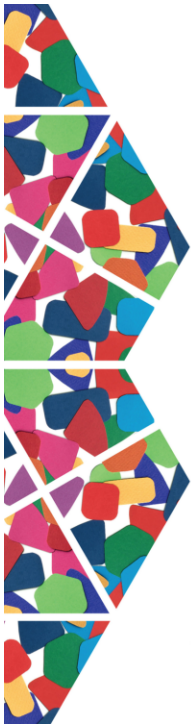


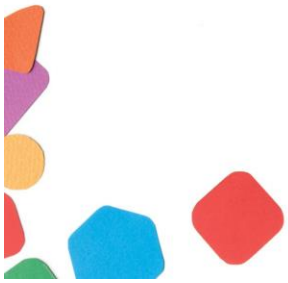
GeneXus Consulting

# TESTING LAB

Gonzalo Mancebo – gmancebo@genexusconsulting.com
Guillermo Skrilec – gskrilec@genexusconsulting.com

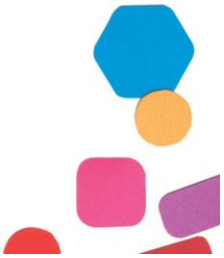
#GX27

# TESTING BEST PRACTICES



**Who likes Testing?**

**Who tests?**



# WHAT IS TESTING?

# EVALUATION OF A PRODUCT THROUGH EXPLORATION AND RESEARCH TO PROVIDE INFORMATION

# POSSIBLE APPROACHES:

# SUPPORT THE TESTING PROCESS

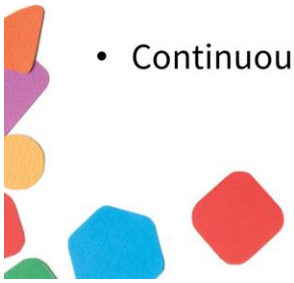
# PERFORM TESTS!

# SUPPORT THE TESTING PROCESS

- Give information about the implementation of the product
- Report known bugs
- Fix the bugs and give feedback
- Reuse of code (to reduce testing effort)
- **Testability**

# PERFORM TESTS!

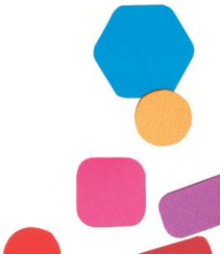
- Unit testing
- Test Driven Development
- Continuous Integration



# UNIT TESTING

# OBJECTIVES

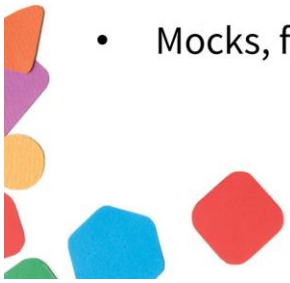
- Test units of the source code
- A unit is the smallest testable part of an application
- Each test case is independent from the others
- Used to ensure that the code behaves as intended



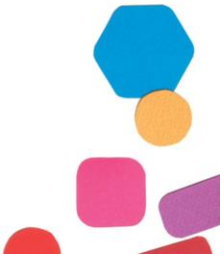
# COMMON PROBLEM

- The units need other elements or programs to run
- The developer needs to model a reality that emulates the behaviors of elements outside the unit
- Examples: data, third-party web services, other classes

# SOLUTIONS



- The unit has to be tested alone so it cannot be affected by other units
- Mocks, fakes or stubs should be created



# UNIT VS INTEGRATION TESTS

**Unit Test**

- Verify that a relatively small piece of code is doing what is intended to do
- Narrow in scope
- Easy to write and execute
- Test useful for the developer
- No dependencies on external components
- External components are mocked or stubbed out
- Test internal consistency

**Integration Test**

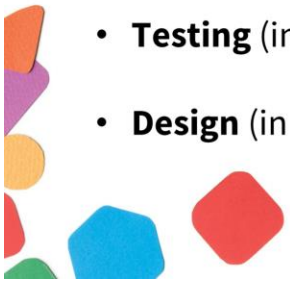
- Verify that different pieces of the system works correctly together
- Covers whole business requirements/features
- Require more resources to complete
- Test useful for business
- Dependencies are required because the holistic approach
- Test consistency between components

# TEST DRIVEN DEVELOPMENT (TDD)

# WHAT IS TDD?

It refers to a style of programming where three activities are tightly interwoven:

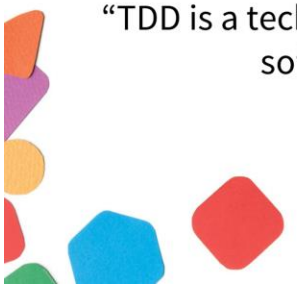
- **Coding**
- **Testing** (in the form of **Unit Testing**)
- **Design** (in the form of **Refactoring**)



# WHAT IS TDD?

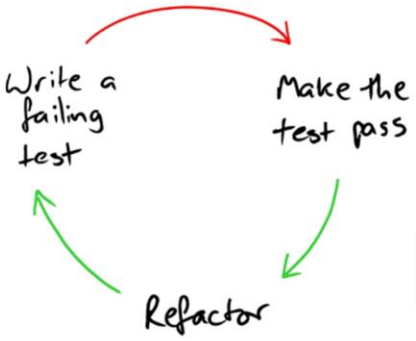
"TDD is a software development approach where tests are written before the funcional code"

"TDD is a technique of using automated unit tests to drive the design of software and **force decoupling of dependencies**"

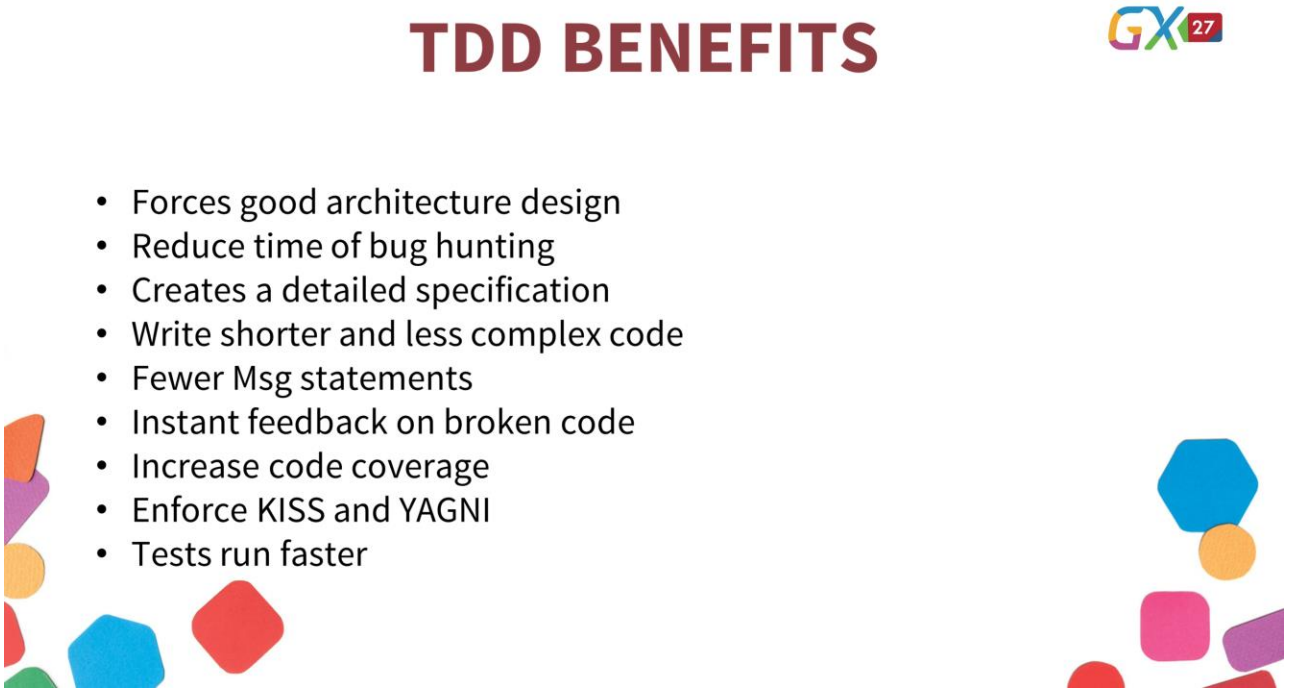


# TDD PROCESS

1. Write a test
2. Run all tests (the new should fail)
3. Write some code
4. Run tests
5. If tests passes, refactor
6. Repeat step 1



# TDD BENEFITS




KISS – Keep it Simple

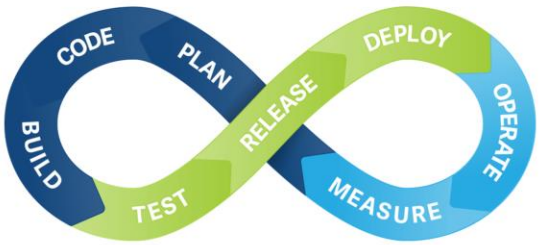
YAGNI – You aren’t gonna need it

# CONTINUOUS INTEGRATION (CI)

# WHAT IS CI?

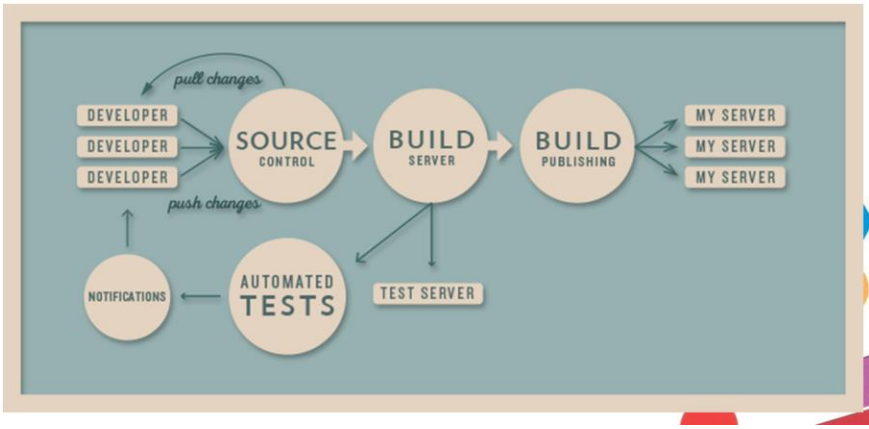
Development practice that requires developers to **integrate** code into a shared repository several times a day.

Each check-in is then verified by an automated build, allowing teams to detect problems early.



# CI PROCESS

1. Get latest version of the code
2. Generate and compile
3. Code review
4. Deploy
5. Run tests



# CI BENEFITS

- Eradication of manual deployment
- Prevention and reduction of staging errors
- Analysis of the health of the code
- Reduce overheads across development and deployment process
- Quality Assurance in early stages

# TESTABILITY

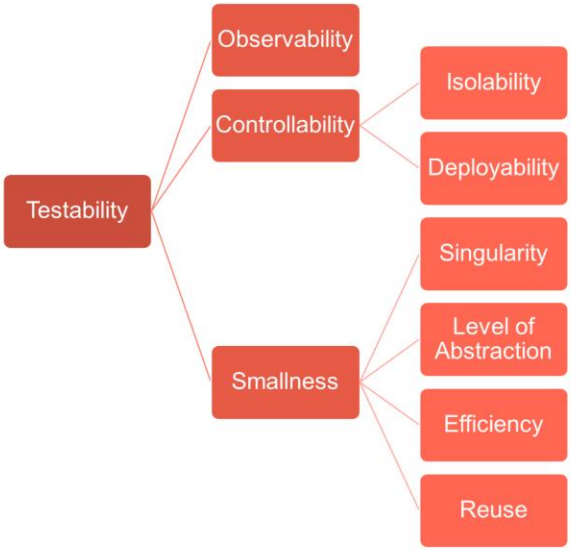
# WHAT IS TESTABILITY?

If testing is **questioning a product in order to evaluate it**, then testability is anything that makes it easier to question or evaluate that product.

So testability is **anything that makes the program faster or easier to test on some level**.

Anything that slows down testing or makes it harder reduces testability, which gives bugs an opportunity to hide for longer, or to conceal themselves better.

# BUILDING TESTABLE APPS



# BUILDING TESTABLE APPS


- Logging of inputs, outputs, or activities within the program. Structure helps; time stamps help; a variety of levels of detail help.
- Real-time monitoring of the internals of the application via another window, a debug port, or output over the network—anything like that. Printers, for example, often come with displays that can tell us something about what's going on.
- Internal consistency checks within the program. For example, if functions depend on network connectivity, and the connection is lost, the application can let us know that instead of simply failing a function.

# BUILDING TESTABLE APPS

- Use of resource files for localization support, rather than hard-coding of location-dependent strings, dialogs, currencies, time formats, and the like.
- Readable and maintainable code, thanks to pairing or other forms of technical review, and to refactoring.
- Restraint in feature support. Very generally, and all other things being equal, the more features, the longer it takes to test a program.

# BUILDING TESTABLE APPS

- Scriptable interfaces to the product, so that we can drive it more easily with automation.
- Overall simplicity and modularity of the application.
- Finally, but perhaps most importantly, an application that's in good shape before the testers get to it.

# PRACTICE TIME

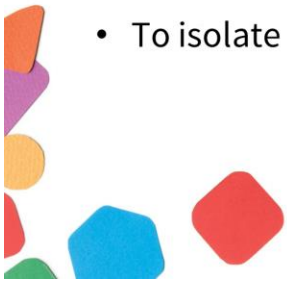
# MOCKING

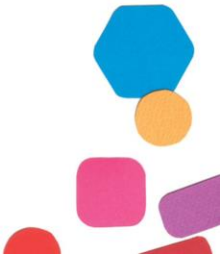
GeneXus Consulting

# OBJECTIVES

GeneXus Consulting

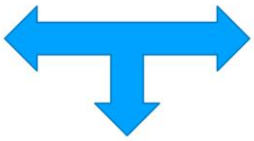
- Have total availability of a web service
- Have a scheduled response
- To isolate a method to be tested through unit test





# HOW?

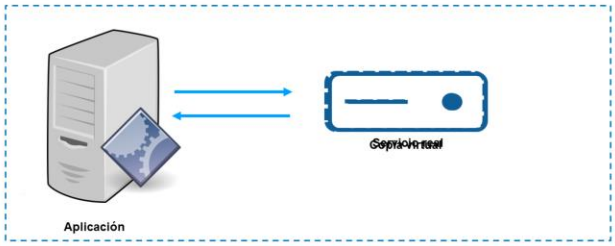
Servicio real



Copia virtual

- Características
- Comportamiento
- Control total

# MOCK TAKES THE PLACE OF REAL SERVICE



# BEST PRACTICES

**Virtualize what you need**

How real will it be?

Create -Expand

Team asset

# MOCKING IS NOT ONLY FOR WEB SERVICES

# NOW PRACTICE TIME