System Testing

What, Why & How.

Maximilian Meffert

A Motivational Question

What is the deliverable
of a software development process?

A Motivational Question

What is the deliverable of a software development process?

a) A collection of artifacts

b) UX and behavior

My Personal Answer

What is the deliverable of a software development process?

a) ~~A collection of artifacts~~ **Just the medium!**

b) **UX and behavior**

My Little Testing Glossary

White Box Testing

The investigation from an *internal perspective* whether a program works as expected.

Examines the source code of a program, e.g. control flow, data flow, coverage, etc.

Black Box Testing

The investigation from an *external perspective* whether a program works as expected.

Examines the functionality of a program, i.e. whether it is fit to fulfill its purpose.

My Little Testing Glossary

Unit Testing

The investigation whether an *individual program unit* works as expected.

Examines units independently, i.e. in isolation.

Units may be sets of one or more routines:

- procedures, functions or modules (in Procedural and Functional Programming)
- methods, class or interface signatures (in Object Oriented Programming)

Integration & Integrated Testing

The investigation whether *multiple program units in combination* work as expected.

An informal distinction:

- *Integration Testing* involves third party units.
- *Integrated Testing* does not involve third party units.

System Testing

The investigation whether *all program units in combination* (the entire system) work as expected

My Little Testing Glossary

Acceptance Testing

The investigation whether *all requirements of a specification are met*.

Regression Testing

The investigation whether all requirements of a specification are still met *after a change* was introduced.

My Little Testing Glossary

Testing terminology can be grouped by:

Perspective of the test conductor

- White Box Testing
- Black Box Testing

Properties of the test subject

- Unit Testing
- Integration & Integrated Testing
- System Testing

Purpose of the test

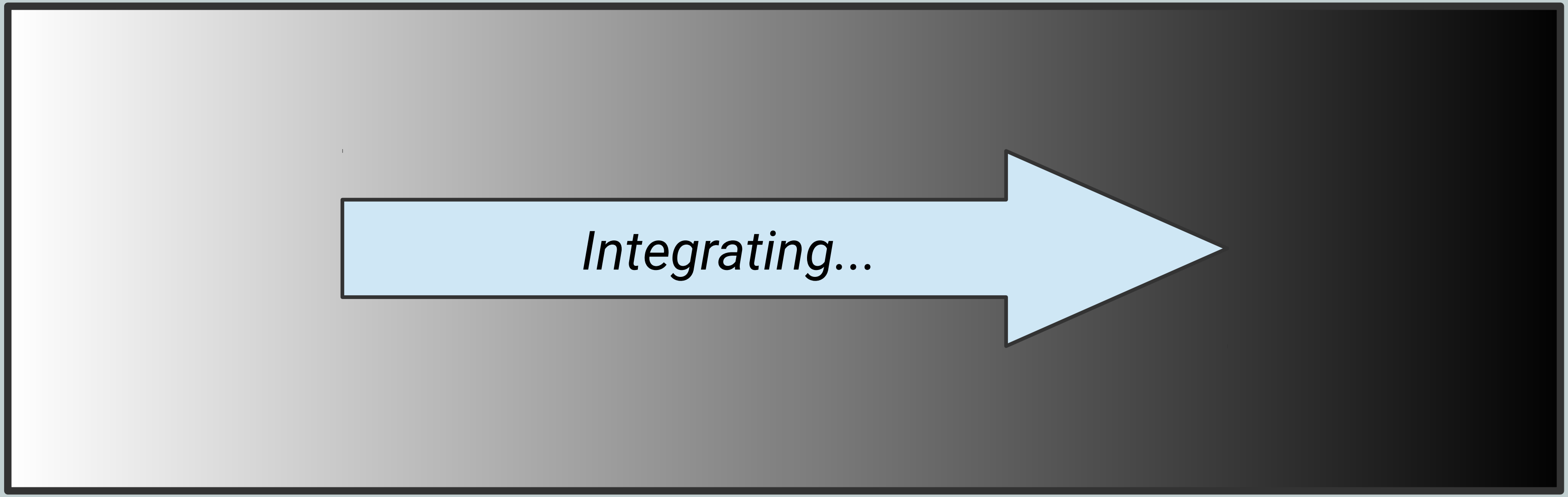
- Acceptance Testing
- Regression Testing

Fifty Shades of Testing

A Rule of Thumb

White Box Testing

Black Box Testing



Unit Testing

Integration & Integrated Testing

System Testing

Behavior Driven Development

An extension of Test Driven Development.

Advocates Acceptance Testing focusing on the behavior of a program.

Tests have a fixed format.

May involve a Domain Specific Language and other tooling for specification.

Behavior Driven Development

```
Scenario: John wants to withdraw money from his bank
account at an ATM
    Given John has a valid Credit or Debit card
    And his account balance is $100
    When he inserts his card
    And withdraws $45
    Then the ATM should return $45
    And his account balance is $55
```

SpecFlow / Cucumber Style Scenario Definition

Behavior Driven Development

Enforced BDD Test Format:

Given

an initial context and/or a set of pre-conditions passes

When

an event occurs or action is executed

Then

a set of post-conditions must pass

Proposed TDD Test Format:

Arrange

all necessary preconditions and inputs

Act

on the unit under test

Assert

that expected results have occurred

Principles of SOLID Design

5 Design Principles for creating understandable and maintainable software:

Single Responsibility Principle

Open/Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Principles of SOLID Design

S

Single Responsibility Principle

A class should only have one responsibility, i.e. *“reason to change”* [Robert C. Martin a.k.a. Uncle Bob].

- A class only establishes one, and only one reason for its existence.
- A class only has only job, and one job alone.
- …

Principles of SOLID Design

O

Open/Closed Principle

“Modules should be both open (for extension) and closed (for modification).” [Bertrand Meyer]

Clients of interface methods or clients of abstract methods of abstract base classes are closed for modification by providing a fix signature, but are still open for extension through the possibility of implementation.

E.g.: Template Methods, Strategies, Commands, Plugin Architectures, etc.

Principles of SOLID Design

L

Liskov Substitution Principle

Given a program P containing a type T and its sub-type S: Let q(P,x) be a provable property in P for all instances x of T, then q(P,y) should be true in P for all instances y of S. [Barbara H. Liskov, Jeannette M. Wing]

x may be substituted with y, hence “Substitution Principle”.

The overall behavior of a program regarding one type should not differ or change for any of its sub-types.

Principles of SOLID Design

I

Interface Segregation Principle

Interfaces should be designed from the perspective of its clients.

An interface only exposes methods a client, i.e. classes using the interface, necessarily needs to know in order to do its job.

Principles of SOLID Design

D

Dependency Inversion Principle

1) *“High-level modules should not depend on low-level modules. Both should depend on abstractions.”*

2) *“Abstractions should not depend on details. Details should depend on abstractions.”*

For instance, business logic should only interact with concrete environment logic or third party logic (e.g. file system API) through abstraction; see Interface Segregation Principle or Open/Closed Principle.

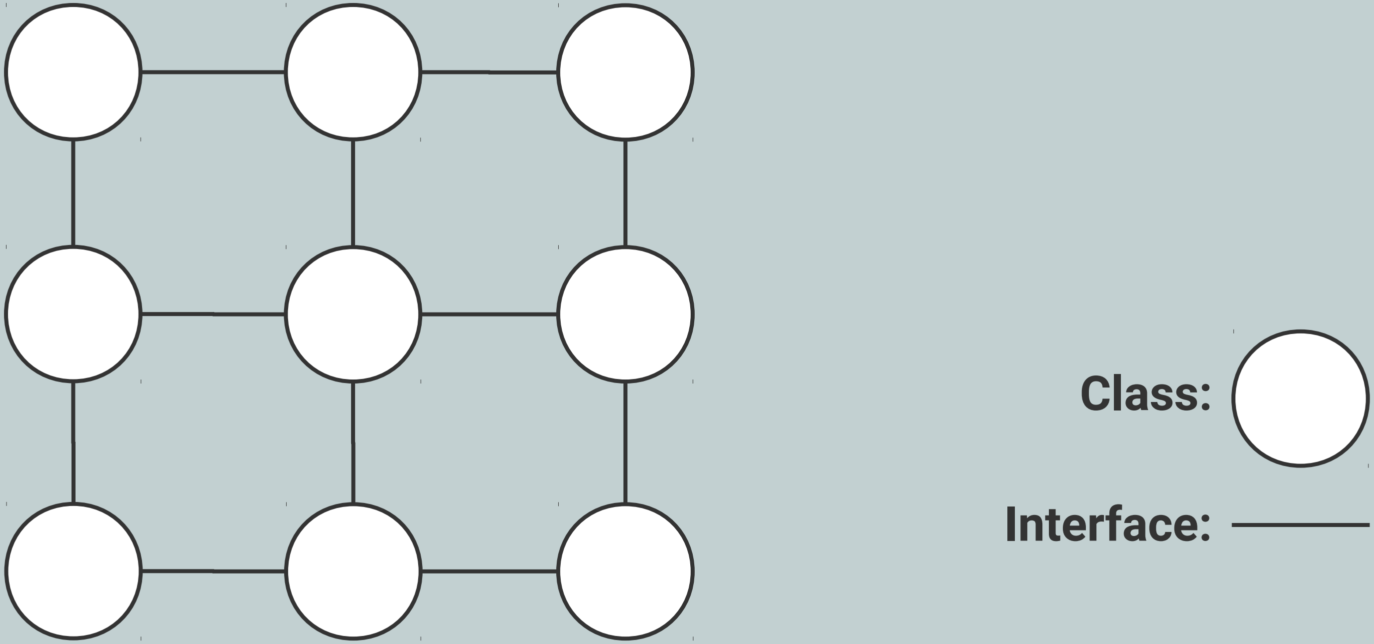
Principles of SOLID Design

Consequences of SOLID Design:

Single Responsibility Principle	→ many relatively small classes
Open/Closed Principle	→ classes may introduce more interfaces
Liskov Substitution Principle	→ beware of inheritance (between classes)
Interface Segregation Principle	→ many specific interfaces
Dependency Inversion Principle	→ interaction through interfaces

Principles of SOLID Design

Idea of ideal Design



Class: ○

Interface: —

Principles of SOLID Design

SOLID Design Principles provide ***quality criteria for*** the internal structure of software, i.e. ***the medium***.

It leads to ***many relatively small and independent units***, which know little to nothing of each other.

Hence, it is ***easy to unit test***.

… so SOLID Design usually is heavily united tested.

A Blind Spot

But, do all these units play well with each other?

A Blind Spot

But, do all these units play well with each other?

We can’t tell from unit tests!