StatSVN: Statistics for SVN Repositories Based on the Open Source Project StatCVS

Jason Kealey and Gunter Mussbacher

***Abstract*—Effective project management requires reliable and accurate software metrics in order to make informed decisions about a software project. This paper introduces StatSVN, a new open source software metrics tool which calculates lines of code (LOC) metrics from Subversion (SVN) repositories. The software design of StatSVN is based on the design of the open source StatCVS by implementing a new input layer for SVN repositories but reusing the model and output layers of StatCVS. Major design decisions made for StatSVN are explained and the new tool is validated by comparing metrics compiled by StatSVN with metrics compiled by StatCVS.**

***Index Terms*—Software Metrics, Configuration Management Systems, SVN, CVS, Open Source Software, StatCVS**

I. Introduction

SOFTWARE metrics give the project manager the means to assess the state of a software project more accurately and reliably. Based on this data, the project manager can then make decisions regarding the project in a more educated way. One such metric is lines of code (LOC). Although LOC is not the best metric to indicate the size of a project, it is a metric which can easily be automated. Using a configuration management system (CMS), LOC metrics can be calculated for all revisions in a repository. These metrics combined allow a depiction of the history of a project, directory, or file. Furthermore, historical LOC metrics can then also be compiled for each contributor to the CMS.

Subversion (SVN) [15] is an open source CMS which addresses some of the shortcomings of the popular open source tool CVS (Concurrent Versions System) [3]. It is a rather recent development – its beginnings trace back to early 2003 – and good open source metrics tools are not yet available for SVN. Our tool StatSVN wants to change this.

Our main goal is to provide LOC metrics for software projects that use SVN for configuration management. There are, however, secondary goals which have to be considered for this project. We have received strong encouragement to support the open source community. Reducing effort is always an issue and more so in a time-constrained project such as ours. Finally, it is important to ensure that the initial feature set can be increased over time to make use of new SVN functionality. The oval-shaped *goal* nodes in Fig. 1, Fig. 2, and Fig. 3 represent our four goals.

Note that Fig. 1 to Fig. 5 have been created according to the User Requirements Notation (URN) with the jUCMNav tool (for more information on the notation and tool see [1], [8], [9], [10], [16], and [17]).

There are several alternatives to consider regarding the development of this new tool (see the or-breakdown of the main goal into hexagon-shaped *task* nodes in Fig. 1, Fig. 2, and Fig. 3). The tool could be built from scratch or could be based on other tools already available (see section II for details about such tools). Commercially available SVN tools are unfortunately closed source and are therefore not further taken into account (see the *Break* contribution of *based on commercial SVN tools* to *support open source* in Fig. 1). All other alternatives support open source (see four *Make* contributions in Fig. 1).

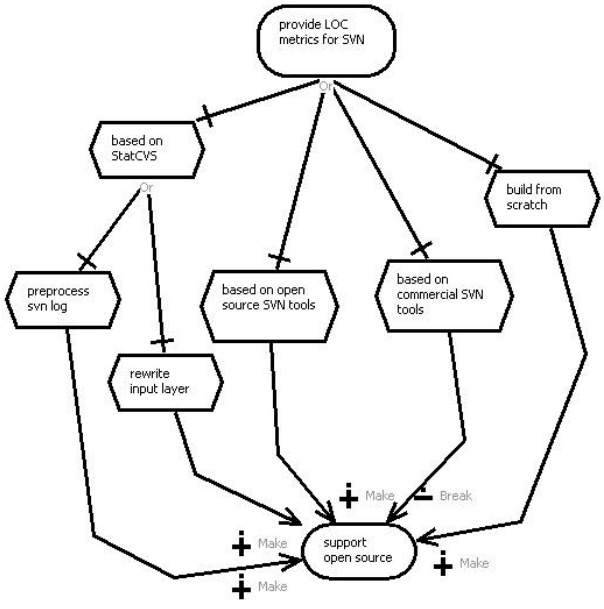

Fig. 1. Assessment of Alternatives for "Support Open Source"

Building from scratch requires the most effort and therefore is not a viable option in the context of this project (see *Hurt* contribution in Fig. 2). The current open source SVN tools that provide metrics about repositories either do not provide LOC metrics or do not offer as many features as StatCVS

Manuscript submitted for review on March 21, 2006.

J. Kealey is with the School of Information Technology and Engineering (SITE) at the University of Ottawa, Ottawa, Ontario, K1N6N5, Canada (e-mail: jkealey@ shade.ca).

G. Mussbacher is with the School of Information Technology and Engineering (SITE) at the University of Ottawa, Ottawa, Ontario, K1N6N5, Canada (e-mail: gunterm@ site.uottawa.ca).

[13]. These tools will consequently require less effort (see *Help* contribution in Fig. 2) than building from scratch but more effort than approaches based on StatCVS (see two *Make* contributions in Fig. 2). Therefore, we decided to implement StatSVN based on StatCVS.

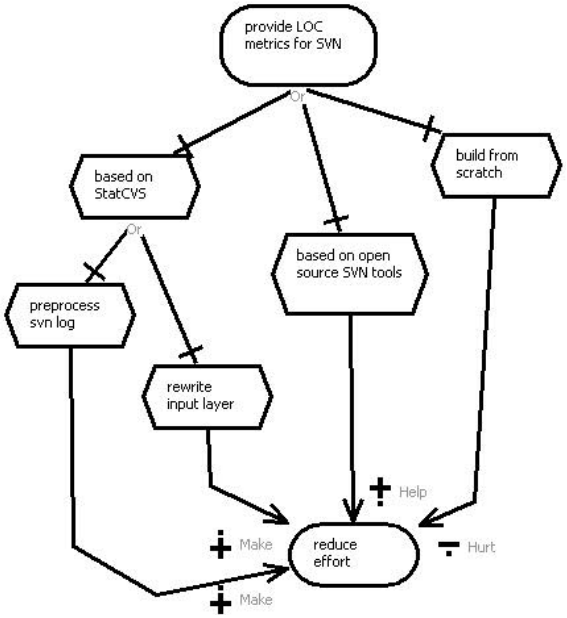

Fig. 2. Assessment of Alternatives for “Reduce Effort”

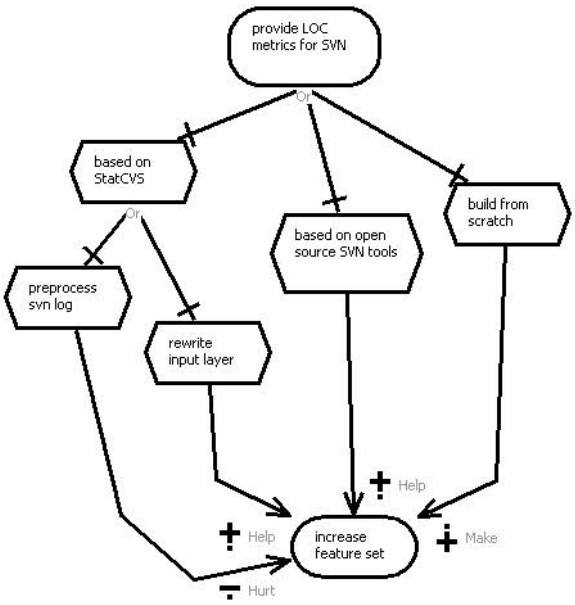

Fig. 3. Assessment of Alternatives for “Increase Feature Set”

Two alternative solutions based on StatCVS were discussed. The SVN log file generated by SVN could be converted by a preprocessor into a CVS log file. This causes problems going forward as some features of SVN are not supported by CVS. It may not be possible to express these features with a CVS log. Furthermore, changes to the StatCVS implementation may be required (see *Hurt* contribution in Fig. 3). Therefore, we decided to adapt StatCVS for our purposes by rewriting the input layer of StatCVS as a first step. Such an approach does not preclude the addition of new SVN features in the future (see *Help* contribution in Fig. 3). Initially, only a subset of the functionality of StatCVS will be supported. For an exact listing of all features of StatCVS and the intended features of StatSVN see section II.

The remainder of the document gives an overview of the differences between CVS and SVN, lists the features of StatCVS, and compares existing SVN metrics tools in Section II. Section III first summarizes the design of StatCVS and then details the changes as well as major design decisions made for StatSVN. Section IV validates our approach by comparing the outputs of StatSVN against the outputs of StatCVS. Finally, section V summarizes our contributions and identifies future work.

II. Background

In order to understand the design decisions for StatSVN, it is necessary to consider the differences of CVS and SVN. Even more importantly, the log files generated by CVS and SVN have to be discussed as both, StatCVS and StatSVN, base their metrics calculation on log files for performance reasons. Both tools calculate metrics in similar ways, first requiring a checkout of the project to be analyzed into a working directory. The metrics are then generated with the help of the log file, the working directory, and the repository. The following discussion of CVS vs. SVN is not exhaustive but rather focuses on the differences which have an impact on the calculation of LOC metrics from log files. In general, log files have entries which identify “add”, “modify”, and “delete” actions in the repository.

a) SVN’s revision numbers are per-commit, corresponding to atomic commits. CVS’s revision numbers are per-file. Therefore, the SVN log is organized by commit, not per file.

b) Directories are versioned in SVN. Therefore, actions on directories appear in the log. These entries only implicitly define actions on files while CVS logs explicitly mention such files.

c) Renames are versioned in SVN and the history of the file is retained. This is not supported by CVS and therefore not taken into account by StatCVS. A CVS log simply shows that one file was deleted and another was added. The SVN log contains such entries for renamed files which will have to be interpreted appropriately.

d) File meta-data ("properties") are versioned in SVN. This is not supported by CVS. If such properties are changed, the SVN log contains an entry which has to be interpreted as a change where no lines were added or removed.

e) As SVN replaces CVS’s symbolic names for tags with an underlying "copy" operation to a user-specified directory, the handling of tags is simplified. The SVN log, however, does not contain any symbolic names or explicit tags since it simply shows the copied files and directories. This makes the identification of releases considerably more difficult.

f) As SVN replaces CVS’s branches with an underlying "copy" operation to a user-specified directory, the handling of branches is simplified. The SVN log, however, does not

contain any explicit branches since it simply shows the copied files and directories. A limitation of StatCVS is that it only takes the main branch into account and ignores side branches. In the case of StatSVN, the user can decide what to include in the report since metrics are always compiled for a specific directory, regardless of it being the main branch, a side branch, or the root directory.

g) The SVN log does not indicate binary files whereas the CVS log does. Binary files have to be identified through other means for StatSVN.

h) Finally, the SVN log does not contain counts for lines added and removed by a revision. A feature request in this regard has already been turned down by the developers of SVN [14]. This has the biggest impact of all differences since line counts now have to be retrieved by a "diff" operation on the repository server. As this is a very expensive operation, we expect it to be StatSVN's limiting factor in terms of scalability.

TABLE 1
FEATURES OF STATCVS AND STATSVN

Feature	StatCVS	StatSVN
Historical LOC metrics for project	Yes	Yes
Historical LOC metrics for directories	Yes	Yes
Historical LOC metrics for files	Yes	Yes
Historical LOC metrics for authors	Yes	Yes
Commit Log with LOC changes	Yes	Yes
Accurate LOC for existing files	Yes	Yes
Accurate LOC for deleted files	No	No
Metrics for main branch and side branches	No	Partial
Support for binary files	No	No
Support for renamed, copied, and replaced files	No	No
Configurable input log file	Yes	Yes
Configurable "working copy" directory	Yes	Yes
Configurable output directory	Yes	Yes
Configurable caching directory	N/A	Yes
Support for include and exclude file patterns	Yes	Yes
Identification of releases (symbolic names)	Yes	No
Configurable start and end date of report	Yes	No
Configurable title and description of project	Yes	Yes
Configurable css file	Yes	Yes
Optional logging (verbose/debug)	Yes	Yes
Support for ViewVC[a]	Yes	Yes
Support for CvsWeb[a]	Yes	N/A[b]
Support for Chora[a]	Yes	Yes
Support for ant task	Yes	Yes

[a]ViewVC [18] (formerly ViewCVS), CvsWeb [6], and Chora [2] are web-based tools for browsing SVN or CVS repositories.
[b]CvsWeb does not support SVN.

Table 1 lists the features provided by StatCVS and intended for StatSVN. StatSVN supports most features of StatCVS. Most notably, both tools only provide accurate LOC if the file still exists in the working copy of the current revision. This is due to the calculation of LOC which starts with the latest version of a file and works its way back to the first version using the counts for added and removed lines. If the latest version is not available (as is the case with deleted files), StatCVS only approximates the LOC of the file. StatSVN is intended to do the same, but this could be improved by querying the repository server. Such an approach may cause only a small additional performance loss since "diff" operations already have to be performed on the server.

StatSVN compiles metrics for a directory regardless of whether it is the main branch, a side branch, or a tagged release. This feature, however, is only supported partially since the files in branches or tags are interpreted as new files without taking their history into account. StatCVS compiles metrics for the main branch only.

Neither StatCVS nor StatSVN supports line counts on binary files which are deemed to contain zero lines. Binary files, however, are listed in the commit logs.

The initial version of StatSVN is not intended to support renamed, copied, and replaced files. Similarly to StatCVS, such files are treated as new files without taking their history into account. However, it is possible to do so and should be considered for future versions.

StatCVS does not need to query the server for any additional information, and therefore does not need a directory to cache such information for performance reasons.

As the identification of releases is difficult, StatSVN is not intended to include this feature in its initial version. It, however, seems possible to do so and should be considered for future versions.

StatCVS allows reports for given start and end dates indirectly by restricting the entries in the log file to a date range. The initial version of StatSVN does not provide this feature. Future versions may support it if start and end dates are provided as command line parameters because SVN log files cannot be restricted to a certain date range. The usefulness of such a feature, however, is still being discussed.

In summary, the initial version of StatSVN will offer all features of StatCVS listed in Table 1 except identification of releases and configurable start and end dates. The initial version also deals with all differences discussed in II.a) to II.h) except for support of renamed, copied, and replaced files (II.c) and the aforementioned identification of releases (II.e).

Other metrics tools for SVN exist but quality tools are rare since SVN is relatively new. The most promising competitor is integrated into the Polarion [12] web-based software project management framework. However, it is not open source nor is it free to use. Furthermore, it can produce statistics only for users of the Polarion framework.

Open source alternatives worth mentioning are MPY SVN Stats [11] and CVSAnalY [5], both of which offer a similar feature set. The main problem with these tools is they do not take LOC metrics into account for SVN repositories. Finally, GForce [7] is an open source collaborative software development tool which offers LOC metrics for SVN repositories using PHP scripts which dynamically query the repository. It suffers from very poor usability because of its latency and lack of a high-level view of the repository activity.

III. Approach

As a first step, the source code of StatCVS was analyzed to verify that the *input* package is sufficiently separated from the *model* package and output layer of StatCVS. Note that the output layer contains several packages which together form a cohesive unit. The following architectural design decisions were recovered:

a) Only *Main.generateDefaultHTMLSuite()* uses the *input* package.

b) Only the *input* package uses the *model* package to create the representation of the repository.

c) The output layer only retrieves information from the *model* package but does not add anything to it (note that *ModulePage* in the output layer does create new commits with *Commit()* and *Commit.addRevision()* but only to duplicate a subset of already available information relevant to a particular directory).

d) The purpose of the *ant* and *util* packages is evident through their respective names. These two packages have only a minor impact on the required changes for StatSVN and are therefore out of scope for this document.

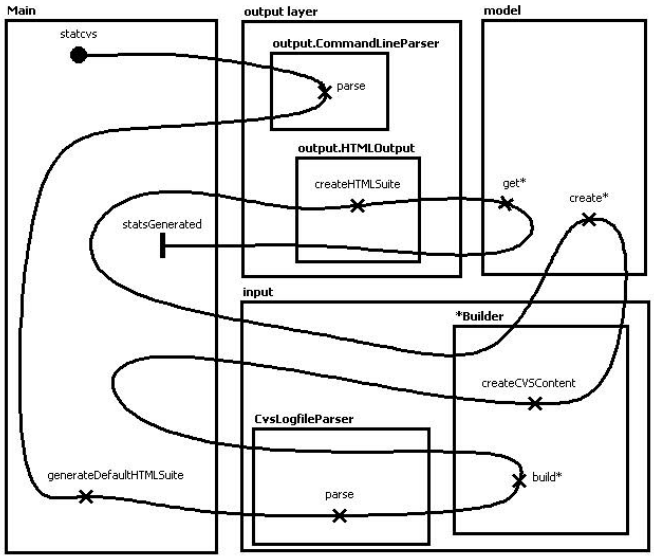

Fig. 4. Overview of StatCVS

Fig. 4 shows the main use case of StatCVS. After parsing the command line (*CommandLineParser.parse*), the default HTML suite is generated by first parsing through the CVS log file (*CvsLogfileParser.parse*), then creating the model (*createCVSContent*), and finally creating the output (*createHTMLSuite*). *CvsLogfileParser.parse* provides (*build**) data to several builders (**Builder)* which interact with the *model* to create the representation of the repository (*create**). During the last step, *createHTMLSuite* retrieves the repository information (*get**) and uses it to populate the output.

The design of StatCVS makes use of the MVC pattern. *Main* is a very slim controller since StatCVS is not an interactive application. The output layer represents the view. The *model* in the design of StatCVS is the model in the pattern. The builder pattern has also been applied several times in the *input* package to further separate the objects in the *model* package from their construction process.

As we wanted to reuse as much of the existing code as possible, the decision was taken to not make any changes to the *model* package and output layer. Furthermore, the builders are to be modified minimally and ideally only *CvsLogfileParser* is to be replaced by *SvnLogfileParser*. Considering time constraints, it was also decided to simply rename CVS classes to SVN classes and modify the renamed classes. This was done instead of more sophisticated refactoring which would have introduced subclasses for CVS specific and SVN specific behavior.

The eventual design of StatSVN closely followed the above mentioned rules even though three factors complicated the scenario. First, performance testing revealed serious delays for "diff" operations required to retrieve line counts. The decision was taken to persist line counts (see section II.h). Second, binary files have to be identified by querying SVN meta-information in the local working copy of the repository. As this only works for files which are not deleted, the decision was taken to also persist the binary status (see section II.g). Third, implicit actions on files have to be derived from actions on directories. This requires further processing of the data from the SVN log (see section II.b). All other supported changes due to the differences of SVN log files and CVS log files required relatively minor changes to the implementation of StatCVS and did not have an impact on the overall structure of StatCVS.

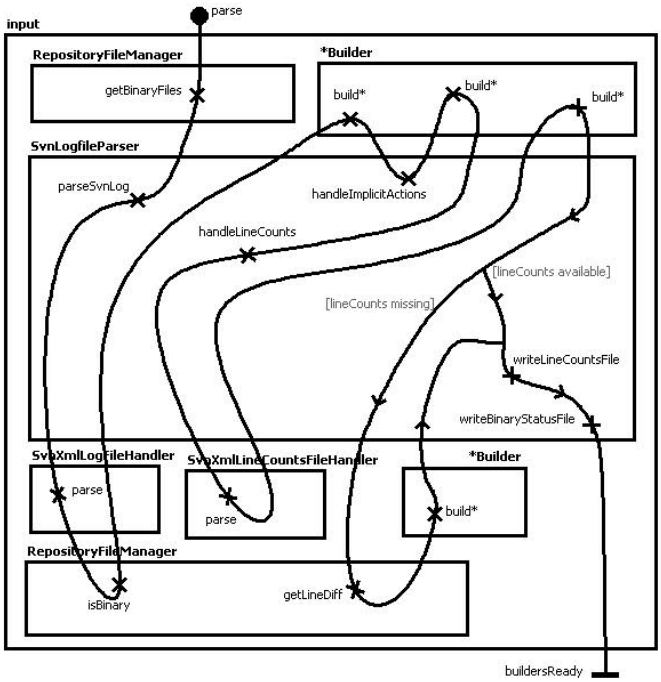

Fig. 5. Details of SvnLogfileParser

Fig. 5 shows the details of *SvnLogFileParser* which replaced *CvsLogfileParser* and the *parse* and *build** responsibilities in Fig. 4. There are four steps to the gathering of repository information for StatSVN. First, the list of binary files is compiled using the local working copy (*getBinaryFiles*).

Second, the SVN log file is parsed (*parseSvnLog*). Third, file actions implicitly defined in the SVN log file are handled (*handleImplicitActions*). Fourth, line count information is collected (*handleLineCounts*).

Steps two, three, and four forward data to the builders (*build**). The second step *parseSvnLog* requires the help of the *RepositoryFileManager* to retrieve the binary status (*isBinary*). The fourth step *handleLineCounts* parses the line counts file to retrieve persisted line counts data. If revisions are still missing line count data, the RepositoryFileManager once again helps with retrieving the data. This involves accessing the SVN server in order to perform a "diff" operation (*getLineDiff*). At the end, the line counts and binary status files are resaved to make the newly collected line counts data and binary status data available to the next execution of StatSVN (*writeLineCountsFile* and *writeBinaryStatusFile*).

During the fourth step, line count data is deemed missing and a "diff" operation will be performed if

a) no line count data was retrieved from the line counts file,
b) the file is not binary,
c) the revision is not a "delete" action,
d) an earlier revision of the file exists, and
e) the earlier revision of the file is not a "delete" action.

The first step *getBinaryFiles* can only identify binary files which still exist in the working copy (i.e. they have not been deleted). Therefore, the binary status is persisted to identify binary files even if they do not exist in the current working directory. It, however, is still possible that a "diff" operation may be attempted on a deleted binary file. In this case, the "diff" operation fails and the file is logged in the binary status file as potentially binary. Once a file has been identified a certain number of times as potentially binary, the file is assumed to be binary and the "diff" operation will not be attempted anymore.

IV. Validation

In order to verify that the statistics generated by StatSVN were correct, we compared its output with that of StatCVS for the same repository. We migrated the jUCMNav [10] repository from CVS to SVN using the cvs2svn script [4] and tested the output of both tools on the 100 KLOC project. The results were identical except for slight differences in line counts for some files. Investigating further, we discovered that the SVN "diff" operation does not produce exactly the same information as the line counts provided in the CVS log. The SVN "diff" operation produces minimal differences whereas the "diff" operation used by CVS log does not.

In addition, we tested on a few other SVN repositories to verify our parser on varying input sources. However, more extensive tests need to be performed, especially on large repositories, to verify our tool's correctness and performance implications.

V. Conclusion

StatSVN provides LOC metrics for software projects that use SVN for configuration management. Given time constraints imposed on our project, we are proud of the open source extension we have built. We limited our work to replicating StatCVS's behavior but have developed the groundwork for future work on SVN-specific enhancements. We, therefore, have achieved our main and three secondary goals identified in section I.

The main enhancement to be made is to move away from the simplistic view of copy/rename/replace actions and take advantage of SVN's intrinsic support for these operations. While our modifications in this iteration were restrained to the input layer, this improvement will impact the whole system. We have also identified performance improvements which will be listed on the forthcoming website for the StatSVN project.

Once our tool has been extensively tested, the merger of StatCVS and StatSVN should be considered. Although support for both configuration management systems might not be of great value to the end users, their merger will greatly simplify the engine's evolution from a developer's standpoint.

References

[1] D. Amyot, "Introduction to the User Requirements Notation: Learning by Example," *Computer Networks*, vol. 42:3, pp 285-301, 21 June 2003.
[2] Chora website. http://www.horde.org/chora (accessed March 2006).
[3] Concurrent Versions System website. http://www.nongnu.org/cvs (accessed March 2006).
[4] cvs2svn website. http://cvs2svn.tigris.org/ (accessed March 2006).
[5] CVSAnalY website. http://libresoft.urjc.es/Tools/CVSAnalY (accessed March 2006).
[6] CvsWeb website. http://www.freebsd.org/projects/cvsweb.html (accessed March 2006).
[7] GForce website. http://dforge.cse.ucsc.edu/svn/?group_id=63 (accessed March 2006).
[8] *Goal-oriented Requirement Language (GRL)*, URN Focus Group – Draft Recommendation Z.151, Geneva, Switzerland, September 2003.
[9] J. Kealey, E. Tremblay, J.-P. Daigle, J. McManus, O. Clift-Noël, and D. Amyot, "jUCMNav: une nouvelle plateforme ouverte pour l'édition et l'analyse de modèles UCM," *5ième colloque sur les Nouvelles Technologies de la Répartition (NOTERE 2005)*, pp 215-222, August 2005.
[10] jUCMNav website. http://jucmnav.softwareengineering.ca/twiki/bin/view/ProjetSEG/WebHome (accessed March 2006).
[11] MPY SVN Stats website: http://mpy-svn-stats.berlios.de/ (accessed March 2006).
[12] Polarion website. http://www.polarion.com/ (accessed March 2006).
[13] StatCVS website. http://statcvs.sourceforge.net (accessed March 2006).
[14] Subversion Issue 2206. http://subversion.tigris.org/issues/show_bug.cgi?id=2206 (accessed March 2006).
[15] Subversion website. http://subversion.tigris.org (accessed March 2006).
[16] *Use Case Map Notation (UCM)*, URN Focus Group – Draft Recommendation Z.152, Geneva, Switzerland, September 2003.
[17] *User Requirements Notation (URN) – Language Requirements and Framework*, ITU-T Recommendation Z.150, Geneva, Switzerland, 2003.
[18] ViewVC website. http://www.viewvc.org (accessed March 2006).