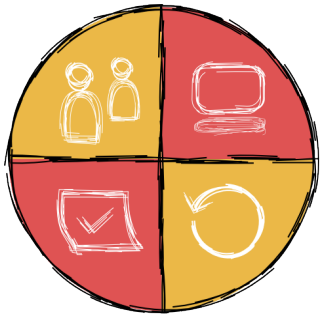
Refactoring with Connascence



Softwerkskammer Thüringen
11.04.2019

Intro

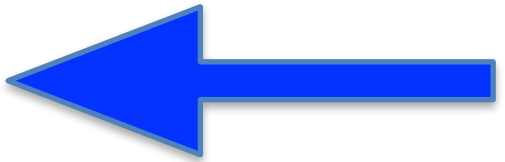
Thorsten Brunzendorf

codecentric

Nürnberg

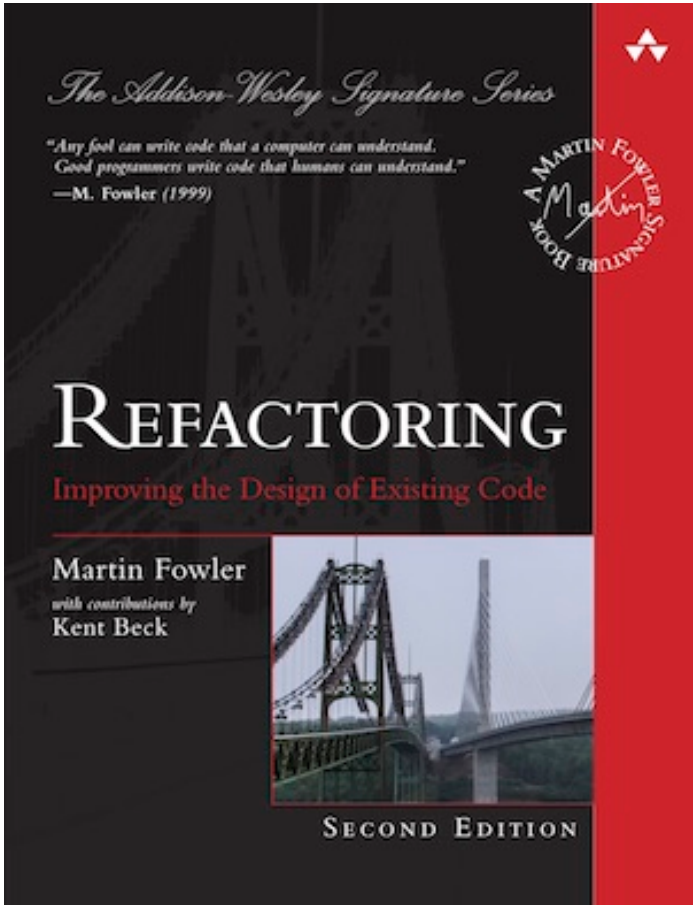
TDD Loop

- **Red**
  - Write a failing test
- **Green**
  - Make the test pass
- **Refactor**
  - Improve the design
- **Repeat**



Refactoring

"Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior."
(Martin Fowler)



Refactoring Workflows

- TDD Refactoring
- Litter-Pickup Refactoring (boy-scout rule)
- Comprehension Refactoring (better names)
- Preparatory Refactoring

  > "for each desired change, make the change easy (warning: this may be hard), then make the easy change" (Kent Beck)

- Planned Refactoring
- Long-Term Refactoring

Refactoring is a tool

- What tools are there for refactoring?

- Start in green and stay in green (Test Runner)
- Small mechanical steps (Refactoring book)
- IDE with Refactoring support

- Most important tool?

The most important tool?

- Our brain

  !?!

- TDD cycle really is

  Think-Red-Think-Green-Think-Refactor-Think ...

- What can help us to reason about our code?

Sandro Mancuso
@sandromancuso

Folgen

I believe software design should be taught before TDD. TDD can't lead to good design if we don't know what good design looks like.

17:47 - 15. Apr. 2015

245 Retweets **126** „Gefällt mir"-Angaben

29 245 126

https://codurance.com/2015/05/12/does-tdd-lead-to-good-design/

Better Design?

- There are **principles**
- E.g. **SOLID**
  - **S**ingle Responsibility
  - **O**pen/Closed
  - **L**iskov Substitution
  - **I**nterface Segregation
  - **D**ependency Inversion

Better Design?

- And there are **rules**
- E.g. the **4 elements of simple design**
  (Kent Beck, here J.B. Rainsberger's version)
  - Passes its tests
  - Minimizes duplication
  - Maximizes clarity
  - Has fewer elements

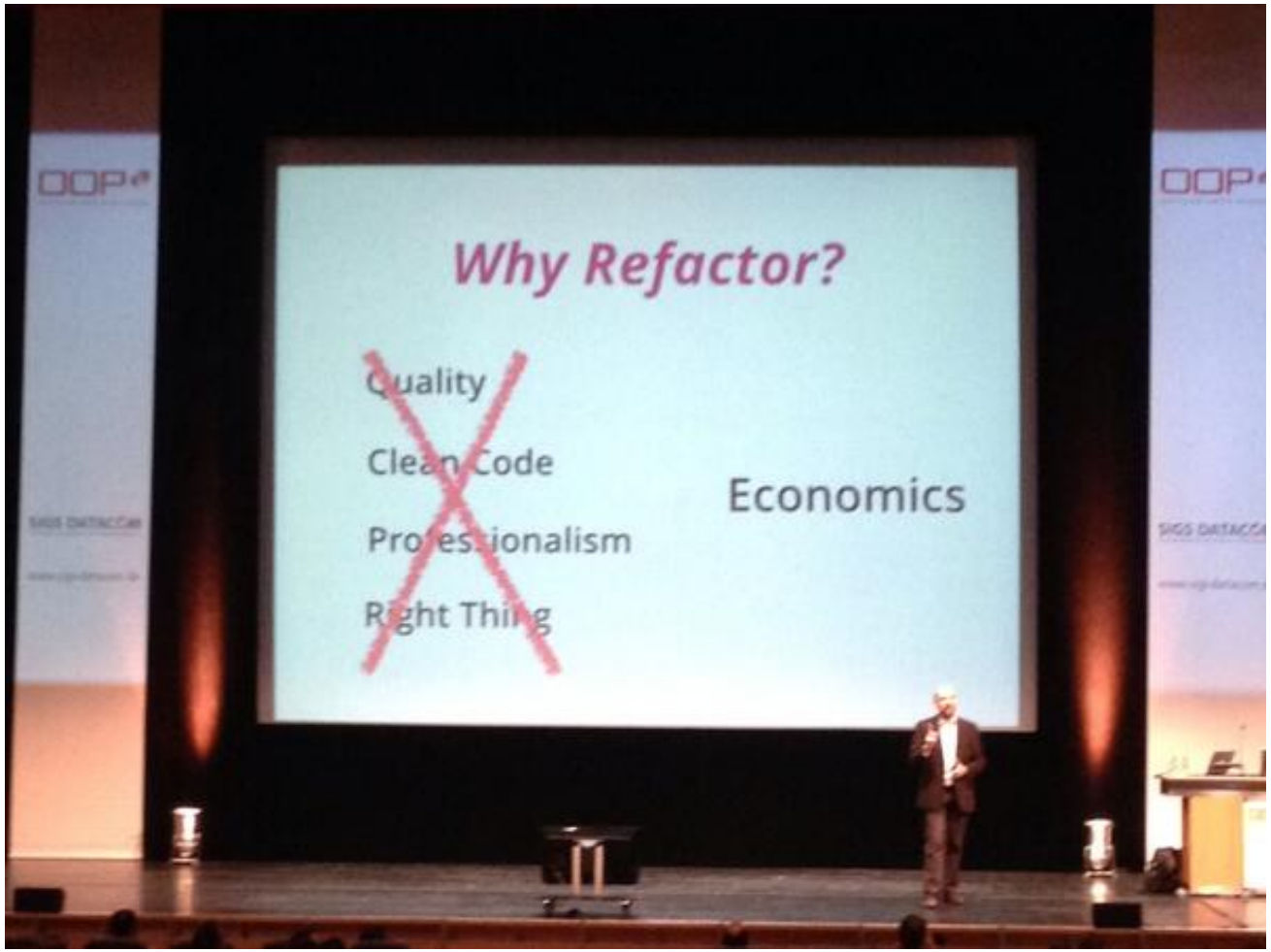
Better Design?

- And there are even **laws**
- E.g. the Law of Demeter
  - “Don’t talk to strangers“
- More specifically
  - “Use only one dot“

Better Design?

- But also **subjective opinions**
  - Listen to your tests
    - Bad tests signal bad design
  - Code smells
    - Kevin Rutherford: The problem with code smells
  - Gut Driven Development?
    - Kevin Mahoney: Understand your biases, e.g.
    - Something is better because it is familiar to you
    - Something is better because it is new
    - **So examine your opinions and find some objective advantage**

Why Refactor?



Economics of software change

- Cost of **reading** and understanding
- Cost of **finding** the things that have to change
- Cost of **making** the change
- Cost of **testing** the change
- Cost of **building** and **deploying** the change
- Cost of **maintaining** the change

Coupling and Cohesion

- Coupling
  - Degree of interdependence between software components
- Cohesion
  - Degree to which the elements of a component belong together
- Good design: low coupling, strong cohesion
  - So cohesion is "good coupling" because it is local!?
- Let’s go multi-dimensional!

Connascence

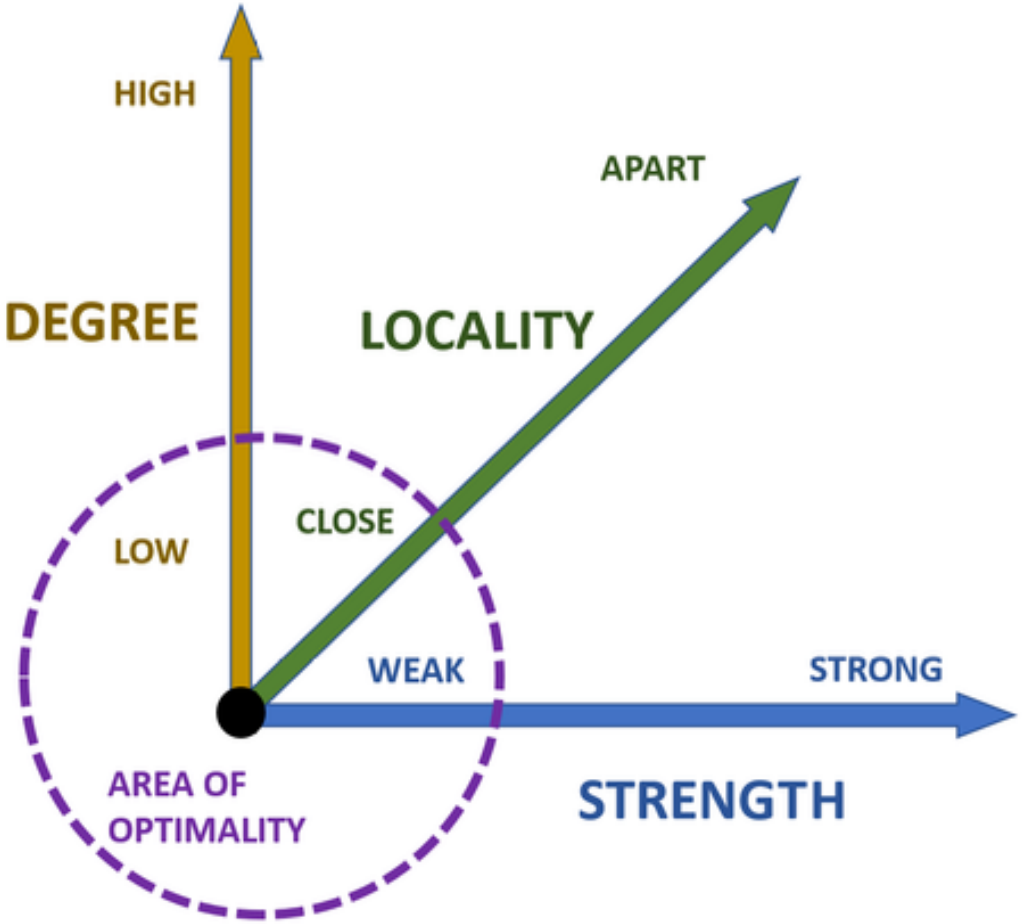
Meaning

- “Being born and growing together“

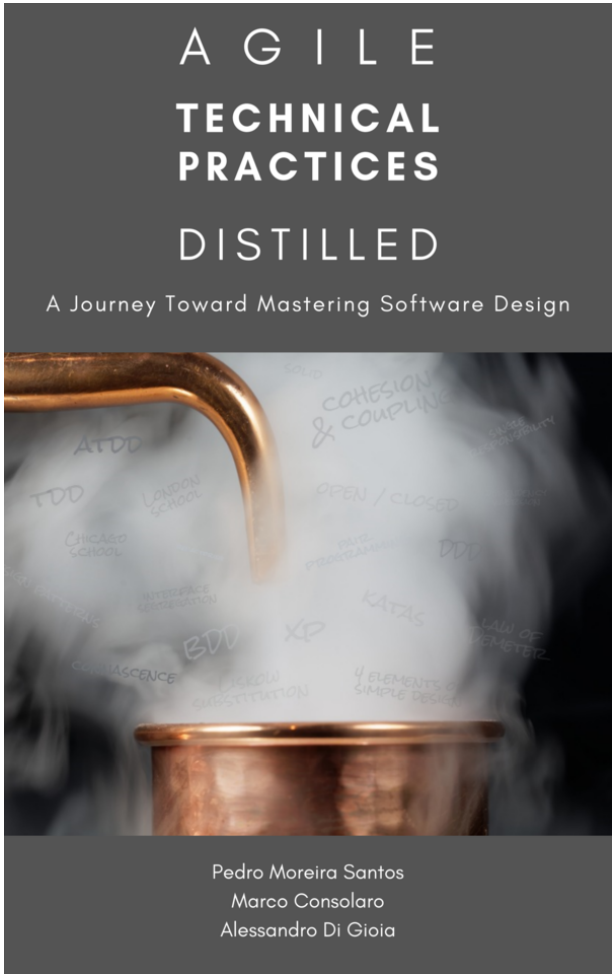
Definition (software)

- Two or more elements are connascent when a change in one element requires a change in the others in order to keep the system working correctly

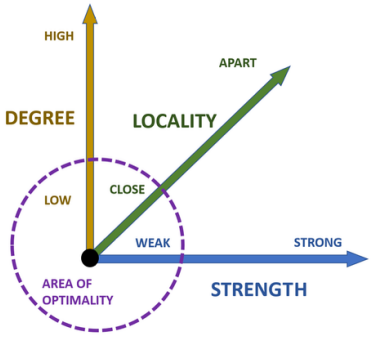
3 Dimensions of Connascence



- Pedro Moreira Santos, Marco Consolaro and Alessandro Di Gioia: Agile Technical Practices Distilled
- https://leanpub.com/agiletechnicalpracticesdistilled

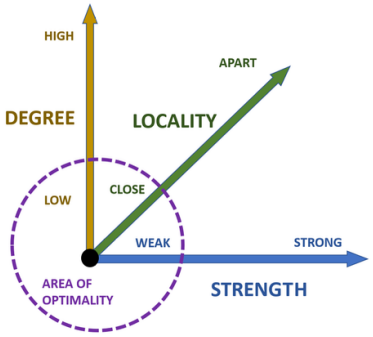


Degree



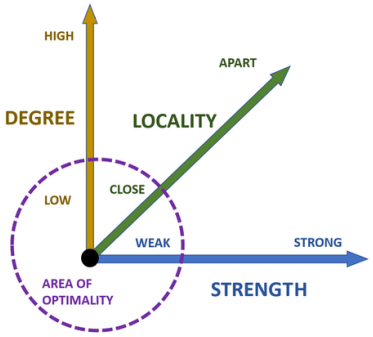
- How many elements
  are involved in this connascence
  (**multiplicity**)
- Higher degree means
  - Harder to discover
  - Especially all of them
- Lower is better

Locality



- How close together
  in the **structure of our codebase**
  (functions, objects, packages, modules, applications, …)
  are the elements of this connascence
- More apart means
  - Harder to discover
- Closer is better

Strength



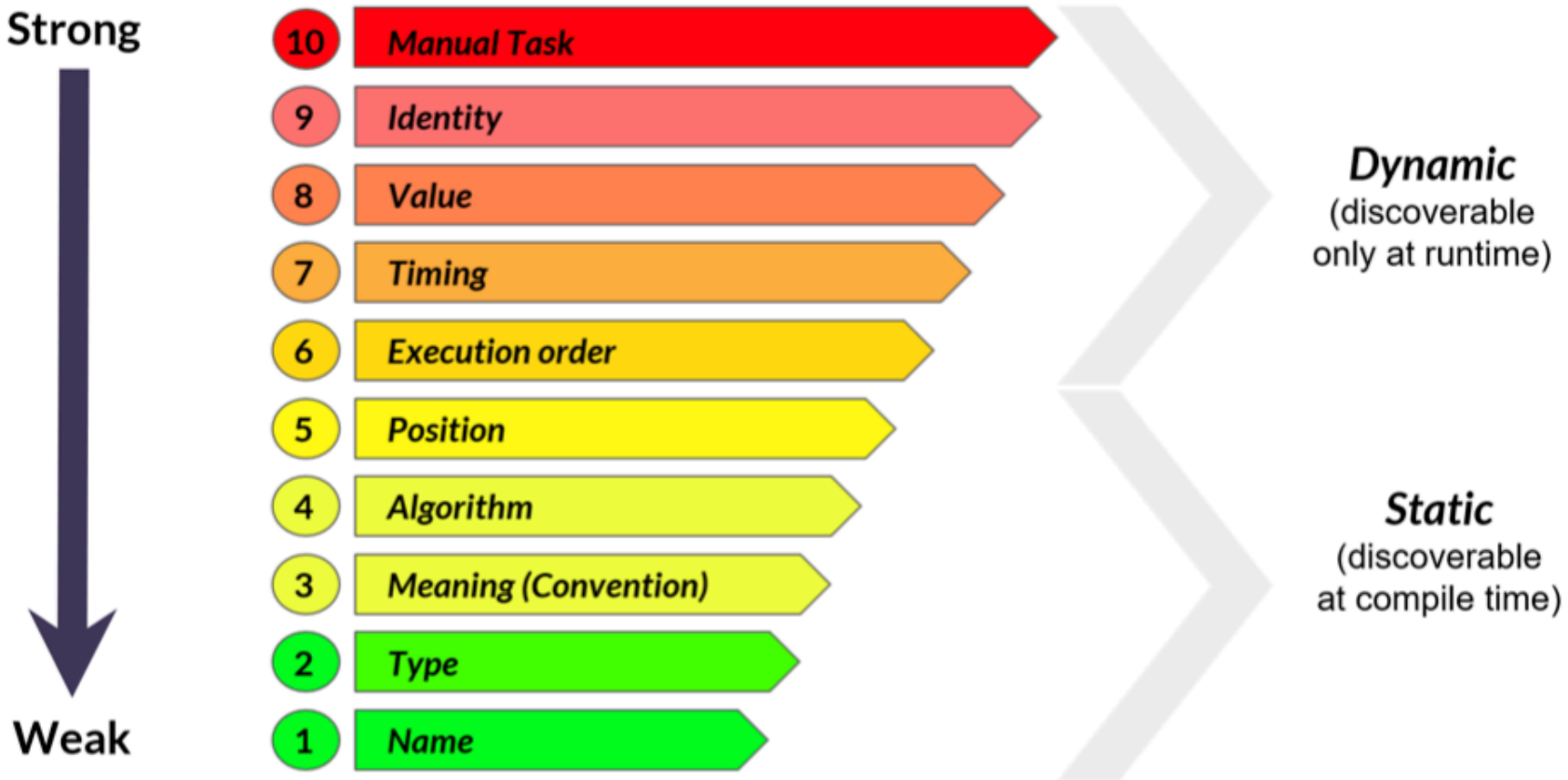
- How strong is the **type** of this connascence
- Stronger means
  - Harder to discover
  - And harder to refactor
- Weaker is better

3 Refactoring Options

Towards better design

1. Reduce degree
2. Increase locality
3. Reduce strength

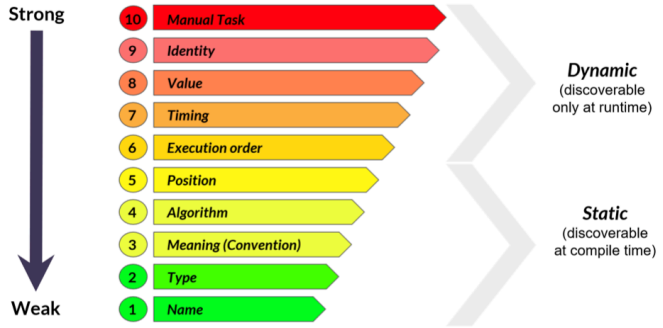
Types of Connascence



Connascence of Name

Obvious, because
callers have to know the

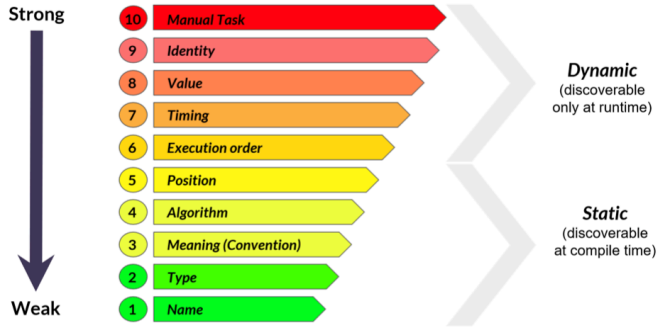
- Name of the method
- Name of the variable
  upon which the method is called



Connascence of Type

Obvious, because
callers have to know the

- Type of object
  to know what methods
  or messages are available
- Type of parameters and return values
  (obvious in static languages, but
  not so obvious in **dynamic** languages)



Connascence of Type

Example

- Signature

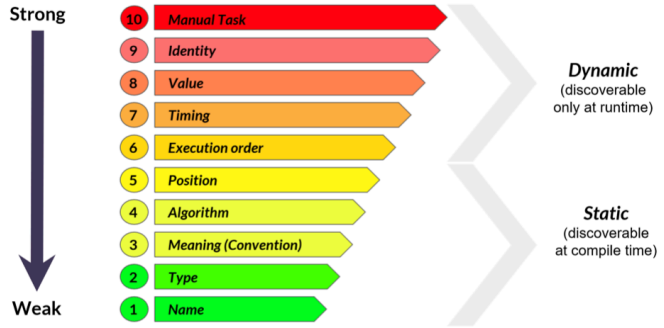
```
age (day, month, year)
```

- What is right?

```
age (1, 1, 1970)
age ("1", "1", "1970")
age (1, "January", 1970)
```

Connascence of Convention

When
two or more components
must agree on
the meaning
of specific values,
hence using a convention



Aka Connascence of Meaning

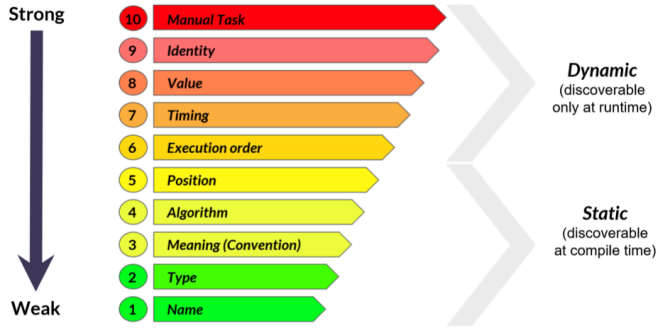
Connascence of Convention

Examples

- Java Comparable.compareTo return values: -1, 0, +1
- Flags and modes: true, false => avoid boolean arguments!
- Meaning of null return value: not found, invalid argument, ...
- Money representation

Connascence of Algorithm

When
two or more
components
must agree on
using a particular algorithm



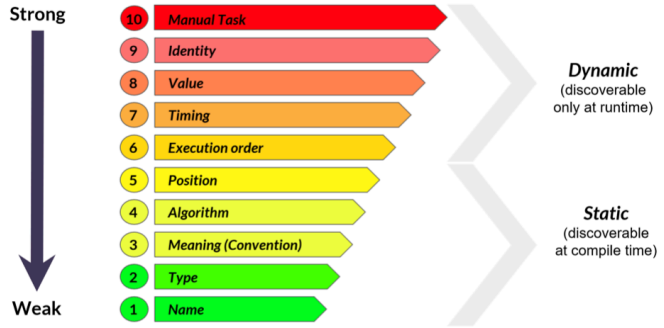
Connascence of Algorithm

Examples

- Frontend / backend validation
- All kinds of symmetries
  - encrypt/decrypt
  - encode/decode
  - compress/decompress

Connascence of Position

When
multiple components
must be adjacent
or must appear
in a particular order



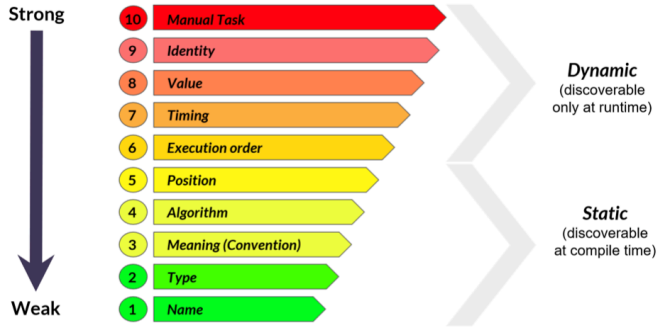
Connascence of Position

Examples

- Often when passed within a positional structure like an array or a tuple
- Constructor or method parameters all strings
  - Better: Builder pattern or named parameters
- List where entries have different meaning depending on the order
  - Better: Map with named of entries

Connascence of Execution Order

When
the caller of a component
must have some
unexpressed knowledge
about the correct order
of the methods to be called



Connascence of Execution Order

Example

- Stateful usage
  - Order: set, set, get

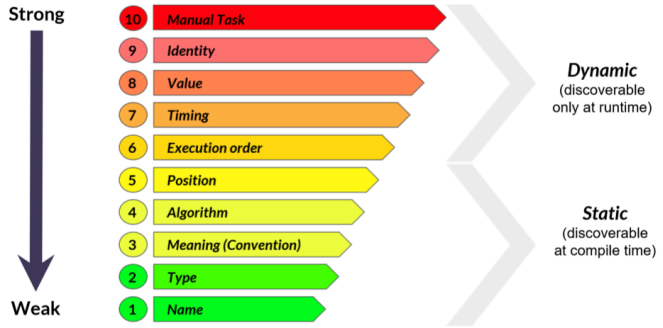
```
vacation.setStart(2019,4,1)
vacation.setEnd(2019,4,8)
val duration = vacation.getDuration()
```

  - Better

```
val duration =
  vacation.getDuration(start = Date(2019,4,1),
                       end = Date(2019,4,1)
```

Connascence of Value

When
two or more
components' values
are related
or have an intrinsic range of validity
in their input
not expressed by their primitive types



Connascence of Value

Examples

- Same values in production-code and test-code
- Special values known in lots of places
- Implicit validity of values

Locality and Remote APIs

- Originally connascence was more about function level and class level
- How about Microservices?
- How about Serverless?

Microservices and Rest APIs

- Same types of connascence
- Service discovery, alias and redirection, api gateways
- Which commands are valid?
  - Duplicate logic or HAL resources and links
- Encode / decode and assemble / parse json documents
  - Connascence of Algorithm, duplicate conventions etc

Generic APIs

"Ironically

the more generic

your API

the more coupling

you create“

Oliver Drotbohm:
REST Beyond the Obvious - API Design for Ever Evolving Systems

Degree and Remote APIs

- As an API provider do I know all consumers of my API?
- **NO**
- Then use documentation, schemas, non breaking changes
- **YES**
- Then use Consumer Driven Contracts

Back to the TDD Loop

- **Red**
  - Write a failing test
- **Green**
  - Make the test pass
- **Refactor**
  - Improve the design
- **Repeat**

Test Driven Design

- **Transformation Priority Premise**
  - Tells us **what test to write next**
  - What is the next behavior that requires the least complex code
- **Connascence**
  - Tells us **what refactoring to apply next**
  - What refactoring would have the most impact on reducing connascence in our code

Hands-on coding

- Kata applying connascence
  - “Back to the Checkout“ (Dave Thomas @PragDave)
  - http://codekata.com/kata/kata09-back-to-the-checkout/
- Your task
  - Work in pairs and use TDD for this kata
  - For refactoring steps discuss the connascence in your current code and try to reduce it

Back to the Checkout

This week, let's implement the code for a supermarket checkout that calculates the total price of a number of items. In a normal supermarket, things are identified using Stock Keeping Units, or SKUs. In our store, we'll use individual letters of the alphabet (A, B, C, and so on). Our goods are priced individually. In addition, some items are multipriced: buy n of them, and they'll cost you y cents. For example, item 'A' might cost 50 cents individually, but this week we have a special offer: buy three 'A's and they'll cost you $1.30. In fact this week's prices are:

```
Item   Unit      Special
       Price     Price
--------------------------
  A     50       3 for 130
  B     30       2 for 45
  C     20
  D     15
```

Our checkout accepts items in any order, so that if we scan a B, an A, and another B, we'll recognize the two B's and price them at 45 (for a total price so far of 95). Because the pricing changes frequently, we need to be able to pass in a set of pricing rules each time we start handling a checkout transaction.

The interface to the checkout should look like:

```
co = CheckOut.new(pricing_rules)
co.scan(item)
co.scan(item)
    :    :
price = co.total
```

Short Retrospective

- What did you observe?
- How did you feel about it?
- What would you like to try next time?

More Reflection

- Kata “Back to the Checkout“ applying connascence exercised by Kevin Rutherford
- Blog posts
  - https://silkandspinach.net/2015/01/22/connascence-of-value/
- Video
  - https://www.youtube.com/watch?v=fSr8LDcb0Y0

Contact

Thorsten Brunzendorf

 @thbrunzendorf

✉ thorsten.brunzendorf@codecentric.de

Thank you!

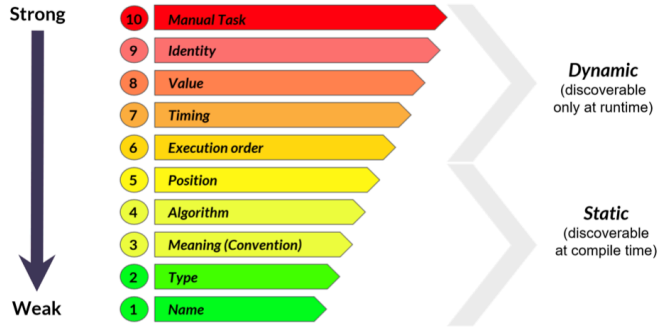
References

https://refactoring.com/
https://martinfowler.com/articles/workflowsOfRefactoring/
https://twitter.com/kentbeck/status/250733358307500032
https://blog.jbrains.ca/permalink/the-four-elements-of-simple-design
https://silkandspinach.net/2012/09/03/the-problem-with-code-smells/
http://kevinmahoney.co.uk/articles/gut-driven-development/
https://twitter.com/sandromancuso/status/588503877235781632
https://codurance.com/2015/05/12/does-tdd-lead-to-good-design/
https://twitter.com/mbohlende/status/431446680874258433
https://leanpub.com/agiletechnicalpracticesdistilled
https://speakerdeck.com/olivergierke/rest-beyond-the-obvious-api-design-for-ever-evolving-systems
https://twitter.com/olivergierke/status/997735413606420480
https://www.codesai.com/2017/01/about-connascence
https://silkandspinach.net/tag/connascence/
http://xpsurgery.com/resources/connascence/
https://silkandspinach.net/2015/05/20/red-green-what-now/
http://connascence.io/
http://confreaks.tv/videos/aac2009-the-grand-unified-theory
http://codekata.com/kata/kata09-back-to-the-checkout/

Backup

Connascence of Timing

- When
  the success
  of two or more calls
  depends on the timing
  of when they occur



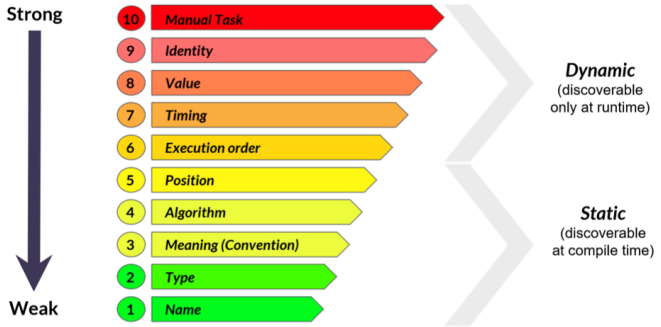
Connascence of Timing

Example

- Threads

Connascence of Identity

- When
  one or more components
  must reference
  exactly one particular
  instance of another entity
  to work correctly



Connascence of Identity

Examples

- Message queues and listeners for topics
  - Where in test environment there is only one, in production there are many instances
  - Messages in production are about of order
  - Better: Bind queues / listeners to topics
- Using ORM frameworks
  - Different instances of entities
  - State and re-reading from persistence