Analysis Programming

Remarks

- Please feel free to
  - ask at any time,
  - give helpful comments.

- Be aware, that the difficulty level of the various topics is not entirely constant. If things are
  - too simple, perhaps you can help others,
  - too much at once, don’t be frustrated: It is not that important to understand everything.

Goals of Good Code

- **Correctness**
- **Reasonable Efficiency**

} @ ANY TIME

- Maintainability
- Readability
- Clarity

} **Prerequisite, because**

- **libraries, hardware, etc. may change;**
- **programming = reading code ~90% of the time;**
- **debugging is painful, less need, easier with clear code.**

- Reusability

This is mostly important for library developers.

Issues of this Course

- **Preparation, Organisation, Testing**
- General Best Practice Coding Rules
- Tools I
- STL Usage
- Design Patterns
- Paradigms
- Tools II
- Efficiency
- Templates

Preparation, Organisation, Testing

Coding Conventions

- Many developers have their own style of coding and their own naming conventions.
  BUT Big Benefits from using standards in a project, as they
  - make it easier to understand and use code,
  - allow the use of preprocessor macros.

  - Identifiers for multiple objects: **objects** OR **object_list** OR...
  - Function **doStuff()** OR **do_stuff()** OR … **do_Stuff()** …
  - Often classes are written in capital letters, functions etc. in small letters,...
  - Special indication of members, e.g. **m_object**.

  Think about this **before starting** a big project!

General Abstract Coding Conventions

- Keep it short & simple!
  - Is the functionality, that is used most of the time easy to use?
- Use informative names!
  - The names itself should be a kind of short comment
- Don’t make huge “God classes”, that try to make everything!
- Suppress excessive coupling &(!) Interfacing!
  - One important point: Avoid exposing data members of classes in all but the most simple structs.
- Avoid dead code, that is never reached.
- Avoid code duplication!
  - Even a single large calculation expression is better put into an intelligently named function, when this is used twice. Others then can be sure, that twice really the same kind of calculation is done.

Comments on Comments

- Some people don't like comments:
  - 'Code is the best form of documentation',
  - 'When code is changed, but comments not, they are more confusing than helping',
  - 'Comments are often just restating the obvious';
    - `int callCount = 0;  //Declares an integer variable`
- But you should agree on comments for additional info:
    - `int callCount = 0;  //holds number of calls to <function>`
  - Makes sense, especially in context of IDE, where you can see `callCount` definition without being at the place where it is defined.
    - `unsigned int findTime(unsigned int distance, unsigned int speed)`
  - Comment can tell you about units of I/O (but should probably be project specific convention),
  - limitations (max. distance),
  - reasons for I/O types (e.g. why unsigned, why int).

  Putting all this info in the name overloads it.

Ctd.

- You should still try to keep you comments short.
- You can as well use comments to explain
  - what the function should do in the future,
  - what improvements are possible,
  - why some design decision was taken over another appealing one,
  - ….

Of course this means, you have to change comments, when the code is changed!

→ Understand comments as part of the code; if comments aren't reflecting what is done, they are **dead** code.

Keep comments alive!

Comments for Web-Documentation

- Use of special comments structure to produce documentation e.g. for the web or as booklet...
  - As in the ROOT documentation... (not very nice)
  - Doxygen (recommended and very configurable) http://www.stack.nl/~dimitri/doxygen/

```
/** Getter for integrated charge.
 *
 *  Note: This is the integrated charge over the cell.
 *        In principle, this charge can come from more than just the
 *        track, this hit belongs to.
 */
unsigned short getADCCount() const {
  return m_adcCount;
}
```

Short description.

unsigned short **getADCCount** () const
Getter for integrated charge.

Long description.

unsigned short getADCCount ( ) const

Getter for integrated charge.

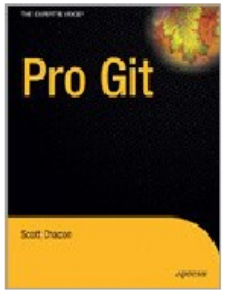
Note: This is the integrated charge over the cell. In principle, this charge can come from more than just the track, this hit belongs to.

Definition at line 131 of file CDCHit.h.

Version Control System

- ***svn*** seems now most popular in HEP, but I recommend the use of a 2-tiered system like ***git*** or ***hg:***
  - You can make **frequent local commits**, which is difficult with only the central repository, because
    - the central rep. should always compile,
    - often all commits notify conveners, etc.
    - sometimes you don't have connection, e.g. because you travel or the central server is down.
  - Developments, that turn out to be not useful don't need to be pushed up ever, so you can keep the central rep. cleaner.
  - Easy integration of local patches, that other people don't need.

Documentation



Pro Git
Scott Chacon

CC BY-NC-SA 3.0, http://git-scm.com/book

git ready

http://www.gitready.com
http://de.gitready.com

git cheat sheet

http://zrusin.blogspot.de/2007/09/git-cheat-sheet.html

Automatic Build and Test-Systems

- Your project should be built and tested all the time.
  - See immediately, when someone commits code, that breaks the build/tests.
    → notify immediately in case of failure.

- Support for
  - Building: Buildbot (buildbot.net)
  - Static Code analysis: cppcheck,...
  - Memory leaks etc.: Tools (hear later more about this) like valgrind
  - ….
  - **TESTING**: boost unit tests, google test,..., as well for test driven development (TDD as a paradigm, see later)

Example of Unit Test

```
TEST(ClusterCache, FindNeighbours)
{
  ClusterCache cache;
  ClusterCandidate cls;

  cache.setLast(2, 0, &cls);
  //Right pixel belongs to cluster
  EXPECT_EQ(&cls,                 cache.findCluster(3, 0));
  //Next to right is not clustered
  EXPECT_EQ((ClusterCandidate*)0, cache.findCluster(4, 0));
  //one down, two columns left is to far away
  EXPECT_EQ((ClusterCandidate*)0, cache.findCluster(0, 1));
  //We also want the bottom left pixel
  EXPECT_EQ(&cls,                 cache.findCluster(1, 1));
  //the bottom pixel
  EXPECT_EQ(&cls,                 cache.findCluster(2, 1));
  //and of course the bottom right.
  EXPECT_EQ(&cls,                 cache.findCluster(3, 1));
  //But everything farther away does not belong to the same
  //cluster
  EXPECT_EQ((ClusterCandidate*)0, cache.findCluster(4, 1));
  //Two rows down is also excluded
  EXPECT_EQ((ClusterCandidate*)0, cache.findCluster(0, 2));
  EXPECT_EQ((ClusterCandidate*)0, cache.findCluster(1, 2));
  EXPECT_EQ((ClusterCandidate*)0, cache.findCluster(2, 2));
  EXPECT_EQ((ClusterCandidate*)0, cache.findCluster(3, 2));
  EXPECT_EQ((ClusterCandidate*)0, cache.findCluster(4, 2));
}
```

There are other statements establishing <,>, and floating point compares, as well abort tests, etc.

```
Running main() from gtest_main.cc
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from ClusterCache
[ RUN      ] ClusterCache.FindNeighbours
[       OK ] ClusterCache.FindNeighbours (0 ms)
[ RUN      ] ClusterCache.SkipRow
[       OK ] ClusterCache.SkipRow (0 ms)
[ RUN      ] ClusterCache.Merging
[       OK ] ClusterCache.Merging (0 ms)
[----------] 3 tests from ClusterCache (0 ms tot

[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (0 ms
[  PASSED  ] 3 tests.
```

Automatic High-Level Testing

- Execute example user scripts, e.g.
  - comparisons of histograms of a reconstruction script with some expected result like DAQ control,
  - some fit result,...

  - check log text against expected result;

Code Reviews

- Finding bugs fast is imperative:
  - developer still familiar with the code producing the error;
  - same developer is still available;
  - other people haven't build their code on the wrong code;
    - e.g. for a long time in some lib there was a wrong sign in some octant of some trigonometric fuction;
      → users knowing this flipped the sign of that function in their code;
      → correction would break these users' code!

- Find errors fast by testing and by code reviews!

Code Reviews

- Code Reviews should make sure the code is
  [from Martin Schmetzstorff “Bessere Software Kompakt”]
  - **Correct;** e.g. does the `findTime` function return the correct time?
  - **Appropriate;** e.g. no tons of useless features!
  - **Complete;** e.g. the features that should be there are there.
  - **Comprehensible;** necessary for maintainability.

- Code Reviews need some exercise;
  - In the beginning often only focus on formal correctness etc. but over time go deeper (unclear comments, repetition, “shaming people into writing proper code”).
- Plan about 1 hour together with the developer for a ~500 line class.

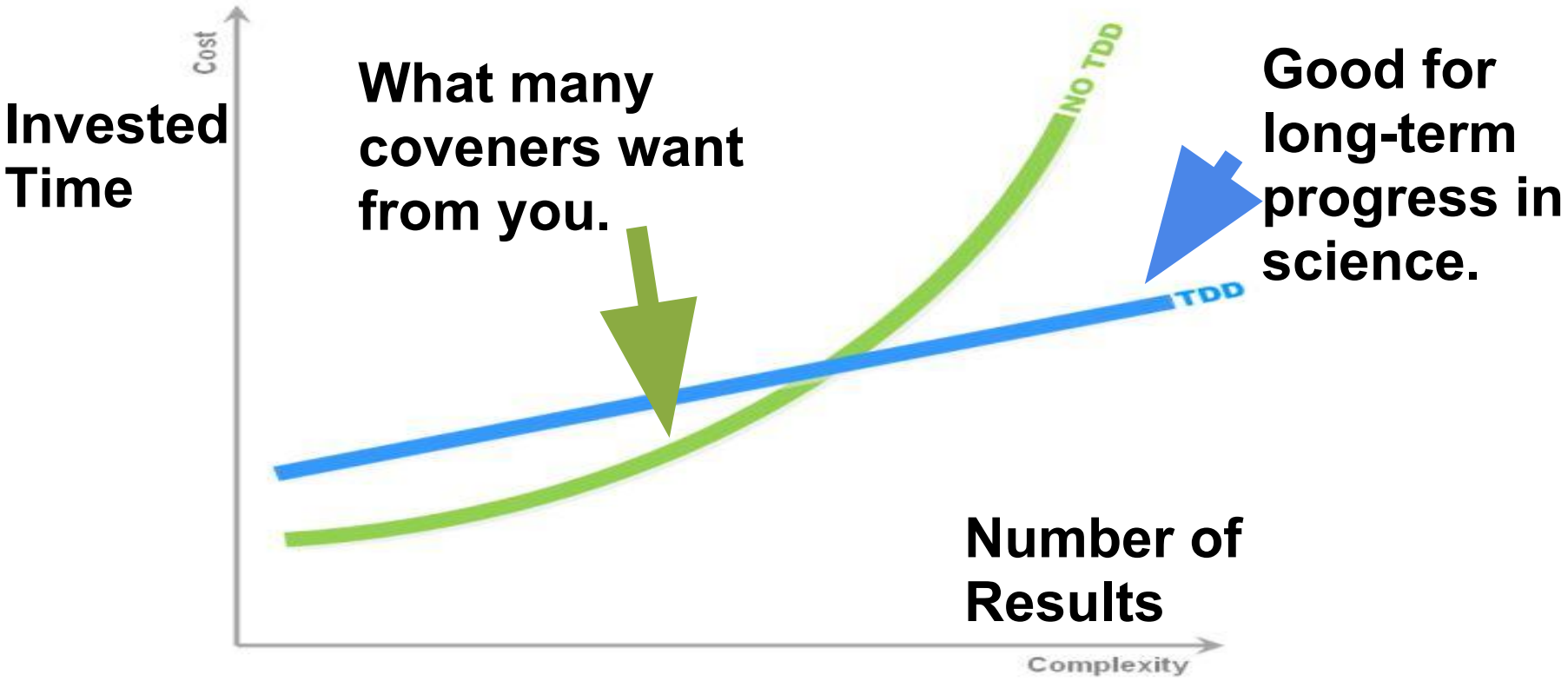
Invested Time Pays Off

- Industry experience with code reviews (again from BSK):
  - HP estimates an RoI of 10 for its “Inspection Program”;
  - Inspection of 2.5 million lines of real-time-code from Bell Northern Research saved an estimated 33 hours of maintenance per error;
  - IBM reports 1 hour of inspection saved an estimated 20 hours of testing and 82 hours of rework, if the product would have been delivered with the errors.
  - Imperial Chemical Industries had 1/10 the maintenance costs for group of programs, that were reviewed, compared with a group that wasn't.

- In science:
  - Wrong results, e.g. due to miscalculation of efficiency happened already...

More Pay-Off

- Code-Review has similar advantages as TDD:



Pair-programming and Collective Code Ownership

- As kind of a level-0 code review, you can do pair-programming;
  - Try it out, if it suits you!
  - Experience from software industry is fairly positive.

- Collective Code ownership means, that no code belongs to one person alone.
  Advantages:
  - If you need to push one analysis fast, you can have several people work on the same code without much extra effort.
  - If someone leaves, there is always someone, who can directly take over
    - e.g. doctorate funding of 3 years, but analysis still needs polishing.

Use an IDE

- Some learning is required, but then there a lot of useful functionality, like
  - easy navigation between related files,
  - display of environment around definition when pointing on an identifier,
  - tool integration (gdb,...),
  - smart refactoring across files,
  - …

- However, for small examples it may be easier to use classical editors, as well when working remote.

Issues of this Course

- Preparation, Organisation, Testing
- **General Best Practice Coding Rules**

  See next file!
- Tools I
- STL Usage
- Design Patterns
- Paradigms
- Tools II
- Efficiency
- Templates