CODEMOTION WORKSHOP

CLEAN CODE & DESIGN PRINCIPLES IN ACTION

DEVELOP HIGH QUALITY APPLICATIONS, FASTER

CARLO BONAMICO / carlo.bonamico@nispro.it / @carlobonamico

ANDREA PASSADORE / andrea.passadore@nispro.it

MODULE 00

Introduction

WHILE YOU ARE WAITING...

- download the labs from
  - https://github.com/carlobonamico/clean-code-design-principles-in-action

```
git clone https://github.com/carlobonamico/clean-code-design-principles-in-action
```

or plain "Download Zip" from browser

CLEAN CODE AND DESIGN PRINCIPLES IN ACTION:

DEVELOP QUALITY APPLICATIONS, FASTER

As developers, we often feel that we are always asked for more: more features, more bugfixes, more code to get our application done, faster. In the workshop you'll learn first hand how applying Clean Code and Design Principles will help you complete solid & maintainable applications in less time.

- http://milan2015.codemotionworld.com/workshop/clean-code-and-design-principles-in-action-develop-quality-applications-faster/

ABSTRACT

We are often asked for more features, more bugfixes, faster. But is "running" always the fastest way to get things done? For a mountain climber, "running" means more mistakes, falling off more often,thus proceeding more slowly.

So like a climber gets to the top through a continuous chain of small, safe steps, we can improve coding by making design & implementation steps clean, safe and incremental.

In the workshop, starting from concrete examples, you will learn first hand how applying Clean Code and Design Principles will help you complete more solid & maintainable applications in less time

AUDIENCE

Basically, to all developers! Independently from the language / platform you are developing on, and from your expertise level, if you are interested in improving your approach to coding, and develop higher quality applications with more productivity, this workshop is for you.

PREREQUISITES

Working knowledge and practical experience in one programming language (you should be able to write/compile/test/debug by yourself a program which reads and parses input and presents output either on the command line or in a simple GUI). Basic knowledge of HTTP.

HARDWARE AND SOFTWARE REQUIREMENTS

- Participants are required to bring their own laptop. The labs require an HTML5 Browser (Chrome or Firefox), text editor or IDE, supporting HTML5, CSS3, JavaScript.
- Modern Browser (Chrome - Firefox)

Text Editor (Sublime, Atom, Visual Studio Code,...)

- http://brackets.io/
- http://atom.io

and/or IDE (Eclipse, NetBeans, Intellij, Visual Studio,..)

- http://www.eclipse.org
- http://netbeans.org
- WebStorms / Visual Studio Community
- `python -m SimpleHTTPServer` vs https://code.google.com/p/mongoose/

TOPICS

- How does our code become unmanageable? A practical example
- What can we do about that?
  - Clean Code, Design Principles and Lean to the rescue
- Clean Code by example: key concepts
  - Concept 1 - Naming
  - Concept 2 - Formatting
  - Concept 3 - What's in a good function?
  - Concept 4 - What's in a good class? Design Principles

TOPICS

- Concept 5 - Making our code Testable
- Concept 6 - Making debug and troubleshooting easier
- Concept 7 - Refactoring
  - The principles: quality vs productivity - continuous safe steps
  - Incremental development and evolutionary design
  - A more complex example - applying the method to real-world problems
  - How to continue by yourself: references for further learning

APPROACH

For each module, hands-on lab will include

- quizzes (which of these variants is better? trade-offs)
- interactive examples to complete and modify in an online IDE

KEY REFERENCES

- All Labs and links available at
  - https://github.com/carlobonamico/clean-code-design-principles-in-action
- Clean Code: the book
  - https://books.google.it/books/about/Clean_Code.html?id=hjEFCAAAQBAJ

(images/CleanCode.png)

REVISING CORE JAVASCRIPT CONCEPTS

- Yakov Fain - Advanced Introduction to Javascript
  - https://www.youtube.com/watch?v=X1J0oMayvC0
  - http://enterprisewebbook.com/appendix_a_advancedjs.html
    - https://github.com/Farata/EnterpriseWebBook
    - https://github.com/Farata/EnterpriseWebBook_sources

MODULE 01

Clean Code: Why does it matter?

TOPICS

How does our code become unmanageable? A practical example

- fast-forward demo through the life of an (apparently) trivial function

LEGACY CODE ANONYMOUS ...

Hello... My name is Carlo,

and I have been writing Legacy Code for 15 years...

THIS WORKSHOP IS NOT ABOUT THE "PERFECT" WAY TO DEVELOP SOFTWARE

BUT MORE ON THE NEED FOR NEVER STOP IMPROVING OUR APPROACH

MY PHONE DOES NOT WORK

Can you have a look?

THE REQUIREMENTS

How does our code become unmanageable? A practical example

- fast-forward demo through the life of an (apparently) trivial function

Write an expense report tool that

- runs in a folder where employee expense excel (csv for now) are stored
- a file for each month (e.g. 01.csv for January)

SPECIFICATIONS

INPUT

```
10/01/2015, 10.50
11/01/2015, 8.50
12/01/2015, 5.50
15/01/2015, 8.50
```

EXPECTED OUTPUT

- compute the monthly total and produce <>-report.txt

```
Month January

Expenses: 4
Amount: 33
```

CONVERT COFFEE TO CODE...

Version 1... implemented in 15 minutes! :-)

not good but not so bad

labs/clean-code-expenses-bad

CAN YOU ALSO...

- add a type of expense
- skip an header row

```
Date, Type, Amount
10/01/2015, Train, 10.50
```

- produce an html report
- compute expenses per category
- detect if the expenses are more than threshold

THE RESULTS

another 15 minutes - I am a 10x-programmer!

120 Lines, and already unmaintainable

labs/clean-code-expenses-ugly

THE EFFECTS:

- code-writing time -> decreases
- application-ready time -> never done
- time needed for bug fixes and new features -> increases

WHAT HAPPENS AFTER 6 MONTHS?

The original developer has left the company and your boss asks you:

Can you simply automatically do this for all employees? and fix the case where the input file is incorrect and fix the rounding errors And keep two intermediate totals for cash and credit card expenses?

And integrate with the acccounting application?

:-(

Well... it will take N weeks just to understand and fix the bugs in the original code...

Which become months with regression testing

WHY THIS HAPPENS?

the "deadly sins" of development

- cut & paste,
- optimization lust
- haste
- false savings
- naming avarice
- trial and error wrath
- my code is perfect pride
- making it right is too hard - discouragement

LAB 01

Lab

- write down 3 problems with the expense report code

EXERCISES

- try to keep track of how much time you waste because of bad code
  - just tick a mark every time it happens

LEARN MORE

- Symptoms of Rotten Design
  - http://www.objectmentor.com/resources/articles/Principles_and_

WHAT CAN WE DO ABOUT THAT?

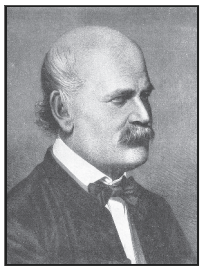
Clean Code, Design Principles and Lean to the rescue

- improving our code
- improving our design
- practice, practice, practice and continuous / daily improvement (Kaizen)

WHAT CAN WE DO ABOUT THAT?

- Ideas -> 3 post it

DO YOU KNOW THIS MAN?



IT IS *NOT* BRAIN SURGERY

- Ignaz Semmelweis
  - http://www.npr.org/sections/health-shots/2015/01/12/375663920/the-doctor-who-championed-hand-washing-and-saved-women-s-lives
  - http://semmelweis.org/about/dr-semmelweis-biography/
- He championed washing hands before childbirth and surgery

THE SIMPLES THINGS ARE THE MOST DIFFICULT TO DO

- he was obstracized by the medical community!
  - it can't be that simple...
  - we just don't have time...
- And now?

CLEAN CODE

- It cannot solve all development problems...
- But it can make them way more tractable!

DESIGN PRINCIPLES

Once we have got the basics covered, then we will need to understand the Software Dynamics

- vs the nature (and Laws) of Software

Take them into account => Design Principles

Basically, Common Sense applied to software design

Treat your code like your kitchen C.B., about 2013

IMPROVE OUR CODE

It takes a Deliberate approach and constant effort

To complicate is easy, to simplify is hard To complicate, just add, everyone is able to complicate Few are able to simplify Bruno Munari

PRACTICE

If I don't practice for a day, I notice If I don't practice for two days, my orchestra notices If I don't practice for three days, the public notice Claudio Abbado

- practice, practice, practice and continuous / daily improvement (Kaizen)
  - *deliberate practice* -> iterate small skills until >90% perfect

PRACTICE AND KATAS

Code Katas

CLEAN CODE BY EXAMPLE: KEY CONCEPTS

CONCEPT 1 - NAMING

- reading code vs writing code
- what is a good name?
- same but different: the importance of conventions
- be meaningful
- aside: commit messages

CODE ?!#%

WHY DOES THIS MATTER?

A little experiment

Write down:

- what the software does
- how long it took to understand it
- which bugs you can find in the code
- how long it took to find them

READY - SET - GO!

Group A: go to http://plnkr.co/edit/dQldXF

Group B: go to http://plnkr.co/edit/zPXf70?

READING CODE VS WRITING CODE

What is written without effort is in general read without pleasure.

Samuel Johnson

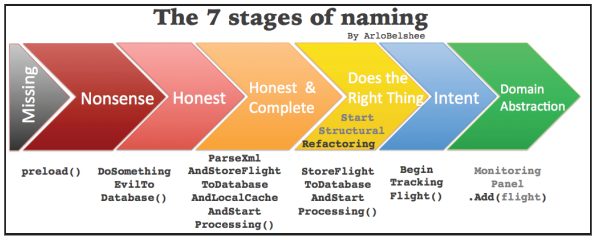
Most code is written once, but read

- every time you need to fix a bug
- to add new features
- by other developers
  - including your future self

WHAT IS A GOOD NAME?

- Ideas?

WHAT IS A GOOD NAME



- nonsense
- honest
- honest & complete
- does the right thing
- intent
- domain abstraction

http://llewellynfalco.blogspot.it/p/infographics.html

BE MEANINGFUL

- good code shouldn't almost require naming comments

SAME BUT DIFFERENT: THE IMPORTANCE OF CONVENTIONS

How do you read this code:

```
float tempC;
float farTemp
```

IF IT IS DIFFERENT, IT MUST HAVE A MEANING

```
float celsiusTemperature;
float farenheitTemperature;
```

LAB

Lab 1: write down a function for computing the amount of days between two dates

- jot it down
- check that it works
- review the naming of each variable and function
- check that it works
- review the naming - again
- discuss it with your neighbour

LAB 2

Lab 2: refactor the first example

- http://plnkr.co/edit/lVa8ws?p=preview
- http://plnkr.co/edit/zPXf70?p=preview

TIP:

- avoid mental mapping

```
formatMessage("File not found",2);
formatMessage("File not available",1);
formatMessage("File loaded",3);
```

WHY?

- Mental energy is finite
  - attention over time
  - amount of information: 7 +/- 2
- do not waste it in useless mappings
  - see http://www.amazon.it/Badass-Making-Awesome-Kathy-Sierra/dp/1491919019

ASIDE: COMMIT MESSAGES

Everything is like code

- configuration files
- infrastructure
- scripts And
- commit log messages
  - a message to your future self (and team mates) `git commit -m "bug fix"`

LEARN MORE

- https://en.wikipedia.org/wiki/Leap_year#Algorithm
- http://blog.speziale.it/post/Scorporo-dei-prezzi-ivati-problema-del-e2809ccentesimo-pazzoe2809d.aspx

CONCEPT 2 - FORMATTING

- making code readable
- making code diff-friendly & commit-friendly
- making code modification-friendly

CODE ?!#%

What does this code do? http://plnkr.co/edit/mZtyDG?p=info

MAKING CODE READABLE

- the IDE does most of that for you
- share a team guideline
  - possibly, a shared style template

MAKING CODE DIFF-FRIENDLY

- diffs work mainly line by line
- each line should have a different reason / time to change

HTML EXAMPLE

```
<div>Threshold <input type="number" style="color : red" min ="5"
  max = "57" class ="outline"
  ></div>
```

vs

```
<div>
  Threshold
  <input type="number" min ="5" max = "57"
         style="color : red" class ="outline"
  >
</div>
```

MAKING CODE COMMIT-FRIENDLY

- this makes thing also merge-friendly

MAKING CODE MODIFICATION-FRIENDLY

- A modification per line

LAB

Refactor this

- http://plnkr.co/edit/91t0kv?p=preview

CONCEPT 3 - WHAT'S IN A GOOD FUNCTION?

- single responsibility
- separing inputs from outputs
- if you have to do 3 things, make 4 functions
- primitives and orchestrators

CODE ?!#%

See the `gen()` function again

SINGLE RESPONSIBILITY

Each function should do 1 thing

Or even better, have a single responsibility

- and reason to change

HOW TO FIND RESPONSIBILITIES?

Ask yourself questions...

- What?
- Who?
- When?
- Why?
- Where?

And put the answer in different sub-functions

INPUTS VS OUTPUTS

- make inputs clear
- limit / avoid output parameters

3 THINGS, 4 FUNCTIONS

If your function needs to perform a non-trivial task:

- import data, transform it and store it in the DB

Instead of

```
readData(){
  file.open();
  while(..)
    {
      line = readLine();
      obj = trasformLine(line);
      saveInDB(obj);
    }
}
```

what's better?

3 THINGS, 4 FUNCTIONS

```
importData(){
  data = readData();
  obj = transformData (data);
  saveInDB(obj);
}
```

- a function for each step
- a function to call the steps

PRIMITIVES, ORCHESTRATORS, LEVEL OF ABSTRACTION

- Primitives: small, focused, typically use-case independent
- Orchestrators: implement use-cases by combining primitives
- rinse and repeat over multiple levels of abstraction
- benefits:
  - more reusable
  - easier to test

LAB

- Parse the Meteo Data file and compute the weekly min and max temperature

CONCEPT 4 - WHAT'S IN A GOOD CLASS? DESIGN PRINCIPLES

- Single Responsibility Principle
- collaborating with other classes
- composition vs inheritance (and the Open/Closed principle)
- Dependency Injection
- interfaces and the importante of Contracts

SINGLE RESPONSIBILITY PRINCIPLE

Have you ever seen your grandmother put dirty clothes in the fridge?

Or biscuits in the vegetable box?

So, why to we do this all the time in our code?

SINGLE RESPONSIBILITY PRINCIPLE

Responsibility == reason to change

SRP - AGAIN

A class should do one thing

- and have a single reason to change

Consequences:

- classes should be small
- classes should be focused
- classes need to collaborate to perform complex tasks

LAB

- Take the "ugly" code or any other code example
- Paste it in word / Google Docs
- Outline in different colors the various responsibilities

COLLABORATING CLASSES

- We need a way of making collaboration easier
- With Dependency Injection
  - separate creation of classes from linking instances
  - create A
  - create B
  - something else passes B to A
- You do not need a framework for that...

INHERITANCE - WITH CAUTION

- Inheritance is the strongest link between classes
- useful with caution

PREFER COMPOSITION

- combine parts
- a derived class becomes the composition of a base behaviour + additional custom behaviour

OCP

Open for extension, Closed for Modification

INTERFACES AND CONTRACTS

- esplicit vs implicit
- Decoupling changes and detecting regressions
- separate clean parts from dirty code

LAB

- Design the classes for the additional requirements for the expense report
- Can you simply automatically do this for all employees?
- and fix the case where the input file is incorrect
- keep two intermediate totals for cash and credit card expenses?
- integrate with the acccounting application

CONCEPT 5 - MAKING OUR CODE TESTABLE

- avoid statics
- testable code vs good design

CODE ?!#%

TIPS AND TRICKS

- avoid statics
- decouple code

TESTABLE CODE AND GOOD DESIGN ALIGMENT

Things that make code testable

- clear interfaces
- small classes / functions
- decoupled
- composition

Things that make code well-designed, easy to evolve

- clear interfaces
- small classes / functions
- decoupled
- composition

MORE PRACTICE AND KATAS

- http://matteo.vaccari.name/blog/tdd-resources

LEARN MORE

- http://misko.hevery.com/code-reviewers-guide/

CONCEPT 6 - MAKING DEBUG AND TROUBLESHOOTING EASIER

- one task - one statement
- make return values visible
- logging

CODE ?!#%

```
        BufferedReader reader = new BufferedReader(new FileReader(fileName));

        String line = reader.readLine();
        while(line != null)
        {
            String [] lineElements = line.split(",");
            forecasts.add(new MeteoForecast(lineElements[1],
                    formatter.parse(lineElements[0]),
                    Integer.parseInt(lineElements[2]),Integer.parseInt(lineElements[3].trim

            );
```

It contains a mistake - go debug it! ./labs/lab-debug-01

ONE TASK - ONE STATEMENT

The `forecast.add(...` line both

- has multiple responsibilities
- manipulates multiple values

So split it!

Even one statement -> multiple lines,

- where the language supports it

RETURN VALUES

Having temporary variables for return values

- makes debugging easier
  - inspecting the value *during* construction
  - inspecting the value *before* returning
    - even modifying it in some debuggers
- makes type errors more visible

Also modern debuggers are able to display the value inline

- http://plnkr.co/edit/WoP0Ry?p=preview

LOGGING

- make log messages understandable
  - avoid mental mapping
- make them easy to search

LAB

Refactor the example

IF YOU ONLY REMEMBER 1 THING

Your code should read like a terse prose of *simple* statements

- this also helps with debugging!

CONCEPT 7 - REFACTORING

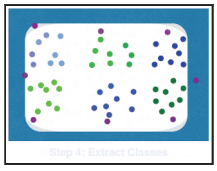
- from bad code to good code
- in steps
- learn your IDE refactoring tools
- The "Boy Scout Rule"
- Why we need unit tests?

CODE ?!#%

- Go back to the "ugly" example

FROM BAD TO GOOD

Incremental transformation



IN STEPS

- Each step should not change the functional properties of the system
- and improve the non-functional ones
- separate adding features from refactoring
  - don't do both in the same step

IDE REFACTORING TOOLS

- if you have to pick one: Find all references
- Refactor / Rename
- Extract Method
- Extract Interface
- http://refactoring.com/

THE BOY SCOUT RULE

Leave the campsite a little better than you found it

Every time you touch some code, leave it a little better

The power of compounding many small changes *in the same direction*

- 1% time

WHY DO WE NEED TESTS?

- be confident that we can change the code with very low risk
- simple refactorings are performed at the AST level in the IDE
  - renaming
  - extracting method

LAB

- Refactor the initial Expense Report example
- name things
- extract sub-methods
- create classes
- extract interfaces

THE PRINCIPLES

So what did we just do? Understand the principles

- the relationship between quality and productivity
- the need for a continuous chain of small, safe steps of design & implementation

QUALITY VS PRODUCTIVITY

Traditionally, Quality is seen as an alternative to raw development Speed

- this is partly true only in the short term

Four quadrants:

- high quality, high productivity -> tends to further improve
- high quality, low productivity -> tends not to improve, and go to
- low quality, low productivity -> tends to get worse
- low quality, high productivity -> tends to go to the previous one
- Productivity curves at different quality ratio

CONTINUOUS CHAIN

- Faster small steps beat bigger steps
- also easier to parallelize
- The smaller the better

SAFE STEPS

- you need to be able to check that everything works
- run frequently
- test frequently

EVOLUTIONARY DESIGN VS EMERGENT DESIGN

- Making things easier to change
- This does not mean that you do not have a vision
- Plan the overall Path
  - but execute a step at a time

LAB

Define the main structure Split in sub-tastks with post-its Discuss the optimal order Introduce mock / support steps

MORE PRACTICE AND KATAS

- Elefant Carpaccio
  - https://docs.google.com/document/u/1/d/1TCuuu-8Mm14oxsOnlk8DqfZAA1cvtYu9WGv67Yj_sSk/pub

INCREMENTAL DEVELOPMENT AND EVOLUTIONARY DESIGN

- how to do everything incrementally
- Separation of Concerns in practice: ask yourself questions!
- incremental implementation: in-application Mocks & the Walking Skeleton approach
- how to keep track of what you do and what is missing
- how to manage incremental commits

HOW TO DO EVERYTHING INCREMENTALLY

- You can do *everything* incrementally
  - decouple release from deployment
  - branch by abstraction
  - do both
  - expand-contract

HOW TO MAKE IT SMALLER - ASK YOURSELF QUESTIONS

- what if instead I only do X?
- A & B -> A then B

IN -APP MOCKING

- like in the tests

WALKING SKELETON

- entire application / workflow structure
- made of empty (or logging-only) components
- incrementally filled-in
- also useful for testing

HOW TO KEEP TRACK OF WHAT YOU DO AND WHAT'S MISSING

- Write it down
- comment it with temporary comments
- code it!

INCREMENTAL COMMITS

- Each commit should start from a stable state and lead to a stable but more complete state
- Push vs commit in DVCS

LINK: CONTINUOUS DELIVERY

http://continuousdelivery.com/

LAB

Define the main structure Split in sub-tastks with post-its Discuss the optimal order Introduce mock / support steps

A MORE COMPLEX EXAMPLE - APPLYING THE METHOD TO REAL-WORLD PROBLEMS

- the feedback loop
- splitting the problem
- getting more feedback
- getting feedback more frequently
- model the problem
- avoid trial and error, but if you need it, do it fast

CODE ?!#%

When it does not work...

And we do not know why...

THE FEEDBACK LOOP

- Idea --> Change --> Observation --> Evaluation

Learning

THE TIME SINK

- limit in the amount of changes
- time required
  - to effect change
  - to observe the result
- information collected after the change

EXPERIMENTS

- split the problem
- make it smaller
- make it independent

GETTING MORE FEEDBACK

- adding more log / monitoring
- adding higher quality log info
- adding dedicated code / validation logic
- better tools (inspectors...)

GETTING FEEDBACK MORE FREQUENTLY

- make the loop faster
  - e.g. jrebel
  - gulp serve
  - livereload

MODEL BASED, SYSTEM THINKING

- Hypothesis
- Which test do I need?
- write it down
- does the result conforms to the expectation?
- reduce uncertainty

AVOID TRIAL AND ERROR

Or at least do it fast

MORE PRACTICE AND KATAS

- http://codekata.com/
- https://www.industriallogic.com/blog/modern-agile/

IMPROVEMENT CULTURE

- https://codeascraft.com/2012/05/22/blameless-postmortems/

LEARNING TO LEARN

- Kathy Sierra
- https://www.youtube.com/watch?v=FKTxC9pl-WM

BEYOND CLEAN CODE

CLEAN PROJECTS

- http://misko.hevery.com/2008/07/16/top-10-things-i-do-on-every-project/

MODULE

References

TO LEARN MORE

- Online tutorials and video trainings:
  - https://cleancoders.com
- Full lab from my Codemotion Workshop
  - https://github.com/carlobonamico/clean-code-design-principles-in-action

HOW TO CONTINUE BY YOURSELF: REFERENCES FOR FURTHER LEARNING

- Principles of Package Design
  - http://www.objectmentor.com/resources/articles/Principles_and_
- More on TDD
  - http://matteo.vaccari.name/blog/tdd-resources
- Modern Agile
  - https://www.industriallogic.com/blog/modern-agile/
- Lean, Quality vs Productivity and DevOps
  - http://itrevolution.com/books/phoenix-project-devops-book/

JAVASCRIPT

- http://humanjavascript.com/
- http://javascript.crockford.com/
- http://yuiblog.com/crockford/
- Free javascript books
- http://jsbooks.revolunet.com/

THANK YOU

- Other trainings
  - https://github.com/carlobonamico/
- My presentations
  - http://slideshare.net/carlo.bonamico
- Follow me at @carlobonamico / @nis_srl
- Contact me carlo.bonamico@nispro.it / carlo.bonamico@gmail.com andrea.passadore@nispro.it

#

THANK YOU FOR YOUR ATTENTION

CARLO BONAMICO

@CARLOBONAMICO / @NIS_SRL

CARLO.BONAMICO@NISPRO.IT

CARLO.BONAMICO@GMAIL.COM

HTTP://MILANO.CODEMOTIONWORLD.COM

HTTP://TRAINING.CODEMOTION.IT/

{codemotion}