Embrace the Duplication

Or: why CQRS is cool and similar things must not always be the same

Disclaimer (as I was saying)

All code will be valid code and (usually) not *inherently* wrong.

Your job is to choose the right way in the right moment for the right scope and the right problem and the right… you get the idea.

What's wrong here?

```
int maxSearchDepth = 10^10; // We didn't get more memory from management
```

What's wrong here? Let's refactor …

```
int maxSearchDepth = 10^10;
```

Extract ...

```
int exponent = 10;
int maxSearchDepth = 10^exponent;
```

See same constant value and replace …

```
int exponent = 10;
int maxSearchDepth = exponent^exponent;
```

See same constant value and replace …

```
int exponent = 10;
int maxSearchDepth = exponent^exponent;
```

Similar things must not always be the same!

```
int base = 10;
int exponent = 10;
int maxSearchDepth = base^exponent;
```

Know when to sthaaap

```
int common_base_and_exponent_value = 10;
int base = common_base_and_exponent_value;
int exponent = common_base_and_exponent_value;
int maxSearchDepth = base^exponent;
```

Another example from another perspective …

```
class Customer {
  String name;
  boolean isAdmin;
}

// In your service/controller
void login(Customer c);

void orderThings(Customer c, List<Items> basket);

void dropAllDatabases(Customer c); // Memo to Joe: check the isAdmin flag!!1!
```

Let's help Joe

```
class Customer {
  String name;
}
```

```
class Admin {
  String name;
}
```

```
// In your service/controller
void login(Customer c);

void orderThings(Customer c, List<Items> basket);

void dropAllDatabases(Customer c); // Memo to Joe: check the isAdmin flag!!1!
```

Let's help Joe

```
class Customer {
  String name;
}
```

```
class Admin {
  String name;
}
```

```
// In your service/controller
void login(Customer c); // Sorry Joe, login with token from server cp12971

void orderThings(Customer c, List<Items> basket);

void dropAllDatabases(Admin a);
```

Adding another layer of indirection …

```
class Customer : User {
}
```

```
class Admin : User {
}
```

```
abstract class User {
  String name;
}
```

```
// In your service/controller
void login(User u);

void orderThings(Customer c, List<Items> basket);

void dropAllDatabases(Admin a);
```

Another common example

```
class ToDo {
  String what;
  User by;
}
```

```
// In your service/controller
void create(ToDo t);
Todo getNext();
```

Feature Request: Add Notes to ToDo

```
class ToDo {
  String what;
  User by;
}
```

```
// In your service/controller
void create(ToDo t);
Todo getNext();
```

Feature Request: Add Notes to ToDo

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
}

// In your service/controller
void create(ToDo t);
Todo getNext();
```

Feature Request: Add Notes to ToDo

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
}

// In your service/controller
void create(ToDo t); // Memo to Joe: init with empty Notes List!!1!
Todo getNext();
```

Poor Joe

Let’s help him again!

Are all “ToDo” equal?

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
}

// In your service/controller
void create(ToDo t); // Memo to Joe: init with empty Notes List!!1!
Todo getNext();
```

Are all “ToDo” equal?

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
}

// In your service/controller
void create(ToDo t); // Memo to Joe: init with empty Notes List!!1!
Todo getNext();
```

CQRS* to the rescue! (“Command and Query Responsibility Segregation”)

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
}
```

```
class NewToDo {
  String what;
  User by;
}
```

```
// In your service/controller
void create(ToDo t); // Memo to Joe: init with empty Notes List!!1!
Todo getNext();
```

CQRS* to the rescue! (“Command and Query Responsibility Segregation”)

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
}
```

```
class NewToDo {
  String what;
  User by;
}
```

```
// In your service/controller
void create(NewToDo t); // Map NewToDo → ToDo is trivial
Todo getNext();
```

New Feature Request: Add Created Date

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
}
```

```
class NewToDo {
  String what;
  User by;
}
```

```
// In your service/controller
void create(NewToDo t); // Map NewToDo → ToDo is trivial
Todo getNext();
```

New Feature Request: Add Created Date

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
  Date created;
}
```

```
class NewToDo {
  String what;
  User by;
}
```

```
// In your service/controller
void create(NewToDo t); // Map NewToDo → ToDo is trivial
Todo getNext();
```

Again: know when to stop

```
class ToDo : BaseToDo {
  List<String> notes;
  void addNote(String n);
  Date created;
}
```

```
class NewToDo : BaseToDo
{
}
```

```
abstract class BaseToDo {
  String what;
  User by;
}
```

```
// In your service/controller
void create(NewToDo t); // Map NewToDo → ToDo is trivial
Todo getNext();
```

Why? Change Request: Create now only with UserId

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
  Date created;
}
```

```
class NewToDo {
  String what;
  User by;
}
```

```
// In your service/controller
void create(NewToDo t); // Map NewToDo → ToDo is trivial
Todo getNext();
```

Why? Change Request: Create now only with UserId

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
  Date created;
}
```

```
class NewToDo {
  String what;
  int byUserId;
}
```

```
// In your service/controller
void create(NewToDo t); // Map NewToDo → ToDo is trivial
Todo getNext();
```

Why? Change Request: Create now only with UserId

```
class ToDo {
  String what;
  User by;
  List<String> notes;
  void addNote(String n);
  Date created;
}
```

```
class NewToDo {
  String what;
  int byUserId;
}
```

```
// In your service/controller
void create(NewToDo t); // Map NewToDo → ToDo is trivial-ish, diverging models
Todo getNext();
```

Why I like CQRS

- Your commands usually have different *needs* than your queries.
  - Using the same models for Commands and Queries results in all sorts of violations, starting with Interface Segregation, Least Astonishment etc.
- Separation of Concerns (the “Responsibility Separation” part) makes your code easier to read, because you (h0m@n) can follow it in smaller chunks.
- Usually easier to extend as seen (*pluginability*).
- Usually also easier to test.

“We can solve any problem by introducing an extra level of indirection …”

- CQRS leads to more advanced concepts like request/response handling on single handler, which are common in the eventing world (@CSS: Metro, Kafka, keyword “Event Sourcing”)
- All those pattern try to solve the same problem: separate the processing of your “events” by also separating the input/output models.
    - So you can turn it around: by having separate command and query *models*, you enable - or even force - command and query separation.
- Your REST-Controllers most probably have the same “problem”!
    - Separate Request and Response Objects are cool too ;)
    - Have a look at **PipelineR** (Java) or **MediatR** (C#)

“… except the problem of too many layers of indirection”

- Don’t be afraid from too many Classes/Records
  - Especially in big complex systems like at CSS, the flexibility they give is priceless
  - *However*, be mindful on how you structure your *packages* of your CQRS-Models/DTOs. Separate by (sub-)domain or by command type or … well what works for **you**. Keyword for DDD: Bounded Context
  - Be vigilant in consistent naming (*New*ToDo vs *Create*User).
- It’s no silver bullet, use with care (overuse can make systems needlessly complicated).

Questions?

I might have answers. Maybe. Or more questions.

Acknowledgement and Contact

All Code written by me or “in spirit” by my wonderful colleagues at CSS.

Quotes by David J. Wheeler and Kevlin Henney.

CQRS by Greg Young and explained by Martin Fowler, based on CQS by Bertrand Meyer.

david.thalmann@gmail.com / david.thalmann@css.ch

github.com/boast