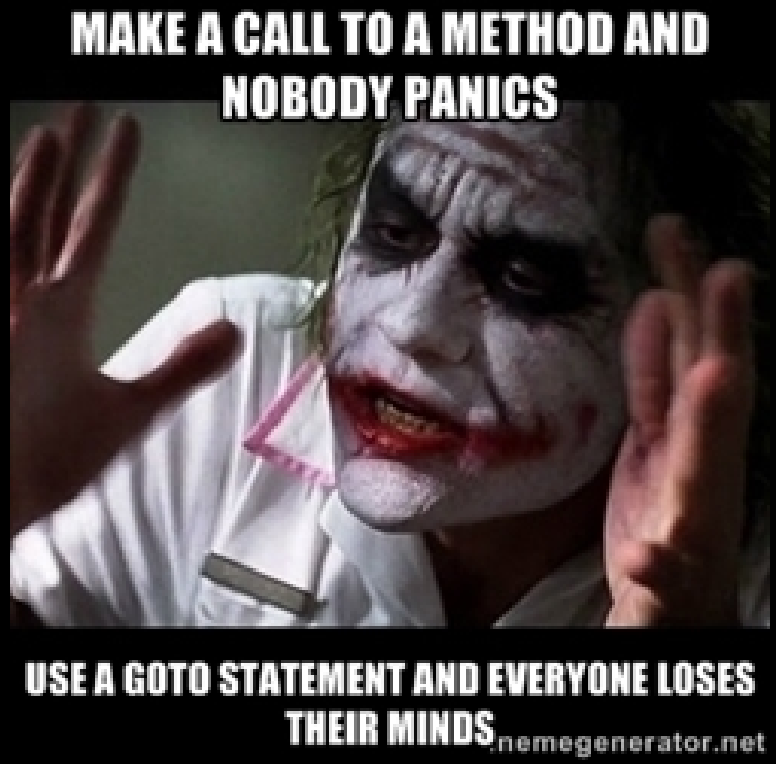
Messages are the gotos of OOP

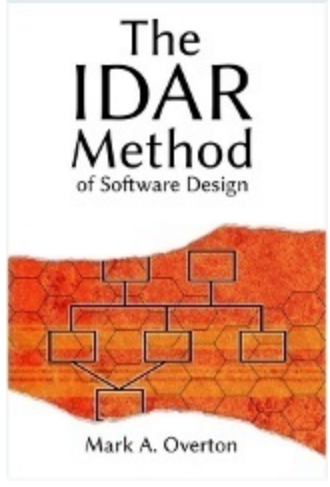


Messages are the gotos of OOP

Messages are the gotos of OOP

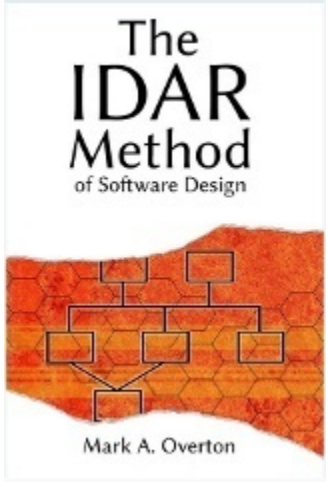
- Traditional OOD/OOP does not encourage or enforce the use of levels of abstraction, so designs usually slide into messiness, as code did with goto statements. --*Mark A. Overton*

Messages are the gotos of OOP



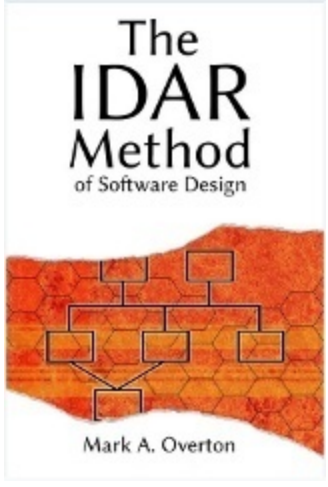
Messages are the gotos of OOP

- A significant improvement in software design.



Messages are the gotos of OOP

- A significant improvement in software design.
- A better method of designing object-oriented software was recently invented.



Samenvatting

Samenvatting

Samenvatting

lets genuanceerder:

Samenvatting

lets genuanceerder:

- Er zit een interessant idee in dit boek.

Samenvatting

lets genuanceerder:

- Er zit een interessant idee in dit boek.
- De auteur verpakt het niet zo best.

Samenvatting

lets genuanceerder:

- Er zit een interessant idee in dit boek.
- De auteur verpakt het niet zo best.
- De auteur leeft in zijn eigen wereld.

Samenvatting

lets genuanceerder:

- Er zit een interessant idee in dit boek.
- De auteur verpakt het niet zo best.
- De auteur leeft in zijn eigen wereld.
- Schoenmaker blijf bij je eigen leest.

"Ignorance"

"PHP encourages business logic and presentation logic to be mixed together, but I think that's poor practice because ..."

"Ignorance"

"PHP encourages business logic and presentation logic to be mixed together, but I think that's poor practice because (1) it means that changing one can accidentally introduce bugs in the other ..."

"Ignorance"

"PHP encourages business logic and presentation logic to be mixed together, but I think that's poor practice because (1) it means that changing one can accidentally introduce bugs in the other, and (2) such mixing reduces the learnability of both. ..."

"Ignorance"

"PHP encourages business logic and presentation logic to be mixed together, but I think that's poor practice because (1) it means that changing one can accidentally introduce bugs in the other, and (2) such mixing reduces the learnability of both. Keep business logic and presentation logic separate, as shown in Figure 6.8."

"Ignorance"

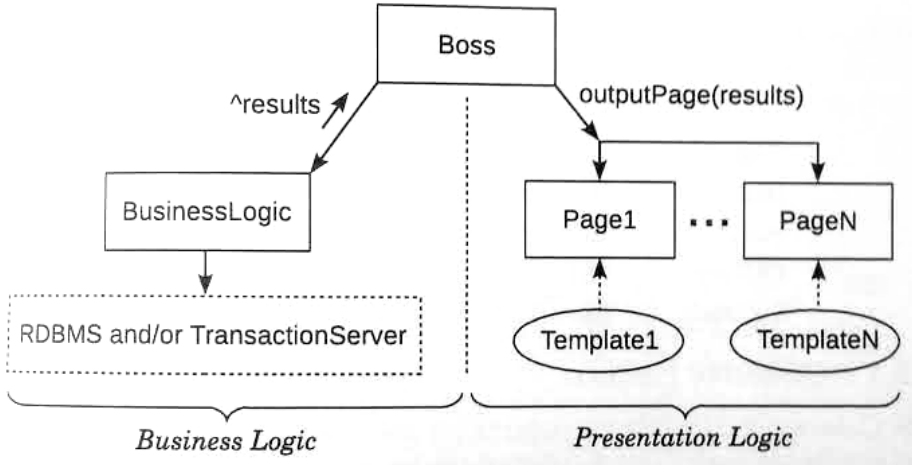


Figure 6.8: Web App Server pattern

"Ignorance"

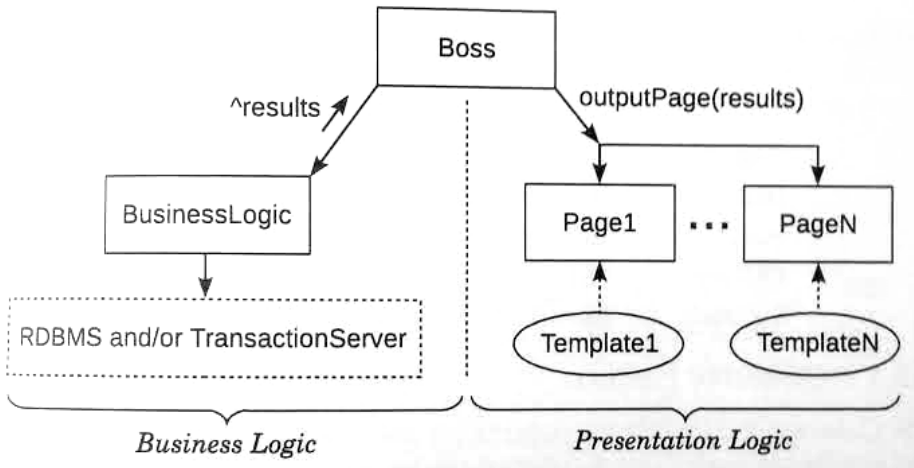


Figure 6.8: Web App Server pattern

"This Web App Server pattern is similar to the MVC (Model-View-Controller) architecture pattern."

Waarom dit onderwerp?

Waarom dit onderwerp?

- Herken toch een kern van waarheid

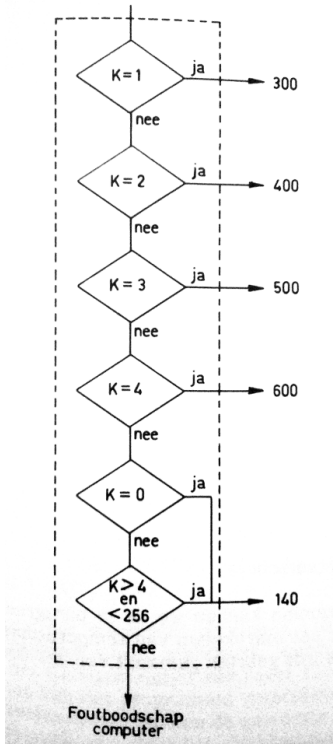
Waarom dit onderwerp?

- Herken toch een kern van waarheid
- Niet echt tijd meer om te switchen van onderwerp ;-)

Goto statements

Goto statements

```
100 REM * OEFENING MET ON ... GOTO *
110 CLS
120 INPUT "Uw keuze";K
130 ON K GOTO 300,400,500,600
140 PRINT "K=0 of K>4 maar <256"
150 PRINT:GOTO 120
300 PRINT "Celcius naar Fahrenheit"
310 PRINT:GOTO 120
400 PRINT "Fahrenheit naar Celcius"
410 PRINT:GOTO 120
500 PRINT "Celcius naar Kelvin"
510 PRINT:GOTO 120
600 PRINT "Fahrenheit naar Kelvin"
610 PRINT:GOTO 120
```



Goto statements

```
01.     vector<int> squareVec1(vector<int> v)
02.     {
03.         auto it = begin(v);
04.         loopBegin:
05.         if (it == end(v))
06.             goto loopEnd;
07.         *it = *it * *it;
08.         ++it;
09.         goto loopBegin;
10.         loopEnd:
11.         return v;
12.     }
```

Goto statements 

```
01.     vector<int> squareVec1(vector<int> v)
02.     {
03.         auto it = begin(v);
04.         loopBegin:
05.         if (it == end(v))
06.             goto loopEnd;
07.         *it = *it * *it;
08.         ++it;
09.         goto loopBegin;
10.         loopEnd:
11.         return v;
12.     }
```

Goto statements

```
01.     vector<int> squareVec2(vector<int> v)
02.     {
03.         auto it = begin(v);
04.         while (it != end(v))
05.         {
06.             *it = *it * *it;
07.             ++it;
08.         }
09.         return v;
10.     }
```

Goto statements

```
01.     vector<int> squareVec3(vector<int> v)
02.     {
03.         for (auto it = begin(v); it != end(v); ++it)
04.         {
05.             *it = *it * *it;
06.         }
07.         return v;
08.     }
```

Goto statements

```
01.     vector<int> squareVec4(vector<int> v)
02.     {
03.         for (int& i : v)
04.         {
05.             i = i * i;
06.         }
07.         return v;
08.     }
```

Goto statements

```
01.     vector<int> squareVec5(vector<int> v)
02.     {
03.         transform(begin(v), end(v), begin(v), [](int i)
04.         {
05.             return i*i;
06.         });
07.         return v;
08.     }
```

Goto statements

```
01.     vector<int> result;
02.     for (int i : v)
03.     {
04.         if (i % 2 == 0)
05.             result.push_back(i);
06.     }
```

Goto statements

```
01.     vector<int> result;
02.     for (int i : v)
03.     {
04.         if (i % 2 == 0)
05.             result.push_back(i);
06.     }
07.
08.     vector<int> result;
09.     copy_if(begin(v), end(v), back_inserter(result), [](int i)
10.     {
11.         return i % 2 == 0;
12.     });
```

Goto statements

```
01.     template<typename Container, typename Functor>
02.     Container transformCont(Container xs, Functor op)
03.     {
04.         transform(begin(xs), end(xs), begin(xs), op);
05.         return xs;
06.     }
```

Goto statements

```
01.     template<typename Container, typename Functor>
02.     Container transformCont(Container xs, Functor op)
03.     {
04.         transform(begin(xs), end(xs), begin(xs), op);
05.         return xs;
06.     }
07.
08.     vector<int> squareVec6(const vector<int>& v)
09.     {
10.         return transformCont(v, [](int i)
11.         {
12.             return i*i;
13.         });
14.     }
```

Goto statements

Come from statements

The author feels that the COME FROM will prove an invaluable contribution to the field of computer science.

Come from statements

```
10 J=1
```

Come from statements

```
10 J=1

11 COME FROM 20
```

Come from statements

```
10 J=1

11 COME FROM 20

12 WRITE (6,40) J STOP
```

Come from statements

```
10 J=1

11 COME FROM 20

12 WRITE (6,40) J STOP

13 COME FROM 10
```

Come from statements

```
10 J=1

11 COME FROM 20

12 WRITE (6,40) J STOP

13 COME FROM 10

20 J=J+2
```

Come from statements

```
10 J=1

11 COME FROM 20

12 WRITE (6,40) J STOP

13 COME FROM 10

20 J=J+2

40 FORMAT (14)
```

Voorbeeld van die chaos van messages

Voorbeeld van die chaos van messages

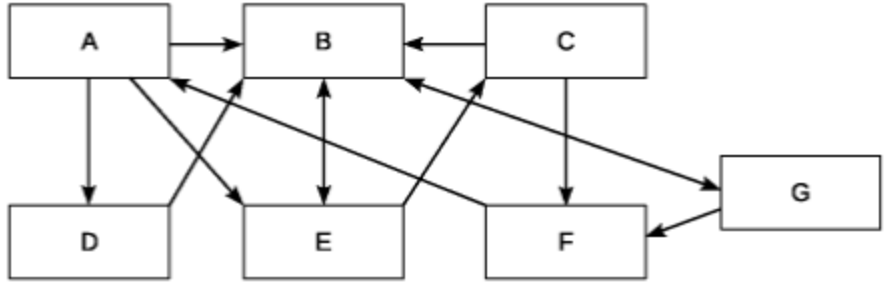


Figure 1.1: Unrestricted messages encourage disorganization

Voorbeeld van die chaos van messages

- Today's OOP is naturally disorganized because messages (i.e., method-calls) among objects are not constrained in any way.

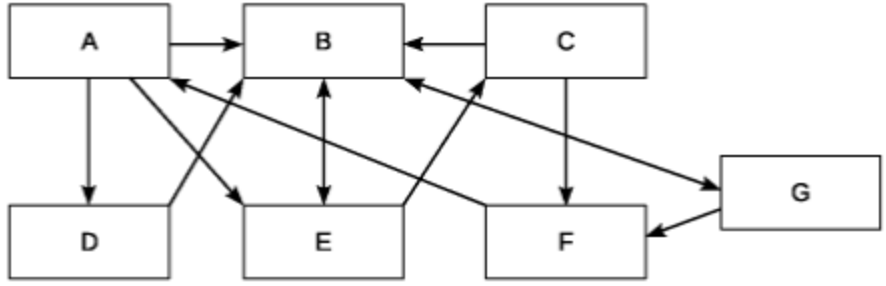


Figure 1.1: Unrestricted messages encourage disorganization

Voorbeeld van die chaos van messages

- Today's OOP is naturally disorganized because messages (i.e., method-calls) among objects are not constrained in any way. You are free to send a message to (call a method in) any object you wish.

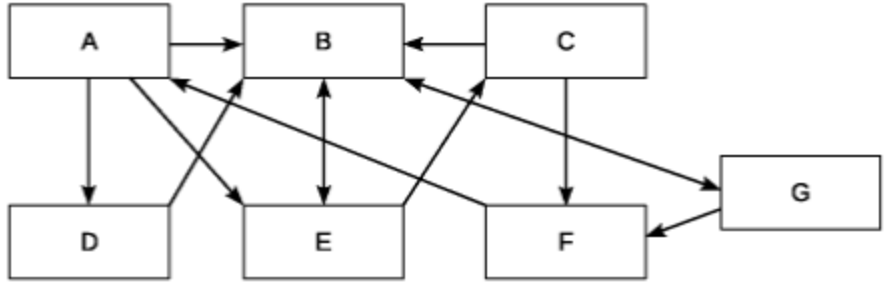


Figure 1.1: Unrestricted messages encourage disorganization

Voorbeeld van die chaos van messages

- Today's OOP is naturally disorganized because messages (i.e., method-calls) among objects are not constrained in any way. You are free to send a message to (call a method in) any object you wish. Like unrestricted `goto` statements, unrestricted messages soon descend into a mess, as shown in fig.1.1.

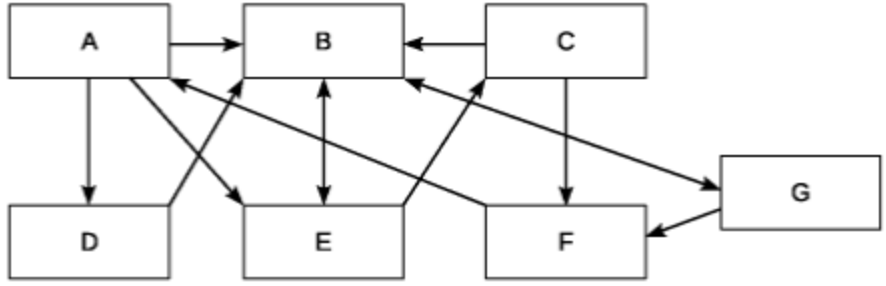


Figure 1.1: Unrestricted messages encourage disorganization

Voorbeeld van die chaos van messages

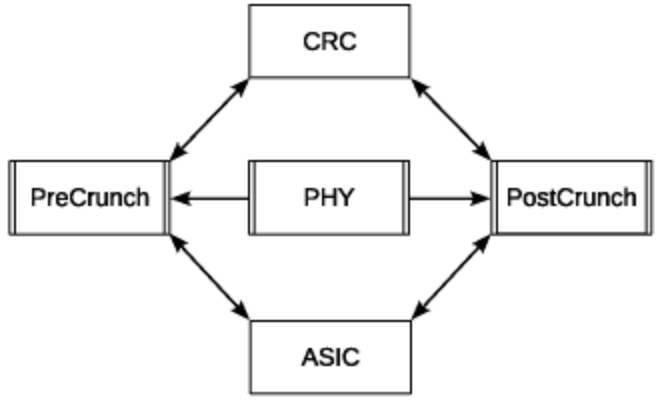


Figure 1.3: UML communication diagram of a PHY

Voorbeeld van die chaos van messages

- "Object-oriented design is fundamentally different from procedural design. Objects are structured in a network and not a hierarchy." -- Wirfs-Brock and McKean

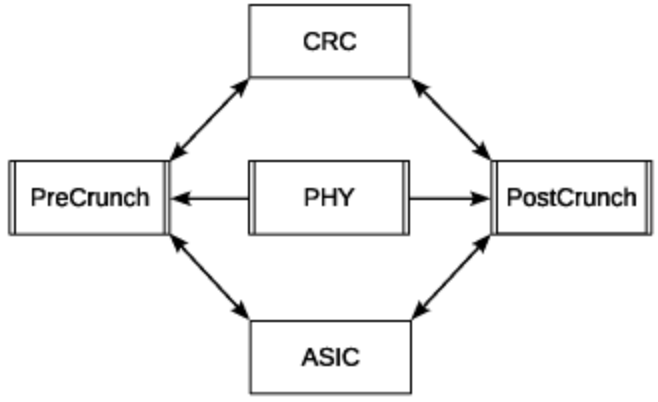


Figure 1.3: UML communication diagram of a PHY

Voorbeeld van die chaos van messages

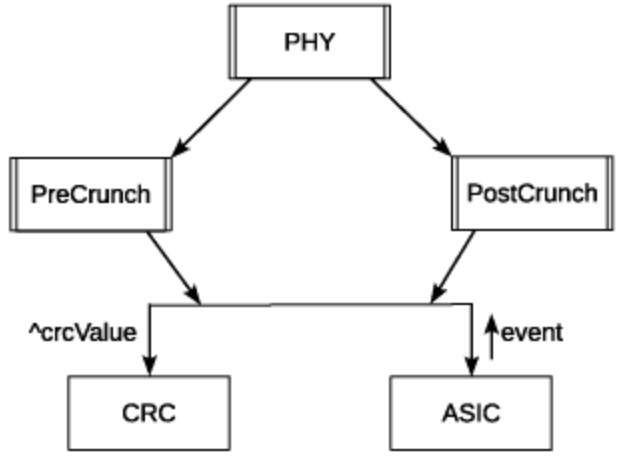


Figure 1.5: IDAR graph of a PHY

Voorbeeld van die chaos van messages

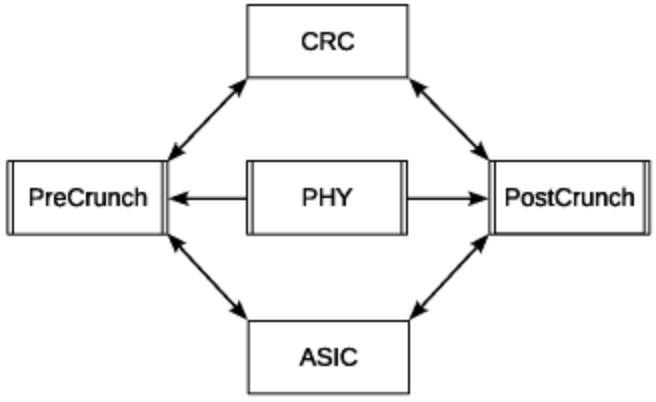


Figure 1.3: UML communication diagram of a PHY

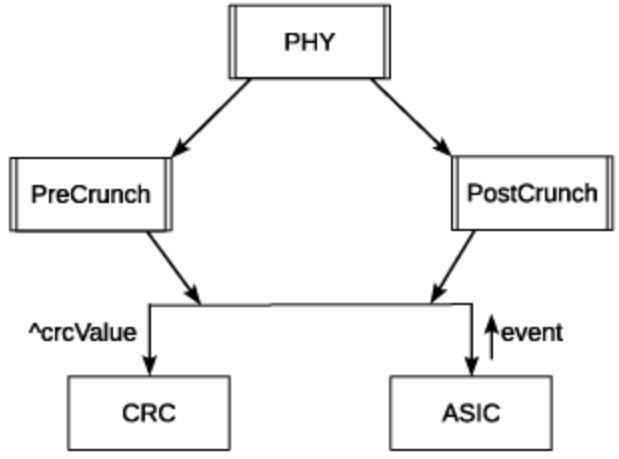


Figure 1.5: IDAR graph of a PHY

Voorbeeld van die chaos van messages

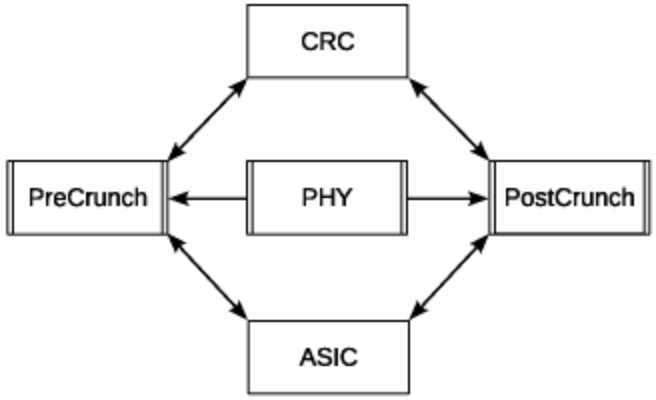


Figure 1.3: UML communication diagram of a PHY

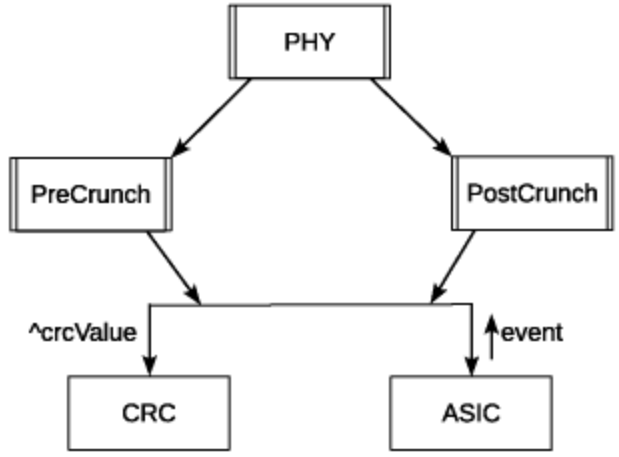


Figure 1.5: IDAR graph of a PHY

"None of the UML diagrams can portray levels because object-oriented design (surprisingly) does not possess the concept of levels of abstraction amongst objects. So UML will not help you." -- Mark.

IDAR Method

Rules:

- **Identify**
- **Down**
- **Aid**
- **Role**

IDAR Method

Rules:

- **Identify**: Every public method in each class must be identified as either a *command* or *notice*.
- **Down**
- **Aid**
- **Role**

IDAR Method

Rules:

- **Identify**: command or notice
- **Down**: When graphing calls to commands among objects the arrows must point down.
- **Aid**
- **Role**

IDAR Method

Rules:

- **Identify**: command or notice
- **Down**: arrows down
- **Aid**: Each notice must aid one or more commands in its class.
- **Role**

IDAR Method

Rules:

- **Identify**: command or notice
- **Down**: arrows down
- **Aid**: Each notice must aid one or more commands in its class.
- 1) The command must need the information conveyed in the notice
- **Role**

IDAR Method

Rules:

- **Identify**: command or notice
- **Down**: arrows down
- **Aid**: Each notice must aid one or more commands in its class.
- 1) The command must need the information conveyed in the notice
- 2) If a prior call to that command did not finish all of its actions, the notice is allowed to perform some or all of those unfinished actions.
- **Role**

IDAR Method

Rules:

- **Identify**: command or notice
- **Down**: arrows down
- **Aid**: Each notice must aid one or more commands in its class.
- 1) The command must need the information conveyed in the notice
- ~~2) If a prior call to that command did not finish all of its actions, the notice is allowed to perform some or all of those unfinished actions.~~
- **Role**

IDAR Method

Rules:

- **Identify**: command or notice
- **Down**: arrows down
- **Aid**: notice aids command{s}
- **Role**: For each class and command write a role describing the gist of the service it provides.

IDAR Method

Rules:

- **Identify**: command or notice
- **Down**: arrows down
- **Aid**: notice aids command{s}
- **Role**: determine exact roles.

IDAR Method

Rules:

- **Identify**: command or notice ==> Command/Query separation
- **Down**: arrows down ==> Hierarchy ("Directed acyclic graph")
- **Aid**: notice aids command{s} ==> ???
- **Role**: determine exact roles. ==> API docs

IDAR graph

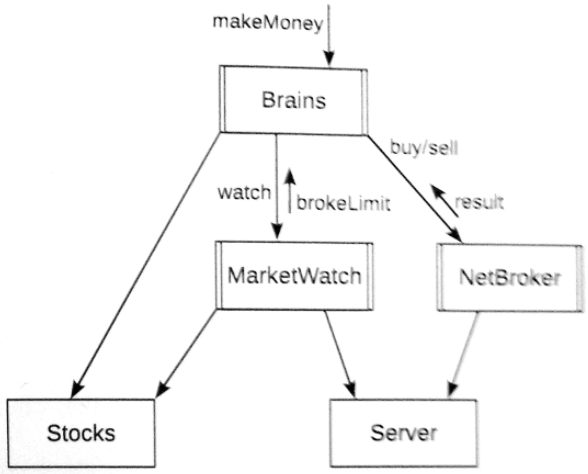


Figure 3.1: Computerized stock-trading program

Method	C/N	Sender	Recipient	Role
watch	C	Brains	MarketWatch	Watch list of stocks
brokeLimit	N	MarketWatch	Brains	Stock exceed limits
buy/sell	C	Brains	NetBroker	Buy/sell this stock
result	N	NetBroker	Brains	Result of buy/sell

Table 3.1: Commands and notices used in stock-trading program

IDAR graph

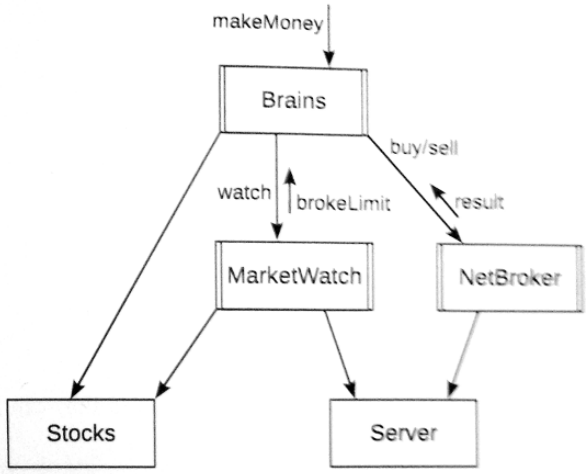


Figure 3.1: Computerized stock-trading program

Method	C/N	Sender	Recipient	Role
watch	C	Brains	MarketWatch	Watch list of stocks
brokeLimit	N	MarketWatch	Brains	Stock exceed limits
buy/sell	C	Brains	NetBroker	Buy/sell this stock
result	N	NetBroker	Brains	Result of buy/sell

Table 3.1: Commands and notices used in stock-trading program

```
01. class Brains
02. {
03.     public function makeMoney()
04.     {
05.         while (true) {
06.             $this->marketWatch->watch();
07.         }
08.     }
09.     public function brokeLimit($limit)
10.     {
11.         if ($limit < self::someThreshold) {
12.             $this->netBroker->buy();
13.         }
14.         elseif ($limit > self::someThreshold2) {
15.             $this->netBroker->sell();
16.         }
17.     }
18.     public function result(...)
19.     {
20.         ...
21.     }
22. }
```

IDAR graph

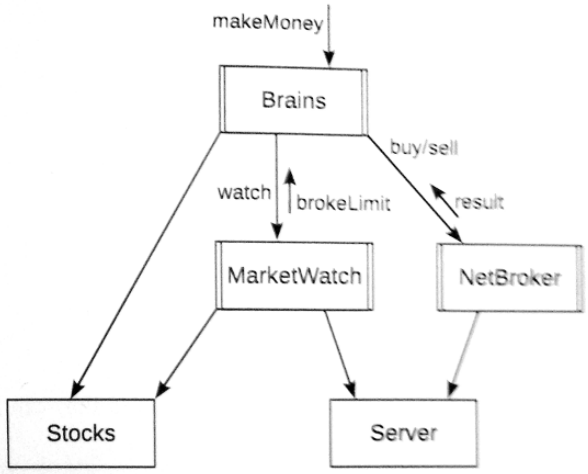


Figure 3.1: Computerized stock-trading program

Method	C/N	Sender	Recipient	Role
watch	C	Brains	MarketWatch	Watch list of stocks
brokeLimit	N	MarketWatch	Brains	Stock exceed limits
buy/sell	C	Brains	NetBroker	Buy/sell this stock
result	N	NetBroker	Brains	Result of buy/sell

Table 3.1: Commands and notices used in stock-trading program

```
01. class MarketWatch
02. {
03.     public function watch()
04.     {
05.         $this->server->getCurrentPrice();
06.     }
07.
08.     public function currentPrice($currentPrice)
09.     {
10.         $this->currentPrice = $currentPrice;
11.         $this->stocks->getBoughtPrice();
12.     }
13.
14.     public function boughtPrice($boughtPrice)
15.     {
16.         $limit = $this->currentPrice;
17.         $limit /= $boughtPrice;
18.         $this->brains->brokeLimit($limit);
19.     }
20. }
```

IDAR graph

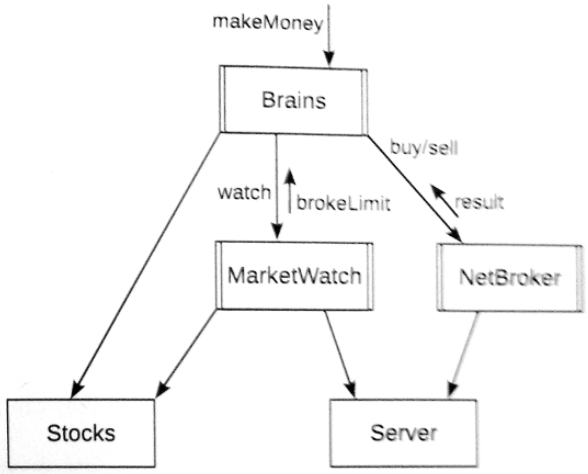


Figure 3.1: Computerized stock-trading program

Method	C/N	Sender	Recipient	Role
watch	C	Brains	MarketWatch	Watch list of stocks
brokeLimit	N	MarketWatch	Brains	Stock exceed limits
buy/sell	C	Brains	NetBroker	Buy/sell this stock
result	N	NetBroker	Brains	Result of buy/sell

Table 3.1: Commands and notices used in stock-trading program

```
01. class MarketWatch
02. {
03.     public function watch()
04.     {
05.         $this->server->getCurrentPrice();
06.         $this->stocks->getBoughtPrice();
07.
08.         $limit = $this->currentPrice;
09.         $limit /= $this->boughtPrice;
10.
11.         $this->brains->brokeLimit($limit);
12.     }
13.
14.     public function currentPrice($currentPrice)
15.     {
16.         $this->currentPrice = $currentPrice;
17.     }
18.
19.     public function boughtPrice($boughtPrice)
20.     {
21.         $this->boughtPrice = $boughtPrice;
22.     }
23. }
```

IDAR graph

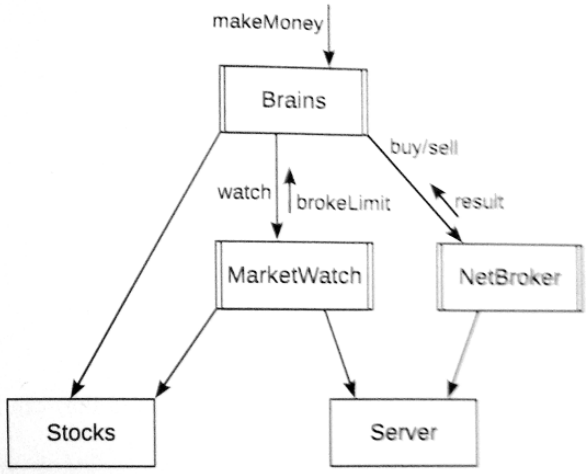


Figure 3.1: Computerized stock-trading program

Method	C/N	Sender	Recipient	Role
watch	C	Brains	MarketWatch	Watch list of stocks
brokeLimit	N	MarketWatch	Brains	Stock exceed limits
buy/sell	C	Brains	NetBroker	Buy/sell this stock
result	N	NetBroker	Brains	Result of buy/sell

Table 3.1: Commands and notices used in stock-trading program

```
01. class MarketWatch
02. {
03.     public function watch()
04.     {
05.         $this->brains->brokeLimit(
06.             $this->server->getCurrentPrice() /
07.             $this->stocks->getBoughtPrice()
08.         );
09.     }
10. }
```

4

5

Circular references

```
01.     function displayMemoryUsage($desc) {
02.         $size = memory_get_usage(true);
03.         $unit = array('b','kb','mb','gb','tb','pb');
04.
05.         printf("%-40s%s\n", $desc, @round($size/pow(1024,($i=floor(log($size,1024))))),2).'
06.     }
07.
```

Circular references

```
01.     displayMemoryUsage('before allocating a big string'); // 256 kb
02.
```

Circular references

```
01.     displayMemoryUsage('before allocating a big string'); // 256 kb
02.
03.     $a = str_repeat('A', 10 * 1024 * 1024);
04.
```

Circular references

```
01.     displayMemoryUsage('before allocating a big string'); // 256 kb
02.
03.     $a = str_repeat('A', 10 * 1024 * 1024);
04.
05.     displayMemoryUsage('after allocating'); // 10.5 mb
06.
```

Circular references

```
01.     displayMemoryUsage('before allocating a big string'); // 256 kb
02.
03.     $a = str_repeat('A', 10 * 1024 * 1024);
04.
05.     displayMemoryUsage('after allocating'); // 10.5 mb
06.
07.     unset($a);
08.
```

Circular references

```
01.     displayMemoryUsage('before allocating a big string'); // 256 kb
02.
03.     $a = str_repeat('A', 10 * 1024 * 1024);
04.
05.     displayMemoryUsage('after allocating'); // 10.5 mb
06.
07.     unset($a);
08.
09.     displayMemoryUsage('after unsetting'); // 256 kb.
```

Circular references

```
01.     class Foo {
02.         private $data = null;
03.         private $bar = null;
04.
05.         public function __construct() {
06.             $this->data = str_repeat('A', 10 * 1024 * 1024);
07.         }
08.         public function __destruct() {
09.             $this->setBar(null);
10.         }
11.         public function setBar(Bar $bar = null) {
12.             $this->bar = $bar;
13.         }
14.     }
15.
```

Circular references

```
01.     class Bar {
02.         private $data = null;
03.         private $foo = null;
04.
05.         public function __construct() {
06.             $this->data = str_repeat('A', 10 * 1024 * 1024);
07.         }
08.         public function __destruct() {
09.             $this->setFoo(null);
10.         }
11.         public function setFoo(Foo $foo = null) {
12.             $this->foo = $foo;
13.         }
14.     }
15.
```

Circular references

```
01.     {
02.         $foo = new Foo();
03.         $bar = new Bar();
04.
05.         // create circular reference
06.         $foo->setBar($bar);
07.         $bar->setFoo($foo);
08.     }
```

Circular references

```
01.     {
02.         $foo = new Foo();
03.         $bar = new Bar();
04.
05.         // create circular reference
06.         $foo->setBar($bar);
07.         $bar->setFoo($foo);
08.
09.         displayMemoryUsage('after allocating two classes'); // 20.75 mb
10.     }
```

Circular references

```
01.     {
02.         ...
03.
04.         displayMemoryUsage('after allocating two classes'); // 20.75 mb
05.
06.         unset($bar);
07.         $bar = null;
08.
09.         displayMemoryUsage('after deallocating bar'); // 20.75 mb
10.     }
```

Circular references

```
01.     {
02.         ...
03.
04.         displayMemoryUsage('after allocating two classes'); // 20.75 mb
05.
06.         unset($bar);
07.         $bar = null;
08.
09.         displayMemoryUsage('after deallocating bar'); // 20.75 mb
10.
11.         unset($foo);
12.         $foo = null;
13.         displayMemoryUsage('after deallocating foo'); // 20.75 mb
14.     }
```

Circular references

```
01.     {
02.         ...
03.
04.         displayMemoryUsage('after allocating two classes'); // 20.75 mb
05.
06.         unset($bar);
07.         $bar = null;
08.
09.         displayMemoryUsage('after deallocating bar'); // 20.75 mb
10.
11.         unset($foo);
12.         $foo = null;
13.         displayMemoryUsage('after deallocating foo'); // 20.75 mb
14.     }
15.     displayMemoryUsage('after scope exit'); // 20.75 mb
16.
```

Circular references

```
01.     {
02.         ...
03.
04.         displayMemoryUsage('after allocating two classes'); // 20.75 mb
05.
06.         $bar->setFoo(null); // <<< fixes the problem
07.         unset($bar);
08.         $bar = null;
09.
10.         displayMemoryUsage('after deallocating bar'); // 20.75 mb
11.
12.         $foo->setBar(null); // <<< fixes the problem
13.         unset($foo);
14.         $foo = null;
15.         displayMemoryUsage('after deallocating foo'); // 20.75 mb
16.     }
17.     displayMemoryUsage('after scope exit'); // 256 kb <<< now destructors were called
18.
```

Circular references

```
01.     {
02.         $foo = new Foo();
03.         $bar = new Bar();
04.
05.         $foo->setBar($bar);
06.         $bar->setFoo($foo);
07.
08.         unset($bar);
09.         $bar = null;
10.
11.         unset($foo);
12.         $foo = null;
13.     }
14.     displayMemoryUsage('after scope exit'); // 20.75 mb
```

Circular references

```
01.     {
02.         $foo = new Foo();
03.         $bar = new Bar();
04.
05.         $foo->setBar($bar);
06.         ...
07.     }
08.     displayMemoryUsage('after scope exit'); // 20.75 mb
09.
10.     gc_enable();
11.     if (!gc_enabled()) {
12.        throw new RuntimeException;
13.     }
14.     gc_collect_cycles();
15.
16.     displayMemoryUsage('after garbage collector cleanup call'); // 256 kb
```

Circular references

```
01.     {
02.         $foo = new Foo();
03.         $bar = new Bar();
04.
05.         $foo->setBar($bar);
06.         ...
07.     }
08.     displayMemoryUsage('after scope exit'); // 20.75 mb
09.
10.     gc_enable();
11.     if (!gc_enabled()) {
12.        throw new RuntimeException;
13.     }
14.     gc_collect_cycles();
15.
16.     displayMemoryUsage('after garbage collector cleanup call'); // 256 kb
17.
18.     print phpversion(); // 5.5.3-1ubuntu2.1
```

Circular references

```
01.         public function setFoo(Foo $foo = null) {
02.             $this->foo = $foo;
03.         }
04.
05.         public function setBar(Bar $bar = null) {
06.             $this->bar = $bar;
07.         }
```

Circular references

```
01.         public function setFoo(Foo $foo = null) {
02.             $this->foo = $foo ? clone($foo) : null;
03.         }
04.
05.         public function setBar(Bar $bar = null) {
06.             $this->bar = $bar ? clone($bar) : null;
07.         }
```

Conclusie (over dit boek)

Het idee van de auteur had wellicht beter uit de verf gekomen als een blogpost of paper.

Conclusie (over dit boek)

Het idee van de auteur had wellicht beter uit de verf gekomen als een blogpost of paper.

- Software Architecture maar ook UML draait om verschillende views.

Conclusie (over dit boek)

Het idee van de auteur had wellicht beter uit de verf gekomen als een blogpost of paper.

- Software Architecture maar ook UML draait om verschillende views.
- Daarom: lekker UML gebruiken. i.p.v. IDAR graphs

Conclusie (over dit boek)

Het idee van de auteur had wellicht beter uit de verf gekomen als een blogpost of paper.

- Software Architecture maar ook UML draait om verschillende views.
- Daarom: lekker UML gebruiken. i.p.v. IDAR graphs
- Hierarchie projecteren op objecten vind ik wel een zinvol "principe".

Conclusie (over dit boek)

Het idee van de auteur had wellicht beter uit de verf gekomen als een blogpost of paper.

- Software Architecture maar ook UML draait om verschillende views.
- Daarom: lekker UML gebruiken. i.p.v. IDAR graphs
- Hierarchie projecteren op objecten vind ik wel een zinvol "principe".
- Event-based aspect van IDAR method niet super zinvol binnen webdev.

Conclusie (over dit boek)

Het idee van de auteur had wellicht beter uit de verf gekomen als een blogpost of paper.

- Software Architecture maar ook UML draait om verschillende views.
- Daarom: lekker UML gebruiken. i.p.v. IDAR graphs
- Hierarchie projecteren op objecten vind ik wel een zinvol "principe".
- Event-based aspect van IDAR method niet super zinvol binnen webdev.
- Wel met messaging aan de gang? ==> frameworks zoals ZeroMQ

Wat moet je hier nu eigenlijk mee?

Wat moet je hier nu eigenlijk mee?

Concreet maken?

Wat moet je hier nu eigenlijk mee?

Concreet maken?

- Tool support: cyclic dependencies checken

Wat moet je hier nu eigenlijk mee?

Concreet maken?

- Tool support: cyclic dependencies checken
- Annotaties wie "commands" wie?

Wat moet je hier nu eigenlijk mee?

Concreet maken?

- Tool support: cyclic dependencies checken
- Annotaties wie "commands" wie?
- Dependency injection container op configuratie niveau?

Wat moet je hier nu eigenlijk mee?

Concreet maken?

- Tool support: cyclic dependencies checken
- Annotaties wie "commands" wie?
- Dependency injection container op configuratie niveau?
- Regels voor "doorgeven" van referenties naar instances?

Wat moet je hier nu eigenlijk mee?

Concreet maken?

- Tool support: cyclic dependencies checken
- Annotaties wie "commands" wie?
- Dependency injection container op configuratie niveau?
- Regels voor "doorgeven" van referenties naar instances?

Maar voor nu kom ik niet verder dan:

Wat moet je hier nu eigenlijk mee?

Concreet maken?

- Tool support: cyclic dependencies checken
- Annotaties wie "commands" wie?
- Dependency injection container op configuratie niveau?
- Regels voor "doorgeven" van referenties naar instances?

Maar voor nu kom ik niet verder dan:

- Het idee ter harte nemen om hierarchie te projecteren over je objecten.

References

- http://www.idarmethod.com
- https://www.fortran.com/come_from.html
- http://www.isr.uci.edu/architecture/c2StyleRules.html
- https://github.com/Dobiasd/articles/blob/master/from_goto_to_std-transform.md
- http://zguide.zeromq.org/page:all

Betere tips

Betere tips

Een model geinspireerd door functioneel programmeren lijkt me een beter "alternatief":

Betere tips

Een model geinspireerd door functioneel programmeren lijkt me een beter "alternatief":

- Classes als "types" (goed gebruik van constructors, destructors en exceptions)

Betere tips

Een model geinspireerd door functioneel programmeren lijkt me een beter "alternatief":

- Classes als "types" (goed gebruik van constructors, destructors en exceptions) ==> Design by Contract zit dan verwerkt in je types.

Betere tips

Een model geinspireerd door functioneel programmeren lijkt me een beter "alternatief":

- Classes als "types" (goed gebruik van constructors, destructors en exceptions) ==> Design by Contract zit dan verwerkt in je types.
- Command / Query separation (CQS)
- Dependency injection

Betere tips

Een model geinspireerd door functioneel programmeren lijkt me een beter "alternatief":

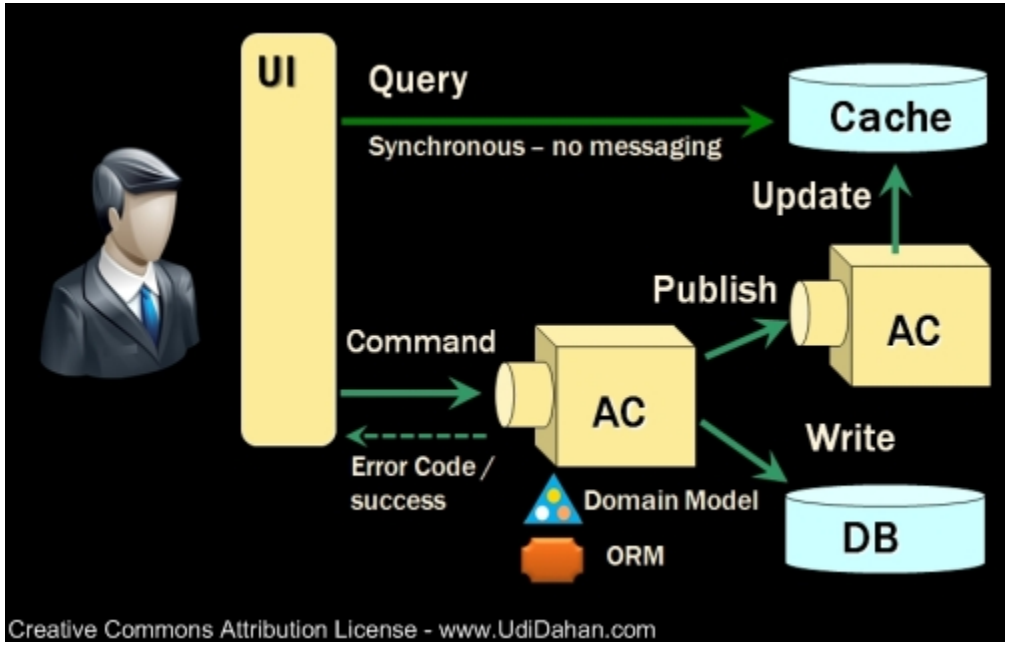
- Classes als "types" (goed gebruik van constructors, destructors en exceptions) ==> Design by Contract zit dan verwerkt in je types.
- Command / Query separation (CQS) ==> Command Query Responsibility Segregation (CQRS).
- Dependency injection

Betere tips

Een model geinspireerd door functioneel programmeren lijkt me een beter "alternatief":

- Classes als "types" (goed gebruik van constructors, destructors en exceptions) ==> Design by Contract zit dan verwerkt in je types.
- Command / Query separation (CQS) ==> Command Query Responsibility Segregation (CQRS) ==> Domain Driven Design (DDD).
- Dependency injection

CQRS


UI
Query
Synchronous – no messaging
Cache
Update
Publish
AC
Command
AC
Write
Error Code /
success
Domain Model
ORM
DB
Creative Commons Attribution License - www.UdiDahan.com