Chapter 24
A Framework for Testing Code in Computational Applications

Diane Kelly
Royal Military College, Canada

Daniel Hook
Engineering Seismology Group, Canada

Rebecca Sanders
EA Pogo, Canada

ABSTRACT

The aim of this chapter is to provide guidance on the challenges and approaches to testing computational applications. Testing in our case is focused on code testing for accuracy as opposed to validating the science models or testing user interfaces. A testing framework is used to present the different challenges. Discussions cover topics such as test oracles and the tolerance problem, testing to address specific goals rather than testing as a process, areas of risk inherent in developing and using computational software, a testing mindset, and the use of technical reviews. Three observational studies are included to illustrate different techniques, problems, and approaches. There is no prescribed way of testing computational code. Instead, an awareness of risks and challenges inherent in computational software can provide the necessary guidance.

INTRODUCTION

Mistakes find their way into all nontrivial pieces of software. This is supported by both our experiences and by published research. For example, Les Hatton (1997) conducted a series of experiments in which he found that some scientific programs thought to be "fully tested" (p. 30) harboured serious code faults.

DOI: 10.4018/978-1-4666-4301-7.ch024

For scientific software to be trusted, the developers of scientific software must make a reasonable effort to detect and correct the faults in their code. This reality is strongly expressed by Donoho, Maleki, Shahram, Ur Rahman, & Stodden (2009) in an article on reproducible computational research in which they write:

Many scientists accept computation (for example, large-scale simulation) as the third branch [of science—alongside deductive and empirical

branches]...However, it does not yet deserve elevation to third-branch status because current computational science practice doesn't generate routinely verifiable knowledge. Before scientific computation can be accorded the status it aspires to, it must be practiced in a way that accepts the ubiquity of error, and work then to identify and root out error. (pp. 8-9).

Many activities may be involved in the quest to identify and root out errors in artifacts of scientific processes. For example, to help root out errors in deductive science and mathematics the resulting artifacts (for example, equations) are subjected to peer review. Similarly, computational artifacts should be scrutinized. However, just as artifacts of deductive science cannot be reviewed in the same way as artifacts of empirical science (such as physical measurements), reviews of computational artifacts must be carried out in a way uniquely suited to the principal artifact of the computational process, program code. In this chapter we will focus on two approaches to the review of program code: *code testing* and *technical review*. Both of these approaches will be grouped under the umbrella term *code scrutinization*.

Some topics are not addressed in this chapter. Firstly, we do not discuss the validation of the scientific models that underlie scientific programs. Although it is critical that scientific programs be built from appropriate scientific models, it is also critical that models are realized in code reasonably and accurately. Scientists are experts at evaluating scientific models, but they are not necessarily experts at evaluating codes that realize these models. In our research (Sanders and Kelly, 2009) and work experiences, we have found that strong model validation practices are often not matched by strong code scrutinization practices. For that reason, this chapter avoids discussions of model validation and devotes itself to code scrutinization.

Secondly, we do not discuss numerical methods. Selection of numerical methods, solution techniques, and algorithms can have a strong influence on the accuracy of a program, but it is not our aim to instruct the reader on how to choose appropriate algorithms. Numerous introductory and advanced textbooks already offer good coverage of the topic. However, we encourage strong code scrutinization practices to help scientists discover excessive inaccuracies resulting from weak algorithms.

Thirdly, we do not discuss the testing of routines that interact with the world outside the program. Instead, we focus primarily on the testing of computational engines.

A Note on Terminology

In the remainder of this chapter, when we use the word *error* we mean the quantitative difference between a measured or calculated value of a quantity and what is considered to be its actual value. To indicate a code mistake we will use the word *fault*. Note, therefore, that a fault is not an error, but a fault can lead to an error.

DESCRIPTION OF A TESTING FRAMEWORK

In general, testing is an investigative activity done to improve knowledge about the state of the software under test. Each test is an experimental trial of the software. Tests contribute empirical data required to answer questions about the software. A testing effort will have knowledge goals that tests should fulfill when taken in aggregate.

We describe a testing framework that allows the scientist to better understand how to match their situation to a testing approach. It requires defining the *context* of the testing effort by gathering the right information and asking the right questions, establishing achievable *goals*, designing a set of tests that can achieve the goals using appropriate testing *techniques*, and using the data from test executions to decide whether testing *adequately* achieves the knowledge goals.