VERSION CONTROL

Content from Chapter 6 of "Head First Software Development", Pilone et al.

Miami University Software Technology & Analysis Group (MUSTANG)
Computer Science & Software Engineering
Miami University, Oxford, Ohio, USA

ADMIN

- Customer meetings continuing
  - Don't forget to email agendas and deliverables beforehand
  - Follow deliverables guidelines and rubric
- Iteration #1 Demo
  - March 30th ➔ extended to April 6th
  - Presenting product up to this point (Week 3 and 4 deliverables, week # 5 is not included)
  - Iteration 1 retrospective
  - Burndown and updates
  - Rubric online
- Midcourse Evaluation

MIDCOURSE EVALUATION

- Bare Bones Questioning Technique (Snooks et al., 2004)
  - What (if anything) is interfering with your learning? (**STOP**);
  - What suggestions do you have to improve your learning? (**START**);
  - What is your instructor doing that helps you to learn? (**CONTINUE**).

- The feedback will be used to make changes during the current course.

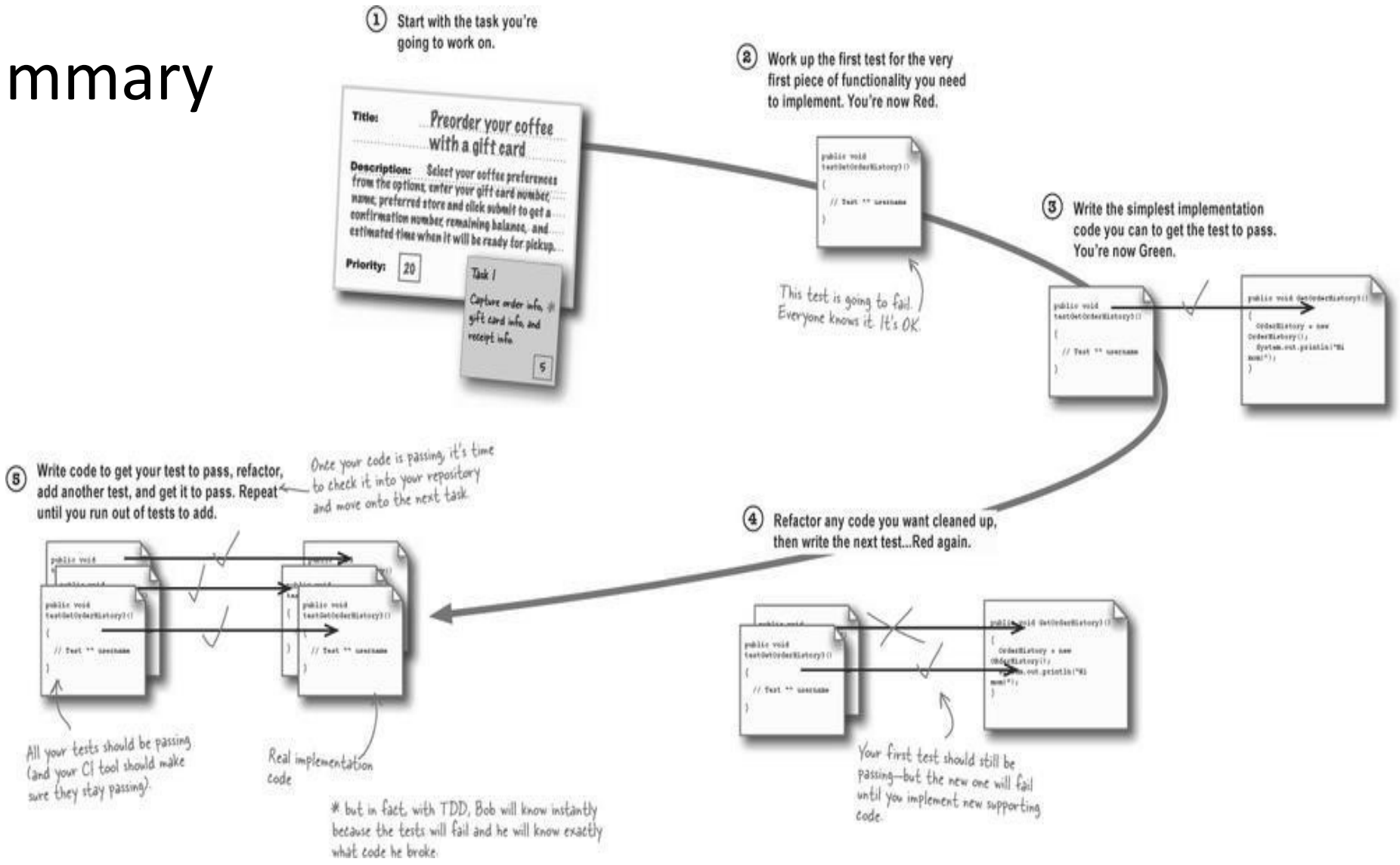
DELIVERABLES FOR 5TH PROJECT WEEK

- Due April 2nd
- Review on Canvas – Week-10 module

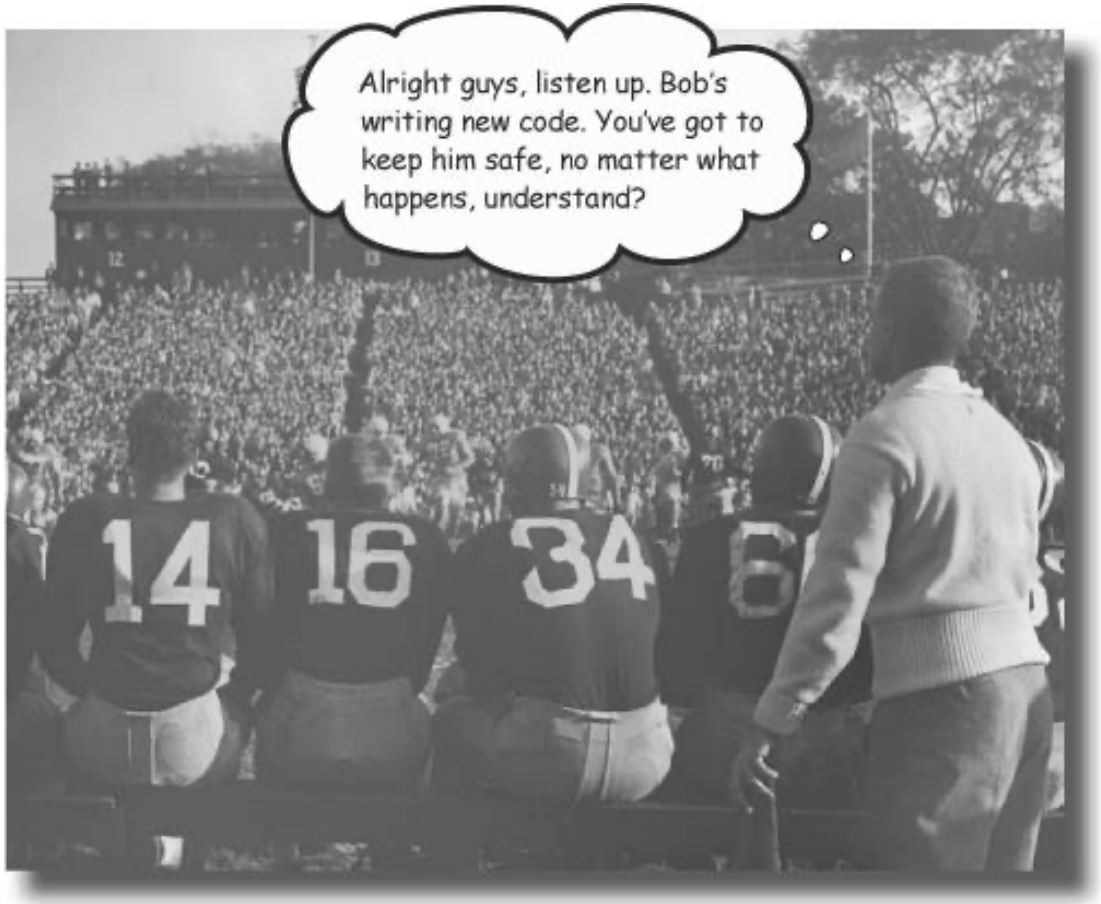
QUESTIONS FROM LAST CLASS?

- TDD
- Red, Green, Refactor
- Benefits of TDD
- Unit testing

TDD Summary


1 Start with the task you're going to work on.
Title: Preorder your coffee with a gift card
Description: Select your coffee preferences from the options, enter your gift card number, name, preferred store and click submit to get a confirmation number, remaining balance, and estimated time when it will be ready for pickup.
Priority: 20
Task 1
Capture order info, gift card info, and receipt info
5
2 Work up the first test for the very first piece of functionality you need to implement. You're now Red.
This test is going to fail. Everyone knows it. It's OK.
3 Write the simplest implementation code you can to get the test to pass. You're now Green.
4 Refactor any code you want cleaned up, then write the next test...Red again.
Your first test should still be passing—but the new one will fail until you implement new supporting code.
5 Write code to get your test to pass, refactor, add another test, and get it to pass. Repeat until you run out of tests to add.
Once your code is passing, it's time to check it into your repository and move onto the next task.
All your tests should be passing (and your CI tool should make sure they stay passing).
Real implementation code
* but in fact, with TDD, Bob will know instantly because the tests will fail and he will know exactly what code he broke.

VERSION CONTROL: DEFENSIVE DEVELOPMENT



Version Control

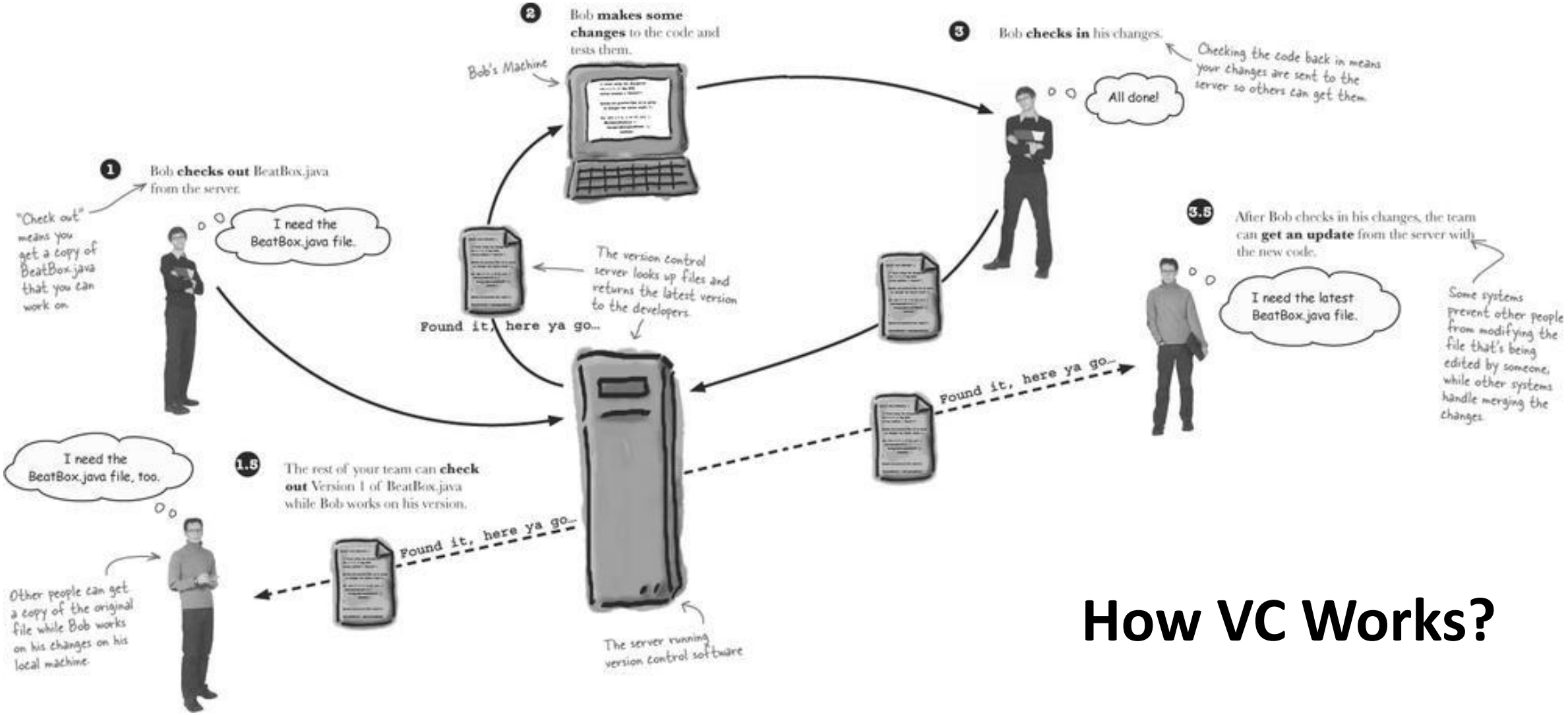
- Always use version control!
  - Code is always backed up
  - Compare versions of your code
    - Helps with debugging
  - Great (essential) way to collaborate with other programmers

SAFETY FIRST

- Writing great software isn't easy
- Need to make sure it keeps working
  - In lieu of typos, bad decisions, hard drive crash, or other
- VC ensures code is always safe in a repository
- Undo mistakes
- Bug fixes

VERSION CONTROL ← → CONFIGURATION MANAGEMENT

- Any serious software project needs VC
  - Also called configuration management (CM)
- VC is a tool
  - Keeps track of changes to your files
  - Helps you coordinate different developers working on different parts of the system at the same time.


2 Bob makes some changes to the code and tests them.
Bob's Machine
3 Bob checks in his changes.
Checking the code back in means your changes are sent to the server so others can get them.
All done!
1 Bob checks out BeatBox.java from the server.
"Check out" means you get a copy of BeatBox.java that you can work on.
I need the BeatBox.java file.
The version control server looks up files and returns the latest version to the developers.
Found it, here ya go...
3.5 After Bob checks in his changes, the team can get an update from the server with the new code.
I need the latest BeatBox.java file.
Some systems prevent other people from modifying the file that's being edited by someone, while other systems handle merging the changes.
Found it, here ya go...
I need the BeatBox.java file, too.
1.5 The rest of your team can check out Version 1 of BeatBox.java while Bob works on his version.
Found it, here ya go...
Other people can get a copy of the original file while Bob works on his changes on his local machine.
The server running version control software

How VC Works?

Many Version Control Systems (VCS) Exist

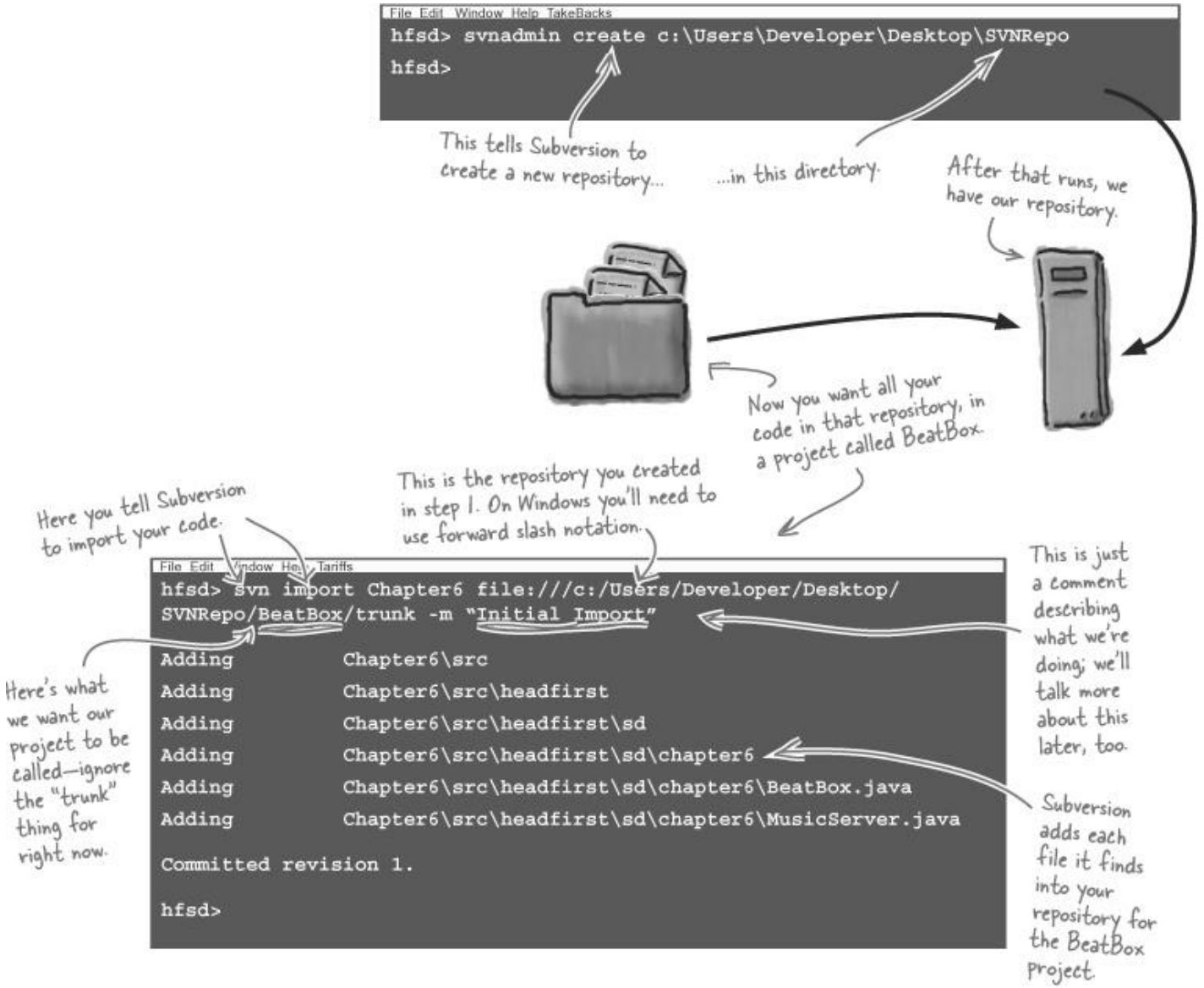
- GitLab (today)
- Subversion (book, Ch6)
- Mercurial
- CVS
- Others

DIFFERENCES

- Git and Mercurial are both “distributed revision control systems” (DRCS)
  - Distributed version control
  - More like peer-to-peer version control
  - Every copy is the complete repo (w/ history)
- Subversion and CVS are more traditional
  - Centralized version control
  - You can get the full Subversion documentation here: http://svnbook.red-bean.com/

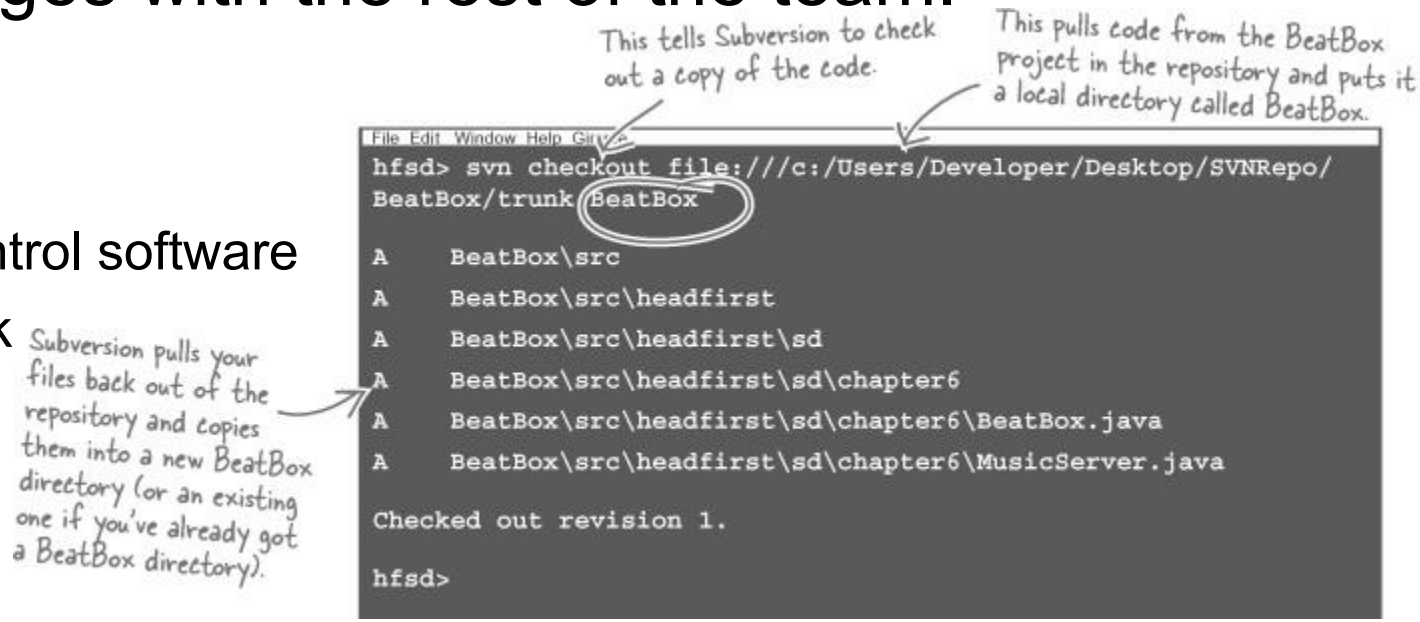
SVN- Don't Need to Learn/Memorize Commands

- First create the repository
  - Only need to do this once for each VC install
- Then, import first version of code into the repository
  - Go to the directory above your code and tell your version control server to import it.


File Edit Window Help TakeBacks
hfsd> svnadmin create c:\Users\Developer\Desktop\SVNRepo
hfsd>
This tells Subversion to create a new repository...
...in this directory.
After that runs, we have our repository.
Now you want all your code in that repository, in a project called BeatBox.
This is the repository you created in step 1. On Windows you'll need to use forward slash notation.
Here you tell Subversion to import your code.
File Edit Window Help Tariffs
hfsd> svn import Chapter6 file:///c:/Users/Developer/Desktop/SVNRepo/BeatBox/trunk -m "Initial Import"
Adding Chapter6\src
Adding Chapter6\src\headfirst
Adding Chapter6\src\headfirst\sd
Adding Chapter6\src\headfirst\sd\chapter6
Adding Chapter6\src\headfirst\sd\chapter6\BeatBox.java
Adding Chapter6\src\headfirst\sd\chapter6\MusicServer.java
Committed revision 1.
hfsd>
This is just a comment describing what we're doing; we'll talk more about this later, too.
Here's what we want our project to be called—ignore the "trunk" thing for right now.
Subversion adds each file it finds into your repository for the BeatBox project.

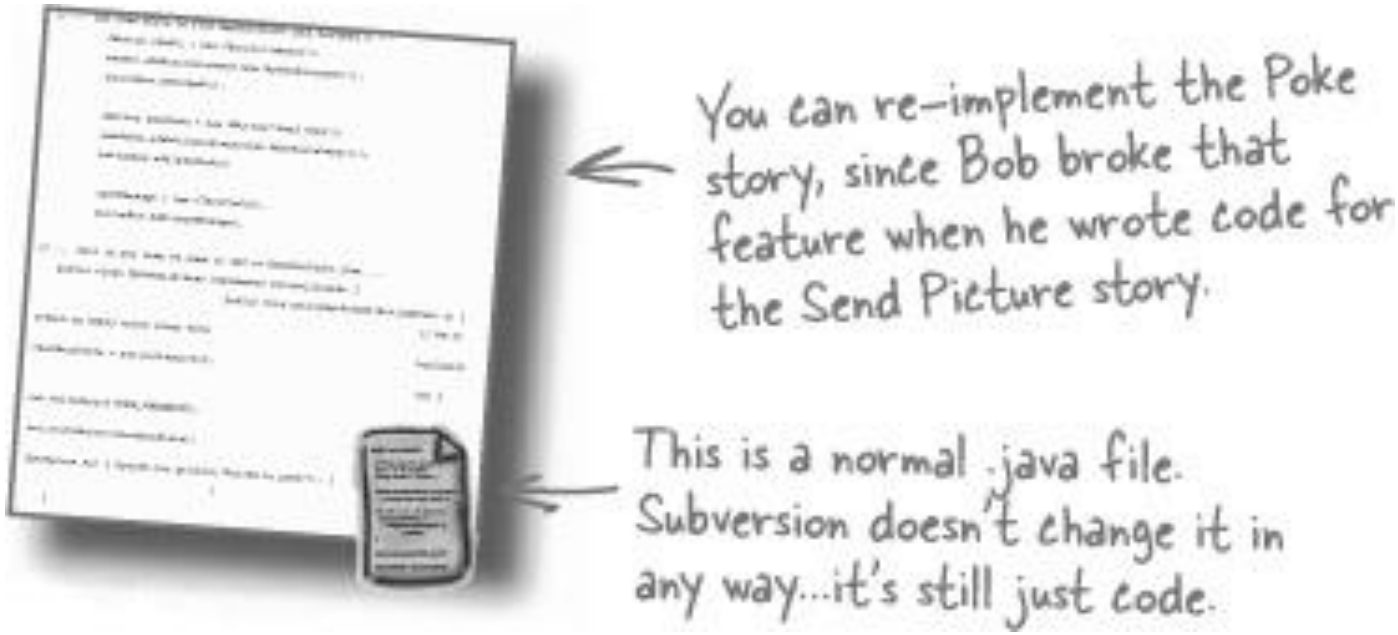
Then Code can be Checked In and Out

- Check it out, make changes, and check it back in.
- VC system (VCS) keeps track of original code, all the changes, and handles sharing changes with the rest of the team.
  - Check out your code
    - You just tell your version control software what project you want to check out, and where to put the files you requested.

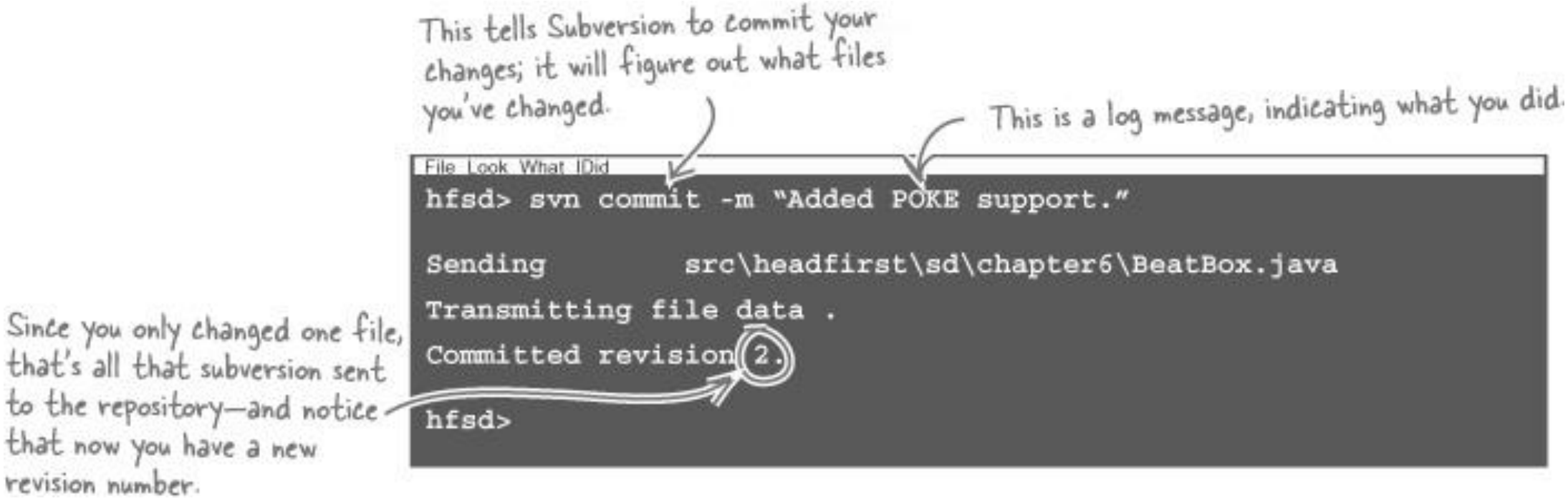


MAKE CHANGES LIKE YOU NORMALLY WOULD

- Work directly on the files you checked out from VCS, compile, and save.

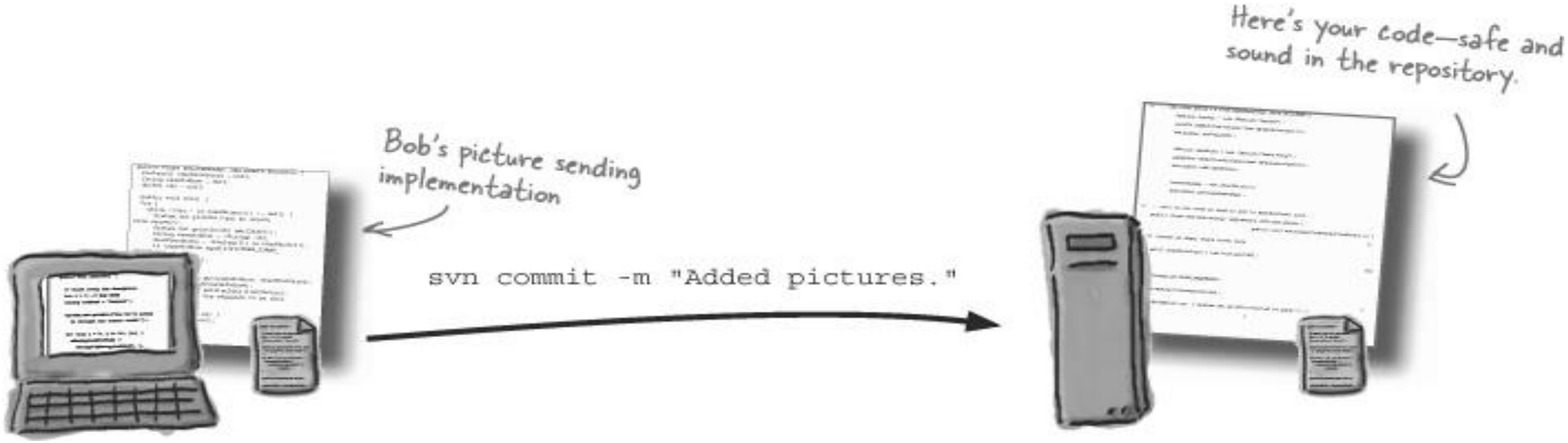


THEN YOU COMMIT YOUR CHANGES BACK TO THE REPOSITORY

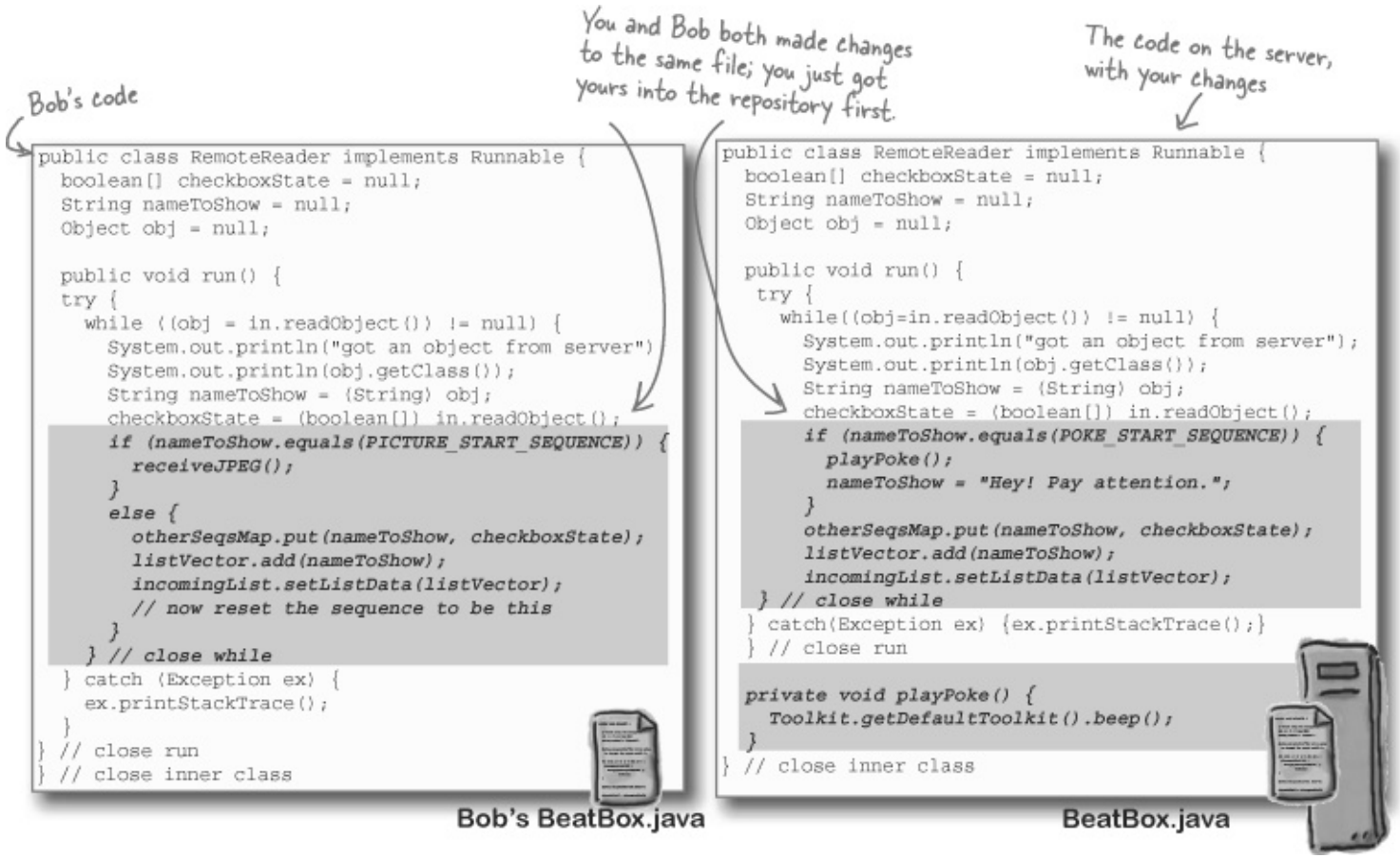


A TOOL SHOULD SOLVE A TOUGH PROBLEM!

- Conflicts & Simultaneous Updates



A TOOL SHOULD SOLVE A TOUGH PROBLEM!




```
public class RemoteReader implements Runnable {
  boolean[] checkboxState = null;
  String nameToShow = null;
  Object obj = null;

  public void run() {
  try {
    while ((obj = in.readObject()) != null) {
      System.out.println("got an object from server")
      System.out.println(obj.getClass());
      String nameToShow = (String) obj;
      checkboxState = (boolean[]) in.readObject();
      if (nameToShow.equals(PICTURE_START_SEQUENCE)) {
        receiveJPEG();
      }
      else {
        otherSeqsMap.put(nameToShow, checkboxState);
        listVector.add(nameToShow);
        incomingList.setListData(listVector);
        // now reset the sequence to be this
      }
    } // close while
  } catch (Exception ex) {
    ex.printStackTrace();
  }
} // close run
} // close inner class
```

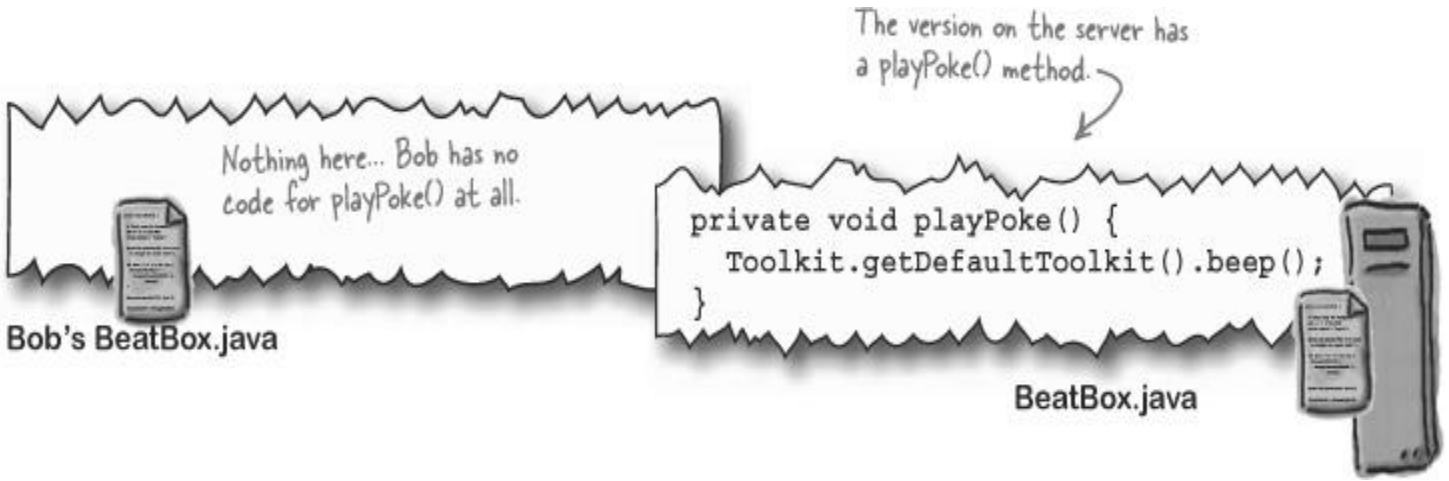
```
public class RemoteReader implements Runnable {
  boolean[] checkboxState = null;
  String nameToShow = null;
  Object obj = null;

  public void run() {
   try {
     while((obj=in.readObject()) != null) {
       System.out.println("got an object from server");
       System.out.println(obj.getClass());
       String nameToShow = (String) obj;
       checkboxState = (boolean[]) in.readObject();
       if (nameToShow.equals(POKE_START_SEQUENCE)) {
         playPoke();
         nameToShow = "Hey! Pay attention.";
       }
       otherSeqsMap.put(nameToShow, checkboxState);
       listVector.add(nameToShow);
       incomingList.setListData(listVector);
   } // close while
  } catch(Exception ex) {ex.printStackTrace();}
  } // close run

  private void playPoke() {
    Toolkit.getDefaultToolkit().beep();
  }
} // close inner class
```

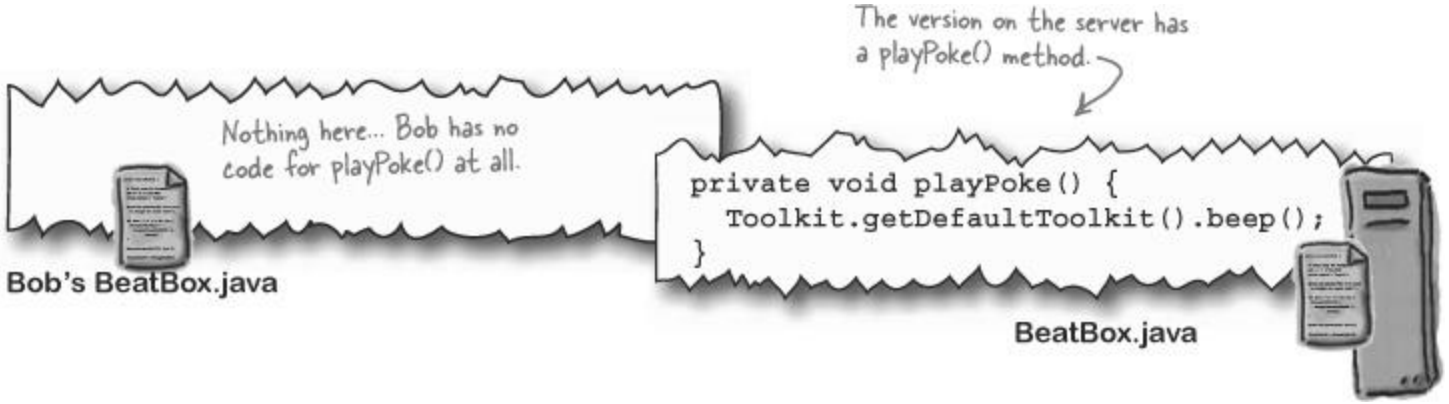
SERVER TRIES TO MERGE YOUR CHANGES

- Most VCS try to merge changes together.
- Isn't always what you want, but most of the time it works great.
- Nonconflicting code and methods are easy.

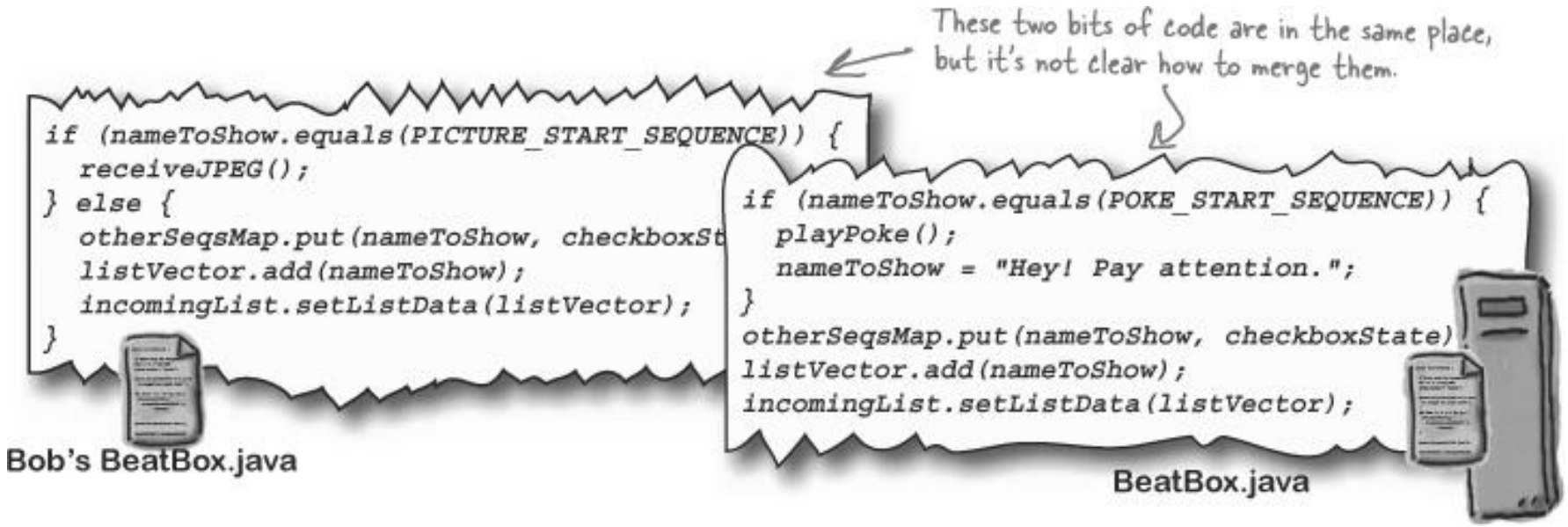


MERGE THROUGH COMBINATION

- VCS can simply combine the two files
- `playPoke()` gets combined with nothing.
  - Think about the Math/set theory.

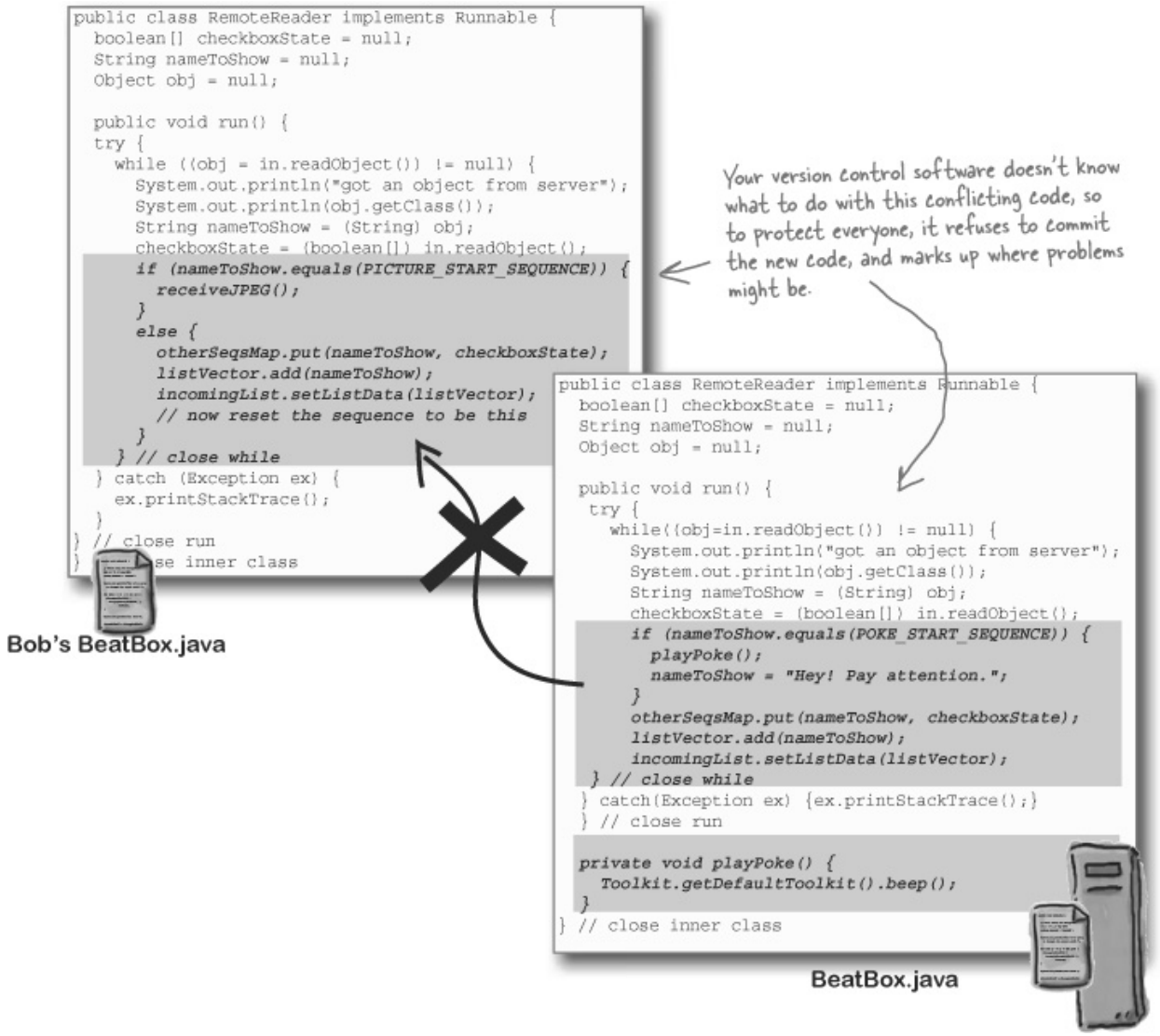


CONFLICTING CODE IS A PROBLEM



CAN'T MERGE? ➔ CONFLICT

- Most systems "Punt"
  - Kick the file back to the person trying to commit the code and ask them to sort the problems out.
- Subversion rejects your commit.
  - You can use the `update` command to pull the changes into your code, and Subversion will mark the lines where there are conflicts in your files...
  - After you sort out the conflicts, you can recommit.


public class RemoteReader implements Runnable {
boolean[] checkboxState = null;
String nameToShow = null;
Object obj = null;
public void run() {
try {
while ((obj = in.readObject()) != null) {
System.out.println("got an object from server");
System.out.println(obj.getClass());
String nameToShow = (String) obj;
checkboxState = (boolean[]) in.readObject();
if (nameToShow.equals(PICTURE_START_SEQUENCE)) {
receiveJPEG();
}
else {
otherSeqsMap.put(nameToShow, checkboxState);
listVector.add(nameToShow);
incomingList.setListData(listVector);
// now reset the sequence to be this
}
} // close while
} catch (Exception ex) {
ex.printStackTrace();
}
} // close run
} // close inner class
Bob's BeatBox.java
Your version control software doesn't know what to do with this conflicting code, so to protect everyone, it refuses to commit the new code, and marks up where problems might be.
public class RemoteReader implements Runnable {
boolean[] checkboxState = null;
String nameToShow = null;
Object obj = null;
public void run() {
try {
while((obj=in.readObject()) != null) {
System.out.println("got an object from server");
System.out.println(obj.getClass());
String nameToShow = (String) obj;
checkboxState = (boolean[]) in.readObject();
if (nameToShow.equals(POKE_START_SEQUENCE)) {
playPoke();
nameToShow = "Hey! Pay attention.";
}
otherSeqsMap.put(nameToShow, checkboxState);
listVector.add(nameToShow);
incomingList.setListData(listVector);
} // close while
} catch(Exception ex) {ex.printStackTrace();}
} // close run
private void playPoke() {
Toolkit.getDefaultToolkit().beep();
}
} // close inner class
BeatBox.java

TRUNK (OR HEAD)

- By default, your version control software gives you code from the trunk.
  - Some systems call this the Head
- Checking out code checks out the trunk, which is the latest code by default and has all the latest changes

Remember the trunk thing that keeps coming up? That's the place where all the latest and greatest code is stored.

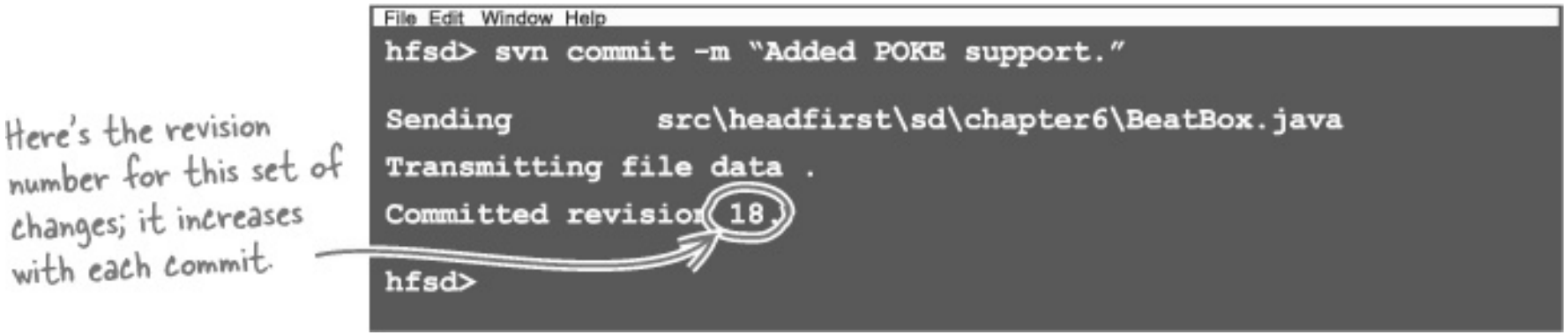
```
File Edit Window Help
hfsd> svn checkout file:///c:/Users/Developer/Desktop/SVNRepo/
BeatBox/trunk BeatBox
A    BeatBox\src
A    BeatBox\src\headfirst
A    BeatBox\src\headfirst\sd
A    BeatBox\src\headfirst\sd\chapter6
A    BeatBox\src\headfirst\sd\chapter6\BeatBox.java
A    BeatBox\src\headfirst\sd\chapter6\MusicServer.java

Checked out revision 1.

hfsd>
```

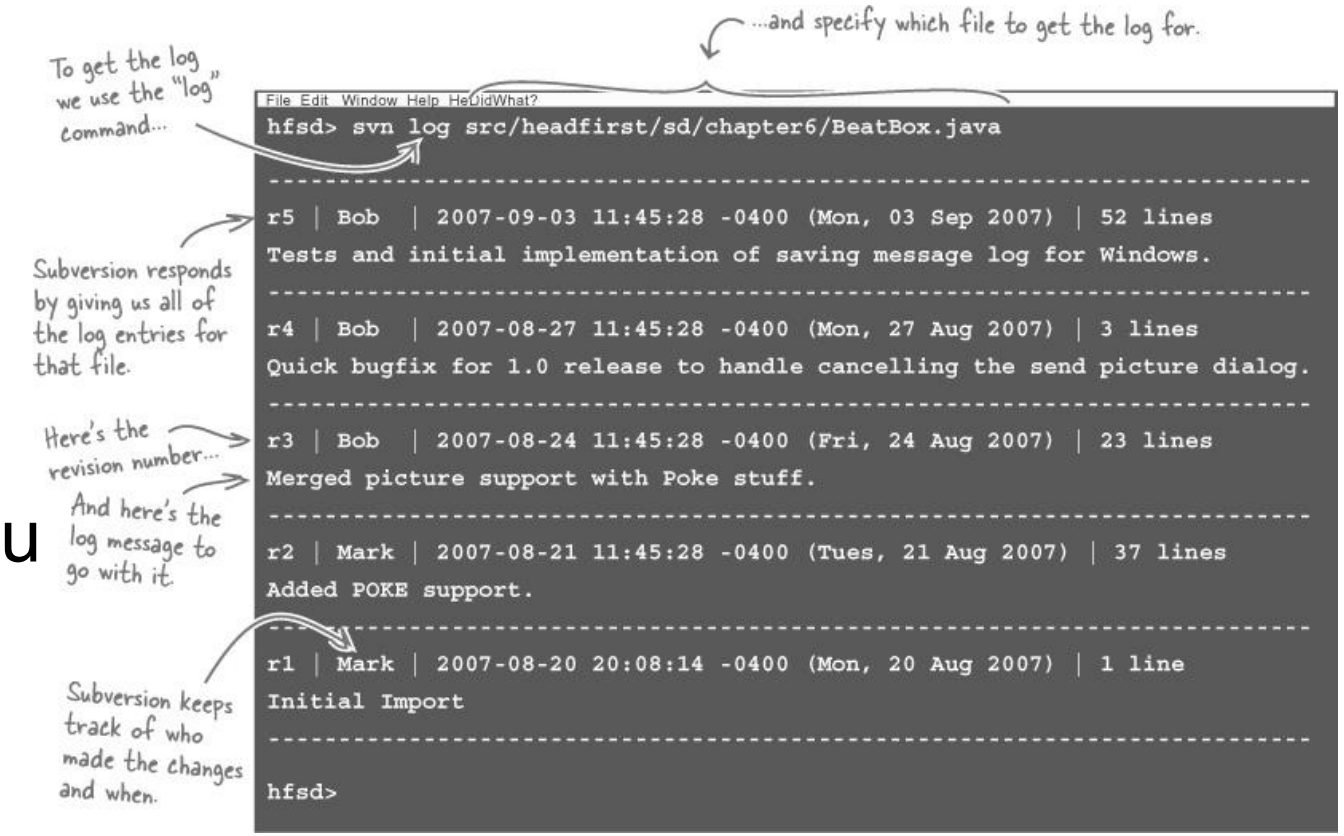
Checking out Previous Version?

- Version control software stores ALL your code.
  - Every time you commit code into your version control system, a revision number was attached to the software at that point.
- So, if you can figure out which revision of your software was released as Version 1.0, you're good to go.

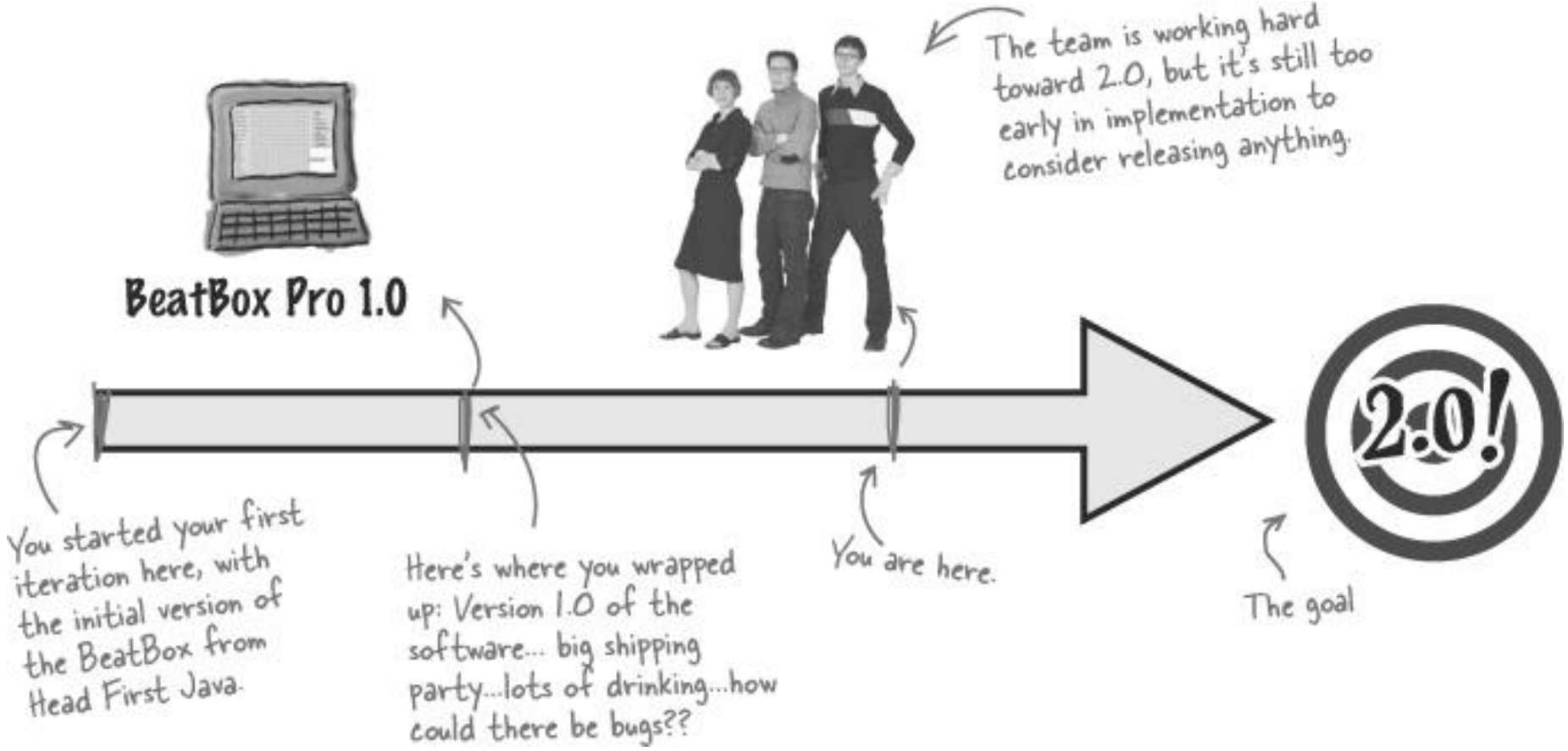


DOCUMENT YOUR CHECKINS & CHANGES!!

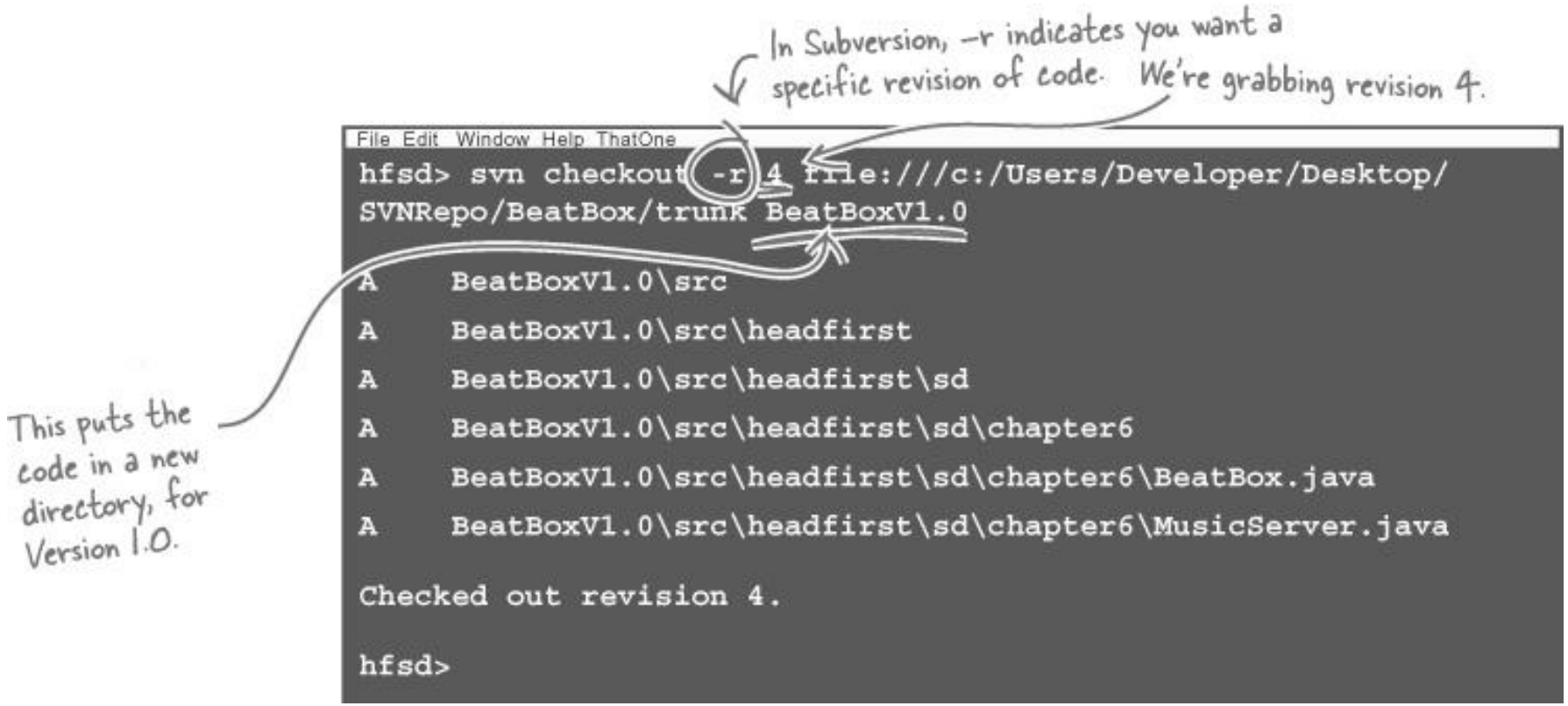
- They matter.
- Just as each commit gets a revision number, your VCS also keeps your commit messages associated with that revision number, and you can view them in the log.



Bugs in Old Versions



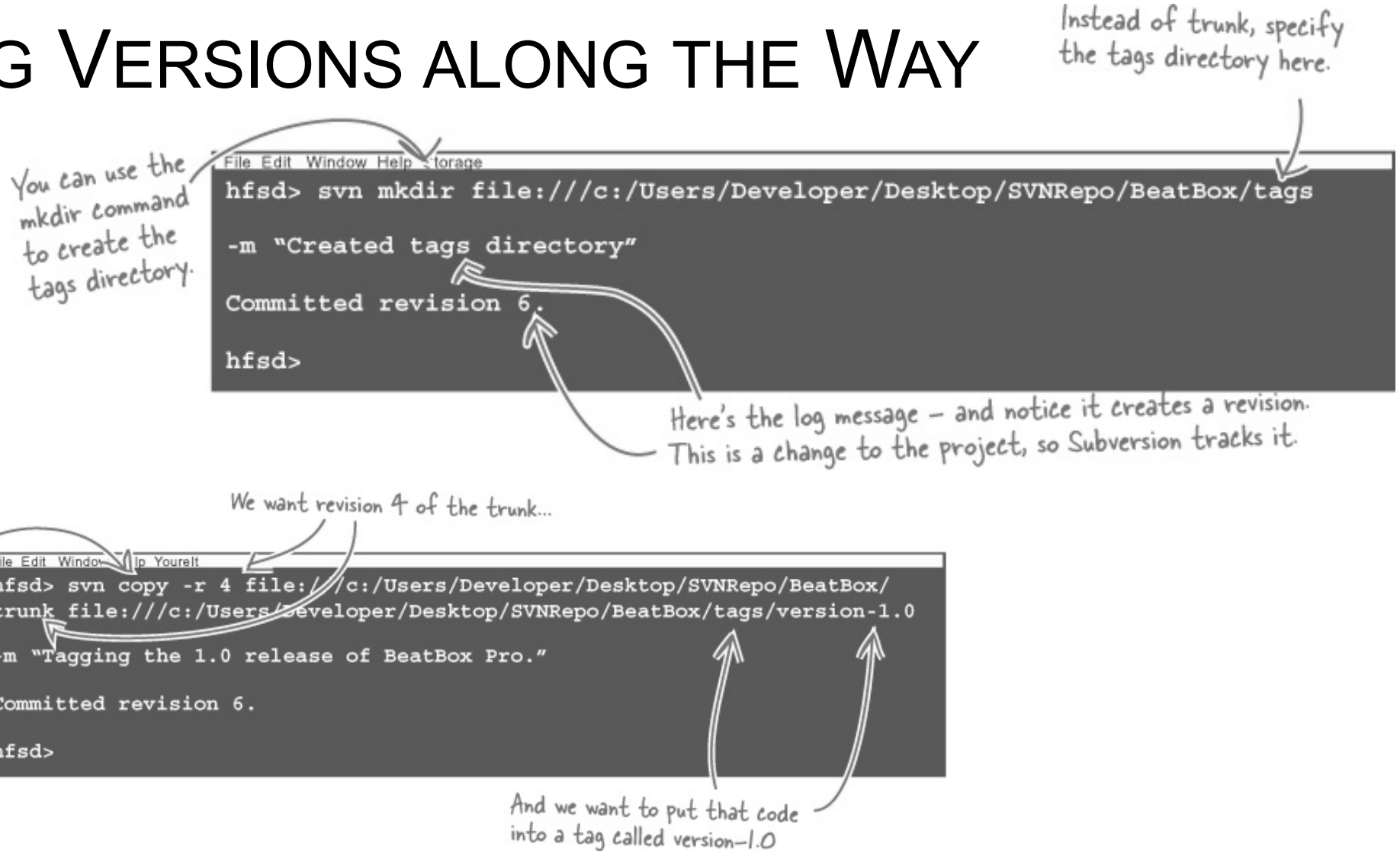
CHECKING OUT OLD VERSION



Fix and Check Back In?

- Nope!

TAG VERSIONS ALONG THE WAY



How Does That Help?

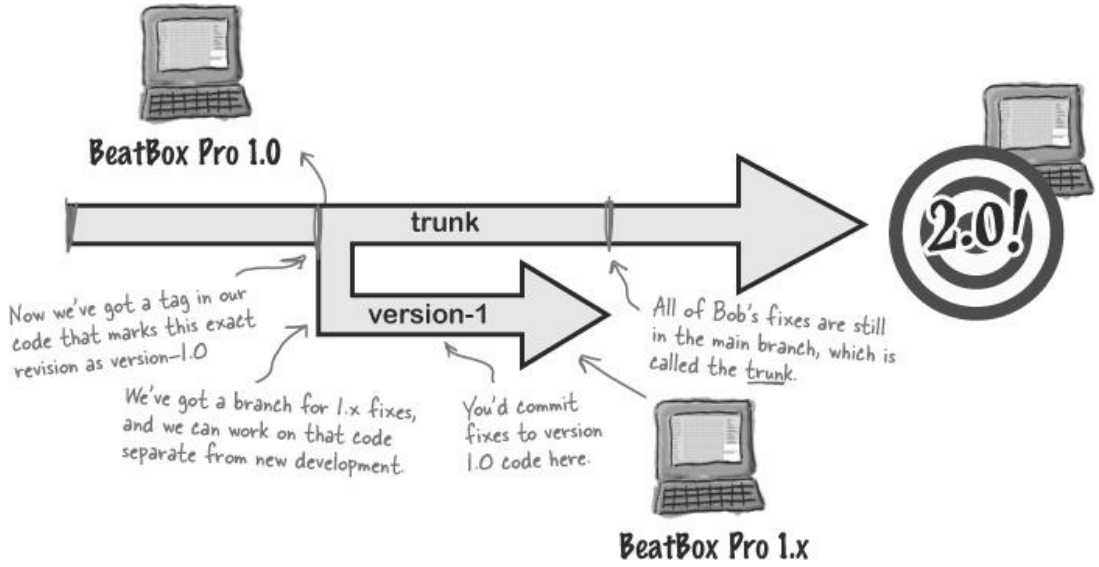
- Instead of needing to know the revision number for version 1.0 and saying `svn checkout -r 4 ...`, you can check out Version 1.0 of the code like this:

```
svn checkout file:///c:/Users/Developer/Desktop/SVNRepo/BeatBox/tags/version-1.0
```

- Do not commit changes into the the Version 1.0 tag
  - Rather, make a copy of revision 4 that we can commit changes to, called a branch
- So a **tag** is a snapshot of your code at a certain time, and a **branch** is a place where you're working on code that isn't in the main development tree of the code.

TAGS, BRANCHES, AND TRUNKS, OH MY!

- Your VCS has a lot going on, but most of the complexity is handled by the server
  - **Tags are snapshots of your code. You should always commit to a branch, and never to a tag.**



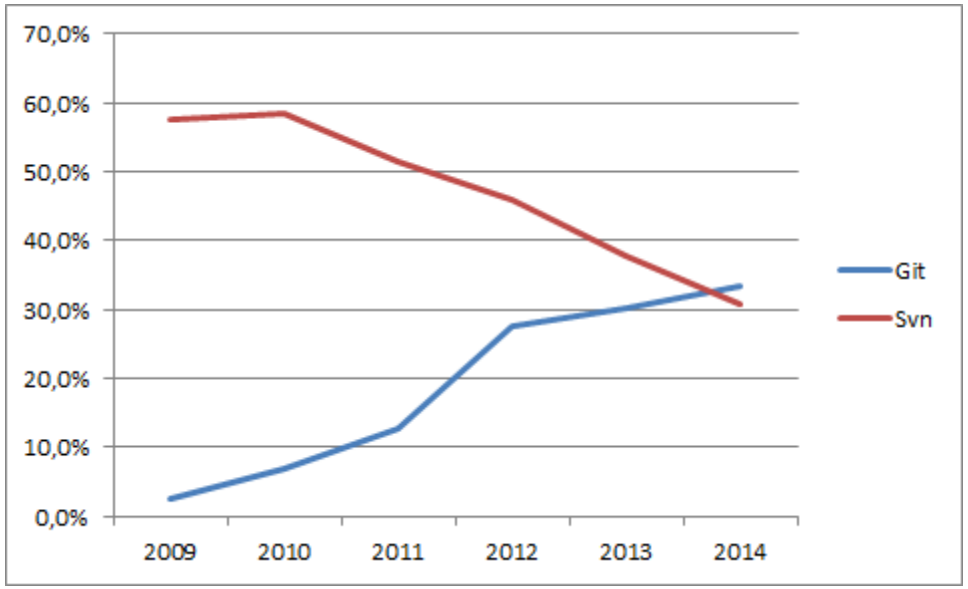
To Branch or Not to Branch?

- Branch when…
  - You have released a **version of the software** that you need to maintain **outside of the main development cycle**.
  - You want to try some **radical changes to code** that you might need to throw away, and you **don't want to impact the rest of the team** while you work on it.
- Do not branch when…
  - You can accomplish your goal by **splitting code into different files** or libraries that can be built as appropriate on different platforms.
  - You have a bunch of developers that can't keep their code compiling in the trunk so you try to **give them their own sandbox** to work in.

GIT

GitHub

- GitHub is a hosting service for git repositories
- Most popular code repository for open source projects
  - Some industry personnel prefer SVN



GIT & GITHUB

- Git is a very popular version control system
- You may use github/lab as your remote repository
- You get free private repositories as a student
  - http://github.com/edu

GITLAB

- Miami gives you access
  - https://gitlab.csi.miamioh.edu/users/sign_in
- Gitlab is more for development teams and associated features

GitLab – Great for Agile

- Source of this material –
  - https://about.gitlab.com/blog/2018/03/05/gitlab-for-agile-software-development/
- Ever wondered if GitLab supports Agile methodology? If you're considering using GitLab it might not be obvious how its features correspond with Agile artifacts, so we've broken it down for you…
  - Do not need to memorize all of this
  - Do need to remember the mappings

Mapping Agile Artifacts to GitLab Features

Agile artifact	GitLab features
User story	Issues
Task	Task lists
Epic	Epics
Points and estimation	Weights
Product backlog	Issues lists & prioritized labels
Sprint/iteration	Milestones
Burndown chart	Burndown charts
Agile board	Issue board

User Stories ➔ GitLab Issues

- In Agile, you often start with a user story that captures a single feature that delivers business value for users.
- In GitLab, a single issue within a project serves this purpose.

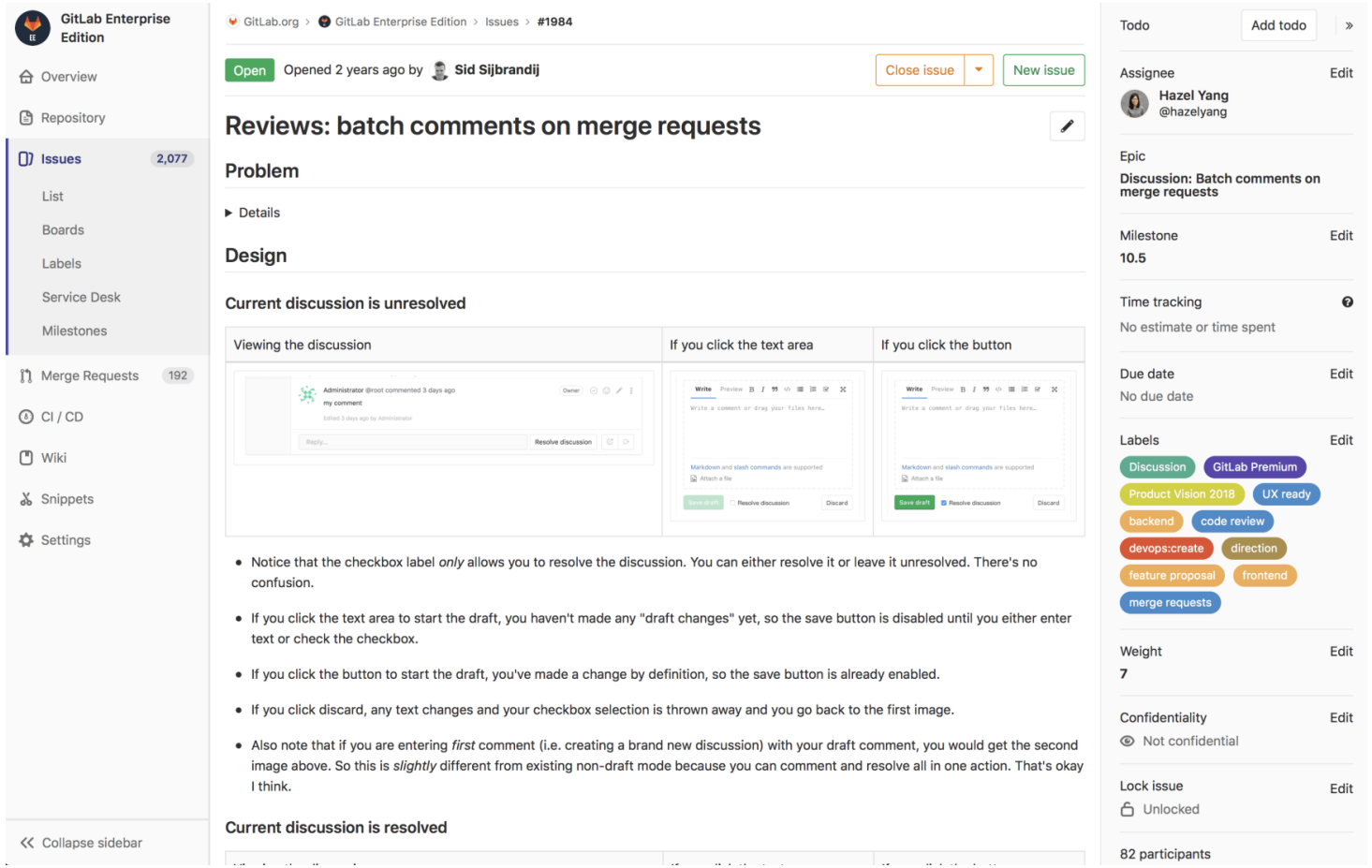
TASK ➔ GITLAB TASK LISTS

- Often, a user story is further separated into individual tasks.
- You can create a task list within an issue's description in GitLab, to further identify those individual tasks.

EPICS ➔ GITLAB EPICS

- Some Agile practitioners specify an **abstraction above user stories**, often called an epic, that indicates a larger user flow consisting of multiple features.
- In GitLab, an epic also contains a title and description, much like an issue, but it allows you to attach multiple child issues to it to indicate that hierarchy.

EPICS ➔ GITLAB EPICS



PRODUCT BACKLOG ➔ GITLAB ISSUE LISTS AND PRIORITIZED LABELS

- The product or business owners typically create user stories to reflect the needs of the business and customers.
- They are prioritized in a product backlog to capture urgency and desired order of development.
- The product owner communicates with stakeholders to determine the priorities and constantly refines the backlog.
- In GitLab, there are dynamically generated issue lists which users can view to track their backlog.
- Labels can be created and assigned to individual issues, which then allows you to filter the issue lists by a single label or multiple labels.
- This allows for further flexibility. Priority labels can even be used to also order the issues in those lists.

SPRINTS ➔ GITLAB MILESTONES

- A sprint represents a finite time period in which the work is to be completed, which may be a week, a few weeks, or perhaps a month or more.
- The product owner and the development team meet to decide work that is in scope for the upcoming sprint.
- GitLab's milestones feature supports this: assign milestones a start date and a due date to capture the time period of the sprint.
- The team then puts issues into that sprint by assigning them to that particular milestone.

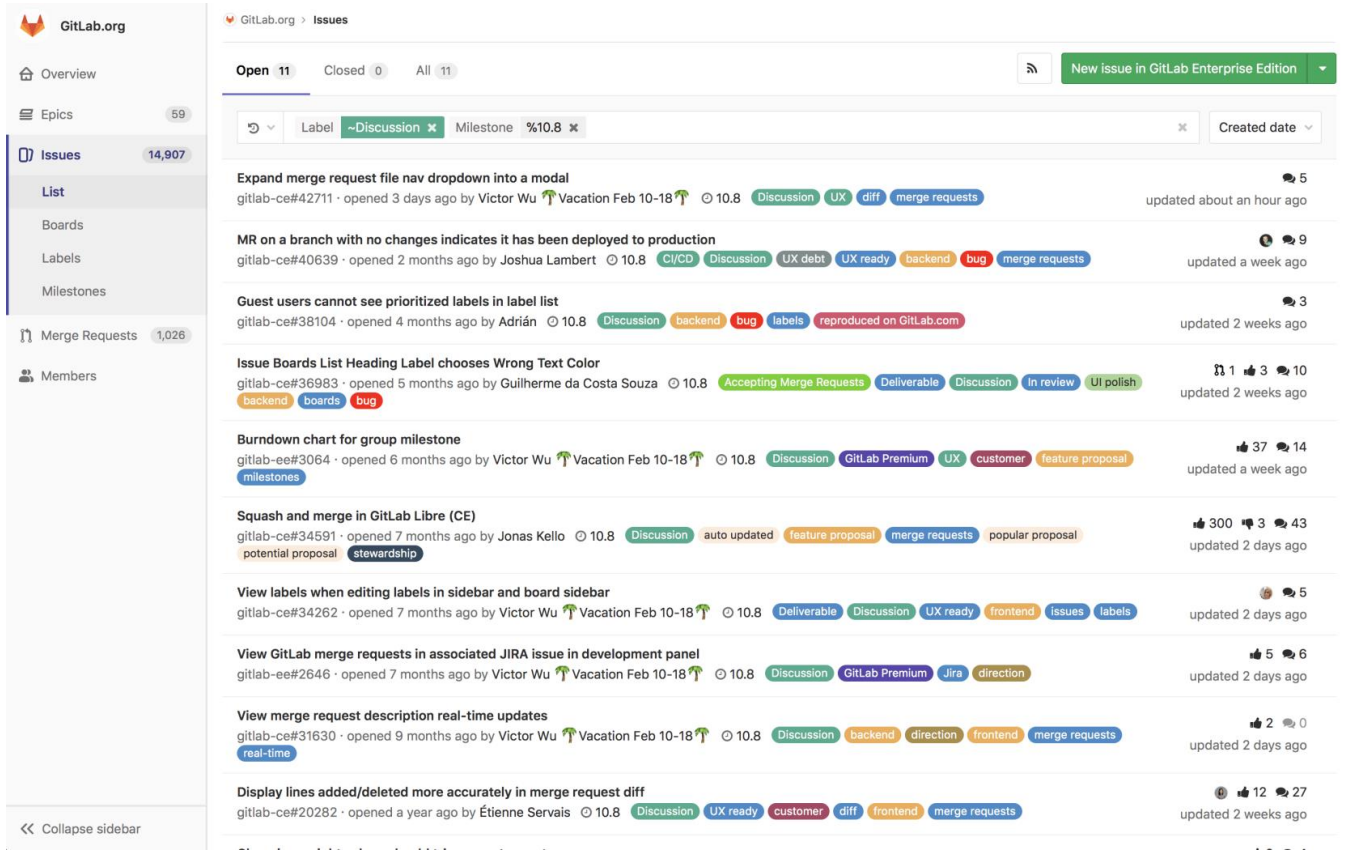
POINTS AND ESTIMATION → GITLAB ISSUE WEIGHTS

- User stories are communicated, and **the level of technical effort** is estimated for each in-scope user story
  - In GitLab, issues have a weight attribute, which you would use to indicate the estimated effort
- User stories are further broken down to **technical deliverables**, sometimes documenting technical plans and architecture
  - In GitLab, this information can be documented in the issue, or in the merge request description, as the merge request is often the place where technical collaboration happens
- During the sprint (GitLab milestone), development **team members pick up user stories to work on**, one by one
  - In GitLab, issues have assignees. So you would assign yourself to an issue to reflect that you are now working on it

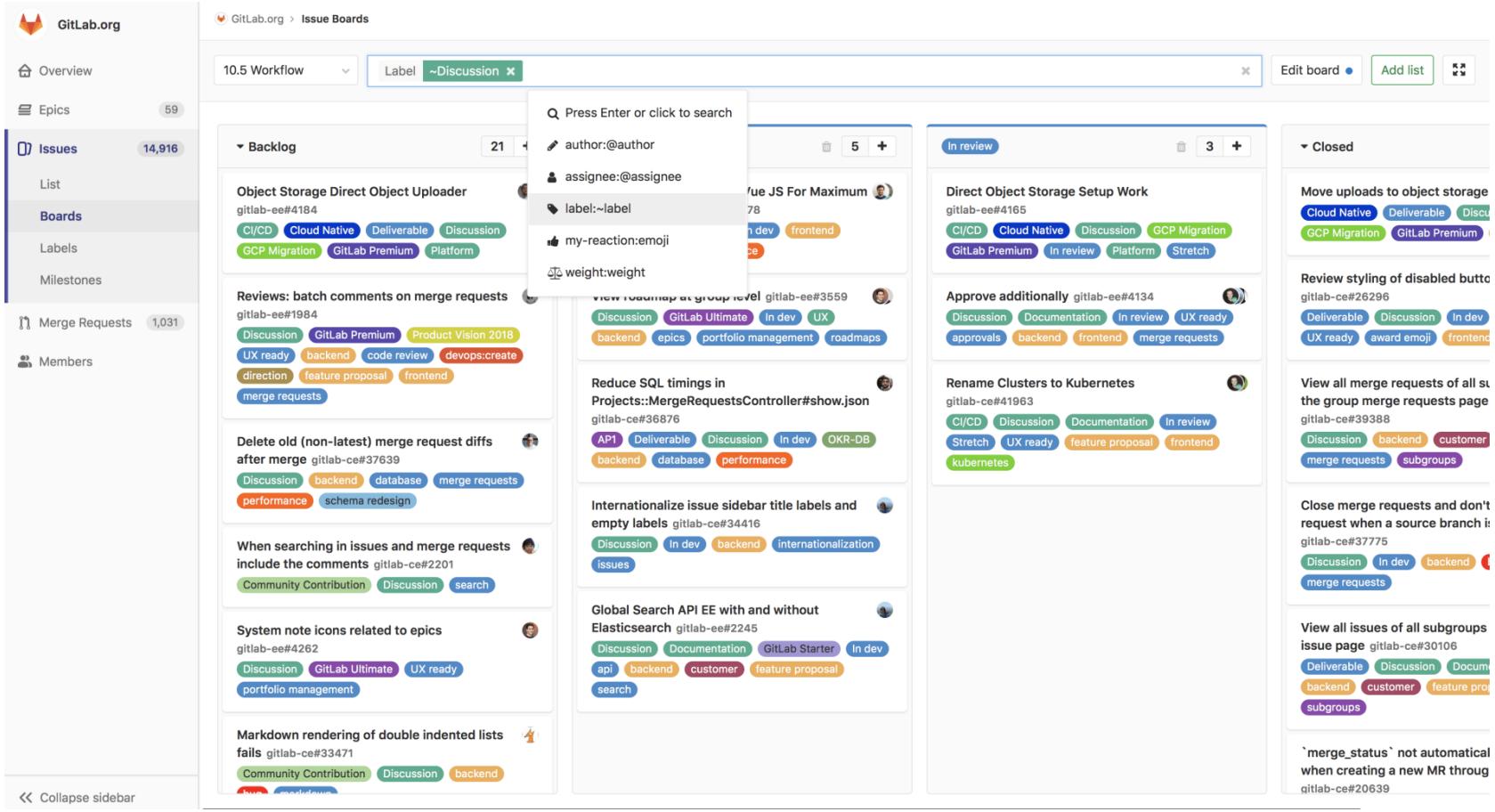
AGILE BOARD ➔ GITLAB ISSUE BOARDS

- Throughout the sprint, issues move through various stages:
  - For example, *Ready for dev, In dev, In QA, In review, Done*,
- Typically these are columns in an Agile board
  - In GitLab, issue boards allow you to define your stages and enable you to move issues through them. The team can configure the board with respect to the milestone and other relevant attributes.
- During daily standups, the team looks at the board together, to see the status of the sprint from a workflow perspective

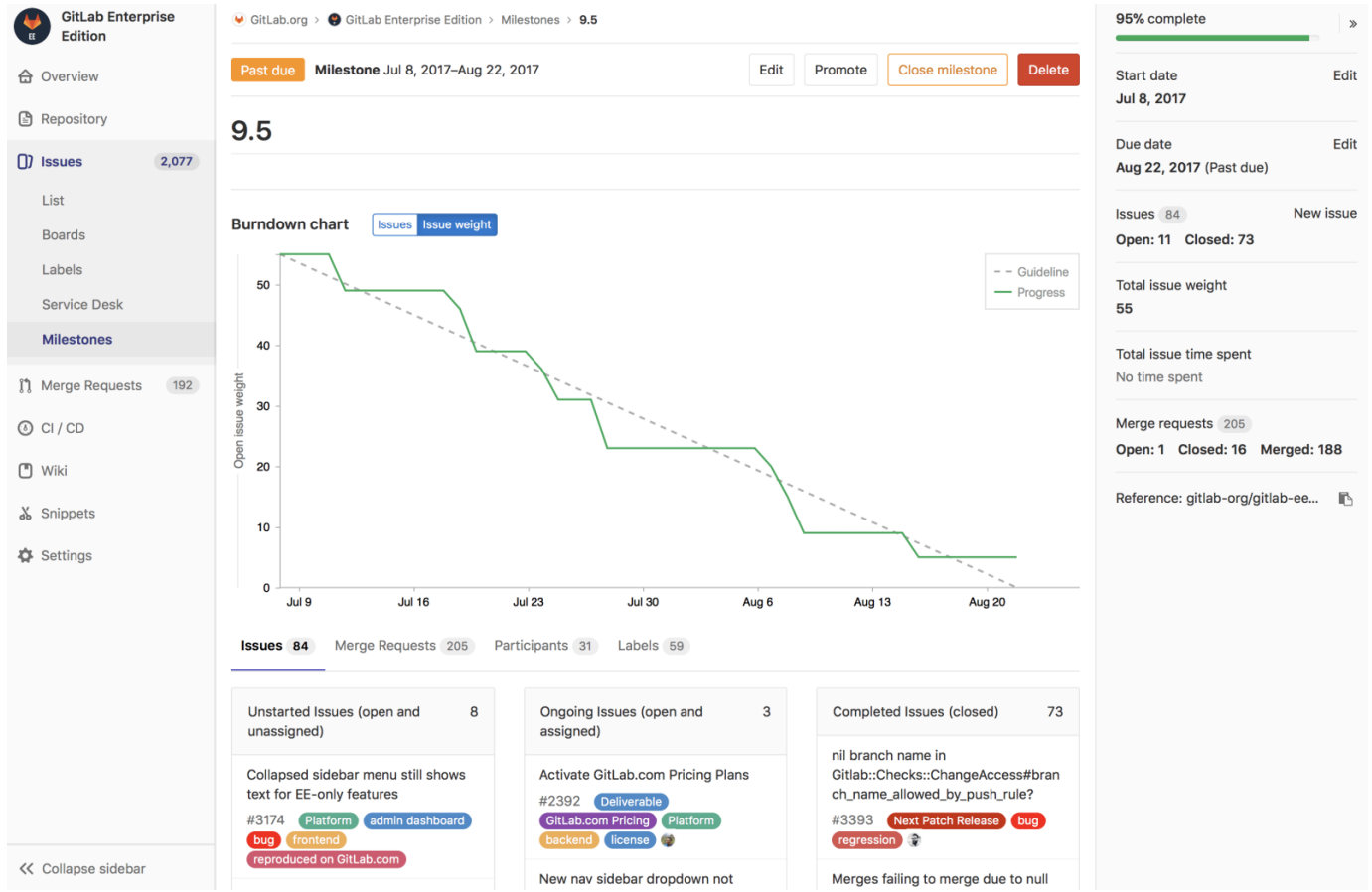
AGILE BOARD ➔ GITLAB ISSUE BOARDS



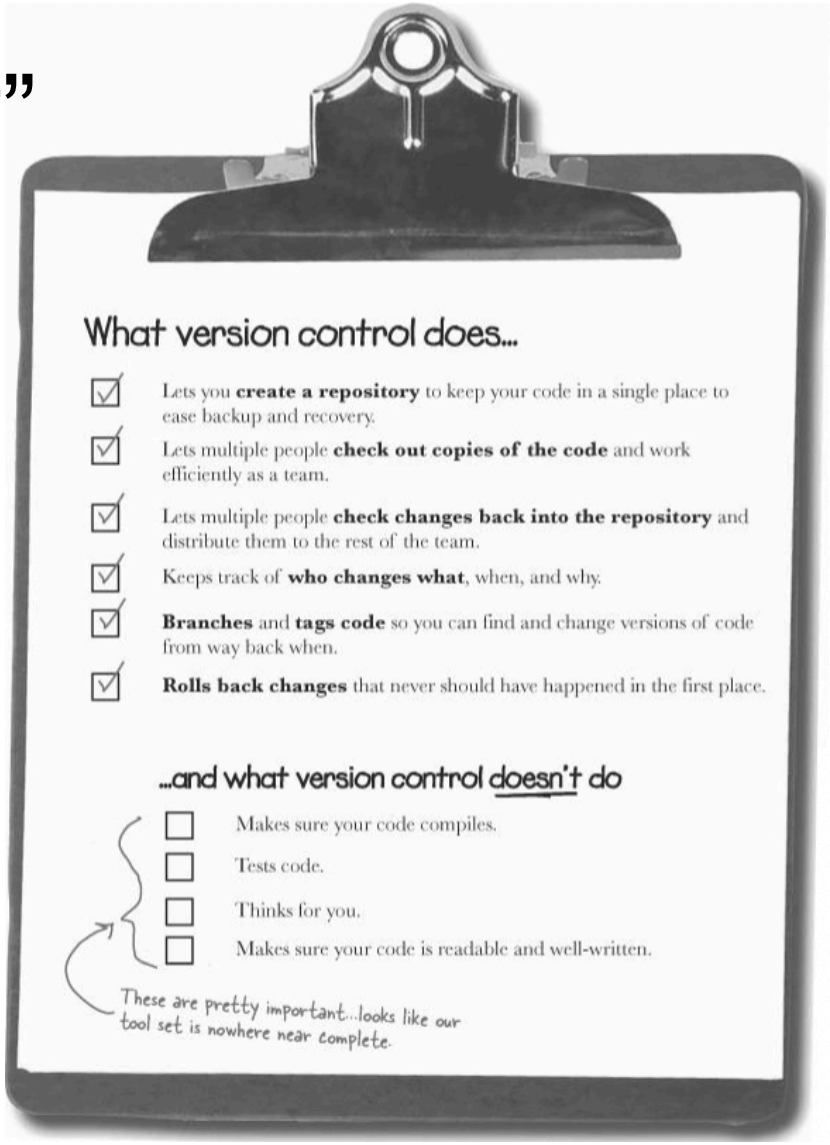
AGILE BOARD ➔ GITLAB ISSUE BOARDS



BURNDOWN CHARTS ➔ GITLAB BURNDOWN CHARTS



VCS IS NOT THE "SILVER BULLET"



Retrospective Questions

- Distributed VCS? Example?
- Centralized VCS? Example?
- SVN general process and terms?
- Gitlab and its relation to Agile? Name some mappings.