An Analysis of Process Characteristics for Developing Scientific Software

Diane Kelly, Royal Military College of Canada, Canada

ABSTRACT

The development of scientific software is usually carried out by a scientist who has little professional training as a software developer. Concerns exist that such development produces low-quality products, leading to low-quality science. These concerns have led to recommendations and the imposition of software engineering development processes and standards on the scientists. This paper utilizes different frameworks to investigate and map characteristics of the scientific software development environment to the assumptions made in plan-driven software development methods and agile software development methods. This mapping exposes a mismatch between the needs and goals of scientific software development and the assumptions and goals of well-known software engineering development processes.

Keywords: Extreme Programming, Professional End-User Developer, Scientific Software, Scientific Software Development, Software Development Process

1. INTRODUCTION

The methods for developing scientific software have recently garnered the attention of the software engineering community, resulting in several studies of scientists as “end-user developers”. Segal (2004) differentiated scientists as “professional” end-user developers, pointing out their extensive knowledge in a technical domain and their high comfort level in writing code. There have been suggestions (Ackroyd et al., 2008; Pitt-Francis et al., 2008; Segal, 2005; Wood & Kleb, 2003) that agile software development methodologies (Highsmith & Cockburn, 2001) rather than “plan-driven” methodologies (Royce, 1970) are better suited to the development of scientific software.

This paper examines the characteristics of the environment in which scientists develop their own software and compares these to the assumptions and goals of plan-driven and agile software development methodologies. The conclusion is that there is a mismatch in both types of methodologies when applied to the development of scientific software.

We first define what we mean by scientific software and discuss characteristics of its development. In Section 3, we make initial observations about these characteristics and the assumptions inherent in plan-driven and agile software development methods. Section 4 uses a variety of models to further explore the
characteristics of scientific software development as opposed to inherent assumptions that underlie plan-driven and agile development methods. Section 5 summarizes our findings and Section 6 concludes.

2. SCIENTIFIC SOFTWARE

2.1. Definition of Scientific Software Development

A project that we categorize as scientific software development has the following three characteristics. First, the software is written to answer a scientific question. The question can be general, such as, “Is it safe to operate this nuclear generating station?” or specific, such as, “Can I track satellites with this telescope?” Second, the writing of the software necessitates the close involvement of someone with deep domain knowledge in the application area related to answering the question. The person with the deep domain knowledge, the scientist, usually writes the software. This is Segal’s professional end-user developer. We have observed cases where the scientist has not written the software, but the software developer has had to become enough of an expert in the domain to understand and implement what the scientist is describing. Third, the software provides output data to support the scientific initiative. A human is in the system loop to examine the data and make observations and ultimately, answer the scientific question. The expectation is that the data provided by the computer solution is correct and will not misguide the scientist answering the question.

The science application can be solving large systems of differential equations, analyzing immense amounts of data such as imagery or bioinformatics, and computing using analytical and empirical models. Our definition of scientific software, however, excludes the following: control software whose main functioning involves the interaction with other software and hardware; user interface software that may provide the input for and report of scientific calculations; and any generalized tool that scientists may use in support of developing and executing their software, but does not of itself answer a scientific question.

Scientific software may have an extensive graphical user interface or interact with other software or hardware to obtain data. It may run on complex multi-processors and require middleware support to make use of hardware capabilities. But in the following discussion, we are talking exclusively about the code that implements the science application whether it is designed as a separate module or inserted into a product that includes these other parts. We contend that this core scientific part of the product requires different considerations when it is being developed.

2.2. Characteristics of Scientific Software

Several studies have looked at the characteristics that differentiate the environment and activities of scientific software development from the development of other types of software (Basili et al., 2008; Boisvert & Tang, 2001; Matthews et al., 2008; Post & Kendall, 2004; Sanders & Kelly, 2008; Segal, 2003, 2004). Some characteristics are invariant whether the software is developed by a scientist working alone (Sanders & Kelly, 2008) or a large distributed team (Matthew et al., 2008), or whether the code is a thousand lines of code or over a million.

The most noteworthy of the characteristics is the focus on the need for correctness, or trust, in the software. Some scientists stated baldly, “the software must never lie to me” (Sanders & Kelly, 2008). As explained by Kelly (2008) “This [trustworthiness] takes priority over efficiency, reliability, user-friendliness, cost-effectiveness, and portability… If the software gives the wrong answer, all other qualities become irrelevant.” Hatton et al. (1994, 2007) and Miller (2006) both give examples of cases where correctness of the software was the sole factor in success or failure. Hatton describes an example of industrial software where the success of oil-exploration
Related Content

A Theoretical Investigation and Extension of a Model of Information Technology Architecture Maturity
Randy V. Bradley and Terry A. Byrd (2011). Organizational and End-User Interactions: New Explorations  (pp. 352-375).
www.igi-global.com/chapter/theoretical-investigation-extension-model-information/53099?camid=4v1a

A Model of System Re-Configurability and Pedagogical Usability in an E-Learning Context: A Faculty Perspective
www.igi-global.com/chapter/model-system-configurability-pedagogical-usability/62795?camid=4v1a

From E-Learning Tools to Assistants by Learner Modelling and Adaptive Behavior
www.igi-global.com/chapter/learning-tools-assistants-learner-modelling/24482?camid=4v1a
A Survey of the AACSB Universities’ Policies on Software Copying