BESSIG Meeting Wed, Sep 18, 4 - 6 PM

> *"Code without tests is bad code. It doesn't matter how well written it is; it doesn't matter how pretty or object-oriented or well-encapsulated it is. With tests, we can change the behavior of our code quickly and verifiably. Without them, we really don't know if our code is getting better or worse." [FEATHERS]*

A strong statement, but it does bring home the vital role of testing in software development. Join us at the Boulder Outlook Hotel for:

Strategies, motivations, and influencing adoption of testing for scientific code

Ian Truslove, Erik Jasiak, NSIDC

Computation and programming are increasingly inescapable in modern Earth Sciences, but scientists and researchers receive little or no formal software engineering or programming training. At the same time, research into the reproducibility of other academic papers exposing disappointingly low rates of repeatability and high-profile retractions due to computational or data errors increase the onus on researchers to write repeatable, reliable, even reusable programs; in other words, "write better code".

Software engineering has plenty to say on the matter of "better code": metrics, methodologies, processes, tools... Of course, none are indisputable and none provide absolute guarantees. One seemingly obvious technique - testing - has enjoyed a renaissance in incarnations such as unit testing, and with approaches such as test-driven development (TDD) and behavior-driven development (BDD).

Based on our experience at the National Snow and Ice Data Center (NSIDC) with unit testing, TDD and BDD, we present a set of recommendations to scientific and research programmers about some techniques to try in their day to day programming, and possibly provide some inspiration to aim for more comprehensive approaches such as BDD. We will highlight some use cases of various types of testing at the NSIDC, discuss some of the cultural and management changes that occurred for programmers, scientists and project managers to consider and adopt processes such as TDD, make recommendations about how to introduce or expand rigorous code testing practices in your organization, and discuss the likely benefits in doing so.

[Scroll down to see post presentation references and material.]

Schedule

4:00 - 5:00 Presentation

5:00 - 6:00 Social

All are welcome.

Post Presentation References and Material

Presentation slides

Wilson et al, "Best Practices for Scientific Computing", highly recommended!

Merali, "Computational science: ... Error" in Nature. "As a general rule, researchers do not test or document their programs rigorously, and they rarely release their codes, making it almost impossible to reproduce and verity published results generated by scientific software, say computer scientists."

Good books for unit testing, TDD, and higher-level tests

Freeman and Pryce, Growing Object-Oriented Software, Guided by Tests

Beck, Test Driven Development: By Example

Fowler, Mocks Aren't Stubs - Martin Fowler on the terminology and usage of mocks, stubs, test doubles - all those "fake collaborators"

A couple of Bob Martin's books are particularly noteworthy for covering lots and lots of desirable attributes for code

Martin, Agile Software Development, Principles, Patterns, and Practices

Martin, Clean Code: A Handbook of Agile Software Craftsmanship

Some test frameworks

JUnit - the original (Java)

RSpec (Ruby)

Behave (Python)

Jasmine (JavaScript)

mgunit (IDL)

pfUnit (FORTRAN)

Cucumber (Acceptance tests, lots of languages)

Also

Feathers, Michael C., Working Effectively with Legacy Code , Prentice Hall, 2005, p. xvi.

Snowden, Cynefin: Wikipedia on Cynefin, David Snowden introducing Cynefin (video) - applicable to knowledge management, cultural change, and community dynamics, and has also involved issues of organizational strategy.

Snowden, Boone, 2007 "A Leader's Framework for Decision Making" (must pay for access from Harvard Business Review, though perhaps available elsewhere)