



# Object-Centric Instrumentation with Pharo

Steven Costiou

Square Bracket tutorials

February 14, 2019

Copyright 2017 by Steven Costiou.

The contents of this book are protected under the Creative Commons Attribution-ShareAlike 3.0 Unported license.

You are **free**:

- to **Share**: to copy, distribute and transmit the work,
- to **Remix**: to adapt the work,

Under the following conditions:

**Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page:  
<http://creativecommons.org/licenses/by-sa/3.0/>

Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.



Your fair dealing and other rights are in no way affected by the above. This is a human-readable summary of the Legal Code (the full license):  
<http://creativecommons.org/licenses/by-sa/3.0/legalcode>

# Contents

<b>Illustrations</b>	<b>ii</b>
<b>1 Talents</b>	<b>1</b>
1.1 Example . . . . .	1
1.2 Evaluation . . . . .	2
<b>Bibliography</b>	<b>3</b>

# Illustrations

# Talents

Talents are originally behavioral units, that can be attached to an object to add, remove or alter behavior [RGN<sup>+</sup>14]. Only the object to which a talent is attached is affected by behavioral variations. The latest talent implementation relies on trait definition.

## 1.1 Example

Talents are based on traits. Objects can answer to the `#addTalent:` messages, which takes a `Trait` as parameter. All behavior defined in the trait is flattened in the object. In the following illustration, we instantiate an anonymous trait, and we compile a method in this trait. That method is an instrumented version of the original `name` method of the class `Person`. This new method replaces the original one, until the talent is removed from the object.

```
|person talent|
  person := Person new.
  talent := Trait new.
  talent
    compile:
      'name: aName
        self tag: aName.
        name := aName'.
  person addTalent: talent. "adds the talent to the object"
  person removeTalent: talent. "removes the talent from the object"
```

## 1.2 Evaluation

**Manipulated entity: Trait.** Behavioral variations are expressed using traits. It can be Traits defined in the image or anonymous trait instances in which specific behavior is manually compiled by the developer.

**Reusability: Yes.** A trait can be added as a Talent to any number of objects.

**Flexibility: Partial.** Using anonymous traits forces the user to manually compile code in the method. This is however necessary to achieve a sub-method granularity. Conflicts must be resolved manually when Traits are composed.

**Granularity: Method.** Traits add, remove or alter (through aliasing) the behavior of a method. It can be done at a sub-method level (*e.g.* inserting a statement in the body of a method), but that requires manual rewriting of the method in the Trait.

**Integration: Partial.** The object is migrated to an anonymous subclass, which does not break system tools. However, it may break libraries that uses classes and class names as a discriminator.

# Bibliography

- [RGN<sup>+</sup>14] Jorge Ressia, Tudor Gîrba, Oscar Nierstrasz, Fabrizio Perin, and Lukas Renggli. Talents: an environment for dynamically composing units of reuse. *Software: Practice and Experience*, 44(4):413–432, 2014.

