

When To Abstract

SymfonyCon Berlin 2016

Kore Nordmann (@koredn / @qafoo)
1st December, 2016



Hi, I'm Kore (@koredn)



A close-up photograph of a dense field of bright green, succulent-like plants. The plants have small, pointed, overlapping leaves and tiny yellow flowers. The background is dark, making the vibrant green of the plants stand out. A semi-transparent dark horizontal band is overlaid across the middle of the image, containing white text.

You: A Pattern Recognition Machine

Fast Category Learning*

Look:
A table



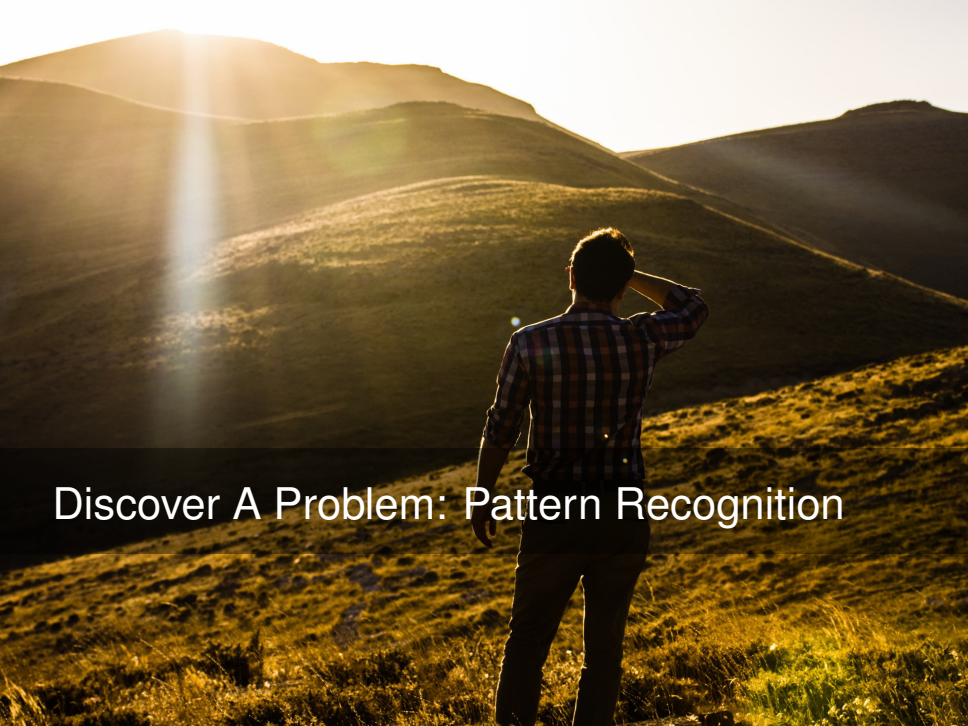
Table?

Table.

Table!



*Psychology of Learning and Motivation (Band 43)
Academic Press, 04.09.2003

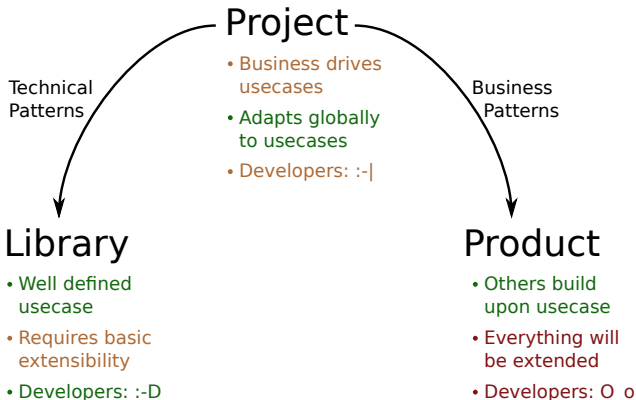


Discover A Problem: Pattern Recognition



When To Stop?

Types Of Development



Pareidolia



TOAST



ORANGE



DOG

A young woman with blonde hair and blue eyes, wearing a blue blazer over a white shirt, is looking intently at another woman. The second woman, seen in profile, has dark hair and is wearing a dark blue top and a watch. They appear to be in a professional or business setting, possibly a meeting or conference. The background is slightly blurred, showing other people and a bright light source.

Discover Business Patterns

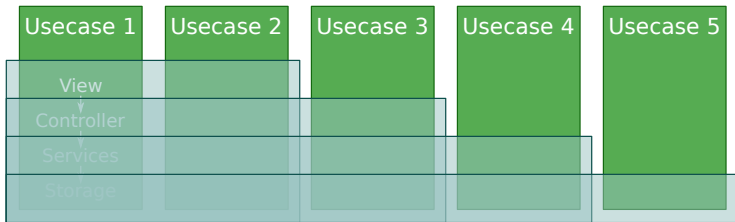
OK – what should we do?

Business Requirements Will Never Converge



Split vertically instead of horizontally

- ▶ How much code can be re-used for that new user registration through Facebook?



Avoid Pareidolia when creating abstractions

- ▶ Never abstract before you have seen the second / third usage of something
- ▶ “Duplication is better than the wrong abstraction”¹

¹<https://medium.com/@rdsbhas/>

10-modern-software-engineering-mistakes-bc67fbef4fc8

Logic in Value Objects should model **eternal truth**

- ▶ There is almost no eternal truth in products
- ▶ There might be some eternal truth in libraries
- ▶ There is much global truth encoded in your project already

Facades are good if they **simplify** multiple cases of usage of externals

- ▶ Do not implement the next generic query / mapper / ...



The Joy Of Refactoring

“Legacy Code” already contains all the “stable”
business logic

- ▶ Let the patterns emerge
- ▶ Rewrites would, as always, forget some (40%) of the domain rules
- ▶ When refactoring, a developer can do Domain Driven Design on its own

Refactoring is default behaviour

- ▶ Just like testing, you refactor code when it makes sense
- ▶ The Domain changes – so does the code

When shall I refactor?

- ▶ The more you wait the more you know
- ▶ The more technical dept there is, the harder it gets
- ▶ Same as testing: Keep the crucial parts clean & tested

Summary

Library	Project	Product
Extensibility +	Extensibility -	Extensibility +++
API-Stability +++	API-Stability -	API-Stability +
Modifiability +	Modifiability +++	Modifiability ++
Optimize for stability with customizations	Optimize for change	Optimize for modifications with stable core

Do **not** abstract!*

You are not Symfony

* Except you develop a product or library or it makes sense.



<https://qafoo.com/newsletter>

THANK YOU

Rent a quality expert
qafoo.com