Documenting the code

Victor Ananyev, QFT
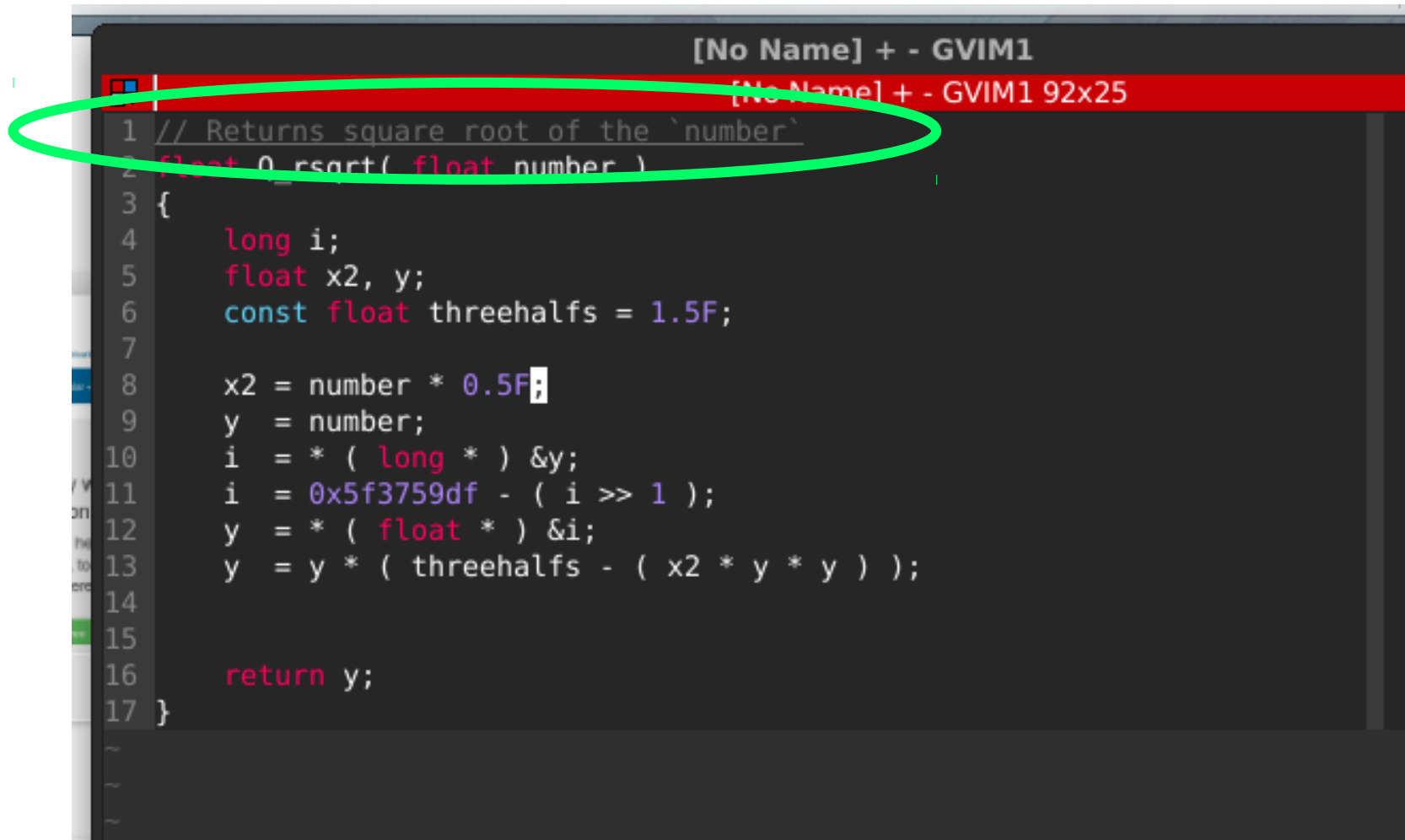
Why documenting code?

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;
    x2 = number * 0.5F;
    y  = number; i  = * ( long * ) &y;
    i  = 0x5f3759df - ( i >> 1 );
    y  = * ( float * ) &i; y  = y * ( threehalfs - ( x2 * y * y ) );

    return y;
}
```

What is the application of this function?

Now it is clear.


```
 1  // Returns square root of the `number`
 2  float Q_rsqrt( float number )
 3  {
 4      long i;
 5      float x2, y;
 6      const float threehalfs = 1.5F;
 7
 8      x2 = number * 0.5F;
 9      y  = number;
10      i  = * ( long * ) &y;
11      i  = 0x5f3759df - ( i >> 1 );
12      y  = * ( float * ) &i;
13      y  = y * ( threehalfs - ( x2 * y * y ) );
14
15
16      return y;
17  }
```
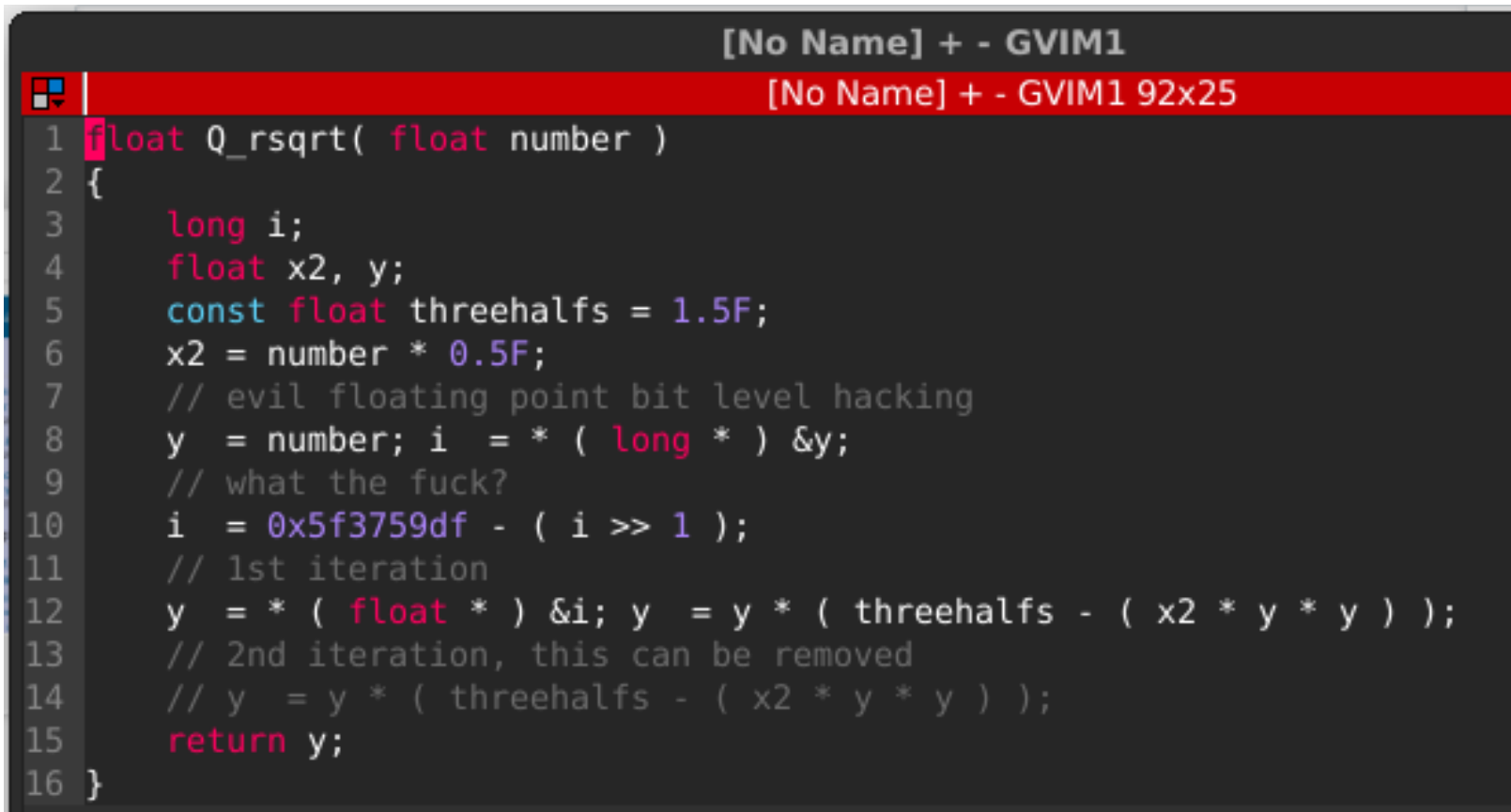
[No Name] + - GVIM1

[No Name] + - GVIM1 92x25

Would you like to see the original?

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;
    x2 = number * 0.5F;
    // evil floating point bit level hacking
    y  = number; i  = * ( long * ) &y;
    // what the fuck?
    i  = 0x5f3759df - ( i >> 1 );
    // 1st iteration
    y  = * ( float * ) &i; y  = y * ( threehalfs - ( x2 * y * y ) );
    // 2nd iteration, this can be removed
    // y  = y * ( threehalfs - ( x2 * y * y ) );
    return y;
}
```

The code above is the fast inverse square root implementation from Quake III Arena, stripped of C preprocessor directives, but including the exact original comment text

More stuff:
https://www.quora.com/What-is-the-most-complex-line-of-C-code-you-have-created-or-encountered

Approaches

- **"Documentation schmocumentation..."** It does not work for anybody.

- **"Write a {La}TeX document"** school, where the documentation is seen as a standalone product, which is produced independently. This works great for PDF output, and sucks royally for HTML and TXT output.

- **"Literate programming"** school, which abandons readability of both the program source code and the documentation source, which seems to be one of the best access protections one can put on the source code of either.

- **"DoxyGen"** school, which lets a program collect a mindless list of hyperlinked variable, procedure, class and filenames, and call that "documentation".

- **OpenOffice, LyX or Word, etc.** Anything that uses a fileformat that cannot be put into a version control system, because it is binary and non-diff'able.

Source: https://varnish-cache.org/docs/trunk/phk/sphinx.html

Literate programming example

```
32   #---------------------------------------------
33   {{{2 <h2>2. Lie algebras</h2>
34   #---------------------------------------------
35
36   <b>Definition 1: </b> <b>Lie algebra ``\mathfrakg``</b> is a linear space with the bil:
37   <ul> ``[ , ]:\mathfrak{g}\otimes \mathfrak{g}\to \mathfrak{g}``</ul>
38   with the additional property that Jacoby identity holds <i>(eq 2)</i>
39   <ul>
40   ``[x,[y,z]]+\text{(cyclic permutations)}=0``</ul>
41
42   Lie algebras can be finite- or infinite-dimensional. Finite-dimensional Lie algebras a
43
44   <p>
45   We will represent Lie algebra in the code as the object of class <i>lie-algebra</i>. W
46   Lie algebra is a vector space, so we really should create class for the vector space wl
47
48   <<Lie algebra class .scm>>=
49   (class 'lie-algebra 'object
50        <<Lie algebra methods .scm>>)
51   @
52
53
```

Source: https://github.com/unixtechie/Literate-Molly

Doxygen. Intro

```
/**
 *  A test class. A more elaborate class description.
 */

class Javadoc_Test
{
  public:

    /**
     * An enum.
     * More detailed enum description.
     */

    enum TEnum {
          TVal1, /**< enum value TVal1. */
          TVal2, /**< enum value TVal2. */
          TVal3  /**< enum value TVal3. */
        }
      *enumPtr, /**< enum pointer. Details. */
      enumVar;  /**< enum variable. Details. */

    /**
     * A constructor.
     * A more elaborate description of the constructor.
     */
    Javadoc_Test();
```

Source: https://www.stack.nl/~dimitri/doxygen/manual/docblocks.html

Results into

Test Class Reference `abstract`

A test class. More...

Public Types

enum	**TEnum** { **TVal1**, **TVal2**, **TVal3** }
	An enum. More...

Public Member Functions

Test ()	
	A constructor. More...

Constructor & Destructor Documentation

Test::Test ( )

A constructor.

A more elaborate description of the constructor.

Results into

Detailed Description

A test class.

A more elaborate class description.

Member Enumeration Documentation

enum Test::TEnum

An enum.

More detailed enum description.

Enumerator	
TVal1	enum value TVal1.
TVal2	enum value TVal2.
TVal3	enum value TVal3.

Source:
https://www.stack.nl/~dimitri/doxygen/manual/examples/jdstyle/html/class_test.html#details

Commenting functions and methods

```
/**
 * a normal member taking two arguments and returning a
 * @param a an integer argument.
 * @param s a constant character pointer.
 * @see Javadoc_Test()
 * @see ~Javadoc_Test()
 * @see testMeToo()
 * @see publicVar()
 * @return The test results
 */
int testMe(int a,const char *s);
```

* Description of parameters and return values are usually placed near the declaration line

** Notice *@param*, *@see*, *@return* commands

Results in

Member Function Documentation

```
int Test::testMe ( int           a,
                   const char *  s
                 )
```

a normal member taking two arguments and returning an integer value.

Parameters

> a an integer argument.
>
> s a constant character pointer.

See also

> Test()
>
> ~Test()
>
> testMeToo()
>
> publicVar()

Returns

> The test results

Formulae

Using **inline** formulae that appear in the running text. These formulae should be put between a pair of \f$ commands

```
The distance between \f$(x_1,y_1)\f$ and \f$(x_2,y_2)\f$ is
\f$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}\f$.
```

Results in:

The distance between (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

Formulae

Unnumbered displayed formulas that are centered on a separate line. These formulas should be put between \f[and \f] commands

```
\f[
   |I_2|=\left| \int_{0}^T \psi(t)
           \left\{
               u(a,t)-
               \int_{\gamma(t)}^a
               \frac{d\theta}{k(\theta,t)}
               \int_{a}^\theta c(\xi)u_t(\xi,t)\,d\xi
           \right\} dt
         \right|
\f]
```

Results in:

$$|I_2| = \left| \int_0^T \psi(t) \left\{ u(a,t) - \int_{\gamma(t)}^a \frac{d\theta}{k(\theta,t)} \int_a^\theta c(\xi)u_t(\xi,t)\,d\xi \right\} dt \right|$$

Referencing

Links to functions are created if one of the following patterns is encountered:

1. <functionName>"("<argument-list>")"

2. <functionName>"()"

3. "::"<functionName>

4. (<className>"::")n<functionName>"("<argument-list>")"

5. (<className>"::")n<functionName>"("<argument-list>")"<modifiers>

6. (<className>"::")n<functionName>"()"

7. (<className>"::")n<functionName>

More: https://www.stack.nl/~dimitri/doxygen/manual/autolink.html

Example of referencing

```
/*!
  Since this documentation block belongs to the class Autolink_Test no link to
  Autolink_Test is generated.

  Two ways to link to a constructor are: #Autolink_Test and Autolink_Test().

  Links to the destructor are: #~Autolink_Test and ~Autolink_Test().

  A link to a member in this class: member().

  More specific links to the each of the overloaded members:
  member(int) and member(int,int).

  A link to the variable #var.

  A link to the global typedef ::B.

  A link to the global enumeration type #GlobEnum.

  A link to the define ABS(x).

  A link to a variable \link #var using another text\endlink as a link.

  A link to the enumeration type #EType.

  A link to some enumeration values: \link Autolink_Test::Val1 Val1 \endlink and ::GVal1.

  And last but not least a link to a file: autolink.cpp.

  \sa Inside a see also section any word is checked, so EType,
      Val1, GVal1, ~Autolink_Test and member will be replaced by links in HTML.
*/
class Autolink_Test
{
  public:
    Autolink_Test();                   //!< constructor
```

Results in

Detailed Description

Since this documentation block belongs to the class **Autolink_Test** no link to **Autolink_Test** is generated.

Two ways to link to a constructor are: **Autolink_Test** and **Autolink_Test()**.

Links to the destructor are: **~Autolink_Test** and **~Autolink_Test()**.

A link to a member in this class: **member()**.

More specific links to the each of the overloaded members: **member(int)** and **member(int,int)**.

A link to the variable **var**.

A link to the global typedef **B**.

A link to the global enumeration type **GlobEnum**.

A link to the define **ABS(x)**.

A link to a variable **using another text** as a link.

A link to the enumeration type **EType**.

A link to some enumeration values: **Val1** and **GVal1**.

And last but not least a link to a file: **autolink.cpp**.

See also

Inside a see also section any word is checked, so **EType**, **Val1**, **GVal1**, **~Autolink_Test** and **member** will be replaced by links in HTML.

Bonus. Diagramms

If you do it right, Doxygen will generate inheritance diagramms for you automatically:

A Class Reference

`#include <diagrams_a.h>`

Inheritance diagram for A:
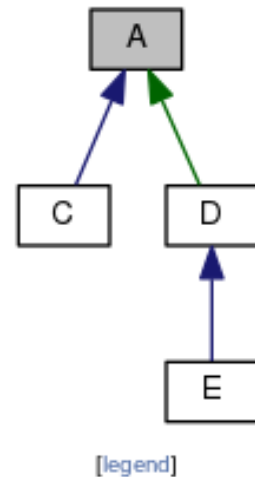
Collaboration diagram for A:

Too many colors?
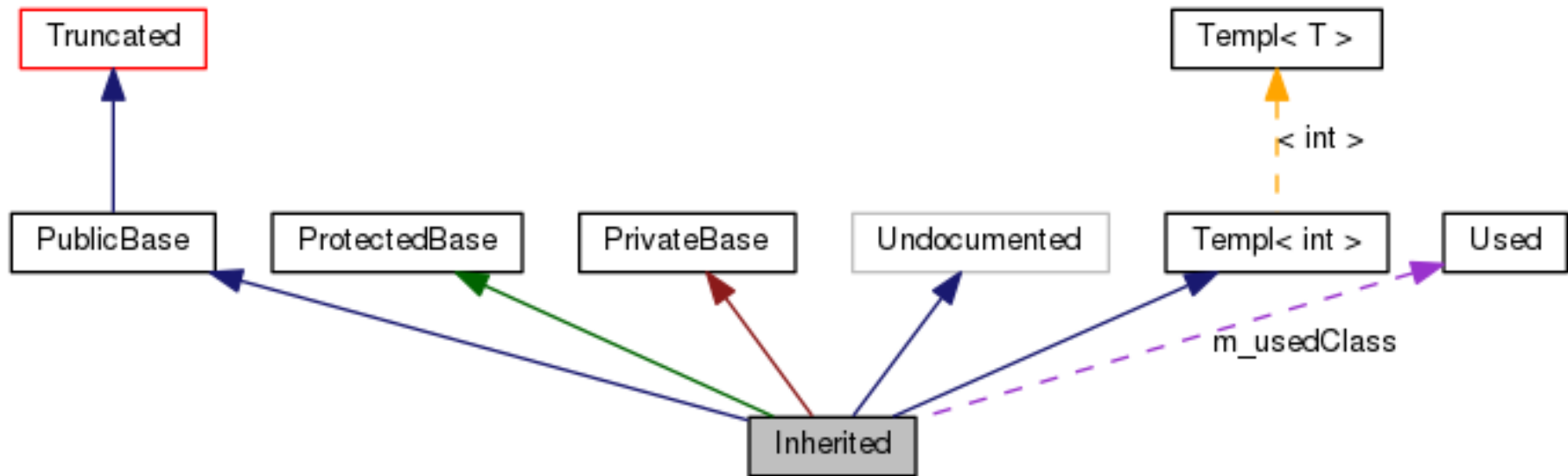
- **A filled gray box** represents the struct or class for which the graph is generated.
- **A box with a black border** denotes a documented struct or class.
- **A box with a gray border** denotes an undocumented struct or class.
- **A box with a red border** denotes a documented struct or class forwhich not all inheritance/containment relations are shown. A graph is truncated if it does not fit within the specified boundaries.

Too many colors?
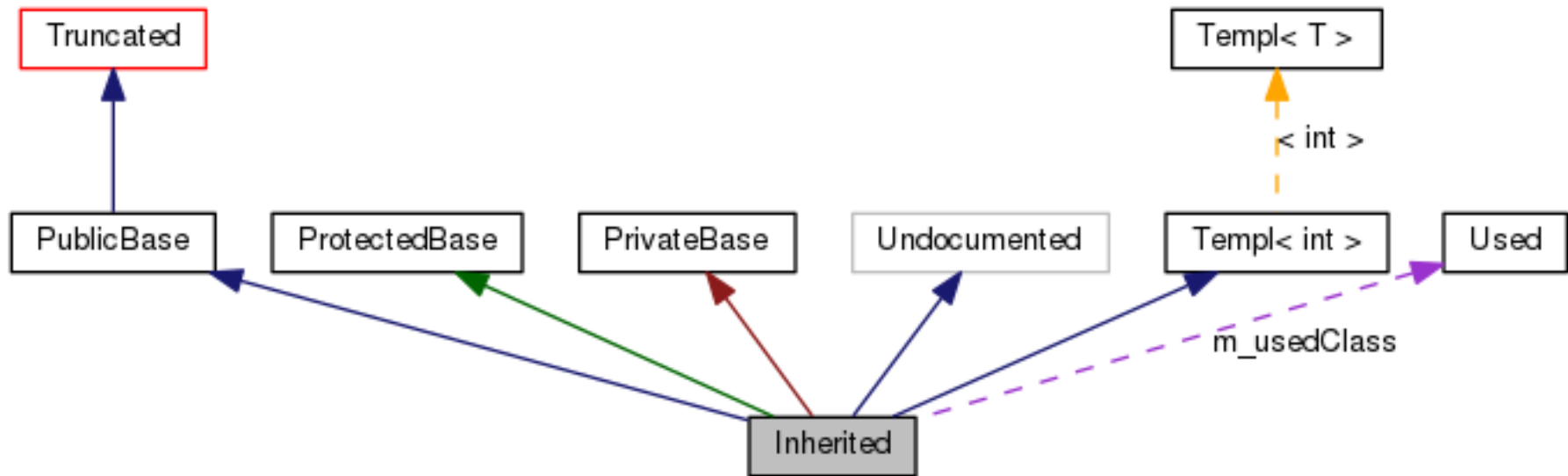
- **A dark blue arrow** is used to visualize a public inheritance relation between two classes.

- **A dark green arrow** is used for protected inheritance.

- **A dark red arrow** is used for private inheritance.

- **A purple dashed** arrow is used if a class is contained or used by another class. The arrow is labelled with the variable(s) through which the pointed class or struct is accessible.

- **A yellow dashed** arrow denotes a relation between a template instance and the template class it was instantiated from. The arrow is labelled with the template parameters of the instance.

Running Doxygen

Firstly, you should create Doxyfile – configuration file for doxygen engine

Basic directives in Doxyfile

- **PROJECT_NAME** = "My Project" // Project name, it's clear.

- **PROJECT_BRIEF** = "Brief description of your project" // Together with PROJECT_NAME will be shown in the title of the Doc.

- **OUTPUT_DIRECTORY** = doc // A place to where doxygen will generate all output. For instance, html documentation is under doc/html/index.html

- **CREATE_SUBDIRS** = YES // It forces doxygen to classify files into subdirs. Makes the output more structured.

- **FILE_PATTERNS** = *.py // You can leave it as is, but for readability of your Doxyfile, it is better to leave here only needed extensions to be searched for comments

- **RECURSIVE** = YES // Set it to YES, if you have subdirs in your project structure. Usually you have those.

- **INLINE_SOURCES** = YES // If set to yes, code listings will be included into documentation.

- **USE_MATHJAX** = YES // Use online compilation of LaTeX, but expressions become prettier.

- **GENERATE_XML** = YES // This you will need for thrid-party applications to use your docs

- **INPUT_FILTER** = "doxypypy -a -c" // Command to run on source code before passing it to doxygen. Directive is useful to convert Google Code Style standards for Python into Doxygen compliant. About this further.

Running

Now simply call doxygen in your project's root
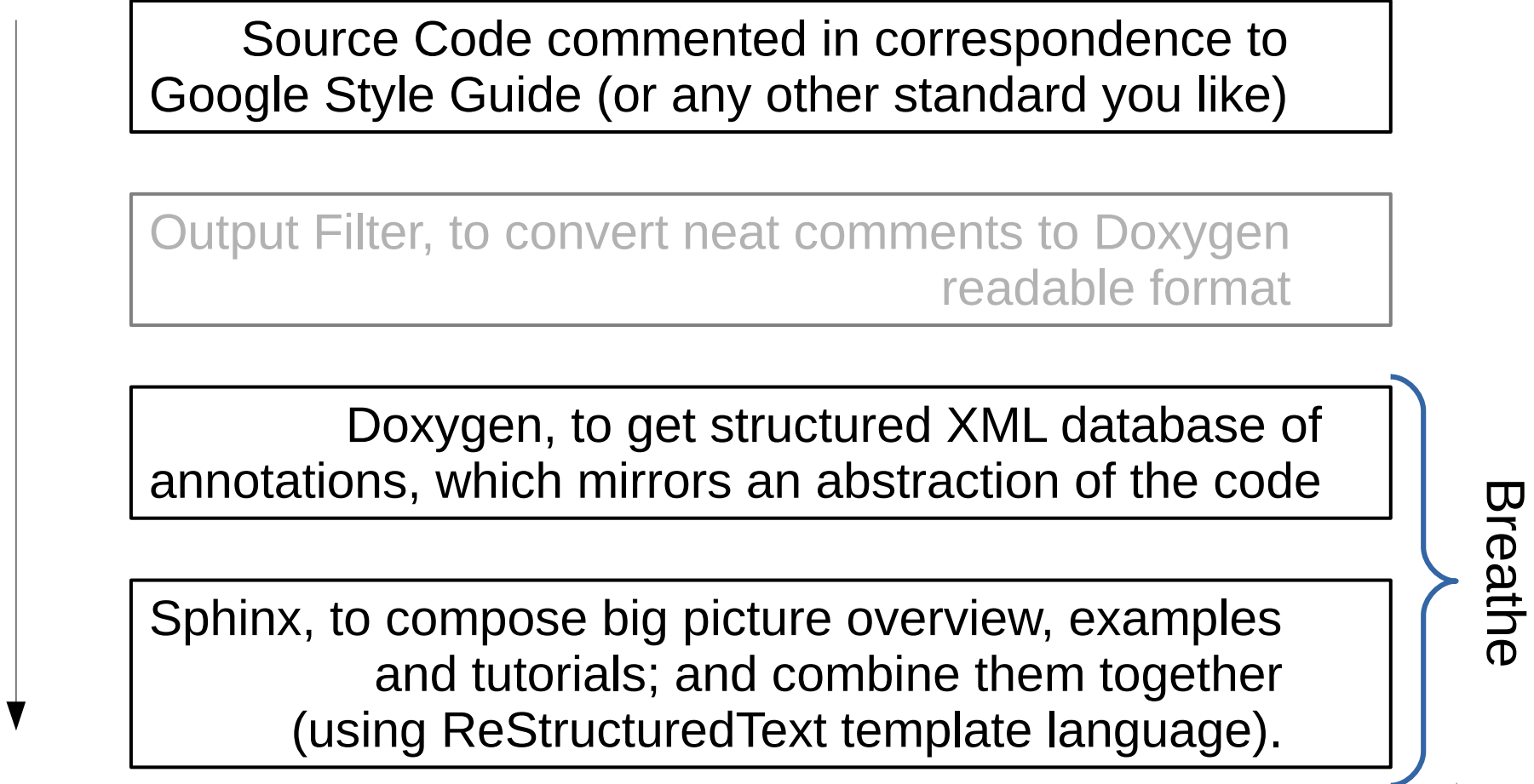

Results in:

Pros and cons of Doxygen

- :-) Doxygen does an excellent job in gathering source code annotations into a database (XML tree structure).

- !!! It forces you to mix templating and code description inside your sources.

Why does mixing is wrong?

- Let programmer describe his code, and let designer decide how the documentation should look.

- Code descriptions should be readable without third-party compilers (like Doxygen engine). Template directives make it worse readable.

Solution

Source Code commented in correspondence to Google Style Guide (or any other standard you like)

Output Filter, to convert neat comments to Doxygen readable format

Doxygen, to get structured XML database of annotations, which mirrors an abstraction of the code

Sphinx, to compose big picture overview, examples and tutorials; and combine them together (using ReStructuredText template language).

Breathe

Source:
librelist.com/browser//breathe/2014/6/19/a-successful-documentation-workflow-using-sphinx-doxygen-and-breathe/

What is Sphinx?

We'll use Sphinx to glue annotations extracted by Doxygen into Tutorial

template <class *PtrType*>
class **SmartPtr**

Public Functions

SmartPtr(PtrType *ptr*)

Initialization constructor.

Parameters

- `ptr`: a pointer to data

~SmartPtr()

Destructor to clean data referenced by stored pointer.

PtrType &**operator***() *const*

Dereference operator has to be overloaded to change stored data.

Now you can describe applications, or even provide some examples:

```
{
    SmartPtr<double> dptr(new double(1.278);
    # memory will be cleaned here
}
```

Sphinx's language is ReST


```
1 Hello world!
2 -----------
3
4 This is my first *instruction*.
5
6 Let's look at annotation for `SmartPtr` class:
7
8 .. doxygenclass:: SmartPtr
9     :members:
10
11 Now you can describe applications, or even provide some examples::
12
13     {
14         SmartPtr<double> dptr(new double(1.278);
15         # memory will be cleaned here
16     }
17
18 Hope you'll find it funny.
```
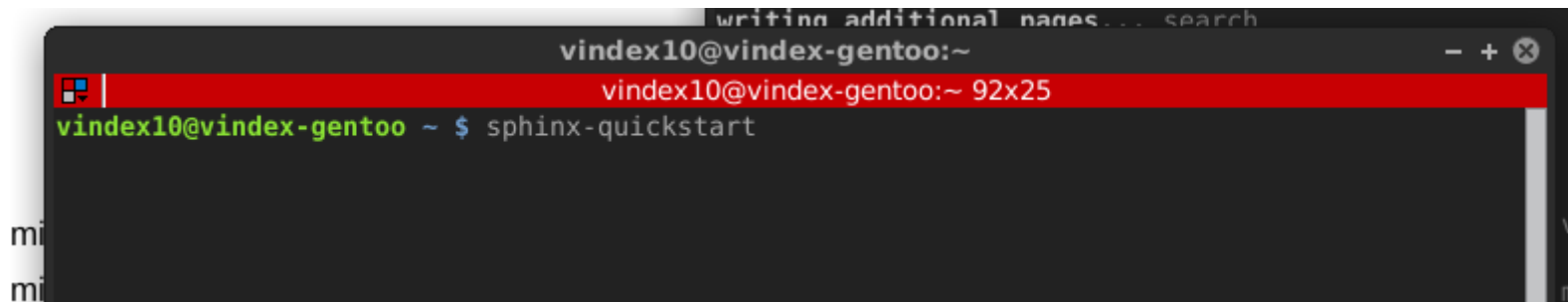
More about ReST syntax: http://www.sphinx-doc.org/en/1.6.4/rest.html

Where is Doxygen?


```
 1 Hello world!
 2 -----------
 3
 4 This is my first *instruction*.
 5
 6 Let's look at annotation for `SmartPtr` class:
 7
 8 .. doxygenclass:: SmartPtr
 9    :members:
10
11 Now you can describe applications, or even provide some examples::
12
13     {
14         SmartPtr<double> dptr(new double(1.278);
15         # memory will be cleaned here
16     }
17
18 Hope you'll find it funny.
```

More on Doxygen specific commands: http://breathe.readthedocs.io/en/latest/quickstart.html

How to breathe? Init Sphinx.

* Assuming, Doxygen has been already configured, and XML output has been generated.

** Assuming, Python packages Sphinx and Breathe have been already installed.

Firstly, we have to initialize Sphinx:

How to breathe? Check Sphinx.

Then, you can try to build basic doumentation, to check if everything is right:

```
(.env) vindex10@vindex-gentoo ~/downloads/doxytest $ sphinx-build -b html doc/sphi
 doc/sphinx/build/
Running Sphinx v1.6.3
loading pickled environment... not yet created
building [mo]: targets for 0 po files that are out of date
building [html]: targets for 2 source files that are out of date
updating environment: 2 added, 0 changed, 0 removed
reading sources... [100%] instruction
looking for now-outdated files... none found
pickling environment... done
checking consistency... done
preparing documents... done
writing output... [100%] instruction
generating indices... genindex
writing additional pages... search
copying static files... done
copying extra files... done
dumping search index in English (code: en) ... done
dumping object inventory... done
build succeeded.
```

How to breathe? Configure breathe.

Fill some configs at `doc/sphinx/source/conf.py`

```
18 #
19 import os
20 # import sys
21 # sys.path.insert(0, os.path.abspath('.'))
22
23 # -- General configuration ------------------------------------------------
24
25 # If your documentation needs a minimal Sphinx version, state it here.
26 #
27 # needs_sphinx = '1.0'
28
29 # Add any Sphinx extension module names here, as strings. They can be
30 # extensions coming with Sphinx (named 'sphinx.ext.*') or your custom
31 # ones.
32 extensions = ['sphinx.ext.autodoc',
33     'sphinx.ext.todo',
34     'sphinx.ext.mathjax',
35     'sphinx.ext.ifconfig',
36     'sphinx.ext.viewcode',
37     'sphinx.ext.githubpages',
38     'breathe']
39
40 breathe_projects = {"smartptr": os.path.abspath("../../doxygen/xml")}
41 breathe_default_project = "smartptr"
42
```
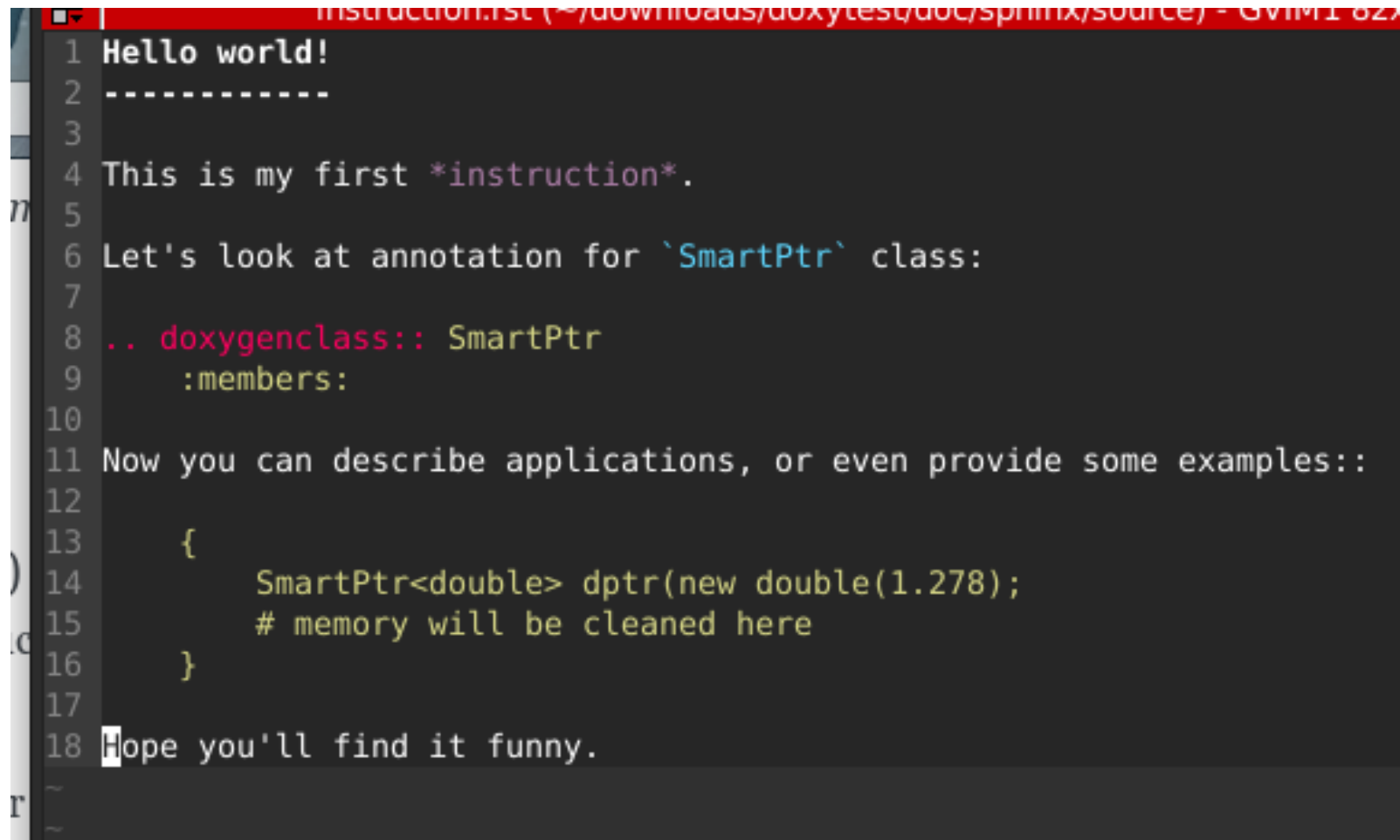
How to breathe? Basic template

Add an entry into Table of Contents at `doc/sphinx/source/index.rst`

```
 1 .. Smart Pointer documentation master file, created
 2    sphinx-quickstart on Tue Oct 10 01:05:44 2017.
 3    You can adapt this file completely to your liking,
 4    contain the root `toctree` directive.
 5
 6 Welcome to Smart Pointer's documentation!
 7 =========================================
 8
 9 .. toctree::
10    :maxdepth: 2
11    :caption: Contents:
12
13    instruction
14
15
16
17 Indices and tables
18 ==================
19
20 * :ref:`genindex`
21 * :ref:`modindex`
22 * :ref:`search`
```

How to breathe? Basic template

Create corresponding file `doc/sphinx/source/instruction.rst`

```rst
1  Hello world!
2  -----------
3
4  This is my first *instruction*.
5
6  Let's look at annotation for `SmartPtr` class:
7
8  .. doxygenclass:: SmartPtr
9      :members:
10
11 Now you can describe applications, or even provide some examples::
12
13     {
14         SmartPtr<double> dptr(new double(1.278);
15         # memory will be cleaned here
16     }
17
18 Hope you'll find it funny.
```

Deep breathe and Run

Run Sphinx again, now it is able to use Doxygen commands

Results in

template <class *PtrType*>
class **SmartPtr**

Public Functions

SmartPtr(PtrType *ptr*)

    Initialization constructor.

    ### Parameters

        • ptr: a pointer to data

~SmartPtr()

    Destructor to clean data referenced by stored pointer.

PtrType &**operator***() *const*

    Dereference operator has to be overloaded to change stored data.

Now you can describe applications, or even provide some examples:

```
{
    SmartPtr<double> dptr(new double(1.278);
    # memory will be cleaned here
}
```

See full example on Github: https://github.com/vindex10/codoc

Thanks