SPK System Deployment Guide

This guide describes a process for placing new versions of the System for Population Kinetics (SPK) in production. It is meant to be used in conjunction with the process described in the SPK System Testing Guide [1].

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

The Resource for Population Kinetics (RFPK), in addition to its roll as an academic research group and the principal developer of the System for Population Kinetics (SPK), operates a demonstration installation of the software. This installation runs every day, around the clock, providing service to selected researchers. It is a test-bed not only for the software but also the processes that support a production-quality service.

In the life-cycle of SPK, deployment is the stage which moves a minor version of the software into production. The entire process of producing a new minor version is the following:

1. Designing, programming and unit-testing an enhancement or modification.
2. System testing, as described in the SPK System Testing Guide [2]. The output of the system testing phase is a *candidate* for deployment.

   A candidate is *staged* for deployment but is not, in general, deployed immediately. Candidates accumulate according to the rhythm of software development. Deployment occurs according to an operations schedule.
3. Deployment, which performs a *regression test* on the most recent candidate and, if the test is successful, moves the new minor version into production. The purpose of the regression test is to insure that the software continues to run everything correctly that it was able to run before the change; in other words, that the software has *not* regressed.

Regression Testing

As explained in the Introduction, the purpose of the regression test is to demonstrate that the software has *not* regressed. The test is run by issuing the command **regression_test** [3], which is a shell script that, in turn, calls the perl script **regression_test.pl** [4]. A successful test returns zero as the value of the exit status.

The regression test consists of two parts, which will now be described.

Client-Side Regression Test

This test has not yet been developed.

Server-Side Regression Test

The server side of SPK goes into action when the client program, known as the MDA, submits a job. The responsibility of the server side is to process the job and to return results to the client. A regression test for the server side consists, quite naturally, of a set of jobs along with their expected results. The jobs are run and the results compared with what was anticipated. If there are no significant differences, the test is a success. The entire process of extracting the set of selected jobs from the database, of running them, and of comparing results is handled automatically by regression_test.pl

The jobs to use are specified for regression_test.pl in a configuration file. Here are some guidelines for selecting jobs:

1. Jobs should be good representatives of the sort of jobs that scientific researchers create, but they should be selected or modified so that they do not take very long to run. As new features are added to SPK, the set of jobs in the regression test increases in size, and so does the quantity of resources consumed each time the test is run.
2. Coverage should be strived for. Every computational behavior of SPK should be exercised by at least one job.
3. Jobs which produce errors or fail to converge should be included, to test that error handling capabilities have not regressed.

Deploying the Candidate

Once the candidate has passed the regression test, the installation of the software on the servers where it is required is handled automatically by the **deploy_candidate.pl** [5] perl script.

Publishing Release-Notes

As explained in the Introduction, software engineers develop a sequence of deployment candidates between actual deployments of the software to the production environment. Each candidate represents a potential minor version. Each candidate could be one of those that is deployed.

Whenever a software engineer stages a candidate, it is her or his responsibility to write a brief set of release-notes describing what changes are included in the candidate. The software engineer does this by editing a text file that is created, in skeletal form, by the **stage_candidate.pl** [6] perl script.

When a candidate is deployed, all of the release-notes files accumulated since the last deployment are concatenated to form the release-notes for the deployed minor version. This is done automatically by running the **release_notes** [7] shell script.

Deploying a Minor Version, Step-by-Step

1. There must be at least one candidate staged for deployment. For more details, see the SPK System Testing Guide [8].
2. Obtain exclusive use of the test environment, by negotiating with the other users of this facility.
3. Prepare for the regression test by logging into *aspkserver* as root.

   ```
   ssh aspkserver
   su -
   ```

   If the changes to your software require structural changes to the database, you must have a database modification script present in the current directory. You will have already used this script for unit testing and system testing. An example [9], in the "MySQL and SPK" howto document, shows how to create such a script.

   Run the regression test.

   ```
   regression_test
   regression_test --pvm
   regression_test --pvm --parallel
   ```

4. If the regression test runs successfully, skip to the "After the regresssion test" step.
5. The regression test may have failed because of formatting changes in the output files. If you decide that all errors were due to this cause, run

   ```
   recalibrate_regression.pl
   ```

   and return to step 3.
6. The descrepancies for some jobs may have been due to formatting changes, but others may be due to real errors in the code. Create a configuration file for calibrate_regression.pl, which lists only the jobs for which the errors were due solely to formatting changes, then run

   ```
   recalibrate_regression.pl --config-file="name-of-file"
   ```

   then proceed to the next step.
7. If any of the discrepencies were real, they must be corrected, unit tested, system tested and this procedure repeated from the beginning. *In the meantime, no other candidates should be staged.*
8. **After the regression test**, *if the modifications being tested do not involve any structural modification of the database* , skip to the "Deploy" step.
9. Stop the web server, so that changes to the software and changes to the database can be synchronized.

   ```
   ssh webserver
   su -
   /etc/rc.d/init.d/tomprod    stop
   /etc/rc.d/init.d/tomprodssl stop
   ```

10. **Deploy** the software to the production environment by running

    ```
    deploy_candidate.pl --prod
    ```

Note that without the "--prod" option, the script install the latest candidate into the test environment by default.

11. If your changes did not require any modification of the database, skip to the "Publish" step.
12. Run your modification script against the *production* database. Use **mysql** to verify the changes.
13. If this deployment also requires the deployment of a modified web server application, do that at this time.
14. Restart the webserver.

```
ssh webserver
su -
/etc/rc.d/init.d/tomprod    start
/etc/rc.d/init.d/tomprodssl start
```

15. After production has started, but before any additional testing can occur, *take_snapshot.pl* may have to be modified, especially if the database modifications added to the *job* table either a blob field or any field normally NULL at the time a job is submitted. *Do this before releasing the test environment.*
16. Release the test environment by informing other software developers that the environment is now free.
17. **Publish** the release notes:

```
release_notes
```

This will concatenate the release notes together and copy them to the web server.

To make the web notes accessible, add a url to the software team web page, by editing the file `index.html` in the svn directory `<YOUR_LOCAL_R2_COPY_ON_ASPKSERVER>/r2/src/web`, then executing **make** in that directory as a root. NOTE that this must be done on Aspkserver due to firewalls.

Copyright Notice

Notes

1. ../system_test/system_test.html
2. ../system_test/system_test.html
3. ../../ops/utils/rt_script.html
4. ../../ops/utils/regression_test.html
5. ../../ops/utils/deploy_candidate.html
6. ../../ops/utils/stage_candidate.html
7. ../../ops/utils/rn_script.html
8. ../system_test/system_test.html
9. ../../howto/rhel3/mysql/mysql.html#example

10. http://www.opencontent.org/openpub/