

15-413

Lecture Notes on Software Configuration Management

**Guenter Teubner
Technische Universitaet Muenchen
Institut fuer Informatik**

29 September 1998

Outline of the Lecture

- ❖ **Software Configuration Management (SCM)**
 - ◆ **Motivation: Why software configuration management?**
 - ◆ **Definition: What is software configuration management?**
- ❖ **Terminology and Methodology**
 - ◆ **What are Configuration Items, Baselines, etc. ?**
 - ◆ **What goes under version control?**
- ❖ **Software Configuration Management Plans**
 - ◆ **Standards (Example: IEEE 828-1990)**
 - ◆ **Basic elements of IEEE 828-1990**
- ❖ **Configuration Management Tools**
- ❖ **CVS at a glance**
 - ◆ **The basics of CVS**
 - ◆ **Examples for the most common use cases**

Why Software Configuration Management ?

- ❖ **The problem:**
 - ♦ **Multiple people have to work on software that is changing**
 - ♦ **More than one version of the software has to be supported:**
 - **Released systems**
 - **Custom configured systems (different functionality)**
 - **System(s) under development**
 - ♦ **Software must run on different machines and operating systems**

- ⇒ *Need for coordination*

- ❖ **Software Configuration Management**
 - ♦ **manages evolving software systems**
 - ♦ **controls the costs involved in making changes to a system**

What is Software Configuration Management

❖ Definition:

- ◆ A set of management disciplines within the software engineering process to develop a baseline.**

A thought bubble with a grey fill and a black outline, containing the text "Forward Definition!". It is connected to the text above by three small circles of decreasing size.

❖ Description:

- ◆ Software Configuration Management encompasses the disciplines and techniques of initiating, evaluating and controlling change to software products during and after the software engineering process.**

❖ Standards (approved by ANSI)

- ◆ IEEE 828: Software Configuration Management Plans**
- ◆ IEEE 1042: Guide to Software Configuration Management**

SCM Activities

- ❖ **Software Configuration Management (SCM) Activities:**
 - ◆ **Configuration identification (labeling and identification)**
 - ◆ **Baseline management**
 - ◆ **Change control (mechanism needed to coordinate parallel activities)**
 - ◆ **Reviews (status accounting, audits)**
 - ◆ **Release management**

- ❖ **No fixed rules:**
 - ◆ **SCM functions are usually performed in different ways (formally, informally) depending on the project type and life-cycle phase (research, development, maintenance).**

Terminology and Methodology

❖ What are

- ◆ Configuration Items**
- ◆ Baselines**
- ◆ SCM Directories**
- ◆ Versions, Revisions and Releases**

⇒ **The usage of the terminology presented here is not strict but varies for different configuration management systems. We will see for example that the configuration management system used for this class uses different names than those mentioned in the IEEE standards.**

Terminology: Configuration Item

“An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process.”

- ❖ **Software configuration items are not only program code segments but all type of documents according to development, e.g**
 - ⇒ **all type of code files**
 - ⇒ **drivers for tests**
 - ⇒ **analysis or design documents**
 - ⇒ **user or developer manuals**
 - ⇒ **system configurations (e.g. version of compiler used)**

- ❖ **In some systems, not only software but also hardware configuration items (CPUs, bus speed frequencies) exist!**

Finding Configuration Items (CIs)

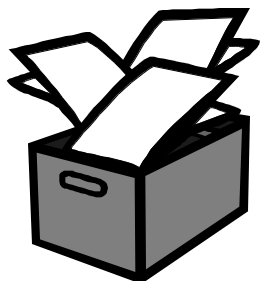
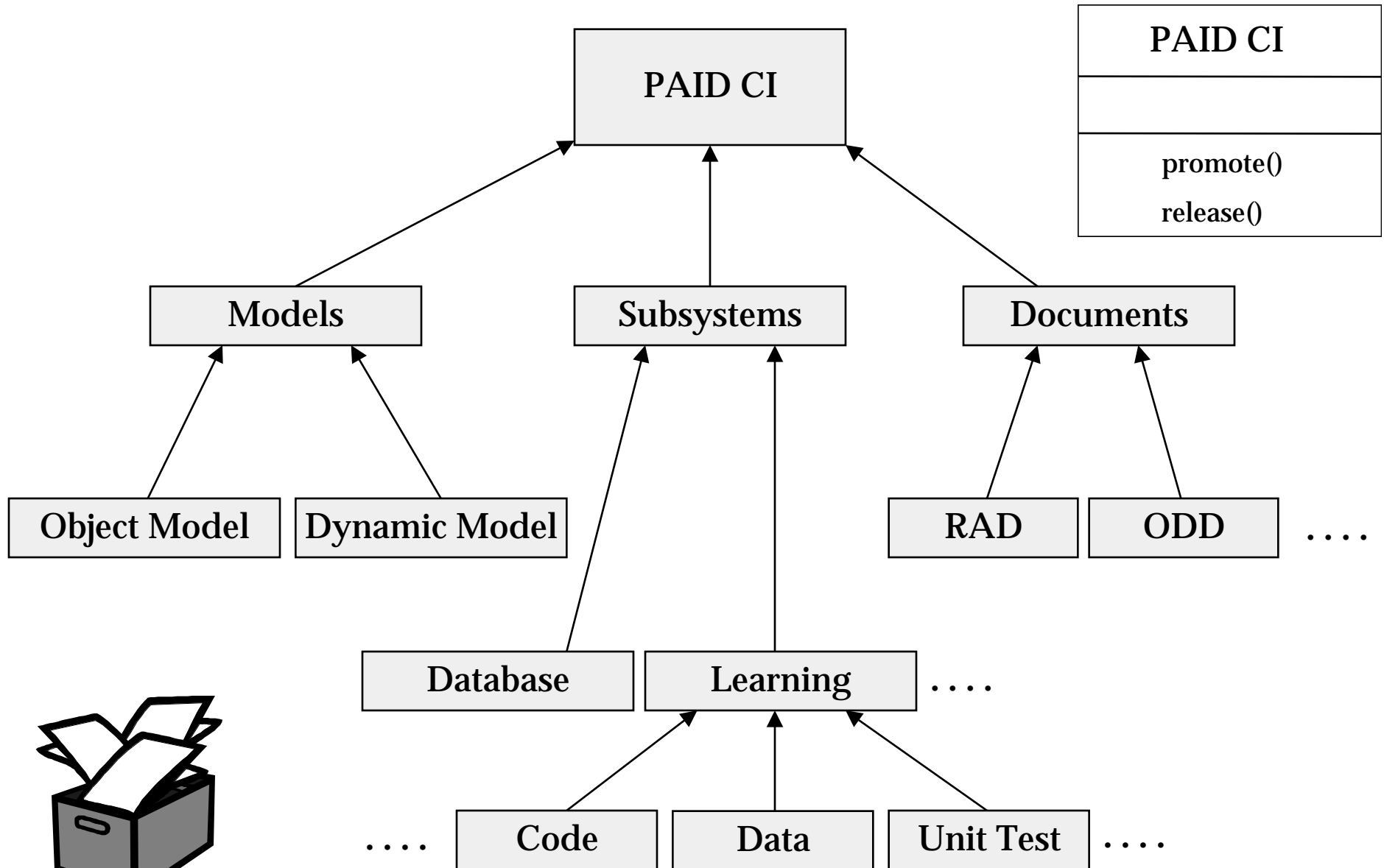
- ❖ Large projects typically produce thousands of entities (files, documents, ...) which must be uniquely identified.
- ❖ But not every entity needs to be configured all the time.
Issues:
 - ◆ **What: Selection of CIs (What should be managed?)**
 - ◆ **When: When do you start to place an entity under configuration control?**
- ⇒ Starting too early introduces too much bureaucracy
- ⇒ Starting too late introduces chaos

Finding Configuration Items (continued)

- ❖ **Some of these entities must be maintained for the lifetime of the software. This includes also the phase, when the software is no longer developed but still in use; perhaps by industrial customers who are expecting proper support for lots of years.**
- ❖ **An entity naming scheme should be defined so that related documents have related names.**

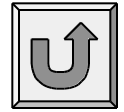
- ❖ **Selecting the right configuration items is a skill that takes practice**
 - ◆ **Very similar to object modeling**
 - ◆ **Use techniques similar to object modeling for finding CIs**

Configuration Identification is similar to Object Identification



Tasks for the Configuration Managers in PAID

Define configuration items



Terminology: Baseline

“A specification or product that has been formally reviewed and agreed to by responsible management, that thereafter serves as the basis for further development, and can be changed only through formal change control procedures.”

Examples:

Baseline A: The API of a program is completely defined; the bodies of the methods are empty.

Baseline B: All data access methods are implemented and tested; programming of the GUI can start.

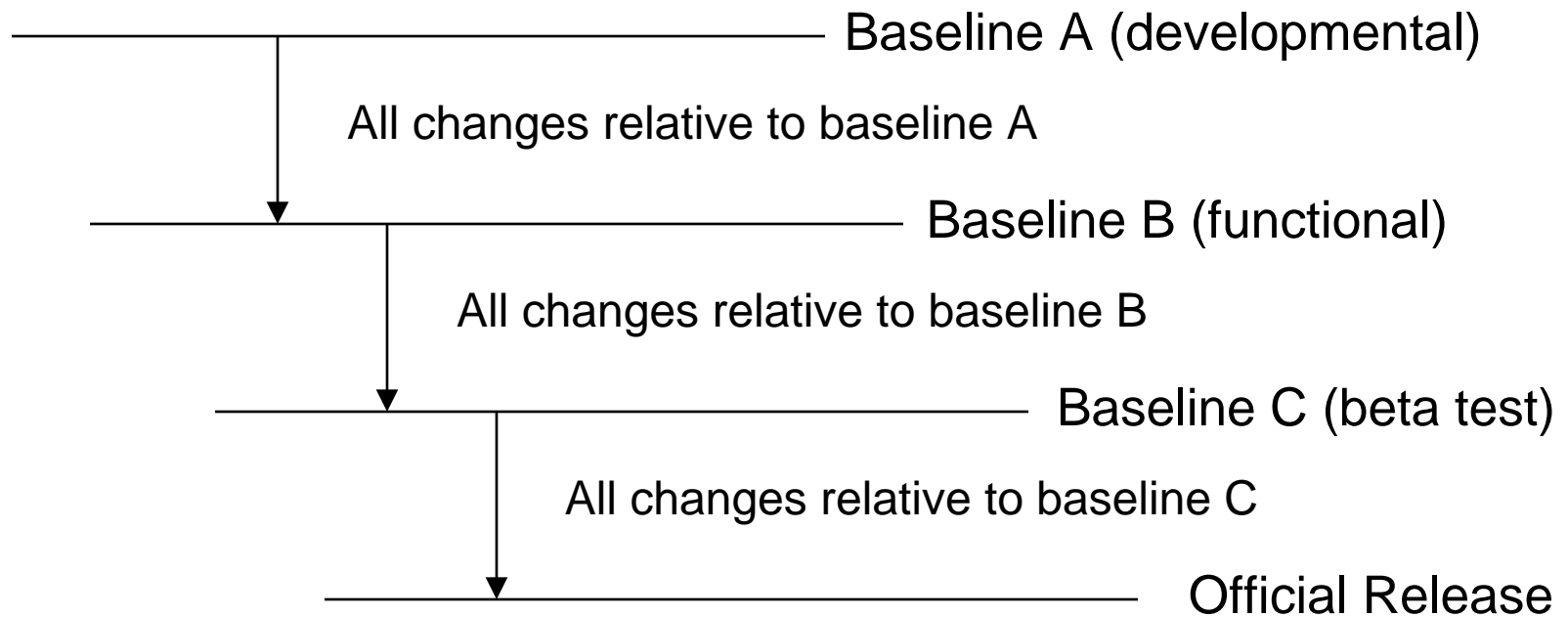
Baseline C: GUI is implemented, test-phase can start.

More on Baselines

- ❖ As systems are developed, a series of baselines is developed, usually after a review (analysis review, design review, code review, system testing, client acceptance, ...)
 - ♦ **Developmental baseline** (RAD, SDD, Integration Test, ...)
 - Goal: Coordinate engineering activities.
 - ♦ **Functional baseline** (first prototype, alpha release, beta release)
 - Goal: Get first customer experiences with functional system.
 - ♦ **Product baseline** (product)
 - Goal: Coordinate sales and customer support.
- ❖ Many naming scheme for baselines exist (1.0, 6.01a, ...)
- ❖ 3 digit scheme:



Baselines in SCM



SCM Directories

- ❖ **Programmer's Directory (IEEE: Dynamic Library)**
 - ◆ **Library for holding newly created or modified software entities. The programmer's workspace is controlled by the programmer only.**
- ❖ **Master Directory (IEEE: Controlled Library)**
 - ◆ **Manages the current baseline(s) and for controlling changes made to them. Entry is controlled, usually after verification. Changes must be authorized.**
- ❖ **Software Repository (IEEE: Static Library)**
 - ◆ **Archive for the various baselines released for general use. Copies of these baselines may be made available to requesting organizations.**

Standard SCM Directories

❖ Programmer's Directory

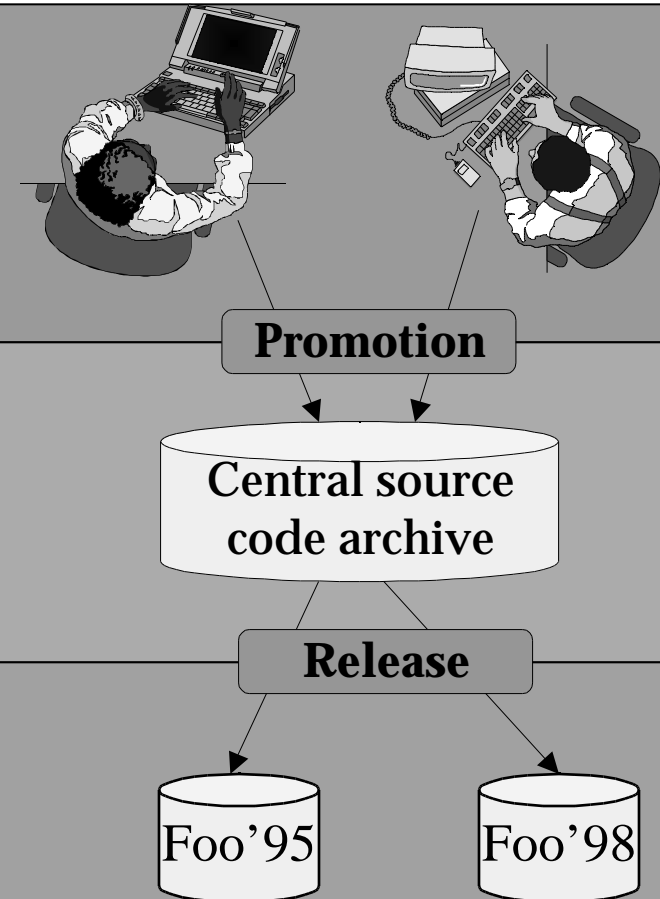
- ◆ (IEEE Std: "Dynamic Library")
- ◆ Completely under control of one programmer.

❖ Master Directory

- ◆ (IEEE Std: "Controlled Library")
- ◆ Central directory of all promotions.

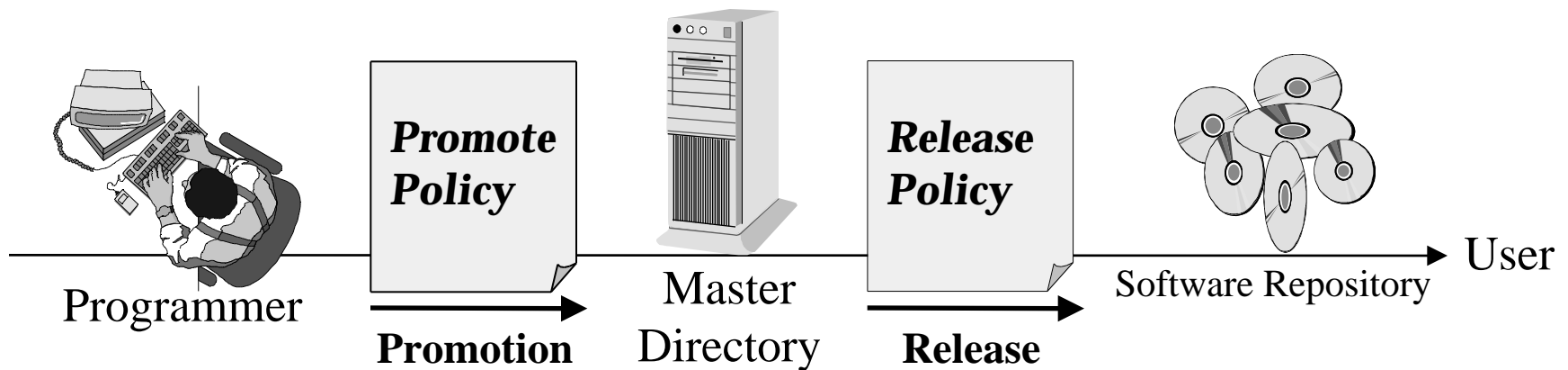
❖ Software Repository

- ◆ (IEEE Std: "Static Library")
- ◆ Externally released baselines.



Controlling Changes

- ❖ Two types of controlling change:
 - ♦ **Promotion:** The internal development state of a software is changed.
 - ♦ **Release:** A set of promotions is distributed outside the development organization.



- ❖ Approaches for controlling change to libraries (Change Policy)
 - ♦ **Informal** (good for research type environments)
 - ♦ **Formal approach** (good for externally developed CIs and for releases)

Change Policies

- ❖ Whenever a promotion or a release is performed, one or more policies apply. The purpose of change policies is to guarantee that each version, revision or release (see next slide) conforms to commonly accepted criteria.
- ❖ Examples for change policies:
 - No developer is allowed to promote source code which cannot be compiled without errors and warnings.**
 - No baseline can be released without having been beta-tested by at least 500 external persons.**

Tasks for the Configuration Managers in PAID

Define configuration items

Define promote /release policies

Version vs. Revision vs. Release

❖ Version:

- ◆ An *initial* release or re-release of a configuration item associated with a *complete compilation* or recompilation of the item. Different versions have different functionality.

❖ Revision:

- ◆ *Change* to a version that corrects only errors in the design/code, but does not affect the documented functionality.

Quiz: Is Windows98 a new version or a new revision compared to Windows95 ?

❖ Release:

- ◆ The *formal distribution* of an approved version.

SCM planning

- ❖ Software configuration management planning starts during the early phases of a project.
- ❖ The outcome of the SCM planning phase is the *Software Configuration Management Plan (SCMP)* which might be extended or revised during the rest of the project.
- ❖ The SCMP can either follow a public standard like the IEEE 828, or an internal (e.g. company specific) standard.

The Software Configuration Management Plan

- ❖ Defines the *types of documents* to be managed and a document naming scheme.
- ❖ Defines *who takes responsibility* for the CM procedures and creation of baselines.
- ❖ Defines *policies for change control* and version management.
- ❖ Describes the *tools* which should be used to assist the CM process and any limitations on their use.
- ❖ Defines the *configuration management database* used to record configuration information.

Outline of a Software Configuration Management Plan (SCMP, IEEE 828-1990)

- ❖ 1. Introduction
 - ◆ **Describes purpose, scope of application, key terms and references**
- ❖ 2. Management (WHO?)
 - ◆ **Identifies the responsibilities and authorities for accomplishing the planned configuration management activities**
- ❖ 3. Activities (WHAT?)
 - ◆ **Identifies the activities to be performed in applying to the project.**
- ❖ 4. Schedule (WHEN?)
 - ◆ **Establishes the sequence and coordination of the SCM activities with project mile stones.**
- ❖ 5. Resources (HOW?)
 - ◆ **Identifies tools and techniques required for the implementation of the SCMP**
- ❖ 6. Maintenance
 - ◆ **Identifies activities and responsibilities on how the SCMP will be kept current during the life-cycle of the project.**

Tasks for the Configuration Managers in PAID

Define configuration items

Define promote /release policies

Define responsibilities

Tailoring the SCMP

- ❖ The IEEE standard allows quite a bit flexibility for preparing an SCMP.

- ❖ To conform to the rest of the project, the SCMP may be
 - ◆ **tailored upward:**
 - to add information
 - to use a specific format
 - ◆ **tailored downward**
 - **Some SCMP components might not apply to a particular project.**
 - **Instead of omitting the associated section, mention its applicability.**
 - **Information that has not been decided on at the time the SCMP is approved should be marked as “to be determined”.**

Conformance to the IEEE Standard 828-1990

- ❖ **Presentation format & Minimum information**
 - ♦ **A separate document or a section embedded in another document titled “Software Configuration Management Plan”.**
 - ♦ **6 Sections: Introduction, Management, Activities, Schedules, Resources and Plan Maintenance**
- ❖ **Consistency Criteria:**
 - ♦ **All activities defined in the SCMP are assigned to an organizational unit or person and they are associated with resources to accomplish the activities.**
 - ♦ **All identified Configuration items have defined processes for baseline establishment and change control.**
- ❖ **If the above criteria are met, the SCMP can include the following sentence:**
 - ♦ ***“This SCMP conforms with the requirements of IEEE Std 828-1990.”***

Tools for Software Configuration Management

- ❖ **Software configuration management is normally supported by tools with different functionality.**

- ❖ **Examples:**
 - ◆ **RCS**
 - **very old but still in use; only version control system**
 - ◆ **CVS**
 - **based on RCS, allows concurrent working without locking**
 - ◆ **Perforce**
 - **Repository server; keeps track of developer's activities**
 - ◆ **ClearCase**
 - **Multiple servers, process modeling, policy check mechanisms**

Tasks for the Configuration Managers in PAID

SCMP following the IEEE 828-1990 standard

Define configuration items

Define promote /release policies

Define responsibilities

Set up configuration management system

Summary

- ❖ Software Configuration Management is an elementary part of the project management plan to manage evolving software systems and coordinate changes to them.
- ❖ SCM is performed by following a SCM plan. This plan can either follow a public standard (e.g. IEEE 828) or an internal standard.
- ❖ It is necessary to tailor a standard to a particular project:
 - ♦ Large projects need detailed plans to be successful
 - ♦ Small projects can't afford the bureaucracy of such plans
- ❖ SCM is supported by tools. Their functionality varies from simple version storage tools to very sophisticated systems with automated procedures for policy checks and support for the creation of SCM documents.

CVS

Concurrent Version System

CVS at a glance

- ❖ We will use CVS (Concurrent Version System) as version management system for this project during development. CVS is a shell-based, freely available configuration management system with a very short learning time for standard users.
- ❖ For easy access, we provide also a web-interface to the repository which allows no changes to the repository.
- ❖ The *Configuration Manager* of the project is Kent Ma. He will install and operate the system and provide basic help for the other students.

How CVS works ...

❖ Configuration Manager

- ◆ Creates one central repository for all developers.**
- ◆ Structures the repository by defining *modules* which represent directory trees.**

❖ Developer

- ◆ Has his/her own working directory.**
- ◆ Selects the part of the repository (modules) he wants to work with.**
- ◆ Receives copies of all these modules. He can then work on his local copies.**
- ◆ Adds new files to the repository.**
- ◆ Modifies existing ones.**
- ◆ *Resolves conflicts* when two developers have edited the same part of the same file simultaneously.**

CVS Tutorial Operations

- ❖ **Setting up a master directory (CVS: Repository)**
- ❖ **Creating the programmers directory**
- ❖ **Adding a new file to the repository**
- ❖ **Getting a file from the repository**
- ❖ **Updating a file in your local directory**
- ❖ **Editing a file and resolving a conflict**
- ❖ **Promoting your changes to a file**
- ❖ **Getting information about a file**
- ❖ **Deleting a file from the repository**

Creating the programmers directory

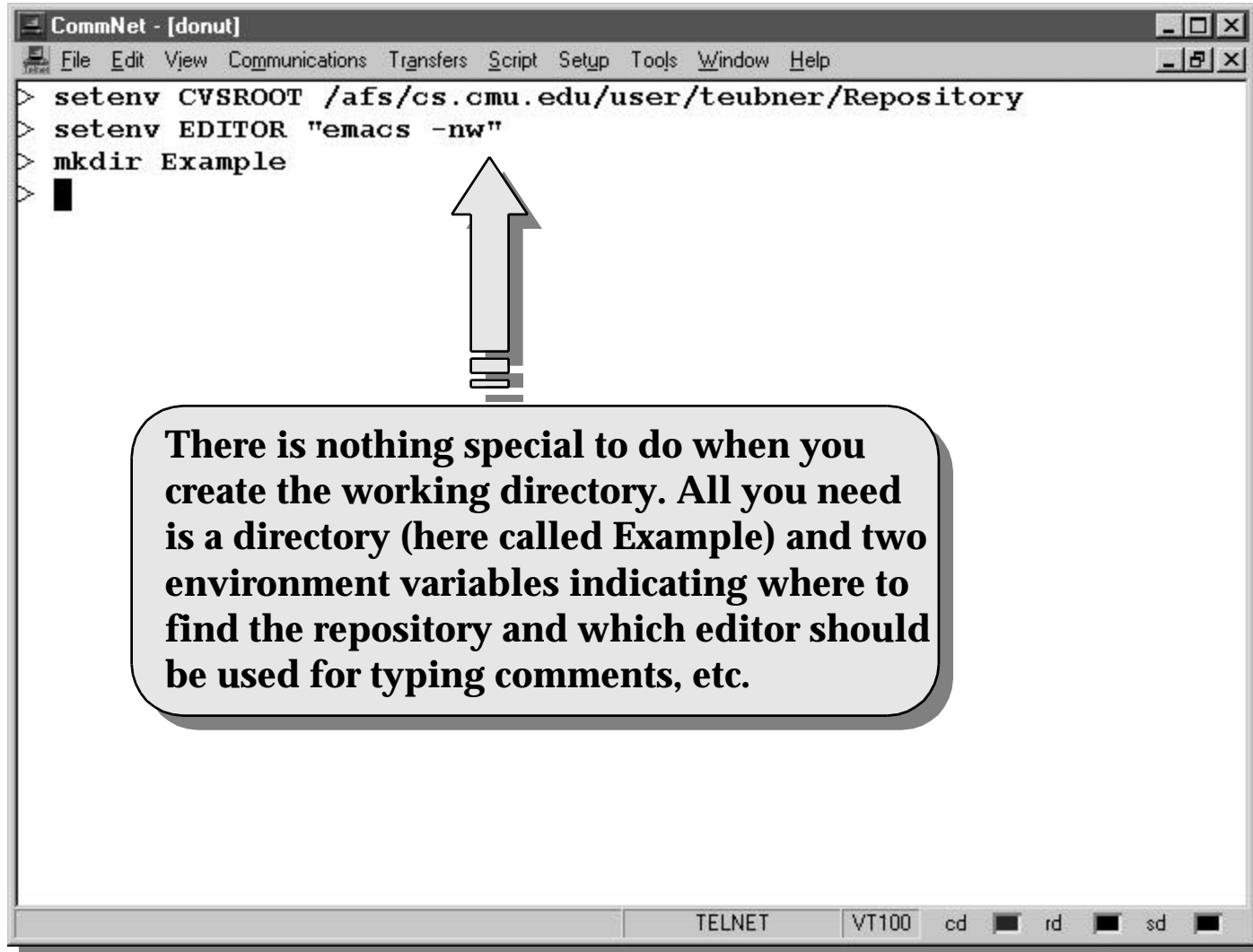
- ❖ Before you can use CVS, you have to set the following two environment variables:
 - ◆ *Variable:* **CVSROOT**
 - ◆ *Value:* **<Directory has to be defined>**
 - ◆ *Variable:* **EDITOR**
 - ◆ *Value:* **<Your preferred editor>**
- ❖ Set these variables in your login-scripts so that you can be sure, that they definitely exist when you are working in the computer lab.
- ❖ Create a directory for the PAID, enter it and type
`cvs checkout <module name>`

Creating the programmers directory (continued)

- ❖ What is this “*module name*”?
 - ◆ **CVS doesn't work on ordinary directory trees; you need to work within a directory that CVS created for you. Just as you check out a book from a library before taking it home to read it, you use the `cv` checkout command to get a directory tree from CVS before working on it. The module name specifies, which tree you get from the repository as there can be more than one.**

- ❖ **Only the configuration manager can create modules for you. If you want to establish a new module (e.g. for bringing also the HTML documentation under version control), you have to contact Kent Ma.**

Creating the programmers directory



```
CommNet - [donut]
File Edit View Communications Transfers Script Setup Tools Window Help
> setenv CVSROOT /afs/cs.cmu.edu/user/teubner/Repository
> setenv EDITOR "emacs -nw"
> mkdir Example
> █
```

There is nothing special to do when you create the working directory. All you need is a directory (here called Example) and two environment variables indicating where to find the repository and which editor should be used for typing comments, etc.

TELNET VT100 cd rd sd

Getting files from the repository

- ❖ **Situation:** You want to get a copy of a file you have currently not in your local directory. If the file belongs to a different module, you have to checkout the module.
- ❖ **Command:** `cvs checkout module`
- ❖ **If you deleted a file by accident and want it back,** simply request the most recent version from the repository. *Remember that this will not bring back the changes you made locally before!*
- ❖ **Command:** `cvs update filename`

Getting files from the repository

```
CommNet - [donut]
File Edit View Communications Transfers Script Setup Tools Window Help
> cd Example/
> cvs checkout HelloWorld.java
U ./HelloWorld.java
> cvs checkout docs
cvs checkout: Updating docs
U docs/rad.html
U docs/spmp.html
```

The letter in front of the file name indicates the performed operation on the file. The U means *updated*.

The *cvs checkout* command creates copies of the files in the repository in your local directory. You can retrieve either single files or complete directories .

TELNET VT100 cd rd sd

Working with your files

The main terminal window, titled "CommNet - [donut]", shows the following commands and output:

```
> javac HelloWorld.java
> java HelloWorld
Hello World
>
> javac GoodbyeWorld.java
> java GoodbyeWorld
Goodbye World
>
```

An inset window titled "GoodbyeWorld - Source" shows the source code for `GoodbyeWorld.java`:

```
// The mother of all programs
public class GoodbyeWorld {
    public static void main(String args[])
    {
        System.out.println("Goodbye World");
    }
}
```

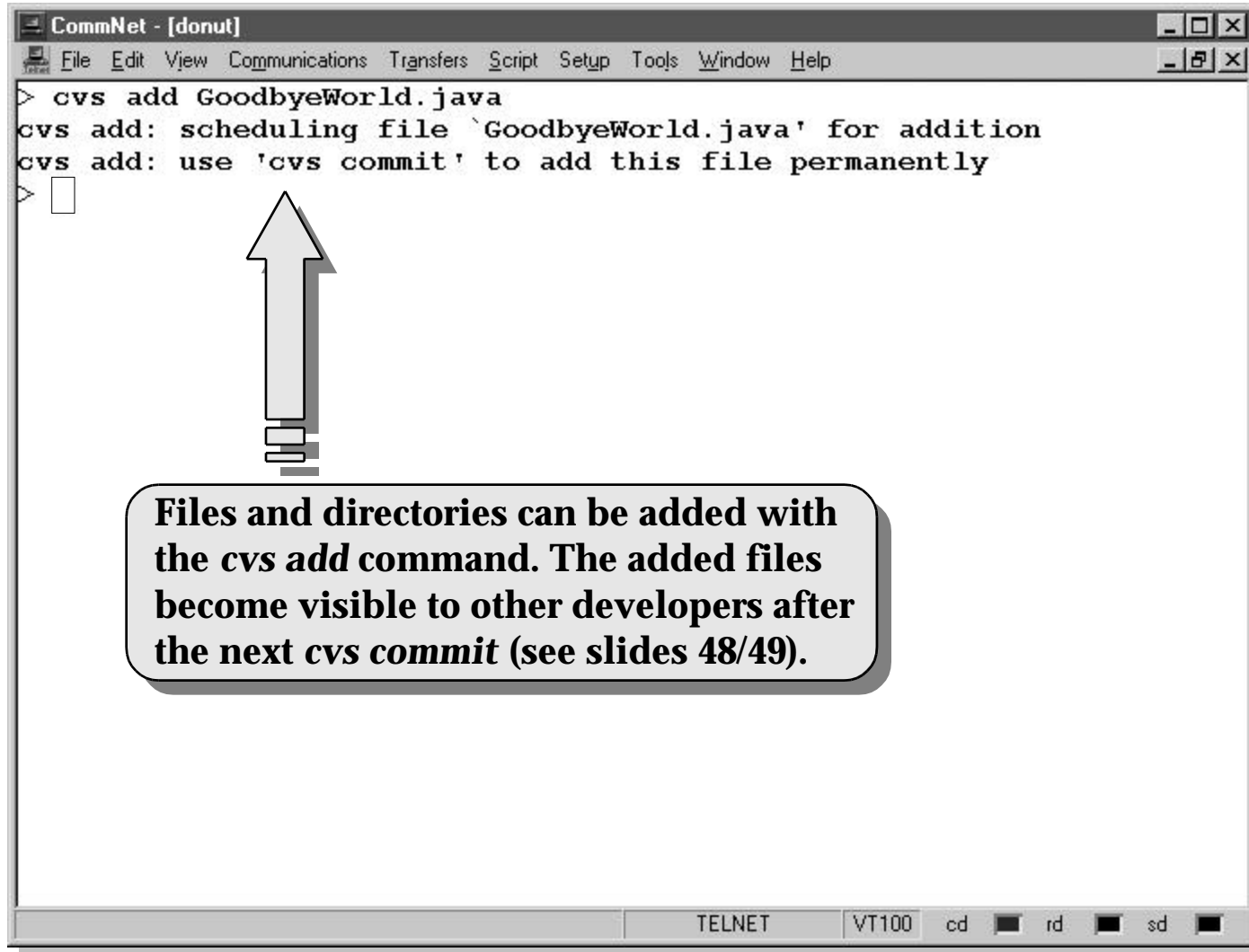
A callout bubble with an arrow pointing to the `javac GoodbyeWorld.java` command in the terminal says: "Create a new file in the editor".

Another callout bubble at the bottom left says: "You can work with the files in the working directory as if CVS does not exist. You can edit or compile them and you can also create new files (here *GoodbyeWorld.java*)." The terminal status bar at the bottom shows "TELNET VT100 cd rd sd".

Adding a file to the repository

- ❖ **Situation:** A file which is currently not under version control has to be added to the repository.
- ❖ **Command:** `cvs add filename`
- ❖ **You still have to do a `cvs commit` after this command to make the addition(s) actually take affect.**
- ❖ **You may make any number of new files in your programmers directory, but they will not be committed to the central repository unless you do a `cvs add`.**

Adding a file (example)



```
CommNet - [donut]
File Edit View Communications Transfers Script Setup Tools Window Help
> cvs add GoodbyeWorld.java
cvs add: scheduling file `GoodbyeWorld.java' for addition
cvs add: use 'cvs commit' to add this file permanently
> 
```

Files and directories can be added with the *cvs add* command. The added files become visible to other developers after the next *cvs commit* (see slides 48/49).

TELNET VT100 cd rd sd

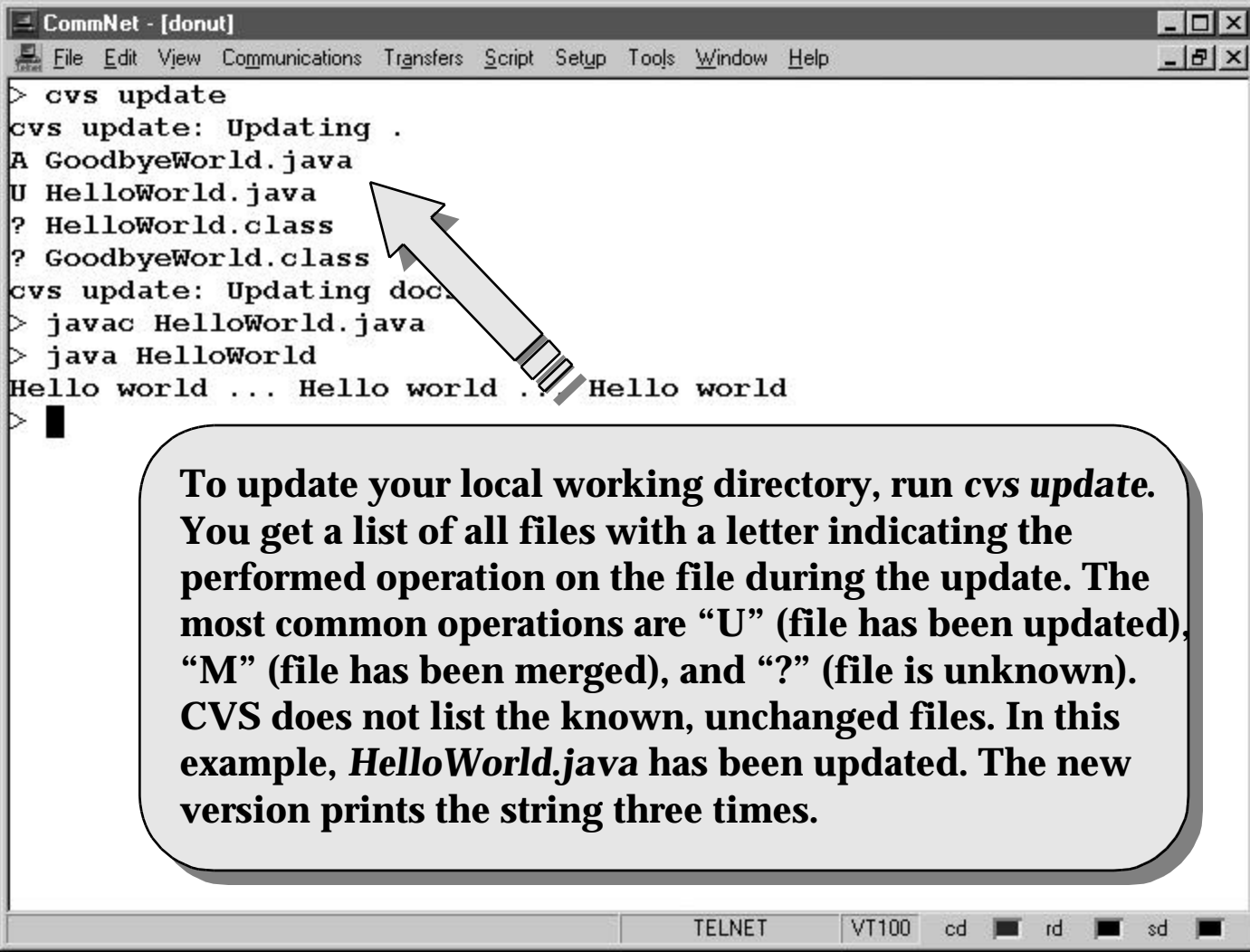
Updating a local file

- ❖ **Situation:** You have a local, possibly outdated copy of a file which you have edited. You now want to get the most recent version with all changes done by other developers (merged with your changes).

- ❖ **Command:** `cvs update filename`

- ❖ **CVS will inform you, if the file**
 - ◆ **has not changed in the meantime**
 - ◆ **has been updated (new code only from other developers)**
 - ◆ **has been merged (new code also from you)**
 - ◆ **couldn't be completely merged, because a conflict was found**

Updating a local file



```
CommNet - [donut]
File Edit View Communications Transfers Script Setup Tools Window Help
> cvs update
cvs update: Updating .
A GoodbyeWorld.java
U HelloWorld.java
? HelloWorld.class
? GoodbyeWorld.class
cvs update: Updating doc
> javac HelloWorld.java
> java HelloWorld
Hello world ... Hello world ... Hello world
>
```

To update your local working directory, run *cvs update*. You get a list of all files with a letter indicating the performed operation on the file during the update. The most common operations are “U” (file has been updated), “M” (file has been merged), and “?” (file is unknown). CVS does not list the known, unchanged files. In this example, *HelloWorld.java* has been updated. The new version prints the string three times.

TELNET VT100 cd rd sd

Resolving a conflict

- ❖ **Situation:** You asked for an update of one or more files. During this process, you got at least one message looking like the following:

```
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in <filename>
```

- ❖ **To resolve the conflict,** start your favorite editor and search the file for the following construct:

```
<<<<<< filename
  if (! error) {
    exit (1);
  }
=====
  if (! error) {
    exit (0);
  }
>>>>>> 1.9
```

Resolving a conflict (continued)

- ❖ The text from your working file appears at the top, after the `<<<` characters; below it is the conflicting text from the other developer. The revision number `1.9` indicates that the conflicting change was introduced in version 1.9 of the file, making it easier for you to check the logs, or examine the entire change with `cv diff`.
- ❖ Once you've decided how the conflict should be resolved, remove the markers from the code, and put it in its proper state.
- ❖ Repeat this process for all files with an error message and all occurrences within them.

Resolving a conflict

You can't commit your changes if your version of a file is not up-to-date!

```
CommNet - [donut]
File Edit View Communicat...
> cvs commit HelloWorld.java
cvs commit: Up-to-date check failed for `HelloWorld.java'
cvs [commit aborted]: correct above errors first!
>
> cvs update
cvs update: Updating .
A GoodbyeWorld.java
RCS file: /afs/cs.cmu.edu/user/teubner/Repository/./HelloWorld.java,v
retrieving revision 1.2
retrieving revision 1.3
Merging differences between 1.2 and 1.3 into HelloWorld.java
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in HelloWorld.java
C HelloWorld.java
? HelloWorld.class
? GoodbyeWorld.class
cvs update: Updating docs
> █
```

The *cvs update* command throws a warning when a merging conflict occurs. You have to resolve this conflict manually before you can commit your changes.

Resolving a conflict (continued)

```
CommNet - [donut]
File Edit View Communications Transfers Script Setup Tools Window Help
Buffers Files Tools Edit Search Java Help
// THE mother of all programs

public class HelloWorld {

    public static void main(String argv[])
    {
<<<<<<< HelloWorld.java
        System.out.println("Hello world ...");
=====
        System.out.println("Hello universe");
>>>>>>> 1.3
    }
}

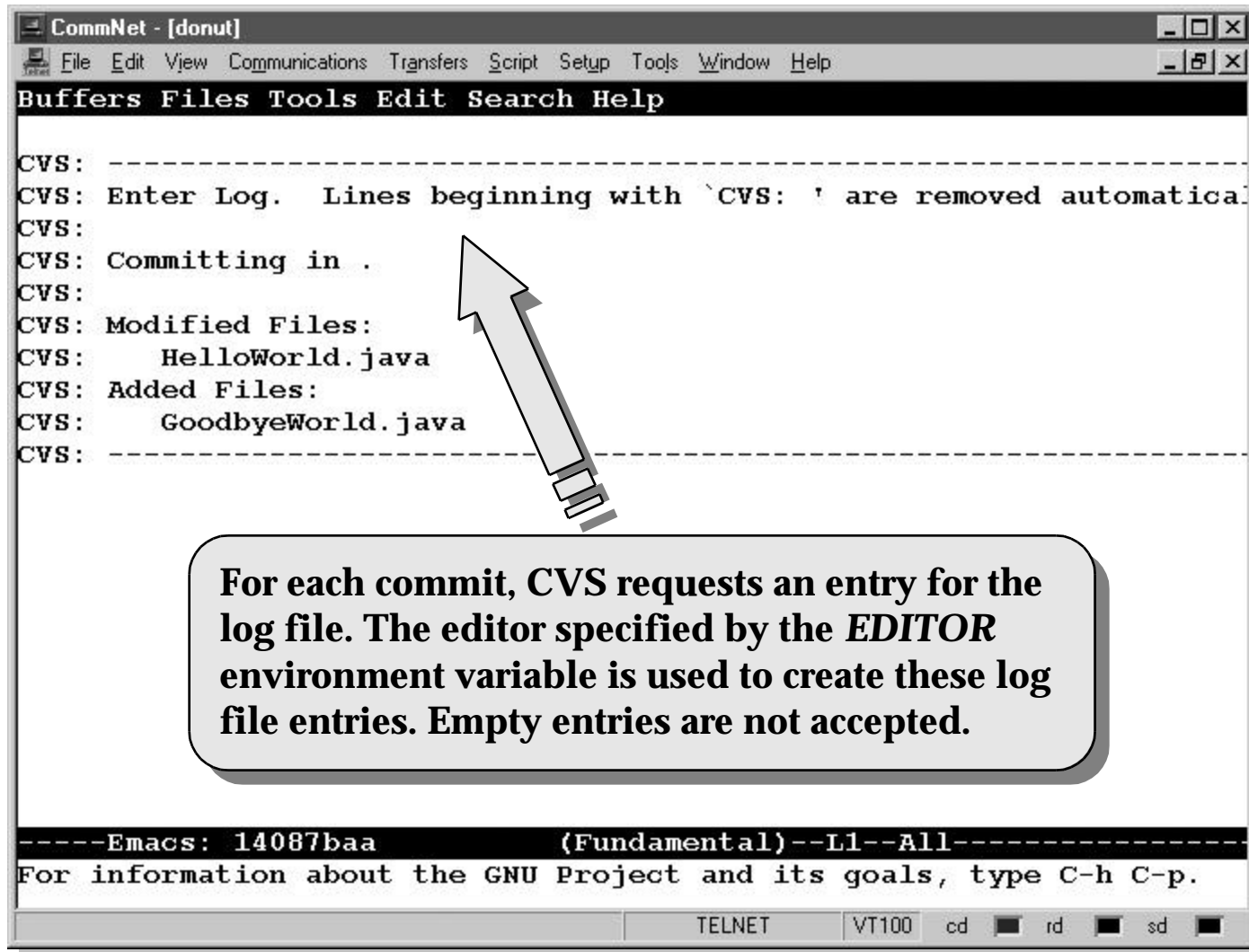
-----Emac-----All-----
TELNET VT100 cd rd sd
```

These are the conflicting lines of code. The developer has to select the correct version and remove the

Committing your changes

- ❖ **Situation:** You have brought your sources up-to-date with the rest of the group and tested them, so you are ready to commit your changes to the repository and make them visible to the rest of the group.
- ❖ **Command:** `cvsv commit filename`
- ❖ **At this point,** CVS will start up your favorite editor and prompt you for a log message describing the change. When you exit the editor, CVS will commit your change, it is now visible to the rest of the group. When another developer runs `cvsv update`, CVS will merge your changes to '*filename*' into their working directory.

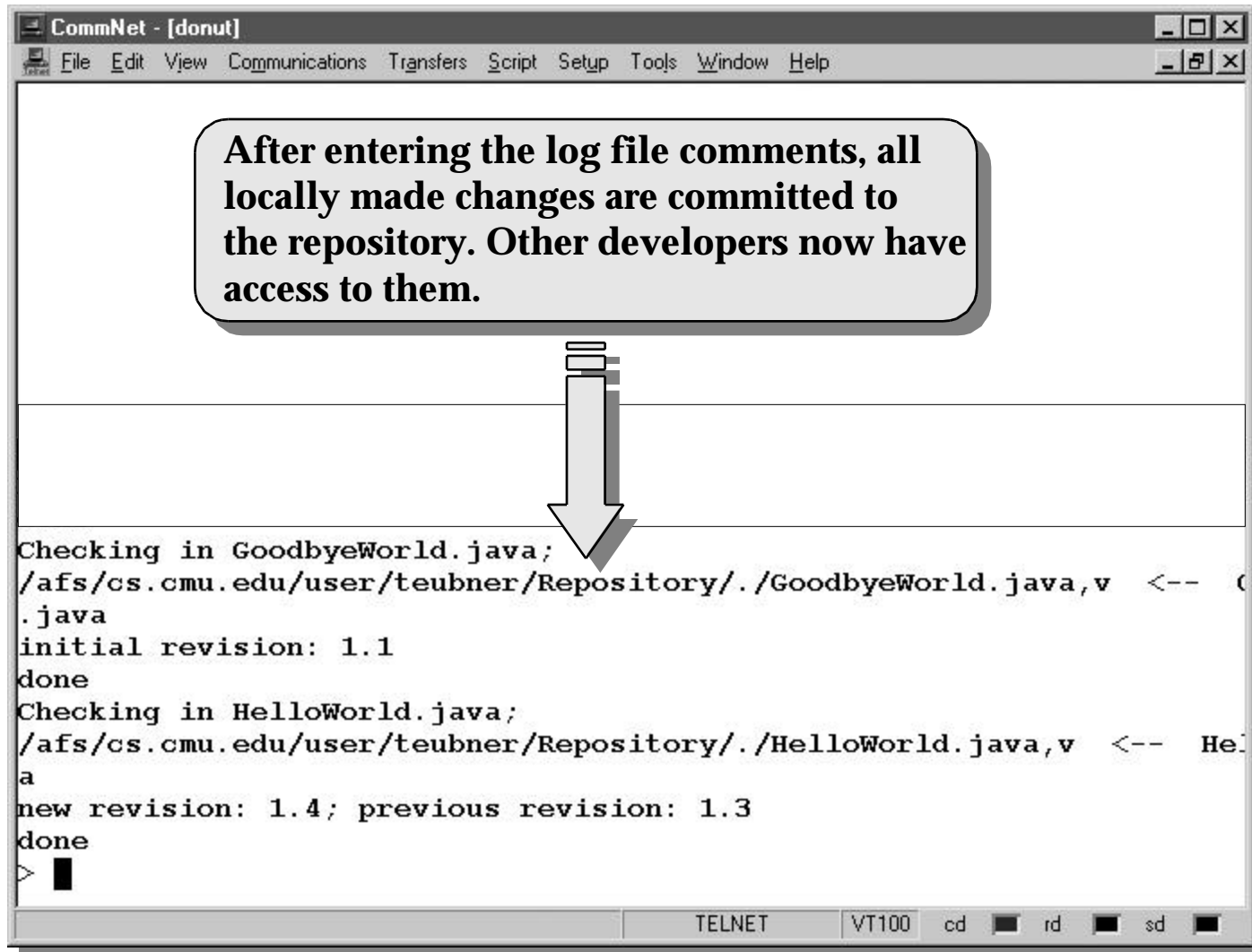
Committing your changes



```
CommNet - [donut]
File Edit View Communications Transfers Script Setup Tools Window Help
Buffers Files Tools Edit Search Help
CVS: -----
CVS: Enter Log.  Lines beginning with `CVS: ' are removed automatica
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:   HelloWorld.java
CVS: Added Files:
CVS:   GoodbyeWorld.java
CVS: -----
-----Emacs: 14087baa      (Fundamental)--L1--All-----
For information about the GNU Project and its goals, type C-h C-p.
TELNET VT100 cd rd sd
```

For each commit, CVS requests an entry for the log file. The editor specified by the *EDITOR* environment variable is used to create these log file entries. Empty entries are not accepted.

Committing your changes (continued)



CommNet - [donut]
File Edit View Communications Transfers Script Setup Tools Window Help

After entering the log file comments, all locally made changes are committed to the repository. Other developers now have access to them.

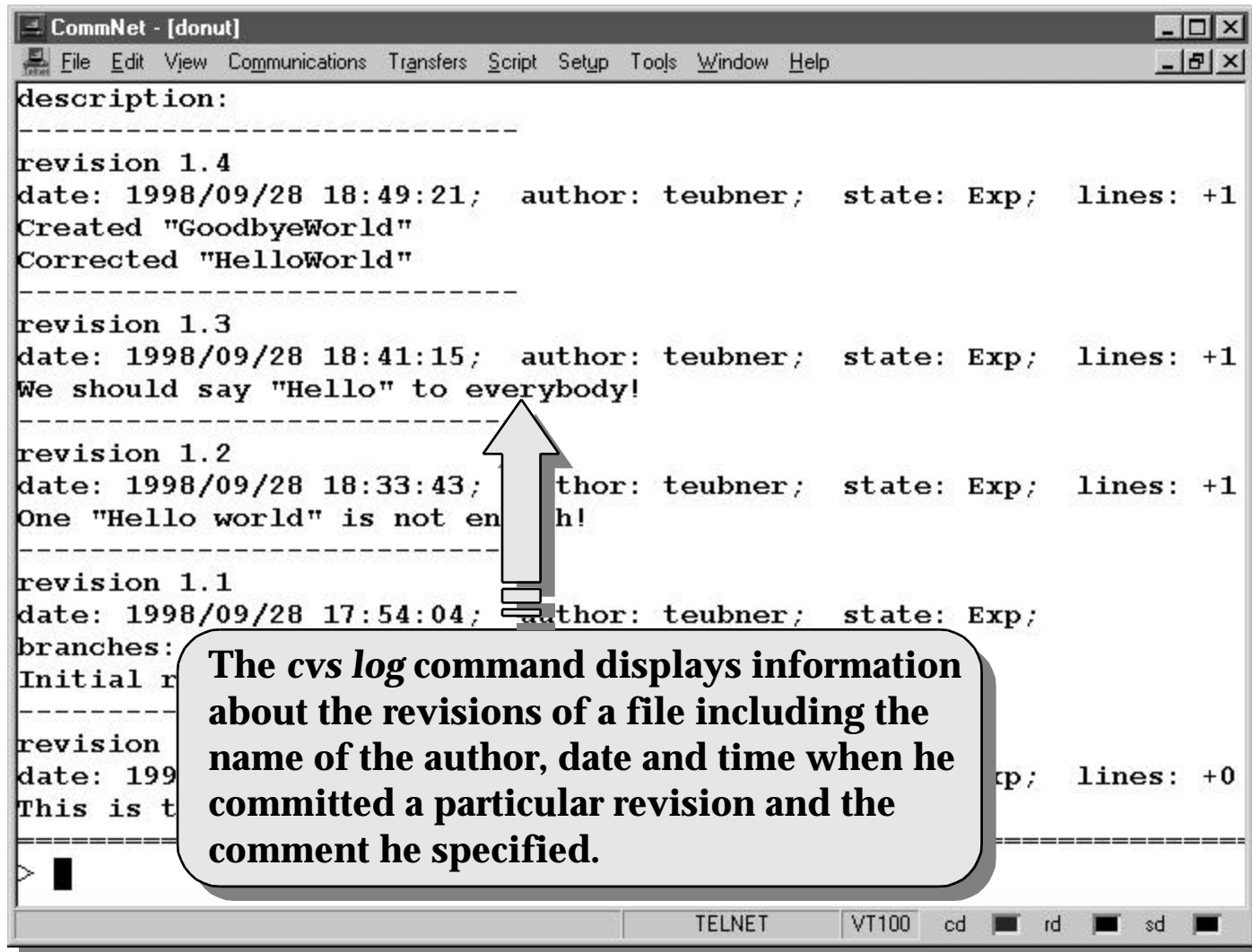
```
Checking in GoodbyeWorld.java;  
/afs/cs.cmu.edu/user/teubner/Repository/./GoodbyeWorld.java,v <-- C  
.java  
initial revision: 1.1  
done  
Checking in HelloWorld.java;  
/afs/cs.cmu.edu/user/teubner/Repository/./HelloWorld.java,v <-- He  
a  
new revision: 1.4; previous revision: 1.3  
done  
> █
```

TELNET VT100 cd rd sd

Getting some information about a file

- ❖ **Situation:** You might well be curious what changes the other developers made to a file.
- ❖ **Command:** `cvls log filename`
- ❖ You will see the log for this particular file. You get a list of all revisions including the name of the developer, the message describing the changes he/she did, the date and time of the commit and the differences between the revisions.
- ❖ You can find a description of the diff-format in the CVS documentation on the homepage of this course!

Getting some information about a file



```
CommNet - [donut]
File Edit View Communications Transfers Script Setup Tools Window Help
description:
-----
revision 1.4
date: 1998/09/28 18:49:21; author: teubner; state: Exp; lines: +1
Created "GoodbyeWorld"
Corrected "HelloWorld"
-----
revision 1.3
date: 1998/09/28 18:41:15; author: teubner; state: Exp; lines: +1
We should say "Hello" to everybody!
-----
revision 1.2
date: 1998/09/28 18:33:43; author: teubner; state: Exp; lines: +1
One "Hello world" is not enough!
-----
revision 1.1
date: 1998/09/28 17:54:04; author: teubner; state: Exp;
branches:
Initial revision
-----
revision 1.0
date: 1998/09/28 17:54:04; author: teubner; state: Exp; lines: +0
This is the first revision
-----
>
```

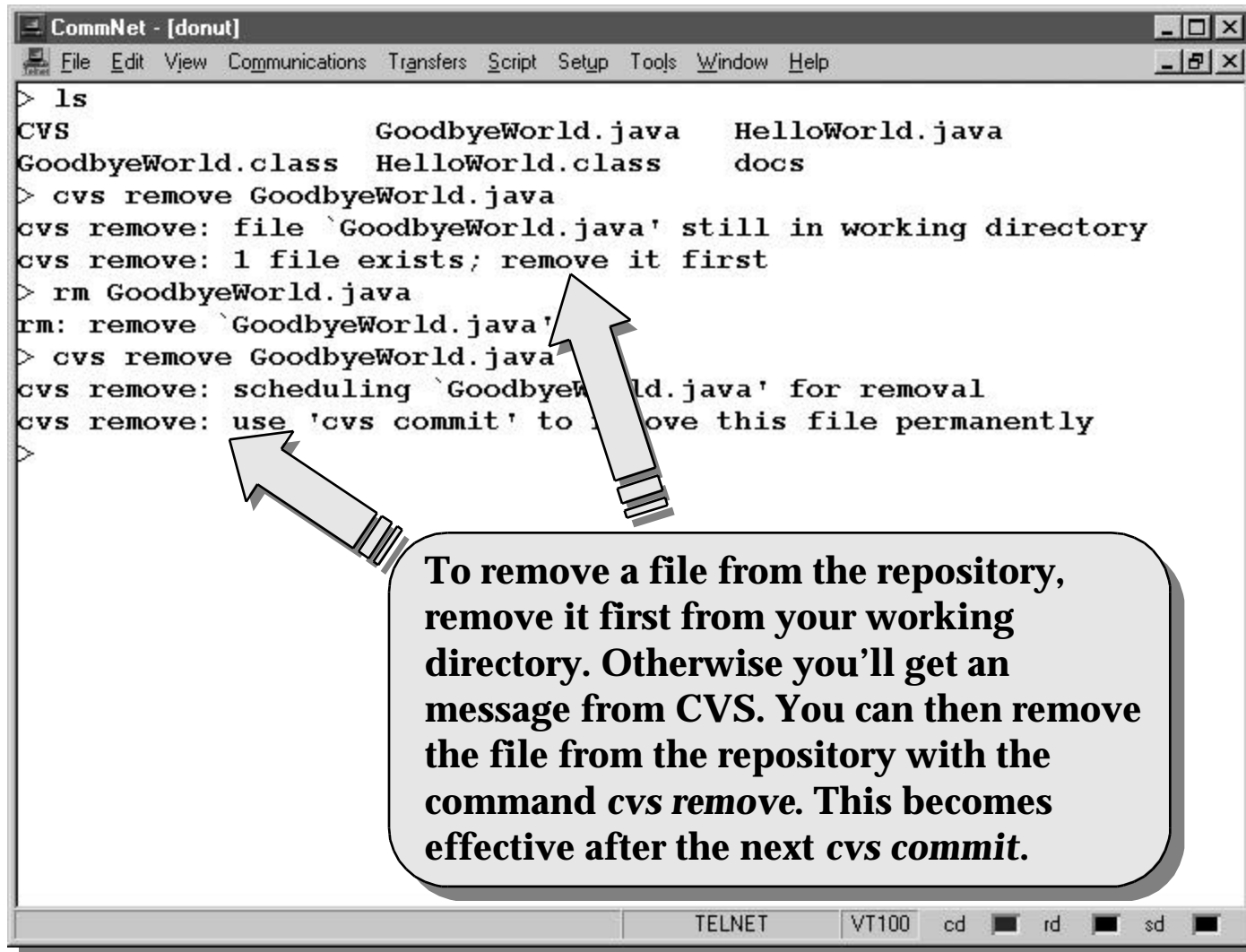
The `cvs log` command displays information about the revisions of a file including the name of the author, date and time when he committed a particular revision and the comment he specified.

TELNET VT100 cd rd sd

Deleting a file from the repository

- ❖ **Situation:** One or more files are not longer being used and have to be removed from the repository.
- ❖ **Command:** `cvs remove filename`
- ❖ **You still have to do a `cvs commit` after this command to make the remove(s) actually take affect.**
- ❖ **Note:** `cvs remove` does not actually remove the files from the repository. It only moves them from the "current list" to the CVS Attic. When another person checks out the module in the future they will not get the files that were removed. But if you ask for older versions, the file will be checked out of the Attic.

Deleting a file (example)



```
CommNet - [donut]
File Edit View Communications Transfers Script Setup Tools Window Help
> ls
CVS                               GoodbyeWorld.java    HelloWorld.java
GoodbyeWorld.class  HelloWorld.class    docs
> cvs remove GoodbyeWorld.java
cvs remove: file `GoodbyeWorld.java' still in working directory
cvs remove: 1 file exists; remove it first
> rm GoodbyeWorld.java
rm: remove `GoodbyeWorld.java'
> cvs remove GoodbyeWorld.java
cvs remove: scheduling `GoodbyeWorld.java' for removal
cvs remove: use 'cvs commit' to remove this file permanently
>
```

To remove a file from the repository, remove it first from your working directory. Otherwise you'll get an message from CVS. You can then remove the file from the repository with the command `cvs remove`. This becomes effective after the next `cvs commit`.

TELNET VT100 cd rd sd

CVS Basic Commands (Summary)

- ❖ Adding a file to the repository
 - ◆ **cv**s add *filename*
- ❖ Getting a module (file tree)
 - ◆ **cv**s checkout *filename*
- ❖ Updating a local file
 - ◆ **cv**s update *filename*
- ❖ Promote all changes done to a file
 - ◆ **cv**s commit *filename*
- ❖ Deleting a file from the repository
 - ◆ **cv**s delete *filename*
- ❖ See version history for a file
 - ◆ **cv**s log *filename*

Security aspects of CVS

- ❖ **Direct access (through the file system) to the repository is restricted by using the access control mechanisms of the Linux file system.**

- ❖ **Web access is restricted by**
 - ◆ **using a password protected website**
 - ◆ **allowing only read-access to the repository through the web**

- ❖ *Important: Every developer has to protect his private working directory by establishing appropriate access rights (e.g. read and write access only for him/her).*

Where to find help on CVS?

- ❖ The direct way
 - ◆ `cvsh`
- ❖ The unix way
 - ◆ `man cvs`
- ❖ The comprehensive way
 - ◆ **Read the documentation for CVS. You'll find a copy on the webpage of the project.**
- ❖ Where to start in the web?
 - ◆ `http://www.cycltic.com`