The Freescale Cup

- Software development proccess overview -
- Configuration management -
- Versioning control system -
- TortoiseSVN tool -
- Naming conventions -

- Software development proccess overview -

- **How do you define process?**
- A process is a set of practices performed to achieve a given purpose (in this case to develop software); it may include tools, methods, materials, and/or people.

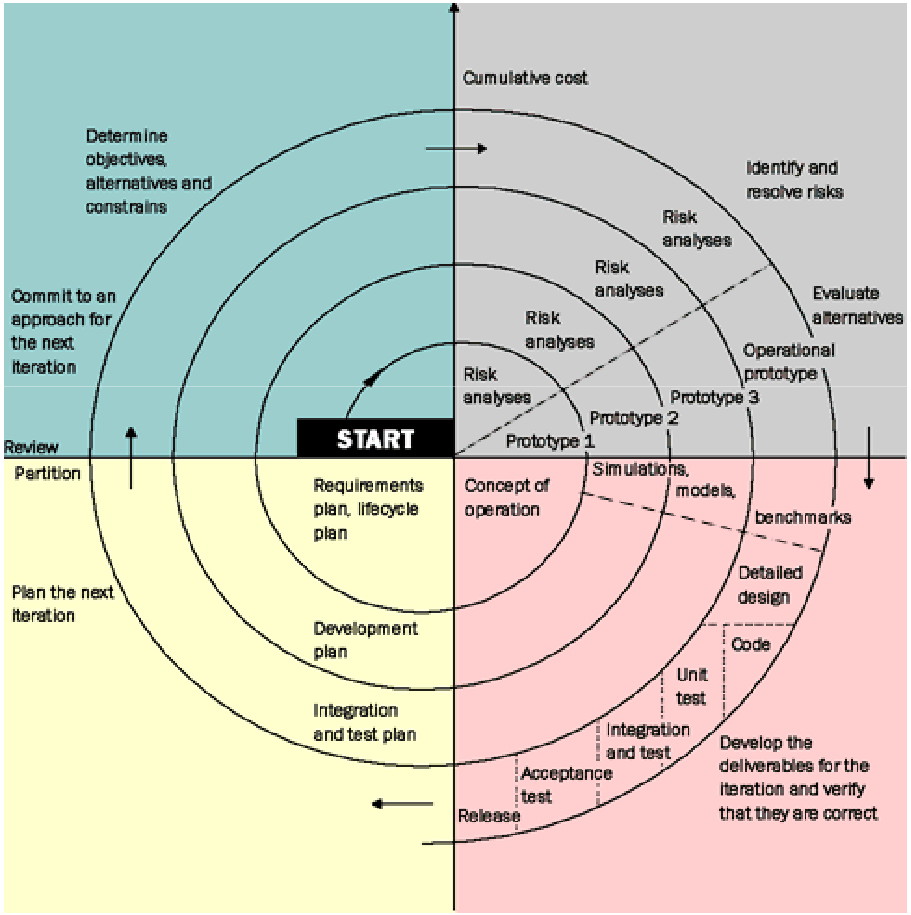
- **What Is a Process Model?**
- A model is a structured collection of elements that describe characteristics of ***effective processes*** (included are those proven by experience to be effective***)***.

- A model provides
  - **a place to start**
  - **the benefit of a community's prior experiences**
  - **a common language and a shared vision**
  - **a framework for prioritizing actions**

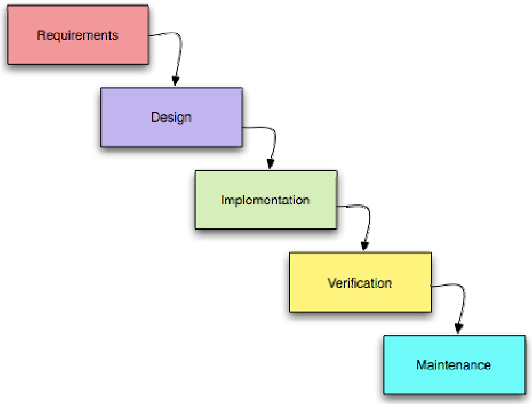
- Software development proccess overview -

- Some examples of development model:

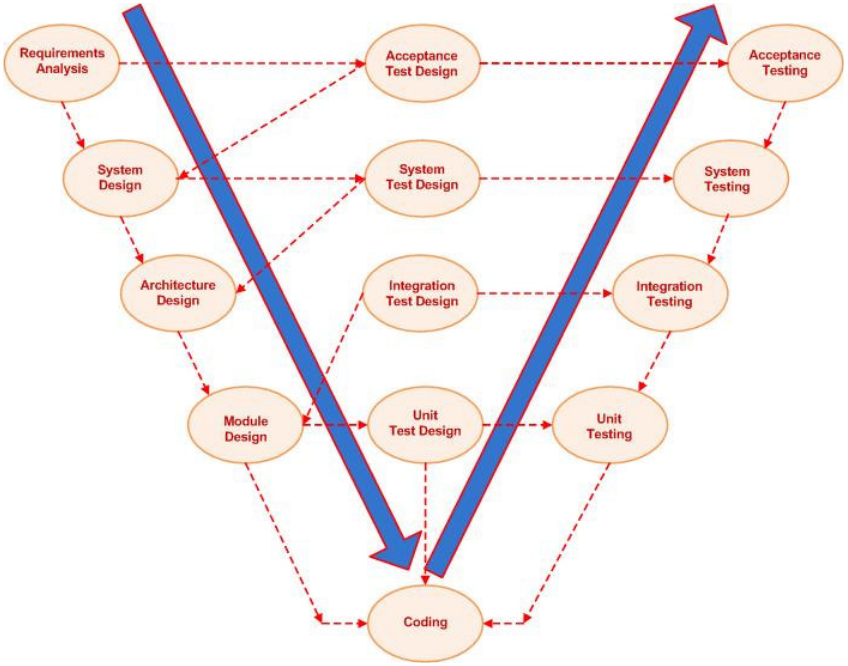
- Spiral model



- Waterfall model

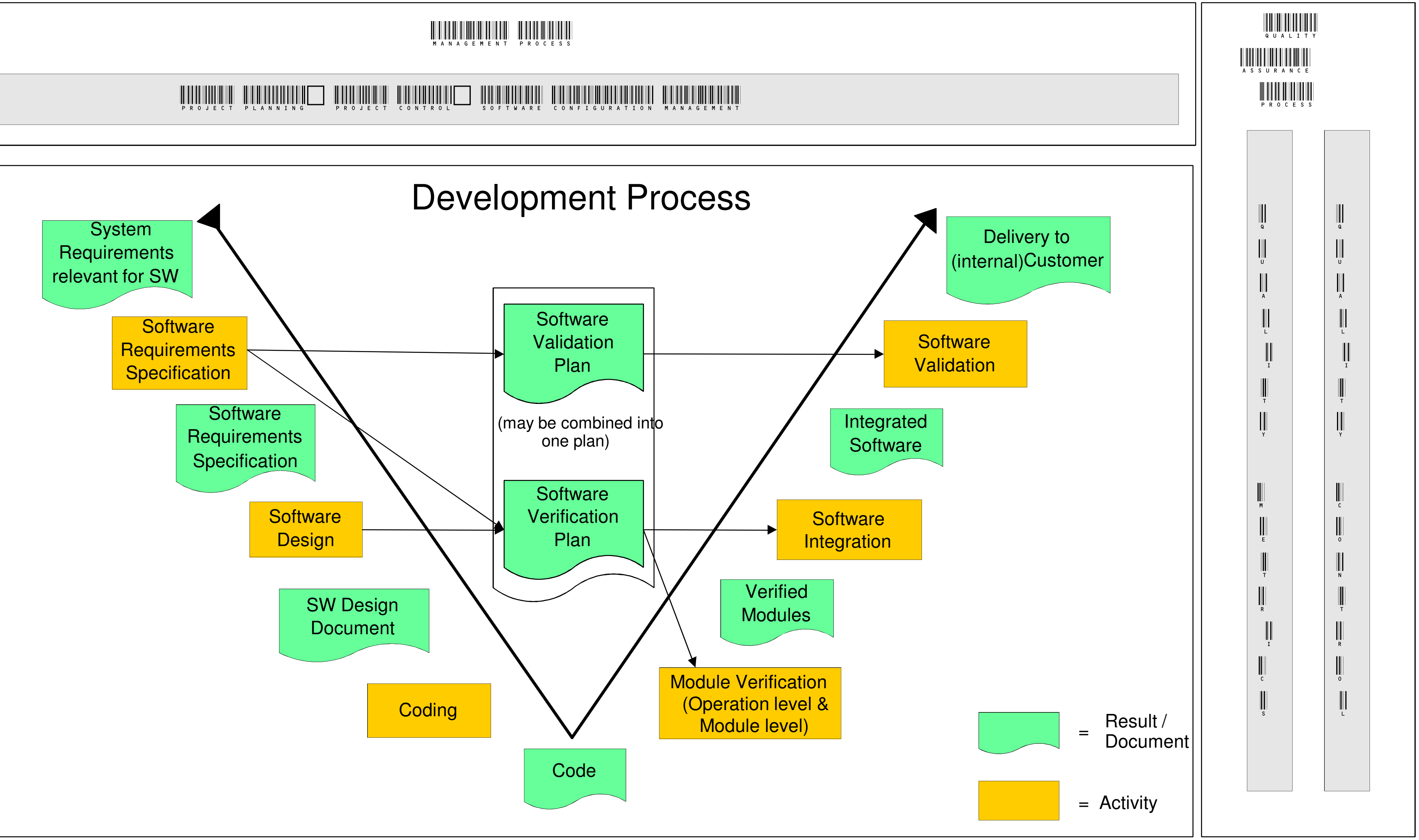


- "V cycle" model



- Software development proccess overview -

A quick look to the "V cycle" model



MANAGEMENT PROCESS

PROJECT PLANNING □ PROJECT CONTROL □ SOFTWARE CONFIGURATION MANAGEMENT

QUALITY ASSURANCE PROCESS

QUALITY METRICS

QUALITY CONTROL

Development Process

System Requirements relevant for SW

Software Requirements Specification

Software Requirements Specification

Software Design

SW Design Document

Coding

Code

Software Validation Plan

(may be combined into one plan)

Software Verification Plan

Delivery to (internal)Customer

Software Validation

Integrated Software

Software Integration

Verified Modules

Module Verification (Operation level & Module level)

🟩 = Result / Document

🟨 = Activity

- Configuration management -

- The purpose of *Software Configuration Management* (SCM) is to establish and maintain the integrity of the outputs of the software project throughout the life cycle.

- The three parts of (Software) Configuration Management
  - Version control
  - Change control
  - Build control

- **Activities** of SCM:
  - **identifying** and defining the **configuration items** of a system
  - **controlling** the **release** and **change** of all configuration items
  - **recording** and reporting the **status** of configuration items and **change requests**
  - **verifying** the **completeness** and correctness of configuration items

- Configuration management -

- Why to use configuration management?

I CANNOT UNDERSTAND
YESTERDAY IT WORKED FINE □
I HAVE JUST MODIFIED
A SINGLE LINE

WHICH VERSION
OF THE BASIC SW
ARE WE USING □

WHO HAS DELETED
THIS FILE □

WHO HAS CHANGED THIS FILE □
I HAVE LOST MY MODIFICATIONS

I DO NOT UNDERSTAND □
THIS BUG WAS ALREADY FIXED

I THOUGHT I MADE
A BACKUP OF THIS VERSION □

→ SYMPTOM OF CONFUSION

- Configuration management -

- What can be under control?
  - anything we can store as a file
- What should be under control
  - SW development environment (tools, process)
  - source files
  - produced files (map file, executable)
  - documentation
- Needed activities:
  - Version management
  - Baselining
  - Change management
  - Configuration reporting and review
  - Status accounting
  - Archiving

➢ A tool is required!

- EVERYTHING WE DELIVER TO A CUSTOMER HAS TO BE UNDER CM.
- PAPER DOCUMENTS ARE MANAGED IN THE SW PROJECT FOLDER

- Versioning control system -

VERSIONING / REVISION CONTROL SYSTEM

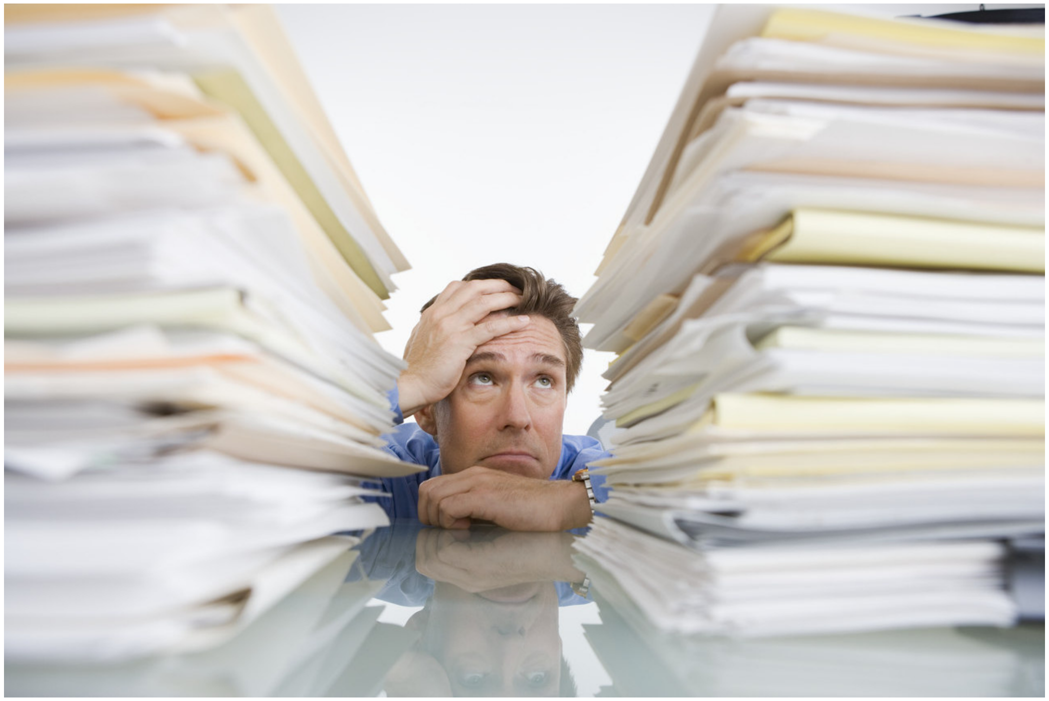
System for management of multiple revisions of the same unit of information. It is most commonly used in engineering and software development to manage ongoing development of digital documents like application source code and other projects that may be worked on by a team of people.

Changes to these documents are usually identified by incrementing an associated number or letter code, termed the "revision number", "revision level", or simply "revision" and associated historically with the person making the change.

CHARACTERISTICS

A versioning control system must provide:

- Some mechanism to **store** items under versioning control.
- The ability to **make changes** over stored items. (partial changes, add, delete, move, rename items, etc.)
- **Historical record** of performed actions over items or groups of items. (Enabling the posibility to roll-back to older versions, to view changes over time, etc.).



- Versioning control system -

CLASIFICATION

Main clasification of versioning control systems is based on the storage model of items.

- *Centralized system*: There is a centralized repository which contains all items under versioning control. There are owned and managed by one responsible user (or a group of them). Administrative tasks are simplified but reducing the flexibility. Strong tasks must be first approved by responsible.
- *Distributed system*: Adds more flexibility to the system, increases distributed decision capabilities but some tasks like synchronization can be very complex.

COMMON VOCABULARY

Baseline

An approved revision of a document or source file from which subsequent changes can be made. See the discussion of baselines, labels, and tags.

Branch

A set of files under version control may be branched or forked at a point in time so that, from that time forward, two copies of those files may be developed at different speeds or in different ways independently of the other.

- Versioning control system -

... Common vocabulary.....

Change

A change (or diff, or delta) represents a specific modification to a document under version control. The granularity of the modification considered a change varies between version control systems.

Change list

On many version control systems with atomic multi-change commits, a changelist, change set, or patch identifies the set of changes made in a single commit. This can also represent a sequential view of the source code, allowing source to be examined as of any particular changelist ID.

Checkout

A check-out creates a local working copy from the repository. Either a specific revision is specified, or the latest is obtained.

Commit / Checkin

A commit (checkin) occurs when a copy of the changes made to the working copy is written or merged into the repository.

Conflict

A conflict occurs when two changes are made by different parties to the same document, and the system is unable to reconcile the changes. A user must resolve the conflict by combining the changes, or by selecting one change in favour of the other.

- Versioning control system -

... Common vocabulary.....

Export

An export is similar to a check-out except that it creates a clean directory tree without the version control metadata used in a working copy. Often used prior to publishing the contents.

Head

The most recent commit.

Import

An import is the action of copying a local directory tree (that is not currently a working copy) into the repository for the first time.

Mainline

Similar to Trunk, but there can be a Mainline for each branch.

- Versioning control system -

... Common vocabulary...

Merge

A merge or integration brings together two sets of changes to a file or set of files into a unified revision of that file or files.

- This may happen when one user, working on those files, updates their working copy with changes made, and checked into the repository, by other users. Conversely, this same process may happen in the repository when a user tries to check-in their changes.
- It may happen after a set of files has been branched, then a problem that existed before the branching is fixed in one branch and this fix needs merging into the other.
- It may happen after files have been branched, developed independently for a while and then are required to be merged back into a single unified trunk.

Reconcile

Look for differences between the repository and the working area to updated either repository based on work area or work area based on repository.

Repository

The repository is where the current and historical file data is stored, often on a server. Sometimes also called a depot (e.g. with SVK, AccuRev and Perforce).

Resolve

The act of user intervention to address a conflict between different changes to the same document.

Reverse integration

The process of merging different team branches into the main trunk of the versioning system.

- Versioning control system -

... Common vocabulary...

Revision

Also version: A version is any change in form. In SVK, a Revision is the state at a point in time of the entire tree in the repository.

Tag / Label

A tag or label refers to an important snapshot in time, consistent across many files. These files at that point may all be tagged with a user-friendly, meaningful name or revision number. See the discussion of baselines, labels, and tags.

Trunk

The unique line of development that is not a branch (sometimes also called Baseline or Mainline)

Update

An update (or sync) merges changes that have been made in the repository (e.g. by other people) into the local working copy.

Working copy / working area

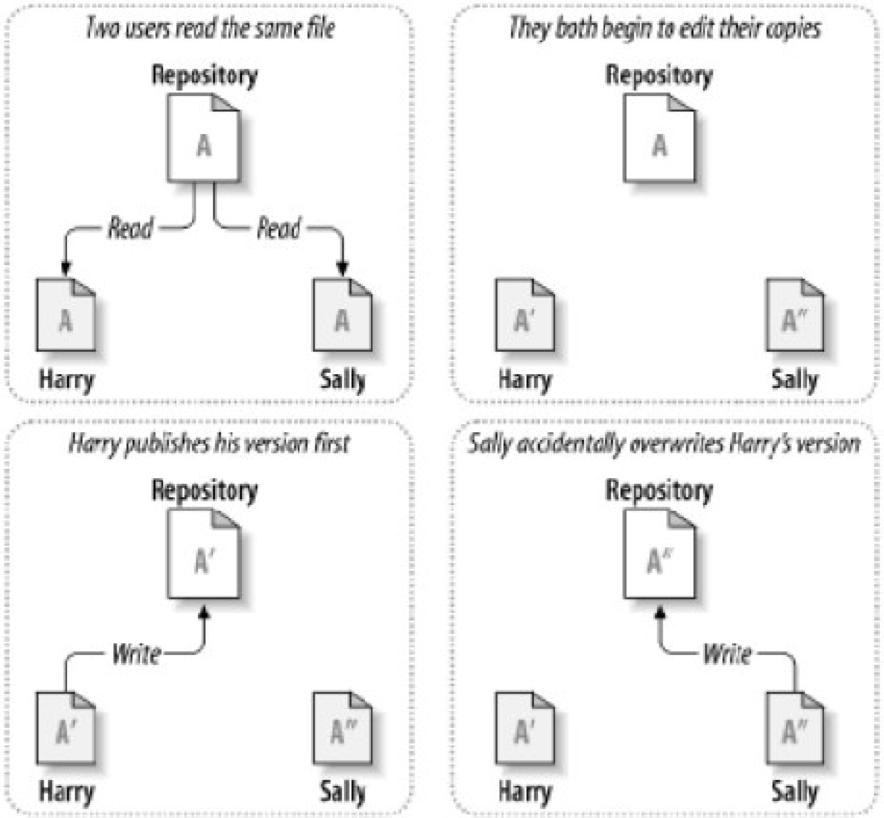
The working copy is the local copy of files from a repository, at a specific time or revision. All work done to the files in a repository is initially done on a working copy, hence the name. Conceptually, it is a sandbox.

- Versioning control system -

USE CASES

All version control systems have to solve the same fundamental problem: how will the system allow users to share information, but prevent them from accidentally stepping on each other's feet? It's all too easy for users to accidentally overwrite each other's changes in the repository.

The Problem of File-Sharing


Two users read the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Harry publishes his version first
Repository
A′
Write
A′
A″
Harry
Sally
Sally accidentally overwrites Harry's version
Repository
A″
Write
A′
A″
Harry
Sally

Consider this scenario: suppose we have two co-workers, Harry and Sally. They each decide to edit the same repository file at the same time. If Harry saves his changes to the repository first, then it's possible that (a few moments later) Sally could accidentally overwrite them with her own new version of the file.

While Harry's version of the file won't be lost forever (because the system remembers every change), any changes Harry made won't be present in Sally's newer version of the file, because she never saw Harry's changes to begin with. Harry's work is still effectively lost - or at least missing from the latest version of the file - and probably by accident. This is definitely a situation we want to avoid!

- Versioning control system -

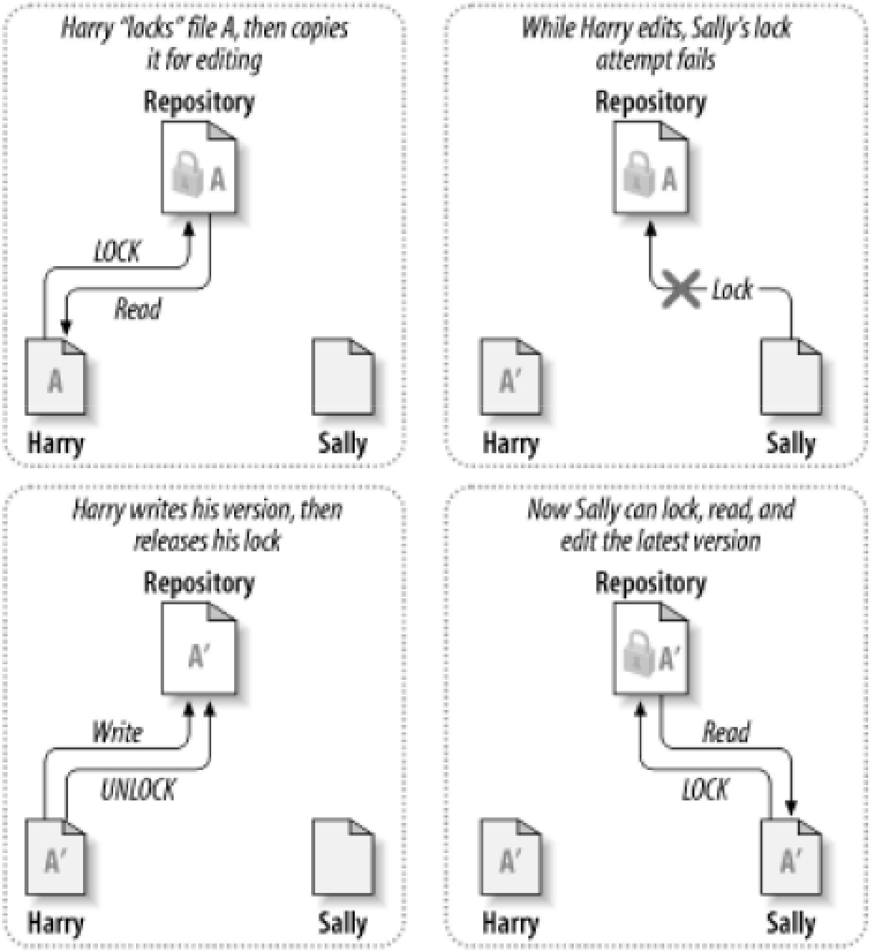
... use cases ...

The Lock-Modify-Unlock Solution

Many version control systems use a lock-modify-unlock model to address this problem, which is a very simple solution. In such a system, the repository allows only one person to change a file at a time. First Harry must lock the file before he can begin making changes to it. Locking a file is a lot like borrowing a book from the library; if Harry has locked a file, then Sally cannot make any changes to it. If she tries to lock the file, the repository will deny the request. All she can do is read the file, and wait for Harry to finish his changes and release his lock. After Harry unlocks the file, his turn is over, and now Sally can take her turn by locking and editing.

Some disadvantages:

- Locking may cause administrative problems.
- Locking may cause unnecessary serialization.
- Locking may create a false sense of security.

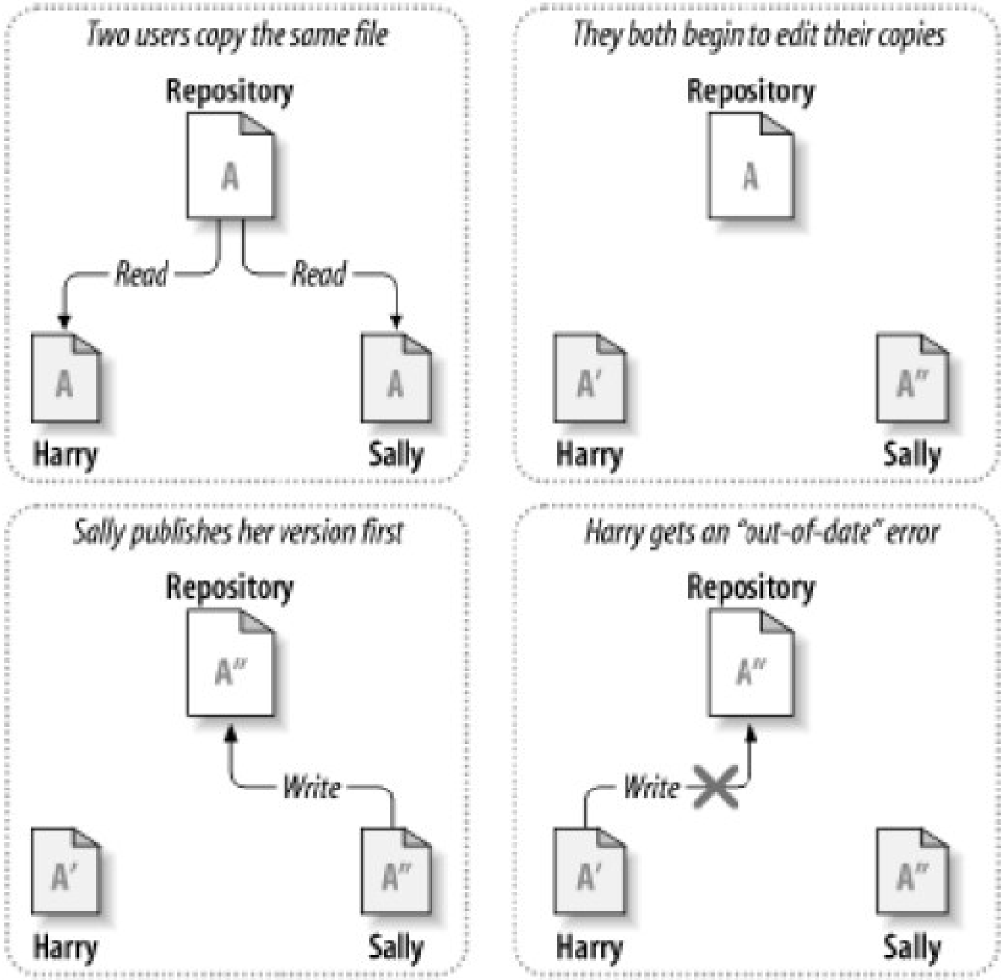

Harry "locks" file A, then copies it for editing
Repository
A
LOCK
Read
A
Harry
Sally
While Harry edits, Sally's lock attempt fails
Repository
A
Lock
A'
Harry
Sally
Harry writes his version, then releases his lock
Repository
A'
Write
UNLOCK
A'
Harry
Sally
Now Sally can lock, read, and edit the latest version
Repository
A'
Read
LOCK
A'
A'
Harry
Sally

- Versioning control system -

... use cases ...

The Copy-Modify-Merge Solution

In this model, each user's client reads the repository and creates a personal working copy of the file or project. Users then work in parallel, modifying their private copies. Finally, the private copies are merged together into a new, final version. The version control system often assists with the merging, but ultimately a human being is responsible for making it happen correctly.



Two users copy the same file
Repository
A
Read
Read
A
Harry
A
Sally
They both begin to edit their copies
Repository
A
A′
Harry
A″
Sally
Sally publishes her version first
Repository
A″
Write
A′
Harry
A″
Sally
Harry gets an "out-of-date" error
Repository
A″
Write
A′
Harry
A″
Sally

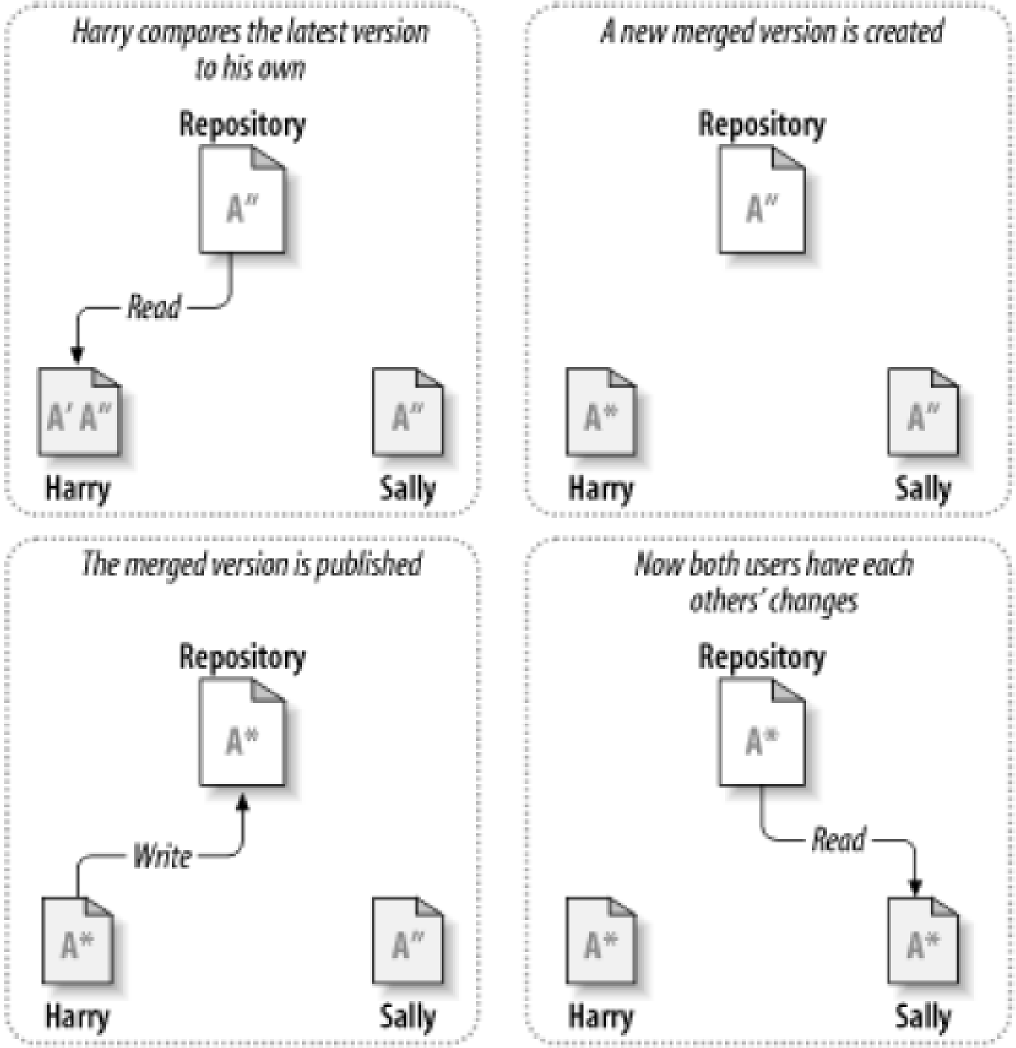
- Versioning control system -

... use cases ...

... The Copy-Modify-Merge Solution ...

The copy-modify-merge model may sound a bit chaotic, but in practice, it runs extremely smoothly. Users can work in parallel, never waiting for one another. When they work on the same files, it turns out that most of their concurrent changes don't overlap at all; conflicts are infrequent. And the amount of time it takes to resolve conflicts is far less than the time lost by a locking system.

There is one common situation where the lock-modify-unlock model comes out better, and that is where you have unmergeable files. For example if your repository contains some graphic images, and two people change the image at the same time, there is no way for those changes to be merged together.


Harry compares the latest version to his own
Repository
A″
Read
A′ A″
Harry
A″
Sally
A new merged version is created
Repository
A″
A*
Harry
A″
Sally
The merged version is published
Repository
A*
Write
A*
Harry
A″
Sally
Now both users have each others' changes
Repository
A*
Read
A*
Harry
A*
Sally

- Versioning control system - (Tools)

SUBVERSION (SVN)

Overview

Subversion (SVN) is a version control system initiated in 2000 by CollabNet Inc. It is used to maintain current and historical versions of files such as source code, web pages, and documentation. Its goal is to be a mostly-compatible successor to the widely used Concurrent Versions System (CVS).

Features

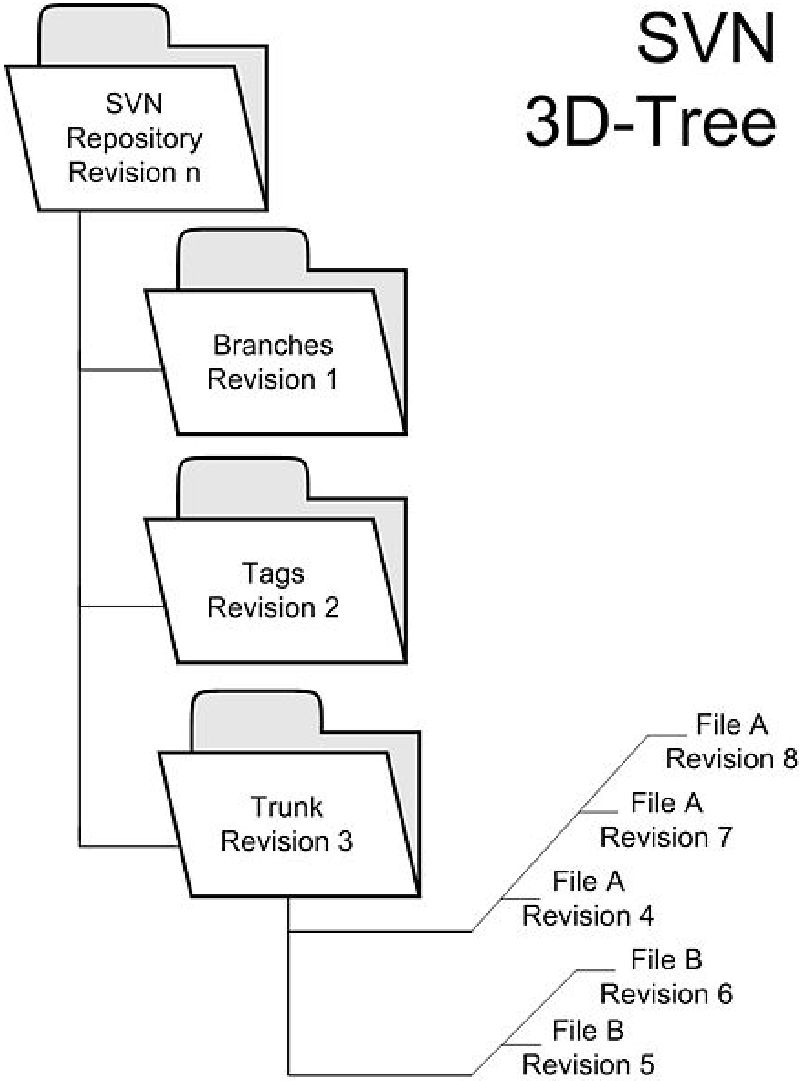
- Commits are true atomic operations. Interrupted commit operations do not cause repository inconsistency or corruption.
- Renamed/copied/moved/removed files retain full revision history.
- Entire directory trees can be moved around and/or copied very quickly, and retain full revision history.
- Native support for binary files, with space-efficient binary-diff storage.
- Apache HTTP Server as network server, WebDAV/DeltaV for protocol.
- Branching and tagging are cheap operations, independent of file size, though Subversion itself does not distinguish between a tag, a branch, and a directory
- Costs are proportional to change size, not data size.
- Open source licensed — "CollabNet/Tigris.org Apache-style license"
- File locking for unmergeable files ("reserved checkouts").

- Versioning control system - (Tools)

- ... SUBVERSION (SVN)...

Filesystem

The Subversion filesystem can be described as a three dimensional filesystem. Since most representations of a directory tree (e.g., tree view) are two dimensional, the added dimension is that of revisions. Each revision in a Subversion filesystem has its own root, which is used to access contents at that revision. Files are stored as links to the most recent change; thus a Subversion repository is quite compact. The storage space used is proportional to the number of changes made, not to the number of revisions.



SVN
3D-Tree

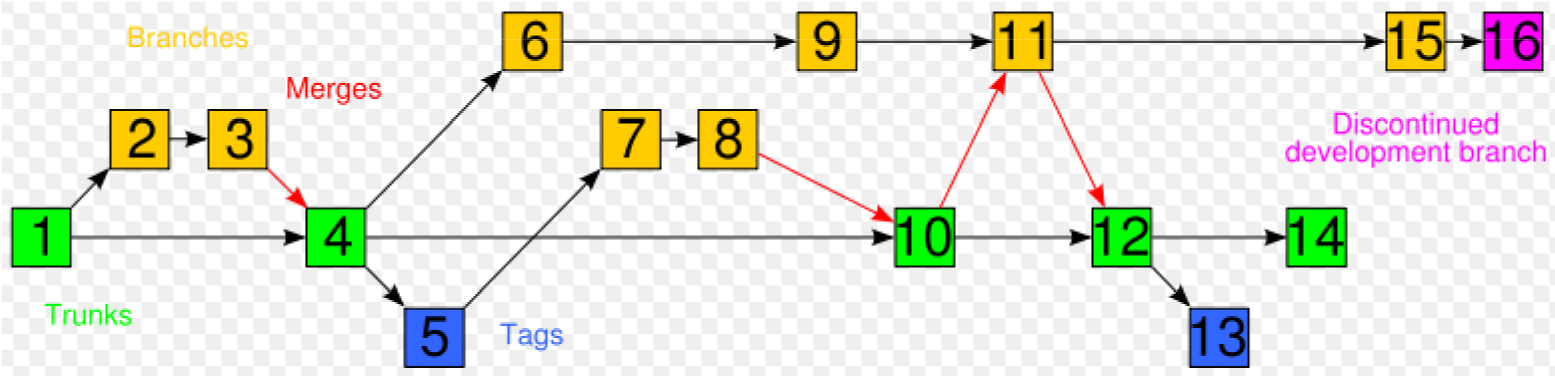
- Versioning control system - (Tools)

... SUBVERSION (SVN)...

Branching and Tagging

Branching is the ability to isolate changes onto a separate line of development. Tagging is the ability to associate additional information - such as a build environment - with a particular revision

All the files in each branch maintain the history of the file up to the point of the copy, plus any changes made since. Changes can be 'merged' back into the trunk or between branches. To Subversion, the only difference between tags and branches is that changes should not be checked into the tagged versions. Due to the differencing algorithm, creating a tag or a branch takes very little additional space in the repository.



Useful links:

http://subversion.tigris.org/ (oficial site)

http://svnbook.red-bean.com/ (SVN book)

- TortoiseSVN tool -

TortoiseSVN is a really easy to use Revision control / version control / source control software for Windows. Since it's not an integration for a specific IDE you can use it with whatever development tools you like. TortoiseSVN is free to use.

TortoiseSVN won the SourceForge.net 2007 Community Choice Award for Best Tool or Utility for Developers.

Easy to use

- All commands are available directly from the windows explorer.
- Only commands that make sense for the selected file/folder are shown. You won't see any commands that you can't use in your situation.
- See the status of your files directly in the Windows explorer descriptive dialogs, constantly improved due to user feedback allows moving files by right-dragging them in the windows explorer

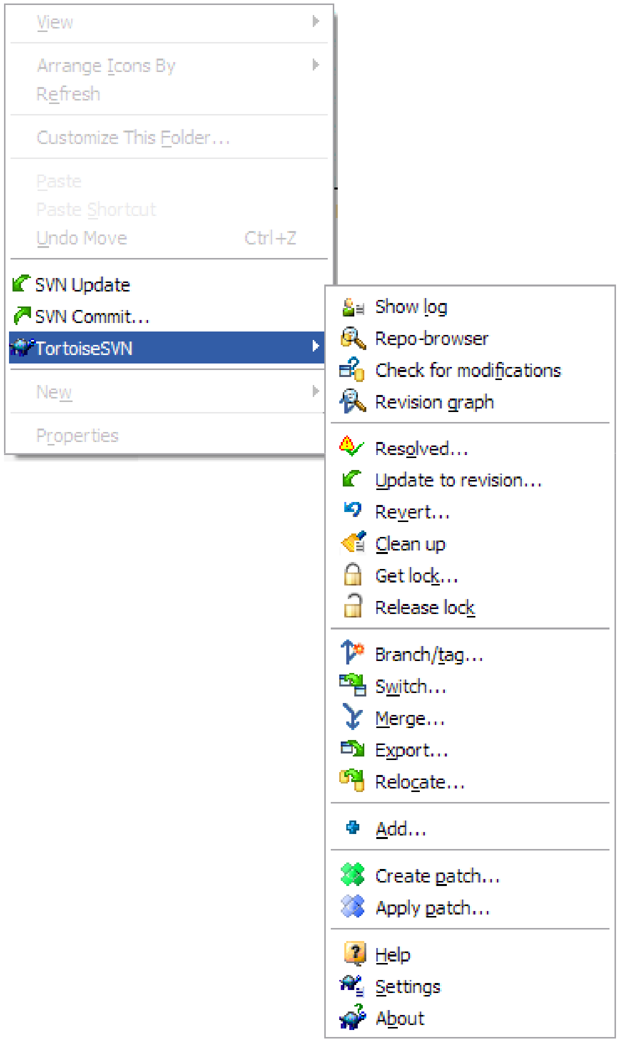
Useful links:

http://tortoisesvn.tigris.org/ (oficial site)

http://tortoisesvn.net/support (TortoiseSVN manuals/documentation)

http://en.wikipedia.org/wiki/Software_configuration_management

http://en.wikipedia.org/wiki/Revision_control



- TortoiseSVN tool -

- To practice with this tool please review file "Control de versiones (Manual de Practica) - 1.doc" after this presentation is finished.

Note: Use of TortoiseSVN tool on the project will be points to be checked for the contest.

- Naming conventions -

- The main goals of naming convention rules are to :

  - standardise the way of naming so the “language” is the same for all the developers
  - minimise the adaptation period for new readers : the software is quickly understandable
  - clarify the writing / reading of the source file
  - clarify the architecture of the software

- **Naming rules for variables**

All variables will be written with a prefix in lower case and the name of the variable. An underscore separate the prefix of the name. An upper case letter should be used to separate the words in the name:

<prefix>_<variablename> = <mxyz>_<VariableName>

The advantage of Hungarian notation comes from the limited number of types and the agreement of all the developers team members to use them.

Note: Naming convention for code will be explained later on other modu

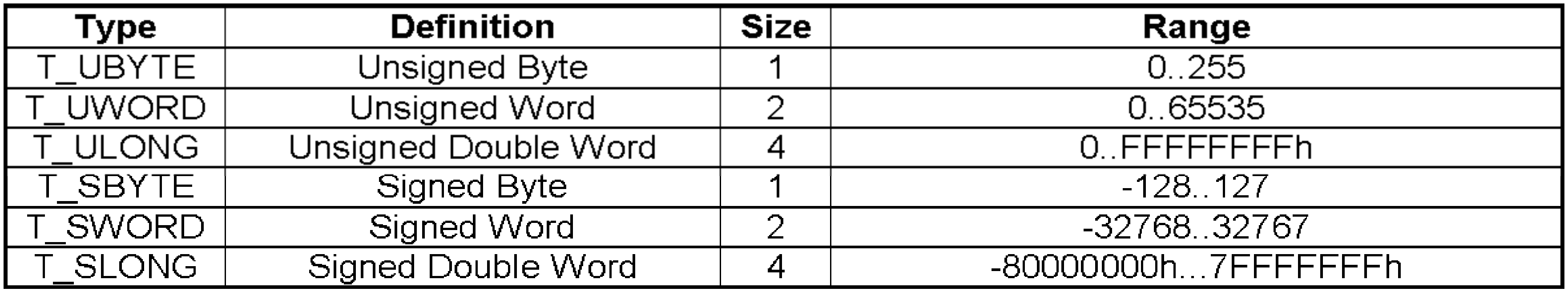
- Naming conventions -

Macros and type definitions

The macros and type definitions will be written in capital letters. If possible try to minimise exported macros and constants for visibility.

To define the type of a variable, we used « typedef » to bypass the C compiler default size definition.

The following tables list the type used :



Type	Definition	Size	Range
T_UBYTE	Unsigned Byte	1	0..255
T_UWORD	Unsigned Word	2	0..65535
T_ULONG	Unsigned Double Word	4	0..FFFFFFFFh
T_SBYTE	Signed Byte	1	-128..127
T_SWORD	Signed Word	2	-32768..32767
T_SLONG	Signed Double Word	4	-80000000h...7FFFFFFFh

- The type definition for simple types, structures, unions or enumerations will be specified in capital letters as follow :

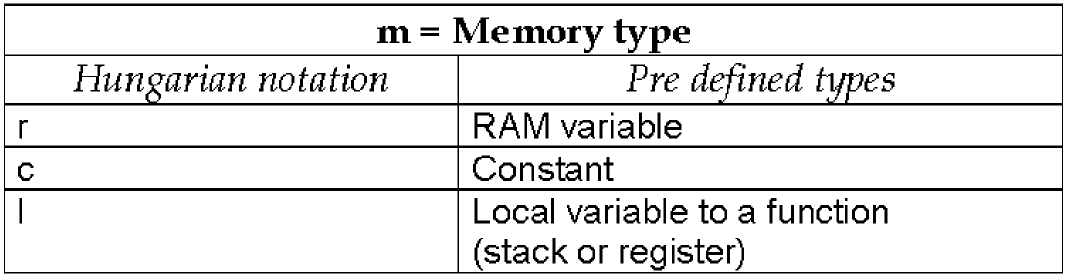
T_NAME	**(Simple Types)**
S_NAME	**(Structures)**
E_NAME	**(Enumerations)**
U_NAME	**(Unions)**

Note: Naming convention for code will be explained later on other modu

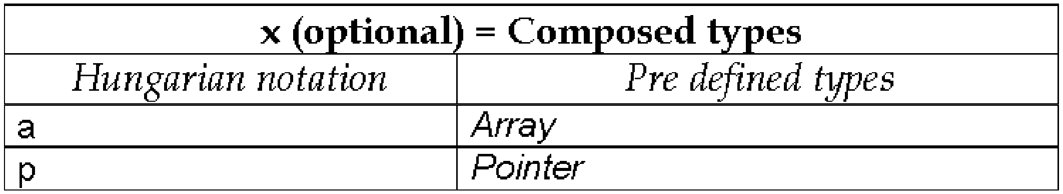
- Naming conventions -

- The following tables list the hungarian notation used :

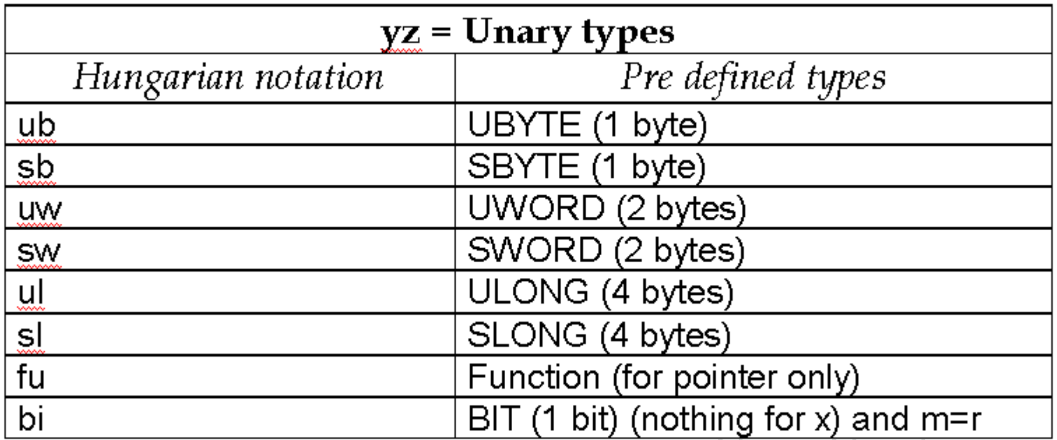
<mxyz>_<VariableName>



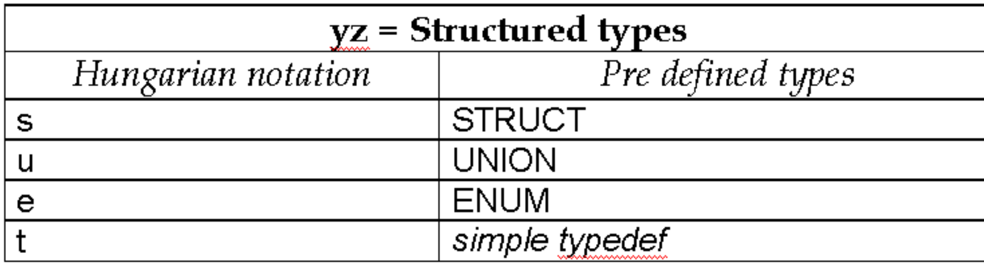
m = Memory type	
Hungarian notation	*Pre defined types*
r	RAM variable
c	Constant
l	Local variable to a function (stack or register)



x (optional) = Composed types	
Hungarian notation	*Pre defined types*
a	*Array*
p	*Pointer*



yz = Unary types	
Hungarian notation	*Pre defined types*
ub	UBYTE (1 byte)
sb	SBYTE (1 byte)
uw	UWORD (2 bytes)
sw	SWORD (2 bytes)
ul	ULONG (4 bytes)
sl	SLONG (4 bytes)
fu	Function (for pointer only)
bi	BIT (1 bit) (nothing for x) and m=r



yz = Structured types	
Hungarian notation	*Pre defined types*
s	STRUCT
u	UNION
e	ENUM
t	*simple typedef*

Note: Naming convention for code will be explained later on other mod

- Naming conventions -

Remember:

m :

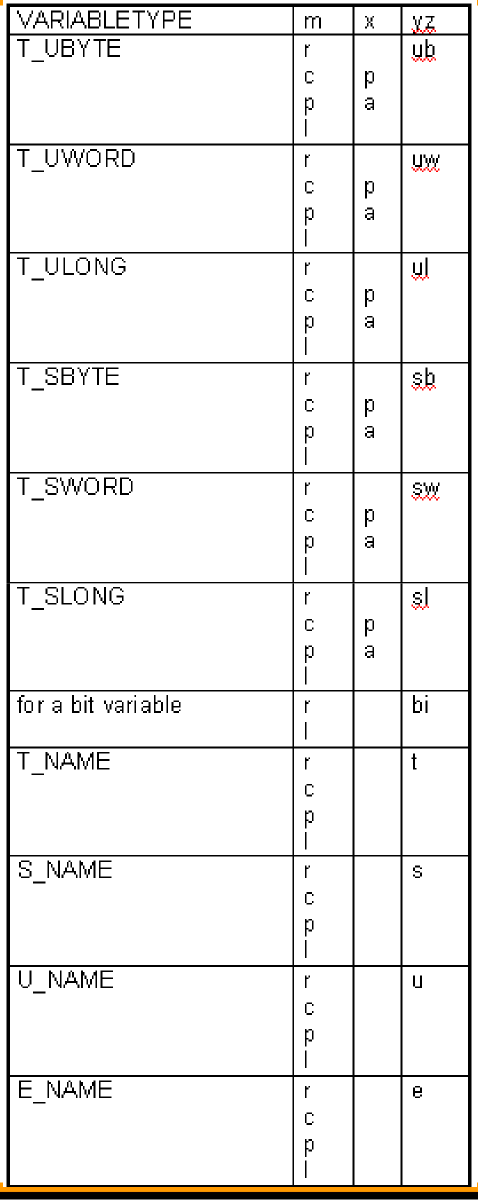
- **r = RAM variable**
- **c = Constant calibration**
- **p= Program constant**
- **l = Local variable**

x : (optional)

- **a = array**
- **p = pointer**

yz :

structured types / Unary types



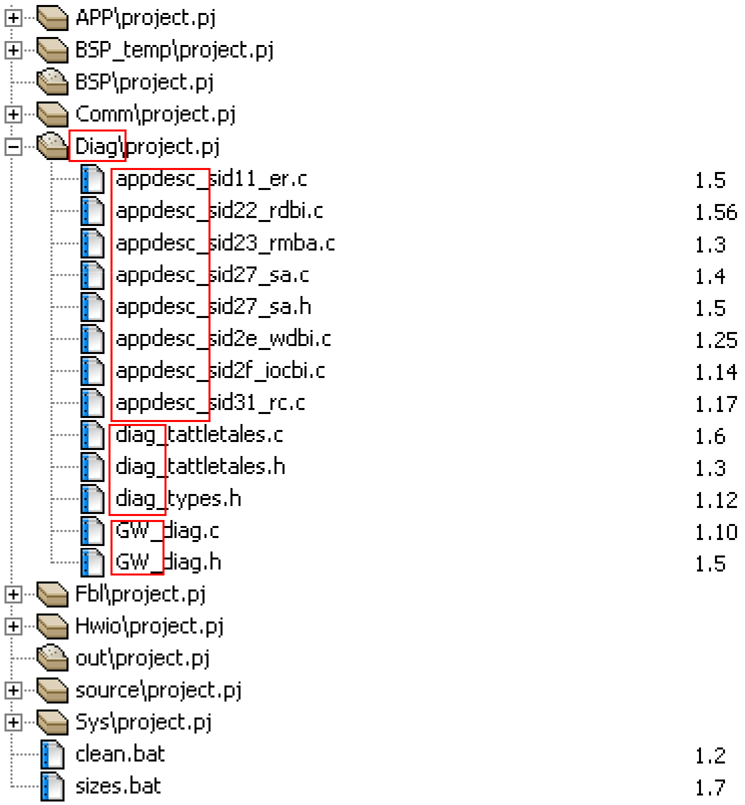
VARIABLETYPE	m	x	yz
T_UBYTE	r c p l	p a	ub
T_UWORD	r c p l	p a	uw
T_ULONG	r c p l	p a	ul
T_SBYTE	r c p l	p a	sb
T_SWORD	r c p l	p a	sw
T_SLONG	r c p l	p a	sl
for a bit variable	r l		bi
T_NAME	r c p l		t
S_NAME	r c p l		s
U_NAME	r c p l		u
E_NAME	r c p l		e

- A real world example:

```
typedef struct {
        T_UBYTE ub_DtcMsb;
        T_UBYTE ub_DtcLsb;
        union {
          struct {
                T_UBYTE bi4_Reserved    :4;
                T_UBYTE bi_ReadinessFlag :1;
                T_UBYTE bi2_StorageState :2;
                T_UBYTE bi_WarnIndRq     :1;
          } bit;
          T_UBYTE val;
        } u_Status;
        T_UBYTE bi_OccFlag   :1;
        T_UBYTE bi7_Reserved :7;
        T_UBYTE ub_OrigOdoMsb;
        T_UBYTE ub_OrigOdoLsb;
        T_UBYTE ub_MostRecentOdoMsb;
        T_UBYTE ub_MostRecentOdoLsb;
        T_ULONG ul_FreqCntr;
        T_UBYTE ub_OpCycCntr;
 } S_KWP_SID17_DTC_RECORD;
```

- Naming conventions -

- Naming conventions rules are also applicable to functions, project folder structure and modules but in this case it is a little different.
- For example:



```
APP\project.pj
BSP_temp\project.pj
BSP\project.pj
Comm\project.pj
Diag\project.pj
    appdesc_sid11_er.c        1.5
    appdesc_sid22_rdbi.c      1.56
    appdesc_sid23_rmba.c      1.3
    appdesc_sid27_sa.c        1.4
    appdesc_sid27_sa.h        1.5
    appdesc_sid2e_wdbi.c      1.25
    appdesc_sid2f_iocbi.c     1.14
    appdesc_sid31_rc.c        1.17
    diag_tattletales.c        1.6
    diag_tattletales.h        1.3
    diag_types.h              1.12
    GW_diag.c                 1.10
    GW_diag.h                 1.5
Fbl\project.pj
Hwio\project.pj
out\project.pj
source\project.pj
Sys\project.pj
clean.bat                     1.2
sizes.bat                     1.7
```

- Look how this project is grouped by modules (folders) and inside those folders the files begings with a prefix also describing what the code is about.
- This also applies for functions, if we have a function inside appdesc_sid11_er.c it should have a prefix "appdesc_" for example appdesc_init();. In this way if you use this function in other files then you know that the function is defined at any of the "appdesc_" file.

Note: Use of naming convention on the project will be points to be checked for the contest.

Thanks!