Travel n Study
SVN Policy

Version 1.0

Revision History

Date	Version	Description	Author
2012-10-25	1.00	First version	Javier Hualpa

Contents

1. Introduction

1.1 Purpose of this document

The purpose of this document is to communicate the set of practices to share working documents and code on the SVN assigned for the project.

1.2 Document organization

The document is organized as follows:

- Section 1, *Introduction*, describes the contents of this policy and its purpose.
- Section 2, *File Naming Convention*, explains techniques for achieving maximum results when working with documents.
- Section 3, *Subversion (SVN)*, explains rules to represent the logical structure of the code accurately and consistently.

1.3 Intended Audience

The intended audience is:

- All team members, supervisors and customers of the StudyAbroad project.

1.4 Scope

This document addresses resolutions regarding SVN folder layout structure, best practices for working collaboratively with code and file naming conventions used for sharing documents.

1.5 Definitions and acronyms

1.5.1 Acronyms and abbreviations

Acronym or abbreviation	Definitions
SVN	Subversion, a software versioning and revision control system

1.6 References

[1] SVN: http://en.wikipedia.org/wiki/Apache_Subversion

2. File Naming conventions

2.1 Naming rules

Below we present the basic rules that will serve as a guideline in structuring folder and file naming conventions:

- Use the underscore (_) as element delimiter. Do not use spaces or other characters such as: ! # $ % & ' @ ^ ` ~ + , . ; =) (

  Do: BillGates_AccountBalance.pdf
  Do Not: BillGates AccountBalance.pdf

  Reason: The underscore (_) is a quasi-standard for field delimiting and is the most visually ergonomic character. Some search tools do not work with spaces and should be especially avoided for internet files. Other characters may be interesting but visually confusing and awkward.

- Use the hyphen (-) to delimit words within an element or capitalize the first letter of each word within an element.

  Do: Bill-Gates_Account-Balance.pdf
  Do Not: Bill Gates Account Balance.pdf

  Reason: Spaces are poor visual delimiters and some search tools and file systems have some issues with spaces. The hyphen (-) is a common word delimiter. Alternatively, capitalizing the words within an element is an efficient method of differentiating words but is harder to read. Abbreviations and acronyms must be on capital letters.

- The order of importance rule holds true when elements include date and time stamps. Dates should be ordered: YEAR, MONTH, DAY. (e.g. YYYYMMDD, YYYYMMDD, YYYYMM). Time should be ordered: HOUR, MINUTES, SECONDS (HHMMSS).

  Do: BillGates_AccountBalance_1998-12-20
  Do Not: BillGates_AccountBalance_Dec-20-1998

  Reason: To ensure that files are sorted in proper chronological order the most significant date and time components should appear first followed with the least significant components.

- An element for version control should start with V followed by at least 2 digits and should be placed as the last most elements. To distinguish between working drafts (i.e. minor revisions) use Vx-01->Vx-99 range and for final draft (i.e. major version release) use V1-00-> V9-xx. (where x =0-9)

  Do: Project_Plan_V09.doc, Project_Plan_V1-02.doc
  Do Not: Project_Plan_09.doc || Project_Plan_V2FinalDraft.doc

  Reason: The “V” helps denote that the element pertains to a version number. A minimum of 2 digits with a leading zero is required to ensure that search results are properly sorted. The intent is to avoid the situation where for example, a filename with a “V1-13” will wrongly appear before an identical filename with a “V1-2” version number when sorted in ascending alphabetical/numerical order. To distinguish between working, review and final draft a single digit prefix followed by hyphen “-” is preferred to facilitate proper sorting; using words in the file name such Final, Draft or Review in the filename affect the order and should be avoided. So the “.” is not meant to be used for versioning just for file name and file extension separation.

2.2 File Naming Standard

All the files names used for the project documents must adhere to the following pattern:

- For most documents: **ProjectName_DocumentName_Date_Version**
  Example for project vision presentation: Travel_ n_ Study_ProjectVision_2012-10-19_V1-00.ppt

- For minutes of meeting documents: **ProjectName_DocumentName_Date**
  Example for minutes of meeting: Travel_ n_ Study_MoM_2012-10-19.doc

- For week reports: **ProjectName_DocumentName_Week_MemberName**
  Example: Travel_ n_ Study_WeekReport_41_JavierHualpa.doc

- For summary week reports: **ProjectName_DocumentName_WeekNumber**
  Example: Travel_ n_ Study_SummaryWeekReport_41.doc

- It is recommended to specify the date as YYYY-MM-DD.

- It is mandatory to specify the project name, document name (except for Minutes of Meeting) and member name as verbose as possible.

2.3 File Format Standard

The document format for all the documents generated is the Microsoft format, this means that we will work with .doc and .ppt files offline and on SVN. When submitting documents to the DSD web site we will work with .pdf and .ppt files. This means that the .doc files should be converted to .pdf.

3. Subversion

The SVN repository for Travel_ n_ Study can be browsed via **svn://lapis.rasip.fer.hr/svn/dsd12/StudyAbroad** or **https://lapis.rasip.fer.hr/svn/dsd12/StudyAbroad**. You may have to provide the credentials that were sent to your personal email. The proposed SVN client is Tortoise (http://tortoisesvn.net/downloads.html) because it features a graphical and command line interface.

3.1 Directory Layout

The project will use the mainline model software development lifecycle which consist essentially in a main codeline that forms the trunk from which be branch release and development codelines.

The SVN repository layout is as follows:

- Repository_Root
  - branch
  - tags
  - trunk
    - Docs
      - Documentation
      - Presentations
      - WeekReports
      - MinutesOfMeeting
      - Policies
      - Temp
    - Code
      - To be determined

Essentially it contains one project root, from which we manage documentation artifacts and code artifacts.
The structure of the “Docs” directory mirrors the structure of the DSD web site folders, this way we can work

on drafts documents and review them before uploading the final versions to the DSD web site. Additionally there is a "Temp" folder where team members can store whatever content they need or that does not fit into the existing folders such as temporary documents, training material, research material, etc.
The structure of the "Code" directory will be determined as we dig deeper into the architecture of the solution.
The project root follows the standard layout convention, in which each project root contains:

- trunk subdirectory: The main body of development, originating from the start of the project until the present. It is the development workspace; this is where all development should happen.

- branches subdirectory: A branch is usually used to do something away from the trunk (or other development line) that would otherwise break the build. New features are often built in a branch and then merged back into the trunk. Usually used for major version like 1.0, 1.5, 2.0, etc. Then when you release you tag the branch. This allows you to continue to support a production release while moving forward with breaking changes in the trunk

- tags subdirectory: Contains snapshots of the repository at a particular time. No development should occur on these. They are most often used to take a copy of what was released to a client so that you can easily have access to what a client is using.

3.2 Best Practices

- Never commit code that doesn't compile: Compile the code and correct all errors before committing. Make sure that newly added files are committed. If they are missing your local compile will work fine but everybody else won't be able to compile.

- Test your changes before committing: Run unit and regression tests, if available.

- Double check what you commit: Do a svn update and a svn diff before committing. Take messages from SVN about conflicts, unknown files, etc. seriously. svn diff will tell you exactly what you will be committing. Check if that's really what you intended to commit.

- Always add descriptive log messages: Log messages should be understandable to someone who sees only the log. They shouldn't depend on information outside the context of the commit. Try to put the log messages only to those files which are really affected by the change described in the log message. In particular put all important information which can't be seen from the diff in the log message.

- Honor the project coding policy: Check the corresponding document for this, if you want automated support you can too.

- Announce changes in advance: When you plan to make changes which affect a lot of different code in SVN, announce them in advance. For instance, changes in libraries might break other code even if they look trivial, because an application must also compile with older versions of the library for some reasons. By announcing the changes in advance, developers are prepared, and can express concerns before something gets broken.

- Take responsibility for your commits: If your commit breaks something or has side effects on other code, take the responsibility to fix or help fix the problems.

- Backport bugfixes: If you commit bugfixes, consider porting the fixes to other branches. Use the same comment for both the original fix and the backport, that way it is easy to see which fixes have been backported already.

- Use bug tracking system numbers: If you fix bugs reported on the bug tracking system, add the bug number to the log message. In order to keep the bug tracking system in sync with SVN, you should reference the bug report in your commits, and close the fixed bugs in the bug tracking system.

- Tags and branches: Branches and release tags will be created by the SVN manager in the “branches/studyabroad” and “tags/studyabroad directories”. The SVN manager should place all branches which are aimed to be released in “branches/studyabroad” and name them like the release, e.g. “branches/studyabroad/1.0”. For all release tags, “tags/studyabroad” is the right place. All temporary working branches (which should be deleted after work has completed) should be located in “branches/work” with some name describing which work is done there.

- Don't add generated files to the repository: Files generated at build time shouldn't be checked into the repository because this is redundant information and might cause conflicts. Only real source files should be in SVN.

- Commit complete changesets: SVN has the ability to commit more than one file at a time. Therefore, please commit all related changes in multiple files, even if they span over multiple directories at the same time in the same commit. This way, you ensure that SVN stays in a compileable state before and after the commit, that the commit history (svn log) is more helpful. Commits should be preferably atomic and not splittable. That means that every bugfix, feature, refactoring or reformatting should go into an own commit. This, too, improves the readability of the history.

- Don't mix formatting changes with code changes: Changing formatting like indenting or white spaces blows up the diff, so that it is hard to find code changes if they are mixed with re-indenting commits or similar things when looking at the logs and diffs later. Committing formatting changes separately solves this problem.