Literate programming IDE for LAR-CC [*]

Alberto Paoluzzi

January 20, 2015

Abstract

This document introduces the developer of geometric libreries and applications to the integrated development environment (IDE) set up for documentation and multilanguage development using LAR-CC, the Linear Algebraic Representation for geometry, manufacturing and physics with *Chains* and *CoChains*. This IDE is strongly based on the literate programming tool *Nuweb*, aiming at embedding the code in the documentation, and not vice-versa. The main goal of this framework is to facilitate how to express the *why* of software design decisions, and not only the tricky details of low level coding. I would recommend writing programs as if they were research papers and treat the code as you would write mathematical expressions in a research paper. Using multiple programming languages is allowed and even encouraged in `larcc`. When possible, the same functions coded in different languages should stay close within the same document subsection. The `larcc` IDE integrates a few programming, documentation and version control tools, including LaTeX, *Nuweb*, *Pandoc*, *Git*, and Leo editor.

Contents

[*]This document is part of the *Linear Algebraic Representation with CoChains* (LAR-CC) framework [CL13]. January 20, 2015

1 Up and running

1.1 Prerequisites

LaTeX The `larcc` IDE requires the users to embed the compute code within LaTeX files written for documenting their work. Therefore the first requirement is a working LaTeX environment. “As TeX Live is the basis of MacTeX, and is the TeX system for Unix, if you work cross-platform and want an identical system on all of your machines, then TeX Live is the way to go” [Wri12].

Python As of today, most of `larcc` development was done in Python. Hence a working Python environment is required, including three packages: `scipy`, `pyopengl`, and `pyplasm`. On a Mac, Python is installed by default, whereas `scipy` and `pyopengl` may be installed in the terminal by doing

```
$ sudo easy-install scipy
$ sudo easy-install pyopengl
```

Finally, to install `pyplasm` look at the README file in its downloaded directory:

```
$ git clone https://github.com/plasm-language/pyplasm
```

Nuweb In 1984, Knuth introduced the idea of literate programming. The idea was that a programmer writes just one document, the web file—with suffix `.w`, that combines the documentation with the code. *Nuweb* works with any programming language and LaTeX, and is probably the simplest incarnation of the Knut’s original work. The web site of the tool is sourceforge.net/projects/nuweb/. A revised version of source files can be found on code.google.com/p/nuweb. This package can build using the standard tools:

```
$ cd <path-to>/nuweb/
$ ./configure
$ make
$ sudo make install
```

For some documentation read the wiki page. Test your installation by just compiling to *pdf* the `nuweb.w` document itself, whose chapter one contains the user documentation:

```
$ nuweb nuweb.w
```

Of course, in order to extend `larcc` and/or to make an efficient use of it, you are supposed to read carefully the first chapter of the *nuweb.pdf* document.

1.2 Download

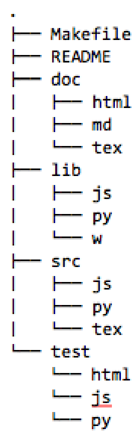
You may or may not put your IDE under the protection of a version control system. The `larcc` project comes from *Github* equipped with an integrated *git* system, that you are free either to use or not to use. Of course, my advice is of making the best use of it. Hence write in the terminal:

```
$ git clone https://github.com/cvdlab/larcc
```

Thats all. Now move to the `larcc` directory and give a look at its content, a bunch of directories, subdirectories and files, of course. Write

```
$ tree -L 2
```

and you will see something like



```
.
├── Makefile
├── README
├── doc
|   ├── html
|   ├── md
|   └── tex
├── lib
|   ├── js
|   ├── py
|   └── w
├── src
|   ├── js
|   ├── py
|   └── tex
└── test
    └── html
    └── js
    └── py
```

Figure 1: A sample of the `larcc` framework

1.3 Working-file

For the impatient, open a terminal, and change directory to `larcc`, the root of the LAR-CC project, if not already there, whenever you hold it on your system.

Then create your working-file `src/tex/<name>.tex` as a copy of `src/tex/template.tex`. Open the working-file with an editor, possibly with one aware of the LaTeX syntax, and make few mandatory changes on the template text:

Title change the `title` command, substituting the “Title” string with the actual sentence to be used as document title;

Author do the same for the `author` command, substituting the "TheAuthor" string with the actual document author;

Bibliography substitute the "template" string with the actual working-file name (without the file extension).

Test files folder within the `test/py/` folder, create a folder named `<name>` — the working-file name (without the file extension) — where to put the example files and the file used for unit testing of the module.

`Makefile` **editing** edit the first non-comment line of the `Makefile` writing

```
NAME = <name>
```

1.4 Using the IDE

Finally, you may starting the real work by writing the documentation/code within your `<name>.tex` file, using the simple mark-up rules of Nuweb.

In short, in order to use the development environment, you must (a) open your terminal and move to the `larcc` directory, (b) write a `tex` file, including documentation and suitably tagged computer code, (c) save it in the `src` subdirectory, and (d) execute a `make` command, asking either for generation of the `pdf` or the `html` documentation, or execution of unit testing, or simply for the compilation of the source code.

1.4.1 Using Leo

Currently, `Leo` is not being used. I found it extremely interesting, but the learning curve is too hard for me.

Leo is a multi-platform and open-source outliner and *non-linear* text editor allowing for fundamentally different ways of using and organizing data, programs and scripts. Leo has been under active development for 15 years and has an active group of developers and users. The Leo environment allow to structure either simple or very complex projects as an acyclic graph, where nodes (said clones), and hence the subgraphs rooted in them, may have more than one father. Accordingly, every update to a clone node immediately extends to every its instances within the tree-like walk-through of the whole outline.

1.4.2 Using make

When using the IDE, the user must open the `Makefile` with any text editor, and modify the current values of two user-definable variables, according to his will:

```
NAME = <name>
LANGUAGE = <language>
```

where `<name>` is name of the new working document, and `<language>` may be only `py` (for Pyton). Soon such values will be extended to include `<js>` for Javascript and `<lhs>` for Literate Haskell. The make *targets* currently available in the Makefile are the following:

all is the default option. Its execution produces a *pdf* document in the `doc/pdf` subdirectory, and a pair of *tex*/*bbl* documents in the `doc/tex` subdirectory, all using name `<name>`;

html similar to the default option, but produces a directory named `<name>` wth a bunch of *html* pages, located in the `doc/html` subdirectory;

test to execute the tests contained in the directory `test/<language>/<name>`

exec to execute *nuweb* on the working document, i.e. on `<name>.tex`[1]; this execution generates both the LaTeX documentation file and the source output files (for example the unit tests) written in the coding `<language>`;

clear to remove all the working files from the root directory. Used by other commands. To be invoked by the user just in case that something did not work out.

2 Structure of `larcc`

The `larcc` project is hinged around four subdirectories (see Figure 1) and a Makefile. The meaning and function of the four subdirectories are listed below.

src (for *Source*) is the directory `src` that contains all the source documents, and in particular the `tex` files including the code of the algorithms and the tests developed in the project. It is divided in subdirectories related to the type of the source file itself. For example a `html` directory will contain the user-defined `css` source files, and the `lhs` directory the *literate Haskell* sorce files, to be processed directly by the Haskell compiler *GHC*. Such directories will also contain other programming resources needed to build the libreries or the applications developed in the LAR-CC project.

test test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: write a "unit test", get it to pass, run all tests, clean up the code (see Figure 2). The subdirectory `test` is the repository of test suites, collection of test cases, and of unit test files, possibly grouped depending on the source language, to be launched either individually, while writing each single software function or application, or collectively before committing or pushing novel developments or subsystems.

[1] Actually, `<name>.tex` is internally copied to a scratch file `<name>.w`, in order to allow the user to work comfortably with an editor knowledgeable of the LaTeX syntax.

lib is the repository of compiled and/or executable programs. In particular, it is the place where to store and retrieve all the libreries or modules or applications developed by compilation of any document within the `scr` subdirectory of the `larcc` system, excluding the documentation.

doc conversely contains all the documentation generated by the system, once again subdivided depending on the language and tools used for its reading or examination.

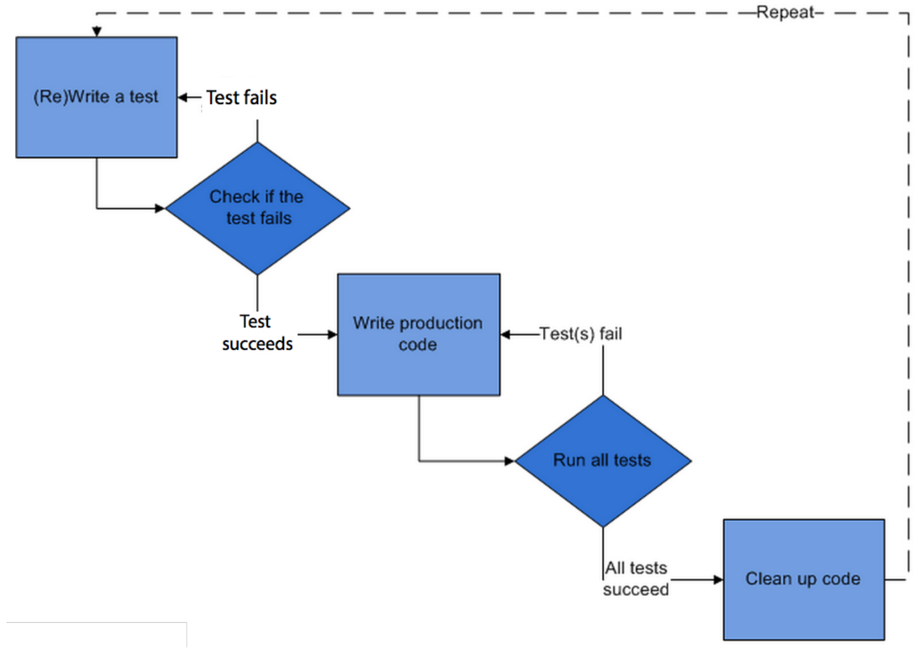


Figure 2: test-driven development (TDD) cycle (from Wikipedia)

References

[CL13] CVD-Lab, *Linear algebraic representation*, Tech. Report 13-00, Roma Tre University, October 2013.

[Wri12] Joseph Wright, *TEX on Windows: MiKTEX or TEX Live?*, TUGboat **33** (2012), no. 1.