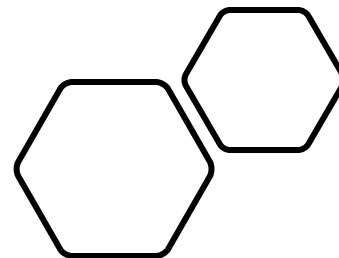


# Analizando o modelo de camadas

Prof. Bernardo Copstein



# O que tem de errado com camadas?

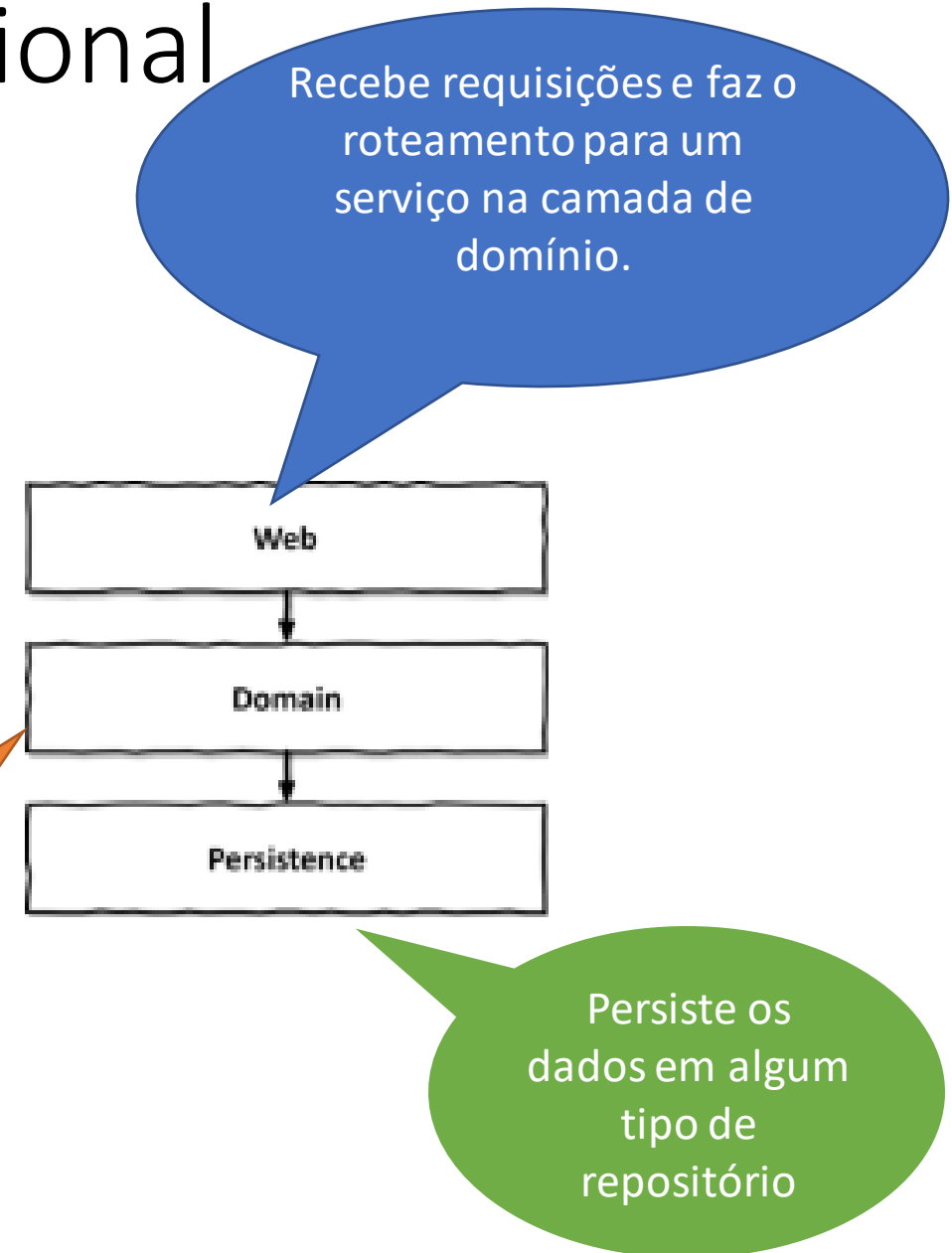


Leituras recomendadas:

- Get Your Hands Dirty on Clean Architecture. Tom Hombergs

# O modelo de camadas tradicional

- Quase todo mundo já ouviu falar no uso do modelo de 3 camadas para a construção de aplicações WEB.
- Diversos títulos na literatura procuraram explorar esta abordagem
- Na verdade “camadas” é um padrão arquitetural sólido. Se bem usado consegue-se manter a lógica do domínio independente das camadas de WEB e de persistência. Pode-se trocar as tecnologias destas camadas sem afetar a lógica. Também é possível acrescentar novas funcionalidades sem afetar as existentes.



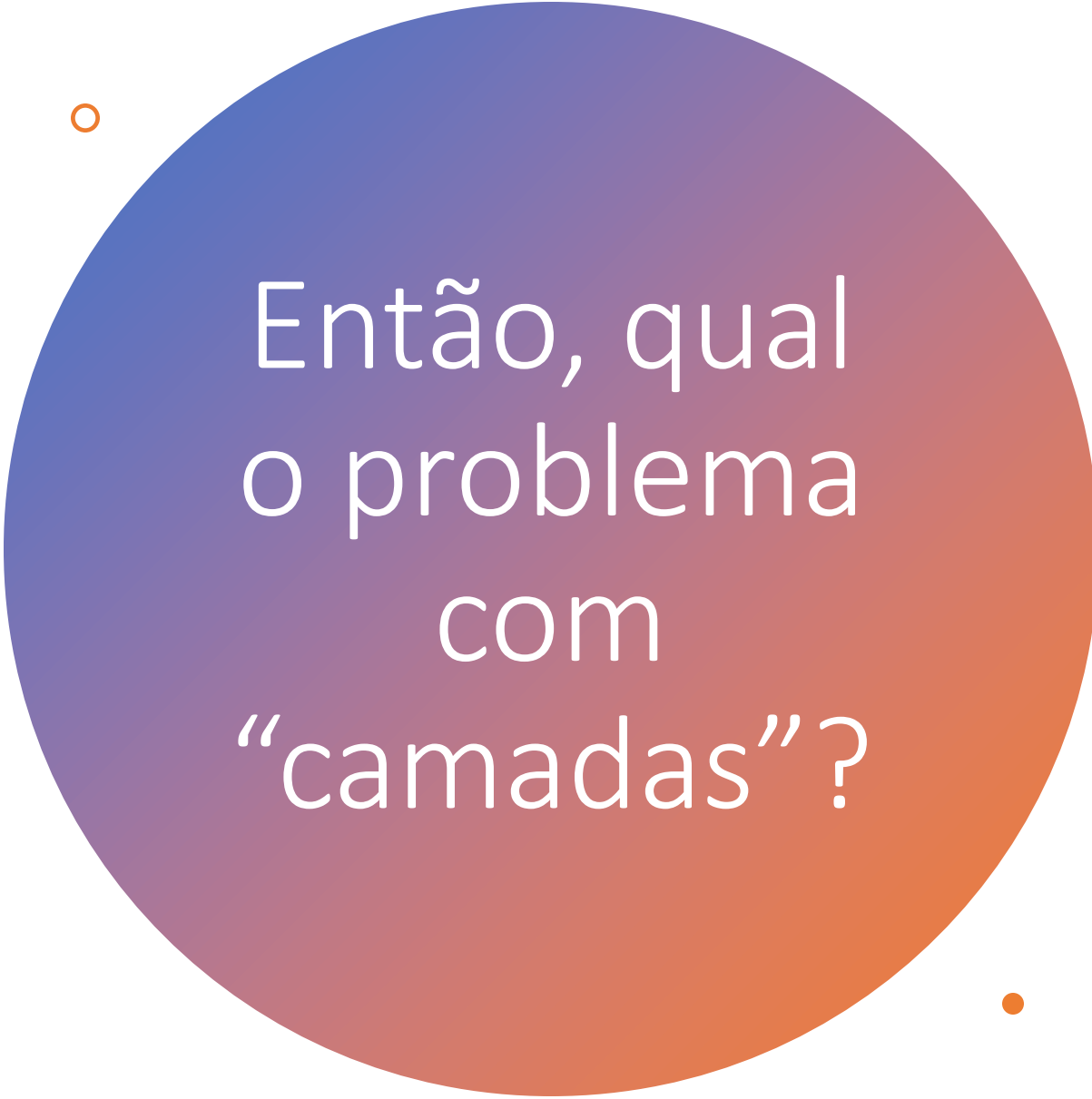

+

•


○

# Uma boa opção


Com uma boa arquitetura em camadas mantemos nossas opções em aberto e estamos aptos para rapidamente adaptar o sistema a mudanças nos requisitos e fatores externos. E se acreditamos em Bob Martin, isso é exatamente o que significa arquitetura de software.



Então, qual  
o problema  
com  
“camadas”?



Uma arquitetura em camadas deixa muitos flancos abertos que levam a maus hábitos. Estes maus hábitos tendem a tornar o software difícil de alterar com o tempo.



# P1: Promove projeto orientado pelo BD



- Em uma arquitetura em camadas tudo é construído a partir da camada de persistência (do banco de dados em última análise)
- Quando se modela uma aplicação, o principal é entender o comportamento do negócio/domínio.
  - Casos de uso ou histórias são usados para definir comportamento não estado
  - O banco de dados armazena o estado do sistema
- A estrutura de dependências do modelo em camadas nos leva a desenvolver primeiro o banco de dados já que tudo é construído a partir dele.
- Isso pode fazer sentido em função do fluxo de dependências, mas não faz nenhum sentido do ponto de vista da lógica de negócios

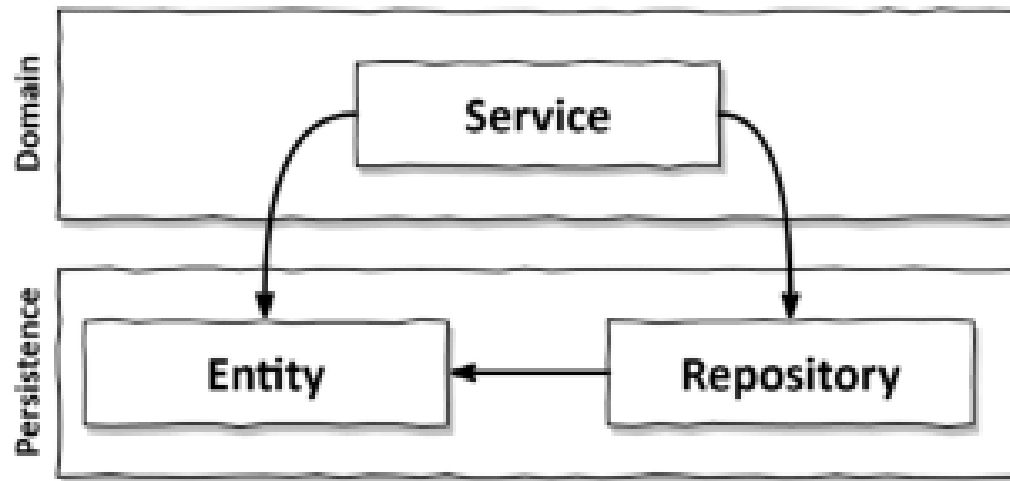
---

# O ideal seria ...

- Iniciar a modelagem pela lógica da aplicação
- Somente quando tivermos entendido e modelado corretamente a lógica da aplicação é que deveríamos construir a persistência e a interface ao redor dela



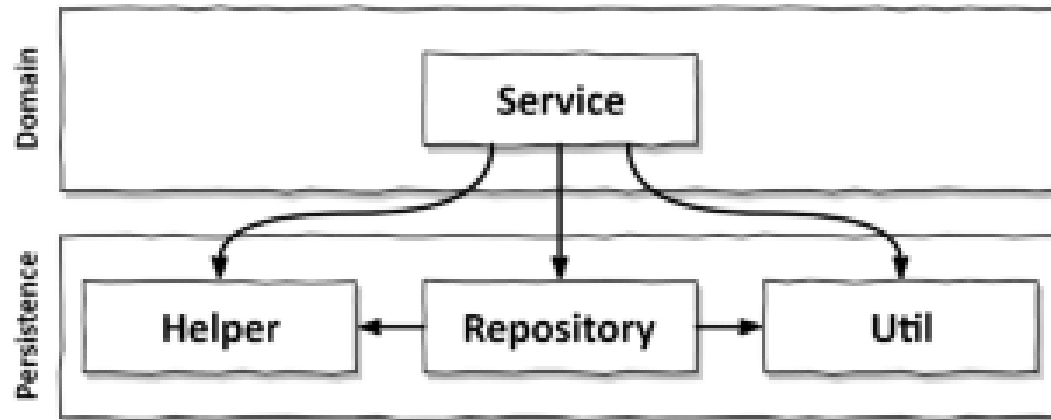
# E tudo piora com o uso de uma API ORM



- APIs de ORM (JPA, Hibernate) facilitam enormemente o desenvolvimento
- ORM + camadas = mistura das regras de negócio com persistência
- Normalmente entidades gerenciadas pelo ORM fazem parte da camada de persistência
- Nada impede que a camada de domínio acesse estas entidades visto que estão em uma camada inferior
- Isso leva à um acoplamento forte entre a camada de domínio e a camada de persistência
- Esse acoplamento implica que se torna difícil alterar um sem alterar o outro

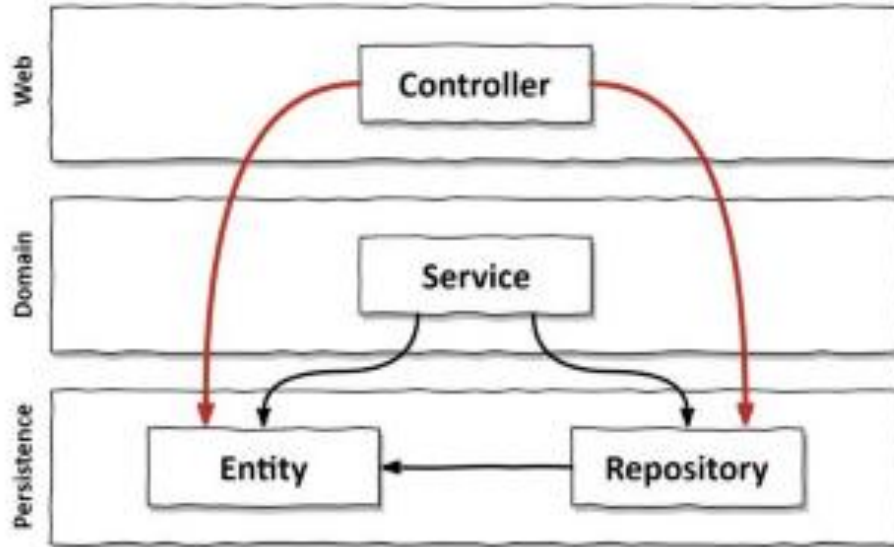


# P2: São propensas à atalhos



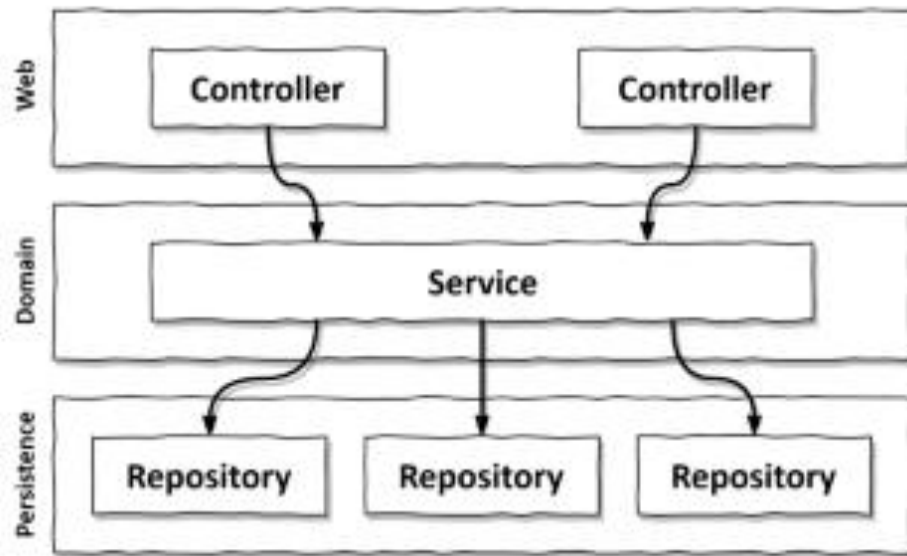
- Em um modelo de camadas um componente só pode acessar outros na mesma camada ou em uma camada abaixo
- Se surge a necessidade de acessar um componente que está em uma camada acima basta puxar ele para baixo: problema resolvido !!
- Fazer isso de vez em quando não é um problema, mas ...
- Com o tempo a tendência é que a camada de persistência “engorde”

# P3: Os testes tendem a complicar



- É comum em uma arquitetura em camadas “pular” camadas
- Se uma funcionalidade corresponde a um acesso ao banco é normal que a camada de interface acesse diretamente a camada de persistência
- Problemas:
  - Mistura parte da lógica na camada de interface
  - Torna-se difícil testar a camada de interface. Será necessário criar doubles tanto para a lógica como para a persistência

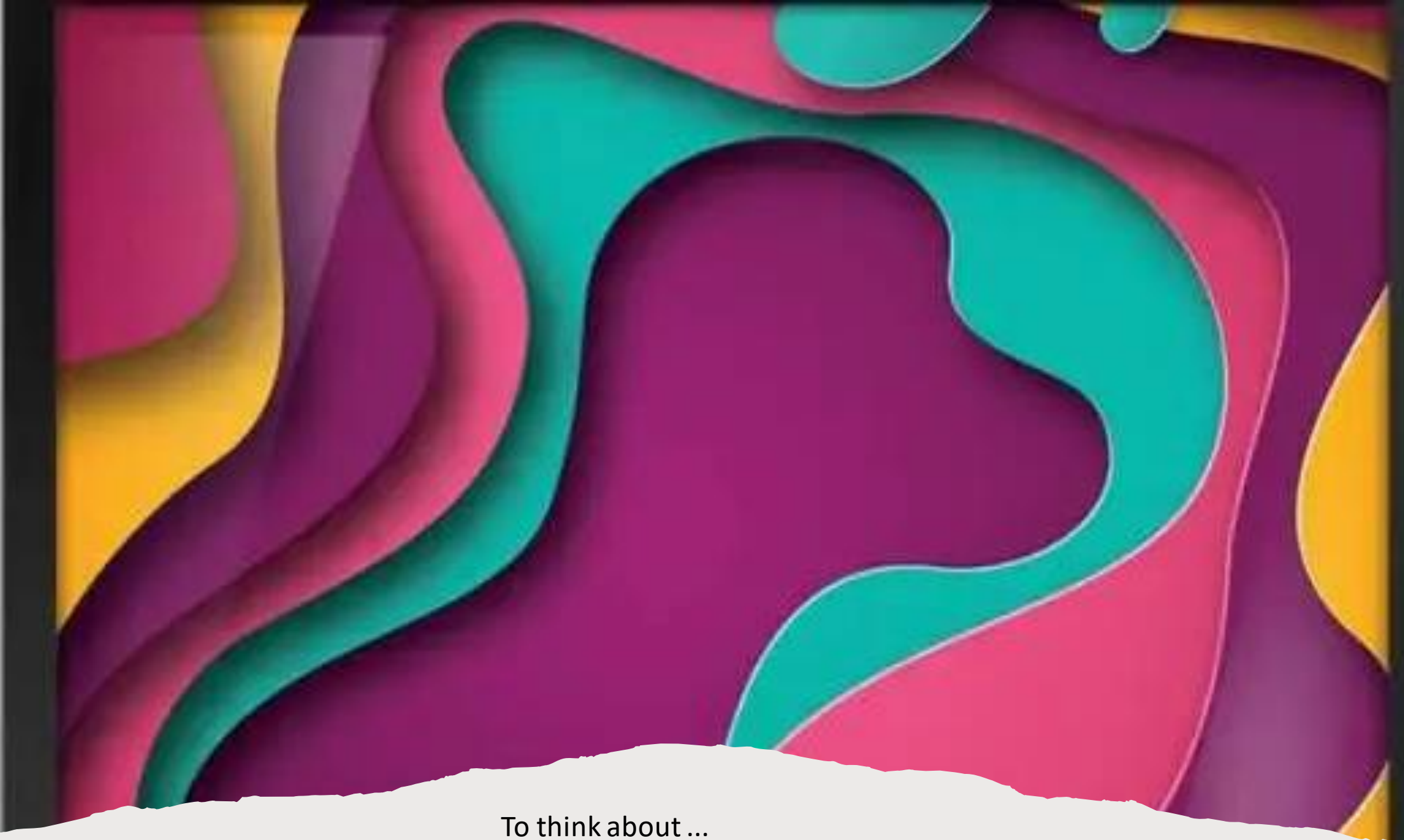
# P4: Tendem a esconder os casos de uso



- Em uma arquitetura de camadas facilmente a lógica “se esparrama” pelas camadas.
  - Pode existir na camada de interface se pular a camada de negócios nos casos de uso “simples”
  - Pode existir na camada de persistência se tivermos empurrado algum componente para baixo
- Isso torna complicado localizar em que camada novas funcionalidade devem ser incluídas
- Além disso nada no modelo determina o “tamanho” de um serviço. Com o tempo eles crescem em excesso atendendo a muitos casos de uso.
- Serviços grandes dependem de muitos componentes na camada de persistência e muitos componentes da camada de interface dependem dele
  - O grau de acoplamento do sistema cresce !!

# P5: Torna difícil o trabalho em paralelo

- Os problemas anteriormente mencionados fazem com que, com o tempo, o trabalho em paralelo se torne impossível
- O grau de acoplamento do sistema faz com que qualquer alteração implique em mudanças em todas as camadas
- O desenvolvimento de novos casos de uso em paralelo leva os mesmos componentes a serem editados simultaneamente o que leva a conflitos que tem de ser resolvidos



To think about ...