SPK System Testing Guide

This guide describes a process that can be followed by software developers to perform end-to-end system testing whenever enhancements or modifications are made. This process is critical for maintaining and improving the quality of the system.

The information in this guide is essential to individuals and groups participating in the development of the System for Population Kinetics (SPK), an open source project of the RFPK Laboratory of the Department of Bioengineering of the University of Washington. RFPK's work is supported, in part, by grant P41 EB-001975 of the National Institutes of Health (NIH) of the U.S. Department of Health and Human Services.

Table of Contents

Introduction

Three stages of testing are employed in the enhancement and modification of SPK:

1. Unit testing -- responsibility of the developer who is making the changes and is normally performed on the developer's workstation.
2. System testing -- also the responsibility of the developer making the changes, but is performed in a special test environment. *This stage of testing is the subject of this guide.*
3. Regression testing -- responsibility of the deployment coordinator, who may be a developer or may be operations staff. This is an automated test in which a set of jobs is run and the results compared with results previously judged to be acceptable. In effect, the regression test insures that the software has *not regressed*. Because this test takes considerable time to run and consumes significant computational resources, it is not routinely run as part of a system test. It must, however, be run before any new or modified software is deployed to the production environment.

SPK is a complex system, comprising many software modules, a number of computers, and several technologies. When any software is modified, it must be tested thoroughly before being put into production. To minimize the risk that modifications would destabilize the system, it is crucial that system-wide testing be performed on exactly the same software and hardware infrastructure as is used in production. It is for this reason that an SPK installation includes both production and test environments. Both are identical in terms of the following infrastructure components:

- Database server (dbserver)

- Web server (webserver)
- Job-queue server (jobqserver)
- Application server for population kinetics (aspkserver)
- Computational server for population kinetics (cspkserver)

One of the key concepts in the architecture[1] of SPK is that of the *job*. With the help of the Model Design Agent (MDA), which is the internet client portion of SPK, a user creates or imports a model and a dataset, combines them as a job, and then submits the job to be processed by the rest of the system. To test the system, it is necessary to have jobs which will exercise portions of the system which have been added or modified. The principal inputs to the system testing process are, therefore, sets of jobs. These jobs come from two sources:

1. The production database, *spkdb*. The utility program **take_snapshot.pl** can be used to copy a set of previously run jobs from *spkdb*, creating files that can be used by another utility, **load_spktest.pl** to initialize the test database, *spktest*, with the selected set of jobs along with all related jobs and data. This is most useful for testing modifications that either fix bugs or enhance existing functions.
2. The MDA. Sometimes new features cannot be tested with existing jobs because those jobs would not have requested the features. In this case, the tester can use the MDA to create new jobs.

Sometimes a test set must be run many times, as the software is debugged. Eventually, all jobs in the set should run as expected. At that point, the tester runs a utility called **stage_candidate.pl**, which saves all the software in the test environment for subsequent deployment to the production environment.

RFPK recommends a policy of deploying modifications to production on a scheduled basis, rather than doing it every time that some software has passed a system test. Instead, a number of deployment candidates will accumulate during the period between scheduled deployments, and it will be the most recent candidate that finally moves into production. This strategy has a number of advantages:

- Users learn to expect the behavior of the system to change on a scheduled basis, rather than randomly. This improves the perception of stability.
- Communication of the changes can occur in an orderly fashion.
- Before deployment, a recursion test should be run to insure that the modifications do not "break" something that worked previously. Scheduled deployments reduce the frequency with which the recursion test must be executed.

The process of deployment is covered in another document, titled *SPK System Deployment Guide*. Some organizations may wish to assign the responsibility of deployment to a different group than the one responsible for system testing, based on the theory that an "arms-length" relationship between development and operations improves quality.

Testing a Deployment Candidate, Step-by-Step

The following procedure should be followed whenever changes are made to any of the server side SPK software.

1. All new and modified software should first be unit-tested in your own workspace.
2. Do *not* commit changes to CVS, at this time. Changes must not be committed to the CVS repository until they have passed a full system test.
3. As explained in the introduction, regression testing is not a routine part of system testing. If you have reason to believe that your modifications might "break" some existing features, you can use the regression testing infrastructure to prepare a small automated test to include as part of your system test. See the manual pages for **regression_test.pl** for details.
4. Obtain exclusive access to the test environment. This must be done through coordination with other developers who might also need this environment.
5. It is convenient at this point to open two new windows: one to *aspkserver*, and the other to *cspkserver*.

   Note: the minus sign in the **su -** commands, below, is necessary.

   In a new terminal window:

   ```
   ssh jobqserver
   su -
   ```

   In another new terminal window:

   ```
   ssh aspkserver
   su -
   ```

   In another new terminal window:

   ```
   ssh cspkserver
   su -
   ```

6. Use the most recent deployment candidate to initialize the test environment; for example, /usr/local/include/spktest/* on all relevant machines.

   In your *aspkserver* root window:

   ```
   deploy_candidate.pl --test
   ```

7. You may want to test the (supposedly) initialized test environment is clean before you add your new modifications. Run the regresssion test suite in the test environment if you prefer.

   Continuing on *aspkserver*:

   ```
   regression_test --ignore-candidate
   regression_test --ignore-candidate --pvm
   regression_test --ignore-candidate --pvm --parallel
   ```

8. Make sure that the test daemons are *not* running.

   In your *aspkserver* root window:

   ```
   spkinit.sh spktest stop
   ```

9. Take a test snapshot of the production database. Pick jobs that are related to the bug fix or enhancement. For example, Job 382 is an individual analysis and Job 390 is a population analysis. You may want to change to a temporary directory.

   Continue as root:

   ```
   ssh dbserver
   cd /tmp
   take_snapshot.pl 382 390
   ```

   If you do not want any jobs that have previously run, use **take_snapshot.pl** without providing any arguments. No jobs will be copied from *spkdb*, but the files to build a valid *spktest*, containing no jobs, will be created.

10. If your changes require the modification of the structure of the database, you will need a database modification script. An example [2] of a modification script can be found in the *MySQL and SPK* howto document. Place the script, which should be named `modscript.sql` in the current directory on *dbserver*, along with the `basedata.sql`, `schema.sql` and `userdata.sql` files that were just created by **take_snapshot.pl**.

11. Create a test database from the `.sql` files that were created by **take_snapshot.pl** and, only if you are modifying the structure of the database, `modscript.sql` file.

    Continuing on *dbserver*:

    ```
    load_spktest.pl
    ```

    You now have a test database. If you need to add additional jobs to exercise new features, go to the test instance of the web site and use the MDA to create the additional jobs you need and to submit them. You can find the URL to use in Appendix C.

12. If you have added jobs using the MDA, make a backup copy of your test database, in case you need to run your tests several times before your modifications work correctly.

    Continuing on *dbserver*:

    ```
    dump_spktest.pl
    ```

13. Transfer source code to the target host(s). Consider these two distinct cases:

   a. *Source that you have added or modified* should not be committed to cvs until after a successful system test, hence cvs update cannot be used for the transfer. *Use scp instead of cvs.*

   b. *All other source required for your build* should be updated by using the command **cvs update -dP** in your cvs workspace on the target host.

14. Build your software on the target host and install the object code into the correct test directories (see Appendix B).

15. Start the Job-queue server and the daemons, so that the jobs will compile and run.

    In your root window to *aspkserver*:

    ```
    spkinit.sh spktest start
    ```

    In these terminal windows, you can verify that the daemon you started there is actually running

    On *aspkserver*

    ```
    ps -efl | grep spktest
    ```

    should show you that **spktest/spkcmpd.pl** is running.

    On *cspkserver*, the same command should show that **spktest/spkrund.pl** is running.

    In the system log, the daemons record many important events, including their own starting and stopping, but also the beginning and completion of jobs and the occurrence of errors. In each of your two windows, you can view the latest 40 messages in the system log with the following command:

    ```
    tail -n 40 /var/log/messages
    ```

16. Verify the output.

    a. Check the status of your jobs by examining the database. As an ordinary user on your workstation, open up a window and run the MySQL client on any machine:

    ```
    mysql -hdbserver -ureader -preader
    use spktest;
    ```

    Periodically check the status of jobs in the test database:

    ```
    select job_id, state_code, end_code, abstract from job;
    ```

    When the value of *state_code* for a given job changes to *end*, the job has completed.

    b. If a job completes with the value of *end_code* equal to *srun*, no faults or exceptions were detected, and you can skip the next step.

    c. If *end_code* is equal to any value other than *srun*, you should check the system log for error messages and the working directory for additional evidence of what went wrong.

    You can determine whether the job terminated on *aspkserver* or on *cspkserver* by consulting the *history* table in the test database.

    In your database client window:

```
select * from history where job_id=n;
```

where *n* is the *job_id* of the job that has terminated. The *host* field will have a value that is the host name of the computer on which the event occurred. Look for the host on which the final event for the given job took place.

Now you can check the system log by executing **tail** in your root window to the server on which the job terminated:

```
tail -n40 /var/log/messages
```

d. Now you can also examine the contents of the working directory, which will be named `/tmp/spkcmptest-job-n`, if the job terminated on *aspkserver* or `/tmp/spkruntest-job-n`, if the job terminated on *cspkserver* (where *n* is the job_id number of the job.) The working directory contains inputs, outputs, source code and object code.

e. Now you may want to run a comprehensive list of system tests, which may take up to 1 hour to complete. This is not mandatory but highly recommended to make sure your will-be candidate is in a correct state.

   On *dbserver* as a root (i.e. 'su -')

```
regression_test --ignore-candidate
regression_test --ignore-candidate --pvm
regression_test --ignore-candidate --pvm --parallel
```

   The option "--ignore-candidate" forces the system to retain the installation of your modifications.

f. Log into the test web site (find the URL to use in Appendix C). Use the MDA to examine the output.

17. Stop both test daemons and Job-queue server when the database shows that all your jobs have *end* as the value for *state_code*.

    In your root window to *aspkserver*:

```
/etc/rc.d/init.d/spkcmptestd stop
```

    and in your root window to *cspkserver*

```
/etc/rc.d/init.d/spkruntestd stop
```

    and in your root window to *jobqserver*

```
/etc/rc.d/init.d/jobqtestd stop
```

18. If the test fails and you want to make a few changes and try again:

    a. Restore your test database.

       In the same directory on dbserver where you made the backup copy of your test database

```
load_spktest.pl
```

   b. Modify your source code.
   c. Return to step 10.

19. If the test fails and you do not want to make changes and retest immediately:
    a. Inform the other developers that the test environment is now available.
    b. Exit this step-by-step procedure.

20. If the test is successful:
    a. You can now commit your additions and modifications to the cvs repository.
    b. Add your candidate to the list of deployment candidates.

       In your root window to *aspkserver*:

       ```
       stage_candidate.pl
       ```

       You may want to write down the name of the release-notes file created by **stage_candidate.pl**, because you will need to edit it.

    c. Inform the other developers that the test environment is now available.
    d. Edit the release-notes file created by the execution of **stage_candidate.pl**
    e. Exit this step-by-step procedure.

Appendix A: Production Files

Jobqserver

All files in the following directory:

```
/usr/local/bin/spkprod
```

Aspkserver

All files in the following directories:

```
/usr/local/bin/spkprod
/usr/local/lib/spkprod
/usr/local/include/spkprod
```

Cspkserver

All files in the following directories:

```
/usr/local/src/spkprod
/usr/local/bin/spkprod
/usr/local/lib/spkprod
/usr/local/include/spkprod
```

Webserver

```
/usr/local/tomcat/instance/prodssl/webapps/user.war
```

Appendix B: Test Files

Jobqserver

All files in the following directory:

```
/usr/local/bin/spktest
```

Aspkserver

All files in the following directories:

```
/usr/local/bin/spktest
/usr/local/lib/spktest
/usr/local/include/spktest
```

Cspkserver

All files in the following directories:

```
/usr/local/src/spktest
/usr/local/bin/spktest
/usr/local/lib/spktest
/usr/local/include/spktest
```

Webserver

```
/usr/local/tomcat/instance/testssl/webapps/user.war
```

Appendix C: Test Web Site

The following URLs are those used for the SPK demonstration site operated by RFPK. At other SPK installations, the domain names will most certainly be different, but the IP addresses to be used behind the firewall may be the same, depending on whether or not RFPK practice was followed in setting up the web site.

There are two instances of the web application running on the same machine and using the same web server software. For production and test, the two URLs for access from the public internet are, respectively,

http://spk.rfpk.washington.edu [3]

http://spk.rfpk.washington.edu:8080 [4]

For access from behind the firewall, the corresponding URLs are

https://192.168.2.2/info/ [5]

https://192.168.2.2:8443/info/ [6]

Copyright Notice

Notes

1. ../../whitepaper/overArch/overArch.html
2. ../../howto/rhel3/mysql/mysql.html#example
3. http://spk.rfpk.washington.edu
4. http://spk.rfpk.washington.edu:8080
5. https://192.168.2.2/info/
6. https://192.168.2.2:8443/info/
7. http://www.opencontent.org/openpub/