

# 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 30<sup>th</sup> → extended to April 6<sup>th</sup>
  - 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.

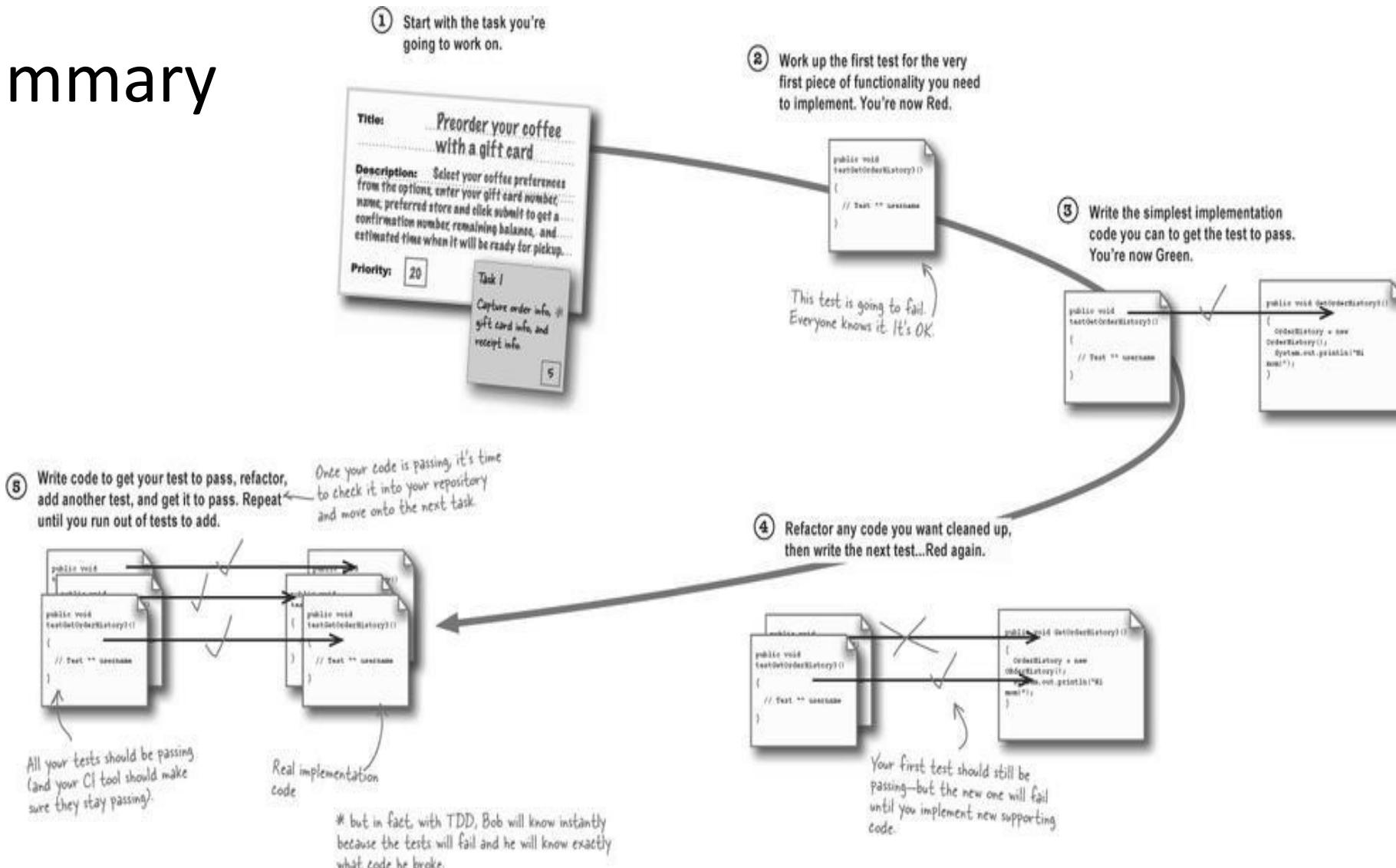
# DELIVERABLES FOR 5<sup>TH</sup> PROJECT WEEK

- Due April 2<sup>nd</sup>
- Review on Canvas – Week-10 module

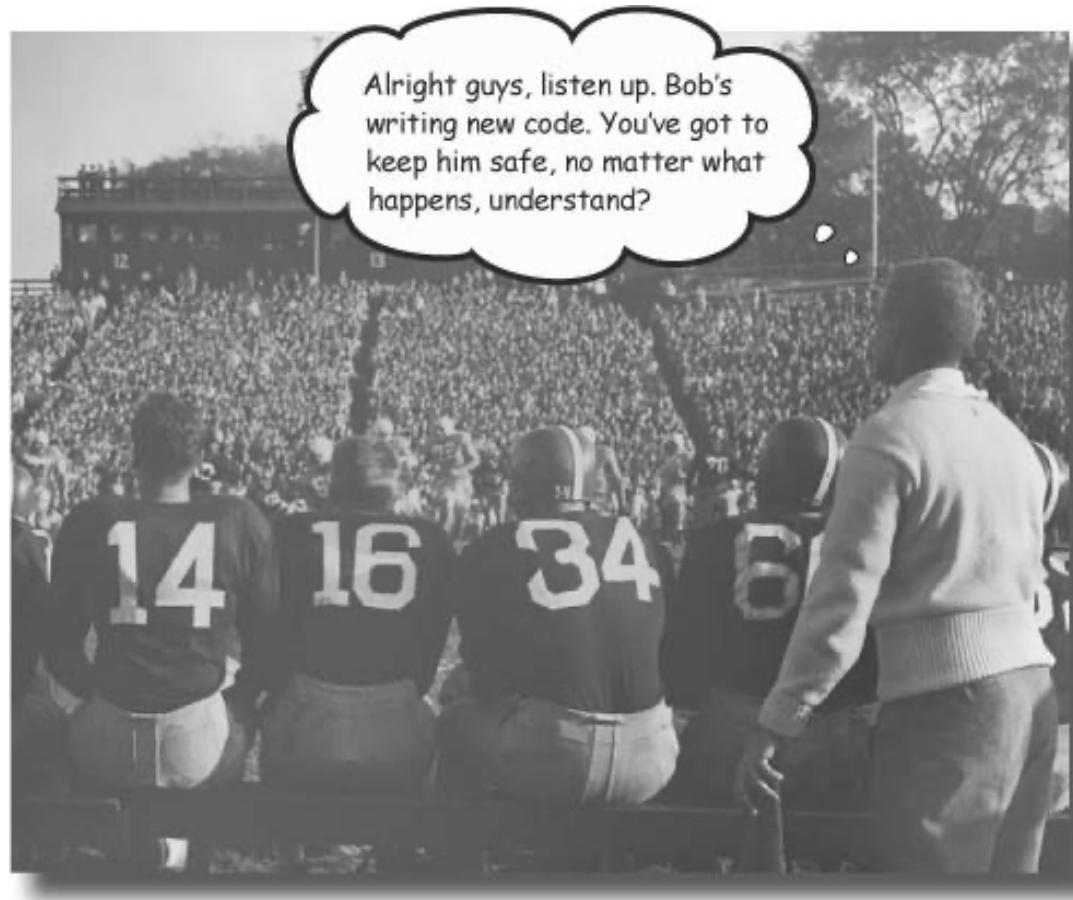
# QUESTIONS FROM LAST CLASS?

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

# TDD Summary



# 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



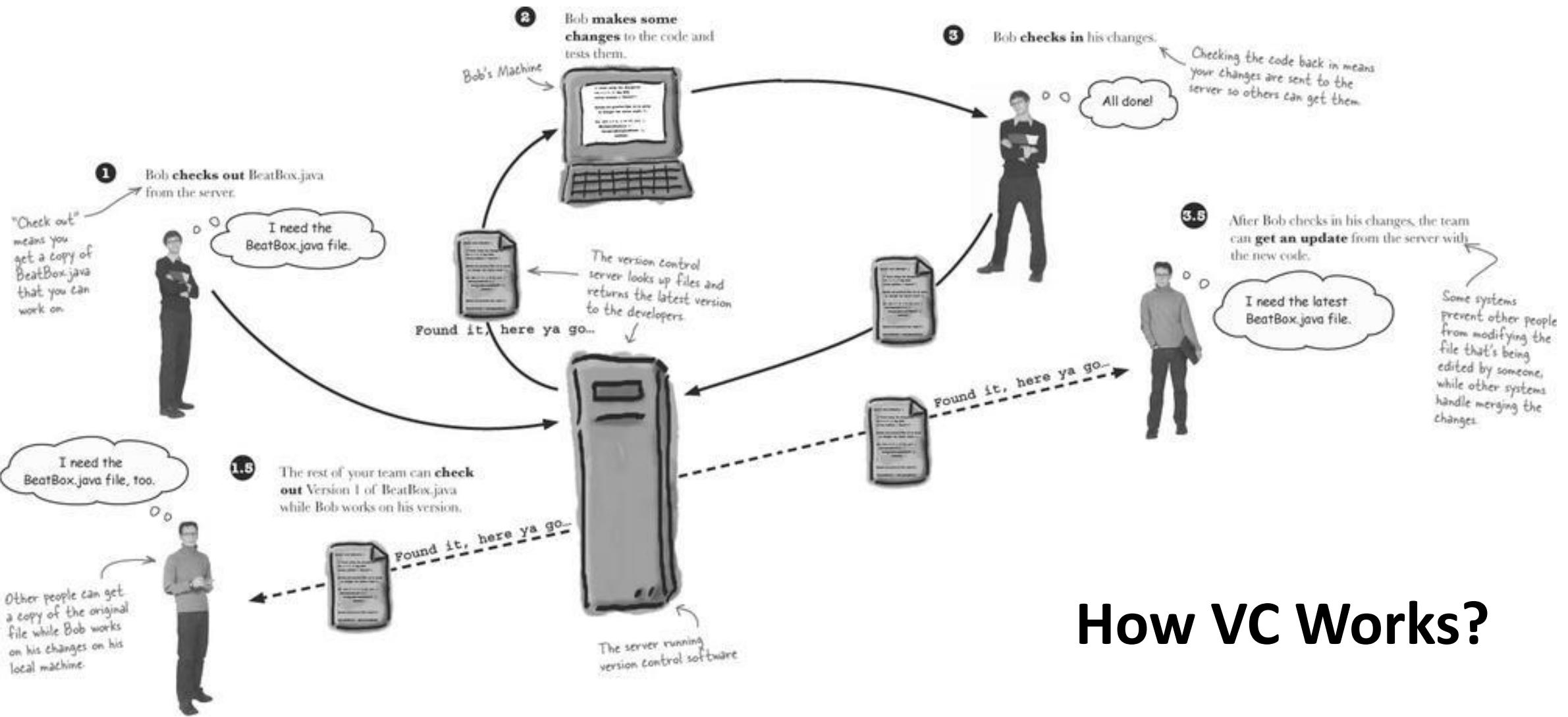
# SAFETY FIRST

- 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.



## How VC Works?

# MANY VERSION CONTROL SYSTEMS (VCS) EXIST

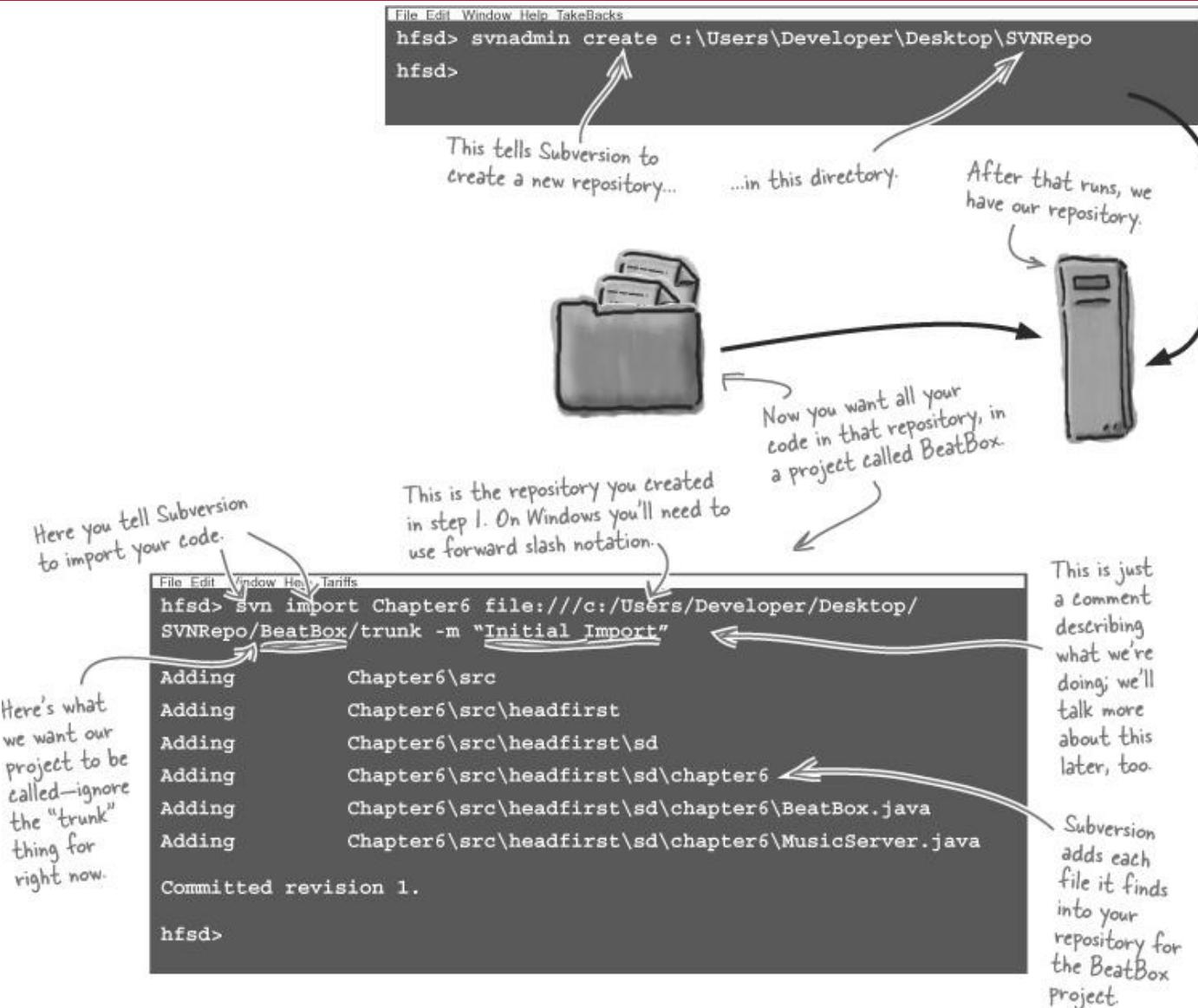
- GitLab (**today**)
- Subversion (**book**, Ch6)
- Mercurial
- CVS
- Others

# DIFFERENCES

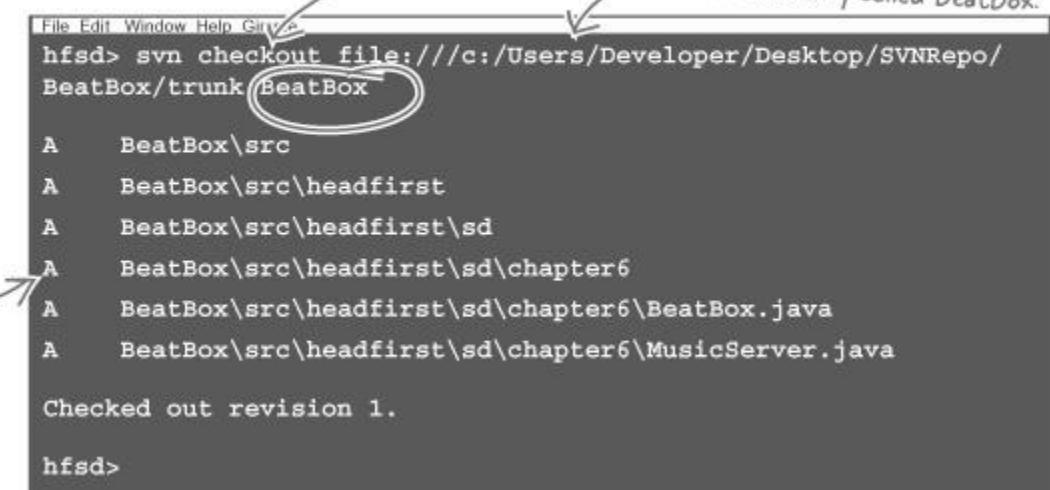
- Git and Mercurial are both “distributed revision control systems” (DRCS)
  - Distributed version control
  - More like peer-to-peer version control
  - Every copy is the complete repo (w/ history)
- Subversion and CVS are more traditional
  - Centralized version control
  - You can get the full Subversion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



# 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.
- This tells Subversion to check out a copy of the code.*
- This pulls code from the BeatBox Project in the repository and puts it in a local directory called BeatBox.*
- 
- ```
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>
```
- Subversion pulls your files back out of the repository and copies them into a new BeatBox directory (or an existing one if you've already got a BeatBox directory).*

# MAKE CHANGES LIKE YOU NORMALLY WOULD

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



You can re-implement the Poke story, since Bob broke that feature when he wrote code for the Send Picture story.

This is a normal .java file.  
Subversion doesn't change it in any way...it's still just code.

# THEN YOU COMMIT YOUR CHANGES BACK TO THE REPOSITORY

This tells Subversion to commit your changes; it will figure out what files you've changed.

This is a log message, indicating what you did.

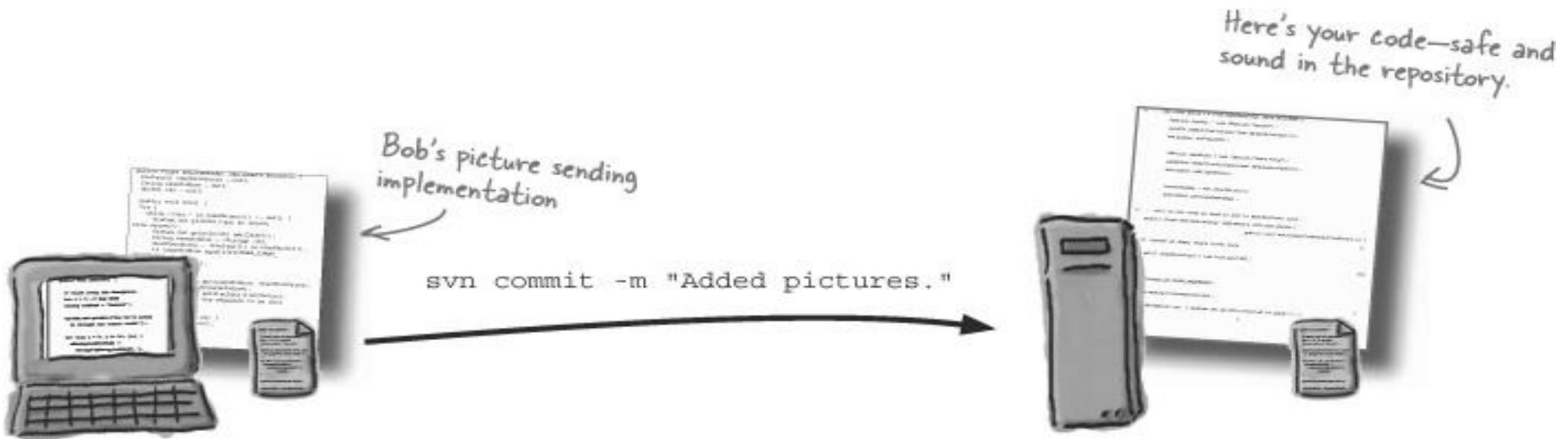
```
File Look What I Did
hfsd> svn commit -m "Added POKE support."
```

```
Sending      src\headfirst\sd\chapter6\BeatBox.java
Transmitting file data .
Committed revision 2.
```

Since you only changed one file, that's all that subversion sent to the repository—and notice that now you have a new revision number.

# A TOOL SHOULD SOLVE A TOUGH PROBLEM!

- Conflicts & Simultaneous Updates



# A TOOL SHOULD SOLVE A TOUGH PROBLEM!

You and Bob both made changes to the same file; you just got yours into the repository first.

The code on the server, with your changes

*Bob's code*

```

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*

*Bob's code*

```

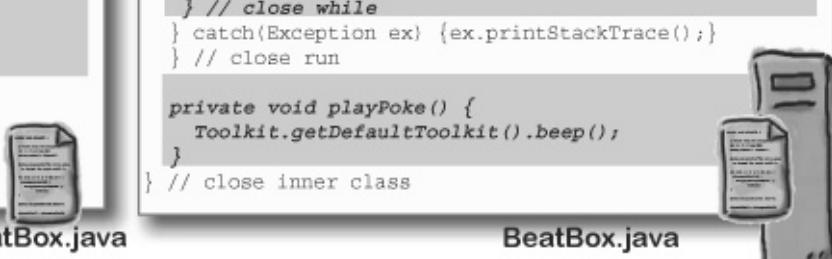
public class RemoteReader implements Runnable {
    boolean[] checkboxState = null;
    String nameToShow = null;
    Object obj = null;

    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

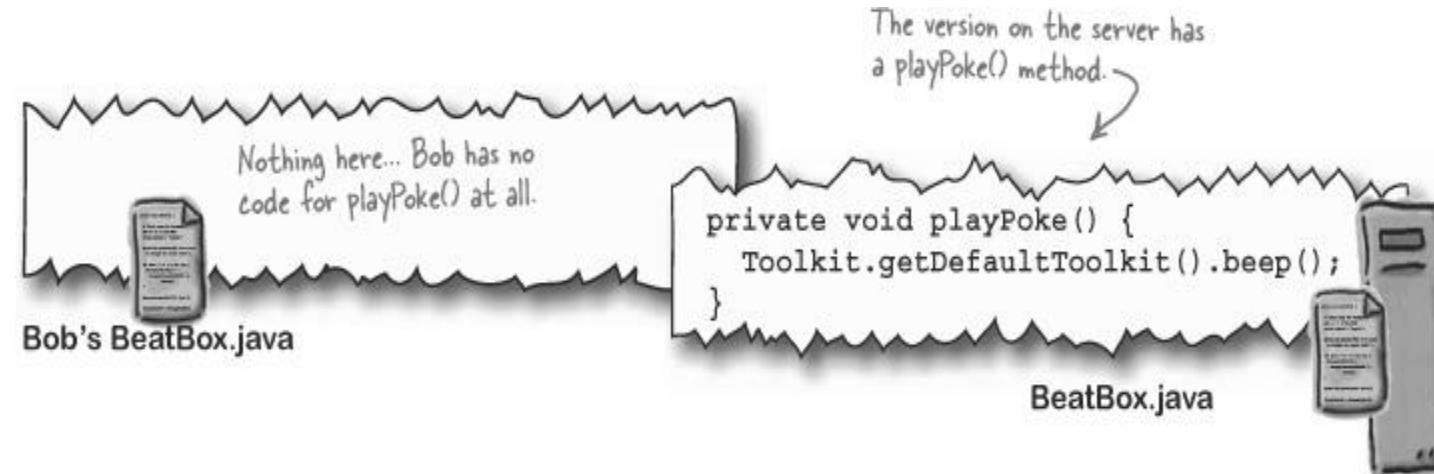
```

*BeatBox.java*



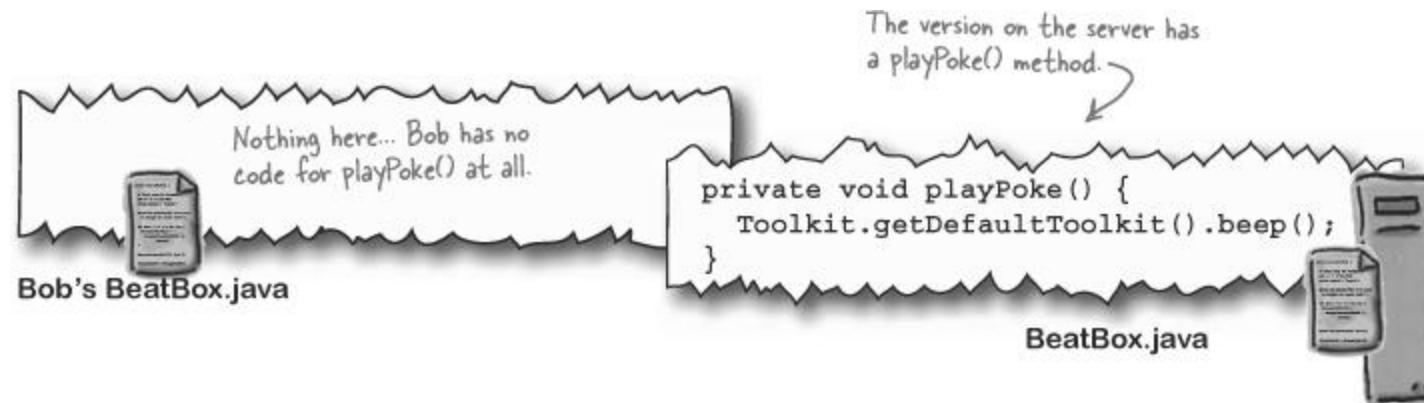
# SERVER TRIES TO MERGE YOUR CHANGES

- 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.

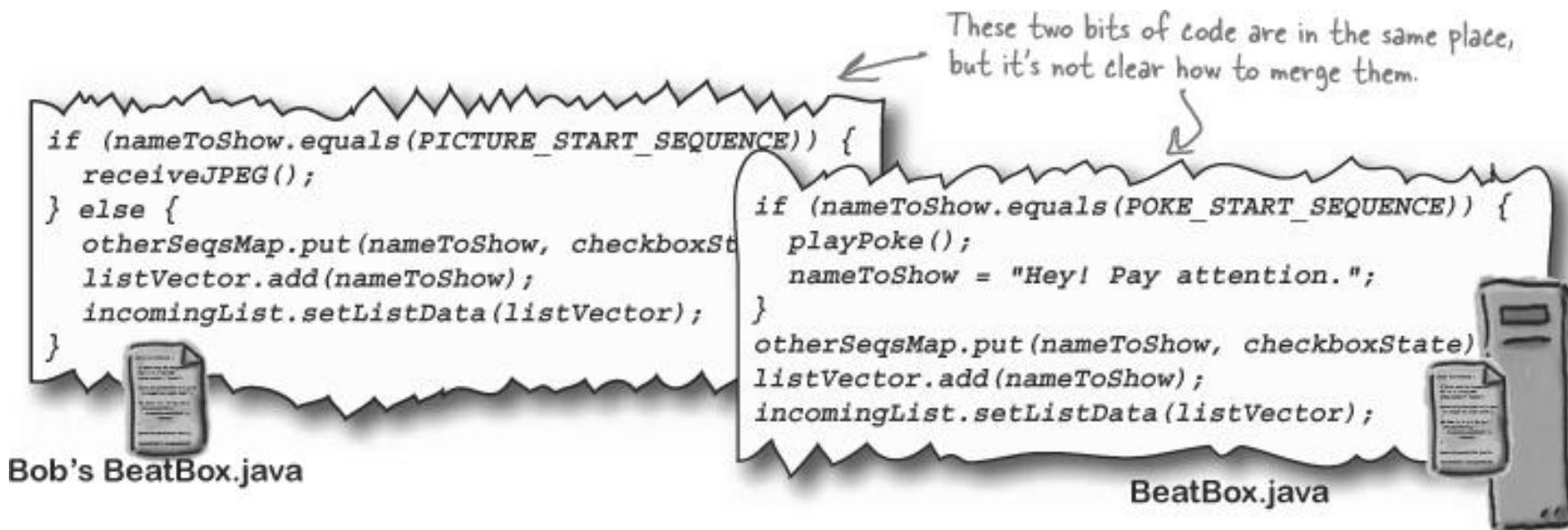


# MERGE THROUGH COMBINATION

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



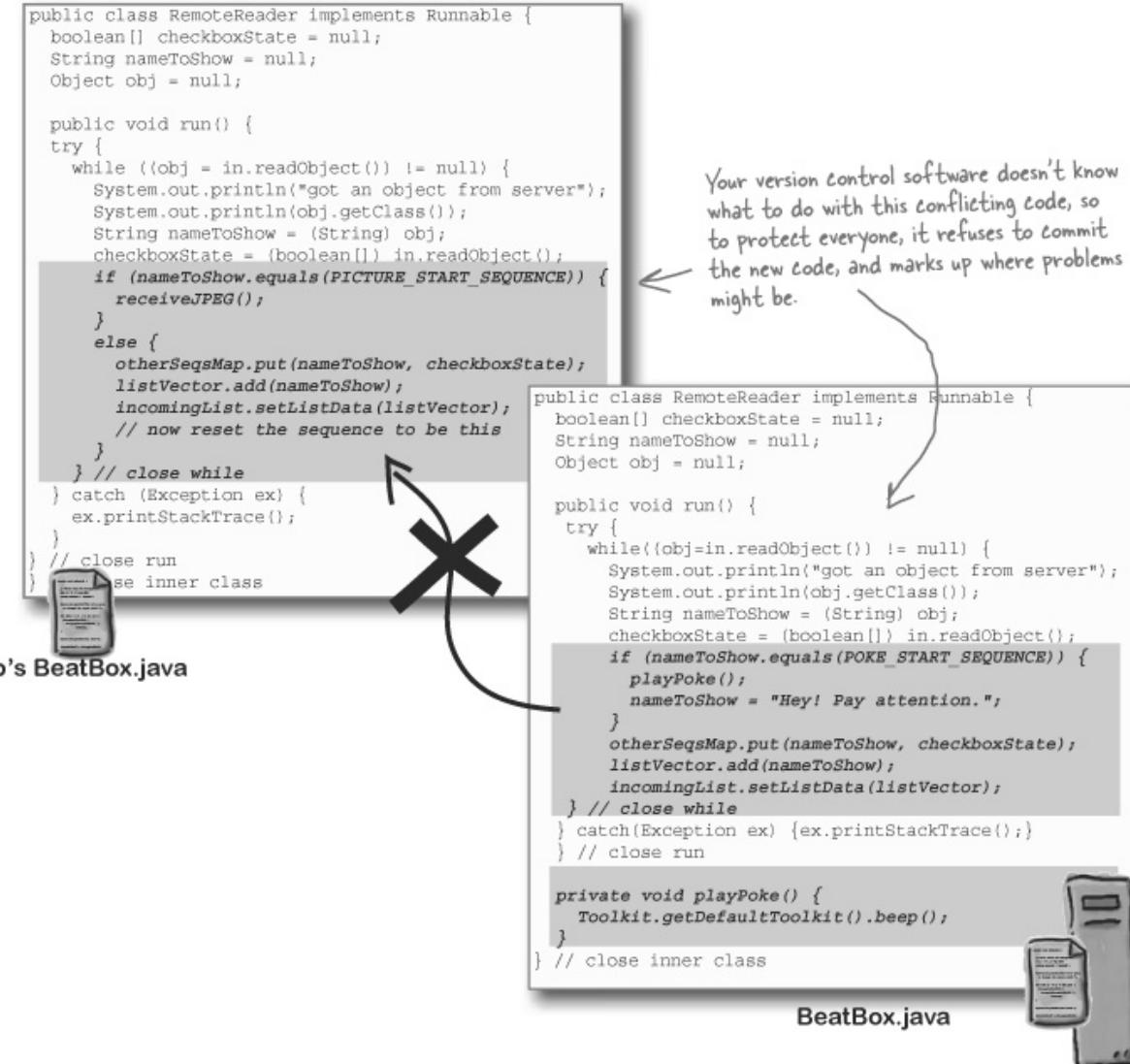
# CONFLICTING CODE IS A PROBLEM



# 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.

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(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**

```

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

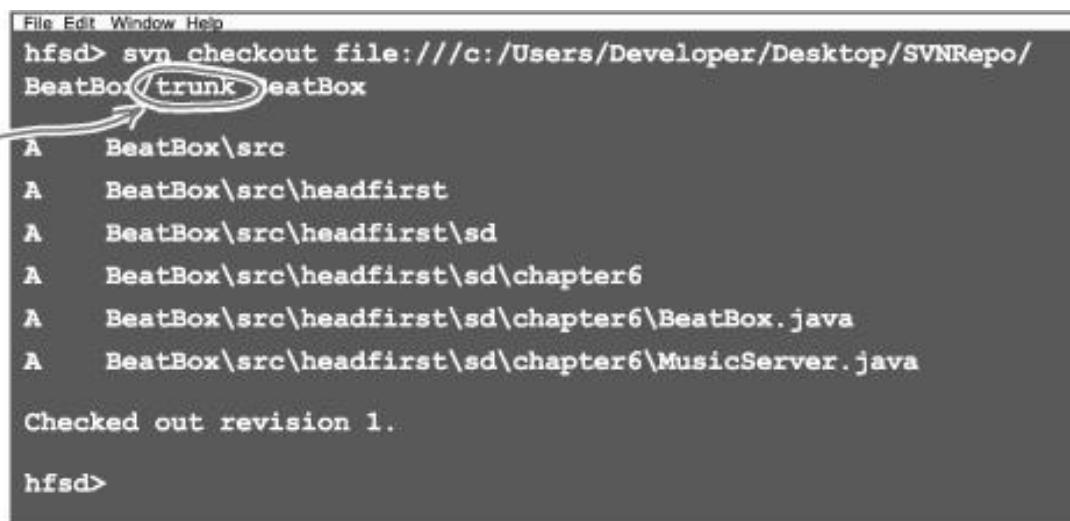
```

**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

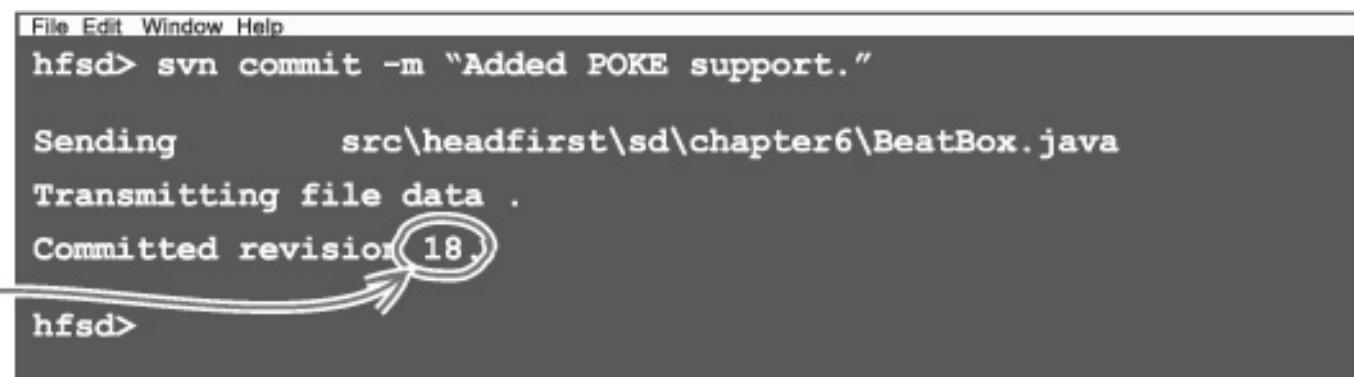
Checked out revision 1.

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.

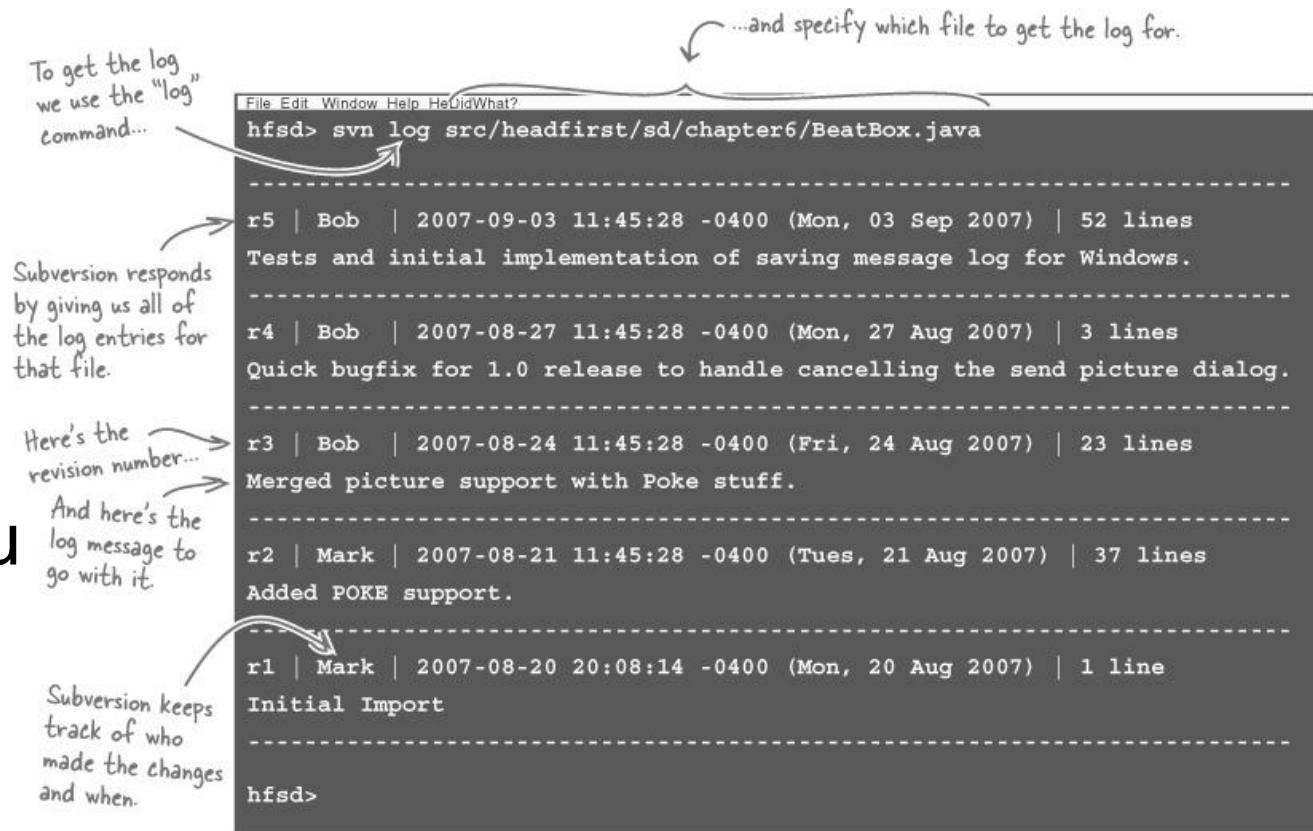
Here's the revision  
number for this set of  
changes; it increases  
with each commit.



```
File Edit Window Help
hfsd> svn commit -m "Added POKE support."
Sending      src\headfirst\sd\chapter6\BeatBox.java
Transmitting file data .
Committed revision 18.
hfsd>
```

# 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.



To get the log, we use the "log" command...

Subversion responds by giving us all of the log entries for that file.

Here's the revision number...

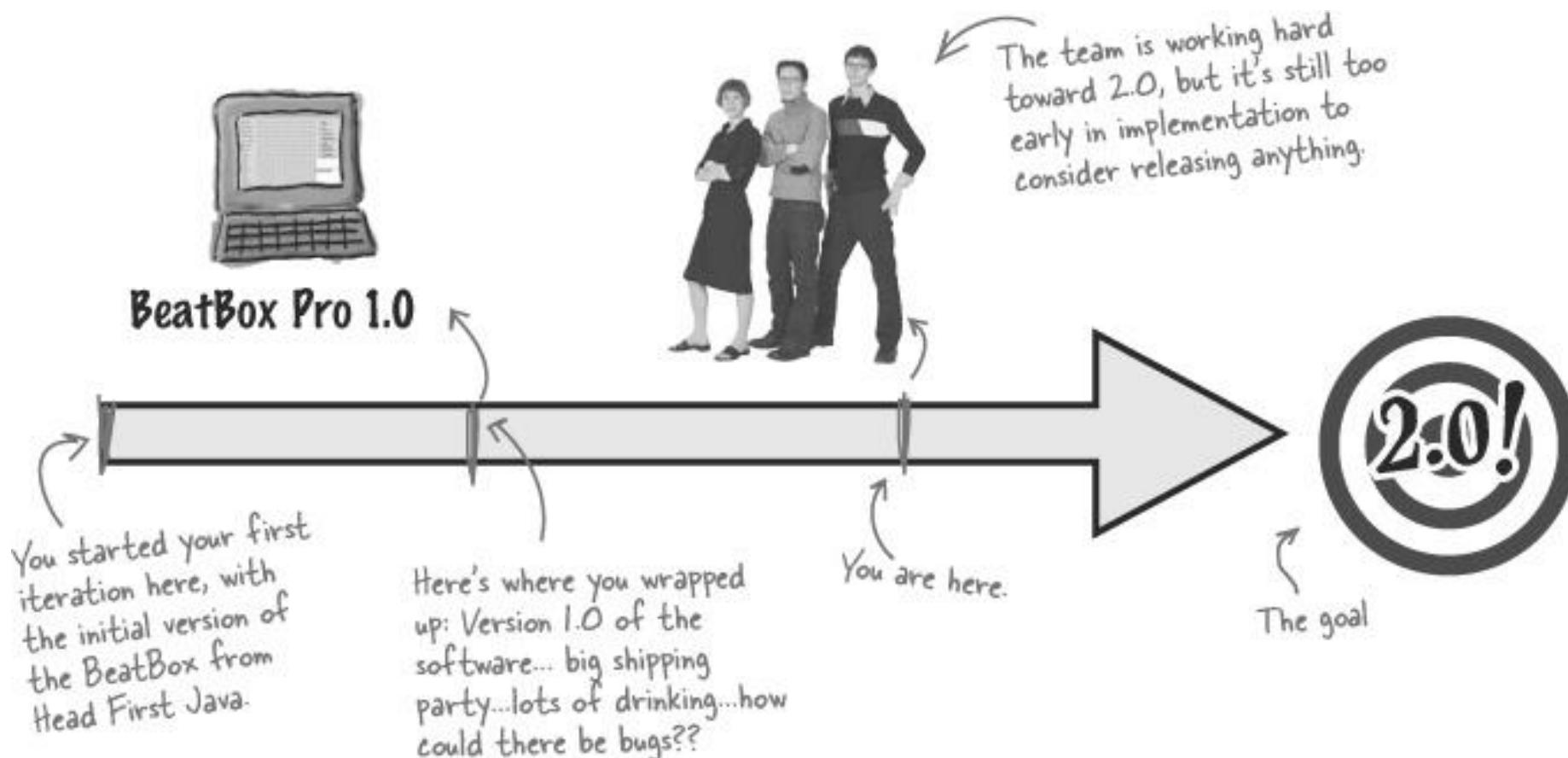
And here's the log message to go with it.

Subversion keeps track of who made the changes and when.

...and specify which file to get the log for.

```
File Edit Window Help HeDidWhat?
hfstd> svn log src/headfirst/sd/chapter6/BeatBox.java
-----
r5 | Bob | 2007-09-03 11:45:28 -0400 (Mon, 03 Sep 2007) | 52 lines
Tests and initial implementation of saving message log for Windows.
-----
r4 | Bob | 2007-08-27 11:45:28 -0400 (Mon, 27 Aug 2007) | 3 lines
Quick bugfix for 1.0 release to handle cancelling the send picture dialog.
-----
r3 | Bob | 2007-08-24 11:45:28 -0400 (Fri, 24 Aug 2007) | 23 lines
Merged picture support with Poke stuff.
-----
r2 | Mark | 2007-08-21 11:45:28 -0400 (Tues, 21 Aug 2007) | 37 lines
Added POKE support.
-----
r1 | Mark | 2007-08-20 20:08:14 -0400 (Mon, 20 Aug 2007) | 1 line
Initial Import
-----
hfstd>
```

# BUGS IN OLD VERSIONS



# CHECKING OUT OLD VERSION

This puts the code in a new directory, for Version 1.0.

✓ In Subversion, -r indicates you want a specific revision of code. We're grabbing revision 4.

```
File Edit Window Help ThatOne
hfsd> svn checkout -r4 file:///c:/Users/Developer/Desktop/
SVNRepo/BeatBox/trunk BeatBoxV1.0
A   BeatBoxV1.0\src
A   BeatBoxV1.0\src\headfirst
A   BeatBoxV1.0\src\headfirst\sd
A   BeatBoxV1.0\src\headfirst\sd\chapter6
A   BeatBoxV1.0\src\headfirst\sd\chapter6\BeatBox.java
A   BeatBoxV1.0\src\headfirst\sd\chapter6\MusicServer.java

Checked out revision 4.

hfsd>
```

# FIX AND CHECK BACK IN?

- Nope!

```
File Edit Window Help Trouble
hfsd> svn commit src/headfirst/sd/chapter6/BeatBox.java -m
"Fixed the critical security bug in 1.0 release."
Sending      src\headfirst\sd\chapter6\BeatBox.java
svn: Commit failed (details follow):
svn: Out of date: '/BeatBox/trunk/src/headfirst/sd/chapter6/
BeatBox.java' in transaction '6-1'
hfsd>
```

Uh oh, looks like  
the server isn't  
happy with your  
updated code.

# TAG VERSIONS ALONG THE WAY

You can use the  
mkdir command  
to create the  
tags directory.

```
File Edit Window Help Storage
hfsd> svn mkdir file:///c:/Users/Developer/Desktop/SVNRepo/BeatBox/tags
-m "Created tags directory"
Committed revision 6.
hfsd>
```

Instead of trunk, specify  
the tags directory here.

Here's the log message – and notice it creates a revision.  
This is a change to the project, so Subversion tracks it.

With Subversion,  
you create a tag by  
copying the revision  
you want into the  
tags directory.  
Subversion actually  
just relates that  
version tag to the  
release.

```
File Edit Window Help Yourself
hfsd> svn copy -r 4 file:///c:/Users/Developer/Desktop/SVNRepo/BeatBox/
trunk file:///c:/Users/Developer/Desktop/SVNRepo/BeatBox/tags/version-1.0
-m "Tagging the 1.0 release of BeatBox Pro."
Committed revision 6.
hfsd>
```

And we want to put that code  
into a tag called version-1.0

# How DOES THAT HELP?

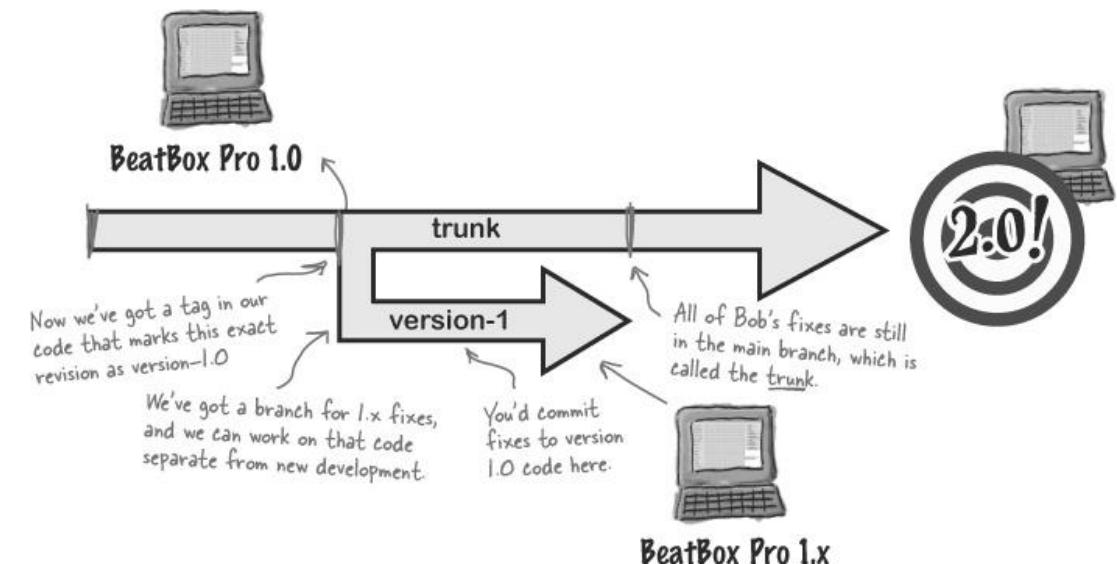
- Instead of needing to know the revision number for version 1.0 and saying `svn checkout -r 4 . . .`, you can check out Version 1.0 of the code like this:

```
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...

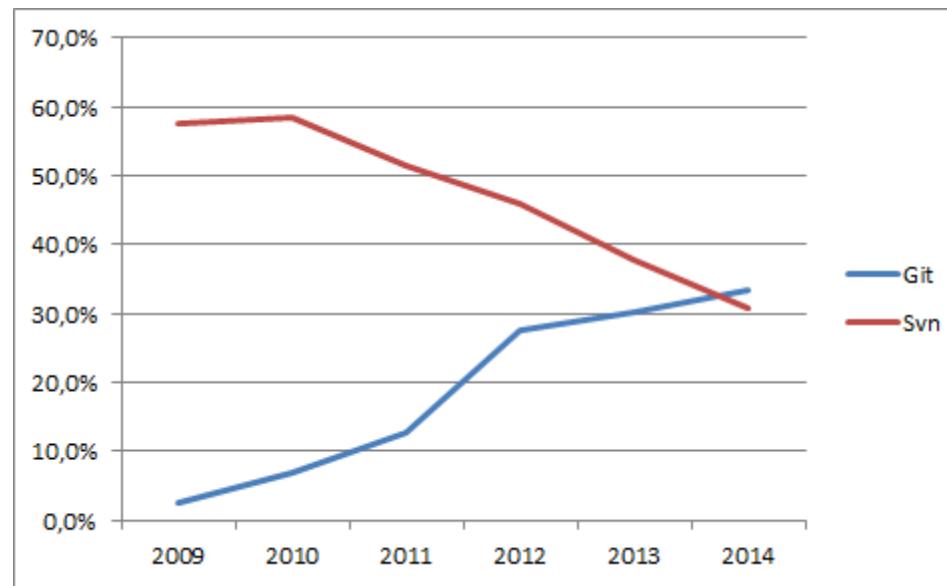
- You can accomplish your goal by **splitting code into different files** or libraries that can be built as appropriate on different platforms.
- You have a bunch of developers that can't keep their code compiling in the trunk so you try to **give them their own sandbox** to work in.

# 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](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            | <a href="#">Issues</a>                                |
| Task                  | <a href="#">Task lists</a>                            |
| Epic                  | <a href="#">Epics</a>                                 |
| Points and estimation | <a href="#">Weights</a>                               |
| Product backlog       | <a href="#">Issues lists &amp; prioritized labels</a> |
| Sprint/iteration      | <a href="#">Milestones</a>                            |
| Burndown chart        | <a href="#">Burndown charts</a>                       |
| Agile board           | <a href="#">Issue board</a>                           |

# 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

GitLab Enterprise Edition

Overview

Repository

Issues 2,077

List

Boards

Labels

Service Desk

Milestones

Merge Requests 192

CI / CD

Wiki

Snippets

Settings

Collapse sidebar

GitLab.org > GitLab Enterprise Edition > Issues > #1984

Open Opened 2 years ago by  **Sid Sijbrandij**

[Close issue](#) [New issue](#)

## Reviews: batch comments on merge requests

### Problem

► Details

### Design

Current discussion is unresolved

| Viewing the discussion                                                             | If you click the text area                                                          | If you click the button                                                             |
|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
|  |  |  |

- Notice that the checkbox label *only* allows you to resolve the discussion. You can either resolve it or leave it unresolved. There's no confusion.
- If you click the text area to start the draft, you haven't made any "draft changes" yet, so the save button is disabled until you either enter text or check the checkbox.
- If you click the button to start the draft, you've made a change by definition, so the save button is already enabled.
- If you click discard, any text changes and your checkbox selection is thrown away and you go back to the first image.
- Also note that if you are entering *first* comment (i.e. creating a brand new discussion) with your draft comment, you would get the second image above. So this is *slightly* different from existing non-draft mode because you can comment and resolve all in one action. That's okay I think.

### Current discussion is resolved

**Todo** [Add todo](#)

**Assignee**  **Hazel Yang** [@hazelyang](#) [Edit](#)

**Epic**  
**Discussion:** Batch comments on merge requests

**Milestone** **10.5** [Edit](#)

**Time tracking** [?](#)   
No estimate or time spent

**Due date** **No due date** [Edit](#)

**Labels** [Discussion](#) **GitLab Premium** [Product Vision 2018](#) [UX ready](#) [backend](#) [code review](#) [devops:create](#) [direction](#) [feature proposal](#) [frontend](#) [merge requests](#) [Edit](#)

**Weight** **7** [Edit](#)

**Confidentiality**  **Not confidential** [Edit](#)

**Lock issue**  **Unlocked** [Edit](#)

**82 participants**

# PRODUCT BACKLOG → GITLAB ISSUE LISTS AND PRIORITYIZED 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.

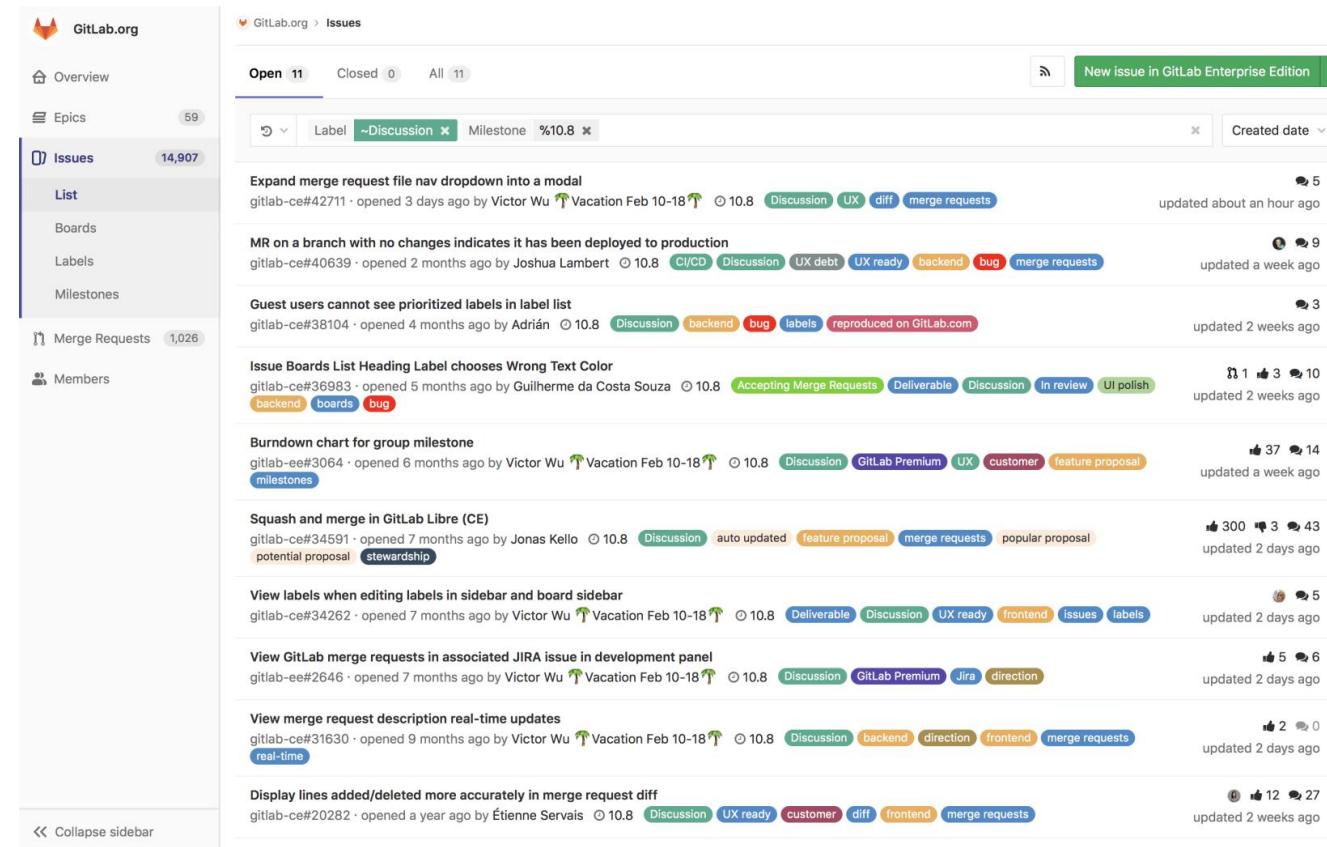
# POINTS AND ESTIMATION → GITLAB ISSUE WEIGHTS

- 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

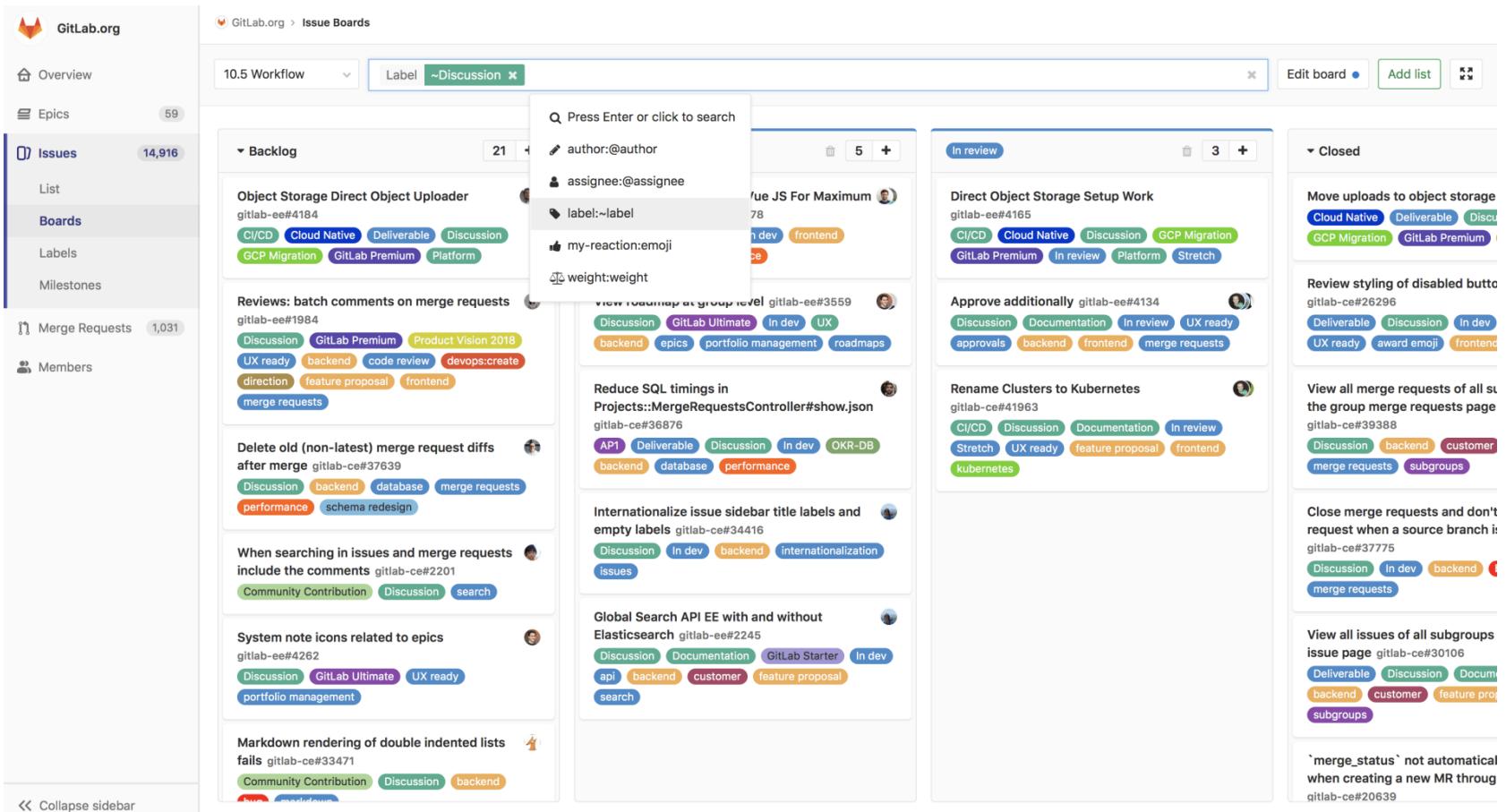


The screenshot shows the GitLab.org Issues page with the following details:

- Overview:** Shows 14,907 issues.
- Issues:** Filtered by "Label" (Discussion) and "Milestone" (%10.8).
- Issues List:** A list of 11 open issues, each with a title, description, creation date, and update information.

| Issue Title                                                                 | Description                                                        | Created   | Last Updated      |
|-----------------------------------------------------------------------------|--------------------------------------------------------------------|-----------|-------------------|
| Expand merge request file nav dropdown into a modal                         | gitlab-cef#42711 · opened 3 days ago by Victor Wu                  | Feb 10-18 | about an hour ago |
| MR on a branch with no changes indicates it has been deployed to production | gitlab-cef#40639 · opened 2 months ago by Joshua Lambert           | Oct 10.8  | a week ago        |
| Guest users cannot see prioritized labels in label list                     | gitlab-cef#38104 · opened 4 months ago by Adrián                   | Oct 10.8  | 2 weeks ago       |
| Issue Boards List Heading Label chooses Wrong Text Color                    | gitlab-cef#36983 · opened 5 months ago by Guilherme da Costa Souza | Oct 10.8  | 2 weeks ago       |
| Burndown chart for group milestone                                          | gitlab-eef#3064 · opened 6 months ago by Victor Wu                 | Feb 10-18 | a week ago        |
| Squash and merge in GitLab Libre (CE)                                       | gitlab-cef#34591 · opened 7 months ago by Jonas Kello              | Oct 10.8  | 2 days ago        |
| View labels when editing labels in sidebar and board sidebar                | gitlab-cef#34262 · opened 7 months ago by Victor Wu                | Feb 10-18 | 2 days ago        |
| View GitLab merge requests in associated JIRA issue in development panel    | gitlab-eef#2646 · opened 7 months ago by Victor Wu                 | Feb 10-18 | 2 days ago        |
| View merge request description real-time updates                            | gitlab-cef#31630 · opened 9 months ago by Victor Wu                | Feb 10-18 | 2 days ago        |
| Display lines added/deleted more accurately in merge request diff           | gitlab-cef#20282 · opened a year ago by Étienne Servais            | Oct 10.8  | 2 weeks ago       |

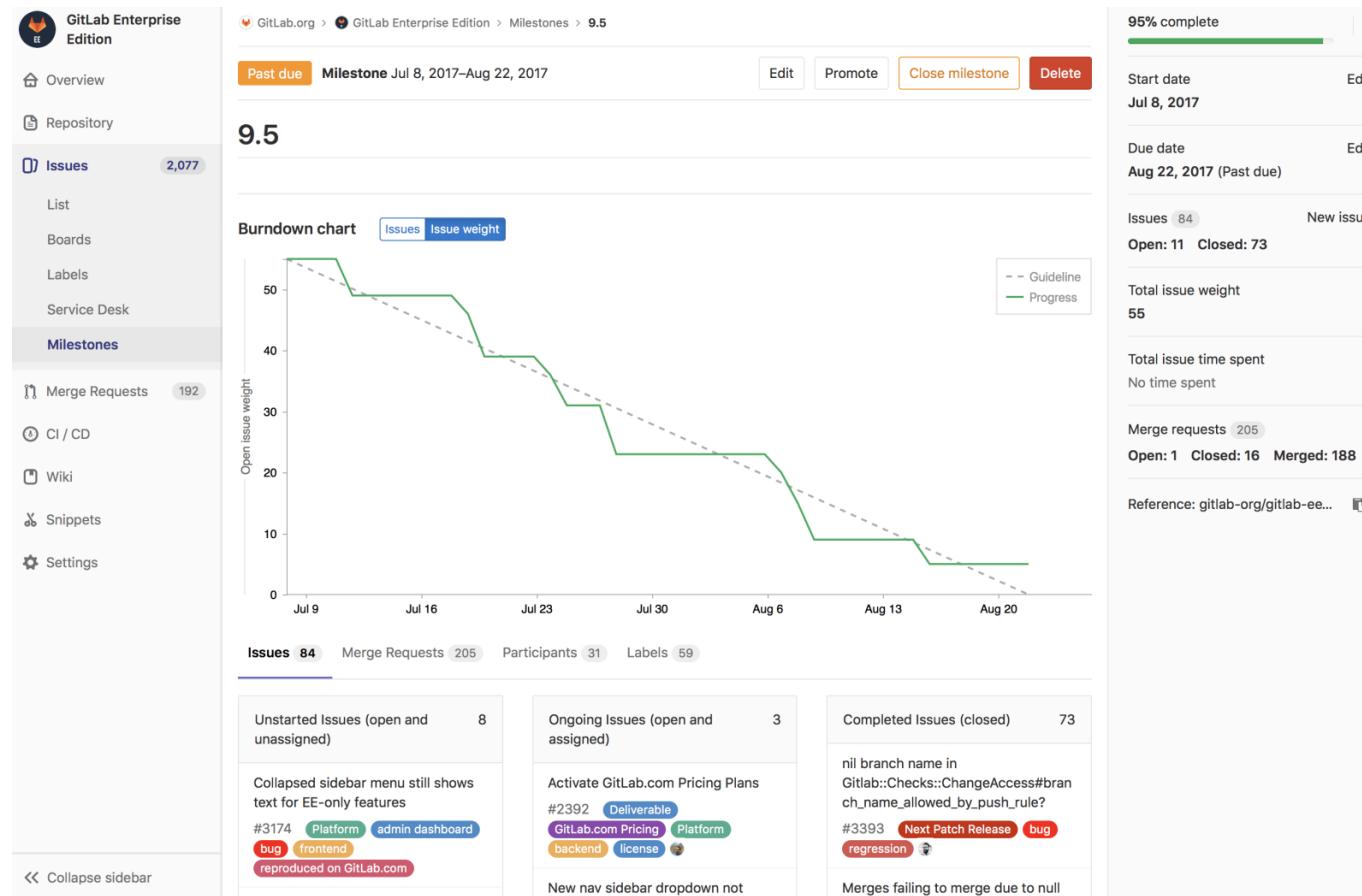
# AGILE BOARD → GITLAB ISSUE BOARDS



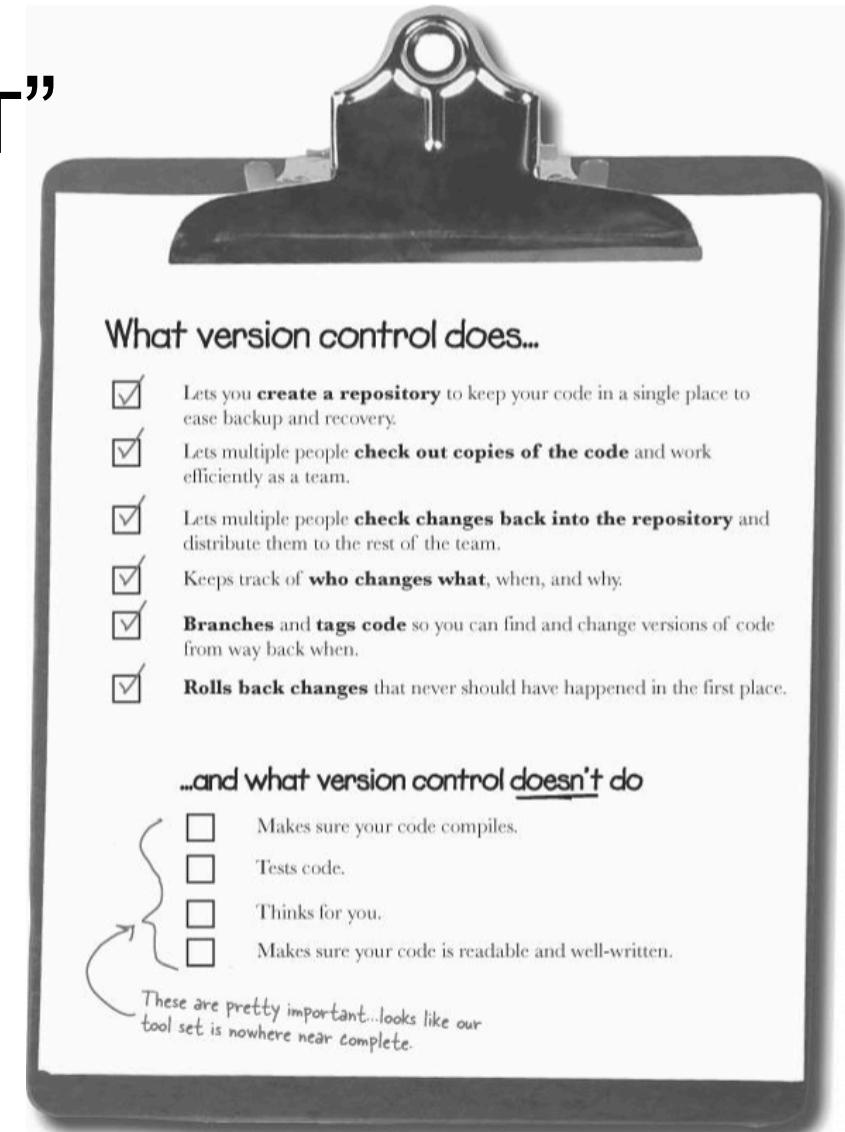
The screenshot shows the GitLab.org interface for issue boards. The left sidebar includes links for Overview, Epics, Issues (14,916), Boards (selected), Labels, Milestones, Merge Requests (1,031), and Members. The main area displays a board titled "10.5 Workflow" with a filter for "Label ~Discussion". The board is divided into sections: Backlog, In review, and Closed.

- Backlog:**
  - Object Storage Direct Object Uploader (gitlab-ee#4184) - CI/CD, Cloud Native, Deliverable, Discussion, GCP Migration, GitLab Premium, Platform
  - Reviews: batch comments on merge requests (gitlab-ee#1984) - Discussion, GitLab Premium, Product Vision 2018, UX ready, backend, code review, devops:create, direction, feature proposal, frontend, merge requests
  - Delete old (non-latest) merge request diffs after merge (gitlab-ce#37639) - Discussion, backend, database, merge requests, performance, schema redesign
  - When searching in issues and merge requests include the comments (gitlab-ce#2201) - Community Contribution, Discussion, search
  - System note icons related to epics (gitlab-ee#4262) - Discussion, GitLab Ultimate, UX ready, portfolio management
  - Markdown rendering of double indented lists fails (gitlab-ce#33471) - Community Contribution, Discussion, backend
- In review:**
  - Direct Object Storage Setup Work (gitlab-ee#4165) - CI/CD, Cloud Native, Discussion, GCP Migration, GitLab Premium, In review, Platform, Stretch
  - Approve additionally (gitlab-ee#4134) - Discussion, Documentation, In review, UX ready, approvals, backend, frontend, merge requests
  - Rename Clusters to Kubernetes (gitlab-ce#41963) - CI/CD, Discussion, Documentation, In review, OKR-DB, backend, database, performance
  - Internationalize issue sidebar title labels and empty labels (gitlab-ce#34416) - Discussion, In dev, backend, internationalization, issues
  - Global Search API EE with and without Elasticsearch (gitlab-ee#2245) - Discussion, Documentation, GitLab Starter, In dev, api, backend, customer, feature proposal, search
- Closed:**
  - Move uploads to object storage (gitlab-ee#4165) - Cloud Native, Deliverable, Discussion, GCP Migration, GitLab Premium
  - Review styling of disabled buttons (gitlab-ce#26296) - Deliverable, Discussion, In dev, UX ready, award emoji, frontend
  - View all merge requests of all subgroups in the group merge requests page (gitlab-ce#39388) - Discussion, backend, customer, merge requests, subgroups
  - Close merge requests and don't request when a source branch is merged (gitlab-ce#37775) - Discussion, In dev, backend, merge requests
  - View all issues of all subgroups issue page (gitlab-ce#30106) - Deliverable, Discussion, Documentation, backend, customer, feature proposal, subgroups
  - 'merge\_status' not automatical when creating a new MR through the UI (gitlab-ce#20639)

# 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.