"Clean" Architecture

2016, by Manfred Touron (@moul)

overview

- the "clean" architecture, "Yet Another New Architecture"
- by uncle Bob
- discovered 3 months ago at OpenClassrooms with Romain Kuzniak
- recent, no real spec, no official implementation
- I don't use "clean" architecture in production
- I'm not a "clean" architecture expert

design slogans 1/2 [1]

- YAGNI (You Ain't Gonna Need It)
- KISS (Keep It Simple, Stupid)
- DRY (Don't Repeat Yourself)
- S.O.L.I.D (SRP, OCP, LS, IS, DI)
- TDD (Test Driven Development)

---

[1] more info: http://fr.slideshare.net/RomainKuzniak/design-applicatif-avec-symfony2

design slogans 2/2 [1]

- BDD (Behavior Driven Development)
- DDD (Domain Driven Design)
- ...

[1] more info: http://fr.slideshare.net/RomainKuzniak/design-applicatif-avec-symfony2

design types [1]

- MVC
- N3 Architectures
- Domain Driven Design
- Clean Architecture

---

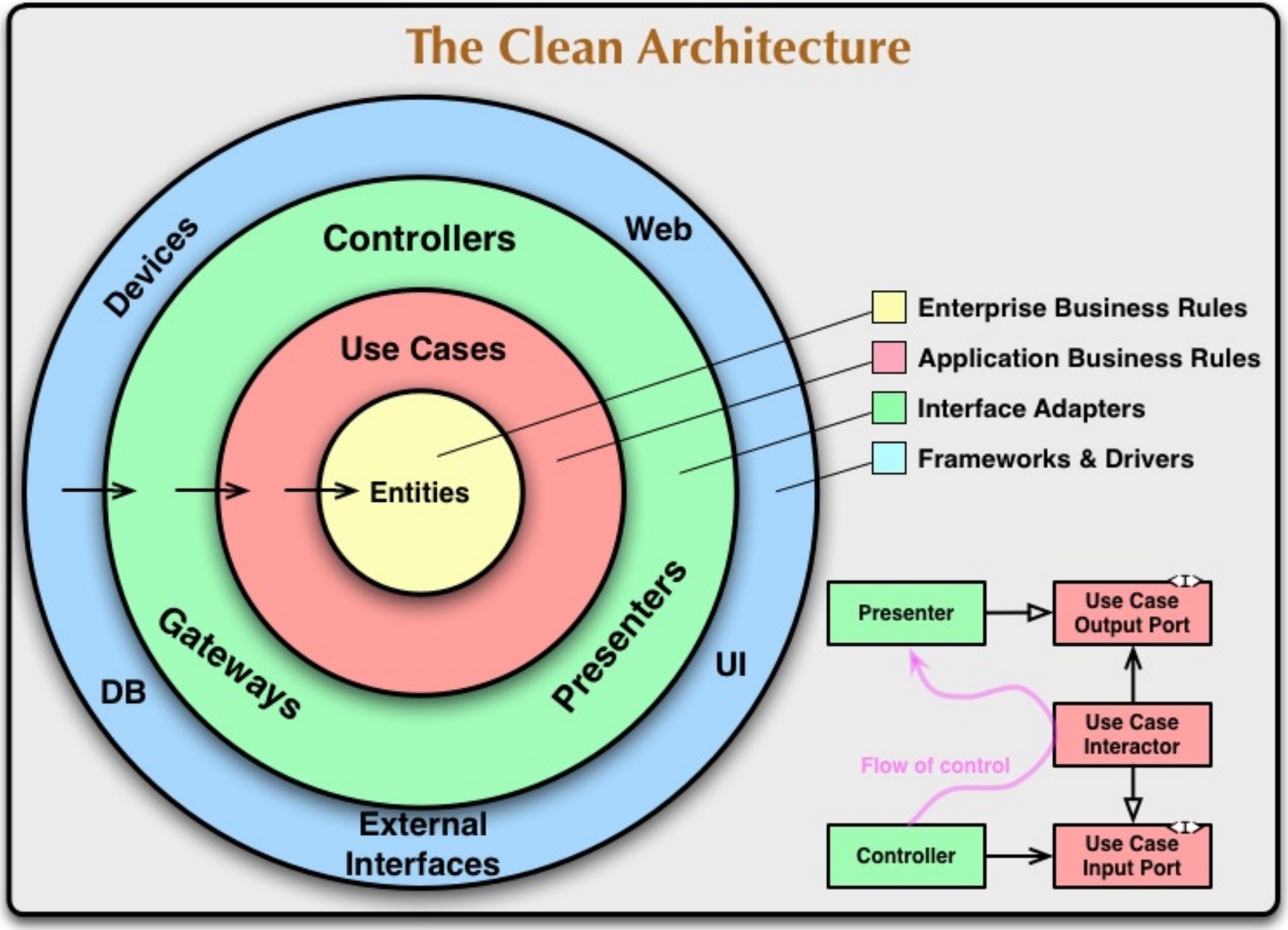
[1] more info: http://fr.slideshare.net/RomainKuzniak/design-applicatif-avec-symfony2

the "clean" architecture [2]

- Not a revolution, a mix of multiple existing principles
- The other designs are not "dirty" architectures
- Recent examples: Hexagonal Architecture, Onion Architecture, Screaming Architecture, DCI, BCE
- Dependency injection at the buildtime or at leat at the runtime init

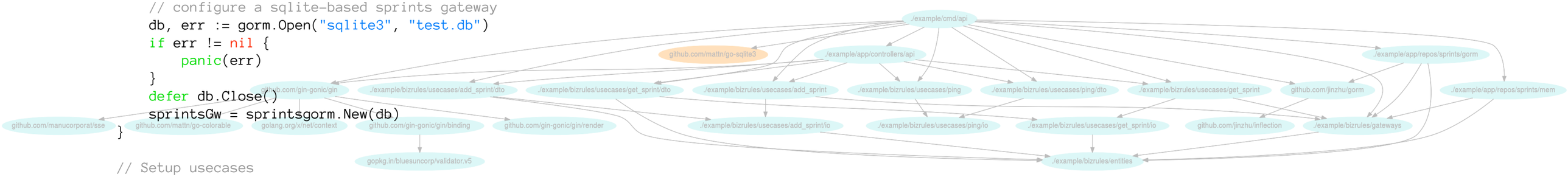
---

[2] https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port

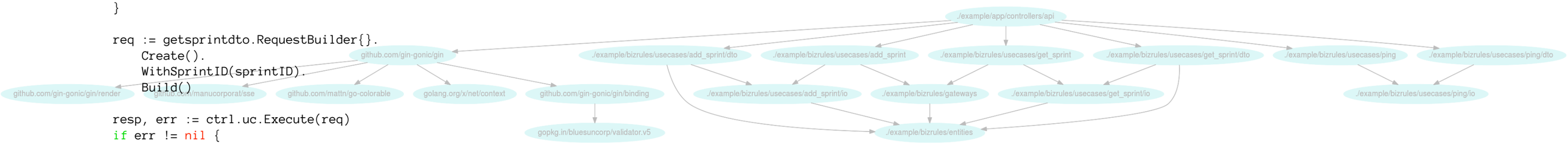
./cmd/api

```
func main() {
    // Setup gateways
    var sprintsGw gateways.Sprints
    if len(os.Args) > 1 && os.Args[1] == "--mem" {
        // configure a memory-based sprints gateway
        sprintsGw = sprintsmem.New()
    } else {
        // configure a sqlite-based sprints gateway
        db, err := gorm.Open("sqlite3", "test.db")
        if err != nil {
            panic(err)
        }
        defer db.Close()
        sprintsGw = sprintsgorm.New(db)
    }

    // Setup usecases
    getSprint := getsprint.New(sprintsGw, getsprintdto.ResponseAssembler{})
    addSprint := addsprint.New(sprintsGw, addsprintdto.ResponseAssembler{})
    ping := ping.New(pingdto.ResponseAssembler{})
    //closeSprint := closesprint.New(sprintsGw, closesprintdto.ResponseBuilder{})

    // Setup API
    gin := gin.Default()
    gin.GET("/sprints/:sprint-id", apicontrollers.NewGetSprint(&getSprint).Execute)
    gin.POST("/sprints", apicontrollers.NewAddSprint(&addSprint).Execute)
    gin.GET("/ping", apicontrollers.NewPing(&ping).Execute)
    //gin.DELETE("/sprints/:sprint-id", apicontrollers.NewCloseSprint(&closeSprint).Execute)

    // Start
    gin.Run()
    }
```



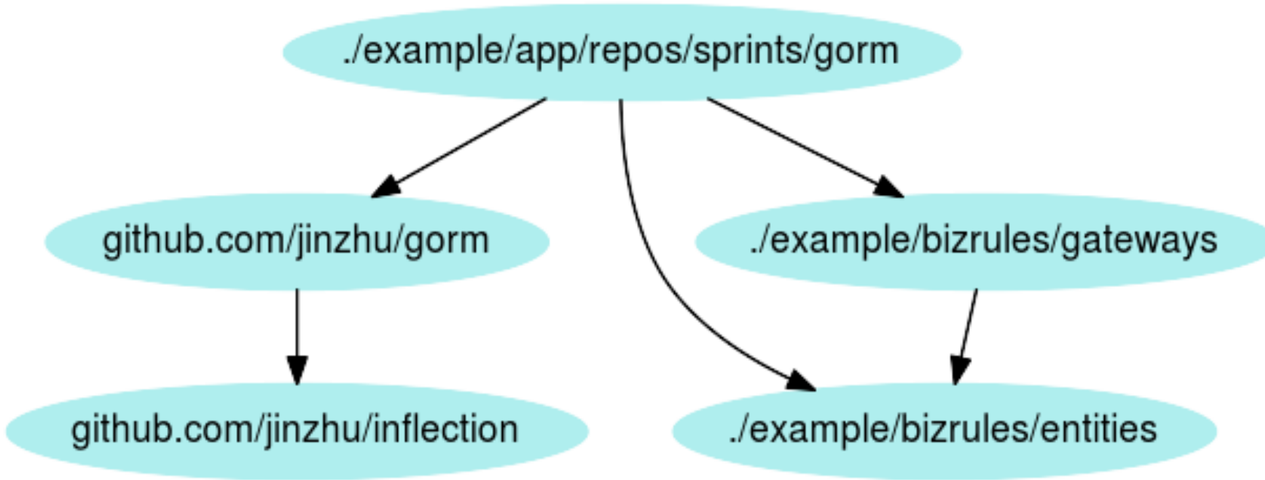
./app/controllers/api

```go
type GetSprint struct {
    uc *getsprint.UseCase
}

func (ctrl *GetSprint) Execute(ctx *gin.Context) {
    sprintID, err := strconv.Atoi(ctx.Param("sprint-id"))
    if err != nil {
        ctx.JSON(http.StatusNotFound, gin.H{"error": "Invalid 'sprint-id'"})
        return
    }

    req := getsprintdto.RequestBuilder{}.
        Create().
        WithSprintID(sprintID).
        Build()

    resp, err := ctrl.uc.Execute(req)
    if err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{"error": fmt.Sprintf("%v", err)})
        return
    }

    ctx.JSON(http.StatusOK, gin.H{"result": GetSprintResponse{
        CreatedAt:         resp.GetCreatedAt(),
        EffectiveClosedAt: resp.GetEffectiveClosedAt(),
        ExpectedClosedAt:  resp.GetExpectedClosedAt(),
        Status:            resp.GetStatus(),
    }})
}

type GetSprintResponse struct {
    CreatedAt         time.Time `json:"created-at"`
    EffectiveClosedAt time.Time `json:"effective-closed-at"`
    ExpectedClosedAt  time.Time `json:"expected-closed-at"`
    Status            string    `json:"status"`
}
```



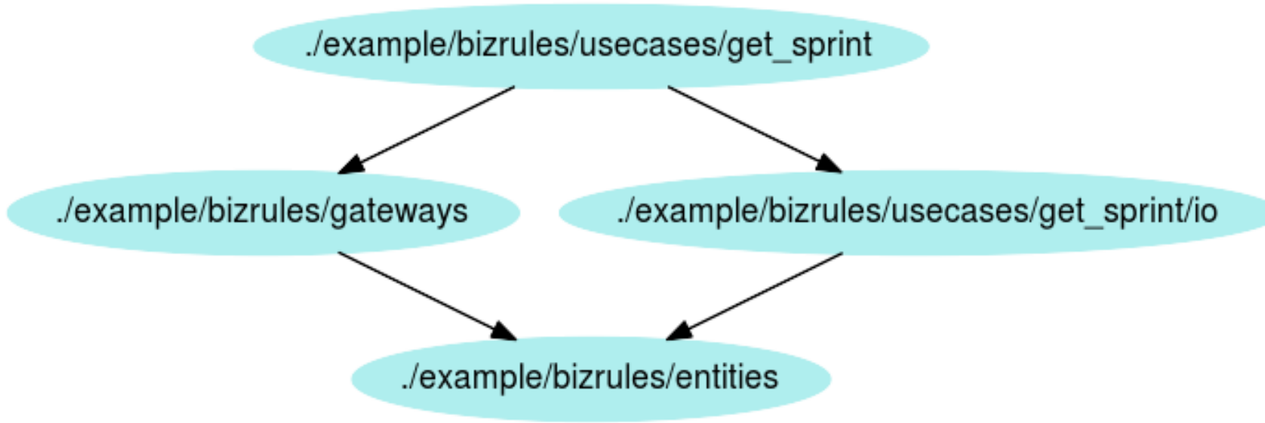
./app/repos/sprints/gorm

```go
type Repo struct { // implements gateways.Sprints
    db *gorm.DB
}

func (r Repo) Find(id int) (*entities.Sprint, error) {
    obj := sprintModel{}
    if err := r.db.First(&obj, "id = ?", id).Error; err != nil {
        return nil, err
    }

    ret := entities.NewSprint()
    ret.SetCreatedAt(obj.CreatedAt)
    ret.SetID(int(obj.ID))
    ret.SetStatus(obj.status)
    ret.SetEffectiveClosedAt(obj.effectiveClosedAt)
    ret.SetExpectedClosedAt(obj.expectedClosedAt)

    return ret, nil
}
```



./bizrules/usecases/get_sprint

```
type UseCase struct {
    gw   gateways.Sprints
    resp getsprintio.ResponseAssembler
}

func (uc *UseCase) Execute(req Request) (Response, error) {
    sprint, err := uc.gw.Find(req.GetID())
    if err != nil {
        return nil, err
    }

    return uc.resp.Write(sprint)
}
```



pros 1/2

- highly reusable
- separate business rules <-> drivers
- ease of switching to new backends
- "LTS" business rules - heritage
- unit-tests friendly
- keep "good" performances (perhaps specific with Go (no needs for reflect))

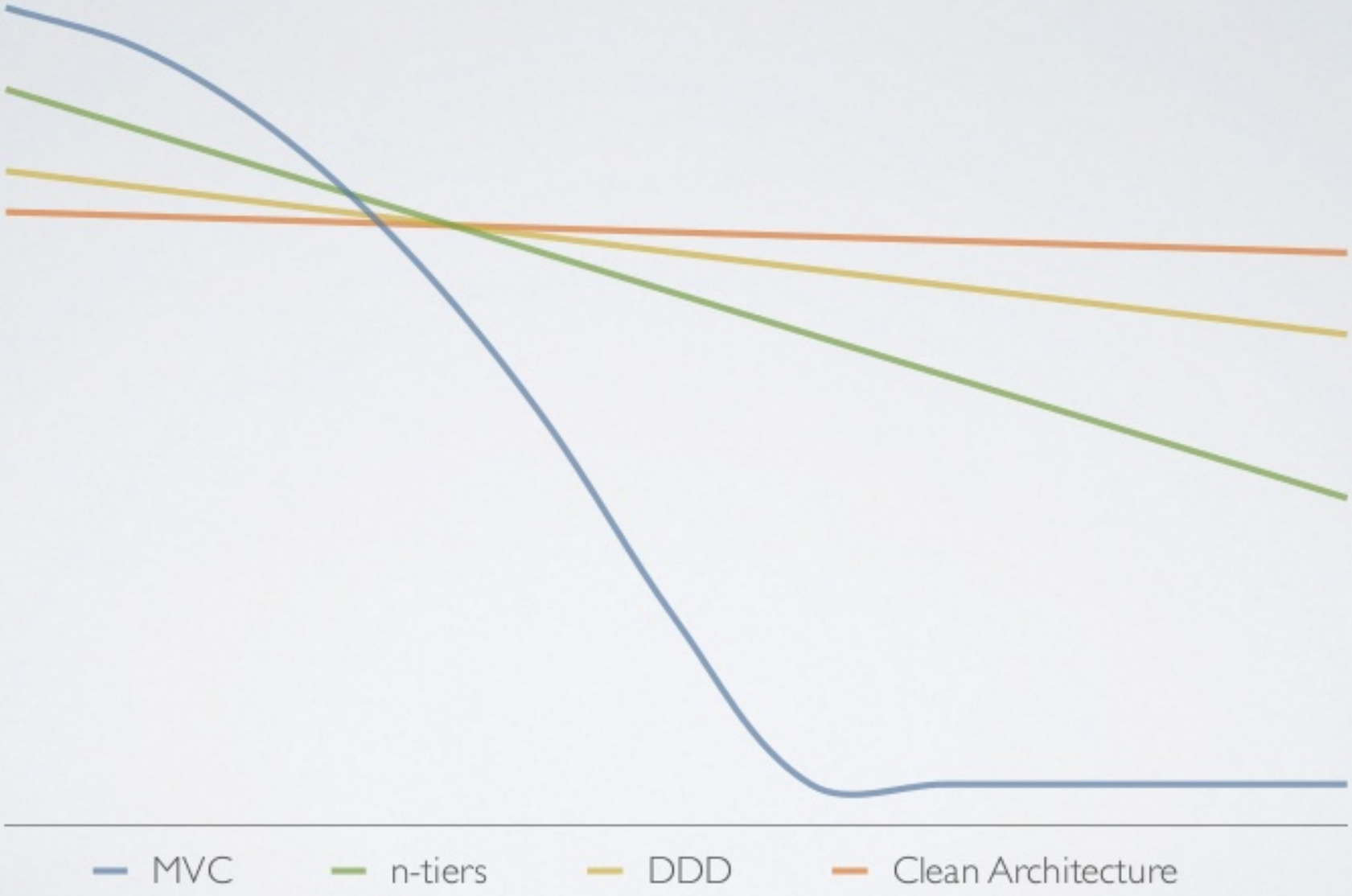
pros 2/2

- TDD friendly (Test Driver Development)
- BDD friendly (Behavior Driven Development)
- TDD + BDD drives to good designs
- ease of switching to new interfaces (or have multiple ones)
- standardize exchanges; unit-tests requests and responses
- the boundaries are clearly defined, it forces you to keep things at the right place

cons

- a looooooot of files, classes, ... (annoying for creating new entities, usecases...)
- code discovery, classes not directly linked to real objects, but to interfaces
- make some optimizations harder, i.e: transactions

Evolution de la productivité



improvements ideas

- gogenerate: less files, more readable code
- add stats on the GitHub repo (impact on performances, LOC, complexity)
- ...

conclusion

- it's a Gasoil, the learning curve (start) is long
- interesting for big projects, overkill for smaller, the center domain needs to be rich enough
- should be done completely, or not at all
- needs to be rigorous with the main and unit tests

questions ?

github.com/moul/cleanarch
@moul