27-1-2020

Harvey D. Bastidas C.

INGENI-US eNGINEERING

Singularity

Use case for decentralized trading agent optimization.

# Introduction

This document describes the design and implementation details for an automated forex trading agent that is continuously training in a decentralized network using the singularity platform.

# Problem Description

To find the best way to automate a multi-symbol trading agent while continuously optimizing its performance by comparing 4 automation methods:

* Multi-Timeseries Singular Spectrum Analysis (MSSA) to make a prediction of a trading signal used by the agent.
* DCN/LSTM and MSSA to predict a trading signal.
* Autoencoder for using its decoder as feature extractor.
* NEAT for the dense layer, either using a DCN/LSTM or autoencoder based feature extractor.
* The performance ***X*** of an agent in a dataset ***D*** is measured as the net profit ***p*** divided by the risk ***r***.

*X(D) = p / r*

The profit *p*  is the variation in the balance of an agent in a forex broker account simulator S after evaluating the market data contained in the dataset D. The risk is a weighted average of all orders individual risk.

The individual order risk is the sum of the instantaneous profit in pips in every tick of the order multiplied by the maximum ticks per order divided by the stop loss multiplied by the length of the order in ticks.

***TODO***

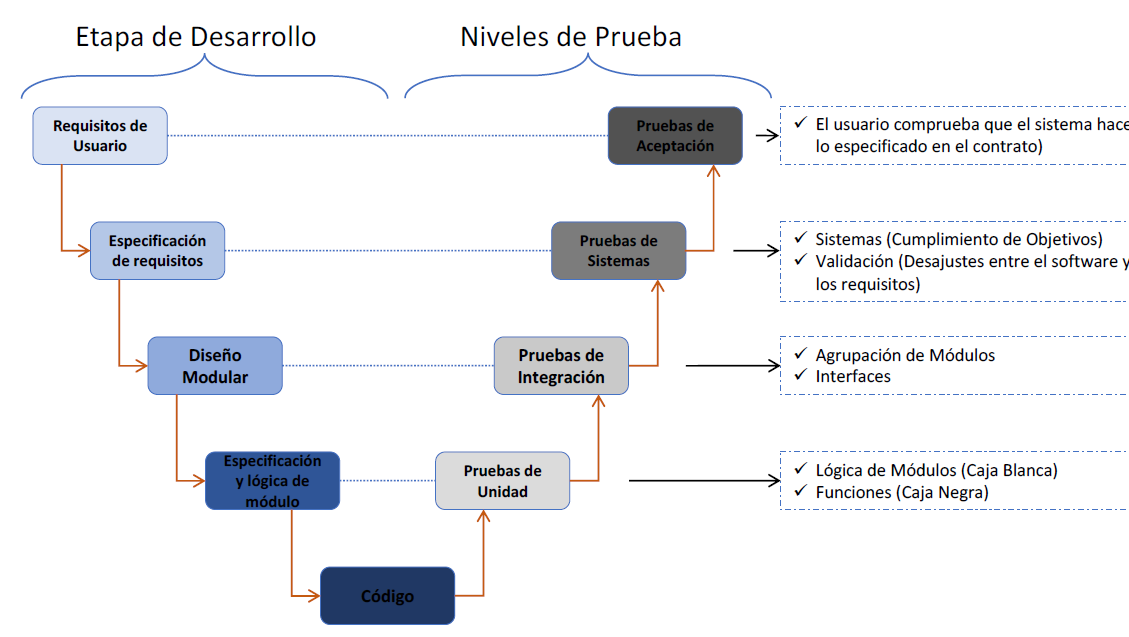
*p, r = S(D)*

# Design Methodology

The design process for the agent is described in this section. This process is based on the Attribute-Driven Design (ADD) iterative design methodology, and the Test-Driven Development (TDD) methodology. The platform design and development process had the following steps:

* Define the stakeholders.
* Define high level architecturally significant requirements.
  1. Define **acceptance tests**.
* Identify the elements that compose the desired system, their behaviors and their interactions.
  1. Define **system tests** (using all components).
* For each of the elements identified:
  1. Identify all the requirements associated with the designed element
  2. Define **integration tests** (for each element’s requirement)
  3. Generate a design solution for the selected element with the following steps:
     1. Select the architectural pattern (AP) that fits the best to its requirement
     2. Select the tactics that are more suitable for implementing the selected AP
     3. Select the technologies used for implementing the selected tactic
     4. Define **unit tests.**
* Setup continuous integration.

The following is a diagram of the described process.



The following is the structure and the lifecycle of the performed tests :

Test case components:

* Input Values
* Pre-conditions in test environment
* Expected results (post-conditions)
* Test execution, instructions or script
* Post-conditions and test results

Test case life cycle:

* Planning and test design
* Data and test environment preparation
* Execution
* Evaluation and log of results
* Acceptation and test end or error notification and request for solution

# Stakeholders

The stakeholders are the users that interact Awith the program before, during and after development. In this case they are:

* **Agent** **owner**: he wants to maximize his profits by trading with an agent while it is being optimized.
* **Administrator:** configures the optimization process.
* **Developer**: he wants to deliver a scalable maintainable optimization platform.

# User Requirements and Acceptance Tests

This section defines the requirements and acceptance tests for each stakeholder and the acceptance alpha and optionally beta tests they will perform to validate the correct behavior of the developed program. The requirements/tests are divided into functional and non-functional.

## Functional Requirements

**Agent Owner:**

* Should be able to install the agent and all its requirements for simulation.
* Should be able to configure the agent.
* Should be able to start, pause and stop multiple agents.
* Should be able to generate and download a report detailing the trading operations.

**Developer**:

* Should be able to update the source code.
* Should be able to test the whole program after an update.

**Administrator:**

* Should be able to add or remove optimization nodes.
* Should be able to use multiple agent automation methods.
* Should be able to perform multiple optimization processes simultaneously.

### FUNCTIONAL TESTS

**Agent Owner:**

* Should be able to install the agent and all its requirements for simulation.
* Should be able to configure the agent.
* Should be able to start, pause and stop multiple agents.
* Should be able to generate and download a report detailing the trading operations.

**Developer**:

* Should be able to update the source code.
* Should be able to test the whole program after an update.

**Administrator:**

* Should be able to add or remove optimization nodes.
* Should be able to use multiple agent automation methods.
* Should be able to perform multiple optimization processes simultaneously.

## Non-Functional Requirements

### Non-FUNCTIONAL TESTS

* Define high level architecturally significant requirements.
  + Define **acceptance tests**.
* Identify the elements that compose the desired system, their behaviors and their interactions.
  + Define **system tests** (using all components).
* For each of the elements identified:
  + Identify all the requirements associated with the designed element
  + Define **integration tests** (for each element’s requirement)
  + Generate a design solution for the selected element with the following steps:
    - Select the architectural pattern (AP) that fits the best to its requirement
    - Select the tactics that are more suitable for implementing the selected AP
    - Select the technologies used for implementing the selected tactic
    - Define **unit tests.**
* Setup continuous integration.