Testing in Analysis (using GoogleTest)

Nils Krumnack (Iowa State University)

Introduction

- have been playing around with GoogleTest since last TIM
  - updated some existing tests to GoogleTest
  - wrote some new tests for existing (un-tested) code
  - for new code wrote unit tests at the same time (where possible)
- only did tests for my own code so far

- overall impression positive:
  - generally easy to use
  - lots of integrated features and tests
  - used boost unit tests a while back, this seems at least as good

- though some caveats:
  - some code isn't written unit-test friendly
  - some of our use cases seem missing
  - need more support infrastructure

Reminder: GoogleTest

- each test file contains a series of tests
  - generally break down in setup and checks
- the checks should be minimal and focused (ideally one per test)
  - allows to see easily what fails
- can group common setup code in fixtures
  - shared and/or re-instantiated per test
- can pass parameters into fixtures (template or value)
  - all tests get executed for each value of the parameter
  - check multiple implementation to adhere to common interface
  - check implementation to work for multiple parameter values

- convenience features for running (selection):
  - (temporarily) disable tests by pre-fixing name with DISABLED_
  - run only select tests (for debugging)
  - re-run test N times (test for spurious failures & resource leaks)
  - missing: start process for each test (useful for crashes)

EventLoop tests

- reminder: EventLoop does non-Athena job management
  - uses drivers to implement different locations (local, batch, grid…)
- testing EventLoop itself using GoogleTest
  - replaces some older, murkier tests
- not your typical unit test:
  - needs to process a larger input dataset
  - large overhead for starting batch jobs (minimize this)
  - probably indicative of tests with large input datasets

- implemented tests using parametric fixture:
  - passing in the driver as parameter
  - on first test: generates input datasets and runs jobs
  - separate (small) tests check job outputs
- roughly matches structure of old tests:
  - generally a functional structure
  - GoogleTest provides easier checks and better accounting

EventLoop problems

- will always run all jobs on first test:
  - includes one job on large dataset to test recursive hadd, etc.
  - slow and unnecessary when debugging other tests
- could do one fixture per dataset, but:
  - requires each test file to instantiate multiple fixtures
  - doesn't work for enabling/disabling individual algorithms
- ideally have multi-step process:
  - determine tests to run
  - run only what I need for those tests
  - run the actual tests
- also: rerunning tests via GoogleTest doesn't re-run batch jobs

- can't disable single test for single driver:
  - can disable all tests for single driver
  - can disable single test for all drivers
  - problem for all parametric tests

GoogleMock

- haven't used GoogleMock in ATLAS code, but private code (so far)

- e.g. used GoogleMock to mock a random number generator
  - allows to set the exact sequence of random numbers
  - very easily explore every code path
  - get predictable outputs

- extends ability to check inputs/outputs of code:
  - control information passed in/out through function calls
  - control number/sequence of calls to external objects

- improves testability of code:
  - can test with inputs you want, instead of the inputs you get
  - can test unusual/rare inputs (or input sequences)
  - can pick more "readable" numbers, e.g. 2 instead of 1.0375…

GoogleMock & ATLAS

- GoogleMock should lend itself to our component model:
  - could use mock versions of tools and services
- didn't use it in my tests so far:
  - don't use that many components to begin with
  - for the cases I had a custom mock object seemed better

- possible issue: mock objects designed to live on the stack
  - can easily assign them to a ToolHandle in RootCore
  - Athena simply doesn't allow that
  - run such tests RootCore-only?

- sometimes custom mock objects are better:
  - easier for complex behavior and analysis (reused across tests)
  - easier to work with cloning, streaming, etc.

Algorithm Testing

- multiple algorithm types in analysis realm:
  - EventLoop, QuickAna, Gaudi/Athena, various analysis frameworks
- algorithms can be tricky to test well:
  - interface designed around when and how they are called
  - algs (can) expect a standard environment setup
  - algs (can) access a number of services
  - algs may need to fulfill various guarantees
- stayed away from algorithm testing so far

- one workaround: move alg code into functions/tools
  - for complex/long algs that may be better anyways
  - seems overkill for simple algorithms
  - also: leaves some "glue" code untested
- better: provide algorithm testing setup
  - need to revisit/redesign EL/QA algs anyways
  - hope to address testability at the same time

misc. items

- don't have a uniform naming and calling convention
  - at least not implemented in RootCore & cmt (& cmake)

- no test meta-information, e.g.:
  - human readable description
    - would help reading the log file
    - currently just a comment in the test source file
  - test dependency information,
    i.e. say that test X will likely fail if test Y failed
    - useful if you have ~hundred tests fail

- how to check error conditions properly?
  - normally print error message and return FAILURE
  - ideally had a way to test messages generated
    - testing messages could also be useful for other things
  - or return failure modes differently?

misc. items II

- how to handle tests for specific build configurations?
  - e.g. testing checks disabled with NDEBUG
  - disable? remove? remove content? don't test?

- can't nest parametric tests
  - i.e. can't make a matrix of test for two parameters
  - useful in parametric tests to check adherence to an interface
    - one parameter from test file, other from the library

- some tests create temporary files
  - RootCore/cmt start them in separate directories for that
  - when running manually that doesn't happen
    - easily clobbers directories and breaks (some) tests
  - doesn't play nice with test auto-repeats
  - could use a helper function that makes unique sub-directories per test