TDD And Algorithm Development

Contents

[Introduction 1](#_Toc47817888)

[Benefits of Testing 1](#_Toc47817889)

[Building Functionality 1](#_Toc47817890)

[Stability through Regression Testing 1](#_Toc47817891)

[The Environment Setup 1](#_Toc47817892)

[Create a Project 1](#_Toc47817893)

[Define the Concepts - Interfaces 1](#_Toc47817894)

[Initial Tests 1](#_Toc47817895)

[Algorithm Development 1](#_Toc47817896)

[Simple Additive Notation 1](#_Toc47817897)

[Subtractive Notation 1](#_Toc47817898)

[Resources 2](#_Toc47817899)

[References 2](#_Toc47817900)

# Introduction

See <https://www.linkedin.com/pulse/test-driven-development-tdd-really-works-donald-trummell-1c>.

I attended an excellent lecture on Test Drive Development given by Mark Shead (see Resource #1), in which he used a “Arabic to Roman Numeral” conversion algorithm to demonstrate how TDD enabled exploring code and algorithm options. He of course started with the TDD mantra of:

* Write a failing test first.
* Write code that makes the test succeed.
* Repeat.

This approach works well then the algorithm being implemented is straight-forward and does not require significant abstraction. Given the purpose of his lecture, this orthodox approach was reasonable.

Often however, algorithms will require significant abstraction. For example, suppose we are defining the notion of distance between two “points”. We might start with a one dimensional point, and compute the distance between two one-dimensional points on the number line:

* D = ABS(X1 – X2)

We write the appropriate tests for one dimension. Next we solve the problem in two dimensions:

* D = SQRT((X1 – X2)^2 + (Y1 – Y2)^2)

Again we write our tests and proceed to the next dimension. Soon we realize that there is a general formula for ***N*** *> 0* dimensions. Now we have to refactor both our solution and tests to take this more general approach. A little research in advance would have made

# Benefits of Testing

## Building Functionality

## Stability through Regression Testing

# The Environment Setup

## Create a Project

## Define the Concepts - Interfaces

## Initial Tests

# Algorithm Development

## Simple Additive Notation

## Subtractive Notation

# Resources

1. Mark Shead (<https://www.linkedin.com/in/markshead/>), president of Xeric Corporation (<http://www.xeric.net/>.)
2. My previous article on TDD: <https://www.linkedin.com/pulse/test-driven-development-tdd-really-works-donald-trummell-1c/>.
3. GitHub repository for Roman Numeral Example: <https://github.com/DonaldET/DemoDev>

# References

1. Wikipedia discussion of Roman Numerals: