Universidad de Pamplona

Facultad de Ingenierías y Arquitectura

Programa de Ingeniería de Sistemas

Tema:

DISEÑO DE UN PROCEDIMIENTO DE PRUEBAS PARA LOS PROYECTOS DE DESARROLLO DE SOFTWARE DE LA EMPRESA BEGO

Autor:

Hernán Javier Guio Carrillo

Pamplona, Norte De Santander

Diciembre 2020

Universidad De Pamplona

Facultad De Ingenierías y Arquitectura

Programa De Ingeniería De Sistemas

Trabajo de grado presentado para optar al título de Ingeniero de Sistemas.

Tema:

DISEÑO DE UN PROCEDIMIENTO DE PRUEBAS PARA LOS PROYECTOS DE DESARROLLO DE SOFTWARE DE LA EMPRESA BEGO

Autor:

Hernán Javier Guio Carrillo

Director:

William Mauricio Rojas Contreras

Magíster en ciencias computacionales.

Pamplona, Norte de Santander.

Diciembre 2020.

Resumen

La automatización de pruebas es la práctica que permite controlar la ejecución de un producto software de manera automática, comparando los resultados obtenidos con los resultados esperados mediante el uso de un software especial que pretende simular la interacción humana. Esta práctica permite no solo realizar pruebas repetitivas dentro de un proceso sino probar ejecuciones que manualmente podrían originar fallas de regresión, de integración y fallas funcionales. Gracias a la implementación de pruebas se puede contar con métricas e información objetiva e independiente sobre la calidad del producto asegurándose que cumpla con prácticas y estándares de programación adecuadas entregando versiones confiables y además ahorrando tiempo y recursos.

En el presente documento se expone el diseño de proceso de pruebas que permitirán a la empresa BeGo la optimización de recursos y tiempos, logrando así tener control y seguimiento de las diferentes funcionalidades y procesos de desarrollo de software que se están llevando a cabo. Inicialmente se plasma un estado del arte para el proceso de automatización de pruebas en empresas dedicadas al desarrollo de Software con el fin de tener una base sólida para determinar qué herramientas, técnicas y softwares dedicados a pruebas automáticas se adaptan a los procesos operativos de la empresa, para su posterior selección y definición de una política adecuada. También describe el estudio realizado para la construcción de un prototipo de aplicación hibrida diseñado mediante la librería de React Native. Culminando con el diseño de un procedimiento uniendo las técnicas seleccionadas junto con las herramientas de software.

Abstract

The automation of tests is the practice that allows to control the execution of a software product automatically, comparing the results obtained with the expected results through the use of special software that aims to simulate human interaction. This practice allows not only to perform repetitive tests within a process, but also to test executions that could manually cause regression, integration and functional failures. Thanks to the implementation of tests, it is possible to count on metrics and objective and independent information on the quality of the product, ensuring that it complies with appropriate programming practices and standards, delivering reliable versions and also saving time and resources.

This document presents the design of the testing process that will allow the BeGo company to optimize resources and times, thus achieving control and monitoring of the different functionalities and software development processes that are being carried out. Initially, a state of the art for the test automation process is embodied in companies dedicated to software development in order to have a solid base to determine which tools, techniques and software dedicated to automatic tests are adapted to the operational processes of the company. , for its subsequent selection and definition of an adequate policy. It also describes the study carried out for the construction of a hybrid application prototype designed using the React Native library. Culminating in the design of a process bringing together the selected techniques together with the software tools

Tabla de contenidos

1 Descripción del proyecto 11
1.1 Planteamiento del problema. 11
1.2 Justificación 13
1.3 Delimitación 14
1.3.1 Objetivo General 14
1.3.2 Objetivos Específicos 14
1.4 Acotaciones 15
1.5 Metodología 16
2 Marco teórico y estado del arte 17
2.1 Marco conceptual 17
2.1.1 Pruebas manuales 17
2.1.2 Pruebas automatizadas 17
2.1.3 Pruebas funcionales 18
2.1.4 Pruebas de desempeño 20
2.1.5 React 21
2.1.6 Metodologías de desarrollo ágil 21
2.1.6.1 Metodología scrum 21
2.1.7 Integración Continua 22
2.1.8 Open Source: 23
2.2 Herramientas para la realización de pruebas 24
2.2.1 Herramientas de evaluación estática del código 25
2.2.2 Herramientas para planificación y gestión de pruebas 25
2.2.3 Herramientas para las pruebas de automatización 25
2.3 Estado del arte 26
2.3.1 Internacional 26
2.3.2 Nacional 27

2.3.3 Regional 28
3 Análisis preliminar 31
3.1 Herramientas para pruebas de automatización 31
3.2 Herramientas para planificación y gestión 32
3.2.1 Asana 33
3.2.2 TestLink 34
3.2.3 Trello 35
3.2.4 Jira 36
3.3 Costos de realizar pruebas de software 38
3.4 Herramientas de desarrollo WEB y de aplicaciones móviles 42
4 Definición base 45
4.1 Selección de la herramienta de gestión 45
4.1.1 Clasificaciones según el sector al que se enfocan 45
4.1.2 Identificación de las ventajas y desventajas 46
4.2 Selección del tipo de framework de desarrollo 50
4.2.1 Identificación de las ventajas y desventajas de cada framework 50
4.2.2 Diferencias entre los frameworks 53
4.2.3 Agrado y aceptación de la comunidad 53
4.3 Selección para las herramientas de pruebas automatizadas 55
5 Diseño 57
5.1 Procedimiento para plan de pruebas 57
5.1.1 Realización de análisis de requerimientos de desarrollo de software 57
5.1.2 Detectar las nuevas funcionalidades que deben ser probadas 57
5.1.3 Identificar las funcionalidades de sistemas existentes que deben probarse 58
5.1.4 Definir la estrategia de pruebas 59
5.1.5 Definir los criterios de inicio, aceptación y suspensión de pruebas 59
5.1.6 Identificar los entornos requeridos 61
5.1.7 Determinar necesidades de personal y entrenamiento 62

5.1.8 Establecer la metodología y procedimientos de prueba 62
5.1.9 Elaborar la planificación de las pruebas 62
5.1.10 Identificar los riesgos y definir planes de respuesta 63
5.2 Instalación de versiones y herramientas que se van a trabajar 64
5.3 Definición del flujo de trabajo para errores 64
5.3.1 versionamiento de código y versionamiento de relases 64
5.3.2 Flujo de trabajo 65
5.3.3 Flujo de trabajo de un bug 66
5.3.4 Principales atributos de un problema 67
5.4 Implementación de las pruebas con Jest 68
5.4.1 Configuración del entorno de pruebas 68
5.4.2 Estructura de las pruebas 71
5.4.3 Matchers 72
5.5 Tipos de pruebas que se pueden realizar con Jest 73
5.6 Implementación de las pruebas con PostMan 74
5.7 Responder a solución de problemas con Smart Commits 75
6 Validación 77
6.1 Configuración de las herramientas 77
6.2 Gestión de proyectos 77
6.3 Realización de Test 79
6.4 Al encontrar bugs 84
6.5 Herramientas de desarrollo 84
7 Conclusiones 86
8 Recomendaciones y trabajos futuros 88
9 Bibliografía. 89

Tablas

Tabla 1Descripción de diferentes softwares para Test 32

Tabla 2 Ventajas y desventajas del software de Asana 46

Tabla 3 Ventajas y desventajas de Trello 47

Tabla 4 Ventajas y desventajas de Jira 47

Tabla 5 Ventajas y desventajas de TestLink 48

Tabla 6 Ventajas y desventajas de Angular 51

Tabla 7 Ventajas y desventajas de React 51

Tabla 8 Ventajas y desventajas de Vue 52

Tabla 9 Diferencias entre frameworks (Luismi Gracia, 2020) 53

Tabla de Figuras

Ilustración 1 Pruebas funcionales y no funcionales 20

Ilustración 2 Proceso Ágil de Desarrollo de Software (Rina Elizabeth López Menéndez de Jiménez, 2015) 22

Ilustración 3 Flujo de trabajo de la integración continua (Jorge Turrado, 2019)..... 23

Ilustración 4 Aspectos conceptuales de Jira..... 37

Ilustración 5 Coste relativo de corregir un error 40

Ilustración 6 Anti patrón de las prioridades de las pruebas 41

Ilustración 7 Prueba de automatización ideal 42

Ilustración 8 Frameworks con más aceptación por la comunidad 54

Ilustración 9 Frameworks más queridos 54

Ilustración 10 Tendencias de descargas por NPM 56

Ilustración 11 Flujo de trabajo de las versiones 65

Ilustración 12 Flujo de trabajo predeterminado en Jira (YANA GUSTI, 2019)..... 66

Ilustración 13 Ilustración de la creación de un problema 67

Ilustración 14 Estados de problemas 68

Ilustración 15 Niveles de priorización en Jira..... 68

Ilustración 16 Configuración para ejecutar las pruebas..... 70

Ilustración 17 Ejemplo de un resultado de Jest Coverage (Matthew Garcia, 2016) 70

Ilustración 18 Formato de una prueba con Jest 71

Ilustración 19 Resultado de ejemplo de ejecución de pruebas..... 72

Ilustración 20 Formato de una prueba escrita con PostMan 74

Ilustración 21 Resultado de la ejecución de pruebas con PostMan 74

Ilustración 22 Commit inteligente, Jira 75

Ilustración 23 Representación grafica del procedimiento de pruebas 76

Ilustración 24 Herramientas utilizadas 77

Ilustración 25 Ilustración de manejo de errores 79

Ilustración 26 Entornos de despliegue del sistema 81

Ilustración 27 Pruebas internas en dispositivos móviles 82

Ilustración 28 Resultado por consola de las pruebas 83

Ilustración 29 cobertura de código 83

Ilustración 30 Representación de una tarea de solución de error terminado 84

1 Descripción del proyecto

1.1 Planteamiento del problema.

En la actualidad las empresas dedicadas a la producción de software deberían tener diseñado un proceso de pruebas que permitan optimizar de forma planeada y disciplinada el encontrar diferencias entre el comportamiento esperado con el comportamiento observado en proyectos de desarrollo de software.

La empresa BeGo aborda los requerimientos de desarrollo mediante la asignación de tareas y responsabilidades o en ocasiones por medio de descripciones realizadas de manera informal en donde se especifican de forma breve los comportamientos que se deben esperar para cumplir con las exigencias propuestas. Al completar o finalizar la actividad de desarrollo se inicia la primera etapa de pruebas donde una persona distinta a la que desarrolló la funcionalidad se encarga de realizar una prueba tratando de simular con la mayor precisión posible el uso normal que tendría la nueva adición funcional para verificar el correcto funcionamiento o detectar fallas potenciales, si la ejecución marcha adecuadamente se procede a pasar la segunda etapa de pruebas en la que otra persona nuevamente trata de encontrar defectos o validar el funcionamiento, si todo sale bien la funcionalidad entra en espera para desplegarse en producción, por el contrario, si falla regresa a la fase de solución de problemas y se especifica mediante una evidencia el por qué ha fallado para que comience de nuevo el ciclo de corrección y pruebas. Además, la empresa cuenta con un aplicativo móvil construido en el entorno de desarrollo de Apache Cordova el cual presenta incompatibilidad con las distintas versiones de Android haciendo difícil la tarea de lanzar la

aplicación a los diversos dispositivos que se encuentran en el mercado.

Al no tener ningún procedimiento establecido en la empresa se ve perjudicada la parte organizacional debido a que se reprocesa el análisis causando que los tiempos de la implementación de las tareas pueden aumentar más de lo esperado provocando una falta de optimización en los proyectos. Es por esto que surge la necesidad de establecer una guía que permita a la empresa BeGo llevar un mayor control y orden sobre las tareas que realizan los desarrolladores permitiendo así ahorrar tiempo y recursos, y asimismo proporcionar un mejor entorno para la creación del nuevo aplicativo de programación hibrida mediante el uso de la librería React Native.

1.2 Justificación

Implementar un sistema de automatización de pruebas de software proporciona diversos beneficios tales como una mayor organización que conlleva a un análisis más estructurado sobre los procesos de desarrollo, mayor número de pruebas ejecutadas en lapsos de tiempo cortos, mejor comunicación con el equipo de desarrollo, estabilización temprana del código y mayor confiabilidad en los resultados. Actualmente a la hora de desarrollar un nuevo proceso de software en la empresa BeGo se identifican los detalles y requerimientos necesarios para poder documentarse y planear la manera en la que se llevará a cabo distribuyendo responsabilidades y estimando tiempos de ejecución para su realización. Establecer un procedimiento de pruebas es una necesidad de la empresa BeGo que perfeccionará las fases de desarrollo de su plataforma, así como también proporciona estándares de control, planeación y protocolos de funcionamiento. Este proyecto busca lograr definir un modelo para pruebas de software que mejore el rendimiento para el equipo de desarrollo, puesto que no sería necesario la inversión de gran cantidad de tiempo en realizar manualmente pruebas al sistema.

1.3 Delimitación

1.3.1 Objetivo General

- Diseñar un procedimiento de pruebas para los proyectos de desarrollo de software de la empresa BeGo.

1.3.2 Objetivos Específicos

- Establecer el estado del arte del proceso de pruebas en empresas dedicadas al desarrollo de Software.
- Analizar las herramientas y técnicas para implementar el proceso de pruebas en la empresa BeGo.
- Identificar las categorías y propiedades de los elementos que conforman el procedimiento de pruebas de la empresa BeGo.
- Validar el procedimiento de pruebas en el proyecto de diseño de una nueva versión del aplicativo móvil soportado en programación híbrida mediante el uso de la librería React Native.

1.4 Acotaciones

En el presente trabajo se emplearán únicamente herramientas de integración y pruebas de código libre (Open Source), además los prototipos de las pruebas del sistema y los procesos serán implementados solamente en el ámbito de las pruebas funcionales aplicados al servicio de “Mercar” que ofrece la plataforma de la empresa BeGo.

Debido a que el proceso de integración de software está muy asociado al proceso de pruebas, se hará uso e implementación de integración en la realización de las actividades propuestas para este trabajo.

Del mismo modo cabe aclarar que la información, descripción, demostraciones funcionales o resultados serán limitadas bajo los acuerdos de confidencialidad de la empresa donde se realiza la práctica

1.5 Metodología

El método que se utilizará para el desarrollo del proyecto se basa bajo un tipo de investigación directa y documental, en donde se utilizará el conocimiento existente sobre el tema y se empleará con el fin de analizar la manera más optima en la que se utilizan los recursos para obtener resultados logrando dar realización a los objetivos planteados para el proyecto.

El proyecto cuenta con diferentes secciones; inicialmente se realizó un estudio para conocer el material existente sobre la temática y explorar que herramientas se usan en la actualidad para la gestión de pruebas en las empresas dedicadas al desarrollo de software, posteriormente en la siguiente sección se utilizará la información adquirida para analizar que software y que técnicas son más convenientes para aumentar el rendimiento en cuestión de desarrollo de la empresa BeGo, seguidamente se hará una capacitación con el equipo de desarrolladores de BeGo para incentivar a implementar estas nuevas metodologías y realizar una evaluación para determinar el rendimiento y hacer una retroalimentación para ver qué factores se pueden mejorar, por último se generará el procedimiento de pruebas como un documento el cual contendrá una serie de procesos para escribir código de manera estructurada y disciplinada aumentando la calidad y confiabilidad del sistema basándose en los elementos teóricos de las fases anteriores.

Para este proyecto la recopilación de información se establece partiendo de información secundaria como artículos, libros e internet.

2 Marco teórico y estado del arte

2.1 Marco conceptual

- **Pruebas de software**: Las pruebas de software son un conjunto de procesos con los que se pretende probar un sistema o aplicación en diferentes momentos para comprobar su correcto funcionamiento. Este tipo de pruebas abarca cualquier estado del desarrollo del sistema, desde su creación hasta su puesta en producción. Lo interesante de las pruebas es que se puedan ejecutar de manera automática, para determinar en cualquier momento si se tiene una aplicación estable o si, por el contrario, un cambio en una parte ha afectado a otras partes sin darse cuenta (Turrado, 2020).

2.1.1 Pruebas manuales

Las pruebas manuales son realizadas por personas, las cuales interactúan con el sistema usando herramientas y técnicas adecuadas en cada caso. Estas pruebas por lo general resultan costosas, ya que se requiere contar con un equipo profesional encargado de realizar esta labor para crear y configurar un entorno donde puedan ejecutar las pruebas.

Estas pruebas pueden estar expuestas al error humano como por ejemplo saltarse pasos o cometer errores tipográficos.

2.1.2 Pruebas automatizadas

Las pruebas automatizadas son llevadas a cabo por máquinas, que ejecutan un script de pruebas establecido previamente.

Estas pruebas pueden variar dependiendo de la complejidad, desde comprobar que el método de una clase trabaje correctamente, hasta verificar que una secuencia de acciones en la Interfaz de usuario se lleve a cabo correctamente y devuelvan los resultados esperados.
Estas pruebas son más rápidas y confiables respecto a las que se llevan a cabo manualmente, pero la calidad de estas pruebas automatizadas depende de qué tan bien escritos se encuentren los scripts de pruebas.

2.1.3 Pruebas funcionales

Las pruebas funcionales se definen teniendo como fuente los requisitos del sistema, estas pruebas validan y verifican que el producto cumple con lo especificado y hace lo que debe y cómo lo tiene que hacer dando también una idea del grado de calidad del software. En el blog de TesterHouse el autor clasifica las pruebas funcionales en las siguientes categorías.

- **Pruebas Exploratorias:** El término "pruebas exploratorias" fue introducido por Cem Kanter, consiste en ejecutar las pruebas a medida que se piensa en ellas, sin gastar demasiado tiempo en prepararlas o explicarlas, confiando en los instintos.
- **Pruebas de Regresión:** Las pruebas de regresión están pensadas para evitar el efecto onda en un producto estable en el momento de introducir un cambio.

  Es decir, evita que al introducir nuevos cambios en un software se obtengan comportamientos no deseados o defectos en otros módulos no modificados.

- **Pruebas de Compatibilidad de entorno:** Las pruebas de compatibilidad son pruebas en las que se ejecuta el mismo producto en diferentes entornos, para chequear que funcionalmente se comportan igual.
- **Pruebas Libres:** Son las pruebas que se ejecutan sin un plan determinado. Comparte la misma filosofía que las pruebas exploratorias. Son un complemento ideal para la fase final de ejecución de un proyecto donde el tester ya ha ejecutado el plan de pruebas y puede probar fuera de un guion lo que él considere menos maduro o más propenso a fallo sin atender tanto a la burocracia de la ejecución del plan de pruebas.
- **Pruebas de Humo:** Este tipo de pruebas es una revisión rápida inicial de la versión de software entregada por desarrollo donde se verificará de forma general sin entrar en detalle las principales funcionalidades del mismo y se asegurará que no tiene defectos que interrumpa el funcionamiento básico, para que el equipo de testing pueda seguir probando entrando más en detalle.
- **Pruebas de Mono:** Son las pruebas que se hacen sin atender mucho al funcionamiento teórico del producto, simplemente consiste en navegar por los distintos caminos del software sin un orden determinado e intentando ejecutar todas las opciones posibles. El comportamiento esperado de estas pruebas no es más que la robustez del programa.
- **Pruebas de Sanidad:** La idea principal de las pruebas de sanidad es parecida a las de regresión, pero suelen ser menos exhaustivas.

Se refieren a un conjunto de pruebas que se ejecutan para comprobar que todo funciona correctamente después de alguna intervención o modificación (Pruebas funcionales / No funcionales: ¿Qué son y para qué sirven?, 2019).

2.1.4 Pruebas de desempeño

Este tipo de pruebas encuentra diferencias entre las especificaciones no funcionales y el sistema. El autor Bernd Bruegge en su libro cataloga las pruebas de desempeño dependiendo de factores como el manejo de grandes cantidades de datos, fallas de seguridad, capacidad del sistema de recuperarse de errores.

En la siguiente figura se expone de manera resumida las pruebas funcionales contra las pruebas de desempeño o no funcionales:

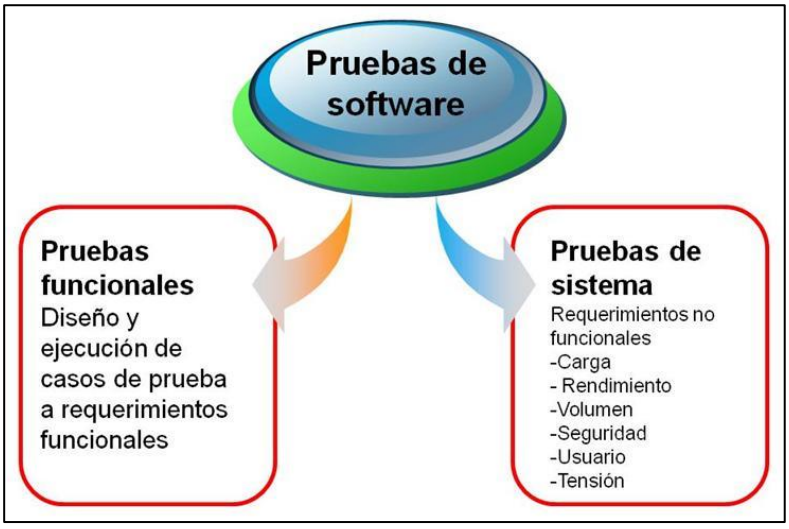


Ilustración 1 Pruebas funcionales y no funcionales

2.1.5 React

React es una librería JavaScript enfocada en el desarrollo de interfaces de usuario. Esta librería permite crear aplicaciones web, SPA (Single Page Application) o incluso aplicaciones para móviles. Para ello, alrededor de React existe un completo ecosistema de módulos, herramientas y componentes capaces de ayudar al desarrollador a cubrir objetivos avanzados con relativamente poco esfuerzo.

Por tanto, React representa una base sólida sobre la cual se puede construir casi cualquier cosa con JavaScript. Además, facilita mucho el desarrollo, ya que ofrece muchos componentes libres y reutilizables.

2.1.6 Metodologías de desarrollo ágil

En febrero de 2001, nace el término "ágil" aplicado al desarrollo de software. En donde un grupo de expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software esbozaron valores y normas que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.

Este término surgió como una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

2.1.6.1 Metodología scrum

Describe un marco para la gestión de proyectos, que se ha sido aceptado y utilizado con éxito durante los últimos.

Esta metodología esta especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración promedia entre una a cuatro semanas. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración (Issi, 2003).

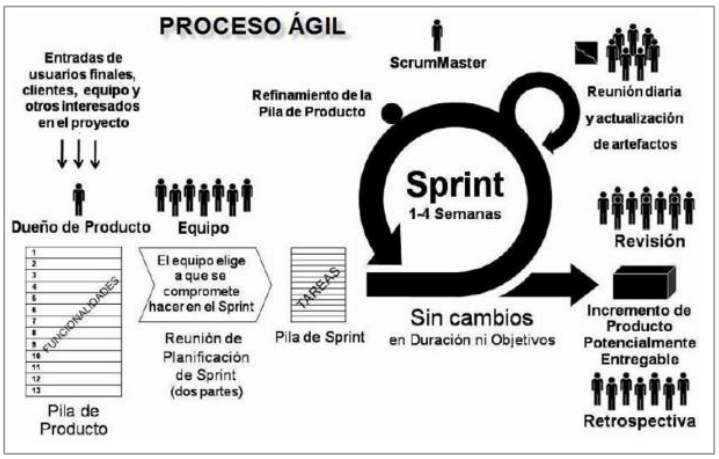


Ilustración 2 Proceso Ágil de Desarrollo de Software (Rina Elizabeth López Menéndez de Jiménez, 2015)

2.1.7 Integración Continua

La integración continua es una práctica que se aplica al desarrollo de software realizado con metodologías ágiles, que se puede ejecutar siempre y cuando se implementen automatizaciones en sus procesos de requerimientos, diseño, desarrollo, pruebas y despliegues, lo que permite realizar cambios en el código, compilar y probarlo el mismo día, las veces que sea necesario, siendo reglamentario que todos los desarrolladores del equipo integren su parte de código por lo menos una vez al día.

En la figura que se muestra a continuación se puede observar un ciclo básico que tendría la integración continua.

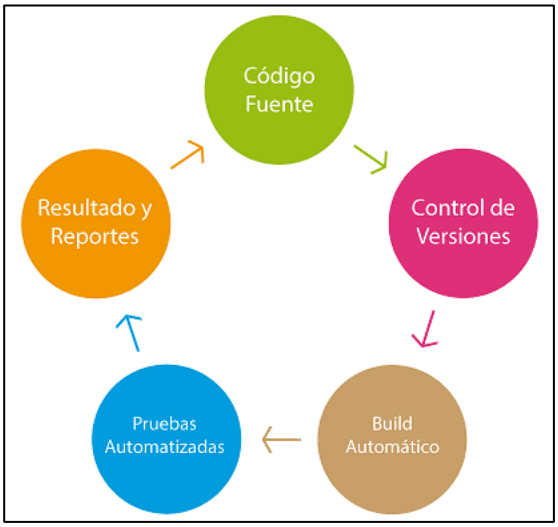


Ilustración 3 Flujo de trabajo de la integración continua (Jorge Turrado, 2019)

Aseguramiento de calidad del software (QA)

El aseguramiento de calidad es de suma importancia dentro del proceso de desarrollo de software ya que corresponde con las tareas de planeación, organización, dirección y control de la calidad de un sistema con el fin de dar a los clientes una calidad adecuada en su producto.

2.1.8 Open Source:

"Diseñado de manera que sea accesible al público: todos pueden ver, modificar y distribuir el código de la forma que consideren conveniente. El software open source se desarrolla de manera descentralizada y colaborativa, así que depende de la revisión entre compañeros y la producción de la comunidad. Además, suele ser más económico, flexible y duradero que sus alternativas propietarias, ya que las encargadas de su desarrollo son las

comunidades y no un solo autor o una sola empresa.” (RedHat, s.f.) . Existe una diferencia entre código libre y software libre, suelen ser lo mismo en la mayor parte de los casos con excepciones. En algunos casos se basan principalmente en el acuerdo de licencias de uso y distribución.

2.2 Herramientas para la realización de pruebas

Existe un conjunto de herramientas las cuales permiten apoyar diversos aspectos en las pruebas, la autora (OROZCO, 2015) menciona algunas de estas herramientas que son:

- Pruebas estáticas: herramientas para apoyar el proceso de revisión, herramientas para el análisis estático y herramientas de modelado.
- Administración Y proceso de pruebas: herramientas para la administración de las pruebas, para el seguimiento de incidentes, para la gestión de la configuración y para la administración de requerimientos.
- Especificación de las pruebas: herramientas para el diseño de las pruebas y para la preparación de datos de prueba.
- Ejecución de las pruebas: herramientas de ejecución de casos de prueba, herramientas de pruebas unitarias, comparadores, herramientas de medición del cubrimiento, herramientas de seguridad. Desempeño y monitorización: herramientas de análisis dinámico, herramientas de desempeño, de carga y de estrés, herramientas de monitorización.

2.2.1 Herramientas de evaluación estática del código

La evaluación estática del código consiste en buscar defectos dentro de un script sin la necesidad de compilar ni ejecutar el código. Para esto se pueden emplear diversas técnicas descritas por (Lluna, 2011) tales como analizar posibles valores extremos, analizar flujo de control, revisiones simbólicas de código, detectar variables o funciones sin declarar, código muerto o inaccesible y posible inexistencia de lógica. Estas herramientas son de gran ayuda para los programadores ya que permite identificar tempranamente posibles fallas en sus funcionalidades permitiendo dar solución temprana a los errores presentados.

- ESLint: Es una herramienta de análisis estático de código creada para el lenguaje de JavaScript, ESLint permite cubrir temas que se relacionan con una buena calidad de programación y un adecuado estilo en el que se escribe código.

2.2.2 Herramientas para planificación y gestión de pruebas

En el contexto de las pruebas es importante tener herramientas que ayuden a las personas que realizan los test a documentar y evaluar los casos de pruebas. Este tipo de herramientas tiene como objetivo brindar mecanismos que permitan realizar de una manera óptima y eficiente el mantenimiento, gestión y documentación de los resultados. También es común dentro de las herramientas que gestionan las pruebas, las herramientas que gestionan las incidencias para su análisis, gestión y distribución dentro de un entorno de un proyecto.

2.2.3 Herramientas para las pruebas de automatización

Estas herramientas tienen como objetivo el diseño de scripts en diferentes tipos de leguajes los cuales permiten probar diferentes funcionalidades dentro de los sistemas.

2.3 Estado del arte

2.3.1 Internacional

Lugar: Costa Rica

Autor: Christian Quesada-López

Título: Factores asociados a prácticas de desarrollo y pruebas de software en Costa Rica: Un estudio exploratorio

Resumen: "La industria de desarrollo de software costarricense se desenvuelve en entornos cambiantes y competitivos. Para enfrentar los continuos desafíos adoptan distintas prácticas ingenieriles que les permitan cumplir las metas de calidad establecidas e incrementar la productividad. Este estudio realiza un análisis cruzado de factores entre las prácticas de ingeniería de software utilizadas en la industria versus los factores demográficos de los profesionales. Para esto, exploramos las relaciones y patrones identificados para las prácticas de desarrollo y pruebas de software. Nuestro análisis considera las respuestas de 135 profesionales que participaron en una encuesta basada en el cuerpo de conocimiento en la ingeniería del software (SWEBOK). En nuestra investigación encontramos que no existe una asociación significativa entre el tamaño de la organización y las metodologías de desarrollo. Además, un conjunto de asociaciones identificadas con distintos factores de contexto es discutidas. Los resultados sugieren que las organizaciones y sus equipos de desarrollo adaptan las prácticas ingenieriles en sus procesos de acuerdo a las necesidades específicas y el contexto de negocios en el que

desarrollan sus proyectos. Se requieren estudios similares que analicen las tendencias de uso y adopción de las prácticas de la ingeniería del software en distintos contextos.” (Quesada-López & Jenkins, 2018)

Aporte: En este documento la autora señala que el principal instrumento para asegurarse que una correcta calidad de las aplicaciones es un plan de calidad, el cual está constituido en reglas o procedimientos. Estos procedimientos son diferentes para cada organización, pero lo importante es que estén escritos, personalizados, adaptados a los procesos de la organización y que se cumplan disciplinadamente.

2.3.2 Nacional

Lugar: Popayán, Colombia

Autor: Julián Andrés Mera-Paz

Titulo: Análisis del proceso de pruebas de calidad de software

Resumen: “Este artículo es producto de la lectura, revisión y análisis de libros, revistas y artículos reconocidos por su calidad científica e investigativa que han abordado el proceso de pruebas de calidad de software. El autor, con base en su experiencia laboral en empresas de desarrollo de software, docencia y otras, ha recopilado y seleccionado información para argumentar y sustentar el porqué de la importancia del proceso de pruebas de calidad de software. Metodología: se analizó la literatura existente sobre el proceso de pruebas de calidad de software en un contexto local, nacional y mundial. Se revisaron de forma exhaustiva las bases de datos ScienceDirect, Elseiver, Springer, Wiley Online Library, proquest, Enginneering Village, Scopus, Dialnet. Resultados: se describen conceptos de

gran importancia en el proceso de pruebas, características, metodologías y marcos de referencia enfocados a la adecuada implementación de un proceso de pruebas. Conclusiones: se generan unos resultados y conclusiones con el fin de que las empresas de desarrollo de productos software mejoren el rendimiento y la efectividad, así como la optimización de los procesos de pruebas de calidad de software, que además es base fundamental para iniciar procesos de investigación en calidad de software." (Paz, 2016).

Aporte: En este trabajo el autor recalca la suma importancia de hacer pruebas sólidas al software mencionando que ya hemos incorporado tecnología y sistemas dentro de nuestra vida cotidiana y que una falla podría resultar costosa. Menciona también la importancia de que se cuenten con unos principios, que, a través de un plan de pruebas, sirvan de guía para el equipo de desarrollo, para que utilicen de la mejor manera los recursos y herramientas y los enfoquen en aumentar la calidad como factor primordial.

2.3.3 Regional

Lugar: Pamplona, Colombia

Autor: Sanguino

Título: Procedimiento para la gestión de pruebas funcionales de software web en ambientes de desarrollo colaborativo y distribuido.

Resumen: "La necesidad de producir un software que sea de utilidad común para muchas organizaciones como es el caso de un software para la gestión académica y administrativa de instituciones educativas, se puede abordar de diversas maneras: tal como se viene haciendo, cada Universidad invierte recursos de su propios para el desarrollo de una

aplicación a su medida (adecuada a sus procesos), algunas Universidades deciden comprar un software licenciado y adaptar sus procesos al software, una alternativa podría ser participar en la creación de una comunidad que desarrolle un software bajo conceptos Open Source, en donde todas las Universidades pudiesen participar, obteniendo un producto genérico de fácil adaptación a cada uno de los contextos establecidos por cada una de las Universidades participantes.

Dado que un proceso de software está constituido de muchos subprocesos, este trabajo se enfoca en el subproceso de pruebas de software, por lo que se desarrolló un procedimiento apoyados en el software de Bugzilla junto a su extensión de Testopia para gestionar las pruebas de software y llevar un seguimiento de las fallas que se presenten en éste.

El procedimiento está basado en la experiencia obtenida de diferentes autores en cuanto a la participación en comunidades de Open Source y de la colaboración en el desarrollo de Academusoft 4.0 el cual pertenece a uno de los productos de software de la Universidad de Pamplona, el fin de este procedimiento es otorgarle a la comunidad educativa libre de la Universidad un medio para mejorar y aliviar la fatal de ésta en el desarrollo de software" (Sanguino, 2016) .

Aporte: En este trabajo el autor menciona como los softwares de Bugzilla y Testopia le fueron útiles a la hora de definir casos de prueba ayudando a encontrar flujos alternativos dentro del software que no se habían pensado, además de recalcar la importancia de tener una herramienta la cual permita monitorear y ver el estado de las pruebas hasta su solución

y lo importante que es definir un plan el cual proponga los pasos que se deben seguir considerando todo el ciclo de pruebas.

3 Análisis preliminar

A continuación, se describen aspectos de importancia que hacen referencia a todo el proceso de gestión y realización de pruebas con el fin de servir como base para tomar la decisión que más se adecue a los procesos operativos de la empresa BeGo.

Del mismo modo se describirán diversos tipos de frameworks para su posterior selección y determinación para la creación de la nueva interfaz de la plataforma.

En el análisis preliminar se pretende buscar puntos de referencia importantes con el fin de tener conocimientos sobre las distintas técnicas y herramientas para su posterior selección y decisión.

3.1 Herramientas para pruebas de automatización

Este tipo de herramientas tiene como objetivo la creación de rutinas las cuales permitan correr un lenguaje de programación específico para ejecutar pruebas funcionales automáticamente.

A continuación, el autor (Jash Unadka, 2019), menciona algunas de ellas.

Framework	Descripción
	➢ Permite hacer test por snapshot (test para asegurarnos de que nuestra interfaz de usuario no cambia). ➢ Posibilidad de crear Spies para métodos y mocks de manera muy sencilla. ➢ Es el framework de test de Facebook por lo tanto cuenta con gran documentación y comunidad activa. ➢ No necesita de gran configuración para ser usado.

	➢ Permite la ejecución de pruebas en paralelo.
	➢ Facilidad a la hora de realizar pruebas asíncronas ➢ Ejecución de pruebas en serie por lo tanto arroja informes precisos y flexibles ➢ Flexibilidad a la hora de mapear excepciones no detectadas ➢ Cobertura de ejecución de pruebas en tiempo real
	➢ Tiene una curva de aprendizaje suave. ➢ No depende de diferentes marcos de JavaScript y no necesita la interacción con el DOM. ➢ Contiene dependencias mínimas hacia otros frameworks y componentes.
	➢ Admite pruebas remotas directamente desde un terminal o IDE. ➢ Es agnóstico del marco, lo que significa que se pueden describir pruebas con marcos populares como Mocha, Jasmine. ➢ Permite realizar pruebas en diferentes entornos
	➢ Permite gestionar las peticiones a las APIs siendo útil para probar el Backend de las aplicaciones. ➢ Arroja información detallada sobre los errores en los test si llegan a existir. ➢ Se incorpora con facilidad con los sistemas de integración continua para llamadas automáticas de pruebas

Tabla 1Descripción de diferentes softwares para Test

3.2 Herramientas para planificación y gestión

Este tipo de herramientas busca brindar a los desarrolladores una guía la cual permita planear y documentar los distintos casos de pruebas e incidencias, y además gestionar los

resultados. Estas herramientas nos permiten tener una visión general de los casos de prueba del proyecto.

A continuación, se presentarán algunas de ellas.

3.2.1 Asana

Asana es una herramienta de productividad en línea para la planificación de proyectos y la gestión de tareas en un entorno de trabajo colaborativo. El objetivo principal de Asana es facilitar el trabajo en equipo para que el personal se concentre en los objetivos a alcanzar y en el trabajo a realizar en lugar de enfocarse en la organización del proyecto.

En el blog de (Anaraya Albornoz, 2020) se presentan varias de sus características:

- **Transparencia**

  Poco importa el número de tareas, proyectos, documentos o informaciones, Asana permite conservar un espacio legible e intuitivo.

- **Agilidad**

  Asana simplifica y flexibiliza la organización tradicional del trabajo. Todos los participantes pueden alimentar una o varias tareas y el responsable de proyecto se encarga de controlar la viabilidad y ejecución. La herramienta es transparente y reúne estrategias y operaciones para el éxito del proyecto.

- **Adaptabilidad**

  Asana permite diferentes vistas, filtros, clasificaciones y organizaciones para que cada colaborador pueda navegar en la herramienta y acceder a la información según su funcionamiento personal y sus necesidades profesionales.

- **Integración**

  Asana puede ser considerado un hub colaborativo para gestionar la colaboración en empresas, la comunicación formal e informal, la gestión documental y mucho más gracias a cientos de integraciones posibles (Google Drive, DropBox, Slack, Gmail, etc.). La centralización de la información refuerza la capacidad de la herramienta para aumentar la productividad.

- **Universalidad**

  Asana es un software que conviene tanto al freelance que busca un gestor de tareas como a las grandes corporaciones, con cientos de empleados, que buscan una solución para la gestión de carteras de proyectos o PPM (Project Portfolio Management) para la gestión de proyectos complejos. Las diferentes vistas y la flexibilidad en la experiencia del utilizador permiten a todos los miembros del equipo sentirse cómodos en la utilización del programa.

3.2.2 TestLink

Posee una interfaz simple y fácil de usar, permite crear y gestionar casos de pruebas y organizarlos dentro de planes de pruebas. Esta herramienta permite crear planes de pruebas para ser ejecutados posteriormente. Permite asignar y priorizar tareas, así como también generar informes.

TestLink no tiene su propio rastreador de errores, pero se integra con éxito con otros sistemas similares para el seguimiento de problemas.

Características:

- Crear y describir el requisito de su producto
- Crear Casos de prueba Sobre la base de estos requisitos.
- Agrupar sus casos de prueba en una Plan de prueba
- Cubrir los requisitos del cliente (o los suyos propios) con casos de prueba.
- Seleccione una persona para la prueba
- Recibir el informe una vez finalizada la prueba

3.2.3 Trello

Esta es una herramienta digital, entre muchas otras de su tipo, que permite organizar visualmente contenidos diversos y compartirlos con otras personas. Contiene una interfaz visual que funciona de forma muy intuitiva. Además, es muy flexible. Actualmente es usada por millones de personas y empresas de todos los tipos y tamaños, en especial para apoyar la gestión de proyectos y tareas en equipo.

Para que el uso de Trello se sostenga en el tiempo y genere valor, es fundamental que la cultura organizacional, propia del equipo, se oriente al uso efectivo de la herramienta. El autor (Javier Guillot, 2019) menciona mas a detalle los aspectos relevantes en su blog de internet.

- **Asegurar un propósito compartido:**

  Identificar colectivamente la necesidad de gestionar tareas en equipo. Para ello, es útil invitar a un ejercicio grupal de reflexión en el que se identifiquen los “dolores colectivos” relacionados con la ausencia de procesos y herramientas compartidas para gestionar tareas (ejemplos: ineficiencias, errores de comunicación, dificultades de

coordinación) y los posibles beneficios de usar una única herramienta para este propósito.

- **Asegurar el liderazgo:**

  Es imprescindible que las personas que ocupan roles de liderazgo promuevan el uso de la herramienta mediante su ejemplo constante, usándola para la gestión de sus propias tareas y como referencia para la coordinación, seguimiento y reporte.

- **Balancear reglas fijas de uso en equipo con flexibilidad de uso individual**

  Es necesario construir y compartir reglas en equipo para que el uso como herramienta de gestión y coordinación sea consistente. También es necesario dejar espacio para la flexibilidad en la aproximación individual a la definición de tareas, en especial motivando a que cada persona agregue sus propias tareas y las actualice según su progreso.

- **Adoptar una lógica de experimentación iterativa:**

  Evaluar periódicamente el uso de Trello y recabar evidencia sobre fortalezas y oportunidades de mejora (por ejemplo, mediante el uso de formularios anónimos en línea).

3.2.4 Jira

Es una herramienta de seguimiento de problemas o gestión de proyectos. Por lo general, se utiliza para el seguimiento de errores o problemas, y la gestión de proyectos.

Los aspectos fundacionales conceptuales de JIRA son el problema, el proyecto y el flujo de trabajo. Seguidamente se ilustra como lo explica (YANA GUSTI, 2019) en su apartado WEB

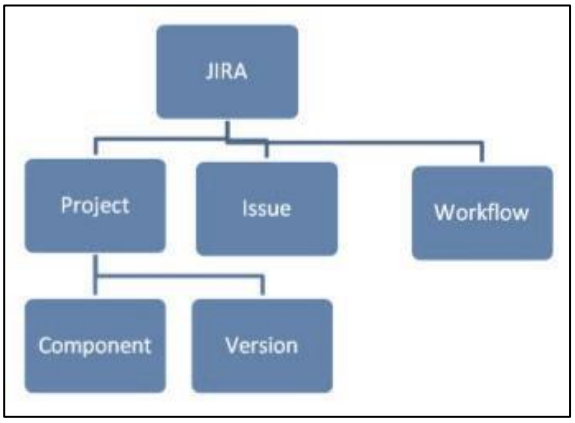


Ilustración 4 Aspectos conceptuales de Jira

- **Issue (Problema):** se puede considerar como cualquier actividad que se cree y se rastree a través de Jira como, por ejemplo:
    - Error de software
    - La tarea de un proyecto
    - El formulario de solicitud de licencia
    - Ticket de ayuda.
- **Flujo de trabajo:** Conjunto de estados y transiciones por los que una incidencia se mueve durante todo su ciclo de vida, representando los procesos internos de la organización, estas etapas son:
    - Tema abierto
    - Problema resuelto
    - Problema en progreso

  - Problema reabierto
  - Problema cerrado.

- **Proyecto:** Consiste en una colección de temas, sus principales atributos son:
  - Nombre seleccionado por el administrador
  - Identificador para el nombre del proyecto
  - Agrupación lógica de problemas dentro de un proyecto

Este software de gestión proporciona múltiples utilidades, una de las más relevantes es que permite tratar con errores o bugs dentro del software El potente motor de flujo de trabajo de Jira garantiza que los errores se asignen y prioricen automáticamente una vez que se capturan. Luego, los equipos pueden rastrear un error hasta su finalización.

3.3 Costos de realizar pruebas de software

Las pruebas software son de vital importancia dentro del ciclo de vida del desarrollo de un producto. Proporcionan información valiosa sobre la calidad alcanzada y son en sí mismas un mecanismo efectivo para descubrir fallos en el funcionamiento del software.

Es necesario comprender que como cualquier otra fase del desarrollo de software conlleva a gastos de tiempo, dinero y recursos. Es por esto importante analizar y determinar en qué se va a invertir los esfuerzos con el fin de no malgastar ninguna clase de bien.

El autor (Cecilia García García, 2019) ha señalado distintos costos que se pueden evidenciar en equipos de trabajo:

- **Costos iniciales:**
  - Evaluar y seleccionar la herramienta adecuada.

- Comprar la herramienta o adaptar las herramientas open source, o bien desarrollar una herramienta personalizada.
- Priorizar los casos de prueba a ser automatizados. Por ejemplo, se puede considerar primero en la priorización los casos de prueba que tengan mayor tasa de reúso (se ejecutan con la mayor frecuencia en el proceso actual: en pruebas de humo, en pruebas de regresión, en pruebas de sanidad)
- Generar los datos de prueba a consumir por los scripts de prueba.
- Aprender a usar la herramienta apropiadamente. Esto incluye los costos de transferencia de conocimiento dentro de la organización y la generación de nuevo conocimiento; por ejemplo: diseñar y documentar la nueva arquitectura de pruebas.
- El costo de integrar la herramienta con el proceso de pruebas actual, la integración con otras herramientas existentes (por ejemplo, la herramienta de gestión de casos de prueba o gestión de defectos, o integración continua) y la integración con el equipo de pruebas actual.

- **Costos recurrentes:**
  - Mantener la herramienta y los scripts de prueba. Este costo está asociado a la durabilidad de un script, es decir cuánto tiempo dura el script antes de que sea modificado.
  - Mantener los scripts de generación de datos de prueba.

- Ejecución de los scripts de pruebas y analizar los resultados. Este costo puede ser estimado mediante la multiplicación del tiempo en correr los scripts por la frecuencia de ejecución y por la tarifa por unidad de tiempo.
- Extender la cobertura a nuevas funcionalidades de la aplicación bajo pruebas, o nuevas aplicaciones.

Los aspectos anteriores hacen referencia a los gastos que conllevan la gestión de pruebas, de la misma manera, hay que tener presentes los gastos cuando se solucionan los errores. “Se ha demostrado que cuanto antes se detecte un error mucho menor será el coste de corregirlo, reduciendo así su impacto en el proyecto”(Montenegro, 2017).

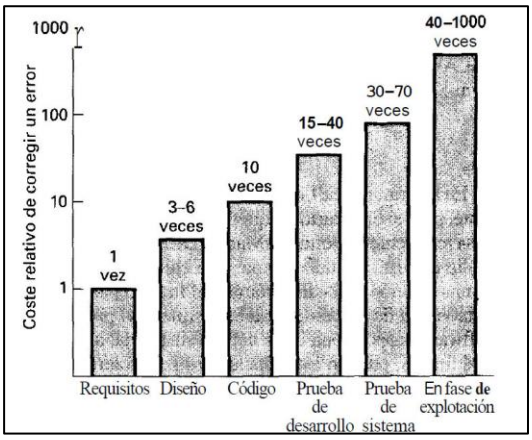


Ilustración 5 Coste relativo de corregir un error

Se recomienda que el proceso de pruebas debe de comenzar lo antes posible y comenzar desde lo más pequeño hasta lo más grande. Las pruebas se van ejecutando conforme el sistema vaya creciendo en función de la fase de desarrollo.

Al momento de comenzar con la automatización de un proyecto, se debe de contar con una base sólida comenzando con los casos de pruebas unitarios evitando la mayor cantidad de errores posibles.

Esta automatización debe contar con una retroalimentación inmediata y continuar sucesivamente a las diferentes capas. De esta forma, las pruebas manuales y exploratorias son más valiosas en el nivel de la interfaz de usuario, centrándose en aquellas pruebas que no son posibles de automatizar.

En la siguiente ilustración se presenta la manera común en la cual se focalizan los esfuerzos, gastando demasiado tiempo en construir pruebas automatizadas que prueben la interfaz.

Este tipo de esfuerzo no es recomendable ya que este tiempo podría ser mejor aprovechado siendo usado en las pruebas unitarias.

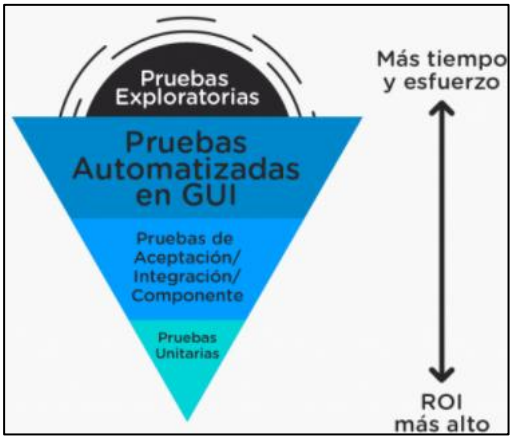


Ilustración 6 Anti patrón de las prioridades de las pruebas

En la siguiente figura, se puede apreciar la manera ideal de concentrar los esfuerzos de las pruebas, en donde las pruebas unitarias tienen el mayor peso, luego los test de componentes e integración y por último se debería de poner menos esfuerzos en las pruebas de interfaz de

usuario. La explicación de la ilustración de la pirámide enseña que se debe tratar de anticipar los errores que se pueden producir para así tener controlados los elementos de más bajo nivel para que no se vayan a propagar hacia arriba. Además, el hecho de tener una amplia cobertura de tests unitarios documenta nuestro producto y hace que sepamos exactamente qué y dónde está fallando y facilita su lectura, de modo que cuando se incorpora una persona nueva puede ver más rápidamente lo que se está haciendo. Resulta todo ser más ventajoso porque los tests que más rápido se ejecutan siempre son los tests unitarios.

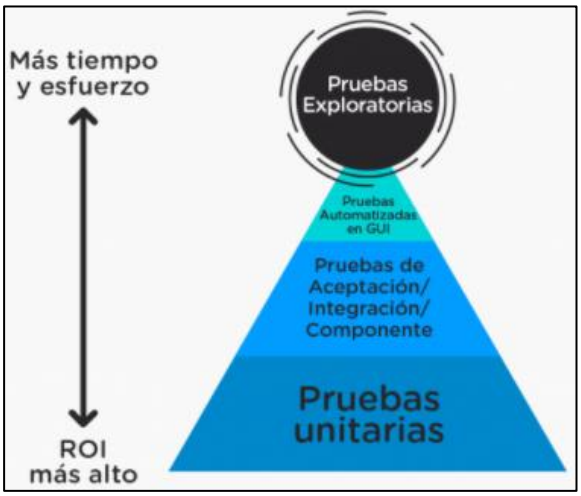


Ilustración 7 Prueba de automatización ideal

3.4 Herramientas de desarrollo WEB y de aplicaciones móviles

Desarrollo web significa construir y mantener sitios web; es el trabajo que tiene lugar en un segundo plano y que permite a una página web tener una apariencia impecable, un funcionamiento rápido y un buen desempeño para permitir la mejor experiencia de usuario.

El desarrollo web se hace a través de diversos lenguajes de programación. El lenguaje que se usa en cada momento depende del tipo de tarea que se esté haciendo. El desarrollo web se divide, de forma general, en Frontend (la parte cliente) y Backend (la parte servidor) (Mercedes, 2017) .

Existen diversos frameworks entre ellos se mencionan los más populares a continuación

- **Angular**

  Es un framework opensource desarrollado por Google para facilitar la creación y programación de aplicaciones web de una sola página, las webs SPA

  Angular separa completamente el frontend y el backend en la aplicación, evita escribir código repetitivo y mantiene todo más ordenado gracias a su patrón MVC asegurando los desarrollos con rapidez, a la vez que posibilita modificaciones y actualizaciones (QUALITY DEVS, 2019).

- **React**

  Esta librería está basada en un paradigma llamado programación orientada a componentes en el que cada componente es una pieza con la que el usuario puede interactuar. Estas piezas se crean usando una sintaxis llamada JSX permitiendo escribir HTML y opcionalmente CSS dentro de objetos JavaScript.

  Estos componentes son reutilizables y se combinan para crear componentes mayores hasta configurar una web completa. Esta es la forma de tener HTML con toda la funcionalidad de JavaScript y el estilo gráfico de CSS centralizado y listo para ser abstraído y usado en cualquier otro proyecto.(Albert Campillo, 2020).

- **Vue js**

  Es un tipo de framework progresivo, esto quiere decir que podemos crear todo tipo de desarrollos con él. Podrían ser componentes sencillos, que implementan una parte determinada de una aplicación web, pero también aplicaciones frontend completas, con su sistema de enrutamiento y cantidad de lógica de negocio.

  VueJS implementa lo que se conoce como arquitectura de componentes. Permite dividir las aplicaciones en bloques con funcionalidades independientes, llamados componentes. Esos bloques podrían ser una cabecera, un menú, un listado, una ficha de producto. En realidad, cualquier cosa que se pueda necesitar puede ser un componente. Además, unos componentes se pueden apoyar en otros, de modo que en un listado de productos se pueden tener fichas de productos, que a su vez pueden estar compuestas por datos, botones, desplegables con información, entre otros(Jose M Baquero García, 2020).

4 Definición base

Con los fundamentos encontrados y conocimientos adquiridos en el capítulo anterior se puede realizar la toma de decisiones sobre qué tipo de herramienta de gestión, que software de testing y que frameworks conviene usar comparando diversos aspectos y determinar cuáles de ellos se adaptan mejor a los procesos operativos de la empresa BeGo. En este capítulo se mostrará a detalle el por qué se seleccionaron dichas herramientas realizando un análisis comparativo examinando sus ventajas, desventajas, dificultades y costos para generar el mayor beneficio para la empresa.

4.1 Selección de la herramienta de gestión

Dadas las herramientas de gestión consultadas se evaluarán comparándolas con el fin de examinar cual de todas ellas resultaba ser más provechosa para la empresa. Uno de los principales factores que se pretende encontrar es que la herramienta sirva tanto para la gestión de proyectos como para la gestión de errores e incidencias, puesto que, se decidió con el equipo de desarrollo que sería más complicado tener estas dos herramientas separadas.

4.1.1 Clasificaciones según el sector al que se enfocan

Los distintos softwares de gestión fueron creados tratando de ser útiles para sectores de trabajo en específico o siendo lo suficientemente versátil para adaptarse a cualquiera sin problema.

En la siguiente tabla se muestra para que sectores fueron desarrollados las herramientas de gestión.

	Nube/Instalado	**Trabajo remoto (App)**	**Sector**

Trello	Nube	Android, iOs	Cualquier tipo de sector o proyecto
Jira	Nube	Android, iOs	Marketing, operaciones, legal, finanzas, TI, ingeniería, desarrollo de software
asana	Nube	Android, iOs	Ingeniería, diseño, ventas, RRHH, desarrollo de producto, asistentes administrativos
TestLink	Nube	No	Ingeniería, desarrollo de software

4.1.2 Identificación de las ventajas y desventajas

Para la comparación se utilizó información útil hallada en la página escrita por (JAVIER GOBEA, 2018) en donde menciona bajo su experiencia el resultado de trabajar con dichas herramientas y señala diferentes puntos a favor y en contra de las herramientas.

- **Asana**

Ventajas	Desventajas
• Actualizaciones en tiempo real que permiten ver rápidamente los últimos cambios hechos. • Interfaz sencilla e intuitiva que facilitan mucho el uso de la herramienta y reducen la curva de aprendizaje. • Función de prioridad de tareas, para ayudarte a ti y a tu equipo a ser más productivo y eficaz. • Versión gratuita muy completa. • Visualización de objetivos generales para supervisar el progreso del proyecto.	• No tiene acceso fuera de línea. • La función de búsqueda necesita algo más de desarrollo, ya que es un poco torpe. • Los planes de pago incluyen funcionalidades que pueden resultar poco útiles.

Tabla 2 Ventajas y desventajas del software de Asana

- **Trello**

Ventajas	Desventajas
• Sistema de arrastrar y soltar muy sencillo que permite cambiar el estado de una tarea de forma muy cómoda. • Eficaz a la hora de compartir información relacionada con una tarea: puedes adjuntar explicaciones y archivos para cada tarea en su correspondiente tarjeta y estar disponible para todos los miembros de tu equipo. • Tableros visuales que facilitan la organización, gestión y el seguimiento de tareas. • Posibilidad de ampliar el plan gratuito a uno de pago para conseguir más funcionalidades, integraciones y seguridad avanzada.	• Puede resultar demasiado simple si quieres gestionar proyectos complejos • No puedes filtrar tareas por usuario o fechas límite. • Si una columna tiene demasiadas tarjetas de tareas, se pierde la posibilidad de verlo todo de un vistazo porque obliga a hacer scroll con el ratón.

Tabla 3 Ventajas y desventajas de Trello

- **Jira**

Ventajas	Desventajas
• Colaboración en tiempo real entre los miembros del equipo. • Integración con otras herramientas de trabajo. • Actualización continua del desarrollo de los flujos de trabajo. • Permite gestionar y mantener las pruebas, así como las incidencias • Permite trabajar bajo un marco como scrum	• Personalización limitada. • Base de conocimiento no integrada. • No hay gestión de los elementos de configuración.

Tabla 4 Ventajas y desventajas de Jira

- **TestLink**

Ventajas	Desventajas

• Los casos de prueba se pueden organizar por una estructura jerárquica modulo – caso de uso, y pueden ser utilizados en diferentes iteraciones, proyectos y ejecuciones. • El diseño del caso de prueba se adapta al formato definido en el SGI, lo que permite configurar un resultado esperado en cada uno de los pasos. • Permite diseñar casos de prueba genéricos para ser reutilizados en diferentes proyectos • Permite manejar las versiones de los casos de pruebas.	• No permite hacer el seguimiento de bugs. • No se puede clasificar los errores desde su inicio hasta su finalización. • No tiene aplicaciones móviles para facilitar el trabajo remoto

Tabla 5 Ventajas y desventajas de TestLink

Bajo la necesidad de la empresa BeGo para comenzar a organizar de una manera más formal la planeación de sus proyectos se determinó que por cuestiones de factibilidad había que buscar un software de gestión que además se integrara con las demás necesidades como lo son el proceso de integración continua y gestión de pruebas, es por esto que la herramienta de TestLink se ve descartada ya que dentro de sus funcionalidades no se enfoca de manera profunda en la gestión de proyectos y no tiene soporte para aplicaciones móviles haciendo complicada la comunicación en el equipo de trabajo.

En la empresa BeGo se decidió hacer uso durante un periodo determinado de las otras herramientas de gestión escogidas para lograr obtener un mayor apoyo y consistencia de la información recogida.

Primeramente, se empezó a utilizar Asana, antes de esto, el equipo de desarrollo no había utilizado ninguna herramienta de gestión por lo que resulto un poco costoso acostumbrarse

en cuestión de tiempos, no hubo una buena adaptación debido a que se necesitaba clasificar de mejor manera las tareas y la herramienta resulto poco flexible.

Posteriormente se migro a Trello en donde se lograba una personalización bastante mayor y mayor fluidez a la hora de crear y ejecutar tareas, además de una interfaz mucho más fácil e intuitiva de manejar. Sin embargo, surgía la necesidad de implementar una metodología ágil en los proyectos e integrar el trabajo con un repositorio, es por esto que se buscó una herramienta más completa la cual podría cumplir con los requerimientos por el equipo de desarrollo.

Como siguiente alternativa se tomó Jira el cual permite integrarse con un repositorio para el control de versiones y presentaba un flujo de trabajo más completo además de poder incluir de manera controlada una metodología ágil como lo es en este caso Scrum, el equipo se adaptó de manera rápida a esta herramienta ya que sus funcionalidades le resultaban mejor y más útiles que las otras herramientas de gestión ya antes mencionadas.

Después de haber realizado varios Sprints de prueba se concluyó que Jira seria la nueva herramienta para trabajar del equipo de desarrollo dado que facilitaba hacer el seguimiento de estado y control de las tareas, además de ser utilizada como herramienta colaborativa para evitar pérdida de información, mejorar la gestión y almacenamiento de la documentación y conseguir un buen trabajo en equipo.

4.2 Selección del tipo de framework de desarrollo

Para esta selección se realizó una investigación para determinar cuáles frameworks se adaptan mejor al equipo teniendo en cuenta sus conocimientos y experiencia además se realizó una revisión de la popularidad de estos frameworks a nivel mundial.

4.2.1 Identificación de las ventajas y desventajas de cada framework

Para la comparativa se utilizó información ya existente de (Juan Amat, 2020) autor conocedor en los 3 frameworks que se van a examinar, para esto se plasmará tablas de ventajas y desventajas para determinar cuál resultaría mejor. Las tablas con sus respectivos resultados se muestran a continuación.

- **Angular**

Ventajas	Desventajas
• Está creado para ser usado junto con TypeScript. • Tiene características como una generación de bibliotecas npm basadas en el CLI del mismo. • Posee una documentación detallada. Sin embargo, esto provoca una curva de aprendizaje un poco más grande. • Enlace de datos unidireccional que permite un comportamiento singular para la aplicación que minimiza los riesgos de posibles errores. • MVVM (Model-View-ViewModel), permite a los desarrolladores trabajar por separado en la misma sección de la aplicación utilizando el mismo conjunto de datos.	• La variedad de estructuras diferentes hace que sea un poco más difícil de aprender en comparación con React y Vue.js, que tienen un único "Componente" en mente. • Rendimiento relativamente más lento, según diferentes puntos de referencia. Por otro lado, puede abordarse fácilmente utilizando la llamada "ChangeDetectionStrategy", que ayuda a controlar el proceso de renderizado de componentes de forma manual.

• Tiene una estructura y arquitectura específicamente creadas para una gran escalabilidad del proyecto.	

Tabla 6 Ventajas y desventajas de Angular

- **React**

Ventajas	Desventajas
• Es muy fácil de aprender gracias a su diseño simple, el uso de JSX y una documentación muy detallada. • Los desarrolladores pasan más tiempo escribiendo JavaScript moderno y menos tiempo preocupándose por el código específico del framework. • Es extremadamente rápido, cortesía de la implementación del Virtual DOM de React y varias optimizaciones de renderizado. • Tiene un gran soporte para la representación del lado del servidor. • Soporte de primera clase de Progressive Web App (PWA). • También tenemos Redux, el framework más popular para administrar el estado de las aplicaciones en el mismo, es fácil de aprender y dominar. • Implementa conceptos de Programación Funcional. • Las aplicaciones se pueden hacer de tipo seguro con TypeScript. • En general, migrar entre versiones es muy fácil, ya que Facebook proporciona "codemods" para automatizar gran parte del proceso. • Las habilidades aprendidas en este se pueden aplicar al desarrollo React Native.	• La comunidad está dividida sobre la mejor manera de escribir CSS en el mismo, dividida entre hojas de estilo tradicionales (módulos CSS) y CSS-in-JS (es decir, componentes de estilo y emoción). • Se está alejando de los componentes basados en clases, lo que puede ser una barrera para los desarrolladores que están más cómodos con la Programación Orientada a Objetos (POO). • Al principio, mezclar plantillas con lógica (JSX) puede ser confuso para algunos desarrolladores.

Tabla 7 Ventajas y desventajas de React

- **Vue.js**

Ventajas	Desventajas
• Vue.js tiene muchas características similares con el framework de Google y esto puede ayudar a optimizar el manejo de bloques HTML con el uso de diferentes componentes. • Vue.js tiene documentación muy circunstancial que puede acelerar la curva de aprendizaje para los desarrolladores y ahorrar mucho tiempo para desarrollar una aplicación utilizando sólo los conocimientos básicos de HTML y JavaScript. • Vue.js se puede usar tanto para crear aplicaciones de una sola página como para interfaces web de aplicaciones más complejas. Lo principal es que las partes interactivas más pequeñas se pueden integrar fácilmente en la infraestructura existente sin ningún efecto negativo en todo el sistema. • Vue.js puede ayudar a desarrollar plantillas reutilizables bastante grandes que se pueden hacer sin gastar tiempo por su estructura simple. • Vue.js puede pesar alrededor de 20 KB manteniendo su velocidad y flexibilidad que le permite alcanzar un rendimiento mucho mejor en comparación con otros frameworks.	• Lamentablemente Vue.js todavía tiene una participación del mercado bastante pequeña en comparación con los anteriores, lo que significa que el intercambio de conocimientos en este framework aún está en la fase inicial. • A veces, Vue.js puede tener problemas al integrarse en grandes proyectos y todavía no hay posibles soluciones, pero definitivamente vendrán pronto.

Tabla 8 Ventajas y desventajas de Vue

4.2.2 Diferencias entre los frameworks

A continuación, se presenta una comparación entre distintos atributos de los frameworks en donde se quiere ver la diferencia en cuestiones de espacio en el disco, velocidad de desarrollo, documentación disponible y rendimiento.

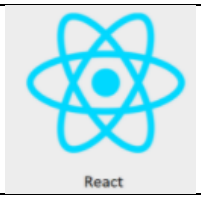
Atributos			
Tamaño producción	167 KB	110 KB	30 KB
Tamaño en desarrollo	1200 KB	774 KB	279 KB
Velocidad de programación	Lenta	Normal	Rápida
Documentación	Extensa	Extensa	No muy Extensa
Rendimiento	Alto	Alto	Alto
DOM	Regular	Virtual	Virtual
Tiempo de inicio	Largo	Corto	Corto

Tabla 9 Diferencias entre frameworks (Luismi Gracia, 2020)

4.2.3 Agrado y aceptación de la comunidad

Se ha revisado Stack Overflow el cual es un sitio de preguntas y respuestas sobre temas relacionados a lenguajes de programación en donde la comunidad se ha ido fortaleciendo y creciendo en los últimos años. Esta plataforma en febrero de 2020 realizo una encuesta a 65000 desarrolladores para que votaran con que tecnologías se adeudaban más.

A continuación, se muestran sus resultados.

- Resultados con respecto a los Frameworks Web

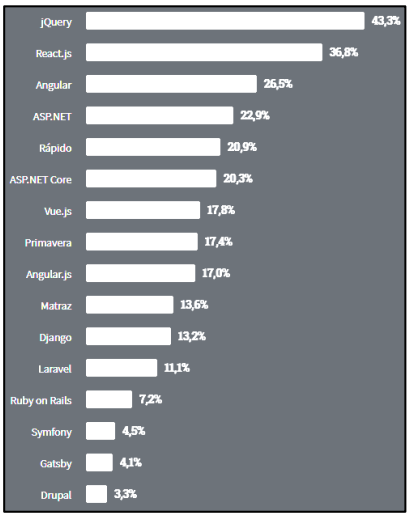


Ilustración 8 Frameworks con más aceptación por la comunidad

Se puede apreciar que jQuery es el más aceptado por la comunidad, pero poco a poco está perdiendo terreno frente a React.js y Angular año tras año.

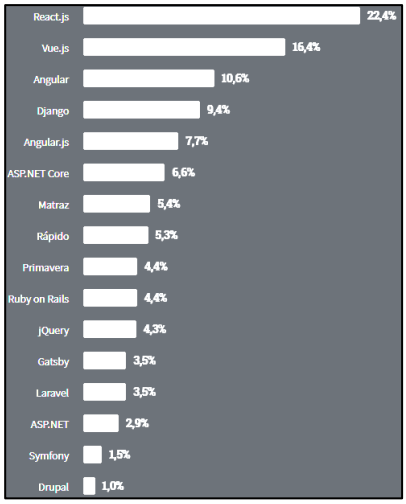


Ilustración 9 Frameworks más queridos

Entre los frameworks más queridos por la comunidad de desarrollo es React.

Por temas de poco crecimiento del framework de Vue y el hecho de que no haya incursionado mucho en temas de desarrollo móvil se ha decidido descartar, además de que la comunidad

no es tan grande, entonces si se presenta un problema sería más compleja la búsqueda de la posible solución.

Se ha presenciado en la investigación de los frameworks que angular es un marco bastante integral, aunque se puede ver esto como una característica negativa cuando los desarrolladores no necesitan la mayoría de las características que esta herramienta de trabajo ofrece causando que la aplicación final y de desarrollo tenga un peso mayor al necesitado.

El hecho de que React funcione bajo un DOM virtual garantiza mejores rendimientos que angular.

El autor de este trabajo junto con el equipo de desarrollo ha decidido que el framework que mejor se adapta a la empresa de BeGo es React, puesto que trabajar en la construcción de un aplicativo WEB y móvil es muy similar resulta beneficioso debido a que los conocimientos se pueden reutilizar y así ahorrar tiempos en las implementaciones y capacitación del personal.

4.3 Selección para las herramientas de pruebas automatizadas

Se evaluará cual es el software de testing más utilizado en la actualidad gracias a los resultados de tendencias de NPM (Node Package Manager). NPM es un gestor de paquetes, el cual facilita el trabajar con Node, ya que, gracias a él se puede acceder a cualquier librería disponible con solo una línea de código, NPM ayuda a administrar los módulos, distribuir paquetes y agregar dependencias de una manera sencilla (Alexander Guevara Benites, 2018).

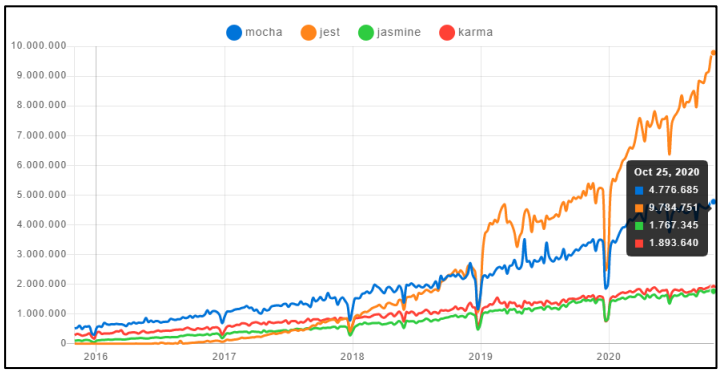


Ilustración 10 Tendencias de descargas por NPM

En esta ocasión se decidió por Jest para el Frontend debido a que se integra muy bien con el framework de React y es el más aceptado por la comunidad, además que cuenta con una documentación bastante detallada y completa gracias a que es desarrollada por el equipo de Facebook. En el caso del BackEnd dado que la arquitectura de la empresa cuenta con diferentes lenguajes no se puede utilizar Jest, por lo tanto, se decidió usar la herramienta de PostMan para realizar la escritura de pruebas que validen la comunicación con el servidor.

5 Diseño

Dadas las decisiones y la información establecida en el capítulo 4 se obtuvo cuales herramientas son más beneficiosas para implementar en la empresa de desarrollo BeGo. Ahora bien, se explicará detalladamente el uso y la manera en la que se establecieron estas herramientas en conjunto con el fin de implementar de manera adecuada un procedimiento para que el rendimiento y la calidad del producto software mejoren en consideración con su trabajo previo.

5.1 Procedimiento para plan de pruebas

Es necesario contar con un procedimiento de pruebas para determinar los objetivos, la estructura, los métodos, los pasos a seguir y los demás aspectos que deberían de ser empleados en las pruebas. A continuación, se presentarán una serie de 10 aspectos importantes para elaborar un plan de pruebas del software, consultados en el blog (PMOinformatica, 2016) donde se detalla con mayor profundidad su contenido.

5.1.1 Realización de análisis de requerimientos de desarrollo de software

Primero antes de elaborar un plan de pruebas es necesario comprender de manera profunda los requerimientos del sistema, puesto que son el sujeto de la verificación de la calidad que se va a realizar.

5.1.2 Detectar las nuevas funcionalidades que deben ser probadas

Es importante identificar y adicionar en el plan de pruebas de software la lista de las características nuevas en el sistema.

5.1.3 Identificar las funcionalidades de sistemas existentes que deben probarse

Hay que identificar las funcionalidades existentes que estén siendo impactadas por el desarrollo actual, considerando todos los componentes afectados en todas las capas de la arquitectura de software.

Se ha determinado mediante la práctica que realizar pruebas a la interfaz de usuario resulta costoso en cuestiones de gasto de tiempo, es por esto necesario decidir si realmente vale la pena escribir casos de prueba para validar la interfaz o determinar si resulta más provechoso hacer test exploratorios de forma manual para comprobar que todo funcione de acuerdo a como debería. Se recomienda que se establezca la mayor cantidad de tiempo para escribir pruebas funcionales.

Situaciones que se puede encontrar al identificar estas funcionalidades:

1. **Funcionalidades modificadas de la interfaz**: verificar si una funcionalidad está siendo modificada agregando más pantallas o cambios a su flujo de proceso, debe ser incluida en el plan de pruebas de software.
2. **Funcionalidades modificadas en sus componentes internos**: Son funcionalidades modificadas manteniendo la misma interfaz gráfica y flujo de procesos, sin embargo, si se modifican componentes internos que comparten con otras funcionalidades del sistema, en las capas de lógica de negocio o acceso a datos. Estas deben incluirse en el plan de pruebas de software para determinar a partir de ellas pruebas de regresión a realizar.

5.1.4 Definir la estrategia de pruebas

Consiste en seleccionar cuáles son los tipos de pruebas de software que se deben realizar.

Es recomendable seguir un marco de referencia para determinar los tipos de prueba, como, por ejemplo:

- **Pruebas funcionales:** Se determinan los conjuntos de pruebas a realizar, correspondiente con cada funcionalidad nueva o existente que se esté modificando.

  Se tienen distintos tipos de pruebas funcionales, por ejemplo, las pruebas de sistema, que se realizan después que el equipo de desarrollo ha integrado los componentes de distintas capas.

  Las pruebas funcionales son definidas como pruebas basadas en especificación. Son diseñadas usando técnicas de diseño de pruebas de caja negra.

- **Pruebas de regresión**: Se definen sobre las funcionalidades modificadas en sus componentes internos.
- **Pruebas exploratorias**: Se deben de acostumbrar realizar cada cierto tiempo con el fin de probar la robustez del sistema y verificar de manera general si todo hace lo que se supone que debe hacer.

5.1.5 Definir los criterios de inicio, aceptación y suspensión de pruebas

- **Criterios de aceptación o rechazo:**

Para definir los criterios de aceptación o rechazo, es necesario definir el nivel de tolerancia a fallos de calidad. Si la tolerancia a fallos es muy baja puede definirse como criterio de aceptación que el 100% de los casos de prueba estén sin incidencias. Lograr

este margen en todos los casos de prueba principales y caso borde será muy difícil, y podría comprometer los plazos del proyecto, pero asegura la calidad del producto.

Por otra parte, puede ser que la intención sea realizar un mínimo producto viable, en ese caso se podría definir como criterio de aceptación el 100% de los casos de prueba principales y 20% de casos de prueba no principales (casos bordes).

Una vez logradas las condiciones, se darán por aceptadas las pruebas y el desarrollo de software.

- **Criterios de inicio o reanudación**

Definen las condiciones que deben cumplirse para dar inicio o reanudar las pruebas. Por ejemplo, en el caso de inicio la condición podría ser la instalación de los componentes de software en el ambiente y que los casos de pruebas de verificación de ambiente sean exitosos.

Para el caso de la reanudación las condiciones están relacionadas, se determina a partir de cuales criterios de suspensión se presentaron para detener las pruebas. Una vez que estás condiciones ya no existan (sean solventadas) se procede con la reanudación.

- **Criterios de suspensión:**

Las condiciones van a depender de los acuerdos de nivel de servicio internos de la organización y también de los acuerdos establecidos en cada proyecto individual.

Por ejemplo, si se tiene un equipo de pruebas que comparte su esfuerzo entre varios proyectos, se puede definir un criterio de suspensión exigente, un determinado porcentaje

de casos fallidos que resulten en incidencias. Si la condición se cumple, se detienen las pruebas y se dedica el personal a otras actividades,

Por otra parte, si se tiene un equipo de pruebas con personal dedicado, el criterio de suspensión puede ser poco exigente, por ejemplo, solo ocurriendo si se bloquean por incidencia todos los casos de prueba.

5.1.6 Identificar los entornos requeridos

Posteriormente se definen y documentan las características de los entornos de Hardware y Software necesarios para realizar la ejecución de las pruebas de software.

Esta información se obtiene a partir de los integrantes del equipo de desarrollo, quienes pueden suministrar los requisitos mínimos y óptimos para la operación del sistema.

Como mejor práctica, el ambiente de pruebas de software debería ser lo más similar posible al ambiente de producción, sin embargo, no siempre es posible debido a limitaciones de recursos (financieros). En estos casos debe estudiarse cuales son los requisitos que aseguran un mínimo de confiabilidad de estas pruebas respecto al entorno de producción.

Además, en esta sección del plan de pruebas, también se definen los requisitos de sistemas operativos, software y herramientas de las estaciones de trabajo de los Testers.

Si el alcance del proyecto incluye pruebas de aplicaciones (Apps) para móviles, es necesario definir los emuladores y teléfonos inteligentes, con sus respectivos requisitos.

También deben definirse los requisitos de hardware y software para los siguientes componentes:

- Herramienta de gestión de calidad de software.

- Herramientas para automatización de pruebas.
- Herramientas de Testing de Web Services.

5.1.7 Determinar necesidades de personal y entrenamiento

Debe completarse previamente la estimación del esfuerzo de pruebas a partir del diseño de casos de prueba.

Si aún no se cuenta con la estimación, se puede comenzar por definir los tipos de perfiles de habilidades y conocimientos en Software Testing que se necesitan.

Para ello se puede buscar la respuesta a las siguientes preguntas:

- ¿Qué conocimientos de procesos de negocio se necesitan?
- ¿Qué sistemas se están probando y quienes tienen experiencia en su funcionamiento?
- ¿Cuáles herramientas de gestión de calidad de software se van a utilizar?
- ¿Se necesitan conocimientos en herramientas técnicas como Lenguajes de programación o herramientas de pruebas de webservices?

5.1.8 Establecer la metodología y procedimientos de prueba

La metodología de pruebas de software dependerá de la que se esté utilizando para la gestión del proyecto. Dependiendo de la metodología hay que documentar los procedimientos para diseño y ejecución, siguiendo el orden de los pasos definidos, flujos de procesos, condiciones para tomar decisiones, y demás aspectos.

5.1.9 Elaborar la planificación de las pruebas

La planificación de las pruebas abarca:

- **Asignación de responsabilidades**

Esta se define con perfiles genéricos o inclusive con el equipo de trabajo si ya se conoce cuál es el que será asignado.

- **Cronograma**

Elaborado a partir de la estimación de las actividades de Software Testing realizada por el equipo.

Para elaborar un cronograma real, es importante definir actividades críticas como por ejemplo los tiempos de instalación de versiones en los entornos de pruebas, pruebas de validación de ambientes antes de comenzar a hacer las pruebas y las iteraciones por incidencias, que es el tiempo invertido en volver a probar los casos de prueba fallidos.

- **Premisas**

Son las condiciones que deben cumplirse para que el cronograma sea realizable, estas se determinan a partir de la documentación de entornos y de los requisitos de personal. Por ejemplo, disponibilidad de ciertos entornos, disponibilidad de personal con algún conocimiento técnico especifico y la metodología que se va a utilizar.

5.1.10 Identificar los riesgos y definir planes de respuesta

Para el Testing de Software, los riesgos por lo general están vinculados con factores como:

- Posibles dificultades en la disponibilidad de entornos.
- Pruebas que dependen de factores externos al proyecto y la organización.
- Disponibilidad de personal con conocimientos especializados en alguna herramienta, o en la funcionalidad especifica que se está desarrollando.
- Dependencias con otros proyectos.

- Posibilidad que alguna premisa no se cumpla.

Para identificar los riesgos es necesario enumerar cada una de estas dependencias y por medio de mesas de trabajo y tormentas de ideas pensar en las posibilidades de que algo salga mal.

Luego de la identificación, es necesario también definir planes de respuesta, los cuales deben ser específicos para cada situación particular y riesgo.

5.2 Instalación de versiones y herramientas que se van a trabajar

Dentro del diseño también es importante especificar el uso de las siguientes herramientas en las correspondientes versiones.

- Se instaló Jest en su versión estable 26.6
- Se instaló React junto con todas sus dependencias en su versión 16.8.6
- Se instaló React Native en su última versión 0.63
- Se instaló la versión 7.34.0 de Post Man
- Se utilizó la última versión disponible del software de Jira. 8.13.1

5.3 Definición del flujo de trabajo para errores

La herramienta de Jira permite realizar el seguimiento de Bugs y errores de un proyecto, asimismo no se enfoca solamente en defectos, sino también en hacer seguimiento de tareas de desarrollo y documentación entre otras.

5.3.1 versionamiento de código y versionamiento de relases

Dentro de las actividades del proyecto es importante que cuente con un sistema de versionamiento de código y versionamiento de relases.

Las versiones sirven de gran ayuda para asociar los defectos encontrados con:

- Versiones afectadas: en qué versiones del proyecto se ha verificado que el problema existe.
- Versiones corregidas: en qué versión fue arreglado el problema.

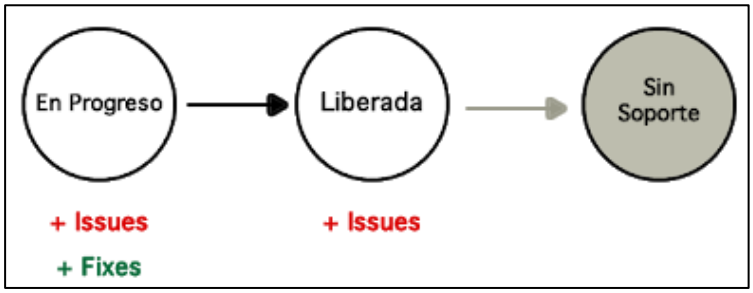


Ilustración 11 Flujo de trabajo de las versiones

En las versiones liberadas indica que no se pueden introducir más cambios, en tal caso habría que liberar otra versión a partir de ella. Las soluciones de los defectos se realizan solo sobre versiones en progreso.

5.3.2 Flujo de trabajo

Conjunto de estados y transiciones por los que se mueve un problema durante su ciclo de vida y generalmente representa procesos dentro de su organización, para ampliar la información se puede visitar el sitio oficial (Jira, 2020).

A continuación, se muestra un ejemplo de un flujo de trabajo predeterminado

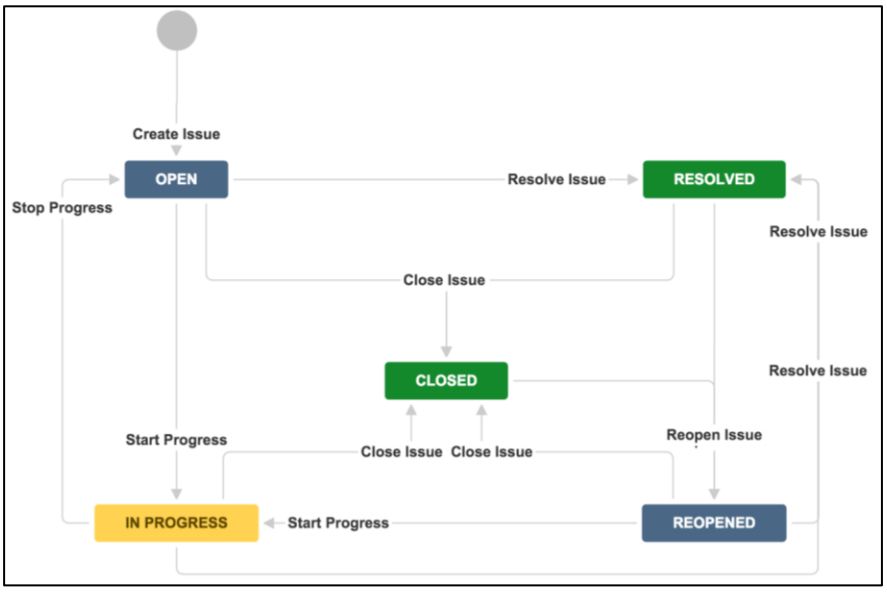


Ilustración 12 Flujo de trabajo predeterminado en Jira (YANA GUSTI, 2019)

5.3.3 Flujo de trabajo de un bug

Cuando se haya identificado un error, se debe de crear una incidencia y añadir todos los datos de importancia, incluidas

- Descripciones
- Nivel de gravedad
- Capturas de pantalla
- Versión
- Una persona que reporta el problema
- Una persona, a quien se asigna la tarea

Las incidencias pueden representar cualquier cosa, como un error de software, una tarea de proyecto o un formulario de solicitud de abandono, y cada tipo único de incidencia puede tener su propio flujo de trabajo personalizado.

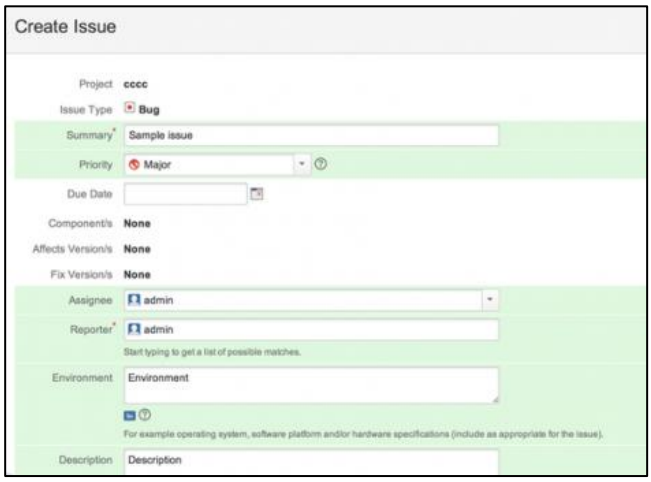


Ilustración 13 Ilustración de la creación de un problema

5.3.4 Principales atributos de un problema

Los principales atributos del problema son: Estados, Resoluciones y Prioridades.

- **Estado de emisión**: Indica el progreso de un proyecto.
    - **Open**: El problema está abierto para asignarlo y comenzar a trabajar en él.
    - **En progreso:** El problema está activado en el momento en que fue asignado.
    - **Reabierto:** El problema ya fue resuelto en algún momento. Pero la resolución estuvo incorrecta.
    - **Resuelto:** Se ha tomado una resolución y está pendiente de verificación por parte del reportero.
    - **Cerrado**: El problema fue considerado cerrado. La solución es correcta.

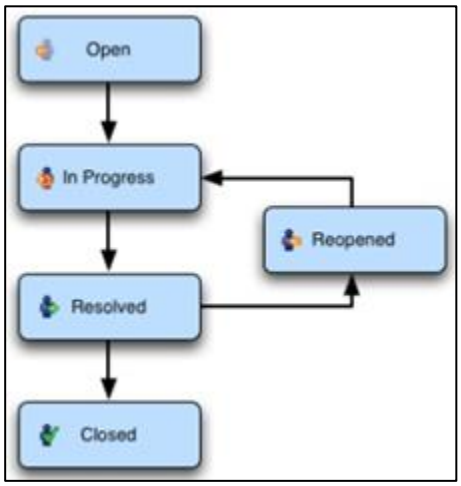


Ilustración 14 Estados de problemas

- **Priorización de los problemas**

Al encontrar un error, se debe de clasificar y priorizar según la importancia y la urgencia de la incidencia y la carga de trabajo del equipo.

Ilustración 15 Niveles de priorización en Jira

5.4 Implementación de las pruebas con Jest

5.4.1 Configuración del entorno de pruebas

Esencialmente se necesita contar con un proyecto que sea gestionado bajo el gestor de módulos de NPM, lo primero que se deberá de hacer es iniciar el proyecto con Jest, para esto se puede ejecutar el comando a continuación:

```
npm init -y
```

El cual creará toda la configuración por defecto de un proyecto e instanciará los módulos iniciales principales.

Para la instalación con todos los complementos de Jest se debe correr la siguiente instrucción en consola ubicados en la carpeta donde esta nuestro proyecto

```
npm install --save-dev jest
```

Ahora para poder ejecutar correctamente todos los módulos de nuestra aplicación procederemos a instalar otras dependencias necesarias con los comandos a continuación

```
npm i -D babel-jest babel-polify
```

```
npm i -D babel-preset-es2015
```

Ahora se creará un aun archivo llamado *./.babelrc* el cual permitira configurar las herramientas con el fin de que se puedan correr sentencias de las últimas versiones de JavaScript. Dentro de este archivo deberá ir la siguiente instrucción:

```
{"presets": ["es2015"]}
```

Por defecto dentro de la raíz de nuestro proyecto se deberá crear una carpeta especial donde se ejecutarán y almacenarán todas las pruebas, estas carpetas por convención se deberá denominar de la siguiente manera *__tests__* y todos los archivos de pruebas deberán de contener la extensión *.test.jsx* la cual permitirá que el intérprete de las pruebas las ejecute adecuadamente.

Por último, se deberá configurar el entorno para determinar de qué modo se ejecutarán las pruebas, esto se hace en el archivo que está en la raíz denominado *./package.json* el cual contiene las siguientes reglas para pruebas

```
"scripts": {
  "start": "webpack-dev-server --mode development --open --hot",
  "build": "webpack --mode production",
  "eject": "react-scripts eject",
  "test": "jest",
  "test:watch": "jest --watch",
  "test:coverage": "jest --coverage",
  "test:coverage-all": "jest --coverage --watchAll"
},
```

Ilustración 16 Configuración para ejecutar las pruebas

En donde se puede presenciar que se cuenta con diferentes opciones para pruebas las cuales se detallan a continuación:

- **Modo Watch:** Queda pendiente de los archivos para detectar cuando hay nuevos cambios y ejecuta pruebas relacionadas con los archivos que sufrieron modificaciones.
- **Modo watchAll:** Observa todos los archivos en busca de cambios y vuelve a ejecutar todas las pruebas cuando algo cambie.
- **Coverage**: Indica que información de cobertura de prueba debería ser recolectada y registrada en la salida.

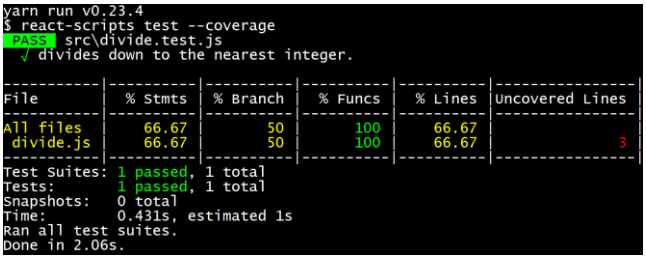


Ilustración 17 Ejemplo de un resultado de Jest Coverage (Matthew Garcia, 2016)

5.4.2 Estructura de las pruebas

A continuación, se presentará la estructura de un ejemplo de una prueba sencilla, la cual contiene entradas, funciones y resultados esperados.

```
// ejemplo.test.tsx
describe('Describe elementos comunes a probar', () => {
    test('Ejecuta un test dentro de un contexto', () => {
      expect(CodigoAProbar(Parametro0, Parametro1)).toBe(true);
    });

    test('Ejecuta otro test comun al anterior', () => {
      expect(FuncionVerdadera()).toBeTruthy();
    });
})
```

Ilustración 18 Formato de una prueba con Jest

- **Método describe**: se utiliza para contener una o más pruebas que se encuentren relacionadas, este método necesita de dos argumentos; una cadena para describir el conjunto de pruebas y una función que envolverá la prueba real.
- **Método test**: Este método contendrá una única prueba unitaria a ser ejecutada.
- **Expect**: se utiliza para poder acceder a un comparador de Jest para ejecutar la prueba.

Por último, para ejecutar el conjunto de pruebas se podrá correr el siguiente comando.

```
npm run test
```

Por último, el resultado por pantalla es el que se muestra a continuación, en donde se muestra un resumen detallado de cuales scripts de test se van a ejecutar, el tiempo que tomo cada uno de ellos y el total de todas las pruebas realizadas.

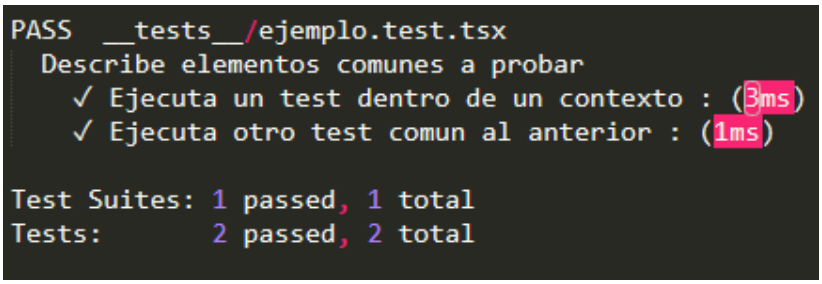


Ilustración 19 Resultado de ejemplo de ejecución de pruebas

5.4.3 Matchers

Jest tiene implementados Matchers los cuales permiten probar los distintos valores que se puedan originar en el código.

La función expect se utiliza cada vez que se desea testear un valor, esta función estará acompañada con una función de comparación para lograr afirmar algo sobre un valor.

Algunos de estas funciones más usadas y útiles en Jest son:

- **toBe:** Esta función suele ser usada para comparar valores primitivos o verificar si se está tratando de instancias del mismo valor.
- **toEqual:** Es usado para comparar recursivamente todas las propiedades de un objeto, también conocido como igualdad profunda.
- **toBeLessThan:** Es utilizado para comparar si el valor que se recibió es menor al que se espera.
- **toBeGreaterThan:** Es utilizado para comparar si el valor que se recibió es mayor al que se espera.
- **toBeTruthy**: verifica que, sin importar el contexto, el valor sea verdadero en un contexto booleano.

- **toBeFalsy**: verifica que, sin importar el contexto, el valor sea verdadero en un contexto booleano.
- **toBeUndefined**: Determina si el valor que ha llegado no está definido
- **toBeNull**: Se utiliza para verificar que el valor al que estamos apuntando es nulo.

5.5 Tipos de pruebas que se pueden realizar con Jest

Funciones lógicas: Son el tipo de funciones que más se utilizan, consisten en la llamada de componentes ya existentes que contengan extraído la lógica de una cierta funcionalidad, en donde dado un conjunto de entradas se logre inferir una salida.

Funciones Mock: Este tipo de pruebas permiten testear vínculos entre código deshaciéndose de la implementación real de una función, capturando llamadas a la función, capturando instancias de funciones de construcción cuando estas se crean instancias con new, y permitiendo configuración de valores de retorno al momento de probar. Por ejemplo, resulta bastante útil al momento en que se realiza una petición como cliente, puesto que, se puede mediante un mock hacer que la petición en vez de enviarse al servidor se envíe así mismo localmente y se puede dar una respuesta estándar.

Prueba de instantáneas: Las pruebas de instantáneas permiten asegurarse que la interfaz de usuario no cambie inesperadamente.

Un caso de prueba de instantánea típico representa un componente de la interfaz de usuario, toma una instantánea y luego la compara con un archivo de instantánea de referencia almacenado junto con la prueba. La prueba fallará si las dos instantáneas no

coinciden: el cambio es inesperado o la instantánea de referencia debe actualizarse a la nueva versión del componente de la interfaz de usuario.

5.6 Implementación de las pruebas con PostMan

La aplicación de PostMan permite realizar pruebas enfocadas al backend del sistema orientadas a simular la comunicación que tendría la estructura de cliente/servidor, el formato de escritura de una prueba en PostMan es bastante similar a las de Jest, estas comprender la sintaxis que se mostrará a continuación.

```
pm.test("Descripcion de la prueba que se ejecutara", function () {
    //sentencia de prueba
    pm.expect(pm.response.status).to.be(true)
});
```

Ilustración 20 Formato de una prueba escrita con PostMan

En donde se puede ejecutar la cantidad de pruebas necesarias para determinar si los datos que se envían desde el servidor sean correctos y correspondan de manera adecuada a las peticiones de los clientes

El resultado que arroja la consola de tiene el formato que se puede apreciar a continuación

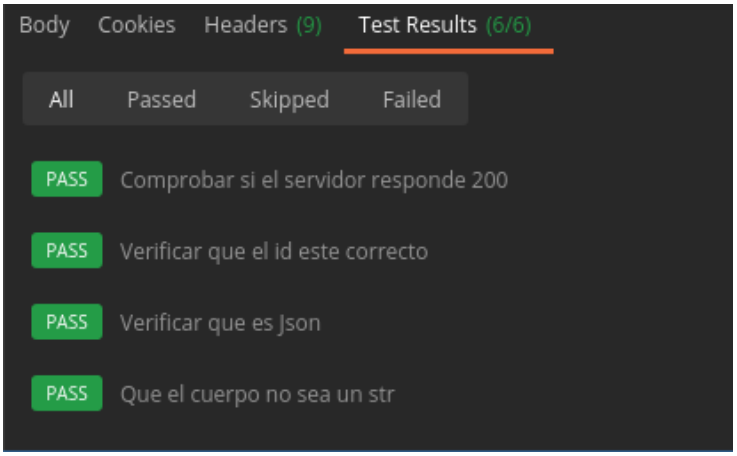


Ilustración 21 Resultado de la ejecución de pruebas con PostMan

5.7 Responder a solución de problemas con Smart Commits

Utilizando Jira y GitHub se puede hacer seguimiento al trabajo realizado que se dio para responder a una incidencia de un error propuesta gracias a las confirmaciones inteligentes y de esta manera llevar un control mayor estructurado sobre estas.

Para hacer uso de esta funcionalidad se debe seguir los pasos a continuación:

- Crear una incidencia en Jira donde se denote el error a tratar
- Hacer los cambios necesarios hasta solucionar el error propuesto
- Extraer el identificador único de la tarea en Jira
- Escribir un commit con el identificador de la tarea
- Subir los cambios realizados al repositorio remoto

Una vez completados los pasos anteriores GitHub se encargará de disparar una acción con el fin de que Jira mueva el problema como solucionado y quede el registro y soporte de que Commits y en que rama fue resuelto el error.

Ilustración 22 Commit inteligente, Jira

En la siguiente ilustración bajo la estructura de BPMN se presenta un gráfico en donde se describe el procedimiento de pruebas que se adapta de mejor manera a los procesos operativos de la empresa BeGo.

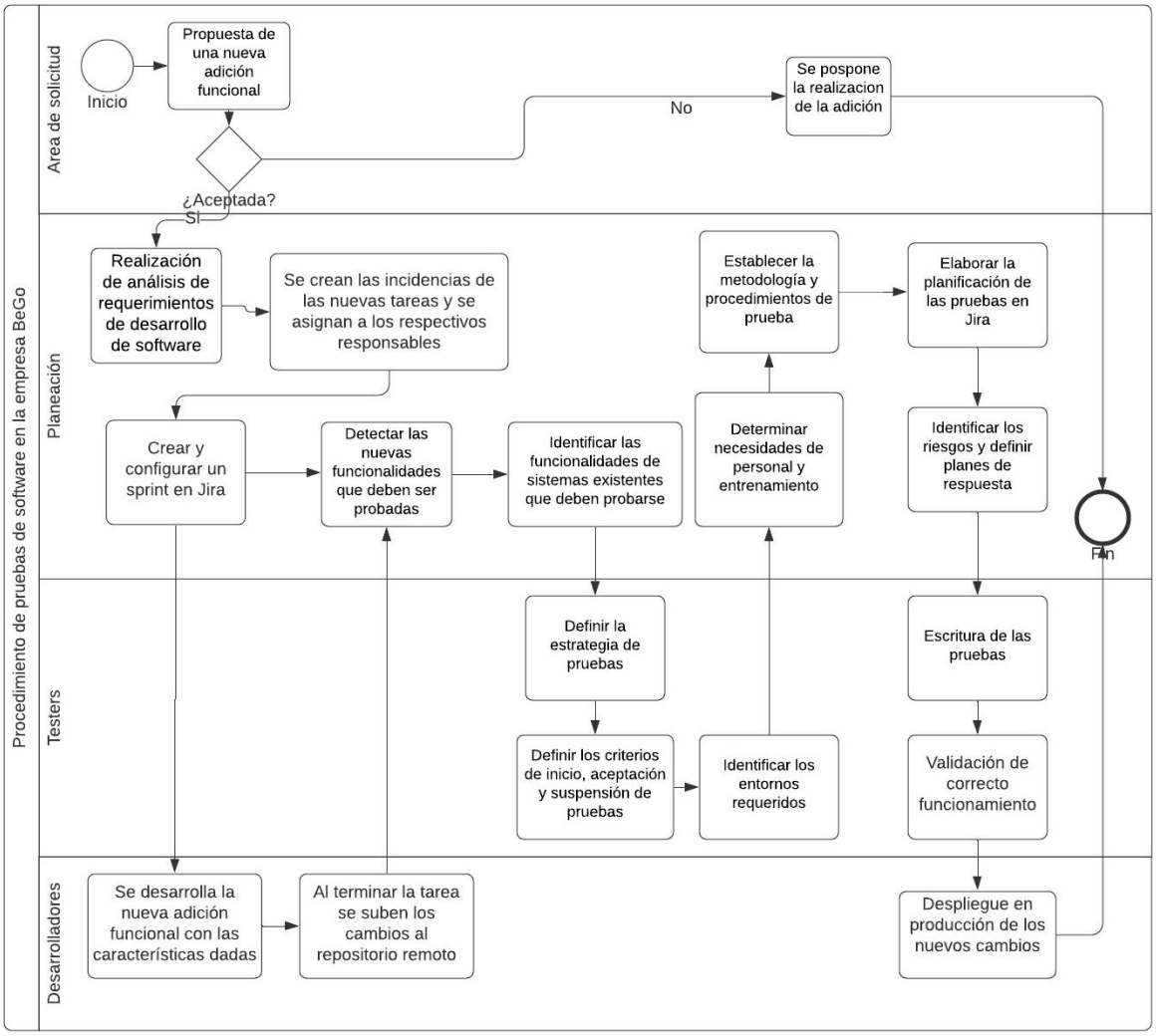


Ilustración 23 Representación grafica del procedimiento de pruebas

6 Validación

Para validar todo el procedimiento que comprende las pruebas se mostrará el trabajo realizado en la plataforma que cuenta la empresa BeGo, esta plataforma ya contiene un entorno de desarrollo el cual se puede adaptar para realizar la implementación que conlleva la automatización de pruebas, para esto, es necesario adaptar los nuevos cambios operativos con el fin de que se vea reflejado el antes y el después de la implementación del proceso de pruebas.

6.1 Configuración de las herramientas

Para materializar la comprobación del procedimiento planteado para pruebas sobre el entorno de trabajo de la empresa BeGo se realizó la instalación de las herramientas descritas en el capítulo 5.1 donde se describe que utilidades y versiones son necesarias.

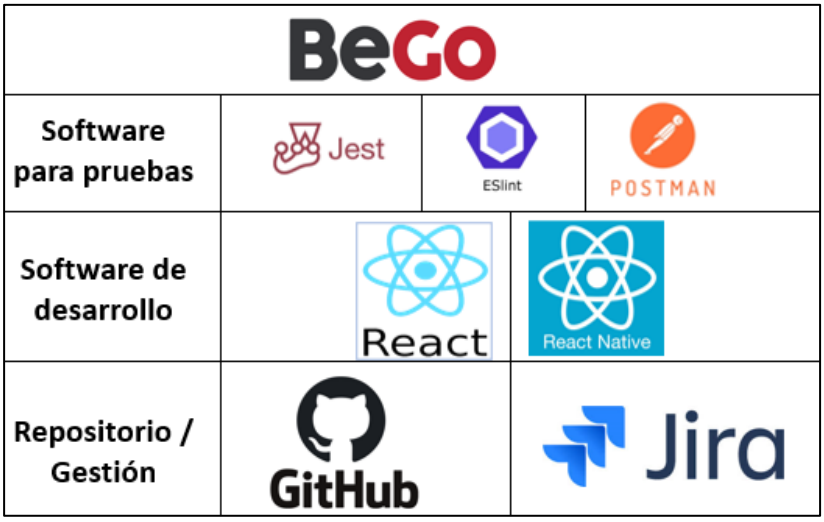
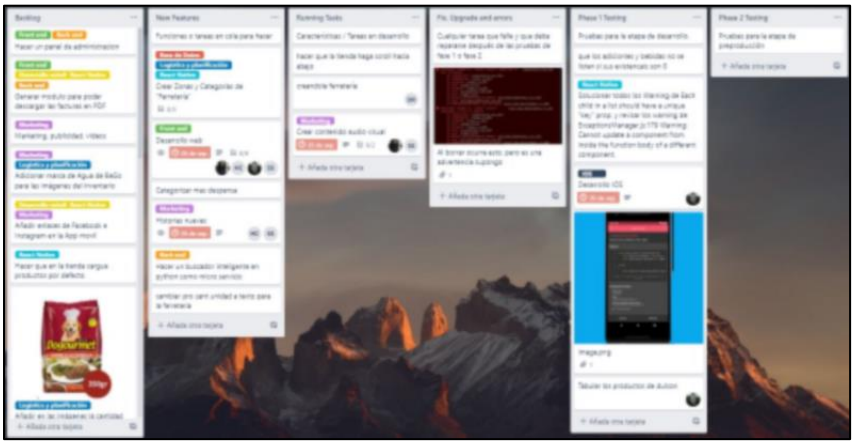


Ilustración 24 Herramientas utilizadas

6.2 Gestión de proyectos

Antes

La empresa de desarrollo BeGo abordaba los requerimientos de desarrollo por medio de la asignación de tareas y responsabilidades descritas de manera informal en donde se especificaba de manera verbal los comportamientos que se debían de esperar para cumplir con las exigencias propuestas.



Ahora

En la actualidad se planea y se prioriza las tareas actuales, así como también los proyectos a futuro utilizando la herramienta de Jira, allí se describe a detalle cada tarea asignando su nivel de importancia, clasificándola en los distintos proyectos destinando a las personas involucradas y seccionándolos como corresponda. Estas tareas planificadas se listan en el Backlog del proyecto esperando para ser asignadas y comenzar un sprint para su desarrollo.

Esta herramienta proporciona bastantes beneficios al momento de organizar los proyectos tales como clasificarlos en distintas versiones de lanzamiento para llevar un control estructurado sobre que adiciones y diferencias existen en cada versión.

También permite trabajar bajo una metodología de desarrollo ágil tal como scrum para impulsar el rendimiento del equipo de desarrollo.

6.3 Realización de Test

Antes

Cuando se terminaba la realización de las actividades de desarrollo se iniciaba la primera etapa de realización de pruebas en donde la persona que programo la nueva funcionalidad junto con una persona aparte de ella se debían encargar de realizar pruebas exploratorias tratando de simular con la mayor precisión posible el uso normal que tendría la nueva adición funcional para verificar el correcto funcionamiento o detectar fallas potenciales. Si la ejecución marchaba adecuadamente se procede a pasar a la segunda etapa de pruebas en la que otra persona nuevamente trata de encontrar defectos o validar el funcionamiento, si todo sale bien la funcionalidad entra en espera para desplegarse en producción.

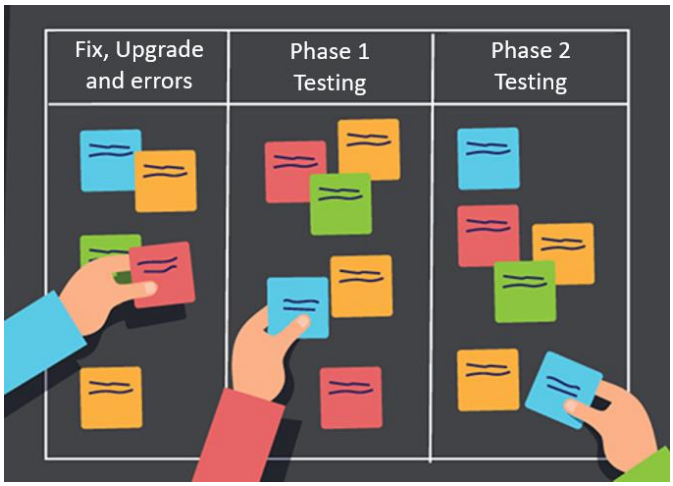


Ilustración 25 Ilustración de manejo de errores

Ahora

- *Se determina que funcionalidades necesitan realización de pruebas*

Al momento que se planea el desarrollo de una nueva adquisición funcional se planea también el desarrollo de su prueba, es decir, a la persona se le asignan dos tareas, una el desarrollo planteado y dos, la escritura de pruebas que valide que el desarrollo se ejecuta sin inconvenientes.

- *Determinación del tipo de prueba*

Se determina si las nuevas adiciones funcionales requieren únicamente de pruebas funcionales aplicadas a pequeños módulos, si no es así, se planea la creación de una prueba que interactúe con diferentes componentes para realizar una prueba de integración y que abarque más contenido dentro del sistema.

Del mismo modo se determina si vale la pena invertir la cantidad de tiempo necesaria para automatizar una prueba que simule la interacción con la interfaz.

- *Decisión sobre la estrategia de pruebas*

Se decide si los cambios añadidos necesitan de realizar pruebas de regresión o exploratorias para determinar si todo sigue funcionando de manera correcta.

- *Definición de los criterios de aceptación*

Decisión sobre la madurez con la que se puede lanzar el sistema a producción teniendo en cuenta los resultados de las pruebas y que tanto abarcan sobre el sistema

- *Determinación sobre el entorno requerido*

Se especifica si se requiere de alguna condición necesaria como el establecimiento de base de datos, servicios externos requeridos, plataformas, sistemas operativos o navegadores web

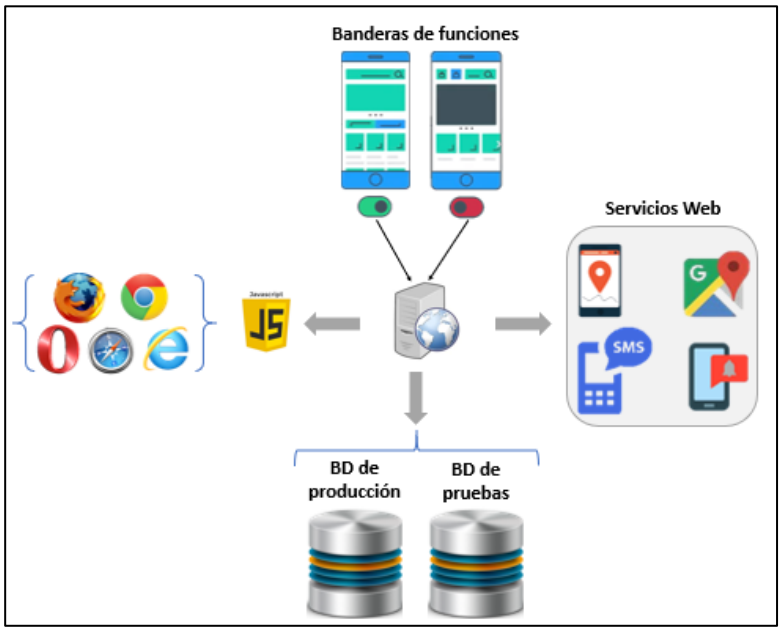


Ilustración 26 Entornos de despliegue del sistema

En el grafico anterior se muestran partes del entorno necesario para la ejecución de pruebas sin afectar el ambiente de producción. Se cuenta con una característica conocida como feature flag que permite un cambio de ambiente con respecto a la base de datos y los servicios web externos que se usan, así se pueden realizar pruebas con total tranquilidad sin temor a causar daños en la base de datos real.

Ilustración 27 Pruebas internas en dispositivos móviles

También es posible gracias a las consolas de desarrollo con las que cuenta la empresa de desarrollo lanzar versiones de los aplicativos a un grupo de personas reducidas con el fin de que se realicen Test a una muestra pequeña de la población para poder así determinar si la aplicación esta lista para desplegarse a todos los usuarios o requiere algún tipo de corrección.

- *Definición del plan de pruebas*

Se determina que personas son las que más conviene que realicen las pruebas asignando sus respectivas responsabilidades

- *Ejecución de las pruebas automatizadas*

Al ejecutar las pruebas desde la terminal de proyecto y esperado el tiempo de respuesta los resultados que se muestran son los siguientes

```
> jest --coverage

 PASS  __test__/src/redux/actions/actionsCart.test.tsx (47.134 s)
 PASS  __test__/src/assets/constants.test.ts (49.042 s)
 PASS  __test__/src/redux/actions/actionsProducts.test.tsx (0.226 s)
 PASS  __test__/src/redux/reducers/reducerProducts.test.tsx (0.226 s)
 PASS  __test__/src/components/views/ContainerMetodosDePago.test.tsx (6.226 s)
 PASS  __test__/src/components/utils/UtilsButtomAgregar.test.tsx (61.111 s)

Test Suites: 19 passed, 19 total
Tests:       78 passed, 78 total
Snapshots:   13 passed, 13 total
Time:        142.736 s
Ran all test suites.
```

Ilustración 28 Resultado por consola de las pruebas

En donde se muestra información sobre algunos de los scripts de test que se corrieron y su tiempo de ejecución, además también muestra el número de pruebas que pasaron correctamente y también mostraría los casos de pruebas fallidos en caso de existir

También puede mostrar el resultado de que tantas líneas de código se tienen cubiertas con las escrituras de pruebas y demás información que puede llegar a ser útil como estadísticas para entender el estado de las pruebas.

File		Statements		Branches		Functions		Lines	
assets		53.77%	57/106	22.55%	23/102	53.85%	7/13	53.77%	57/106
components/cards		100%	1/1	100%	0/0	100%	1/1	100%	1/1
components/containers		100%	3/3	100%	0/0	100%	2/2	100%	3/3
components/utils		85%	17/20	75%	9/12	83.33%	5/6	84.21%	16/19
components/views		100%	1/1	100%	0/0	100%	1/1	100%	1/1
services		45%	4/10	0%	0/7	0%	0/5	45%	4/10

Ilustración 29 cobertura de código

- *Subiendo los cambios de solución al repositorio*

Por último, después de haber solucionado los bugs es necesario subirlos al repositorio de control de versiones, es recomendable hacer un comentario donde se especifique que se hizo

y adicionar el identificador para que así la tarea se marque como realizada de manera automática gracias a los comentarios inteligentes que proporcionan Git y Jira.

En la siguiente ilustración se muestra una captura de la información presente en Jira sobre la solución de un error

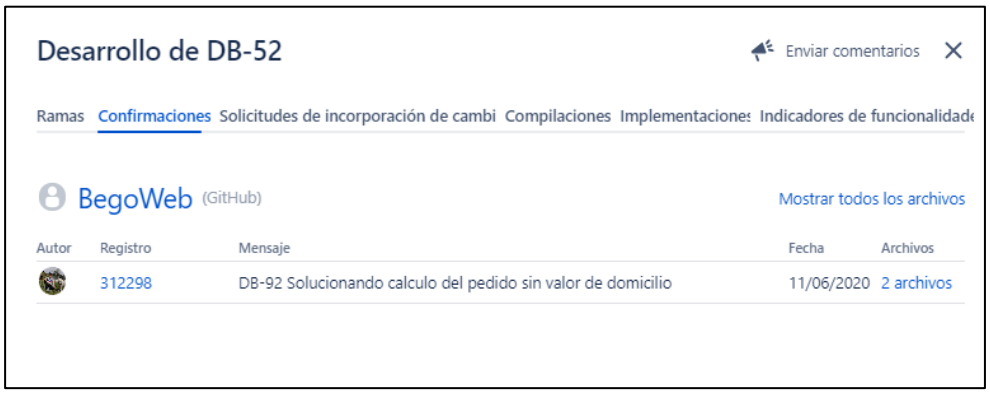


Ilustración 30 Representación de una tarea de solución de error terminado

6.4 Al encontrar bugs

Antes

Si durante la fase de pruebas alguna adición funcional presentaba alguna falla o defecto, se debía regresar a la fase de solución de problemas y se especifica mediante una evidencia el por qué ha fallado para que comience de nuevo el ciclo de corrección y pruebas.

Ahora

Si durante los ciclos de pruebas se llega a encontrar algún error se procede a crear un nuevo Issue en la plataforma de Jira, en donde se debe especificar con el mayor detalle posible cual fue el error encontrado para proceder a asignarse a un caso para su solución.

6.5 Herramientas de desarrollo

Antes

La empresa contaba con un aplicativo móvil construido en el entorno de desarrollo de Apache Cordova el cual presenta incompatibilidad con las distintas versiones de IOs y Android haciendo difícil la tarea de lanzar la aplicación a los diversos dispositivos que se encuentran en el mercado, además de la página web la cual se volvía cada vez más compleja de mantener y modificar.

Ahora

Se decidió migrar la plataforma web, así como la móvil a un framework el cual permitiera facilidad a la hora de desarrollar, en este caso se vio React como el más ventajoso además que permitía integrarse fácilmente con varias utilidades que necesitaba la empresa de desarrollo.

7 Conclusiones

- ✓ Teniendo en cuenta todos los aspectos relevantes que se lograron investigar con respecto a las pruebas de software se logró la identificación de un procedimiento el cual permitiera adaptarse a los procesos operativos de la empresa BeGo para implementar pruebas automatizadas.
- ✓ Al implementar el uso de una metodología ágil y una herramienta de gestión de proyectos y de pruebas como Jira se observó una mejora considerable con respecto a la organización y planeación de los proyectos con los que cuenta la empresa.
- ✓ Se logró validar el procedimiento de pruebas en la nueva implementación de React mostrando una mejora con respecto con la plataforma con la que se contaba anteriormente.
- ✓ Gracias al estudio realizado se logró determinar que aplicar el proceso de pruebas es una metodología importante que deben implementar todas las empresas sin importar que tan grandes sean ya que proporcionan aseguramiento de la calidad y mayor control sobre el sistema.
- ✓ Se logró la adopción adecuada de las utilidades, configuración y puesta en marcha de la librería de Jest y PostMan las cuales permitieron realizar las validaciones de funcionamiento del aplicativo.
- ✓ Se ve claramente que desarrollar un procedimiento para la gestión de pruebas de software es totalmente factible ya que proporciona al equipo de desarrollo un

estándar para saber que hacer frente al descubrimiento de un bug o error y como abordarlo para solucionarlo.

8 Recomendaciones y trabajos futuros

- Se recomienda incentivar a un mayor estudio con respecto a las pruebas de software ya que es un factor indispensable en las empresas que se encargan de desarrollar software.
- Como trabajo futuro se propone realizar un estudio para la ejecución de un proceso de pruebas no funcionales que se adapten a pequeñas y medianas empresas.
- Diseñar una arquitectura y planeación organizacional para empresas que determine recomendaciones que permita realizar pruebas automatizadas, integración continua, entrega continua, despliegue continuo y manejo de repositorios de control versiones de manera óptima.
- Creación de un estándar para planear en que situaciones dentro del desarrollo de sistemas conviene la realización de escritura de pruebas para interfaces

9 Bibliografía.

Pruebas funcionales / No funcionales: ¿Qué son y para qué sirven? (2019). Tester-H. https://testerhouse.com/teoria-testing/pruebas-funcionales/

Turrado, J. (2020). Qué son las pruebas de software | campusMVP.es. Campusmvp. https://www.campusmvp.es/recursos/post/que-son-las-pruebas-de-software.aspx

B. Bruegge. (2002). Ingeniería de software orientada a objetos, ISBN 970-260010-3

redhat. (s.f.). redhat. Obtenido. https://www.redhat.com/es/topics/open-source/what-is-open-source

Albert Campillo. (2020, January 27). *¿Qué es React y para qué sirve?* https://www.drauta.com/que-es-react-y-para-que-sirve

Alexander Guevara Benites. (2018). *¿Qué es npm?* https://devcode.la/blog/que-es-npm/

Anaraya Albornoz. (2020, October 8). *Planificación de proyecto con Asana*. https://www.appvizer.es/revista/organizacion-planificacion/gestion-proyectos/asana-espanol-planificacion-de-un-proyecto#-que-es-asana

Cecilia García García. (2019, May 3). *Costos de la automatización de pruebas*. https://www.avantica.com/es/blog/costos-de-la-automatización-de-pruebas

Jash Unadka. (2019, November 23). *Javascript Testing Frameworks Browser Stack*. https://www.browserstack.com/guide/top-javascript-testing-frameworks

Javier Guillot. (2019, October 6). *Uso de Trello como herramienta para la gestión de tareas | by Equipo de Innovación Pública (EiP) | Medium*. https://medium.com/@eipdnp/uso-de-trello-como-herramienta-para-la-gestión-de-tareas-4138ec8153e2

Jira. (2020, June 23). *Trabajar con flujos de trabajo | Administración de aplicaciones de Jira Data Center and Server 8.13 | Documentación de Atlassian*. https://confluence.atlassian.com/adminjiraserver/working-with-workflows-938847362.html

Jorge Turrado. (2019, July 3). *Integración continua: qué es y por qué deberías aprender a utilizarla cuanto antes | campusMVP.es*. https://www.campusmvp.es/recursos/post/integracion-continua-que-es-y-por-que-deberias-aprender-a-utilizarla-cuanto-antes.aspx

Jose M Baquero García. (2020, May 27). *Vue.js: Qué es y por qué usarlo como framework de referencia* -. https://www.arsys.es/blog/vuejs/

Juan Amat. (2020, May 30). *React vs Angular vs Vue.js para JavaScript en 2020 | Talent Republic*. https://www.talent-republic.tv/developer/desarrollo-de-software/react-vs-angular-vs-vue-js-para-javascript-en-2020/

Lluna, E. (2011). *Análisis estático de código en el ciclo de desarrollo de software de seguridad crítica* (Vol. 7, Issue 3).

Luismi Gracia. (2020, January 15). *React vs Angular vs Vue – Un poco de Java y +*. https://unpocodejava.com/2020/01/15/react-vs-angular-vs-vue/

Matthew Garcia. (2016, May 24). *Introduction to Testing With Jest | DigitalOcean*. https://www.digitalocean.com/community/tutorials/testing-jest-intro

Mercedes. (2017, September 11). *Desarrollo web - OpenClassrooms ES*. https://blog.openclassrooms.com/es/2017/09/11/que-es-el-desarrollo-web/

Montenegro, R. C. (2017). Testing y calidad de software. automatización de pruebas con selenium webdriver. *Cadime*. http://oa.upm.es/49320/1/PFC_RAFAEL_CUBAS_MONTENEGRO.pdf

Paz, J. M. (2016). Análisis del proceso de pruebas de calidad de software. *Ingeniería Solidaria*, *12*(20), 163–176. https://revistas.ucc.edu.co/index.php/in/article/view/1482/1724

PMOinformatica. (2016, January 18). *Pasos para elaborar el plan de pruebas - La Oficina de Proyectos de Informática*. http://www.pmoinformatica.com/2016/01/elaborar-plan-pruebas-software.html

POR JAVIER GOBEA. (2018). *19 herramientas de gestión de proyectos para emprendedores digitales*. https://hormigasenlanube.com/herramientas-de-gestion-de-proyectos/

QUALITY DEVS. (2019, September 16). *¿Qué es Angular y para qué sirve?* https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/

Quesada-López, C., & Jenkins, M. (2018). Factores asociados a prácticas de desarrollo y pruebas de software en Costa Rica: Un estudio exploratorio. *Avances En Ingenieria de Software a Nivel Iberoamericano, CIbSE 2018*, 149–162.

Sanguino, D. A. A. (2016). *Procedimiento para la gestión de pruebas funcionales de software web en ambientes de desarrollo colaborativo y distribuido.* Universidad de Pamplona.

YANA GUSTI. (2019, July 8). *JIRA moderno caso de prueba y herramienta de gestión de*

proyectos. https://geteasyqa.com/es/blog/jira-test-case-project-management-tool/