Code Reviews - Best Practices

Gerhard Brandt
(University of Heidelberg)
Contents

- Code Review in a Top-Down Approach
- Documentation from Code
- Code Evolution: Best Practices

Not Contents of this lecture:
- Brilliant theories – just many small tips
- Professional engineering – just practical experience in poorly equipped HEP environments
- Dilbert Cartoons
Why Code Reviewing?

- Other people have engineered code for you
  - Maintenance
    - It's your honour to adjust this code where it shows suboptimal behaviour ( = fix bugs )
  - Evolution
    - You need to add a feature. But where and how?
  - Learning
    - They were not completely stupid: You can learn from their ingenuity
Approaching an unknown body of code

- “Eagle method”: Stay on top - dive in only as required!
  - Don't try to read 100k lines of code from the beginning to the end
- Read in increasing level of detail
  1) Directory Level
  2) Structure Level
  3) Codeline level
Approaching an unknown body of code: Tools and Example

• Tools used in this lecture:
  - Free and Simple
  - Easily available (come with Linux'es / Downloadable)
  - No IDEs (not available everywhere)
  - Mostly Cmdline and WWW based

• Example used: The ROOT Source Code
  - Used by many HEP physicists in practice
  - Never hurts to know something about it
  - (More suggestions for practice: Geant4, Mozilla, Linux Kernel, offline sw of your experiment, ...)

• Starting point:

  root_v4.00.08.source.tar.gz
Reading Code: Things to notice at Directory Level

- The Shell: First tool, even before the editor
- Size & Complexity?
  - No. of Packages, Files, Classes, Lines of Code
- Documentation?
  - Standard Set of README Files?
- Build Process?
  - Configuration? Compilation? Linkage?
- Unit Tests?
  - What is code, what are tests?

At Directory Level: **Project Organization**
Reading Code: Size and Complexity with `ls` and `wc`

- **Most powerful tool:** `ls`
  - Show organization
  - See filename conventions

- **Pipe output into `wc` for size estimates**

- **Example: Size of ROOT:**
  - `ls | wc` ~ 90 top level directories (Packages)
  - `ls -R1 * | wc` ~ 6285 files
    ~ 772 *.cxx files (Classes)
  - `cat */src/*.cxx | wc` ~ 660k lines of code: a lot if read sequentially ...

**Example**

```
/root/html/Module.mk
/root/html/inc/THtml.h
/root/html/src/THtml.cxx
```
Reading Code
Know your Editor

• We now start up or favorite editor
  – Lucky people have IDEs
  – The others have at least vi, Emacs, nedit, kate, ...
• Learn to profit from their features. They know:
  – Searching / Regular Expressions
  – Syntax Highlighting
  – ctags / idutils Index files
  – Block (un)indentation
  – Column selection
  – Block collapsing
  – File Browser / Tabbed Windows
  – ...
Reading Code: Survey using Birdseye Views

- Have a look at code from above using a tiny text size / multiple pages (Print Preview)
- Use a signature survey script
  - (http://c2.com/doc/SignatureSurvey/)
  - Strip code, show only brackets / delimiters
- Use syntax highlighting to lowlight comments, emphasize structure

```
Example
Signature of java/awt/print
```

```
/java/awt/print
409 Book ;{{}}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}{}{{}}
Reading Code: Things to at structural level

- What **Design Patterns** are used? How do they look like?
- What **Data Structures** are used? What is their interface?
- What are the **Framework Facilities**?
  - Error Handling / Logging / Steering / Cmdline Parsing
  - Maths
  - GUI / Graphics
  - I/O
  - Wrappers / Interfaces to legacy code (FORTRAN )
- If you happen upon these during browsing: Remember them!
  - Either ... you must use them anyway
  - ... if not, you avoid reinventing the wheel
Reading Code: Things to notice at Line Level

- Coding Conventions used
  - Naming Conventions
  - Formatting Rules: Layout, Indentation?
  - Commenting Rules / Comment Enrichment
  - Control Structures
- What Subset of C++ is used/allowed? STL? Templates?
- C++ Coding Standards in HEP are quite similar to each other
  - Taligent based: ROOT, ATLAS, ...
- Remember:
  - Advantage of Coding Standards comes mostly from **Consistent Use**
  - Even if they are suboptimal/outdated, continuing them makes sense within the same project
Reading Code: Things to skip at line level

- For quick reading, it's crucial to bypass skin and bones and get to the meat right away.
- Skip
  - Preprocessor Statements
  - Initialization
- Instead: Look for
  - Text (like window titles) or print-out you have seen
  - Comments marking important sections, like //FIXME
  - Tutorial Markers
  - Inner Loops
Scientific Studies exist on what is best (see Refs)
- But most important is to be consistent
- Advantages only gained when being consequent

Also it is known what is not
- Spaghetti Code
- Inverse Polish Christmas Tree Notation (Align operators in center)
- Dangling else

Code beautifiers exist: indent (C/C++), Jalopy (Java)

But Caveat:
- Colleagues could get lost if you reformat their code
- Time “lost” formatting code properly is regained only on second iteration (reviewing)
Reading Code
Searching and RegExps

• Most powerfull tool: *grep*
• Searching covers code and comments
• Stay general - Use word stem
• Chain *grep* to narrow your search
Documentation from Code

Introduction

- Tools exist to convert code into readable, navigable formats (HTML, LaTeX, PDF ...)
- Source: Code itself + enriched comments
- Progenitor: javadoc (by Sun for Java)
- Many different tools exist
  - ~40 listed on Doxygen page
    - http://www.stack.nl/~dimitri/doxygen/
- Mostly incompatible formats - chose wisely before coding
- Examples:
  Javadoc, Doxgen, Thtml, LXR, custom
• Example: Convert java code to HTML

```java
/**
 * Get a dummy object
 * @param name An unused string
 * @return Nothing (Null)
 * @see Dummy
 */
public Dummy getDummy(String name) {
    return null;
}
```

**getDummy**

```java
public Dummy getDummy(String name) {
    Get a dummy object

    Parameters:
    name - An unused string

    Returns:
    Nothing (Null)

    See also:
    Dummy
```
Documentation from Code
Doxygen

- “King” of doc tools (popular)
- Output to LaTeX, RTF, PS, PDF, HTML, man
- Good results for any (unenriched) source
- Create indices, graphs, diagrams ...
- Too many bells & whistles?
  - Some people prefer less features

Example
Mozilla Code Documentation
Documentation from Code THtml

- Doctool for the ROOT world
- Classes must be linked to ROOT executable
  - ClassImp, ClassDef Macros required
  - Non-C++-Files not documented

**Example**
Class TObject in HTML Format

- Inofficial outlook: THtml2
  - ROOT team choice: rewrite doc tool from scratch, incl. C++ parser etc.
  - more output formats, code browsing, ...
Documentation from Code LXR

- Perl to HTML – Source Code Cross Referencing Script
- Serves pages through webserver
- Used by Mozilla, FreeBSD, ROOT, ...
- Freetext search possible
- Updates several times a day
  but: not current state of repository!

**Example**

Class THtml in LXR
Documentation from Code GraphViz

- Free Graph generation package from BellLabs
- Simple Syntax - can be generated automatically
- Graphical representation of code structure
- Used by Doxygen for its graphs

Example

H1 Analysis Software Package Dependencies Graph (Perl to GraphViz Script)
Docu from Code: Do it yourself!

- Your doctool is missing a feature? Write your own tool!
- Code is written to be parsed – do so
- Possible in *ix world since 30 years
  - Classic tools: grep, sed, ...
- And of course there are Perl, Python, Ruby ...

Example

Package Index HTML page: Hyperized Directory Listing (Perl Script)
Code Evolution

• When adding new code it is time to apply the best practices learned in code reading
• If possible use tools to check contributions
• Compiler
  – Is a professional code reader
  – Tell him to be verbose, eg. on gcc use -Wall
• Regression Testing
  – Remember junit, cppunit
  – Often simpler testing possible
  – For HEP software exploit that the output must make sense in terms of physics
• cvs diff
Evolution of Code
CVS Browsers

- Most useful tools for manual checks of changes to code
- View the CVS Repository in the web browser
- Popular: CVSweb, ViewCVS
Summary

• Reading Code
  – Is a "soft skill" to be learned by experience
  – Can be automized using tools: General ones (ls, wc),
    Dedicated ones (doctools) and your own
  – Goes hand in hand with writing new code

• Documentation from Code
  – Is easy to extract using doctools

• Evolution of Code
  – Can be monitored by tools
Bibliography

- Spinellis D., *Code Reading*, Addison Wesley 2003
- McConnell S., *Code Complete*, Microsoft Press, 2\textsuperscript{nd} ed. 2004
- Gamma, E. et al., *Design Patterns, Elements of Reusable Software*, Addison Wesley 1995
- ...