Survival guide for maintaining Legacy Sytems

DataArt

Jeronimo Martinez
Software Engineer

@jeronimomtnz

jeronimo.mtnz@gmail.com

DataArt
Technology Consulting & Solution Design

21

years in operation

2600+

specialists

20

cities

3500+

projects

10

countries

$140M

revenue in 2018

What is a legacy software system?

A system that is in use, because it’s valuable for a business or it’s users…

… but it's difficult to update and improve, or even to keep it functioning correctly.

Why is it a problem?

Technical problems

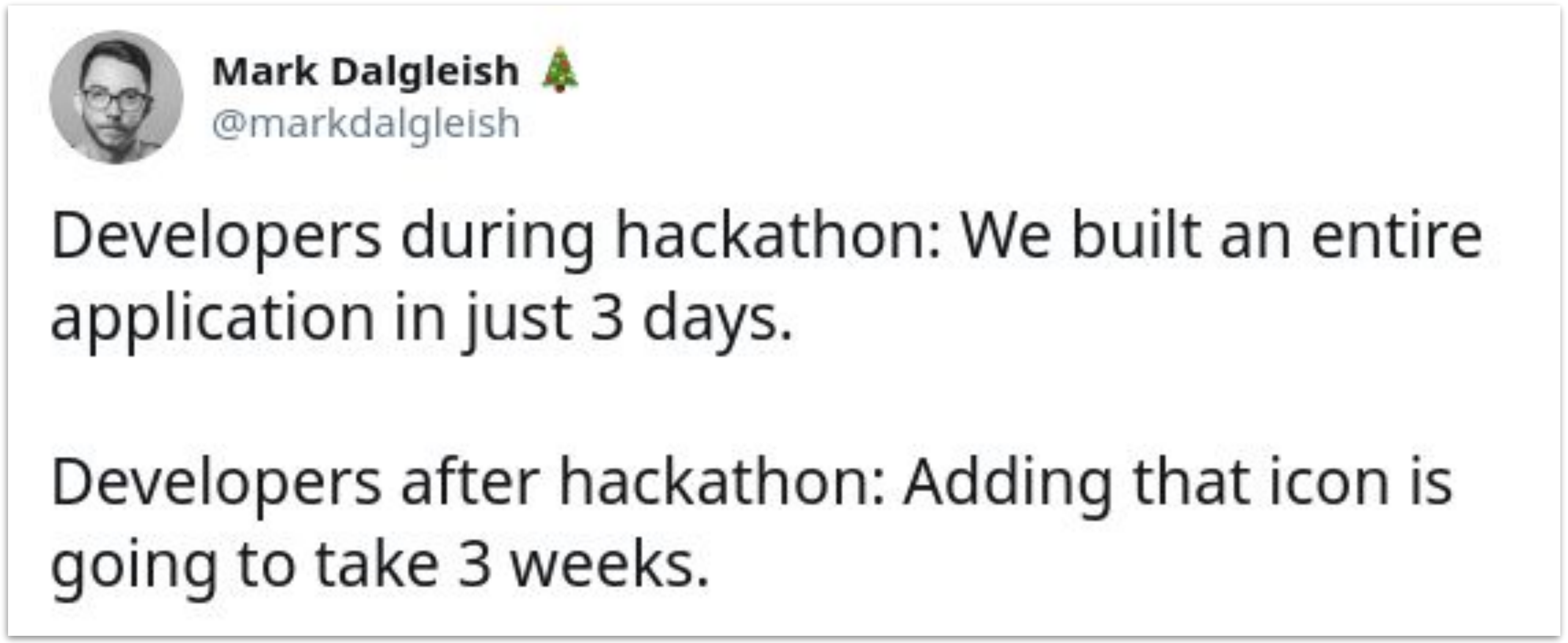
Business problems

- Can’t add new features easily
- Very difficult to fix bugs
- Can’t improve the design to prepare it for future changes
- It’s slow and using too many resources
- Developers don’t want to deal with these systems
- More time putting down fires than creating value

Why do systems become legacy?

- Scarcity of software maintenance knowledge base.
- Excessive amount of dependencies.
- Technical debt.
- Fear of making changes.
- Miscommunication between Business and Technical teams.

Fear of making changes



Mark Dalgleish 🎄
@markdalgleish

Developers during hackathon: We built an entire application in just 3 days.

Developers after hackathon: Adding that icon is going to take 3 weeks.

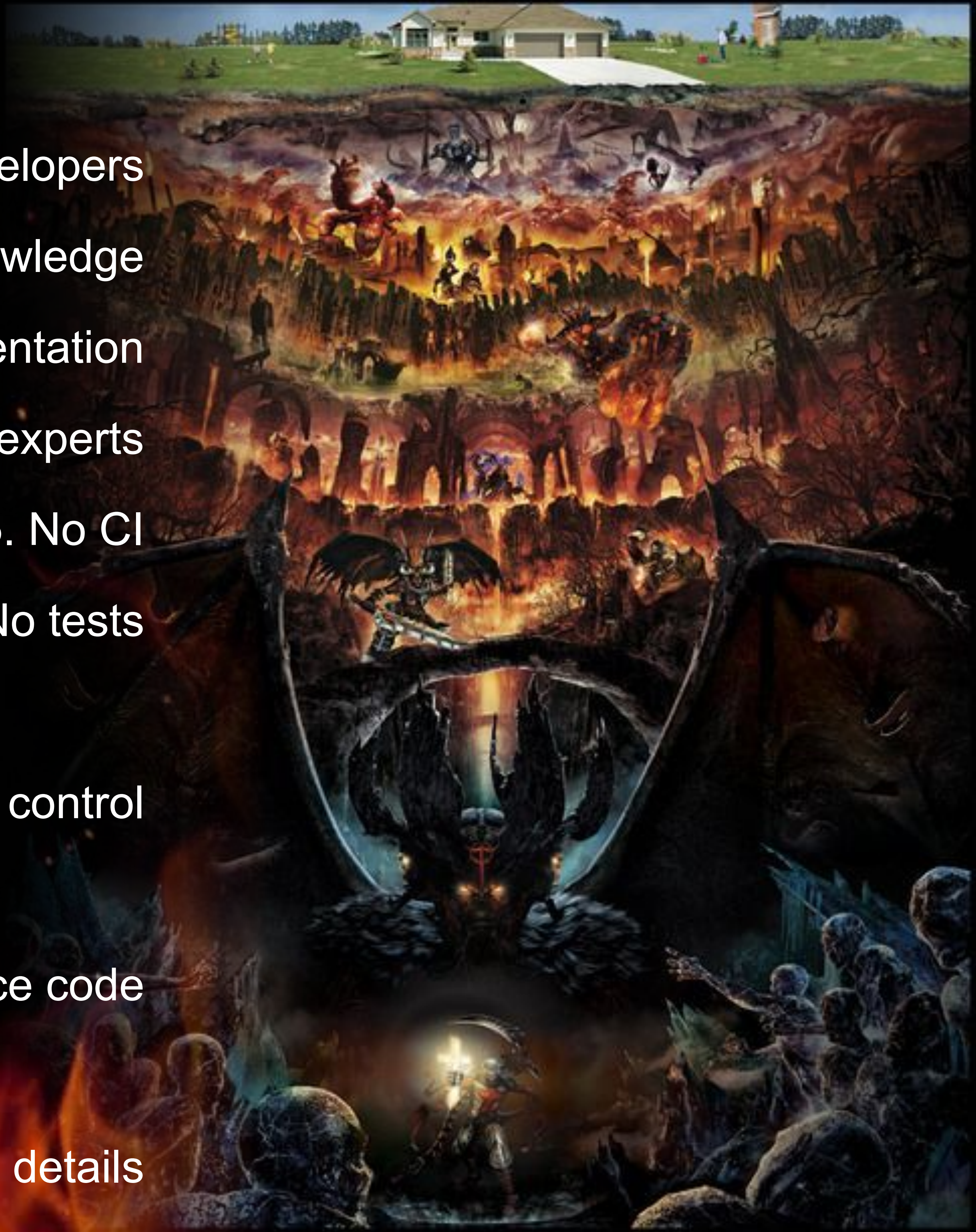
- Code is not an asset, it’s a liability, tests and technical documentation also need to be maintained and updated.
- The real value in the team is the knowledge of the business process acquired, not the code or the documentation. You need the why.
- Objective: Having the business knowledge, systems architecture and lessons learned into a single brain.

change wrong architecture → quick & dirty hack → push wrong abstraction → more hacks → can't reason about code → fear → frustration → developers leave → fear & confusion

Legacy systems are a communication problem

- Educate the business to make them aware of the real cost of maintaining a production system, and the consequences of not doing it properly.
- Use analogies. E.g. Housekeeping, remodeling a building without blueprints
- Maintaining code needs a budget, you need to negotiate it.

Nine circles of legacy hell



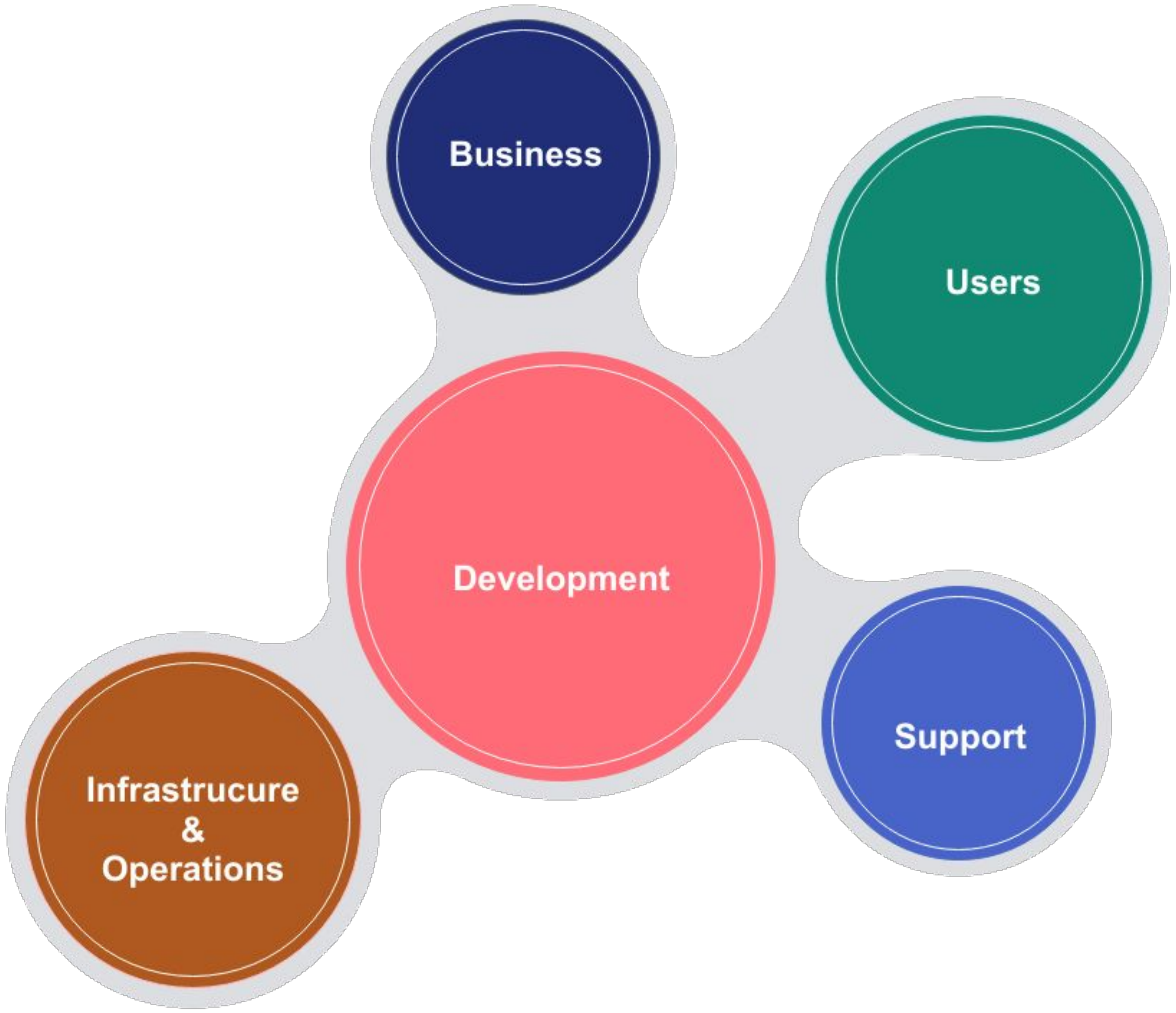
1. No original developers
2. No developers with knowledge
3. No documentation
4. No access to users or domain experts
5. No CI
6. No tests
7. No version control
8. No source code
9. No deployment details

Five Stages of Grief

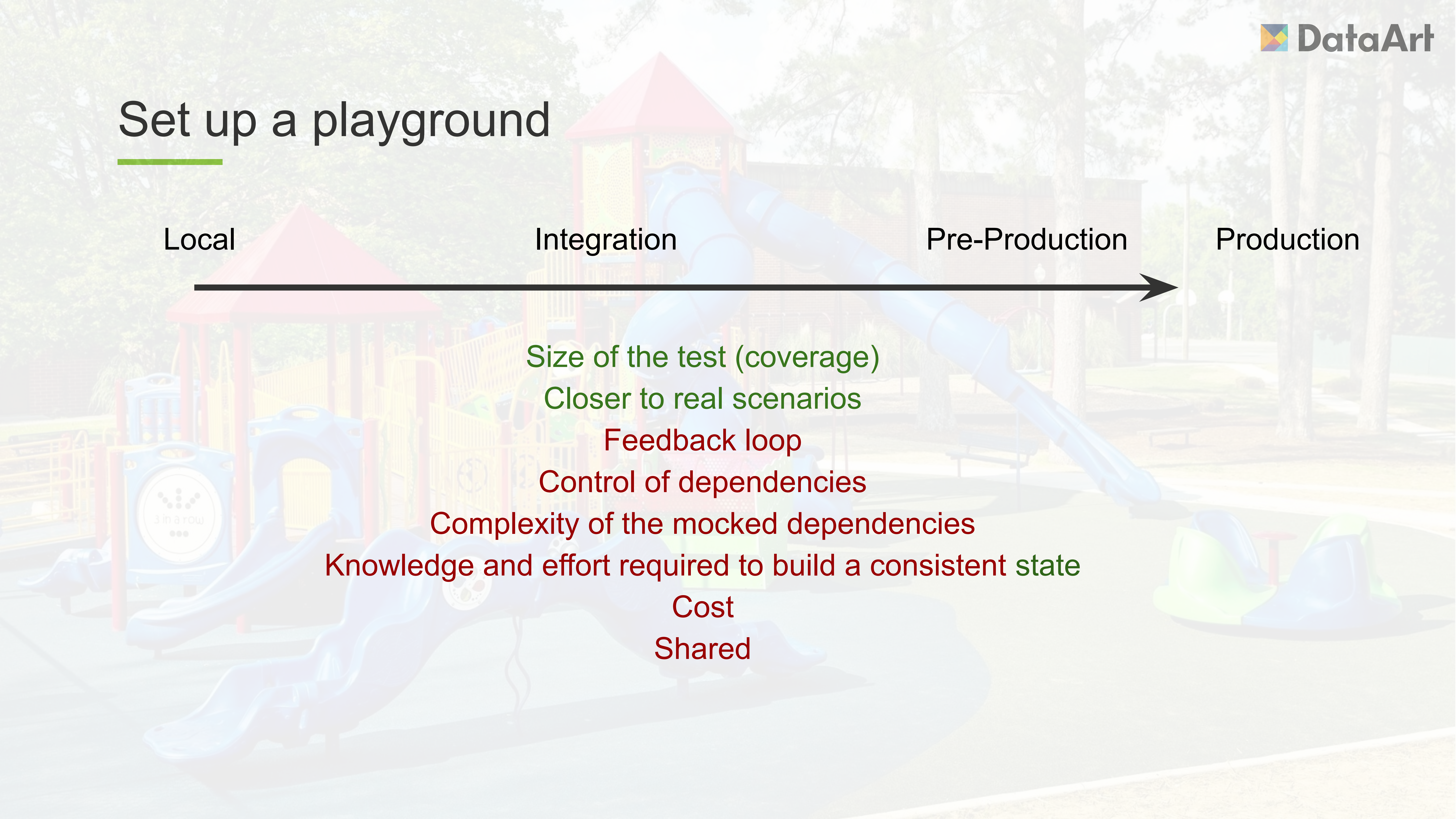
- **Denial** -> It can't be that bad, users have been using it for years
- **Anger** -> Who the f*** designed this system? and why did they do this? why????
- **Bargaining/Realization** -> Can I do a re-write and kill the old version?
- **Depression/Despair** -> There is no hope, I'm just wasting my time in this project.
- **Acceptance** -> It's the same everywhere, at least I can learn a few lessons and give a talk about it.

Understanding the system

Extract domain knowledge from people!



- Development team: Overview of the system
- Users: Job shadowing
- Business: Domain experts, which problem is the system solving?
- Support: What are the common problems with the system?
- Inf & Ops: Which resources does the system need?
- Write all that down!



Set up a playground

Local → Integration → Pre-Production → Production

Size of the test (coverage)

Closer to real scenarios

Feedback loop

Control of dependencies

Complexity of the mocked dependencies

Knowledge and effort required to build a consistent state

Cost

Shared

Look at the data!

“Show me your flowcharts and conceal your tables,

and I shall continue to be mystified.

Show me your tables,

and I won’t usually need your flowcharts;

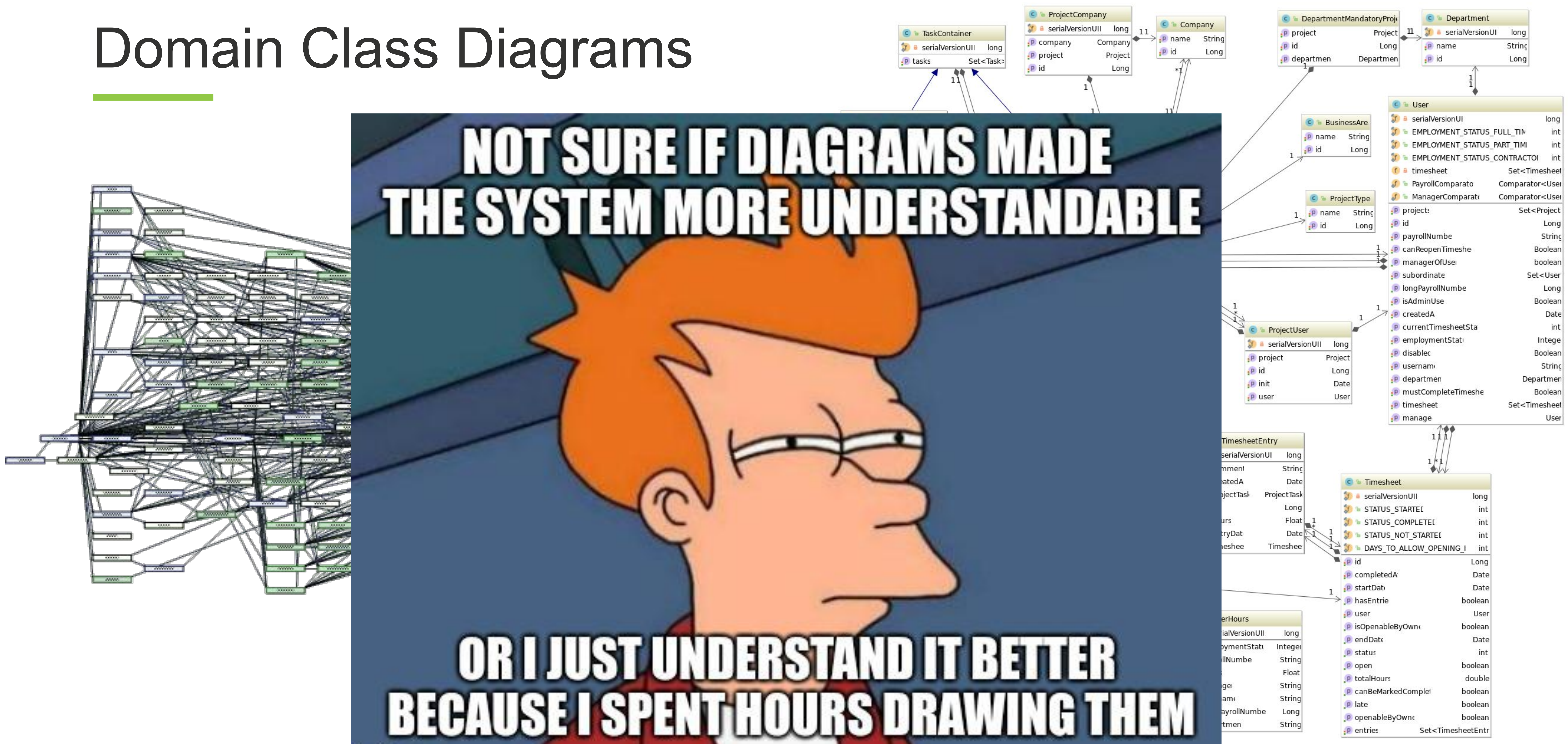
they’ll be obvious.”

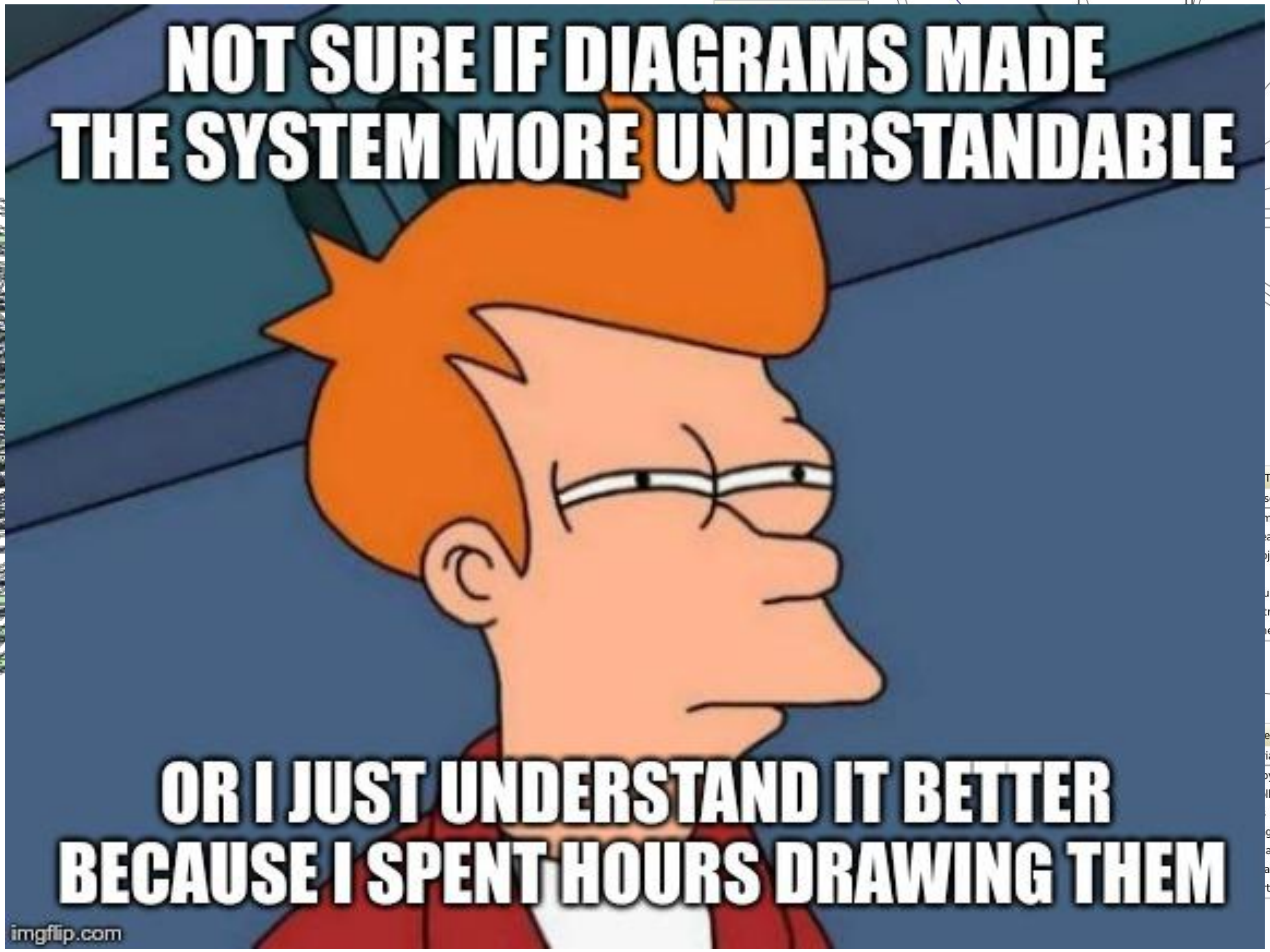
- Fred Brooks, (The Mythical Man-Month)

DB diagrams

- Generate diagrams from the DB schema:
  - SQLDeveloper (Oracle), MySqlWorkbench (MySql), pgModeler (PostgreSQL)
  - Schemaspy
- For more than 20 tables split it in clusters
  - DBeaver + yED
- Common problems:
  - Missing constraints (Unique, not null, PK, FK) or NoSQL
  - Physical schema all relationships are potentially N to 1
- Solutions
  - Look at the domain classes and ORM mappings
  - Statistics about on the actual content (Data exploration): Tableau, Looker, Jupyter Notebooks, Facets

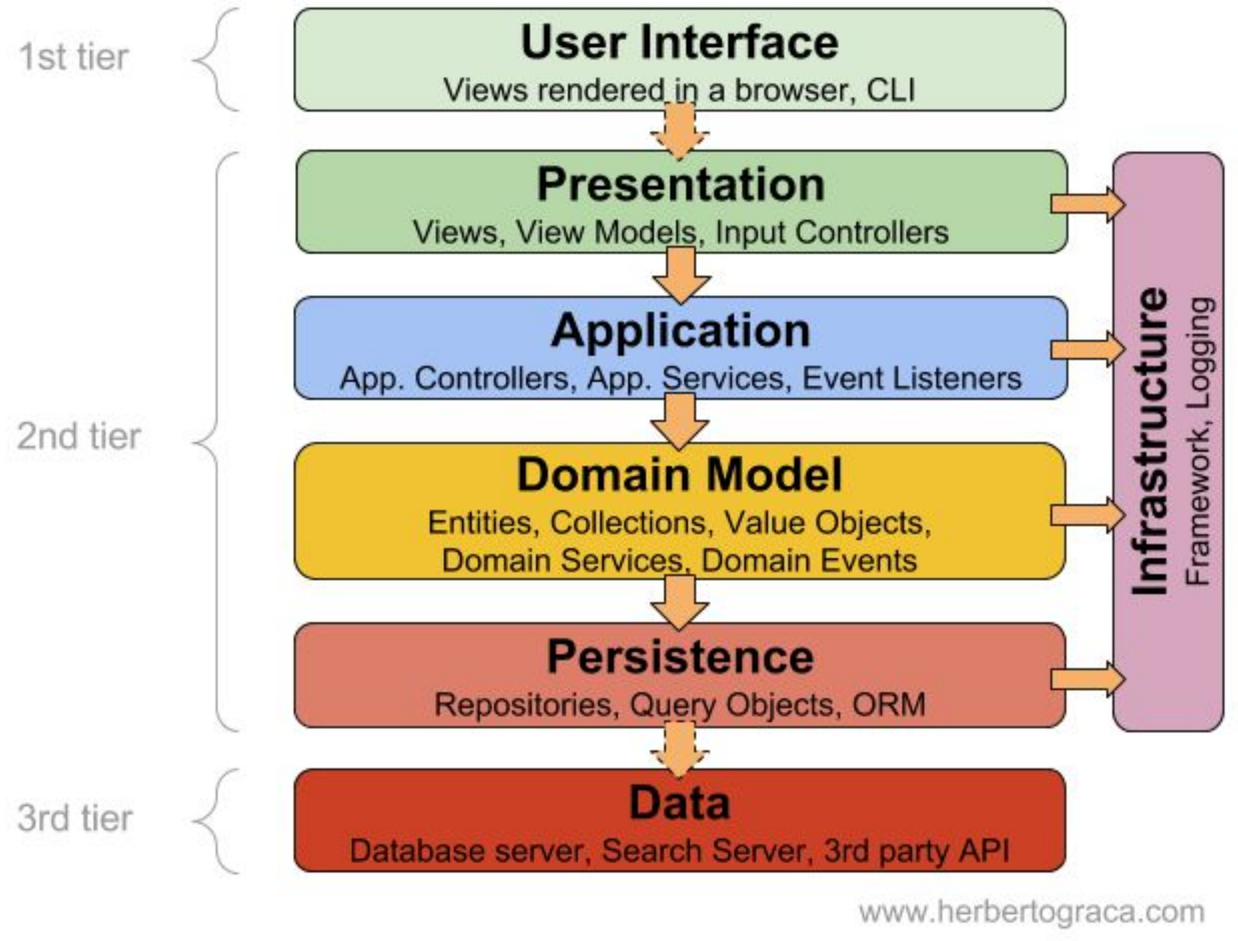
Domain Class Diagrams

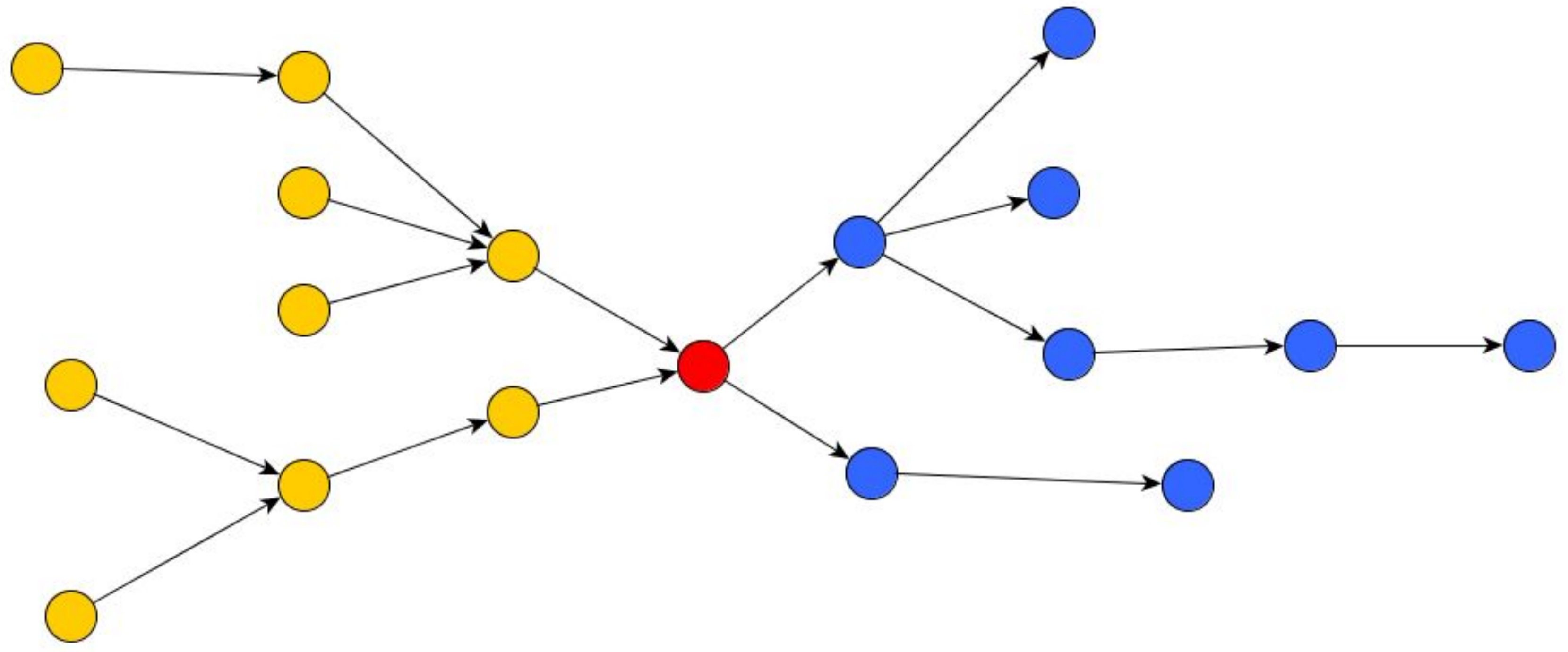



NOT SURE IF DIAGRAMS MADE THE SYSTEM MORE UNDERSTANDABLE
OR I JUST UNDERSTAND IT BETTER BECAUSE I SPENT HOURS DRAWING THEM
imgflip.com

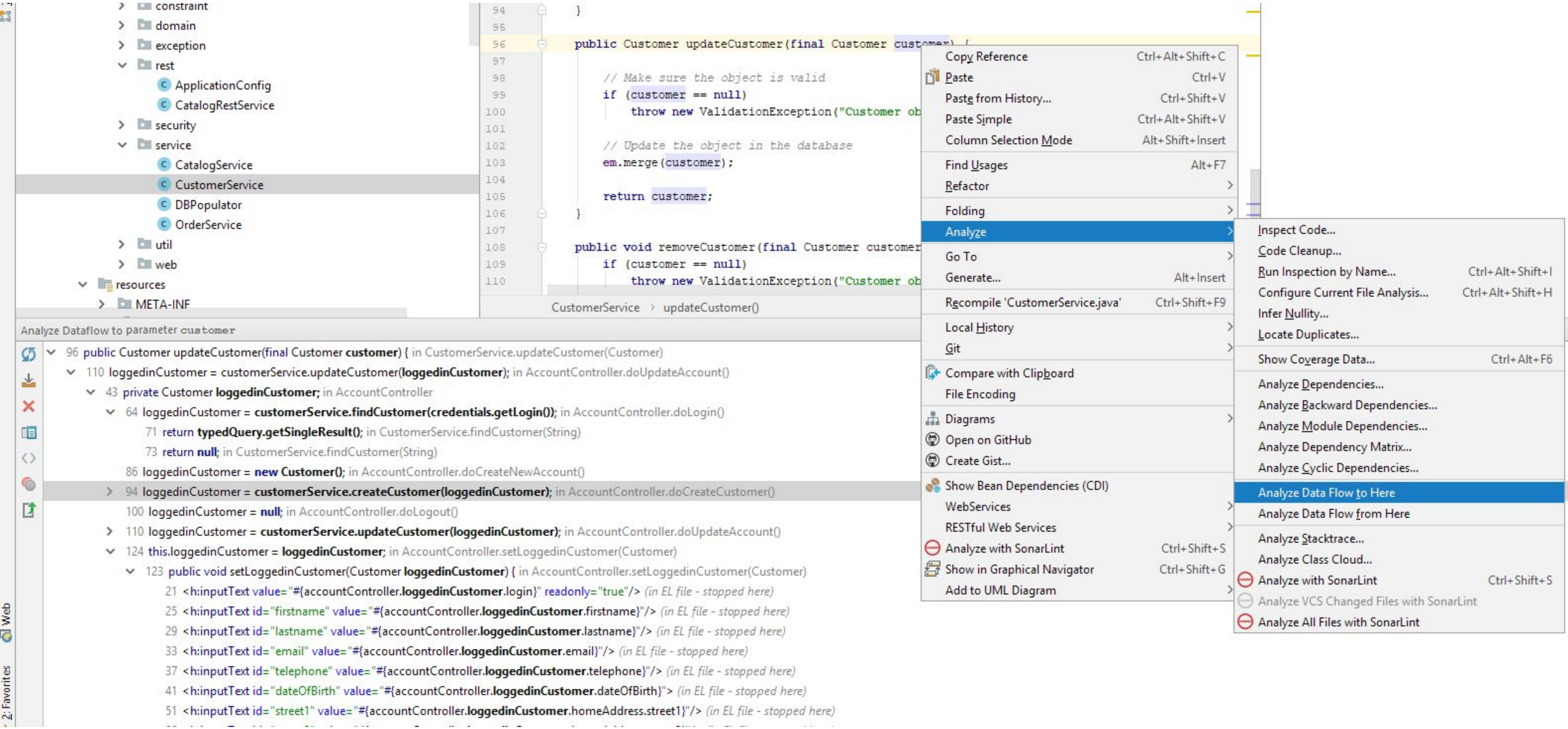
Identify processes and data flows

- Track the code from opposite directions: Interface and Data layer
- Debugging and breakpoints
- Extra verbose logging: structured, ad-hoc during debugging
- Scratch branch for extra comments and logging
- Make wild assumptions about the behaviour and confirm them
- Sequence diagrams, Call hierarchy, Analyze data flow from/to


1st tier
User Interface
Views rendered in a browser, CLI
2nd tier
Presentation
Views, View Models, Input Controllers
Application
App. Controllers, App. Services, Event Listeners
Domain Model
Entities, Collections, Value Objects, Domain Services, Domain Events
Persistence
Repositories, Query Objects, ORM
Infrastructure
Framework, Logging
3rd tier
Data
Database server, Search Server, 3rd party API
www.herbertograca.com



DataArt



```
> constraint
> domain
> exception
v rest
    ApplicationConfig
    CatalogRestService
> security
v service
    CatalogService
    CustomerService
    DBPopulator
    OrderService
> util
> web
v resources
  > META-INF
```

```
    }

    public Customer updateCustomer(final Customer customer) {

        // Make sure the object is valid
        if (customer == null)
            throw new ValidationException("Customer ob

        // Update the object in the database
        em.merge(customer);

        return customer;
    }

    public void removeCustomer(final Customer customer
        if (customer == null)
            throw new ValidationException("Customer ob
```

CustomerService › updateCustomer()

Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Paste from History... Ctrl+Shift+V
Paste Simple Ctrl+Alt+Shift+V
Column Selection Mode Alt+Shift+Insert
Find Usages Alt+F7
Refactor
Folding
Analyze
Go To
Generate... Alt+Insert
Recompile 'CustomerService.java' Ctrl+Shift+F9
Local History
Git
Compare with Clipboard
File Encoding
Diagrams
Open on GitHub
Create Gist...
Show Bean Dependencies (CDI)
WebServices
RESTful Web Services
Analyze with SonarLint Ctrl+Shift+S
Show in Graphical Navigator Ctrl+Shift+G
Add to UML Diagram
Inspect Code...
Code Cleanup...
Run Inspection by Name... Ctrl+Alt+Shift+I
Configure Current File Analysis... Ctrl+Alt+Shift+H
Infer Nullity...
Locate Duplicates...
Show Coverage Data... Ctrl+Alt+F6
Analyze Dependencies...
Analyze Backward Dependencies...
Analyze Module Dependencies...
Analyze Dependency Matrix...
Analyze Cyclic Dependencies...
Analyze Data Flow to Here
Analyze Data Flow from Here
Analyze Stacktrace...
Analyze Class Cloud...
Analyze with SonarLint Ctrl+Shift+S
Analyze VCS Changed Files with SonarLint
Analyze All Files with SonarLint

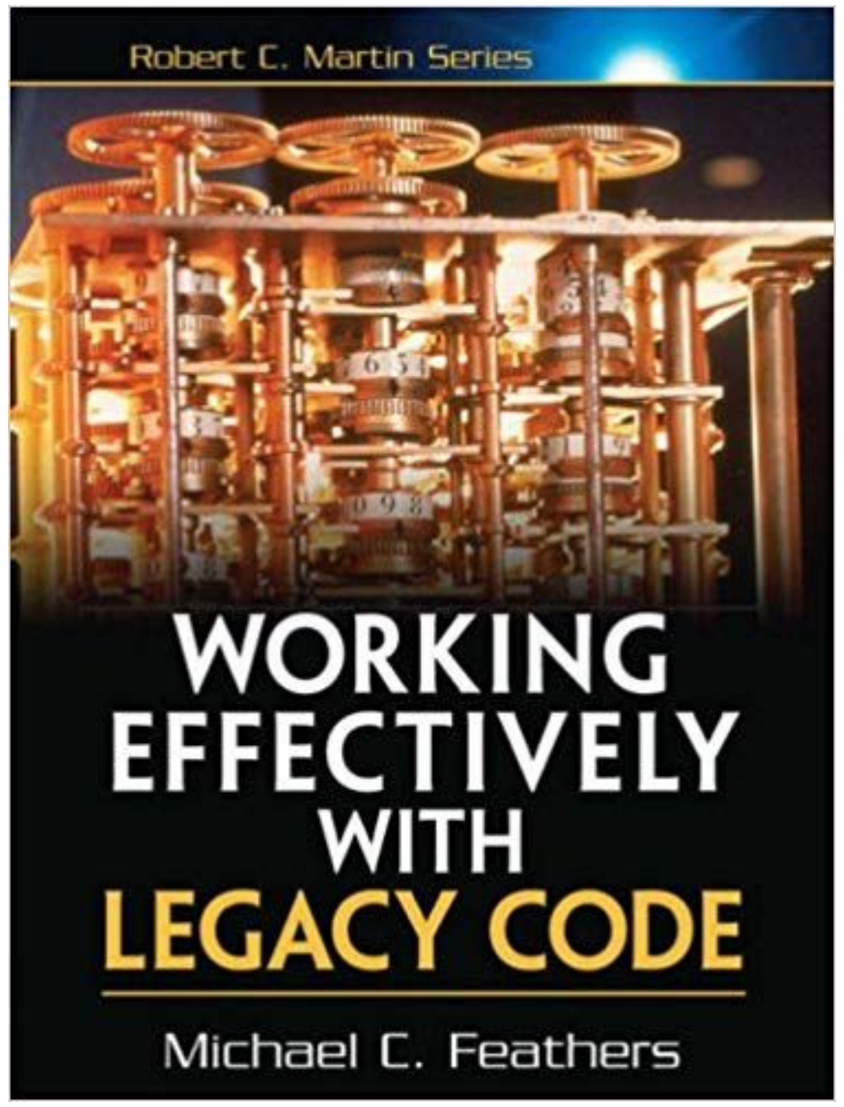
Analyze Dataflow to parameter `customer`

```
v 96 public Customer updateCustomer(final Customer customer) { in CustomerService.updateCustomer(Customer)
  v 110 loggedinCustomer = customerService.updateCustomer(loggedinCustomer); in AccountController.doUpdateAccount()
    v 43 private Customer loggedinCustomer; in AccountController
      v 64 loggedinCustomer = customerService.findCustomer(credentials.getLogin()); in AccountController.doLogin()
          71 return typedQuery.getSingleResult(); in CustomerService.findCustomer(String)
          73 return null; in CustomerService.findCustomer(String)
        86 loggedinCustomer = new Customer(); in AccountController.doCreateNewAccount()
      > 94 loggedinCustomer = customerService.createCustomer(loggedinCustomer); in AccountController.doCreateCustomer()
        100 loggedinCustomer = null; in AccountController.doLogout()
      > 110 loggedinCustomer = customerService.updateCustomer(loggedinCustomer); in AccountController.doUpdateAccount()
      v 124 this.loggedinCustomer = loggedinCustomer; in AccountController.setLoggedinCustomer(Customer)
        v 123 public void setLoggedinCustomer(Customer loggedinCustomer) { in AccountController.setLoggedinCustomer(Customer)
            21 <h:inputText value="#{accountController.loggedinCustomer.login}" readonly="true"/> (in EL file - stopped here)
            25 <h:inputText id="firstname" value="#{accountController.loggedinCustomer.firstname}"/> (in EL file - stopped here)
            29 <h:inputText id="lastname" value="#{accountController.loggedinCustomer.lastname}"/> (in EL file - stopped here)
            33 <h:inputText id="email" value="#{accountController.loggedinCustomer.email}"/> (in EL file - stopped here)
            37 <h:inputText id="telephone" value="#{accountController.loggedinCustomer.telephone}"/> (in EL file - stopped here)
            41 <h:inputText id="dateOfBirth" value="#{accountController.loggedinCustomer.dateOfBirth}"> (in EL file - stopped here)
            51 <h:inputText id="street1" value="#{accountController.loggedinCustomer.homeAddress.street1}"/> (in EL file - stopped here)
```

Test, Modify & Refactor

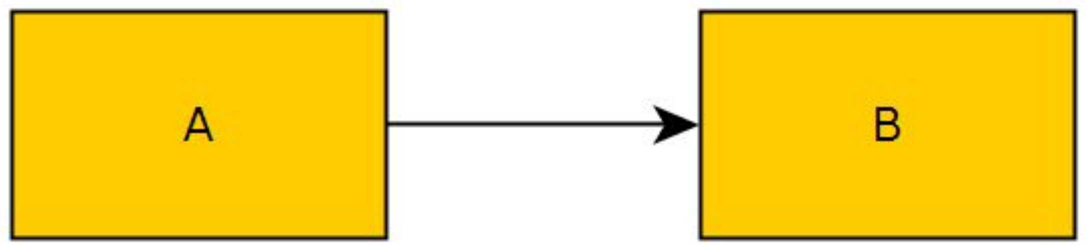
The need for tests

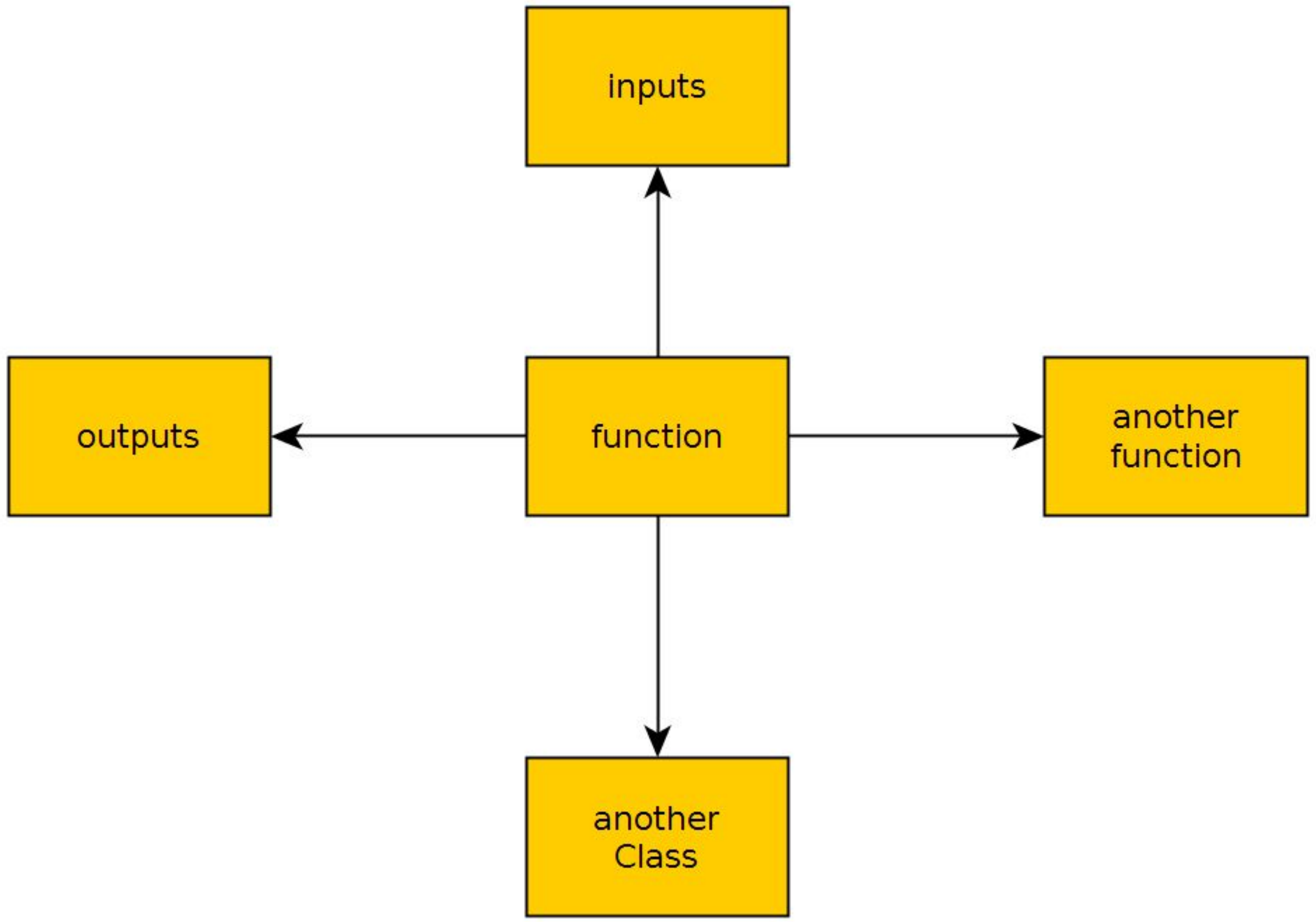
- It's the only sane way of changing code.
- Nobody is infallible, not even yourself!



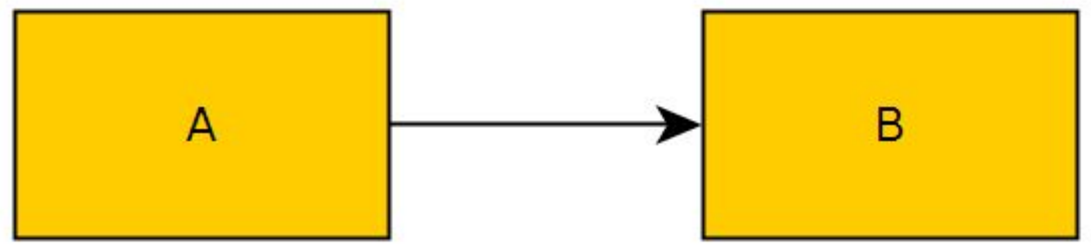
1. Test before change.
2. Dependencies are what makes testing hard.
3. Make the minimal changes needed in order to break them and create seams.
4. Use the seams to isolate the code under test and mock the rest.
5. Write tests for the existing behaviour.
6. Refactor.
7. Write tests for the new behaviour.
8. Make changes.
9. Repeat, keep increasing the test-coverage.

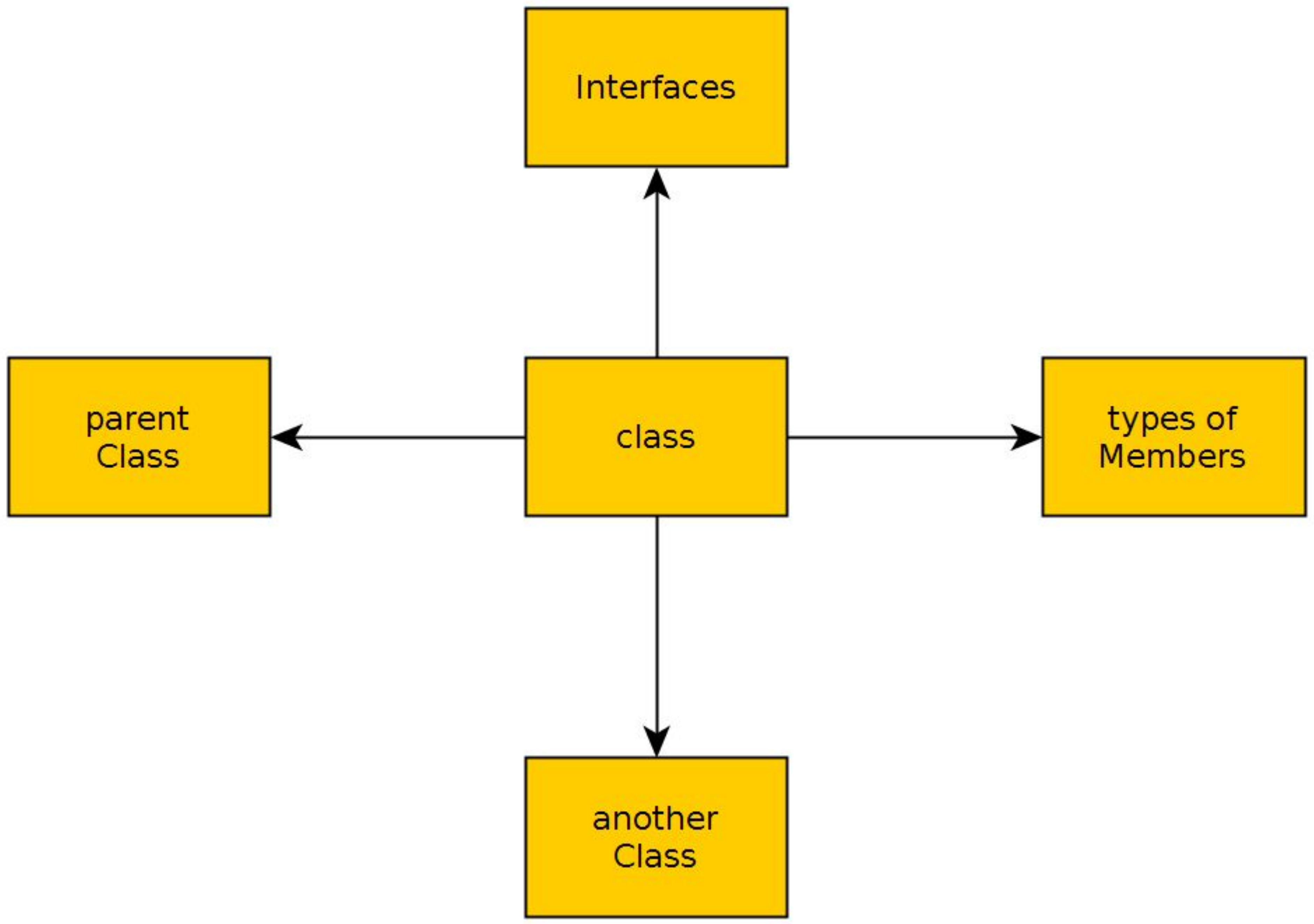
Dependencies



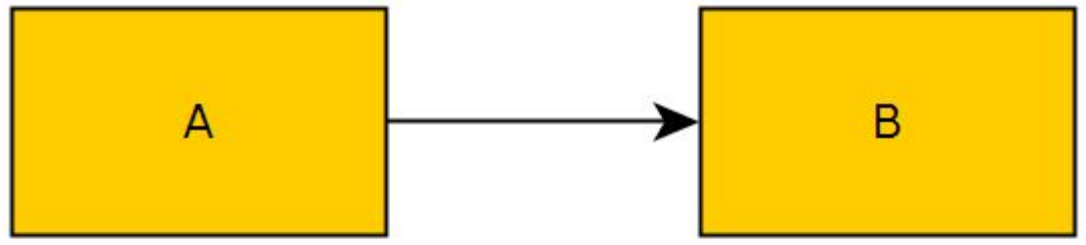


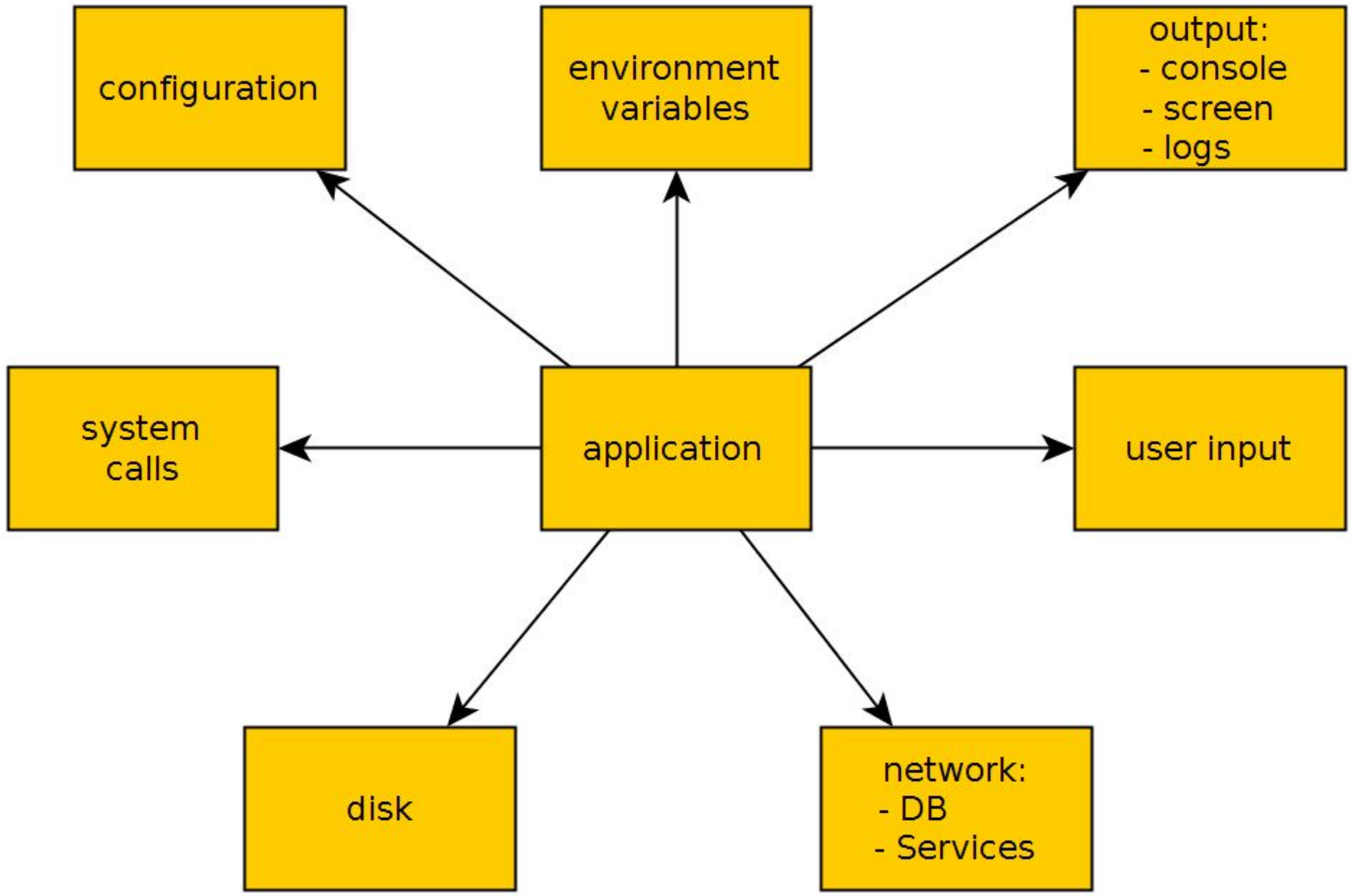
Dependencies



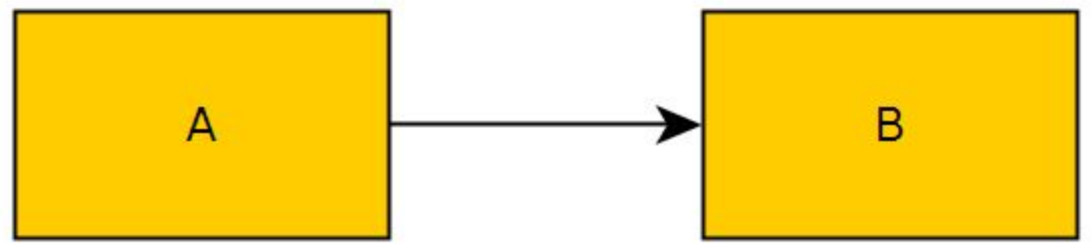


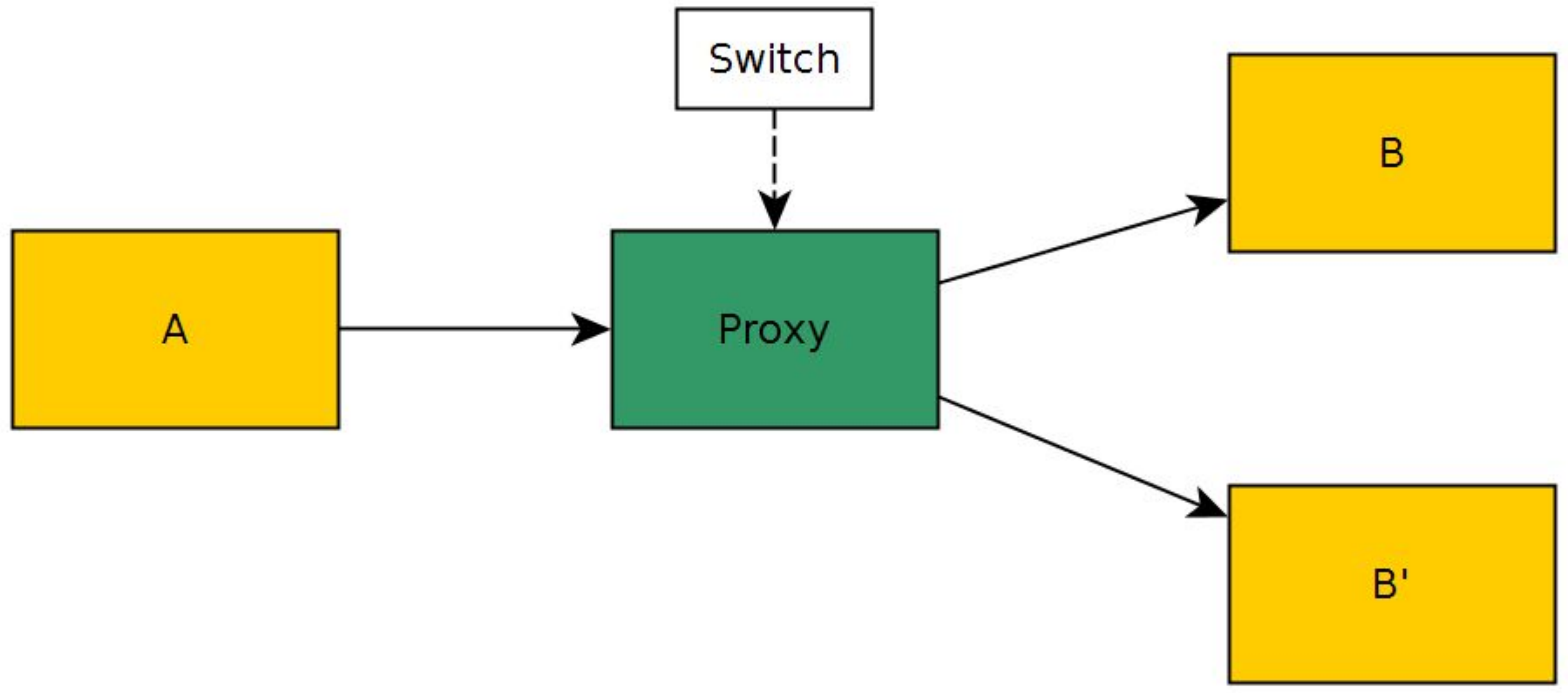
Dependencies



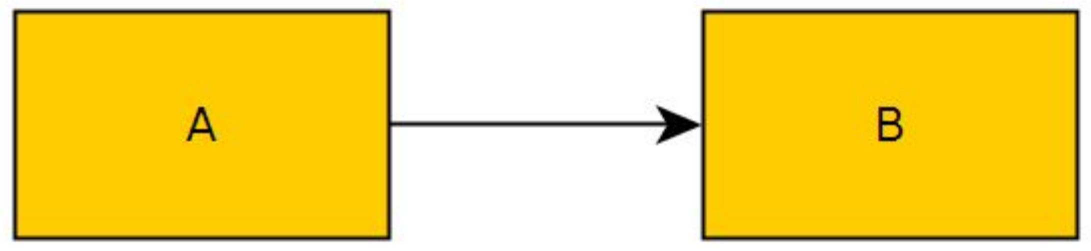


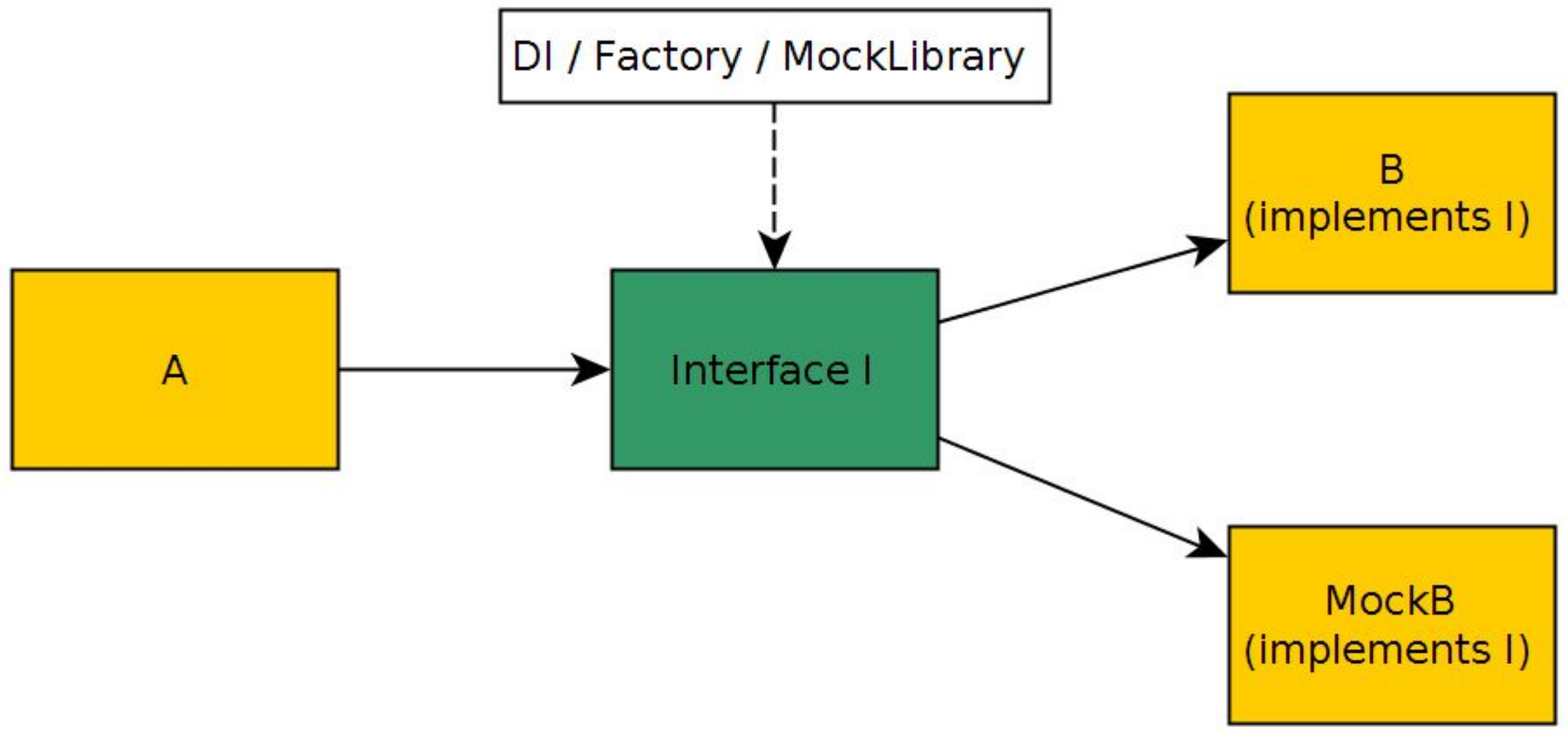
Seams



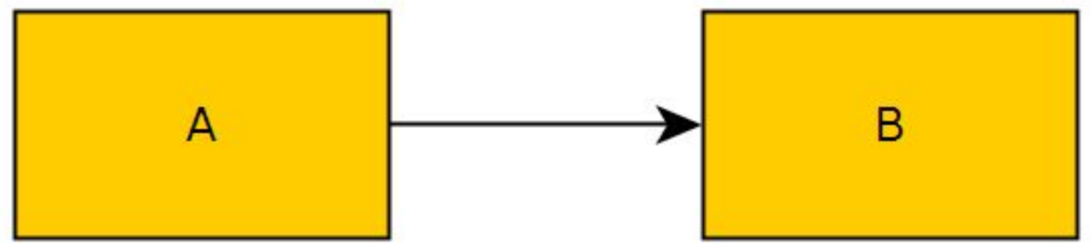


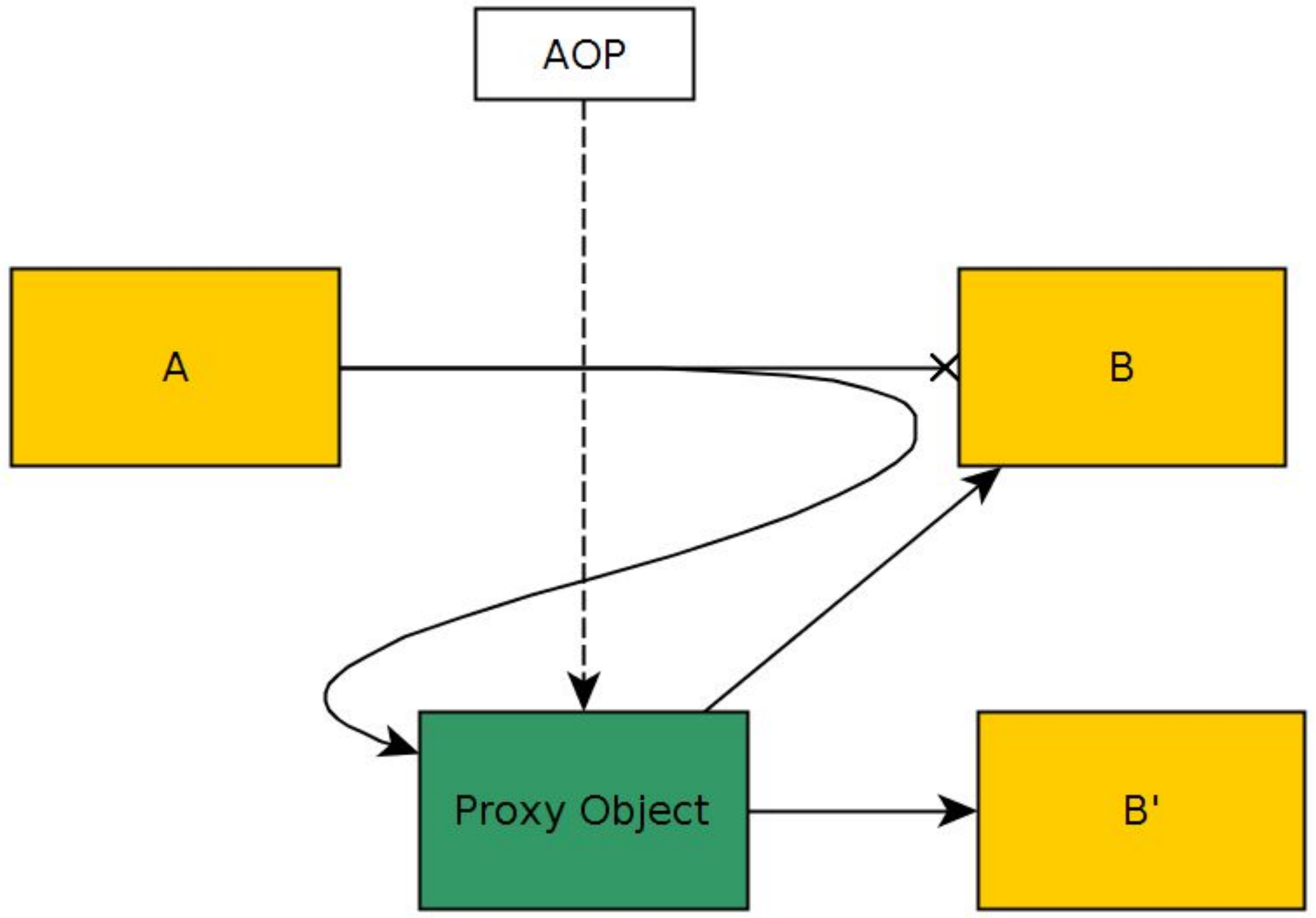
Seams


A
B


DI / Factory / MockLibrary
A
Interface I
B (implements I)
MockB (implements I)

Seams





```java
class Employee extends Person {
    ...
    void calculatePaycheck() throws Exception {
        log.debug("Calculating paycheck for employee {} on {}", this.getEmployeeId(), new Date());
        DateRange currentMonth = DateUtils.getMonth(new Date());

        double seniorityBonus = employmentHistory.getSeniority(currentMonth.getStartDate());
        PaycheckGenerator payCheckGen = new PaycheckGenerator(this);
        double totalGrossPay = 0;
        for (Calendar date = currentMonth.getStartDate(); date.before(currentMonth.getEndDate()); date.add(Calendar.DATE, 1)) {
            int hours = workSchedule.getWorkedHours(date);
            double grossDayPay = salaryHistory.salaryAt(date) * seniorityBonus * hours;
            totalGrossPay += grossDayPay;
        }
        double pensionContribution;
        try {
            pensionContribution = PensionManagerService.getPensionContribution(this.getEmployeeId());
        } catch (Exception e) {
            log.error("WTF! this should never happen!");
            throw e;
        }
        totalGrossPay -= pensionContribution;
        double taxRate = totalGrossPay >= TaxBands.Threshold1 ? TaxBands.Rate1 : TaxBands.Rate2;
        double taxDeduction = this.getMaritalStatus() == MaritalStatus.Married ? TaxDeduction.Type2 : TaxDeduction.Type1;
        double totalTax = totalGrossPay * taxRate - taxDeduction;
        payCheckGen.setGross(totalGrossPay);
        payCheckGen.setTax(totalTax);
        payCheckGen.setPensionContribution(pensionContribution);
        Paycheck paycheck = payCheckGen.generate();
        log.debug("Calculating paycheck for employee {} for during {}, resulting in {}",
                this.getEmployeeId(), currentMonth, paycheck);
        this.addPaycheck(paycheck);
    }
```

DataArt

Summary

- Legacy code is a communication problem. As a developer you need to educate, build realistic expectations and negotiate.
- With the right amount of time, resources and techniques, it does not have to be painful.
- Maintaining legacy code is the ultimate training for learning how to write maintainable code.
- Discovering the insights of a business and rescuing a legacy system can be very rewarding.

I ♥
Legacy
Systems