CS 4500
Software Development

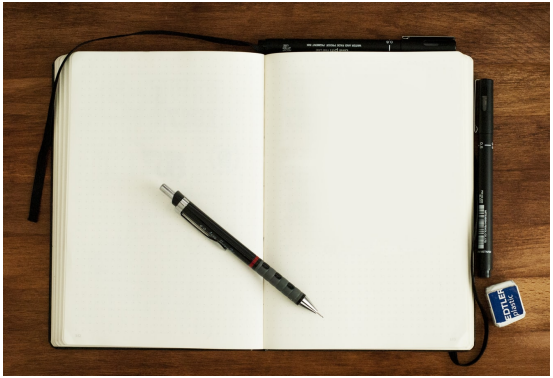
[Code, pt 1]

Ferdinand Vesely
(partially based on Clean Code by Robert C. Martin)

September 17, 2019

Lab / Log Book

- Get a Lab Book
- Small notebook
- Notes about
  - assignment and project work
  - group/pair work
  - meetings

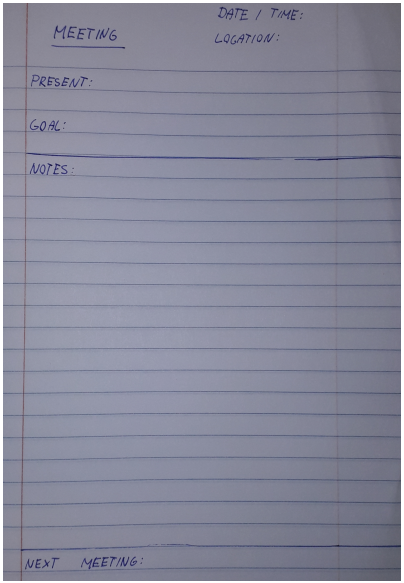


Lab / Log Book

- Make notes about group
- 4 types of pages:
  1. Group info page: nickname, names, cell phones, emails or social network handles
  2. Meeting notes page
  3. Weekly project cover page: title + time estimate + notes
  4. Weekly project conclusion page: time needed, reflection

Meeting Page

1. Date/time
2. Location
3. Members present
4. Goal
5. Notes
6. Duration
7. Next meeting



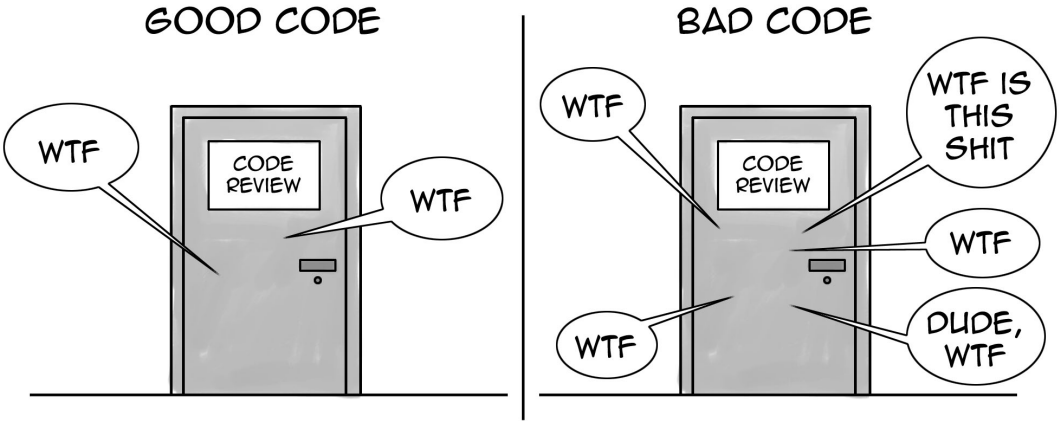
Meetings

If somebody doesn't show up:

- write down steps taken to reach them
- if they excused themselves, write down reasons

About Code

Good Code vs. Bad Code



Code

Can we get rid of code?
In a sense…

- Represents the *details* of the requirements
- Specifying requirements in such detail that a machine can execute them = programming
- Such a specification *is* code
- Good code matters and will matter

What often happens

- Write a promising prototype

What often happens

- Write a promising prototype
- Turn into product, roll it out

What often happens

- Write a promising prototype
- Turn into product, roll it out
- No time to think about code quality

What often happens

- Write a promising prototype
- Turn into product, roll it out
- No time to think about code quality
- Product is succesfull → feature requests

What often happens

- Write a promising prototype
- Turn into product, roll it out
- No time to think about code quality
- Product is succesfull → feature requests
  - Also: bugs get discovered more frequently

What often happens

- Write a promising prototype
- Turn into product, roll it out
- No time to think about code quality
- Product is succesfull → feature requests
  - Also: bugs get discovered more frequently
- Pile new code on top of old one

What often happens

- Write a promising prototype
- Turn into product, roll it out
- No time to think about code quality
- Product is succesfull → feature requests
  - Also: bugs get discovered more frequently
- Pile new code on top of old one
- Code becomes unmaintainable

What often happens

- Write a promising prototype
- Turn into product, roll it out
- No time to think about code quality
- Product is succesfull → feature requests
  - Also: bugs get discovered more frequently
- Pile new code on top of old one
- Code becomes unmaintainable
- Longer and longer release cycles

What often happens

- Write a promising prototype
- Turn into product, roll it out
- No time to think about code quality
- Product is succesfull → feature requests
  - Also: bugs get discovered more frequently
- Pile new code on top of old one
- Code becomes unmaintainable
- Longer and longer release cycles
- New bugs introduced, old bugs not fixed

What often happens

- Write a promising prototype
- Turn into product, roll it out
- No time to think about code quality
- Product is succesfull → feature requests
  - Also: bugs get discovered more frequently
- Pile new code on top of old one
- Code becomes unmaintainable
- Longer and longer release cycles
- New bugs introduced, old bugs not fixed
- Users abandon product

What is good code?

- Clean code

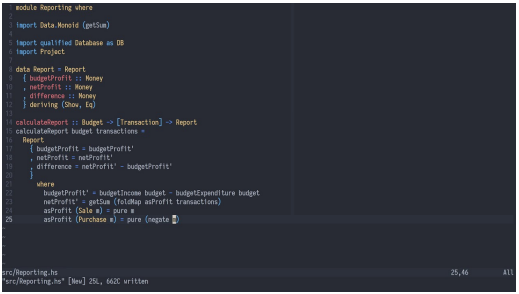
  You know you are working on clean code when each routine you read turns out to be pretty much what you expected. (Ward Cunningham)

What is good code?

- Reads like well-written prose
- Never obscures designer's intent
- Clear abstractions & straightforward lines of control
- Easy for other people to read and enhance. Literate

Good Code

- Ratio of reading vs. writing code is high
- Making it easy to read makes it easier to write



Names

Names

- In programming: most of what we read or write are names:
  - variables
  - methods/functions
  - classes
  - packages
  - constants
  - macros
  - …
- Names are everywhere!
- We should be motivated to choose them well
- Principles

Indicate Intent

- A name should answer all big questions
- Shouldn't need a comment
- Comment doesn't travel with the variable name
- Even though tools support fast lookups

Indicate Intent

- Consider:

```
int d;    // elapsed time in days
```

- Fine … but several lines down, something like:

```
d = Scanner.nextInt();

if (d > dMax) {
  ...
}
else {
  ...
}
```

Indicate Intent

- d on its own tells me nothing
- How about:

```
int elapsedTimeInDays;
int daysSinceCreation;
int daysSinceModification;
int fileAgeInDays;
```

Indicate Intent

```
public List<int[]> getThem() {
  List<int[]> list1 = new ArrayList<int[]>();

  for (int[] x : theList)
    if (x[0] == 4)
      list1.add(x);

  return list1;
}
```

- Not *explicit*

Indicate Intent

```
public List<int[]> getFlaggedCells() {
  List<int[]> flaggedCells = new ArrayList<int[]>();

  for (int[] cell : gameBoard)
    if (cell[STATUS_VALUE] == FLAGGED)
      flaggedCells.add(cell);

  return flaggedCells;
}
```

Indicate Intent

- Even better: name types!

```
public List<Cell> getFlaggedCells() {
  List<Cell> flaggedCells = new ArrayList<Cell>();

  for (Cell cell : gameBoard)
    if (cell.isFlagged())
      flaggedCells.add(cell);

  return flaggedCells;
}
```

Meaningful Distinctions

For the purpose of distinguishing:

- Avoid number series

```
public static void copy(char a1[], char a2[]) {
  for (int i = 0; i < a1.length; i++)
    a2[i] = a1[i];
}
```

Meaningful Distinctions

For the purpose of distinguishing:

- Avoid number series

```
public static void copy(char a1[], char a2[]) {
  for (int i = 0; i < a1.length; i++)
    a2[i] = a1[i];
}
```

vs.

```
public static void copy(char source[], char destination[]) {
  for (int i = 0; i < source.length; i++)
    destination[i] = source[i];
}
```

Meaningful Distinctions

For the purpose of distinguishing:

- Avoid “noise words”, such as `info`, `data`:

```
account
accountInfo
accountData
```

- What is the difference?

Choose Names You Can Pronounce

- If you can't pronounce it, you can't discuss it

```
class DtaRcrd102 {
  private Date genymdhms;
  private Date modymdhms;
  private final String pszqint = "102"; /* ... */
}
```

Choose Names You Can Pronounce

- If you can't pronounce it, you can't discuss it

```
class DtaRcrd102 {
  private Date genymdhms;
  private Date modymdhms;
  private final String pszqint = "102"; /* ... */
}
```

vs.

```
class Customer {
  private Date generationTimestamp;
  private Date modificationTimestamp;
  private final String recordId = "102"; /* ... */
}
```

Choose Names You Can Search

- Constants – usually hardly searchable
  - e.g., 5 vs. `WORK_DAYS_PER_WEEK`
- Short names – hardly searchable: e, a, ...
- Single letter – Only very local variables, e.g., `i`, `j`, `k` in for-loops

Choose Names You Can Search

```
for (int j=0; j<34; j++) {
  s += (t[j]*4)/5;
}
```

Choose Names You Can Search

```
for (int j=0; j<34; j++) {
  s += (t[j]*4)/5;
}
```

vs.

```
int realDaysPerIdealDay = 4;
const int WORK_DAYS_PER_WEEK = 5;
int sum = 0;
for (int j=0; j < NUMBER_OF_TASKS; j++) {
  int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;
  int realTaskWeeks = (realTaskDays / WORK_DAYS_PER_WEEK);
  sum += realTaskWeeks;
}
```

Solution vs. Problem Domain Names

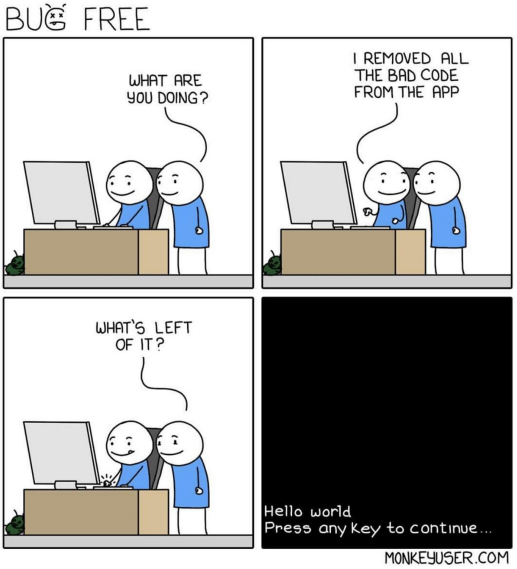
- Use CS terms (“solution domain”) when it makes sense:
  - names of algorithms, patterns, standard data structures, etc.
  - communicating to programmers
  - e.g., `JobQueue`, `calculateChecksum`
- Use problem domain terms when dealing with problem domain concepts
  - describing problem domain

Meaningful Context

- Provide context to ambiguous names
- E.g., what does `state` represent?
- Clearer when seen in context:
  `street`, `houseNumber`, `city`, `state`, `zipCode`
- Provide context in name: `addrStreet`, `addrCity`, `addrState`, ...
- Provide context by bundling: `class Address { ...`

Summary

- We *read* code most of the time
- Good code reads well – it flows
- Minimizes distractions
- Names should:
  - indicate intent
  - use meaningful distinctions
  - be pronounceable
  - be searchable
  - relate to the appropriate domain (problem vs. solution)
  - give enough context


BUG FREE
WHAT ARE YOU DOING?
I REMOVED ALL THE BAD CODE FROM THE APP
WHAT'S LEFT OF IT?
Hello world
Press any key to continue...
MONKEYUSER.COM