	Quelle	Typ	Steht zur Verfügung?	US
[OC]	(1) Object Calisthenics - Thoughtworks by Jeff Bay	Buch	✓	✓
	(2) Eclipse documentation	Internet	✓	–
	(3) Github Object Calisthenics Daniel Gindner	Internet	✓	–
[GoF]	(4) Gang of Four Design Patterns	Buch	✓	✓
[CC]	(5) Clean Code C. Martin	Buch	✓	✓
	(6) Refactoring Martin Fowler	Buch	✓	–
[DDD]	(7) Domain Driven Design Addison Wesley	Buch	✓	✓
	(8) Wiki Thoughtworks	Internet	✓	✓
	(9) Wikki Cohesion	Internet	✓	✓
	(10) Wiki Encapsulation	Internet	✓	✓
	(11) Wiki Liskov Substitution	Internet	✓	✓
	(12) Wiki Domain Driven Design	Internet	✓	✓
	(13) Blg Fowler Tell don't ask	Internet	✓	✓

used somewhere?

(A1) Prg programmer.com Internet ✓ ✓

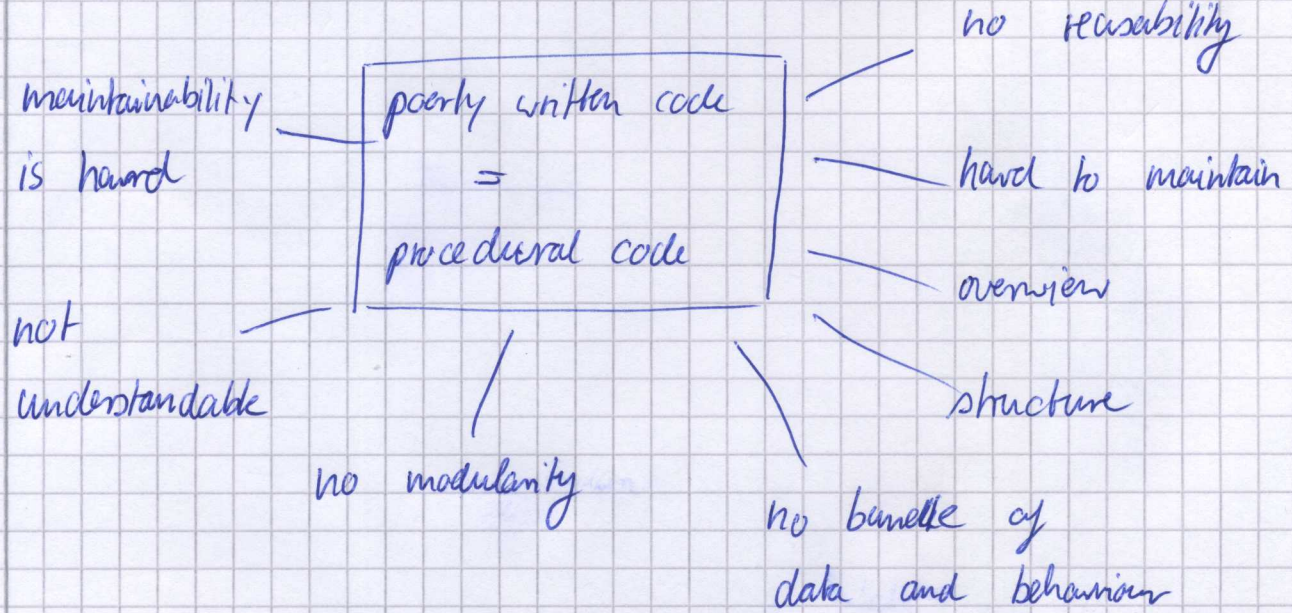
Main Source: The Thoughtworks Anthology
Chapter 6 - Jeff Bay Page 70
Year: 2008

ThoughtWorks describes itself as:

"...a software company and community of passionate individuals whose purpose is to revolutionize software design, creation and delivery, while advocating for positive social change."
[thoughtworks.com/about-us]

[OC, 70] Content.

Problem description:



⇒ Object Orientation Saves us!

But: procedural thinking is still in the minds of many developers:

because:

- they are used to old habits
- „easy to write".

Comparison of procedural vs. object oriented programming

procedural:

- step by step
- seldom information hiding
- actions manipulate data

object oriented

- „bundle" with capabilities
- hides structures
- delegates tasks that are not done by the object itself to other objects
- objects model real world behaviour
- decoupled and separated in different modules.

„what"

↓ leads to

„how":

- encapsulation
- abstraction
- inheritance
- polymorphism

→ Advantages of ocp:

- modules
- reusable
- maintainable
- simplicity

Core concept or „Qualities":

How focused is a class/module?

cohesion: How strong do the classes belong together?

cohesion metric: „modules are grouped because they all contribute to a single, well-defined task [wiki: Cohesion]

Qualities

loose coupling:

The degree of knowledge of surrounding objects

„Lack of coupling means that the elements of our system are better isolated from each other and from change" [CC: 150]

„DIP Principle" [CC: 150]

Qualities

Zero duplication:

If no duplication reusable components that don't have to be rewritten similar

Metric: code duplication

Encapsulation

Hide internal state with methods that are operations to execute ("commands")

[wiki: encapsulation]

- restrict component's access
- bundle of methods that implements operations on data
  → data is hidden behind specified methods

Testability

Is the code / class / module / method testable?

Metric: code coverage.

↳ amount of code that is tested

- function cov.
- statement cov.
- branch cov.
- condition cov.

Readability

easiness to understand

- documentation quality
- Amount of lines per unit
- „Top down narrative"

One indicator as example:

[CC: 37] The Stepdown Rule

Qualities

Focus

One object is responsible for one thing

? Metric: Lines per unit

Related to cohesion

Page 21 [OC]: Use all principles

The Exercise:

- o strict coding standards
- o used without doubts in the exercise during

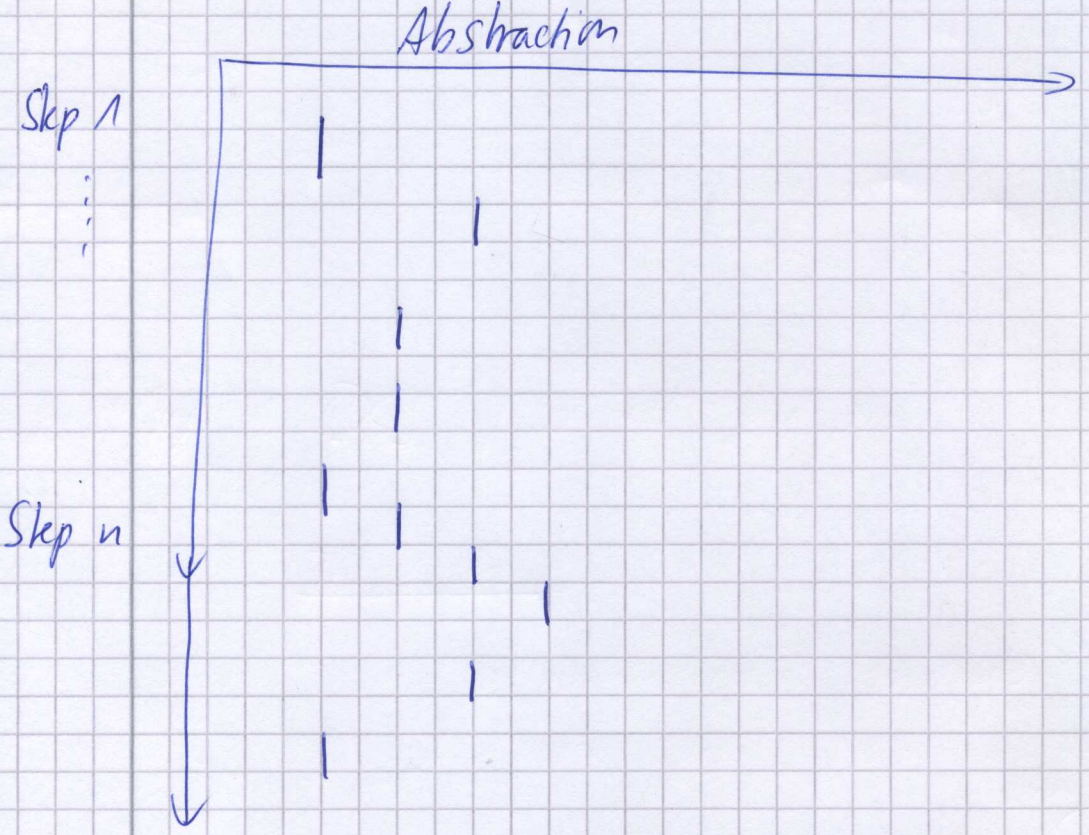
→ Understand 100% pure Object Orientation „in perfection" in a small project

⇒ Remember the „lessons learned" in the „wildlife"

Rule 1:

One level of indentation per method

- „Do one thing" [CC: 44] and small
- Cohesion - but not on class level, but on method level
- Describes how focused a method is.
- One level of abstraction [CC: 304]



⇒ Makes every method.

- easier to understand
- readable
- testable

} because the purpose is clearer

Ensures that the level only contains method calls that operate on the same level of abstraction (the „next" abstraction level)) ↔ Mac Info in [CC] 6

Rule 2

Do not use the else keyword

Learned early: [OC]

- if/else branches and conditions

→ Restrict Use

→ Restrict Non-Positive Case

There are three (3) ways to get around the „else" keyword:

I early return

II Null Object Pattern

III Strategy/State Pattern: Polymorphism

I Early return:

- No either or decision, but:

  Clear if condition and „return" keywords marks the end of the method

II Null Object Pattern

- Often null is returned in methods
- Null reference has to be checked because if not exception

⇒ Create reference with empty implementation of deactivating a reference

\+ Executable at runtime

III Use of Polymorphism.

Strategy Pattern:

Middle Block aus GOF!

→ GOF Pattern

→ DSP version?

→ break procedural text lines into units [GOF]

+ different algorithms executable

+ interchangable

(+ Liskov [wiki: Liskov Principle] ?) added by Fabian

„Same behavior as abstract specification"

guarantees no wrong or unexpected behaviour

if subtyp otherwise: documentation

→ less duplication

+ „configurable" class / algorithm with different behaviour" [GOF]

+ different variants or different implementations of the Algorithm [GOF]

→ „Move multiconditionals and branches to own Strategy class" [GOF]

---

out of topic:

→ Fabian Null object Pattern helps if there might be a if condition wit no early return. („Empty Strategy")

⇒ „Move different decisions (conditionals) and consequences (branches) to other classes, hidden behind a common interface" [GOF]

State Pattern

- Strategy changes context and changes next chosen Strategies to execute
- Similar to strategy
- GOF example: Nein, da ähnlich.

I and II and III help with

- cohesion
- duplication
- readable: - top down narrator
  - encapsulated behavior
  - focus
- configurable with polymorphism

→ "Implementation details are hidden in Polymorphism" [GOF]

Rule 3

Wrap all Primitives and Strings

Primitive types:

- Reflect a meaning without meaning or behaviour
- Parameter list of primitives in method's signature: Only the Methodname and not the parameters are expressing the meaning [OC]

-> Add a meaning to types (≙ behaviour = operations)
-> Type safety by Compiler [OC]
-> No accidential exchange of parameters (unlikely)
-> Bundle behaviour „close to data as possible
  -> Bundle „in Domain"
    -> Clear for everyone where to find operations on the datatype
    -> No „scattering" of behaviour

Domain Model: [Wiki DDD]

- domain is <u>not</u> trivial
- bundles behaviour and data in one type

DDD

squeeze down logic

- Entity: -- „Top level" ?!! - conceptual identity
- Value Object: Contains attributes but has no conceptual identity.

value data

- Aggregate: Collection of objects bundled
  Aggregate root guarantees consistency of changes.
  External changes to it's references (members of aggregate) are forbidden!

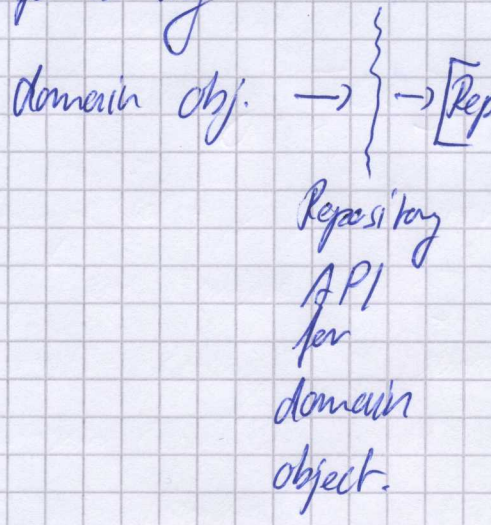
squeeze down

DDD Book

Wiki

Service: Operations that do not belong to the domain knowledge of an object.

Repository: Retrieves domain object (value, aggregate or Entity) and provides methods for storing



Factory: Methods for creating domain objects should delegate to a specialized factory object to make domain object alternative implementations possible and interchangeable

[DDD book]: „Model as distilled knowledge"
- bundeling data and behavior.
Encapsulate the behavior in common types

SAP Suite Mode !?

All in all:

- compiler → type safety
- encapsulation → with possible operations
- readability: → clear where to find

cycle in Rule 9

Rule 4

Use only one dot per line

[OC: 75]

Lines of code with multiple dots:

- „misplaced responsibility":

„Object is middleman"

- knows too much about too many people

→ Move operations in one of these objects.

Bad: - encapsulation → loose coupling and LoD
- cohesion
- bad testability

[OC] LoD: „Object is manipulating internal state of other objects instead of doing the task itself or delegating the work"

#1

[OC] Encapsulation: „Do not reach across boundaries into types that you do not know about"

Testability: - Many Mock objects behave differently
→ Represent similar behaviour.
Have to re-implement with many other mock objects because they ~~have to~~ (=might) cover all the touched boundaries like the original objects

better - when applying the rule:
- easy overriding of methods.
- Mocks can use direct methods of related mock/original objects.

→ Adaptable implementations/mocks are easier.

#1 LoD: Special case of cohesion
"principle of least knowledge" [citi] } instead: direct friends (1)

— Unit should be limited to direct friends and neighbours

⇒ Unit only talks to these friends

What is a „friend":
- own class
- parameters
- created objects
- associated classes

LoD:
- maintainability
- adaptable

⇒ less dependent, more focused

Rule 5

Don't abbreviate

Abbreviation problem:

- confusing
- bad readability:

→ Do you want to read a book full of abbreviations?

No long names:

- long method names with many words: [CC]
  doesn't do one thing

- do one thing
  - 2 words maximum [CC]

Help/Tip:

- Do not use classnames twice

~~order.shipOrder()~~

order.ship()

→ When assuming good variable names no duplication is necessary

→ [CC] „straightforward and concise" method and class names.

Rule 6

Keep Entities small

[OC 77] „do one thing" [OC] 50
„do it right" [CC]

→ focus, readability, overview

↑ Purpose per class: small units that „have an identity" [OC]

\+ „fit in one screen"

Packages are „clustered classes" [Fabi] 10
like classes: „Work together and achieve one goal"

Fabi: Bundle different special behaviours / implementation variations in one package (doing and achieving one thing)

↑

Rule 7

Don't use any classes with more than two instance variables

· Decrease of cohesion

[OC:77]

- manages single instance variable
- manages coordination of two instance variables

Similar to rule 3 and 7

→ bundle to 1 behaviour

Example Name:

Full Name → Prename, Name

└ List<Name>

Domain Driven:

Decomposition of a model into multiple smaller models representing the model

→ Refer to DDD (already explained -)

→ Leads to:

- squeeze down model
- separate model concerns

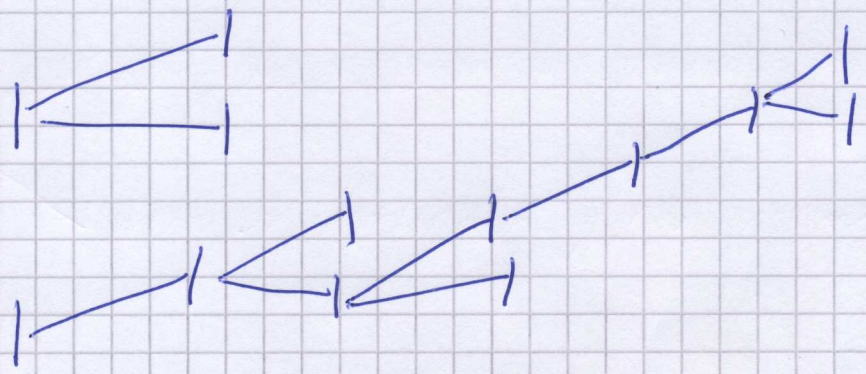
DD: Aggregate

split down vs. pick up.

Similar to rule one: applied on object structure:

- same degree of abstraction in one class

Leads to:



Example: SAP Sailly Domain Model?

Rule 8

First class collections

- Encapsulation: Hide internal implementation
- Hide „standard" manipulation methods that are framework specific and make sure only the necessary meheds are provided
- Like Rule 5: „Domain Drive"
- behavior & data together

Rule 9

Tell, don't ask (No Getter/Setter/Properties)

Also famous under the name „Tell, don't ask"

Claim: Encapsulation does not mean to add getter/setter/properties afterwards.

→ Object would „expose it's implementation details directly" [OC]

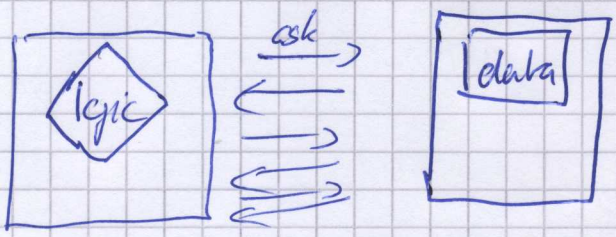
→ Place behaviour into the object instead of exposing it's data and letting others manipulate it.

→ less duplication

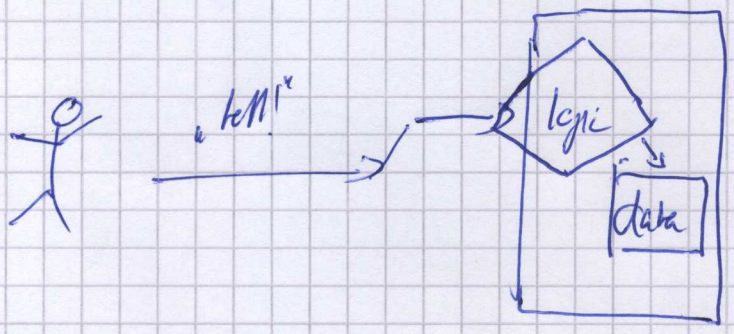
→ Better localization of behaviour

Martin Fowler plg

→ Bundle the data with the behaviour



- exposes implementation details
- behaviour is outside
- behaviour is „littered around"



- operations to execute
- more like „message sending"

- data and behaviour are in one place
- „Tell" don't ask!"
- Easily testable, because the ~~data~~ logic will not be rewritten, but is part of the things that are tested.
- Mocks are easy: behaviour and data are bundled in one object with one specification

⇒ No ask about internal state, but „command style"

⇒ Single responsibility [CC]
⇒ Do one thing [CC]

+ testability
+ maintainability
+ focus (clearer api specification)

Prog Programmer.com

Seperates between

- Query: Returns aggregated Entity, Side effect free
- Command: Operation that internally changes the object's state

[CC] Tell, don't ask!

All in all

- behavior and operation oriented
- LeD & loose coupling
  - ↳ talk to friends
  - ↳ talk the protocol specified by the object's operation

Conclusion

[OC]
- 8 Rules Encapsulation
- No else: Appropriate use of polymorphism
- Naming Strategy:
  - concise
  - without abbreviation

All in all:

[OC]
- No duplication in
  - code
  - idea

„Simple and elegant Abstractions"

Recommendation

Exercise:
- Spend 20 hours with it.
- 1000 LoC that confirm the rules 100%.
  - → break habits
  - → Follow the rule and start to solve problems differently

[OC] „Old obvious habits and solutions are not available anymore" because of the restriction

⇒ get used to rules

Objective: Conform Rules without any effort.

[CC] Finish

- Author has finished a 100-000 line Project written in this style
- Programmers followed the rules routinely
  → less tiresome development when „embracing deep simplicity"