h1. Discussion of Oct 26

All tasks will be done by the integration project members, unless otherwise indicated.

artlite::Event

- C++ access
  - access data products, providing all parts of the identifier [10 days, done]
  - access data product using only module label + instance name [10 days]
  - making Ptrs readable (read and dereference) [10 days]
  - making FindOne and FindMany templates work [5 days]
  - handle reading a sequence of files [5 days]
- Python access (these time estimates are very uncertain)
  - make the Python access work, using the C++ underpinnings [5 days]

Nonfunctional requirements

- tests for all new code [included in estimates above]

Note: 'demonstrate' implies that the resulting code is not production quality.

Refactoring to minimize dependencies. Note we are not beholden to the names we have invented.

- demonstrate refactoring of lardata into 'larlitedata' and 'lardata':
  - clone the lardata repository [1 hr]
  - remove the data product classes that have dependencies on things other than nutools, art data products and ROOT. Note that nutools will be dealt with later. [3 days]
  - This includes building a larlitedata UPS product to be used in the demonstrators.
  - NOTE: Use this UPS product in demonstrators for use cases 1 and 2.
  - Produce the updated lardata and official larlitedata UPS products based on the refactoring above [LArSoft team, 5 days]
  - NOTE: From here on, we use 'larlitedata' to denote the result of this factorization
- reconcile the larlitedata and LArLite data product classes:

  - identify and remove from LArLite classes that are copies of those in larlitedata [LArLite team, 5 days]
  - identify classes in larlitedata and lardata and LArLite that provide the same functionality with different implementations. [LArLite team, 2 day]
  - reconcile the classes identified and put the implementations into larlitedata [LArLite and LArSoft teams, 10 days]
  - NOTE: when finished, LArLite will contain only classes not intended for LArSoft, and not replicating functionality of larlitedata.
  - NOTE: This is the time estimate of which we are least sure.

- perform refactoring of art into an 'art' and 'artlite' UPS products:
  - identify classes necessary to support artlite::Event (above) [2 days]
  - refactor directories and libraries within art to group the above classes [3 days]
  - NOTE: At the end of this task, art is still building one UPS product, but with more libraries.
  - refactor art to create two or more UPS products, following the grouping above [1 day]
  - write script(s) (for use by experiments) to adjust #include paths, to deal with the above refactoring [2 days]
  - make necessary adjustments in SSI build systems to deal with refactoring of art [art team, 1 day]
  - make necessary adjustments in delivery systems to deal with refactoring of art [art team, 2 day]
  - provide documentation to direct experiments on how to modify their build systems [1 day]
  - make necessary adjustments in experiment build systems to deal with refactoring of art [each experiment, 5 days]
  - NOTE: This time may vary widely between experiments

- demonstrate refactoring of nutools into 'nutools' and 'nulitedata' UPS products:
  - clone the nutools repository
  - remove the data product classes that have dependencies on things other than artlite and ROOT.
  - Use this modified repository in demonstrators for use cases 1 and 2.
  - nutools team will use this information to refactor nutools into two or more UPS products (nulitedata and nutools)
  - From here on, we use 'nulitedata' to denote the result of this factorization

- reconcile the lartlite, nulitedata, and larlitedata data product classes:
  - identify LArLite classes that depend only upon nulitedata, artlite and ROOT
  - move these classes from larlite to larlitedata
  - identify and remove from LArLite classes that are copies of those in nutools (note, not nulitedata)
  - identify classes in nulitedata and nutools and LArLite that provide the same functionality with different implementations.
  - reconcile the classes identified and put the implementations into nulitedata [LArLite and nutools developers]
  - when finished, LArLite will contain only classes not intended for LArSoft or nultools, and not replicating functionality of larlitedata or nulitedata
- reconcile the remainder of LArSoft with the previous refactorizations [LArSoft team, 5 days]

Demonstrator for Use Case 1

- create makefiles that build a newly-invented LArLite-using program, using an installed LArSoft version.
  - create a repository to be used as an example for others to follow
- write documentation for how to create makefiles that use LArSoft in this way
  - the documentation should live in a different repository from the example created above
  - documentation should also be printable
  - documentation must be tested by a LArLite user/developer
- develop scripts to generate makefile fragments that work this way
  - the scripts should live in the same repository as the documentation
  - scripts must be tested by a LArLite user/developer

Demonstrator for Use Case 2

- repository containing toy "user of LArLite" products and algorithms.
- fork of that repository, suitable for building UPS products from the appropriate code

- repository with wrapper code to build LArSoft modules using the forked repository.
- write documentation
  - documentation should live in the same repository as the use case 1 documentation
  - documentation must be tested by a LArLite user/developer on his or her own code
- if scripts are necessary for user support, develop them
  - any scripts should live in the same repository as the documentation
  - any scripts must be tested by a LArLite user/developer on his or her own code