Good Comments

EJUG – Good Code Lightning Talks

March 23, 2011

Warren Blanchet – wskb.ca

Why Comments?

Maintenance

- 10 Generations of programmers work on your code before it is rewritten
- 50% - 60% of change time is spent on understanding the existing code
- You have to communicate with these people

Citations:

(Both via Steve McConnell. 2004. *Code Complete*, *Second Edition*. Microsoft Press.)

Richard A. Thomas. 1984. "Using comments to aid program maintenance." *BYTE*, May, 415-22

G. Parikh, N. Zvegintzov. 1983. *Tutorial on software maintenance*. IEEE Computer Society Press.

Inaccuracy vs. Distance

Distance

	Logic	Comments	Req's	Unit Test
Storage	Source File A	Source File A	Folder of Word Files	Source File ATest
Author	Coder	Coder	Multiple	Coder
Authoring Tool	IDE	IDE	Word	IDE
Consumer	Coder	Coder	Multiple	Coder

What do I mean by distance?

Several concerns: Where is it stored? Who writes it? With what? For whom?

Some examples: Program Logic, Source Code Comments, Requirements, Unit Test Logic

Distance

	Logic	Comments	Req's	Unit Test
Storage	Source File A	Source File A	Folder of Word Files	Source File ATest
Author	Coder	Coder	Multiple	Coder
Authoring Tool	IDE	IDE	Word	IDE
Consumer	Coder	Coder	Multiple	Coder

Hey, look, these are pretty close.

Comments are good for documenting program logic

Distance

	Logic	Comments	Req's	Unit Test
Storage	Source File A	Source File A	Folder of Word Files	Source File ATest
Author	Coder	Coder	Multiple	Coder
Authoring Tool	IDE	IDE	Word	IDE
Consumer	Coder	Coder	Multiple	Coder

This is good too, but that's another presentation.

Distance

	Logic	Comments	Req's	Unit Test
Storage	Source File A	Source File A	Folder of Word Files	Source File ATest
Author	Coder	Coder	Multiple	Coder
Authoring Tool	IDE	IDE	Word	IDE
Consumer	Coder	Coder	Multiple	Coder

This doesn't match well

Comments are not great for requirements

Comment Taxonomy

- Marker
- Summary
- Explanation: intent + rationale
- API = Summary + reuse manual

Marker

License and copyright

- Audience is coder, sometimes
- Stick in header - ignorable when not relevant

LicenseAndCopyright.java

```
/*
 * Copyright (C) 2011 by Warren S.K. Blanchet
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

package ca.wskb.example;

public class LicenseAndCopyright {
```

LicenseAndCopyright.java

```
* Copyright (C) 2011 by Warren S.K. Blanchet

package ca.wskb.example;

public class LicenseAndCopyright {
```

Version Control

```
//  public void foo() {
//
//      System.out.println(System.getProperty( "otherproperty" ));
//  }
}
```

I might need this later

Version Control

SourceControl.java

```
/*
 * This file does nothing
 *
 * Date         Who   Change
 * ------------------------------------------------------------
 * Mar 2009     wb    Added extra nothing
 * 2010-02-29   wsb   Tweaked slightly
 * 2011-03-11   wsb   Added nothing for customer request #3324314
 */
```

History is nice to have

Version Control

```
public void bar() {

    // #4323432 - 2011-02-29 - wb - added
    // #MG34DF3 - 2011-03-11 - wsb - changed from bad value
    String property = System.getProperty( "someproperty" );

    // #MG34DF3 - 2011-03-11 - wsb - Changed from err to out
    System.out.println(property);
}
```

Good to know what changed

Version Control

- Did you know?
  - There are tools for version control
  - Many are free
- Advantages
  - Up to date
  - Accurate
  - Better tool support
  - Only visible when relevant

Task

```
public Date calculateImportantDateThing() {
    // TODO: handle leap year
```

Tasks

4 items

Description	Resource
TODO : implement me!	Task.java
TODO : implement me!	Task.java
TODO : handle leap year	Task.java

Useful:

-Tool support: highlighting, lists

Task

```
public void doImportantThing() {

    // TODO: implement me!
}
```

```
public int calculateImportantThing() {

    throw new UnsupportedOperationException("not yet implemented");
}
```

Todo advantages:

-in tool lists

Exception advantages:

-Fails tests

-Can be used when returning value

Task

```
public int deriveImportantValue() {

    // TODO: implement me!
    throw new UnsupportedOperationException("not yet implemented");
}
```

Combine approaches

Task

```
public Color getBackgroundColor() {

    // TODO: figure out what this should be
    return Color.WHITE;
}
```

Bad: Requirements not program logic

But it's not for requirements

How is this todo to get done?

Wish it were BASIC

```
for( Object item : items  ) {
    if( item != null ) {
        bar(item);
    } // end if item is null
} // end for
```

See what distance does?

Wish it were BASIC

```
for( Object item : items  ) {
    if( item != null ) {
        bar(item);
    } // end if item is null
} // end for
```

Indentation and matching braces already do the job

Tools

OK

```
public class Foo {

    private int bar = 12; // NOPMD - used by native method
}
```

Better

```
public class Foo {

    @SuppressWarnings( "unused" ) // used by native method
    private int bar = 12;
}
```

Annotations make it easier on tools and coders: it's harder to get it wrong

Summary

```
// multiply by two
sleepTimeRange = sleepTimeRange * 2;
// ensure no more than max
sleepTimeRange = Math.max( sleepTimeRange, MAX_SLEEP_TIME_RANGE );
```

Not concise - adds no value

```
// exponential backoff
sleepTimeRange = sleepTimeRange * 2;
sleepTimeRange = Math.max( sleepTimeRange, MAX_SLEEP_TIME_RANGE );
```

Better

```
sleepTimeRange = exponentialBackoff( sleepTimeRange );
```

Best - in the code itself

Explanation

```java
private int exponentialBackoff( int sleepTimeRange ) {

    sleepTimeRange = sleepTimeRange * 2;
    sleepTimeRange = Math.max( sleepTimeRange, MAX_SLEEP_TIME_RANGE );
    return sleepTimeRange;
}
```

Simple code - no explanation necessary

```java
private int exponentialBackoff( int sleepTimeRange ) {

    // multiply by two
    sleepTimeRange = sleepTimeRange << 1;
    sleepTimeRange = Math.max( sleepTimeRange, MAX_SLEEP_TIME_RANGE );
    return sleepTimeRange;
}
```

Now strange: explain intent

```java
private int exponentialBackoff( int sleepTimeRange ) {

    /*
     * Multiply by two using left shift;
     * increases performance under NicheCorp's java compiler 7.2
     */
    sleepTimeRange = sleepTimeRange << 1;

    sleepTimeRange = Math.max( sleepTimeRange, MAX_SLEEP_TIME_RANGE );
    return sleepTimeRange;
}
```

Better: explain intent and rationale

API: JavaDoc

```java
/**
 * Returns the new range for sleep time between retries, as calculated
 * using exponential backoff. The next attempt should be retried after a
 * random wait time bounded by the return value.
 *
 * @param sleepTimeRange the existing range for sleep time
 *
 * @return the new range for sleep time
 */
public int exponentialBackoff( int sleepTimeRange ) {
```

Summary + how to use

```java
/**
 * Returns the new range for sleep time between retries, as calculated
 * using exponential backoff. The next attempt should be retried after a
 * random wait time bounded by the return value.
 * <p>
 * This method is thread-safe.
 *
 * @param sleepTimeRange the existing range for sleep time
 *
 * @return the new range for sleep time
 *
 * @throws IllegalArgumentException if the passed value is negative
 */
public int exponentialBackoff( int sleepTimeRange ) {

    if( sleepTimeRange < 0 ) {

        throw new IllegalArgumentException("sleepTimeRange must be non-negative");
    }
```

Also include:

-thread safety

-Thrown runtime exceptions (used for validation, etc.)

Miscellaneous

- Avoid net emptiness
- Avoid @author - version control's job
- Use @link

Net emptiness: when the comment doesn't say anything more than the method signature:

-empty @param

-empty @return

-restatement of method name

-Example: getters/setters

Comment Presentation

```
/* ***************************************
 * How can I make this harder to         *
 * maintain? I'm not sure. Maybe some    *
 * ASCII word art?                       *
 *                                       *
 *  _     _    _____   _____   _____     *
 *  | \   | |  | ___ | |__ __| | ____|  *
 *  |  \  | |  ||   ||   | |   ||___    *
 *  | |\\ | |  ||   ||   | |   | __|    *
 *  | | \\| |  ||___||   | |   ||____   *
 *  |_|  \__|  |_____|   |_|   |_____|  *
 *                                       *
 * *************************************** */
```

Don’t do this: Presentation for presentation’s sake

```
//
// I'm a summary comment about the next section
//

// I'm an explanation comment about the next line or block
blah("blah");
YADA.yada();

//
// next section
//
```

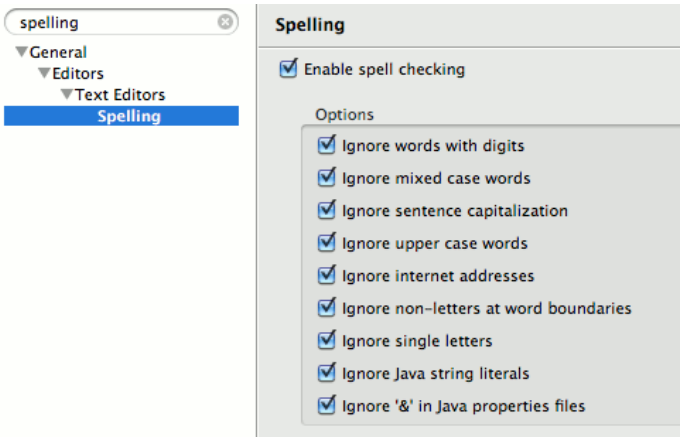
One strategy: use presentation to distinguish between different types of comments

```
//::// I'm a summary comment about the next section

/*
 * I'm a long explanation comment about the next line
 * or block
 */
blah("blah");
YADA.yada();

//::// next section
```

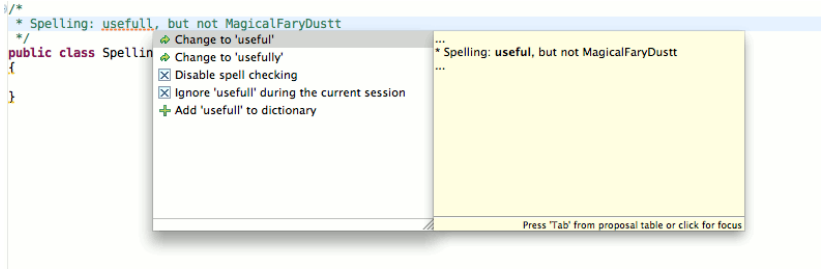
Same strategy, different implementation

Tips for Eclipse users

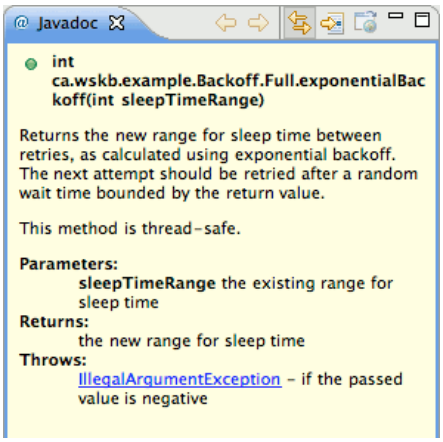
Spelling


spelling
General
Editors
Text Editors
Spelling
Spelling
Enable spell checking
Options
Ignore words with digits
Ignore mixed case words
Ignore sentence capitalization
Ignore upper case words
Ignore internet addresses
Ignore non-letters at word boundaries
Ignore single letters
Ignore Java string literals
Ignore '&' in Java properties files

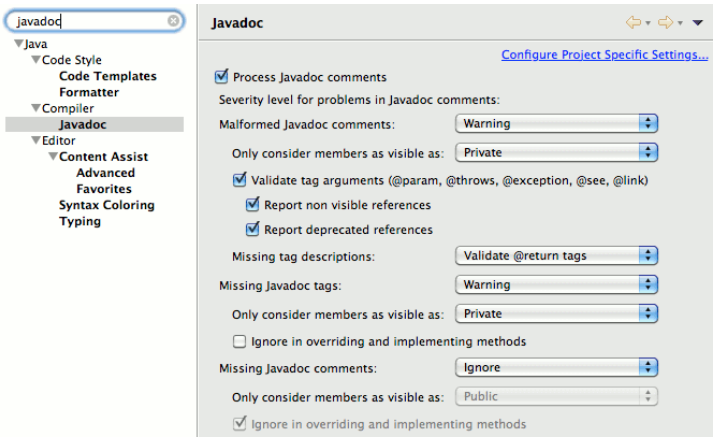
Spelling


/*
* Spelling: usefull, but not MagicalFaryDustt
*/
public class Spellin
{
}
Change to 'useful'
Change to 'usefully'
Disable spell checking
Ignore 'usefull' during the current session
Add 'usefull' to dictionary
...
* Spelling: useful, but not MagicalFaryDustt
...
Press 'Tab' from proposal table or click for focus

JavaDoc view



JavaDoc validation



JavaDoc validation

```
/**
 * This class is useful. See {@link #primary()}.
 */
public class JavaDocValidation {

    /**
     * This method does something useful
     * @param input the input
     * @param configVal the configuration value
     */
    public String primary( String input ) throws IOException {
```

I am always right,
except when I am not

Comments are communication

- Read books — I did:
  - Code Complete, Second Edition
  - Effective Java, Second Edition
- Hear from speakers
- Talk to colleagues
- Listen to customers

Good Comments

- “Good” is subjective and context-sensitive
- Expose yourself to ideas - no obligation
- Know the rationale behind your practices
- Be able to explain them

Good Comments

EJUG – Good Code Lightning Talks

March 23, 2011

Warren Blanchet – wskb.ca