Software Engineering Coding Practises


Show
Me
(don't
tell
me)

Why it matters ?

- Improve your craft as engineers
- Reduce software error
- Code always evolves

Why it is hard ?

- Individual silos
- Code is frozen
- Hard to refactor code
- No pride in code

Write Code for Humans not Machines

SOFTWARE IS NEVER DONE

- Bugs are discovered
- Customers want new features
- Market demands new functionality
- Actively refactor code

Code for Change

Things You Should hate

- Dead code
- Unused variables
- Long parameter lists
- Long functions
- Inconsistency

Deep Dive

- Naming
- Code Comments
- Beautiful Code

Examples in book chapters

Naming

```
/*  Naming */
// BAD
int i1, i2, i3, i4, l1, l2, l3, l4;
void callServerAndReplayMessages();
long nActs, int c;


// GOOD
int input1, input2, input3, input4;
int output1, output2, output3, output4;
void doWork();
long nActivities, count;

/* Name same things same */

// alive/living mean same thing
void human(int id) {
    int alive = !planet.find(id).dead;
    eat(alive);
}

void eat(boolean living);


/* Make up you mind */

int numSucesss;
int errors;
int number_exceptions;


class DeviceManager {
    int userLogin();
}

class LoginManager {
    int login();
}
```

Avoid Code Comment

```
function length(area) {
    // square root
    a = 1
    b = area
    while (abs(a-b)>ErrorMargin)
       a = (a+b)/2
       b = x/a
    endwhile
    return a;
}
```

```
function length(area) {
    function squareRoot(area) {
        a = 1
        b = area
        while (abs(a-b)>ErrorMargin)
           a = (a+b)/2
           b = x/a
       endwhile
       return a
    }
    return squareRoot()
}
```

Avoid
Code
Comment

```
function saveUser(user) {
    // connect to db
    db = DB::open()
    // saving the use
    db.save(user)
    // close the database
    db.close()
}
```

Avoid
Code
Comment

```
starbucks.order(3); // large
```

```
enum CoffeeSize {
    LARGE=3,
....
}
starbucks.order(CoffeeSize.LARGE)
```

Beautiful Code - Example 1

NO

```
class Discounter :

    @classmethod
    def discount(self, u):
        if (not u.age > 12):
            return 50
        elif (u.age <= 18 and u.gender == 'F'):
            return 25
        if (u.age>= 65):
            if (u.preferred):
                return 45
            else:
                return 40
        else:
            if (u.preferred):
                return 5
            else:
                return 0
```

YES

```
class Discounter:

    @staticmethod
    def discount(user):
        def preferred(user):
            if (user.preferred):
                return 5
            return 0

        if (user.age <= 12):
            return 50
        if (user.age <= 18 and user.gender == 'F'):
            return 25
        if (user.age >= 65):
            return 40 + preferred(user)
        return 0 + preferred(user)
```

Beautiful Code - Example 2

```python
class Sum1:
    @staticmethod
    def doit(l):
        averages = []
        idx = 0
        while idx < len(l) -1:
            e = (l[idx] + l[idx+1])/ 2.0
            idx += 1
            averages.append(e)
        return averages

class Sum2:
    @staticmethod
    def doit(l):
        averages = []
        for i in range(0,len(l) - 1,1):
            e = (l[i] + l[i+1])/ 2.0
            averages.append(e)
        return averages


class Sum3:
    @staticmethod
    def doit(l):
        averages = [(x + y) / 2.0 for (x, y) in zip(l[:-1], l[1:])]
        return averages
```

Testing

TESTING FRAMEWORK

- a common test framework
- test framework can be run by all engineers
- easy to add tests
- on test failure the framework makes it easy to debug
- options to run a particular test or a full suite
- the build test cycle for a developer must be quick
- the tests run on every commit
- on failure team members are notified

MORE TESTING ...

- build more complicated tests on unit tests
- reported bugs recreate in test
- verify code with test
- apply good code practices when writing tests
- …. over time build more tests on existing test suite