The LArSoft code analysis process

Erica Snider

Fermilab

June 23, 2016
General goals of the process

• Writing good code is a process.
  - Can't just sit down and “bang it out”

• With analyses, hope to improve quality and usability of code
  - Usability
    • Repeated design / usage patterns following from adherence to common principles
      - Easier to learn and use
    • Create more of a toolkit through improved design
    • Allow more sophistication through layering of algorithms
  - Quality
    • The code does what it was intended to do (from the coding point of view)
    • Good computing resource utilization
    • Maintainable: more modular, easier to test, internally well documented, ...
      Particularly important – experiments long lived. Support needs to transition
  - Include full scope of code: algorithm implementations, infrastructure
General goals of the process

• Writing good code is a process.
  - Can't just sit down and “bang it out”

• With analyses, hope to improve quality and usability of code
  - Usability
    • Repeated design / usage patterns following from adherence to common principles
      - Easier to learn and use
    • Create more of a toolkit through improved design
    • Allow more sophistication through layering of algorithms
  - Quality
    • The code does what it was intended to do (from the coding point of view)
    • Good computing resource utilization
    • Maintainable: more modular, easier to test, internally well documented, ...

Particularly important – experiments long lived. Support needs to transition

- Include full scope of code: algorithm implementations, infrastructure
General goals of the process

- Writing good code is a process.
  - Can't just sit down and “bang it out”

- With analyses, hope to improve quality and usability of code

  - Repeated design / usage patterns following from adherence to common principles
    - Easier to learn and use
  - Create more of a toolkit through improved design
  - Allow more sophistication through layering of algorithms

- Quality
  - The code does what it was intended to do (from the coding point of view)
  - Good computing resource utilization
  - Maintainable: more modular, easier to test, internally well documented, ...
    - Particularly important - experiments long lived. Support needs to transition
  - Include full scope of code: algorithm implementations, infrastructure

Code analysis is a strategic initiative strongly backed by SCD

Expect strong technical support to be available for an on-going program of LArSoft code analyses
General attributes / guidelines

- The analysis procedure shall:
  - Not be overly prescribed
  - Be as light-weight as possible for the situation
  - Be performed collaboratively with the code author(s)
  - Have clear objectives in each instance
  - Have adequate time and effort available for the review
  - Have adequate effort allocated in advance to implement recommendations
  - Have a written report to the authors, requesters, experiment offline coordinators, LArSoft community

Want analyses to be manifestly useful to authors/experiments.

  We should all want to have our code reviewed/analyzed!
General attributes / guidelines

- The analysis procedure shall:
  - Not be overly prescribed
  - Be as light-weight as possible for the situation
  - Be performed collaboratively with the code author(s)
  - Have clear objectives in each instance
  - Have adequate time and effort available for the review
  - Have adequate effort allocated in advance to implement recommendations
  - Have a written report to the authors, requesters, experiment offline offline coordinators, LArSoft community

Want analyses to be manifestly useful to authors/experiments.

We should all want to have our code reviewed/analyzed!
Specific goals of each analysis

- Ensure compliance with *art* / LArSoft design principles
  - The common principles underlying the design
- Evaluate / improve code performance
  - CPU and memory
- Ensure the use of best practices in coding
  - Typically aimed at either performance or maintainability
  - The context of *art* / LArSoft is an important component of this
- Evaluate and improve high and low-level architecture
  - Class structure, data product design, interface design...
- No change to the physics output
  - NOT aimed at physics performance. Just make it do what it does now better
How the procedure was developed

• What we did
  - Created a committee of experts, people with experience with code reviews
  - Designed a set of guidelines based on that experience
  - Ran a trial review, keeping careful notes of issues related to the process
  - Wrote recommendations for the process

• The committee
  - Software engineering experts: Chris Jones, Jim Kowalkowski, Marc Paterno
  - LArSoft domain experts: Gianluca Petrillo, Erica Snider
  - Outside domain expert: Rob Kutschke
  - Analysis trial code authors: Dorota Stefan, Robert Sulej
Participants / roles

- Each review will involve the following parties
  - Code authors
  - Two software engineering experts
  - An additional problem domain expert
    - e.g., someone from LArSoft team, SCD reconstruction group, additional experiment members
  - Additional experts considered to be necessary or useful
  - Also, observers with no official role are welcome

- The following roles will be assigned
  - Review leader, who will facilitate the process
    - Should be one of the experts
  - Scribe, who will record important conclusions or points of discussion
    - Should not be the review leader
Conduct of the analysis

Each analysis has five basic steps:

1) Initiating the review
2) Preparatory work
3) Review meeting(s)
4) Review report
5) Follow-up work
Conduct of the analysis

1) Initiating the review

2) Preparatory work

3) Review meeting(s)

4) Review report

5) Follow-up work

Initiating an analysis

- Basically anyone can request a review.
  - The code authors
  - An experiment representative
  - Core LArSoft team
  - The result of some future policy (e.g., for each “pull request” equivalent)

- Several types of analyses are possible
  - General design, class structure, interfaces
  - Computing performance, resource bottlenecks
  - Compliance with coding or C++ practices

- At the time of the request
  - Agree on type and scope of the review, the charge to the analysis team, objectives and any metrics needed to assess “success”
Conduct of the analysis

1) Initiating the review

2) Preparatory work

- Prerequisites
  - Assemble analysis team
  - Agreements with people who will implement the agreed upon changes

- To make meetings as productive as possible
  - Common work areas for building, profiling
  - Obtain test fcl and input data
  - A set of tests to validate changes
  - Gather viewing aids: diagrams, repository clones
  - Generate reports from static analysis tool
  - Generate preliminary time and memory profiles
  - Preview of the code by the analysis team

3) Review meeting(s)

4) Review report

5) Follow-up work
Review meetings

- No pre-defined structure or format
- The following considered useful
  - Agreement on how to allocate time
  - Overview presentation or discussion by authors
    - Introduction to major concepts and abstractions
  - Discussion of targets of opportunity
    - e.g., “While we were in there, we noticed that 90% of the time was spent here...”
- Sessions should last about four hours
  - Number of sessions depends on the analysis
Conduct of the analysis

Review report
- Should include
  - Recommended
    - changes
    - items for more investigation, thought, discussion
    - conclusions regarding targets of opportunity
  - Additional items worthy of note
  - Before / after metrics, when they exist
  - Comments on the process
- Should be co-authored by all involved
  • Need only be long enough to specify problems and solutions

1) Initiating the review
2) Preparatory work
3) Review meeting(s)
4) Review report
5) Follow-up work
Follow-up work

- Scope defined by code authors / experiments in consultation with Core LArSoft team
  - Create prioritized task list
- To facilitate implementation
  - Define milestones in LArSoft issue tracker that describe overall targets
  - Enter work tasks into LArSoft issue tracker
    - Each task should be under a milestone
    - Experiments may choose to track tasks also
  - Reports on the progress toward these milestones
    - e.g., at experiment / LArSoft meetings
- Document lessons learned
  - Make this list available to LArSoft community
  - A brief report presented when of broad interest
A recent example: the PMA analysis

- Analyzed Pattern Matching Algorithm code
  - Authors Dorota Stefan and Robert Sulej
  - Analysis team: Chris Jones, Jim Kowalkowski, Rob Kutschke, Marc Paterno, Gianluca Petrillo, Erica Snider

- Focused on
  - High-level design
  - Computing performance (authors priority)
  - Low-level coding practices

and

- the analysis process
A recent example: the PMA analysis

• About a week in advance
  - Prepared shared repositories on GitHub
  - The team read and commented on the code
  - Identified testing fcl and data

• The meetings
  - Three hours on one day, two hours the next
    • Each meeting focused more narrowly on particular topics
  - Ran memory and CPU profiling using `igprof` (+ `valgrind`, but was less useful)
  - Quickly identified performance issue with use of `TVector` in a tight loop
    • Change resulted in factor of 2 in CPU speed (profiling reported 10—20%)
    • A second suggestion led to another 30% improvement, for a total factor of 3
  - Identified various possible structural and low-level improvements
A recent example: the PMA analysis

- The report
  - Everyone met a third time for about one hour to write the report
  - Recommended changes, further consideration in five areas
    - Design / architecture
    - LArSoft coding guidelines
    - art coding guidelines
    - C++ coding practices
    - Code management
A recent example: the PMA analysis

- Created first “lessons learned” list

From the report

- Keeping transient data as module state between events can significantly and unnecessarily increase the memory footprint of modules.
- TObject memory and CPU overheads can have a significant impact on overall performance in some cases. Care should be exercised when choosing to use TObjects sub-classes.
- IgProf was much more convenient than Callgrind for profiling work.
- Cloning the repository and using local tools was faster for looking at large amounts of code.
- The GitHub repository was very convenient for commenting.
  - For structural reviews, the GitHub commenting was not very useful.
  - For code conformance and best practices comments, the GitHub commenting was very useful.
- The search facilities in git itself (e.g. git grep) are useful in looking at code. The git history facilities are also useful.
The authors' description of the experience

Summary

- instructive and for sure helpful for the further PMA development; a lot of collaborative work with experts
- otherwise never have time to look at our own code
- reasonable amount of recommendations, should be feasible to implement in parallel with other developments

- as of today: code x3 faster

- such reviews are required by DUNE!
- LArTPC reconstruction is still huge R&D effort, but it looks like we are moving towards well organised, high-quality software culture
Important tools for analyses

- Collaborative
  - GitHub.com
    - Used for annotating code, chats between analysis team members

- Performance profilers
  - IgProf ([http://igprof.org](http://igprof.org))
  - Valgrind ([http://valgrind.org](http://valgrind.org))

- Static code analyzers
  - Clang static analyzer ([http:clang-analyzer.llvm.org](http:clang-analyzer.llvm.org))
    - Used by CMS, works at Fermilab

- Class structure diagramming
  - No general tools available that works well
    (Just used, so just used what team was familiar with)
Summary / conclusions

- LArSoft hopes to create a culture that seeks code analysis
  - Assist non-expert code authors in writing expertly crafted code
  - More time to think about physics

- Had good experience with the first analysis
  - Tangible improvements: 3x faster
  - Design improvements: create independent algorithmic components
  - Authors are happy!
  - Experiment, LArSoft team, computing providers all happy at the outcome

- Will take the next steps here, today
  - Thank you Mike Wallbank (EM shower reconstruction) and Bruce Baller (clustering) for volunteering their code!!
The end