Tools, Trends and Techniques for Developing Scientific Software

Tom Clune - NASA GSFC
Brice Womack - NASA/NGC-TASC
Brian Foote - The Refactory, Inc.
Jeffrey Overbey - University of Illinois

ECMWF - Nov. 2, 2006

ASTG

- Advanced Software Technology Group
  - Part of the Software Integration and Visualization Office (SIVO) within Earth Science Division at NASA Goddard Space Flight Center.
  - *Not* formally part of NASA HPC computing.
  - Assists NASA scientists is development, optimization, and porting scientific models - primarily climate/weather and atmospheric chemistry.
  - Primary clients include
    - Global Modeling and Assimilation Office - **GEOS-5**
    - Goddard Institute for Space Studies - **modelE**
    - Various atmospheric chemistry groups
- How can the ASTG most effectively aid such a wide variety of research teams/codes?
  - Interesting constraint: In most instances, ASTG does not own/control source code.

ASTG's Support Activities

- ASTG is assisting modelers in modifying software in a variety of manners:
  - Parallelization
  - Componentization (migration to ESMF)
  - Adopting new computational grids (cubed-sphere)
  - Exploring new/exotic architectures
    - Blue Gene/L ?
    - Cell processor ??
    - Field Programmable Gate Arrays ???
- Common theme - potentially require large, pervasive modifications throughout source code.
  - However - answers must not change
  - Legacy code is often difficult to modify without introducing unintended errors.

Accruing Code Debt

- Expediency often conflicts with long-term software development/maintenance issues.
  - “This is a temporary kludge …”
  - “We’ll use x set to -9999. to signal …”
  - “We’ll just add another argument to the routine …”
  - “We’ll just cut-and-paste the loop from over there …”
- Scientific programmers are so accustomed to many bad programming practices that we often forget why the practices are bad!
- “Code Debt” is an apt metaphor
  - Accrues interest - cost per change increases
  - Never goes away on its own
  - Can grow to unmanageable size
- Code debt can seriously frustrate attempts to introduce significant new capabilities in a legacy system.
- Worth noting - code debt also increases startup costs for new developers.

Getting Out of Code Debt?

- Refactoring: *intentionally* modifying code so as to improve *quality* without modifying behavior
  - Common examples:
    - Breaking large routine into smaller, more manageable routines.
    - Replacing “magic” numbers with named constants.
    - Replacing common snippets of code with procedure call
    - Changing local variable into dummy argument.
- The challenge is to reduce the cost and risk of refactoring such that developers can and will refactor on a regular basis.
  - Risks - unintentionally altering behavior
  - Costs - changes often involve deeply rooted constructs throughout code.

Trends in IT industry

- Agile software processes
  - Short development cycles (1-2 week iterations)
  - Adapts to customer's changing requirements
  - Test Driven Development(TDD)
    - Implement, build and verify cycle
  - Relies on new generation of tools to allow/encourage fast, repeatable testing of code

Test harness

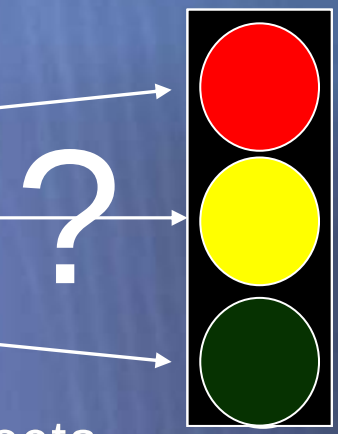
- A test harness is a system which verifies (tests) some aspects of behavior for an existing system.
  - Used to identify unintended changes in behavior.
    - Discover it *now*, not later!
  - Improves developer confidence - compare to a net used to catch trapeze artists when they practice new stunts.

Software Time Scale

- Both development and maintenance can be characterized by the above cycle.
- Long cycles are undesirable
  - Developers tend to make many/larger changes before verification.
  - More difficult to isolate cause of defects.
  - Bugs are discovered when developer's memory is stale.
- Many teams work with cycle times in hours, days, or even weeks.
  - Common practice for agile software developers are in the 1-10 minute range.

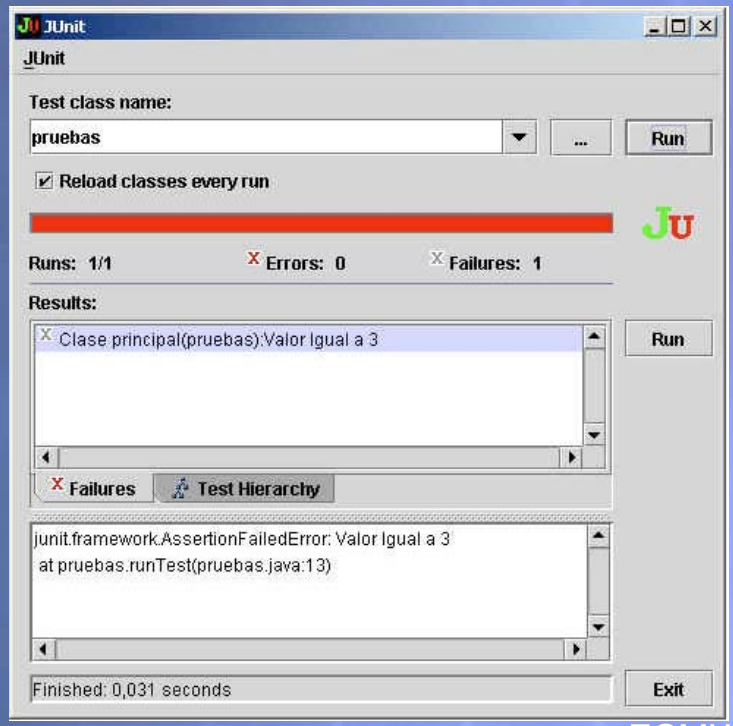
Test Driven Development

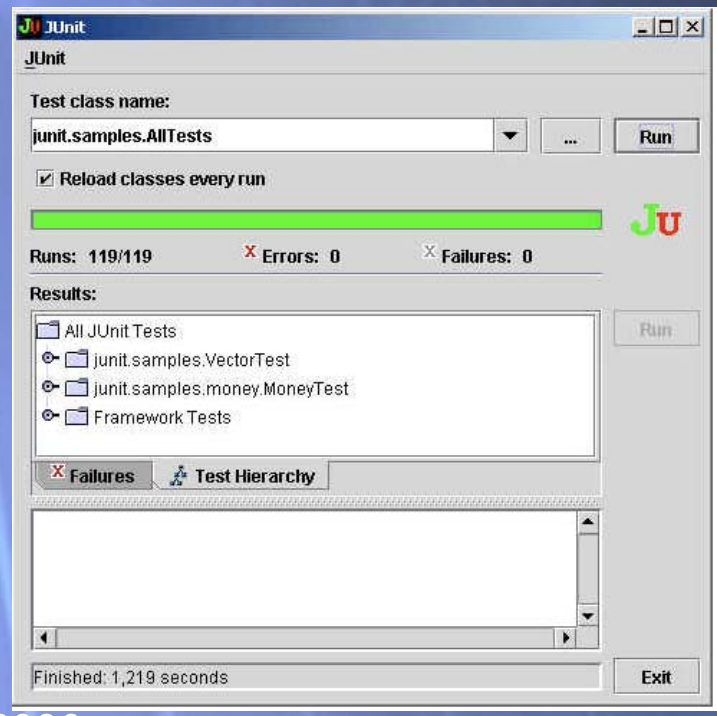
- TDD - very fast cycle
  - Write test for new behavior
  - Write code to pass test
  - Remove redundancy
- Advantages
  - Early detection and fast isolation of defects
  - Reduced development and maintenance costs
  - Large degree of confidence!
  - Always ready for release.
  - ***Better design!?***
- Costs
  - 2-3x more lines of source code
  - Requires discipline and adequate support from tools.
- How applicable is this to numerical routines?



Testing Frameworks

- Enables developer to easily create, group, and execute collection of tests.
  - Support for many languages: JUnit, cppUnit, pyUnit, …
- New psychology of development: “Green Bar” addiction





Refactoring Tools

- Provide high-level, *semantic*, and *safe* transformations of source code
  - Examples include
    - Rename
    - Extract method
  - Best if used in conjunction with testing harness.
- Combining with TDD total development time is significantly reduced for legacy applications:
  - Fewer development cycles due to larger changes within a cycle with minimal risk.
  - Faster, reliable cycles from TDD☐
- Eclipse - popular open source IDE
  - Provides powerful refactorings for JAVA and C++

Tools for Fortran

- Testing frameworks
  - More difficult to develop in Fortran due to relatively limited abstraction capabilities.
  - Nonetheless several have been developed:
    - pFUnit - full featured, includes support for MPI
    - Funit - full featured, built using Ruby
    - FRUIT - Limited features, primarily an Assert package
  - These Fortran testing frameworks are well suited for many development efforts, but provide no effective capabilities for dealing with most legacy software.☐☐

pfUnit - Parallel Fortran Testing Framework

- Developed at NASA GSFC (Clune & Womack)
  - Used internally by ASTG for several projects
  - Recently released under NASA open source license
    - http://sourceforge.net/projects/pfunit
    - User documentation and useful examples are still being created.
- Bootstrap development via TDD
  - Bundled with self tests.
  - F95 based implementation
  - Augmented with minimal amount of C
- Supports HPC unique test cases
  - Parallel MPI unit tests
  - Extensive Assert library for floating point
  - Parameterized unit tests

Other Tools for Fortran

- Photran - Refactoring tool for Fortran
  - Provided as plug-in for Eclipse
    - Integrated with CVS
    - View outline of file
    - Jump to source line for error
  - Refactoring capabilities under development
    - Rename: “a2s” -> “convertToString”
    - Extract Subprogram: replace section of code with call to new subroutine. Tool prompts user for information - dummy args, routine name, etc.
  - Future refactorings - which should be highest priority?
    - Replace Common, Add Argument, Move Subprogram, Remove Continue, …
    - Each new refactoring requires nontrivial development effort to make automatic and robust. I.e. need more funding.

Legacy Software?

- By themselves, testing frameworks are inadequate for applying TDD to legacy software.
  - Difficult to bootstrap:
    - Need tests to make changes
    - Need changes to make testable
  - Quite often the only available tests are to check that original behavior is preserved.
- Typical fortran legacy applications involve additional difficulties:
  - Data not passed through formal interface:
    - Vars in common blocks, module variables
    - SAVE'd local variables
  - Conditional compilation and deeply nested conditional blocks limit test coverage.
  - Large routines (1000+ lines) are difficult beasts to engage: Where do you start?

FFTT

- Fast Fortran Transformation Toolkit
  - Toolkit to assist placing test harness around legacy code.
- General approach:
  - Provide methods to capture existing (empirical) behavior for legacy routines.
    - Store state of subsystem before and after procedure call.
  - Create tests based upon stored behavior and incorporate them into test suites.
  - Rely on OO capabilities in F2003 to maximize flexibility and power. (Will *not* require F2003 for actual user application!)
- Timeline for development:
  - Conceptual design is complete
  - Prototype/demonstration in legacy applications ~ April 2007.
    - (Will be developed using TDD methodology.)
  - Open-source release ~ October 2007.

Summary

- Important to remain aware of capabilities in general software development community.
  - Investments in scientific development are dwarfed by the investments made in other software areas.
- Opportunities for significant long-term productivity gains will be missed unless appropriate investment are made in tools/training for developers of technical software.
- Some minimal capabilities are on the near-term horizon.

References

- Testing Frameworks:
  - JUnit - Erich Gamma and Kent Beck
    - http://www.junit.org/index.htm
  - pFUnit - Tom Clune and Brice Womack
    - http://opensource.gsfc.nasa.gov/projects/funit/pfunit.php
    - http://sourceforge.net/projects/pfunit
  - Funit - Bil Kleb et al
    - http://funit.rubyforge.org
  - FRUIT - Andrew Chen
    - http://sourceforge.net/projects/fortranxunit
- IDE's
  - Eclipse - IBM et al
    - http://www.eclipse.org
  - Photran - Jeffrey Overbey et al
    - http://www.eclipse.org/photran
- Books
  - Test Driven Development - by Example - Kent Beck
  - Refactoring - Martin Fowler