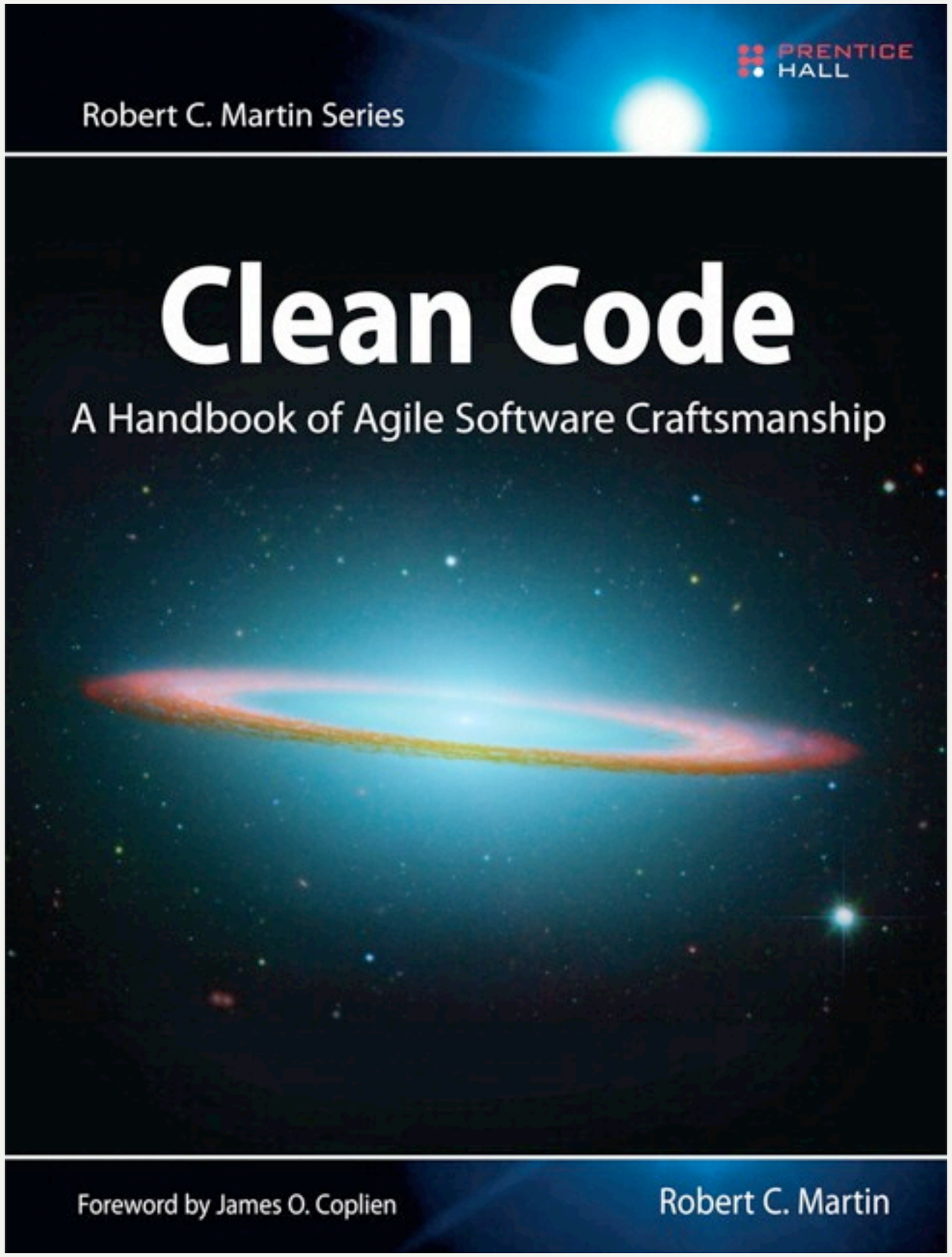
Clean code

Kien Nguyen Trung

Why?

- Everyone wants to write code which other people can understand
- Everyone want to be better programmer


PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

How?

- Meaningful names
- Clean, small functions
- Comments
- Good classes design

Meaningful names

Avoid disinformation name

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

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

Use intention reveal name

```java
// BAD
public List<int[]> getThem() {
    List<int[]> list1 = new ArrayList<int[]>();
    for (int[] x : theList)
        if (x[0] == 4)
            list1.add(x);
    return list1;
}

// GOOD
public List<Cell> getFlaggedCells() {
    List<Cell> flaggedCells = new ArrayList<Cell>();
    for (Cell cell : gameBoard)
        if (cell.isFlagged())
            flaggedCells.add(cell);
    return flaggedCells;
}
```

Use pronounceable name

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

// GOOD
class Customer {
    private Date generationTimestamp;
    private Date modificationTimeStamp;
    private final String recordId = "102";
}
```

Naming consistence

- Classes and objects should have noun
  - `Manager, Generator, Processor`
- Method should have verb
  - `isValid(), getUsername()`

Functions

45 lines function

```
public static String testableHtml(
        PageData pageData, boolean includeSuiteSetup) throws Exception {
    WikiPage wikiPage = pageData.getWikiPage();
    StringBuffer buffer = new StringBuffer();
    if (pageData.hasAttribute("Test")) {
        if (includeSuiteSetup) {
            WikiPage suiteSetup = PageCrawlerImpl.getInheritedPage(
                    SuiteResponder.SUITE_SETUP_NAME, wikiPage);
            if (suiteSetup != null) {
                WikiPagePath pagePath =
suiteSetup.getPageCrawler().getFullPath(suiteSetup);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -
setup .").append(pagePathName) .append("\n");
            }
        }
        WikiPage setup = PageCrawlerImpl.getInheritedPage("SetUp",
wikiPage);
        if (setup != null) {
            WikiPagePath setupPath =
wikiPage.getPageCrawler().getFullPath(setup);
            String setupPathName = PathParser.render(setupPath);
            buffer.append("!include -
setup .").append(setupPathName) .append("\n");
        }
    }
```

```
    buffer.append(pageData.getContent());
    if (pageData.hasAttribute("Test")) {
        WikiPage teardown = PageCrawlerImpl.getInheritedPage("TearDown",
wikiPage);
        if (teardown != null) {
            WikiPagePath tearDownPath =
wikiPage.getPageCrawler().getFullPath(teardown);
            String tearDownPathName = PathParser.render(tearDownPath);
            buffer.append("\n").
                append("!include -teardown .").
                append(tearDownPathName) .append("\n");
        }
        if (includeSuiteSetup) {
            WikiPage suiteTeardown = PageCrawlerImpl.getInheritedPage(
                    SuiteResponder.SUITE_TEARDOWN_NAME, wikiPage);
            if (suiteTeardown != null) {
                WikiPagePath pagePath =
suiteTeardown.getPageCrawler().getFullPath (suiteTeardown);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -teardown .").
                    append(pagePathName) .append("\n");
            }
        }
    }
    pageData.setContent(buffer.toString()); return pageData.getHtml();
}
```

What about this?

```
public static String renderPageWithSetupsAndTeardowns(
        PageData pageData, boolean isSuite) throws Exception {
    boolean isTestPage = pageData.hasAttribute("Test");
    if (isTestPage) {
        WikiPage testPage = pageData.getWikiPage();
        StringBuffer newPageContent = new StringBuffer();
        includeSetupPages(testPage, newPageContent, isSuite);
        newPageContent.append(pageData.getContent());
        includeTeardownPages(testPage, newPageContent, isSuite);
        pageData.setContent(newPageContent.toString());
    }
    return pageData.getHtml();
}
```

2 rules of function

- Functions should be small
- Functions should be smaller than that
- Function should has < 20 lines, each line has < 80 characters !?

What about this?

```
public static String renderPageWithSetupsAndTeardowns(
        PageData pageData, boolean isSuite) throws Exception {
    if (isTestPage(pageData))
        includeSetupAndTeardownPages(pageData, isSuite);
    return pageData.getHtml();
}
```

Do one thing

- How to know what is one thing?
- If you only need write one test to test it !?

Parameters

- Number of parameters should be as small as possible
- Zero is ideal number
- One, two, three is alright
- > 3 should be avoid

No side effects

```
public class UserValidator {
    private Cryptographer cryptographer;

    public boolean checkPassword(String userName, String password) {
        User user = UserGateway.findByName(userName);
        if (user != User.NULL) {
            String codedPhrase = user.getPhraseEncodedByPassword();
            String phrase = cryptographer.decrypt(codedPhrase, password);
            if ("Valid Password".equals(phrase)) {
                Session.initialize();
                return true;
            }
        }
        return false;
    }
}
```

checkPassword can be called only 1 time

DRY (don't repeat yourself)

- Duplication may be the root of all evil in software
- Avoid copy and paste
- Make functions/classes/libraries to reuse your code

Comments

Comments

- Don't make comments for bad code
- If code is expressive enough, comments are rarely need
- Explain yourself in code, instead of comments

Comments

```
// Check to see if the employee is eligible for full benefits
if ((employee.flags & HOURLY_FLAG) && (employee.age > 65))
```

```
if (employee.isEligibleForFullBenefits())
```

Classes

Class design

- Classes should be small
- Classes should be smaller than that

Single responsibility principle

- Class should do only one thing
- Class should have one responsibility - one reason to change
- Separate big class into many small classes.
- Each class should have small number of methods

God class

```
public class SuperDashboard extends JFrame implements MetaDataUser {
    public String getCustomizerLanguagePath();
    public void setSystemConfigPath(String systemConfigPath);
    public String getSystemConfigDocument();
    public void setSystemConfigDocument(String systemConfigDocument);
    public boolean getGuruState();
    public boolean getNoviceState();
    public boolean getOpenSourceState();
    public void showObject(MetaObject object);
    public void showProgress(String s);
    public boolean isMetadataDirty();
    public void setIsMetadataDirty(boolean isMetadataDirty);
    public Component getLastFocusedComponent();
    public void setLastFocused(Component lastFocused);
    public void setMouseSelectState(boolean isMouseSelected);
    public boolean isMouseSelected();
    public LanguageManager getLanguageManager();
    public Project getProject();
    public Project getFirstProject();
    public Project getLastProject();
    public String getNewProjectName();
```

```
    public void setComponentSizes(Dimension dim);
    public String getCurrentDir();
    public void setCurrentDir(String newDir);
    public void updateStatus(int dotPos, int markPos);
    public Class[] getDataBaseClasses();
    public MetadataFeeder getMetadataFeeder();
    public void addProject(Project project);
    public boolean setCurrentProject(Project project);
    public boolean removeProject(Project project);
    public MetaProjectHeader getProgramMetadata();
    public void resetDashboard();
    public Project loadProject(String fileName, String projectName);
    public void setCanSaveMetadata(boolean canSave);
    public MetaObject getSelectedObject();
    public void deselectObjects();
    public void setProject(Project project);
    public void editorAction(String actionName, ActionEvent event);
    public void setMode(int mode);
    public int getMajorVersionNumber();
    public int getMinorVersionNumber();
    public int getBuildNumber();
}
```

Separate class

```
class VersionController {
    public int getMajorVersionNumber();
    public int getMinorVersionNumber();
    public int getBuildNumber();
}
```

Cohesion

- Classes should have small number of instance variables
- Each of the methods of a class should manipulate one or more instance variables

Cohesion

```
public class Stack {
    private int topOfStack = 0;

    List<Integer> elements = new LinkedList<Integer>();

    public int size() {
        return topOfStack;
    }

    public void push(int element) {
        topOfStack++;
        elements.add(element);
    }

    public int pop() throws PoppedWhenEmpty {
        if (topOfStack == 0)
            throw new PoppedWhenEmpty();
        int element = elements.get(--topOfStack);
        elements.remove(topOfStack);
        return element;
    }
}
```

Refactoring

Refactoring rules

- Don't afraid of changing code to make it better
- Prepare test for functions/classes before refactoring (apply TDD ?!)
- Review other code when they commit

Reviewing process

- Use GIT
- Create new branch for new feature/bug
- Commit to feature branch (not master)
- When feature done, ask other people for reviewing, modify and merge it to master branch after reviewing

Conclusion

- Choose expressive name for variables/ functions/classes
- Write small functions (< 20 lines)
- Write small classes
- Don't repeat yourself

References

- Clean code (Robert Martin) -highly recommended
- Refactoring - Improving design of Existing Code (Robert Martin, Kent Beck)

Question