Getting clean

How to apply clean code principles

Andrea Huber
techdev Solutions GmbH
04.11.2015

Conclusion

- Clean Code is doable.
- But only if both software developers and companies make an effort.

© xkcd.com
By Randall Munroe

© dilbert.com
By Scott Adams

Who can do what to make it work?

Agenda

0. Why care?

1. What can software developers do in order to learn clean code principles?

2. What can companies contribute?

Why care?

- Readable code, easily comprehensible by others
- Extensible code
- Interchangable components

} Maintainable code

- „Software craftsmanship“: Pride in good codebase

Prerequisites

- Here's what software developers need in order to learn clean code:
    1. Mindset
    2. Guidelines
    3. Feedback
    4. Practice

Mindset

- Be prepared to learn programming twice:
  - First time: Learn how to get it to work.
  - Second time: Learn to code well/professionally/clean.
- You will write bad code. It's ok.
  Just do not leave it at that.

Mindset

- Think of your code as a contribution to a common code base.
- This brings more responsibility:
  - Your code must be easily comprehensible by others.
  - You must make sure you approve of the code others check in.

Mindset

- Internalize the following principle:
  Once the code works and the tests run, the first part of your job is done. The second part is cleaning up / refactoring your code.
- Work in 3 phases:
  implementation → test → refactoring
  test → implementation → refactoring

Guidelines

- Keep a checklist of the most important clean code principles your team agreed on. Check your code against these principles before checking it in.

  1. Could my names (variables, methods, classes) be more expressive?
  2. Does any of my methods do more than one thing, i.e. does it violate the Single Responsibility Principle?
  3. Do I have duplicate code?
  4. Did I write unneccessary comments? On the other hand, did I comment everything that needs commenting?
  5. Did I use Javadoc correctly?
  6. Did I return null instead of throwing an exception?
  7. Are exceptions handled in the appropriate level? Are they documented?
  8. Do I have full test coverage?
  9. Is my code properly formatted (i.e. according to the team formatting rules)?

Comment examples

Unneccessary comment:

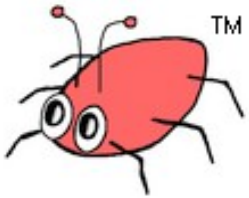
```
<!-- PAGINATION START -->
<div th:include="fragments/pagination-navigation-bar :: page_nav"></div>
<!-- PAGINATION END -->
```

Neccessary comment:

```
<th:block th:fragment="agree-checkbox-sender-without-creditcard">
    <!-- tabindex is needed for data-trigger="focus" to work on an i element -->
    <i id="checkbox_disabled" class="icon-checkbox-figure fa fa-check-square fa-2x pull-right"
       data-toggle="popover" data-placement="auto" data-trigger="focus" tabindex="0"
       th:attr="data-content=#{shypp-details.statebar.provideCreditCard}"></i>
    <script type="text/javascript">
        $(function() {
            $('[data-toggle="popover"]').popover({
                html: true
            });
        });
    </script>
</th:block>
```

Guidelines

- Let tools for static code analysis assist you with your code quality:
  - FindBugs
  - Checkstyle
  - PMD
  - SonarQube



checkstyle

sonarqube

PMD
DON'T SHOOT THE MESSENGER

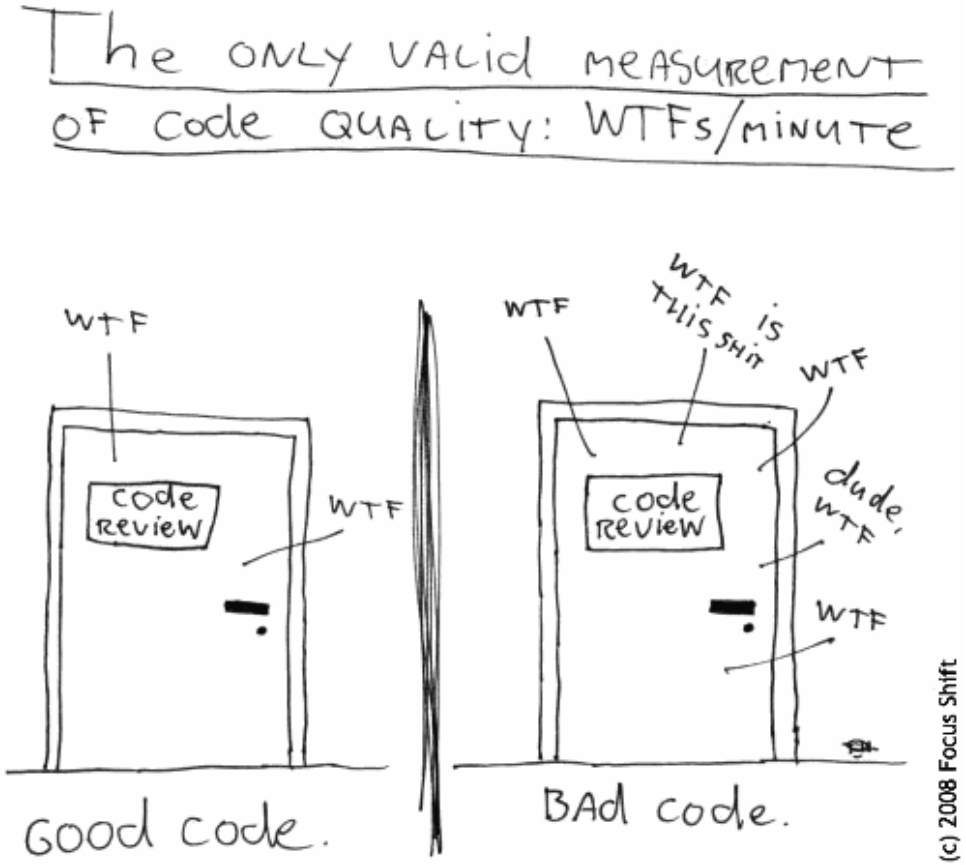
Guidelines

- Keep a checklist of your favourite mistakes. Check your code against these before checking it in.
- For example:

    1. Copy & paste error: Did I adapt names? Does every aspect of the old code fit the new usage?
    2. Did I write anything myself my framework offers out-of-the-box, or I could have used a library for?
    3. Did I auto-format my code?

Guidelines

- Think of your code as prose.
  Learn to ask yourself: Does it read well?



techdev SOLUTIONS

Guidelines

- Be nitpicky about tests. Make sure your tests match the following criteria:

  1. Do they cover everything, also corner cases
  (tools like SonarQube help with determining test coverage)?

  2. Do I have one concept per test
  (not neccessarily one assert-statement)?

  3. Do they have expressive names? Is it clear from the name what is being tested, and what the expected outcome is?

  4. Do their names correspond to the convention your team agreed on?

  5. Did I use the right test scope, i.e. did I write an integration tests where a unit test would suffice? Did I test anything unneccessary?

Guidelines

- Clean versioning:
- Make sure your commit messages are expressive and correspond to the team standard.

SHYPP-423 Changes date format in edit trip

```
Before, departure date in register/edit trip had been in English format. Formats date according to
the locale.
```

- If you use Git, learn squash and rebase in order to have a clean commit history.

Guidelines

- Keep your iteration cycles small
  (test → implementation → refactoring).
  Let your pull request deal with one ticket only.
- No premature optimization. Do only the task at hand.

Feedback

- Do code reviews as often as you can:
  have your code reviewed and review others' code.
- If possible, integrate code reviews in your team's workflow.
- Leave your ego at the door when your code is reviewed. Don't hesitate to ask for positive feedback – makes criticism much easier to handle!

Practice

- ...practice, practice...
- Regularly check your workflow and adapt it if you can think of improvements.
  This goes for the team's workflow as well as for individual workflows.
- ...and practice.

Company perspective

- Software developers have to learn clean code themselves. But a company can offer a good learning environment.
- Reminder: What do software developers need in order to learn clean code?

1. Mindset
2. Guidelines
3. Feedback
4. Practice

Mindset

- Communicate the importance of clean code in your company. Encourage your developers to learn clean code. Again and again.
- But do not leave it at that – offer good conditions to apply and practice clean code principles.

Project workflow

- These activities should be part of your developers' daily workflow:
  - Writing tests.
  - Refactoring code.
  - Reviewing code.
- Make sure everybody does these activities regularly.
- Allow enough time for these activities in project planning - avoid too much technical debt.

Company standards

- Agree on certain standards (naming conventions, most important clean code principles, formatting rules, commit message conventions etc.) in a team / in the company.
- Write them down.
- Make sure people understand WHY these standards are important.

Real life example

```java
public class TestPunkt{

    public static void main(String[] args) {
        /*
         * Deklaration der Testkonstanten, um keine magic numbers zu verwenden.
         */
        final int TEST1 = 5;
        final int TEST2 = 7;
        final int TEST3 = 9;
        final int ZERO = 0;

        Punkt p1 = new Punkt();

        p1.setX(ZERO); p1.setY(ZERO);                    // Zuerst wird x und y ein Wert zugewiesen.
        /*
         * Jetzt wird verschiebeInnerhalb(deltaX, deltaY) mit Werten, die eine Verschiebung erlauben getestet.
         */
        p1.verschiebeInnerhalb(TEST1, TEST1);
```


techdev SOLUTIONS

Staffing

- Hire enough seniors / people with clean code experience to teach juniors.
- Make it part of their job description to maintain code standards in the team, offer guidance and provide regular feedback on code.

Infrastructure

- Offer infrastructure, i.e.
  - tools for code reviews (GitHub etc.),
  - tools doing static code analysis (SonarQube, FindBugs, checkstyle, PMD etc.),
  - good IDE support (IntelliJ Ultimate, WebStorm etc.).

sonarqube

Training

- Provide opportunities for developers to improve their code quality:
- Hire experts e.g. for refactoring workshops.
- Send your staff to conferences, hackatons etc.
- Organize in-house tech talks on the subject to spread knowledge.
- If you complain that you cannot find good junior software developers: why not offer clean code / refactoring workshops at universities as a recruiting initiative?

Conclusion

- Clean code can work, if both software developers and companies make an effort.

Remember: it's a process!
Forget clean. Try to get cleaner every day.

Thank you for your attention.