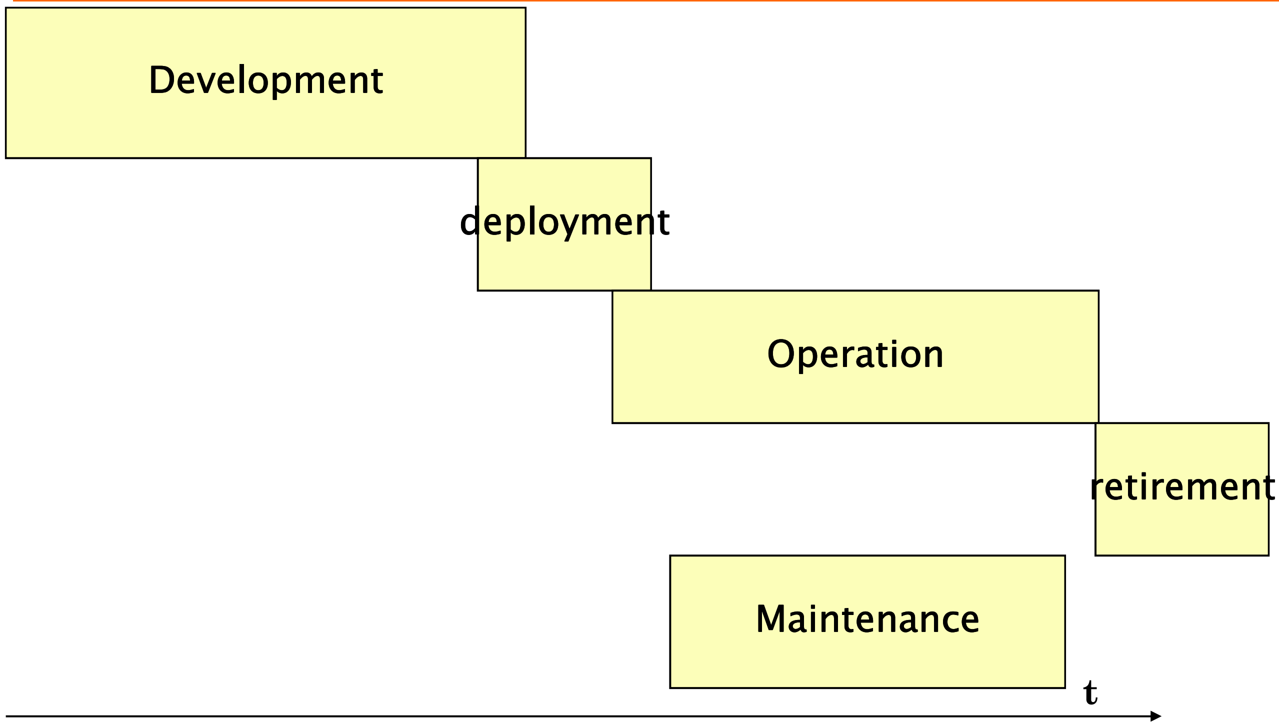
Configuration management

Outline

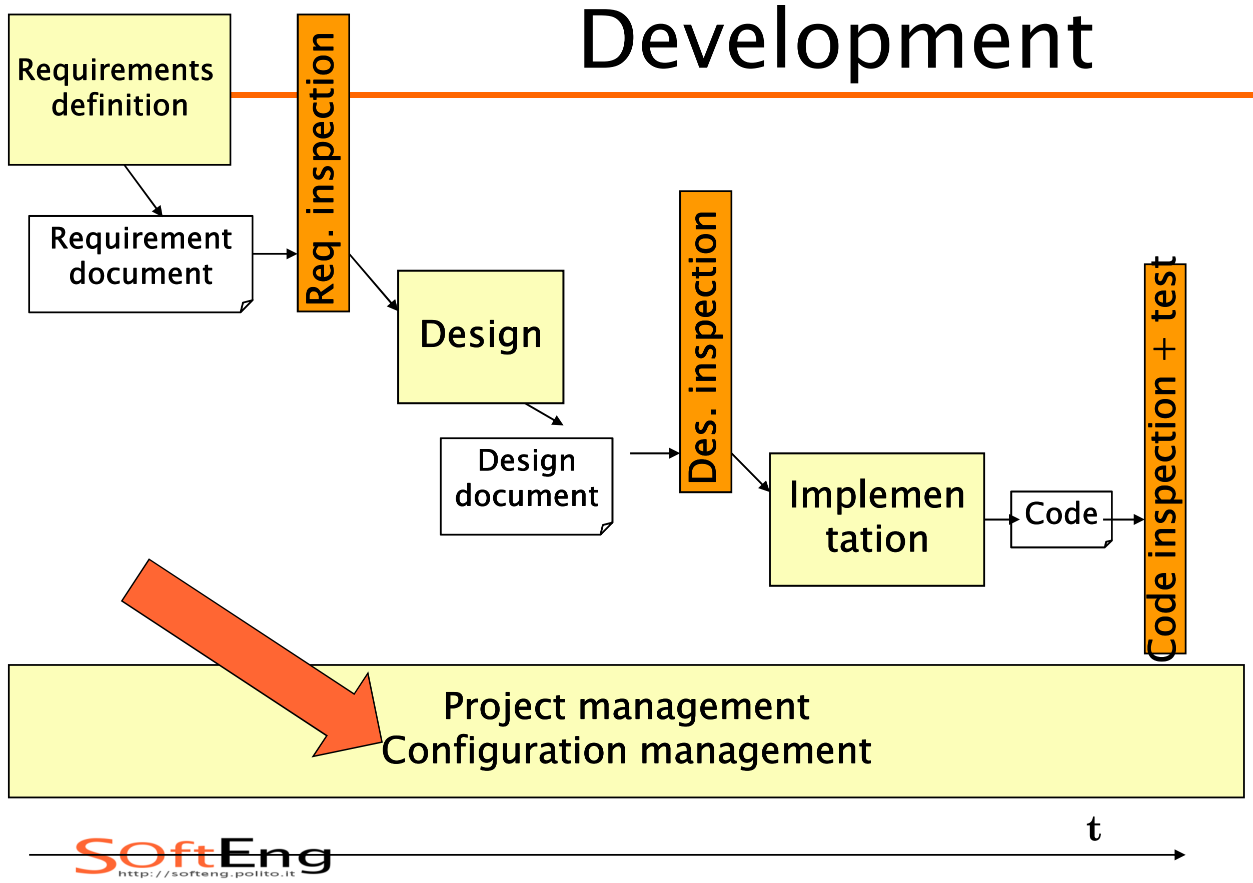
- Motivation
- Versioning
- Configuration items, configurations, baselines
- Change control
- Build
- Configuration management plan
- Configuration management tools

Motivation

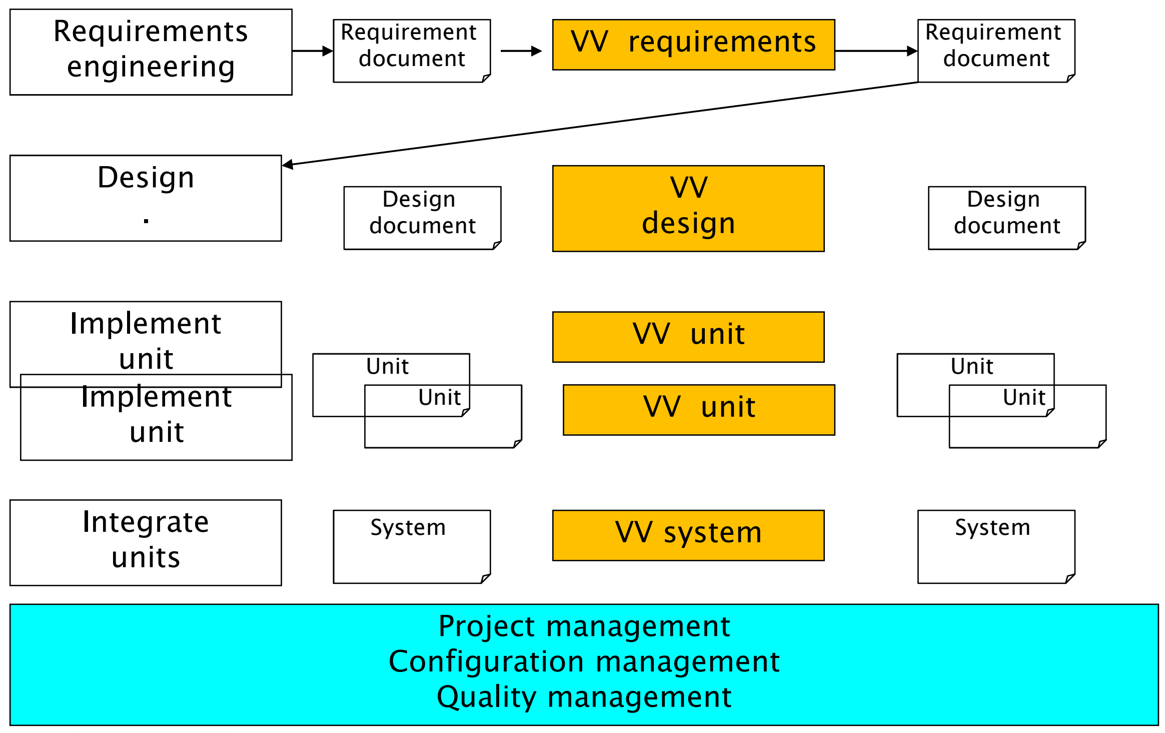
Main Phases



Development



Development



Time and space dimensions

- Space
  - System made of many parts (documents, code)
  - System (and parts) adapted for many situations
- Time
  - Parts, and system, change over time

Software – space

- Made of many parts
  - Documents
  - Programs
- With different instantiations
  - Customers, platforms
- Thousands of separate documents are generated for a large software system

Time – Change is inevitable

- A software system changes
  - Different instantiations of software for different customers
  - Same software changes over time

"No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle." [Bersoff et al., 1980]

Issues

- What is history of document?
  - versioning
- What is the correct set of documents for a specific need?
  - configuration
- Who can access and change what?
  - Change control
- How the system is obtained?
  - build

Goals of CM

- Identify and manage parts of software
- Control access and changes to parts
- Allow to rebuild previous version of software

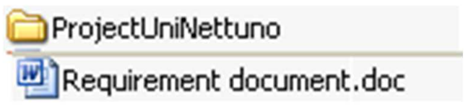
Versioning

Terms

- Configuration item (CI)
- Configuration Management aggregate
- Configuration
- Version
- Baseline

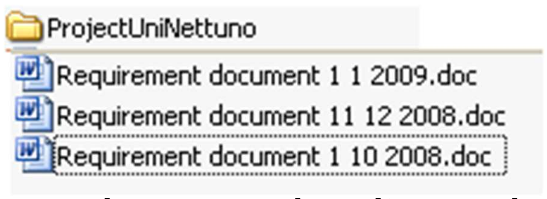
Versioning

- No history



ProjectUniNettuno
Requirement document.doc

- Different names



ProjectUniNettuno
Requirement document 1 1 2009.doc
Requirement document 11 12 2008.doc
Requirement document 1 10 2008.doc

- Tool capable of keeping track of versions
  - Same name
  - Different version name (ex 1.0 2.0 2.1 or 1,2,3,)
  - User decides when to change version (commit)
  - Always possible to recover a past version

Versioning – CI

- CI, configuration item
- Element (file) under configuration control
  - Has a name and a version number
  - All its version numbers are kept
  - User decides to change version number with specific operation (commit)
  - It is possible to retrieve any version

Configuration Item

- Unit for the CM system

  a work product or piece of software that is treated as a single entity for the purpose of configuration management.

  - May correspond to one/more document(s), one/more programs
    - Simple example of CIs
      - Requirement document
      - Design document
      - Source code module

Version

- Instance of CI
    - Ex Req document 1.0
    - Req document 1.1

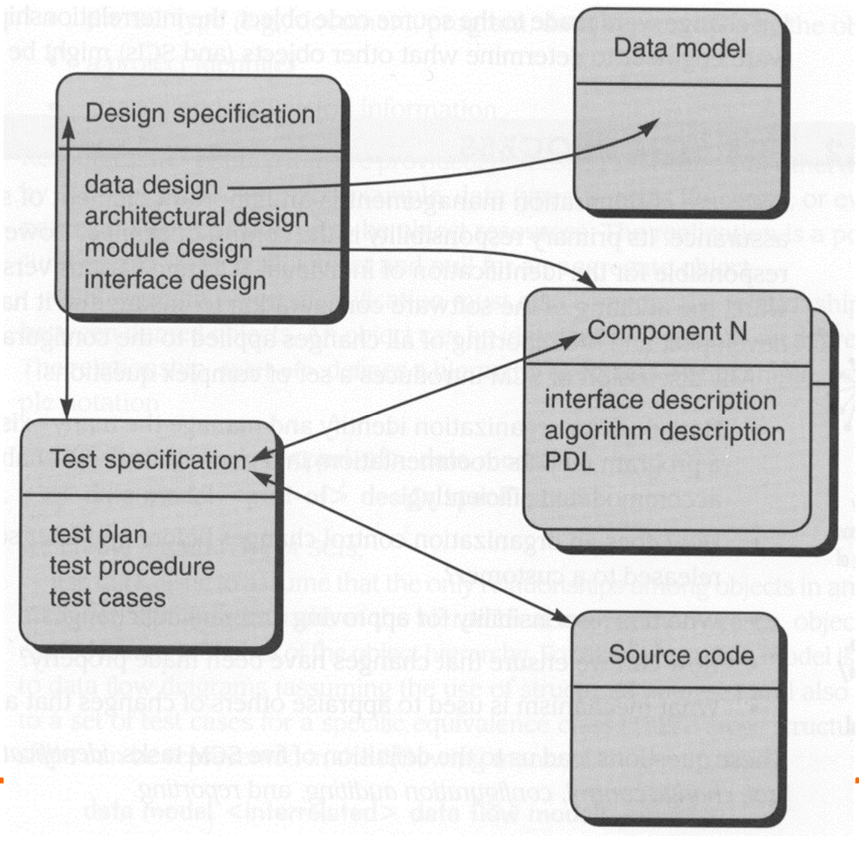
Version identification

- Procedures for version identification should define an unambiguous way of identifying component versions
- basic techniques for component identification
    - Version numbering
    - Attribute-based identification

Version numbering

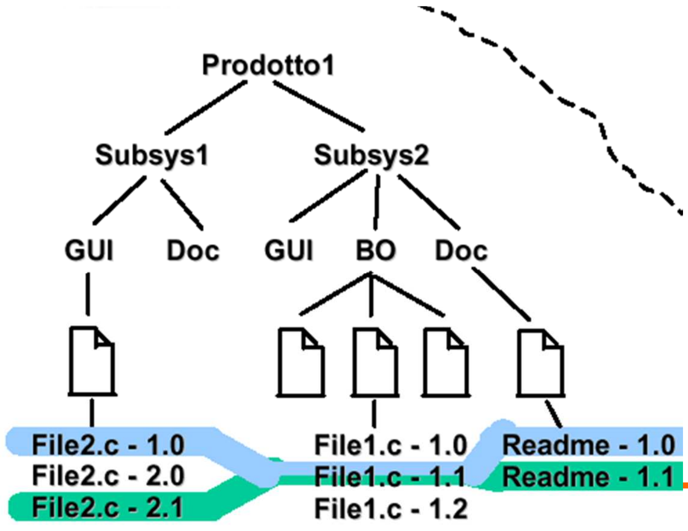
- Simple naming scheme uses a linear derivation
  e.g. V1, V1.1, V1.2, V2.1, V2.2 etc.
- Actual derivation structure is a tree or a network rather than a sequence
- Names are not meaningful.
- Hierarchical naming scheme may be better

Links between CIs



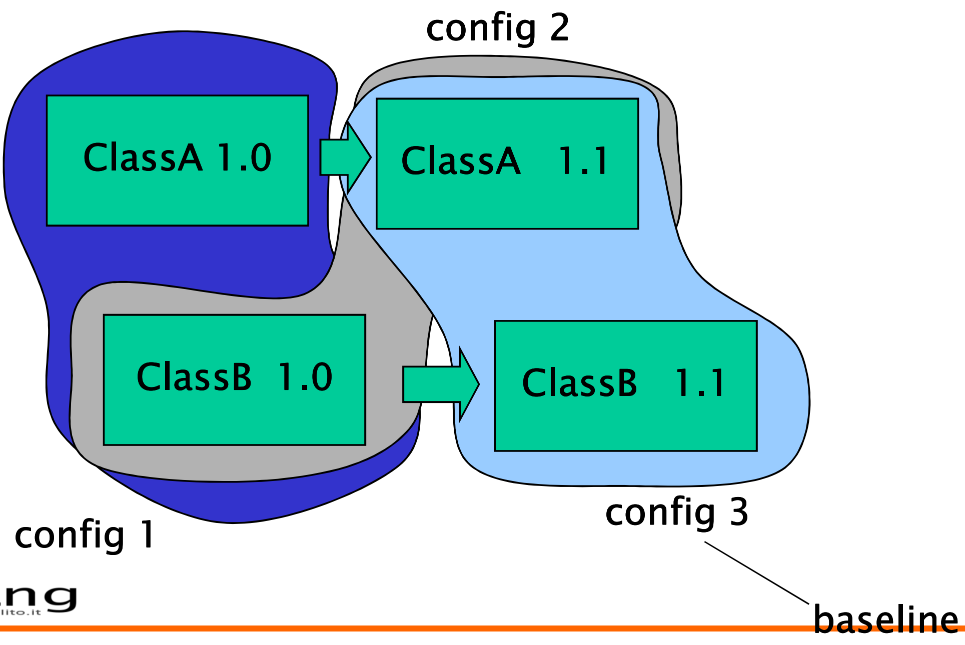
Ex.

- System 1.0 (configuration 1.0)
  - File2.c 1.0 + File1.c 1.0 + Readme 1.0
- System 1.1 (configuration 1.1)
  - File2.c 1.0 + File1.c 1.1 + Readme 1.0



Configuration

- Set of CIs, each in a specific version



- Choices of CM system
  - What parts of software system become CIs
  - (not all documents may become CIs)
- Changes to CI are subject to procedures defined by CM system
  - Typically, change must be approved and recorded
  - New version of CI must be generated

Configuration

- Snapshot of software at certain time
  - Various CIs, each in a certain version
  - Same CI may appear in different configurations
  - Also configuration has version

Baseline

- configuration in stable, frozen form
  - Not all configurations are baselines
  - Any further change / development will produce new version(s) of CI(s), will not modify baseline
- Types of baselines
  - Development – for internal use
  - Product – for delivery

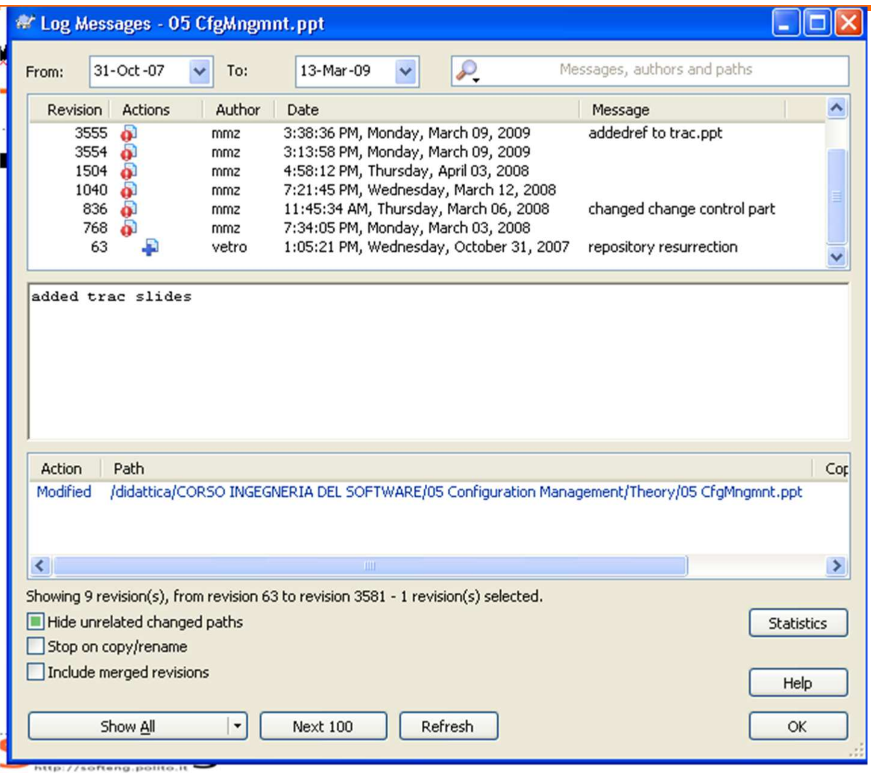
Derivation history

- Record of changes applied to a document or code component
- Should record, in outline, the change made, the rationale for the change, who made the change and when it was implemented
- May be included as a comment in code. If a standard prologue style is used for the derivation history, tools can process this automatically

Component header info

```
// PROTEUS project (ESPRIT 6087)
//
// PCL-TOOLS/EDIT/FORMS/DISPLAY/AST-INTERFACE
//
// Object: PCL-Tool-Desc
// Author: G. Dean
// Creation date: 10th November 1998
//
// © Lancaster University 1998
//
// Modification history
// Version          Modifier  Date        Change        Reason
// 1.0     J. Jones  1/12/1998   Add header   Submitted to CM
// 1.1     G. Dean   9/4/1999 New field  Change req. R07/99
```

Derivation history – svn



Revision	Actions	Author	Date	Message
3555		mmz	3:38:36 PM, Monday, March 09, 2009	addedref to trac.ppt
3554		mmz	3:13:58 PM, Monday, March 09, 2009	
1504		mmz	4:58:12 PM, Thursday, April 03, 2008	
1040		mmz	7:21:45 PM, Wednesday, March 12, 2008	
836		mmz	11:45:34 AM, Thursday, March 06, 2008	changed change control part
768		mmz	7:34:05 PM, Monday, March 03, 2008	
63		vetro	1:05:21 PM, Wednesday, October 31, 2007	repository resurrection

```
added trac slides
```

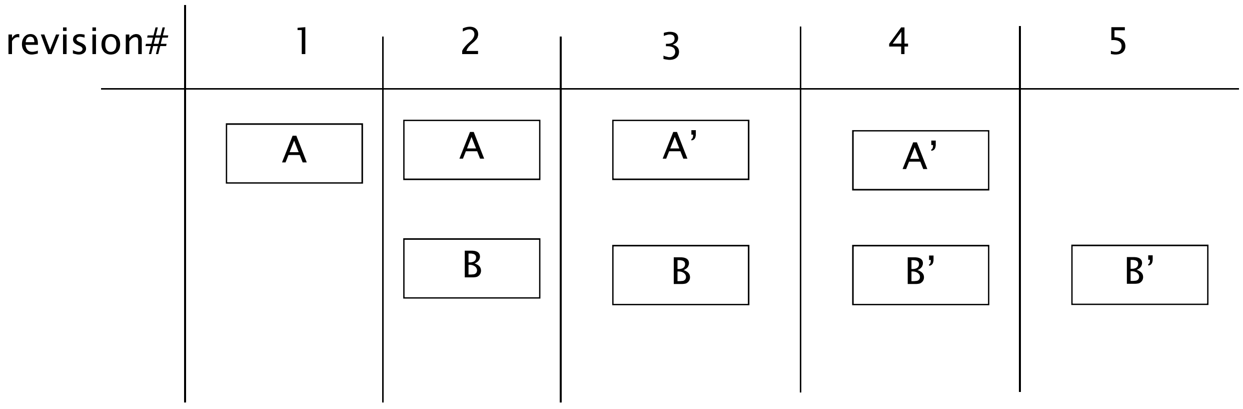
Action	Path
Modified	/didattica/CORSO INGEGNERIA DEL SOFTWARE/05 Configuration Management/Theory/05 CfgMngmnt.ppt

Showing 9 revision(s), from revision 63 to revision 3581 - 1 revision(s) selected.

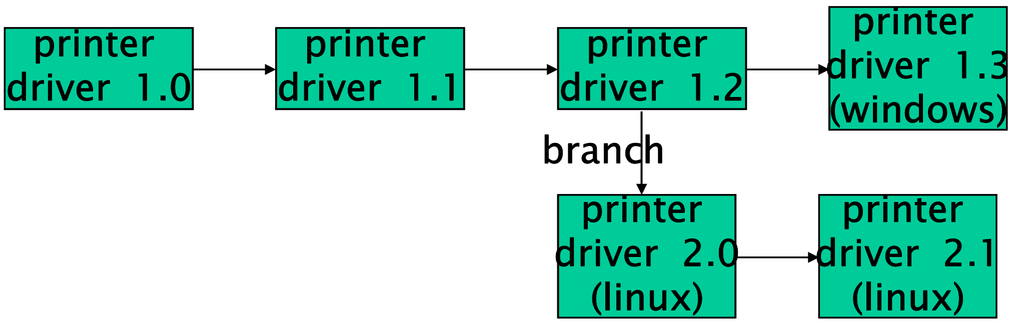
- [✓] Hide unrelated changed paths
- [ ] Stop on copy/rename
- [ ] Include merged revisions

Svn – version identification

- In subversion a version is called → revision
- Each configuration has a new number
- Each element changes revision, even if has not been changed

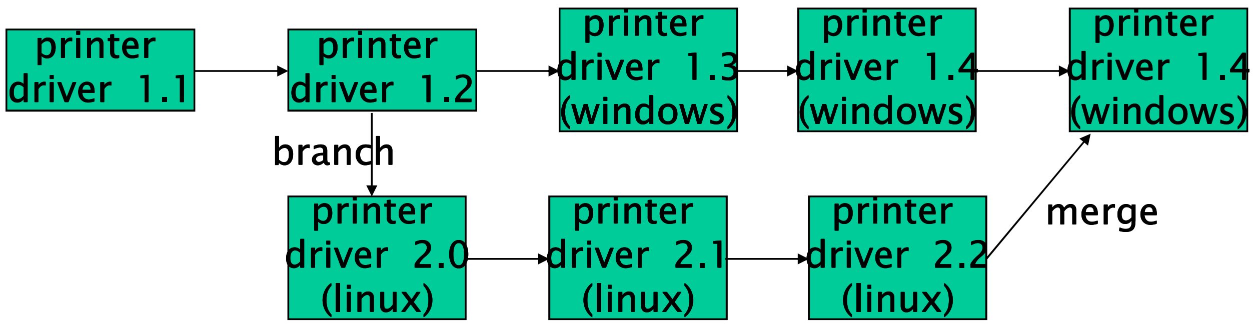


Branches



printer driver 1.0
printer driver 1.1
printer driver 1.2
printer driver 1.3 (windows)
branch
printer driver 2.0 (linux)
printer driver 2.1 (linux)

Merge



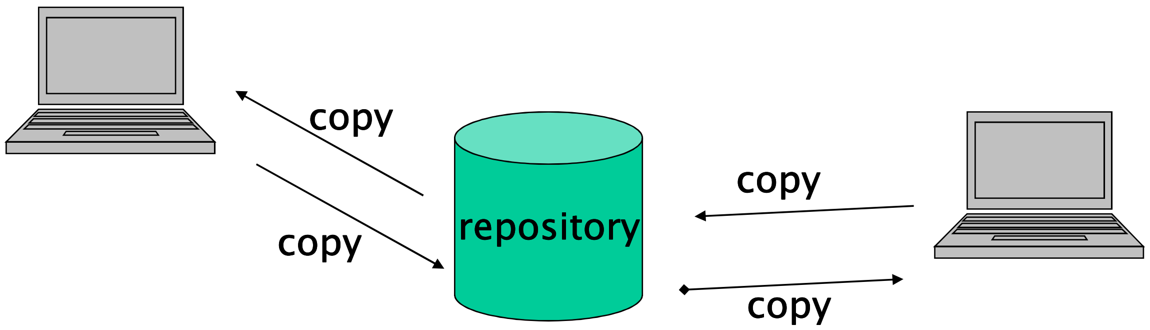
printer driver 1.1
printer driver 1.2
printer driver 1.3 (windows)
printer driver 1.4 (windows)
printer driver 1.4 (windows)
branch
printer driver 2.0 (linux)
printer driver 2.1 (linux)
printer driver 2.2 (linux)
merge

Change control

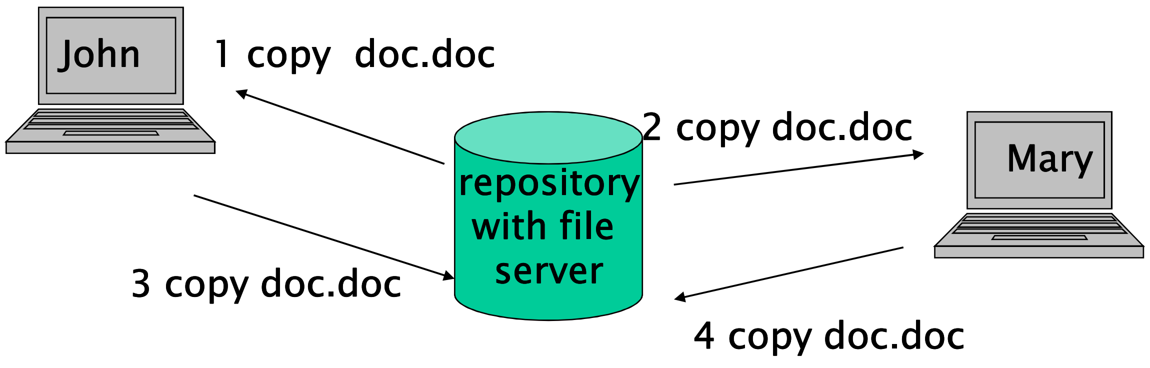
Typical situation

- Team develops software
- Many people need to access parts of software
  - Common repository (shared folder), all can read/write documents/ programs

Change control – repository



Repository – file server

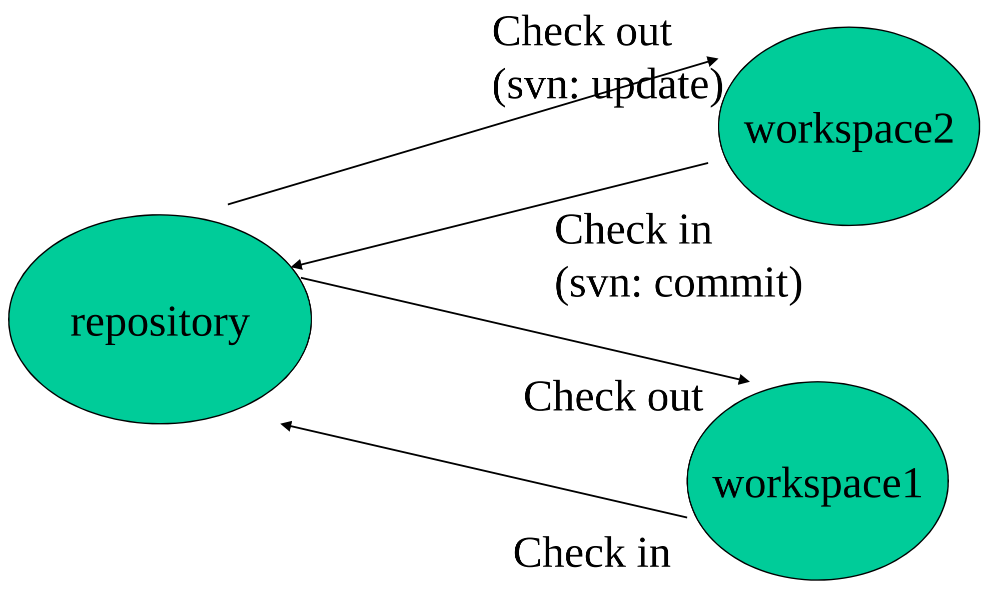


Changes by John are lost

Change control

- Changes must be disciplined
  - Who controls
  - What is controlled
  - How control is implemented
- Approaches
  - Check in – check out model, Workspaces
  - CCB
    - On top of check in check out

Workspace and check in/out



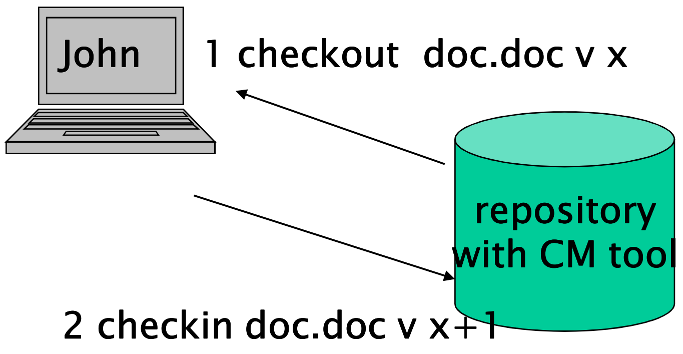
Check-in check-out

- Check-out
  - Extraction of CI from repository
    - with goal of changing it or not
    - After checkout next users are notified
- Check-in
  - Insertion of CI under control

Workspace

- 'Private' space where developer has full control

Repository – check in checkout



Checkin checkout vs. file system

Check in /out	File system
▪ CIs are in repository	▪ Files are in shared directory
▪ To rd/wr CI user needs to do check out	▪ Any user can get copy of file, or work on original
▪ After checkout next user knows that CI is used by someone else	▪ Users can work on copies of file without knowing that others are doing the same

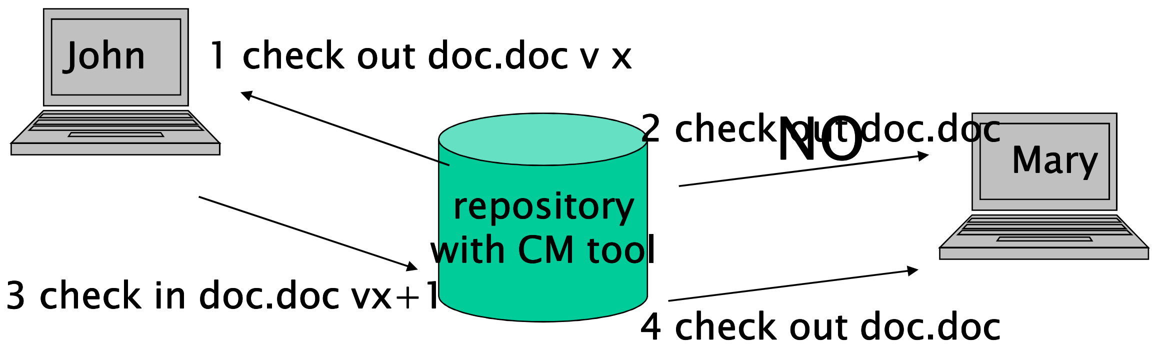
Check in/out – choices

- Who can do check in/out
- Checked-out CI is locked or not
  - If locked, one writer, many readers
    - One only can modify
- Checked-in CI increments version or not
  - If not, old version is lost

Check in / check out – scenarios

- Lock modify unlock (or serialization)
  - One can change at a time
- Copy modify merge
  - Many change in parallel, then merge

Lock modify unlock



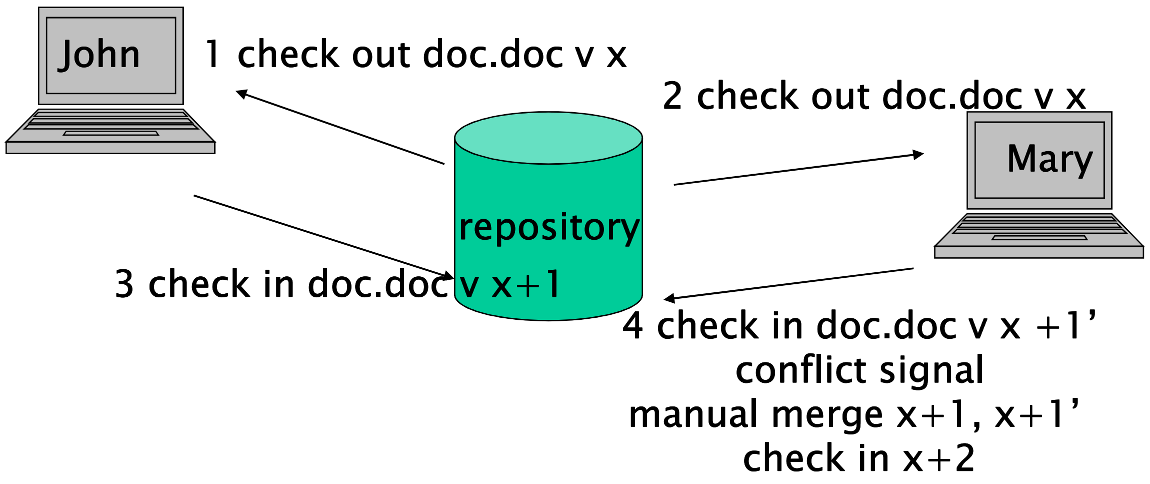
First check out locks the file
No other checkouts are allowed until check in

Problems – lock

- Locker forgets to unlock
- No parallel work, delays

SoftEng http://softeng.polito.it

Copy modify merge



The second check in signals conflict
User has to do a manual merge of x+1 and x+1' in x+2

SoftEng http://softeng.polito.it

- Pro
  - More flexible
- Cons
  - Requires care to resolve the conflict

CM Tools

- CVS
- Subversion
- Clearcase
- Bitkeeper
- ..

CCB

- Configuration Control Board
  - Authorizes changes to a baseline
    - Corrective maintenance
  - Defines what will be in next baseline
    - Perfective maintenance

Configuration control board

- Changes should be reviewed by an external group who decide whether or not they are cost-effective from a strategic and organizational viewpoint rather than a technical viewpoint
- Should be independent of project responsible for system. The group is sometimes called a change control board
- May include representatives from client and contractor staff

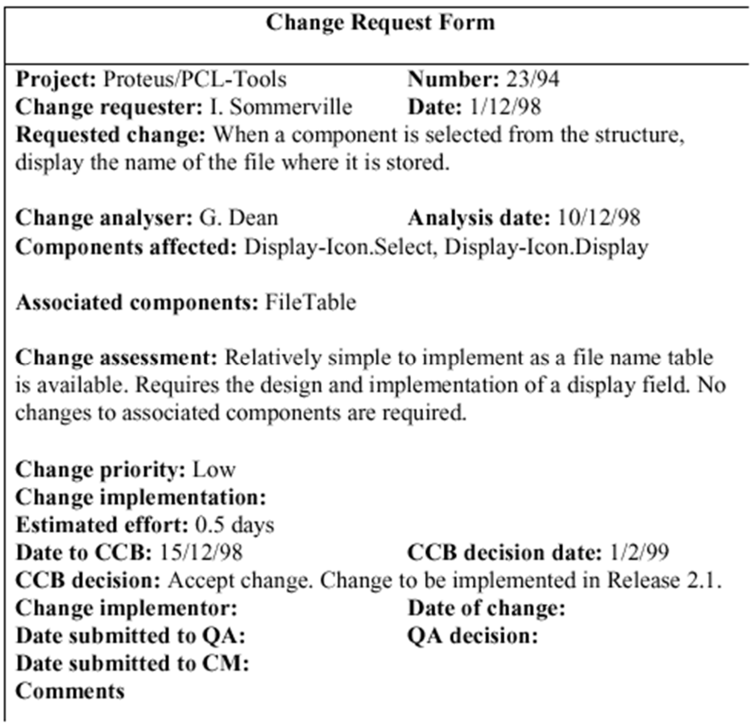
Change procedure

```
Request change by completing a change request form
Analyze change request
if change is valid then
   Assess how change might be implemented
   Assess change cost
   Submit request to change control board
   if change is accepted then
      repeat
      make changes to software
      submit changed software for quality approval
      until software quality is adequate
   create new system version
 else
   reject change request
else
   reject change request
```

Change request form

- Definition of change request form is part of the CM planning process
- Records change required, suggestor of change, reason why change was suggested and urgency of change (from requestor of the change)
- Records change evaluation, impact analysis, change cost and recommendations (System maintenance staff)

Change request form



Change Request Form

Project: Proteus/PCL-Tools **Number:** 23/94
Change requester: I. Sommerville **Date:** 1/12/98
Requested change: When a component is selected from the structure, display the name of the file where it is stored.

Change analyser: G. Dean **Analysis date:** 10/12/98
Components affected: Display-Icon.Select, Display-Icon.Display

Associated components: FileTable

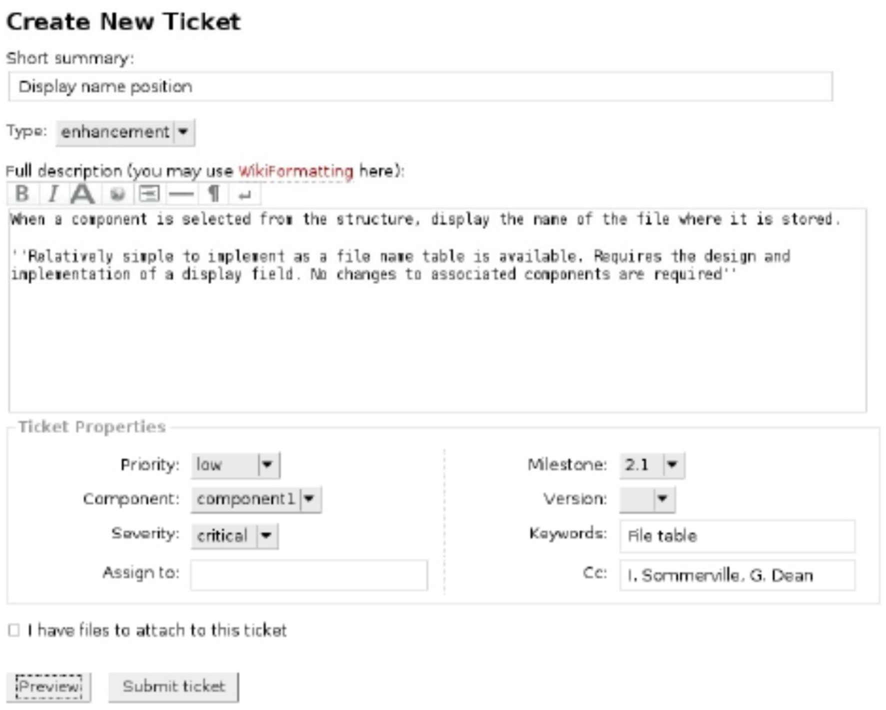
Change assessment: Relatively simple to implement as a file name table is available. Requires the design and implementation of a display field. No changes to associated components are required.

Change priority: Low
Change implementation:
Estimated effort: 0.5 days
Date to CCB: 15/12/98 **CCB decision date:** 1/2/99
CCB decision: Accept change. Change to be implemented in Release 2.1.
Change implementor: **Date of change:**
Date submitted to QA: **QA decision:**
Date submitted to CM:
Comments

Tools to support change process

- Trac, Jira, Bugzilla, ..
- See Trac demo
  - For trac: change → ticket
  - Usr demo pwd demo
- http://www.hosted-projects.com/trac/TracDemo/Demo

Trac – create ticket


Create New Ticket

Short summary: Display name position

Type: enhancement

Full description (you may use WikiFormatting here):

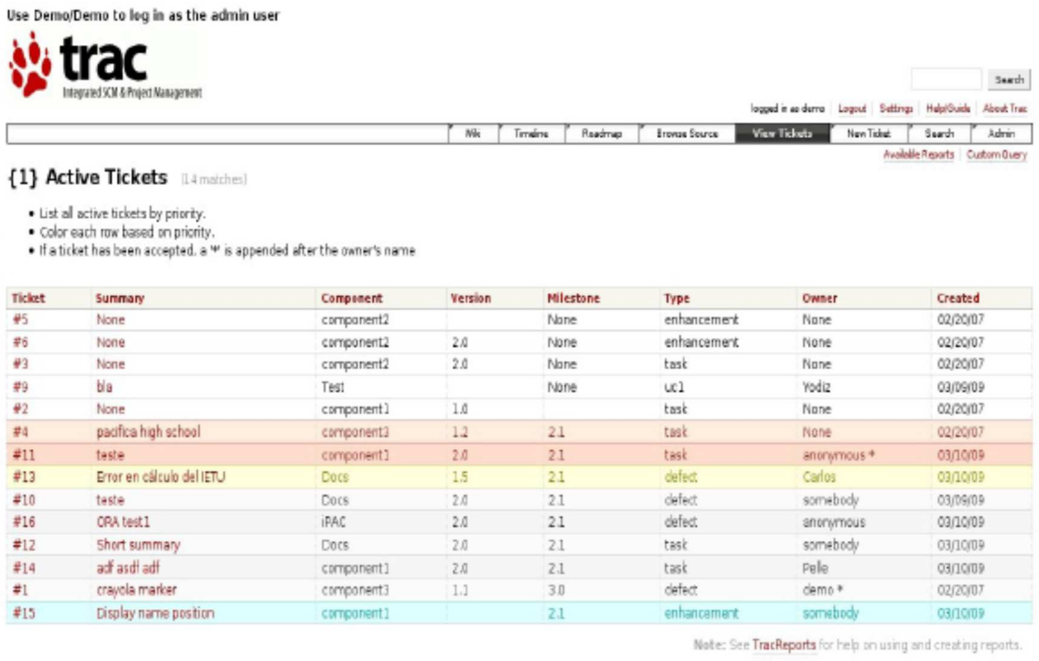
```
When a component is selected from the structure, display the name of the file where it is stored.

''Relatively simple to implement as a file name table is available. Requires the design and
implementation of a display field. No changes to associated components are required''
```

Ticket Properties

Priority:	low	Milestone:	2.1
Component:	component1	Version:	
Severity:	critical	Keywords:	File table
Assign to:		Cc:	I. Sommerville, G. Dean

☐ I have files to attach to this ticket

Trac – see all (active) tickets


Use Demo/Demo to log in as the admin user

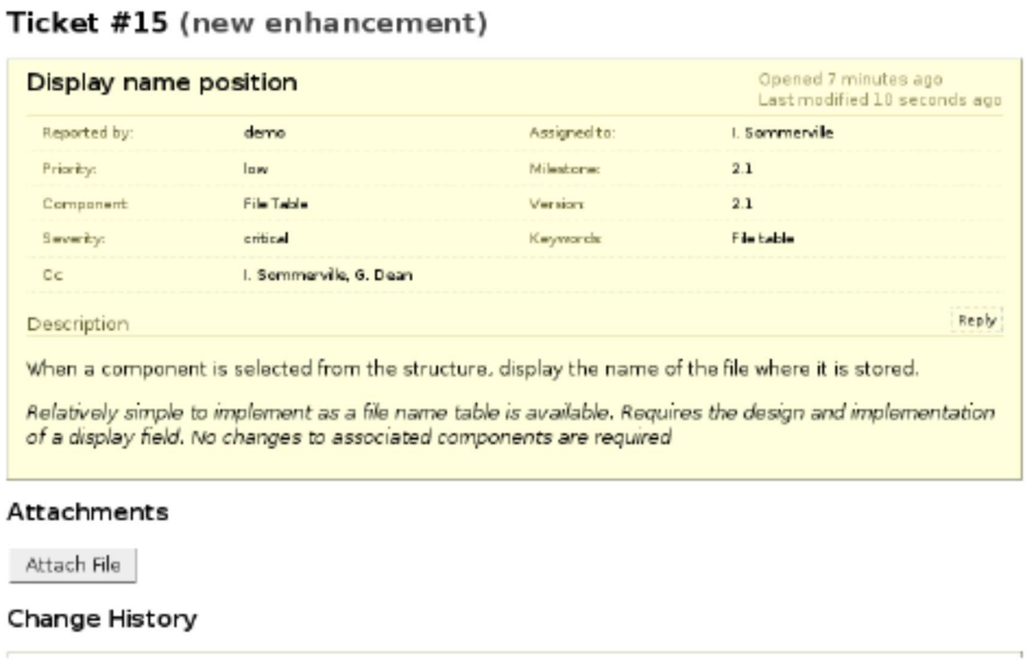
{1} Active Tickets (14 matches)

- List all active tickets by priority.
- Color each row based on priority.
- If a ticket has been accepted, a '*' is appended after the owner's name

Ticket	Summary	Component	Version	Milestone	Type	Owner	Created
#5	None	component2		None	enhancement	None	02/20/07
#6	None	component2	2.0	None	enhancement	None	02/20/07
#3	None	component2	2.0	None	task	None	02/20/07
#9	bla	Test		None	uc1	Yodiz	03/09/09
#2	None	component1	1.0		task	None	02/20/07
#4	pacifica high school	component3	1.2	2.1	task	None	02/20/07
#11	teste	component1	2.0	2.1	task	anonymous *	03/10/09
#13	Error en cálculo del IETU	Docs	1.5	2.1	defect	Carlos	03/10/09
#10	teste	Docs	2.0	2.1	defect	somebody	03/09/09
#16	ORA test1	iPAC	2.0	2.1	defect	anonymous	03/10/09
#12	Short summary	Docs	2.0	2.1	task	somebody	03/10/09
#14	adf asdf adf	component1	2.0	2.1	task	Pelle	03/10/09
#1	crayola marker	component3	1.1	3.0	defect	demo *	02/20/07
#15	Display name position	component1		2.1	enhancement	somebody	03/10/09

Note: See TracReports for help on using and creating reports.

Trac – open ticket



Ticket #15 (new enhancement)

Display name position

Opened 7 minutes ago
Last modified 10 seconds ago

Reported by:	demo	Assigned to:	I. Sommerville
Priority:	low	Milestone:	2.1
Component:	File Table	Version:	2.1
Severity:	critical	Keywords:	File table
Cc	I. Sommerville, G. Dean		

Description

Reply

When a component is selected from the structure, display the name of the file where it is stored.

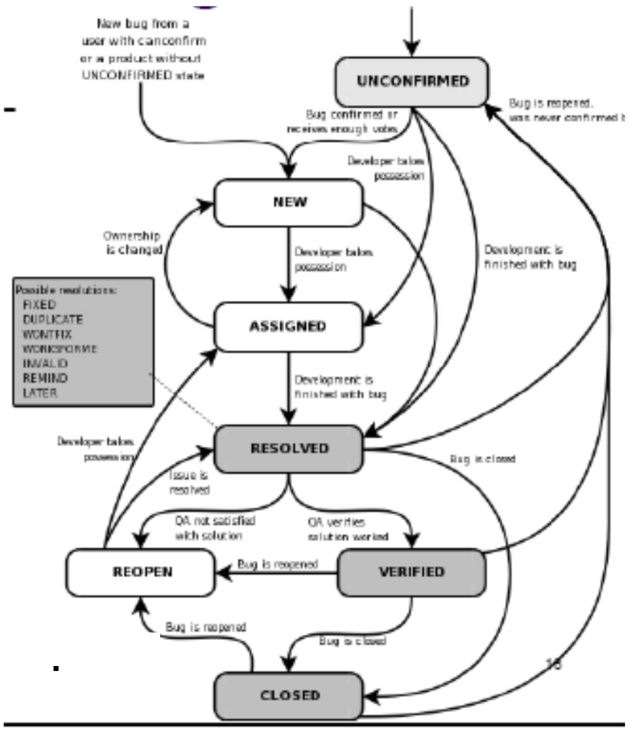
Relatively simple to implement as a file name table is available. Requires the design and implementation of a display field. No changes to associated components are required

Attachments

Attach File

Change History

Lifecycle for change (bug)


New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug is reopened, was never confirmed
Bug confirmed or receives enough votes
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions:
FIXED
DUPLICATE
WONTFIX
WORKSFORME
INVALID
REMIND
LATER
ASSIGNED
Development is finished with bug
Developer takes possession
RESOLVED
Bug is closed
Issue is resolved
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

CM Planning

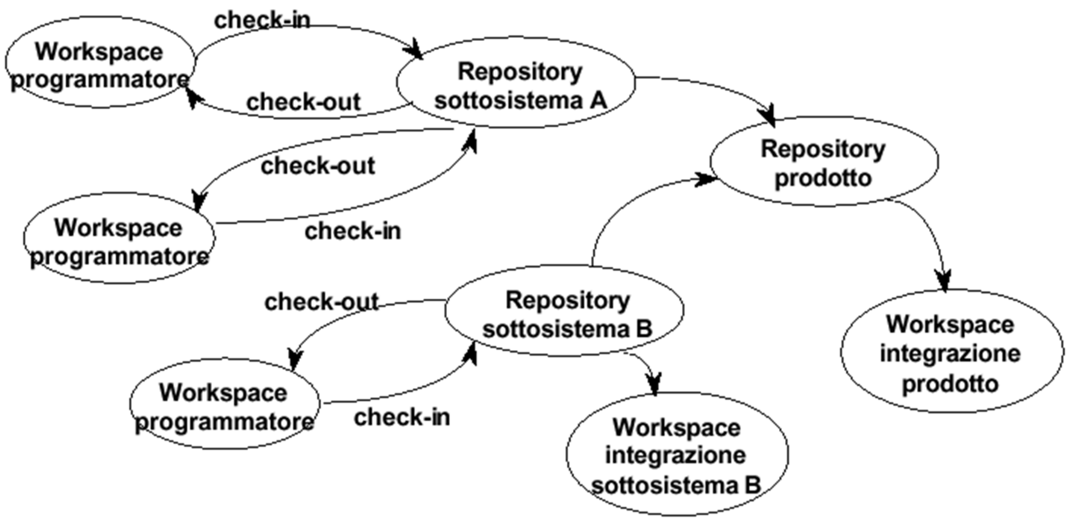
CM plan

- Contains key CM related choices and policies in a project
  - Using or not a CM tool, what tool
  - What should and should not be a CI
  - Change control policy
  - Who is CM manager

Example

- The product
  - Several subsystems, each subsystem an executable and several source files (modules)
  - Hierarchy
- The team
  - One person responsible per module
  - One person responsible per subsystem
- The repository
  - One repository per subsystem
  - Check in/out
  - Workspace per person

Example

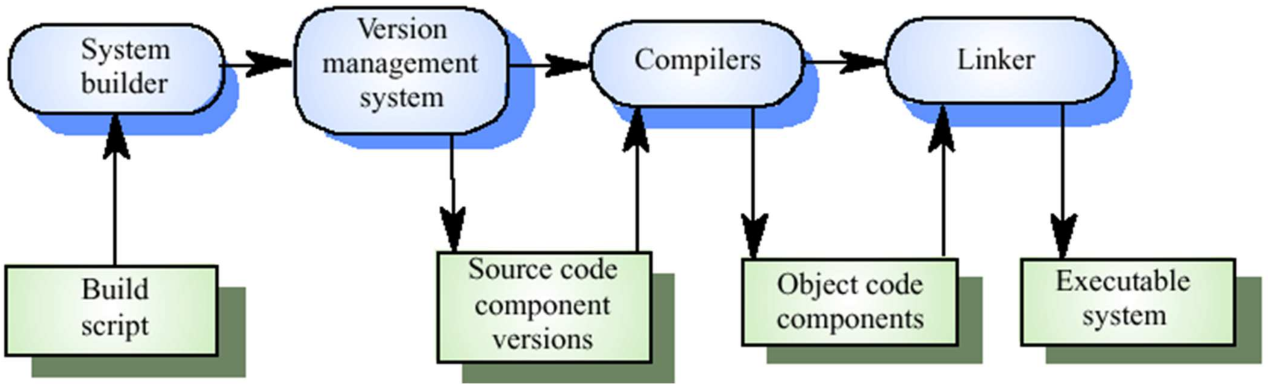


Build

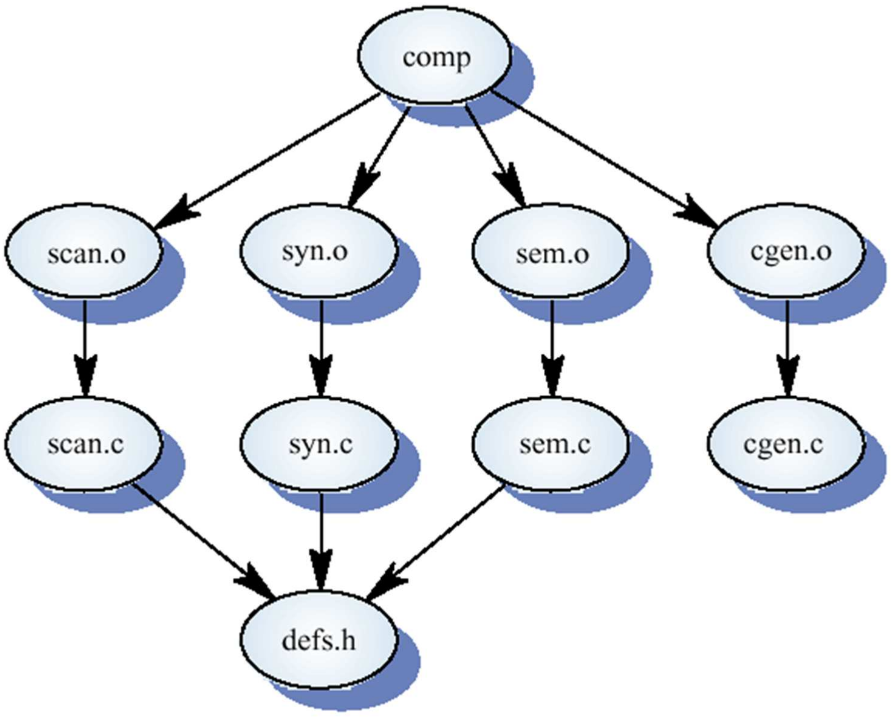
System building

- The process of compiling and linking software components into an executable system
- Different systems are built from different
combinations of components
- Invariably supported by automated tools that are driven by ‘build scripts’

System building



Component dependencies



System building problems

- Do the build instructions include all required components?
  - When there are many hundreds of components making up a system, it is easy to miss one out. This should normally be detected by the linker

- Is the appropriate component version specified?
  - A more significant problem. A system built with the wrong version may work initially but fail after delivery
- Are all data files available?
  - The build should not rely on 'standard' data files. Standards vary from place to place

System building problems

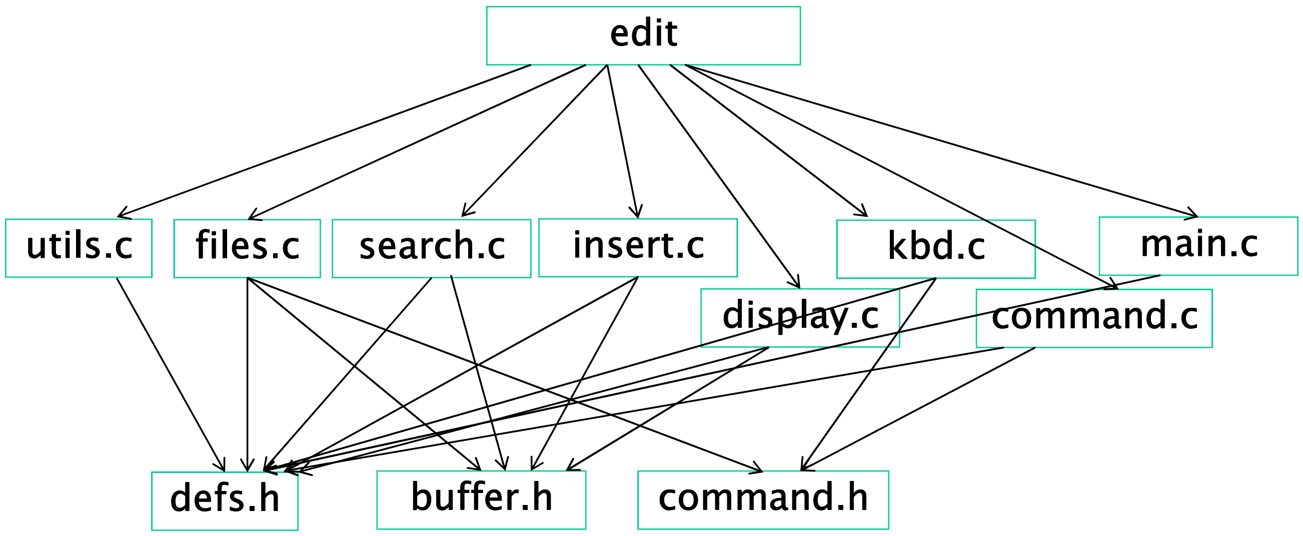
- Are data file references within components correct?
  - Embedding absolute names in code almost always causes problems as naming conventions differ from place to place
- Is the system being built for the right platform
  - Sometimes must build for a specific OS version or hardware configuration

- Is the right version of the compiler and other software tools specified?
  - Different compiler versions may actually generate different code and the compiled component will exhibit different behaviour

System representation

- Systems are normally represented for building by specifying the file name to be processed by building tools. Dependencies between these are described to the building tools
- Mistakes can be made as users lose track of which objects are stored in which files
- A system modelling language addresses this problem by using a logical rather than a physical system representation

Dependencies

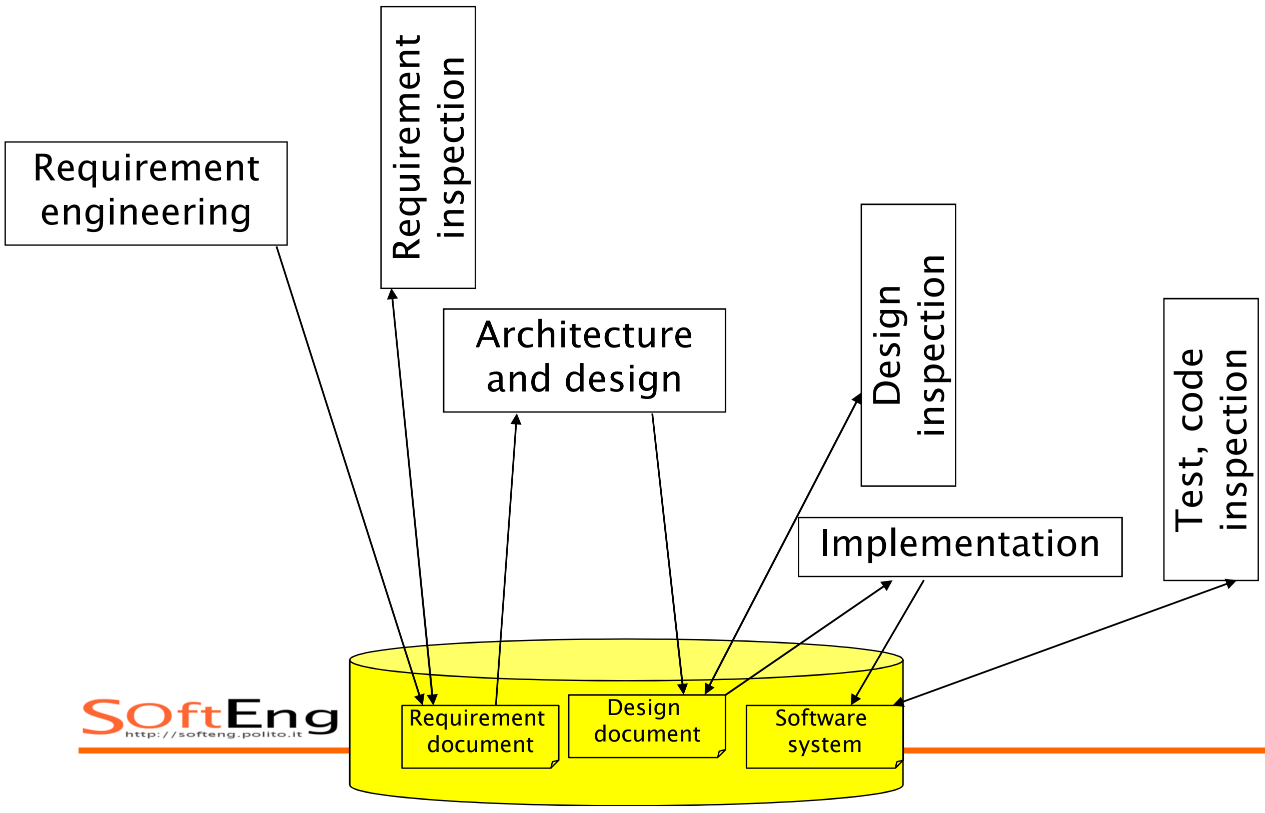


```
edit : main.o kbd.o command.o display.o insert.o search.o files.o utils.o
    cc -o edit main.o kbd.o command.o display.o insert.o  search.o files.o
    utils.o
main.o : main.c defs.h
     cc -c main.c
kbd.o : kbd.c defs.h command.h
     cc -c kbd.c
command.o : command.c defs.h command.h
    cc -c command.c
display.o : display.c defs.h buffer.h
    cc -c display.c
insert.o : insert.c defs.h buffer.h
   cc -c insert.c
search.o : search.c defs.h buffer.h
   cc -c search.c
files.o : files.c defs.h buffer.h command.h
   cc -c files.c
utils.o : utils.c defs.h
   cc -c utils.c
clean :
   rm edit main.o kbd.o command.o display.o insert.o  search.o files.o utils.o
```

Build tools

- Make
- Ant
- Maven

The process and CM



Summary

- CM is about
  - allowing to retrieve different (past) states of a document (versioning),
  - keeping track of consistent sets of documents (configurations and baselines)
  - offering a place to store documents (repository) and safe to ways to access them (lock modify unlock or copy modify merge)

References and Further Readings

- “Software configuration management: A roadmap”, J.Estublier, Proc. INt. Conf.onSoftware Engineering, 2000, IEEE Press.
- IEEE STD 1042 – 1987 IEEE guide to software configuration management
- IEE STD 828-2005 Standard for Software Configuration Mangement Plans
- “Configuration Management Principle and Practice”, A.M.J.Hass,2002, Addison Wesley

Tools

- CM + VM
  - RCS
  - CVS
  - SCCS
  - PCVS
  - Clearcase
  - Subversion
  - BitKeeper
- Build
  - Make
  - Ant
  - Maven

RCS

- Unit is file
- Baseline
- Check in check out
  - CI command
    - Inserts file in baseline
    - Associates comment explaining the change
    - Associates new version number (automatically or not)
  - CO command
    - Extracts file, in Rd or Wr mode

- Ident command
  - Associates name to file, starting from attributes (name author version)
- Rlog
  - Extracts from baseline description
    - List of composing files
    - Comments attached to files

- Storage of versions based on delta
  - Storage space saved
  - Check in / out can be slow
- Lock mechanism (default)
  - At checkout file is locked
  - Checkin possible only if user did checkout

CVS

- Built on top of RCS
- Client server
- Unit is file or directory
- Same commands as RCS (if applied to directory they are applied to all contained files)
- Check out with lock or not
  - Concurrent work on file possible
  - Reconciliation at checkin (semi automatic)

PCVS

- Client server
- Concepts
  - Project = set of files + directories
  - Archive = set of all versions of file
  - Revision = version of file
- Suite of tools
  - Version manager
  - Configuration builder (to support creation of release)
  - Tracker to support change request
  - Notify (via email) to notify changes

Functions

- Create project
- Browse project
- Check out (w w/out lock)
- Check in
- Reports
- Branch merge management

Svn – subversion

- See slides

Make

- Part of Unix
- Allows to describe components and dependencies among components
- Allows to describe operations to build system from components
- Builds system – recompiles only if component was changed (using data tag)

SoftEng http://softeng.polito.it

```
edit : main.o kbd.o command.o display.o insert.o search.o files.o utils.o
     cc –o edit main.o kbd.o command.o display.o insert.o  search.o files.o
     utils.o
main.o : main.c defs.h
     cc –c main.c
kbd.o : kbd.c defs.h command.h
     cc –c kbd.c
command.o : command.c defs.h command.h
    cc –c command.c
display.o : display.c defs.h buffer.h
    cc –c display.c
insert.o : insert.c defs.h buffer.h
   cc –c insert.c
search.o : search.c defs.h buffer.h
   cc –c search.c
files.o : files.c defs.h buffer.h command.h
   cc –c files.c
utils.o : utils.c defs.h
   cc –c utils.c
clean :
   rm edit main.o kbd.o command.o display.o insert.o  search.o files.o utils.o
```

ANT

- A build tool like make
- Open source
  - from the Apache Jakarta project
  - http://jakarta.apache.org/ant
- Implemented in Java
- Used to build many open source products
  - such as Tomcat and JDOM

Why Use Ant Instead of make?

- Ant is more portable
  - Ant only requires a Java VM (1.1 or higher)
  - make relies on OS specific commands to carry out it’s tasks
- Ant targets are described in XML
  - make has a cryptic syntax
  - make relies proper use of tabs that is easy to get wrong
    - you can’t see them
- Ant is better for Java-specific tasks
  - faster than make since all tasks are run from a single VM
  - easier than make for some Java-specific tasks such as generating javadoc, building JAR/WAR files and working with EJBs

How Does Ant Work?

- Ant commands (or tasks) are implemented by Java classes
  - many are built-in
  - others come in optional JAR files
  - custom commands can be created
- Each project using Ant will have a build file
  - typically called build.xml since Ant looks for this by default
- Each build file is composed of targets
  - these correspond to common activities like compiling and running code
- Each target is composed of tasks
  - executed in sequence when the target is executed
  - like make, Ant targets can have dependencies
    - for example, modified source files must be compiled before the application can be run

How ..

- Targets to be executed
  - can be specified on the command line when invoking Ant
  - if none are specified then the default target is executed
  - execution stops if an error is encountered so all requested targets may not be executed
- Each target is only executed once
  - regardless of the number of other targets that depend on it, ex:
    - the "test" and "deploy" targets both depend on "compile"
    - the "all" target depends on "test" and "deploy"
    - but "compile" is only executed once when "all" is executed
- Some tasks are only executed when they need to be
  - for example, files that have not changed since the last time they were compiled are not recompiled

Build file example (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="Web App." default="deploy" basedir=".">

  <!-- Define global properties. -->
  <property name="appName" value="shopping"/>
  <property name="buildDir" value="classes"/>
  <property name="docDir" value="doc"/>
  <property name="docRoot" value="docroot"/>
  <property name="junit" value="/Java/JUnit/junit.jar"/>
  <property name="srcDir" value="src"/>
  <property name="tomcatHome" value="/Tomcat"/>
  <property name="servlet" value="${tomcatHome}/lib/servlet.jar"/>
  <property name="warFile" value="${appName}.war"/>
  <property name="xalan" value="/XML/Xalan/xalan.jar"/>
  <property name="xerces" value="/XML/Xalan/xerces.jar"/>
```

relative directory references are relative to this

target that is run when none are specified

Some of these are used to set "**classpath**" on the next page. Others are used in task parameters.

Where possible, use **UNIX-style paths** even under Windows. This is not possible when Windows directories on drives other than C must be specified.

Build file example (2)

```
<path id="classpath">
  <pathelement path="${buildDir}"/>
  <pathelement path="${xerces}"/>
  <pathelement path="${xalan}"/>
  <pathelement path="${servlet}"/>
  <pathelement path="${junit}"/>
</path>

<target name="all" depends="test,javadoc,deploy"
 description="runs test, javadoc and deploy"/>
```

used in the compile, javadoc and test targets

means that the test, javadoc and deploy targets must be executed before this target

doesn't have any tasks of its own; just executes other targets

Build file example (3)

```
<target name="clean" description="deletes all generated files">
  <delete dir="${buildDir}"/> <!-- generated by the prepare target -->
  <delete dir="${docDir}/api"/> <!-- generated by the javadoc target -->
  <delete>
    <fileset dir=".">
      <include name="${warFile}"/> <!-- generated by the war target -->
      <include name="TEST-*.txt"/> <!-- generated by the test target -->
    </fileset>
  </delete>
</target>

<target name="compile" depends="prepare"
 description="compiles source files">
  <javac srcdir="${srcDir}" destdir="${buildDir}" classpathref="classpath"/>
</target>

<target name="deploy" depends="war,undeploy"
 description="deploys the war file to Tomcat">
  <copy file="${warFile}" tofile="${tomcatHome}/webapps/${warFile}"/>
</target>
```

means that the prepare target must be executed before this target

compiles all files in or below srcDir that have no .class file or have been modified since their .class file was created; don't have to list specific file names as is common with make

makes the servlet available through Tomcat; Tomcat won't expand the new war file unless the corresponding webapp subdirectory is missing

Build file example (4)

```
<target name="dtd" description="generates a DTD for Ant build files">
  <antstructure output="build.dtd"/>
</target>

<target name="javadoc" depends="compile"
 description="generates javadoc from all .java files">
  <delete dir="${docDir}/api"/>
  <mkdir dir="${docDir}/api"/>
  <javadoc sourcepath="${srcDir}" destdir="${docDir}/api"
   packagenames="com.ociweb.*" classpathref="classpath"/>
</target>

<target name="prepare" description="creates output directories">
  <mkdir dir="${buildDir}"/>
  <mkdir dir="${docDir}"/>
</target>
```

generates a DTD that is useful for learning the valid tasks and their parameters

generates javadoc for all .java files in or below srcDir.

can't just use a single * here and can't use multiple *'s

creates directories needed by other targets if they don't already exist

Build file example (5)

```
<target name="test" depends="compile" description="runs all JUnit tests">
  <!-- Delete previous test logs. -->
  <delete>
    <fileset dir=".">
      <include name="TEST-*.txt"/> <!-- generated by the test target -->
    </fileset>
  </delete>

  <taskdef name="junit"
   classname="org.apache.tools.ant.taskdefs.optional.junit.JUnitTask"/>
  <junit printsummary="yes">
    <classpath refid="classpath"/>
    <batchtest>
      <fileset dir="${srcDir}"><include name="**/*Test.java"/></fileset>
      <formatter type="plain"/>
    </batchtest>
  </junit>
</target>
```

runs all JUnit tests in or below srcDir

junit.jar must be in the **CLASSPATH** environment variable for this to work. It's not enough to add it to <path id="classpath"> in this file.

** specifies to look in any subdirectory at any depth

Build file example (6)

```
<target name="undeploy" description="undeploys the web app. from Tomcat">
  <delete dir="${tomcatHome}/webapps/${appName}"/>
  <delete file="${tomcatHome}/webapps/${warFile}"/>
</target>

<target name="war" depends="compile" description="builds the war file">
  <war warfile="${warFile}" webxml="web.xml">
    <classes dir="${buildDir}"/>
    <fileset dir="${docRoot}"/>
  </war>
</target>

</project>
```

makes the servlet unavailable to Tomcat

creates a web application archive (WAR) that can be deployed to a servlet engine like Tomcat

contains HTML, JavaScript, CSS and XSLT files

Download

- http://ant.apache.org
- http://ant.apache.org/manual

Commands

- ant [*option*s] [*target-name*s]
  - omit target-name to run the default target
  - runs targets with specified names, preceded by targets on which they depend
  - can specify multiple target-names separated by spaces
  - -D option specifies a property that can be used by targets and tasks
    - *-Dproperty-nam*e=*property-value*
- ant -help
  - lists other command-line options

Core tasks (some)

- Chmod
- Concat
- Copy
- Cvs
- Delete
- Exec
- Java
- Javac
- Javadoc
- Mail
- Mkdir
- Move
- Sleep
- Sql
- Tar
- Zip
- Unzip