Code Review

Today

- What is code review?
- Code review in practice
  - Review under Googlecode

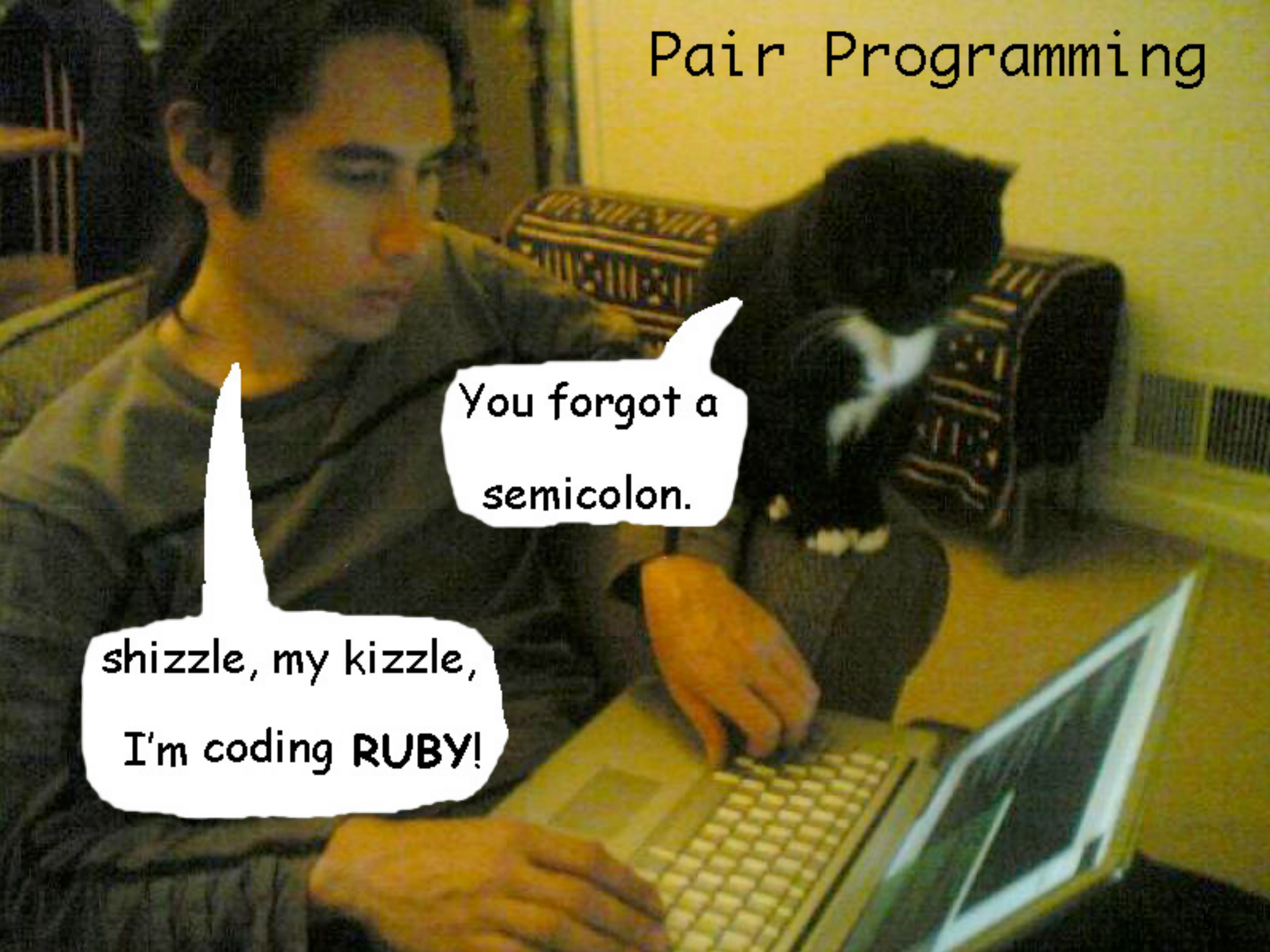
What is code review?

- Off-line version of pair-programming
- Review code written by other developers
- Not finding faults in others, but collaboration
- Voluntary review is important
- Or you will be forced to review other developers' broken code

Why code review?

- Two or more eyes/brains are better than one
  - Find mistakes easily and quickly
- Motivations to follow code guide lines
- New developer education
- Learning by mistakes
- Of course, reliable software

Pair Programming



Pair programming

- **pair programming**: 2 people, 1 computer
  - take turns "driving"
  - rotate pairs often
  - pair people of different experience levels

- pros:
  - Can produce better code
  - An inexperienced coder can learn from an experienced one

- cons:
  - Some people don't like it

Motivation for reviews

- Can catch most bugs, design flaws early.
- > 1 person has seen every piece of code.
  - Prospect of someone reviewing your code raises quality threshold.
- Forces code authors to articulate their decisions and to participate in the discovery of flaws.
- Allows junior personnel to get early hands-on experience without hurting code quality
  - Pairing them up with experienced developers
  - Can learn by being a reviewer as well
- Accountability. Both author and reviewers are accountable for the code.
- Explicit non-purpose:
  - Assessment of individuals for promotion, pay, ranking, etc.
  - Management is usually not permitted at reviews

Types of code review

- What is reviewed:
  - A specification
  - A coherent module (sometimes called an “inspection”)
  - A single checkin or code commit (incremental review)
- Who participates:
  - One other developer
  - A group of developers
- Where:
  - In-person meeting
    - Best to prepare beforehand: artifact is distributed in advance
    - Preparation usually identifies more defects than the meeting
  - Email/electronic

Code reviews in industry

- Code reviews are a **very** common industry practice.
- Made easier by advanced tools that:
  - integrate with configuration management systems
  - highlight changes (i.e., diff function)
  - allow traversing back into history
  - E.g.: Eclipse, SVN tools

My approach

- Distribute code (or other artifacts) ahead of time
  - Common pagination
  - Documentation is required (as is good style)
  - No extra overview from developer
- Each reviewer focuses where he/she sees fit
- Mark up with lots of comments
- Identify 5 most important issues
- At meeting, go around the table raising one issue
  - Discuss the reasons for the current design, and possible improvements
- Author takes all printouts and addresses all issues
  - Not just those raised in the meeting

Exercise

"Code review" this checkin. What feedback would you give the author? What changes would you request before checkin?

```
public class Account {
   double principal,rate;   int daysActive,accountType;

   public static final int STANDARD=0, BUDGET=1,
   PREMIUM=2, PREMIUM_PLUS=3;
}
...
public static double calculateFee(Account[] accounts)
{
   double totalFee = 0.0;
   Account account;
   for (int i=0;i<accounts.length;i++) {
      account=accounts[i];
      if ( account.accountType == Account.PREMIUM ||
           account.accountType == Account.PREMIUM_PLUS )
         totalFee += .0125 * (                // 1.25% broker's fee
            account.principal * Math.pow(account.rate,
            (account.daysActive/365.25))
            - account.principal);             // interest-principal
   }
   return totalFee;
}
```

Improved code (page 1)

```
/** An individual account.  Also see CorporateAccount. */
public class Account {
   private double principal;
   /** The yearly, compounded rate (at 365.25 days per year). */
   private double rate;
   /** Days since last interest payout. */
   private int daysActive;
   private Type type;

   /** The varieties of account our bank offers. */
   public enum Type {STANDARD, BUDGET, PREMIUM, PREMIUM_PLUS}

   /** Compute interest. **/
   public double interest() {
      double years = daysActive / 365.25;
      double compoundInterest = principal * Math.pow(rate, years);
      return compoundInterest - principal;
   }

   /** Return true if this is a premium account. **/
   public boolean isPremium() {
      return accountType == Type.PREMIUM ||
             accountType == Type.PREMIUM_PLUS;
   }
```

Improved code (page 2)

```
/** The portion of the interest that goes to the broker. **/
public static final double BROKER_FEE_PERCENT = 0.0125;

/** Return the sum of the broker fees for all the given accounts. **/
public static double calculateFee(Account accounts[]) {
   double totalFee = 0.0;
   for (Account account : accounts) {
      if (account.isPremium()) {
         totalFee += BROKER_FEE_PERCENT * account.interest();
      }
   }
   return totalFee;
}

}
```

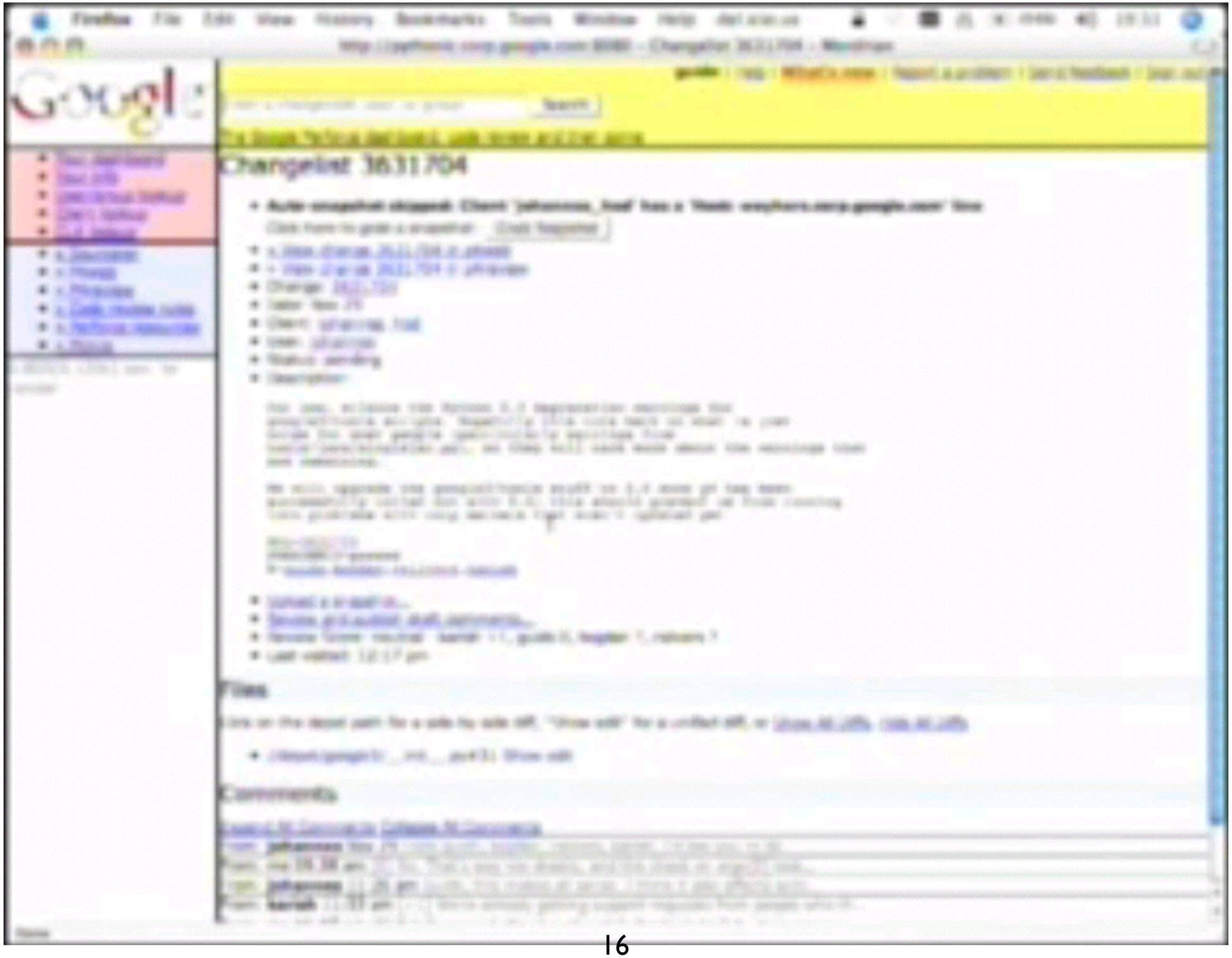
Code review in open source

- Author & reviewer on separate computers
- Author invokes "diff –u" to create patch file
- Author mails patch file to reviewer
  - Or uploads to e.g. SourceForge patch manager
- Reviewer users "patch" to recreate the files
- Then email back-and-forth a few times
- Finally, reviewer submits into svn/cvs/etc
  - Patch author often has no privileges (yet)
  - Process helps "voting" new developers

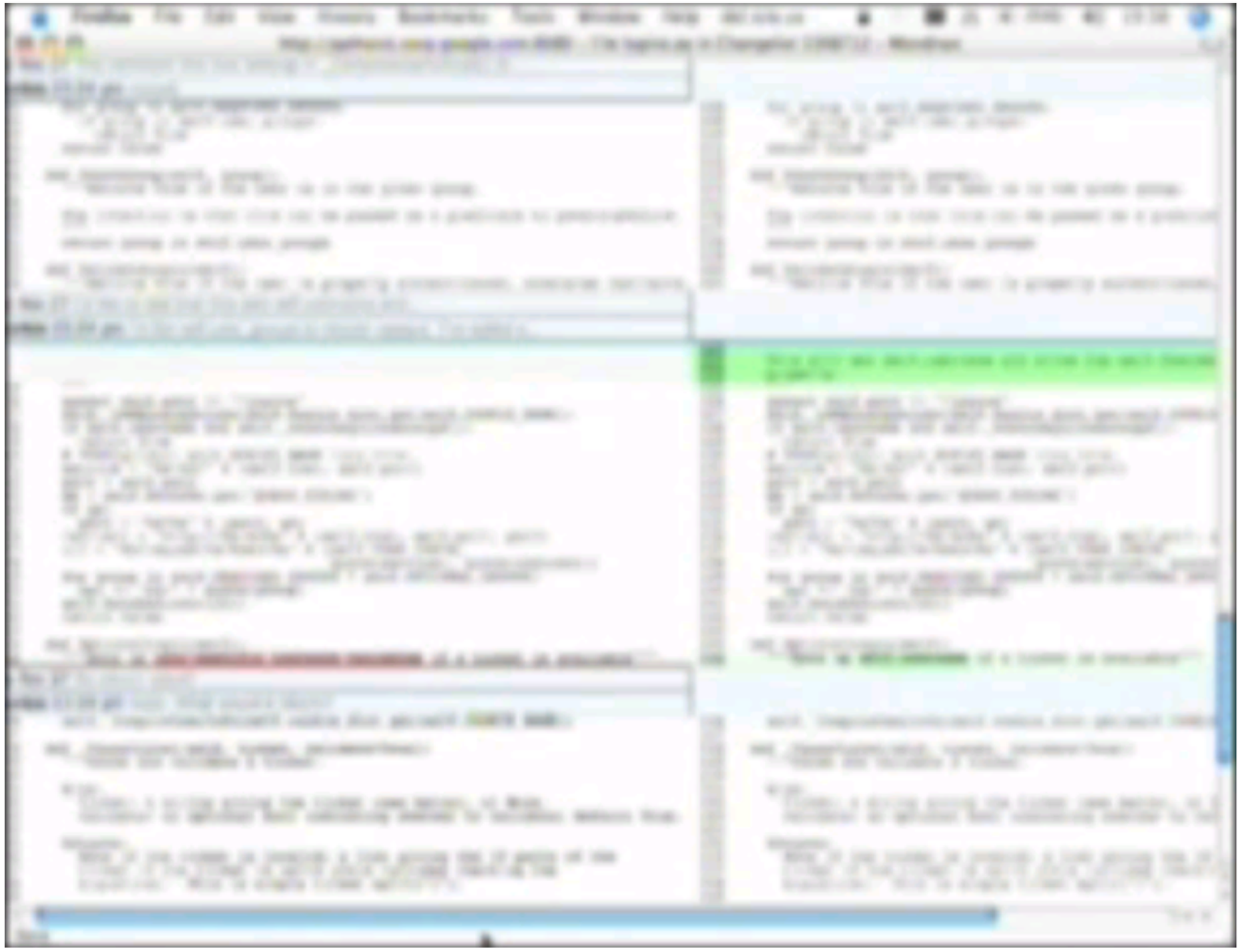
Google Mondrian

- Web based system
- Work with email for comparability
- Side-by-side/line-by-line comments
- Review process monitoring

Mondrian – Change



Mondrian – Diff



Mondrian at Google

- Used by many inside google projects
- Voluntarily
- Work with bug tracking systems and FindBugs
- A great talk given by Guido Van Rossum http://www.youtube.com/watch?v=sMql3Di4Kgc

Code review (googlecode)

Project Home | Issues | Source | Administer

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

Source path: svn/ ‹r248 r249

- svn
  - branches
  - tags
  - trunk
    - java
      - data
      - lib
      - script
      - src
        - com
          - googlecode
            - crashma
              - findbugs
              - stacktrace
              - util
              - weka
    - paper

Filename	Size	Rev	Date	Author
Analysis.java	5.1 KB	r22	Jan 10, 2008	hunkim
CombineData.java	1.4 KB	r88	Mar 18, 2008	hunkim
EntityData.java	1.5 KB	r22	Jan 10, 2008	hunkim
GetMethodNames.java	5.7 KB	r88	Mar 18, 2008	hunkim
HandleStackTrace.java	6.1 KB	r88	Mar 18, 2008	hunkim
HandleStackTraceRemoveDuplicates.java	8.6 KB	r19	Jan 10, 2008	shay.artzi
ReadEntityData.java	3.8 KB	r14	Jan 01, 2008	hunkim

Browse source code to review

Project Home | Issues | Source | Administer

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

Source path: svn/ trunk/ java/ src/ com/ googlecode/ crashma/ ReadEntityData.java ‹r11 **r249**

```
1  package com.googlecode.crashma;
2
3  import java.io.BufferedWriter;
4  import java.io.FileInputStream;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.io.Writer;
8  import java.util.Hashtable;
9  import java.util.Set;
10 import java.util.zip.GZIPInputStream;
11
12 import au.com.bytecode.opencsv.CSVReader;
13
14 import com.googlecode.crashma.util.FileUtil;
15
16 public class ReadEntityData {
17         private Hashtable<String, EntityData> data = new Hashtable<String, Ent
18
19         /**
20          * Check if the key is contained
21          *
22          * @param key
23          * @return
24          */
25         public boolean contain(String key) {
26                 return data.get(key) != null;
27         }
28
29         public EntityData getEntityData(String key) {
30                 return data.get(key);
31         }
32
33         public Set<String> getDataKeySet() {
34                 return data.keySet();
35         }
36
```

Hide details

Change log

r14 by hunkim on Jan 01, 2008 Diff
Updated data

Go to: ...code/crashma/ReadEntityData.java

Hide published comments
Double click a line to add a comment

Older revisions

r11 by hunkim on Dec 30, 2007 Diff
r9 by hunkim on Dec 30, 2007 Diff
r7 by hunkim on Dec 29, 2007 Diff
All revisions of this file

File info

Size: 3842 bytes, 173 lines
View raw file

In-line review

Project Home | Issues | Source | Administer

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

Source path: svn/ trunk/ java/ src/ com/ googlecode/ crashma/ ReadEntityData.java ‹r11 **r249**

```
1  package com.googlecode.crashma;
2
3  import java.io.BufferedWriter;
4  import java.io.FileInputStream;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.io.Writer;
8  import java.util.Hashtable;
9  import java.util.Set;
10 import java.util.zip.GZIPInputStream;
11
12 import au.com.bytecode.opencsv.CSVReader;
13
14 import com.googlecode.crashma.util.FileUtil;
15
16 public class ReadEntityData {
17         private Hashtable<String, EntityData> data = new Hashtable<String, Ent
18
19         /**
20          * Check if the key is contained
21          *
22          * @param key
23          * @return
24          */
25         public boolean contain(String key) {
```

Draft comment:

Why do we need this key?

Save Cancel

```
26                 return data.get(key) != null;
```

Hide details

Change log

r14 by hunkim on Jan 01, 2008 Diff

Updated data

Go to: ...code/crashma/ReadEntityData.java

Hide published comments

Double click a line to add a comment

Older revisions

- r11 by hunkim on Dec 30, 2007 Diff
- r9 by hunkim on Dec 30, 2007 Diff
- r7 by hunkim on Dec 29, 2007 Diff

All revisions of this file

File info

Size: 3842 bytes, 173 lines

View raw file

In-line review

Project Home | Issues | Source | Administer

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

Source path: svn/ trunk/ java/ src/ com/ googlecode/ crashma/ ReadEntityData.java ‹r11 r249

```
1  package com.googlecode.crashma;
2
3  import java.io.BufferedWriter;
4  import java.io.FileInputStream;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.io.Writer;
8  import java.util.Hashtable;
9  import java.util.Set;
10 import java.util.zip.GZIPInputStream;
11
12 import au.com.bytecode.opencsv.CSVReader;
13
14 import com.googlecode.crashma.util.FileUtil;
15
16 public class ReadEntityData {
17         private Hashtable<String, EntityData> data = new Hashtable<String, Ent
18
19         /**
20          * Check if the key is contained
21          *
22          * @param key
23          * @return
24          */
25         public boolean contain(String key) {
```

Draft comment: Edit

Why do we need this key?

```
26                 return data.get(key) != null;
27         }
28
29         public EntityData getEntityData(String key) {
30                 return data.get(key);
31         }
```

Hide details

Change log

r14 by hunkim on Jan 01, 2008 Diff

Updated data

Go to: ...code/crashma/ReadEntityData.java

Hide published comments

Publish your comments

Older revisions

r11 by hunkim on Dec 30, 2007 Diff

r9 by hunkim on Dec 30, 2007 Diff

r7 by hunkim on Dec 29, 2007 Diff

All revisions of this file

File info

Size: 3842 bytes, 173 lines

View raw file

Diff

Review in the diff

Publish!

Project Home | Issues | Source | Administer

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

Changes to /trunk/java/src/com/googlecode/crashma/ReadEntityData.java r11 vs. r14 Edit ‹r11 r14

Code review of r14 Hide published comments Go to: ...code/crashma/ReadEntityData.java Publish your comments

/trunk/java/src/com/googlecode/crashma/ReadEntityData.java r11

```
1  package com.googlecode.crashma;
2
3  import java.io.BufferedWriter;
4  import java.io.FileInputStream;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.io.Writer;
8  import java.util.Hashtable;

9  import java.util.zip.GZIPInputStream;
10
11 import au.com.bytecode.opencsv.CSVReader;
12
13 import com.googlecode.crashma.util.FileUtil;
14
15 public class ReadEntityData {
16         Hashtable<String, EntityData> data = new Hashtable<String,
   EntityData>();
```

Draft comment:

Hmm?

Save Cancel

/trunk/java/src/com/googlecode/crashma/ReadEntity

```
1  package com.googlecode.crashma;
2
3  import java.io.BufferedWriter;
4  import java.io.FileInputStream;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.io.Writer;
8  import java.util.Hashtable;
9  import java.util.Set;
10 import java.util.zip.GZIPInputStream;
11
12 import au.com.bytecode.opencsv.CSVReader;
13
14 import com.googlecode.crashma.util.FileUtil;
15
16 public class ReadEntityData {
17         private Hashtable<String, EntityData> data
   Hashtable<String, EntityData>();
```

Publishing review (others can see)

Your review

General comment

Rewrite the code please (This is a test)

Review score

- ○ Positive: I support this change
- ○ Neutral: I am ambivalent about this change
- ◉ Negative: I oppose this change, and I am willing to explain why.

Line-by-line comments

/trunk/java/src/com/googlecode/crashma/ReadEntityData.java r11 line 16:

```
16:     Hashtable<String, EntityData> data = new Hashtable<String, EntityData>();
```

Hmm?

/trunk/java/src/com/googlecode/crashma/ReadEntityData.java r14 line 25:

```
25:     public boolean contain(String key) {
```

Why do we need this key?

Submit

Review results in the changes

Project Home | Issues | Source | Administer

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

Committed Changes 249 - 225 of 249 Older ›

Rev	Scores	Commit log message	Date	Author
r249	-1	Submitted copy added under dir Submission.	Sep 07, 2009	r.premraj
r248		Second submission.	Sep 07, 2009	r.premraj
r247		First submission.	Sep 07, 2009	r.premraj
r246		Updated author order consulting with Sung and Tom.	Sep 06, 2009	r.premraj
r245		Release Candidate 1.	Sep 06, 2009	r.premraj
r244		Sung's and Shiv's comments incorporated.	Sep 05, 2009	r.premraj
r243		Section 6 on ReCrash updated.	Sep 05, 2009	r.premraj
r242		Most of nico's changes factored in.	Sep 04, 2009	r.premraj
r241		Removed the naive bayes stuff and updated bayes	Sep 04, 2009	sshivaji

Detailed review information

Project Home | Issues | Source | Administer

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

Revision: r249

Author: r.premraj
Date: Sep 07, 2009

Review scores:

-1

Negative:
hunkim

Log message

```
Submitted copy added under dir Submission.
```

Affected files expand all collapse all

⊞Add /trunk/paper/FSE/Submission diff

⊞Add /trunk/paper/FSE/Submission/Kim-Predicting_Method_Crashes.pdf diff

Start a code review ›
Add comments to files in th
them with the author and c
Learn more

Comment by hunkim, Today (19 minutes ago)
⊞ 2 line-by-line comments

Comment by hunkim, Today (16 minutes ago)

```
Good
```

⊞ 1 line-by-line comment

Comment by hunkim, Today (moments ago)

```
Rewrite the code please (This is a test)
```

⊞ 2 line-by-line comments

Summary

- Code Review
  - What, Why, and How
- Code review in practice
  - Mondrian
- Code review in action (googlecode)
- Review your (project) code!

Defensive Programming

Be a skeptical, very skeptical

1. Defensive Programming

It's like "Defensive Driving"

- You *expect* "unexpected" problems to crop up
  - her turn signal is on, but car doesn't turn
  - you brake, but he doesn't slow down behind

- Some areas for "suspicious" programming:
  - input data, not as specified
    - function arguments, file contents, human input
  - module behaviour, not as specified

Three General Strategies

- Stop the world ==> Use assertions;
- Manually handle errors ==> Exception systems;
- Record what happens ==> Logging

No idea how to continue!

- Check for cases which “can't occur” just in case!
- Check for extreme values - negative values, huge values, etc.

```
    if ( grade < 0 || grade > 100 ) /* impossible grades!
*/

        letter = '?';

    else if ( grade >= 80 )

        letter = 'A';
```

Assertions are your friend!

Java Example of an Assertion

```
assert denominator != 0
       : "denominator is unexpectedly equal to 0.";
```

```
if (i % 3 == 0) {
    ...
} else if (i % 3 == 1) {
    ...
} else {
    assert i % 3 == 2 : i;
    ...
}
```

What to Check For

- Check pre- and post-conditions
  - before and after a critical piece of code, check to see if the necessary pre- and post-conditions (value of variables, etc.) hold true
  - Famous divide by zero error:

    On the USS Yorktown (guided-missile cruiser), a crew member entered a zero, which resulted in a divide by zero, the error cascaded and shut down the propulsion system and the ship drifted for a couple of hours.

Java Example

```
void setRefreshRate(int rate) {
    // Enforce specified precondition in public method
    if (rate <= 0 || rate > MAX_REFRESH_RATE)
        throw new IllegalArgumentException("Illegal rate: " +
 rate);

    setRefreshInterval(1000/rate);
}

void setRefreshInterval(int interval) {
    // Confirm adherence to precondition in nonpublic method
    assert interval > 0 && interval <= 1000/
MAX_REFRESH_RATE : interval;

    ... // Set the refresh interval
}
```

Error Handling Strategy

- Defensive programming uncovers "errors"
  - **Now, what are you going to do about them?**

```
if (error happened) {
what code goes here? }
```

- Best to have thought-out **strategy** regarding errors
  - Don't leave up to individual coder's judgment
  - Don't handle on ad hoc basis
- Otherwise, code quality (re error checking & handling) will vary widely across system

Striking a Balance

- *One extreme:*
  - Check for every conceivable (and inconceivable) error condition
  - Downside:
    - Costs time/money to write that code
    - Error checking/handling code must be tested, too!
    - Takes up memory with code never likely to run
- *Other extreme:* "nothing will go wrong!"
    - Fragile system, late discovery of hard-to-locate bugs

Recognize *Two Kinds* of Errors

1. Problems with external data/conditions
   - User/operator should be informed; don't crash!
2. Erroneous internal usage, rare conditions
   - Module A calls module B with bad args
   - Out of memory/disk space
   - Unexpected error return/result from library function call
   - *Testing/debug phase:*
     - Let programmers know before crashing
   - *Production phase:*
     - Keep running, recover gracefully if possible

Recognize *Two Severities*

1. "Fatal" errors:
   - Meaningless to continue execution
     - e.g., out of memory, nonsensical value of critical variable
   - Best to abort, but inform if possible
     - Otherwise, how will anyone find the bug?
2. "Nonfatal" errors:
   - Recovery possible, may want to inform

After Detection, Who Handles?

- General principle:
  - **Handle** errors in context, in the same place where you **detected** them

- You're aware an error may occur, might as well write the code to handle it *while you know what the problem is*

Benefits of Local Error Handling

- Avoids letting invalid state **propagate** elsewhere through other modules
- Self-documenting:
  - Clear idea what the (usually complex) error handling code is supposed to be there for
- Keeps the complex code for particular error **contained and localized** instead of smeared throughout the system

When to Reflect Errors Upward

- **Utility** packages are exception!
  - Can detect errors, but may not know how to handle in way acceptable to application
    - E.g., utility printing error message usually not appropriate
    - App. may have special use of stdout/stderr streams, may be running in GUI
  - Best: reflect error status up to caller
    - *Caller* applies "handle error in context" principle to suit nature of application

Record what happened

- Java logging utility (Reference only)

ConsoleHandler This `Handler` publishes log records to `System.err`.

ErrorManager ErrorManager objects can be attached to Handlers to process any error that occur on a Handler during Logging.

FileHandler Simple file logging `Handler`.

Level The Level class defines a set of standard logging levels that can be used to control logging output.

Logger A Logger object is used to log messages for a specific system or application component.

LogManager There is a single global LogManager object that is used to maintain a set of shared state about Loggers and log services.

LogRecord LogRecord objects are used to pass logging requests between the logging framework and individual log Handlers.

MemoryHandler `Handler` that buffers requests in a circular buffer in memory.

SimpleFormatter Print a brief summary of the LogRecord in a human readable format.

SocketHandler Simple network logging `Handler`.

StreamHandler Stream based logging `Handler`.

XMLFormatter Format a LogRecord into a standard XML format.

Logging levels
(Reference Only)

The Level class defines a set of standard logging levels that can be used to control logging output. The logging Level objects are ordered and are specified by ordered integers. Enabling logging at a given level also enables logging at all higher levels.

Clients should normally use the predefined Level constants such as Level.SEVERE.

The levels in descending order are:

- SEVERE (highest value)
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST (lowest value)

In addition there is a level OFF that can be used to turn off logging, and a level ALL that can be used to enable logging of all messages.

On to Testing!

- Defensive programming = some degree of self-testing code
  - Tests itself at development time
  - Continues testing in production
- Still need to methodically apply tests to modules, system as whole
  - Practices very well developed in SW engr.