COMPARACIÓN MÉTODOS DE VERIFICACIÓN DE SOFTWARE

APLICACIÓN A UNA MÁQUINA TOSTADORA DE CAFÉ AUTOMÁTICA

TESIS DE MAESTRÍA EN INGENIERÍA MECÁNICA *MSc*

SERGIO ALEJANDRO NARANJO CASALLAS

INGENIERO MECATRÓNICO

sa.naranjo11@uniandes.edu.co

ASESOR DE TESIS

GIACOMO BARBIERI

PH.D. INGENIERO MECÁNICO

g.barbieri@uniandes.edu.co

UNIVERSIDAD DE LOS ANDES, COLOMBIA

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA MECÁNICA

BOGOTÁ D.C, DICIEMBRE DE 2017

Abstract

At present, machine design techniques are being implemented virtually, trying to reduce the time, costs, and quality of the products. For this, simulation tools are used in the different stages of development. An integrated simulation of the system requires not only the dynamic behavior of the production system, but also to test the operating situations and possible failures to which the device is subjected. This thesis project compares two verification techniques for the control of a coffee roaster, from the design of the same to the testing of different situations to which it is subjected, implementing the same control in two programming languages, Arduino and PLC, for a physical prototype and a virtual prototype respectively. A physical prototype was designed and this was used for the simulation. For the modeling of the control the UML language was used, and the RUP methodology for software development.

Keywords: Arduino, PLC, UML, virtual commissioning.

Resumen

En la actualidad se están implementando técnicas de diseño de máquinas de forma virtual, intentando reducir los tiempos y costos, y fortaleciendo la calidad de los productos. Para esto se utilizan herramientas de simulación en las diferentes fases del desarrollo. Una simulación integrada del sistema requiere no sólo el comportamiento dinámico del sistema de producción, sino también probar las situaciones de funcionamiento y posibles fallos a los que está sometido el dispositivo. Este proyecto de tesis compara dos técnicas de verificación para el control de una máquina tostadora de café, desde el diseño del mismo hasta la prueba de diferentes situaciones a las que está sometida la misma, implementando el mismo control en dos lenguajes de programación, Arduino y PLC, para un prototipo físico y uno virtual respectivamente. Se diseñó un prototipo físico y se utilizó este mismo para la simulación. Para la modelación del control se utilizó el lenguaje UML, y la metodología RUP para el desarrollo del software.

Palabras Clave: Arduino, PLC, UML, virtual commissioning.

Agradecimientos

Quiero agradecer al Universo infinito por su sabiduría y paciencia en esta etapa de mi vida, tantas señales para crecer pensando bonito a la humanidad.

Agradezco especialmente a mi familia por su apoyo, amor, paciencia y comprensión en el desarrollo de mis estudios.

Agradezco a mis compañeros en la Universidad de los Andes que compartieron conmigo el desarrollo del proyecto y las diferentes etapas que acarrea el mismo, con todos sus estados de ánimo.

Agradezco a mi asesor Giacomo Barbieri por su sincero acompañamiento y consejo en el desarrollo del proyecto.

“Perfecciono con el fin de reflejar

Produciendo el orden

Sello la matriz del sinfín

Con el tono planetario de la manifestación

Me guía el poder de la muerte

Yo Soy un Portal de Activación Galáctica, entra en mi”

Tabla de contenido

Lista de Figuras

Lista de Tablas

Lista de Gráficas

Introducción

En la actualidad, industrialmente se están implementando sistemas automatizados para mejorar la producción de diferentes productos de alta demanda en el mercado. Para esto existen metodologías de diseño, las cuales dan una aproximación a la solución para el proceso estudiado, sin embargo, no aplicable la misma para productos similares, por lo tanto, es necesario investigar en una forma de solucionar las diferentes etapas de un proceso que se adapte a la industria en general, ya que, la complejidad a nivel funcional se incrementa cada vez más y es necesario tener métodos efectivos para el diseño y verificación del proceso.

Usualmente los sistemas de fabricación y procesado, son dinámicos y los cambios de estado de éstos se deben a eventos concurrentes, lo que describe un comportamiento discreto. La simulación de éstos es la forma frecuente de verificar el comportamiento de un sistema (Lee & Park, 2014).

Por otra parte, la industria manufacturera, para mantener su competitividad, debe estar mejorando los productos y el sistema que lo produce, por consiguiente, es necesario tener un ambiente de creación de prototipos eficiente, con lo cual se introdujo el concepto de fabricación virtual (Virtual Commissioning-VC), basado en modelos asistidos por computadora (CAD), sensores y actuadores, para simular procesos de fabricación (Onosato & Iwata, 1993), (Hibnio, Inukai, & Fukuda, 2006).

Según (Lee & Park, 2014), VC permite la verificación completa de un sistema de producción realizando una simulación entre una planta virtual y un controlador real o virtual. Para esto, el modelo de planta virtual describe completamente a nivel de sensores y actuadores. Generalmente en la industria, un sistema de fabricación se estabiliza implementando plantas y controladores reales, lo cual es costoso y demorado. Por lo tanto, el VC sirve para identificar y abordar fallas de diseño y operacionales sin planta real ni controladores, de modo que se puedan lograr ahorros significativos en la implementación del sistema de producción.

Esta teoría hace parte del fundamento que se tuvo en cuenta para resolver el interrogante: ¿Cómo es el comportamiento de dos formas de verificación de software: prototipo vs. virtual commissioning? Para lo cual se formularon los siguientes objetivos:

General:

Comparar el comportamiento de dos formas de verificación de software: prototipo vs. virtual commissioning.

Específicos:

- Diseñar e implementar un prototipo de máquina tostadora basado en los requerimientos de ingeniería para implementar un control en lenguaje Arduino
- Requerimientos de software: definir eventos que pueden hacer fallar la maqueta y pensar en métodos para su implementación real y virtual
- Diseñar el control de una máquina para automatizar el proceso de tostado de café y robusta a los eventos identificados
- Comparar las dos técnicas de verificación a través de los indicadores definidos.

Se utilizó un método de énfasis comparativo, caracterizado por la aplicación de una regla lógica que consiste en variar un fenómeno con la intención de eliminar variables y factores para llegar a lo constante y fundamental. Este método provee una base para realizar afirmaciones sobre regularidades experimentales, lo cual ayuda a describir y sintetizar elementos diferenciadores y comunes sobre el estudio (Caïs, 1997). Se debe tener en cuenta que los conceptos si se puedan comparar, por lo que se debe definir bien las propiedades y atribuciones de los casos a cotejar. En la aplicación de este método se utilizó una estrategia de análisis de casos para sistemas similares, en el que se toma una cantidad de éstos comparando los efectos en el sistema según un mismo evento aplicado. Estos casos son de prueba del software, utilizando una lógica inductiva se analiza cualitativamente las variables y su naturaleza, y se generan descripciones sobre la reacción del sistema.

I. I Estado Del Arte

Para la consolidación de este proyecto fue necesario en primera instancia, realizar una documentación teórica que fundamentó a los procesos tanto investigativo como al de desarrollo de software, a las herramientas de modelado y la verificación del mismo, y que se referencia a continuación.

Metodologías de desarrollo de software

Para el desarrollo de software existen diferentes metodologías a seguir según la aplicación a implementar, éstas se usan para estructurar, planificar y controlar el proceso de diseño de un sistema. Generalmente, se basan en actividades técnicas, diferenciadas y relacionadas con análisis, diseño, programación y prueba. Lo que la diferencia es el grado de énfasis que pone cada una en las diferentes actividades. Se identifican generalmente dos tipos: las metodologías tradicionales o pesadas, y las metodologías ágiles (Canós, Letelier, & Penadés, 2003) .

En cuanto a metodologías tradicionales, es importante recordar que sus inicios se remontan a los años 70’s donde fue necesario el desarrollo de software acorde a la tecnología desarrollada en el momento. Estas metodologías se caracterizaron por ser completamente secuenciales, puesto que, antes de hacer cualquier implementación se plantean detalladamente las actividades y su respectiva documentación y así desarrolla completamente el software (Avison & Fitzgerald, 1988). Se les llama también pesadas, porque si se tienen cambios en el proceso de diseño, por ejemplo, en algún requerimiento del cliente, se generan altos costos en la modificación del desarrollo del mismo. Estas metodologías son poco adaptables a cambios, por lo que no son aconsejables cuando el entorno de desarrollo tiene requisitos no predecibles o pueden variar en el desarrollo. En cambio, son muy prácticas para desarrollos muy pequeños y con muy bajo nivel de riesgo, tales como aquellos que tienen definidos claramente los requerimientos y no sufren cambios drásticos en el proceso a implementar. (Pressman, 1993)

Con respecto a las metodologías ágiles, se puede afirmar que éstas se centran más en el factor humano o el producto de software; entre sus características se destaca que dan mayor valor a los clientes,

con retroalimentaciones de éste e incremento de las etapas del desarrollo, avanzando en fases más cortas. Este proceso iterativo hace más fácil realizar cambios en cada etapa hasta la aprobación por el cliente y así seguir con el desarrollo de la siguiente, ahorrando costos en estos cambios. Son metodologías útiles para desarrollar software muy complejo de forma rápida cuyos requisitos cambian bastante, pero se desea tener una alta calidad en el producto final. Así, se busca simplificar el proceso de desarrollo, en el código menor posible, y entregar la documentación indispensable del software. Sus principales características se muestran en la Tabla 1*Tomado de*

. (Canós, Letelier, & Penadés, 2003).

Tabla 1.Diferencias entre metodologías ágiles y no ágiles.

Metodologías Tradicionales	**Metodologías Ágiles**
Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo	Basadas en heurísticas provenientes de prácticas de producción de código
Cierta resistencia a los cambios	Especialmente preparados para cambios durante el proyecto
Impuestas externamente	Impuestas internamente (por el equipo)
Proceso mucho más controlado, con numerosas políticas/normas	Proceso menos controlado, con pocos principios
Existe un contrato prefijado	No existe contrato tradicional o al menos es bastante flexible
El cliente interactúa con el equipo de desarrollo mediante reuniones	El cliente es parte del equipo de desarrollo
Grupos grandes y posiblemente distribuidos	Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio
Más artefactos	Pocos artefactos
Más roles	Pocos roles
La arquitectura del software es esencial y se expresa mediante modelos	Menos énfasis en la arquitectura del software

Tomado de (Canós, Letelier, & Penadés, 2003)

En este orden de ideas, cabe considerarse otras metodologías muy utilizadas en el desarrollo de software, tradicionales y ágiles como las descritas a continuación:

1) Metodología de cascada

Esta metodología fue la primera en ser utilizada para el desarrollo de software, se caracteriza por tener fases secuenciales donde sólo se empieza la siguiente hasta que se termina la otra. El énfasis de ésta está en la extensa planificación y documentación de todas las etapas, de tal forma que en caso de parar el desarrollo en alguna, cualquier desarrollador puede continuar el trabajo leyendo la documentación de donde va éste. En esta metodología es poco recomendable devolverse en las fases, ya que se estaría perdiendo tiempo y se tendría que hacer muchos cambios en las fases posteriores (Royce, 1970).

Ventajas (Friedman & Cornford, 1989):

- La secuencia ordenada de etapas de desarrollo y controles estrictos de documentación y diseño ayuda a garantizar la calidad, confiabilidad y mantenimiento del software desarrollado.
- Ideal para apoyar equipos de desarrollo y gerentes con menos experiencia, o equipos de desarrollo cuya composición fluctúa.
- El progreso del desarrollo del sistema es mensurable.

Desventajas (Friedman & Cornford, 1989):

- Es Inflexible, lenta, costosa y engorrosa debido a la estructura secuencial y los controles estrictos.
- Los problemas a menudo no se descubren hasta la prueba del sistema.
- El rendimiento del sistema no se puede probar hasta que el sistema esté casi completamente codificado, y la falta de capacidad puede ser difícil de corregir.
- Responde difícilmente a los cambios. Los cambios que ocurren más adelante en el ciclo de vida del software son más costosos de implementar por lo que es muy frecuente que no se hagan y se diseñe un nuevo software.

2) Metodología de Prototipado

Esta metodología consiste en que, con base en requerimientos y necesidades del cliente, se realiza de forma rápida un prototipo, debido a que al contar con las bases necesarias cualquier programador llega fácilmente al código final. Con esto se intenta reducir el riesgo inherente al desarrollo, gracias a dividir éste en segmentos más pequeños y proporcionar facilidad de cambio durante el proceso. Los prototipos a pequeña escala del sistema se desarrollan siguiendo un proceso iterativo de modificación hasta que evolucione para cumplir con los requisitos del usuario, llevando poca documentación para su rápida implementación. A veces la modificación de los requerimientos del cliente genera un prototipo totalmente nuevo. En algunos casos es posible evolucionar de un prototipo a un sistema en funcionamiento, donde se necesita una comprensión básica del problema principal para evitar resolver el problema equivocado.

Ventajas (Green & DiCaterino, 1998):

- Se puede usar para modelar de forma realista aspectos importantes de un sistema durante cada fase del ciclo de vida tradicional.
- Especialmente útil para resolver objetivos poco claros; desarrollar y validar los requisitos del usuario; experimentando o comparando varias soluciones de diseño; o investigando el rendimiento del desarrollo.
- Existe potencial para explotar el conocimiento adquirido en una iteración temprana a medida que se desarrollan iteraciones posteriores
- Ayuda a identificar fácilmente las funciones confusas o difíciles y la funcionalidad faltante
- Proporciona una implementación rápida de una aplicación incompleta pero funcional.
-

Desventajas (Green & DiCaterino, 1998):

- El proceso de aprobación y control no es estricto por su rápida producción de código fuente.

- Puede ocurrir un análisis incompleto o inadecuado del problema por el cual solo se atenderán las necesidades más obvias y superficiales, lo que dará lugar a que las prácticas ineficientes actuales se incorporen fácilmente en el nuevo sistema.
- Los diseñadores pueden descuidar la documentación, lo que resulta en una justificación insuficiente para el producto final y registros inadecuados para el futuro.
- Puede llevar a falsas expectativas, donde el cliente cree erróneamente que el sistema está "terminado" cuando en realidad no lo está; el sistema se ve bien y tiene interfaces de usuario adecuadas, pero no es realmente funcional.
- Las iteraciones se agregan a los presupuestos y programas del proyecto, por lo tanto, los costos adicionales deben sopesarse frente a los beneficios potenciales.

3) Metodología en Espiral

Esta metodología es una combinación entre el modelo de cascada y el modelo de prototipado, sin embargo, a éste se añade la gestión de riesgos, algo que en los modelos anteriores ni siquiera se menciona. Su principal característica se centra en evaluar y minimizar los riesgos del proyecto, al dividir éste en segmentos más pequeños, y proporcionar facilidad de cambio durante el proceso de desarrollo, así como brindar la oportunidad sopesar la consideración de la continuación del proyecto a lo largo del ciclo de vida. Es un modelo evolutivo, que conforme avancen los ciclos, irá incrementando el nivel de código fuente desarrollada, un incremento en la gestión de riesgos y en los tiempos de ejecución y planificación del sistema. Cada ciclo implica una progresión a través de la misma secuencia de pasos, para cada porción del producto y para cada uno de sus niveles de elaboración, desde un concepto general de documento de operación hasta la codificación de cada programa individual. (Zhu, 2005). Comienza cada ciclo con una identificación de los objetivos y sus condiciones de cumplimiento, y finaliza con la revisión de los mismos. Es principalmente utilizada para el desarrollo de grandes proyectos como la creación de un sistema operativo.

Ventajas (Zhu, 2005):

- Mejora la prevención de riesgos.
- Útil para ayudar a seleccionar la mejor metodología a seguir para el desarrollo de una iteración de software dada, basada en el riesgo del proyecto.
- Puede incorporar metodologías de cascada, prototipado, y metodologías incrementales como casos especiales para un caso dado y proporcionar orientación sobre qué

combinación de estos modelos se ajusta mejor a una iteración de software dada, según el tipo de riesgo del proyecto.

Desventajas (Zhu, 2005):

- Es un desafío para determinar la composición exacta de las metodologías de desarrollo a usar para cada iteración alrededor de la Espiral.
- Altamente personalizada para cada proyecto, y por lo tanto es bastante compleja, lo que limita la reutilización.
- No hay controles establecidos para pasar de un ciclo a otro ciclo. Sin controles, cada ciclo puede generar más trabajo para el próximo ciclo.
- No hay plazos firmes. Los ciclos continúan sin una condición de finalización clara, por lo que existe un riesgo inherente de no cumplir el presupuesto o el cronograma.
- Existe la posibilidad de que el desarrollo implementado se halla hecho utilizado solamente utilizando la metodología de cascada.

4) **Metodología de Proceso Racional Unificado** (Rational Unified Process- RUP)

Esta metodología está basada igualmente en el modelo de cascada, sin embargo, se usa de forma iterativa, teniendo una plataforma flexible para el desarrollo de software, que ayuda brindando guías consistentes y personalizadas de procesos para todo el equipo de desarrollo. Crea y mantiene modelos, en lugar de enfocarse en la producción de una gran cantidad de documentación. Esto permite seleccionar el conjunto de componentes de actividades que se ajustan a las necesidades específicas de un proyecto. Esta metodología propone un proceso iterativo e incremental en donde el trabajo se divide en partes más pequeñas o mini proyectos, permitiendo que el equilibrio entre casos de uso y arquitectura se vaya logrando durante cada uno, así durante todo el proceso de desarrollo. Cada mini proyecto es una iteración de la cual se obtiene un incremento que produce un crecimiento en el producto (artefacto) (Kruchten, 2001).

Ventajas (Kruchten, 2001):

- Se evalúa cada fase lo cual permite cambios de objetivos, o requerimientos del cliente.

- Funciona bien en proyectos de innovación, por la flexibilidad en la aplicación según el proyecto, no es necesario aplicar todas las actividades de la metodología.
- Se lleva un seguimiento detallado en cada una de las fases, por medio de los modelos se evidencia el desarrollo del producto.

Desventajas (Kruchten, 2001):

- En ocasiones es excesiva la flexibilidad para algunos proyectos puede generar deficiencia en la calidad.
- El cliente deberá ser capaz de describir y entender a un gran nivel de detalle el proyecto para poder acordar un alcance del proyecto.

5) Metodología Scrum

La metodología Scrum es una de las más conocidas para implementación de desarrollo de software, funciona de forma incremental, sin utilizar conceptos de metodologías secuenciales, el desarrollo de las actividades se va a incrementando sin importar un orden entre en el que se llevan a cabo estos procesos. Existen roles bien definidos y actividades a veces diarias que contribuyen al desarrollo de un proyecto. Esta metodología por ser ágil tiene un fuerte componente en el factor humano como indicador de calidad, donde es más importante la auto organización y el conocimiento de los equipos de trabajo (Schwaber, Beedle, & Martin, 2001). El trabajo en equipo es fundamental para el desarrollo del producto, siendo la comunicación entre éstos fundamental para que todos sepan en qué estado está el desarrollo, de esta forma es más fácil lograr los objetivos planteados.

Ventajas (Schwaber, Beedle, & Martin, 2001):

- Como no hay un orden establecido, con cada iteración se pueden entregar avances reales de objetivos más sencillos, haciendo que el cliente esté enterado del desarrollo del producto. Gracias a los avances mostrados, el cliente puede retroalimentar el proceso.

- Del mismo modo, los riesgos que pueden afectar a un proyecto son gestionados en el mismo momento en que aparecen. La intervención de los equipos de trabajo es inmediata para su solución.
- Cada persona que participa en el proyecto, incluso el cliente, sabe que es lo que tiene que hacer y no es necesario estar reorganizando una y otra vez los roles de cada uno.
- Es adaptable a cualquier contexto, área o sector de gestión. No es una técnica exclusiva de ninguna disciplina.

Desventajas (Schwaber, Beedle, & Martin, 2001):

- Si una persona renuncia o hay algún cambio, es complicado remplazar ese rol, ya que es la persona que se lleva el conocimiento específico y afecta a todo el proyecto.
- Se hacen demasiadas reuniones y a veces no se ha avanzado mucho en el proyecto, esto puede ser agotador y estresante para tratar el mismo tema, de esta forma algunos van perdiendo el interés en el proyecto.
- Funciona con equipos pequeños, donde deben tener objetivos concretos, si no habrá desorden y no se avanzará en el proyecto. Requiere una exhaustiva definición de las tareas y sus plazos.
- Las personas a participar en el desarrollo del proyecto deben contar con una buena formación y gran experiencia en el área de trabajo, para lograr el éxito del proyecto.

6) Metodología Kanban

Esta metodología se desarrolló en Japón inicialmente para sistemas de producción, aunque es aplicable para todo tipo de proyectos, en donde su distintivo es el uso de tarjetas visuales para mostrar el estado de avance de un proyecto. Se utiliza un tablero donde hay secciones definidas para los diferentes flujos de trabajo, actividades y objetivos; todo el equipo puede verlo. Se destaca, además por promover la calidad antes que la velocidad del desarrollo, teniendo que hacer menos correcciones en el producto. Utiliza el principio YAGNI, el cual se rige por hacer solamente lo estrictamente necesario y requerido para que el producto quede bien, evitando lo superficial o extra, de esta forma ofrece mayo calidad, optimizando tiempos y costos. Por ser ágil, tiene flexibilidad en el desarrollo de procesos, con lo cual se le da prioridad a lo urgente y de esta forma también corregir posibles imprevistos, sin embargo, es importante terminar una

actividad para poder empezar otra, y no recargarse de muchas sin finalizarlas (Moeeni, Sanchez, & Vakha Ria, 1997). Cuando un equipo termina una actividad específica puede ayudar a otro a terminar la actividad que esté realizando este.

Ventajas (Moeeni, Sanchez, & Vakha Ria, 1997):

- Provee información rápida y precisa para todo el equipo.
- Mejora la calidad del producto detectar los defectos del mismo rápidamente.
- Se tiene un control del flujo de trabajo, con lo que se localizan fácilmente los cuellos de botella del desarrollo del producto, revisando el estado del mismo.
- Fácil de aplicar, ya que apenas existen reglas de implementación y se puede llevar a cabo de forma artesanal, no requiere software alguno.
- Promueve el trabajo en equipo, por lo que los desarrolladores se comunican entre sí y se ayudan cuando alguna actividad da problemas, así como generar mejoras en las diferentes partes del proyecto.

Desventajas (Moeeni, Sanchez, & Vakha Ria, 1997):

- Tiene dificultad de realizar las entregas a tiempo en grandes proyectos, ya que no hay un control específico del tiempo de cada actividad, en grandes proyectos pueden acumularse un gran número de pequeños retrasos que provocarían la demora en la entrega del producto final.
- Así como la falta de reglas puede ser una ventaja, también puede ser una desventaja si el equipo desarrollador es inexperto y necesita una guía para realizar su trabajo, por lo que es necesario haber creado hábitos de desarrollo de proyectos.
- Asume sistemas de producción repetitivos, dada la naturaleza de su creación en el área de manufactura, por lo que genera un reproceso a la hora de abordar un proyecto nuevo.

7) **Metodología Extreme** Programing (XP)

Esta metodología es destacada entre las ágiles, ya que se basa en su capacidad de adaptación al producto y no tanto a su planificación. Esta no enuncia todos los requerimientos del cliente al comienzo de un proyecto, dándole cabida a la transformación del mismo mientras avanza. Se hace principalmente de

forma iterativa en espiral, aunque se puede ver como una combinación de todas las metodologías de desarrollo de software, sólo que ésta las va aplicando según sea la necesidad del estado del proyecto. Una de sus principales características es que los programadores trabajan en parejas, para corregir y documentar más fácilmente los códigos fuente, teniendo una comunicación constante entre equipos, así como con el cliente, para la objetiva retroalimentación. Se destaca que busca la simplicidad en el diseño del desarrollo, y su implementación, utilizando nombres amigables para las variables, métodos y clases del código, dándole acceso a cualquier miembro del equipo para realizar modificaciones y correcciones, mejorando en tiempos de desarrollo y evitando de esta forma confusiones y documentación extra (Canós, Letelier, & Penadés, 2003). Si en algún momento se dan cuenta que el código fuente no realiza el objetivo planteado, los programadores deben tener la "valentía" de borrarlo y empezar uno desde cero, sin importar la cantidad de tiempo que se haya gastado en su creación y así, no perder tiempo en correcciones que no permiten llegar a un fin exitoso, y si, llegar quizás a otra solución aprendiendo de lo hecho.

Ventajas (Canós, Letelier, & Penadés, 2003):

- Permite definir en cada iteración cuales son los objetivos de la siguiente y así retroalimentar el equipo y el cliente.
- La programación es más organizada, que conlleva una menor tasa de errores, satisfacción de los programadores.
- Apropiada para entornos volátiles, implementando una forma de trabajo donde se adapta fácilmente a las circunstancias del proyecto, lo que resulta en reducir costos.

Desventajas (Canós, Letelier, & Penadés, 2003):

- No se puede delimitar fácilmente el alcance del proyecto con el cliente, es recomendable emplearlo solo en proyectos a corto plazo.
- Altos costos en caso de fallar en el desarrollo del producto.
- Imposible prever todo antes de programar.

Con respecto a lo anotado hasta el momento, de las metodologías mencionadas se seleccionó la RUP por la flexibilidad que da al desarrollo del producto, teniendo la facilidad de aplicación de las fases de forma iterativa, permitiendo, ver cómo cada actividad seleccionada para el proyecto puede irse completando

en cada iteración, utilizando modelos para su desarrollo y hacer la evaluación de calidad del producto. La estructura de esta metodología se muestra más ampliamente en la sección “descripción de metodología”.

A continuación, se hace referencia a cuatro herramientas de modelado de software utilizadas en la implementación de la metodología de desarrollo.

Herramientas de modelado Software

Así como existen diferentes metodologías, para la implementación de éstas, existen herramientas para modelado de software, siendo fundamental su utilización para entender el proceso del proyecto que se desea implementar. Normalmente se basa en la creación de diagramas que explican el funcionamiento detallado de los actores, variables y procesos que se desea. Los más utilizados en el modelamiento de software y que se ajustan a las metodologías estudiadas, son:

8) **Diagramas de flujo**

Los diagramas de flujos son unas herramientas gráficas muy utilizadas en diferentes disciplinas para mostrar la secuencia de rutinas y su interrelación con los respectivos responsables de su ejecución. Su utilización ayuda a entender correctamente las diferentes fases de un proceso a implementar y de esta forma evidenciar posibles soluciones para un mismo problema. Con estos el desarrollador puede definir, formular, analizar y solucionar un problema dado (Downs, Clare, & Coe, 1988). De esta forma se traza una ruta con base en los requerimientos del problema dado, evidenciando actores y salidas del sistema, cada uno representado por diversos símbolos.

Ventajas (Downs, Clare, & Coe, 1988):

- Al ser una herramienta gráfica permite el fácil entendimiento del proceso por varias personas.
- Permite identificar los procesos y errores de lógica fácilmente, dando facilidades para corregirlos.
- Existen compiladores de código fuente que reciben los diagramas de flujo y ejecutan la secuencia.
-

Desventajas (Downs, Clare, & Coe, 1988):

- Si el proceso es demasiado complejo llegan a ser bastante laboriosos durante la planeación y el diseño del mismo.
- Puede ser difícil el seguimiento del proceso si el diagrama tiene caminos diferentes.
- No tiene normas establecidas para la elaboración, dificultando incluir detalles que el desarrollador quiera.

9) Modelamiento por máquina de estados

El modelamiento por máquinas de estados muestra el comportamiento de un sistema con sus entradas y salidas, donde se tiene memoria de las entradas, lo cual determina la salida. Es un conjunto de estados que muestra la relación del proceso en el que se encuentra el sistema respecto a la entrada o la salida existente. Estos estados muestran todas las posibilidades en las que puede estar en el macro proceso, dependiendo de los eventos ocurridos. De esta forma se evidencia una ruta que puede tomar la información para llegar a determinado estado, con lo que se puede visualizar la ejecución de cada proceso en un sistema, detectando posibles errores en los procedimientos o manejo de señales para ejecutar el proceso requerido (Gill, 1962).

Ventajas (Gill, 1962):

- Es muy utilizada en sistemas interactivos, donde se expresa la intención del usuario, o por medio de sensores el uso que se le quiere dar al sistema.
- Como técnica de extracción de requerimiento permite que el desarrollador se centre en los requerimientos del usuario, evidenciando fácilmente el proceso a seguir, facilitando la identificación de prioridades de ejecución de los procesos.

Desventajas (Gill, 1962):

- La inclusión de muchas relaciones hace que los diagramas sean más difíciles de leer, sobre todo para los usuarios finales del producto.

10) Redes de Petri

Son representaciones gráficas donde se modela el flujo de información de la estructura y comportamiento dinámico de un sistema. Son muy utilizados para procesos discretos concurrentes o paralelos, los cuales se determinan por "lugares", "marcas" y "arcos", sincronizando adecuadamente los subprocesos del sistema en general. Por su estructura se pueden evidenciar fenómenos de funcionamiento especiales o errores en el sistema. Las redes de Petri se pueden representar matemáticamente también, cumpliendo unas simples reglas donde se verifica si se habilita o deshabilita un "lugar" determinado para hacer una transición a otro, lo cual mueve la "marca" a ese "lugar", donde el proceso general continua y así ejecutar paralelamente otro "lugar" si es permitido, evitando conflictos en el funcionamiento. Son idénticos en su planteamiento, ya que cumplen las mismas reglas, solo que puede ser más sencillo verlo gráficamente (Haas, 2002).

Ventajas (Haas, 2002):

- Permite tratar procesos independientes de forma individual.
- Permite modelar procesos concurrentes, o sistemas donde un recurso es compartido.
- Proporciona un modelamiento más exacto que los diagramas de flujo.

Desventajas (Haas, 2002):

- Las redes de Petri generales no pueden modelar ciertas situaciones de prioridad.
- Pueden presentarse problemas de análisis según la complejidad del sistema.
- Si el modelamiento es muy detallado puede generar aumento en el tiempo de desarrollo por su complejidad.

11) Diagramas de lenguaje unificado de modelado

Es el lenguaje de modelado más utilizado en la actualidad (Unified Modeling Language-UML), adoptado por varias metodologías para su desarrollo. Está estandarizado para describir un sistema utilizando

diferentes diagramas para modelar aspectos conceptuales como estructuras, comportamiento e interacción dentro del sistema. Es utilizado en diferentes áreas de conocimiento, ya que permite desglosar un sistema detalladamente para el fácil entendimiento de cualquier persona. Por medio de gráficos se organiza el proceso de desarrollo del software, la utilización de los diferentes diagramas se determina según la complejidad del sistema. No tiene un proceso definido de aplicación, en el cual, se pueden registrar diseños parciales independientes de los procesos a desarrollar. Es un acercamiento a la forma en que va a ser programado el código fuente (Booch, Rumbaugh, & Jacobson, El Lenguaje Unificado de Modelado, 1999).

Ventajas (Booch, Rumbaugh, & Jacobson, El Lenguaje Unificado de Modelado, 1999):

- Se puede usar para diferentes tipos de sistemas.
- Muestra claramente los requerimientos, tareas y necesidades del usuario del sistema.
- Es fácilmente entendible porque consolida muchas de las notaciones y conceptos más utilizados en las diferentes metodologías de desarrollo de software.

Desventajas (Booch, Rumbaugh, & Jacobson, El Lenguaje Unificado de Modelado, 1999):

- No es un método de desarrollo lo que lo hace independiente del ciclo de desarrollo.
- No se presta con facilidad al diseño de sistemas distribuidos.
- No permite determinar los requisitos no funcionales

Es necesario anotar que de las diferentes herramientas de modelado se seleccionó la de UML, ya que se ajusta a la metodología RUP, siendo estandarizada para las diferentes fases de ésta, creando un modelo que refleje lo que se desea para el producto. Es una herramienta bastante clara para visualizar, especificar, construir y documentar el desarrollo del producto por medio de los diferentes diagramas según las necesidades del mismo. Sus componentes se muestran más ampliamente en la sección “Descripción de la metodología”.

Teniendo en cuenta que en las metodologías de desarrollo hay una etapa de prueba de software, en este proyecto se hace énfasis para comprender mejor el proceso y para ello se aclara algunos conceptos:

Prueba de software

Como parte importante del desarrollo de software están las pruebas del correcto funcionamiento y satisfacción de los requerimientos planteados por el cliente para el producto. Las pruebas proporcionan información de riesgos potenciales, además de identificar fallas y errores (Naik & Tripathy, 2008), hay dos análisis para las pruebas:

- **Análisis estático**: Es aquel que se centra en revisar la documentación y modelamiento de requerimientos, diseño, incluso el código fuente. Para esto se revisan los algoritmos y se hacen pruebas de corrección en la lógica, no se hacen pruebas en la ejecución del código en construcción. Se examina el código viendo sus posibles comportamientos durante la ejecución, las mejoras que se hacen en un compilador son las más utilizadas como análisis estático.
- **Análisis dinámico**: Este sucede cuando el software ha sido terminado y se ejecuta para encontrar posibles fallas en el programa, a su vez, se observan las propiedades de rendimiento y comportamiento del mismo. En este análisis se seleccionan cierta cantidad de eventos de forma cuidadosa en los cuales se evidencian comportamientos representativos del programa, y se puede llegar a una conclusión sobre la calidad del mismo.

Además, existen dos conceptos que están muy relacionados con las pruebas de software, validación y verificación (Baresi & Pezzè, 2006), los cuales se definen y diferencian a continuación:

- **Verificación**: Esta actividad ayuda a evaluar el sistema de software determinando si el producto de una fase de una metodología está cumpliendo con los requerimientos establecidos al inicio de esa fase, en pocas palabras esta revisa la exactitud de desarrollo de una fase.

- **Validación**: Esta actividad ayuda a confirmar si el producto de desarrollo cumple con su intención, esta apunta a corroborar si el producto cumple con los requerimientos del cliente. Dicho de otra forma, se enfoca en el producto final, donde se prueba de forma extensa desde el punto de vista del usuario.

En el área de prueba de software, se encuentran tres términos que pueden ser tomados como similares, pero en el fondo tienen distintos significados, éstos son: falla, error y falta. Es necesario definirlos para entender cómo encontrarlos y relacionarlos a la solución adecuada (Naik & Tripathy, 2008).

- **Falla**: Esta ocurre cuando el comportamiento externo del sistema no se ajusta a lo establecido en la especificación del sistema. Esta puede permanecer indetectable por mucho tiempo, hasta que un evento específico la active. Cuando esto pasa generalmente el sistema entra a un estado de error.

- **Error**: Es un estado del sistema. En la ausencia de alguna acción correctiva por el sistema un estado de error puede pasar a ser una falla, la cual no puede ser atribuida a ningún evento proveniente del error
- **Falta**: En general es la causa que lleva a un error.

De esta forma, se comprenden los objetivos de la prueba de software que determinan si es correcto lo desarrollado, al aplicar pruebas parciales a la lógica o revisar los productos al final de cada fase. De igual manera, para encontrar faltas en partes del sistema o en éste, intentando que el sistema falle. Además, de reducir riesgos de falla haciendo varias pruebas, para corregir las faltas encontradas en cada una, bajando la tasa de falla del sistema.

Para las pruebas es necesario hacer casos, en la forma básica de "para tal entrada tal salida", es necesario construirlos utilizando como fuente las especificaciones funcionales y de requerimientos del cliente, así como el dominio de las entradas y las salidas. Para esto se sigue una serie de actividades (Naik & Tripathy, 2008):

- Identificar un objetivo a ser probado. Dicho objeto define la intención de diseñar casos de prueba, asegurando que el programa apoye su objetivo. Cada prueba debe estar asociada a un propósito claro.
- Seleccionar las entradas. Se basa en la especificación de los requisitos, el código fuente o expectativas propias del desarrollador; tales entradas deben seleccionarse manteniendo el objetivo de la prueba en mente.
- Configurar el entorno de la ejecución del programa. En este paso todas las suposiciones externas al programa deben ser satisfechas, por ejemplo, hacer una conexión de red disponible, iniciar remotamente un servidor para probar el código, etc.
- Ejecutar el programa. Para llevar a cabo un caso de prueba, las entradas pueden ser proporcionadas al programa en diferentes ubicaciones físicas y tiempos. El concepto de coordinación de prueba se usa para sincronizar diferentes componentes de un caso de prueba.
- Analizar el resultado de la prueba. Se compara el resultado real de la ejecución del programa con el esperado. La complejidad de la comparación depende de la de los datos observados. Al final del paso de análisis, se asigna un veredicto de prueba al programa.

De acuerdo al autor citado anteriormente, hay tres tipos de veredictos de prueba: aprobado, reprobado y no concluyente. A continuación, se muestra como diferenciarlos.

- Aprobación: Este se da si el programa produce el resultado esperado y se cumple el propósito del caso de prueba
- Reprobado Este se da si el programa no produce el resultado esperado.
- No concluyente: esto se da en algunos casos en los que no es posible asignar uno de los anteriores veredictos claramente, en estos casos se deben realizar más pruebas para refinar el veredicto no concluyente en un veredicto de aprobado o reprobado.

Al terminar estas actividades, se debe escribir un informe de prueba. Así, se puede corregir la falla si ésta revela una. Un informe de prueba contiene los siguientes elementos:

- Explicación de cómo reproducir el error.
- Análisis de la falla para poder describirla
- Un indicador de la actual salida del caso de prueba. Este contiene la entrada que lo produce, la salida esperada y el entorno de ejecución.

Desarrollo del proceso

A continuación, se describe las metodologías del proceso de diseño del software, las herramientas de modelado del mismo, y finalmente la aplicación de la metodología que sirvió para el desarrollo del presente proceso.

A. **Descripción de la metodología**

1) **Metodología de Diseño de Software**

En esta sección se amplía la estructura de la metodología de desarrollo seleccionada, conocida como: Rational Unified Process (RUP), la cual se ajustó a lo esperado en este proyecto.

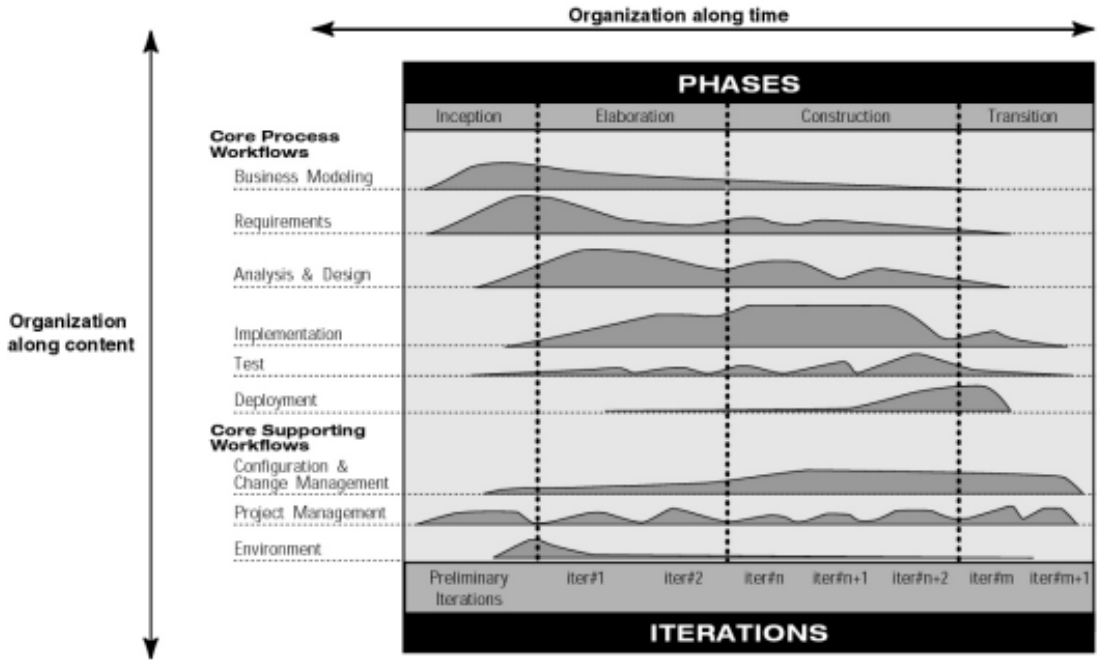
RUP es una metodología de desarrollo de software en la que se hace un acercamiento disciplinado a la asignación de tareas y responsabilidades en un desarrollo. Su propósito es asegurar la producción de software de alta calidad que se ajuste a las necesidades del usuario final con unos costos y calendario predecibles (Kruchten, 2001).

De este modo las actividades de RUP crean y mantienen modelos; no se centra en producir una gran cantidad de documentos, enfatizando el desarrollo de éstos, los cuales son una representación rica del sistema del software en desarrollo. Esta metodología se usa como guía sobre cómo aplicar efectivamente el

lenguaje de modelado unificado (UML), ya que es un lenguaje estándar industrial, que permite comunicar claramente los requisitos, arquitecturas y diseños. Es importante resaltar que, al utilizar los modelos y artefactos creados por esta metodología, se apoya significativamente en la revisión de los cambios generados en cada iteración de su aplicación, caracterizándola como flexible y configurable. Cabe resaltar, que no existe una única forma de implementarla porque se adapta a diferentes composiciones de equipos de desarrollo (Kruchten, 2001).

Según Kruchten (2001) ésta se basa en una arquitectura de proceso en dos ejes, simple y clara. Los aspectos estáticos del proceso en el eje vertical, están descritos en términos de actividades, roles, artefactos y flujos de trabajo. Los aspectos dinámicos en el eje horizontal en los que se representa el tiempo de desarrollo, consta de cuatro fases: Inicio, Elaboración, Construcción, Transición. En la figura 2, se muestra, por ejemplo, la organización de los diferentes flujos de trabajo del desarrollo del producto y cómo se desarrollan en cada fase a través del tiempo. Estas disciplinas pueden variar según el proyecto a desarrollar, las fases dinámicas se explican a continuación.

Figura 1. Fases, Iteraciones y disciplinas.



Tomado de (Process, 1998)

- **Inicio**. Como su nombre lo dice es la fase donde se empieza el proyecto, se plantean los objetivos del producto, si se puede hacer, cuánto va a costar, etc. En esta fase se trata de encontrar los requerimientos del cliente de forma incremental al aplicar iteraciones para cada actividad. Se busca delimitar el proyecto, identificando los actores y actividades que definen la funcionalidad del producto. También se hace una estimación de los costes en recursos y tiempo, así como la estimación de riesgos y lo que puede generarlos.

Como productos de las iteraciones de esta fase de inicio se tienen los objetivos y restricciones del proyecto, llamados también los requerimientos funcionales del producto. Éstos se expresan por medio de modelos de casos de uso.

- **Elaboración.** La intención de esta fase es analizar el dominio del problema, desarrollar el plan del proyecto. se construye un prototipo de la arquitectura del software, que debe evolucionar en las iteraciones sucesivas hasta convertirse en el producto final en las fases siguientes. Para esto se utilizan los casos de uso esbozados en la primera fase. En esta fase se busca definir, validar y cimentar la arquitectura del software, y generar un plan de desarrollo para la de construcción, esto para evitar riesgo y costes.

Como producto se tiene un modelo de casos de uso casi completos, al menos del 80% en la identificación de actores e interacciones con los procesos del sistema.

- **Construcción**. El propósito principal de esta fase es alcanzar el funcionamiento del software desarrollado, haciendo iteraciones para lograrlo. Se implementan todos los componentes, características y requisitos del sistema que no se hayan trabajado a profundidad. A éstos se les realizan pruebas para tener un producto "beta" en el usuario. Esta fase enfatiza el control de las operaciones realizadas, de tal forma que se administren recursos eficientemente y se optimicen los costes y tiempos del desarrollo. Así, tratando de conseguir calidad en el producto, se trata de lograr versiones del mismo tan rápido como se pueda.

Como producto de esta fase se tiene modelos completos de casos de uso, análisis, diseño, etc. Manual inicial de usuario, y un prototipo operacional (beta).

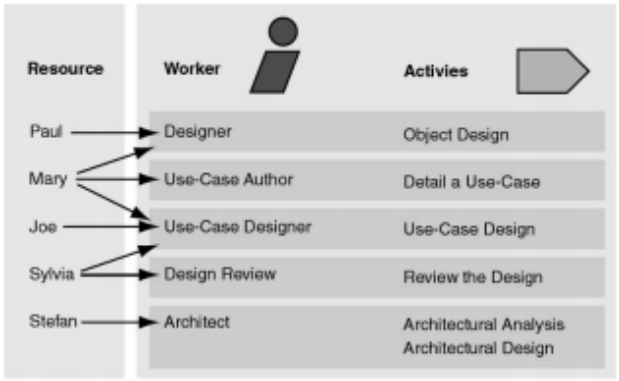
- **Transición**. La finalidad de esta fase es darle al usuario el producto terminado, lo que significa hacer las actualizaciones necesarias para entregar una versión final, con una completa documentación y entrenar al usuario en el manejo del mismo. Es en general darle el poder al usuario del manejo del producto, como llegar a hacer ajustes, configuración, instalación y su uso. En esta fase se hacen las pruebas con el usuario para validar el funcionamiento, de tal manera que el usuario se valga por si mismo al momento de usar el producto final, cumpliendo con los requisitos esperados y que satisfaga suficientemente éstos.

Dentro de este marco, para Kruchten (2001) RUP en su estructura estática está definida por cuatro elementos: roles, actividades, artefactos y flujos de trabajo, cuyas definiciones están a continuación:

Roles. Estos definen el comportamiento y responsabilidades de un individuo o un grupo. Cada individuo puede cumplir uno o más roles en el desarrollo del software, en donde se indica qué actividades va a realizar y qué artefactos va a producir.

Actividades. Una actividad es una unidad de trabajo para un determinado rol. Las actividades se describen claramente con el propósito de la misma, expresada en términos de crear o actualizar algunos artefactos, como un modelo, una clase o un plan. Cada actividad es asignada siempre a un rol definido, desarrollada en tiempos específicos, como pocas horas, hasta pocos días. Son detalladas, por lo que involucran pocos roles y artefactos en el proyecto. Éstas se utilizan como elementos de planeación y progreso. En la Figura 2 se muestra ejemplos de roles y actividades.

Figura 2. Personas, roles y actividades.



Tomado de (Process, 1998)

Artefactos. Es un fragmento de información que es producida, modificada o usada por el proceso de desarrollo. Son productos tangibles del proyecto, las cuales se usan hasta llegar al producto final. Son utilizados en parte como entradas por los roles para realizar una actividad, y a su vez son el resultado de éstas.

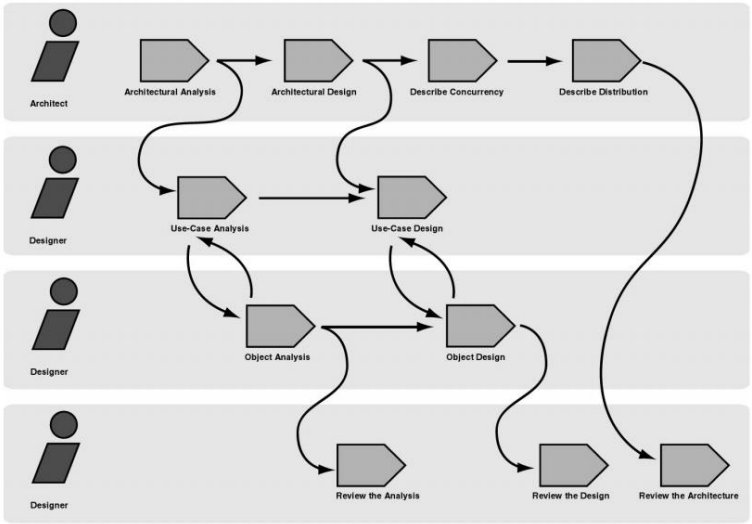
Los artefactos pueden ser muy variados:

- Un modelo, por ejemplo, casos de uso o de diseño.
- Un elemento de los modelos, como una clase o un simple caso de uso o un subsistema.
- Un documento, como el caso de negocio, o el documento de manual de usuario.

- El código fuente del desarrollo
- Los ejecutables del desarrollo.

Flujos de trabajo. Es una secuencia de actividades que producen un resultado de valor observable, en el cual se describen visiblemente cómo interactúan los diferentes roles y sus actividades en una parte del proceso. Puede ser expresado utilizando la notación UML, como un diagrama de secuencia, un diagrama de colaboración o un diagrama de actividades. En la Figura 3 se ve un diagrama de actividades, no siempre es posible o práctico mostrar todas las dependencias entre actividades, a veces dos actividades o más interactúan muy estrechamente, en las que involucra el mismo rol. Como los roles son desarrollados por personas, un flujo de trabajo no puede ser interpretado literalmente como un programa a seguir exactamente y mecánicamente por las personas.

Figura 3. Ejemplo de Flujo de trabajo



Tomado de (Process, 1998)

En la metodología RUP existen nueve flujos de trabajo principales, lo cual ayuda a dividir de forma lógica todos los roles y actividades en grupos. Éstos se dividen en seis principales, llamados de "ingeniería", y tres de "soporte". Sus nombres son similares a las etapas utilizadas en la metodología secuencial de cascada, sin embargo, son diferentes, ya que se aplican de forma iterativa como flujos de trabajo en un ciclo de desarrollo, con diferente énfasis e intensidad según la fase de la metodología en la que se está desarrollando. Éstos se explican a continuación (Process, 1998).

Flujo de trabajo de Ingeniería:

- **Flujo de trabajo de Modelado de Negocios**. Éste ayuda a comunicar efectivamente los grupos de negocios e ingeniería mostrando como crear y mantener una trazabilidad directa entre los modelos de negocios y los de software, evitando que el producto no sea el idóneo como entrada para el uno o el otro. En su modelamiento se documenta el proceso utilizando los llamados casos de uso de negocios. Éstos aseguran un entendimiento común entre todas las partes interesadas, como de qué procesos se necesita el respaldo de la organización desarrolladora. Los casos de uso de negocios son analizados para entender cómo los hechos deberían apoyar el proceso en general, esto es documentado en un modelo de objeto de negocios. Muchos proyectos deciden no hacer este modelamiento.

- **Flujo de trabajo de Requerimientos**. El objetivo de este flujo de trabajo es describir lo que se espera que el sistema haga, y de esta forma, los diseñadores y el cliente puedan estar de acuerdo. Para lograrlo, se obtienen, organizan y documentan las funcionalidades y restricciones requeridas. En un documento de Visión se organizan las necesidades de las partes interesadas, identificando los actores (Usuarios, u otros sistemas que interactúan con el sistema a desarrollar), y los casos de uso que representa el comportamiento del sistema (Estos son desarrollados de acuerdo a las necesidades de los actores). Cada caso de uso es descrito en detalle, en el que se muestra paso a paso la interacción de los actores y que hace el sistema. Los requerimientos no funcionales son descritos en las especificaciones suplementarias. Estos casos de uso son útiles en todo el proceso de desarrollo del software, en las etapas de análisis, diseño y pruebas.
- **Flujo de trabajo de Análisis y diseño**. El fin de éste es mostrar como el sistema va a ser completado en la fase de implementación. Generalmente se quiere que el sistema realice las tareas y funciones descritas en el caso de uso de requerimientos, por lo que el modelo de diseño es una abstracción de cómo está estructurado y va a ser escrito el código fuente que las cumple. Este consiste en el diseño de clases estructuradas en paquetes y subsistemas de diseño con interfaces bien definidas, representado como viene cada componente y su implementación del sistema. Contiene además descripciones de como los objetos de esas clases de diseño colaboran para realizar casos de uso. Las actividades de diseño están centradas en el concepto de arquitectura. La producción y la validación de esa arquitectura es el principal centro de las iteraciones tempranas de diseño, siendo ésta importante para llegar a un buen modelo de diseño e incrementar la calidad de cualquier modelo en el desarrollo. La arquitectura está representada por

diferentes vistas arquitectónicas, las cuales muestran porqué se tomaron las decisiones de diseño; siendo simplificadas, donde las características más importantes son más visibles con anotaciones en detalle a los lados de éstas.

- **Flujo de trabajo de Implementación**. El objetivo es definir la organización del código, en términos de componentes (Archivos fuente, ejecutables), y de probar éstos como unidades. Al implementar estos componentes el sistema quedaría completo, lo que hace que este flujo permita ver qué componentes pueden ser reutilizados de otros desarrollos, o implementar unos nuevos haciendo que el sistema sea más fácil de mantener, e incrementar sus posibilidades de reutilizarse.
- **Flujo de trabajo de Pruebas**. Por ser un proceso iterativo las pruebas se desarrollan a través de todo el proyecto, esto permite encontrar defectos de forma temprana, dentro de lo posible, lo cual reduce el costo de arreglarlos. En este flujo se verifica la interacción entre objetos, la integración correcta de todos los componentes del software, y que todos los requerimientos fueron implementados.

- **Flujo de trabajo de Despliegue**. El objetivo es producir con éxito lanzamientos del producto y entregar el software para su usuario final. Para esto tiene varias actividades que deben ser realizadas, como empaquetar, distribuir e instalar el software, dar asistencia a los usuarios finales. La mayoría de estas actividades se hacen en la fase de transición, este flujo es uno de los que tiene menos información de cómo se va a desarrollar.

Flujo de trabajo de Soporte:

- **Flujo de trabajo de Administración de proyecto**: Éste se enfoca principalmente en especificar aspectos de un proceso de desarrollo iterativo. Proporciona un marco de referencia para manejar proyectos intensos de desarrollo de software y el riesgo de éste. Así mismo, da guías prácticas para la planeación, abastecimiento de personal, ejecución y monitoreo de proyectos.
- **Flujo de trabajo de Configuración y cambio de administración**. Describe como controlar los artefactos producidos por mucha gente en un proyecto. Este control ayuda a evitar costosa confusión, y asegura que los artefactos no estén en conflicto entre si. Este flujo de trabajo sigue las variantes de softwares en evolución, como la versión que se está desarrollando o los parches que se están implementando.
- **Flujo de trabajo de Entorno**. El objetivo es proporcionar a la organización de desarrollo de software, las herramientas y procesos para lograrlo. Se centra en las actividades para configurar el proceso en el contexto de un proyecto. También se enfoca en actividades para desarrollar las pautas

necesarias que apoyan un proyecto. Se proporciona un procedimiento paso a paso que describe cómo implementar un proceso en una organización.

Esta es la información relevante respecto a la metodología seleccionada, implementada en la sección "Caso de estudio". Es importante entonces ver también la información de la herramienta de modelado seleccionada, el lenguaje unificado de modelado (UML).

2) Herramienta de Modelado de Software

UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema de software, donde se centra en representar conceptual y físicamente un sistema, es un estándar para los planos del software. Es un lenguaje que cubre las diferentes vistas de la arquitectura de un sistema mientras evoluciona a través de un ciclo de vida del desarrollo del sistema, UML no dice qué modelos deben ser implementados, éstas son decisiones por parte del equipo desarrollador (Booch, Rumbaugh, & Jacobson, El Lenguaje Unificado de Modelado, 1999).

Para visualizar los artefactos utiliza diagramas con una semántica bien definida, construyendo modelos precisos, sin ambigüedades, cubriendo la especificación de las decisiones de análisis, diseño e implementación. Los modelos pueden conectarse de forma directa a muchos lenguajes de programación, esto permite la generación de código fuente a partir de un modelo de UML, y la ejecución directa de modelos, simulación e instrumentación de sistemas en ejecución. Las siguientes definiciones se toman del libro "El Lenguaje Unificado de Modelado" (Booch, Rumbaugh, & Jacobson, El Lenguaje Unificado de Modelado, 1999).

UML cuenta con tres elementos principales para generar el modelo conceptual de un proyecto, éstos son: 1. Elementos, 2. Relaciones y 3. Diagramas.

1. Los elementos son los que componen el modelo, las relaciones ligan estos elementos entre sí; los diagramas agrupan colecciones de elementos y sus interacciones.

Los elementos a su vez se dividen en cuatro tipos:

- **Elementos estructurales**: En su mayoría son partes estáticas de un modelo, y representa cosas materiales o conceptuales, de estos a su vez se pueden generalizar en 7 tipos.

- **Clase**: Es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones, y semántica. Una case implementa una o más interfaces.
- **Interfaz**: Es una colección de operaciones que especifican un servicio de una clase o componente. Describe el comportamiento visible externamente de ese elemento.
- **Colaboración:** Define una interacción y es una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos.
- **Caso de Uso**: Es una descripción de un conjunto de secuencias de acciones que un sistema ejecuta y que produce un resultado observable de interés para un actor particular. Se utiliza para estructurar los aspectos de comportamiento en un modelo.
- **Clase activa**: Es una clase cuyos objetos tienen uno o más procesos de ejecución y por lo tanto pueden dar origen a actividades de control. El comportamiento de los elementos de esta clase ocurre concurrentemente con el de otros elementos.
- **Componente**: Es una parte física y reemplazable de un sistema que conforma un conjunto de interfaces y proporciona la implementación de dicho conjunto.
- **Nodo**: Es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que por lo general dispone de algo de memoria y, con frecuencia, capacidad de procesamiento. Un conjunto de componentes puede residir en un nodo y también migrar de un nodo a otro.

- **Elementos de comportamiento**: Estos elementos son las partes dinámicas del modelo. Generalmente son los verbos de un modelo, y representan el comportamiento en el tiempo y el espacio. Hay dos tipos principales de estos elementos:
  - Interacción: Es el que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto particular, para alcanzar un propósito específico. Una operación individual o un conjunto de estas puede especificarse con una interacción, involucrando muchos elementos como mensajes, secuencias de acción y enlaces.
  - **Máquina de estados**: Es la que especifica las secuencias de estados por las que pasa un objeto o una interacción durante su vida en respuesta a eventos, junto con sus reacciones a eventos. Una clase individual o un conjunto de estas puede especificarse con una máquina de estados, involucrando elementos como estados, transiciones, eventos y actividades.
- **Elementos de agrupación**: Son las partes organizativas de los modelos UML. Estas son las cajas en las que puede descomponerse un modelo. Solo existe un elemento de agrupación.
  - **Paquete**: Es un mecanismo de propósito general para organizar elementos en grupos, puede incluir todo tipo de elementos. Los paquetes son conceptuales.

- **Elementos de anotación**: Estos elementos son las partes explicativas de los modelos UML. Son comentarios que se pueden aplicar para describir, clarificar y hacer observaciones sobre cualquier elemento de un modelo.
  - **Nota:** Es un símbolo para mostrar restricciones y comentarios junto a un elemento o una colección de estos.

2. **Las relaciones** son bloques básicos de construcción para escribir modelos bien formados, existen cuatro tipos principales:

- **Dependencia:** Es una relación semántica entre dos elementos, en la cual un cambio a un elemento puede afectar a la semántica del otro elemento.
- **Asociación:** Es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos.

- **Generalización**: Es en la cual los objetos del elemento especializado (hijo) pueden sustituir a los objetos del elemento general(padre). De esta forma, el hijo comparte la estructura y el compartimiento del padre.
- **Realización**: Es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Se pueden encontrar entre interfaces y las clases y componentes que las realizan, y también entre los casos de uso y las colaboraciones que las realizan.

3. **Los diagramas** son representaciones gráficas de un conjunto de elementos, visualizados generalmente como un grafo conexo de nodos (elementos) y arcos(relaciones). Un diagrama es una proyección de un sistema, representa una vista resumida de los elementos que constituyen un sistema. En teoría, este puede contener cualquier combinación de elementos y relaciones, en la práctica, sin embargo, solo surge un pequeño número de combinaciones consistentes con las vistas más útiles de la arquitectura de un sistema.

Existen nueve tipos generales de diagramas:

- **Diagrama de Clases**: Muestra un conjunto de clases, interfaces, y colaboraciones, así como sus relaciones. Son los diagramas más comunes en el modelado de sistemas orientados a objetos.

- **Diagrama de objetos**: Muestra un conjunto de objetos y sus relaciones, representan instantáneas de instancias de los elementos encontrados en los diagramas de clases.
- **Diagrama de casos de uso**: Muestra un conjunto de casos de uso y actores y sus clases, cubren la vista de casos de uso estática de un sistema. Son especialmente importantes en el modelado y organización del comportamiento de un sistema.
- **Diagrama de secuencia**: Es un diagrama de interacción que resalta la ordenación temporal de los mensajes entre objetos y sus relaciones.
- **Diagrama de colaboración**: Es un diagrama de interacción que resalta la organización estructural de los objetos que envían y reciben mensajes.
- **Diagrama de estados**: Muestra una máquina de estados que consta de estados, transiciones, eventos y actividades, cubren la vista dinámica de un sistema. Son importantes para el modelado del comportamiento de una interfaz, una clase, o una colaboración, y resaltan el comportamiento dirigido por eventos a un objeto, lo cual es útil en el modelado de sistema reactivo.

- **Diagrama de actividades**: Es un tipo especial de diagrama de estados que muestra el flujo de actividades dentro de un sistema, son importantes para modelar el funcionamiento de un sistema y resalta el flujo de control entre objetos.
- **Diagrama de componentes**: Muestra la organización y las dependencias entre un conjunto de componentes.
- **Diagrama de despliegue**: Muestra la configuración de nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos. (págs. 15-22).

B. Aplicación de la metodología

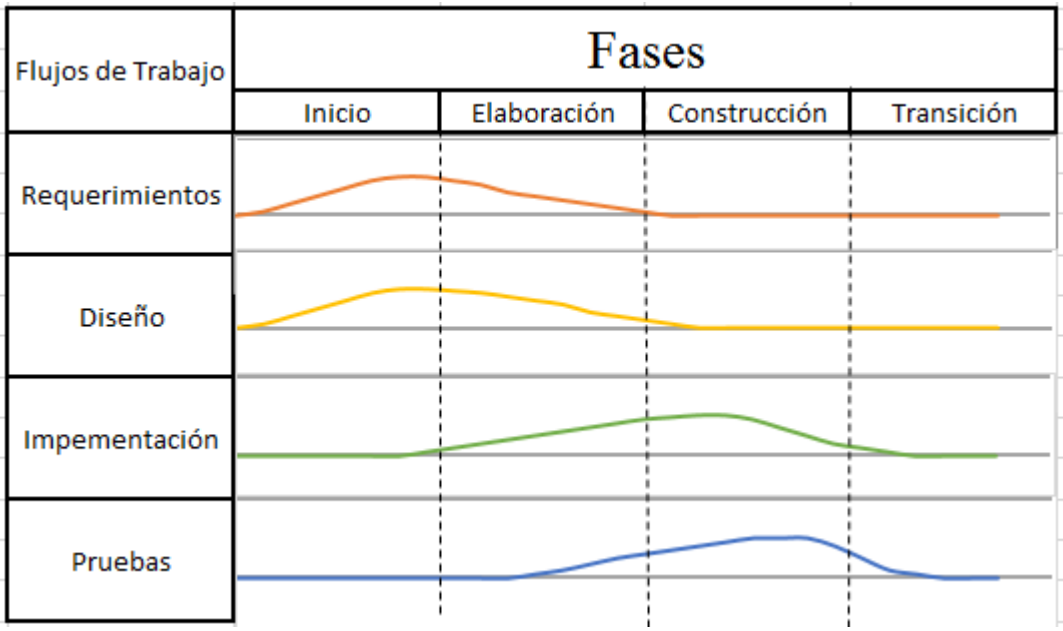
Con la revisión de la metodología y la herramienta de modelado se seleccionaron los diferentes flujos de trabajo aplicados en el proyecto, así como los diferentes diagramas en las fases de desarrollo del software.

Selección de flujos de trabajo

Como en el proyecto se desarrolla un programa de funcionamiento para la máquina, la cantidad de software que se va a generar no es extensa. El flujo de trabajo de modelamiento de negocio no es necesario, ya que el software desarrollado no va a salir a la venta, al igual que el de despliegue, ya que el software diseñado va quedar instalado en la máquina del caso de estudio, teniendo un solo usuario. Tampoco los flujos de trabajo de soporte van a ser implementados porque son más de administración del proyecto. Por esto los flujos de trabajo seleccionados son:

- Flujo de trabajo de requerimientos.
- Flujo de Trabajo de Diseño.
- Flujo de trabajo de Implementación.
- Flujo de trabajo de Prueba.

Figura 4. Diagrama de desarrollo Metodología



Realizado por Sergio A. Naranjo C.

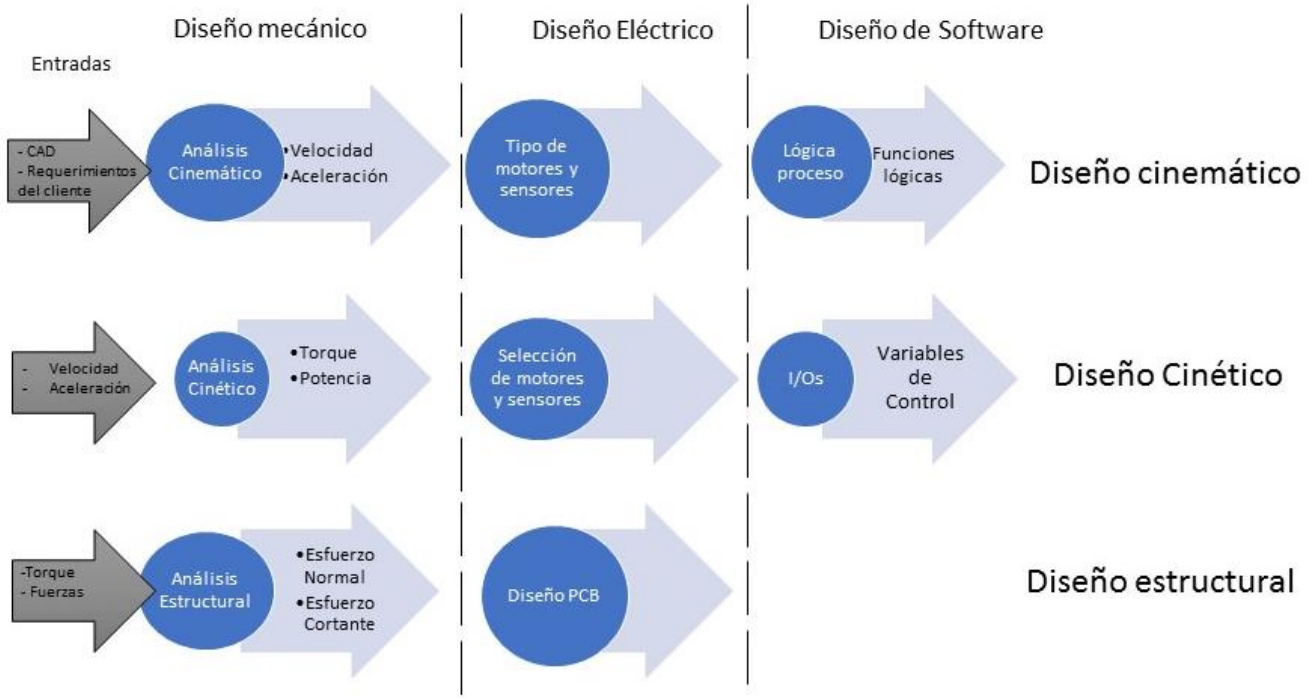
Para el desarrollo de estos flujos de trabajo se seleccionaron como herramientas de modelado de UML los diagramas de Casos de Uso de requerimientos y de diseño, los cuales se desarrollaron principalmente en las fases de inicio y elaboración. A su vez, los diagramas de estados se desarrollaron para el flujo de trabajo de implementación en las fases de elaboración y construcción. Finalmente se aplicaron los casos de prueba en la fase de construcción. En la Figura 4 se observa la intensidad y la fase de aplicación de cada flujo de trabajo.

II. Resultados

Diseño del prototipo de máquina tostadora

Para el diseño del prototipo en el que va a ser probado el software diseñado se implementó una variación de la metodología de diseño de máquinas de (Norton, 2009), la cual se muestra en la Figura 5.

Figura 5. Metodología de diseño utilizada.



Realizada Por Sergio A. Naranjo C.

Para empezar el diseño se tuvo en cuenta los requerimientos del cliente, que se muestran en la *Tabla 1*, con los cuales se identificaron las funcionalidades del sistema, éstas se muestran en la

Figura 6. Se hizo una búsqueda de información sobre el proceso de tostado (Figura 7), donde se encontraron conceptos predominantes en el mercado para este proceso (Tabla 3). Se evaluaron diferentes conceptos para solucionar las funcionalidades y se exploraron los medios físicos para implementarlas (Tabla 4). Así, se modeló en el software CAD Inventor 2018® el prototipo a realizar, se muestra en la

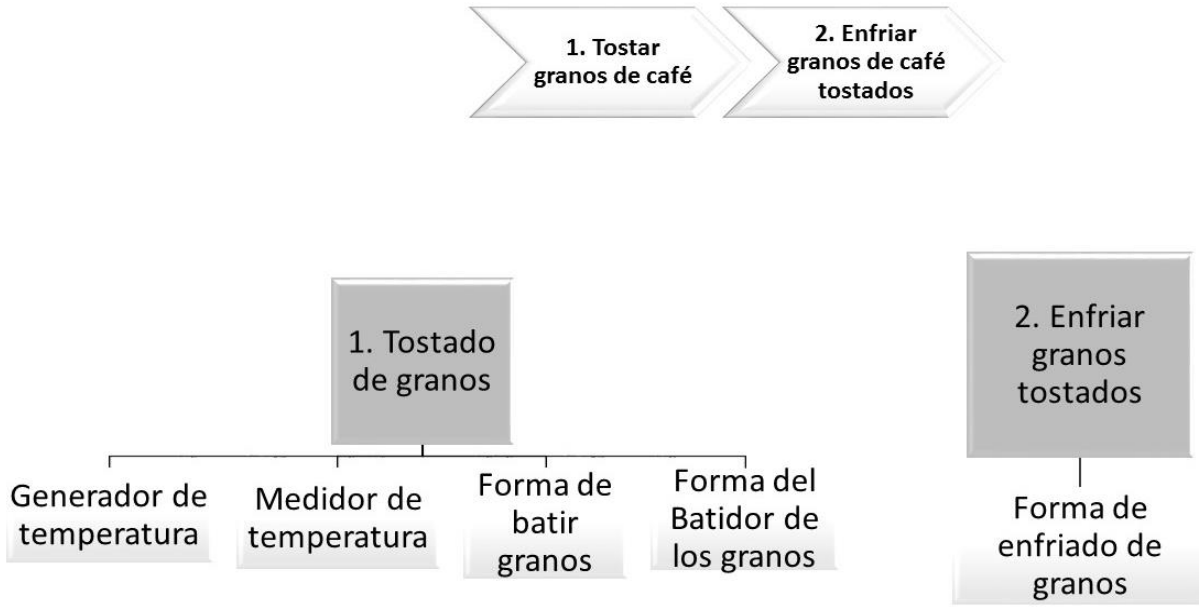
.

Tabla 2.Requerimientos del cliente

Nombre	Objetivo	Descripción
Tostado	• Tostado Americano • Tostado Italiano	Se desea lograr dos tipos de tostado de diferente calidad.
Enfriado	Temperatura del Café a 50°C	El café luego de ser tostado debe ser enfriado a una temperatura ambiente.
Presupuesto	3 SMMLV o menos para componentes y materiales; 3 SMMLV o menos para manufactura y pruebas.	Es el presupuesto máximo autorizado por la Universidad para Proyectos de Maestría
Tiempo	16 semanas, equivalente a 9 horas por semana.	Regulaciones de la Universidad para completar Tesis I
Capacidad	Al menos 1 kilogramo de café tostado en un ciclo de 15 minutos	Limitación del cliente

Fuente: esta investigación 2017

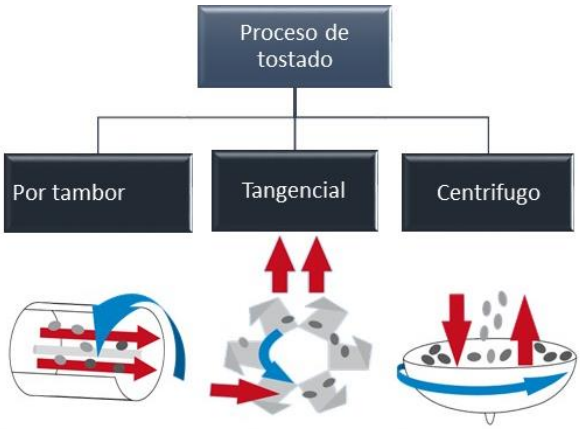
Figura 6.Funcionalidades Primarias Sistema



Realizada por Sergio A. Naranjo C.

Figura 7. Estado del arte Tostado de café.

Tipo de Tostado	Temperatura	Tiempo
Americano	193 °C	6-9 mins
Francés	200 °C	7-10 mins
Italiano	217.5 °C	8- 11 mins



Máquina de tostado	Batch	Tiempo
Por tambor	20-50 Kg	8-20 mins
Tangencial	50-75 kg	5.5- 18 mins
Centrifugo	210-640 Kg	5.5- 18 mins

Tomado de (Veracruz, s.f.) (Probat Saturn Tostadora Centrífuga, 2017) (Probat Neptune Tostadoras de tambor, 2017) (Probat Jupiter Tostadoras Tangenciales, 2017)

Tabla 3. Conceptos de las funciones de la máquina

Funciones	Concepto
Temperatura	El sistema debe mantener una temperatura constante para cumplir con la tostadura deseada de acuerdo a la selección del usuario
Sensores de temperatura	Son necesarios para el monitoreo de la temperatura del proceso y la corrección de la misma por el control.
Forma de batir los granos	El café a ser tostado debe estar en movimiento para asegurar la calidad del proceso

Forma del batidor los granos	La forma del batidor influye en cómo se mueven los granos en el compartimento tostador, se debe garantizar homogeneidad en el movimiento de estos.
Enfriado	El proceso de tostado debe ser detenido por lo que la temperatura debe bajar rápidamente.

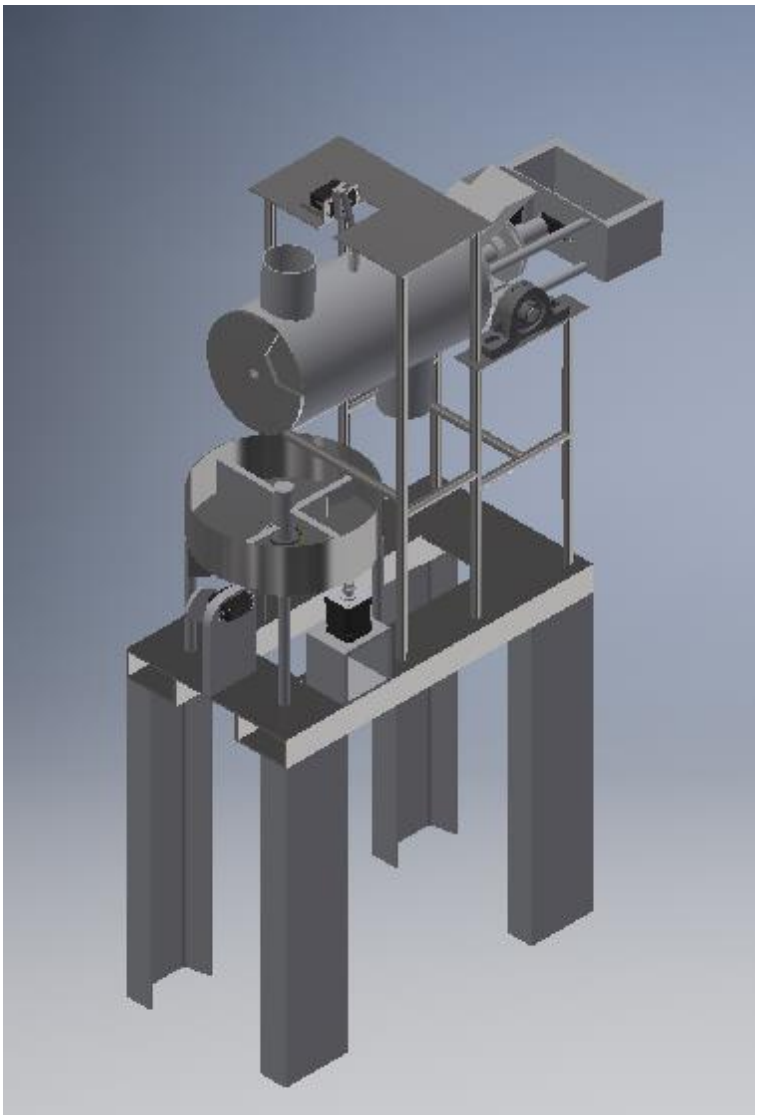
Fuente: esta investigación 2017

Tabla 4. Medios físicos

		Costo		**Rendimiento**		**Tamaño**		**Implementación**		**Confiabilidad**		**Rango**
		0,25		0,25		0,2		0,2		0,1		1
	Alternativas											
Temperatura	Resistencia eléctica	8	2,00	6	1,50	9	1,80	8	1,60	8	0,80	7,7
	Quemador Diesel	5	1,25	8	2,00	3	0,60	4	0,80	8	0,80	5,45
	Quemador de Gas	6	1,50	7	1,75	4	0,80	4	0,80	8	0,80	5,65
Sensores de temperatura	RTD	5	1,25	8	2,00	8	1,60	8	1,60	9	0,90	7,35
	Termistores	7	1,75	6	1,50	8	1,60	8	1,60	6	0,60	7,05
	Termocuplas	8	2,00	7	1,75	8	1,60	9	1,80	8	0,80	7,95
	Termotransistores	9	2,25	6	1,50	8	1,60	8	1,60	7	0,70	7,65
Forma de batir los granos	DC motor	8	2,00	7	1,75	8	1,60	9	1,80	7	0,70	7,85
	Motor paso a paso	9	2,25	6	1,50	8	1,60	7	1,40	8	0,80	7,55
	Servomotor	7	1,75	9	2,25	8	1,60	8	1,60	9	0,90	8,1
Forma del batidor los granos	Liso	9	2,25	7	1,75	8	1,60	9	1,80	8	0,80	8,2
	Con ranuras	7	1,75	9	2,25	8	1,60	9	1,80	9	0,90	8,3
Enfriado	Aspersores agua	7	1,75	8	2,00	7	1,40	6	1,20	8	0,80	7,15
	Aire frio	7	1,75	7	1,75	7	1,40	8	1,60	9	0,90	7,4

Fuente: esta investigación 2017

Figura 8. Modelo CAD Tostadora de café



Fuente: esta investigación 2017

A. Análisis Dinámico

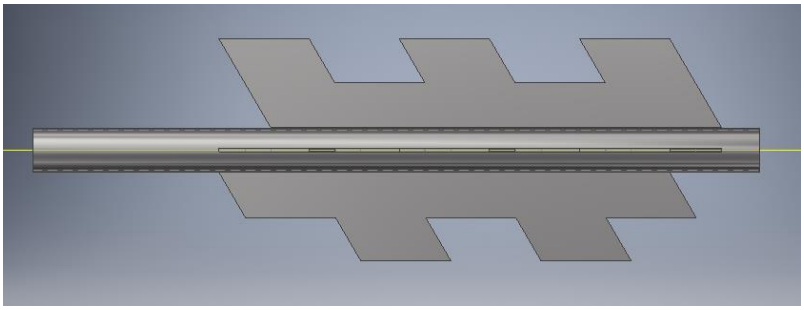
Diseño mecánico

En este análisis se consideró las matrices de inercia de los ejes principales proporcionadas por Autodesk® inventor®, con sus grados de rotación para el eje del respectivo componente. De esta forma, se utilizó la ecuación 1 para realizar los cálculos del par requerido para cada motor, ya que la matriz de inercia está ubicada en la trama del eje de rotación.

$$\boldsymbol{M^{B*} = I^{B/B^*} \cdot {}^{A}_{B}\boldsymbol{\alpha} + {}^{A}_{B}\boldsymbol{\omega} \times (I^{B/B^*} \cdot {}^{A}_{B}\boldsymbol{\omega})} \quad \textbf{ecu.1}$$

Eje batidor del tambor

Figura 8.Eje batidor Tostador



Fuente: esta investigación 2017

La matriz de inercia en los ejes principales para el eje batidor es:

$$\left(\boldsymbol{I^{B/B^*}}\right)_{\boldsymbol{a}} = \begin{bmatrix} 8{,}01634 & 0 & 0 \\ 0 & 75{,}07851 & 0 \\ 0 & 0 & 75{,}10751 \end{bmatrix} Kg\ cm^2$$

Los grados de rotación son:

x	**Y**	**z**	
-25,50	0	0	**Grados**
-0,445058959	0	0	**Radianes**

Con estos datos, la matriz de inercia rotada se calculó, y con una velocidad angular definida por el usuario, el torque requerido para el motor 1 es calculado.

$$\left(\boldsymbol{I^{B/B^*}}\right)_{\boldsymbol{b}} = \begin{bmatrix} I_1 & 0 & 0 \\ 0 & I_2C_\theta^2 + I_3S_\theta^2 & I_3S_\theta C_\theta - I_2S_\theta C_\theta \\ 0 & I_3S_\theta C_\theta - I_2S_\theta C_\theta & I_3C_\theta^2 + I_2S_\theta^2 \end{bmatrix}$$

$$\left(\boldsymbol{I^{B/B^*}}\right)_{\boldsymbol{b}} = \begin{bmatrix} 8{,}01634 & 0 & 0 \\ 0 & 75{,}08338849 & -0{,}01126862 \\ 0 & -0{,}01126862 & 75{,}1021351 \end{bmatrix} Kg\ cm^2$$

La velocidad angular seleccionada es constante, por lo que la aceleración angular no existe. Esta velocidad es seleccionada para batir el café.

$$\prescript{A}{B}{\omega} = \begin{bmatrix} 0{,}5 \\ 0 \\ 0 \end{bmatrix} Rad/s$$

Usando la ecuación 1 se encuentra el torque requerido por el motor 1 para el eje batidor del compartimento de tostado, tal que:

$$\boldsymbol{M^{B^*} = 0{,}196533602Nm}$$

Usando un factor de seguridad de 1.5 se tiene entonces que el factor de seguridad requerido es:

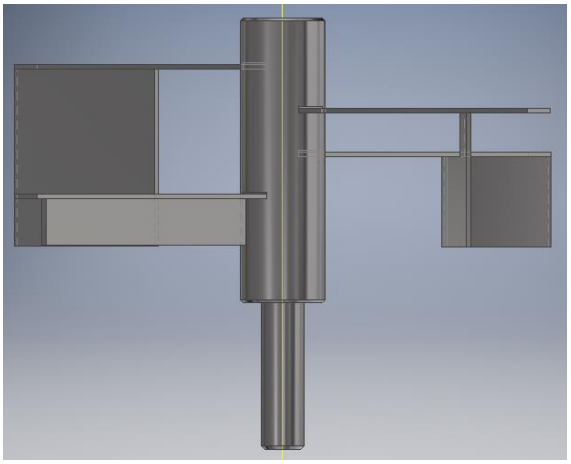
$$\boldsymbol{M^{B^*}} = 0{,}196533602 * 1.5 = \boldsymbol{0{,}2948Nm}$$

La potencia requerida para el movimiento del eje y que debe ser proporcionada por el motor es:

$$\boldsymbol{Potencia = M^{B^*} * \prescript{A}{B}{\omega}} = 0{,}2948 * 0{,}5 = \boldsymbol{0{,}1474\ W}$$

Eje Batidor para el enfriador

Figura 9 Eje batidor Enfriador



Fuente: esta investigación 2017

La matriz de inercia en los ejes principales del eje es:

$$\left(\boldsymbol{I^{B/B^*}}\right)_{\boldsymbol{a}} = \begin{bmatrix} 6{,}85853 & 0 & 0 \\ 0 & 8{,}88171 & 0 \\ 0 & 0 & 8{,}11767 \end{bmatrix} Kg\ cm^2$$

Los grados de rotación son:

X	**Y**	**z**	
-7,61	-14,33	-5,78	**Grados**
-0,132819556	-0,25010568	-0,10088003	**Radianes**

Con estos datos, la matriz de inercia rotada fue calculada, y con esta, más la velocidad angular definida por el usuario, se calculó el torque requerido por el motor. Para calcular la matriz de rotación se utilizó la notación de ángulos de Euler Z-Y-X, de esta forma, utilizando la ecuación 2 se calculó la matriz de inercia rotada.

$$\left(\boldsymbol{I^{B/B^*}}\right)_{\boldsymbol{b}} = {}^{\boldsymbol{A}}_{\boldsymbol{B}}\boldsymbol{R}\left(\boldsymbol{I^{B/B^*}}\right)_{\boldsymbol{a}} {}^{\boldsymbol{A}}_{\boldsymbol{B}}\boldsymbol{R}' \textbf{ ecu.2}$$

$${}^{\boldsymbol{A}}_{\boldsymbol{B}}\boldsymbol{R}_{\boldsymbol{Z'Y'X'}}(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma}) = \boldsymbol{R_z}(\boldsymbol{\alpha})\boldsymbol{R_y}(\boldsymbol{\beta})\boldsymbol{R_x}(\boldsymbol{\gamma}) = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + c\alpha c\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}$$

$${}^{\boldsymbol{A}}_{\boldsymbol{B}}\boldsymbol{R} = \begin{bmatrix} 0{,}963960378 & -0{,}067211538 & -0{,}257416002 \\ 0{,}097575579 & 0{,}989454095 & 0{,}107049525 \\ 0{,}247506354 & -0{,}128309016 & 0{,}960352748 \end{bmatrix}$$

$${}^{\boldsymbol{A}}_{\boldsymbol{B}}\boldsymbol{R}' = \begin{bmatrix} 0{,}963960378 & 0{,}097575579 & 0{,}247506354 \\ -0{,}067211538 & 0{,}989454095 & -0{,}128309016 \\ -0{,}257416002 & 0{,}107049525 & 0{,}960352748 \end{bmatrix}$$

$$\left(\boldsymbol{I^{B/B^*}}\right)_{\boldsymbol{b}} = \begin{bmatrix} 6{,}950606973 & -0{,}161928886 & -0{,}294773237 \\ -0{,}161928886 & 8{,}745999588 & -0{,}113443181 \\ -0{,}294773237 & -0{,}113443181 & 8{,}051303438 \end{bmatrix} Kg\ cm^2$$

La velocidad angular seleccionada es constante, por lo que la aceleración angular no existe. Esta velocidad es seleccionada para batir el café.

$$ {}^{A}_{B}\omega = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} Rad/s $$

Usando la ecuación 1 se encuentra el torque requerido por el motor 1 para el eje batidor del compartimento de tostado, tal que:

$$ \boldsymbol{M^{B^*} = 0,789563 Nm} $$

Usando un factor de seguridad de 1.5 se tiene entonces que el factor de seguridad requerido es:

$$ \boldsymbol{M^{B^*}} = 0{,}789563 * 1.5 = \boldsymbol{1,184345 Nm} $$

La potencia requerida para el movimiento del eje y que debe ser proporcionada por el motor es:

$$ \boldsymbol{Potencia = M^{B^*} * {}^{A}_{B}\omega} = 1{,}184345 * 1 = \boldsymbol{1,184345\ W} $$

Diseño eléctrico

Componentes

En esta sección se muestran las características de motores, sensores y controlador seleccionado para el prototipo desarrollado. La parte más importante del sistema electrónico es el microcontrolador, ya que este permite que el software controle los actuadores, lea y escriba las señales de los sensores.

Hay 5 mecanismos en el sistema, los cuales son rotacionales; dos de estos de giro continuo y los otros tres de posicionamiento. Estos mecanismos son reversibles, por lo que la selección de motores paso a paso para los de giro continuo se dio gracias a su fácil cambio de dirección con el driver correspondiente.

Los 3 mecanismos de posicionamiento se dan gracias al uso de servomotores con optocopladores, lo que hace fácil saber en que posición se encuentra cada uno.

Los motores paso a paso pueden proporcionar un movimiento de rotación de baja velocidad, se agregó una transmisión por correa a una polea acoplada al eje para amplificar el torque necesario para batir el café en los compartimentos de tostado y enfriado. Debido a su facilidad de uso y sin necesidad de tolerancias de posición cerrada, los motores paso a paso se eligieron sobre los servomotores y los motores de CC.

Para el control de este tipo de motores se necesita el driver Allegro A4988 para motores de hasta 2 A de corriente por bobina. Tiene cinco resoluciones diferentes, para esta aplicación se utilizó la de paso completo, tiene control de corriente ajustable que permite ajustar la salida de corriente máxima con un potenciómetro, que le permite utilizar tensiones superiores a la tensión nominal del motor paso a paso para lograr mayores tasas de paso y potencia. Este necesita por parte del controlador dos señales de salida para dar la dirección de giro y la cantidad de pasos a dar.

Los servomotores de media vuelta al tienen una caja reductora con la que proporcionan el torque necesario para mover las partes en la posición deseada. Estos cuentan con un optocoplador en el eje de salida, por lo que es fácil el seguimiento de la posición desde el controlador. Estos necesitan del controlador una salida de modulación por ancho de pulsos, PWM (pulse width modulation), lo cual da la posición exacta del mecanismo.

Como sensores se tiene una termocupla tipo J, con la cual se mide la temperatura de tostado. Esta genera en sus extremos un voltaje el cual es procesado por el microcontrolador con el módulo ADC (Analog Digital Converter).

Para el controlador se seleccionó la tarjeta de desarrollo Arduino Mega tipo open-source que posee pines de entradas y salidas (E/S), analógicas y digitales. Esta tarjeta es programada en un entorno de desarrollo que implementa el lenguaje Processing/Wiring. Esta tarjeta es compatible con los motores y los sensores.

Para los indicadores de proceso se utilizaron diodos emisores de luz (LED), con su respectiva resistencia de protección.

En la Tabla 5. Se resumen las características de los diferentes componentes.

Tabla 5. Resumen características componentes electrónicos

Componente	Características
Motor paso a paso Nema 17	Ángulo de paso:1.8±5% grados Voltage: 8-12V DC Corriente: 0.7 A/phase Par: 400 mN.m Min Peso:275g
Servomotor MG996R	Engranajes de metal Par: 883 mN.m Voltaje: 4,8 - 7.2Volts Peso: 55 g (poco más de 2 oz) de Angulo: 180 grados
Termocupla Tipo J	Positivo: Hierro Negativo: cobre/nickel Rango de temperatura (-180, 750) 42.2
Driver A4988	Voltaje operación: 5 V Corriente de control permitida: 2 A
Arduino Mega 2560	Microcontrolador ATmega2560. Voltaje de entrada de – 7-12V. 54 pines digitales de Entrada/Salida (14 de ellos son salidas PWM). 16 entradas análogas. 256k de memoria flash.
Relé 1	Voltaje operación: 5 V Corriente de control permitida: 3 A
Relé 2	Voltaje operación: 12 V Corriente de control permitida: 24 A
LED Diodo emisor de luz	Voltaje operación: 2.2V

	Corriente de control permitida: 10 mA

Fuente: esta investigación 2017

Diseño de software

Con la selección de los componentes se identificaron las entradas y salidas del sistema sí como los nombres de los mismos, estos se muestran en la Tabla 6.

Tabla 6. Nombres de actuadores y entradas y salidas del sistema

Función	Tipo de Motor	Etiqueta
Mover eje Batidor de café al tostar	Motor Paso a Paso Nema 17	Motor 1
Inclinar compartimiento de tostado	Servomotor MG996R	Motor 2
Abrir tapa de compartimiento tostador	Servomotor MG996R	Motor 3
Mover eje Batidor de café al enfriar	Motor Paso a Paso Nema 17	Motor 4
Abrir tapa de compartimiento enfriador	Servomotor MG996R	Motor 5

Señal de control	I\O's
Botón de inicio proceso	In
Dirección Motor 1	Out
Paso Motor 1	Out
Posición Motor 2	Out
Posición Motor 3	Out
Dirección Motor 4	Out
Paso Motor 4	Out
Posición Motor 5	Out
Sensor temperatura	In
Turbina Soplado de Aire On/Off	Out
Resistencias de Calor On/off	Out
Led de monitoreo 1	Out
Led de monitoreo 2	Out
Led de monitoreo 3	Out
Led de monitoreo 4	Out

Led de monitoreo 5	Out

Fuente: esta investigación 2017

Para el diseño de software inicialmente se describió la rutina más simple para conseguir las funcionalidades principales de la máquina, este proceso se describe a continuación.

Inicio programa

Ingreso café manual

Botón de inicio funcionamiento (entrada de inicio)

Rutina de pre-configuración del sistema para inicio

1. *Soplador de aire caliente apagado (salida calentador LOW)*
2. *Posición 1 tapa 1 tambor (salida servomotor 1 0°)*
3. *Posición 1 tapa 2 enfriador (salida servomotor 2 0°)*
4. *Posición 1 (horizontal) tambor (salida motor paso a paso 3 HIGH hasta entrada final de carrera 1 HIGH)*
5. *Eje batidor tambor apagado (salida motor paso a paso 1 LOW)*
6. *Eje batidor enfriador apagado (salida motor paso a paso 2 LOW)*

   Tostado
7. *Procesos simultáneos*
   - *Inicio de soplado de aire caliente (salida calentador HIGH)*
   - *Inicio giro eje batidor de café tambor (salida al motor paso a paso 1 HIGH 2 rad/s)*
   - *Tiempo de espera de tostado (RETARDO por determinar según calidad de tostado 5-10 mins)*

8. *Fin de soplado de aire caliente (salida calentador LOW)*
9. *Posición 2 tapa 1 tambor (salida servomotor 1 180°)*

Enfriado

10. *Posición 1 (horizontal) tambor (salida motor paso a paso 3 180°)*
11. *Inicio giro eje batidor de café enfriador(salida al motor paso a paso 2 HIGH 1.5 rad/s)*
12. *Tiempo de espera de salida café tambor (RETARDO por determinar según salida del café .5-1 min)*
13. *Fin giro eje batidor de café tambor (salida al motor paso a paso 1 LOW)*
14. *Tiempo de espera de enfriado (RETARDO por determinar según temperatura del café 3-5 mins)*
15. *Posición 2 tapa 2 enfriador (salida servomotor 2 180°)*
16. *Tiempo de espera de salida café enfriador (RETARDO por determinar según salida del café 1-2 min)*

Pre-configuración de la máquina

17. *Eje batidor enfriador apagado (salida motor paso a paso 2 LOW)*
18. *Posición 1 tapa 1 tambor (salida servomotor 1 0°)*
19. *Posición 1 tapa 2 enfriador (salida servomotor 2 0°)*
20. *Posición 1 (horizontal) tambor (salida motor paso a paso 3 HIGH hasta entrada final de carrera 1 HIGH)*
21. *Espera de pulsación botón de inicio (entrada de inicio)*

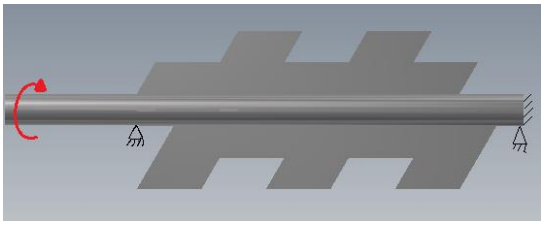
B. Diseño estructural

Diseño mecánico

En el diseño estructural se contemplaron los esfuerzos en los ejes batidores como los principales a sufrir alguna deformación. Esto se muestra a continuación.

Eje batidor del tostador

Figura 10. Eje batidor Tostador Análisis Esfuerzos



Fuente: Esta investigación 2017

Para el análisis de esfuerzo cortante se consideró el peor caso, i.e., donde el eje está anclado al final de su extremo derecho, y el torque calculado es aplicado en su totalidad en el opuesto. Para el análisis del esfuerzo normal flector, el peso del motor conectado se tomó como el momento flector. El esfuerzo permisible del acero inoxidable fue obtenido del estándar "UNE-EN 10088-2:2008".

Eje batidor del Tostador

Entradas	Torque	0,0002948	Nm
	Momento Flector	5	Nm
	Diámetro Externo	0,025	m
	Diámetro Interno	0,0225	m
	Esfuerzo Cortante Permisible	210	MPa
	Esfuerzo Normal Permisible	540	MPa

Fuente: esta investigación 2017

Las siguientes formulas fueron utilizadas para calcular los correspondientes momentos de inercia y los esfuerzos.

$$\tau = \frac{T * c}{J} \; ; J = \frac{\pi}{2}\left(\left(\frac{c_1}{2}\right)^4 - \left(\frac{c_2}{2}\right)^4\right)$$

$$\sigma = \frac{M * c}{I} \; ; I = \frac{\pi}{4}\left(\left(\frac{c_1}{2}\right)^4 - \left(\frac{c_2}{2}\right)^4\right)$$

Por lo tanto, los esfuerzos a los que va a estar sometido el eje se muestran en la Tabla 7.

Tabla 7.Esfuerzos eje tostador

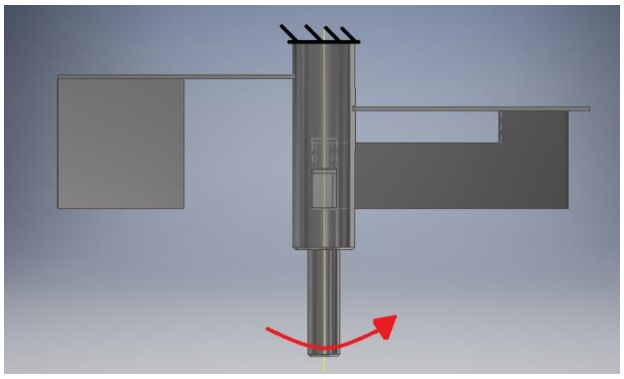
Cálculos	Momento polar de inercia	1,31884E-08	m^4
	Esfuerzo Cortante	**0,297**	**KPa**
	Primer momento de Inercia	6,5942E-09	m^4
	Esfuerzo normal flector	**9,48**	**MPa**

Fuente: esta investigación 2017

Con estos resultados, el eje no va a sufrir deformaciones por los torques aplicados en él.

Eje batidor del enfriador

Figura 11. Eje batidor Enfriador Análisis Esfuerzos



Fuente: esta investigación 2017

Para el análisis de esfuerzo cortante se consideró el peor caso, i.e., donde el eje está anclado en su parte superior, y el torque calculado es aplicado en su totalidad en la parte inferior. Para este caso, el efecto del esfuerzo normal flector no fue considerado.

Eje batidor del enfriador

Entradas	Torque	0,001184345	Nm
	Diámetro 1	0,025	m
	Diámetro 2	0,0127	m
	Esfuerzo cortante permisible	210	MPa

Fuente: esta investigación 2017

Las siguientes formulas fueron usadas para calcular el respectivo momento de inercia y esfuerzo.

$$\tau = \frac{T * c}{J} \; ; J = \frac{\pi}{2}\left(\left(\frac{c_1}{2}\right)^4 - \left(\frac{c_2}{2}\right)^4\right)$$

Tabla 8.Esfuerzos eje batidor enfriador

	Momento polar de inercia 1	3,83495E-08	m^4
Cálculo	Momento polar de inercia 2	2,55396E-09	m^4
	Esfuerzo Cortante 1	**0,386**	**KPa**

Esfuerzo Cortante 2	**2,94**	**KPa**

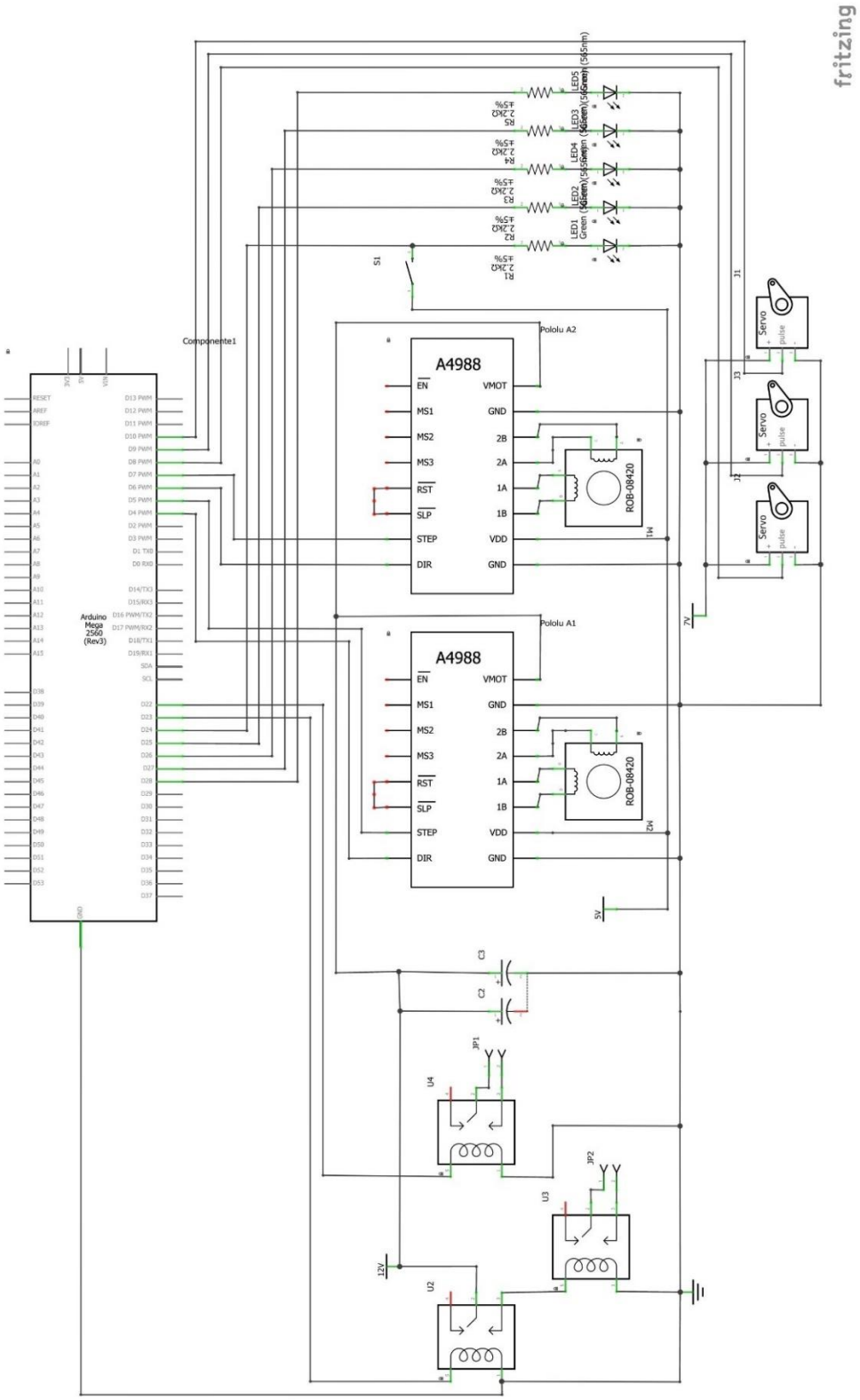
Fuente: esta investigación 2017

Con estos resultados, el eje no sufrirá deformaciones por el torque aplicado en él.

Diseño eléctrico

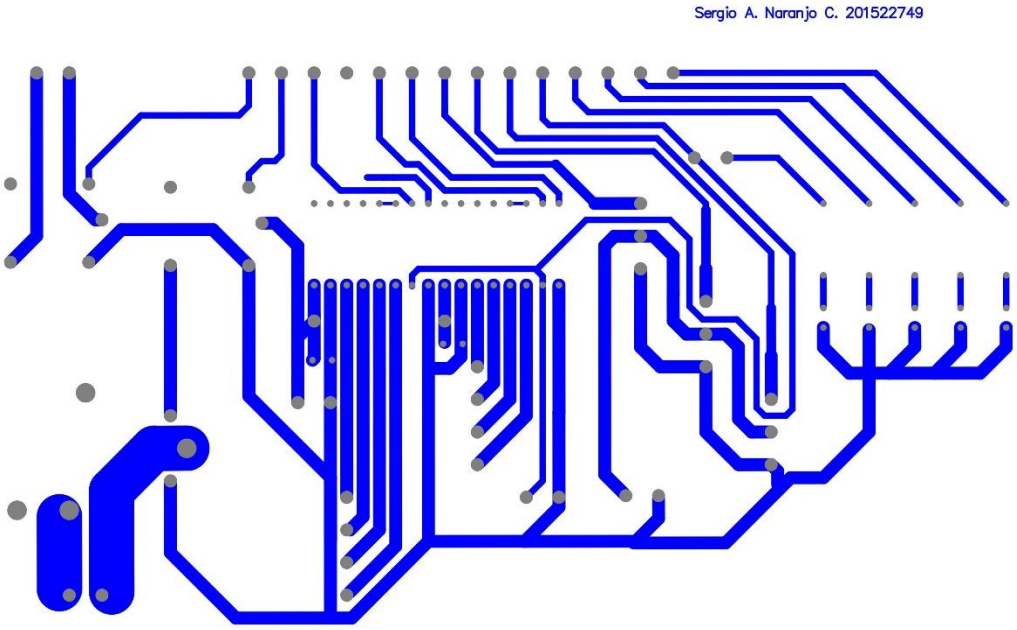
Para la integración de los componentes se diseñó una placa de circuito impreso (PCB), la cual incorpora los indicadores de estado de la máquina, así como los sistemas de activación del sistema de aire forzado caliente (relés). Para el diseño de esta se tuvo en cuenta la cantidad de entradas y salidas del controlador y la alimentación necesaria para los motores, controlador y los contactores y sensores. En la Figura 12 se muestra el circuito esquemático y en la Figura 13 el diagrama de la PCB.

Figura 12.Circuito esquemático



Realizado por Sergio A. Naranjo C.

Figura 13. Diagrama PCB del circuito de control.



Realizado por Sergio A. Naranjo C.

Diseño de software

Como se enunció en la metodología, se desarrollaron las diferentes etapas de RUP; modelando los requerimientos del cliente, los cuales se tienen en un caso de uso de requerimientos (ver figura), y su respectiva descripción. Se tiene un diagrama de Diseño (arquitectura) donde se ven las variables involucradas en el proceso y su interrelación (

Figura 15). Se desarrolló un diagrama de máquina de estados que evidencia el proceso y las señales que se envían (Fuente: esta investigación 2017

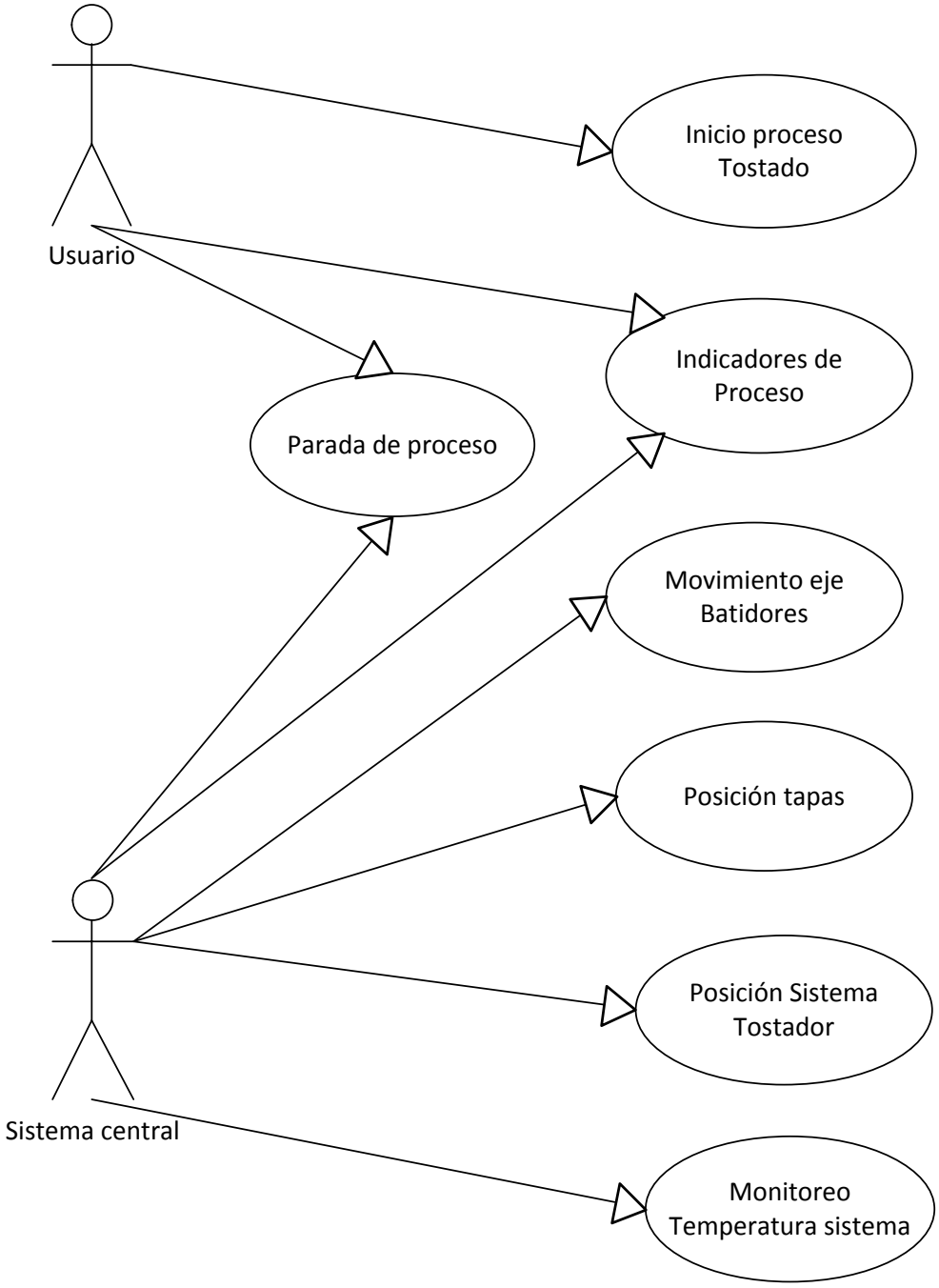
Figura 16,

Fuente: esta investigación 2017

Figura 17,

Figura 18, , Figura 20).

Figura 14. Caso de Usos de requerimientos



Fuente: esta investigación 2017

Descripción de casos de uso

Nombre: Inicio proceso de tostado

Actores: Usuario

Función: Como usuario quiero enviar señal para el inicio del proceso de tostado por medio de un botón

Descripción de casos de uso: Se oprime el botón de encendido para dar inicio a la pre-configuración del sistema.

Nombre: Indicadores de proceso.

Actores: Usuario.

Función: Como usuario quiero ver el estado de la máquina.

Descripción de casos de uso: Según las luces que estén encendidas determinar si la máquina está en estado de espera, tostado, transición, enfriado o alarma.

Nombre: Indicadores de proceso.

Actores: Sistema central.

Función: Como sistema central muestra al usuario el estado de la máquina.

Descripción de casos de uso: Enviar la información del estado de la máquina.

Nombre: Parada de proceso.

Actores: Usuario.

Función: Como usuario quiero oprimir el botón de apagado para terminar con todos los procesos existentes.

Descripción de casos de uso: Se oprime el botón de apagado para terminar con todos los procesos.

Nombre: Parada de proceso.

Actores: Sistema central.

Función: Como sistema central para el sistema para terminar con todos los procesos existentes en caso de emergencia.

Descripción de casos de uso: El sistema central por medio del monitoreo de condiciones del sistema decide para el proceso en casos determinados de emergencia.

Nombre: Movimiento eje de batidores.

Actores: Sistema central.

Función: Como sistema central mueve los ejes de los batidores para batir el café.

Descripción de casos de uso: El sistema central envía señal para mover los ejes y batir el café en el tostado y en el enfriado.

Nombre: Posición de tapas.

Actores: Sistema central.

Función: Como sistema central abre o cierra las tapas en el sistema.

Descripción de casos de uso: El sistema central envía señal para mover las tapas según las necesidades.

Nombre: Posición sistema tostador.

Actores: Sistema central.

Función: Como sistema central posiciona el tostador para tostar el café o pasarlo al enfriador.

Descripción de casos de uso: Se posiciona el tostador horizontalmente para poder tostar el café o de forma inclinada para trasportar el café al enfriador.

Nombre: Monitorear Temperatura sistema.

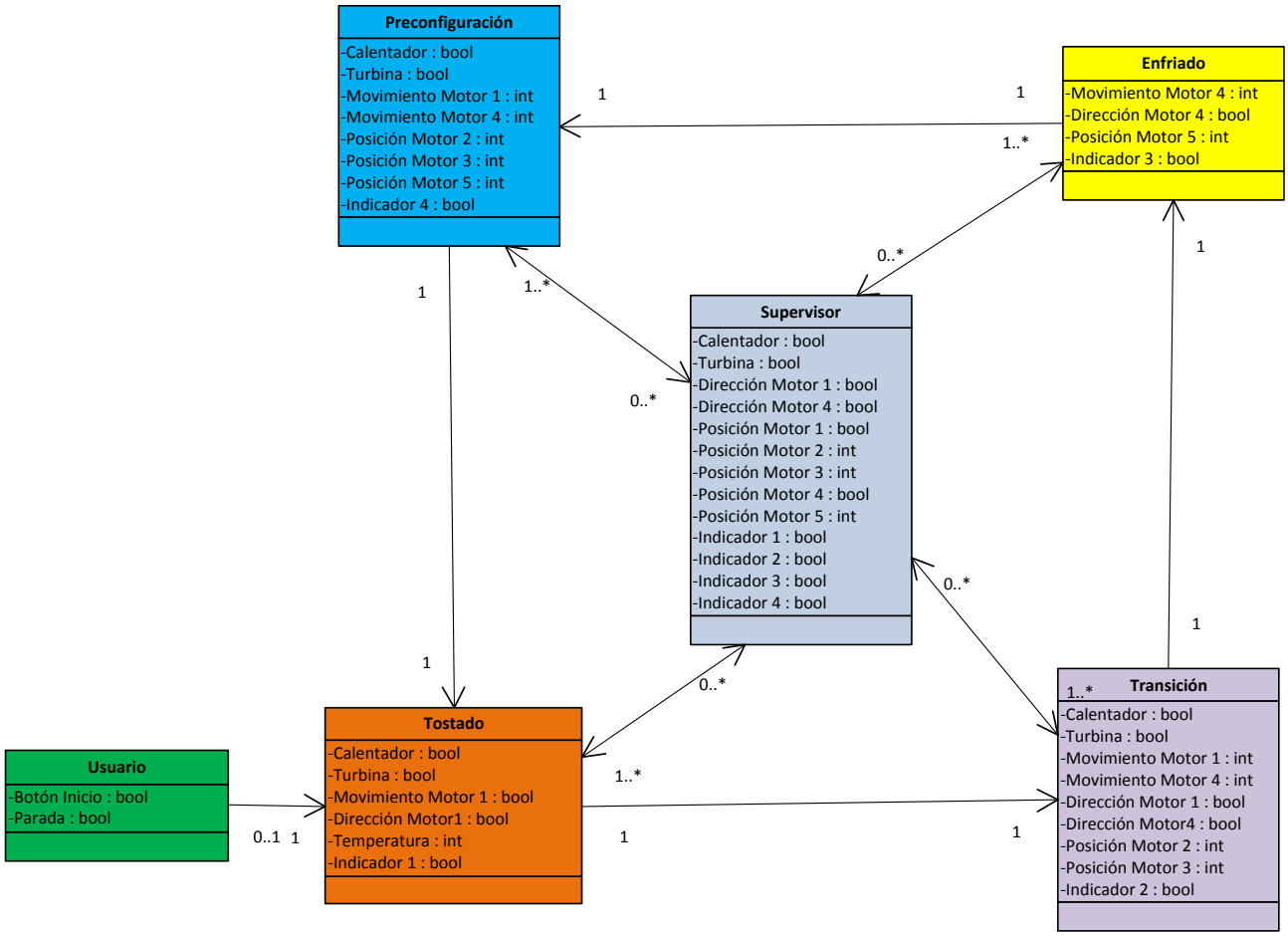
Actores: Sistema central.

Función: Como sistema central monitorea la temperatura del sistema para saber las condiciones del proceso.

Descripción de casos de uso: El sistema central monitorea la temperatura del sistema para tomar decisiones sobre las condiciones del proceso.

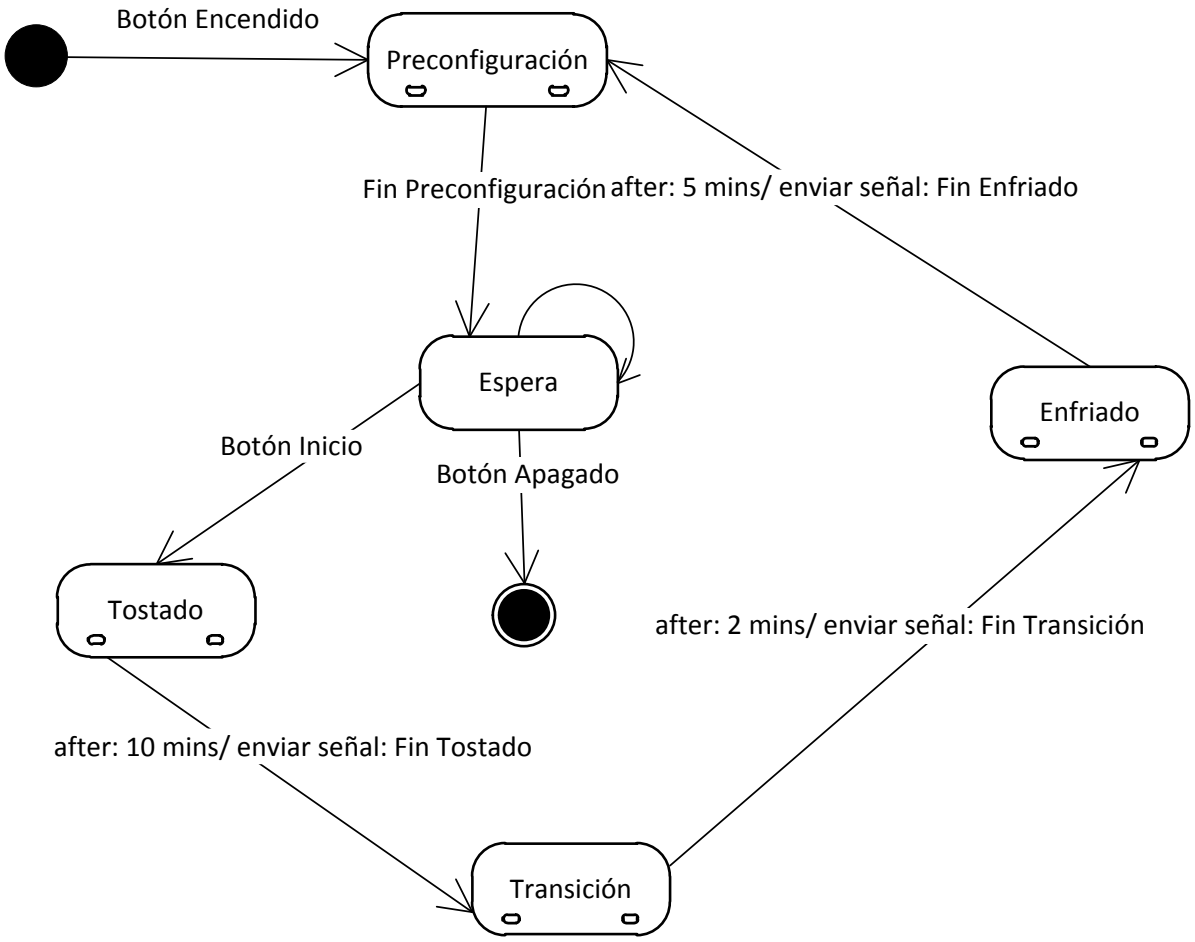
Referencias: Caso de Uso de Requerimientos.

Figura 15. Diagrama de Diseño (Arquitectura de software)



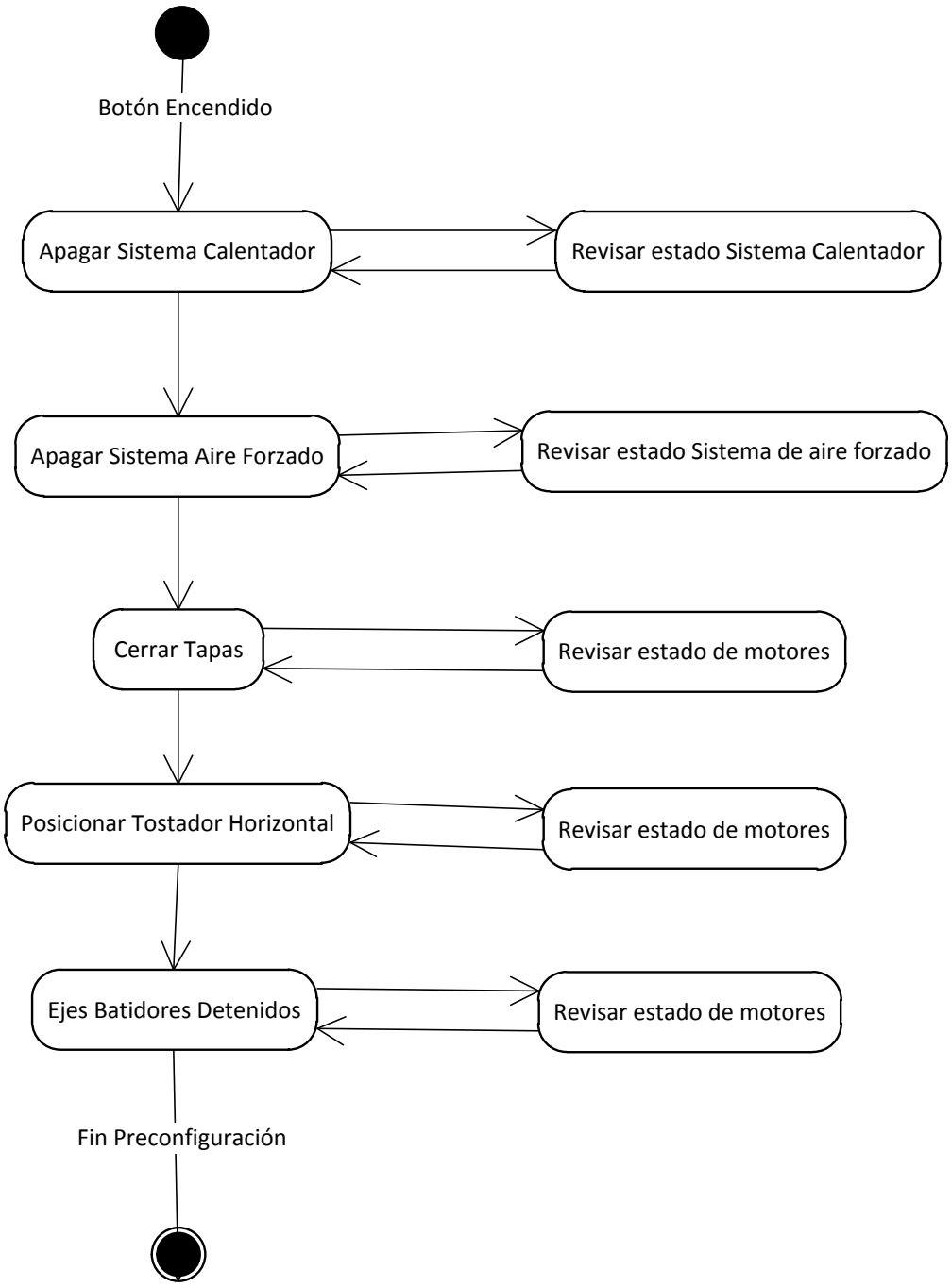
Fuente: esta investigación 2017

Figura 16. Máquina de estados principal



Fuente: esta investigación 2017

Figura 17. Máquina de estados Pre-configuración



Fuente: esta investigación 2017

Figura 18.Máquina de estados tostado

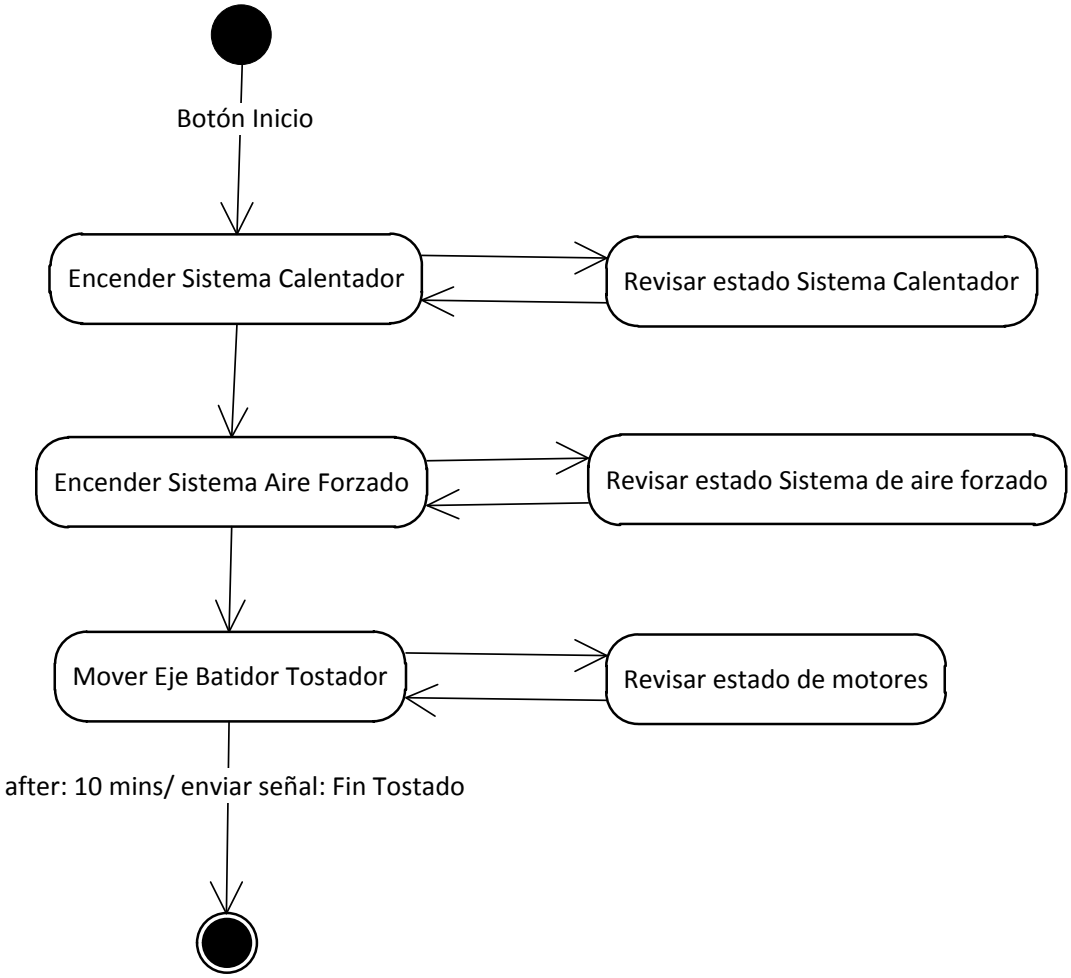
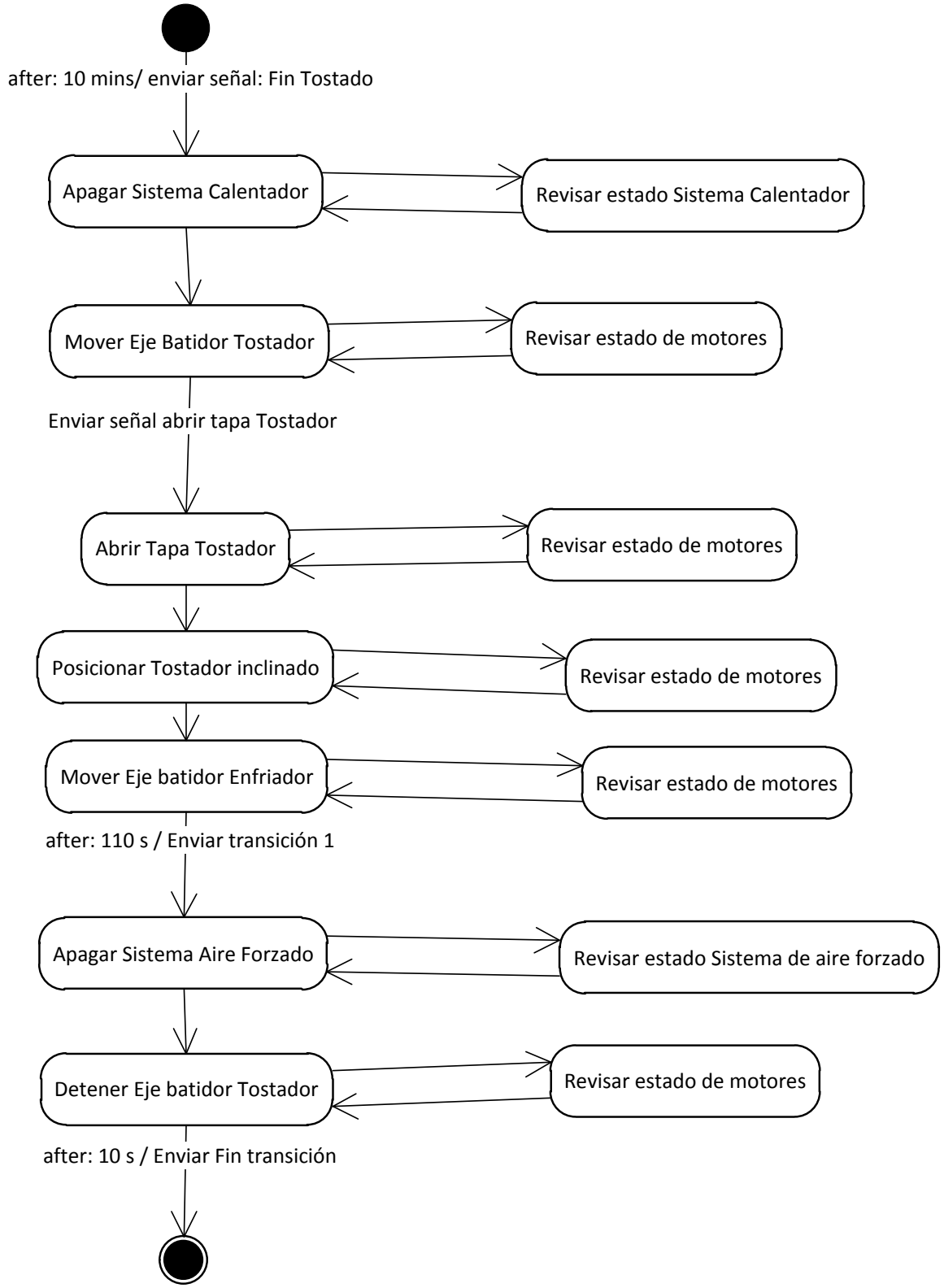
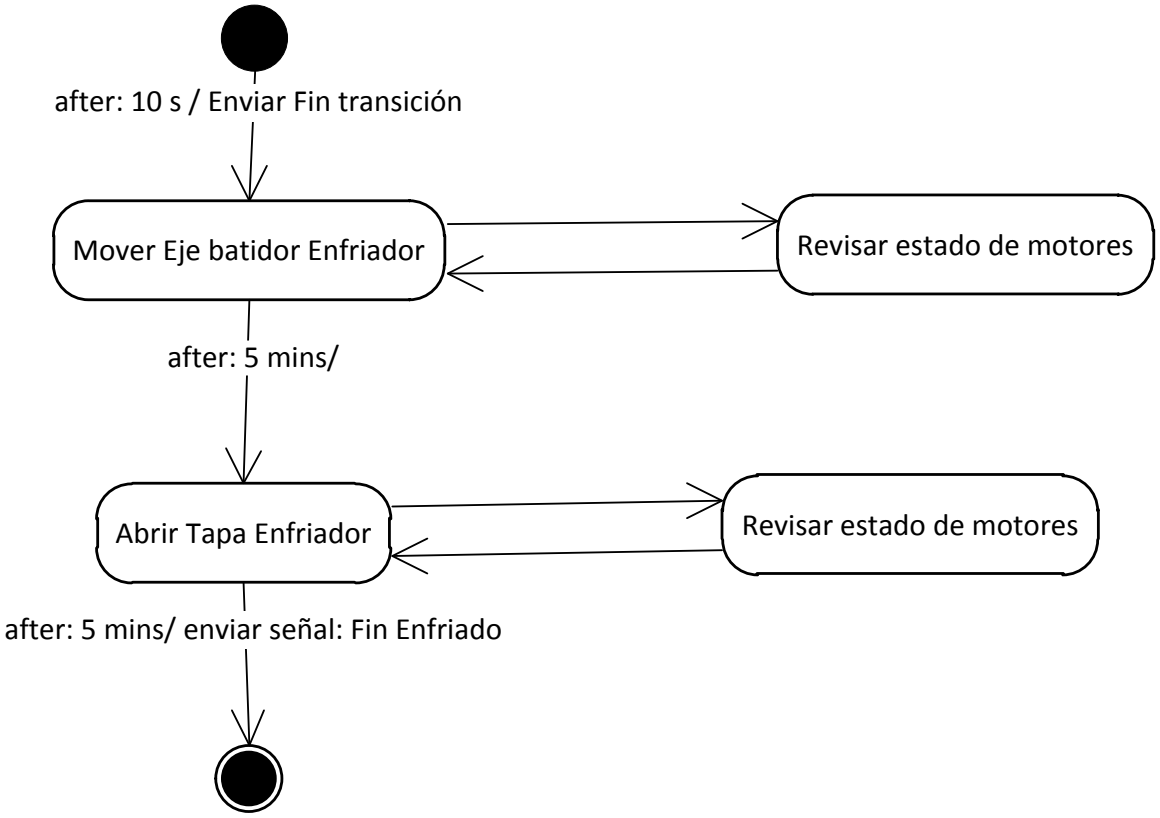


Figura 19. Máquina de estados transición



Fuente: esta investigación 2017

Figura 20 Máquina de estados enfriado



Fuente: esta investigación 2017

Verificación del software.

Para la verificación del software se contemplaron pruebas de tipo funcional, de límites y de fallas inducidas en el sistema para ver el comportamiento, así también los pasos a seguir para la aplicación de la prueba estas se muestran en Tabla 9. A su vez, se tiene la cantidad de pruebas hechas en el prototipo y en la simulación, el resultado esperado y el resultado ocurrido (ver Tabla 10).

Tabla 9. Lista de eventos de prueba Software

No	Prioridad	Tipo	Nombre de la Prueba	Pasos a seguir
TC-1		Modo	Funcionalidad de Tostadora de café	
TC-1.1	Alta	Funcional	Probar Batido de café en el tostador	
TC-1.1-1	Alta		Mover motor 1 sentido horario	Encender el motor 1 con el pin de dirección en alto, ingresar 400 pasos para que de 2 vueltas
TC-1.1-2	Alta		Mover motor 1 sentido Anti-horario	Encender el motor 1 con el pin de dirección en bajo, ingresar 400 pasos para que de 2 vueltas
TC-1.1-3	Alta		Ingresar 0,5 Kg de café y batirlo	Encender el motor 1 ingresar 800 pasos para dar 4 vueltas y 0,5 Kg de café sin tostar
TC-1.1-4	Alta		Ingresar 1 Kg de café y batirlo	Encender el motor 1 ingresar 800 pasos para dar 4 vueltas y 1 Kg de café sin tostar
TC-1.2	Alta	Funcional	Probar Batido de café en el enfriador	
TC-1.2-1	Alta		Mover motor 4 sentido horario	Encender el motor 4 con el pin de dirección en alto, ingresar 400 pasos para que de 2 vueltas
TC-1.2-2	Alta		Mover motor 4 sentido Anti-horario	Encender el motor 4 con el pin de dirección en bajo, ingresar 400 pasos para que de 2 vueltas

TC-1.2-3	Alta		Ingresar 0,5 Kg de café y batirlo	Encender el motor 4 ingresar 800 pasos para dar 4 vueltas y 0,5 Kg de café Tostado
TC-1.2-4	Alta		Ingresar 1 Kg de café y batirlo	Encender el motor 4 ingresar 800 pasos para dar 4 vueltas y 1 Kg de café Tostado
TC-1.3	Alta	Funcional	Probar Inclinación del tambor	
TC-1.3-1	Alta		Mover motor 3 posición 1	Encender el motor 3 y darle la posición de 5°
TC-1.3-2	Alta		Mover motor 3 posición 2	Encender el motor 3 y darle la posición de 150°
TC-1.4	Alta	Funcional	Probar tapa compartimento tostador	
TC-1.4-1	Alta		Abrir tapa compartimento tostador	Encender el motor 2 y darle la posición de 80°
TC-1.4-2	Alta		Cerrar tapa compartimento tostador	Encender el motor 2 y darle la posición de 170°
TC-1.5	Alta	Funcional	Probar tapa compartimento enfriador	
TC-1.5-1	Alta		Abrir tapa compartimento enfriador	Encender el motor 5 y darle la posición de 60°

TC-1.5-2	Alta		Cerrar tapa compartimento enfriador	Encender el motor 5 y darle la posición de 155°
TC-1.6	Alta	Funcional	Probar Sistema de aire forzado Caliente	
TC-1.6-1	Alta		Encender turbina por 5 minutos	Encender Turbina por 5 minutos
TC-1.6-2	Alta		Encender Turbina y calentador por 10 mins	Encender Turbina y calentador por 10 minutos
TC-1.6-3	Alta		Encender Turbina y calentador por 20 mins	Encender Turbina por 20 minutos
TC-1.6-4	Alta		Encender Turbina y calentador por 30 mins	Encender Turbina por 30 minutos
TC-2		Modo	Falla de límites Tostadora de café	
TC-2.1	Media	Límite	Capacidad límite de Tostado	
TC-2.1-1	Media		Ingresar al sistema de tostado 1.5 Kg de café y tostar	Se ingresa al sistema 1.5 kg de café verde y se empieza el ciclo de tostado Normal
TC-2.1-2	Media		Ingresar al sistema de tostado 2 Kg de café y tostar	Se ingresa al sistema 2 kg de café verde y se empieza el ciclo de tostado Normal

TC-2.2	Media	Límite	Capacidad límite de Enfriado	
TC-2.2-1	Media		Ingresar al sistema de tostado 1.5 Kg de café tostado y Enfriar	Se ingresa al sistema 1.5 kg de café tostado y se empieza el ciclo de Enfriado
TC-2.2-2	Media		Ingresar al sistema de tostado 2 Kg de café tostado y Enfriar	Se ingresa al sistema 2 kg de café tostado y se empieza el ciclo de Enfriado
TC-2.3	Media	Límite	Probar Posición límite Motores	
TC-2.3-1	Media		Llevar el motor 2 a posición 180°	Encender el motor 2 y desde la posición 0°, darle la posición de 180°
TC-2.3-2	Media		Llevar el motor 3 a posición 180°	Encender el motor 3 y desde la posición 0°, darle la posición de 180°
TC-2.3-3	Media		Llevar el motor 5 a posición 180°	Encender el motor 5 y desde la posición 0°, darle la posición de 180°
TC-2.4	Media	Límite	Probar Temperatura límite sistema aire forzado	
TC-2.4-1	Media		Llevar temperatura de proceso a 210°C	Encender el sistema de aire forzado caliente hasta alcanzar la temperatura de 210°C
TC-2.4-2	Media		Llevar temperatura de proceso a 220°C	Encender el sistema de aire forzado caliente hasta alcanzar la temperatura de 220°C

TC-3	Media	Modo	Fallas inducidas Tostadora de café	
TC-3-1	Media	Falla	Probar Fallas en los motores Tostadora de café	
TC-3.1-1	Media		Motor 1 Siempre encendido	Alimentar el driver del motor directamente
TC-3.1-2	Media		Motor 2 Siempre encendido	Enviar señal PWM constante
TC-3.1-3	Media		Motor 3 Siempre encendido	Enviar señal PWM constante
TC-3.1-4	Media		Motor 4 Siempre encendido	Enviar señal PWM constante
TC-3.1-5	Media		Motor 5 Siempre encendido	Alimentar el driver del motor directamente
TC-3.1-6	Media		Motor 1 Siempre Apagado	Desconectar alimentación motor
TC-3.1-7	Media		Motor 2 Siempre Apagado	Desconectar alimentación motor
TC-3.1-8	Media		Motor 3 Siempre Apagado	Desconectar alimentación motor

TC-3.1-9	Media		Motor 4 Siempre Apagado	Desconectar alimentación motor
TC-3.1-10	Media		Motor 5 Siempre Apagado	Desconectar alimentación motor
TC-3.1-11	Media		Posición 2 del motor 2 sistema de calefacción encendido	Encender el motor 3 y darle la posición de 170°, encender calentador
TC-3.1-12	Media		Posición 2 del motor 3 sistema de calefacción encendido	Encender el motor 3 y darle la posición de 150°, encender calentador
TC-3.1-13	Media		Posición 2 del motor 5 sistema de calefacción encendido	Encender el motor 5 y darle la posición de 155°, encender calentador
TC-3.2	Media	Falla	Fallas en el sistema de Aire Forzado caliente	
TC-3.2-1	Media		Sistemas de calefacción encendido turbina apagada	encender calentador y apagar turbina

Fuente: esta investigación 2017

Tabla 10. Pruebas y resultados

No	**Número de repeticiones Prototipo**	**Número de Repeticiones Simulación**	**Salida ocurrida**	**Resultado Prototipo**	**Resultado Simulación**
TC-1	Funcionalidad de Tostadora de café				

TC-1.1	Probar Batido de café en el tostador				
TC-1.1-1	5	5	El motor gira en sentido horario 2 vueltas	OK	OK
TC-1.1-2	5	5	El motor gira en sentido antihorario 2 vueltas	OK	OK
TC-1.1-3	5	5	El motor gira moviendo el café	OK	OK
TC-1.1-4	5	5	El motor gira moviendo el café	OK	OK
TC-1.2	Probar Batido de café en el enfriador				
TC-1.2-1	5	5	El motor gira en sentido horario 2 vueltas	OK	OK
TC-1.2-2	5	5	El motor gira en sentido antihorario 2 vueltas	OK	OK
TC-1.2-3	5	5	El motor gira moviendo el café	OK	OK
TC-1.2-4	5	5	El motor gira moviendo el café	OK	OK

TC-1.3	Probar Inclinación del tambor				
TC-1.3-1	5	5	El motor llega a la posición de 5°	OK	OK
TC-1.3-2	5	5	El motor llega a la posición de 150°	OK	OK
TC-1.4	Probar tapa compartimento tostador				
TC-1.4-1	5	5	El motor llega a la posición de 5°	OK	OK
TC-1.4-2	5	5	El motor llega a la posición de 150°	OK	OK
TC-1.5	Probar tapa compartimento enfriador				
TC-1.5-1	5	5	El motor llega a la posición de 5°	OK	OK
TC-1.5-2	5	5	El motor llega a la posición de 150°	OK	OK
TC-1.6	Probar Sistema de aire forzado Caliente				

TC-1.6-1	5	5	La turbina permanece encendida por 5 minutos	OK	OK
TC-1.6-2	5	5	La turbina y el calentador permanecen encendidos por 10 minutos	OK	OK
TC-1.6-3	5	5	La turbina y el calentador permanecen encendidos por 20 minutos	OK	OK
TC-1.6-4	5	5	La turbina y el calentador permanecen encendidos por 30 minutos	OK	OK
TC-2	Falla de límites Tostadora de café				
TC-2.1	Capacidad límite de Tostado				
TC-2.1-1	5	5	El café sale tostado en su mayoría (85%)	OK	OK
TC-2.1-2	5	5	El café sale tostado en partes casi iguales (70%)	NOK	OK
TC-2.2	Capacidad límite de Enfriado				
TC-2.2-1	5	5	El café se enfría en su mayoría (85%)	OK	OK

TC-2.2-2	5	5	El café se enfría en su mayoría (80%)	OK	OK
TC-2.3	Probar Posición límite Motores				
TC-2.3-1	5	5	El motor 2 llega a la posición de 180°	OK	OK
TC-2.3-2	5	5	El motor 3 llega a la posición de 180°	OK	OK
TC-2.3-3	5	5	El motor 5 llega a la posición de 180°	OK	OK
TC-2.4	Probar Temperatura límite sistema aire forzado				
TC-2.4-1	5	5	El sistema de aire forzado caliente alcanza 210°C	OK	OK
TC-2.4-2	5	5	El sistema de aire forzado caliente alcanza 213,5°C	NOK	OK
TC-3	Fallas inducidas Tostadora de café				
TC-3-1	Probar Fallas en los motores Tostadora de café				

TC-3.1-1	5	5	El controlador relaciona el indicador de proceso y determina si debe estar encendido, si no debe estarlo, da la alarma de motor 1 encendido permanentemente	OK	OK
TC-3.1-2	5	5	El controlador relaciona el indicador de proceso y determina si debe estar encendido, si no debe estarlo, da la alarma de motor 2 encendido permanentemente	OK	OK
TC-3.1-3	5	5	El controlador relaciona el indicador de proceso y determina si debe estar encendido, si no debe estarlo, da la alarma de motor 3 encendido permanentemente	OK	OK
TC-3.1-4	5	5	El controlador relaciona el indicador de proceso y determina si debe estar encendido, si no debe estarlo, da la alarma de motor 4 encendido permanentemente	OK	OK
TC-3.1-5	5	5	El controlador relaciona el indicador de proceso y determina si debe estar encendido, si no debe estarlo, da la alarma de	OK	OK

			motor 5 encendido permanentemente		
TC-3.1-6	5	5	El controlador relaciona el indicador de proceso y determina si debe estar apagado, si no debe estarlo, da la alarma de motor 1 detenido.	OK	OK
TC-3.1-7	5	5	El controlador relaciona el indicador de proceso y determina si debe estar apagado, si no debe estarlo, da la alarma de motor 2 detenido.	OK	OK
TC-3.1-8	5	5	El controlador relaciona el indicador de proceso y determina si debe estar apagado, si no debe estarlo, da la alarma de motor 3 detenido.	OK	OK
TC-3.1-9	5	5	El controlador relaciona el indicador de proceso y determina si debe estar apagado, si no debe estarlo, da la alarma de motor 4 detenido.	OK	OK
TC-3.1-10	5	5	El controlador relaciona el indicador de proceso y determina si debe estar apagado, si no debe estarlo, da la alarma de motor 5 detenido.	OK	OK

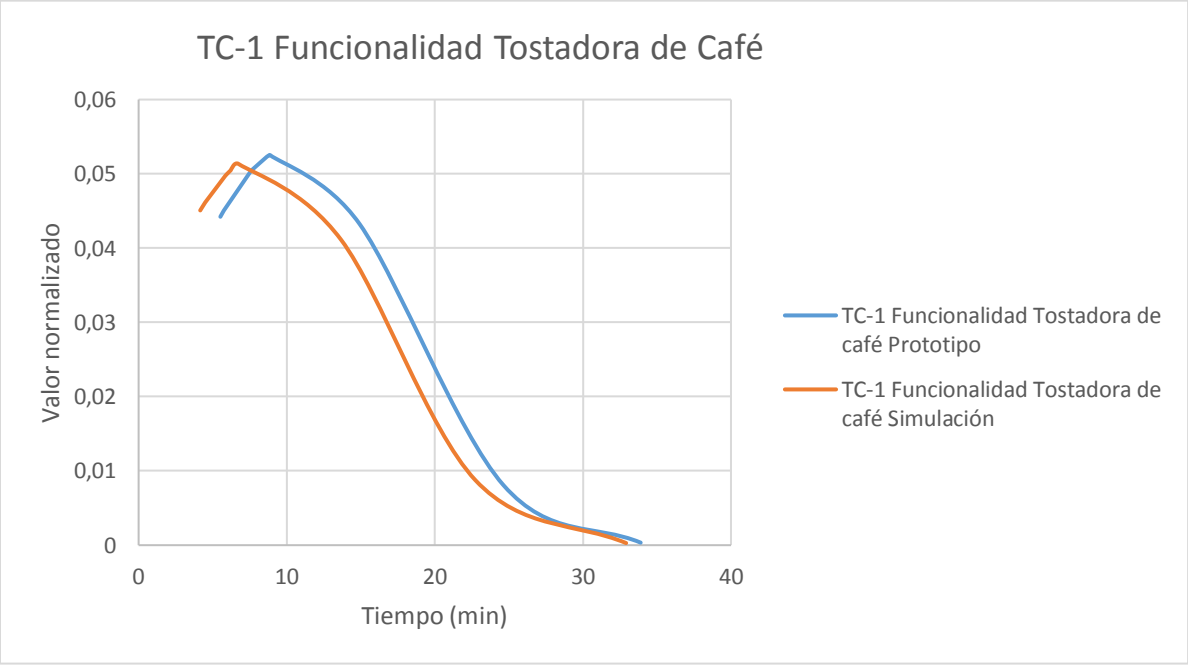
TC-3.1-11	5	5	El controlador relaciona el indicador de proceso y determina si debe estar encendido el calentador, si no debe estarlo, apaga calentador y da alarma.	OK	OK
TC-3.1-12	5	5	El controlador relaciona el indicador de proceso y determina si debe estar encendido el calentador, si no debe estarlo, apaga calentador y da alarma.	OK	OK
TC-3.1-13	5	5	El controlador relaciona el indicador de proceso y determina si debe estar encendido el calentador, si no debe estarlo, apaga calentador y da alarma.	OK	OK
TC-3.2	Fallas en el sistema de Aire Forzado caliente				
TC-3.2-1	5	5	El controlador relaciona el indicador de proceso y determina si debe estar encendido el calentador, si no debe estarlo, apaga calentador. Si debe, enciende la turbina, en caso de que no pueda apagar el calentador o encender la turbina, des energiza sistema y da la alarma.	NOK	OK

Fuente: esta investigación 2017

C. Comparación de las dos técnicas de verificación.

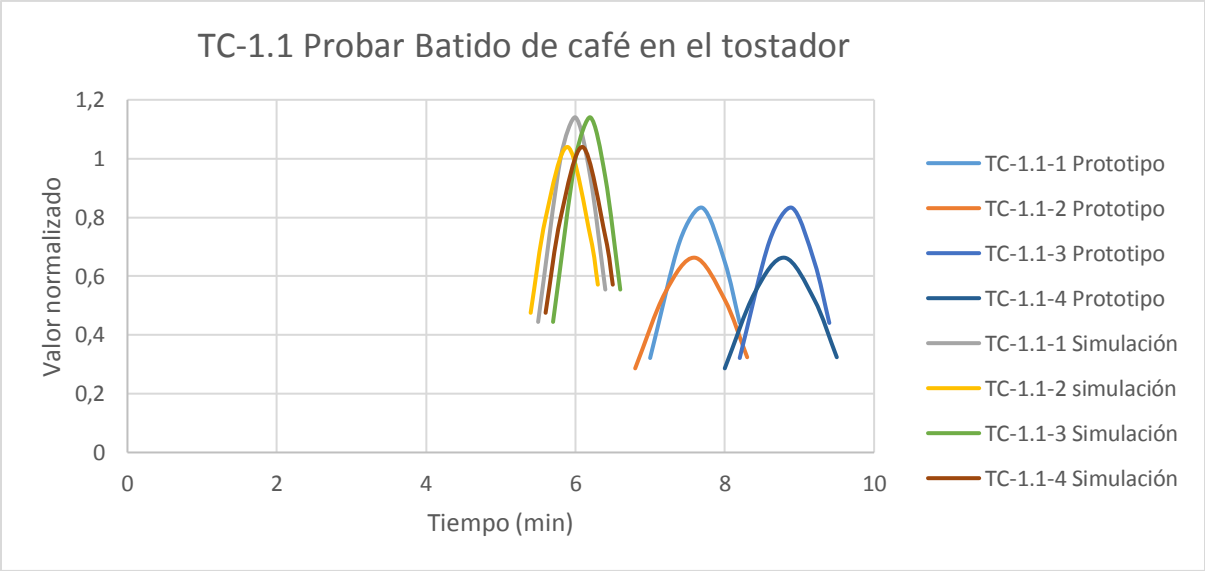
Para la comparación se utilizó el tiempo de implementación de cada caso de prueba, además de que haya cumplido con el resultado esperado. En las gráficas se muestran los resultados de tiempos tomados en cada prueba con lo que se compara la implementación por los dos métodos.

Gráfica 1. Tiempos funcionalidad tostadora de café



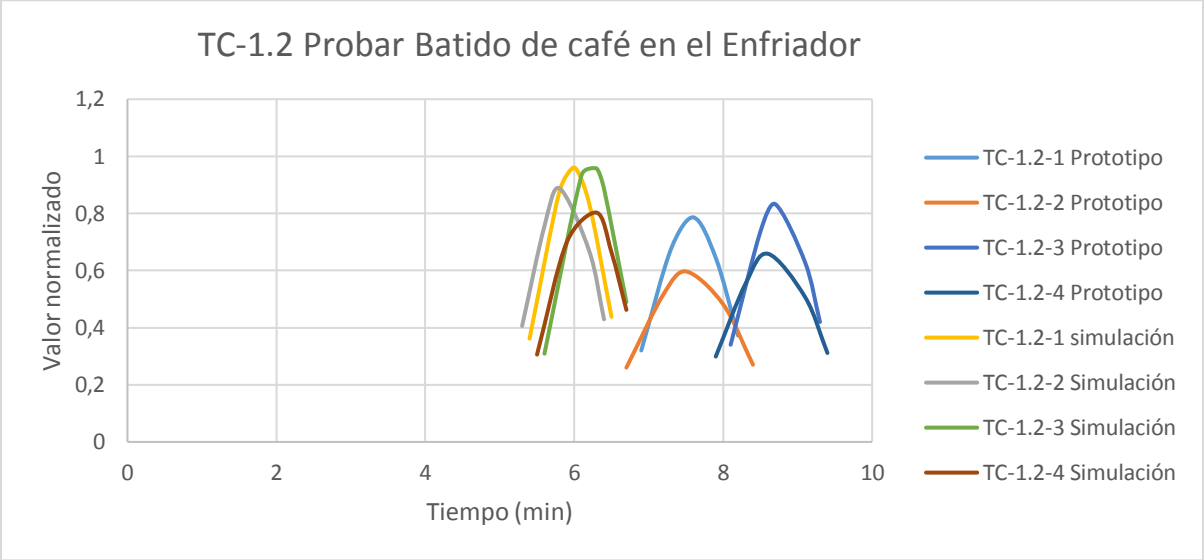
Fuente: esta investigación 2017

Gráfica 2 Tiempos caso de prueba Batido del café en tostador



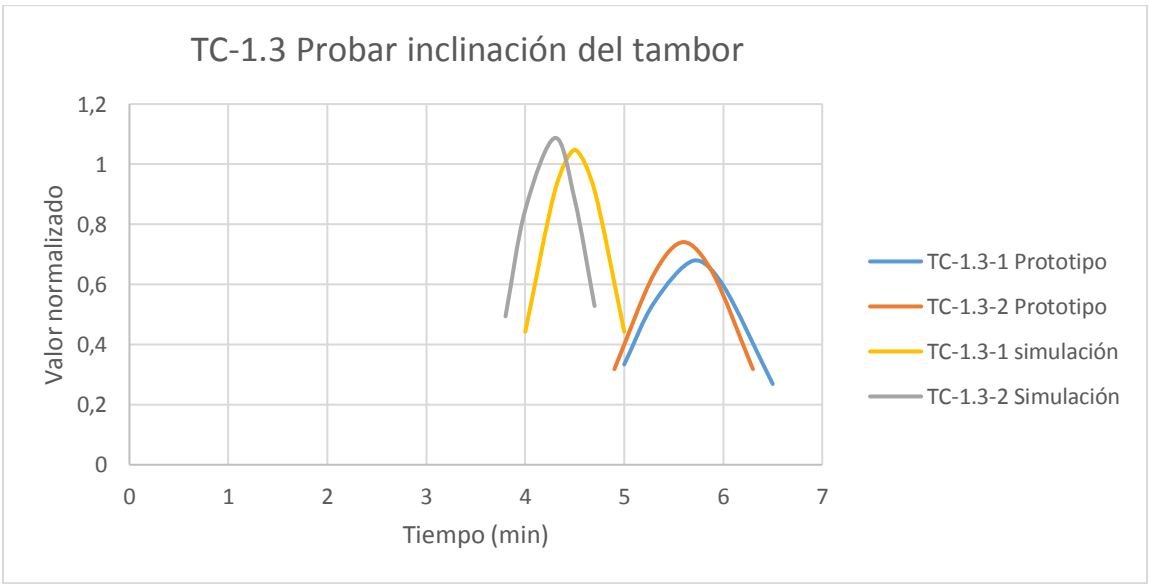
Fuente: esta investigación 2017

Gráfica 3. Tiempos caso de prueba Batido del café en enfriador



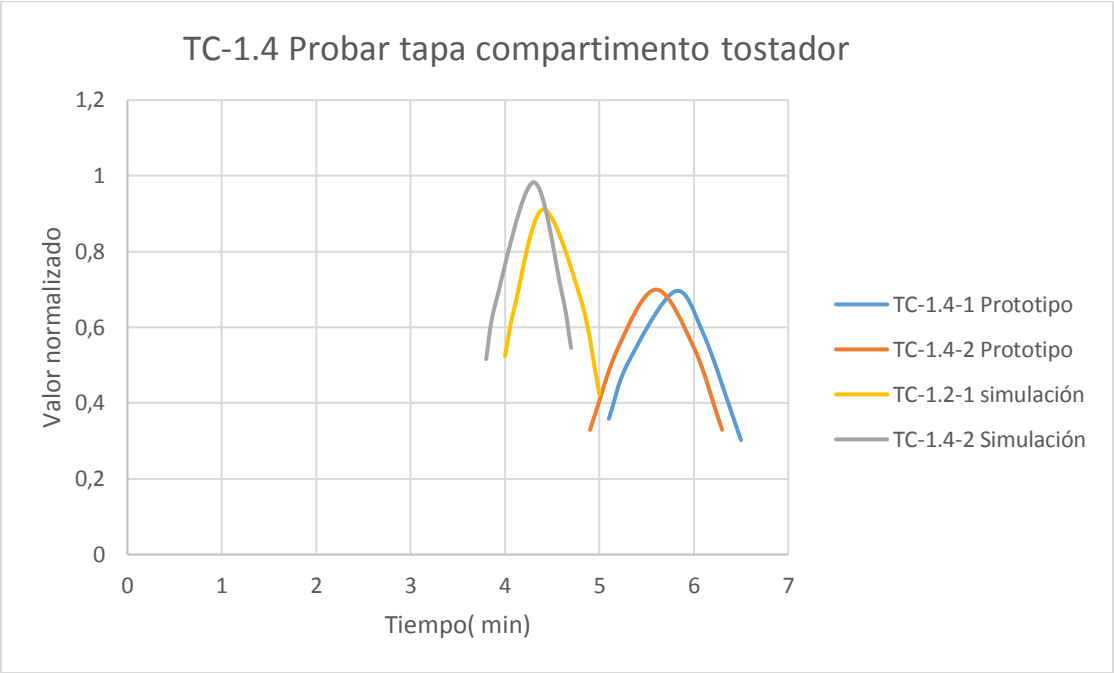
Fuente: esta investigación 2017

Gráfica 4 Tiempos caso de prueba de inclinación de tambor



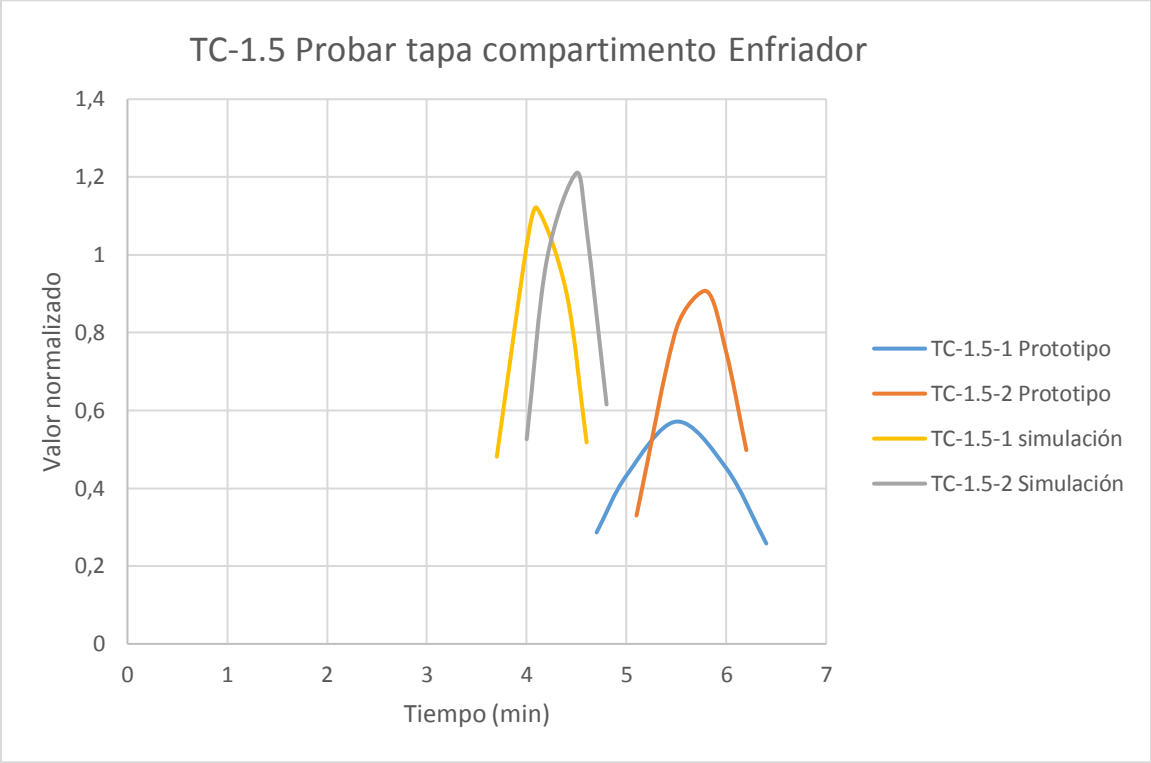
Fuente: esta investigación 2017

Gráfica 5. Caso de prueba tapa compartimento tostador



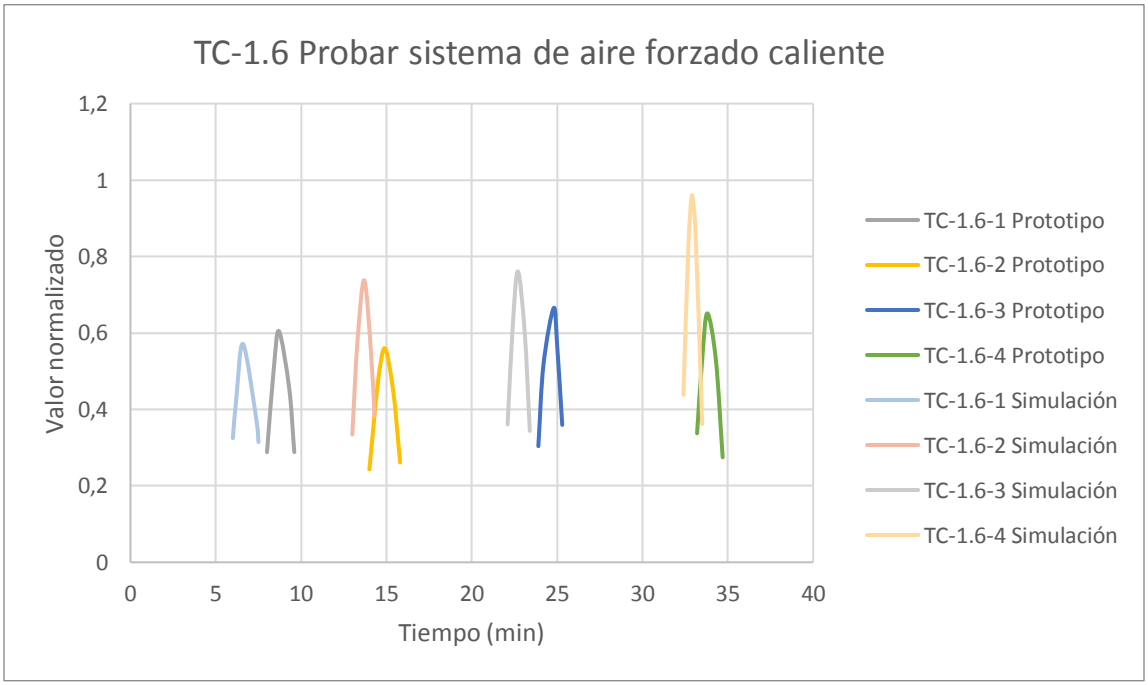
Fuente: esta investigación 2017

Gráfica 6. Tiempos caso de Prueba compartimento enfriador



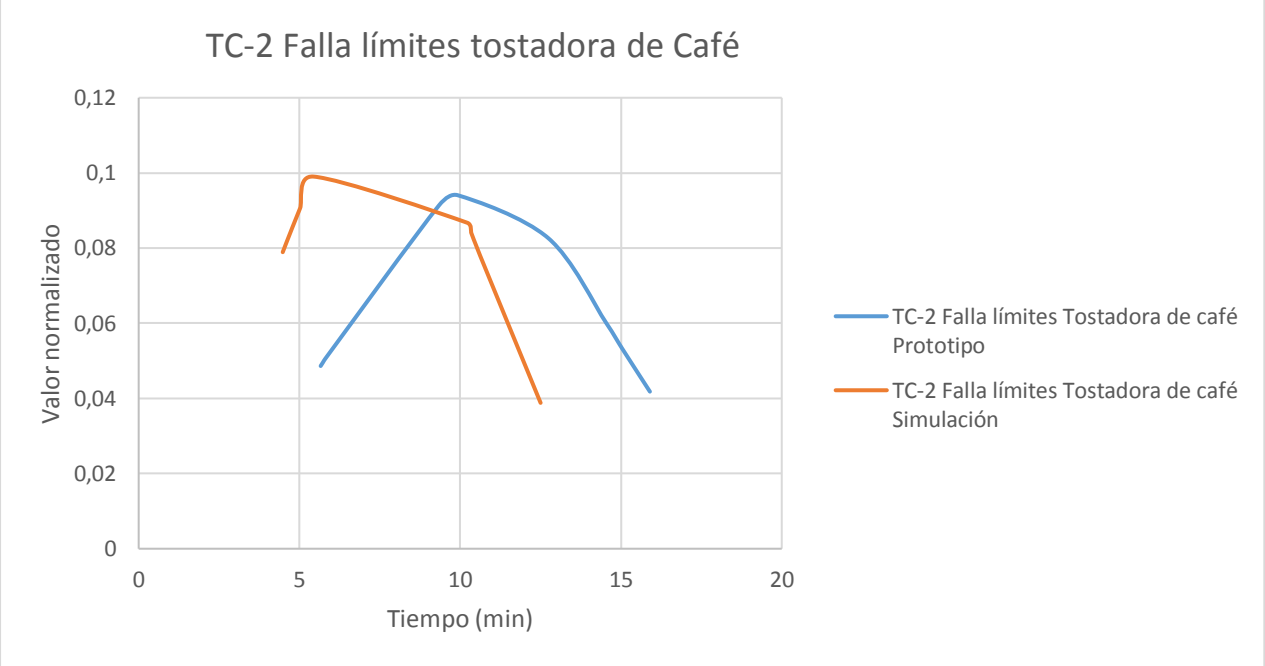
Fuente: esta investigación 2017

Gráfica 7.Tiempos caso de prueba sistema de aire forzado caliente



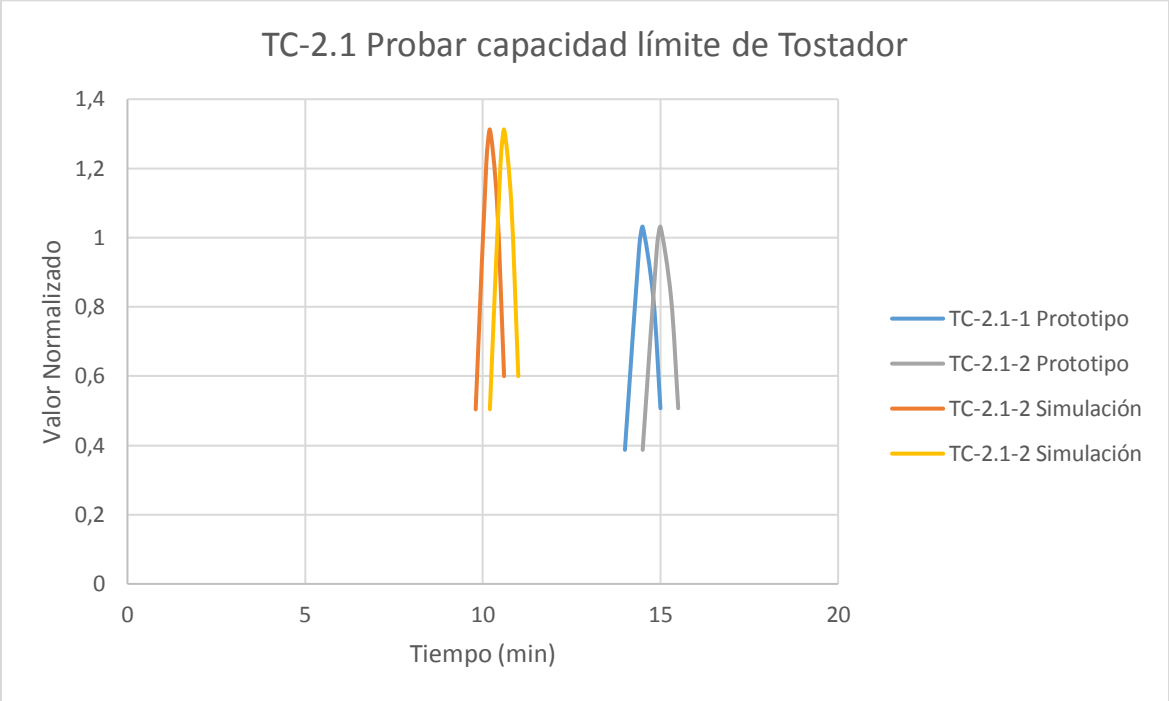
Fuente: esta investigación 2017

Gráfica 8.Tiempos de falla de límites de tostadora de café



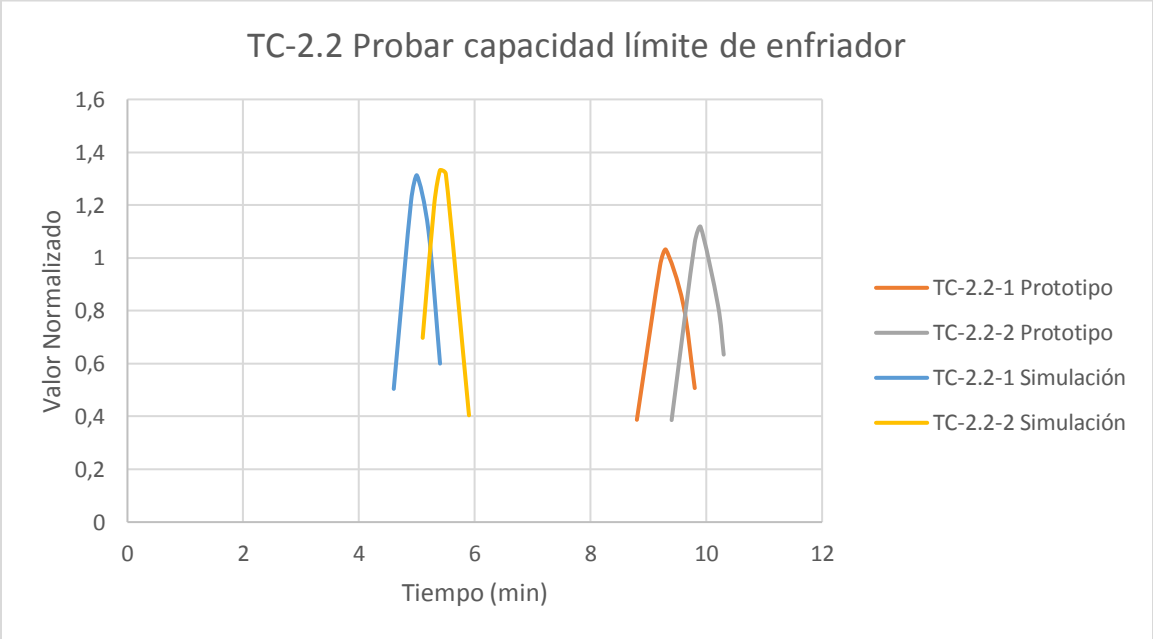
Fuente: esta investigación 2017

Gráfica 9. Tiempos caso de prueba capacidad límite de tostador



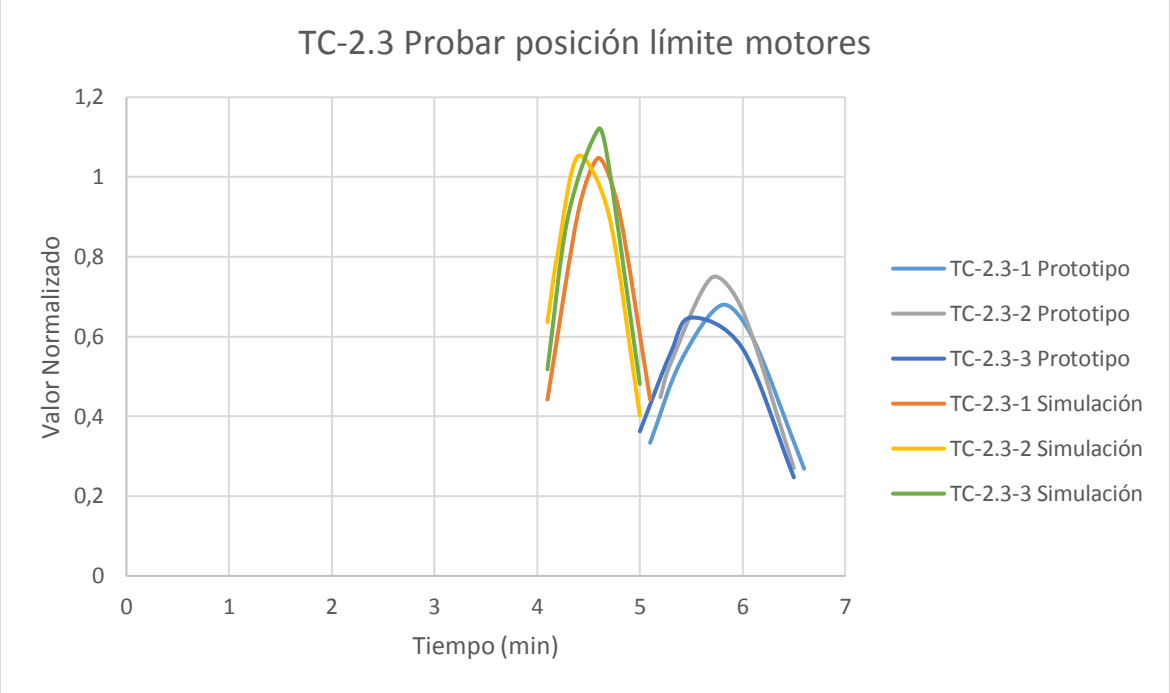
Fuente: esta investigación 2017

Gráfica 10. Tiempos caso de prueba capacidad límite de enfriador



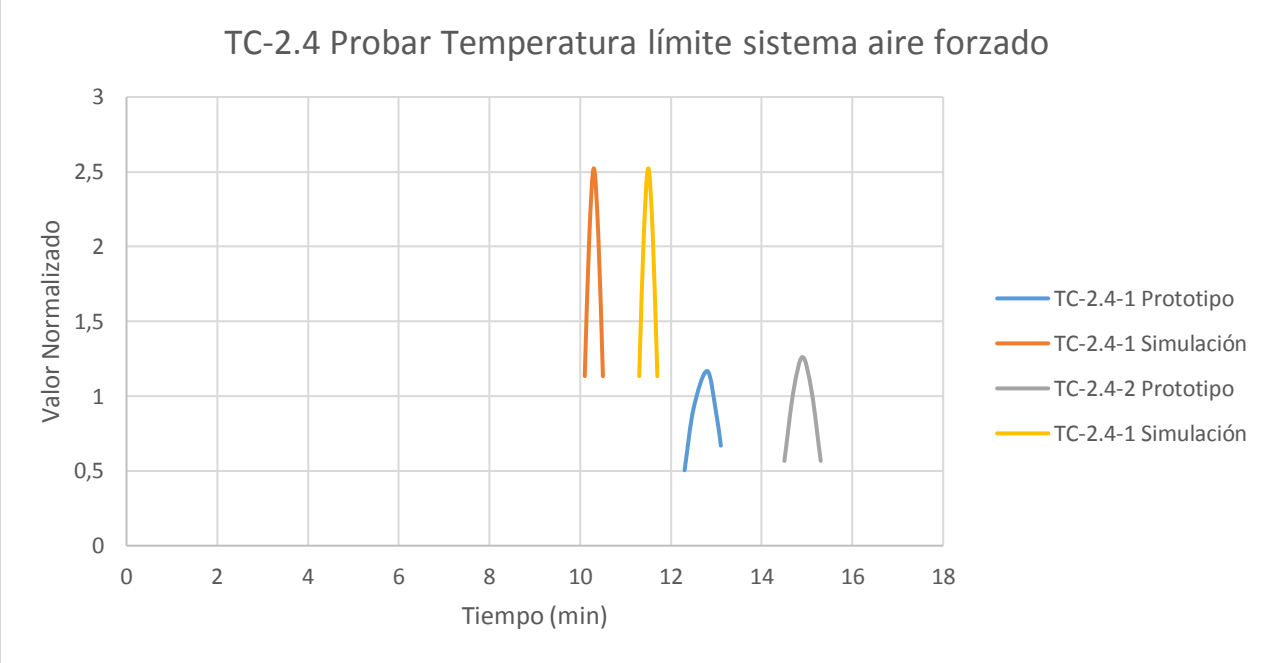
Fuente: esta investigación 2017

Gráfica 11. TIempos caso de prueba límite motores



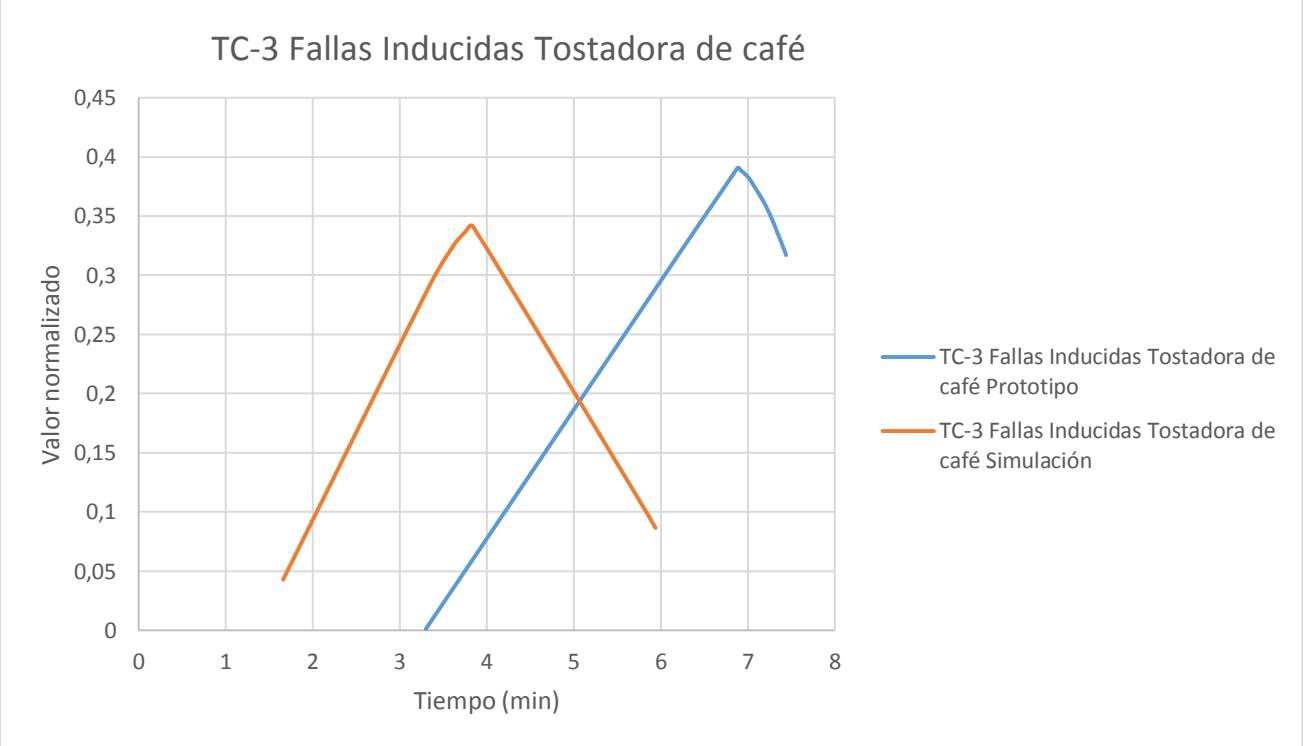
Fuente: esta investigación 2017

Gráfica 12. Tiempos caso de prueba límite sistema de aire forzado caliente



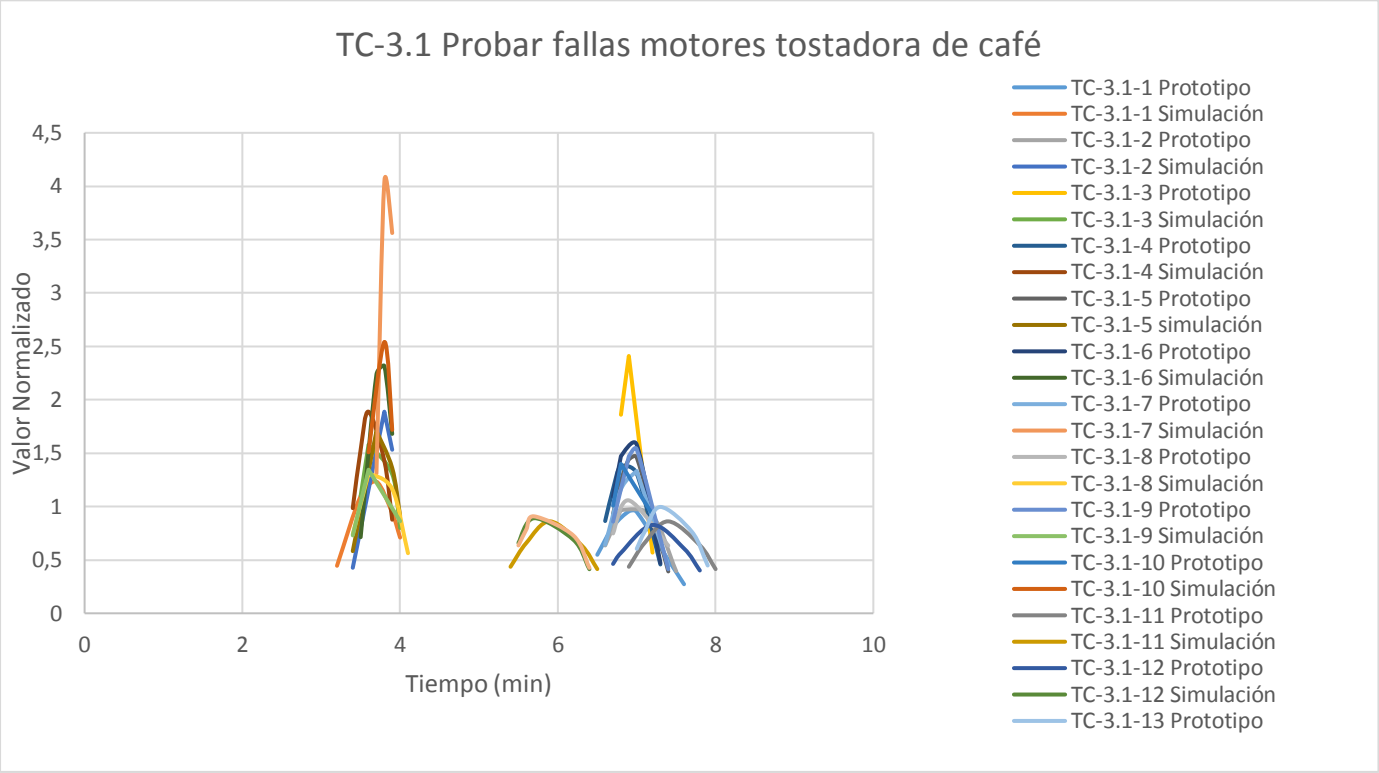
Fuente: esta investigación 2017

Gráfica 13. Tiempos Falla inducidas tostadora de café



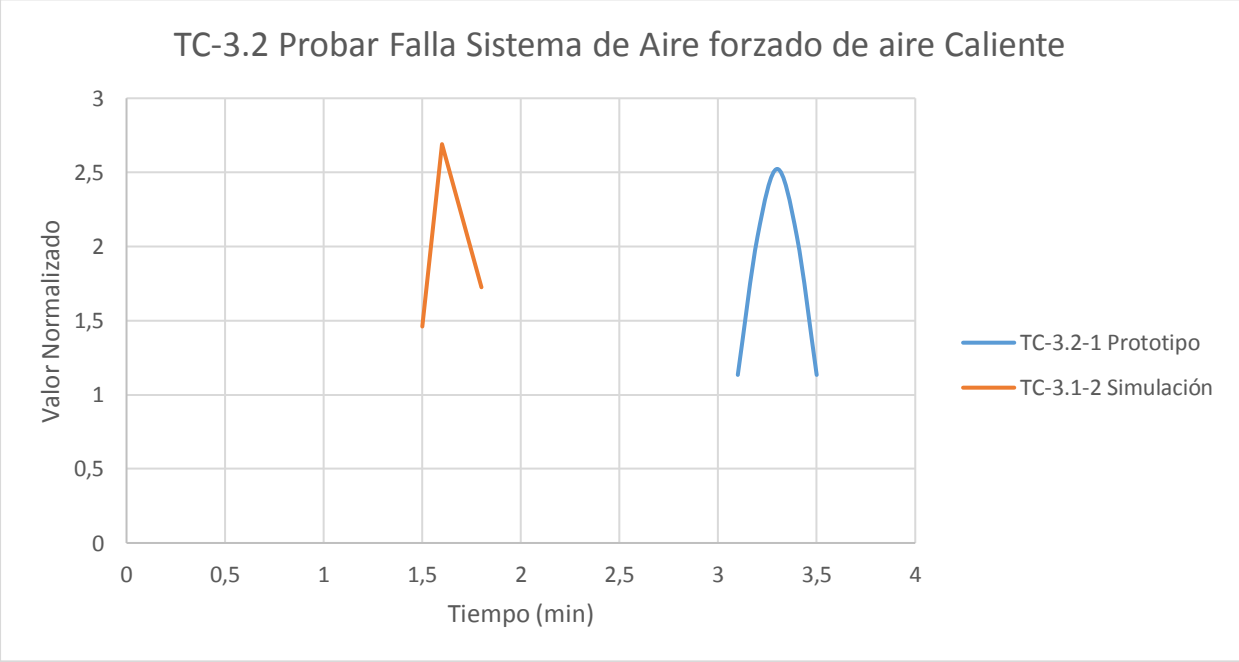
Fuente: esta investigación 2017

Gráfica 14. Tiempos falla motores tostadora de café



Fuente: esta investigación 2017

Gráfica 15. Tiempos de falla de sistema de aire forzado



Fuente: esta investigación 2017

III. Conclusiones

Con este proyecto se puso en práctica el diseño de máquinas con un enfoque mecatrónico, logrando un prototipo automatizado funcional, con alta repetibilidad de sus funciones principales logrando los mismos resultados finales en cada iteración. Esto gracias al diseño de un software verificado con los diferentes casos de prueba.

Sin embargo, la simulación del mismo sistema de control hace que la verificación de software sea más eficiente, teniendo la capacidad de tener mejor respuesta de los sensores virtuales y tiempos menores de implementación las pruebas. Al simular se evitan riesgos como quemaduras por las altas temperaturas del tostador, o atascamientos en los mecanismos; así como daños en motores y sistemas eléctricos.

Los tiempos en la implementación de simulación son más bajos en todos los casos de prueba implementados respecto a la implementación en el prototipo, lo que hace más productivo diseñar en una planta virtual que en una física.

Referencias Bibliográficas

Avison, D., & Fitzgerald, G. (1988). Techniques-Methodologies. En *Information Systems Development: Methodologies, Techniques and Tools* (págs. 65-251). Londres, Inglaterra: Blackwell Scientific Publications.

Baresi, L., & Pezzè, M. (2006). An introduction to software testing. Electronic Notes in Theoretical Computer Science. *1*(148), 89-111. doi:doi:10.1016/j.entcs.2005.12.014

Bishop, R. H. (2002). Mechatronic Design Approach. En *The mechatronics handbook.* Boca Raton, Fla.: CRC Press.

Booch, G., Rumbaugh, J., & I., J. (1999). *El Proceso Unificado de Desarrollo de Software.* Madrid, España: Addison Wesley.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *El Lenguaje Unificado de Modelado.* Madrid, España: Addison Wesley Iberoamericana.

Caïs, J. (1997). *Metodología de Análisis Comparativo.* Madrid, España: Editorial Centro de Investigaciones Metodológicas.

Canós, J., Letelier, P., & Penadés, M. (2003). Métodologías Ágiles en el Desarrollo de Software. En L. P., & S. E. (Ed.), *Metodologías ágiles en el desarrollo de software.* Alicante, España.

Castellanos Arias, J., & Solaque Guzmán, L. (2001). Modelado con redes de petri e implementación con GRAFCET de un sistema de manufactura flexible con procesos concurrentes y recursos compartidos. *Ciencia E Ingeniería Neogranadina, 20*(1), 61-75.

Department of Defense,Systems Management College. (2001). *Systems Engineering Fundamentals.* Fort Belvoir, Virginia: Defense Acquisition University Press.

Downs, E., Clare, P., & Coe, I. (1988). *Structured Systems Analysis and Design Method: Application and Context.* Hemel Hempstead,Herts: Prentice Hall.

Friedman, A., & Cornford, D. (1989). *Computer systems Development: History, Organization and Implementation.* Chichester: Jhon Wiley.

Gill, A. (1962). *Introduction to the theory of finite-state machines* . New York, EE.UU: McGraw-Hill electronic science series.

Green, D., & DiCaterino, A. (1998). *A Survey of system Development Process Models.* Albany, UU.EE: Center for technology in Goverment.

Haas, P. (2002). Stochastic petri nets : Modelling, stability, simulation (Operations research). New York: Springer.

Hibnio, H., Inukai, T., & Fukuda, Y. (2006). Efficient manufacturing system implementation based on combination between real and virtual factory. *International Journal of Production Research, 44*(18), 3897-3915.

Kruchten, P. (2001). *The Rational Unified Process An Introduction.* Boston, EE.UU: Addison Wesley.

Lee, C., & Park, S. (2014). Survey on the virtual commissioning of manufacturing systems. *Journal of Computational Design and Engineering, 1*(3), 213-222. doi:10.7315/JCDE.2014.021

Machineering.de. (Agosto de 2017). *machineering: Real time 3d simulation using industrialPhysics:*. Obtenido de http://www.machineering.de/en/products/industrialphysics-simulation.html

Makris, S., Michalos, G., & Chryssolouris, G. (2012). Virtual Commissioning of an Assembly Cell with Cooperating Robots. *Advances in Decision Sciences*, 1-11.

Moeeni, F., Sanchez, S. M., & Vakha Ria, A. J. (1997). A robust design methodology for Kanban system design. *International Journal of Production Research, 35* (10), 2821-2838.

Naik, S., & Tripathy, P. (2008). *Software Testing and Quality Assurance:Theory and Practice.* Hoboken, N.J.: John Wiley & Sons. doi: doi:10.1002/9780470382844

Onosato, M., & Iwata, K. (1993). Development of a virtual manufacturing system by integrating product models and factory models. *CIRP, 42*(1), 475-478.

Pressman, R. (1993). *Ingeniería del software, un enfoque práctico.* Madrid: McGraw-Hill.

Process, R. U. (1998). Best practices for software development teams. . *A Rational Software Corporation White Paper*.

Risk reduction with the RUP phase plan. (2017). (Ibm.com) Recuperado el 17 de 10 de 2017, de https://www.ibm.com/developerworks/rational/library/1826.html#N100E4

Royce, W. W. (1970). Managing the development of large Software systems. *Proceedings of IEEE WESCON, 26*(8), 328-338.

Schwaber, K., Beedle, M., & Martin, R. (2001). *Agile Software Development with SCRUM.* Prentice Hall.

Sommerville, I. (1985). *Software engineering (2nd ed. ed., International computer science series).* Wokingham, England: Addison-Wesley.

Stevens, W. (1991). *Software design : Concepts and methods (Practical software engineering series).* New York: Prentice Hall.

Ugural, A. C. (2004). *Mechanical design, An integrated approach.* Boston: McGraw-Hill/ HIgher Education.

Yourdon, E., & Constantine, L. (1979). *Structured design : Fundamentals of a discipline of computer program and systems design.* Englewood Cliffs: Prentice-Hall.

Zhu, H. (2005). *Software design methodology.* . Oxford: Elsevier Butterworth-Heinemann.