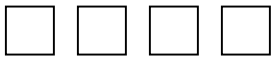


Impact Analysis: An Essential Part of Software Configuration Management

An MKS White Paper

By Steven Church
Product Manager

What is Impact Analysis?

The concepts behind impact analysis have been around for a long time and are not limited to computer software. For years, parents have taught valuable life lessons to their children about the consequences of their actions: "Share the toy or I'll take it away" or "If I have to ask you again…" In short, every action has a consequence, negative or positive.

Impact analysis for computer software is much the same: "If I redefine this variable, what will be the effect?" If software developers cannot answer these simple 'What if…?" questions, then they should not be making the change.

A more formal definition of Impact Analysis is:

> "Software change impact analysis, or *impact analysis* for short, estimates what will be affected in software and related documentation if a proposed software change is made." [Arnold 1996]

As simple as this definition sounds, some software tools that claim impact analysis functionality miss the mark and are incomplete. Their impact analysis functionality is limited to file dependency checking (e.g., foo.c depends on foo.h) which has been around a while in the form of makefiles.

Many more meaningful questions can be answered, however, if you reduce the granularity from files to programming constructs or "tokens" that are parsed[1] [Parr 1997] from source code: "Where is this variable referenced?", "Why is this function so complex?" and "If I add a parameter to this function, what else is affected?" These are all excellent questions, which illustrate the various aspects of impact analysis. Referenced variables indicate code navigation functionality. Complexity measurement leads to source code comprehension functions. Therefore, cause and effect analysis of proposed changes is at the core of impact analysis.

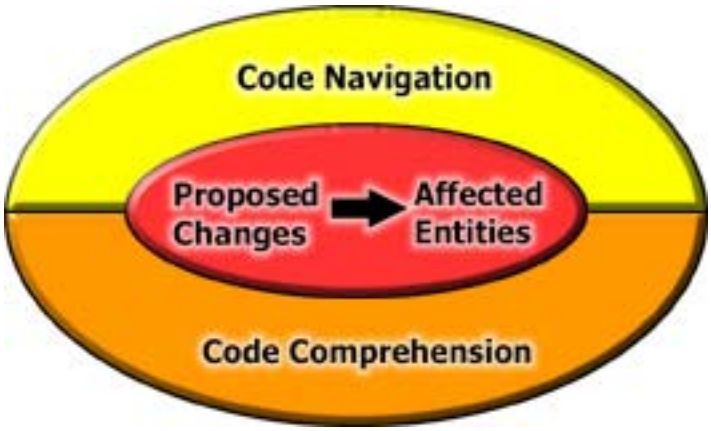
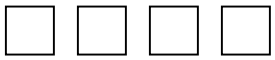


Figure 1: Impact Analysis Functionality

[1] Excerpt from reference found at: http://www.antlr.org/Articles/langparse.html



> Of course, tools alone will not solve the "significant technical challenge" of automated impact analysis. [Arnold 1996]

What is SCM?

The day the first software program was created there was a need to change it [csc-m 2002]. There is a vast body of knowledge available that addresses this very issue as well as the approaches, hurdles, and successes involved in implementing SCM systems. Of all of the definitions that can be found, no definition has captured its essence better than the following:

> "SCM is the discipline of controlling the evolution of software systems."[2]

The main principle that stands out is "controlling". You certainly don't want to prevent the evolution of software but you do need a mechanism for managing and approving proposed changes. You can imagine the chaos that would follow if QA engineers were expected to test software systems where developers had changed the software without adjusting the requirements.

Software development is a complex task. Applying tools and processes (SCM) helps to streamline the effort, ensure repeatability and will lead to productivity and quality improvements.

Managing Change

SCM helps with *how* changes are made to the system; Impact Analysis helps answer these other important questions:

- *Who* is affected by the changes?
- *Where* in the source code do changes need to be made?
- *What* else is affected by these changes?
- *Why* do we need to make all of the changes?
- *When* should all of the changes be made?

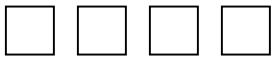
The reasons for changing software can usually be narrowed to one of a few key reasons:

1. Software Defect – As much as we'd like to believe our software is defect-free, bugs do exist.
2. Enhancement– A new feature or a change to an existing feature is requested (hopefully leading to a product improvement).
3. Developer Innovation – This is an often-overlooked aspect of software development projects. Software is changed in subtle ways, often for the better, without an approved change request.

To be sure, fixing known bugs is desirable, but you do not want to fix one bug only to introduce another undetected bug(s). It is critical that you are aware of the impact or consequence of "fixing" the defect.

For similar reasons, you want to be careful about adding new features that may conflict in unknown ways with existing features or other proposed new features. An example of this can be found in Microsoft's Active Desktop feature — it's nice to have, yet it seems to conflict with your

[2] Paraphrased from "*Tools for Software Configuration Management*", Proceedings of the 2nd International Workshop on Software Version and Configuration Control, Walter F. Tichy, 1988.



tropical beach wallpaper image. To see this yourself, right-click on your desktop, select the properties menu item, then choose a background image ensuring that it is stretched to cover your entire desktop. Then right-click on the desktop again and select the "Customize my desktop" menu item. In the dialog box that appears in the "Web" tab, check the "Show Web Content…" checkbox and select "My Current Home Page" from the list. You will notice that your home page now covers your original background image. You can of course resize and reposition your home page but it still covers some of your background image.

Of course not every change made to software is approved or desirable. The "feature creep" that inevitably happens to applications can offer wonderful surprises for users, but at the same time cause Product Managers endless grief. The extras added by developers may not allow for adequate testing; delay release cycles; misalign with the strategic direction; cause conflicts with other features; or, at worst, violate contractual obligations. The product must be improved, but always in a controlled manner.

The following is a basic process for managing software changes with SCM and Impact Analysis tools.

Step 1: Understanding the Change Request

The initial steps of any process are often the most critical. When a software change request is received it is important to fully understand the request before proceeding. The expensive cycle of coding-testing-launching-recoding-retesting can be avoided if you understand the requirements correctly the first time. A mechanism (i.e. MKS Integrity Manager) to capture sufficient details such as: time request was made, expectations for completion, categorization of request, detailed explanations, is invaluable. If a request is not understood, the mechanism should also be able to handle and capture clarification requests and updates to the original change request.

Step 2: Planning Changes

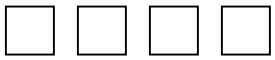
Once the request is fully understood and documented, developers begin the investigation phase by attempting to pinpoint that exact code and locations within the source code that will be affected by the implementation of the change request. In large applications (10 million+ Lines of Code) this can require significant effort. Tools to assist navigation, searching and providing graphical representations of source code can speed up the investigation phase greatly. A sound principle developers should follow in this phase is: "Don't change source code if you don't understand it."

Step 3: Assessing Impact of Proposed Changes

Assuming the code is well understood and the developer has a plan by which coding changes are made, another highly recommended principle is: "assess the impact of proposed changes BEFORE they are made". Often, dependencies between source code objects are not obvious and forging ahead with coding changes can lead to "breaking the build"; defects being found in testing requiring refixing/retesting; or application feature conflicts discovered by users.

Step 4: Approving Changes

Once the full impact of proposed changes is clear, the decision to approve changes can be made with confidence. In some cases, impact analysis reports will indicate a change in the approval due to the fact that the proposed changes cause too much of a "ripple effect" or are too costly to implement.



Step 5: Making the Changes

Proceeding with the approved changes leads into the common developer cycle of Check-out — Edit — Check-in. Additional value can be gained by using this process if the impact analysis tools and the SCM tools are integrated. The impact analysis tool should identify which source files are flagged for modification and then inform the SCM tool to check-out all files in batch mode. This saves the developer the manual effort of identifying the list of files required in the "change package[3]. Ideally, the impact analysis functionality and SCM functionality can be accessed through the Integrated Development Environment (IDE) that the developer is already familiar with.

Step 6: Assessing Quality

After all the changes are made, the last step is to ensure that the quality of the source code has not degraded. A tool to measure quality before and after coding changes is valuable for this. Identifying and correcting quality issues in source code can bring higher quality applications, leading to lower maintenance costs the next time the code needs to be assessed and changed.

MKS Solution

Since 1984, MKS has produced high quality software development tools known for their technical innovation and powerful functionality. While other vendors provide complicated framework solutions that address all facets of the application development lifecycle, MKS is focused on excellence in one area: software development. This enables MKS to deliver products that meet or exceed the needs of the most demanding enterprise software development organization.

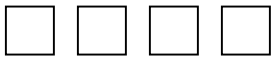
With solutions for software configuration management (SCM), process and workflow management, source code comprehension and analysis, and UNIX/NT interoperability, MKS helps organizations achieve their quality goals while increasing productivity and reducing costs.

MKS Integrity Manager is the most scalable and flexible process and workflow management solution available today for connecting distributed development teams across the enterprise. Integrity Manager allows development teams to implement processes that are as complex or simple as their individual needs or project requirements dictate.

MKS Source Integrity Enterprise Edition is the enterprise choice for comprehensive, cross-platform software configuration management. Its platform transparent, advanced multi-tier architecture provides scalability and security in local and distributed environments while maintaining ease of use and low cost of ownership.

MKS Code Integrity helps software development organizations comprehend complex software systems constructed by the enterprise over time, assess the impact of changes to code before they are implemented, and ultimately improve the quality of software through coding standards automation.

[3] "A change package allows you to specify groups of members that are affected by an issue. For example, a solution's change package might contain files that need to be changed in order to satisfy a problem." – MKS Integrity Manager User Guide.



Conclusions

Both SCM and Impact Analysis tools are invaluable for creating and managing the evolution of quality software. It is the process under which they are employed, however, that exposes their true capabilities. Therefore, when used together in your environment in conjunction with a flexible process these tools will:

1. Lower software development costs
2. Improve application reliability
3. Support collaboration throughout the enterprise

References

[Fowler 1999] 'Refactoring: Improving the Design of Existing Code', Martin Fowler et al, Addison-Wesley, 1999, ISBN: 0201485672

[Arnold 1996] 'Software Change Impact Analysis', Robert S. Arnold, IEEE Computer Society Press, 1996, ISBN: 0818673842

[csc-m 2002] 'comp.software.config-mgmt FAQ: General Questions', http://www.faqs.org/faqs/sw-config-mgmt/faq/, 2002

[Parr 1997] 'Language Translation Using PCCTS & C++', Terence J. Parr, Automata Publishing Company, 1997, ISBN: 0962748854

CIE0203WP