Structure of the ESP-r Source Code Archive
Version of January 5, 2008

Introduction

A version control system (subversion) is used to facilitate the management of the ESP-r source code archive. The archive holds the current and past states of the ESP-r source code and databases and example models and documentation within a database. The database also supports the concept of different simultaneous versions of ESP-r via separate named branches within the repository.

The version control system provides commands to add and remove and update files within the repository based on permissions established by the Archivist. Individuals in the development community *check out* particular versions of ESP-r or states of ESP-r into local *sand boxes* on their computer and changes made within a sand box may eventually be merged back into the repository and shared with others if it passes a testing regime.

To re-iterate, the repository is held remotely. Developers may have one or more local sand boxes which are created via subversion requests to the repository. Alterations in a local sand box remain local to their computer and unknown to the repository until subversion commands (e.g. *add, delete, move, commit*) are given.

For those who require a non-standard version of ESP-r (e.g. for a particular operating system or computer or a version supporting additional geometric complexity) non-developers may also checkout ESP-r in order to compile it rather than using one of the standard binary distributions. ESP-r is available under the GNU public license and Subversion is the mechanism used to distribute the source code (as stipulated within the licence) as well as ensure that those who alter the code are able to conform to the license requirements to return their changes as a contribution to the ESp-r community.

A schematic representation of the version control system's configuration for archiving the source code is given in Figure 1.

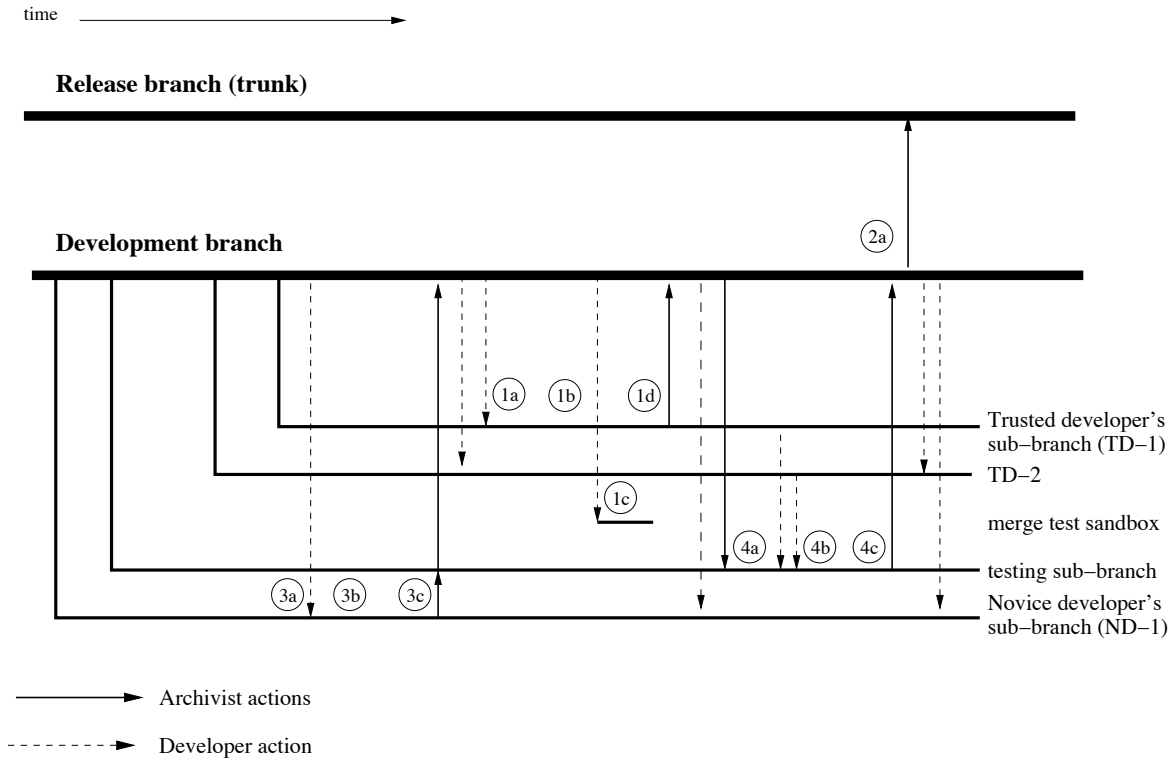

Figure 1: Schematic representation of the ESP-r repository.

The layout of the repository and the management tasks implemented via the use of subversion commands are tightly linked. On first glance the process associated with contributing code is complex. The process is focused on assuring the quality of the ESP-r distribution and identifying potential bugs before they become an issue for the community. This document is one part of a suite of documents that describe how the development process works.

The release branch (in subversion terms the *trunk*) contains the general distribution of ESP-r (source code, databases, example models, documentation and test suite). Any interested party (both users and developers) can, at any time, download the source code representing the latest, fully quality assured version of ESP-r.

Because ESP-r continues to evolve, there is a second branch named *development_branch* which contains the current work-in-progress version of ESP-r. The release branch is updated periodically (about every four months) from the development branch following the completion of a quality assurance procedure (represented by '2a' in Figure 1). This procedure comprises the simulation of a number of pre-constructed models and the comparison of predictions against archived results corresponding to previous releases. The procedure also entails testing on all supported operating systems and several compilers. The details of the quality assurance procedure are given in a separate document.

The released version is accompanied by a summary of changes from the previous version. All ESP-r users have access to the release branch through subversion (all may read, only the Archivist can write to the release branch). The release branch is also the basis for creating the binary distributions of ESP-r for a number of operating systems and computer types.

Working with the repository

Developers who wish to contribute source code should contact the Archivist and request that a branch be created and also to gain access to the *development_branch* (registered users may read and only the Archivist can write to the development branch). Developer-specific sub-branches (e.g. ND-1 and TD-1 in Figure 1) start as a copy of the *development_branch*, and modifications made by a developer remain in the sub-branch until they are *merged* into the development_branch by the Archivist (at which point all others in the development community have access to the changes.

The development community has evolved a number of strategies which enhance working with the repository and others in the community. A few of the strategies are:

- Keep in sync with the development branch.
- Subversion commits should be documented (what issue was addressed, what was changed, what will users notice, what will other developers notice, is it work-in-progress or a completed task, how was it tested, what were the results of the test, what compilers and operating systems were used, results of syntax checks and QA tests run).
- Subversion commits should be atomic. For example, if a common block is altered, change all instances of the common block and, after testing, commit this as one commit. If an example model has also been updated commit this separately from the common block changes. If an equation is changed which will alter predictions commit this change separately.
- If possible compile on more than one platform so that compiler specific issues are identified. If you have access to a syntax checker use this to identify issues before they are committed.
- Take the time to test and debug the code. If an interface dialogue has been updated include interactive tests in addition to the automated test sequence. Open existing models as well as checking that new models or zones or components can be created.
- Back up work-in-progress which has not yet been committed into the repository.
- Periodically review the log of your branch as well as the development branch to ensure that all relevant commits have been accounted for. Annotating a copy of the subversion log file to indicate which commits have been taken and which are pending is a useful technique.

Example of use

The operation of the system is illustrated with the example of the novice developer who 'owns' sub-branch ND-1. Developer ND-1 will perform their development work on their local machine, periodically committing their new and/or modified source code to sub-branch ND-1 in the repository. These periodic code commitments have no impact upon other users/developers as they are strictly available only on the ND-1 sub-branch. Once developer ND-1 feels confident that their code is ready for sharing with the entire ESP-r community, they will perform a number of steps prior to contacting the Archivist.

Firstly, developer ND-1 will synchronise sub-branch ND-1 with the development branch (if they have not already done so). This usually in done via subversion *merge* commands, represented by '3a' in Figure 1. The version control system will facilitate this operation although some human intervention on the part of developer ND-1 will be required if there are conflicts detected during the merge process. If there are no conflicts developer ND-1 should issue a *commit* command immediately after the merge from the development branch. If there are conflicts these should be documented and manually resolved prior to committing the merge. As noted in the strategies above, the prudent developer will synchronise frequently with the development branch to avoid coding conflicts with colleagues.

Developer ND-1 will then ensure that their coding adheres to the ESP-r coding guide (see separate document) and that the necessary testing has been performed (as outlined in the separate quality assurance guide). Depending on the computer the standard *tester.pl* tests in the *tester* folder can take several hours to run. There are also test scripts in the *validation/benchmark/QA/model/cfg* and *validation/benchmark/QA/model1.1/cfg* folders which take only a few minutes to run. This is represented by '3b' in Figure 1.

If the report generated by the testing script indicates no differences with the *development_branch* then the new contributions are ready for the next step. If there are differences reported then discussions must be held with the Archivist to determine whether the differences are expected or have identified an error.

If there are no differences in the test then the novice developer would send an email to the Archivist with a summary of the changes to be taken into the development branch and a list of the revisions to take and the name of the branch. This summary will form part of the eventual release notes and should be formatted and checked for spelling.

The Archivist may merge these commits directly into the development branch or use a temporary branch if further testing is required this is represented by '3c' in Figure 1. When the commits have been merged into the development branch an email (based on the summary sent to the Archivist) will be sent to the development community advising others to merge the newly updated development branch into the sub-branches. Note that the developer who supplied the changes should not merge those same changes back into their own branch.

Experienced developers will follow the same procedure as the novice but with additional steps to ease the work of the Archivists ('1a' '1b'). But prior to emailing the Archivist they would take additional steps to identify potential conflicts with the development branch. Developer TD-1 would perform a test merge of proposed changes by checking out the current *development_branch* into a folder on their computer and performing a merge of the TD-1 branch changes into that local copy of the *development_branch*. This is represented by '1c' in Figure 1. If this test merge is successful the Archivist is notified with the standard summary and instructions about what to take into the development branch. The Archivist will then merge these changes into the development branch ('1d' in Figure 1) and notify the development community.

If the test merge results in conflicts then these must be resolved and separately committed before the Archivist will take in the code. Some conflicts are difficult to merge into the development branch and an alternative procedure may be used. This involves freezing the development_branch for a brief period, performing a high-level merge of the development branch into the users branch (to make it identical to the development branch) and then merging prior commits within the users branch and resolving conflicts and committing all changes and resolves and a single commit which the Archivist tasks.

Remember, good practice is for the development community to update their own branches as soon as practical with changes in the *develpment_branch*. Failure to update can cause problems when it comes to preparing code for submission to the Archivist. Good practice also includes making a backup copy of local sand boxes prior to doing any major merges.

New release testing

The preparation for creating a new official distribution often requires a sequence of commits from several developers who are involved in carrying out the tests. For this task a temporary branch is often created for use by the testing team ('4a'). Issues identified and corrected by the testers will be merged into the temporary branch ('4b'). Once the Archivist is satisfied with the coding contributions, and the quality assurance tests are run on the temporary branch the code is merged into the development branch ('4c') and a new distribution is created ('2a') and announced.

The time delays inherent in the process can impact development teams who need to rapidly progress work. One approach is to create a branch which several people have access to so that each member of the team can commit and update from changes made within the team. Interactions with the Archivist and other members of the development community follow the standard pattern.

Occasionally a change made in one developers branch is of immediate interest to another. While is possible to merge such a change between branches, cross-branch merges should be clearly documented to ensure that only the originator of the change commits it to the development branch.

Further reading

This document assumes proficiency in the use of the version control syntax of subversion. Subversion is well documented on the web and there are a number of tutorials and books on subversion which the novice developers are advised to read.

To generate an A4 postscript document from this file via the groff suite of tools (available for many operating systems) issue the following command:

cat repository.trf | eqn | tbl | groff -mms -dpaper=a4 -P-pa4 > repository.ps