DRAGON AND DONJON: A LEGACY OPEN-SOURCE REACTOR PHYSICS PROJECT AT POLYTECHNIQUE MONTRÉAL

A. HÉBERT
Polytechnique Montréal
Montréal, Canada
Email: alain.hebert@polymtl.ca

Open-Source development is a recognized and accepted practice for developing production software at Polytechnique Montréal. Today, our reactor physics codes DRAGON5 and DONJON5 are used by the Canadian and international Nuclear Industry as components of mission-critical platforms related to safety. In the future, we plan to extent the use of DRAGON5 to medical applications related to radiotherapy.

We present our 30-year experience with Open-Source issues, including LGPL licensing, distribution, long-term maintenance and developer/user interactions. An emphasis is made on quality assurance (QA) issues which become crucial with Open-Source development, especially in a university context. Our QA system will be described in detail

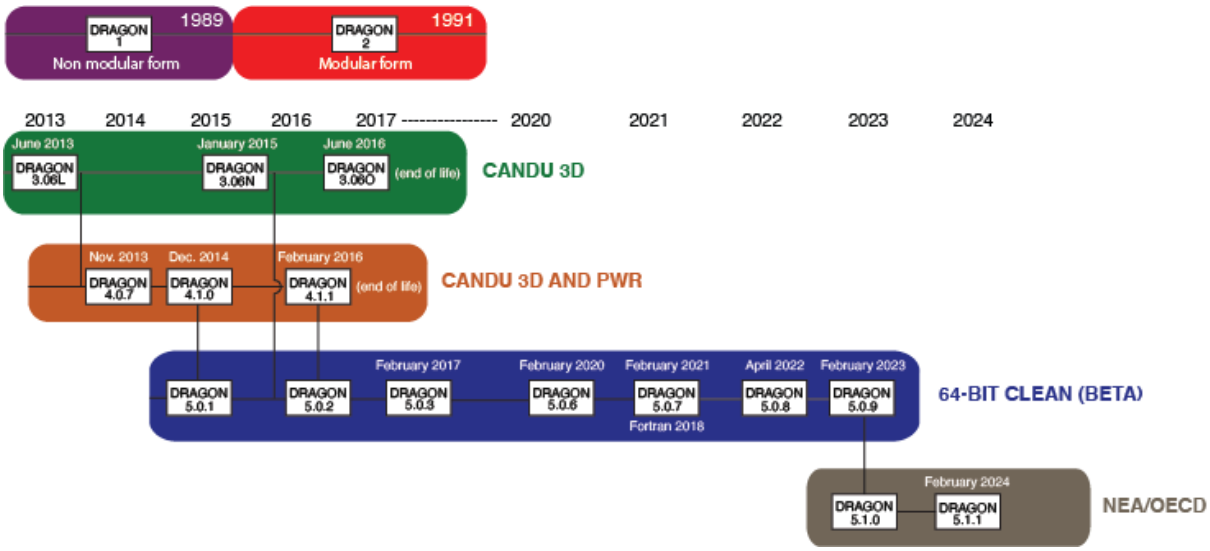


FIG. 1. DRAGON timescale.

As depicted in Fig. 1, the DRAGON project was initiated in the eighties and has been kept alive for more than 35 years at the *Institut de Génie Nucléaire* (IGN) of Polytechnique Montréal. After eight years of internal developments at IGN, DRAGON 1 was released in 1989. DRAGON is now a full feature lattice code with the following capabilities:[1,2]

- solution techniques of the Boltzmann transport equation (BTE) based on collision probabilities (PIJ), method of characteristics (MOC) and discrete ordinates method (SN)
- solution technique of the Boltzmann-Fokker-Planck (BFP) equation using the discrete ordinates method for applications in radiotherapy.
- access to microscopic cross-section libraries in various formats (DRAGLIB, WIMSLIB, APOLIB, MATXS, etc.)
- resonance self-shielding models (equivalence and subgroup)

— burnup calculations with the solution of the Bateman equations
— leakage and diffusion coefficients calculation
— *superhomogénéisation* (SPH) capabilities
— homogenization and condensation of cross sections and diffusion coefficients
— production of burnup-dependent multigroup cross section libraries with local-parameter branching
— capability to design computational schemes (two-level schemes).

DRAGON5 evolved from a modular Fortran-77 project to a Fortran-2013 project in December 2014 and recently to a modern Fortran-2018 project in February 2021. From the beginning, we decide to develop DRAGON as an Open-Source project under the GNU Lesser General Public License (LGPL). DRAGON5 can be freely downloaded from the website at Ref. [3].

DONJON5 is a full-core simulation code with the following capabilities:

— solution of the neutron diffusion equation
— solution of the simplified P_n (or SP_n) equation
— multi-parameter interpolation in the cross-section database
— micro-depletion of particular isotopes (Xe, Sm, etc)
— Boron critical control
— simplified thermo-hydraulics (steady-state and transient) for CANDU and PWR
— management of reactivity devices for CANDU and PWR
— simulation of refuelling strategies in CANDU and PWR, including in-line fuelling in CANDU reactors
— time averaged CANDU core calculations
— pin-flux reconstruction capability in PWR
— 3D neutron kinetics
— capability to design computational schemes (burnup cycles, accident scenario, etc.).

Both DRAGON5 and DONJON5 are collections of independent computational modules that can be linked together using the embedded CLE-2000 scripting language in order to construct custom *computational schemes*, entirely written with CLE-2000 scripting syntax. Computational schemes are specific to each type of reactor or application and contains the intellectual property (IP) of the users. Computational schemes are not subject to the LGPL license of DRAGON. This distinction is possible because DRAGON5 and DONJON5 are released under the “lesser” form of the license and not under the more restrictive GPL version of it. The drawback of this approach is that users need to learn the capability to build their own computational schemes to use the code. This requires much more know-how than using competing codes such as CASMO5.

DRAGON3 is currently an Industrial Standard Toolset (IST) component used by the CANDU Owner’s Group (COG) across the world for the representation of CANDU reactivity devices in 3D lattice geometry. In the actual setup, DRAGON3 is used together with WIMS-AECL for the generation of fuel tables required to complete safety analyses. In the future, the COG expects to replace both DRAGON3 and WIMS-AECL with DRAGON5. The long-term maintenance issue is important in this context as current CANDU reactors are expected to remain in operation until 2060. An agreement has been negotiated between COG and Polytechnique Montreal to provide the required commitment. However, long term commitment was obtained for DRAGON5, not for DONJON5

The long-term maintenance issue is closely related to the quality assurance (QA) issue, as DRAGON5 and DONJON5 are developed under strict QA procedures, as described in Ref. [4]. All interactions between DRAGON5 and DONJON5 users and the development team and all modifications to the code resulting from these interactions are registered and supervised by the QA system. Such an approach is essential in an academic context where student contributions could potentially affect the code stability. This is true for most Open-Source projects. Strict adherence to the QA rules is essential to maintain code stability in the long term.

Three aspects of development procedures are described:

- version control of the project components
- issue tracking, spiral development management, and continuous integration
- configuration management of the code.

Version control is the art of managing changes to information. We have chosen Subversion, an open-source version control system that is free, powerful, well-accepted by computer scientists and widely available.[4] Subversion is a widely used system used to keep track of the historical evolution of the project. We use Subversion for the totality of Version5 components. The information in the repository is organized with a directory structure, as shown in Fig. 2. The repository contains:

- Fortran 2018 and ANSI C software
- Makefiles
- basic CLE-2000 procedures
- Bash and Python scripts
- non-regression tests for the continuous integration procedure
- Latex documentation
- issue tracking information (QA database).

Pre- and post-commit scripts have been added to the repository to validate commit operations and to automatically perform the second automatic issue-tracking commit. Both scripts are written in Python and are based on the pysvn application programming interface.[5] This information is recovered as a directory named issues_wc containing a set of card-index, each of them representing a development issue.

As most software projects, Version5 is evolving with the spiral cycle development model. The spiral development model consists of developing the product as a sequence of cycles; each of them devoted to the development of a single modification (aka project increment) to any of the project components. Each cycle is tagged with an *issue identifier* of the form issuennnnn used as reference through the development cycle. The traceability of the actions made by the developers is made possible by the introduction of this issue identifier. The issue information is stored in a file named issuennnnnn located in the repository. This file is the *card-index* (*fiche d'intervention* in french) characterizing the cycle. An issues_wc working copy is created to facilitate QA operations.

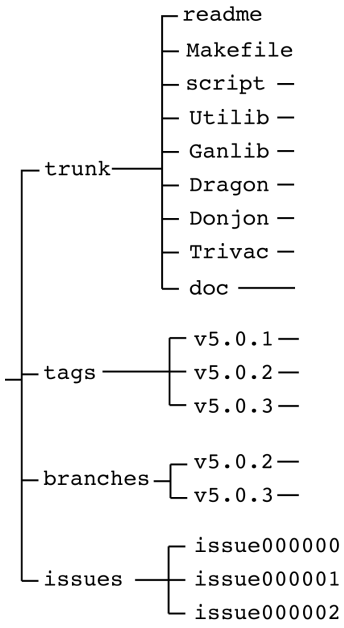


FIG. 2. The Version5 repository structure.

A cycle consists of the following steps:

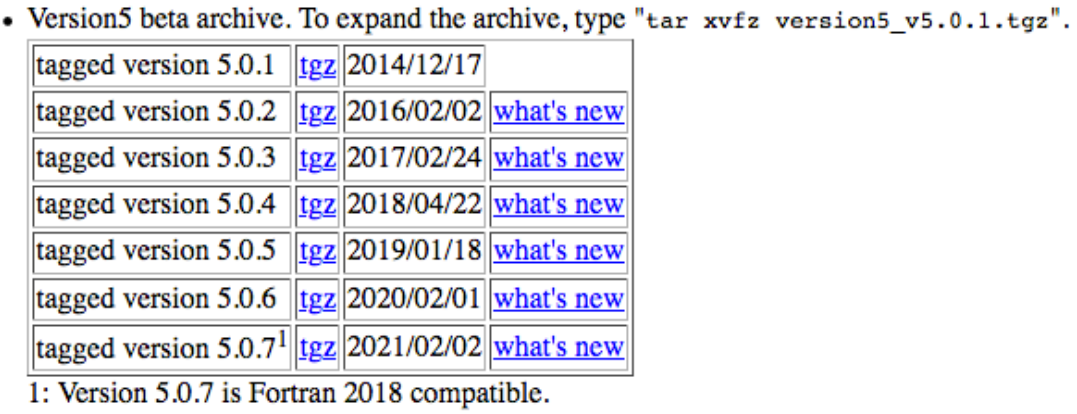
- A development cycle is *always* initiated by a developer at its own initiative or after receiving an issue submission form from a user. An issue submission form can be rejected by the developer in charge of the sub-project or can be accepted. If the issue submission form is rejected, no record is made in the issues_wc working copy. If the issue submission form is accepted, the information relative to the issue is stored in file issuennnnnn.
- The second step include the specification and conception work related to the issue. Here, the amount of work is highly issue dependent. Some issues (such as assistance request) may not even require any specification and/or conception work; others may take years to complete. This step may involve proposed modifications in documentation (LCM object specifications and user's guide of the modules) and proposed non-regression tests. This information is copied in the developer's working copy and can optionally be committed.
- The third step is the programming of the increment and its introduction in the developer's working copy. At the end of this step, the developer performs an update operation on its working copy to make sure that no conflicts with other developers exist.
- The fourth step is the validation of the project increment and its commit in the repository. A set of selected non-regression tests are performed with the developer's working copy. A continuous integration procedure is implemented with the make tests directive. If these tests are conclusive, the issue is closed and an *issue closing report* is written, appended to the card-index named issuennnnnn and emailed to the project user group. References to the issue-related documentation are also added to the card-index. A failure of the non-regression tests may require coming back to step 3 (or even to step 2 if a specification/conception error is detected). In case of success, both the card-index and the developer's working copy are committed to the repository.

The *issue identifier* is of the form issuennnnnn where nnnnnn is equal to the maximum existing value plus one, as assigned automatically by the pre-commit script. If the user sent additional information, it can be included in file issuennnnnn. At any time during the cycle, file issuennnnnn can be updated by the developer in charge of the issue and re-committed as

```
cd issues_wc/
svn commit -m 'issuennnnnn:' .
```

A cycle may require many commits. After each commit of a Version5 item, file issuennnnnn is automatically updated and re-committed by the post-commit script. The issue card-index trace the progress of the work made by the developer(s) to solve the issue.

— Once a year, all project increments are collected into a tagged version identified as v5.n.m. and made available on the official project website of Ref. [2]. The hyperlink "what's new" is a list of issues references and short descriptions relative to this tagged version, as depicted in Fig. 3.



- Version5 beta archive. To expand the archive, type "`tar xvfz version5_v5.0.1.tgz`".

tagged version 5.0.1	tgz	2014/12/17	
tagged version 5.0.2	tgz	2016/02/02	what's new
tagged version 5.0.3	tgz	2017/02/24	what's new
tagged version 5.0.4	tgz	2018/04/22	what's new
tagged version 5.0.5	tgz	2019/01/18	what's new
tagged version 5.0.6	tgz	2020/02/01	what's new
tagged version 5.0.7[1]	tgz	2021/02/02	what's new

1: Version 5.0.7 is Fortran 2018 compatible.

FIG. 3. Version5 tagged versions in website Merlin.

Configuration management is the art of assembling the project components, available in the repository, to build the end product of the project. In case of Version5, the end-product is a set of executables for codes DRAGON, TRIVAC and DONJON on different UNIX-like operating systems (including PCs under Cygwin) and a set of PDF reports.

Version5 configuration management uses the simplest existing approach as it requires relatively simple compiler technologies. In fact, the Version5 sources can be compiled with ISO Fortran--2003 and ANSI C compilers and its documentation can be compiled with Latex. Version 5 configuration uses nothing more sophisticated than the technologies available in usual Unix distributions.

The basic principle of Version5 configuration management consists of executing {\tt install} and/or make scripts (gmake is used) *within* the user's or developer's working copy. Binary files (libraries, executables, PDF files) will be created but will not be managed by the version control system (*we must avoid committing any binary information*). A commit operation can still be performed on the working copy (without committing any binary information) if no add operation is done on this binary information.

For example, an executable of code DRAGON v5.0.1 with its documentation can be constructed using
svn checkout file:///usr/local/Version5_beta/tags/v5.0.7 Version5_wc

```
# build the DRAGON executable
cd Version5_wc/Dragon
make
cd ../..
#
# build the DRAGON documentation
cd Version5_wc/doc/IGE335
./install
cd ../../..
```

The installation of DRAGON5 include the installation of GANLIB5 and TRIVAC5. Similarly, the installation of DONJON5 include the installation of DRAGON5. Note that a generic install script is also available as Version5_wc/script/install to compile the Fortran sources. Each documentation directory has its own install script.

A test-case present in Version5_wc/Dragon/data/ can then be executed using

```
cd Version5_wc/Dragon
./rdragon iaea2d.x2m
```

The directory Version5_wc/Dragon/data/ is containing *non-regression test cases*. They can be executed using the makefile:

```
cd Version5_wc/Dragon
make tests
```

CONCLUSIONS

Procedures presented in this paper are used since the beginning of the development of DRAGON4 in 2004. We expect to upgrade our QA plan around February 2023 and make the transition from this Subversion (svn) approach towards an official GitLab repository hosted at the NEA/OECD Data Bank.

REFERENCES

[1] MARLEAU, G., HÉBERT, A. and ROY, R., "A user guide for Dragon Version5", Report IGE-335, Polytechnique Montréal (2021), 334 pages.
[2] HÉBERT, A., "About Version5", https://www.polymtl.ca/merlin/version5.htm (2021).
[3] HÉBERT, A., "Development procedures for Version5 of reactor physics codes", Report IGE-374, Polytechnique Montréal (2020), 13 pages.
[4] COLLINS-SUSSMAN, B., FITZPATRICK, B.W. and MICHAEL PILATO, C., "Version Control with Subversion," O'Reilly Media Inc., USA, June 2004. See: https://subversion.tigris.org
[5] SCOTT, B.A., "PySVN - The pythonic interface to Subversion". See https://pysvn.sourceforge.io