upcomillas es

upcomillas es

Advanced Computing Tools for Applied Research

Chapter 3. Source code documentation

Jaime Boal Martín-Larrauri
Rafael Palacios Hielscher

Academic year 2014/2015

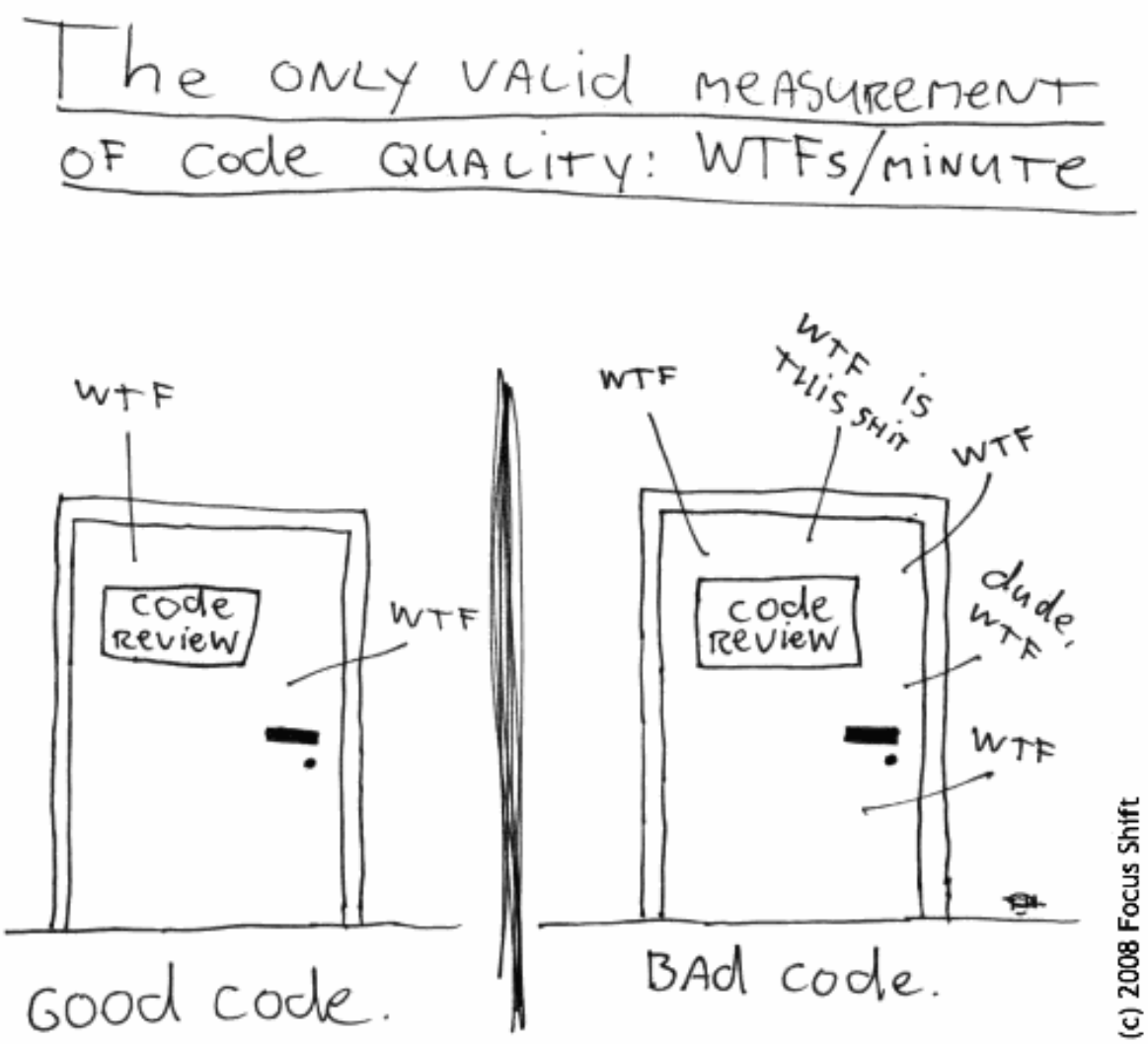
1

Fundamentals

Code quality

- Code is a technical document that might be read by people with different professional backgrounds
  - Developers, testers, end-users…
- Code should be easy to read for everyone
  - Holy grail of legibility ➡ Self-documenting code
  - Coding conventions ➡ Already on the right track but not enough
- Encapsulate algorithms in functions to enhance clarity
- Organize source files in folders following a logical structure
  - Extremely important for big projects

How would you measure code quality?



Comments

- Comments are easier to write poorly than well
- They can be more damaging than helpful
  - What if a comment does not agree with the code?
    - Is the code or the comment right?
    - Actually, a disagreement usually means that both are wrong
- Consider writing the program structure and comments first
- Otherwise comments should be written (and updated!) while programming
  - Documenting code afterwards is a nightmare!

Comments | Types

- Code restatements
  - Useless, gives more to read without providing additional information

```
% Calculate mean temperature
meanTemperature = computeAverage(temperatureArray);
```

- Code explanations
  - Used to explain complicated, tricky, or sensitive pieces of code
  - Generally required because the code is confusing ➡ Rewrite it!

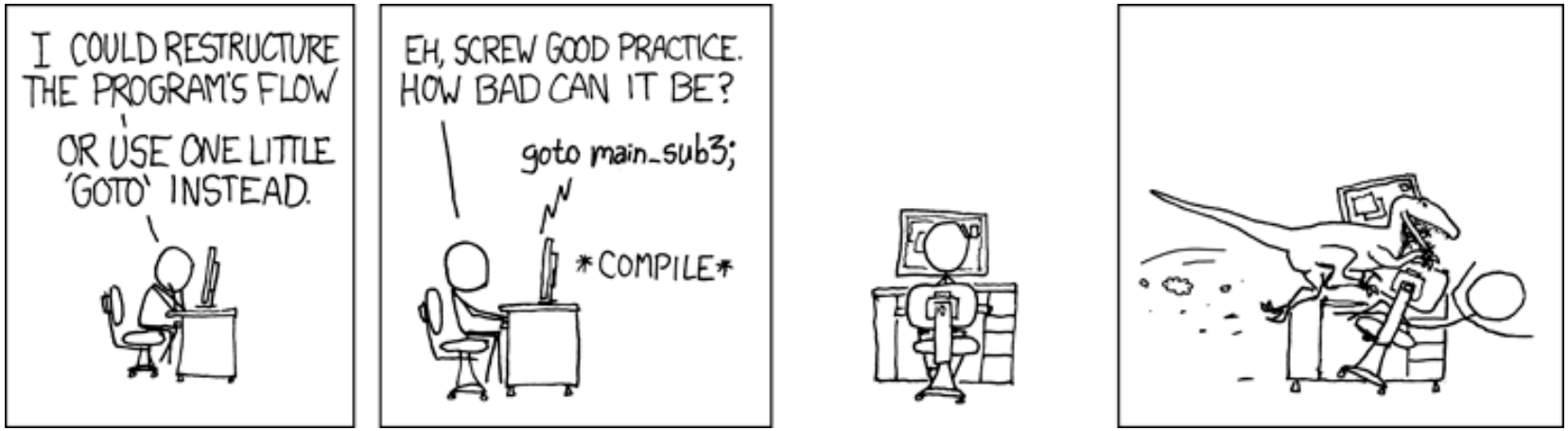


Image source: xkcd.com

Comments | Types

- Code markers
  - Used to indicate that there is work left to be done
  - There should not be any in production code!
  - Use a standard marker ➡ `TODO:`

```
function average = computeAverage(inputData)
  % TODO: Write function body
```

- Summary comments
  - Distill a block of code into one or two sentences
  - Allow to scan code more quickly ➡ Section headings

```
%% Plot results
figure(1); plot(temperatureArray);
figure(2); plot(windSpeedArray);
```

Comments | Types

- Intent comments
  - Explain the purpose of a code fragment
  - Programmer's thoughts when coding

- Information that cannot be expressed by the code itself
  - Copyright and confidentiality notices
  - Version numbers
  - Notes about the code's design
  - References to related requirements or architecture documentation
  - Links to online references…

Only comments that cannot be expressed in code, summary and intent comments are acceptable in production releases

Commenting guidelines | General

1. Make every comment count
   - Focus your documentation efforts on the code itself
2. Do not comment tricky code ➡ Rewrite it!
3. Write syntactically correct English sentences
4. Avoid abbreviations
5. Use comments to prepare the reader for what is to follow
6. Comment anything that gets around an error or an undocumented feature in a language or an environment
7. Justify surprises and violations of good programming style
   - For instance, because there is a big performance improvement

Commenting guidelines | General

8. Use styles that do not discourage modification

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File: badExample.m                                                          %
%                                                                            %
% Author: Jaime Boal                                                         %
%                                                                            %
% Date: 2015-02-16                                                           %
%                                                                            %
% Example of difficult to maintain comment.                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% File: goodExample.m
%
% Author: Jaime Boal
%
% Date: 2015-02-16
%
% Example of easy to maintain comment.
```

Commenting guidelines | Data declarations

9. Use end-of-line comments to annotate data declarations

```
voltage = 220;                  % Input voltage
```

10. Indicate the units of numeric data

```
voltage = 220;                  % Input voltage [V]
```

11. Document the range of allowable numeric values

```
voltage = 220                   % Input voltage [V] {0, 250}
```

Commenting guidelines | Statements and paragraphs

12. Avoid end-of-line comments on single statements
    - It is hard to write a meaningful comment for one line of code

```
area = length * width;  % Compute area
```

13. Write comments at the level of the code's intent

```
% Compute the mean temperature
% NOT: Divide the sum of temperatures by the number of elements
meanTemperature = 0;
nTemperatures   = length(temperatureArray);

for i = 1 : nTemperatures
  meanTemperature = meanTemperature + temperatureArray(i);
end

meanTemperature = meanTemperature / nTemperatures;
```

Commenting guidelines | Control structures

14. Comment the end of long control structures
    - Treat these comments as a warning indicating complicated code

```
for ...
  while ...
    if ...
      ...
    end  % if
  end  % while
end  % for
```

Commenting guidelines | Functions

15. Describe functions in one or two sentences at the top
16. Explain parameters
17. Comment on the limitations
18. Document the source of algorithms that are used
19. It is sometimes useful to provide a usage example

```
function c = addMe(a, b)
% ADDME  Add two values together.
%   c = ADDME(a) adds a to itself.
%   c = ADDME(a, b) adds a and b together.
%
%   See also SUM, PLUS.

  ...
end
```

Commenting guidelines | Functions

```
function foo(x, y, z)
% FOO One-line description goes here
%
%   FOO(x, y, z)
%
%   Multi-line paragraphs of descriptive text go here. It is fine for them to span lines. It's treated as
%   preformatted text; help() and doc() will not re-wrap lines. In the editor, you can highlight
%   paragraphs, right-click, and choose "Wrap selected comments" to re-flow the text.
%
%   More detailed help is in the <a href="matlab: help foo>extended_help">extended help</a>.
%   It is broken out like this so you can keep the main "help foo" text on a single screen.
%
%   Examples:
%     foo(1, 2, 3);
%
%   See also: BAR

disp(x + y + z);

function extended_help
% EXTENDED_HELP Some additional technical details and examples
%
%   Here you would put additional examples, technical discussions, documentation on obscure features...

error('This is a placeholder function just for helptext');
```

Commenting guidelines | Files

20. Give the file a name related to its contents
21. Include a comment block at the beginning of the file
    - Indicate the authorship
    - Include the date and a file version tag
    - Describe its purpose and any limitations
    - Add legal notices

```
% File: wifi.m
%
% Author: John Doe (john.doe@example.com)
%
% Date: 2015-02-16
%
% Version: 0.5.2
%
% Driver to handle Wi-Fi communications.
%
% (c) Copyright 2015 John Doe. All rights reserved.
```

2

Doxygen

What is Doxygen?

- It is the de facto standard tool for automatically generating documentation from annotated C++ sources
  - Natively supports: C, Objective-C, C#, PHP, Java, Python, VHDL…
  - There are filters to add support for MATLAB, Javascript, Visual Basic Perl…

- It can produce outputs in HTML, PDF, LaTeX, RTF…

- Check http://www.stack.nl/~dimitri/doxygen/ for details

Doxygen | Files

```
/**
 * @file        example.h
 *
 * @author      John Doe (john.doe@example.com)
 *              Jane Doe (jane.doe@example.com)
 *
 * @date        2015-02-16
 *
 * @version     0.5.2
 *
 * @copyright   2015 John Doe & Jane Doe. All rights reserved.
 *
 * @brief       Brief description of the file.
 *
 * Detailed description of the file.
 */
```

Doxygen | Functions

```
/**
 * @brief Extracts vertical lines from an image using a triangular kernel.
 *
 * Method adapted from:
 *  - A. Tapus and R. Siegwart, “Incremental robot mapping with fingerprints of places”, in Proc. IEEE/RSJ
 *    Int. Conf. Intelligent Robots and Systems, Edmonton, AB, Canada, Aug. 2–6, 2005, pp. 2429–2434.
 *  - A. Tapus, “Topological SLAM - Topological simultaneous localization and mapping with fingerprint of
 *    places”, PhD Thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 2005. [More detailed]
 *
 * Performs the following steps:
 *  1. Second order derivative of the Sobel operator over the \f$x\f$ direction.
 *  2. Histogram of vertical responses.
 *  3. Smooth the vertical histogram using a triangular kernel.
 *  4. Apply a threshold of value equal to the mean plus a standard deviation to remove false edge fragments.
 *  5. Non-maximum suppression with maxima separation.
 *
 * @param[in]  inputImage             Image to extract the vertical lines from.
 * @param[out] lineColumnIndices      Location of the lines in pixels on the x-axis.
 * @param[in]  minimumLineSeparation  Minimum separation in pixels between consecutive lines.
 * @param[in]  applyRegroupingFilter  If true, check the line separation condition after finding the lines;
 *                                    otherwise, do it simultaneously.
 */
void CLineExtractor::extractKernelVerticalLines(
  const CMatImage&            inputImage,
  std::vector<unsigned int>&  lineColumnIndices,
  const unsigned int&         minimumLineSeparation,
  const bool&                 applyRegroupingFilter);
```

References

- S. McConnell, *Code Complete: A Practical Handbook of Software Construction*, 2nd Ed., Microsoft Press, Redmond, WA, USA, 2004.

- Mathworks
  http://www.mathworks.com/

- M2HTML: Documentation System for MATLAB in HTML
  http://www.artefact.tk/software/matlab/m2html/

- Doxygen
  http://www.stack.nl/~dimitri/doxygen/