Spectrum Software, Inc.
11445 Johns Creek Pkwy.
Suite 300
Duluth, GA – 30097
www.spectrumscm.com

Subject: **SpectrumSCM Concepts and Usage**

Issue Date: **April 6th, 2005**

From: **William C. Brown**
corey@spectrumsoftware.net
(770)448-8662

1.0 What is SpectrumSCM?

SpectrumSCM is a truly integrated Source Configuration and Management (SCM) solution that encapsulates the fundamentals of good configuration management, including Version Control, Issue tracking/Change Management, Process Management and Release Management into a single tool. Unlike some competitive products, SpectrumSCM is not a collection of independent CM related tools that have been integrated or interfaced together. In contrast, SpectrumSCM is a truly integrated tool, built from the ground up on a foundation of solid Source Configuration and Management fundamentals. SpectrumSCM's functional components work together to provide the end user with the information and tools that they need to implement solid, industry accepted best practices for Source Configuration and Management within their organizations. SpectrumSCM can easily answer the following questions:

- **What**: *What sources were changed?*
- **When**: *When did the sources change?*
- **Who**: *Who changed the sources?*
- **Why**: *Why were these sources change?*

It's important to understand that most CM tools, even simple VC (version control) tools, can answer the **What**, **When** and **Who** questions, but most tools fail completely to answer the **Why** question, which unfortunately also happens to be one of the most important questions. Without understanding *Why* the sources have changed, it's almost impossible to determine the full scope of *What* has gone into any particular release of a product.

2.0 SpectrumSCM Concepts

SpectrumSCM is an issue based Source Configuration and Management solution. This means that all changes to repository resources are associated with a series of Change Requests (CRs) that are later used to build physical releases of the end product. Change Requests answer all of the *What*, *When*, *Who* and *Why* questions outlined above. Change Requests represent complete units of work that flow through a user defined Software Development Life Cycle (SDLC) toward a well defined end goal.

Figure #1

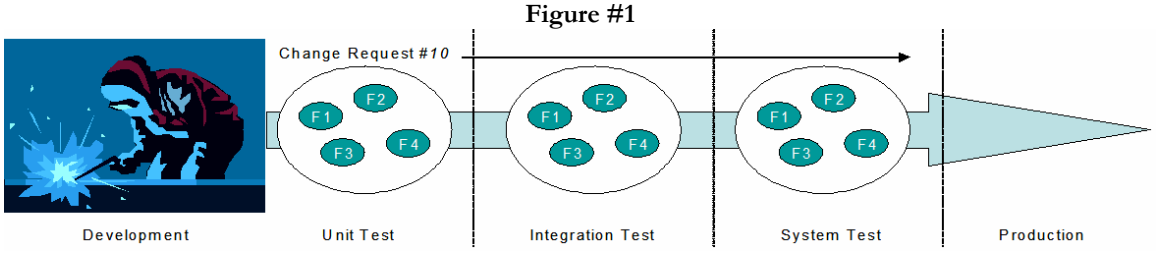


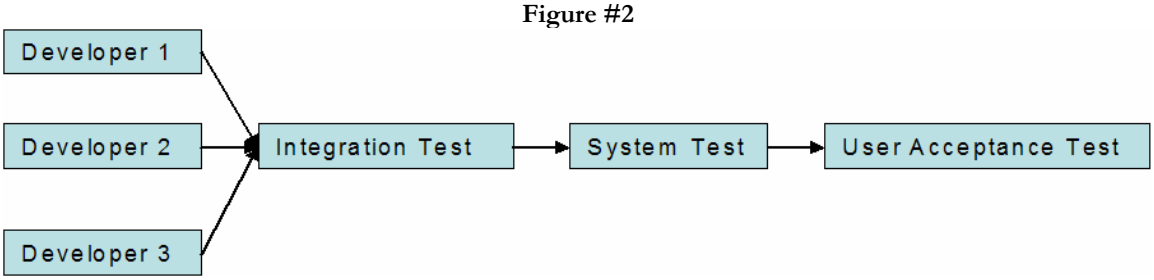
Figure #1 depicts a single developer working within the *Development* phase of a software development life cycle. As the developer completes his/her work, the work is promoted along the SDLC towards the *Production* end

goal. At each phase along the way, the unit of work encapsulated within the Change Request is tested and is either promoted to the next phase or is demoted back to the development phase. The granularity of the work encapsulated within a single CR will change depending on the overall state of the project. For instance, in the early stages of development, the amount of work encapsulated within a single CR may be rather large, e.g. the definition of an entire feature. And in the later stages, such as the testing phases, changes associated with a single CR may only encapsulate a simple bug fix to a single file.

As Change Requests are promoted through the system, they tend to accumulate toward the end of the SDLC. It is at this point that each individual CR may or may not be included within a production release of the system. The decision to include or exclude a CR is left up to the development team or release management but may also be influenced by customer expectations or moving target delivery dates.

In the SpectrumSCM model, the promotion of work along the SDLC is mostly logical instead of physical. This is in contrast to other CM systems that use a branching model to indicate state, as in the following example:

Figure #2



In this example, developers promote code from individual branches into an Integration Test branch. Later the code is then promoted to the System Test branch and then finally to the User Acceptance branch. There are several problems associated with this type of approach:

- **Reliance on physical entities**: While some stages within the SDLC like Integration Test, System Test and User Acceptance Test can be represented by separate physical branches, there are usually other phases within the SDLC that cannot be represented as a physical object. For instance, the SDLC may include phases for CCB (Change Control Board) review, have sign off points for approvals or even integration points with upstream and downstream systems. These types of activities require traceable entry and exit criteria that cannot be represented through a branching model.
- **Reliance on merging**: In the physical promotion model, the end user is required to do several levels of manual merging between the streams. In the best case, automatic merging[1] can be used to merge file content from one branch to another, in the worst case, manual merging is required. The more merging that must take place, the greater the opportunity for error.
- **Lack of Change Management**: Without proper Change Management (issue tracking) there is no way to answer the *What* and *Why* questions. For instance, the model above allows developers and system engineers the ability to see the versions of the files that are available on any particular branch, but the fundamental question of *Why* that work is there and *What* that work actually represents is missing altogether.
- **Lack of file level dependency checking**: Changes promoted from the development branches into the Integration test branch develop interdependencies immediately after the merge/promotion operation has been completed. These interdependencies must be promoted simultaneously or the risk of promoting feature fragments into the later stages of testing becomes highly likely. In the best case, feature fragments can cause the system to fail to build. In the worst case, the system builds and the promotion of untested features can make their way into a production release. In other words, either everything needs to be promoted from one branch to the next or nothing should be promoted at all.

[1] Automatic merging is based off of a line oriented merging technique and is not always 100% accurate. Care must be taken when using an automatic merging facility to verify that the merged code is still semantically as well as syntactically correct.

In the SpectrumSCM model, developers may work together within a single branch line, or may work independently of each other on parallel branch lines. Because the system does not impose a workflow model based on branch hierarchies, the development team is free to use other branching patterns to resolve unique project related problems. A white paper exists on the SpectrumSCM website that details just a few of the branching patterns that users of SpectrumSCM can easily implement. Please see www.spectrumscm.com for more details.

2.1 Change Requests

SpectrumSCM is basically a change management/issue tracking system first and a configuration management system second. The change management system built into SpectrumSCM provides the tool with a solid foundation from which to support proper configuration management.

Change Requests in SpectrumSCM are used to encapsulate work done by developers into individual units of work. This kind of encapsulation can be viewed as a form of Change Set functionality, but change requests are not referred to as Change Sets simply because the functionality provided by SpectrumSCM is far more powerful. Most tools that implement change set functionality do so in an anonymous fashion. In these tools unless the change set is created ahead of time, an anonymous change set is created and the work is assigned at check in time. Change sets are then used to guarantee that the pending action completes in an atomic fashion and that the change set can be used at a later date for simplistic reporting needs. SpectrumSCM takes the concept to the next level and enforces the association of an actual issue to the physical unit of change completed by the developer.

Change requests flow through a user defined development life cycle to one or more end goals. As mentioned above, unlike most other CM systems, change requests are used as input into the release management system. Instead of relying on time sensitive labels, releases in SpectrumSCM are comprised of a series of change requests specifically assigned to a particular release. Change requests can be quickly and easily added to or taken from unlocked releases. This gives the development team an incredible amount of flexibility control over what and when a unit of work is physically assigned to a release.

All work performed on configuration items held within the SpectrumSCM repository is done relative to a CR (change request). A user must have a change request assigned to them before check out for write activities can successfully proceed. Based on the user's role within the project, the user may or may not have the ability to create change requests for themselves. If a user is not allowed to create their own change requests, they must rely on other users such as project leads or senior developers for the creation and assignment of change requests.

Change requests are comprised of the following components:

- **Header**: The CR header is a single line description of the issue itself. In the tool the CR header is usually displayed along with the CR number anywhere where the CR itself is visible.
- **Description**: The CR description is a multi-line textual description of the issue itself. The description is entered by the user during the creation of the CR and can be adjusted at a later date.
- **Attributes/Value**: CR attributes and values are used to capture required information associated with each CR. Each CR attribute can have a pre-defined set of possible values and can be marked as editable so that the user can enter free form text.
- **Attachments**: CRs can have multiple attachments assigned. Attachments can be added, viewed or deleted at any time.
- **History**: Each CR flows through the user defined development lifecycle to one or more end states. As a change request traverses the SDLC, the entry and exit information about each phase that a CR has passed through is recorded directly in the change request itself.

2.2 SDLC and Workflow

As mentioned in the previous section, change requests flow through a user defined SDLC or software development life cycle within SpectrumSCM. Lifecycles are created and held at the system level and thus can be reused as often as necessary. Each project in a SpectrumSCM repository can have a separate and unique life cycle. Once assigned to a particular project, the life cycle can be specialized by adding or subtracting life cycle phases in order to fit the particular needs of the project.

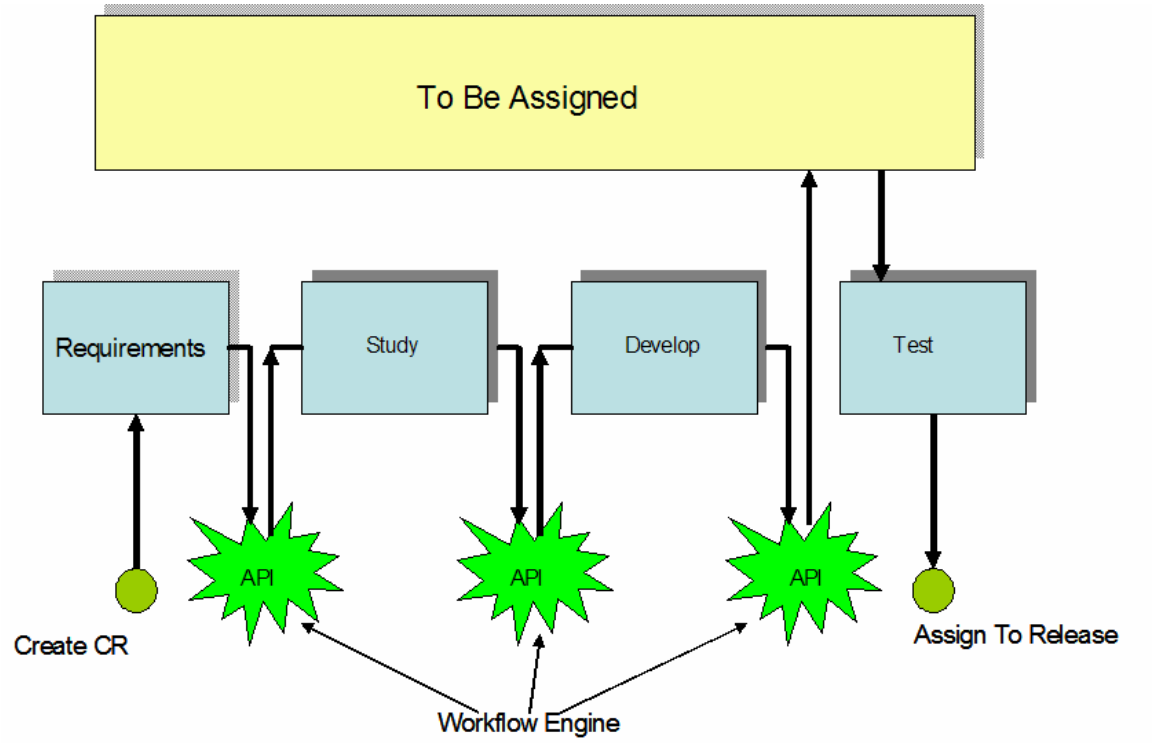
By default, change requests flow through a project's lifecycle in a linear fashion with ad-hoc adjustments. A change request can be created in any phase within a lifecycle and can be assigned to any other phase within the life cycle. As an example, a change request may be created during system testing and assigned back to any other state within the lifecycle, regardless of the connectivity between life cycle phases.

The flow through a life cycle can also be automated to enforce a strict traversal of states within a lifecycle. This is done by simply plugging in one of the supplied workflow engines directly into the SpectrumSCM server. The SpectrumSCM workflow engines are XML driven and can thus be easily customized to work with any defined SDLC. The transition of change requests through the life cycle is controlled by the exit and entry criteria defined in the XML workflow template. End users can use to the template to not only drive the workflow process, but can also use the template to drive external business logic. For instance, the workflow engine can be used to send e-mail notifications based on change request promotions and can also be used to modify change requests attributes and select the next assignment phase and user.

Finally, where the generic workflow engine leaves off, the SpectrumSCM API picks up, i.e. where the general workflow is static, an API defined SDLC can be dynamic. End user organizations can use the SpectrumSCM API to create extremely robust workflow engines and business system connectors. Once created, a workflow engine can be added to a running instance of a SpectrumSCM server without causing a service interruption.

The following diagram illustrates a simple example lifecycle and the insertion points at which a generic or custom built workflow engine would be called.

Figure #4



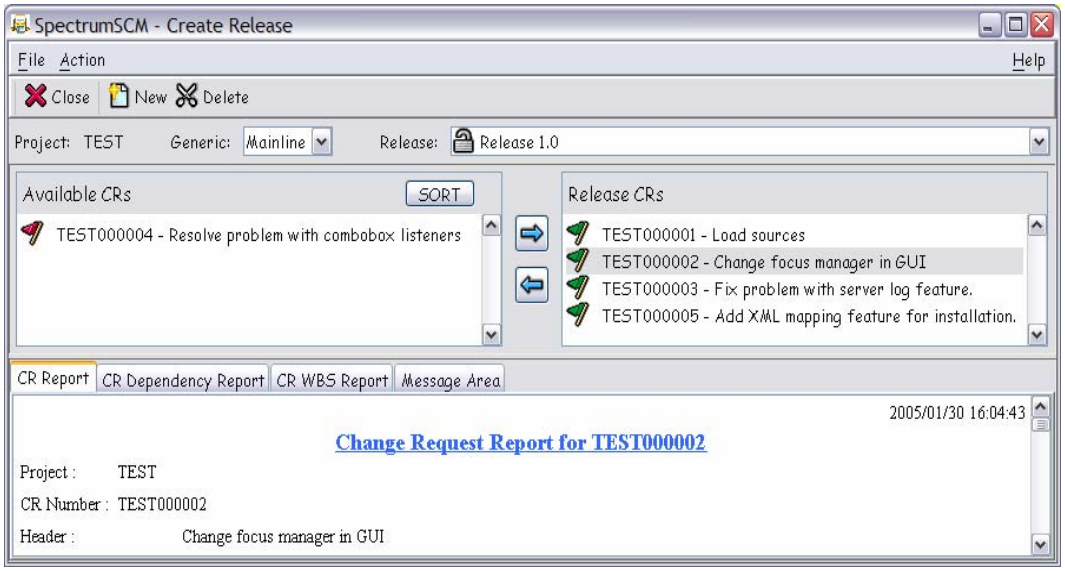
The Yellow "To Be Assigned" state is known as the TBA super state. This state, along with the Completed and Killed states are automatically associated with each user defined life cycle. The TBA state is used to temporarily hold change requests that have been promoted out of a specific state but have yet to be assigned to another state. By default, when a user of SpectrumSCM promotes a change request out of their workspace, the change request is automatically promoted into the TBA super state. This behavior can be changed when a generic or specific workflow engine is applied to the server.

2.3 Release Management

Releases in SpectrumSCM are formed by adding individual CRs into a first class release object. In contrast, most other CM systems rely on a label based approach to associate work with a particular release. The action of applying a label to existing work is a very time ***dependent*** operation. When a label is applied to a repository, it is applied to all files in the repository regardless of the state of those files. At any point in time, some files within the repository will represent work that has been completed and is actually ready to be released, while other files may represent work that has not been completed and is not ready to be released.

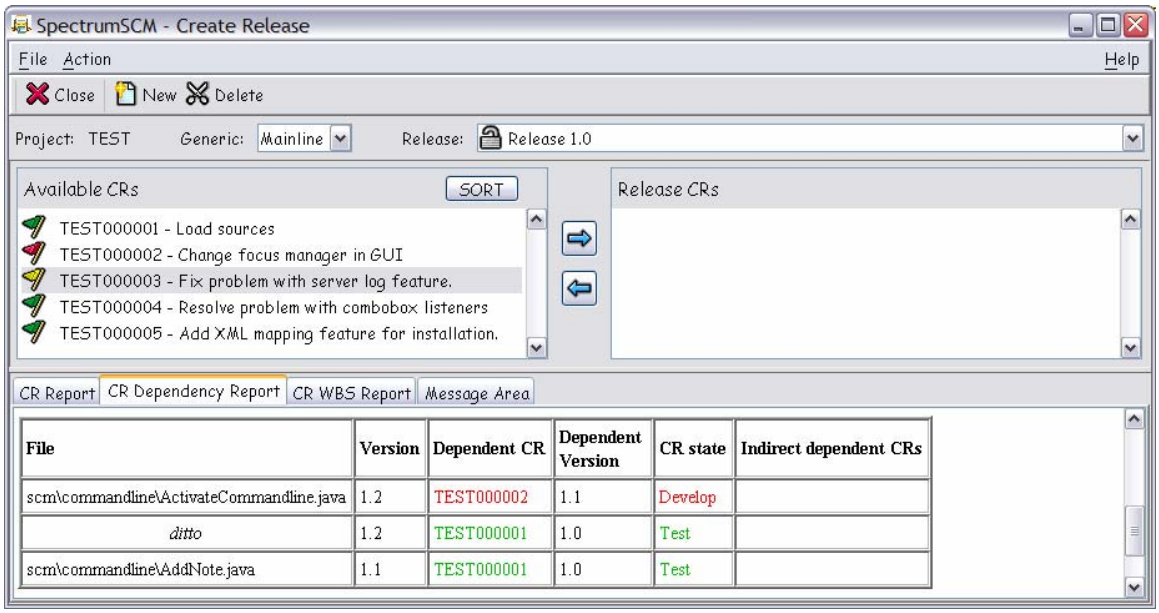
SpectrumSCM's release management system is time ***independent***. Work encapsulated within a Change Request can be added or removed from a release at any time regardless of the state of other work items within the system.

Figure #5



In this example four (4) CRs have already been added to the "Release 1.0" release. Each of the CRs in the release is in a state within the SDLC that permits the Change Request to be added to a release. CR #4, in this case is still in the Development phase and is thus ineligible to be applied to a release. CRs 1,2,3 and 5 are "green flagged" and thus are eligible to be placed into a release. All five CRs represent work that is currently available in the system on a particular branch but at different phases within the SDLC.

Figure #6



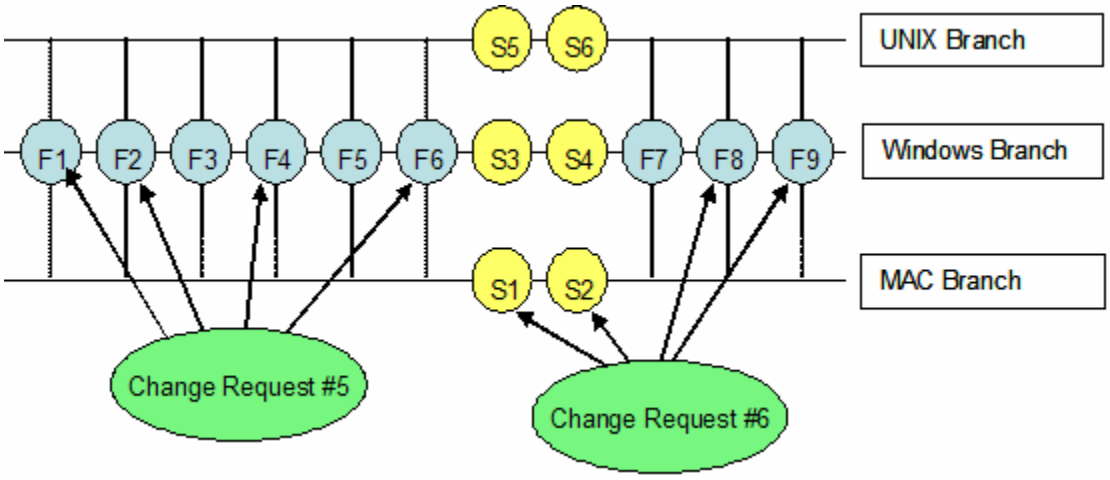
In this example Change Requests 1, 2 and 3 were all used to edit the exact same file. The Release Management facility has detected that there are file level dependencies between the three CRs, and has color coded them accordingly. In this case CR #1 has a green flag status and is actually ready to be included in the release. CR # 2 is red flagged, which is an indication that the CR is still in one of the early phases of the SDLC, and CR #3 is color coded yellow. A yellow flag is an indication that a file level dependency exists between one or more of the CRs in the system. In this example, CR #3 depends on work that was done using CR #2, thus setting up a dependency between the two CRs. The dependency report listing at the bottom of the snapshot shows the files associated with CR #3 and their relationship to other file versions within the repository. Once the dependency has been resolved, CR #3 will be "green flagged" and will be allowed to be placed into the release. Resolving the dependency involves promoting CR #2 into a releasable state.

In all cases, once a series of change requests have been associated with a release, the information concerning that release is available in the form of several reports. The most important of which is the bill of materials report or BOM. The bill of materials report quickly and accurately answers the most important questions concerning the **Who**, **What**, **When** and **Why** of features and bug fixes that have been added to the system. Like all of the reports found in SpectrumSCM, they can be published directly as HTML and thus can be quickly and easily incorporated into internal or external websites.

2.4 *Branching*

Because physical branches in SpectrumSCM don't have to be used to simulate a logical promotional model, developers are free to create a branching model that can actually be used to solve difficult configuration management problems. A prime example of this is the use of branching to support true parallel development. For instance, suppose Company XYZ builds and sells a cross platform editing tool, and this editor is expected to work on at least three different operating systems. The vast majority of the sources that comprise the editor are identical for each supported platform; we can say that these files are ***common*** across the three platforms. There are also a set of files that are specialized for each individual platform that the editor runs on; we can say that these files are ***uncommon*** or have been ***specialized*** for each platform that the editor runs on. When bugs are found in the editor, or new features are to be added to the editor, the changes may all be localized within the common files, or some of them may have to be specialized for each platform. Now suppose that the development of the editor product has been broken up into three individual branches. Each branch supports a particular target platform. With most simple version control tools, the developers make changes to the system by first adding the changes to a particular branch and then either integrating or merging the changes into the other branches. The additional steps required to merge the changes from one branch to the next opens up opportunities for errors, additionally, there is nothing to actually indicate that the changes have actually been merged from the original branch to the other branches. SpectrumSCM improves on this scenario by allowing files to be shared across multiple branches simultaneously. It also allows files to be edited across multiple branches simultaneously. In the example above, when changes need to be added to the editor, if those changes are all localized within the set of common files, then the changes only need to be applied once and are immediately visible on each branch. The Change Request used to make the changes is also immediately visible on all of the shared branches and is available to be placed into a release. There is never any question about whether the files were added to the additional branches, the Change Request information proves that the files have indeed been shared across all of the branches.

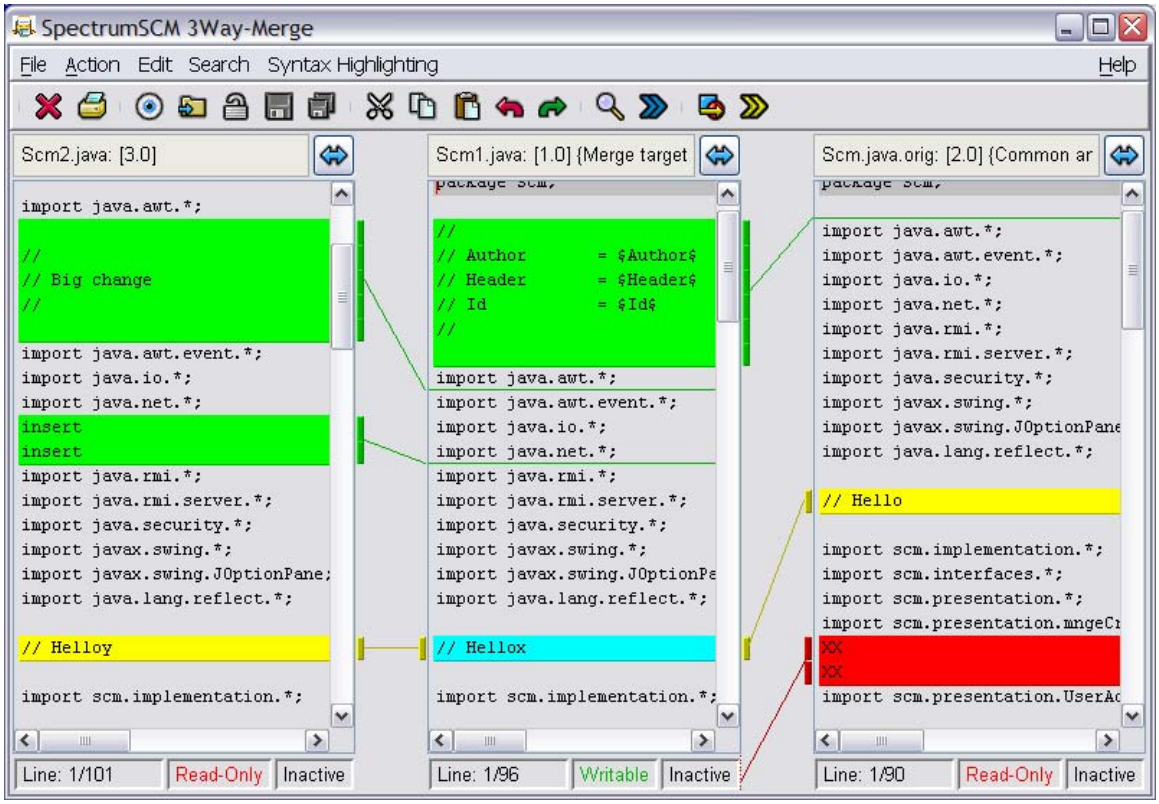
Figure #7



In this example there are three parallel branches. Each branch holds the source resources of the editor for a particular platform. Two Change Requests have been used to edit the files associated with the branches. In this case, CR #5 has been used to edit four (4) files, all of which are common across all three branches. CR #6 was used to also edit four (4) files, two of which are specific to the MAC branch and two of which are common to all of the branches. Both CRs are visible on each branch and can be included in a release. When CR #6 is added to a release on the Windows branch and also on the MAC branch, the release management system within SpectrumSCM determines which files associated with the CR should be extracted for release on each branch. When the release is extracted on the Windows branch, files F8 and F9 are extracted. When the release is extracted from the MAC branch, files S1, S2, F8 and F9 are extracted. CR #6 was used to fix a bug on the MAC branch that spanned all three branches but also required changes to MAC specific files.

SpectrumSCM also supports merging changes from one branch to another, but can also go the extra step to physically combine two separate files. For instance, a simple example is a developer working alone on a set of files within a private sandbox branch. The changes that the developer makes are all uncommon and are thus isolated from the other users of the system. When the developer is ready to share his work with the rest of the team, he has two choices; he can either merge the changes into another branch or he can physically ***re-common*** the specialized files so that they are, once again, shared amongst all or some of the other branches in the project.

Figure #8



There are two ways for a developer to merge or ***re-common*** files. The snapshot in figure #8 depicts the SpectrumSCM 3-way merge tool in action. In this tool the user is able to see the changes to both branch files as well as changes of each branch file relative to the common ancestor of both files. By swapping the positions of the first two edit panes, the user can quickly review the changes to either branch file relative to the common ancestor. Change blocks can be applied to the target editor by right clicking within a change block and then either apply the entire change or just a subset of the change. The tool also supports the ability to automatically merge all of the changes from one editor into the other.

SpectrumSCM supports three different physical branch types. These branch types can be used in multiple ways to implement different branching patterns.

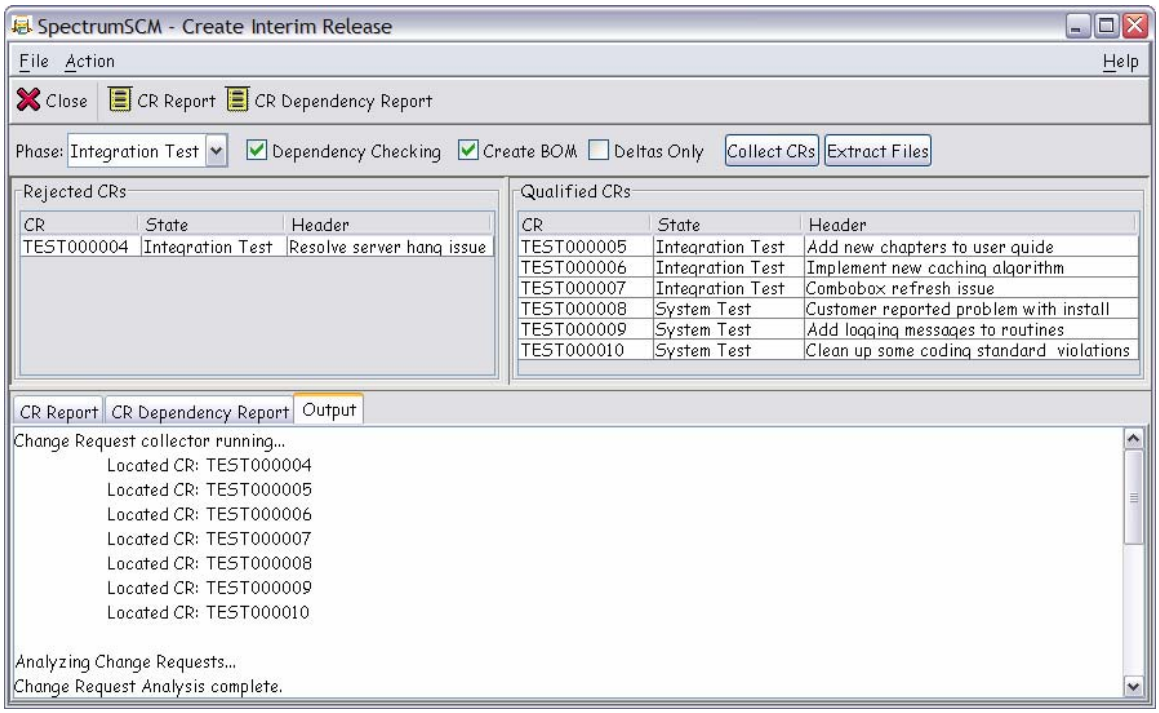
- **Ancestor/Descendent Branches**: This branch type can be used to quickly and easily setup parallel development streams against any existing branch. There is no limit to the number of parallel streams that can be created in parallel to an existing stream.
- **Vendor Branches**: This type of branch can be used to maintain a code base that is related to, but not directly associated with the overall project. As an example, a vendor branch can be used to maintain source or binary distributions of third party libraries.

- **Release Branches**: Release branches are branches formed directly from releases that are already defined in the system. For example, a development organization may need to branch from a previous release of their product in order to patch and then re-release the product.

2.5 Testing Support

The ability to accurately test a software project depends on the CM system's ability to properly identify **What** versions of the configuration items are in each stage of testing and **Why** those items are there in the first place. Simple label based systems cannot provide enough detail to identify the individual items that are supposed to be in integration test versus those items that are supposed to be in system test. In contrast, because SpectrumSCM is an issue based system, the system knows exactly what items are supposed to be in each testing phase, why those items are there, and can easily extract those items into individual testing areas.

Figure #9



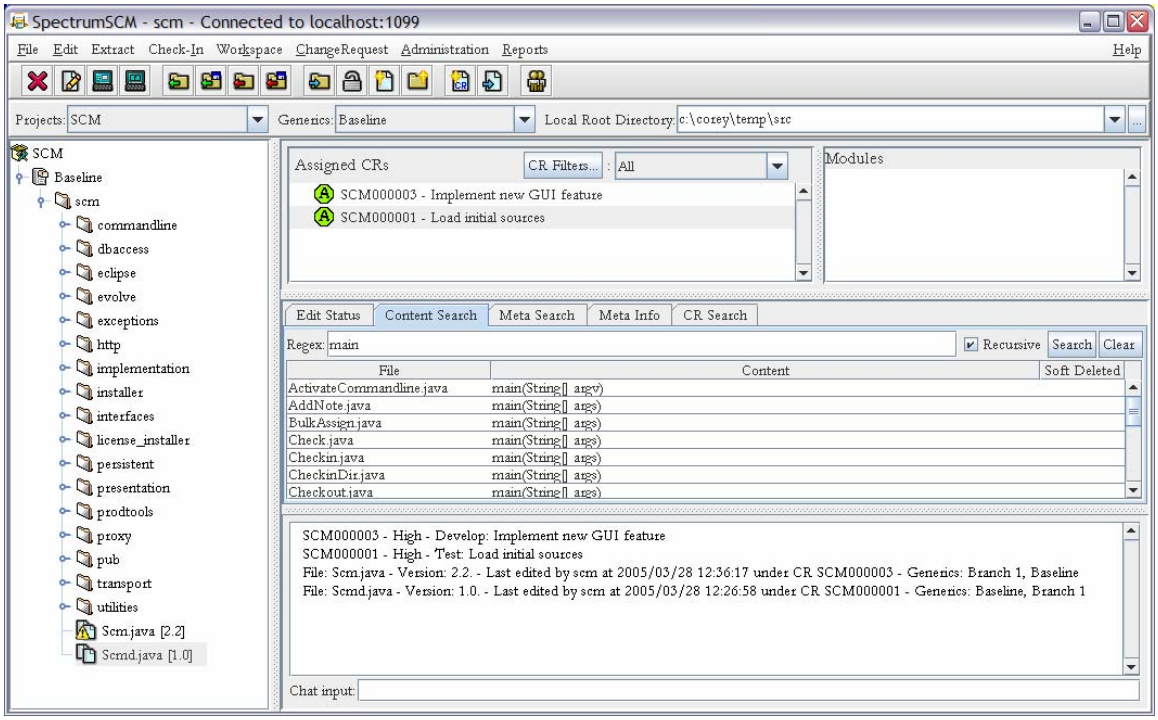
This is an example of the SpectrumSCM Interim Release feature. The Interim Release feature gives QA leads and other testing professionals the ability to extract the correct source versions for each phase of testing into a staging area. In this example the testing lead has entered the Integration Test phase as the base phase for the Interim Release query. When the query is executed, the tool will locate all of the CRs on the current branch that meet or exceed the query criteria. The qualified CRs list in this example contains six (6) CRs that meet or exceed the Integration Test query parameter. On the left hand side of the screen, the system has rejected CR number four (4) due to a file level dependency with another CR that is still in the development phase.
At this point the testing lead could extract the exact version of the system that is represented by this list of qualified CRs and the last full release of the system. When the Interim Release is written out to the filesystem, a bill of materials (BOM) is also written out and contains all of the information about this particular release of the system. This information can then be published to a web site so that all of the other testing team members will know exactly what is in the testing area. By executing multiple queries, one for each phase of testing, multiple simultaneous testing sessions can be executed in parallel.

3.0 SpectrumSCM Usage

The SpectrumSCM GUI can either be installed directly onto a user's workstation or can be accessed directly over the web. Regardless of whether the client is accessed directly from the desktop or over the web, the look, feel, operation and functionality of the tool is exactly the same. This is possible because the SpectrumSCM GUI is 100% Java and runs equally well over the web as it does in a direct installation.

Figure #10 is a snapshot of the SpectrumSCM main display running as a web initiated application.

Figure #10



The SpectrumSCM main display contains the following features:

- **Tree View:** The tree view is the users view into the SpectrumSCM repository. This view displays information containing each files version number, check out status and workspace synchronization status.
- **Active CR List:** The active CR list displays the user's directly assigned change requests. The header information for each change request is displayed just to the right of the change request number itself. The user can obtain a detailed report for each change request simply by double clicking the change request itself.
- **Edit Status and Search tabs:** The edit status and search tabs are located in the center of the screen. The first tabbed area contains an edit status panel and the other tabs contain search panels for file content, meta information and change request information
- **Project Bar::** The project bar is located just above the tree view and contains combo boxes for the user's assigned projects, branches and local workspace.
- **Menu bar and Tool bar:** The menu and tool bar are located at the top of the screen. Most all menu and tool bar functionality is also available in the form of context sensitive menu items.
- **Modules:** Modules allow the user to group files together and associate them with a single name. Later, the user can perform CM operations against this single name as if it were a single file.

Once logged in, the user will be presented with a screen similar to the one in figure #8. If the user already has active change requests assigned to them, they're ready to begin live work against the project. If the user has the proper permissions within his/her assigned role, they may be able to create and self-assign their own change requests.

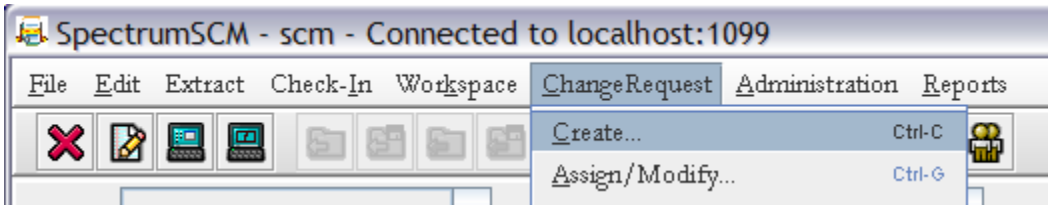
Before file check-out procedures can take place, the user must identify a Local Root Directory (LRD). The local root directory is a location on the users' local workstation where check-out and check-in operations are to take place. When files are extracted from the tool, the path that the file is written to is controlled by the local root directory and the relative path of the file in the file tree. For instance, if the local root directory is set to **"c:\joe"** and the relative path to the file in the tree is **"src\foo.c"**, then the absolute path to the file will be **"c:\joe\src\foo.c"**

3.1 Creating CRs

Before any work can be done on resources controlled by SpectrumSCM, the user must have a change request to perform the work against. Either the user can create change requests for themselves, if they have the proper permissions, or they can have project leads/senior developers create and assign a new change request to them.

To create a new change request, the user can pull down on the Change Request menu item (Figure #11).

Figure #11



Alternatively, the user can press the create change request icon located on the tool bar (Figure #12).

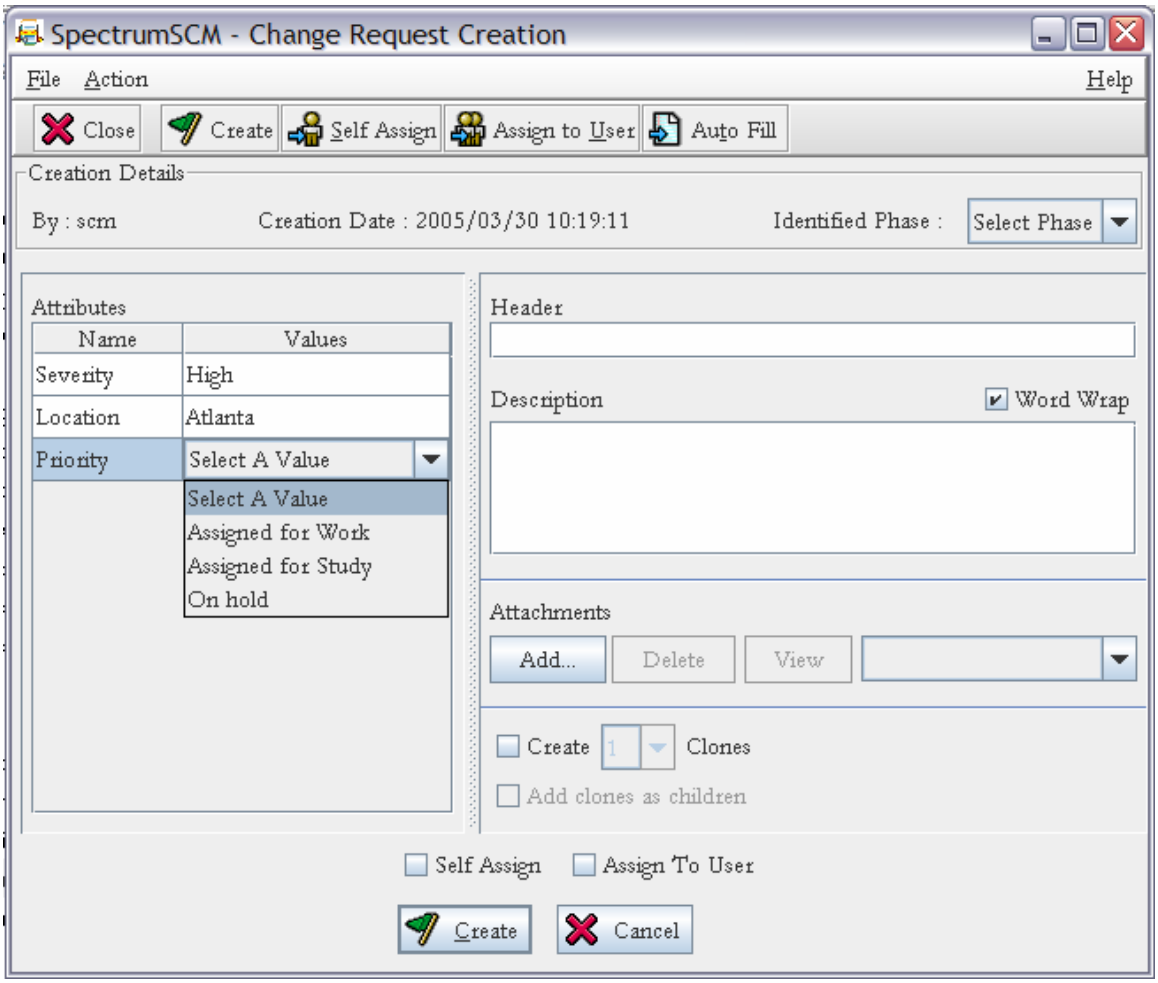
Figure #12

The change request creation screen contains a number of components that must be selected and/or completed before a new change request can be created. The screen components consist of the followings items:

- **Attribute/Value selection:** On the left of the change request creation screen is a table which will contain a series of change request attributes and their associated values. The values are held within a pop-down menu. Double click the default value to raise the pop-down menu.
- **Change Request Header:** The change request header is a one line description of the issue that is displayed along with the change request number itself on all GUI screens.
- **Change Request Description:** The change Request description is a multi-line description field that allows the user to enter as much information as necessary to describe the issue.
- **Identified Phase:** The identified phase is the phase with the current life cycle that this issue has been raised. For instance, if during system testing an issue is found, the user creating the CR would select that phase as the identified phase.
- **User/Self Assign check boxes:** The Self and User assign check boxes will either be enabled or disabled depending on the user's role within the current project. If creating a change request for personal use, select the Self Assign check box. If creating a change request for another user, select the Assign to User check box.

Figure #13 is a snapshot of the change request creation screen. In the figure, the Priority attribute value selection pop-down has been activated with a double click. Also note that the first element in the selection set is the "Select A Value" selection. Mandatory change request attributes will always have this selection as their default selection. Unless the user completes selecting an item for the mandatory attributes, they will no be able to complete the change request creation operation.

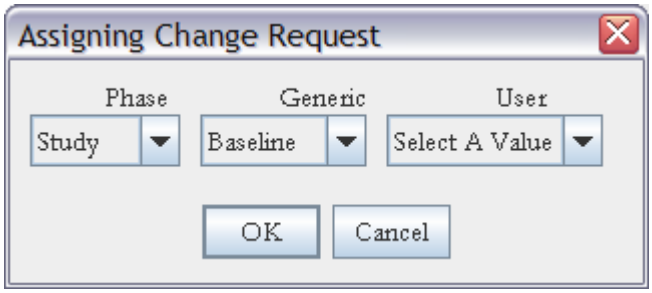
Figure #13



Several other items of interest on this screen include the "Auto Fill" button at the top in the tool bar, the attachments section mid-way to the bottom right and the "Create Clones" section at the bottom right of the screen. The "Auto Fill" functionality allows the user to create a new change requests based on the input values from an already existing change request. The attachments section allows the user to attach, view or delete attachments against the change request. Finally, the create clones section allows users to create a series of change requests, all with the same attributes/values, header and description information. The clones of the original change request are identified as such in their descriptions. Once clones have been created, the user can change the specific header information for each clone in the Change Request Assign/Modify screen. Cloned change requests can be automatically assigned to a WBS (work breakdown structure) by simply checking the "Add clones as children" check box before pressing the Create button.

The assignment of change request to a specific user and/or life cycle phase is handled as part of the confirmation popup that is presented during the create operation (Figure #14).

Figure #14



Change requests can be created and assigned to any phase within the life cycle except for the Completed and Killed phases. When the user has elected to self assign the change request, the User combo box is automatically removed from the Assign Change Request popup.

3.2 Checking out files

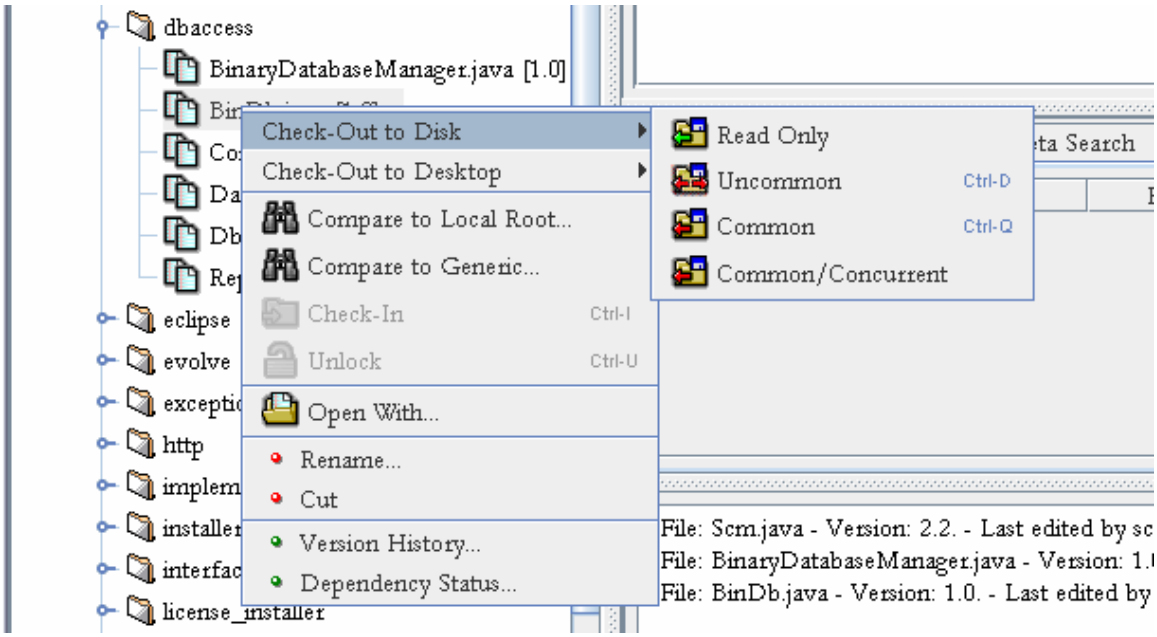
Files can be checked out of the system in one of two modes. Files can be checked out in a read-only mode, or files can be checked out in a writable mode, which means that the files can be modified and then checked back in. Files cannot be checked out for write mode unless the user has also selected an assigned change request. The connection between change request and checked out file is done during the check out operation and not during the check in operation like in some other tools, i.e. the association is not accomplished through the use of check in triggers.

Check out operations can be performed from many different locations. Check out functionality on the main screen exists in several locations:

- Main menu bar
- Main tool bar
- Context Sensitive menu

Other than checking a file out for one of the two modes mentioned above, the user can also specify whether the file should be checked out directly to the user's desktop and whether the check out should be common across all branches or specific to the current branch. These options are all presented on the context sensitive menu accessible by right clicking on any file in the source tree view (Figure #15).

Figure #15



The context sensitive menu in the screen snapshot above illustrates the options available to the user during the check out operation. *Note that in this case the tool has been configured for an advanced user. A normal user will see an abridged context sensitive menu, which simply contains the two options Check-Out to Disk and Check-Out to Desktop with further options for read-only and write access.* The user can choose between one of the four items presented on the pull-right menu section of the popup menu:

- **Read Only:** A check out read only simply places the file on the user's local hard drive, relative to the local root directory (LRD) but without write permissions.

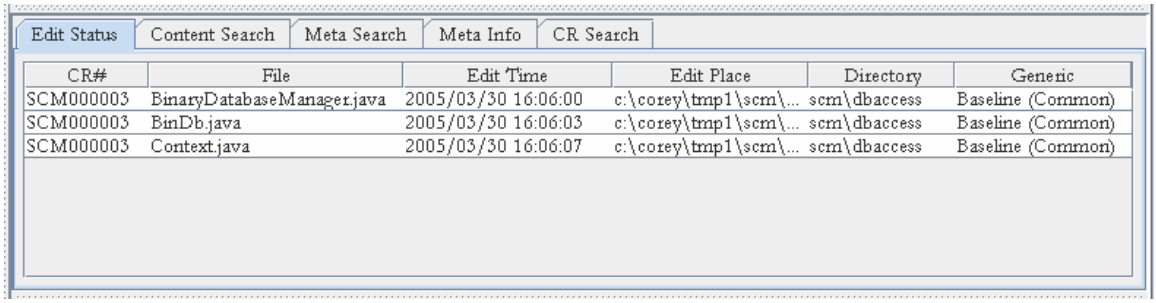
- **Uncommon:** An uncommon check out is a full write-privilege check out that, if the file is associated with multiple branches, will specialize the file into the current branch. Think of this as branching this particular file.
- **Common:** A common check out is a full write-privilege check out that does not break the sharing between
- **Common/Concurrent:** Check a file out in a soft locked mode for write access. When a file is checked out in a concurrent or soft locked mode, the file is still available for check out by other users.

Once a file is checked out, either to the disk or to the desktop, an entry is written into the edit status window on the main display (Figure #16). The foreground color of the file entry in the tree view is also modified to reflect the checked-out status of the file.

3.3 Checking in files

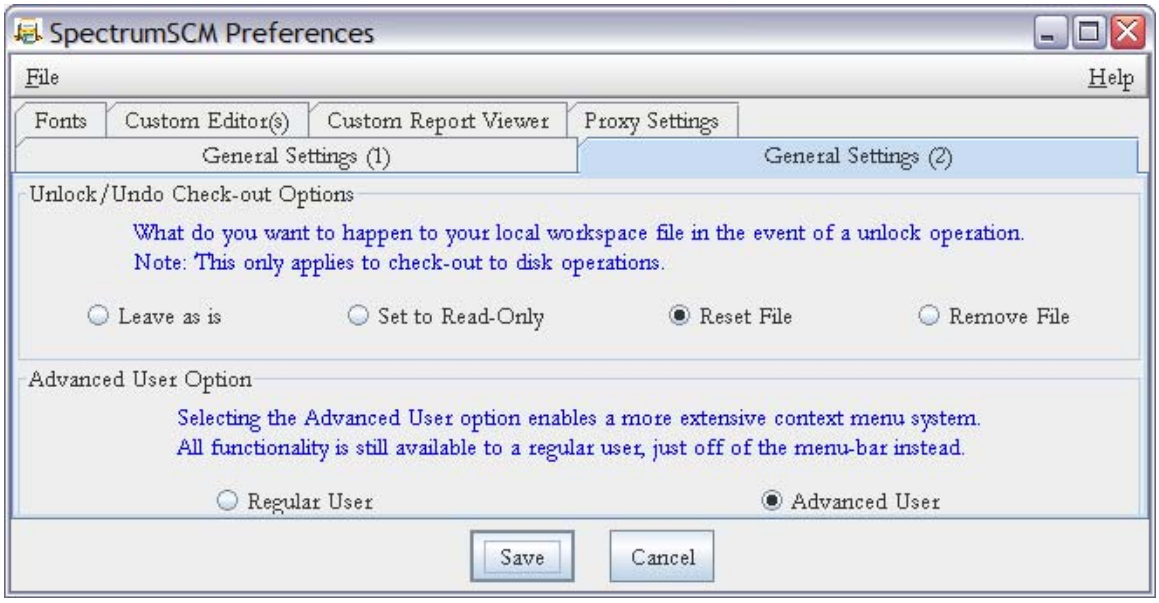
The check-in operation can be performed from several different locations within the main SpectrumSCM GUI. Easily the most elegant location from which to check-in files is on the Edit Status panel (Figure #16 above), which allows the user to multi-select a series of files and then complete the check-in operation.

Figure #16



Files can also be unlocked directly from the edit status panel. The user can multi-select a series of files and then by right clicking the mouse, execute the unlock functionality. What happens to the original file during the unlock operation is controlled by the unlock user preference (Figure #17)

Figure #17



The four (4) unlock options are as follows:

- **Leave as is:** If this option is set, the file is left as is completely intact.
- **Set to Read-Only:** If this option is set, the file attributes are set back to read-only mode.
- **Reset File:** If this option is set, the file is overwritten with the current head revision of the file.
- **Remove File:** If this option is set, the file is removed from the file system.

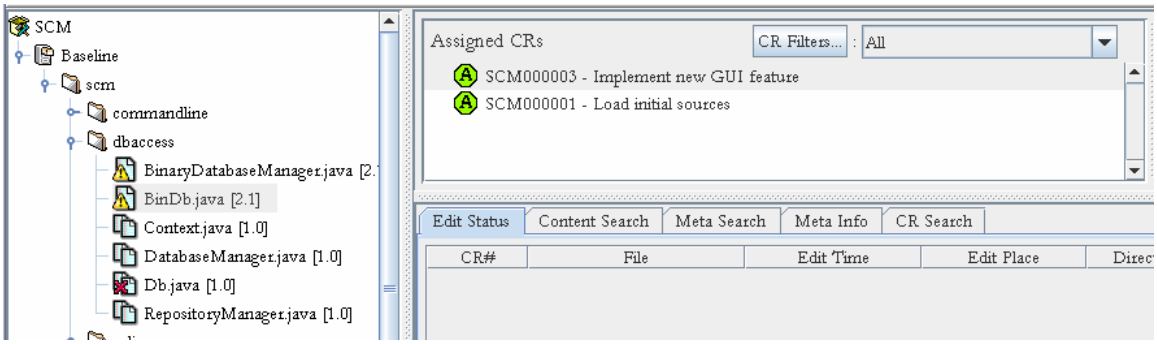
3.4 Synchronizing workspaces

SpectrumSCM has two features that allow users to keep their work areas synchronized with the repository:

- Workspace Analysis
- Workspace Synchronization

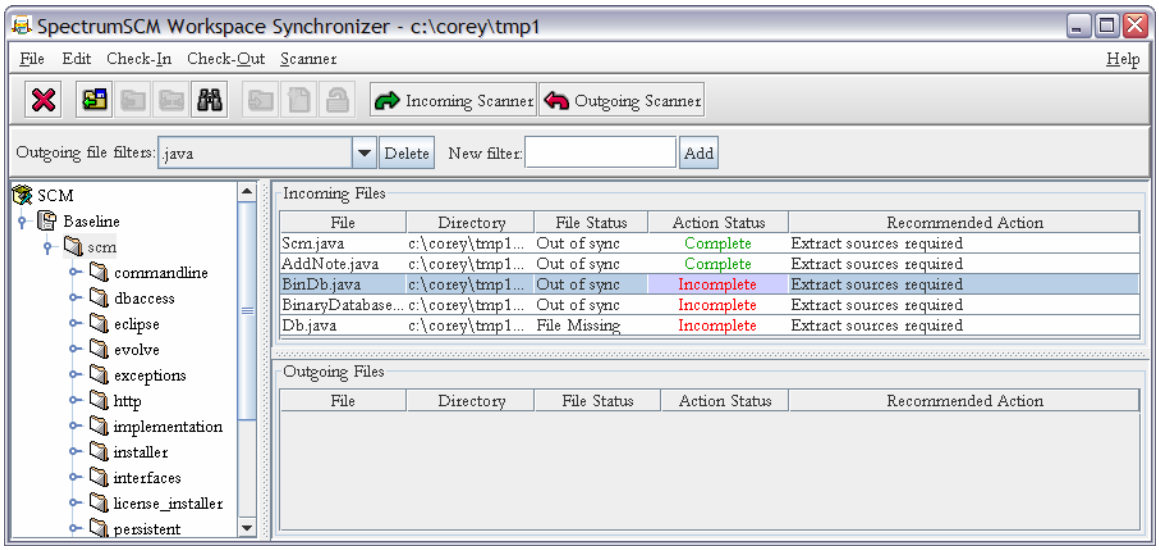
The Workspace Analysis feature automatically analyzes file differences between the user's workspace and the repository as the user interacts with the main GUI. The workspace analyzer decorates the default tree view icons with additional information based on the files state. In the example in **Figure #18**, two files are out of sync with the repository, (marked with yellow triangles) and one file is missing altogether from the user's workspace (marked with a red X). To resolve the synchronization issues, the user can right click and simply extract (check-out for read-only access) the marked files, or the user can use the workspace synchronization feature to resolve the issue.

Figure #18



The workspace synchronizer feature (Figure #19) allows the user to see a consolidated list of all files that need to come into the user's workspace, as well as a list of files that need to be merged from the user's workspace back into the repository. The actions are set up as two separate capabilities so that the user can execute each independently. This allows the user to selectively update specific files or directories within their local workspace based on work done by other users.

Figure #19



.
Once a set of files have been identified as either incoming out outgoing changes, the user can select one or more files and right click to see a set of appropriate actions. For ***incoming files*** the actions are limited to the following choices:

- **Check-Out (Read Only):** Extract the head revision of the file from the repository and overwrite the selected file.
- **Compare:** Compare the local file with the head revision in the repository. The compare is done using the built in 2-way diff/merge tool.

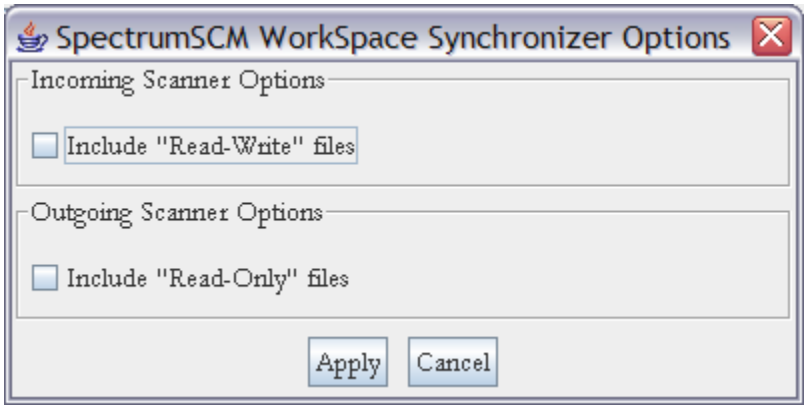
For ***outbound files***, those that need to be merged with the repository, the list of available actions is expanded to include the following:

- **Check-Out (Read Only):** Overwrite the existing file with the head revision of the same file from the repository.
- **Merge (Common):** Merge the local changes into the repository with a common edit. This action will invoke the 2-way merge tool on the selected file and the same file from the repository and allow the user to merge the changes into the repository via a "common" edit. A common edit is an edit that can be seen across all of the branches that the file is "common" or shared with.
- **Merge (Uncommon):** Merge the local changes into the repository with an uncommon edit.
- **Compare:** Compare the local file with the head revision in the repository.
- **Check-in:** If the file is officially checked-out, check the file back in.
- **Add File:** If the file does not exist in the repository yet, add it.
- **Unlock:** If the file is officially checked out, unlock the file.
- **Remove:** If the file has been deleted from the repository, remove the file from the local workspace.

The outbound scanner works by comparing the set of files and their types, located on the users LRD (local root directory), against a set of outbound filters that are defined by the user. Outbound filters are simply the file types (extensions) that the user would like the outbound scanner to examine. Files that are not part of the set of files defined by the outbound filters are ignored during the scanning process.

By default, the incoming scanner will scan only those files that are currently marked as read-only in the user's workspace. And, by default, the outgoing scanner will only scan those files that are marked as writeable. The user can change this behavior by selecting ***Edit->Scanner Options*** on the synchronizer main menu bar and modifying the default options (Figure #20).

Figure #20



The scanner options dialog contains two sections, one for incoming scanning options and one for outgoing. By selecting the "Include Read-Write files" in the incoming section, the incoming scanning operation will include those files that are writable by the user. If the user elects to "Include Read-Only files" in the outgoing scanner options, the outgoing scanner will scan read-only files. This is especially useful if offline work was performed on read-only files as some editors do allow the user to force writes of read only files. Also, the may have received a set of files from another source and by default the read-write flags on those files may be set to read-only.

3.5 Parallel development

Parallel development is a huge subject and the type of branching patterns necessary to properly manage a parallel development effort may vary based on the circumstances. Never the less, this section will cover the most important aspects of the SpectrumSCM branching model and show how the branching features in SpectrumSCM can be used to solve common parallel development problems.

There is an additional white paper on the SpectrumSCM website that discusses some of the more common patterns used to solve most parallel development issues. Please see the following link for additional details. http://www.spectrumscm.com/WhitePapers/BranchingDesignPatternsupdate.pdf

As mentioned in section 2.4, branching in SpectrumSCM is done at the project level first and then secondly at the file level within a branch. One of the chief differences between SpectrumSCM and most version control tools is that files in SpectrumSCM can be shared across multiple branches simultaneously. When a file is shared across two or more branches in SpectrumSCM, the physically shared files are referred to as being ***common*** across the branches. Files can be edited ***common*** in order to extend an editing session across one or more branches. Files can also be edited ***uncommon***, in which case the share is broken with the other branches and the file becomes specialized into the current branch. The commonality between the file and the other branches involved in the original group of common branches, remain intact. For instance, if a file is common across three branches BR1, BR2 and BR3, and the file is checked-out uncommon to BR3, the file remains common across branches BR1 and BR2.

After a file, or set of files, has been specialized into a branch, there are two options for merging branched material back into the other branch lines:

- Merging
- Re-Commoning

When the user elects to ***merge*** the contents of a file on one branch into another file on a different branch, only the contents of the files are merged together. The files themselves remain completely separate entities.

When the user elects to ***re-common*** the contents of a file on one branch into another file on a different branch, the contents of the files are merged together and then the two physical instances of the files are recombined into a single physical instance. At the lowest level this action is similar to forming a symbolic link between two files in different directories. One of the files is physically removed and is replaced with a symbolic link of the other file. This is essentially what is happening under the covers within SpectrumSCM when the user elects to re-common two files back together.

There are some keen advantages to the branching model in SpectrumSCM. For instance, branches become much longer lived entities than they are in plain version control tools. In most version control tools, once a branch is formed from another branch, the new branch and the parent branch begin to deviate from one another. Over time, the gap between the two branches can become quite wide. In order to refresh the branch line from the mainline, a rebasing or global merge operation must take place. This action can prove to be quite painful and is often referred to as merge mania and is avoided as much as possible. To avoid the situation, most branch lines are used for a specific purpose and then abandoned in place. SpectrumSCM improves on this scenario by giving baseline users the ability to perform real-time integration with a branch line (by performing common edits across branches), and by giving branch line users the ability to re-common their work with the baseline. By following this pattern, a branch line is constantly in sync with the parent baseline. For example, suppose there is a baseline branch and a new branch is formed from the baseline in order to complete a long running feature development operation. While the new feature is being developed, the files specific to the new feature are being specialized into the branch line. As work proceeds on the baseline branch, work is done common to the branch line and thus the branch line is constantly integrated with the baseline. Once the feature work has been completed on the branch line, the only difference between the branch line and the baseline is the addition of the new feature itself. Once the new feature is added common to the baseline branch, the two branches will be completely in sync and will stay that way until the next time that the feature branch needs to be used again.

Constant integration and re-commoning allow branches in SpectrumSCM to survive far longer than branch lines in other products. Plus, because branch lines in SpectrumSCM can be reused over and over again, the number of branches formed over time is greatly reduced.

Three different types of branches can be formed in SpectrumSCM:

- Mainline
- Release line
- Vendor line

A Mainline branch is a branch formed as a child of an existing branch. When a mainline branch is formed, all of the files in the parent branch are immediately shared with the new branch. All work that was visible on the parent branch is also immediately visible on the new branch. Use this type of branch when a branch needs to be formed to support parallel development of multiple releases.

A Release Line branch is a branch formed from a previous release of a product. When a branch is formed from a release, all of the files formed on the new branch are rolled back to the versions of the files contained in that specific release. If files in the release have not deviated since the release was formed, those files will remain common across the two branches. Files that have deviated since the release was formed are un-commoned from the parent branch and rolled back to the proper versions in the new release branch. Use this type of branch to form patch releases of existing releases.

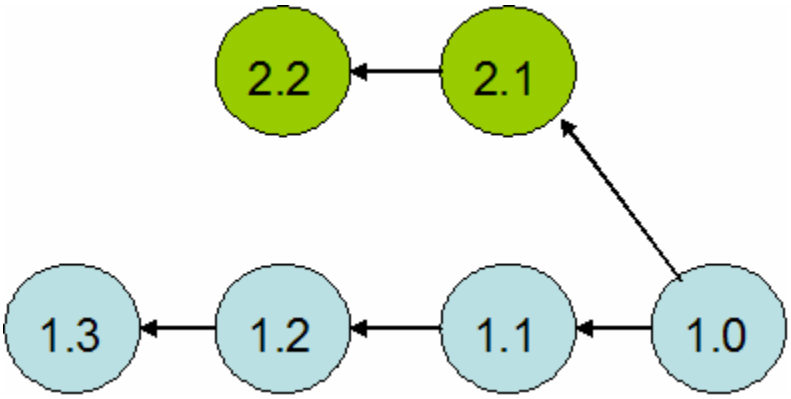
A Vendor branch line is a branch that is not rooted in another branch. Vendor branch lines can be used to support third party code releases from other vendors. Use this type of branch line when a sub-project must be added to a higher level project.

3.6 Merging/Re-commoning

As mentioned in section 3.5, merging is the act of recombining the contents of two files and re-commoning is a merge action along with a symbolic linking action. Where two files existed before, one file will dominate the re-commoning action and will become the file from which the symbolic link is formed. SpectrumSCM doesn't really use symbolic links, they're not platform portable. The concept is just used to aid the reader in understanding what is happening under the covers.

SpectrumSCM supports both a 2-way and a 3-way merge tool for use in merge/re-commoning operations. Each has advantages that work well in certain scenarios. For example, if work is being done on a branch line and some additional work has been done on the parent branch against the same set of files, the user is going to want to use the 3-way merge tool to review the differences between the two. The reason for this is that the 3-way tool can graphically display the differences between the two files in reference to the common ancestor of both files. Figure #21 illustrates the point.

Figure #21



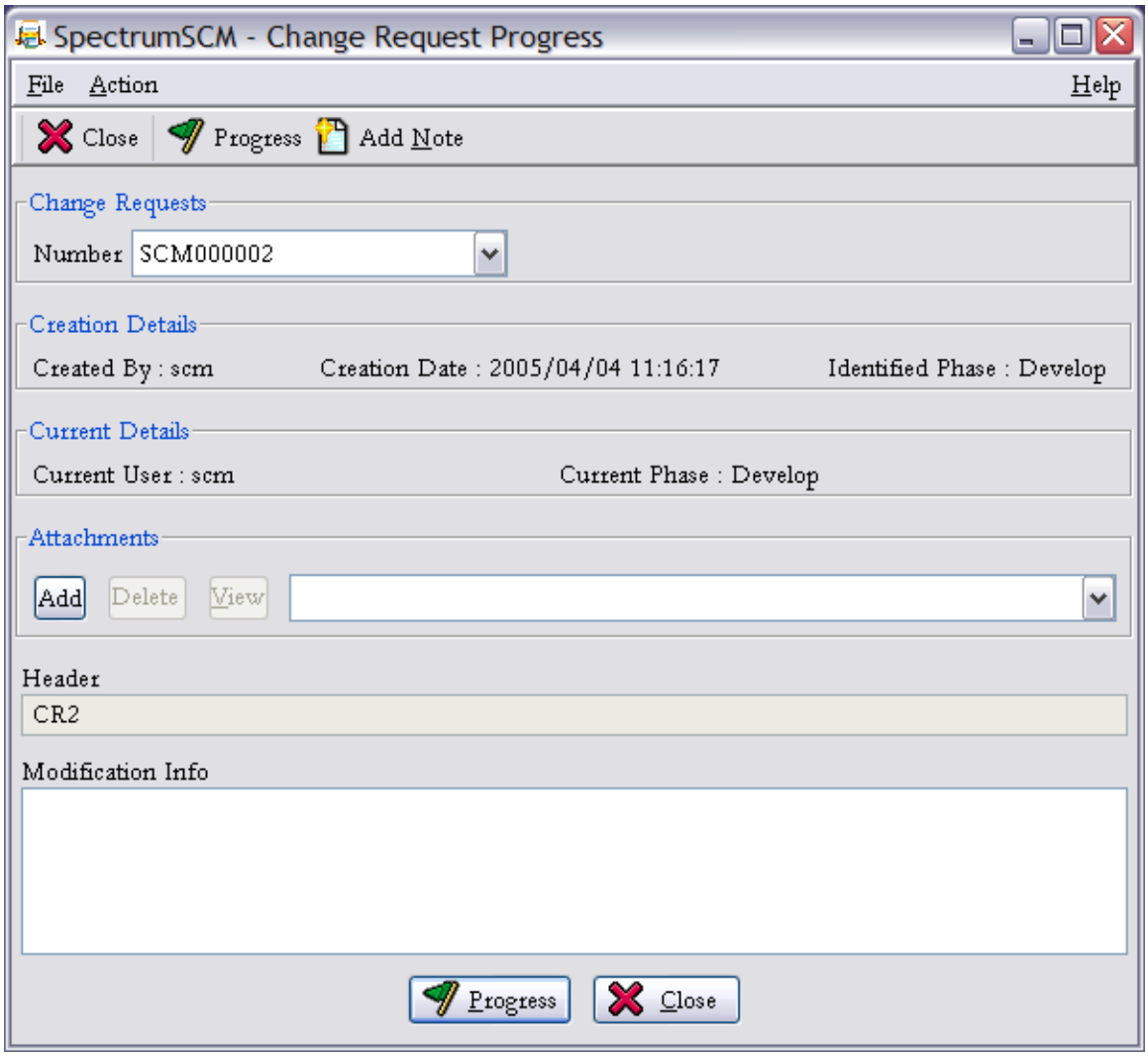
In order to properly merge version 2.2 of the file illustrated in Figure #21 with version 1.3 of the same file on the other branch, the user must also be able to understand what has happened to the target file (version 1.3) relative to the common ancestor, (version 1.0). This is where the 3-way merge tool will help the user tremendously. The user will be able to see the changes made to version 1.3 of the file relative to version 1.0 and he/she will also be able to see the changes made to file version 2.2 relative to the common ancestor (1.0). Figure #8 above contains a snapshot of the 3-way merge tool.

While the 2-way merge tool can be used to merge any two files together, it is best used in situations where the target merge file has not deviated from the common ancestor. For instance, if in Figure #21, version 1.1, 1.2 and 1.3 of the parent branch line had not been created, then the 2-way tool can be easily used to merge the changes from the branch line back into the parent line.

3.7 CR promotion

As previously mentioned, change requests in SpectrumSCM flow through a software development life cycle created by the end user organization. The assignment of change requests into the various states within a life cycle is either done automatically through a workflow engine or is done manually by the user. Only users that have been assigned into a role that allows them to perform change request re-assignments can physically assign a change request into a different phase. Users of the system that don't have the capability to re-assign change requests are only allowed to promote their change requests into the TBA (To Be Assigned) super state. This promotion is done using the change request progress screen (Figure #22)

Figure #22



The change request progress screen allows the user to promote assigned change requests up into the TBA super state. The progression of the change request into the TBA state, like all change request creation and transition activities, automatically sends e-mail notifications to other interested users, e.g. project managers. Before promoting a change request, the user has an opportunity to add notes or modification information directly to the change request. Modification information is different than a note in that this information is associated with the change request transition itself. Notes, on the other hand, can be associated with a change request at any point and are not necessarily associated with phase transitions.

4.0 Summary

The purpose of this paper is to give the reader a quick overview of SpectrumSCM's fundamental building blocks and tools. Readers interested in learning more about specific details related to SpectrumSCM's fundamentals and theory of operation are encouraged to take a look at the full user's guide for the tool, which is published in full on the SpectrumSCM web site, www.spectrumscm.com.

Please contact Spectrum Software, Inc. (support@spectrumscm.com) for further information.