Code Reviews and Software Engineering Best Practices

CUNY Tech Prep

Excellent resource on this topic!

http://web.mit.edu/6.005/www/fa16/classes/04-code-review

What is the purpose of a code review?

Two main purposes

Improve the code

reduce bugs; catch inefficiencies;
maintain clarity and consistency

Improve the programmer

learn new techniques;
teach others your techniques;
communication

What are we looking for in a code review?

Things to look out for

Does the code meet the specification (spec)

Code clarity, consistency, and style

Spotting bugs and bad practices

Let’s review some code

```
public static int doy(int m, int dom, int y) {
   if (m == 2) {
       dom += 31;
   } else if (m == 3) {
       dom += 59;
   } else if (m == 4) {
       dom += 90;
   } else if (m == 5) {
       dom += 31 + 28 + 31 + 30;
   } else if (m == 6) {
       dom += 31 + 28 + 31 + 30 + 31;
   } else if (m == 7) {
       dom += 31 + 28 + 31 + 30 + 31 + 30;
   } else if (m == 8) {
       dom += 31 + 28 + 31 + 30 + 31 + 30 + 31;
   } else if (m == 9) {
       dom += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31;
   } else if (m == 10) {
       dom += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30;
   } else if (m == 11) {
       dom += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31;
   } else if (m == 12) {
       dom += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 31;
   }
   return dom;
}
```

```
public static int doy(int m, int dom, int y) {
   if (m == 2) {
      dom += 31;
   } else if (m == 3) {
      dom += 59;
   } else if (m == 4) {
      dom += 90;
   } else if (m == 5) {
      dom += 31 + 28 + 31 + 30;
   } else if (m == 6) {
      dom += 31 + 28 + 31 + 30 + 31;
   } else if (m == 7) {
      dom += 31 + 28 + 31 + 30 + 31 + 30;
   } else if (m == 8) {
      dom += 31 + 28 + 31 + 30 + 31 + 30 + 31;
   } else if (m == 9) {
      dom += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31;
   } else if (m == 10) {
      dom += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30;
   } else if (m == 11) {
      dom += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31;
   } else if (m == 12) {
      dom += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 31;
   }
   return dom;
}
```

WTH!!!

Question…
What is the spec?

```
/**
* Compute the day of the year given the date (month, day, year).
* @param month month of the year where January=1 and December=12.
* @param dayOfMonth day of the month. requires value 1 <= dayOfMonth <= 31
* @param year year to determine whether it is a leap year
* @return day of the current year, valid values between 1-365, 366 on leap
years
*/
public static int dayOfYear(int month, int dayOfMonth, int year) {
  . . .
}
```

OK… let’s try again

```
public static int dayOfYear(int month, int dayOfMonth, int year) {
    if (month == 2) {
        dayOfMonth += 31;
    } else if (month == 3) {
        dayOfMonth += 59;
    } else if (month == 4) {
        dayOfMonth += 90;
    } else if (month == 5) {
        dayOfMonth += 31 + 28 + 31 + 30;
    } else if (month == 6) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31;
    } else if (month == 7) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30;
    } else if (month == 8) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31;
    } else if (month == 9) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31;
    } else if (month == 10) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30;
    } else if (month == 11) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31;
    } else if (month == 12) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 31;
    }
    return dayOfMonth;
}
```

The good

The bad

The good

Good Names

The bad

A lot of repetition

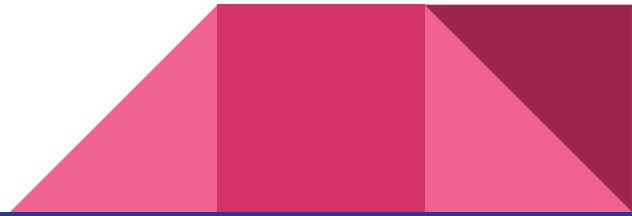
Many **Magic Numbers**

dayOfMonth is reused

Let’s do better

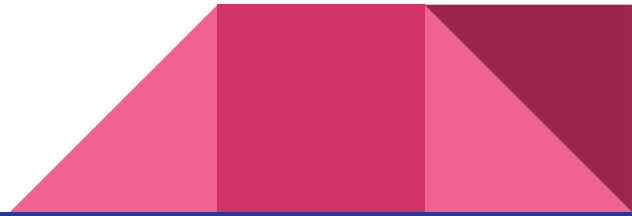
```
public static int dayOfYear(int month, int dayOfMonth, int year) {
    int[] monthLengths = new int[] { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    int result = 0;

    for(int i=1; i < month; i++) {
        result += monthLengths[i-1];
    }

    result += dayOfMonth;

    return result;
}
```

What are other things we can do?



What are other things we can do?

- Fail fast!
  - Check the month and day for validity
- Add comments where needed



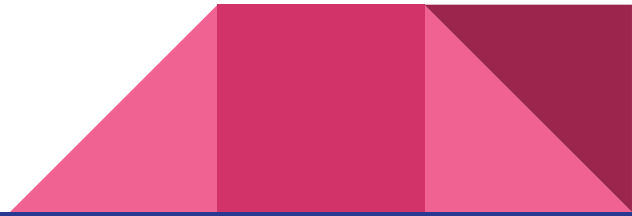
More Examples

```
public static boolean leap(int y) {
    String tmp = String.valueOf(y);
    if (tmp.charAt(2) == '1' || tmp.charAt(2) == '3'
            || tmp.charAt(2) == 5 || tmp.charAt(2) == '7' || tmp.charAt(2) == '9') {
        if (tmp.charAt(3)=='2'||tmp.charAt(3)=='6') return true;
        else
            return false;
    } else {
        if (tmp.charAt(2) == '0' && tmp.charAt(3) == '0') {
            return false;
        }
        if (tmp.charAt(3)=='0'||tmp.charAt(3)=='4'||tmp.charAt(3)=='8')return true;
    }
    return false;
}
```

```
public static int LONG_WORD_LENGTH = 5;
public static String longestWord;


public static void countLongWords(List<String> words) {
    int n = 0;
    longestWord = "";
    for (String word: words) {
        if (word.length() > LONG_WORD_LENGTH) ++n;
        if (word.length() > longestWord.length()) longestWord = word;
    }
    System.out.println(n);
}
```

Summary

- DRY - Don’t Repeat Yourself
- Comment where needed
- Fail fast
- Avoid *magic* numbers
- Use good names
- One purpose for each variable
- No global variables
- Return results, don’t print them
- Use whitespace for readability

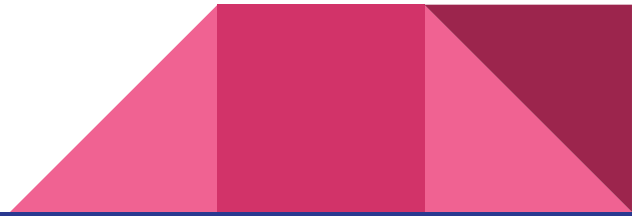


Reviewing the whole project

Look for:

README and Docs

- README should explain
  - Project Dependencies
  - Installation Instructions
  - Basic Usage of codebase
- Docs should document
  - API’s
  - Tools
  - Advanced usage



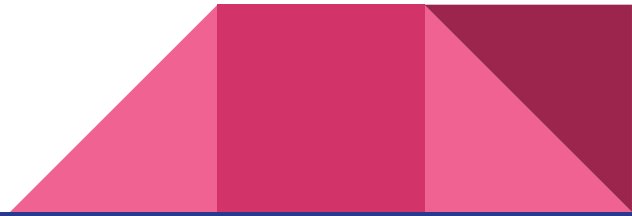
Look for:

Directory Structure

- Create directories for different components
- Follow naming conventions (singular vs plural names)

Filenames

- All file names should be consistent in casing and plurality
- File Extensions should be consistent



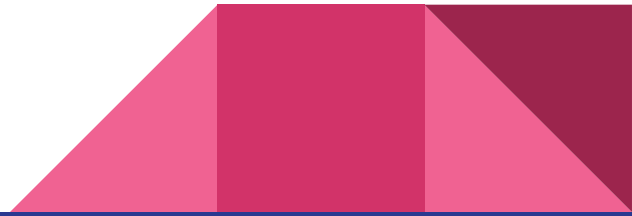
Look for:

Naming consistency

- Are all functions, classes, models named consistently
- Are all controllers and action functions named consistently
- Are CRUD and RESTful functions named in a consistent way

Code Modularity

- Only one module per file
- The code within a module is **Highly Cohesive**
- The code across modules has **Loose Coupling**



Look for:

Tests

- Each module is tested
- Multiple test cases
- Edge test cases

- Replication of bugs (unexpected use cases) in tests

Outcomes

Refactored code

Clearer codebase

Better code modularity and reusability

Catch errors and inefficient code

Naming examples

Which of these names are consistent?

application_controller.js

usersController.js

postController.JS

CommentsController.js

IBM article on code reviews
https://www.ibm.com/developerworks/rational/library/11-proven-practices-for-peer-review/

Code Reviews: Just Do It
https://blog.codinghorror.com/code-reviews-just-do-it/

What to look for in a code review
https://blog.jetbrains.com/upsource/2015/07/23/what-to-look-for-in-a-code-review/