Design Approaches

Simple, RDD, Solid, DBC & KISS

Diego Pacheco

About me...

- ☐ Cat's Father
- ☐ Principal Software Architect
- ☐ Agile Coach
- ☐ SOA/Microservices Expert
- ☐ DevOps Practitioner
- ☐ Speaker
- ☐ Author

diegopacheco

@diego_pacheco

http://diego-pacheco.blogspot.com.br/

Diego Pacheco

Building Applications with Scala

Write modern, scalable, and reactive applications with the power of Scala

Packt>

Diego Pacheco

Building Effective Microservices

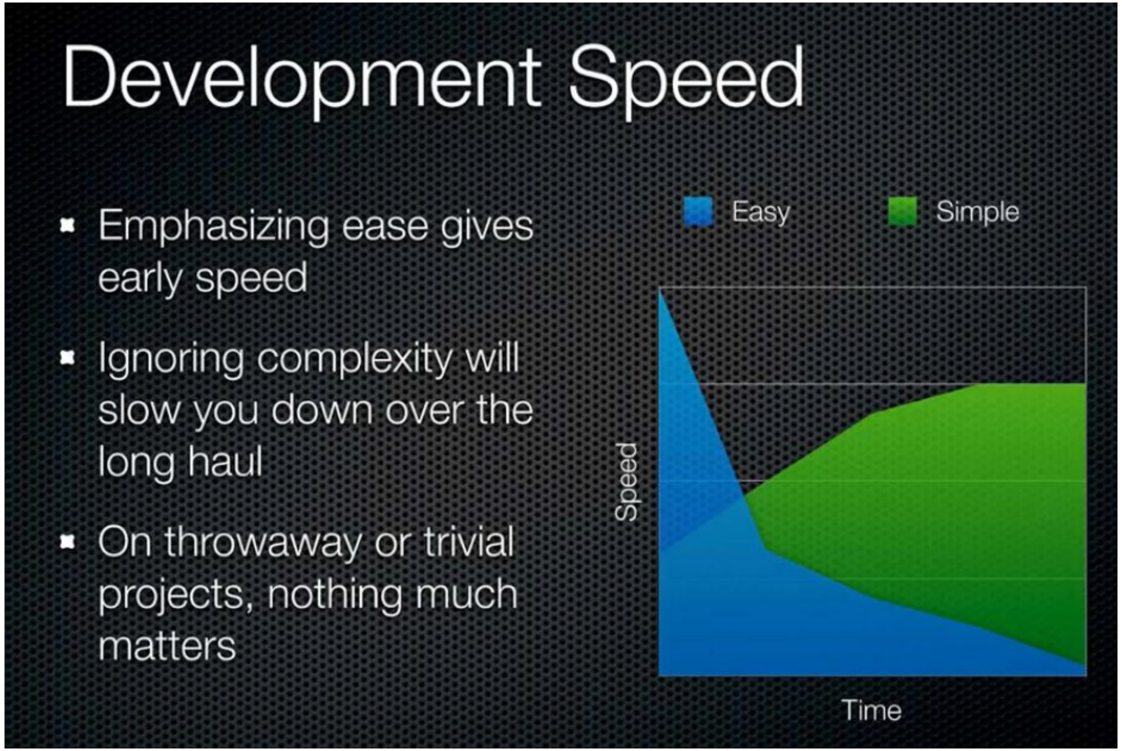
Explore microservices and their implementation hands-on

Packt>

https://diegopacheco.github.io/

Simplicity is the ultimate sophistication.
Leonardo da Vinci
BrainyQuote

Simple: Agility is not about process - is about doing it!

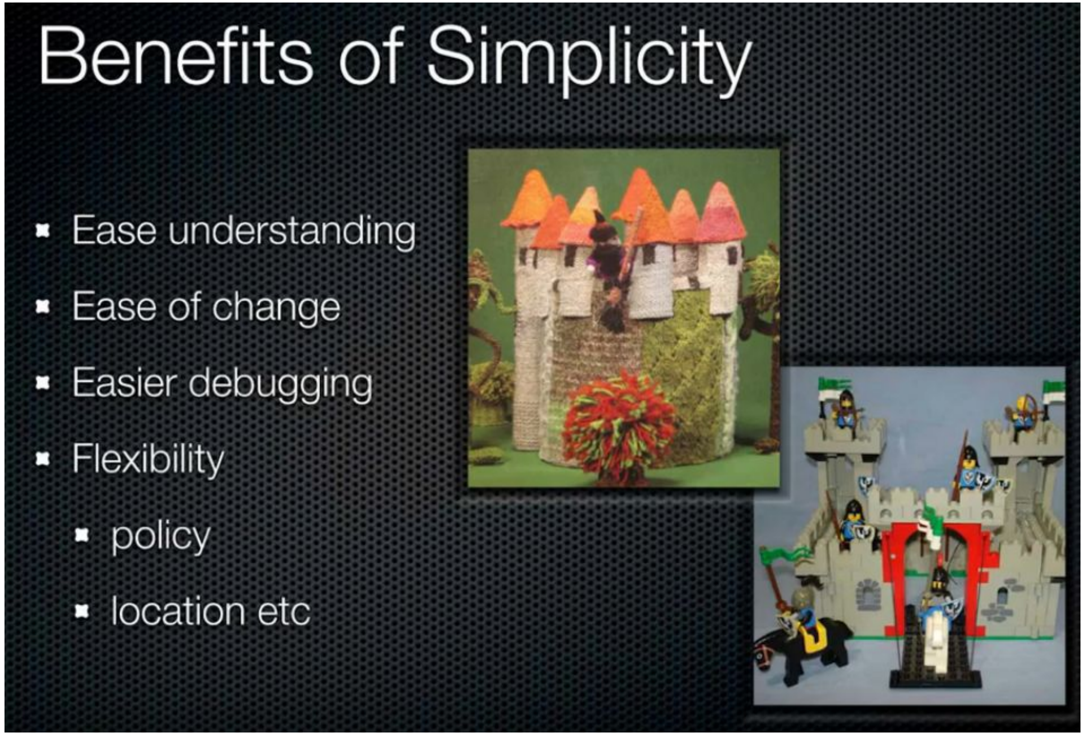


Development Speed

- Emphasizing ease gives early speed
- Ignoring complexity will slow you down over the long haul
- On throwaway or trivial projects, nothing much matters

https://www.youtube.com/watch?v=oytL881p-nQ

Moving Forward: Tests don't make the design better.

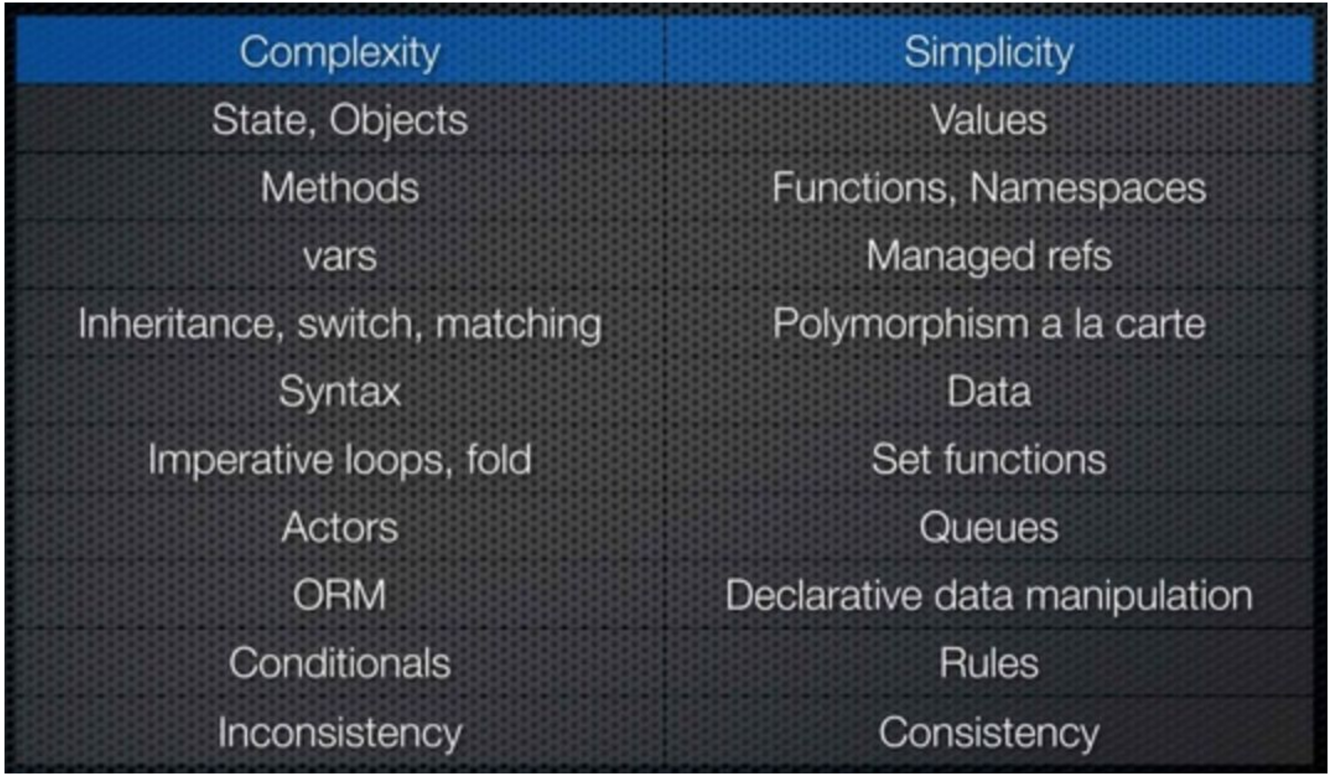


Benefits of Simplicity

- Ease understanding
- Ease of change
- Easier debugging
- Flexibility
  - policy
  - location etc

https://www.youtube.com/watch?v=oytL881p-nQ

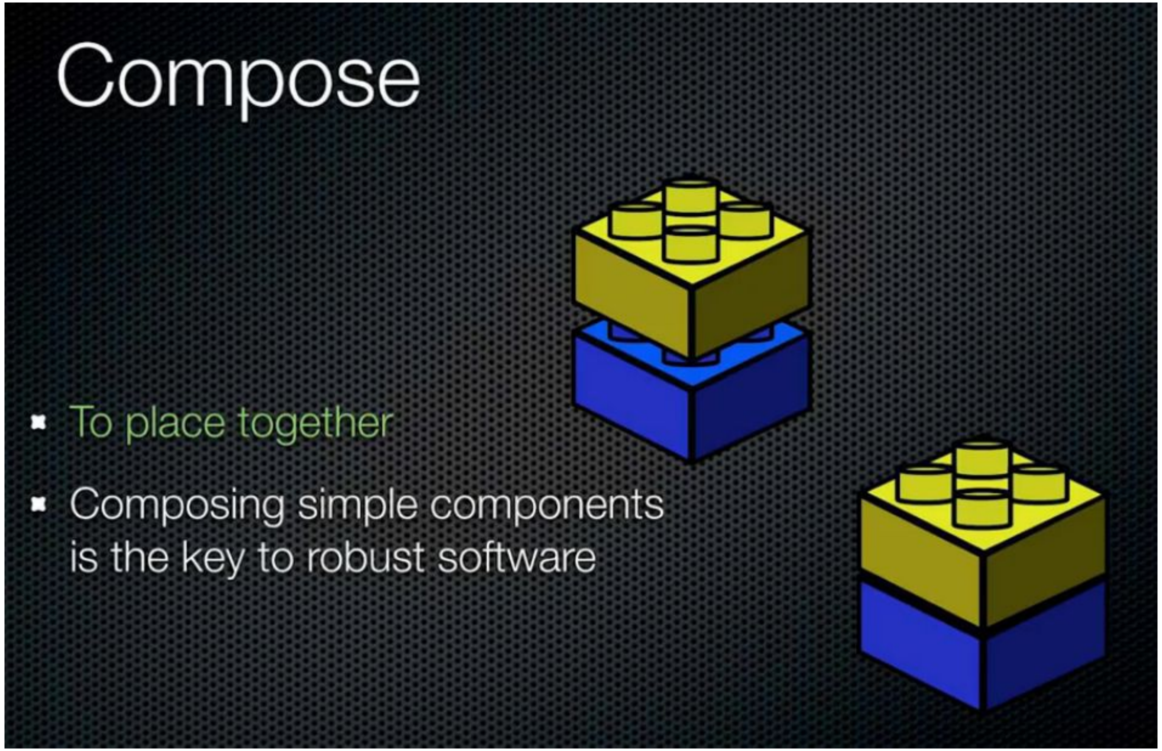
Simple



Complexity	Simplicity
State, Objects	Values
Methods	Functions, Namespaces
vars	Managed refs
Inheritance, switch, matching	Polymorphism a la carte
Syntax	Data
Imperative loops, fold	Set functions
Actors	Queues
ORM	Declarative data manipulation
Conditionals	Rules
Inconsistency	Consistency

https://www.youtube.com/watch?v=oytL881p-nQ

Simple: Nobody thinks about it anymore...

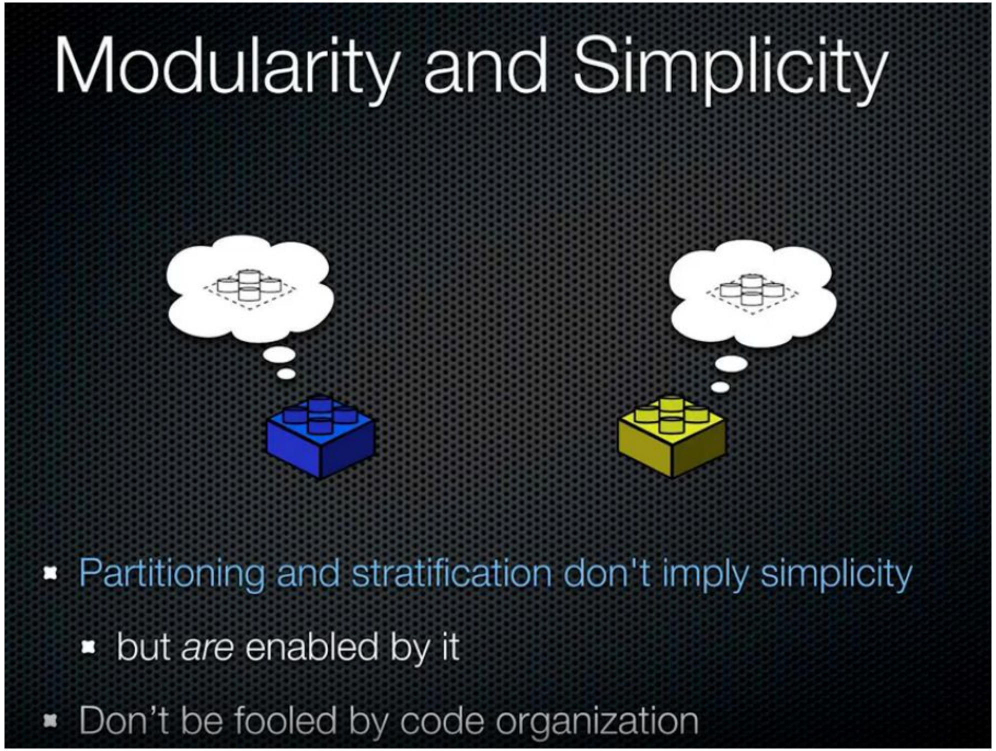


Compose

- To place together
- Composing simple components is the key to robust software

https://www.youtube.com/watch?v=oytL881p-nQ

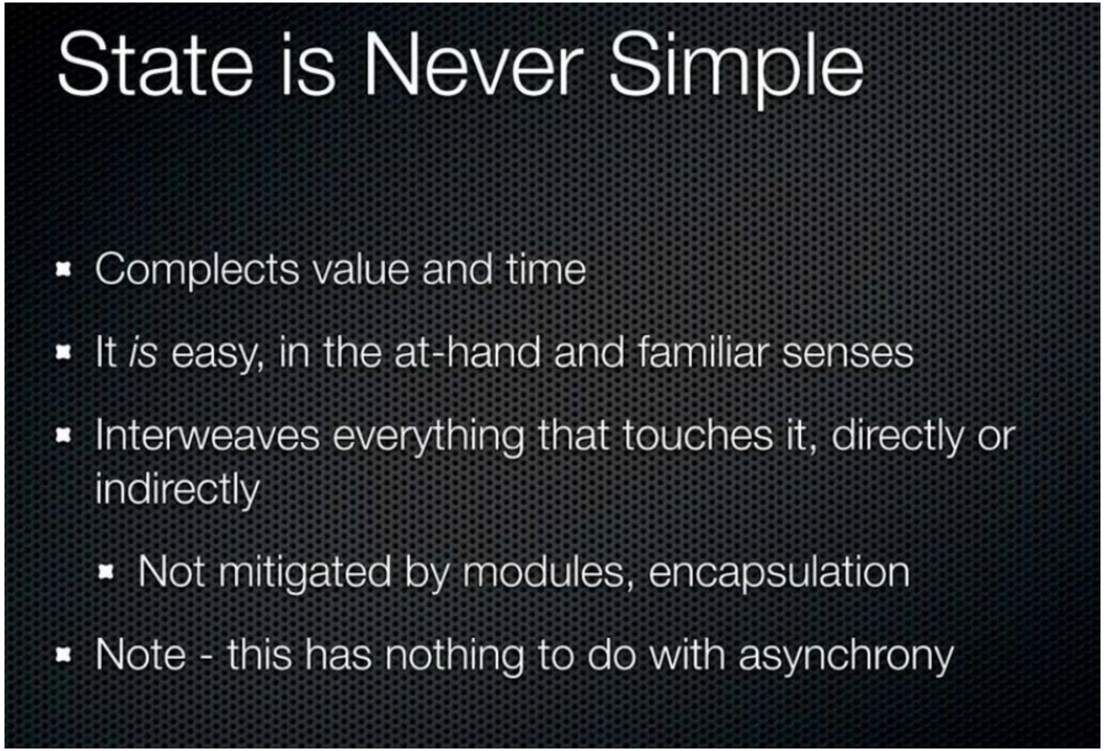
Simple: IF is simple is modular, but modular might not be simple


Modularity and Simplicity

- Partitioning and stratification don't imply simplicity
  - but *are* enabled by it
- Don't be fooled by code organization

https://www.youtube.com/watch?v=oytL881p-nQ

Simple: State is Never Simple!

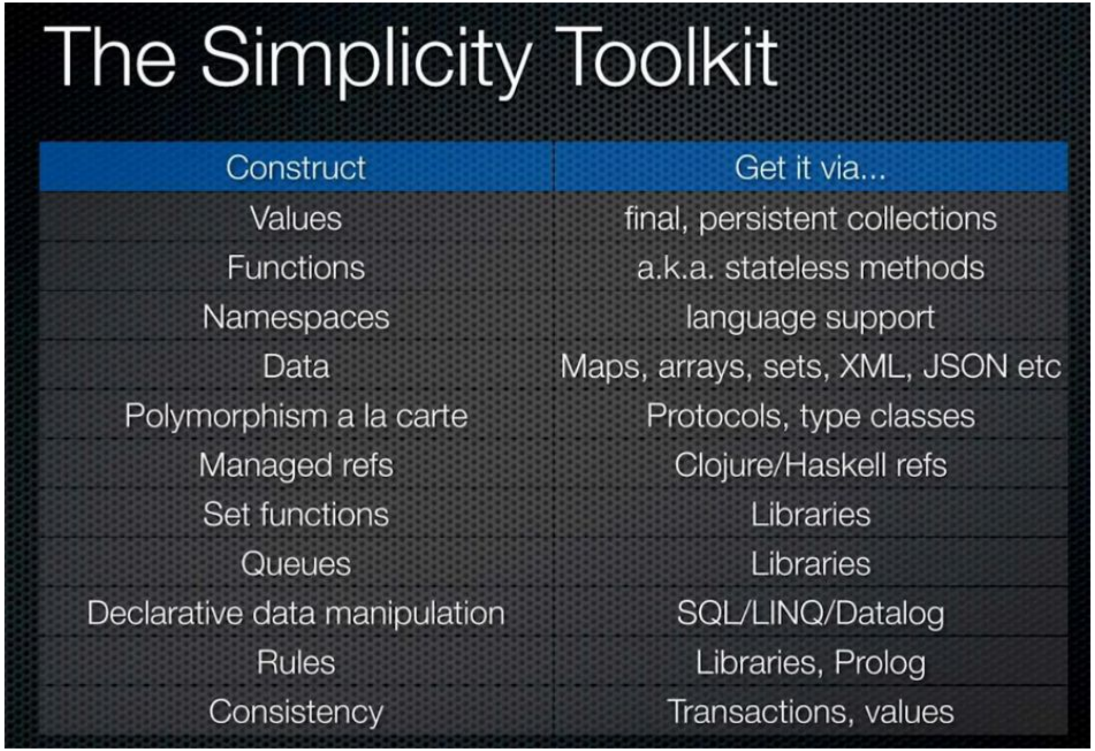


State is Never Simple

- Complects value and time
- It *is* easy, in the at-hand and familiar senses
- Interweaves everything that touches it, directly or indirectly
  - Not mitigated by modules, encapsulation
- Note - this has nothing to do with asynchrony

https://www.youtube.com/watch?v=oytL881p-nQ

Simple



The Simplicity Toolkit

Construct	Get it via...
Values	final, persistent collections
Functions	a.k.a. stateless methods
Namespaces	language support
Data	Maps, arrays, sets, XML, JSON etc
Polymorphism a la carte	Protocols, type classes
Managed refs	Clojure/Haskell refs
Set functions	Libraries
Queues	Libraries
Declarative data manipulation	SQL/LINQ/Datalog
Rules	Libraries, Prolog
Consistency	Transactions, values

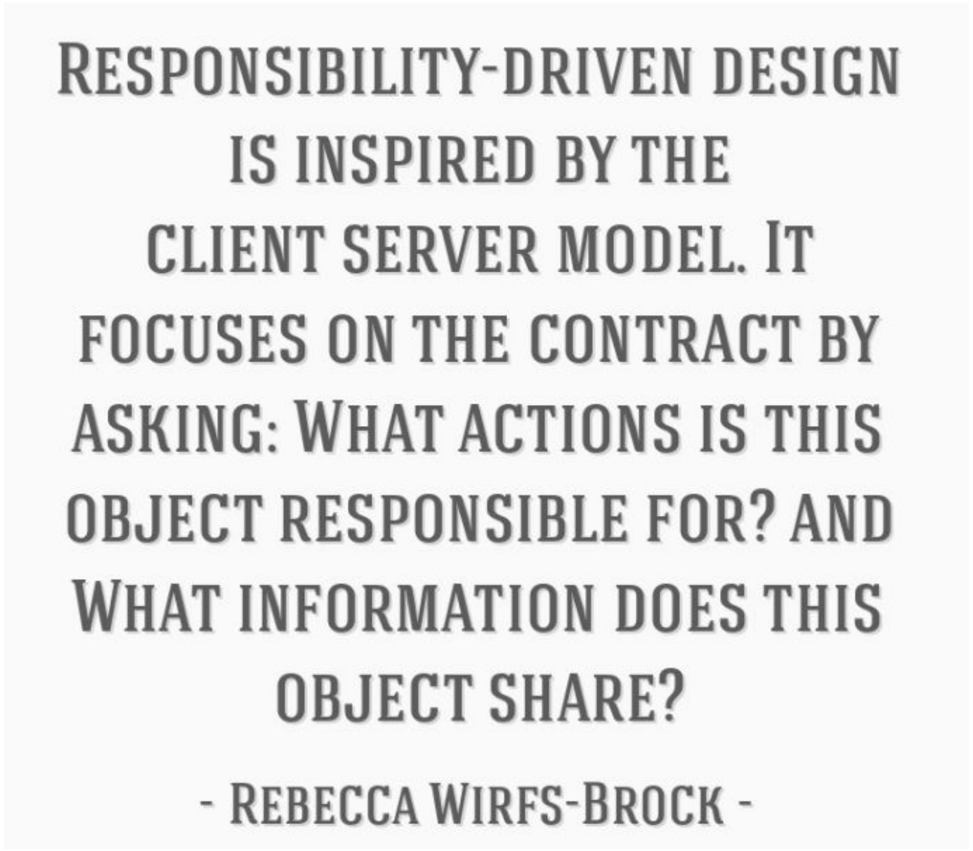
https://www.youtube.com/watch?v=oytL881p-nQ

KISS

KISS

keep.it.simple.stupid.

RDD

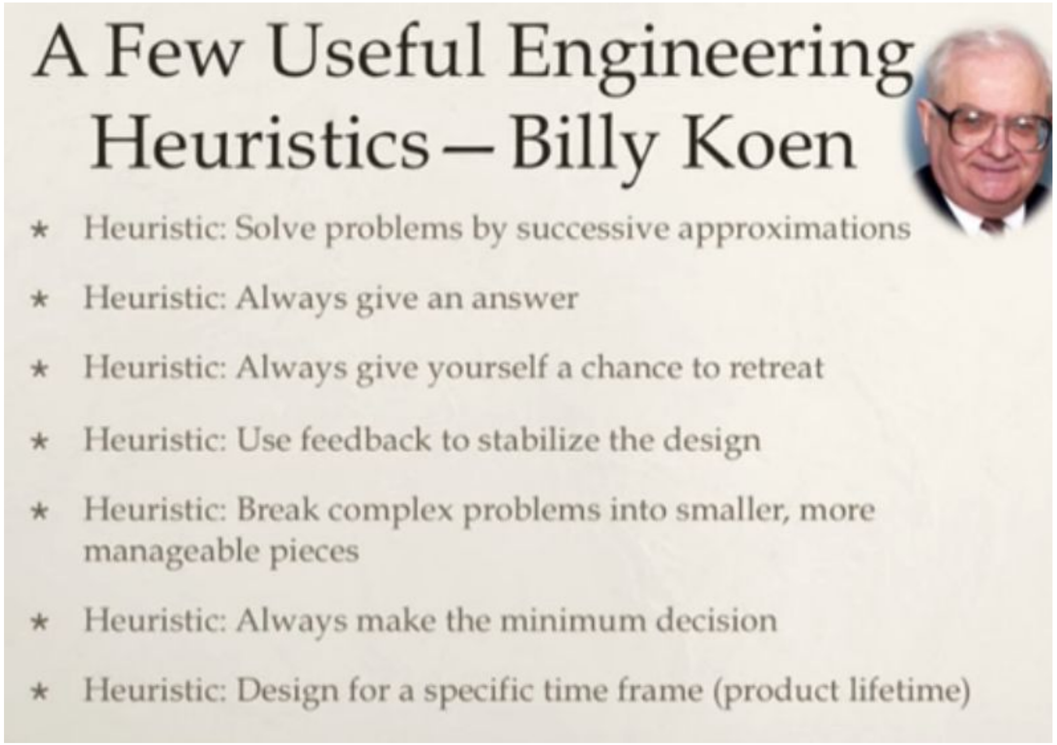


RESPONSIBILITY-DRIVEN DESIGN IS INSPIRED BY THE CLIENT SERVER MODEL. IT FOCUSES ON THE CONTRACT BY ASKING: WHAT ACTIONS IS THIS OBJECT RESPONSIBLE FOR? AND WHAT INFORMATION DOES THIS OBJECT SHARE?

- REBECCA WIRFS-BROCK -

https://www.youtube.com/watch?v=NZ5mI6-tNUc

RDD

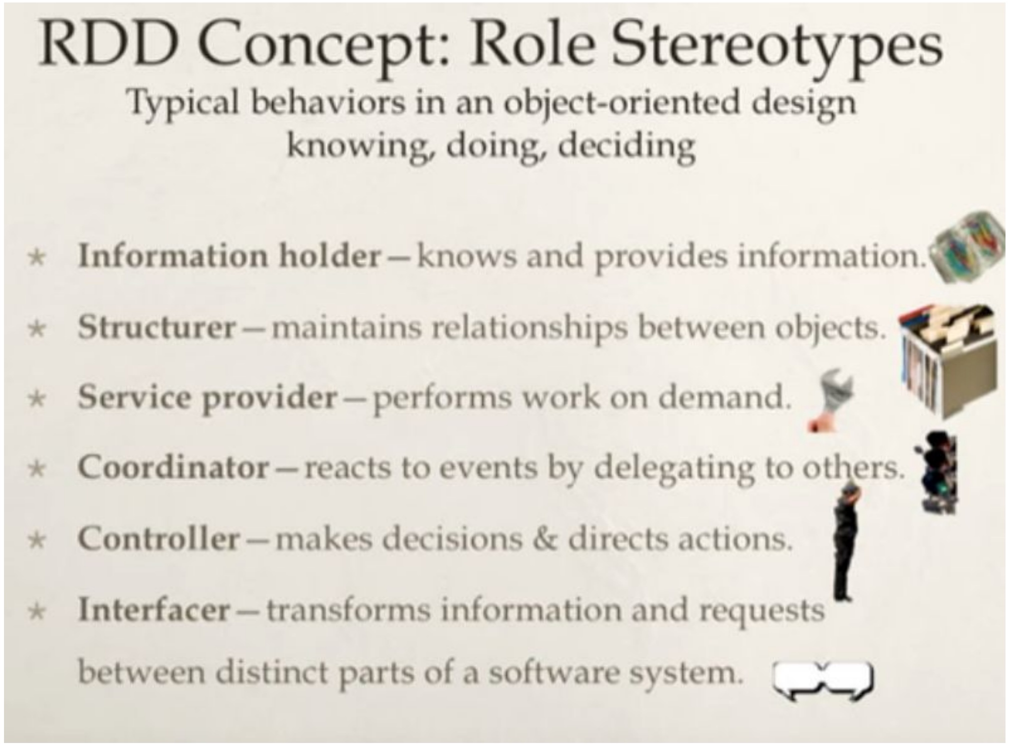


A Few Useful Engineering Heuristics – Billy Koen

- ★ Heuristic: Solve problems by successive approximations
- ★ Heuristic: Always give an answer
- ★ Heuristic: Always give yourself a chance to retreat
- ★ Heuristic: Use feedback to stabilize the design
- ★ Heuristic: Break complex problems into smaller, more manageable pieces
- ★ Heuristic: Always make the minimum decision
- ★ Heuristic: Design for a specific time frame (product lifetime)

https://www.youtube.com/watch?v=NZ5mI6-tNUc

RDD



RDD Concept: Role Stereotypes

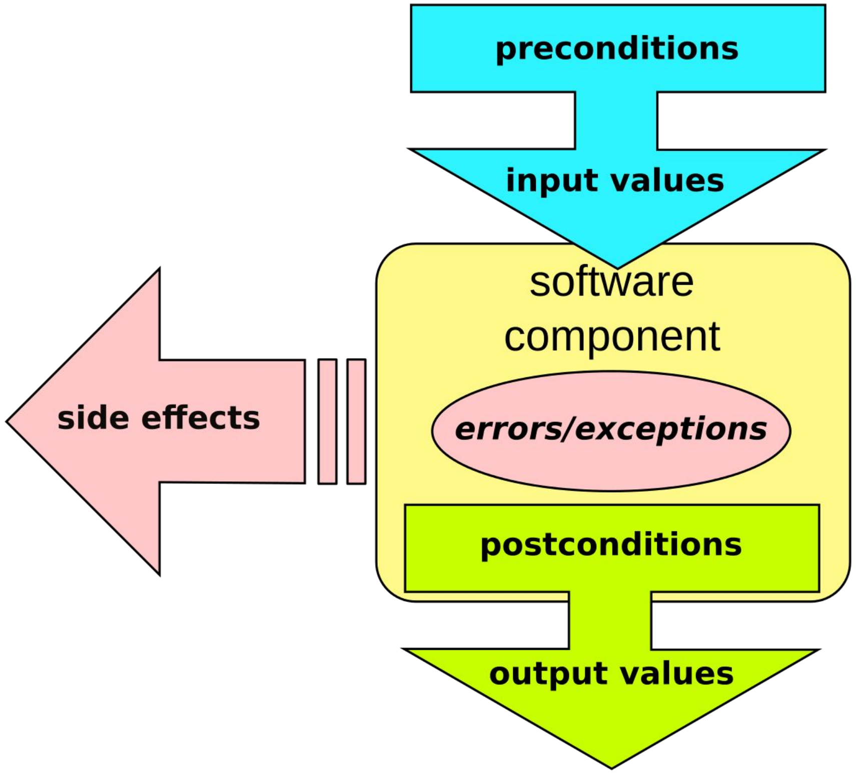
Typical behaviors in an object-oriented design
knowing, doing, deciding

- ★ **Information holder** – knows and provides information.
- ★ **Structurer** – maintains relationships between objects.
- ★ **Service provider** – performs work on demand.
- ★ **Coordinator** – reacts to events by delegating to others.
- ★ **Controller** – makes decisions & directs actions.
- ★ **Interfacer** – transforms information and requests between distinct parts of a software system.

https://www.youtube.com/watch?v=NZ5mI6-tNUc

DBC

DESIGN BY CONTRACT
DESIGN BY CONTRACT LEADS TO BEHAVIOR PRESERVATION. BEHAVIOR PRESERVATION LEADS TO LESS CRAPY SOFTWARE.
DIY.DESPAIR.COM



http://hibernate.org/validator/

DBC

```
@Requires("x >= 0")
@Ensures("result >= 0")
static double sqrt(double x);
```

vs

```
static void sqrt_Requires(double x) {
  assert x >= 0;
}
static void sqrt_Ensures(double result) {
  assert result >= 0;
}
static double sqrt(double x);
```

https://github.com/nhatminhle/cofoja

SOLID

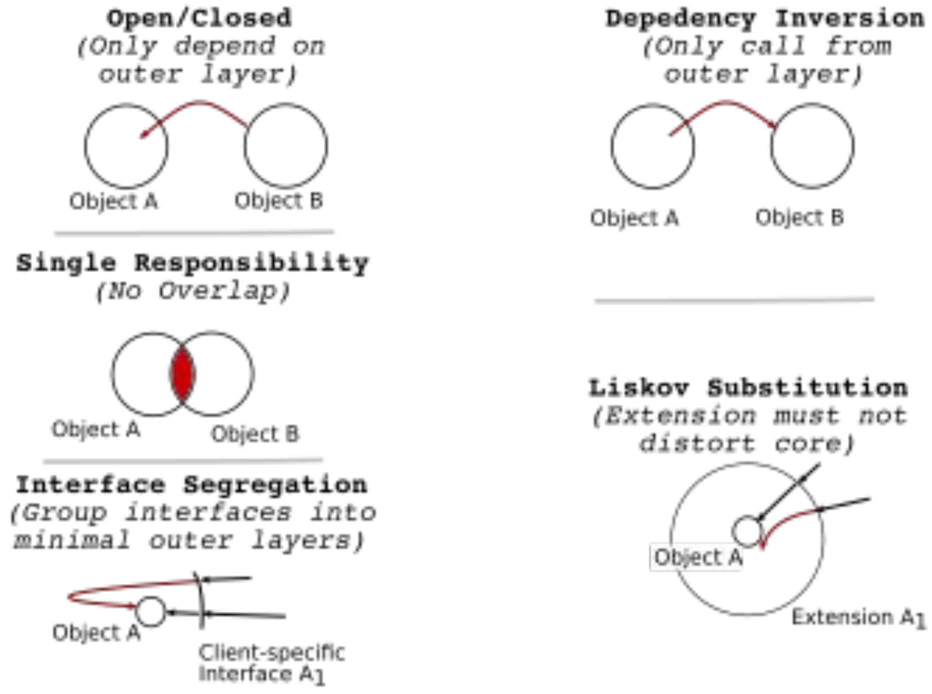
S **ingle Resposibility Principle**
A class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)

O **pen / Closed Principle**
A software module (it can be a class or method) should be open for extension but closed for modification.

L **iskov Substitution Principle**
Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.

I **nterface Segregation Principle**
Clients should not be forced to depend upon the interfaces that they do not use.

D **ependency Inversion Principle**
Program to an interface, not to an implementation.



OOP might be bad...

Aaargh! What is wrong with you?!! You didn't use a framework and you didn't use any design patterns! DEVIANT!!!
I ♥ Laravel
© Allan Bak

https://www.youtube.com/watch?v=QM1iUe6IofM

http://harmful.cat-v.org/software/OO_programming/why_oo_sucks

Design Approaches

Simple, RDD, Solid, DBC & KISS

DIEGO PACHECO