Chapter 6
Pragmatic Software Engineering for Computational Science

David Worth
Science and Technology Facilities Council, UK

Chris Greenough
Science and Technology Facilities Council, UK

Shawn Chin
Science and Technology Facilities Council, UK

ABSTRACT

The purpose of this chapter is to introduce scientific software developers to software engineering tools and techniques that will save them much blood, sweat, and tears and allow them to demonstrate the quality of their software. By introducing ideas around the software development life cycle, source code analysis, documentation, and testing, and touching on best practices, this chapter demonstrates ways in which scientific software can be improved and future developments made easier. This is not a research article on current software engineering methods, nor does it attempt to specify best practices. Its aim is to introduce components that can be built into a tailored process. The chapter draws upon ideas of best practice current in software engineering, but recommends using these only selectively. This is done by presenting details of tools that can be used to implement these ideas and a set of case studies to demonstrate their use.

INTRODUCTION

The premise of this chapter is that most existing scientific software is of low "quality" from the point of view of standards of "best practice" in professional software engineering. Not least because the development of "quality software" of that kind, i.e., involving code that is clearly and manifestly structured, and readily intelligible to new users through reading the code, is not currently a priority for scientific research. Those involved in developing code for CSE research projects no doubt feel justified in believing that their codes are of good "quality", after all they generate valid scientific results that pass peer review, however here we would define that as "effective" software

DOI: 10.4018/978-1-61350-116-0.ch006

rather than good "quality". In this chapter, we argue that scientific software could benefit from the pragmatic application of software engineering tools and techniques in ways that would improve the lot of both developers and users. That is, in terms of increasing productivity and making software not only useful for its dedicated research purpose in a given project, but also from the point of view of making it more readily shareable and re-useable. These features are increasingly important in academic CSE where collaborative teams are becoming more common and repositories of software such as the one hosted by National Centre for Atmospheric Science (NCAS) or the CCPForge facility run by the STFC Computational Science and Engineering Department are being set up by funding bodies.

Our judgement of "quality" may seem harsh, and the authors make it with reluctance, but as experienced scientific programmers we know that this admission is the first step in a process of improvement. There are many reasons why quality may be low, but the foremost is that codes have been developed in a piecemeal way over many years, from what was originally a research code. There is no suggestion that developers set out to write bad code or wish to leave it in such a state.

Software quality has many definitions but in this chapter the term can be taken to mean how easy the code is to maintain and develop. Exactly what this means is dependent on those who develop and use the software and no one set of instructions can cover all situations. As will be made clear in this chapter there is no single process derived from "best practice" that fits scientific software development.

This assessment of quality bears no essential relationship to the scientific outputs of the software. Users are generally happy with the results and publish papers in refereed journals, boosting confidence that the software is adequate. However, the ability to measure, report and improve software quality will have an impact on the users' perception. Being able to demonstrate, for example, that source code has been checked in a particular way, that a certain sort of testing has reached a quantifiable level, or that the introduction of new features has not caused any damage, will increase the users' confidence even further.

Developers, on the other hand, often believe that "software engineering", by adding the burden of documentation and reporting, will retard the introduction of new features into a code. This is why the word *pragmatic* has been specifically used in the title of this chapter. Unthinking implementation of a software engineering methodology picked at random is not the way to proceed. In general, good computational scientists research and evaluate different modelling approaches and investigate solution algorithms before making decisions about what will be implemented. On occasions no single idea will do, and then a selection of ideas will be adapted to suit the situation. The idea of *pragmatic* software engineering is simply the application of these same research skills to the way in which software is developed.

Adopting software engineering ideas in this reasoned manner will improve the developer's lot by making it clear what a particular piece of work is to achieve, making source code more understandable, making it easier to report what has been done, improving testing during development (and as a consequence reducing the bugs in released code) and creating software that is easier to maintain and extend. Going further and creating an agreed process for developing a particular code will ensure that all developers work along similar lines, new developers can "do it right" and all developers work more collaboratively. Overall this should increase the pace of development, creating a better world for all (including funding bodies).

In this chapter we present a wide variety of tools and techniques, along with a range of approaches to the process of software development. No single approach or set of tools can be considered "the best" and applied without thought. They are offered as an aid to good software development, making the life of developers easier, and giving