10
Sun

May
2020

08:10
PM

UTC+4

Imran Yusubov
Crossover

LMS Microservices in production

Proudly supported by

ERPGO

kiss-conf

2 days, 13 speakers

Keep it stupid simple

https://kiss-conf.goupaz.com

Host: Nabi Nabizade

Kiss.Conf 2020

Imran Y

Love learning, sharing, stock investing, business etc.

Contact

Linkedin: https://www.linkedin.com/in/imran-yusubov-9334744a/

Education

Qafqaz Uni(Azerbaijan)

Experience

Current :

Chief Software Architect - Crossover, Global

Founder - Ingress Group, Ingress Academy

Past:

Aurea, Optiva, Versata - Quantum Retails, Sinam, Azerconnect

What is it?

Learning management system,

Technical Highlights

- Mainly Java
- Microservice Architecture
- Stateless
- JWT (Oauth)
- CI/CD
- GKE & GCP
- ~70 80% Unit Test Coverage
- Service Mesh

All development is done by the students of Ingress Academy

Problems

Problems with software development

1. Business does not understand developers
2. Difficult to find experienced developer
3. Education system teaches how to do things wrong than right

All of those factors contribute to the results, and we end up with software that no one needs

What we learn?

Stuff that no one needs, or better to avoid

1. Static methods, static fields, initialization blocks, constructor chaining, checked exceptions, inheritance etc.
2. Singleton, Utility classes, Factory Method, Abstract Factory, Session
3. All sorts of legacy frameworks and libraries (Jsp, Servlet, JSTL and etc)

It takes many more years to learn that we should avoid much of those stuff.

We keep repeating the mistakes over and over.

The motivation

A search for developing maintainable code,

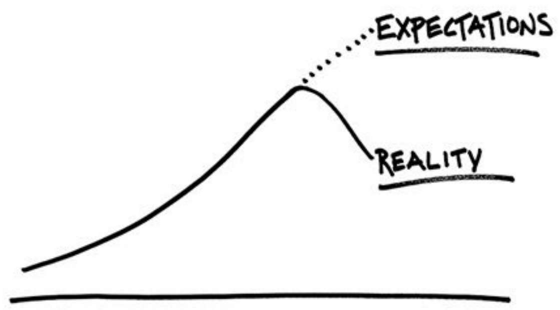
1. A journey for a sustainable business model for a small software company
2. A common platform to learn & share
3. To implement a DevOps pipeline that produces not only software but also developers with good coding habits

Can we develop maintainable software with inexperienced developers (usually experience < a year)

Choose the right architecture

The architecture must be scalable, not only in terms of load but also in terms of development

1. Applications keep getting bigger & bigger
2. Requirements keep changing
3. Technologies keep changing and becoming absolute



The right architecture must be flexible to accommodate all of those changes.

Choose the right infrastructure

Flexible architecture requires flexible infrastructure.

1. Automation can be applied to software defined components
2. Apply infrastructure as code & eliminate manual maintenance efforts
3. Use CI/CD

Agility = right architecture + right infrastructure

The devil is in the detail

The real problems are always in the details

Apply code analysis tools

The combination of tools are able to detect much of the problems

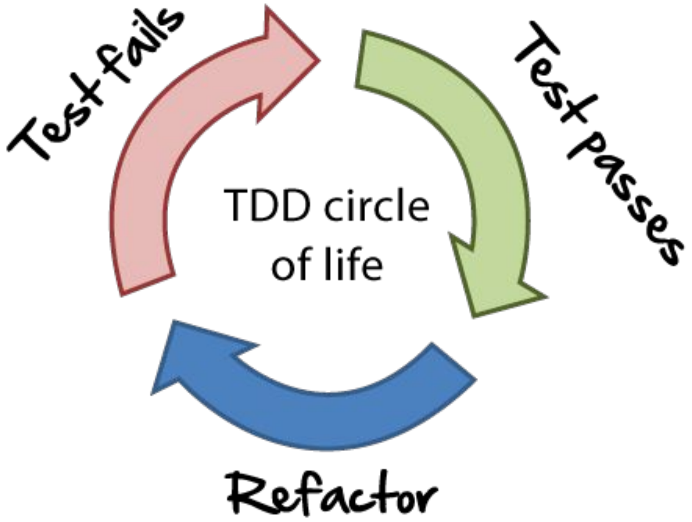
1. Apply code style
2. PMD Java
3. Checkstyle
4. FindBugs
5. Find Security Bugs

Developers learn quickly and produce unified and maintainable (clean) code

Enforce TDD

Enforce unit test coverage targets in your CI pipeline, and it will enforce coding best practices

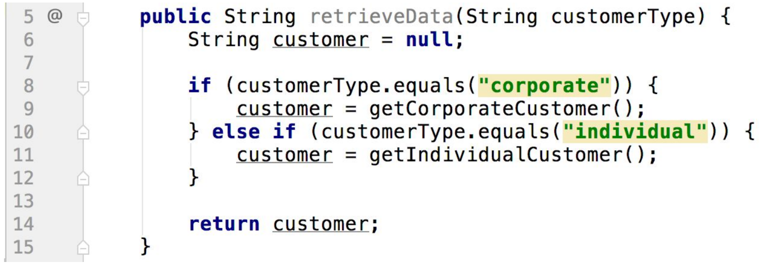
1. Developers can reach the coverage targets only if the code is testable
2. Testable code can be developed when we apply best practices
3. Best practices result in code with less bugs in it
4. Use code coverage tools (e.g. Jacoco)



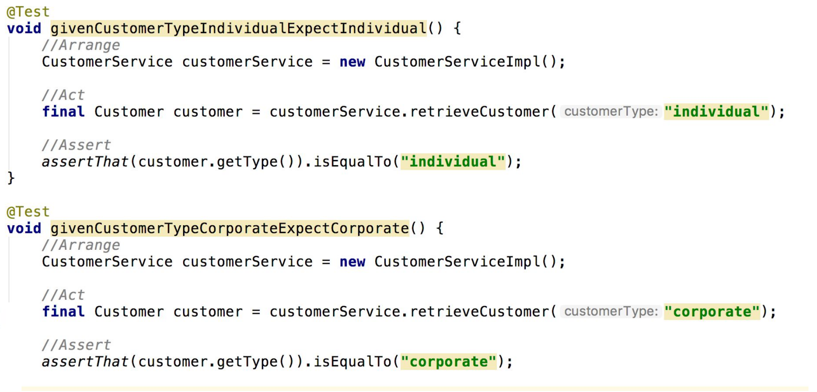
Your test suit is the main guard against 'Integration Hell'

Code coverage vs Test Coverage

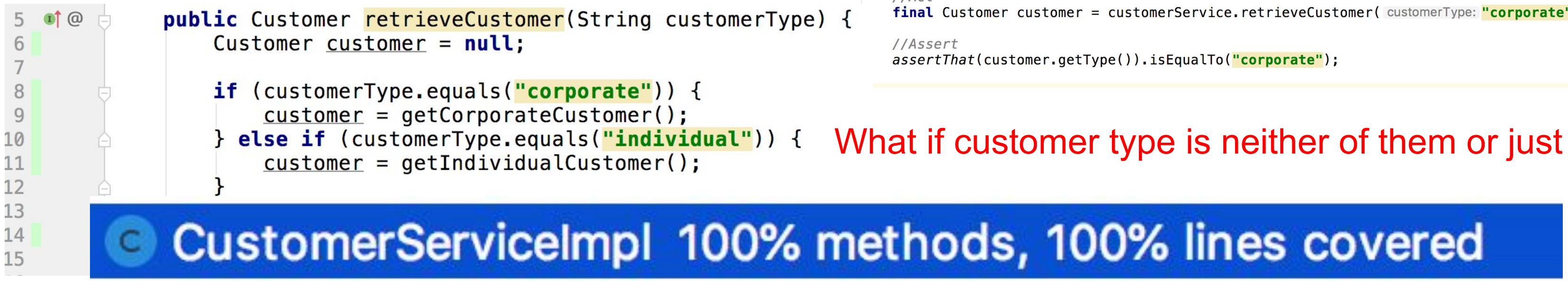
Unit test just for the sake of code coverage is not valuable



```java
public String retrieveData(String customerType) {
    String customer = null;

    if (customerType.equals("corporate")) {
        customer = getCorporateCustomer();
    } else if (customerType.equals("individual")) {
        customer = getIndividualCustomer();
    }

    return customer;
}
```



```java
@Test
void givenCustomerTypeIndividualExpectIndividual() {
    //Arrange
    CustomerService customerService = new CustomerServiceImpl();

    //Act
    final Customer customer = customerService.retrieveCustomer( customerType: "individual");

    //Assert
    assertThat(customer.getType()).isEqualTo("individual");
}

@Test
void givenCustomerTypeCorporateExpectCorporate() {
    //Arrange
    CustomerService customerService = new CustomerServiceImpl();

    //Act
    final Customer customer = customerService.retrieveCustomer( customerType: "corporate");

    //Assert
    assertThat(customer.getType()).isEqualTo("corporate");
```



```java
public Customer retrieveCustomer(String customerType) {
    Customer customer = null;

    if (customerType.equals("corporate")) {
        customer = getCorporateCustomer();
    } else if (customerType.equals("individual")) {
        customer = getIndividualCustomer();
    }
```

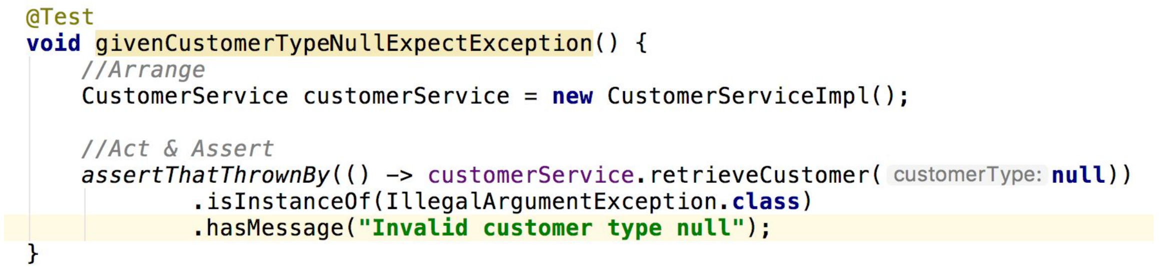
CustomerServiceImpl 100% methods, 100% lines covered

What if customer type is neither of them or just null?

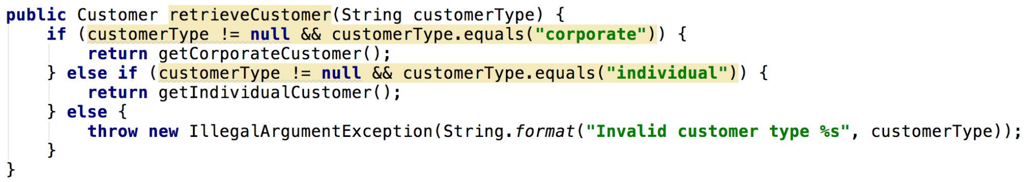
Kiss.Conf 2020

Requirement driven test coverage

Test should be driven by requirement, and should cover all side cases



```
@Test
void givenCustomerTypeNullExpectException() {
    //Arrange
    CustomerService customerService = new CustomerServiceImpl();

    //Act & Assert
    assertThatThrownBy(() -> customerService.retrieveCustomer( customerType: null))
            .isInstanceOf(IllegalArgumentException.class)
            .hasMessage("Invalid customer type null");
}
```



```
public Customer retrieveCustomer(String customerType) {
    if (customerType != null && customerType.equals("corporate")) {
        return getCorporateCustomer();
    } else if (customerType != null && customerType.equals("individual")) {
        return getIndividualCustomer();
    } else {
        throw new IllegalArgumentException(String.format("Invalid customer type %s", customerType));
    }
}
```

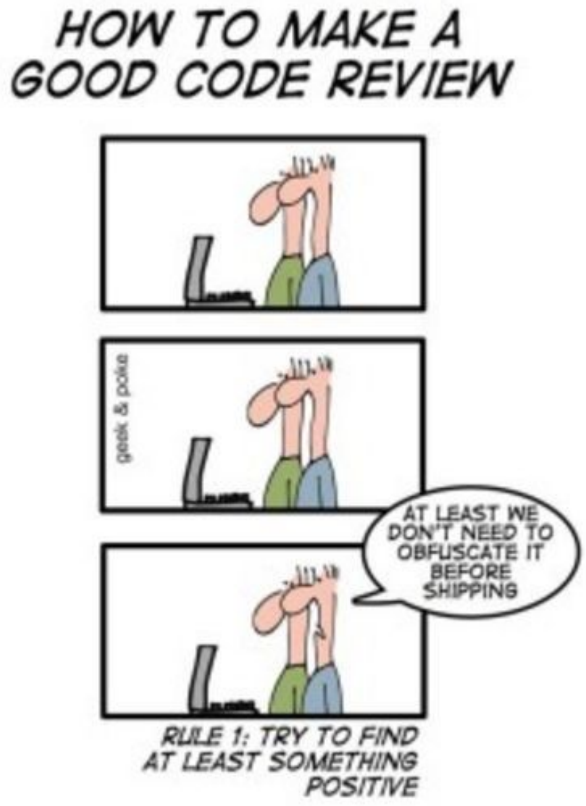
See how much logic we had forgotten, which could cost hours or days to, all involved parties.
Guess who pays for that :) .

Code review

Create a checklist (not to do list) and enforce it,

1. Details matter, they cost and they cost a lot
2. Create a not to do list
3. Many talking about what to do but few talking about what not to do
4. Not everything is detected by code review tools, enforce manual review ‘
5. Encourage peer review as people can learn from others mistakes
6. Always be open to feedback

Code review practices can rapidly increase developer’s skill set


HOW TO MAKE A GOOD CODE REVIEW
geek & poke
AT LEAST WE DON'T NEED TO OBFUSCATE IT BEFORE SHIPPING
RULE 1: TRY TO FIND AT LEAST SOMETHING POSITIVE

Results & Next steps

Make things open source to get community feedback

1. Share what we have learned
2. Encourage collaboration to solve complex problems
3. Involve more developers than just our students

Q&A Discussion

Link to Q&A Panel: https://bit.ly/2KyViHb