

# TESTING

G. Molines

2020-2021



# INTRODUCTION

# Software quality

- What is it?

# Software quality

- What is it?
- No Defects?

# Software quality

- What is it?
- No Defects?
- Or....
- Known defects?

# Known defects???

- Goals of quality are:
  - Know and document bugs
  - Verify them for regression
  - Find workarounds
  - Feed more requirements



# How do you **choose** your defects?

# How do you **choose** your defects?

- Dev methodology
- Team culture
- Good practices
- Correct soft. Architecture
- Early Integration
  
- And...



# How do you **choose** your defects?

- Dev methodology
- Team culture
- Good practices
- Correct soft. Architecture
- Early Integration
- And...


- **Testing**

# Types of tests

- Unit-tests
- Integration tests
- Acceptation tests
- ...ok.... What else?


# Types of tests

- Unit Tests
- Integration Tests
- GUI Tests
- Non-regression Tests
- Coverage Tests
- Load Tests
- Stress Tests
- Performance Tests
- Scalability Tests
- Reliability Tests
- Volume Tests
- Usability Tests
- Security Tests
- Recovery Tests
- L10N/I18N Tests
- Accessibility Tests
- Installation/Configuration Tests
- Documentation Tests
- Platform testing
- Samples/Tutorials Testing
- Code inspections



Seriously, you do all of this stuff???

How the hell do you get organized???



Seriously, you do all of this stuff???

How the hell do you get organized???

# **TEST ARCHITECT**

# TEST ARCHITECT

# Test architect role

- Choose test tools
- Define practices
- Build base frameworks → reusability
- Find test data
- Global test bucket consistency, strategy

# Test Tools

## Criteria

- Language
  - Eg: java -> jUnit and variants (HttpUnit, etc.)
- Type of app
  - Eg: desktop, mobile
- Layer where test is applied
  - Eg: UI → Selenium, RFT and other robots
- Goals
  - Eg: Perf -> jMeter
- Phase
  - Eg: functional vs system, see later



# Define Test Practices

- Coding guidelines
- Test inventory tracking
  - Inventory tooling (RQM and the likes)
- Manual vs automated
  - ROI
  - Exploratory

# Define Test Practices

- Policies
  - Eg: each new defect gets a testcase
- Testability
  - Eg: naming UI elements
- Repeatability
  - Setup / cleanup guidelines
  - Or: VM snapshotting

# Test Data

- Representative samples
- Customer data
  - through support ticket
- Adhoc data
  - Eg: to cause error conditions
- Need be managed
  - Inventoried
  - Anonymized (confidential info)
  - Setup (Eg: prod-like DB)

# Test Strategy

- Aka: where and when to test
- Sources of information
  - User defect reports
  - Coverage data
  - User story / functional understanding
- Measure effectiveness
  - # defects / regressions found
  - Time to find next defect
  - Time spent in maintenance

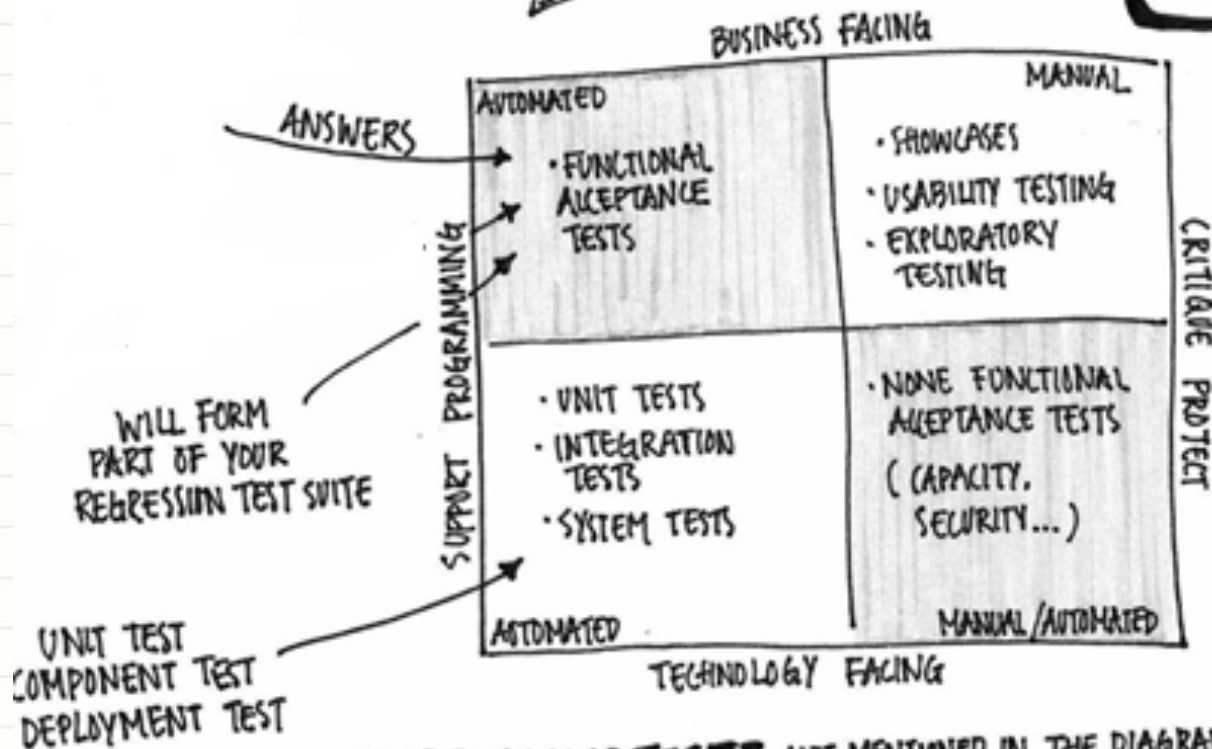
# Test Dimensions

Can be organized based on:

- Product area
- Goals (find defects, measure perf., ...)
- Team in charge
- Dev. Phase
- Execution time

## TYPE OF TESTS

YOUR DEPLOYMENT PIPELINE SHOULD HAVE ALL THESE FOUR TYPE OF TESTS.



**REGRESSION TEST?** NOT MENTIONED IN THE DIAGRAM. THEY ARE CROSSCUTTING CATEGORY.

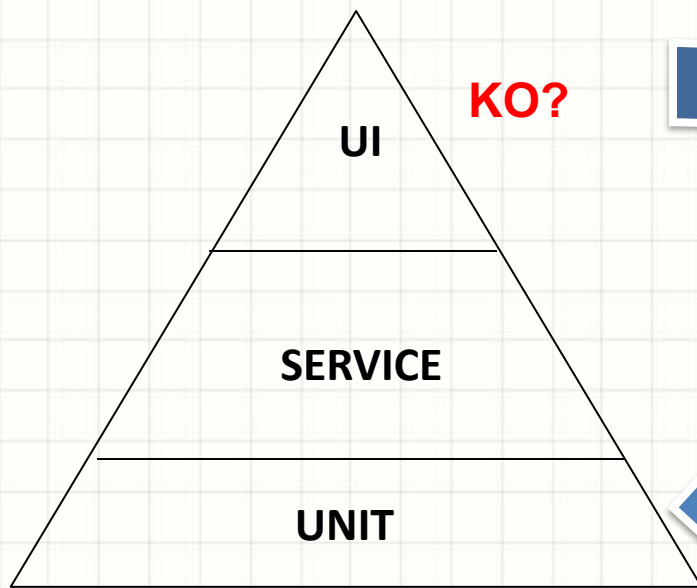
# Test Sizes

- Number
- Execution time
- Depth of paths traversed

*“In particular I always argue that high-level tests are there as a second line of test defense. If you get a failure in a high level test, not just do you have a bug in your functional code, you also have a missing unit test. Thus whenever you fix a failing end-to-end test, you should be adding unit tests too.”*

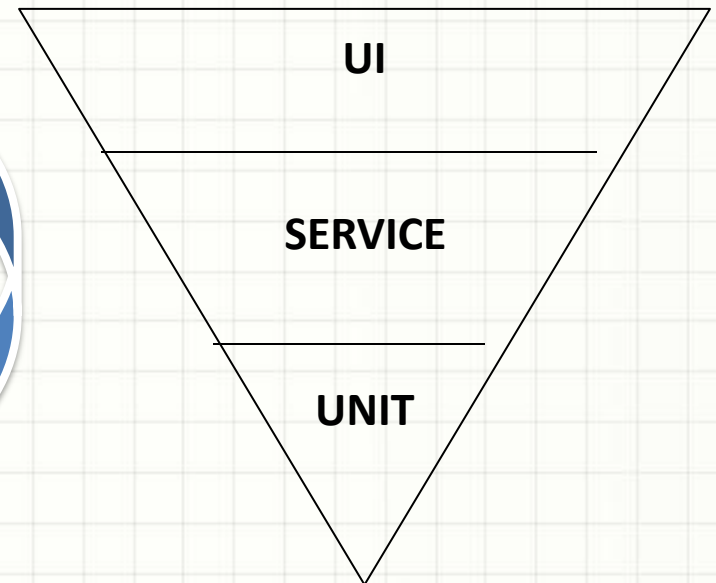
*Martin Fowler*





**KO?**

Volume

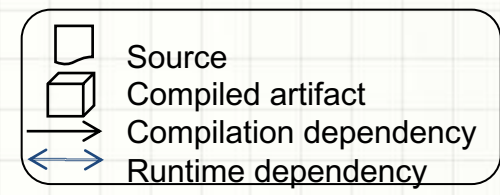
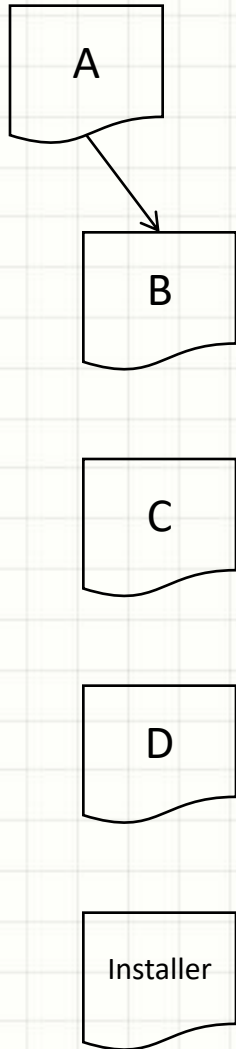


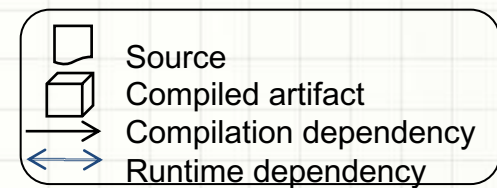
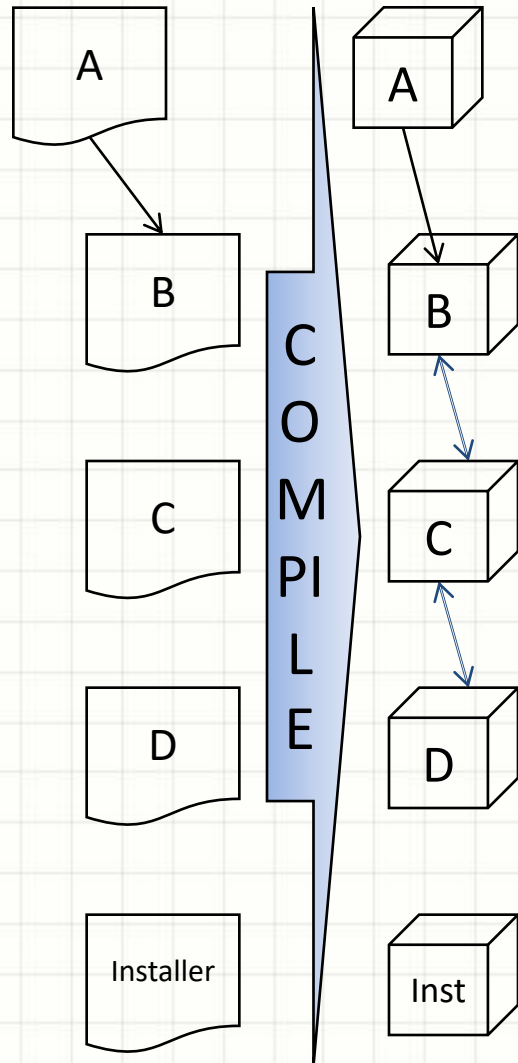
Run time

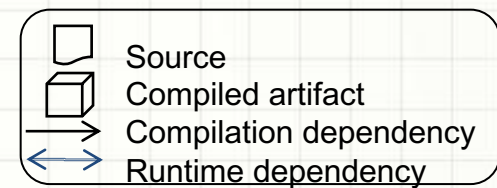
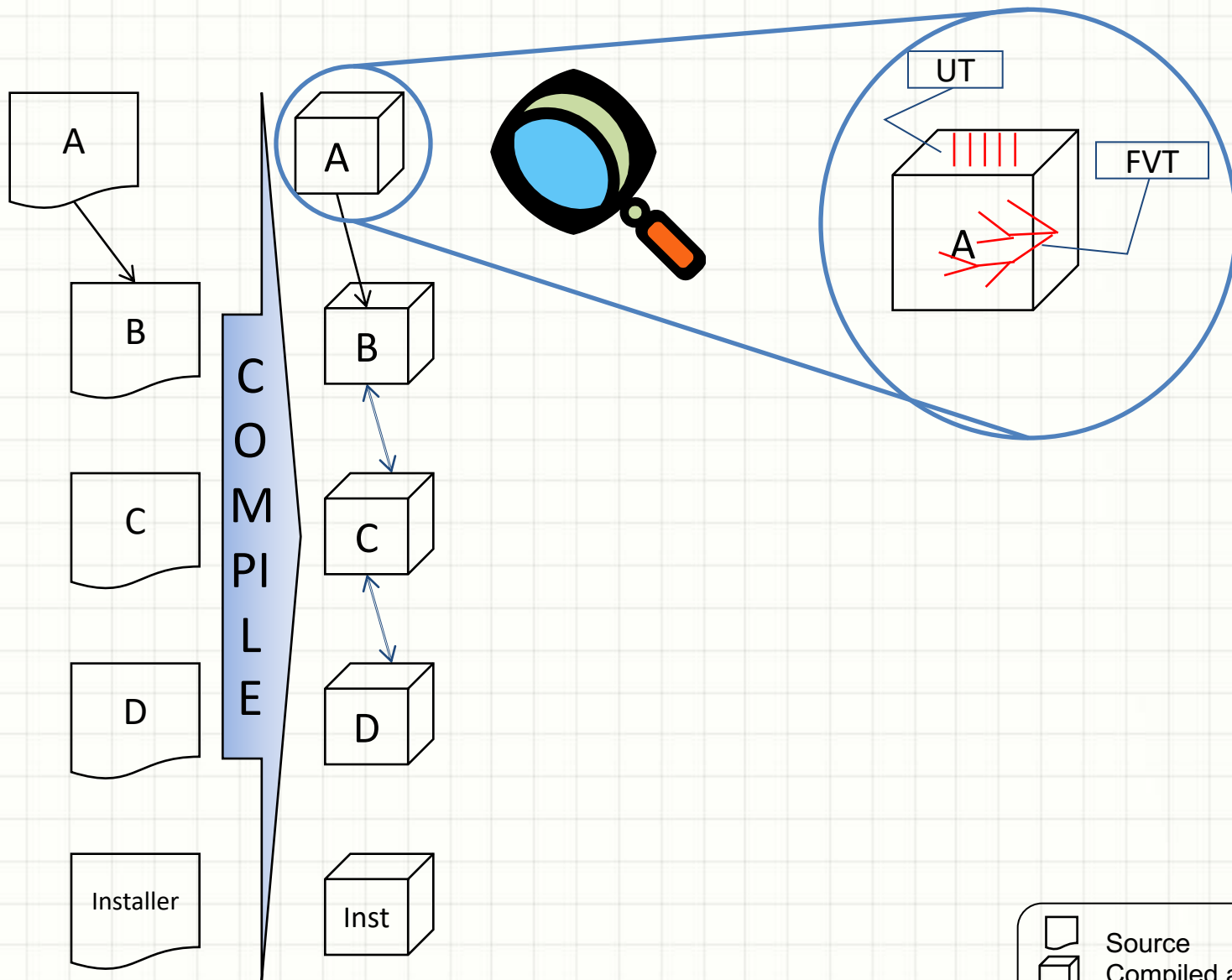


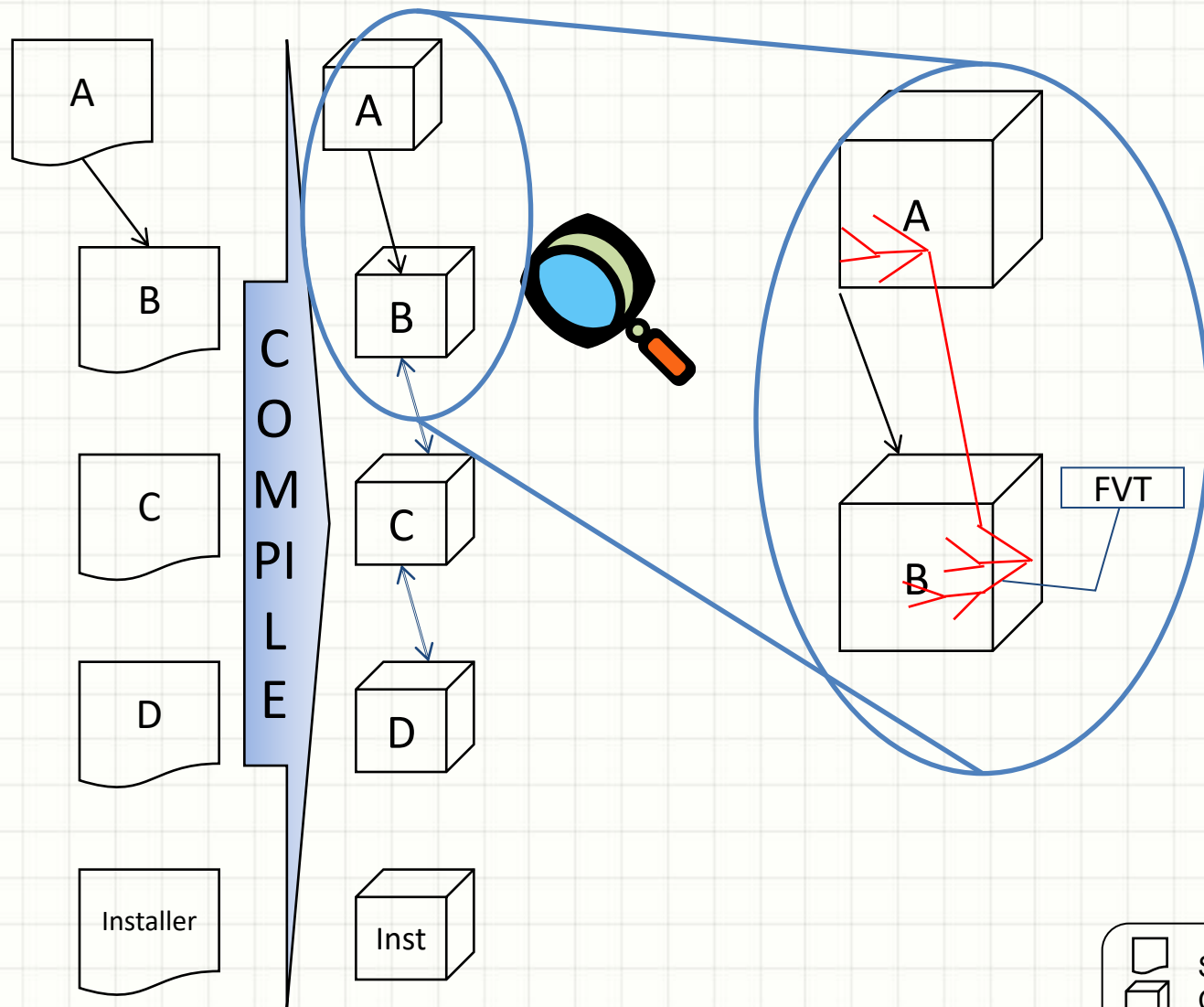
# Functional vs. System

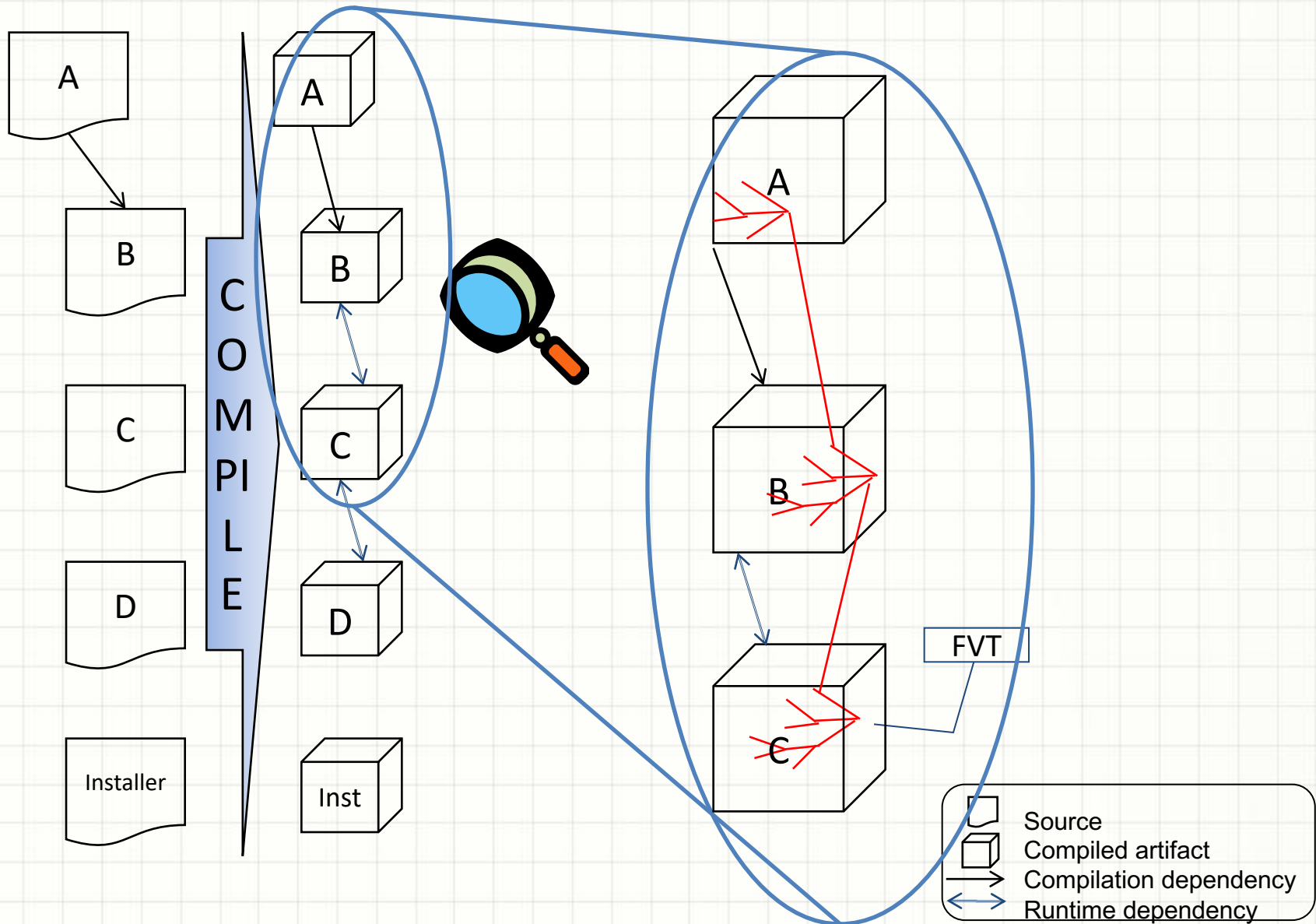
- Some make a big difference between Functional and System testing
- But
- It's just a continuum







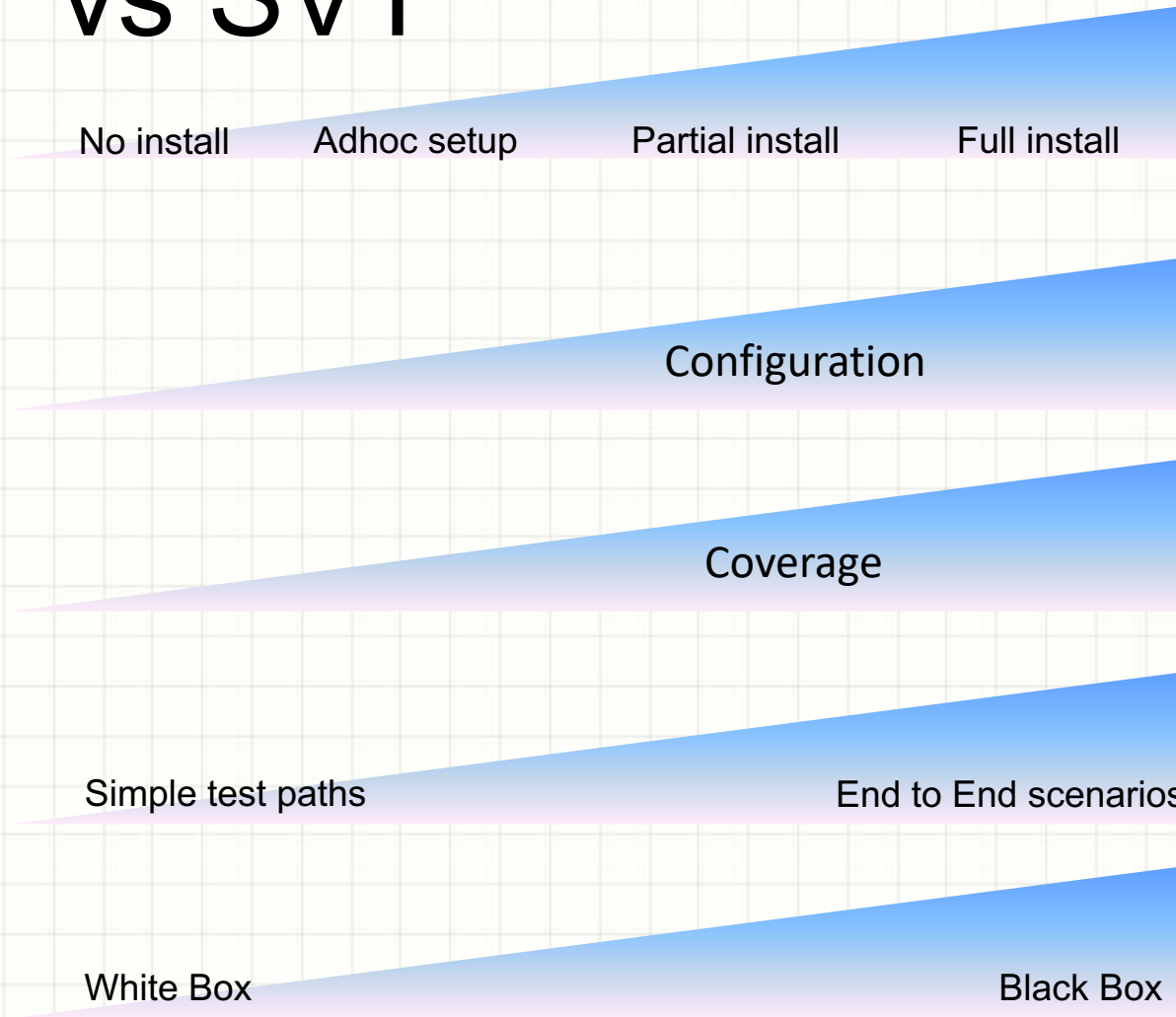


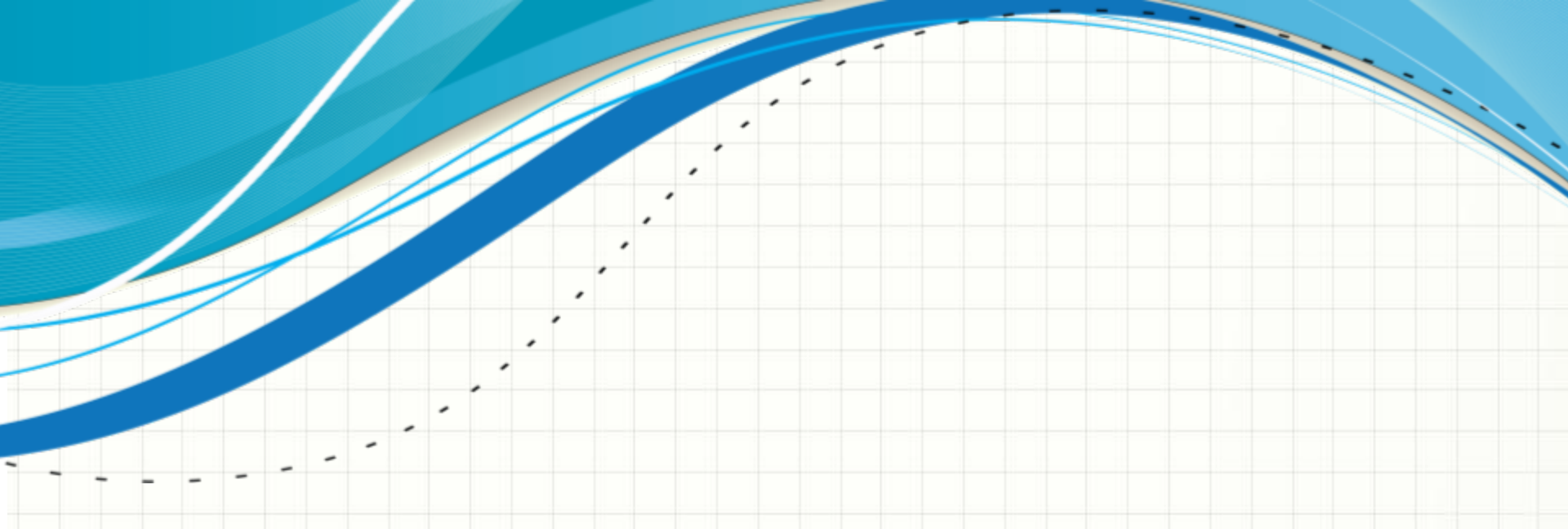


# FVT vs SVT

**FVT**

**SVT**

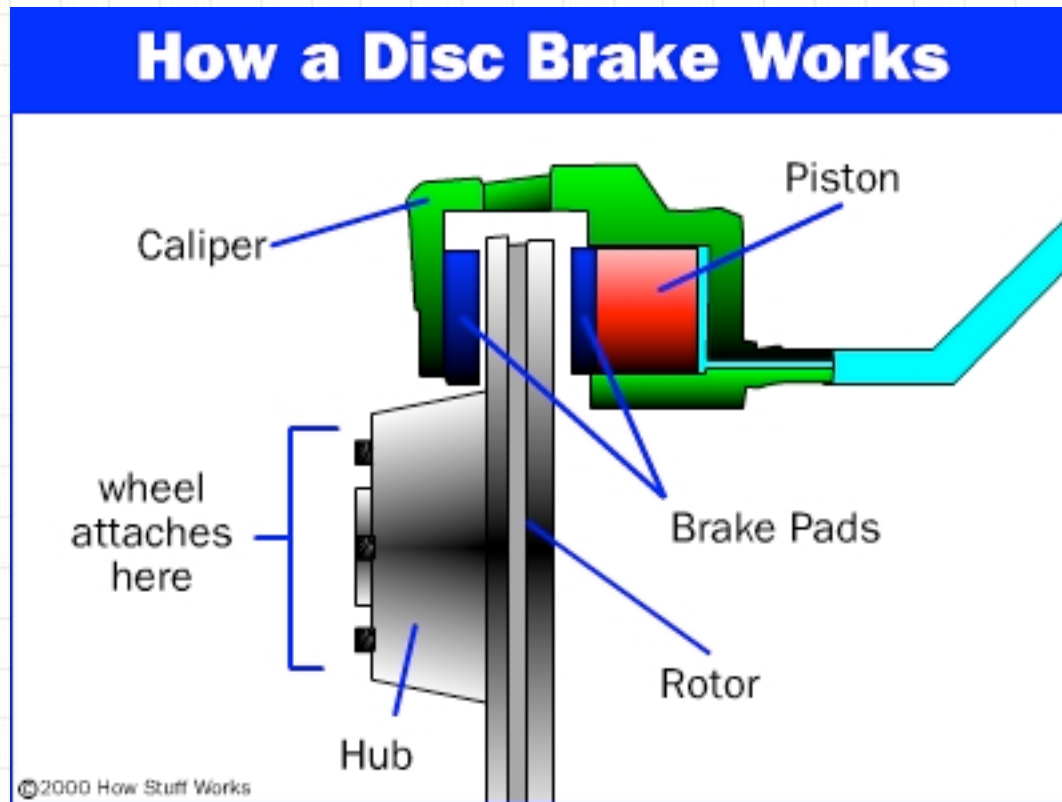




# UNIT-, FUNCTIONAL- & INTEGRATION TESTS



# Unit-Test



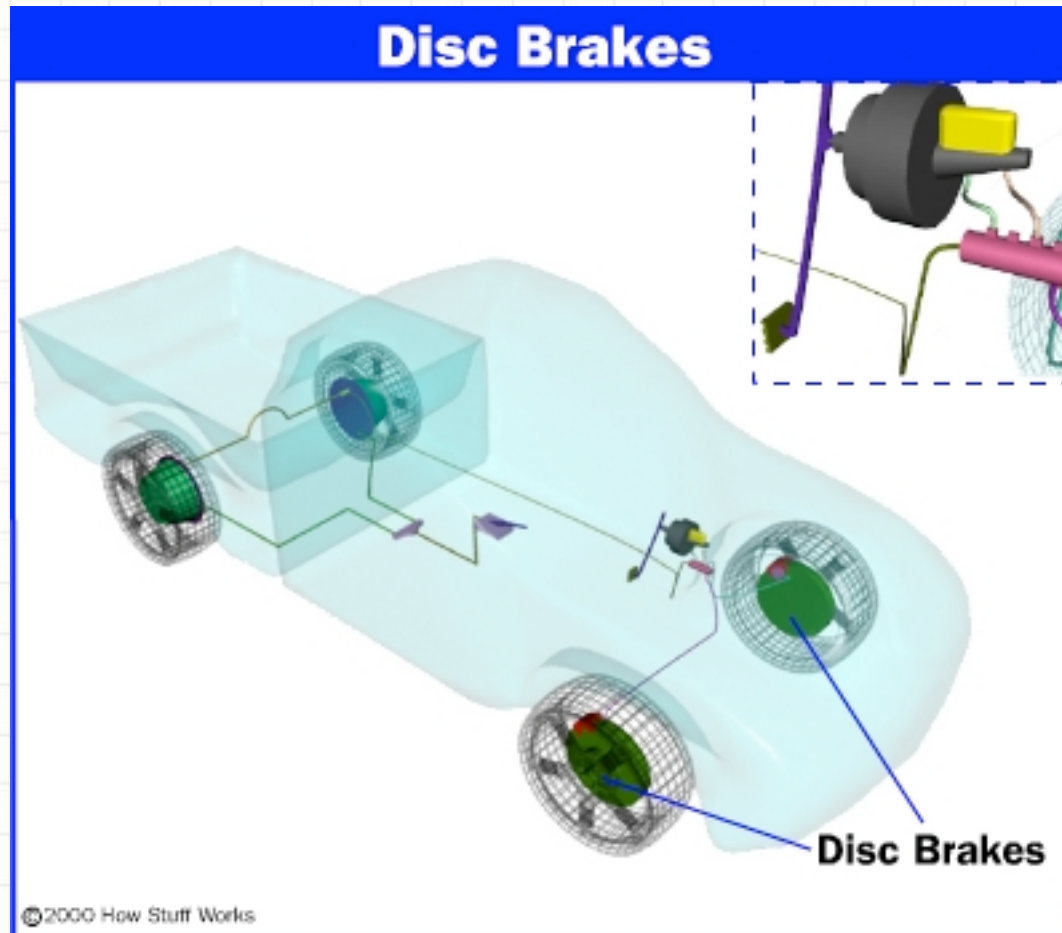
# Unit-Test

Example:

CartModifier add(Customer, Item)

KitchenBean process(Order)

# Functional Test



# Functional Test

Example:

CartWebService

addItemToCustomerCart(String, Item)

# Functional - specs

GIVEN

The Cookie Factory server and its Webservices

The customer Bob

WHEN

Bob calls the CartWebService addItemToCart WS with 3 chocolate cookies

THEN

- Bob's cart is updated (incremented by 3 items)
- TCF inventory is updated (3 cookies removed)
- ...

# Unit vs Functional

Technology <-> Function

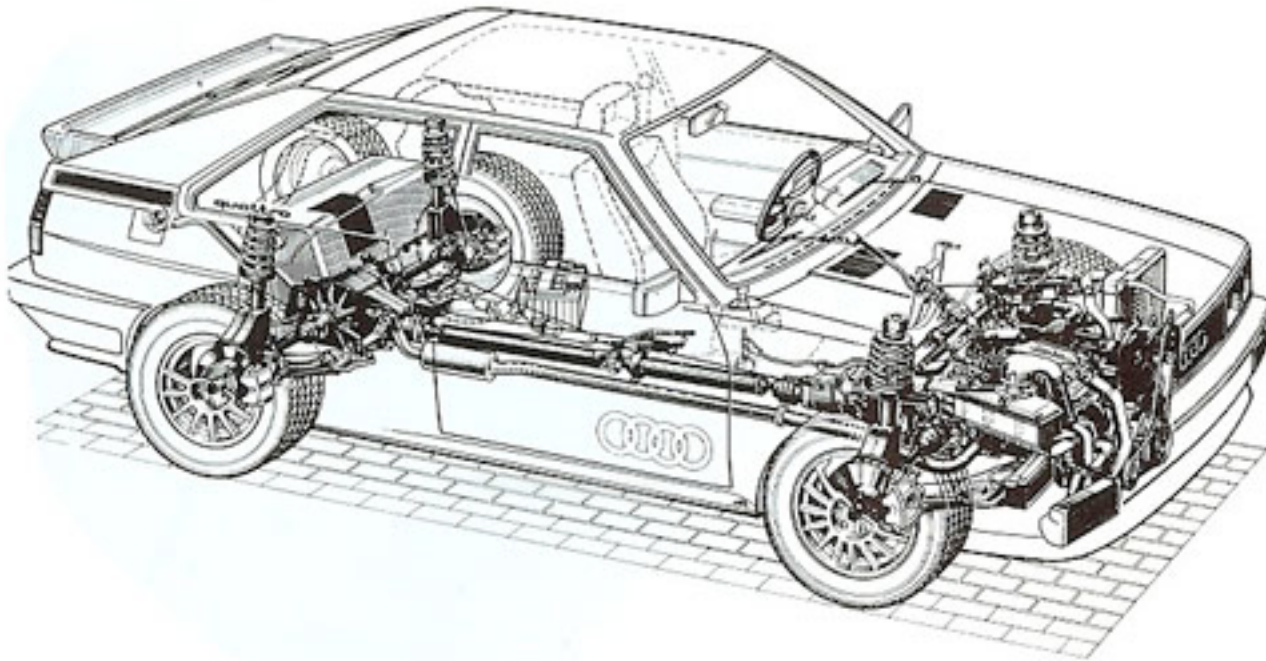
Programmer <-> User

Does it work? <-> Does it implement the function I need?

Small <-> Bigger



# Integration Test



# Integration Test

Example:

CLI

`OrderCookie.execute`



# Integration Test - specs

## GIVEN

- TCF server, connected to “a” (\*) bank system
- The CLI client
- The right sample data (Bob, some cookies...)

## WHEN

- Bob builds a cookie order and calls `OrderCookie.execute`

## THEN

- TCF inventory is updated (3 cookies removed)
- Bob’s cart is updated (added 3 chocolate cookies)
- Bob’s bank accounted is billed

(\*: real or mocked)

# Unit vs Integration



<http://imgur.com/qSN5SFR>  
Another one

# Scenario Testing



# Scenario Testing - specs

## GIVEN

- TCF server, complete
- The CLI client

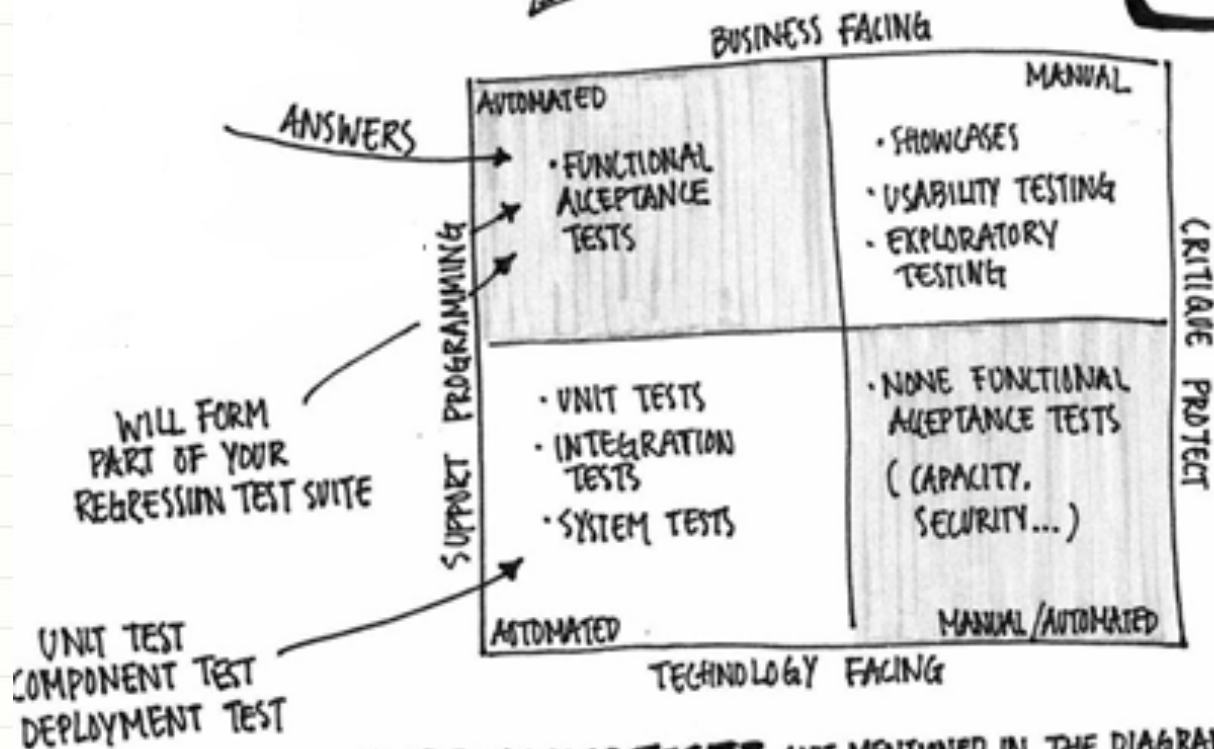
## SCENARIO

- Create a customer Bob through the CustomerCare API
- Add some cookies in the factory
- Bob uses the CLI to order some cookies
- If TCF has enough cookies, fulfill the order and check for appropriate updates
- If TCF is running short on cookies, check for error conditions
- Keep going with more tests
- → What tool would you use?



## TYPE OF TESTS

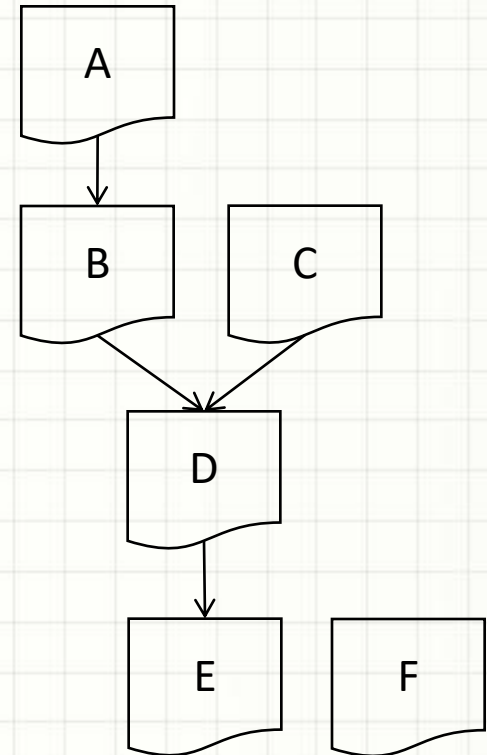
YOUR DEPLOYMENT PIPELINE SHOULD HAVE ALL THESE FOUR TYPE OF TESTS.



**REGRESSION TEST?** NOT MENTIONED IN THE DIAGRAM. THEY ARE CROSSCUTTING CATEGORY.

# Building components

- What does B consume of A?
- Contract
  - API
  - Teams
- Enforced through tests



# WRITING TESTS

# Things to consider

- Start state
- Sample data
- Asserts
- Clean up



# Asserts

```
testSomething {  
    Customer c = customerRegistry.find("Bob");  
    assertNotNull(c, "didn't get Bob");  
    String orderRef = "FR12345";  
    Order o = Util.findOrder(c.getOrders(),  
        orderRef)  
    assertNotNull(o, "didn't get order");  
    PaymentService ps = ...  
    assertNotNull(ps "didn't get PS");  
    boolean status = ps.pay(o, c);  
    assertNotNull(status, "payment didn't work");  
}
```

# Asserts

- Too many or too few asserts?
- Only assert what you're really testing
- But write tests to cover for other aspects
- Each test should have ONE purpose

# More things to consider

- Test maintenance
- Reusability
- → key for functional testing
- → and integration testing
  - Nesting test functions!

## SCENARIO

- Create a customer Bob through the CustomerCare API
- Add some cookies in the factory
- Bob uses the CLI to order some cookies
- If TCF has enough cookies, fulfill the order and check for appropriate updates
- If TCF is running short on cookies, check for error condit
- Keep going with more tests

# Where to run

- Unit
  - Dev command line / dev. Env
  - Build plan, right after compile
  - Prevents artifactory publication
  - Because
    - it's fast
    - It's the contract with dependents

# Where to run

- Functional
  - Dev command line / dev. Env
  - Build plan, dedicated
    - Use profiles
  - Prevents artifactory publication – or not...
  - Not so fast anymore
  - May require to be run in-container
    - Arquillian, Cactus → server-side

# Where to run

- Integration
  - Dedicated env
    - Use VM if possible
  - Separate build plan
  - Long running
  - Requires prod-like env.



**QUESTIONS?**



# Next

- This week:
  - TD: DevOps on TCF Testing
  - TD: Livrair project work





# APPENDIX