# ESP-r Developers' Quality Assurance Checklist
# Last updated: July 6, 2007

## 1. Synopsis

This document outlines the quality assurance testing procedures that you must be complete before contributing your work to the ESP-r archive.

The objectives of these procedures are to:

a) demonstrate to other ESP-r developers and users that your source code additions and modifications function as you expect them to,

b) ensure that the changes you've made behave consistently on all supported platforms, and

c) ensure that your changes do not interrupt the work of other ESP-r users and developers around the world.

*Companion documents*

This document makes frequent reference to other ESP-r documents, all of which are available at *http://www.esru.strath.ac.uk/Programs/ESP-r_central.htm:*

a) The document *ESP-r Coding Guidelines* outlines programming conventions that your code contributions must adhere to.

b) The document *An overview of Subversion for ESP-r Central Users* describes the use of subversion to maintain ESP-r source code.

*Revision history*

This document is under versioning control; suggestions and contributions are strongly encouraged. The troff-formatted source file for the latest version can be obtained at the following url:

*https://esp-r.net/espr/esp-r/branches/development_branch/src/archive/QA_checklist.trf*

## 2. Placing your code under versioning control

In this section, you'll commit your code to your sub-branch, and update your branch to reflect recent changes in the repository.

1. If you have not already done so, commit your code to your sub-branch following the directions in *An Overview of Subversion for ESP-r Central Users*.
2. Move to a new directory, and checkout a fresh copy of your sub-branch. Compile ESP-r and be sure to include support for XML output. Ensure that no compilation errors are reported. If compilation errors are reported, check to see that all of your modified files have been correctly committed to your sub-branch.
3. Synchronize your branch with the development branch, according to the directions in the document: *An Overview of Subversion for ESP-r Central Users*. **IMPORTANT:** Use a clean (that is, unmodified) copy of your branch when synchronizing with the development branch. If you commit your contributions along side revisions from the development branch, the archivist will be unable to incorporate your changes into the development branch!
4. Resolve any conflicts reported during the merge, and compile your updated branch with support for XML output enabled. Ensure that no compilation errors are reported.

## 3. Code standards

These steps will help ensure your code meets standards for inclusion in the ESP-r Archive, and that your modifications don't conflict with recent contributions made by other contributors.

5. Inspect your code for consistency with the *ESP-r Coding Guidelines*.
6. After each commit to your sub-branch, you will receive a test report informing you if ESP-r can compile in its current state, and highlighting any dubious FORTRAN syntax introduced by your last revision. *Read these reports carefully:* you must address any new errors or warnings appearing in the static-analysis portion of the report.

**4. New features and modifications**

These steps will ensure your code behaves as you intended, and that other ESP-r users can use your code with confidence.

7. Select a test case from the test-suite (in the **./tester/test_suite** folder) that exercises the source code you've modified, or if you've added new features to ESP-r, create a new test-case exercising your modifications. Be sure to create a simulation preset within the test case named *test*.
8. Exercise the ESP-r binaries you've modified using this test case on your platform of choice. Ensure your modifications behave as you expect.
9. If your modifications include significant new features (such as a new model), document these results in a written report. Conference papers, theses, reports, and ascii files are all acceptable forms of documentation provided they show your model performs as expected.
10. If your modifications include additions or changes to ESP-r's interface, throughly exercise the menus you've created. Attempt to test every possible combination of inputs and, if possible, ask a colleague unfamiliar with your changes to test the menus.

**5. Automated Testing**

The automated_tests.pl script (in the tester/scripts folder) will help you identify errors in ESP-r introduced by your revisions. automated_tests.pl will test that the version of ESP-r compiles correctly, and determine if your revisions have introduced dubious Fortran syntax or affect simulation results. automated_tests.pl makes use of the tester.pl regression script, and the Forcheck static analysis program. These tools can also be used individually; their use is described in Appendix A.

11. From the tester/scripts folder, invoke the following command:

    ```
    $ ./automated_tests.pl -b development_branch -b <your sub-branch name> -v
    ```

    If you do not have access to the Forcheck static analysis utility, invoke the following command:

    ```
    $ ./automated_tests.pl -b development_branch -b <your sub-branch name> -v --skip-forcheck
    ```

    Be patient—these tests will require several hours.
12. When finished, automated_tests.pl will produce an ASCII report, titled automated_tester_output.txt. This report contains three sections: results from static analysis tests; results from compilation tests; and results from a regression test. Read these sections carefully:
    a) Your modifications should not introduce any new errors or warnings in the static analysis section of the report.
    b) Your branch should compile in all three graphics library configurations (GTK, X11 and no-X).
    c) Your modifications should not introduce any differences into the test results reported from the regression test. If differences are reported, ensure they can be attributed to your intended modifications.
    d) Your modifications should not slow bps down appreciably (review the "dt-CPU (%)" column in the regression test results).

    If the automated tester report identifies errors, address them before submitting your code for incorporation into development branch.

**6. Portability Testing** *(Optional)*

These steps will help ensure your code behaves consistently across all platforms on which ESP-r is supported (currently SUN Unix, SGI, Linux, CYGWIN, MINGW and Mac 4). During portability testing, the predictions of a *test bps* binary are compared with *test bps* binaries compiled on other platforms.

altering ESP-r libraries or installation procedures.

13. Create a results set archive using bps as built from your sub-branch (be sure to enable XML output). The tester.pl script will generate a results set archive for you when invoked with the following command:

    ```
    $ ./tester.pl /path/to/bps --create_historical_archive result_set.tar.gz -v
    ```

13. If you have access to other supported platforms, check-out a copy of your sub-branch on each of these platforms and compile ESP-r. Ensure that you enable XML output and that no compilation errors are reported.
14. Copy your test case results set archive onto each of the the alternate platforms you used in step 7. On each platform, invoke the tester.pl script to exercise your *bps* code and compare its output to the result set archive using the following command:

    ```
    $ ./tester.pl /path/to/bps --historical_archive result_set.tar.gz -v
    ```

15. Review the test report produced by tester.pl (bps_test_report.txt) and ensure that no differences are reported.

### 7. Submitting Your Code to the ESP-r Archivist

These next steps describe how to submit your code for incorporation into the development branch.

16. Prepare a list of the revisions on your sub-branch that comprise the changes you wish to include in the development branch. The list of svn revisions should not include revisions that merge code from development branch. Often, the relevant revision numbers on the branch will be specified as a set of intervals. You can review the complete set of revisions made on your sub-branch using the subversion command:

    ```
    $ svn log --stop-on-copy
    ```

    Note that when specifying an interval of revisions, the first revision must be the revision prior to your commit. For instance, to include revisions 22 and 23 in development branch, you must specify the rang 21–23.
17. Email the ESP-r archivist (Ian Beausoleil-Morrison, Ian_Beausoleil-Morrison@carleton.ca) using the template provided in src/archive/code-contribution.txt.

### Appendix A: Additional testing tools

*Forcheck Static Analyzer*

The Forcheck static analyzer[1] inspects your code for inconsistencies and errors that might otherwise be missed by compilers. To invoke Forcheck, you'll first need to a copy of bps with the debugging symbols included. To do so, answer ''yes'' at the Install script prompts: ''Retain debugging symbols? (y/n) [y]''.

Next, move to the esru folder corresponding to the ESP-r binary you wish to test. For instance, to test *prj*, move to the folder ''esruprj''. To test an ESP-r binary when linked to the X11 graphics library, invoke Forcheck using the following command:

```
$ forchk -I ../include *.F ../lib/esru_ask.F ../lib/esru_blk.F ../lib/esru_libX11.F
```

To test an ESP-r binary when linked with the GTK library, invoke forcheck using the following command

```
$ forchk -I ../include *.F ../lib/esru_ask.F ../lib/esru_blk.F ../lib/esru_libGTK.F
```

Forcheck will produce a report identifying errors in your source. You may expect this report to warn of inconsistencies in other portions of ESP-r, but pay particular attention to portions of the report pertaining to files you've changed.

Forcheck output is generally verbose, and can be even more so if it's not configured to respect language extentions available in modern compilers. A Forcheck configuration file (esp-r.cnf) suitable for use with ESP-r is available in the tester/scripts folder; to use it you must first set the FCKCNF environment variable. Using the bash shell, enter the following command:

```
$ export FCKCNF="/path/to/tester/scripts/esp-r.cnf"
```

[1] http://www.forcheck.nl/

*tester.pl Regression tester*

The tester.pl regression tester will exercise bps binaries over various test cases, and quantitatively compare the results. It can also save test results in compressed archives for use on other platforms. tester.pl can be found in the tester/scripts folder; information on its use is available by running:

```
$ tester.pl --help
```