Version Control using Subversion

Some slides were taken from:

http://www.stat.washington.edu/albert/presentations/2005-11-02-subversion/

By Albert Young-Sun Kim

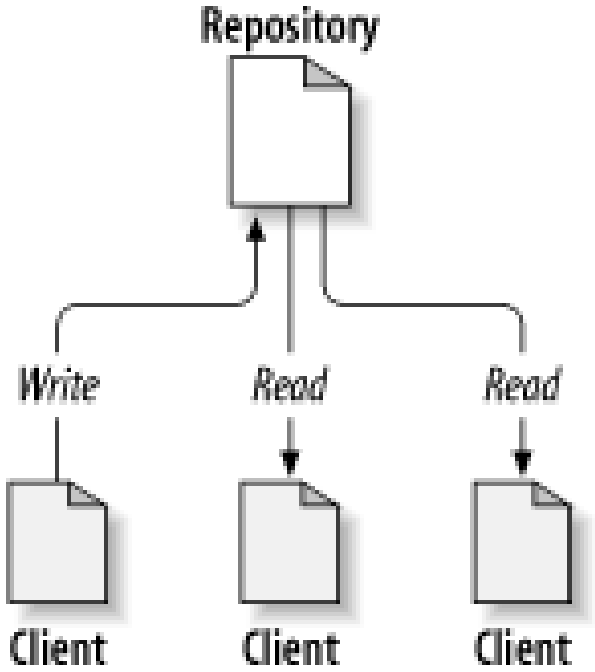
Why use version control?

- Keep past versions of files/directories
- Manage file sharing
  - Specifically: Prevent people from erasing each other's modifications

1. At the Heart of Subversion: *The Repository*

- Typical Client/Server System
- The Repository is a kind of file server.

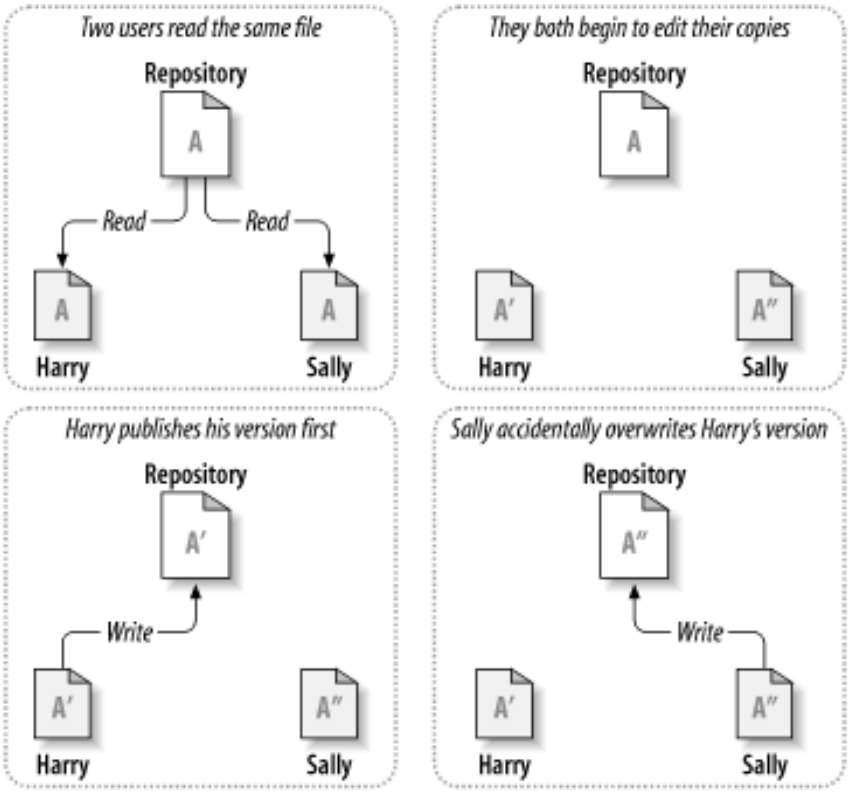
  However, Subversion remembers every change ever written to it!



The Problem of File Sharing

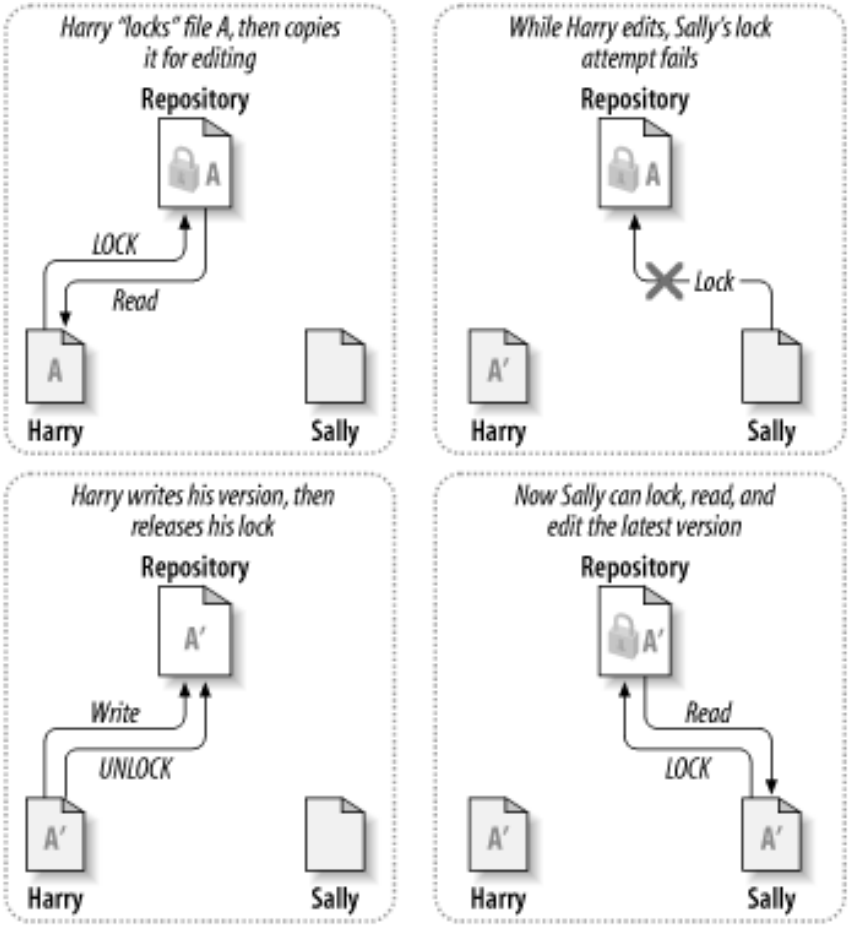
- We want to avoid the following scenario:

  Overwriting each other's modifications



One Solution: Lock-Modify-Unlock

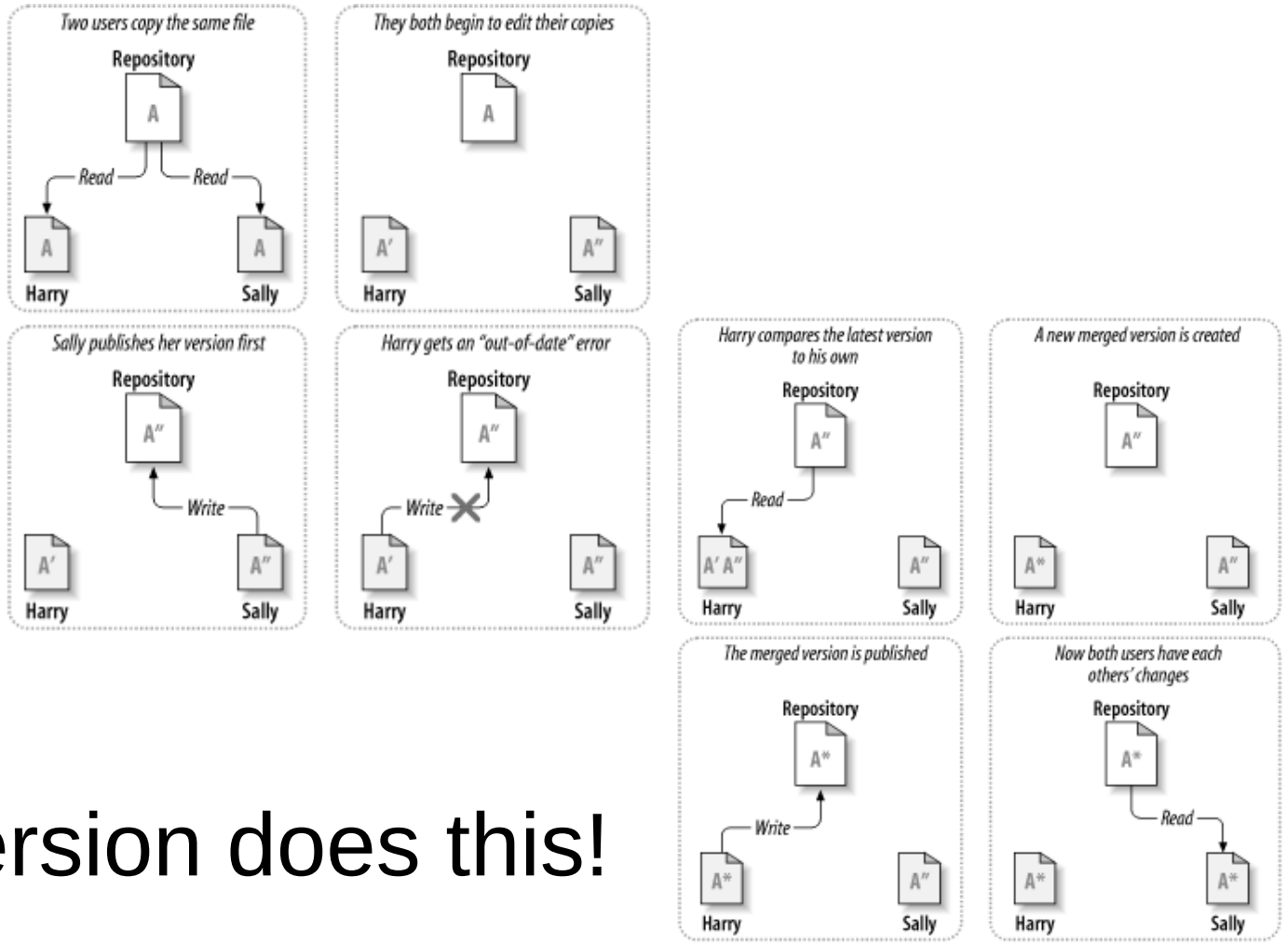
- Only one person may modify a file at any time.
- While this occurs, others can read the file, but not write to it



Problems with Lock-Modify-Unlock

- Can cause unnecessary delays
  - Say Harry forgets to unlock his file before going on vacation
- Even more unfortunate if Harry and Sally's changes don't overlap

Better Solution: Copy-Modify-Merge



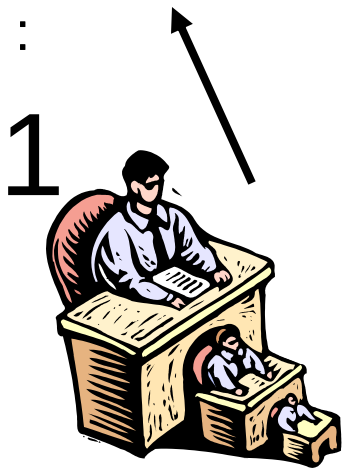
Subversion does this!

Case Study (1)

2 svn update
svn commit

Test.java revision 11

⇒ if (amount > 10) {
Total+=amount;
}
:



1

Harry

- new revision: 12

done

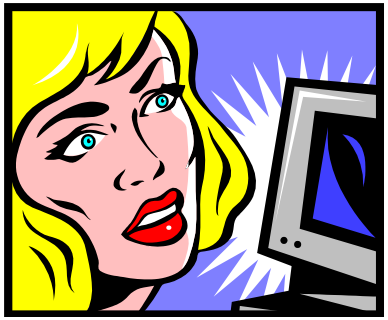
- Merging differences between 11 and 12 into Test.java
- new revision: 13

done

svn update 4
svn commit

Test.java revision 11
:
Total+=amount;
:
⇒ Total++;

3



Sally

Notes on Merge

- When changes don’t overlap, merge is automatic
- When they do overlap, this is called a *conflict*. There are methods to efficiently handle this.
- May seem chaotic, but conflicts are rare and the time it takes to resolve conflicts is far less than the time lost by a locking system.

  (Assuming good communication between users, of course!)

Case Study (2)

2 svn update
svn commit

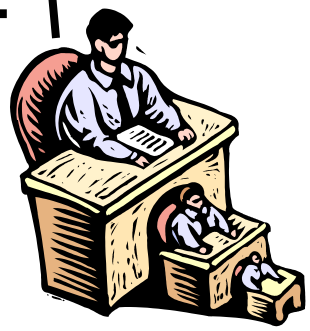
Test.java revision 11

:

Total+=tax;

:

1



Harry

- new revision: 12

done

- Merging differences between 11 and 12 into Test.java

- rcsmerge: warning: conflicts during merge

- cvs update:conflicts found

4

svn update

Test.java revision 11

:

Total+=subtotal;

:

3

Sally

2. Working Copies

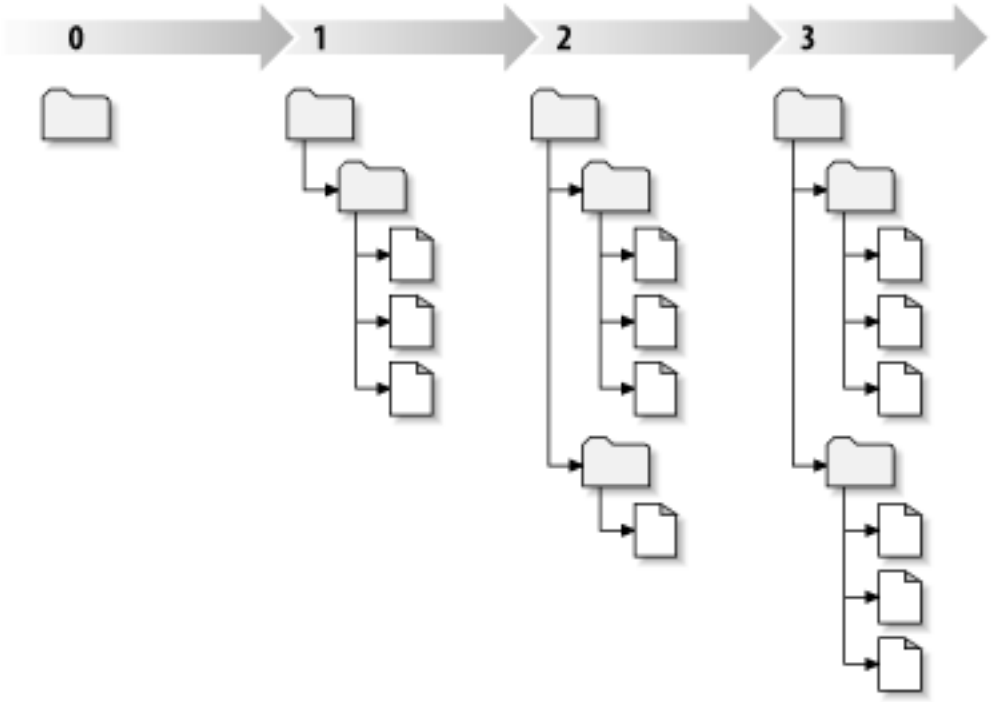
- A Subversion *working copy* is an ordinary directory containing *checked-out* copies of files/directories in the repository

- Your *working copy* is your own private work area:

  Subversion will never incorporate other people's changes, nor make your own changes available to others, until you explicitly tell it to do so

3. Revisions

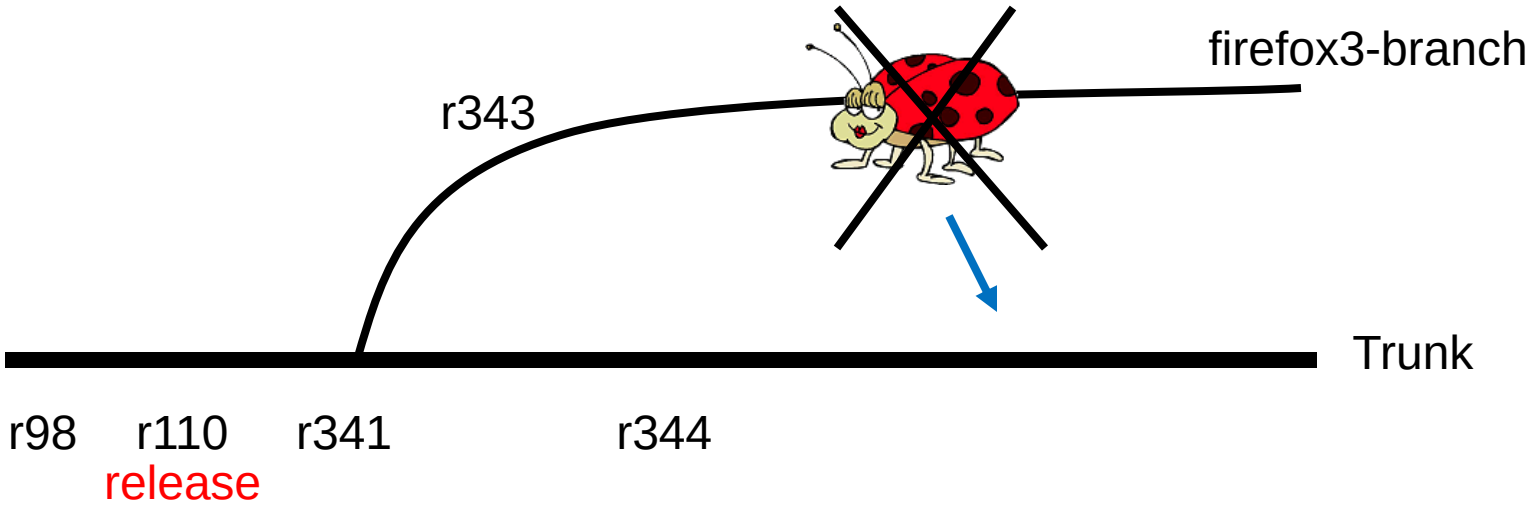
- Each time the repository accepts a commit, this creates a new state of the filesystem tree, called a *revision*.



Each revision is assigned a unique natural number, one greater than the number of the previous revision

Tagging and Branching

- **Tagging**: giving meaningful name to a certain revision
- **Branch** : a fork of the repository (copy)
  - ➢ experimental work, release, or bug fixing



- **Merge**: copying changes between branches

4. Getting Started

1. Create repository
2. Import initial files and directories to repository
3. Initial checkout in order to obtain a working copy
4. Basic Work Cycle

5. Basic Work Cycle:

(Also most common commands)

a) Update your working copy
   - `svn update`

b) Make changes
   - `svn add`
   - `svn delete`
   - `svn copy`
   - `svn move`

c) Examine your changes
   - `svn status`
   - `svn diff`
   - `svn revert`

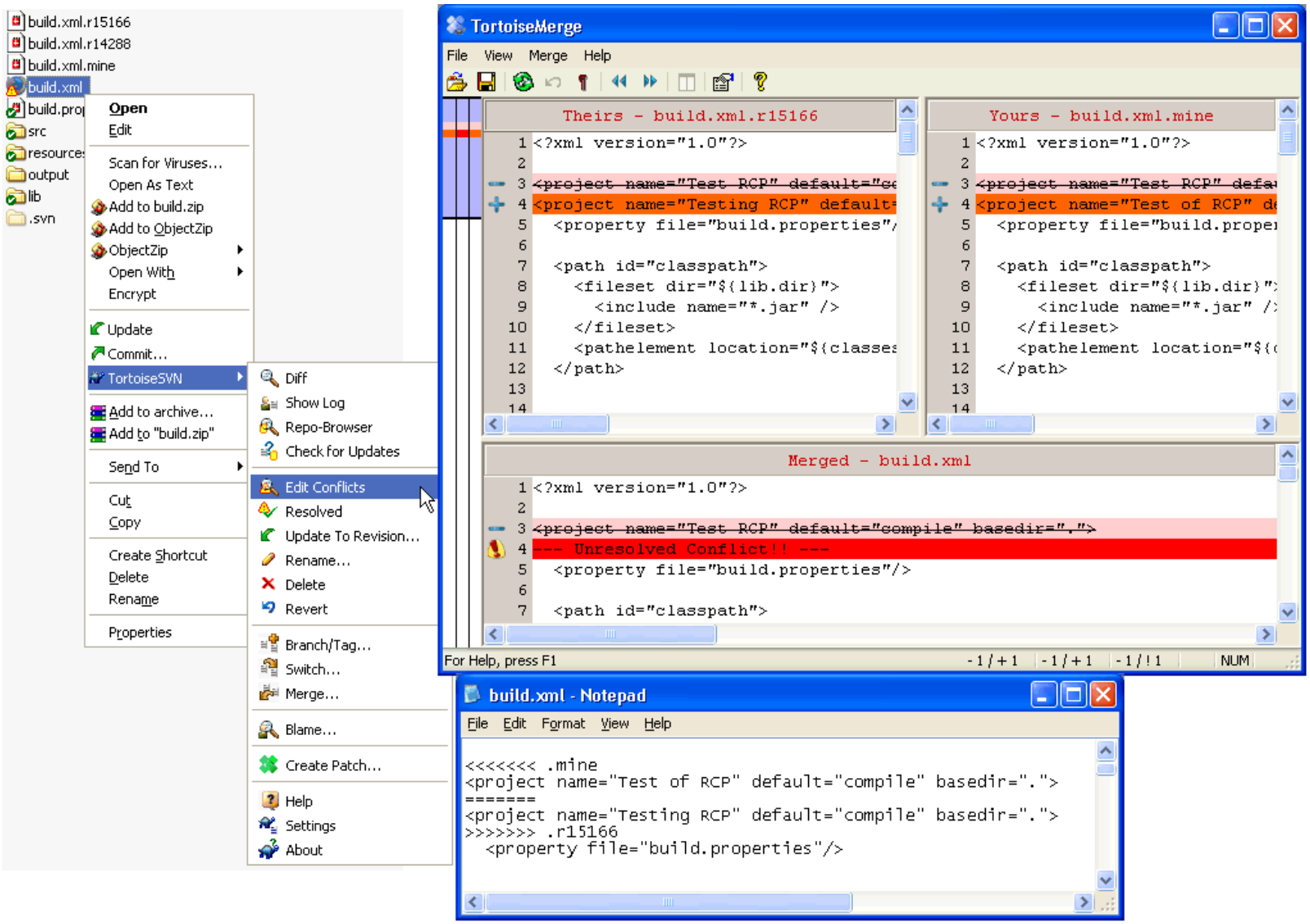
5. Basic Work Cycle

d) Merge other's changes
   - `svn update`
   - `svn resolved`

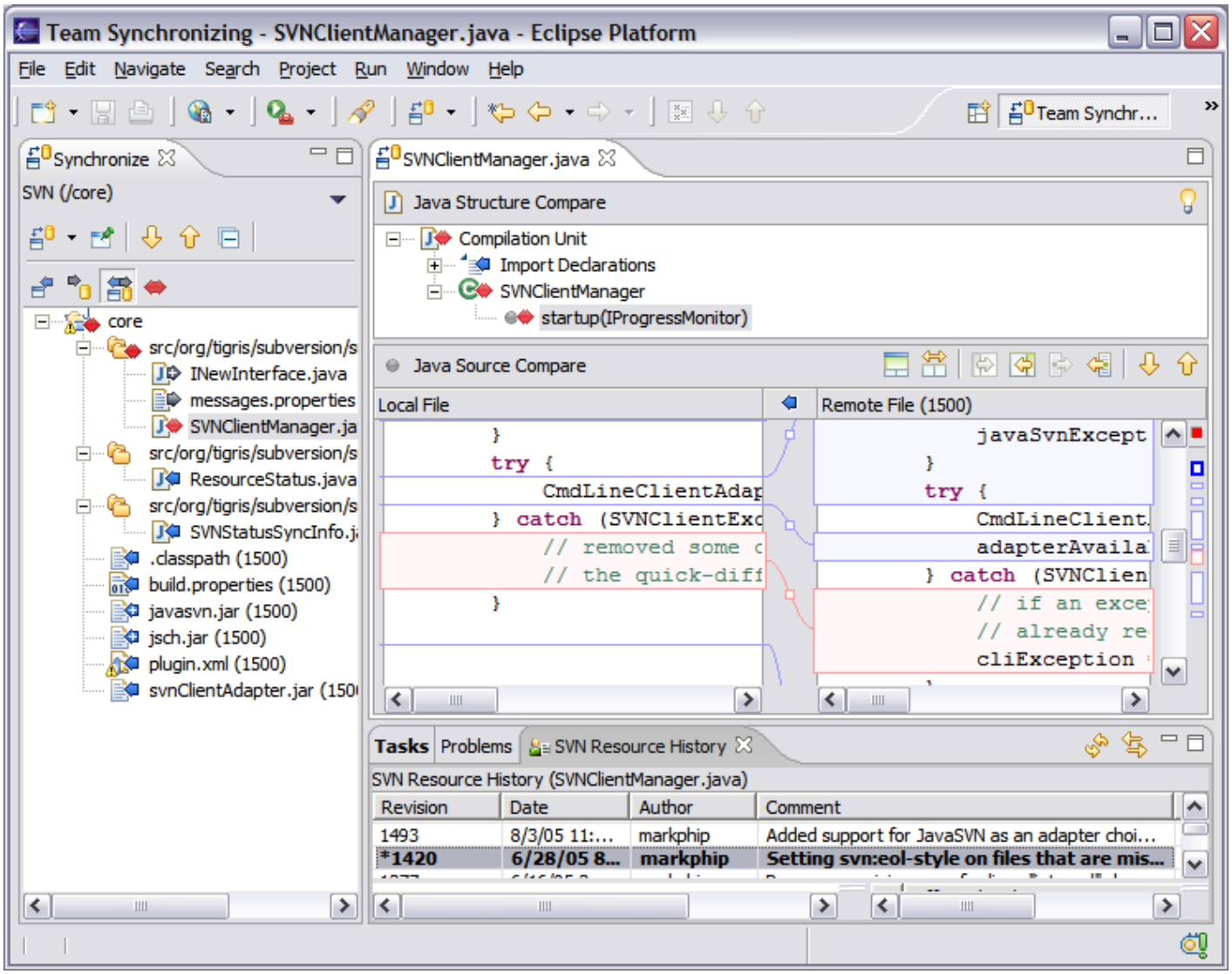
e) Commit your changes
   - `svn commit`

f) (Optional) Examining History
   - `svn log`
   - `svn list`
   - `svn cat`

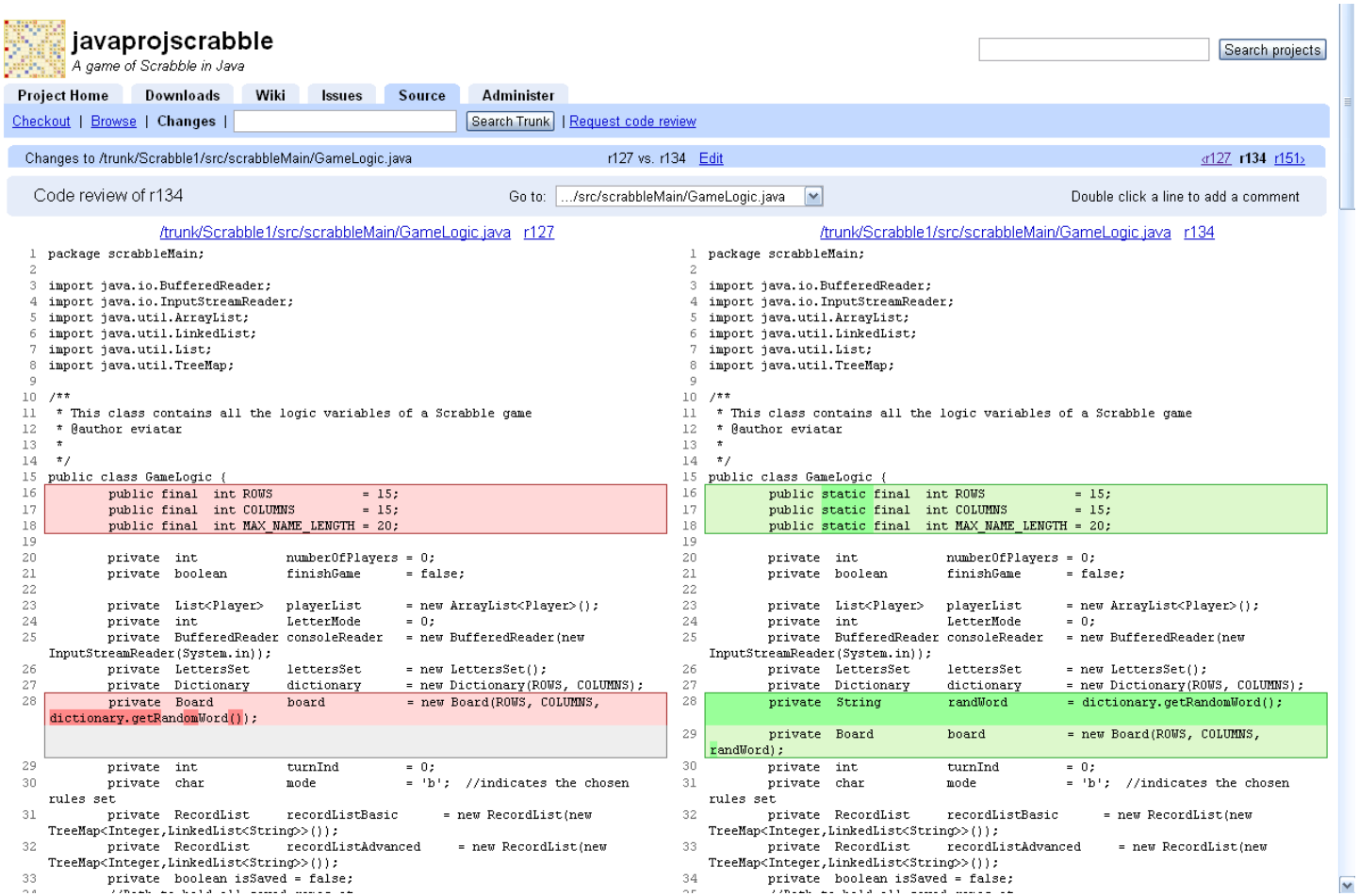
Windows Integration (TortoiseSVN)



Eclipse Integration (subclipse)



SVN on Google Code



javaprojscrabble
A game of Scrabble in Java

Search projects

Project Home | Downloads | Wiki | Issues | Source | Administer

Checkout | Browse | **Changes** | Search Trunk | Request code review

Changes to /trunk/Scrabble1/src/scrabbleMain/GameLogic.java r127 vs. r134 Edit ‹r127 **r134** r151›

Code review of r134 Go to: .../src/scrabbleMain/GameLogic.java Double click a line to add a comment

/trunk/Scrabble1/src/scrabbleMain/GameLogic.java r127

```
1  package scrabbleMain;
2
3  import java.io.BufferedReader;
4  import java.io.InputStreamReader;
5  import java.util.ArrayList;
6  import java.util.LinkedList;
7  import java.util.List;
8  import java.util.TreeMap;
9
10 /**
11  * This class contains all the logic variables of a Scrabble game
12  * @author eviatar
13  *
14  */
15 public class GameLogic {
16         public final  int ROWS           = 15;
17         public final  int COLUMNS        = 15;
18         public final  int MAX_NAME_LENGTH = 20;
19
20         private  int            numberOfPlayers = 0;
21         private  boolean        finishGame      = false;
22
23         private  List<Player>   playerList      = new ArrayList<Player>();
24         private  int            LetterMode      = 0;
25         private  BufferedReader consoleReader   = new BufferedReader(new
InputStreamReader(System.in));
26         private  LettersSet     lettersSet      = new LettersSet();
27         private  Dictionary     dictionary      = new Dictionary(ROWS, COLUMNS);
28         private  Board          board           = new Board(ROWS, COLUMNS,
dictionary.getRandomWord());
29         private  int            turnInd         = 0;
30         private  char           mode            = 'b';  //indicates the chosen
rules set
31         private  RecordList     recordListBasic      = new RecordList(new
TreeMap<Integer,LinkedList<String>>());
32         private  RecordList     recordListAdvanced     = new RecordList(new
TreeMap<Integer,LinkedList<String>>());
33         private  boolean isSaved = false;
```

/trunk/Scrabble1/src/scrabbleMain/GameLogic.java r134

```
1  package scrabbleMain;
2
3  import java.io.BufferedReader;
4  import java.io.InputStreamReader;
5  import java.util.ArrayList;
6  import java.util.LinkedList;
7  import java.util.List;
8  import java.util.TreeMap;
9
10 /**
11  * This class contains all the logic variables of a Scrabble game
12  * @author eviatar
13  *
14  */
15 public class GameLogic {
16         public static final  int ROWS           = 15;
17         public static final  int COLUMNS        = 15;
18         public static final  int MAX_NAME_LENGTH = 20;
19
20         private  int            numberOfPlayers = 0;
21         private  boolean        finishGame      = false;
22
23         private  List<Player>   playerList      = new ArrayList<Player>();
24         private  int            LetterMode      = 0;
25         private  BufferedReader consoleReader   = new BufferedReader(new
InputStreamReader(System.in));
26         private  LettersSet     lettersSet      = new LettersSet();
27         private  Dictionary     dictionary      = new Dictionary(ROWS, COLUMNS);
28         private  String         randWord        = dictionary.getRandomWord();
29         private  Board          board           = new Board(ROWS, COLUMNS,
randWord);
30         private  int            turnInd         = 0;
31         private  char           mode            = 'b';  //indicates the chosen
rules set
32         private  RecordList     recordListBasic      = new RecordList(new
TreeMap<Integer,LinkedList<String>>());
33         private  RecordList     recordListAdvanced     = new RecordList(new
TreeMap<Integer,LinkedList<String>>());
34         private  boolean isSaved = false;
```

Do and Don’t (CVS)

- Do
  - Enter meaningful comments
  - Check in only when files are stable
  - “cvs update” before “cvs commit”
- Don’t
  - Change files in the ‘CVS’ subdirectory
  - Change or create files in repository directly
  - Change layout of a shared file