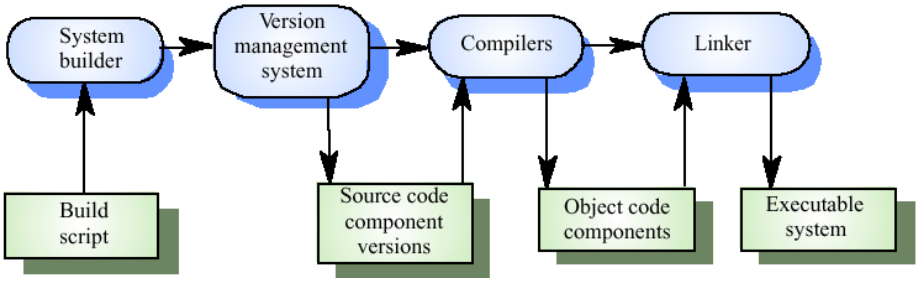
Build

System building

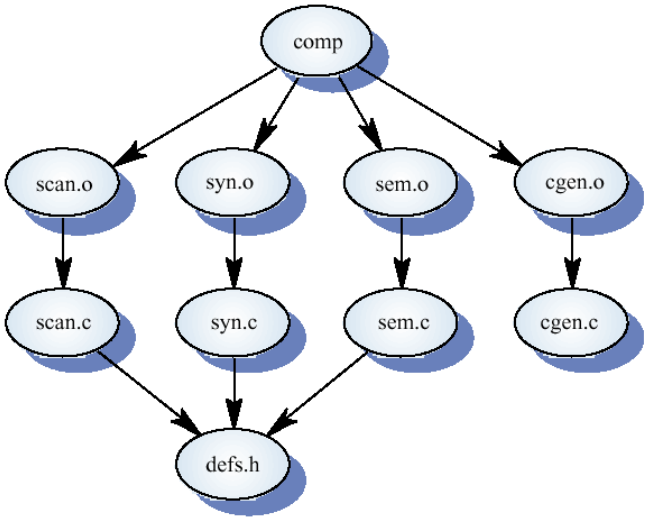
- The process of compiling and linking software components into an executable system
- Different systems are built from different combinations of components
- Invariably supported by automated tools that are driven by ‘build scripts’

System building



SOftEng

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

SOftEng
http://softeng.polito.it

- 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

SOftEng
http://softeng.polito.it

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

SOftEng http://softeng.polito.it

- 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

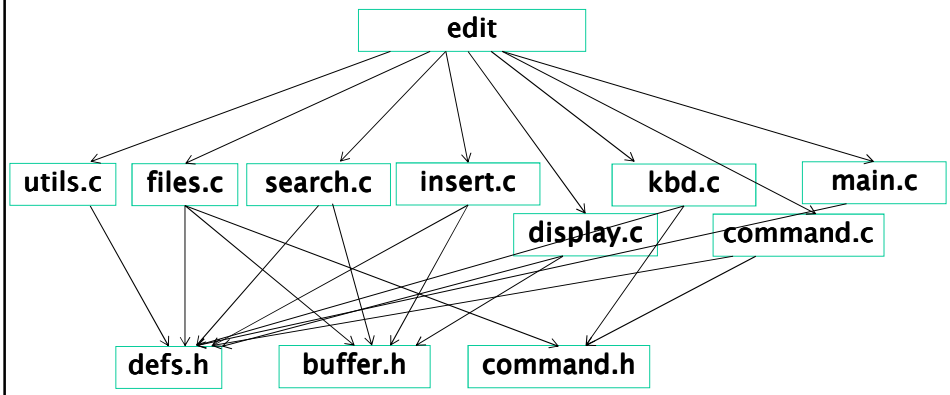
SOftEng http://softeng.polito.it

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

SOftEng http://softeng.polito.it

Dependencies



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
```

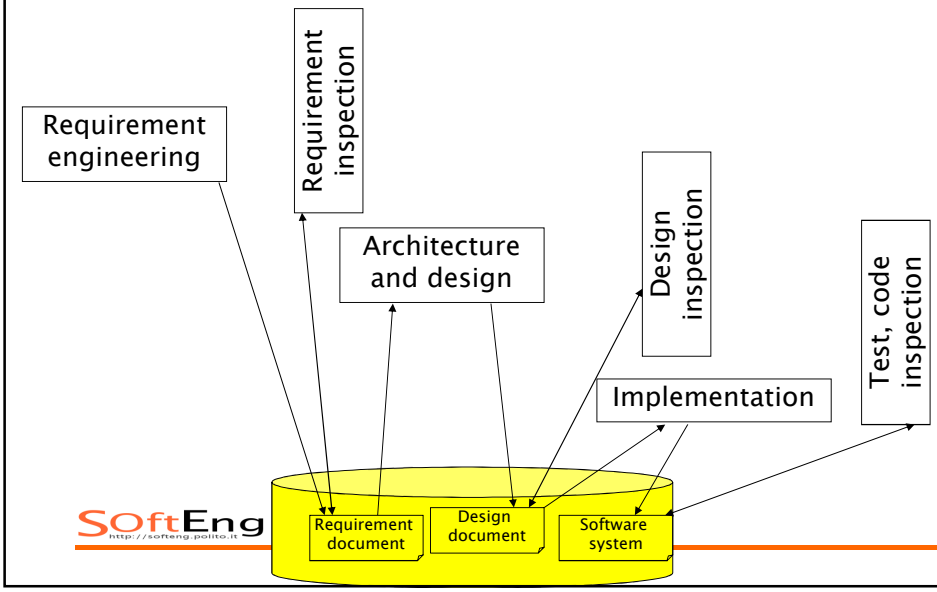
SOftEng http://softeng.polito.it

Build tools

- Make
- Ant
- Maven

SOftEng http://softeng.polito.it

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)

SOftEng
http://softeng.polito.it

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

SOftEng http://softeng.polito.it

Tools

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

SOftEng http://softeng.polito.it

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

SOftEng
http://softeng.polito.it

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)

SOftEng
http://softeng.polito.it

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

SOftEng http://softeng.polito.it

Functions

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

SOftEng http://softeng.polito.it

Svn – subversion

- See slides

SOftEng
http://softeng.polito.it

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

SOftEng http://softeng.polito.it

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

SOftEng http://softeng.polito.it

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

SOftEng http://softeng.polito.it

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

SOftEng http://softeng.polito.it

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-name=property-value*
- ant -help
  - lists other command-line options

SOftEng http://softeng.polito.it

Core tasks (some)

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

SOftEng http://softeng.polito.it

CMAKE

- Cross Platform Make
- open-source build system that enables developers to automate
  - Compiling
  - Testing
  - Packaging

SOftEng http://softeng.polito.it

CMAKE

- Can control different compilers and the build process works in conjunction with native build environments, e.g.:
  - Un*x make
  - Apple's Xcode
  - Microsoft Visual Studio
  - Code::Blocks

SOftEng http://softeng.polito.it