Balancing agility and gate-keeping

*The LArSoft development model*

Erica Snider

*Fermilab*

June 22, 2016
The tension in code development

- Two competing desires common in code development
  - Ability to introduce new ideas, new features quickly
    - Minimal time spent in design
    - Low barriers to writing code, getting into releases
    - Just make it work!
  - Writing production quality code
    - Well tested
    - Well thought-out, integrated design
    - Usually considerable re-writing en route

Like many software projects, LArSoft benefits from both attributes in ample quantities
The tension in code development

- Two competing desires common in code development
  - Ability to introduce new ideas, new features
    - Low barriers to getting code into releases
    - Minimal time spent in design
    - Just make it work!
  - Writing production quality code
    - Well tested
    - Well thought-out, integrated design
    - Usually considerable re-writing enroute

Both important to 'usability'

Users depend on 'production quality'

Developers on 'agility'

*But also 'production quality' as a foundation for development*

So need to examine the LArSoft development model

Like many software projects, LArSoft benefits from both attributes in ample quantities
The LArSoft development model

- Documented on the LArSoft wiki:

- The steps
  - Designing
    - Decide how to structure the solution
  - Writing code
    - Standards!!
  - Building
    - mrb, cmake, maybe cetbuildtools and its configuration for art, LArSoft
  - Testing
  - Documenting
  - Getting code into a LArSoft release
    - Must follow a procedure
The LArSoft development model

For our purposes today, only care about these

- Documented on the LArSoft wiki:

- The steps
  - Designing
    - Decide how to structure the solution
  - Writing code
    - Standards!!
  - Building
    - mrb, cmake, maybe cetbuildtools and its configuration for art, LArSoft
  - Testing
  - Documenting
  - Getting code into a LArSoft release
    - Must follow a procedure
Contributing code

• How do we go about contributing code to LArSoft?
  - Create a branch in some repository
  - Create / modify and test code
  - Next steps depend on the type of change
    • Changes that do not affect behavior
      - Just merge into develop
    • New code or features that do not change dependencies, bug fixes
      - Merge into develop
      - Discuss at LArSoft Coordination Meeting to make people aware of the change
    • New code that introduces new dependencies, that breaks existing code or data, that alters behavior
      - Discuss at LArSoft Coordination Meeting
      - Upon approval, LArSoft team merges into develop during release creation procedure
  - Weekly integration releases to incorporate changes
Contributing code

• How do we go about contributing code to LArSoft?
  - Create a branch in some repository
  - Create / modify and test code
  - Next steps depend on the type of change
    • Changes that do not affect behavior
      - Just merge into develop
    • New code or features that do not change dependencies, bug fixes
      - Merge into develop
      - Discuss at LArSoft Coordination Meeting to make people aware of the change
    • New code that introduces new dependencies, that breaks existing code or data, that alters behavior
      - Discuss at LArSoft Coordination Meeting
      - Upon approval, LArSoft team merges into develop during release creation procedure
  - Weekly integration releases to incorporate changes

In most cases, a discussion of some sort at the LArSoft CM is required

Prior to integrating code, additional work may be requested.
What you need to know first

- Policies, guidelines and standards at all levels
  - LArSoft design principles

LArSoft concepts (on larsoft.org)
Design principles (on larsoft.org)
Architecture document
What you need to know first

- Policies, guidelines and standards at all levels
  - LArSoft design principles
  - Coding guidelines

LArSoft
- Coding guidelines and conventions
- Guidelines on writing/using services
- Guidelines on writing/using algorithms

art
- art module design guide
- Data product design guide
- Guidelines for the use of pointers

C++
- Lots of online resources...
What you need to know first

- Policies, guidelines and standards at all levels
  - LArSoft design principles
  - Coding guidelines
  - Git branching model
What you need to know first

- Policies, guidelines and standards at all levels
  - LArSoft design principles
  - Coding guidelines
  - Git branching model
  - Documentation guidelines

- The context for your code
  - Organization of the code
What you need to know first

- Policies, guidelines and standards at all levels
  - LArSoft design principles
  - Coding guidelines
  - Git branching model
  - Documentation guidelines
- The context for your code
  - Organization of the code
  - art API

Also art.fnal.gov
What you need to know first

- Policies, guidelines and standards at all levels
  - LArSoft design principles
  - Coding guidelines
  - Git branching model
  - Documentation guidelines
- The context for your code
  - Organization of the code
  - art API
  - Relevant existing code
What you need to know first

• Policies, guidelines and standards at all levels
  - LArSoft design principles
  - Coding guidelines
  - Git branching model
  - Documentation guidelines

• The context for your code
  - Organization of the code
  - art API
  - Relevant existing code

• Tools
  - mrb, cmake, cetbuildtools, debuggers, profilers, CI system...
The point of this model

• Focus is on
  - producing shareable, relatively uniform code
  - maintaining a stable development environment
    • This is balanced with recency (i.e., proximity to the head of develop branch)
  - finding consensus across experiments for changes
  - managing integration and deployment of changes

In short: on **gate-keeping**

• Good for users, but takes from agility, usability for developers
  - Steep learning curve
  - Multiple barriers to rapid development
  - Haven't even discussed other aspects of development environment, eg, built speed.
Finding the right balance

• Want to find the sweet spot for the general case
  - Enough agility to keep people interested in producing code
  - Enough gate-keeping to keep people using it

• Are we in the right spot?
  - Meetings?
    • Important to discuss changes before they happen
    • Also facilitate discussions about
      - improvements. other changes affecting lots of people
      - priorities, policies, procedures, etc.
      • But is the present requirement reasonable?
  - Standards and policies?
  - Would structuring releases differently help?
  - Would structuring code differently help?
  - Changes to development environment that might help?
Discuss amongst ourselves...
...then coffee break
The end