Natural Course of Refactoring.
A refactoring workflow.

Mariusz Sieraczkiewicz

@ms_bnsit_pl

http://msieraczkiewicz.blogspot.com

agile PRAGUE CONFERENCE

bns it

Why refactoring is like sex?

Natural Course of Refactoring

(adapted from http://www.thealmightyguru.com/Humor/Docs/ProgrammingIsLikeSex.html)

Why refactoring is like sex?

Once you get started, you'll only stop because you're exhausted.

One mistake and the consequences may be really huge.

(adapted from http://www.thealmightyguru.com/Humor/Docs/ProgrammingIsLikeSex.html)

Why refactoring is like sex?

People more talk about it than actually do.

You can do it for money or for fun.

It’s not really an appropriate topic for dinner conversation.

(adapted from http://www.thealmightyguru.com/Humor/Docs/ProgrammingIsLikeSex.html)

Why refactoring is like sex?

Beginners do a lot of noise about it.

Some people are just naturally good at it,

.. but some people will never realize how bad they are, and you're wasting your time trying to tell them.

bns it

Code readability

Natural Course of Refactoring

Why is it difficult to understand software after some time of its evolution?

.so!changesIntroDuCinG!maYBe,softwaRe,TosO,calLEdprOgRessIve,and.however.cHanGes!Modifies.usuallY,sTRucTureOFmaYbe,hoWEVeRtHecoDE,And!wHaTmaYbecumulatEdhowevEr.anD,mAkes.AnD,,and,the.LeSs!rEAdAblE,aNd.cOdeMAybe.ANd,!and,!Thenumber!of,sO,HoWEvErsodepeNdeNCiES,And.MAybeintErACtIOns!betWeenHoWevEr!differEntsyStem.moDules!inCreasESoThat,iTsO!morE.is.diffiCuLt,To,AnDuNdErstandmodIFy

Why is it difficult to understand software after some time of its evolution?

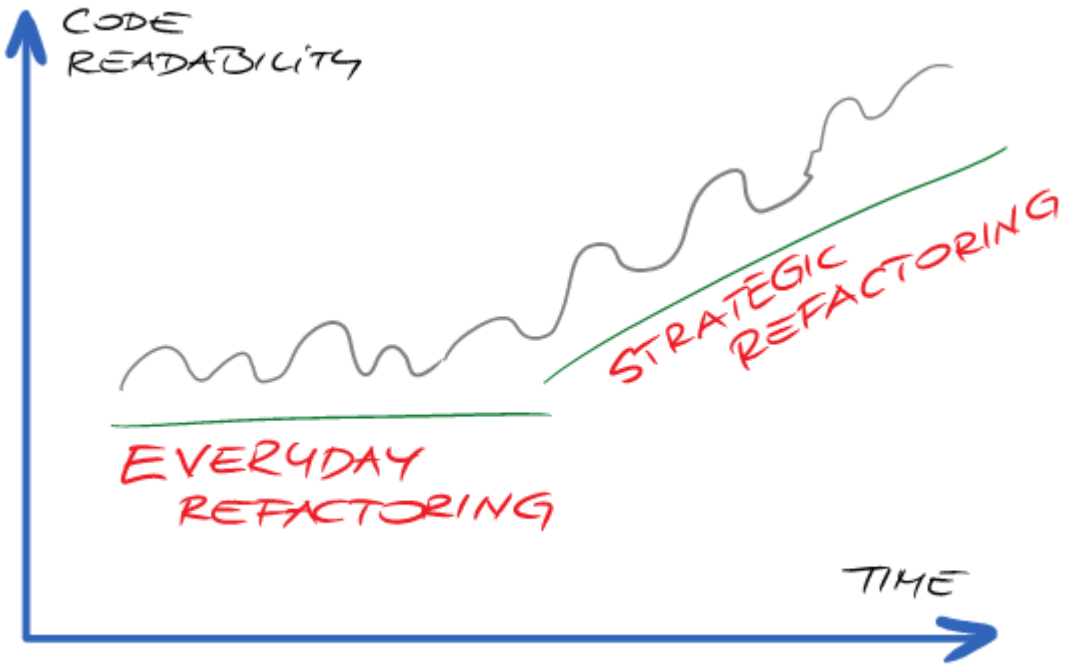
Introducing changes to software (so called progressive changes) usually modifies structure of the code, what cumulated makes the code less readable. The number of dependencies and interactions between different system modules increase, so that it is more difficult to understand and modify.

bns it

Flavours of refactoring

Natural Course of Refactoring

Two flavours of refactoring



Everyday refactoring

Everyday refactoring

Within a reach of every programmer

Can be done in minutes

Mostly safe, IDE-base automatic refactorings

For local health of the code

Part of programming practice

No excuse for not doing it

Strategic refactoring

A team longer term effort

Requires aligning with iteration planning

Generates items in backlog

Risky activity that requires intensive testing (including good tests suite)

Difficult and time-consuming

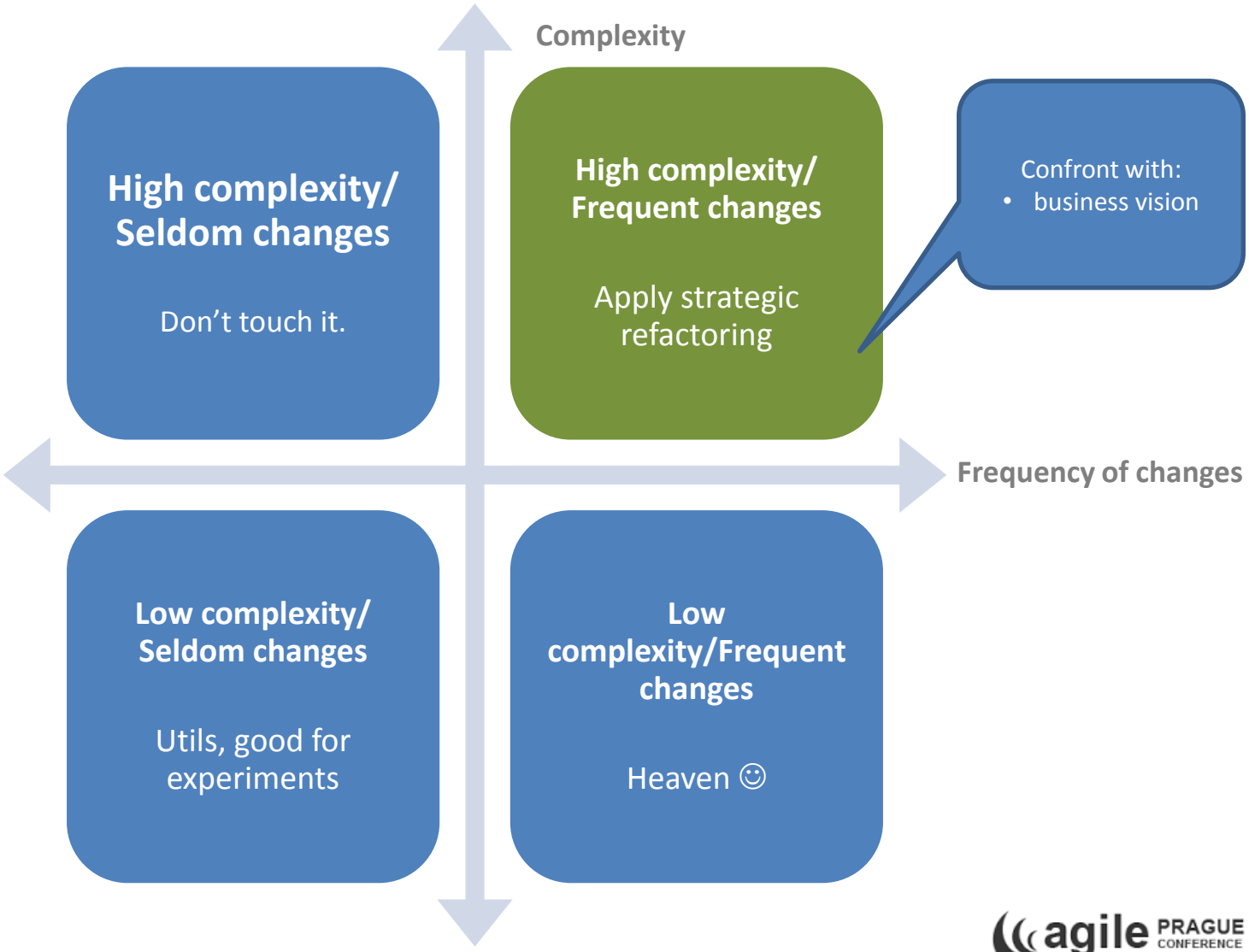
Check carefully if refactoring gives you enough value (Feather's Quadrant)

bns it

When should I do strategic refactoring?

Natural Course of Refactoring

Feather's Quadrant



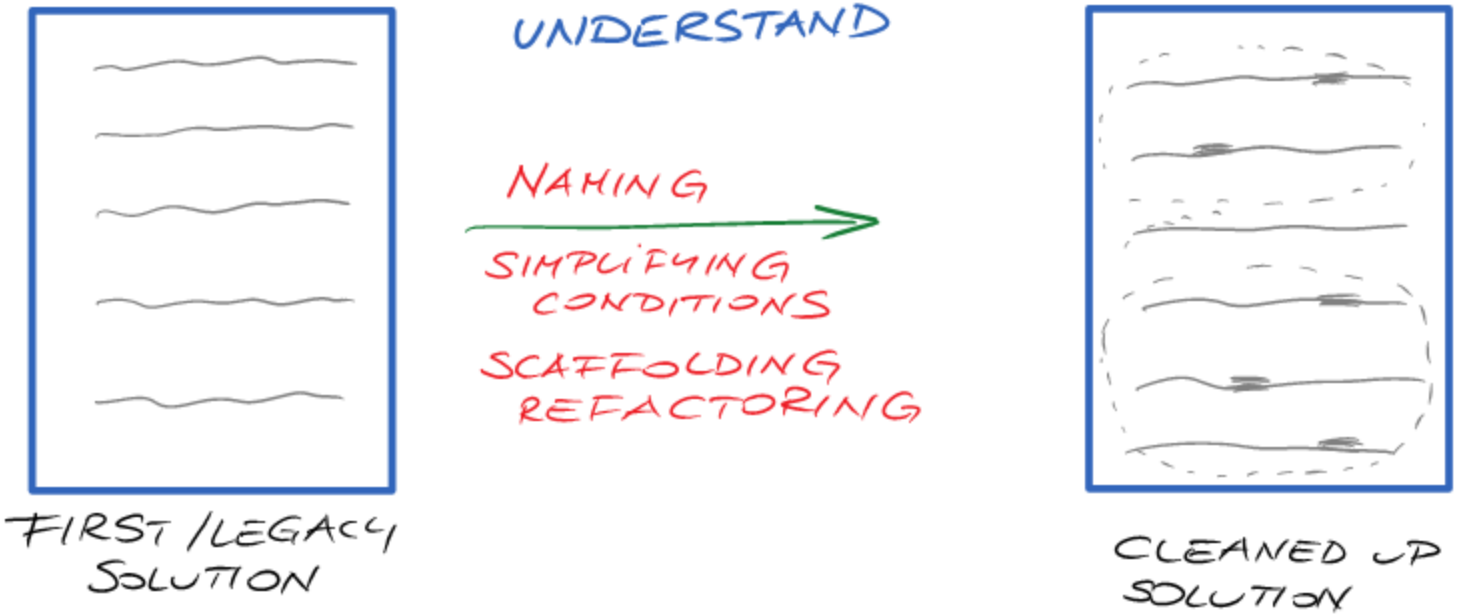
agile PRAGUE CONFERENCE

bns it

Natural Course of Refactoring

The process

Step 0. Understand the code



Understand

Sometimes it is the most difficult point

Try to…

- find an author and ask for help
- find somebody who worked with the code and ask for help
- find somebody who knows the system (or module) and ask for help
- Do it yourself if none of above are possible

Understand

Mental tools

Simple code cleaning

Clean up the names

Add temporary comments to the code

Introduce lazy variables initialization

Make optical cleanup (make more space)

Scratch refactoring

Do some exploratory refactoring to be thrown away

The only goal is to gain more understanding of the code

Introduce refactoring comments

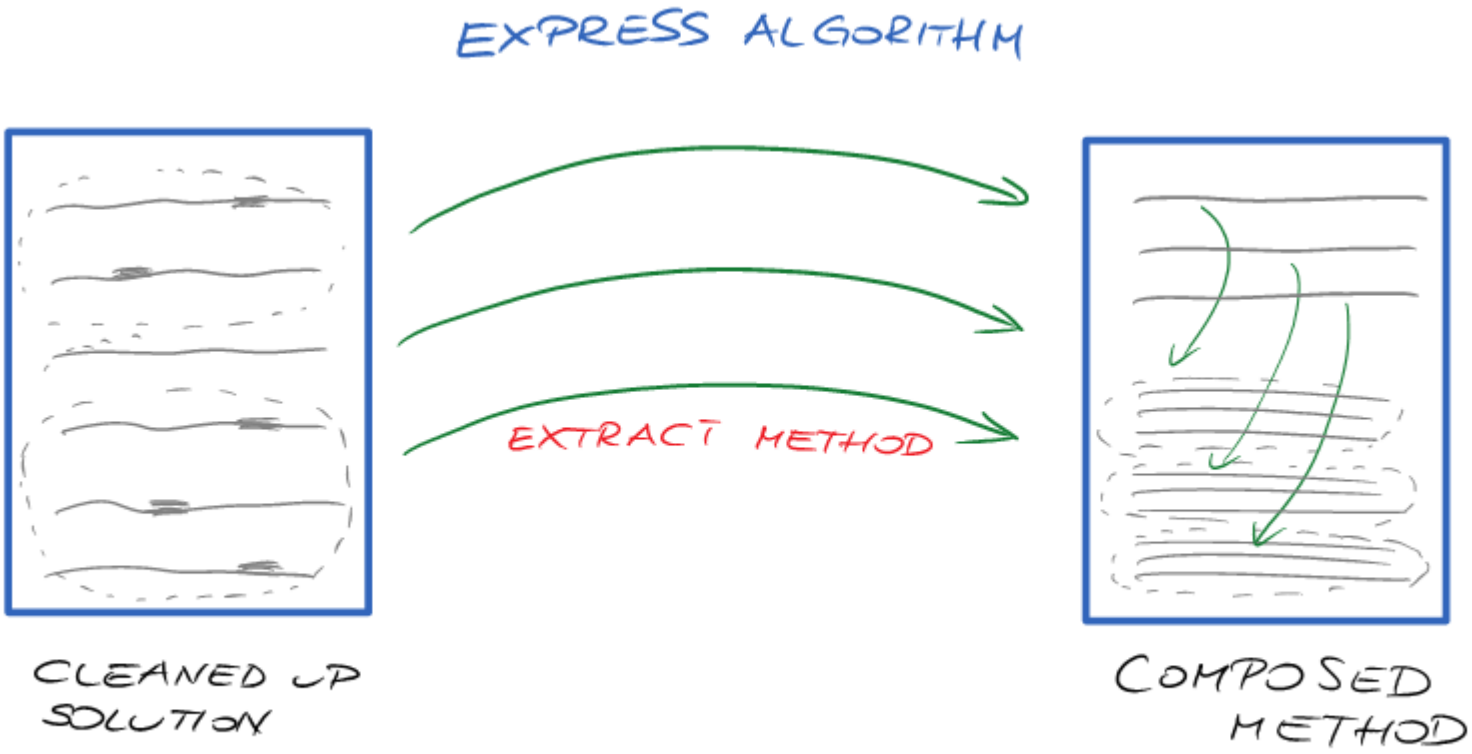
They should be temporary. Delete them after refactoring.

```
// SMELL it doesn't look good, copy-paste antipattern

// REFACTOR introduce factory

// NOTE send a message
```

Step 1. Express algorithm



Express algorithm

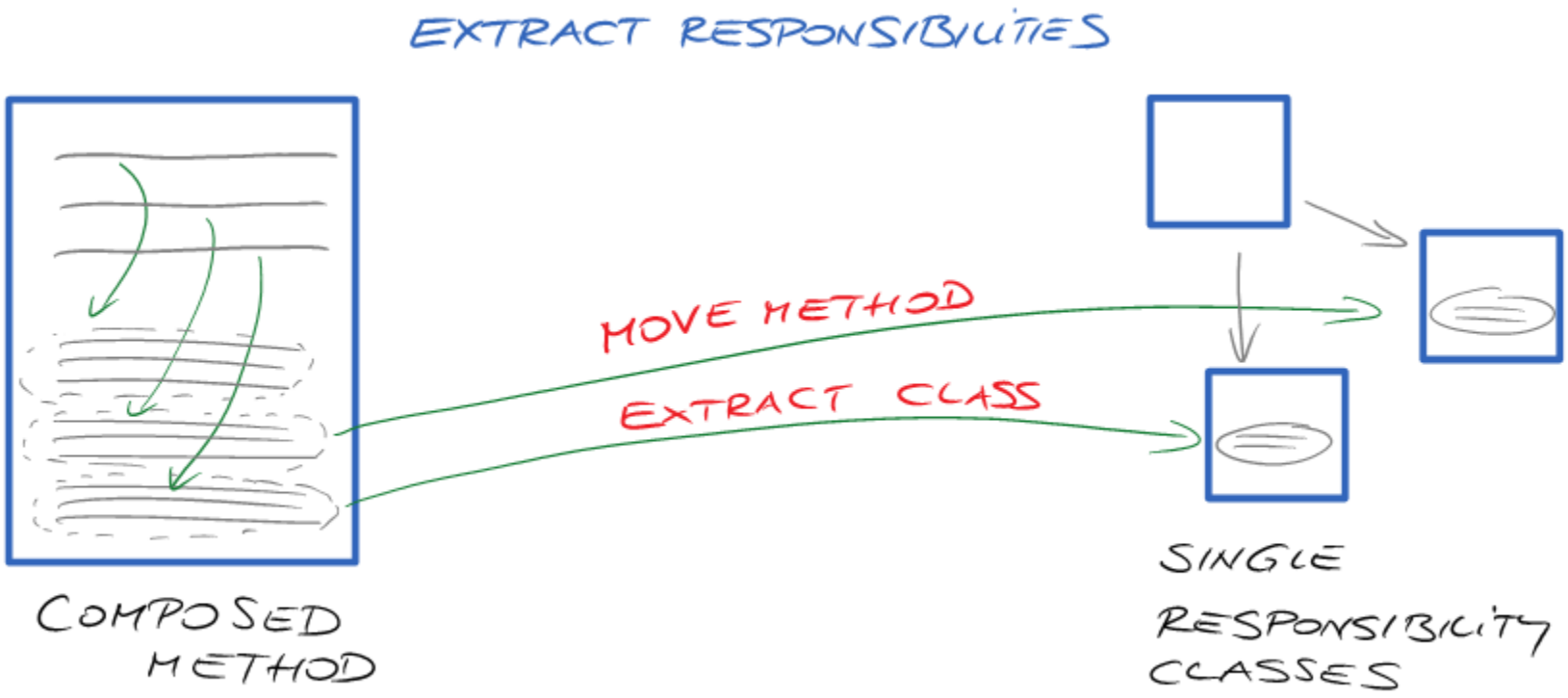
Aim

Code that speaks to you

Mental tools

- Compose method
- Introduce Method Object Refactoring
- Extract method
- Naming conditions

Step 2. Extract responsibilities



Extract responsibilities

Mental tools

Single responsibility principle

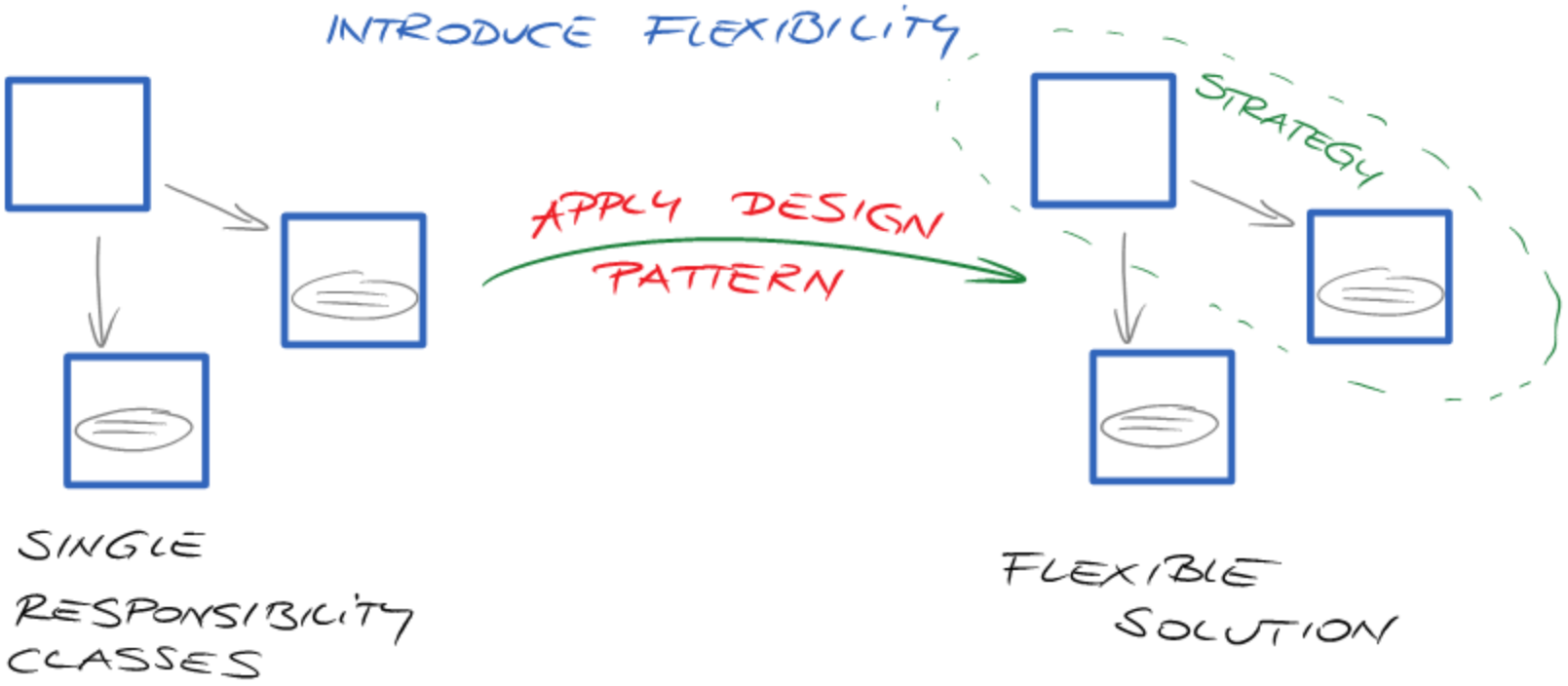
Move method refactoring

Extract class refactoring

Introduce Domain Object

Introduce Value Object

Step 3. Introduce flexibility



Apply design patterns

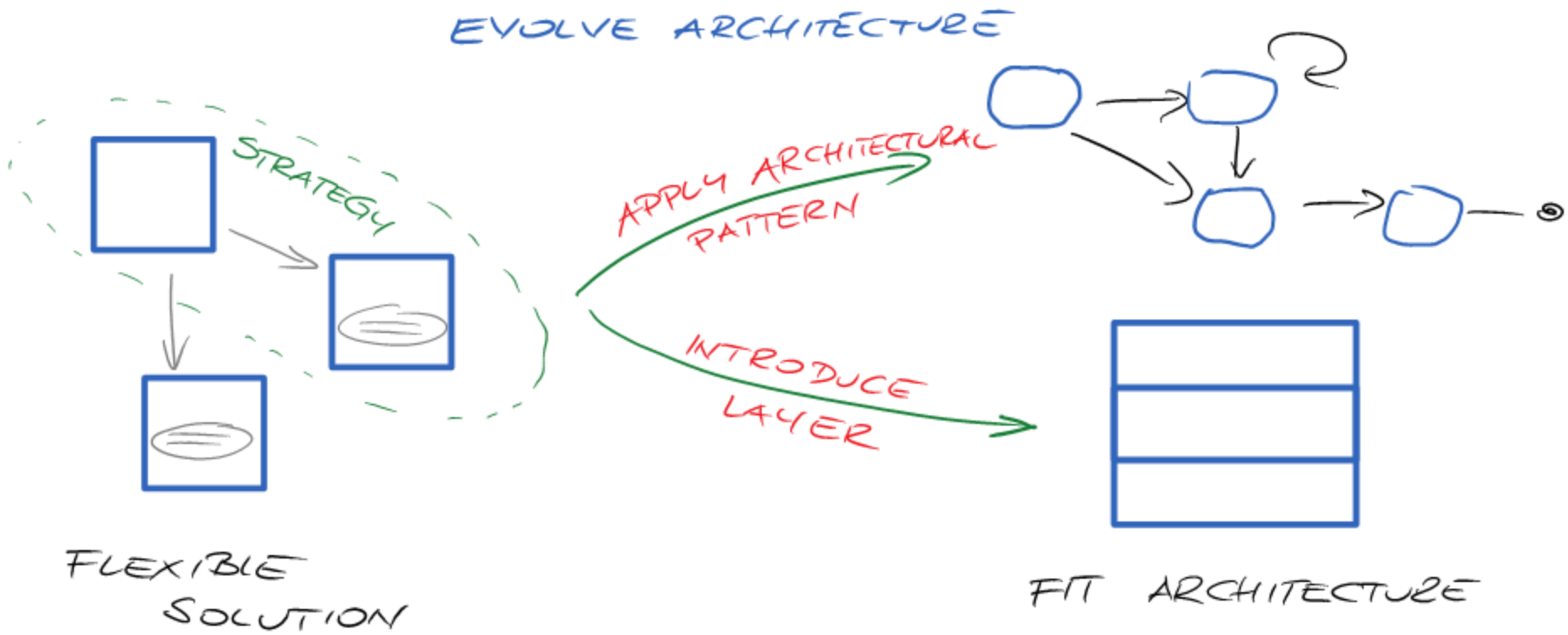
Mental tools

S.O.L.I.D.

Design patterns

Refactoring to patterns

Step 4. Evolve architecture



Evolve architecture

Mental tools

Introducing/removing layers

Introducing or replacing ORM/NoSQL/?

Important change in building blocks

Changing or introducing new framework

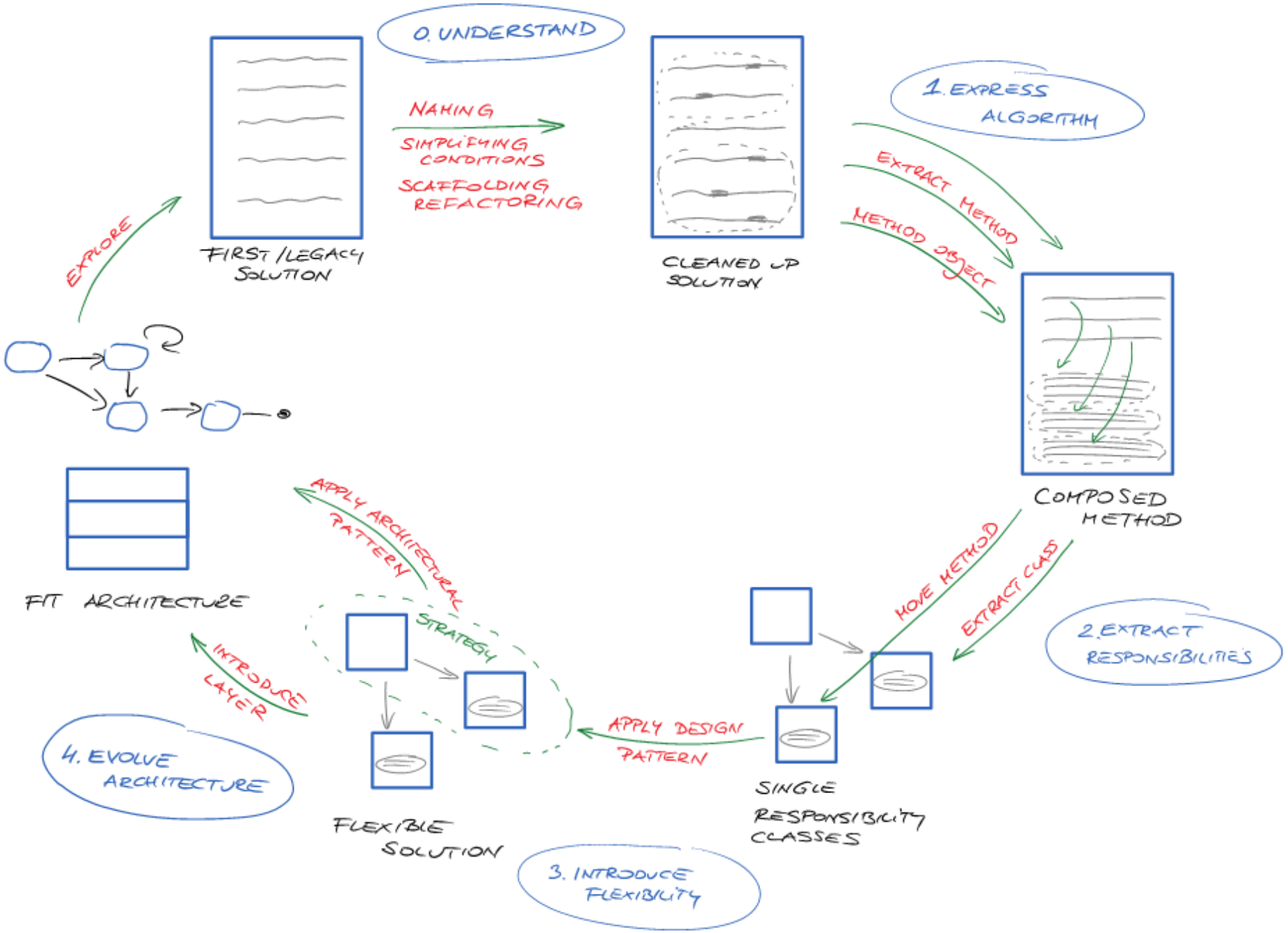
Introducing events

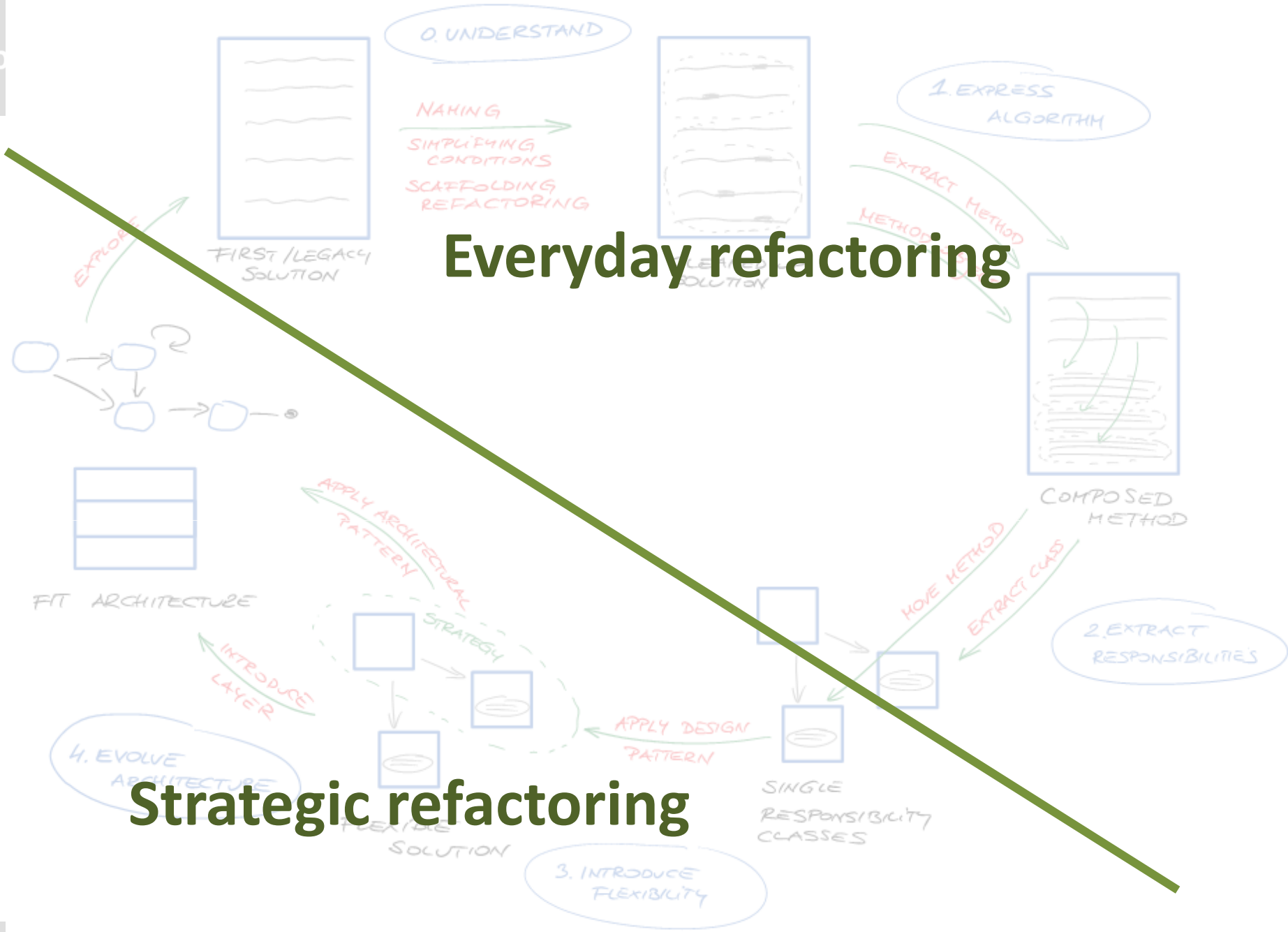
Introducing state machine

Moving towords DDD, Microservices, CQRS

Introducing Bounded-Context (DDD)

Applying Anticorruption Layer


0. UNDERSTAND
FIRST / LEGACY SOLUTION
NAMING
SIMPLIFYING CONDITIONS
SCAFFOLDING REFACTORING
CLEANED UP SOLUTION
1. EXPRESS ALGORITHM
EXTRACT METHOD
METHOD OBJECT
COMPOSED METHOD
EXPLORE
FIT ARCHITECTURE
APPLY ARCHITECTURAL PATTERN
STRATEGY
INTRODUCE LAYER
4. EVOLVE ARCHITECTURE
FLEXIBLE SOLUTION
APPLY DESIGN PATTERN
3. INTRODUCE FLEXIBILITY
MOVE METHOD
EXTRACT CLASS
2. EXTRACT RESPONSIBILITIES
SINGLE RESPONSIBILITY CLASSES



Everyday refactoring

Strategic refactoring

Why NCR?

Easy to teach

Easy to understand and remeber

Separates everyday and strategic refactoring

Indicates the simplest (safe) possible step in the moment

Gives hints what kind of refactorings can be applied in the moment

bns it

Natural Course of Refactoring.
A refactoring workflow.

Mariusz Sieraczkiewicz

@ms_bnsit_pl

http://msieraczkiewicz.blogspot.com

agile PRAGUE CONFERENCE