



university of
groningen

faculty of mathematics
and natural sciences

Adding Git support to SolidSTA for analysing C# metrics and dependencies

Software Maintenance and Evolution

Authors:

T. Back (s3218147)

K.Y. Kliffen (s2369494)

Supervisor:

prof. A. Telea

November 13th, 2016

Contents

1	Introduction	2
2	Related Work	2
3	Architecture Design and Decisions	2
3.1	Source Code Management System	2
3.2	Git Library	3
3.3	Abstract Importer	3
3.4	Improvement of the Abstract Importer	4
3.5	Vertical and Horizontal Storage Structure	5
3.6	Memory Limits	6
3.7	Hard Disk Space	6
3.8	Local Processing	7
3.9	Various Fixes	7
4	Results	8
5	Conclusions	9
6	Future Work and Limitations	9
	References	11
A	SCMServer Abstract Interface	12

1 Introduction

Throughout the life of a software, it is under continuous change. Over time its complexity increases [8] and therefore is more difficult to maintain. In the maintenance process, tools are used to gain a deeper understanding of a software project and its evolution.

There are multiple software analysis tools out there, but none of them offer a complete set of analysis tools. For this project we improve an existing tool (**SolidSTA**) to support the source code management system Git.

Git is the most used version control system [5, 11] for source code at the moment and used for many software projects to organize their code base. Therefore, to analyse also current software projects, support for the most used version control system is needed.

This report is structured as follows. First, the existing versions of the **SolidSTA** tool are discussed. Secondly, the changes to the architecture to support Git are explained. Finally, the results and performance of the new **SCMImporter** utility are discussed and some limitations and future improvements are described.

2 Related Work

In this report we use an extended version of the Solid Sourcecode Trend Analyzer (**SolidSTA**) [1]. The original tool allows the import of SVN and Git repositories as well as performing project and file metrics using static code analysis.

An extended version of the Solid Sourcecode Trend Analyzer uses an utility program called **TFSImporter** to connect to Microsoft's Team Foundation Server (TFS) repositories and calculate extensive code metrics for C# projects [7] and source code dependencies [9]. However, the new metrics are only available for TFS repositories since the analysis is done inside the **TFSImporter**. **SolidSTA** communicates with the command line application **TF-Importer** through an basic text based interface. The dependencies are visualized with an external program called Solid Software Explorer (**SolidSX**).

3 Architecture Design and Decisions

In this section we explain the existing architecture of the **TFS Importer** and how we intend to change it to extend the number of repository types accepted by the application. First, a short description of a Source Code Management System is given. Secondly, the used library is described. Thirdly, an architecture is proposed. Finally, the difference between different kinds of repositories is discussed in terms of querying for changes.

3.1 Source Code Management System

In software development, it is common practice to use a Source Code Management System (SCMS) or a Version Control System (VCS) to collaborate on software projects. In this report

the terms may be used interchangeably.

Examples of such systems are: Git, Mercurial, SVN and Team Foundation Server (TFS).

Most systems have a notion of commit or change set. This is a moment in time when a developer saves a status and thus changes of files to the SCMS. It contains the time of the change, the author of the change and the changed files. Often they also contain a description of the change in a short message.

3.2 Git Library

The version 1.0 of **SolidSTA** requires a native installation of the Git command line tool. To put less dependencies on the environment, we incorporate the Git functionality in the utility using the existing library *LibGit2Sharp* [3]. The library is written in C with a proper C# binding [4], which is needed since the **TFSImporter** is written in C#.

The *LibGit2Sharp* library allows for cloning of a repository and storing it for offline use. This prevents performance issues due to network connectivity, since the files are stored locally on disk. Furthermore it allows for querying commits and changed files per commit. It is also possible to get the file contents of a file, without checking out a given version of the repository.

3.3 Abstract Importer

The extended **SolidSTA** application consists of three major parts:

- **SolidSTA**, graphical user interface for displaying metrics of projects, files as well as evolutions of metrics and user interaction
- **CSharpMetrics**, a library for static code analysis using NRefactory [6]
- **TFSImporter**, an utility for connecting to repositories on the Team Foundation Server

SolidSTA is a python program, which uses the **TFSImporter** utility to connect to a **TFS** repository. This communication is done via a command-line interface. The **TFSImporter** uses the C# metrics library to calculate the code metrics on the project. To support other repositories, the binding between the **TFSImporter** and the **CSharpMetrics** must be abstracted to an *Abstract Importer*.

3.4 Improvement of the Abstract Importer

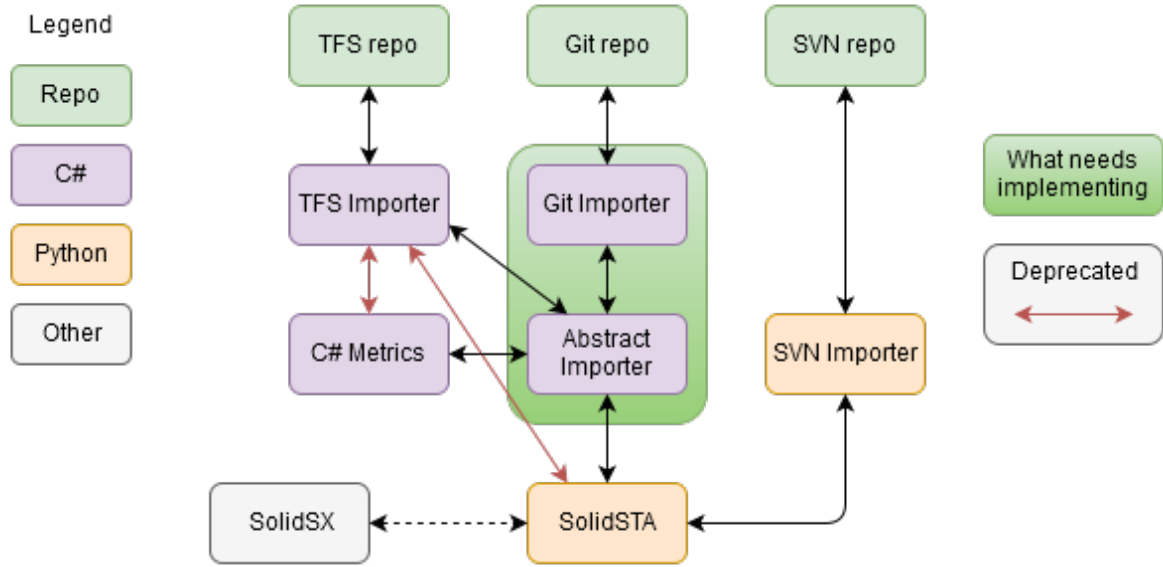


Figure 1: The architecture of the **SCMImporter**

The Figure 1 shows the architecture of the **SCMImporter**. Starting point is **SolidSTA**, which will issue queries to the *Abstract Importer*. In version 1.0 of **SolidSTA** calls the TFS Importer directly. In the new improved version, **SolidSTA** will call instead the Abstract Importer, which provides a transparent interface to Git and TFS repositories. It will also do the metrics calculation.

To start off, the *Abstract Importer* needs a basic interface. It will be the base for all variations of repository importers. Some basic data structure is already given through the **TFSImporter**, which will be used by all implementations of repository specific importers. This data structure is abstracted from the repository and library specific data structures.

Because the **TFSImporter** is a working importer for TFS repositories, the code base of that project is used to extend the functionality. The **TFSImporter** is written in plain C# and uses another C# library. The library *CSharpMetrics* is used for the extraction of software metrics of C# projects. Using another programming language to extend the software or re-engineer the existing software would need significantly more work and will possibly not add the same detailed metrics as with the C# code. Therefore, the support for other repository types will be added to the **TFSImporter** making it the more general **SCMImporter**.

The center of this project is the abstract class *SCMServer* and its implementation for Git. The complete interface is listed in Appendix A.

All of the non-primitive types (such as commits) are defined within the **SCMImporter** and do not have dependencies on other libraries. Wherever necessary a converter method is defined to convert from a specific type used by one of the specific importers to our defined types and also back. This is done for example for Microsoft's TFS library and also for *LibGit2Sharp*, since their libraries use some library specific object types for representing

commits.

3.5 Vertical and Horizontal Storage Structure

In the project we found out that different SCM tools order their data in different kinds of ways. Both types are explained in the following list and also shown in Figure 2.

- Horizontal
In a horizontal data structure the commits are organised around the files. So, when a commit is made, the change is saved at the individual files. In this way, the history of a file can be easily retrieved, whereas its more difficult to see which files have changed at specific commits or moments in time.
- Vertical
In a vertical data structure the basic structure are the commits. It's easy to visualise which files have changes at specific moments in time, but its rather expensive to list how individual files have changed in the duration of the project.

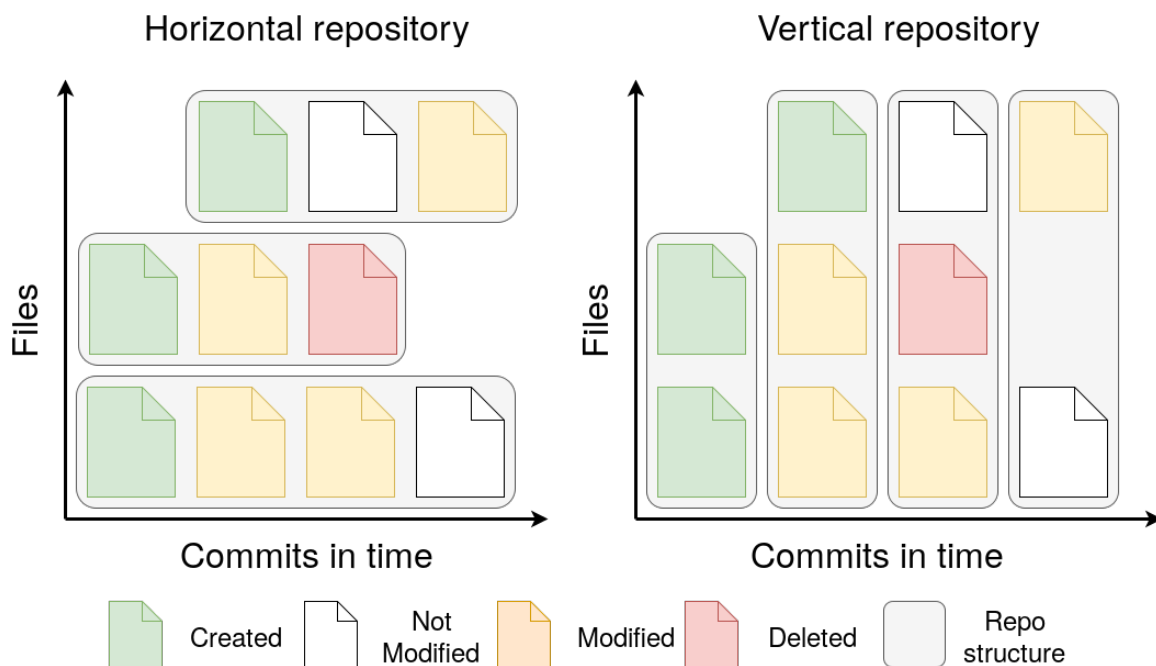


Figure 2: Difference between horizontal and vertical data structure

Whereas Microsoft's TFS uses a horizontal structure, Git uses a vertical structure. This is in general no problem, because it does not matter in which direction a two-dimensional data structure is queried; the result is the same. However, it does concern the speed of the application. Algorithms are usually optimised for a specific data structure and the **TFSImporter** is no exception from that.

By implementing a second version of the querying algorithms, which only changed the way of running through the data, the speed was improved for example for the GIMP project by

an expected factor of $60\times^1$. Thus, the algorithm does matter and is selected based on the SCM data structure (`GetRepoMode()` method in the **SCMServer** interface).

3.6 Memory Limits

In the process of reading in the file contents, we run into an issue with the maximal amount of memory. As it turns out when we read a file from the Git repository - the library returns as expected a `Stream` object - however the `Stream` does not refer to file in the repository with some logic to read that correct file information (lazy loading), but instead the `Stream` is preprocessed and loaded into the memory at request.

When requesting all the file contents of all files in a large repository, a very large amount of memory is consumed - even exceeded the amount that the `LibGit2Sharp` library [4] can handle. The memory issue arises as soon as the memory size exceeds $2^9 + 2^{10} = 1536\text{Megabytes}$. We don't know why it is exactly this number, but since we know about it, we can work around it.

To circumstanced this issue, the **SCMImporter** does process the commits in chunks. We decided to process 100 commits per chunk. This has been decided to be able to handle repositories with large commits, but also being able to process the repository fast by performing batch database insertions. So, a trade-off was found between reliability and speed.

3.7 Hard Disk Space

Before the **SCMImporter** can import any data of the Git repository, the Git repository has to be cloned first. Within the cloning process the whole history of the selected project's primary branch and the content of the files is received and saved to disk. This is the only Git specific extra amount of hard disk space needed.

Additionally, the file contents command copies every source code file change in a local zipped file. So, each file version of supported code files is saved as well. Since different versions of the same file may have a lot of code in common, it can be compressed at a high ratio.

Last, but not least for the calculation of the dependencies, the revisions are checked out, which also take some space during the calculation, since it checks out a given version and performs the static dependency analysis.

Since the TFS part was left untouched, the hard disk requirements are still the same as with the **SolidSTA** for TFS.

¹A projected run would take about 36 hours in horizontal mode and its now down to around 34 minutes in vertical mode

3.8 Local Processing

After the repository was cloned once, with every new run of the **SCMImporter** a fetch is done to include the latest updates. After this, all processing is done offline on the local machine. It's also possible to clone the repository once, then cut the Internet connection and continue to run the **SCMImporter**.

3.9 Various Fixes

Throughout the project we encountered also various other issues on a smaller scale. The ones we feel worth mentioning are listed in the following subsections.

3.9.1 Git Project Clean-up

When cloning a Git repository, all Git related files are marked read-only. **SolidSTA** could not delete these files when the project would be deleted. A small change in the python code was made to force the Git files to be writable and to delete the files then.

3.9.2 Line of Codes Metric

Unfortunately, the line of code (LOC) metric from the **CSharpMetrics** returned a zero value for every file. Since we rely for the analysis of the metric on **NRefactory**, we implemented a basic LOC counter manually in the **SCMImporter** ourselves when metric calculation is requested.

3.9.3 Representation of File Path

After adding the first version of the functionality for the support of git repositories and inserting some basic data into the database, we wondered why these data was not getting displayed in **SolidSTA**. After looking through the code over and over again as well as the structure of the database and its links between the tables, we did not find anything strange. In the end it turns out, that the *Path* in the *Files* table has to follow a very specific format and is not platform independent. All paths have to start with a (forward) slash and may not contain a backslash. The format can be described with: *//folders//filenameWith.Extension*

3.9.4 Unicode Support in Python

Some text displaying functions did not work in **SolidSTA** when the text contained diacritics such as é. This problem was found when a repository was used with foreign names containing these characters. This was fixed by forcing the use of Unicode text in these functions.

4 Results

Besides adding the functionality, also some performance testing was made to evaluate the speed of the new *Abstract Importer*. Since a previous version of **SolidSTA** also had already support for Git repositories integrated, a comparison can be made as well.

For the tests, we decided to run them on two selected repositories:

1. **GIMP**

GIMP [2] is an open-source graphics editing software with almost 20 years of commit history.

2. **LibGit2Sharp**

LibGit2Sharp [4] is the git library that used in the *Abstract Importer* for the usage of Git. It is a bit smaller than GIMP.

The performance of the *Abstract Importer* was evaluated on the following system: Intel Core i5 4570 (3.20 GHz) with 16 GB ram, running with a 5400 RPM hard drive. Version 1.0 of **SolidSTA** was used to compare with the new **SCMImporter** utility. The **SCMImporter** was timed using .NET build-in timer objects and storing the timings inside a file on execution. **SolidSTA** version 1.0 was timed using a stopwatch and executing the different commands by hand. The results for the **SCMImporter** can be seen in Table 1 and for the old **TFS importer** in Table 2.

Repository	# commits	filetree	version info.	version contents	file metrics
GIMP[2]	37,563	4.2 s.	3 m. 20.0 s.	34 m. 2.3 s.	-
LibGit2Sharp[4]	2,109	0.4 s.	18.0 s.	48.8 s.	20.3 s.

Table 1: Performance of **SCMImporter** on different sizes of Git repositories

Repository	# commits	filetree	version info.	version contents	file metrics
GIMP	37,563	< 1 s.	1 h. 30m.	3h. 50m.	-
LibGit2Sharp	2,109	< 1 s.	3 m. 6 s.	-	-

Table 2: Performance of v1.0 **SolidSTA** on different sizes of Git repositories

Some values in both tables are missing, since metrics can only be calculated on C# projects and would otherwise not be comparable between versions. When testing the LibGit2Sharp repository with the old version of **SolidSTA**, it was not possible to perform the content operation, due to some error in the old program, which we could not fix.

Since we use the same data as a basis for measuring the performance of both programs, a comparison can be made. A first look at the numbers already reveals that the new version with the *Abstract Importer* is faster than the previous version. Whereas the time difference for querying all files that are present in the project is quite fast in both versions, the difference in speed is significant for the other tasks. The new importer is probably slower, since version 1.0 of **SolidSTA** requests only the latest files in the repository and does not read the full history of every file that exists. The processing time for gathering just the

information about all commits with times and commit messages has been greatly reduced. This is due to the fact, that the repository is kept locally and no heavy computational operations are necessary. When running the version 1.0 of **SolidSTA** version, it was clear that versions were calculated for each file individually, so a horizontal method was used. This explains the difference in processing time taken by the new implementation. The file contents command involves more steps and is especially resource demanding on IO. But an improvement by the factor of around 8 was achieved by the new implementation, which was also caused by the same horizontal method.

The interface between **SolidSTA** and **SCMImporter** was not changed. It should be possible for a user of **SolidSTA** to use a Git C# repository without needing specific configuration compared to the TFS repositories.

5 Conclusions

In our project we did not focus on Software Maintenance and Evolution in general, but had a rather hands-on experience of program understanding for an existing project and improve it with new functionality. For minimal change impact, we used reverse engineering to provide an abstract interface for different kinds of repositories - which was mostly already provided by the **TFS Importer**. This will allow for easy extension for new kinds of repository in the future.

We did however encounter a difference in execution time depending on the repository structure and query methods. Therefore, we had to define two methods for each kind of repository to be queried efficiently. The execution time of version information and version contents was decreased significantly compared to original **TFS Importer**.

6 Future Work and Limitations

In this section we list some possible improvements and limitations of the **SCMImporter** for **SolidSTA**.

- SVN support is not yet implemented. Using the open source library SharpSVN [12], it should be able to use a similar vertical approach to import from SVN repositories as with Git repositories.
- The LibGit2Sharp library has an issue regarding reading the changes of a project when a new file consisting only of empty lines (end of line characters)² is committed. This issue is fixed in the current version of the library (v0.22), however a new memory leak was introduced, which affects the **SCMImporter** even more. Unfortunately, the thrown `OutOfMemoryException` is not catchable, which makes it impossible to handle the issue. Since LibGit2Sharp is still under development, we hope that this issue will

²This can be observed in the Roslyn repository at <https://github.com/dotnet/roslyn>. An issue regarding this error can be found at <https://github.com/libgit2/libgit2sharp/issues/979>

be fixed in future versions. But until then, the **SCMImporter** is limited to those repositories, which do not contain these kind of empty files.

- To work in conjunction with the Solid Software Explorer, administrator rights are needed. This is necessary because the Solid Software Explorer saves data to its own program directory. In the newer versions of Windows, these folders are write-protected for unprivileged programs and therefore administrator rights are needed for **SolidSTA** when starting **SolidSX**.
- All commits are queried for files that were added, modified or deleted per commit to construct a complete filetree. When a file is deleted however, it still exists in the repository as a file that was never changed after it's deletion. In a future version, the line corresponding to deleted files may end at the commit when it was deleted. This might yield interesting insight into when and how files are deleted during the project.
- With the newly calculated metrics, it might be interesting to use a 2D projection of the high dimensional space from these metrics to visualize evolution of code metrics between commits [10].
- The current version of **SCMImporter** clones only the primary (usually master) branch of a project. A future version might allow for selecting certain branches to be analysed.

References

- [1] SolidSource BV. *SolidSTA tool distribution*. 2008. URL: www.solidsourceit.com.
- [2] Gimp contributors. *GIMP repository*. URL: <https://github.com/GNOME/gimp> (visited on 11/03/2016).
- [3] Libgit2 contributors. *Libgit 2 website*. URL: <https://libgit2.github.com/> (visited on 11/03/2016).
- [4] Libgit2sharp contributors. *Libgit2sharp git repository*. URL: <https://github.com/libgit2/libgit2sharp> (visited on 11/03/2016).
- [5] Google Trends. *Git, Concurrent Versions System, Apache Subversion, Mercurial - Explore - Google Trends*. 2016. URL: https://www.google.com/trends/explore?q=%5C%2Fm%5C%2F05vqwg,%5C%2Fm%5C%2F09d6g,%5C%2Fm%5C%2F012ct9,%5C%2Fm%5C%2F08441_&hl=en-US&tz&tz (visited on 11/06/2016).
- [6] ICSharpCode. *NRefactory repository*. 2010. URL: <https://github.com/icsharpcode/NRefactory> (visited on 11/04/2016).
- [7] Joost Koehoorn. *Software evolution analysis for Team Foundation Server*. 2013. URL: <http://irs.ub.rug.nl/dbi/520a16b64a989>.
- [8] *Metrics and Laws of Software Evolution—The Nineties View*. IEEE. 1997. URL: <http://www.ece.utexas.edu/~perry/work/papers/feast1.pdf>.
- [9] Robbert-Jan Pijpker and Joost Koehoorn. *Report for Software Maintenance and Evoltion*. 2014.
- [10] RRO da Silva et al. "Metric Evolution Maps: Multidimensional Attribute-driven Exploration of Software Repositories". In: ().
- [11] Ian Skerrett. *Eclipse Community Survey 2014 Results*. 2014. URL: <https://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/> (visited on 11/04/2016).
- [12] Sharp SVN. *Sharp SVN website*. URL: <https://sharpsvn.open.collab.net/> (visited on 11/03/2016).

A SCMServer Abstract Interface

The **SCMServer** is defined as the following interface:

```
1 abstract class SCMServer {
2     public abstract void Connect(string url, string directory, string user, string
        password);
3     public abstract string PreparePath(string path);
4     public abstract VersionSpec GetFirstVersionAfter(DateTime dt);
5
6     public abstract IEnumerable<ChangeSet> QueryFileHistory(string path, RecursionType
        recursion, VersionSpec versionFrom, int maxCount, bool includeChanges, bool
        includeDownloadInfo, bool sortAscending);
7     public abstract Item[] GetItems(string path, VersionSpec version, RecursionType
        recursion, DeletedState deletedState, ItemType itemType, bool
        includeDownloadInfo);
8
9     public abstract RepoOperationMode GetRepoMode();
10 }
```