Platinum Sponsor

LUXOFT
Engineer Your Future

EFFECTIVE REFACTORING

Włodek Krakowski

33rd DEGREE

A few words about myself

- **Master's degree in Computer Science (2001)**
- **13 + years of experience, observing things and improving them ☺**
- **Sabre : Junior -> ... -> Team Lead**
- **Lumesse : Application Architect**

A few words about myself

Here is the place where I currently introduce my refactoring ideas...

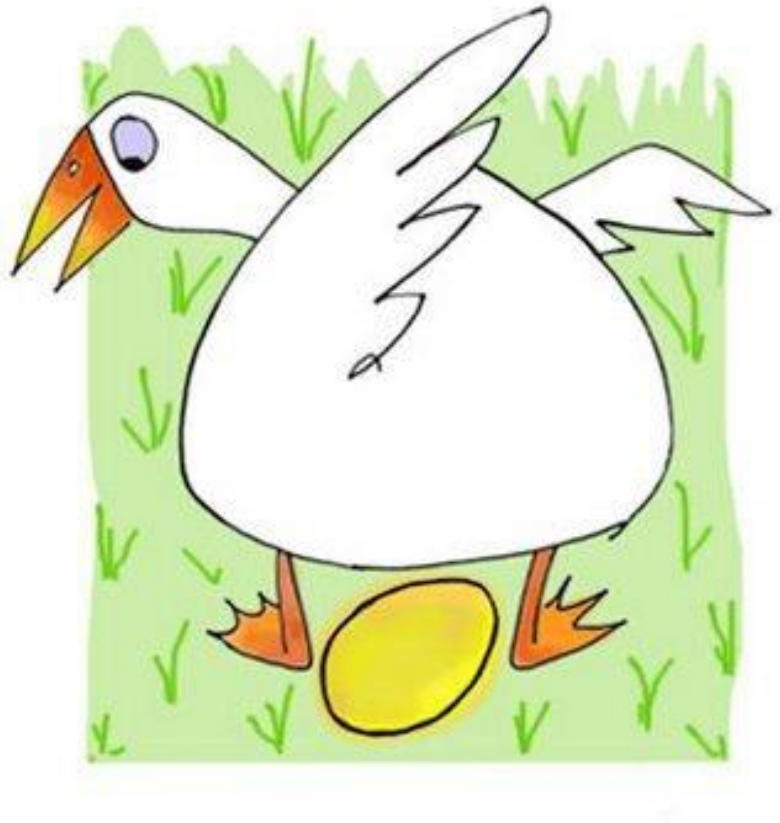
Lumesse
inspiring talent management

What these stories are about?

- **Centuries ago ...**
- **Many years ago ...**
- **Nowadays...**

Centuries ago...

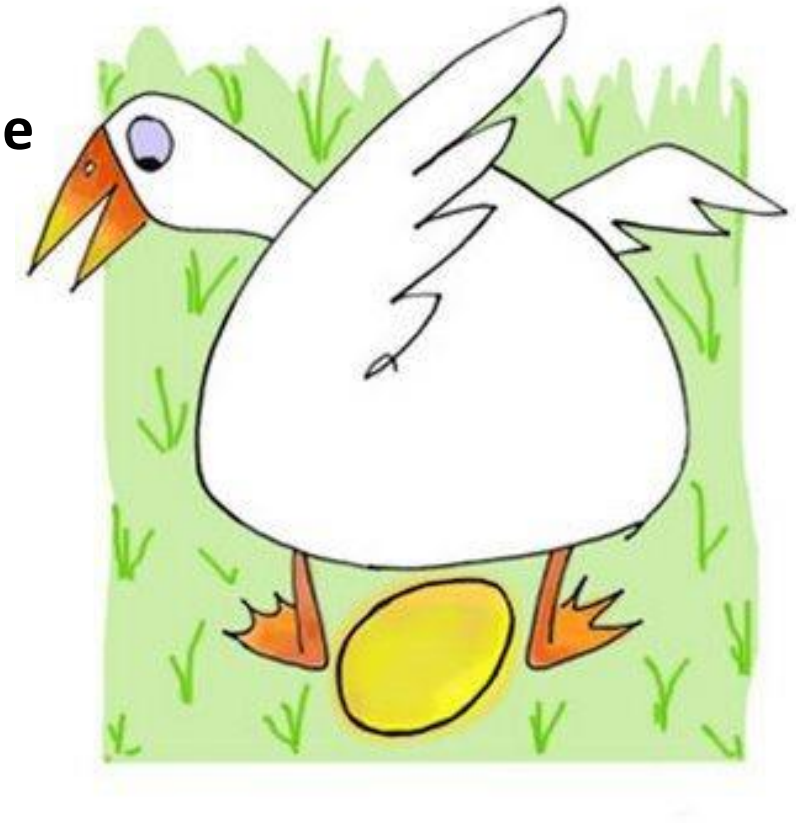
- **Once upon a time there were a farmer and his goose…**
- **One day he noticed that his goose laid down a golden egg…**
- **Since that day happy & rich life was going on…**



Centuries ago...

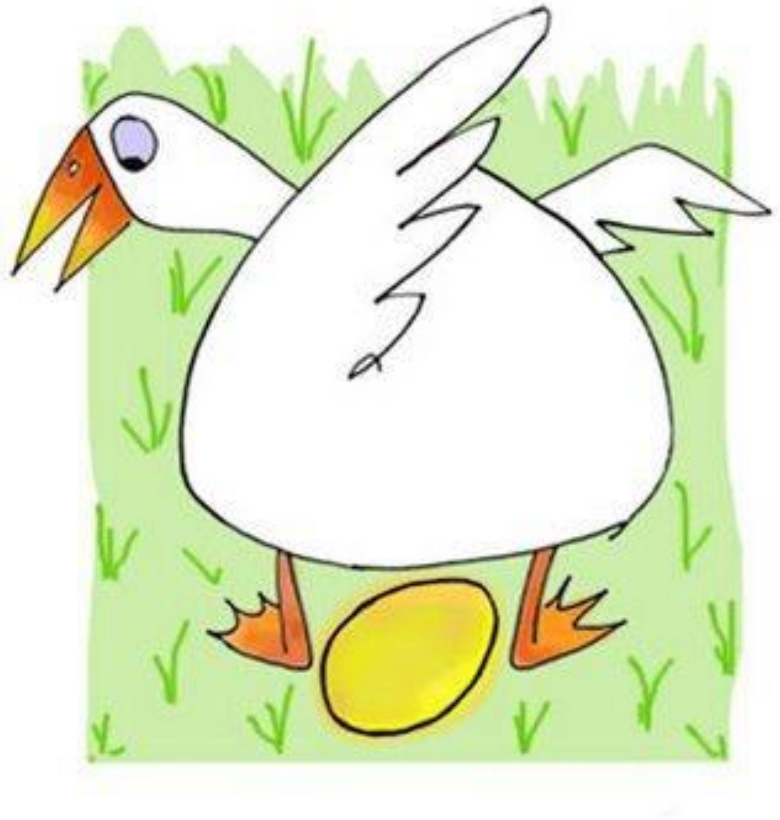
- **But after some time the farmer became very greedy & impatient…**

- **… and finally he decided to kill the goose in order to get all of the gold from inside at once...**

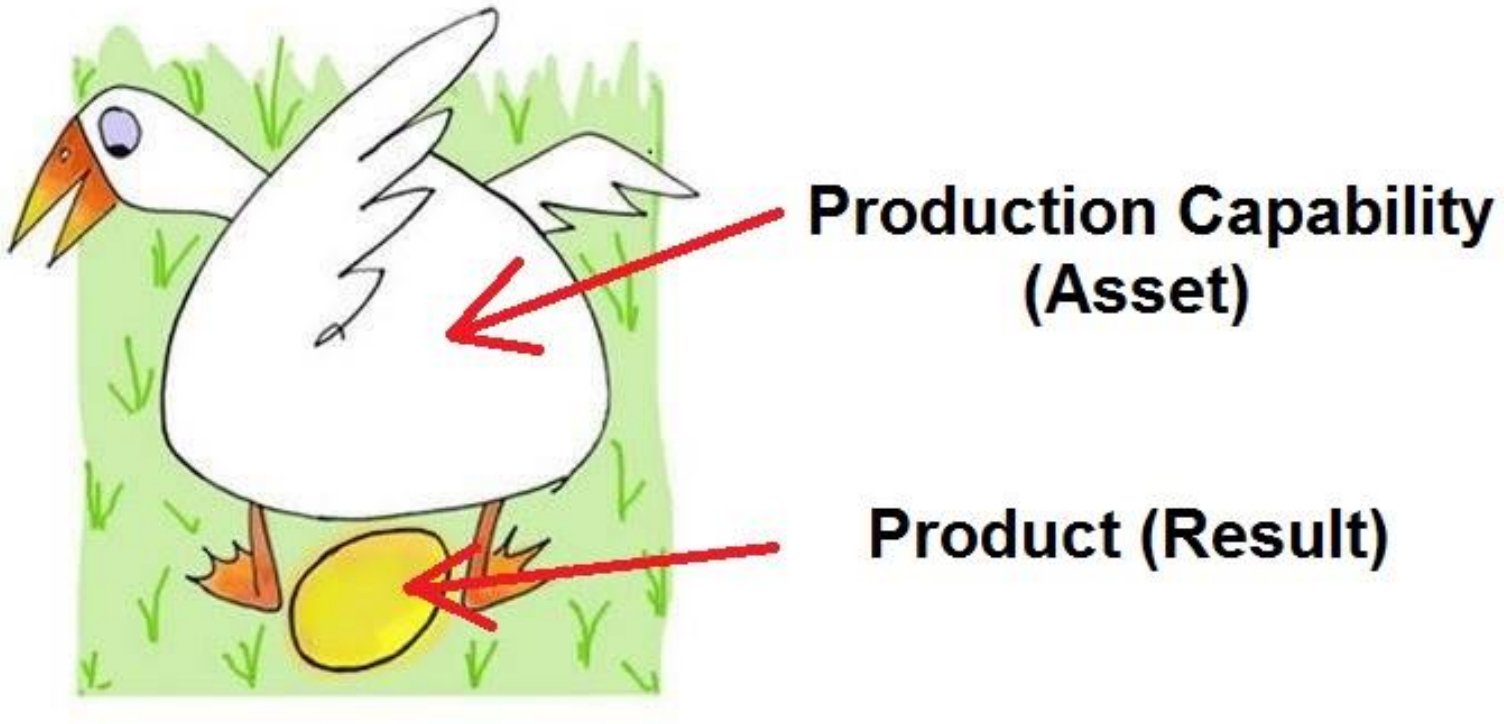


Centuries ago...

- **Surprisingly opposite to his thoughts there was no gold inside the goose at all…**
- **What does the golden egg refer to?**
- **What does the goose refer to?**



Production and Production Capability cannot be considered a separated items



What is effectiveness?

- **At first glance** it looks like the effectiveness seems to be **amount/size** of today's production

- But **diligent leadership** and management needs to consider the **future as well**

Full picture of effectiveness

- At a bigger picture Effectiveness is not only the amount of things being produced today but also ability to continue the production tomorrow

- Therefore there must be a **balance** between today and tomorrow

Many years ago...

Let’s imagine you are working in a factory...

Many years ago...

FACTS

- Any machine can be working very well without any maintenance for some time
- Any machine can be even the best for a while (because when the other machines are being maintained it can be is still working…)

Many years ago...

CONTEST FOR EMPLOYEES :

The one who will produce that biggest amount of the items is the winner

Many years ago...

WINNER OF THE CONTEST WAS...

... engineer who was running his machine 24/7 without any maintanance...

Many years ago…

- Later YOU as new (promoted) owner becomes responsible for such a “best” machine that „won” the contest
- But it breaks more and more often…
- Finally it needs to be **replaced** with a new one… what is more expensive that all savings “achieved” by **lack of** **continues maintenance** in the past

Production and Production Capability cannot be considered a separated items

Production Capability
(Asset)

Product (Result)

NOWADAYS

Is there any difference in our industry?

What is a good product in software development?

- Software that is working as expected and solves business problems
- Clients can use it efficiently
- The number of functionality defects is low
- Software is scalable if the number of input data grows
- The revenue taken is bigger then cost of building it
- …

What is production capability in software development?

- Fact : Software very rarely stays as it is

- The ability to extend the software, fix the defects is purely related to its technical state
  - Readability
  - Testability
  - Reusability
  - Complexity
  - Design
  - Maintainability
  - Extendibility
  - Performance

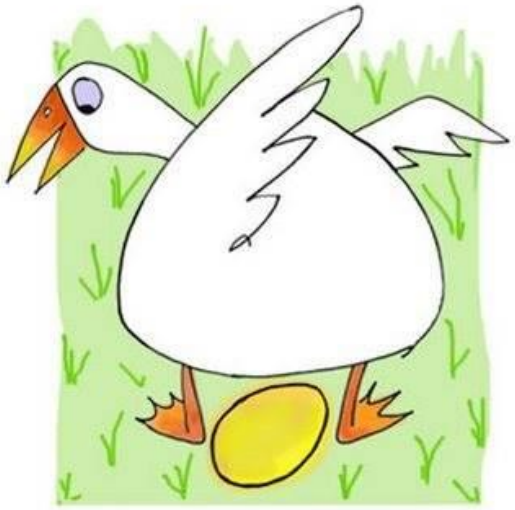
- Production capacity is ability to continue working with the existing software.

What is production capability in software development?

- Second part of production capacity is developer's
  - Knowledge
  - Skills
  - Desire to grow

Principle of balance

Production and production capibility needs to grow together



Balance in software development

Refactoring is a technique that can allow you to keep the
balance when working in software industry

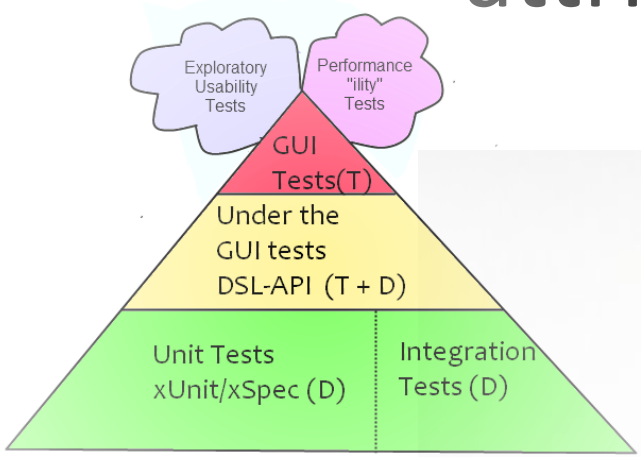
What is correct definition of “refactoring”?

Is it only “*changing code structure without changing its behavior*”?

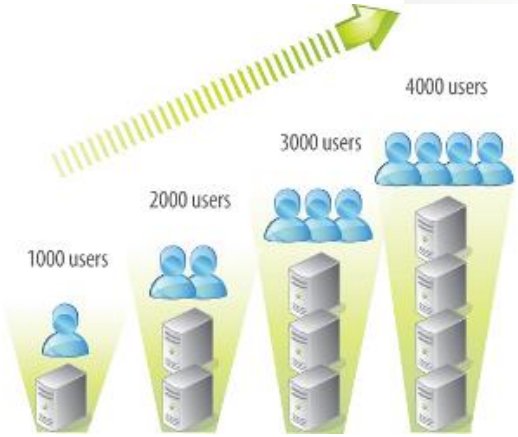
Refactoring definition

- Code refactoring is a "disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior", undertaken in order to **improve** some of the **nonfunctional attributes** of the software.
- Refactoring is not
  - New functionality
  - Fixing bugs
- Although it
  - **follows** them or
  - **proceeds** them

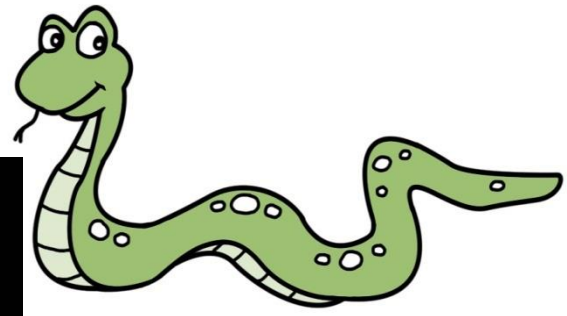
What are non-functional attributes of software?

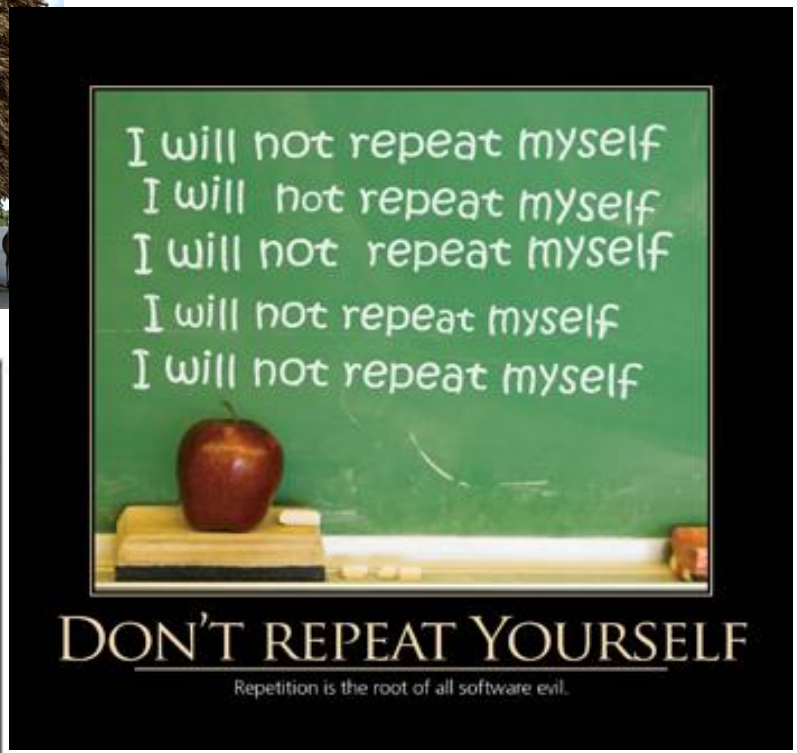


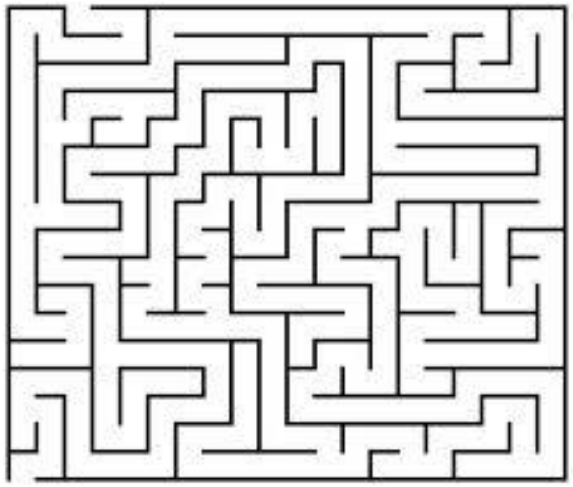
Readability



When do we refactor?

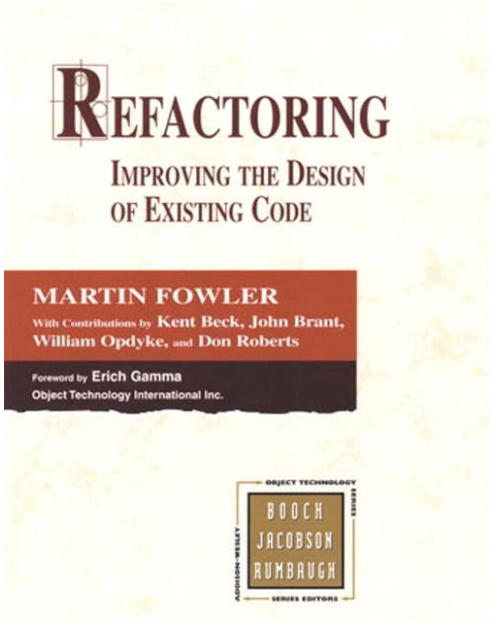




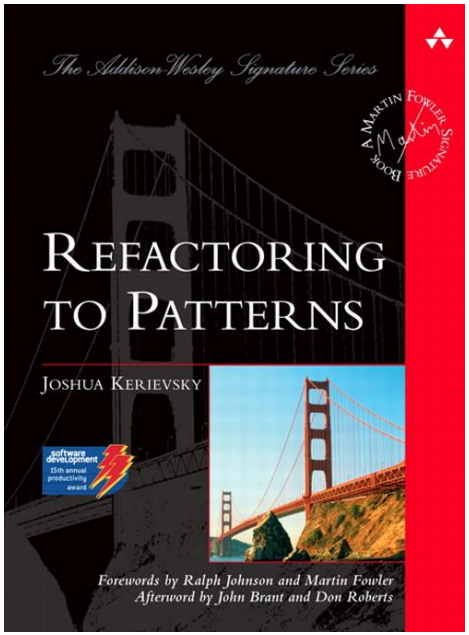


3 Books

It's like Encyclopedia...



Turns it into Use cases....



Presents

End to end flow

A very simple refactoring...

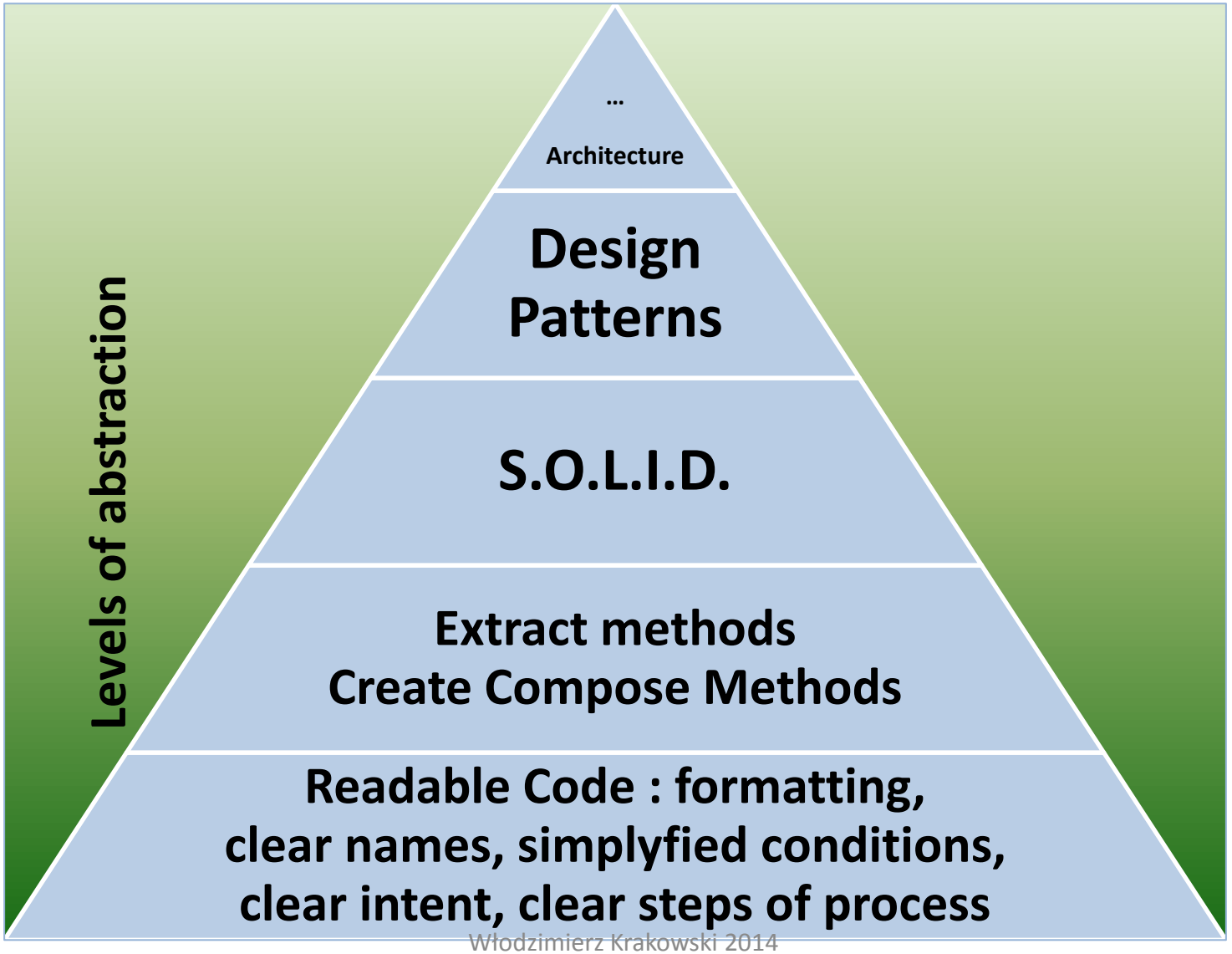
Before

```
double getPrice()
{
    double basePrice = quality * itemPrice;
    double discountFactor;
    if (basePrice > 1000)
        discountFactor = 0.8;
    else
        discountFactor = 1.0;
    double tax;
    if (specialItem)
        tax = 0.08;
    else
        tax = 0.23;
    return basePrice * discountFactor * tax;
}
```

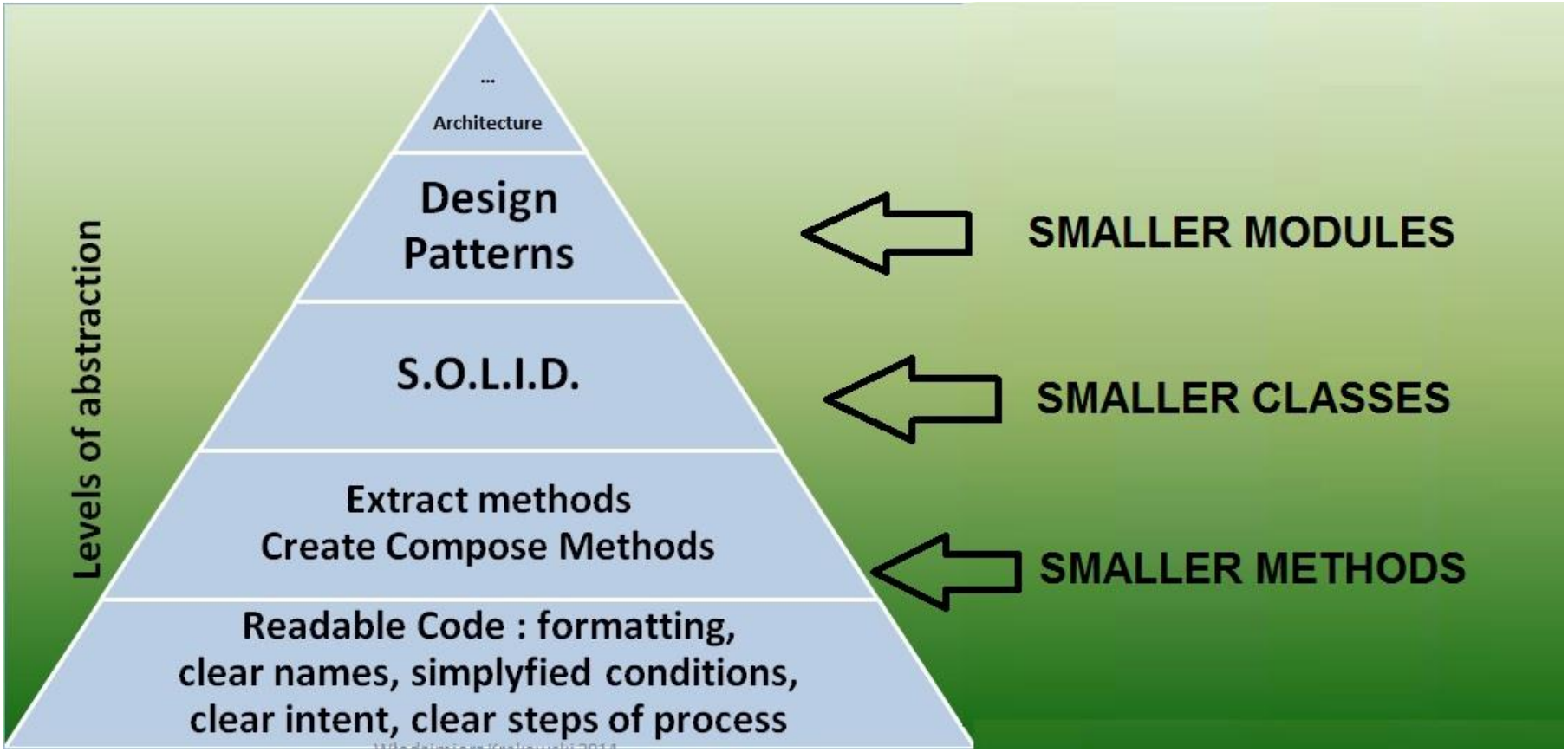
After

```
double getPrice()
{
    return getItemPrice() * getQuantity()
            * getTax() * getDiscount();
}
```

Pyramid of Refactoring



Pyramid of Refactoring



Why is it better to have smaller pieces?

SMALLER SMALLER SMALLER SMALLER SMALLER SMALLER

at least up to some point it is :

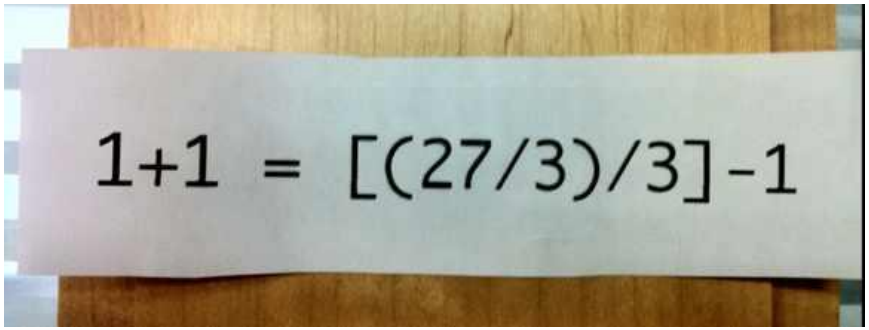
- Easier to read
- Easier to understand
- Easier to divide into related parts / groups
- Easier to move to other classes
- Easier to test
- Easier to reuse !!!

Code segregation vs. garbage segregation

It might seem funny to compare them…

When to stop : once Simple Design is achieved

1. Passes all of the tests.
2. Communicates the programmer's intentions, i.e., has good names for every important concept.
3. Expresses everything once and only once, i.e., it duplicates no code or logic or knowledge.
4. Includes nothing that's unnecessary.



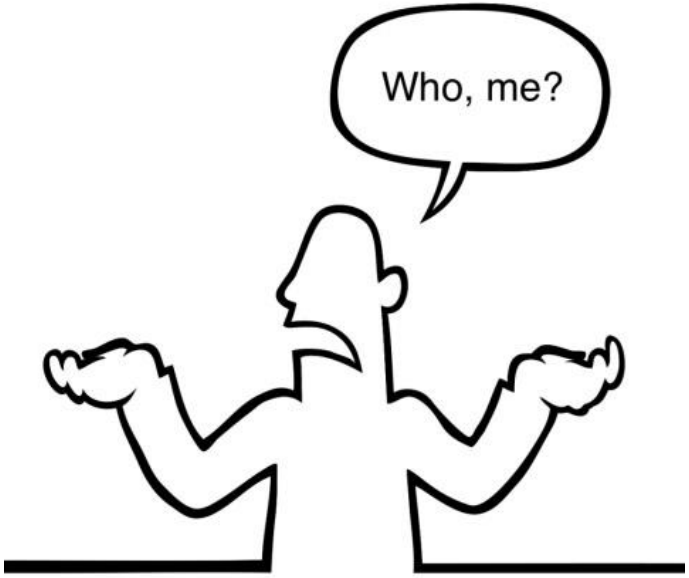
Brainstorming…

What to do to make refactoring **happen** every day in my team?

Answers...

- We need agreement from management
- We need code Reviews
- Clients must understand it
- It must become part of stories
- There must be time
- There must be knowledge
  - Tests are needed
  - We need to prioritize it
  - The architect needs to decide ...
  - ....
  - **How poor we are... (!!!)**

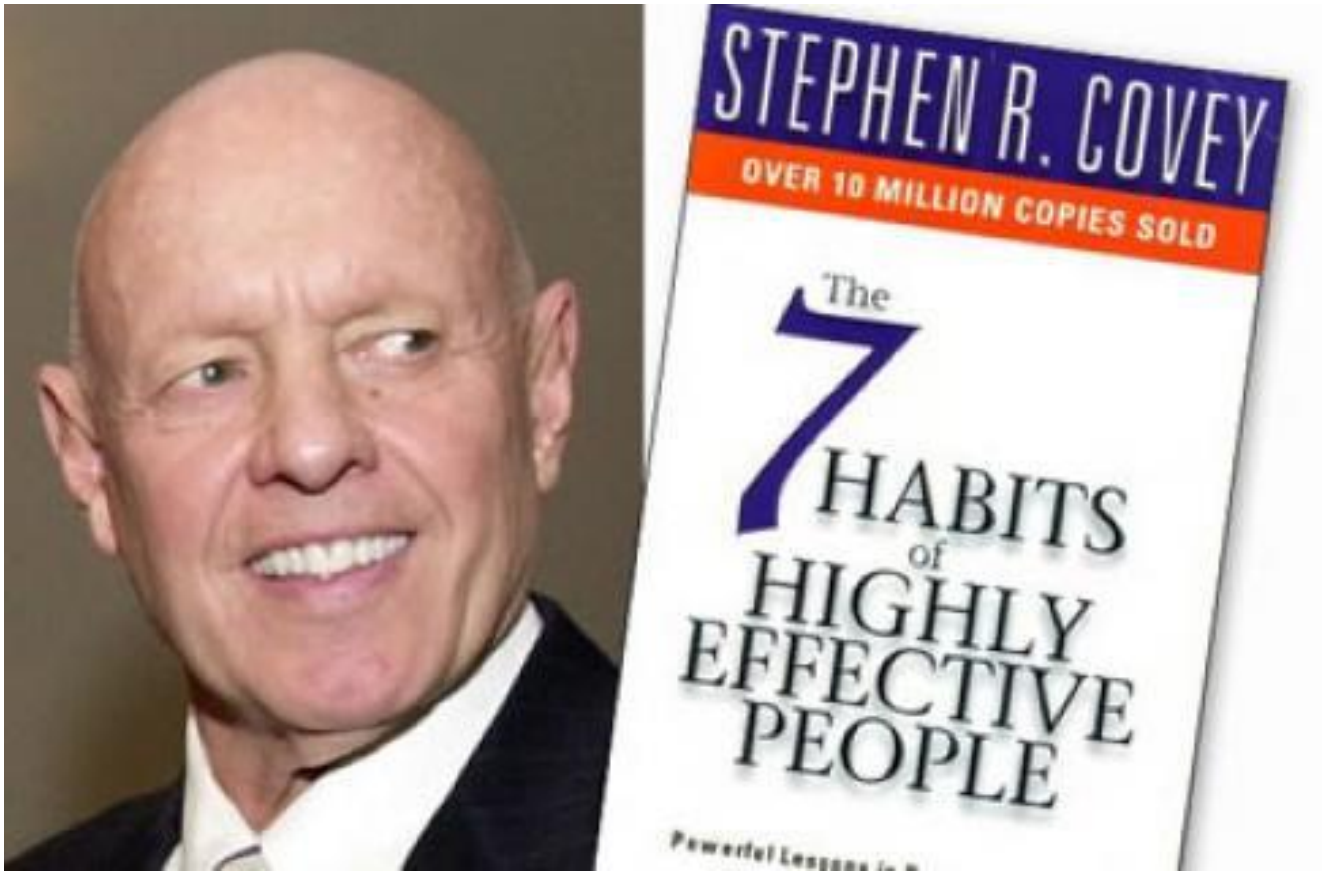
Were these results usefull?



What to do to make refactoring **happen** every day in my team?

What is wrong with this question?

How to take care of effectiveness?



It is not a book about developing software but …

Habit 1: Be Proactive

- The are two options
  - **Be proactive** – you are/become the leader of the change
  - **Be reactive** - you need to follow behind / adjust to the change

- Therefore using refactoring
  - You can be **driver of the change**
  - Or you have to **accept what is going on** (e.g. there is no refactoring at all in the team)

- Be part of the solution – not part of the problem

Habit 1: Be Proactive

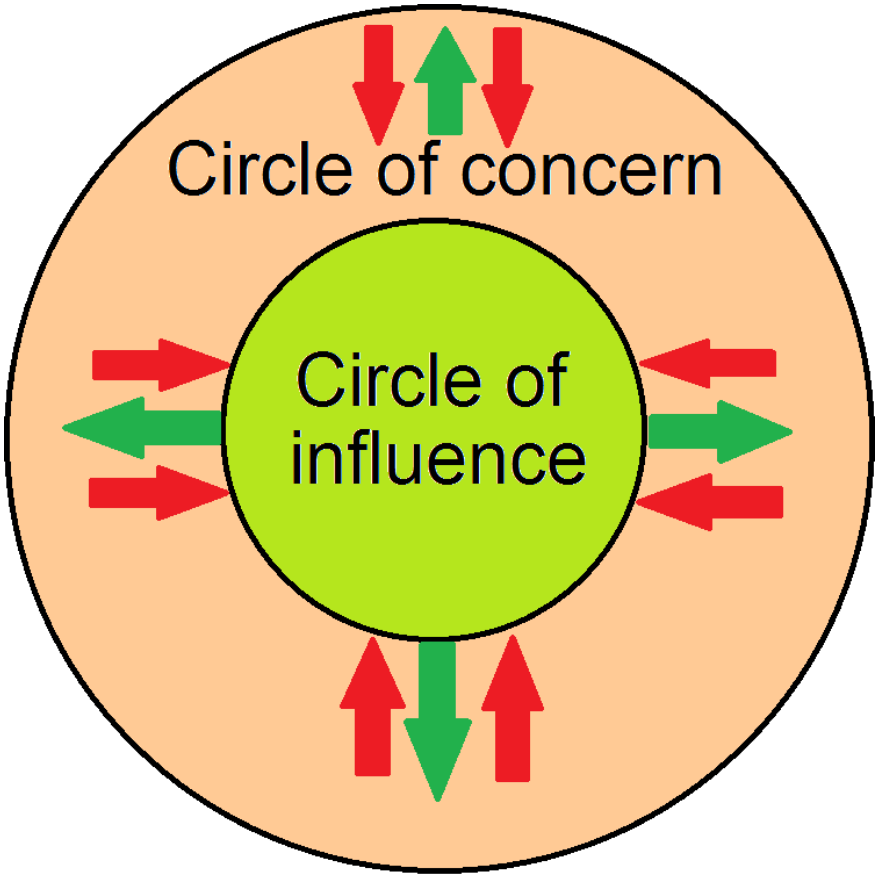
Laugh at it and live with it

You get rid of responsibility

Laugh at it but also take action to solve the problem

You take the responsibility

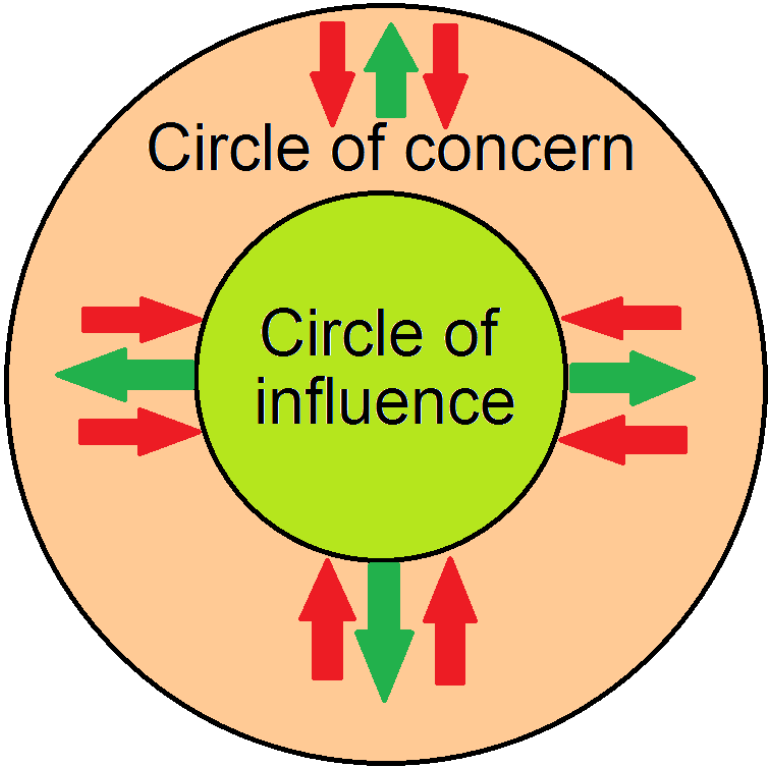
Habit 1: Be Proactive

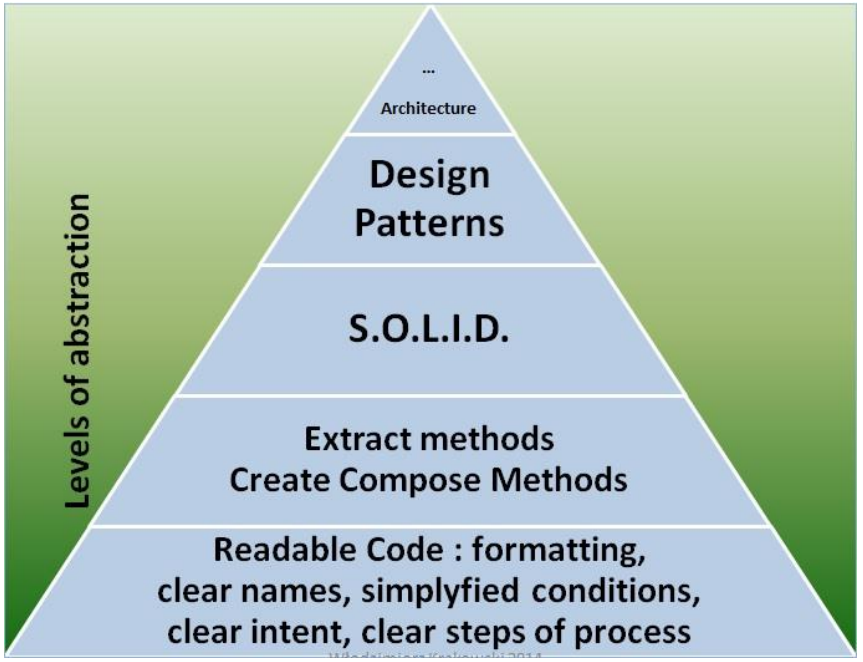


There are always things we can influence…

So start from small refactorings in your daily code!

Pyramid of Refactoring again!





Why aren’t we proactive?

The role of the victim role sounds better?

„It’s not me, it’s the team”

We don’t believe in step by step approach... And therefore do not even take the first step !!!

Why aren’t we proactive?

Maybe we prefer even not to try to avoid failure...

Isn’t it always safe to say „We as a team failed... Shame on us”

Habit 2: Begin with the End in Mind

- Improvements and inventions are usually created twice
  - First time in your mind (leadership)
  - Second time in reality (management)

- Whenever you want to achieve something answer the questions :
  - **What** is my goal?
  - **Why** I should do this?
  - **How** will I do this?

- Be aware of the difference between leadership and management

Habit 2: Begin with the End in Mind

What do you want to do?

- Keep the code at one level of abstraction?
- Keep things separated in different classes?
- Replace conditionals with Strategy Design Pattern?
- Replace conditionals with Polymorphism / Subclasses?
- Get rid of duplication?

Habit 2: Begin with the End in Mind

Why do you want to do this particular code transformation?

- Readability
  - So we save a lot of time
- Extendibility
  - We we add new functionality
- Testability
  - We can can be sure about our solution
- Performance gain
  - So our clients are not frustrated
- …

Habit 2: Begin with the End in Mind

How are you going to do this?

- Automated refactorings from IDE (IntelliJ/Eclipse)
- Small steps
- Two simultaneous solutions for a while?

How do you know you are on track?

- Do you have test coverage?

How will you know you are done?

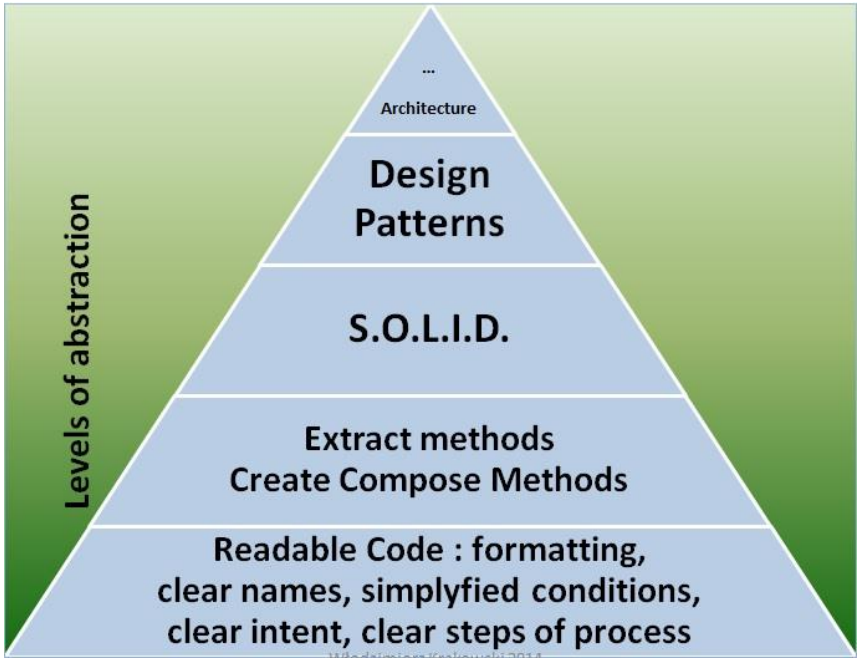
Leadership vs Management

There is a natural relation between
patterns and refactorings.

DESIGN PATTERNS are where you want to be,
REFACTORINGS are ways to get there from somewhere else.

- Martin Fowler

Pyramid of Refactoring again!



Find Your Own Motivator in order to answer these questions

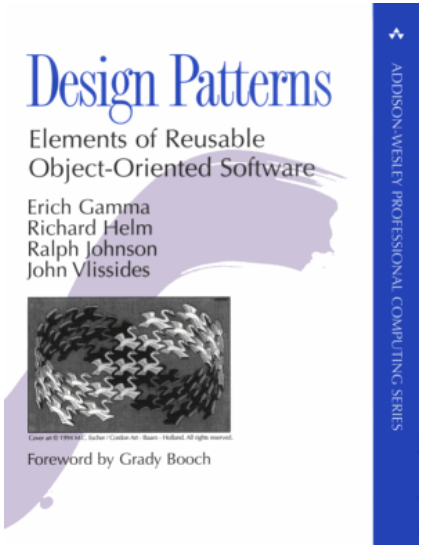
You need to be driven by yourself from inside

Refactoring strategies

Make everything as simple as possible, but not simpler.

Albert Einstein

- Refactor to understand
- Separate things that change from things that don't change



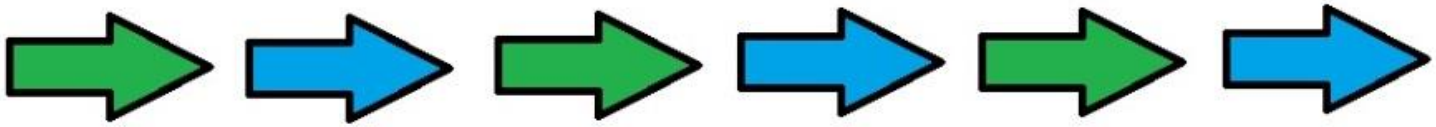
Refactoring strategies

Don’t reinvent the wheel

Heuristics for quickly finding where/what code that should be refactored – Code Smells

- Code duplication
- Long methods
- Large classes
- Data clumps
- Primitive obsession
- Shotgun surgery
- Alternative classes with different interfaces
- …

Common perception of tasks to do



CODING REFACTORING

What is wrong with sequential perception of items to do?

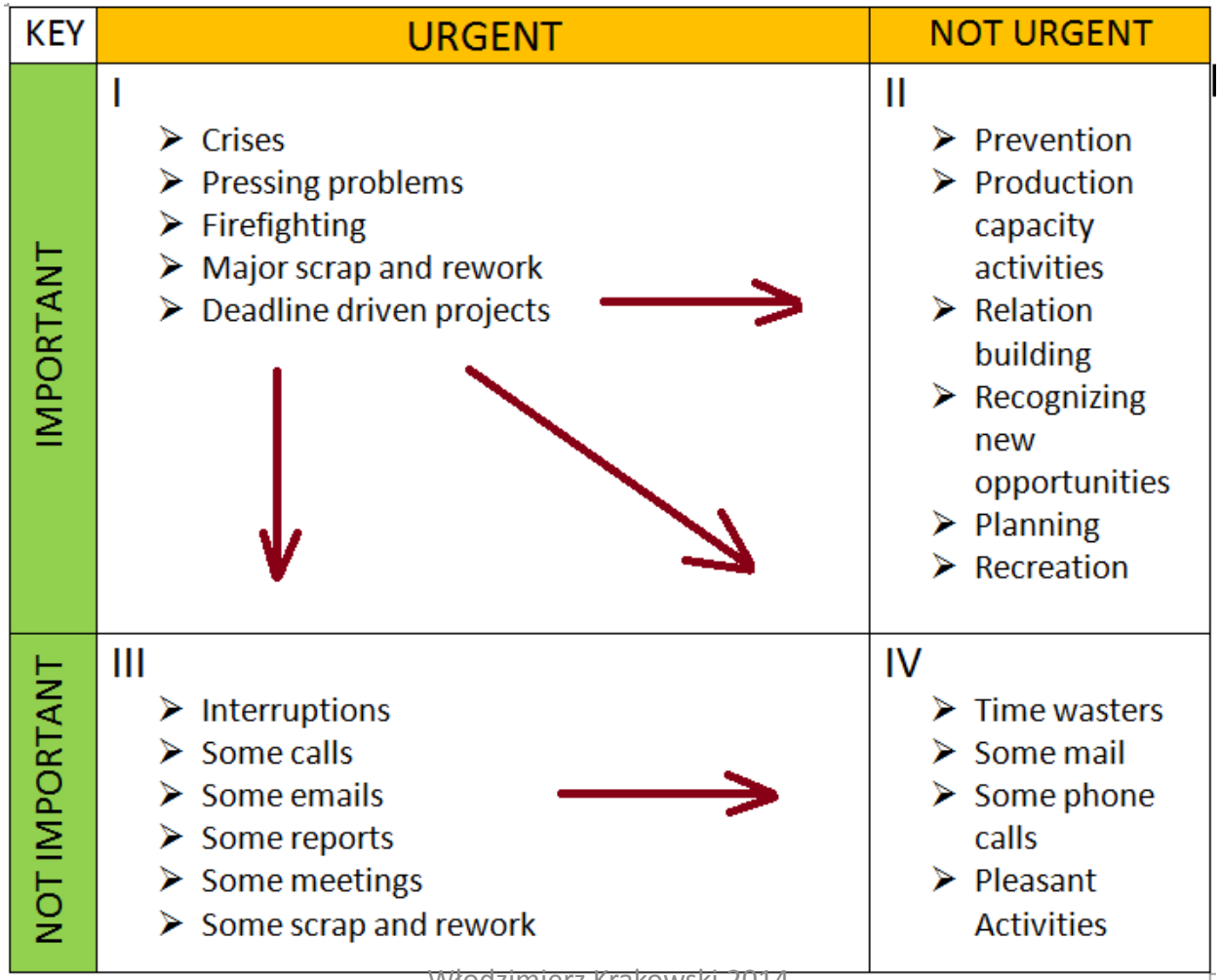
Habit 3: Put First Things First (importance before urgency)

How do we spend time?

KEY	URGENT	NOT URGENT
IMPORTANT	I ➢ Crises ➢ Pressing problems ➢ Firefighting ➢ Major scrap and rework ➢ Deadline driven projects	II ➢ Prevention ➢ Production capacity activities ➢ Relation building ➢ Recognizing new opportunities ➢ Planning ➢ Recreation
NOT IMPORTANT	III ➢ Interruptions ➢ Some calls ➢ Some emails ➢ Some reports ➢ Some meetings ➢ Some scrap and rework	IV ➢ Time wasters ➢ Some mail ➢ Some phone calls ➢ Pleasant Activities

Habit 3: Put First Things First (importance before urgency)

What happens if we do not spent time on prevention?



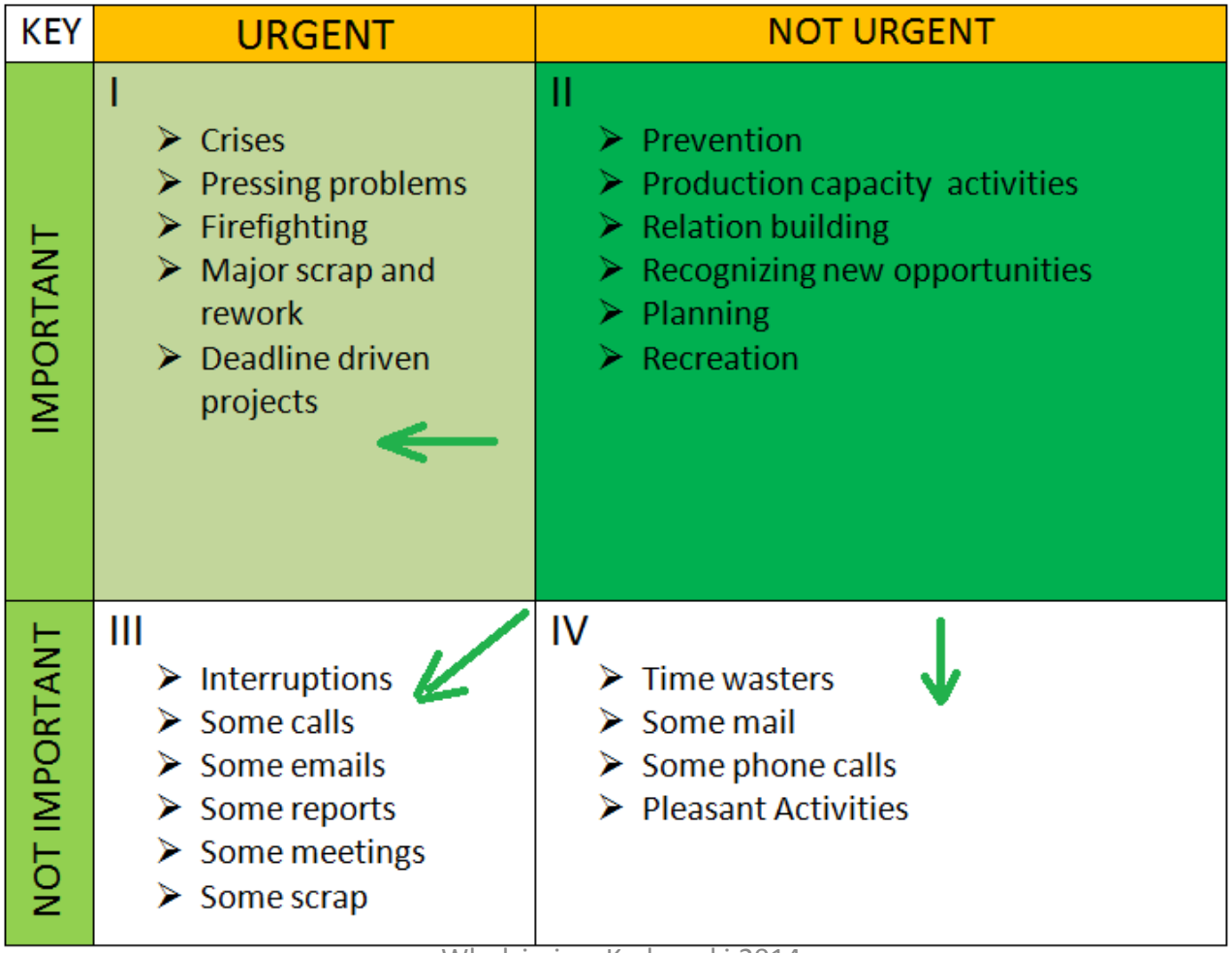
Habit 3: Put First Things First (importance before urgency)

What each quadrant symbolizes?

KEY	URGENT	NOT URGENT
IMPORTANT	I Quadrant of Demand	II Quadrant of The Zone
NOT IMPORTANT	III Quadrant of Illusion	IV Quadrant of Escape

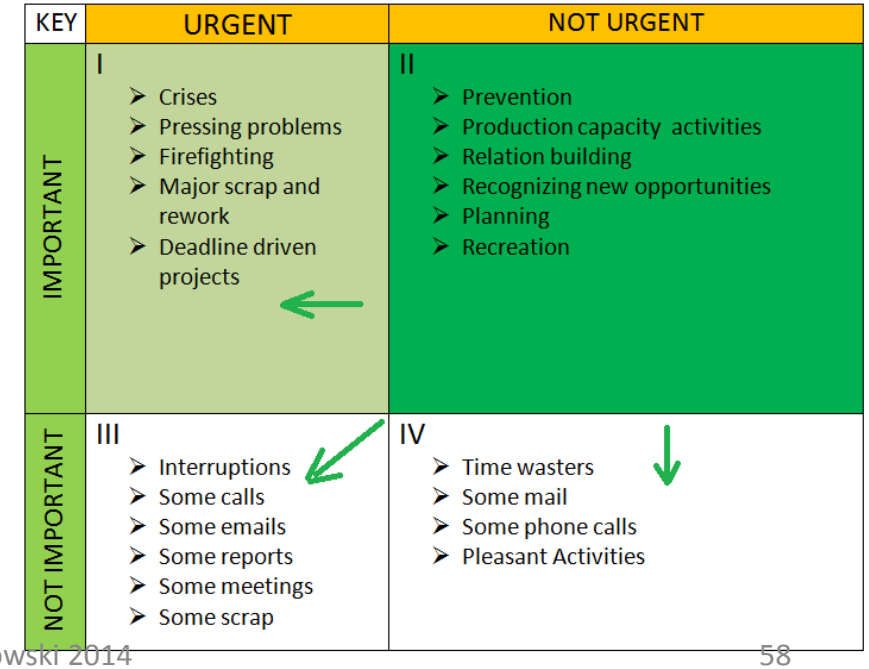
Habit 3: Put First Things First (importance before urgency)

Time for refactoring comes only it is treated as important item



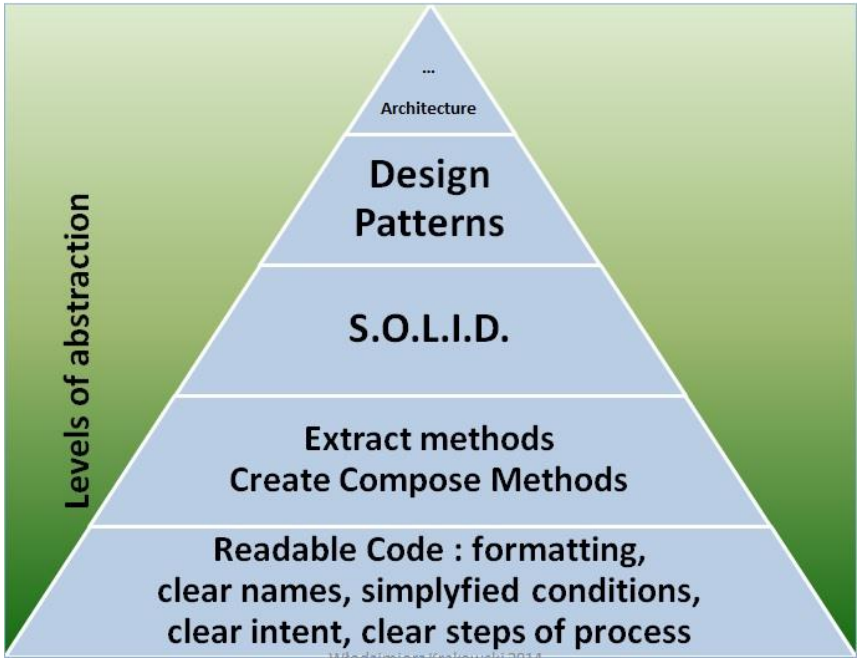
Habit 3: Put First Things First (importance before urgency)

- Do we have to find the time for refactoring?
  - It is not about finding time.
- Do we need to have permission (e.g. from manager) to do the refactoring?
  - Usually it is not about any permission especially if it is everyday practice…
- What is TDD?
  - **Test**
  - **Code**
  - **Refactor**
- Are we doing full cycle of TDD?
- How to "find" time for refactoring?
  - Treat it as **important item**



Pyramid of Refactoring again!

KEY	URGENT	NOT URGENT
IMPORTANT	I ➢ Crises ➢ Pressing problems ➢ Firefighting ➢ Major scrap and rework ➢ Deadline driven projects	II ➢ Prevention ➢ Production capacity activities ➢ Relation building ➢ Recognizing new opportunities ➢ Planning ➢ Recreation
NOT IMPORTANT	III ➢ Interruptions ➢ Some calls ➢ Some emails ➢ Some reports ➢ Some meetings ➢ Some scrap	IV ➢ Time wasters ➢ Some mail ➢ Some phone calls ➢ Pleasant Activities



Why don’t we focus on important?

Additional IF is urgent and important ...

```
if (white)
{ // 10 lines of code}
else if (black)
{ // 10 lines of code}
else if (grey)
{ // 10 lines of code}
else if (yellow)
{ // 10 lines of code}
else if (red)
{ // 10 lines of code}
...
```

But refactoring is only IMPORTANT...

```
interface ColorHandler {...}

class WhiteHandler
   implements ColorHandler {...}
...

Map<Color, ColorHandler> handlers = ...
handler = handlers.get(color);
...
handler.handle(data);
```

Refactoring strategies

<u>Choose</u> the most appropriate time for refactoring

- Extend then refactor
- Refactor then extend
- Debug/fix then refactor
- Refactor then debug/fix

Habit 4: Think Win-Win

What is the problem with competition?

Somebody must lose in order for somebody to win...

Habit 4: Think Win-Win

How does it relate to refactoring?

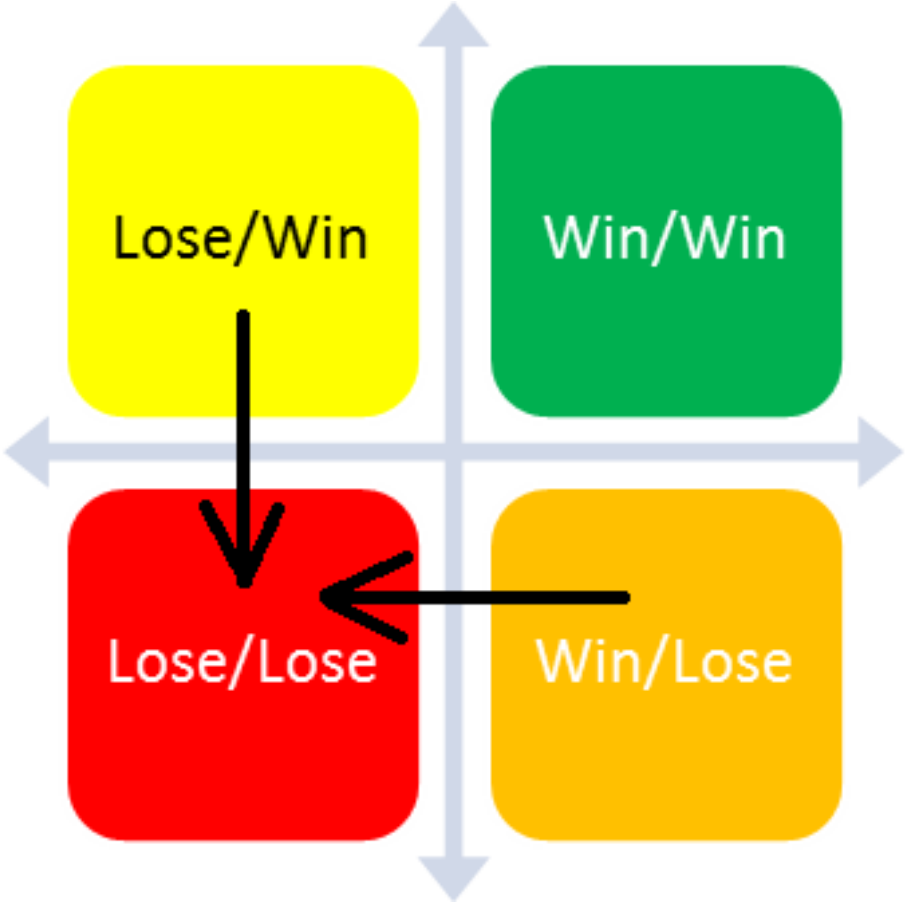
There must be a balance.

- One "WIN" refers to **business quality**
- The other "WIN" refers to **technical quality**

These two must grow together

Otherwise one or the other side feels to be abused and that drives into the conflicts and lack of cooperation.

Habit 4: Think Win-Win

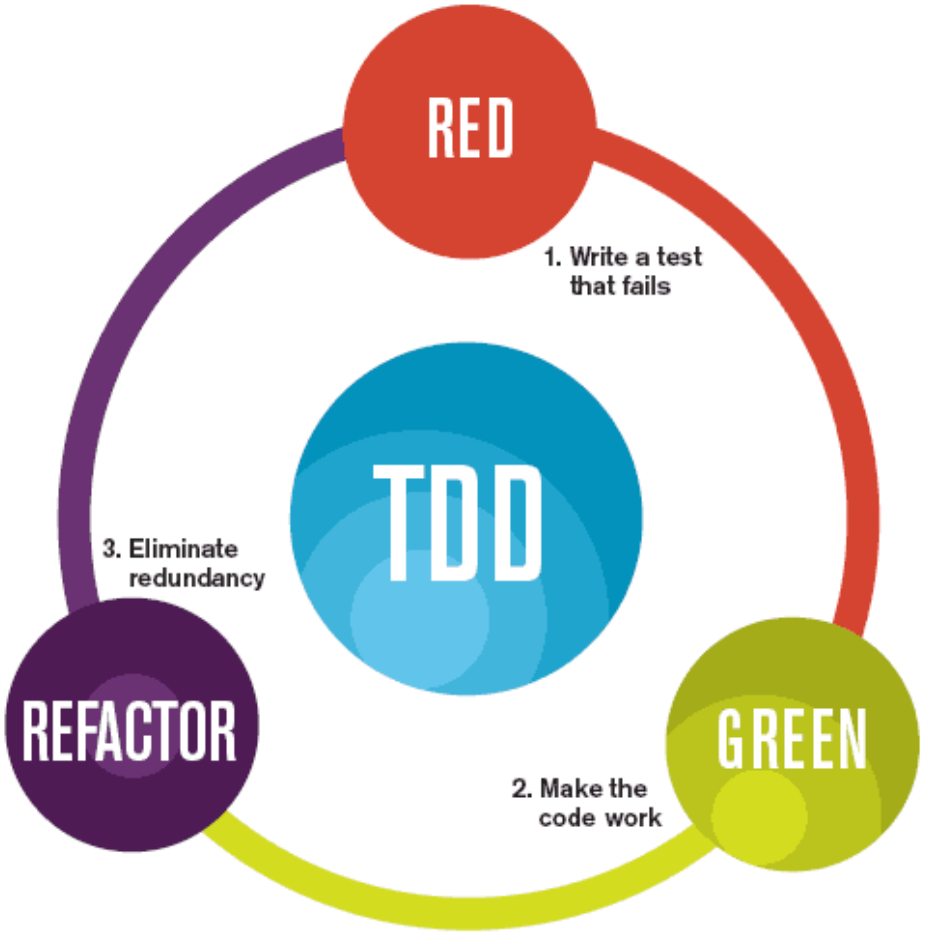


- But in case of relations between people / inside teams the "rat's race" approach is not working…

Habit 4: Think Win-Win

- What is TDD (again)?
  - **Test**
  - **Code**
  - **Refactor**

- **It is nothing less or more then the win-win strategy between functional and technical quality**



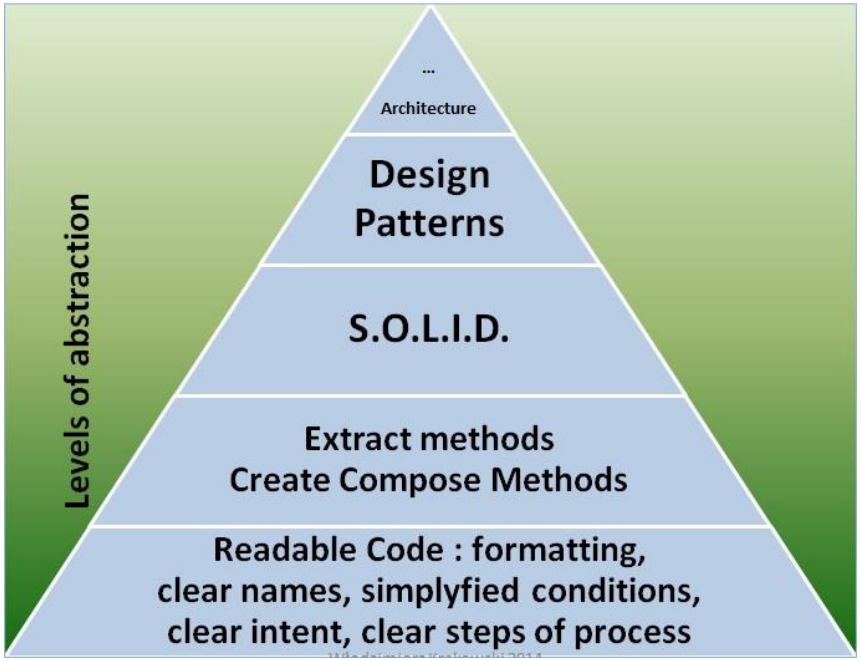
The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Habit 4: Think Win-Win – one more key point

I	You	Alternative
Win	Lose	None
Lose	Win	None
Win	I don't care	None
Lose	Lose	None
Win	Win	None
Win	Win	Or… no deal at all

REMEMBER : *nobody enforces you to stay* if win-win is not possible…

Pyramid of Refactoring again!



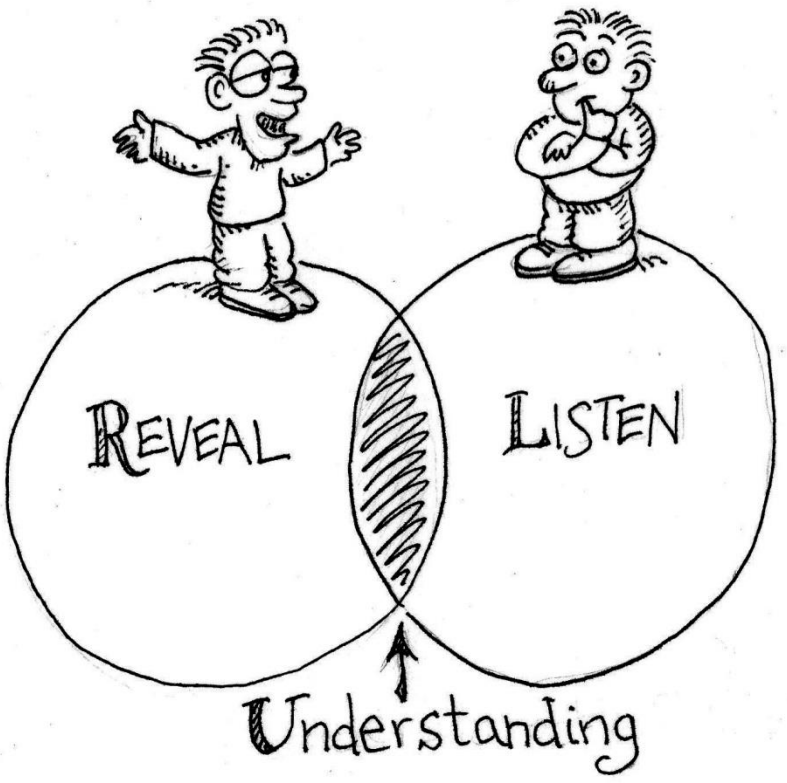
Habit 5: Seek First to Understand, Then to be Understood

Let's imagine two people convincing each other who is right

But what if they focused on listening first instead of talking first?

Habit 5: Seek First to Understand, Then to be Understood

Now let's imagine how each of them feels once they know are have been understood by the other party…



… after a while they want to listen to the other person as well because they feel safe instead of being attacked

Habit 5: Seek First to Understand, Then to be Understood

How does it relate to refactoring?

- Do not enforce your solution at the beginning
- First **understand** the code, the design, the needs, the flow **before you take any actions**

Habit 5: Seek First to Understand, Then to be Understood

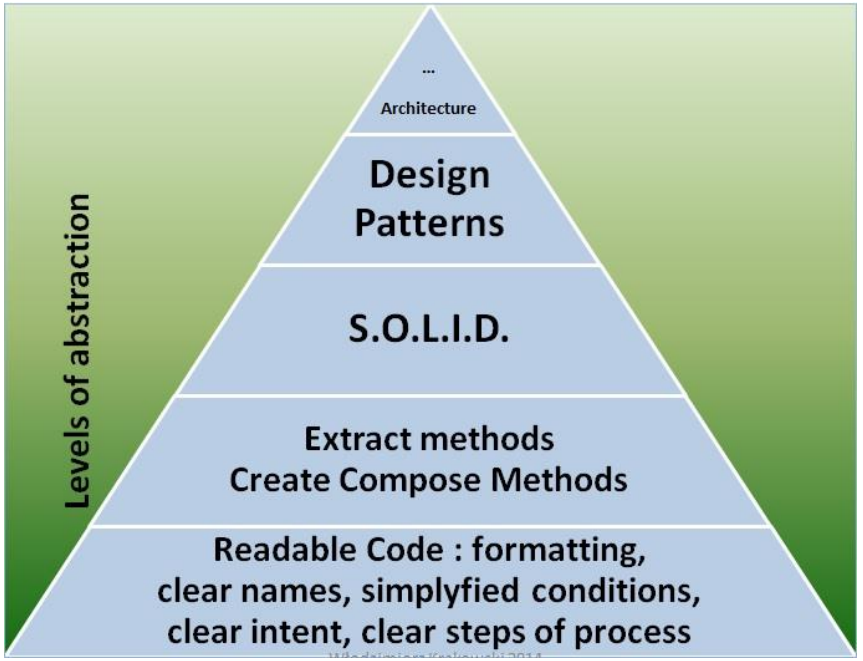
Do not refactor everything that is around you just because now you think your solution is better and the other solution's is worse

Habit 5: Seek First to Understand, Then to be Understood

But once you understand the others then seek to be understood by them as well (win-win)

Teach the others how to refactor in order **to improve** your refactoring skills

Pyramid of Refactoring again!

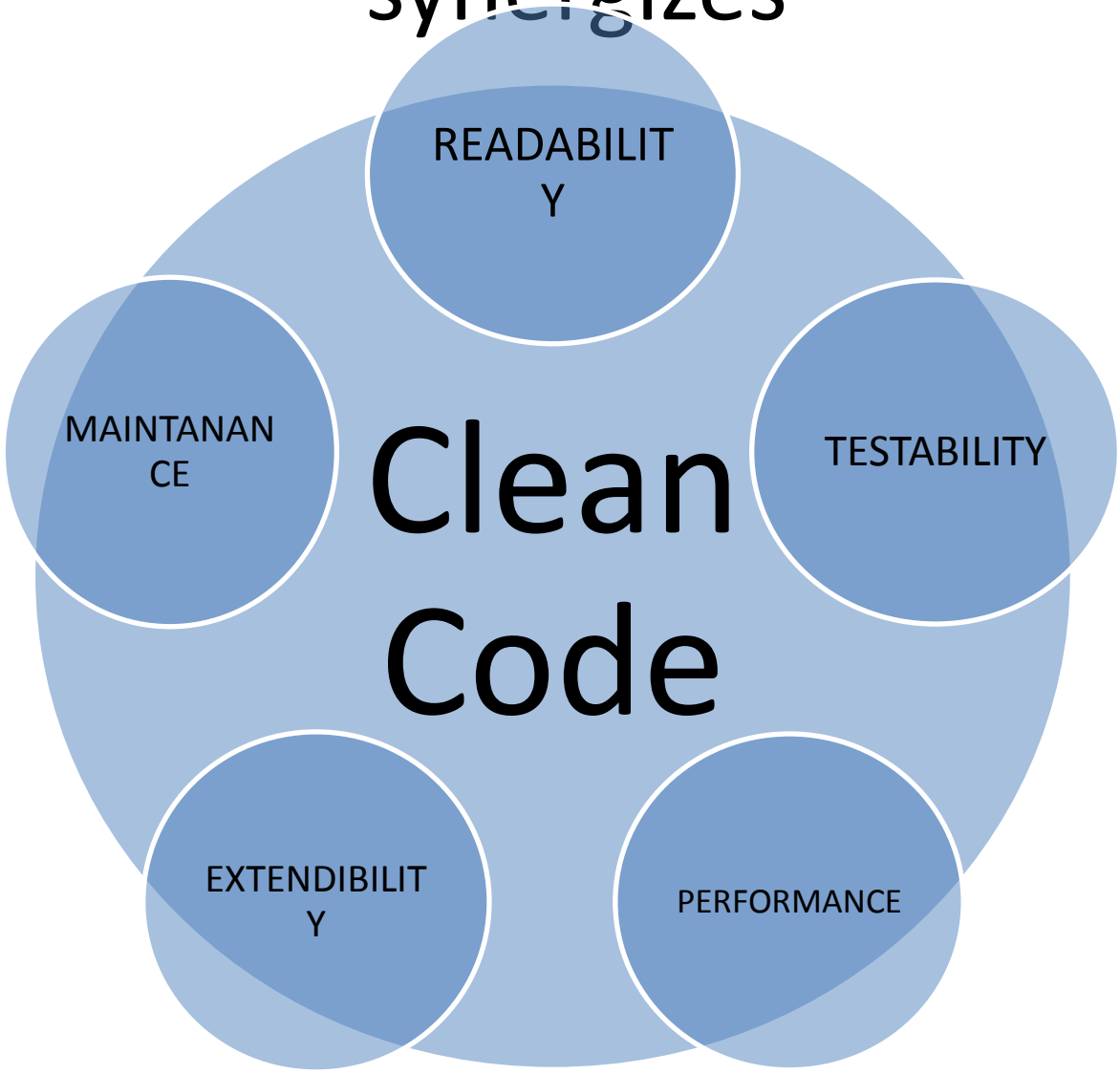


Habit 6: Synergize

- **Synergy happens when the whole system is greater that sum of its components**

- **What does it mean in reality?**
  - It is seeking for 3rd Alternatives
  - Maybe not my solution
  - Maybe not your solution
  - But 3rd solution instead that is better then any solution each of us thought separately

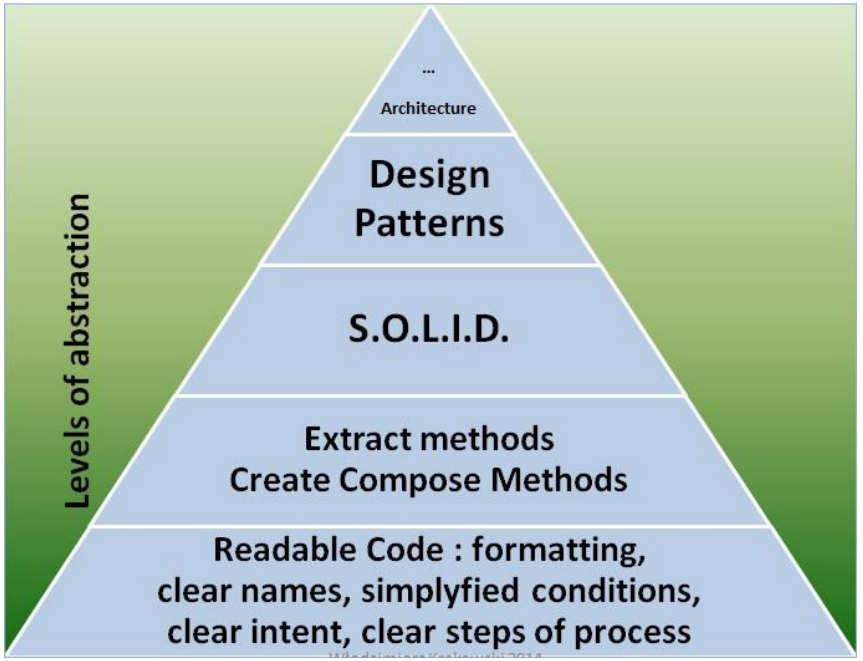
Habit 6: Refactoring – each item synergizes



Habit 6: Refactoring together is a big synergy

- **Let's do a code review before refactoring given code**
- **Let's review after the refactoring if it the improvement is indeed visible**
- **"No one can write good code at first, but we can find faults in other's code really well" (Venkat Subramaniam)**

Pyramid of Refactoring again!



Habit 7: Sharpen the Saw

Do you know the story about a man cutting off a tree with blunt saw and having no time to sharpen it?

Sharpen the Saw means preserving and enhancing the greatest asset you have - you.

Stephen R Covey

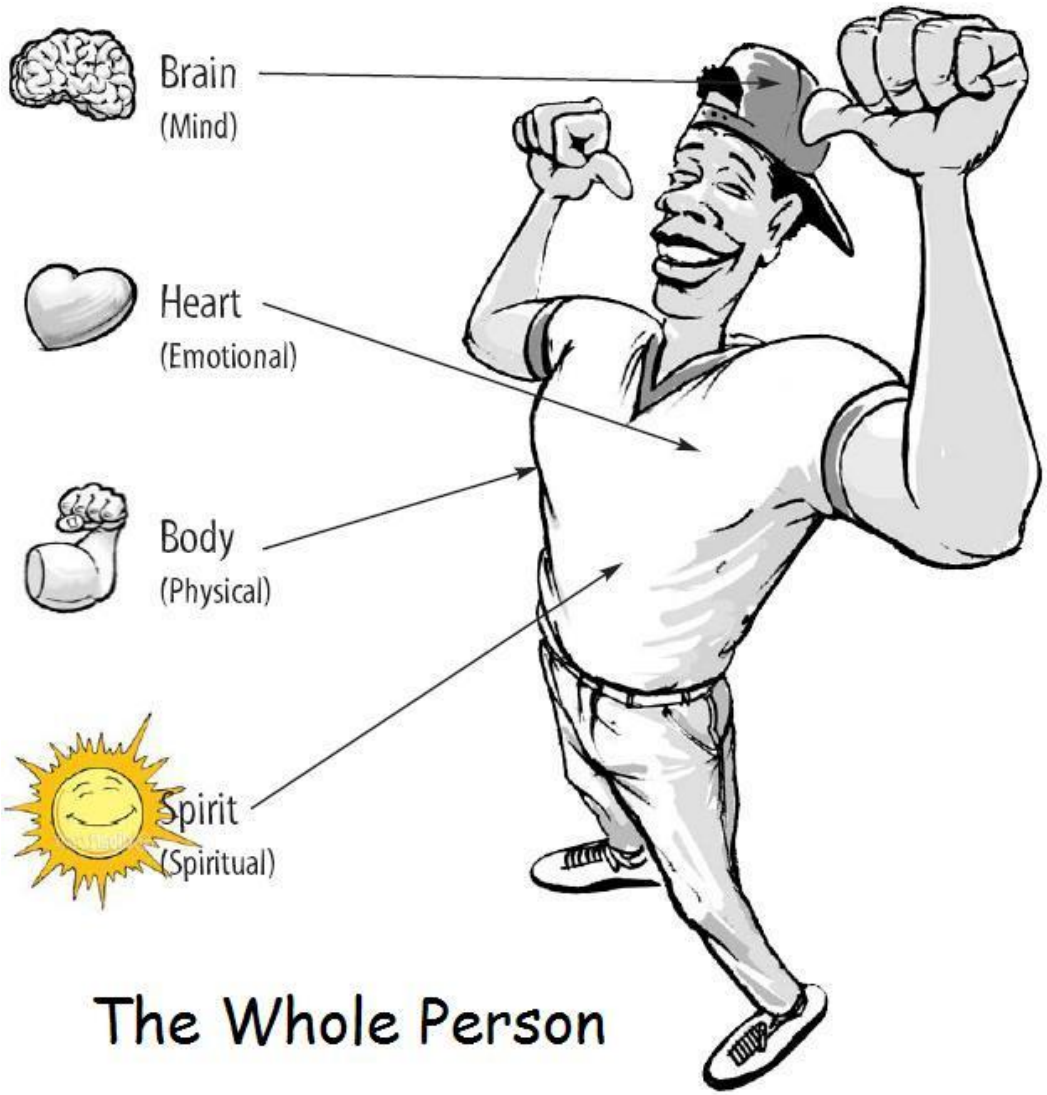
Habit 7: Sharpen the Saw

How does it refer to refactoring then?

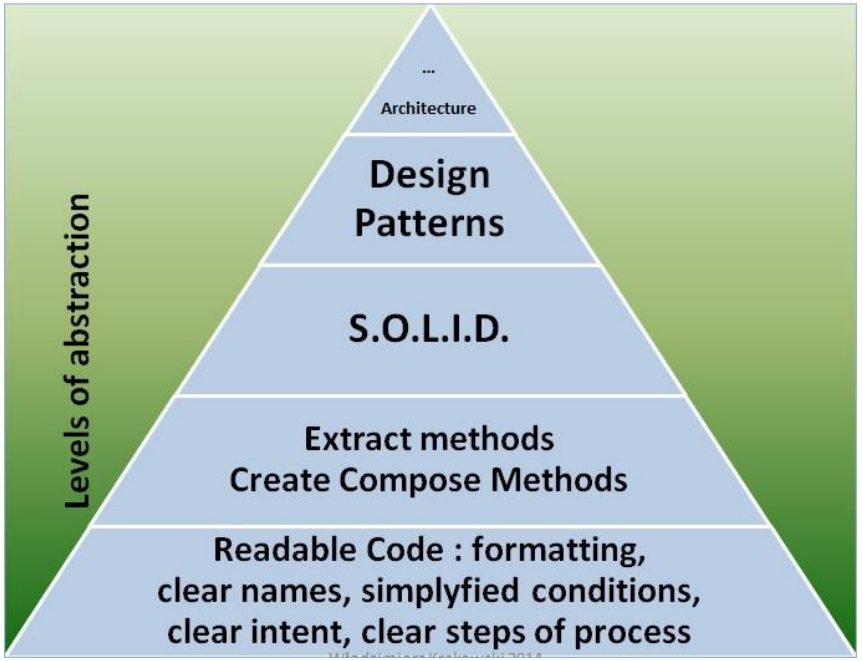
But nothing more?

Habit 7: Sharpen the Saw

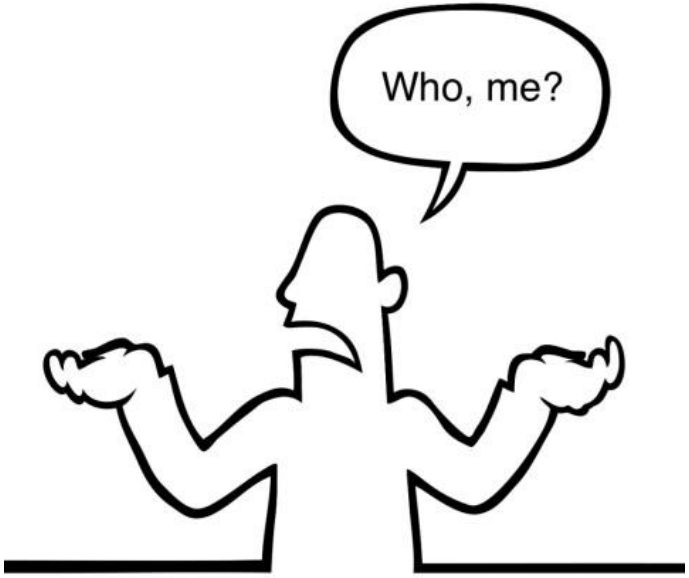
Habit 7: Sharpen the Saw



Pyramid of Refactoring again!



One more brainstorming…



What I can do refactoring **happen** every day in my team?

Isn’t it a better question?

Answers...

- ... Just start doing it
- I can point the places to refactor once I see them
- I will ask/consult with the others dirty code I found
- I will show the refactoring I did yesterday
- I can organize a training to share my skills / knowledge
- I will become a manager and entroduce it ☺
  - Lead / Teach fellow programmers
  - Brown bags
  - Watch presentations together
  - **We are not so poor any longer (!!!)**

7 HABITS OF EFFECTIVE REFACTORING

RENEWAL

MY EFFECTIVENESS

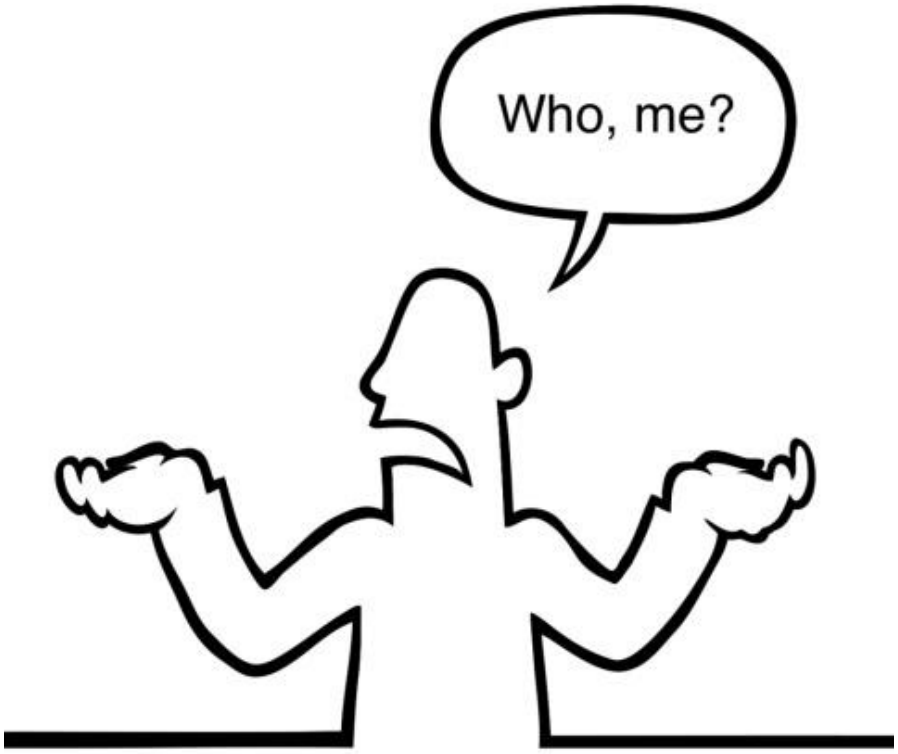
1. Be Proactive
2. Start with the End in Mind
3. Importance before Urgency

MY GROUP EFFECTIVENESS

4. Think Win-Win
5. Undestand before seeking to be undersood
6. Synergize

7. Sharpen the Saw

One more brainstorming…



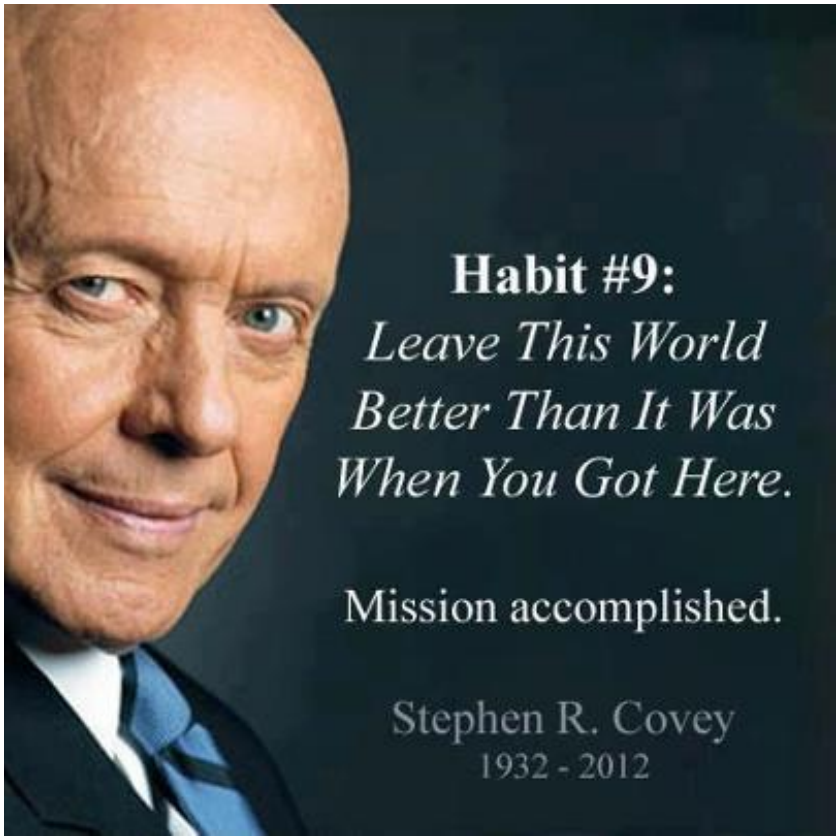
How seven habits of **effective refactoring** can help us achieve the brainstormed ideas?

Answers...

Do It Yourself!!!

BE PROACTIVE ☺

Make the world around you better all the time



"Leave the campground cleaner than you found it"

The Boy Scouts of America

How does it refer to refactoring?

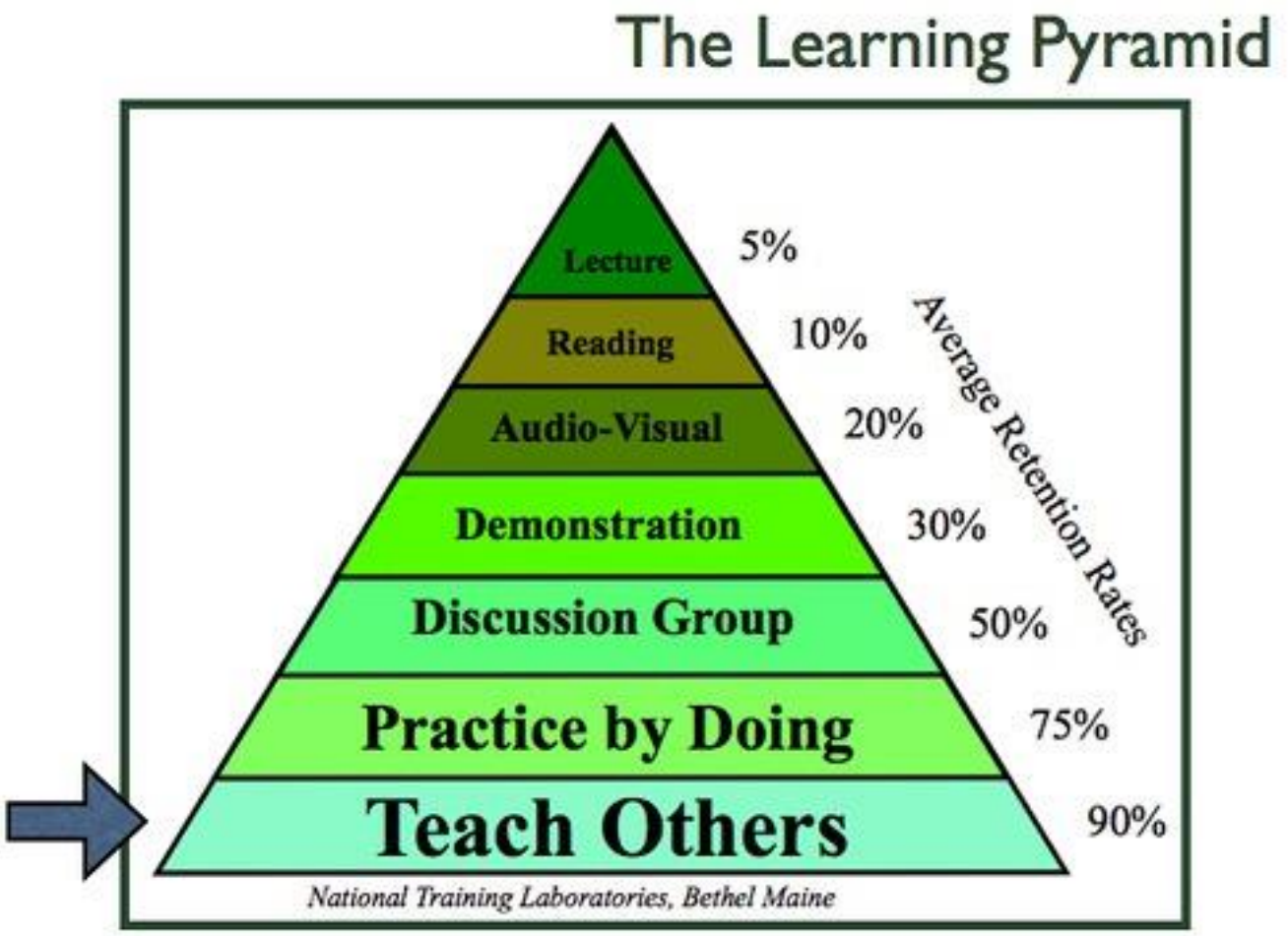
Check-in cleaner code than you checked it out!

Refactoring must become your habit

What is a habit: it is an action performed repeatedly and automatically, usually almost without your awareness

It is a merge of knowledge, skills and desire

Keep growing all the time!



You need to experience to understand

You need to collaborate & teach to grow

Thank you!