

# **CSC444F: Software Engineering I**

**Mou Hu**  
**[mou.hu@utoronto.ca](mailto:mou.hu@utoronto.ca)**

# Lecture 14: Implementation (III)

- Debugging
  - Configuration Management
  - CASE Tools
- 
- Reading: Chapter 4, Sections 19.1, 19.2, 19.3

# Debugging Basics

- Debugging follows...
  - ...“successful” testing
    - the input is a list of tests that failed
- The Debugging Process:
  - requires: a symptom of a problem
  - effects: defect corrected; symptom has disappeared
- Difficulties:
  - The symptom might change if another defect is corrected
  - Some symptoms are hard to reproduce
    - especially timing problems
  - The symptom might not be the result of an defect
    - But assume it is until you can prove otherwise

Adapted from [3]

# Debugging Approaches

- Strategies

- Brute force:

- Memory dumps, runtime tracing, print statements
    - You will be swamped with data!!

- Backtracking:

- Only feasible for small programs

- Cause elimination:

- Use deduction to list all possible causes
    - devise tests to eliminate them one by one
    - (try to find the simplest input that shows the symptom)



## The Scientific Method

### 1) Study the available data

- which tests worked?
- which tests did not work?

### 2) Form a hypothesis

- ...that is consistent with (all) the data

### 3) Design an experiment to refute the hypothesis


- the experiment must be repeatable
- don't try to prove your hypothesis, try to disprove it

Adapted from [3]


# Example

```
boolean palindrome (char *s)
/* effects: returns true if s reads the
   same reversed as it does forward */
```


– Test cases:

- s=“able was I ere I saw elba” returns false 
- s=“deed” returns true 


– Hypothesis 1:

- maybe it fails on odd length strings?
- simple refutation case: s=“r” returns true 

– Hypothesis 2:

- maybe it fails on strings with spaces in them?
- simple refutation case: s=“ ” returns true 

– Hypothesis 3:

- maybe it fails on odd length strings longer than 1?
- simple refutation case: s=“ere” returns false 
- The hypothesis was not refuted, but that doesn't mean it is true!

**Note:**  
At each step we  
have more data.  
  
Each hypothesis  
must be consistent  
with *all* the data

Source: Ref. [3]

# Configuration Management Tasks

- **Version control**
  - keeping track of all the different versions of elements of a system
- **Workflow management**
  - supporting team development
  - managing change request
- **Configuration control**
  - The specific version of each components from which a given version of the complete system is built is called the configuration of that version of the system
  - The configuration management tool offers help in assembling an executable version of the system

One writes a ‘program’ that identifies the various components of the required system and their mutual dependencies.

The system in question is then generated by executing this ‘program’.

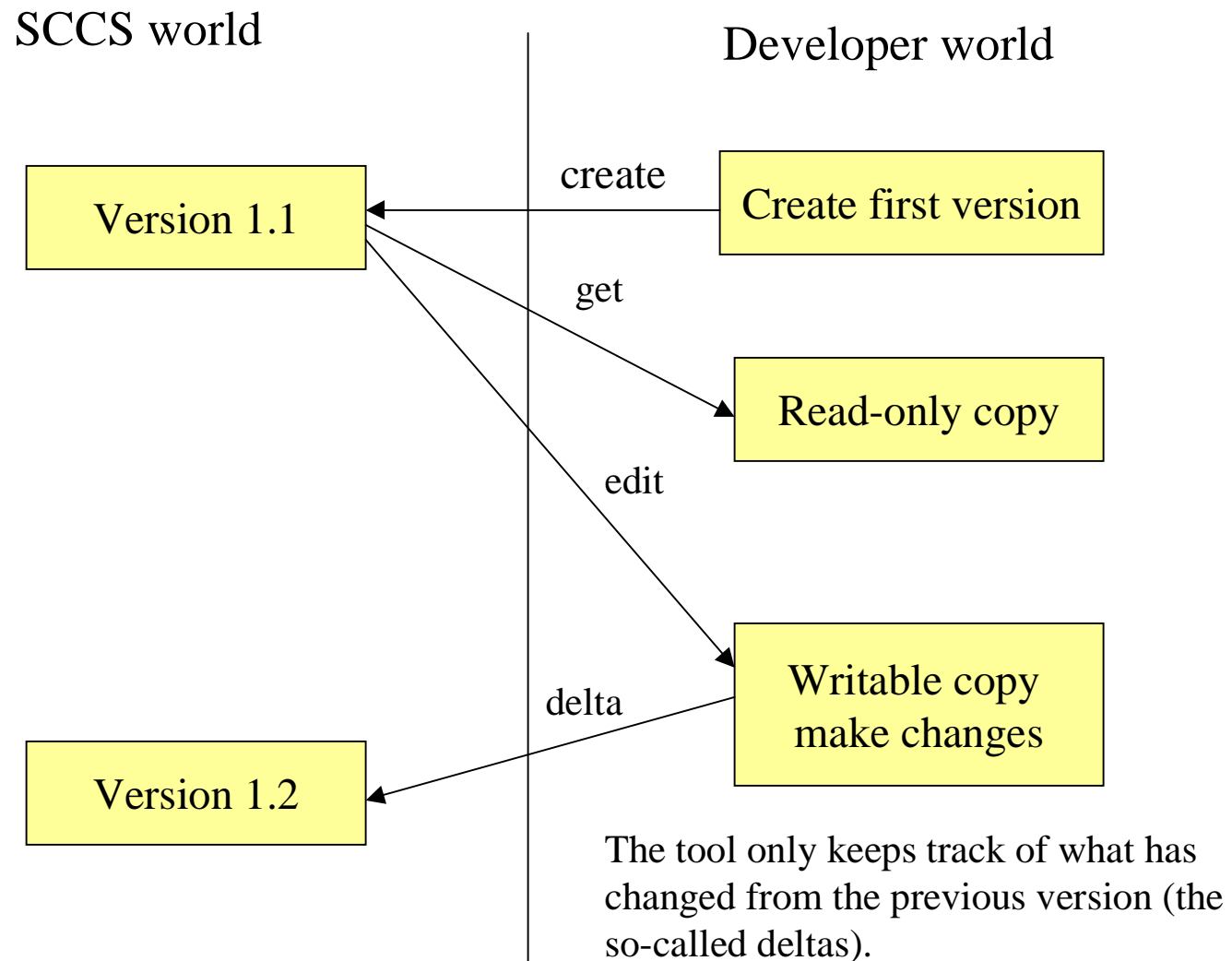
# Configuration Management Basic Concepts

- **Configuration management** is concerned with the management of all artifacts produced in the course of a software development project.
- A **baseline** is a specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.
- Any proposed change to the baseline is called a **change request**.
- A **configuration item** is an aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process.
- Possible configuration items are: source code modules, object code modules, the requirements specification, the design documentation, the test plan, test cases, test results, the user manual.
- Configuration management takes care of these items throughout the software life cycle.

# Configuration Management Tool

- The three major UNIX version-control tools are
  - *sccs* (source code control system)
  - *rcs* (revision control system)
  - *cvs* (concurrent versions system).
- Commercially available configuration management tools:
  - PVCS
  - Microsoft SourceSafe for personal computers
  - ClearCase by Rational

# Version Control Using SCCS

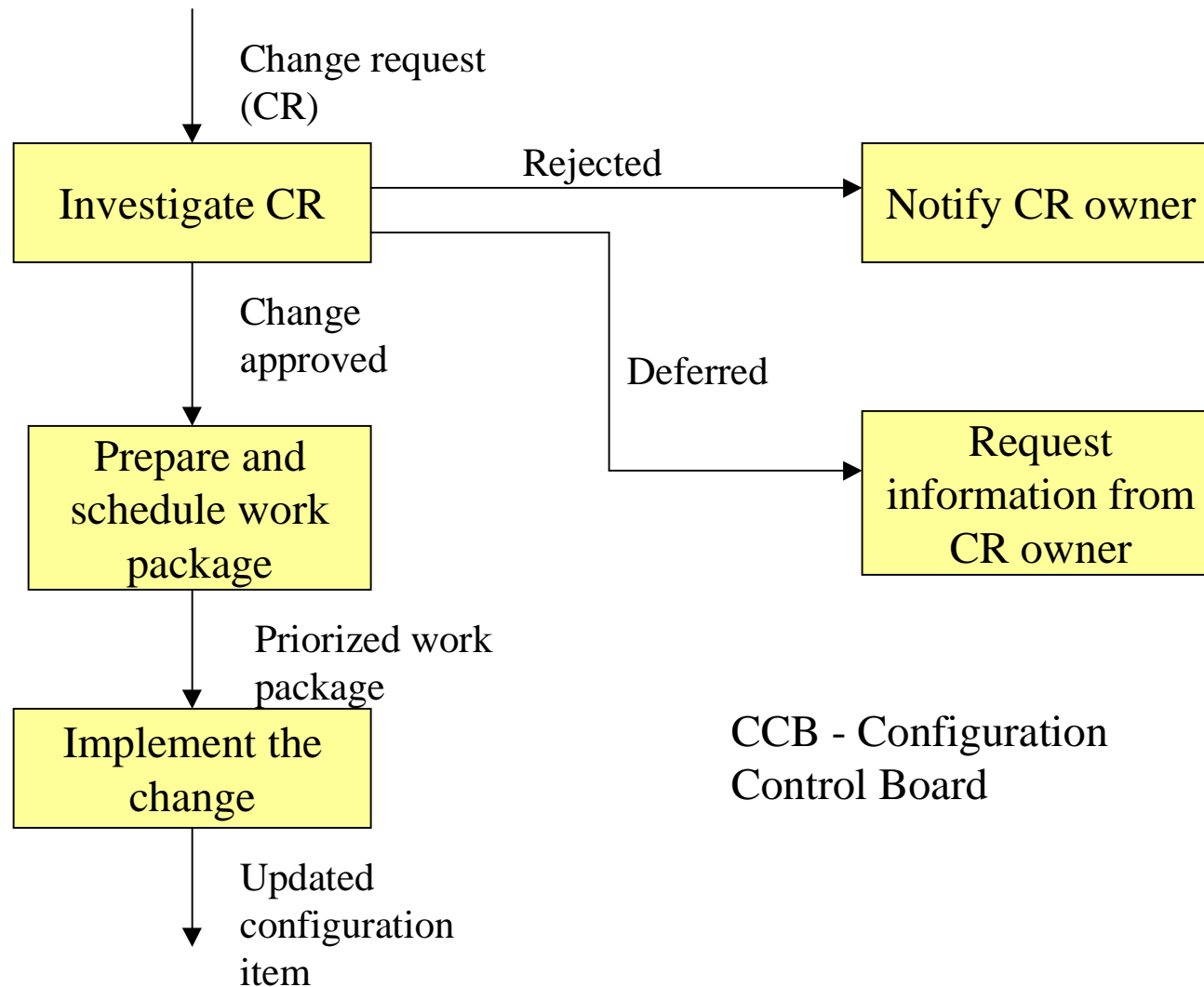


# Workflow Management for Change Request (1/2)

- Keep a record of all changes
  - Comment each procedure / module with
    - author
    - date
    - list of tests performed
    - list of modifications (date, person, reason for change)
- Design a change process
  - Ensure all changes are agreed by the team
  - Use a “request for change” form
  - Distribute the completed forms to clients, subcontractors, etc.
- Don't skip the regression testing!
  - Should run *all* tests again after making any changes.
    - Don't just run the ones that failed...
- Use a configuration management tool
  - ...if you are modifying different parts of the software in parallel

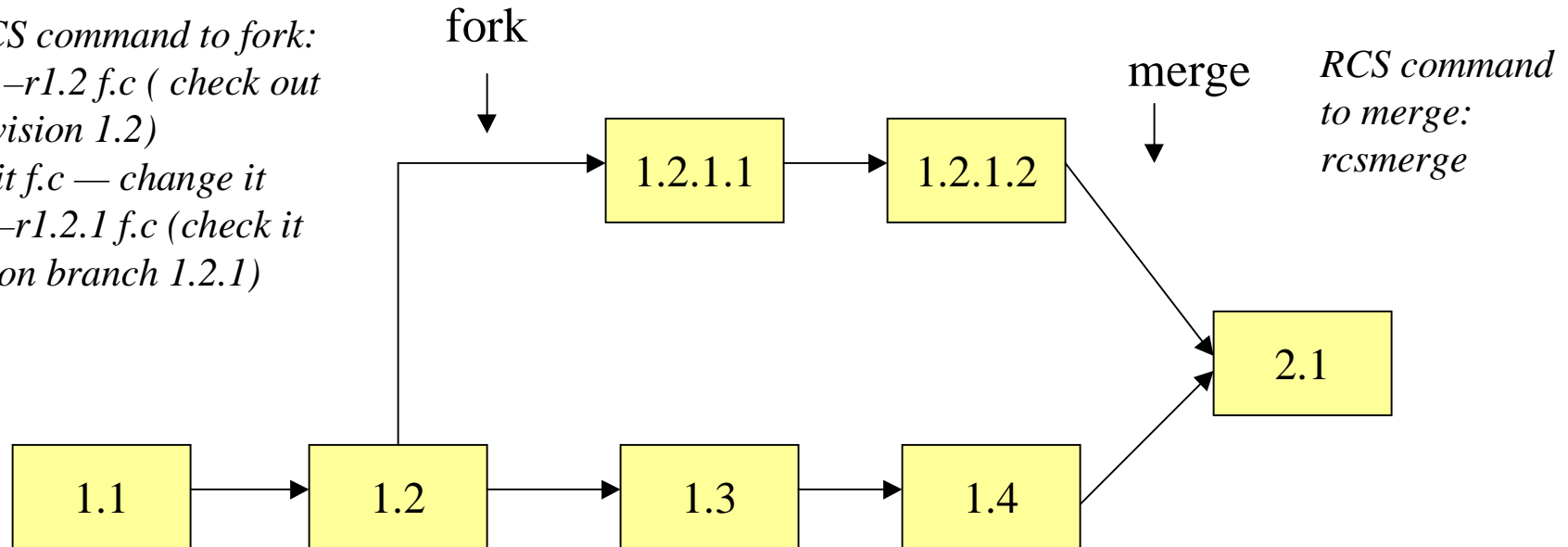
Source: Ref. [3]

# Workflow Management for Change Request (2/2)



# Support Team Development: Forking and Merging

*RCS command to fork:*  
`co -r1.2 f.c` (check out revision 1.2)  
`edit f.c` — change it  
`ci -r1.2.1 f.c` (check it in on branch 1.2.1)

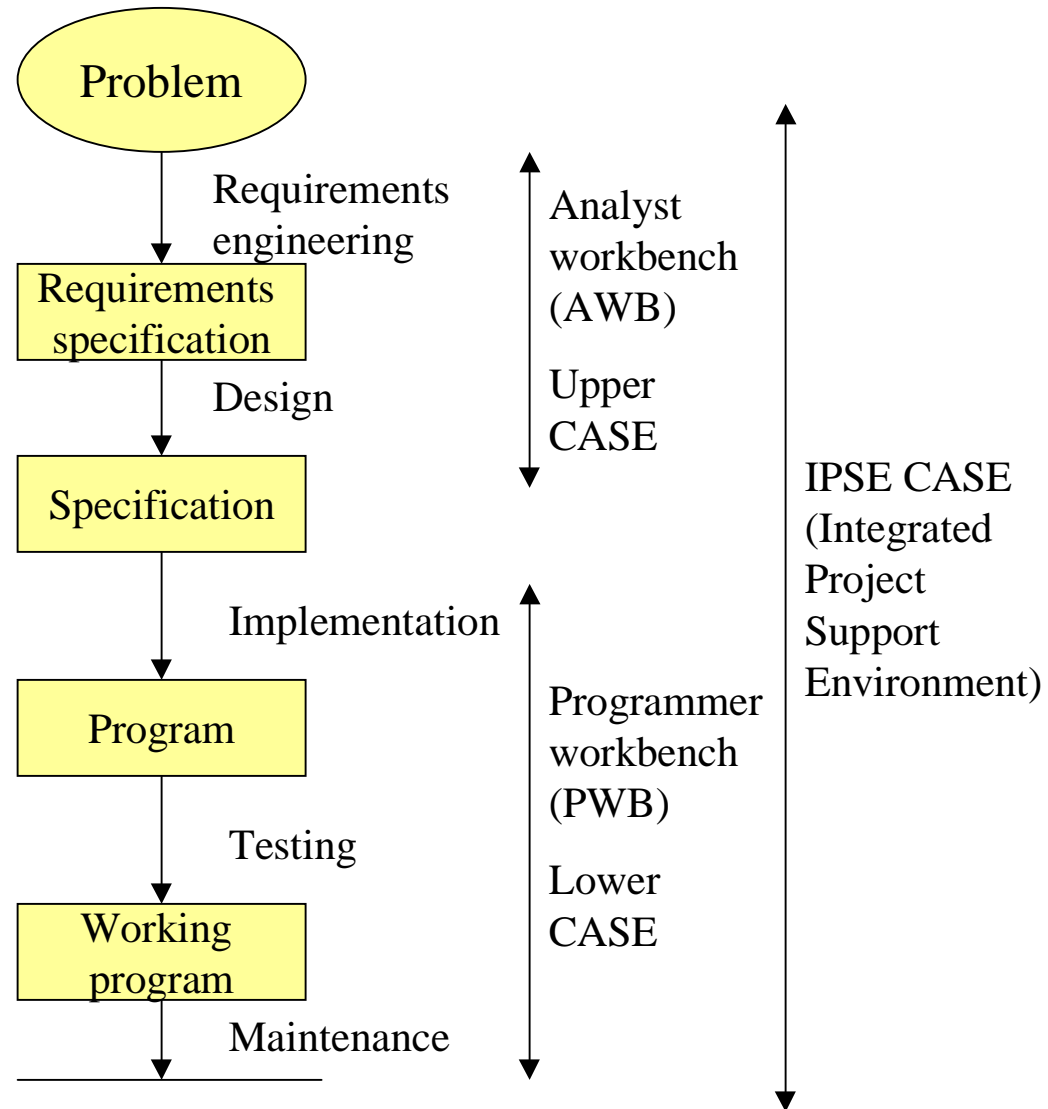


*RCS command to merge:*  
`rcsmerge`

- Reserved checkout (SCCS)
- Unreserved checkout

# Overview of CASE Tools

- CASE (Computer Aided Software Engineering)
- A CASE tool supports a specific task in the software development process.
- A CASE workbench supports a limited set of activities.
- A CASE environment support the entire software process.



# References

- [1] Hans van Vliet, “Software Engineering: Principles and Practice”, John Wiley and Sons, Ltd., 2000.
- [2] Stephen R. Schach, “Object-Oriented and Classical Software Engineering”, McGraw-Hill Companies, Inc., 2002.
- [3] Steve Easterbrook, “Lecture Notes”, University of Toronto, 2001.