Validation, Verification and Testing of Software and Models in FLR

Iago Mosqueira
Cefas

September 2008

Contents

Introduction

Quality of advice in fisheries and natural resources management is tightly coupled with the quality of the mathematical and statistical methods chosen and of the software implementation used. When newly developed software is applied to fisheries problems, assurances are needed on its quality, stability and fidelity at implementing a given statistical model. While formal certification schemes are in place in many fields of application of statistical software, most notably

in critical situations such as medicine, epidemiology or engineering, they have been almost absent in natural resources management and fisheries science.

The FLR initiative [1] has developed a number of mechanisms and protocols for the implementation of software testing and validation at various levels. Building on the systems already in place in the R statistical language, and on well-stablished technologies like Unit Testing[2], we intend to provide a range of tools that will simplify the implementation of testing suites and protocols.

This documents present the current state of testing and validation in the FLR software, reviews a number of key concepts in the field of software Validation, Verification and Testing (VVT), and suggests a number of best practices and protocols that would enhance the quality and testability of fisheries software developed using R and FLR.

1 Validation, Verification and Testing (VVT)

Model and software VVT includes a range of strategies, protocols and mechanisms to ensure that a model is appropriate to the problem at hand, and that the software implemention behaves as expected under a known range of situations. The Federal Drugs Administration (FDA, 2000), defines validation as: "Establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality attributes".

Validation is thus considered a process that is part of the whole model-building exercise, while verification is an activity performed at various steps in the lifecycle within the validation farmework. Software testing is here a form of verification that it is an integral part of the development routine.

FLR depends and is based on the R system for statistical analysis and graphical output, so the accuracy and replicability of most basic operations is assured by the thorough quality standards of the R system. The R Foundation has recently requested formal certification of compliance with FDA guidelines for use of R in clinical trials [3].

Both R and FLR facilitate the verification of both procedures and results by adopting Open Source licenses, in these cases the GNU General Public Li-

1. http://flr-project.org
2. http://en.wikipedia.org/wiki/Unit_test
3. http://www.r-project.org/doc/R-FDA.pdf

cense 2.0 [4]. Source code is open to scrutiny through an open repository, where all changes and additions can be followed in time, thus providing a complete track record of software development.

In addition to this, the FLR initiative has adopted a series of informal testing procedures that attempt to minimise the disruptions caused by changes to the basic software on user-developed code, ensure stability and replicability of results, and improve coherence among methods in FLR and with those in R on which extra functionalities are built.

1.1 Unit testing

To simplify and homogenize all testing efforts inside FLR, the Unit Testing paradigm has been adopted, and is currently being implemented for all packages. In essence, unit testing aims at verifying that individual units of code work as expected. The smallest testable section of code is considered to be a unit.

As FLR is completely based on the S4 object-oriented framework available in R, the smallest testable unit is the method. Both new generics and overloaded methods are tested using the range of tools presented below (see 2.2).

1.2 Open source

In depth inspection and testing of complex software requires validation and testing of individual components. Details of implementation, including the order of certain calculations, fitting and sorting algorithms used and options chosen for them, or how exceptions are handled, can only be fully evaluated if the source code is open to inspection.

By developing an open-source project based on an open-source platform, FLR is open to scrutiny by any interested party down to the very basic operations and calculations.

2 Sofware testing tools in R and FLR

2.1 R package testing

A comprenhensive mechanism for testing packages has always existed in R, through the `R CMD check` command. Packages are checked for compliance with a

4. `http://www.gnu.org/licenses/gpl-2.0.html`

variety of tests [5]. Source code and documentation files are tested for correctness, and all script files inside the `tests` folder are executed. This folder in the package hierarchy [6] stores R scripts used for testing, and its contents are only executed through the `R CMD check` command. If available, the objects generated by those scripts are compared with a set of results stored in the same folder in R binary format. Examples provided as part of the help pages are also executed. See the Writing R Extensions manual for further details.

2.2 FLtest

A simplified implementation of Unit Testing is present in the `FLCore` package, contained in the `R/FLtest.R` file of the source code [7]. The functions there provide a consistent set of basic tests for equality, TRUE/FALSE statements, and whether function and method calls run or fail. In contrast with the basic R test mechanism, test runs do not stop if any test fails. Instead, a note of the result of each test is made on a log file, which should be inspected after each test run. Work is underway for Unit Testing output to be integrated with the main R test log, so a single point can hold all outputs from various tests.

The various comparisons and tests currently included in `FLtest`, are as follows:

`checkTrue` Check whether an operation returns `TRUE`.

`checkFalse` Check whether an operation returns `FALSE`.

`checkIndentical` Check whether two objects are *exactly* equal.

`checkEqual` Check whether two objects are equal within a level of tolerance.

`checkFail` Check whether an operation fails and returns an error.

`checkRun` Check whether an operation runs without any error or warning being thrown back.

2.2.1 An example test script

The following commands are part of the test script for the `FLQuant` class in the `FLCore` package. Each test tries to answer a question, which is presented here before the actual code:

- Is the prototype FLQuant correct?

---

5. `http://cran.r-project.org/doc/manuals/R-exts.html#Checking-packages`
6. `http://cran.r-project.org/doc/manuals/R-exts.html#Package-structure`
7. See latest version at `http://flr.cvs.sourceforge.net/flr/FLCore/R/FLtest.R?view=markup`

  - `checkTrue(validObject(new('FLQuant')))`
- Does the creator return a correct object?
  - `checkTrue(validObject(FLQuant()))`
- Is there any difference between the two?
  - `checkIdentical(new('FLQuant'),FLQuant())`
- Does the FLQuant creator add the units?
  - `checkIdentical(units(FLQuant(unis='kg')),'kg)`
- Does the FLQuant creator acept a vector?
  - `checkRuns(FLQuant(1:10))`
- Does the FLQuant creator refuse to accept a list?
  - `checkFails(FLQuant(list(a=1, b=2)))`

3 Model validation

3.1 An example: FLXSA

The FLXSA package in FLR implements the eXtended Survivors Analysis (XSA) assessment method, commonly used in European stocks.

A set of assessment runs for three stocks (North Sea cod, North Sea herring and Irish Sea plaice), were carried out using both the legacy executable code (used historically in Working Groups) and FLXSA. Results were compared and found to be in agreement within reasonable numerical accuracy limits. A total of 16 sets of assessment options were assembled and again run using both software implementations. The results were again in agreement, and the R objects obtained were saved and added to the FLXSA package. These objects form the reference results set against which further runs of FLXSA on the same stocks and model options are compared.

The model validation process itself was thus based on the detailed inspection of results, carried out both by the Working Groups in charge of these stocks and the FLXSA developers, and the cross-validation provided by two different implementations of the same model based on different programming languages.

Questions of the quality of the model itself as it relates to it's ability to represent the reality of nature are not covered by this validation exercise. The FLR platform, however, provides an appropriate mechanism on which experiments on the stability, limits and capacities of the XSA model could be easily tested.

4 Future developments and needs

As the complexity of models implemented using FLR grows, testing and validation is likely to become an important issue. Users of the platform will want to trust the code and minimize the time spent testing their own models. To fulfill those needs, two lines of work could be developed.

First, a protocol covering most testing needs and validation scenarios should be agreed and distributed. Together with some software tools to help in its implementation, this could have a very positive impact by guiding users turned developers into formal testing for their own code, and by ensuring homogeneus testing styles across the FLR software. This will also simplify the transmission of responsibilities in software maintenance between developers.

A second line of work could be based on mechanisms for automatic generation of a common set of unit tests for every FLR class and method. Any part of the previously mentioned protocol where tests can be precisely defined could be subject to an automatic implementation, in a similar vein to the automatic definition of accessor methods currently in use in `FLCore`.

For example, a series of likely basic tests for any given class and its creator method will check if both return a valid object, if they handle arguments and options in the same way, if slots are filled with the information provided and in the right place, etc. This kind of test is ripe for automation, but only once the protocol that specifies them is well defined, and a number of tests along those lines have been implemented manually and are considered to work properly.

Recent developments in the R world are likely to influence this line of FLR work, for example the development of a doxygen-like engine for R [8], allowing documentation and code to be maintained on a single file, thus reducing the potential for both to become out of phase with each other.

Unit testing is also likely to be more tightly coupled with the R check procedure, by incorporating unit testing output into the log mechanism of `R CMD check`. This would eliminate the need for two log files to be inspected after each test run, and will likely promote the adoption of unit testing in R.

8. `http://roxygen.org/`