INL/EXT-10-17915

SAPHIRE Change Design and Testing Procedure

Curtis L. Smith

February 2010

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-10-17915

SAPHIRE Change Design and Testing Procedure

Curtis L. Smith

February 2010

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Nuclear Regulatory Commission
Washington, DC 20555

SAPHIRE Change Design and Testing Procedure

This document describes the procedure software developers of SAPHIRE follow when adding a new feature or revising an existing capability. This procedure first describes the general approach to changes, and then describes more specific processes. The process stages include design and development, testing, and documentation.

General Approach

The design effort corresponds to the size and complexity of the change. Developers use a cyclical prototyping design methodology as a means to clarify and refine the change. The prototyping process involves the requestor throughout development. The developers will interact with the requestor(s) both initially and throughout the design and development process to ensure that the change accomplishes the expected goal.

Graded Design

Changes and additions to the software vary from very small bug fixes to significant enhancements and new capabilities. The complexity of a change or addition also varies by item. Therefore, the developers use a graded approach to design. They spend more time and effort on larger and/or more complex changes than on relatively simple items. Areas of changes or bugs also dictate the level of effort. For example, problems in cut set generating are much more important than problems in report areas. Enhancements to cut set generation are researched much more carefully than enhancements to reports.

The frequency and formality of communications with the requestor also corresponds to the size and complexity of the change. This ensures that time and money are spent wisely.

Cyclical Methodology

SAPHIRE developers utilize a cyclical, or whirlpool, prototyping software development methodology. The developers prepare prototypes of a proposed change or system, which can then be evaluated by both the developer and requestor, resulting in the development of a more refined prototype. This iteration process helps to clarify requirements, identify weak areas, and evolve and refine the design. Pictorially, the iteration process resembles a spiraling whirlpool or a target, where with each iteration, the cycle becomes smaller and tighter, until the final goal is achieved.

Design and Development Procedure

The cyclical prototyping methodology requires a starting point, which entails a reasonably clear definition of the initial problem and a general solution. When this has been achieved, the iterative development cycle begins.

Initial Problem and General Solution Definition

The first step in designing a change to SAPHIRE requires that the developers and requestors define and discuss the problem and propose a solution. The developer should gain a broad understanding of the goal of the change, and the requestor should understand in general terms how the proposed solution will accomplish the goal.

At this point, the change will be summarized in a SAPHIRE Change Request Form (see Appendix A), where the problem will be summarized and categorized.

Once a clear definition of the change has been identified, additional items are considered, including:

- When applicable, define the necessary inputs and expected outputs.
- Determine the approximate complexity and level of effort required to accomplish the task.
- Consider how existing code functionality can be leveraged to help accomplish the task.
- Consider potential effects on other parts of SAPHIRE.

Design and Development Iterations

The next step is to prove the concept. This means developing key internal functions as well as a rudimentary interface to access and test those functions. This step serves to test the feasibility of the solution, and helps the designers better understand the problem. The results of this step are used for further discussion between the developer and the requestor. This is considered the first iteration of the prototype. Depending upon the results, the design may be modified and refined. The prototype will be modified or rewritten to reflect the information learned.

Each iteration should improve the functionality of the change to bring it closer to its goal. Successive passes, as the design and prototype stabilize, will incorporate more and more of the following items:

- additional supporting functions
- refined and more complete user interface
- integration into the SAPHIRE user interface
- auxiliary functions to facilitate ease of use

Auxiliary functions are niceties that contribute to ease of use. They vary according to the task, but may generally include such things as customizing, sorting, and/or saving data, generating reports, loading and extracting data between projects, toolbar short-cuts, and individual and bulk processing of data. These types of auxiliary functions are added as time and budget permit, upon approval of the customer.

Depending on the scope and complexity of the task, the requestor and the developer maintain contact throughout the development process. Specifically, the requestor or a designated group of users will be given the opportunity to see, try, and comment upon prototypes at logical points.

As a prototype is refined, it approaches a point where satisfies the solution requirements. At this point, the SAPHIRE Change Design and Testing Checklist (see Appendix B) is completed. Completing this checklist will help assure that a standard list of coding issues have been addressed.

Testing

Developers test changes and new features throughout design and development, as part of the process to complete the task. For development to be complete, the change must be tested for acceptance, and where applicable, tested for each software release.

Acceptance Testing

A change is not considered complete until the results have been tested and found reasonable. Developers and key users will test to see that the change works as expected and is free of defects. Changes and new capabilities will not be generally released until the results are deemed satisfactory and correct.

When the change has been accepted, the SAPHIRE Change Form will be updated to document the completion of development.

Prior to official release of a version, SAPHIRE's automated test suite must complete successfully. The success of the suite is a good indicator that the new change does not adversely affect other areas of the code.

Rarely do changes and bug fixes change the acceptable results of the test. On the unusual occasion when this happens, the target test results are modified to match the new accepted results for future runs. The reasons for the results modification are documented and cleared by an authority on the subject matter.

Maintenance Testing

The SAPHIRE automated test suite was designed to verify core operations, such as generating current event data, and solving for cut sets. When the tests produce expected results, the correctness and stability of SAPHIRE is validated. The tests exercise various features on assorted databases, with substantial overlap on key features to provide added confidence.

The test suite is evaluated against significant changes and new features. New tests are developed to check a new feature when the developer and customer agree that it is appropriate. To develop a new test, a suitable test scenario with a database and validated correct answers must be determined.

Documentation

As changes to SAPHIRE are finalized, a description of the change is documented in several places. The developers describe the change when they check-in the altered source code into the version control library. Upon official release, the change is noted in a text file that is distributed with SAPHIRE.

SAPHIRE has an on-line users manual and technical reference manual. Individual changes to the software are not necessarily reflected in this documentation with each release. Many changes are not applicable to this level of the documentation, but some changes and new features do apply. Minor changes, such as wording changes in a screen shot, or removal of an obsolete feature, do not merit immediate inclusion; however, significant new features warrant timely addition. As priorities, time, and budget permit, when such new features are added to the software this documentation is revisited and updated.

Conclusion

The SAPHIRE development team's graded design process involves the initial definition and proof of concept of the change or new capability. Developers use a cyclical prototyping methodology to flesh out the details of a complete, efficient design, including the requestor and key users at logical points in the process. Knowledgeable users test and accept the change before inclusion into a released version of SAPHIRE. Standard test verification and validation ensures that the change remains in good condition. Smaller changes and additions are immediately documented at a low level; bigger changes are documented in the SAPHIRE user and technical reference manuals as time, budget, and prioritization permit.

APPENDIX A

SAPHIRE CHANGE REQUEST FORM

SAPHIRE Change Request Form Instructions

Bug Fixes

When a bug is reported, gather and record the relevant information about the bug on the form. General information should include bug reporter contact information and program version information.

System environment information such as operating system and available memory and disk information should be collected as well, when it appears this information may be a factor into the error.

The problem should be described in sufficient detail as to allow the programmer to reproduce the error. The programmer may request that the bug reporter isolate the problem as much as possible. When necessary, a database should be provided with step by step instructions on how to reproduce the bug.

Additional documentation to support the problem definition should be referenced as needed.

New feature and Improvements

New features and improvements also require contact information; in this case, it should include requestor and subject matter experts. Here, version and system environment information is applicable only to current standards as determined by customers and project management.

A summary of the change should be described, with additional documentation referenced as needed.

Categorization

As the above information is collected, the problem should be categorized as a major bug, minor bug, improvement, or new feature:

- A major bug is defined as an error that stops the user from completing a task and/or adversely affects the core calculational ability of SAPHIRE.

- A minor bug is defined as an error for which a work around is available, or something that affects less essential areas of SAPHIRE, such as a slight user interface malfunction.

- The improvement category is defined as a change that will represent added convenient to the user. For this category, the change is not significant enough to be considered a new feature. Examples of improvements are minor report enhancements, and replacing or adding smoother user interface options.

- A new feature is defined as a significant additional capability to be added. The scope of a new feature is greater than that of an improvement to an existing feature. Examples of new features include new calculation or uncertainty types, new wizards, and new plug-ins.

Prioritization

The priority of a change will generally correlate with the category of the change. Major bugs are generally the highest priority. Minor bugs and suggested improvements are medium to low priority, depending on the pervasiveness of the problem. Customers and project management together prioritize new features. Major customers such as NRC and NASA receive higher priority over minor customers such as the SAPHIRE user's group.

Status Summary

Some Change Request Forms will not be completed. The reasons should be noted, and may include such things as user error, low priority and or lack of funding.

Work on the change is completed in several stages: implementation, validation, documentation, and possibly additional test automation. The SAPHIRE Change Design and Testing Checklist (see Appendix B) details the standard components of each stage. As each stage of the check list is completed, the information should be reflected in the status summary.

The programmer who implemented the change should identify himself, along with any appropriate summarization of the change.

SAPHIRE Change Request

Control Number:	**Category:**		**Priority :**	
Short Title:	Major Bug		High	
	Minor Bug		Medium	
Date:	Improvement		Low	
	New feature			

Program:	**Version:**	**Database:**

Requestor:		**Phone & Address:**
Company:		

Operating System:		RAM:	
Service Pack:		Free Disk Space:	

Description:

Status Summary

	No change (give reason).

Date Completed	
	Change implemented
	Change documented.
	Change validated.
	New test developed.

Comments:

Programmer:	

APPENDIX B

SAPHIRE Change Design and Testing Checklist

SAPHIRE Change Design and Testing Checklist Instructions

The SAPHIRE Change Design and Testing Checklist consists of four sections: Implementation, Documentation, Change Validation, and Test Suite Candidacy. As each section is completed, it should be noted in the SAPHIRE Change Request Form Status Summary section (See Appendix A).

As the implementation of a change nears completion, the programmer should review and mark the items in the first three sections as *required* or *not applicable*. During subsequent reviews, the programmer should note that an item has been completed by filling in the *date completed* column. Comments should be added to clarify any issues.

Implementation

This section addresses a set of standard implementation issues that should be considered for completeness. These items include such things as user interface, reporting, and load/extract capabilities.

Documentation

When a change is made to SAPHIRE, the developer should consider whether the users manual or technical reference manual is affected. New features may merit a new topic, or additional explanation to an existing topic, while others may affect only screen shots or may not affect the documentation at all. It is not practical to update the documentation with every change; however, it is appropriate to make note of the needed documentation changes at the time of the programmatic changes. Periodically, then, the accumulated changes can be added to the documentation.

Change Validation

As with documentation, it is not practical to run the automated test suite with every change. But it is appropriate to consider the potential affects of the change on the test suite for planning purposes, so that the test scripts can be modified to reflect any changed user interface or output format in a timely manner.

Since the automated test suite must be successfully completed just prior to an official release of SAPHIRE, the validation items related to it will generally be completed at that time.

The items relating to code review and user approval should be should be checked off at the end of the active implementation stage. This is also an appropriate time to begin considering whether or not a new automated test should be developed to test the change. The following Test Suite Candidacy section is designed to help determine that answer.

Test Suite Candidacy

This section is designed to help determine whether or not a new test scenario should be added to SAPHIRE's automated test suite. To be a successful candidate, a feature must be significant and unique enough to merit the expense of developing a new test, and mature enough that appropriate baseline data and results are available to test against. Automated test scripts are cost effective to develop only when the user interface does not change much, because changes to things like button and menu labels, and/or output formats require significant test script rework.

SAPHIRE Change Design and Testing Checklist

Control Number:	**Category:**		**Priority :**	
Short Title:	Major Bug		High	
	Minor Bug		Medium	
Date:	Improvement		Low	
	New feature			

Implementation	Req'd	NA	Date Completed	Comments
Related Functionality: Have the following related items been updated?				
• Reports				
• Toolbars				
• Load/Extract				
• Rebuild				
• Other (specify):				
User Interface: Have the following items been implemented/ checked?				
• Appropriate dialog size				
• High and low resolution dialogs available				
• Standard font used				
• Adequate text space allowed for alternate language				
• Initial dialog display is centered				
• Static text/borders ID's set to unused				
• Tab order is logical				
• Default button appropriate				
• ESC key cancels the dialog				
• Clipboard copies text				
Code Review: Have the following items been checked for completeness?				
• String Table updated?				
• Compiler generated warnings reviewed?				
• Change incorporated into other versions (Version 6 & 7, GEM, API)?				

SAPHIRE Change Design and Testing Checklist – Page 2

Control Number:_____

Documentation	Req'd	NA	Date Completed	Comments
User & Technical Reference Manual :				
• Modify existing topic(s)				
• Modify existing screen shots				
• Add new topic(s)sections				

On which operating systems has the change been tested?	Win NT	Win XP	Win 2000	Win 98	Win 95	Comments

Change Validation	Req'd	NA	Date Completed	Comments
• Solution tested with compiler debug options off				
• Change reviewed and approved by the requestor or knowledgeable user(s)				
• Existing test suite passed				
• Test suite scripts changed to reflect interface change				
• Test suite results format changed (i.e., results order changes, significant digit changes, etc.)				
• New test added (See Test Suite Candidacy below)				

Test Suite Candidacy	Yes	No	Date Answered	Comments
• Do other tests cover the feature?				
• Is the feature *significant* enough to merit a special test?				
• Is the feature *mature* enough to build a special test?				
• Can validated *baseline data* be obtained?				