internal audit.

Proposed Deliverables

Our proposed change to an *Agile* software development process calls for only two deliverables from the xLPR groups that develop software:

1) **Deliverable Report**: A draft report shall be submitted for team review one month before the EPRI QA audit scheduled during June 2013. The final report must be completed by the end of the project. Below, an outline of the report is suggested.
    a. Section 1. Introduction
    b. Section 2. List of Requirements from JIRA
    c. Section 3. Overall architectural design showing data flow
    d. Section 4. List of test cases from JIRA
    e. Section 5. Conclusions and lessons learned
2) **Source code/executable** every 6 – 8 weeks of development cycle.

Proposed Schedule

Table 1 presents a proposed schedule which aims to help the integration of all of the modules by early 2013. It shows that each development team will have a short development cycle of 6 to 8 weeks max. In parallel, one deliverable document report will be prepared. Teams are expected to:

- enter (1) requirements and (2) test cases into the JIRA system
- perform the tasks in an *Agile* fashion by the sub-groups
- maintain a documented trace of the evolution of the requirements/tests/bugs/improvements, as well as resolution of the latter, in the JIRA system for the benefit of the team
- review the test results, modify or add new requirements/test cases from the lessons learned in previous iterations
- present a list of requirements/bugs/improvements (taken from JIRA) as a release note at the end of each development cycle
- approve a list of requirements/bugs/improvements (taken from JIRA) to be implemented in the next iteration (deployment)