UNIVERSIDAD NACIONAL DE ITAPÚA

FACULTAD DE INGENIERÍA

Ingeniería en Informática

Trabajo Final de Grado

DESARROLLO DE UN SISTEMA PARA LA EJECUCIÓN DISTRIBUIDA DE PRUEBAS UNITARIAS DE APLICACIONES RUBY ON RAILS UTILIZANDO LA LIBRERÍA RSPEC

Rodrigo Jacinto Fernández Ojeda - Leslie Fabiana López Figueredo

Encarnación - Paraguay
2022

UNIVERSIDAD NACIONAL DE ITAPÚA

FACULTAD DE INGENIERÍA

Ingeniería Informática

DESARROLLO DE UN SISTEMA PARA LA EJECUCIÓN DISTRIBUIDA DE PRUEBAS UNITARIAS DE APLICACIONES RUBY ON RAILS UTILIZANDO LA LIBRERÍA RSPEC

Rodrigo Jacinto Fernández Ojeda - Leslie Fabiana López Figueredo

Trabajo Final de Grado presentado a la Facultad de Ingeniería de la Universidad Nacional de Itapúa, cuyo tema fue aprobado por Res. Dec. 049/2019.

Tutor: Prof. Ing: Aldo Miguel Medina Venialgo

Encarnación - Paraguay
2022

UNIVERSIDAD NACIONAL DE ITAPÚA

FACULTAD DE INGENIERÍA

Ingeniería Informática

HOJA DE EVALUACIÓN DE TFG

INTEGRANTES DE LA MESA EXAMINADORA

- _____
- _____
- _____
- _____
- _____

CALIFICACIÓN FINAL: ___ (_____)

ACTA Nº:_____

FECHA: _____

_____ _____

Secretaria General **Decano**

Encarnación - Paraguay
2022

Dedico este trabajo a mi familia y a mi esposo por estar siempre a mi lado, por la confianza y el apoyo incondicional que me brindaron, sin ellos no hubiera sido posible cumplir esta meta.

Leslie López

Dedico este trabajo a mis padres Silvino y Juana por haberme apoyado durante toda la carrera, y vaya que me tomé mi tiempo, y a mi esposa Maga por estar más emocionada que yo por la culminación de este trabajo.

Rodrigo Fernández

AGRADECIMIENTOS

ÍNDICE

TABLA DE FIGURAS

LISTA DE TABLAS

ÍNDICE DE ANEXOS

LISTA DE ABREVIATURAS

ATDD Desarrollo Orientado a Pruebas de Aceptación (Acceptance Test Driven Development)

BDD Desarrollo Guiado por el Comportamiento (Behavior Driven Development)

PU Pruebas Unitarias

PC Computadora Personal (Personal Computer)

TCP Protocolo de Control de Transmisión (Transmission Control Protocol)

TDD Desarrollo Guiado por Pruebas (Test Driven Development)

UDP Protocolo de Datagramas de Usuario (User Datagram Protocol)

SCP Protocolo de Copia Segura (Secure Copy Protocol)

TFG Trabajo Final de Grado

RESUMEN

En la actualidad las pruebas unitarias son una herramienta fundamental para el aseguramiento de la calidad del software. A medida que se avanza en el desarrollo de un software, la cantidad y complejidad de las pruebas unitarias aumentan y los recursos hardware del desarrollador son casi siempre las mismas que al inicio del proyecto, esto podría incidir negativamente en el tiempo de ejecución del conjunto de pruebas unitarias.

En el presente trabajo final de grado se desarrolló un sistema que permitirá al desarrollador de software, ejecutar pruebas unitarias de código para aplicaciones desarrolladas en Ruby on Rails utilizando la librería RSpec, el sistema soporta una arquitectura distribuida, es decir, la carga de trabajo que conlleva la ejecución de estas pruebas será distribuida en una red de computadoras con el fin de descentralizar la carga de trabajo que conlleva la ejecución de las pruebas unitarias, aumentando la velocidad de ejecución y reduciéndose el tiempo requerido para la obtención de resultados de la ejecución de las pruebas unitarias.

El sistema desarrollado está compuesto por dos aplicaciones, una Aplicación Coordinadora y una Aplicación Agente. La Aplicación Coordinadora se encarga de coordinar el proceso de ejecución del conjunto de pruebas unitarias y del procesamiento de los resultados de esta ejecución. La Aplicación Agente es un servicio en segundo plano, que puede ejecutarse en varias computadoras dentro de una red, y se encarga de ejecutar las pruebas unitarias y retornar los resultados a la Aplicación Coordinadora.

Durante su ejecución, la Aplicación Agente inicia un servidor UDP que se mantiene en espera, mientras que, la Aplicación Coordinadora inicia un cliente UDP y un servidor TCP. El cliente UDP de la Aplicación Coordinadora emite peticiones de difusión (Broadcast) a la red, estas peticiones son procesadas por el servidor UDP de las Aplicaciones Agente que se encuentren en ejecución. Cuando una Aplicación Agente recibe la petición UDP, procede a realizar dos operaciones principales: primero, se conecta a través del protocolo SCP a la computadora donde está ejecutándose la Aplicación Coordinadora y obtiene el código fuente de la aplicación cuyas pruebas unitarias se quiere ejecutar; segundo, inicia un cliente TCP que se conecta con el servidor TCP que se encuentra ejecutándose en la Aplicación Coordinadora, y mediante esta conexión se procede a coordinar la ejecución del conjunto de pruebas unitarias.

A partir de las pruebas llevadas a cabo sobre el sistema implementado, se logró mejorar el tiempo de ejecución de un conjunto de pruebas unitarias usando la arquitectura distribuida propuesta, en comparación a la ejecución de este mismo conjunto de pruebas en una sola computadora.

Palabras claves: Pruebas Unitarias, RSpec, Arquitectura Distribuida, Aplicación Coordinadora, Aplicación Agente.

ABSTRACT

Currently, unit tests are a fundamental tool for software quality assurance. As software development progresses, the number and complexity of unit tests increase and the developer's hardware resources are always the same as at the beginning of the project, this could negatively affect the execution time of the set of tests.

In this final degree project a system was presented that allowed the software developer to execute unit code tests for applications developed in Ruby on Rails using the RSpec library, the system supported by a distributed architecture, that is the workload that entails the execution of these tests will be distributed in a computer network in order to decentralize the workload involved in the execution of the unit tests, increasing the execution speed and reducing the time required to obtain the results of the execution of the unit tests.

To implement the System, different tools were used, such as TCP and UDP sockets for the automatic detection and connection between the different agent machines and the main master machine, the SCP protocol for sending the project in compressed form (zip) and the Rspec gem for running unit tests.

Keywords: Unit tests, Rspec, distributed architecture, agents, master.

1 INTRODUCCIÓN

1.1 PROBLEMÁTICA

> El software se ha convertido en un elemento ubicuo en el actual mundo digital. Esto quiere decir que está presente en todos los aspectos de la vida humana. (...). Desde el punto de vista de la sociedad, el software provee flexibilidad, inteligencia y seguridad a todos los sistemas complejos y equipos que soportan y controlan las diferentes infraestructuras claves de nuestra sociedad como son los: transportes, comunicaciones, energía, industria, negocios, gobierno, salud, entretenimiento, etc. (Tavarez, 2014)

Para desarrollar un software, el proceso de desarrollo en la industria del software debe ser más dinámico y adaptable para hacer frente a las complejidades de la actualidad. Es por eso que desde los años 60 se han propuesto y aplicado varios modelos Ciclo de Vida de Desarrollo de Software (SDLC, del inglés Software Development Life Cycle) para lograr una mejor situación de desarrollo y éxito económico de las mismas. El SDLC representa toda la vida del proceso en función de la especificación, el diseño, la validación y la evolución del software (Chowdhury, Bhowmik, Hasan, y Rahim, 2017).

Según Chowdhury, Bhowmik, Hasan, y Rahim (2017), existen varios modelos de SDLC y un modelo típico de proceso de desarrollo de software tiene generalmente varias etapas como lo son :

- Planificación y recopilación de información;
- Análisis y definición de requisitos;
- Diseño y definición de la arquitectura;
- Desarrollo;
- Pruebas según los requisitos y la perspectiva técnica;
- Despliegue y mantenimiento

Berzal (2004) menciona que las etapas mencionadas anteriormente son:

> un reflejo del proceso que se sigue a la hora de resolver cualquier tipo de problema. (...). Básicamente, resolver un problema requiere: Comprender el problema (análisis). Plantear una posible solución, considerando soluciones alternativas (diseño). Llevar a cabo la solución planteada (implementación). Comprobar que el resultado obtenido es correcto (pruebas).

Las etapas adicionales a éstas como lo son la planificación, despliegue y mantenimiento son necesarias para el desarrollo de un software porque la misma conlleva unos costos asociados, por lo que se hace necesaria la planificación y lo que se pretende una vez construido el software, es que éste debería poder utilizarse (despliegue y mantenimiento) (Berzal, 2004).

Es importante resaltar que durante el proceso de desarrollo de software, continuamente se realizan pruebas con el objetivo de verificar el correcto funcionamiento del software. Según Gómez, Jústiz y Delgado (2013) las pruebas contribuyen a la calidad del software y estas pruebas deben comenzar desde el momento en el que el desarrollador inicia la implementación del software y deben finalizar antes del despliegue del mismo.

Los principales tipos de pruebas que se pueden efectuar en cualquier tipo de software son: pruebas unitarias, prueba de integración, pruebas de regresión, pruebas de humo y pruebas del sistema (Pauta Ayabaca y Moscoso Bernal, 2017).

En la etapa de las pruebas de integración se combinan las diferentes unidades o componentes probados para formar un subsistema que funcione. A pesar de que una unidad ha superado con éxito una prueba unitaria, aún puede comportarse de manera impredecible al interactuar con otros componentes del sistema. Por lo tanto, el objetivo de las pruebas de integración es garantizar la correcta interacción e interconexión entre las unidades en un sistema de software, tal y como se define en las especificaciones de diseño detalladas (Oladimeji, 2007).

Las pruebas de regresión se llevan a cabo cada vez que se modifica el sistema, ya sea agregando nuevos componentes o corrigiendo errores. Su objetivo es determinar si la modificación del sistema ha introducido nuevos errores en el sistema (Oladimeji, 2007).

El objetivo de las pruebas de humo es determinar si la nueva compilación de software es estable o no, de modo que el equipo de control de calidad pueda usarla para realizar pruebas detalladas y el equipo de desarrollo pueda seguir trabajando. La prueba de humo se lleva a cabo para garantizar que las funciones más cruciales de un programa funcionen, pero sin preocuparse por los detalles más finos (Chauhan, 2014).

Las pruebas del sistema comienzan después de completar las pruebas de integración. Las mismas se realizan para demostrar que la implementación del sistema no cumple con la especificación de requisitos del sistema (Oladimeji, 2007). Las pruebas del sistema están enfocadas directamente en los requisitos de negocio, verificando el ingreso, procesamiento, recuperación de los datos y la implementación propiamente dicha (Pauta Ayabaca y Moscoso Bernal, 2017).

La prueba unitaria enfoca los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente o módulo de software (Pressman, 2010).

Las pruebas unitarias normalmente son el primer nivel de prueba de un sistema de software y están motivadas por el hecho de que el costo de encontrar y corregir errores en el momento de la prueba de la unidad, es menor que encontrar y corregir errores que se encuentran durante los otros tipos de pruebas mencionados anteriormente o después del despliegue, las pruebas unitarias facilitan las pruebas de regresión cuando el software cambia porque permiten a los desarrolladores verificar que no hayan dañado la funcionalidad existente (Ganesan et al., 2013).

La popularidad de las pruebas unitarias ha ido aumentando a medida que fueron surgiendo más librerías de testeo, para los distintos lenguajes de programación existentes (Athanasiou, Nugroho, Visser y Zaidman, 2014). En la Tabla 1 se presentan algunas librerías que

permiten implementar pruebas unitarias para ciertos lenguajes de programación, a estas librerías también se le denominan frameworks.

	Librerías de Testeo
Java	Junit5, TestNG
.Net	NUnit
Ruby	UnitTest, Rspec, RubyUnit, MiniTest
Phyton	Unitest, PyUnit, doctest
PHP	SimpleTest, PHPUnit
Javascript	Qunit, Mocha, Jasmine, Chai

Tabla 1: *Librerías de testeo unitario existentes para ciertos lenguajes.*
Elaboración propia.

En nuestra experiencia, cuando el desarrollador requiere ejecutar todas las pruebas unitarias de manera local, el tiempo que lleva la ejecución de estas pruebas unitarias está limitado por las características de procesamiento de su computadora. Es por ello que a medida que va creciendo el software, aumentan la cantidad de pruebas unitarias, por ende, el tiempo de ejecución de las pruebas unitarias también aumentan.

En la Tabla 2 se presenta una comparativa de algunas herramientas de integración continua que mejoran la velocidad de ejecución de pruebas unitarias. Dichas herramientas consisten en la fusión del trabajo de los desarrolladores, permitiendo mejorar la calidad del software (Fitzgerald y Stol, 2017, citado por Shahin, Ali Babar y Zhu, 2017). Esta práctica incluye la ejecución automatizada de las pruebas de software (Leppänen et al., 2015, citado por Shahin et al., 2017).

Como se mencionó anteriormente la integración continua mejora la velocidad de la ejecución de las pruebas unitarias, y ésto lo logra distribuyendo las mismas a través de varias computadoras interconectadas, de esta manera comparten la carga de trabajo y reducen el tiempo de ejecución. Estas herramientas dependen de un repositorio para la obtención del código fuente sobre el cual ejecutar las pruebas, por lo que no soportan la ejecución de

pruebas sobre el código del desarrollador, es decir, el código sobre el cual está trabajando el desarrollador en su computadora local.

	Interfaz	Lenguajes Soportados	Instalación	Licencia
Jenkins	Web	Varios	Linux, Windows, OS X	MIT, Código Abierto
TeamCity	Web	Varios	Linux, Windows, OS X	Comercial, gratis bajo ciertas condiciones
Travis CI	Web	Varios	Servicio en la nube	Comercial, gratis bajo ciertas condiciones
Circle CI	Web	Varios	Servicio en la nube y servidor privado en Linux u OS X	Comercial
Codeship	Web	Varios	Servicio en la nube	Comercial
Dist Test	Web	Java	Linux	Apache License 2.0

Tabla 2: *Comparativa de herramientas que soportan la ejecución distribuida de pruebas unitarias*. Elaboración propia.

Según Wang (2016), cuando la ejecución de las pruebas unitarias toma mucho tiempo se generan los siguientes problemas:

- La productividad del desarrollador se ve afectada porque tiene que esperar la ejecución de todas las pruebas antes de continuar su trabajo.
- El desarrollador se vuelve reacio a agregar más pruebas porque incrementa el tiempo de espera para la ejecución de todas las pruebas.
- La calidad del software baja porque se dejan de escribir pruebas.
- El desarrollador pierde tiempo actualizando las pruebas tratando de disminuir su tiempo de ejecución.
- El desarrollador termina considerando a las pruebas más como una carga que una ventaja adicional para mejorar la calidad.

En 2016, Wang desarrolló un framework denominado Dist Test, orientado al desarrollador, que permite la ejecución distribuida de pruebas unitarias para software desarrollado en el lenguaje Java (Wang, 2016).

Para contrarrestar los problemas citados anteriormente, se propuso el desarrollo de un Sistema similar al mencionado por Wang que dé soporte a software desarrollado en lenguaje Ruby, enfocándose específicamente en la librería de testeo RSpec para Ruby on Rails.

1.2 OBJETIVOS

1.2.1 General

Desarrollar un Sistema que permita reducir el tiempo de ejecución de pruebas unitarias de aplicaciones Ruby on Rails utilizando la librería RSpec.

1.2.2 Específicos

- Analizar y definir los requerimientos del Sistema.
- Implementar el algoritmo de distribución de pruebas unitarias.
- Implementar el algoritmo de distribución de código fuente.
- Configurar el entorno de prueba para el Sistema.
- Implementar las pruebas unitarias que se usarán para evaluar el Sistema.
- Evaluar los resultados obtenidos de la ejecución del Sistema.

1.3 ALCANCE DEL PROYECTO

Se desarrolló un sistema, que permite al desarrollador ejecutar pruebas unitarias de código para aplicaciones Ruby on Rails utilizando la librería RSpec. La ejecución de pruebas unitarias se realiza utilizando una arquitectura distribuida, es decir, la carga de trabajo que conlleva la ejecución de estas pruebas será distribuida en una red de computadoras.

El sistema puede ser usado desde una línea de comandos. Una vez iniciado el proceso de ejecución de pruebas unitarias sobre el código del desarrollador, se irán mostrando los resultados en la interfaz de línea de comandos, indicando las pruebas ejecutadas con éxito o que hayan fallado.

El sistema puede obtener el código fuente sobre el cual se ejecutarán las pruebas unitarias desde dos ubicaciones posibles, una de ellas es la computadora del desarrollador y la otra será desde un repositorio Git.

El sistema soporta la ejecución de pruebas unitarias que requieren conexión a base de datos. Al iniciar un proceso de ejecución de pruebas se crea una base de datos de prueba y al finalizar el proceso la base de datos será eliminada, cada proceso trabaja sobre una base de datos diferente.

El sistema está compuesto por dos aplicaciones, la primera aplicación a la que denominamos Aplicación Coordinadora, se encarga de coordinar el proceso de ejecución de las pruebas, y mostrar los resultados, la segunda aplicación a la que denominamos Aplicación Agente, es un servicio en segundo plano que se encarga de la ejecución de las pruebas bajo las órdenes de la Aplicación Coordinadora. Para que un desarrollador pueda iniciar la ejecución de sus pruebas, debe tener instalado la Aplicación Coordinadora.

En la Figura 1 se presenta una arquitectura compuesta por cuatro computadoras, la Aplicación Coordinadora instalada en PC1 coordinará la ejecución de las pruebas, comunicándose con las Aplicaciones Agentes, las cuales estarán ejecutándose como un servicio dentro de todas las computadoras de la red, eventualmente, las computadoras que componen la red pueden tener instalada una o ambas aplicaciones.

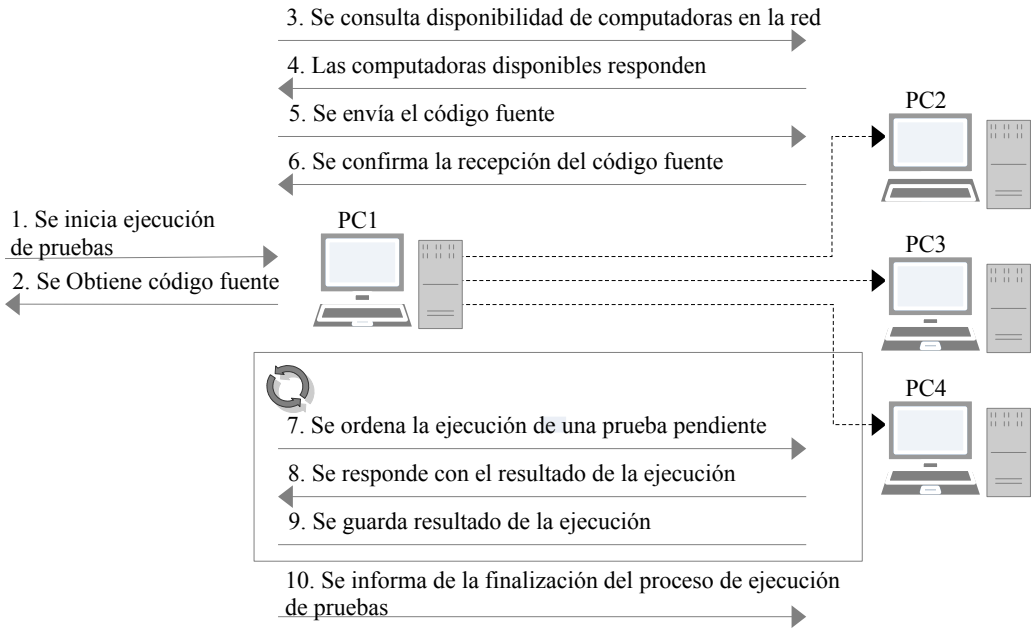

Figura 1: *Prototipo de la Arquitectura del Sistema Implementado.*
Elaboración propia.

1.4 JUSTIFICACIÓN

En grandes proyectos, la ejecución de todas las pruebas después de cada cambio puede convertirse en una operación costosa que requiere varias horas. (Blondeau et al., 2017).

Según Blondeau et al. (2017), en una importante empresa de TI, se encontraron proyectos con un entorno tan complejo y con tantas pruebas que se necesitaban horas para ejecutarlas todas, por este motivo los desarrolladores no se animaban a realizar pruebas periódicas de cada modificación si esto implicaba obtener la respuesta horas después.

A pesar de que los desarrolladores deberían lanzar las pruebas localmente para evitar cometer posibles errores y propagarlos a sus colegas, tienden a delegar esta validación a la integración continua. En consecuencia, se realizan menos pruebas a nivel local y se envían los posibles errores a los demás desarrolladores del equipo (Blondeau et al., 2017).

En nuestra experiencia, a medida que se avanza en el desarrollo de un software, la cantidad y complejidad de las pruebas unitarias aumenta y la potencia de la computadora del desarrollador permanece constante, incidiendo negativamente en el tiempo de ejecución de las pruebas, haciendo que el desarrollador se vuelva reacio a ejecutar las pruebas localmente y delegando la ejecución de las mismas a la herramienta de integración continua.

Con el objetivo de reducir el tiempo de ejecución de pruebas unitarias de código, el Sistema desarrollado en el marco de este TFG tendrá las siguientes ventajas:

- Disminuirá el tiempo de ejecución de pruebas unitarias dividiendo la carga de trabajo entre varias computadoras interconectadas de este modo el tiempo que toma la ejecución de las pruebas unitarias de código no estará limitado por la potencia de la computadora del desarrollador.

- Permitirá la ejecución de pruebas unitarias sobre el código fuente descargado del repositorio, así también sobre el código del desarrollador utilizando la misma arquitectura distribuida mencionada en el punto anterior.

• La cantidad de pruebas unitarias a ejecutar en cada computadora involucrada será determinada de manera automática, en comparación a las herramientas de integración continua existentes que requieren la determinación manual de la cantidad de pruebas a ejecutar en cada computadora cuando realizan una ejecución distribuida.