2023

Análisis de Técnicas de Testing Aplicadas en Metodologías Ágiles

Trabajo Final Integrador

Director:

- Dra. Gutiérrez, M. de los Milagros

Ing. Perez Matías Emmanuel
ESPECIALIZACIÓN EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Agradecimientos:

Quiero expresar mi más sincero agradecimiento a mi directora de tesis, la Dra Gutiérrez, M. de los Milagros por su guía en la elaboración de este trabajo. Su apoyo y consejos han sido fundamentales para alcanzar esta meta. También quisiera expresar mi gratitud a la Dra. Luciana Ballejos cuyo carisma y predisposición hicieron que mi experiencia universitaria fuera sumamente positiva.

Asimismo, agradezco a mi familia y amigos por su apoyo y comprensión en los momentos difíciles.

Finalmente, agradecer infinitamente a Andrea M. mi compañera de vida, cuyos ánimos me impulsaron a mantenerme dentro del camino.

Matías E. Perez

Este trabajo ha sido desarrollado bajo el marco del proyecto

UTN – FRSF, código: SIUTIFE0007757TC

Resumen

Las metodologías ágiles han revolucionado la forma en la que el software es desarrollado, de igual forma han repercutido en la manera en que el software es probado. Asegurar la calidad nunca ha sido una tarea fácil debido a la gran diversidad de atributos que están puestos en juego, sin embargo, podría decirse que al trabajar con las metodologías llamadas "ágiles" dicha tarea resulta aún más compleja.

Es importante destacar que una versión inicial de este trabajo fue presentada y publicada en CONAIISI 2020, lo que representó una valiosa oportunidad para compartir y discutir ideas con la comunidad académica y profesional. A partir de esa experiencia, he tenido la oportunidad de enriquecer este trabajo y profundizar en el análisis y desarrollo del mismo.

El presente trabajo tiene como objetivo hacer una revisión del estado del arte de las técnicas de testing que son aplicadas en proyectos guiados bajo metodologías ágiles, con el fin de recopilar dichas técnicas y analizar su aplicación en la industria. Con esta finalidad se ha realizado una búsqueda sistemática de bibliografía utilizando la metodología propuesta por Medina-López y col. (2010), la cual consta de cinco etapas: (1) Identificación del campo de estudio y período a analizar, (2) Selección de las fuentes de información, (3) Realización de la búsqueda, (4) Gestión y depuración de los resultados y (5) Análisis de los resultados.

Finalmente, se concluye que en el contexto de las metodologías ágiles es fundamental automatizar los casos de prueba para detectar errores tempranamente a través del proceso de integración continua. No obstante, dicha automatización puede generar un problema a medida que avanza el proyecto debido a que al desarrollar más casos de prueba automáticos también aumenta el tiempo de ejecución de los mismos, por lo que dependerá de la experiencia del equipo de calidad saber ejecutar correctamente y con criterio aquellas técnicas de testing que permitan priorizarlos. Por último, además del empleo de herramientas comerciales o ad-hoc (según las particularidades del proyecto-empresa) es importante no perder de vista que al trabajar en un contexto multidisciplinario, los recursos humanos y las relaciones interpersonales que generan estos, asumen un rol igual de importante en el desarrollo del proyecto, por ello es determinante el disponer y cultivar habilidades blandas como la comunicación efectiva, la coordinación y el trabajo en equipo.

Palabras claves:

Técnicas de testing, metodologías ágiles, caja negra, caja blanca, pruebas de regresión, integración continua.

Tabla de contenido

Índice de Figuras

Índice de Tablas

CAPITULO 1

Introducción

Introducción

En la actualidad, el software se ha convertido en un componente esencial de nuestra vida diaria, gracias a su capacidad para simplificar la realización de tareas cotidianas y brindar acceso a una amplia variedad de servicios y productos a través de plataformas digitales, es por ello que se puede decir que el software cumple un doble rol, ya que no solo es un producto en sí mismo, sino que también actúa como un medio para proporcionar otros productos o servicios. No obstante, el aumento de la complejidad del software y la diversidad de los dispositivos y sistemas en los que se utiliza han dado lugar a nuevas problemáticas en el proceso de desarrollo y en las metodologías de prueba utilizadas para validar y verificar su funcionamiento.

La aplicación de técnicas de testing es una práctica fundamental en el desarrollo de software, ya que permite validar su correcto funcionamiento y asegurar su calidad. En las últimas décadas, el surgimiento de metodologías ágiles ha revolucionado la forma en que se desarrolla el software, y ha implicado cambios significativos en el proceso de testing. En este sentido, la literatura científica ha abordado diversas técnicas de testing que se adaptan a las metodologías ágiles, con el objetivo de mejorar la eficiencia y eficacia en la validación del software y es el objetivo de este trabajo recopilar dichas técnicas de testing para analizarlas y así lograr identificar sus principales características, ventajas y desventajas que estas conllevan.

El presente trabajo se estructura en cinco capítulos, cuyo contenido se encuentra organizado de la siguiente manera. El primer capítulo contiene la fundamentación y los objetivos que guían la presente investigación. En el segundo capítulo, se presenta el marco teórico que sustenta el desarrollo de la investigación. El tercer capítulo aborda la metodología de relevamiento utilizada en el estudio. En el cuarto capítulo, se realiza un análisis tanto del conjunto de trabajos seleccionados, como del contenido específico de cada uno, relacionándolos con otros trabajos del mismo conjunto y sintetizando los aspectos más relevantes de cada uno de ellos. Finalmente, en el quinto capítulo se presentan las conclusiones del trabajo, destacando los principales hallazgos y aportes del estudio realizado.

Fundamentación

Al hablar de calidad, debemos entender cuál es el significado que se le da a dicha categoría en el contexto del software y aquí es donde algunos autores han realizado aportes en esta dirección. Pressman (2010) define a la calidad como la “concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente”, fomentando así el uso de procesos estandarizados e intentando minimizar el fracaso del proyecto. Por otro lado, Albin (2003) la define como “una característica directamente relacionada con la habilidad del sistema para satisfacer sus requerimientos funcionales y no funcionales tanto implícitos como explícitos”, siendo ésta una definición bastante similar a la interpretada por ISO (2022) donde define a la calidad del producto software como el “grado en que dicho producto satisface los requisitos de sus usuarios aportando de esta manera un valor”, haciendo omisión al proceso por el cual el software fue construido. Sin embargo, éste aspecto es considerado en la definición proporcionada por IEEE Computer Society (1998) ya que enuncia que: “La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario.” Como se observa, todas las definiciones coinciden en evaluar la calidad del software en función del estudio de los atributos asociados a los requerimientos del sistema, motivo por el cual, es importante tenerlos presentes a todos ellos para que el testing realizado permita evaluar la calidad del software de manera integral.

Sin embargo, como se enuncia en Dijkstra (1969) las pruebas de software pueden ser una manera muy eficaz de mostrar la presencia de errores, pero son totalmente inadecuadas para mostrar su ausencia. Partiendo de esta base, ninguna pieza de software puede considerarse libre de errores y sumado a la dinámica de la industria que se encuentra orientada a desarrollar nuevas funcionalidades en sus aplicaciones (con el fin de mantener o atraer a clientes) en lugar de resolverlos, induce a pensar que el potencial desastre es claro, de hecho, tal es el punto de compromiso por parte de las compañías desarrolladoras de software que muchas veces se prefiere distribuir el software y luego lanzar parches de seguridad o actualizaciones. Por último para ilustrar la tendencia de la industria hacia este tema, algunos softwares evaden la responsabilidad por la calidad y las consecuencias de la ejecución del software al colocar clausulas específicas al respecto en los términos y condiciones de uso.

A pesar de lo mencionado anteriormente, es importante destacar como el compromiso por la calidad del software ha cobrado mayor importancia en los últimos años. Como resultado, han surgido nuevas herramientas, metodologías, roles dentro del proceso de desarrollo y técnicas de testing en general que buscan minimizar los problemas mencionados.

Es interesante observar como las prácticas de ingeniería se han ido incorporando al proceso de desarrollo de software. Aunque inicialmente estas prácticas no se desarrollaron específicamente para este campo, sino para proyectos de ingeniería civil y mecánica (por ejemplo), con el tiempo se han ido incorporando y adaptando al proceso de desarrollo de software para mejorar la calidad del mismo. Gracias al análisis de proyectos fallidos y a la innovación constante, el proceso de desarrollo de software ha ido cambiando y evolucionando, lo que ha llevado a la creación de nuevas metodologías y enfoques para lograr un software de mayor calidad y eficacia.

Se puede definir a un *proceso de desarrollo de software* como el conjunto estructurado de actividades que son requeridas para realizar un sistema de software (Somerville, 2005). Estas actividades comúnmente se agrupan en etapas que usualmente reciben por nombre: especificación de requerimientos, diseño de la solución, codificación, validación (pruebas) y mantenimiento. Cada una de estas etapas persigue un objetivo particular que se podría generalizar de la siguiente forma:

- Especificación de requerimientos: Se trabaja con el cliente y usuarios finales del sistema (si es posible) en una descripción completa del comportamiento del sistema que se va a desarrollar.

- Diseño de la solución: Se establece una arquitectura completa del sistema, se identifican los subsistemas que componen al sistema y se describe su funcionamiento y relaciones entre ellos.

- Codificación: se codifica y prueban los subsistemas identificados asegurándose lo mejor posible que el código haga lo que debería hacer.

- Validación: se realizan pruebas al sistema completo para validar que todos los requerimientos del software se cumplan.

- Mantenimiento: El sistema se instala y se pone en funcionamiento, además se corrigen errores no descubiertos en las etapas anteriores.

Las características relacionadas con la secuencia, la frecuencia, la duración y el orden en que se realizan las distintas etapas del proceso de desarrollo de software dependen de como se desee agrupar este conjunto estructurado de actividades, según los objetivos que se persiguen y el ámbito en el cual cada una

de estas es llevada a cabo. Estas diferentes agrupaciones y perspectivas particulares son expresadas en diversas abstracciones y cada una recibe el nombre de *modelo de desarrollo de software*.

Estos modelos no son descripciones definitivas del proceso de desarrollo del software, son justamente abstracciones y puede pensarse en ellos como un marco de trabajo, el cual puede ser modificado y adaptado para crear un proceso más específico de ingeniería de software que pueda ser efectivamente llevado a cabo por las personas que intervienen.

Existe una gran variedad de modelos de desarrollo de software y los libros de ingeniería de software comúnmente describen aquellos que los autores consideran más importantes, el problema es que estas listas son bastantes disimiles ya que se encuentran sesgadas por la perspectiva de cada autor. Siguiendo a Pressman (2010), se puede realizar una agrupación de esta gran variedad de modelos en i) modelos de proceso prescriptivo, ii) modelos de proceso especializado y iii) modelo de proceso unificado.

i) En los modelos de proceso prescriptivo predomina el orden y la consistencia del proyecto. El autor los llama "prescriptivos" porque prescriben un conjunto de elementos del proceso como por ejemplo las actividades, tareas, aseguramiento de la calidad y los mecanismos de control para cada etapa, además la manera en la que los elementos del proceso se relacionan entre sí.

ii) Modelos de procesos especializados. Son muy similares a la categoría anterior, sólo que estos se aplican en contextos específicos donde la especialización es la clave.

iii) Modelo de proceso unificado: Representa un intento de obtener las mejores características de varios modelos, es un proceso genérico que cada empresa deberá adaptar al contexto y necesidades de la organización.

Si bien hasta ahora se ha hablado de las llamadas metodologías tradicionales, recordemos que estas surgieron para ordenar el proceso de desarrollo de software y poder entregar software confiable, en el tiempo y presupuesto estipulado. Según Sharmaa, Sharmaa, & Gujrala (2015), el software está siendo utilizado a un ritmo cada vez mayor en una amplia variedad de actividades humanas. En consecuencia, la necesidad de reducir el tiempo de entrega y garantizar la calidad del software durante su proceso de desarrollo se convierte en una tarea crítica.

En este contexto aparecen las metodologías ágiles aportando un proceso flexible y ágil que en ambientes como el descripto anteriormente resultan más efectivas que las metodologías tradicionales. En su manifiesto emitido en febrero de 2001, se estipula las características que debe tener un desarrollo ágil y que a pesar de que ciertos atributos son valorados, hay otros, que según los autores, se deben valorar aún más (Beck, y otros, 2001).

Sin embargo, Crispin & Gregory (2014) mencionan que estas nuevas metodologías, impactaron enormemente en las tareas de aseguramiento de la calidad ya que bajo metodologías tradicionales, los ciclos de desarrollo eran generalmente largos y los encargados de probar el software estudiaban los documentos de requerimientos para escribir sus planes de pruebas y luego esperaban a que se les entregue el código para comenzar a probarlo. Los autores remarcan que la transición a iteraciones cortas produjo inicialmente, shock y temor. Los procesos tradicionales de testing ya no cabían dentro del plazo de una iteración y esta práctica también tuvo que ser pensada nuevamente y redefinida. Es allí donde se acuña el concepto de ***agile testing***.

En Razak & Fahrurazi (2011) se define al *agile testing* como una práctica de testing de software que sigue los principios de las metodologías ágiles y que involucra el testing desde la perspectiva del cliente lo más temprano posible, testeando a medida que el código esté disponible y sea lo suficientemente estable. No obstante, Crispin & Gregory (2014) señalan que la práctica de agile testing, puede descuidar otros tipos

de testing tales como: testing de performance, de seguridad, de usabilidad, entre otros. Consideran que estos tipos de pruebas son igualmente importantes en un proyecto ágil, como también lo son en otros proyectos que utilizan otras metodologías de desarrollo. Sin embargo, la dinámica de interacciones cortas dificulta el diseño y ejecución de las pruebas en general y es aquí donde toma relevancia la experiencia del responsable de aseguramiento de la calidad, ya que el hecho de conocer más técnicas de pruebas para aplicarlas al producto, posibilita poder elegirlas adecuadas según que es lo que se desee probar y cual es tiempo disponible.

Afortunadamente, la evolución constante del desarrollo de software conduce a la aparición de nuevas herramientas y metodologías, lo que a su vez genera la necesidad de desarrollar nuevas técnicas de testing de software para asegurar la calidad del producto final. En este contexto, resulta fundamental estar al tanto de los avances en el área de la tecnología y el testing de software para poder aplicarlas en el proceso de pruebas y garantizar la calidad del producto final. La constante aparición de nuevas técnicas de testing de software también implica la necesidad de actualizar los conocimientos y habilidades de los profesionales encargados del testing, para poder adaptarse a los cambios y continuar mejorando los procesos de aseguramiento de la calidad.

Objetivos

General

Evaluar la aplicabilidad en el desarrollo de software guiado por metodologías ágiles de las técnicas de testing ampliamente referenciadas en diversas fuentes bibliográficas para identificar aquellas que faciliten una mejora sustancial en la confiabilidad del software en proceso de validación, contribuyendo así al proceso de aseguramiento de la calidad.

Específicos

Para cumplir con el objetivo general, se proponen los siguientes objetivos particulares:

a) Realizar una búsqueda sistemática de bibliografía en relación al campo de estudio a investigar.
b) Depurar la bibliografía encontrada, de acuerdo a criterios prestablecidos con el fin de trabajar sólo con producciones de interés.
c) Recopilar técnicas de testing utilizadas en la actualidad en proyectos de desarrollos de software guiados con metodologías ágiles.
d) Evaluar la implementación en la industria de las técnicas de testing recopiladas mediante la identificación de sus principales características y las ventajas y desventajas que estas conllevan.

CAPITULO 2

Marco Teórico

Marco teórico

Calidad de software

Si bien uno de los aspectos fundamentales en el desarrollo del software es la calidad, resulta interesante el hecho de que esta palabra pueda ser definida de múltiples maneras y abarcar diferentes conceptos. Esta situación produce que no exista una definición clara sobre que es lo que se entiende cuando hablamos de "calidad del software" y por ende las métricas planteadas para dimensionarla inevitablemente están sesgadas por el punto o la perspectiva de interés en ese momento. Adicionalmente, la calidad del software posee la particularidad de ser subjetiva, ya que depende del conjunto de atributos elegidos para medirla y además es circunstancial, ya que estos atributos pueden ser reemplazados por otros según el contexto.

Sin ir más lejos, existen dos interrogantes (difíciles de responder) estrechamente relacionados, que en la mayoría de las veces se encuentran presentes al momento de pensar sobre la calidad del software, ellos son:

1. ¿Cómo obtener un software de calidad?
2. ¿Cómo evaluar si efectivamente un software dado posee calidad?

Si bien pareciera no haber una definición única, en líneas generales la calidad del software debería ser el cumplimiento y/o grado de satisfacción en relación a determinados requerimientos y/o necesidades que pueden estar explícitamente mencionados en distintos documentos o ser implícitos, basados en la idea de como debería funcionar la pieza de software. Sin una definición clara, concisa y medible de que se entiende por un software con calidad, es imposible realizar evaluaciones sobre que áreas mejorar, que herramientas y técnicas utilizar para mejorar la calidad e incluso puede guiarnos a pensar (erróneamente) que es algo de lo que debemos preocuparnos una vez que se ha generado el código.

Para poder gestionar la calidad del software es necesario realizar cierto tipo de mediciones que permitan caracterizar, evaluar, predecir y mejorar lo que se está midiendo. Para llevarlas a cabo, es necesario definir parámetros, indicadores o criterios de evaluación que proporcionen la información necesaria sobre la calidad que queremos administrar.

En lo que respecta a indicadores, en ISO/IEC 25010 (2022) se presenta el siguiente listado de características de calidad del software, necesarios a la hora de evaluar la misma:

> **Adecuación funcional**
>
> Representa la capacidad del producto software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificadas (…).
>
> **Eficiencia de desempeño**
>
> Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones (…).
>
> **Compatibilidad**
>
> Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno de hardware o software (…).

Usabilidad
Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones (…).

Fiabilidad
Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados (…).

Seguridad
Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos (…).

Mantenibilidad
Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas (…)

Portabilidad
Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, software, operacional o de utilización a otro (…)
(ISO/IEC 25010, 2022)

Dentro de cada grupo, se mencionan subcaracterísticas que especifican en mayor detalle las características principales. Dichas subcaracterísticas se muestran en la **Figura 1.**

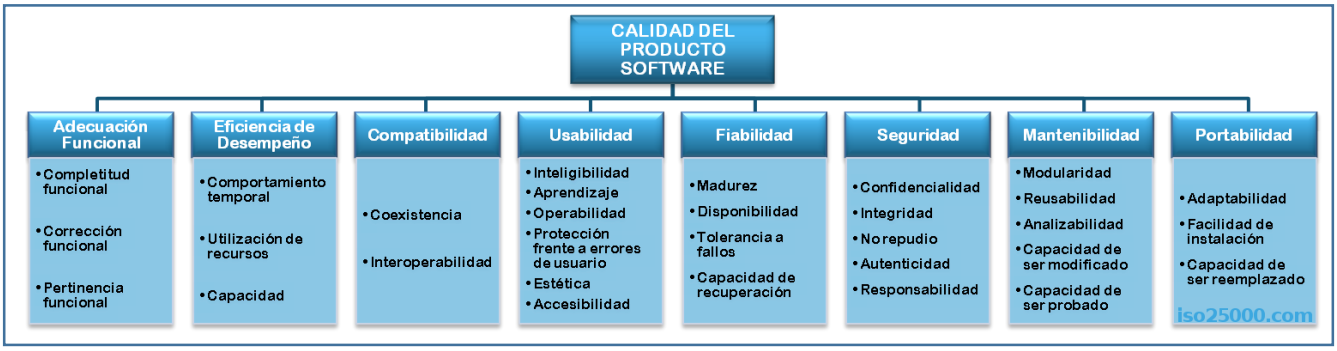

Figura 1. Características de Calidad según ISO/IEC 25010

Para gestionar estas características, existen diferentes modelos de calidad y estándares que proporcionan lineamientos para la correcta administración de la calidad. Por otro lado, el aseguramiento de la calidad del software y por ende el cumplimiento de las características de este, pueden ser analizados (y si amerita, corregidos) a través de la ejecución de diferentes pruebas de software.

Pruebas de Software

En las siguientes secciones se presentarán los diferentes tipos, técnicas y estrategias de testing que pueden ser empleadas según el atributo que se desee evaluar y el nivel en donde se ejecuten. Sin embargo es importante tener en cuenta que no importa cuanto tiempo y dinero se destine a ejecutar pruebas en los diferentes niveles, jamás va a ser posible detectar todos los errores de un software.

En este punto, es oportuno aclarar la diferencia entre *error*, *defecto* y *fallo*. Esta terminología se precisa claramente en el estándar IEEE 610.12-1990, donde se define un *error* como una acción humana que produce un resultado incorrecto, un *defecto* como un desperfecto en un componente o sistema que puede causar que el componente o sistema falle en desempeñar sus funciones y finalmente define al *fallo* como la manifestación física o funcional de un defecto (IEEE Computer Society, 1990). Es decir, un error introduce un defecto y este causa un fallo. Las pruebas detectan fallos y existen diferentes tipos de pruebas para detectar diferentes tipos de fallos, pero es el error el que se debe rastrear y eliminar.

Tipos de Pruebas

Existen diferentes tipos, algunas se concentran en probar una funcionalidad específica como son las pruebas funcionales, otras en probar características no funcionales y por último, están aquellas enfocadas en probar la estructura del software bajo análisis. A continuación, veamos algunos detalles.

Pruebas Funcionales

Se entiende como funcionalidad de un sistema, a lo que el sistema hace. Las funcionalidades suelen estar escritas en una especificación de requerimientos, en especificaciones funcionales, casos de uso e inclusive pueden hasta no estar documentadas, sin embargo para este último caso, se requiere un nivel de experiencia elevado para realizar este tipo de pruebas en este contexto.

Las pruebas funcionales pueden llevarse a cabo en todos los niveles y suelen estar asociadas a las técnicas de diseño de pruebas de caja negra, ya que evalúan el comportamiento externo del software.

Pruebas No Funcionales

Los requerimientos no funcionales representan “como funciona el sistema” (en contraposición con las funcionalidades que definen “lo que el sistema hace”).

Este tipo de pruebas están asociadas a las técnicas de diseño de pruebas de caja negra y su objetivo es probar los requerimientos no funcionales. Algunas pruebas de este estilo son:

- Pruebas de carga: Consisten en diseñar un patrón de demanda, ejecutarlo y observar como reacciona el sistema. Este tipo de pruebas ayudan a identificar la máxima capacidad operativa de una aplicación, así como también, identificar cuellos de botella y las causas de una degradación del servicio.

- Pruebas de estrés: Consiste en someter al sistema a una carga muy superior a la normal y analizar como se ve afectado, con especial interés en la disponibilidad del sistema y el punto de ruptura donde el sistema colapsa.

- Pruebas de volumen: Consisten en validar el funcionamiento del sistema con ciertos volúmenes de datos. El objetivo aquí es ver si dado ciertos volúmenes de datos la aplicación funciona con normalidad.

- Pruebas de configuración: Consiste en verificar si el sistema es capaz de funcionar correctamente en diferentes versiones o configuraciones de entornos.

- Pruebas de usabilidad: Consiste en evaluar que tan fácil de usar es el sistema. Algunos puntos a tener en cuenta en este tipo de pruebas son la facilidad de aprendizaje, eficiencia en las tareas, entre otros.

- Pruebas de seguridad: Consiste en analizar que tan seguro es un sistema, que información expone y como puede ser vulnerado algún acceso.

- Pruebas de escalabilidad: Consiste en verificar la capacidad de una aplicación de escalar cualquiera de sus características, como por ejemplo: carga que soporta, volúmenes de datos, número de transacciones, entre otros.

- Pruebas de recuperación: Consiste en verificar que tan rápido y de que manera se recupera la aplicación luego de experimentar una falla, ya sea de software o de hardware.

- Pruebas de mantenibilidad: Consiste en evaluar que tan fácil es analizar, cambiar y probar algún cambio una vez que el sistema ya esté en producción y requiera algún tipo de mantenimiento.

Pruebas Estructurales

Pruebas estructurales es el término usado por el El Comité Internacional de Certificaciones de Pruebas de Software (ISTQB por sus siglas en inglés) para las pruebas de “Caja Blanca”.

Se utiliza el concepto de “cobertura” como una medida porcentual, para medir el grado en que la estructura ha sido cubierta por un conjunto de pruebas. Idealmente, este porcentaje de cobertura debería ser 100%.

Técnicas de Pruebas

Uno de los objetivos de las pruebas es detectar el máximo número de errores y muchas técnicas se han desarrollado con tal fin. El principio subyacente de estas técnicas radica en ser lo más sistemático posible identificando conjuntos representativos de comportamientos del software.

Estas técnicas pueden clasificarse en tres grandes grupos, los cuales son: técnicas estáticas, técnicas dinámicas y técnicas basadas en la experiencia. Veamos a continuación, cada uno de ellos.

Técnicas estáticas

Este tipo de pruebas son aquellas que no ejecutan el software bajo análisis pero lo analizan con el objetivo de detectar defectos en el código fuente, aspectos sospechosos del código o del diseño.

El mecanismo comúnmente utilizado en este tipo de pruebas es el mecanismo de revisión, el cual consiste en una reunión entre diferentes personas (que pueden ocupar o no algún rol puntual) y en conjunto analizan la pieza de software con el fin de debatir, tomar decisiones, evaluar alternativas, resolver problemas técnicos, validar si se cumplen las especificaciones y/o estándares de diseño, entre otras cuestiones.

Estas revisiones pueden llamarse de diferentes maneras y tener algunas particularidades según el grado de formalidad que posea, pero comúnmente podemos apreciar diferentes estilos como son: revisión informal, revisión guiada, revisión técnica y por último, inspección.

Técnicas dinámicas

La principal diferencia entre las técnicas estáticas y técnicas dinámicas, es que estas últimas realizan pruebas ejecutando el software. Las pruebas dinámicas detectan fallos, en cambio, las pruebas estáticas detectan sus causas, los defectos.

Dentro de este tipo de técnicas, el objeto de estudio puede ser analizado desde una visión interna (técnicas de caja blanca) o desde una visión externa (técnicas de caja negra).

- Técnicas de caja blanca: También pueden ser llamadas como “caja de cristal”. Este tipo de pruebas garantizan que todas las rutas del código sean revisadas al menos una vez y existen diferentes técnicas para alcanzar dicho fin como por ejemplo: pruebas de la ruta básica (o *happy path*), pruebas de bucles, pruebas de condición y condición múltiple.
- Técnicas de caja negra: También pueden ser llamadas pruebas de comportamiento y son las que sin importar la estructura interna del software, analizan si el software cumple con los requisitos funcionales o no funcionales determinados. Comúnmente los errores que se encuentran son funciones faltantes, errores de interfaz, entre otros.
  Existen varios métodos que pueden aplicarse y/o combinarse a la hora de diseñar una prueba de este estilo, ellos son:

  - Particiones de equivalencia: donde los valores de entrada se dividen en diferentes grupos según un criterio de similaridad, con el fin de que todos los elementos de un mismo grupo sean procesados de la misma forma. Estos datos de entrada incluyen todo tipo de valores, tanto válidos como no válidos.
  - Análisis del valor límite: los valores máximos y mínimos de una partición son sus valores límites por lo que el sistema debe responder de la forma esperada ante estos extremos.
  - Pruebas de tabla decisión: aplicables cuando la lógica a probar está basada en decisiones del tipo *If then else*.
  - Pruebas de transición de estado: a partir de un estado inicial se recorren los otros mediante eventos que desencadenan transiciones de estados. Los diagramas de estados (si existiesen) son una fuente de información muy rica para el diseño de este tipo de pruebas.
  - Pruebas de casos de uso: Tomando como punto de partida un documento de caso de uso, donde se expresa la manera de utilizar el sistema para una meta en particular, se diseñan casos de prueba que esperan todo aquello que está documentado y no esperan

todo aquello que está omitido. Este tipo de pruebas tienen una relación directa con las pruebas de aceptación por parte del cliente/usuario.

Técnicas basadas en la experiencia

Las pruebas basadas en la experiencia son aquellas en las que las pruebas derivan de la habilidad e intuición de la persona que la lleva a cabo, así como también, de la experiencia con aplicaciones y tecnologías similares. Esta técnica puede tener distintos grados de efectividad según quien las lleva a cabo.

Dentro de este grupo, existen dos tipos de pruebas que merecen una mencion, ellas son:

- Pruebas de predicción de error: En este tipo de pruebas se toma como referencia fallos encontrados previamente, errores frecuentes de desarrolladores e incluso errores frecuentes de los desarrolladores que pertenecen al equipo. Toda esta información en su conjunto, sumado a la experiencia de quien diseña los casos de prueba permiten generar escenarios de predicción de error.
- Pruebas exploratorias: Este tipo de pruebas fueron introducidas por Cem Kaner y consiste en un estilo de prueba de software que enfatiza la libertad personal y la responsabilidad de quien las lleva adelante, para optimizar continuamente el valor de su trabajo al tratar el aprendizaje relacionado con la prueba, el diseño, la ejecución y la interpretación de los resultados como actividades de apoyo mutuo, ejecutadas en paralelo a lo largo del proyecto (Kaner, 1983).

Este tipo de pruebas es recomendable a la hora asegurar la calidad de un sistema que ya se encuentra desarrollado y se le desconocen sus módulos e interacciones con otros sistemas. Bajo este enfoque, a medida que se va explorando el sistema y aprendiendo sobre él, nuevo conocimiento es adquirido y por ende nuevos escenarios de prueba pueden ser diseñados y ejecutados. Ideales para aquellas personas que ingresan a un equipo con el sistema en marcha.

Estrategias de Pruebas

Las pruebas del software comúnmente se realizan a diferentes niveles durante los procesos de desarrollo y mantenimiento. Dicho de otra manera, el objeto que se prueba puede variar desde un módulo en particular, un conjunto de módulos o el sistema completo. Además de las pruebas llevadas a cabo en estos niveles específicos, existen otras estrategias que son transversales a más de un nivel, aquí podríamos mencionar a las pruebas de humo, de regresión y de aceptación entre otras. A continuación se describirán brevemente cada una de ellas.

Pruebas Unitarias

Este tipo de pruebas verifican el funcionamiento aislado de una unidad de software. Puede definirse a la unidad como un subprograma individual o un componente más grande formado por unidades muy relacionadas. Suele ser llevado a cabo por los desarrolladores y es recomendable que no sólo realicen pruebas de la estructura del código sino también, que se compruebe el funcionamiento del componente según para lo cual fue diseñado.

Pruebas de Integración

Una prueba de integración es aquella que se realiza para comprobar las interacciones entre distintos componentes o sistemas tras su integración. Si bien las pruebas unitarias son por lo general llevadas a cabo por el desarrollador, este tipo de pruebas son llevadas a cabo por equipos más especializados. El hecho de haber comprobado todos los componentes, no asegura el correcto funcionamiento entre ellos y allí es donde radica la importancia de esta prueba.

Existen diferentes enfoques de integración, los cales pueden ser: *top-down*, *down-top*, *ad hoc*, *bing bang* e *híbridos* y cada uno de ellos difiere en el punto de partida y en la manera en que se va integrando con nuevos componentes.

Estas pruebas por lo general encuentran errores de flujo incorrecto de información entre los componentes, cálculos incorrectos, funcionamientos incorrectos propios de la dinámica del producto, entre otros.

Pruebas de Sistema

Este tipo de pruebas tienen como objetivo validar el comportamiento de un sistema completo. Pese a que la mayoría de los errores deberían haber sido identificados en los niveles de pruebas anteriores, existe la posibilidad de que el sistema no cumpla con (como por ejemplo) requisitos no funcionales.

Estas pruebas por lo general encuentran errores de seguridad, velocidad del sistema, confiabilidad, entre otros.

Pruebas de Humo

Son aquellas en las que se realiza una prueba rápida del programa comprobando que las principales funcionalidades siguen desempeñándose según lo esperado, como así también los procesos básicos. Son pruebas económicas en término de tiempo y comúnmente se las ejecuta luego de realizar integraciones de código en un determinado entorno.

Pruebas de Regresión

Se trata de pruebas especialmente elegidas con el fin de verificar que los cambios introducidos no han generado efectos indeseados en otras partes del sistema. Este tipo de pruebas pueden incluir pruebas funcionales, no funcionales y estructurales. Dado que este conjunto de pruebas se ejecutan repetidas veces, es muy común invertir tiempo en intentar automatizarlas de alguna manera.

Pruebas de Aceptación

Se llevan a cabo por el cliente y puede incluir o no a los desarrolladores del sistema. Estas pruebas se realizan en una instancia anterior a que el sistema se ponga en un ambiente productivo y se usan para comparar el comportamiento del sistema con los requisitos del cliente.

Agile Testing

Principios del agile testing

El agile testing es una práctica de pruebas de software que siguen los principios de desarrollo ágil, involucrando en el proceso de pruebas no sólo al equipo de desarrolladores, sino también, líderes del proyecto y clientes.

En este paradigma, se concibe al proceso de pruebas como una actividad que es realizada "durante" el desarrollo del software y las demás actividades y no como una actividad que es realizada luego de completarse el desarrollo de determinada funcionalidad. Esta nueva dinámica, permite encontrar y/o corregir cualquier error o desviación lo más temprano posible a través de una retroalimentación continua y por lo tanto, es necesaria la participación de todo el equipo.

Laing & Greaves (2015) proponen un manifiesto para el agile testing, que como su nombre lo indica, es una analogía con el manifiesto para el desarrollo ágil, pero se encuentra enfocado al testing. Los valores que ellas destacan son:

- Testing durante por sobre testing al final: Implica la priorización de tareas de testing desde el inicio hasta que se finaliza una funcionalidad por sobre el testing como etapa al finalizar un sprint.
- Prevenir bugs por sobre encontrar bugs: Implica incentivar la revisión grupal de las tareas, tener en claro que hay que hacer, como y por que para lograr así, evitar todo tipo de suposiciones y trabajar realmente en equipo con el desarrollador, para evitar la mayor cantidad de bugs posibles en lugar de esperar, sin ningún tipo de interacción, a que el desarrollo esté completo para revisar si se hizo lo que se debería haber hecho y si se lo hizo bien.
- Entender lo que se está testeando por sobre verificar la funcionalidad: Implica entender la necesidad del usuario, para que quiere la funcionalidad y como él lo va a usar de manera tal de realizar pruebas que agreguen un valor al producto y no simplemente verificar si se cumplen las especificaciones.
- Construir el mejor sistema por sobre romper el sistema: Implica generar un ambiente de trabajo donde el equipo vaya mejorando y aprendiendo de sus errores para así en un futuro no cometerlos, en lugar de simplemente intentar romper la funcionalidad.
- El equipo es responsable de la calidad por sobre el tester es responsable de la calidad: La clave está en que el equipo funcione como tal y se comprometa en todas las actividades necesarias para generar valor, por sobre actitudes individualistas.

Es importante aclarar que, al igual que el manifiesto agile la idea es ponderar una postura por sobre la otra, en lugar de simplemente eliminar una, para ser reemplazada por otra.

Cuadrantes del Agile Testing

Crispin & Gregory (2014) crearon la tabla de cuadrantes del Agile Testing donde se permite identificar los diferentes propósitos y tipos de pruebas que se puedan realizar en un entorno ágil. En la **Figura 2** se muestra dicha tabla donde se observa que en un eje se divide a la matriz en pruebas que apoyan al equipo y pruebas que critican al producto y el otro eje divide en pruebas orientadas al negocio y a la tecnología.

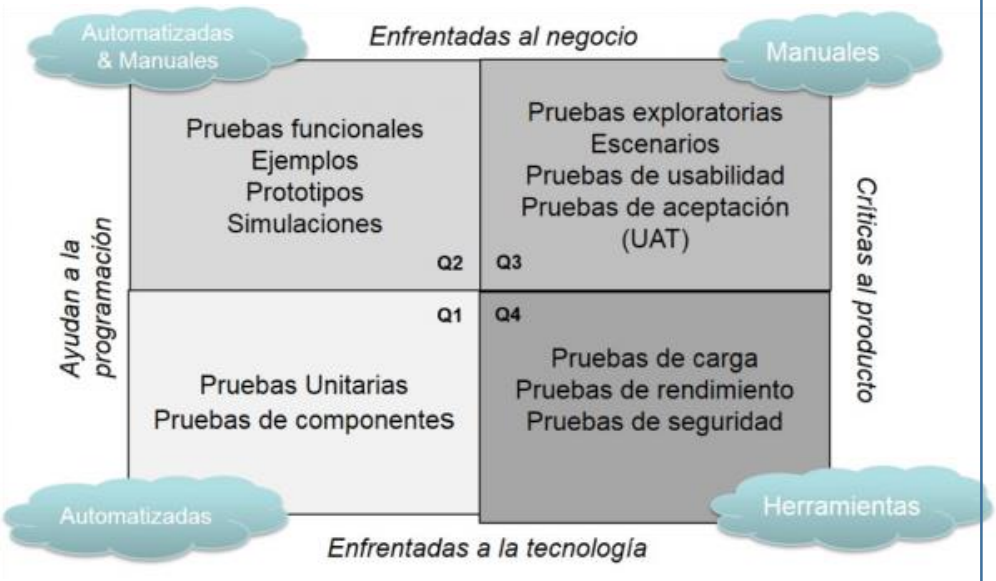

Figura 2. Cuadrantes del Agile Testing. Extraida de Crispin & Gegory (2014)

- Pruebas de apoyo al equipo: Los cuadrantes del lado izquierdo (Q1 y Q2) ayudan al equipo a definir que código necesitan escribir y cuando han terminado de escribirlo. Este concepto es nuevo para muchos testers, ya que el Testing tradicional tiene como objetivo encontrar errores. Las pruebas de estos cuadrantes guían el desarrollo de la funcionalidad y luego cuando son automatizadas sirven para apoyar la refactorización y el agregado de nuevo código sin causar resultados inesperados.
- Pruebas de críticas al producto: Los cuadrantes del lado derecho (Q3 y Q4) ayudan a los clientes a validar si lo que han solicitado se encuentra plasmado en el producto. El término "críticas al producto" no necesariamente debe asociarse a una connotación negativa, en estos cuadrantes también se da espacio a la sugerencia de mejoras.
- Pruebas enfrentadas a la tecnología: Los cuadrantes del lado inferior (Q1 y Q4) son de naturaleza técnica. La idea es someter a pruebas extremas la aplicación y, por ende, son llevadas a cabo generalmente por especialistas técnicos como testers, desarrolladores, arquitectos, entre otros.
- Pruebas enfrentadas al negocio: Los cuadrantes del lado superior (Q2 y Q3) tienen el foco en el negocio permitiendo confirmar el comportamiento de determinada funcionalidad. Son llevadas a cabo por el *product owner* o el *Project manager*.

Para la mayoría de los productos, se necesita como mínimo las pruebas expresadas en los cuadrantes para asegurar que se está entregando un producto con valor y además permitirá que el equipo sepa cuando una función ha cumplido los criterios del cliente en cuanto a funcionalidad y calidad.

Técnicas en el Agile Testing

Las pruebas en entornos ágiles implican cambios importantes respecto a los métodos de trabajo tradicionales. El enfoque de "todo el equipo prueba" tiene como finalidad integrar la calidad al desarrollo del producto desde el primer momento, a diferencia del enfoque tradicional, de primero fabricar el producto y luego inspeccionarlo para determinar su nivel de calidad.

A continuación se presentarán diferentes técnicas y metodologias que refuerzan este nuevo método de trabajo.

Test Driven Development (TDD)

El desarrollo guiado por pruebas, es una técnica basada en tres etapas, las cuales conforman un ciclo. La primera de ellas consiste en escribir una prueba para una funcionalidad que aún no está desarrollada y verificar que falle. A continuación, la siguiente etapa tiene por objetivo incorporar el mínimo código posible (sin que importe el uso de buenas prácticas) para que la prueba desarrollada en la etapa anterior ahora pueda ser ejecutada y los resultados sean los esperados. La última etapa consiste en refactorizar el código para proporcionarle calidad aplicando el uso de buenas prácticas de programación.

Esta metodología exige al desarrollador un cambio en su metodología de trabajo ya que proporciona un orden en el cual debe llevarse a cabo cada etapa y además exige que tenga la capacidad de dividir el problema del desarrollo de una nueva funcionalidad en otros más chicos, de manera tal que pueda cumplir con las etapas mencionadas de una manera sencilla. En caso que el desarrollador deba recurrir a cuestiones complejas para poder expresar una prueba, es un claro indicador de que quizás sea necesaria reconsiderar la estructura de los componentes e intentar simplificarlos.

Como ventajas se pueden mencionar que permite un desarrollo más rápido y se optimiza el mantenimiento y evolución del código ya que cambios grandes pueden ser fácilmente probados debido a que ya existe un conjunto de pruebas que asegura que el comportamiento es el mismo.

Behaviour Driven Development (BDD)

Bajo este enfoque primero se desarrolla una prueba funcional y luego se ejecuta el desarrollo aplicando TDD hasta que la prueba es exitosa. Esta técnica permite desarrollar, probar y pensar el código desde la perspectiva del usuario.

Se trabaja con historias de usuario en un lenguaje compartido entre programadores, el cliente y el software. Este lenguaje común se llama Gherkin (tiene una sintaxis particular) y no se utiliza sólo para describir funcionalidades sino que también es utilizado para crear pruebas automatizadas y generar documentación interna.

Como ventajas se puede mencionar que se mejora la comunicación entre el equipo de desarrollo y el cliente, además que al estar definiendo pruebas en relación a los comportamientos esperados es más sencillo de definir un criterio de aceptación de las funcionalidades previamente a ser desarrolladas.

Continuous Test Driven Development (CTDD)

El desarrollo guiado por Test Continuos es una técnica ágil que extiende a TDD mediante la ejecución automática de pruebas en segundo plano, a veces denominadas pruebas continuas. Es decir, un programador que aplica TDD procede a escribir la prueba manualmente para probar la funcionalidad que aún no ha escrito. En lugar de ejecutar manualmente la prueba para validar que falle y luego de implementado el código, validar que el resultado sea el esperado, el test se ejecutará automáticamente y se proporciona automáticamente el feedback al programador. El programador ahora podrá seguir con sus tareas de escrituras de test y código asociado y el entorno de desarrollo integrado (IDE) será el encargado de ejecutar los test en forma constante e indicar sus resultados.

CAPITULO 3

Metodología de Relevamiento

Relevamiento

Metodología

Para realizar un correcto abordaje sobre las Técnicas de Testing Aplicadas en Metodologías Ágiles, se ha realizado una búsqueda sistemática de bibliografía que permita recopilar información y analizarla posteriormente.

Existen diversas maneras de realizar una búsqueda sistemática, sin embargo para este trabajo se ha utilizado la metodología propuesta por Medina-López, Marin-Garcia, & Alfalla-Luque (2010) que consta de cinco etapas: (1) Identificación del campo de estudio y período a analizar, (2) Selección de las fuentes de información, (3) Realización de la búsqueda, (4) Gestión y depuración de los resultados y (5) Análisis de los resultados (ver **Figura 3**).

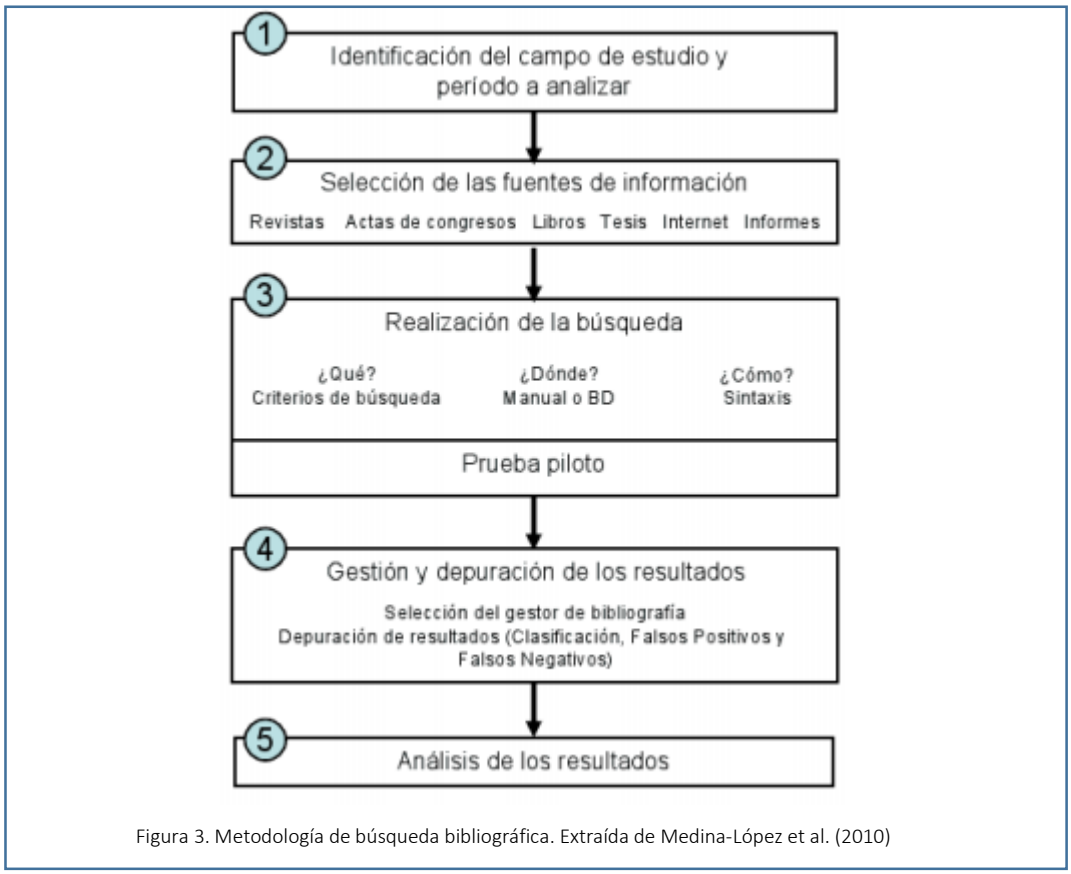

Figura 3. Metodología de búsqueda bibliográfica. Extraída de Medina-López et al. (2010)

Campo de estudio y período a analizar

La primera de las etapas a desarrollar, consiste en identificar el campo de estudio. En este caso, se desea realizar un análisis de aquellas técnicas de testing documentadas en distintas bibliografías y que son aplicadas en desarrollos de software guiado por metodologías ágiles. El resultado de dicho análisis proporcionará un estado del arte de dicho campo. Teniendo en consideración el objetivo planteado en esta investigación, y considerando además que en búsquedas a través de bases de datos Medina-López, Marin-Garcia, & Alfalla-Luque (2010) indicaron “El éxito de la misma y la eficiencia en tiempo vendrá dada, entre otras, por la adecuada selección de dichas palabras clave” (p. 25); se ha establecido en primera instancia

como palabras claves de la investigación: “agile testing”, la cual en etapas posteriores sufrirá modificaciones como se mencionará más adelante en el apartado “realización de búsqueda”.

Debido al dinamismo propio del campo de estudio, el período de trabajos seleccionados será de los últimos 5 años, es decir desde el año 2018 hasta los trabajos presentados durante el año 2022. Este lapso de tiempo se considera suficiente para realizar un correcto estudio de los temas involucrados y disponer de aquellas técnicas que se estén usando actualmente.

Fuentes de información

Dada la importancia que tiene la calidad de los elementos de entrada en todos los sistemas, se ha decidido trabajar con producciones que hayan sido sometidas a un proceso de revisión antes de su publicación y de esta manera, proporcionen calidad en la información a ser procesada. Dichas producciones son libros, secciones de libros, reseñas de libros, artículos de revisión, artículos de investigación, actas de conferencias y además artículos de revistas científicas y profesionales. Además se delimita trabajar con producciones escritas en idioma español o inglés.

Realización de búsqueda

La metodología propone establecer los criterios de búsqueda a emplear con el objetivo de filtrar las referencias de interés. En este proceso, se determinó realizar la búsqueda de referencias de manera automática (usando recursos digitales) consultando para tal fin:

- Bibliotecas Digitales: Biblioteca Electrónica de Ciencia y Tecnología[1].
- Bases de Datos: IEEE Xplore, Google Académico
- Redes sociales académicas: Mendeley

En primera instancia se establecieron como criterios de búsqueda a emplear en las fuentes de información anteriormente mencionadas los siguientes:

- Términos de búsqueda “*agile testing*”, por encontrarlos representativos del campo de estudio a investigar y ser frecuentemente utilizados por quienes se dedican al aseguramiento de la calidad en el ámbito del desarrollo de software.
- Periodo de tiempo comprendido desde el 2018 al 2022.
- Tipo de producciones: libros, secciones de libros, reseñas de libros, artículos de revisión, artículos de investigación, actas de conferencias y además artículos de revistas científicas y profesionales, en idioma español o inglés.

A partir de estos criterios de búsqueda, se han realizado diversas pruebas piloto en diferentes plataformas, lo cual permitió depurar y mejorar la estrategia de búsqueda.

Pruebas Piloto

La primera prueba piloto fue realizada en la plataforma IEEE Xplore. Se realizó una búsqueda avanzada la cual permitió incorporar como filtros los siguientes criterios: términos de búsqueda “*agile testing*” (presentes en el título o resumen o palabras claves) y el período de tiempo a considerar (2018 a 2022). Esta primera búsqueda, arrojó 510 resultados (fecha de consulta 13/06/2022).

[1] https://biblioteca.mincyt.gob.ar/

Posteriormente se ha podido depurar los resultados obtenidos según el campo de estudio de la producción. IEEE Xplore ofrece el filtro "*Publication Topics*" donde se especifica una amplia lista de áreas temáticas, de las cuales se seleccionaron nueve: "Software Prototyping", "Program Testing", "Software Engineering", "Software quality"; "Cloud Computing", "Internet", "Software maintenance", "Formal Specification", "Educational courses"; por encontrarlas en relación con el campo de estudio establecido para la investigación.

Como resultado se obtuvieron 284 producciones. Vale mencionar que no se pudieron aplicar los criterios: tipo de producciones a considerar e idioma. Posteriormente se realizó una depuración manual de los mismos, encontrándose que muchos de ellos eran falsos positivos, es decir que "el trabajo ha sido seleccionado atendiendo a la estrategia de búsqueda establecida, pero a pesar de ello, no se corresponde con el objeto real de la misma." (Medina-López et al., 2010, p. 25). Además se encontró que para el acceso de muchos de los resultados arrojados era necesario un pago, por lo que sumó como criterio de búsqueda, que los recursos sean de libre acceso.

La segunda prueba piloto fue realizada en la plataforma Mendeley con los criterios de búsqueda establecidos inicialmente. Al realizar la búsqueda en dicha plataforma se pudieron aplicar los siguientes criterios: términos de búsqueda, período de tiempo y el tipo de producciones a consultar (*book, book section, conference proceedings, journal*), dando como salida 1.299 resultados (fecha de consulta 13/06/2022). Se infirió que el elevado número, es consecuencia de que la plataforma no permitió filtrar por el campo de estudio (ya que no ofreció una lista de áreas temáticas a considerar), y además no permitió especificar la localización de los términos de búsqueda en el título o resumen o palabras claves como si lo permitió la plataforma anterior. Tampoco se permitió seleccionar el idioma de las producciones a considerar.

A partir de los resultados obtenidos (en la búsqueda automática) se procedió a depurar manualmente dichos resultados. Para ello, se tomó como referencia los siguientes criterios de búsqueda: aparición de los términos de búsqueda en el título o resumen o palabras claves; pertenencia al área temática de interés; idioma inglés o español. En consecuencia, del conjunto de 1.299 producciones se identificaron 576 resultados que si bien cumplían con los criterios mencionados, muchos de ellos eran falsos positivos o requerían de un pago para su acceso.

Modificación de los criterios de búsqueda

En este punto, a partir del conocimiento extraído de las pruebas pilotos, se decidió modificar la estrategia de búsqueda cambiando los términos de búsqueda "*agile testing*" por *"técnicas de pruebas; metodología ágil"* y su correspondientes traducciones en inglés *"techniques testing; agile methodology"*.

Se efectuaron nuevas pruebas pilotos con estos nuevos criterios. La primera prueba fue ejecutada en Google Académico. Esta plataforma permitió establecer los siguientes criterios de búsqueda: *buscar artículos con todas las palabras* "techniques testing agile methodology", *donde las palabras aparezcan* "en todo el artículo" (ya que no se podía indicar que aparezca en secciones específicas como título, resumen o palabras claves, y al seleccionar la opción "*en el título del artículo*", no se producían resultados, ya que la búsqueda resultaba muy restrictiva), y *artículos fechados entre* 2018 y 2022. De esta forma se obtuvo 17.300 resultados (fecha de consulta: 10/10/22).

Se detectó que muchos de los artículos arrojados se encontraban en relación a la temática COVID, es por ello que se precisó la búsqueda aún más mediante la exclusión de dicho término, y además se observó una mayor eficiencia si se introducía como criterio de búsqueda: *con todas las palabras* "agile methodologies"/ "metodología ágil", *con la frase exacta* "techniques testing" / *"*técnicas de pruebas*"; sin las palabras* "covid"; *donde las palabras aparezcan "*en todo el artículo*", y artículos fechados entre* 2018 y

2022. De esta forma se pasó de obtener 17.300 resultados a sólo 228 resultados en total (fecha de consulta 10/10/2022) con los términos seleccionados en ambos idiomas.

Otra prueba fue realizada en la plataforma Mendeley, la cual permitió ingresar los siguientes criterios de búsqueda: términos de búsqueda agile methodology techniques testing / metodología ágil técnicas de pruebas, periodo de años 2018 a 2022, y tipo de acceso "*open access*". Como resultado se obtuvieron 9 producciones (fecha de consulta 10/10/2022) con los términos seleccionados en ambos idiomas.

Ejecución de búsqueda

Finalmente, luego de un análisis de los resultados arrojados a partir de las modificaciones en los criterios de búsqueda, se concluyó que dichas búsquedas habían sido más efectivas que las anteriores, y se procedió entonces a ejecutarla de la manera más parecida posible en las otras plataformas, sin perder de vista que existen diferencias entre ellas y que ofrecen posibilidades de filtrado diferentes. Fecha de consulta 10/10/22.

En relación a las diferencias entre las plataformas de búsquedas seleccionadas, cabe mencionar que al proceder con la "Biblioteca Electrónica de Ciencia y Tecnología", debió realizarse un paso previo a la estrategia de búsqueda establecida, dicha etapa consistió en filtrar de la lista ofrecida en "recursos de acceso abierto" aquellas relacionadas con la temática a investigar, para ello se usaron dentro del filtro *temática* los valores "ingeniería eléctrica, electrónica e informática" y "ciencias de la computación e información", obteniéndose dos resultados: arXiv.org y engrXiv, en las cuales se procedió a realizar finalmente la estrategia de búsqueda.

A continuación se presenta en la **Tabla 1**, los resultados finales del proceso de búsqueda estando satisfecho por la cantidad de producciones a evaluar y la calidad de las mismas.

Plataforma	Criterio de búsqueda	Resultados Obtenidos
Google Académico	agile methodology "techniques testing" –covid	48
	metodología ágil "técnicas de pruebas" –covid	180
IEEEXplore	(("Document Title":agile methodology OR "Document Title":techniques testing) AND ("Abstract":agile methodology OR "Abstract":techniques testing) AND ("Author Keywords":agile methodology OR "Author Keywords":techniques testing)) Filters Applied:2018-2022 "Open Access Only"seleccionado	2
	(("Document Title": metodología ágil OR "Document Title": técnicas de pruebas) AND ("Abstract": metodología ágil OR "Abstract": técnicas de pruebas) AND ("Author Keywords": metodología ágil OR "Author Keywords": técnicas de pruebas)) Filters Applied:2018-2022 "Open Access Only"seleccionado	0
Mendeley	agile methodology techniques testing	8
	metodología ágil técnicas de pruebas	1

Biblioteca Electrónica de Ciencia y Tecnología	Engineering engrxiv archive[2]	metodología ágil	0
		técnicas de pruebas	0
		agile methodology techniques testing	0
	ArXiv[3]	order: -announced_date_first; size: 50; date_range: from 2018-01-01 to 2022-12-31; classification: Computer Science (cs), Electrical Engineering and Systems Science (eess); include_cross_list: True; terms: AND all=techniques testing; AND all=agile methodology	1
			Total de resultados 240

Tabla 1. Cantidad de resultados obtenidos según plataforma y criterio de búsqueda

Gestión y Depuración de los Resultados

El siguiente paso a seguir en la metodología es la depuración de los resultados. De la etapa anterior – realización de la búsqueda – se obtuvieron en primera instancia 240 resultados, de los cuales luego de realizar una primera gestión de los mismos, su número se redujo considerablemente debido a diversas circunstancias como por ejemplo: obtención de la misma producción en diferentes plataformas, lo cual es considerado como un duplicado; obtención de producciones que no pertenecían al grupo determinado como fuentes de información; producciones que no podían ser accesibles debido a que su link estaba roto/caído. De esta manera, al finalizar esta gestión inicial de los resultados el conjunto de trabajos se redujo a un total de 177 producciones, con las cuales se llevó a cabo la depuración.

El proceso de depuración consistió en primera instancia en la clasificación de las producciones obtenidas, en una de las siguientes categorías: *seleccionado, dudoso o falso positivo*, donde en la primer categoría se encuentran los trabajos de interés, en la segunda aquellos trabajos donde fue necesario realizar un análisis más profundo acudiendo a las conclusiones del trabajo o bien secciones específicas que permitan discernir si el trabajo debía ser seleccionado o no y por último, en la tercera categoría se incluyeron aquellos trabajos que han sido seleccionados según la estrategia de búsqueda pero pese a ello, no resulta de interés para este trabajo. El criterio para clasificar las producciones fue la de realizar un análisis del título, abstract (resumen) y palabras claves.

Como resultado de esta etapa, se obtuvieron 12 trabajos seleccionados, 13 dudosos y 152 falsos positivos. Al analizar en profundidad aquellas producciones dentro de la categoría de dudosos, se observó que aquellas se encontraban relacionadas a la temática pero no cumplían con los criterios de selección, por lo que luego de este análisis, se obtuvo finalmente 12 trabajos seleccionados y 165 falsos positivos. Un detalle de este proceso puede observarse en la **Tabla 2.**

Plataforma	Resultados Obtenidos	Etapa 1 – Gestión de Resultados	Etapa 2 – Depuración de Resultados			Etapa 3 – Resultados Finales	
			Seleccionado	Dudoso	Falso Positivo	Seleccionados	Falso Positivo
Google Académico	228	167 (125+42)	6	12	149	6	161
IEEEXplore	2	2	0	0	2	0	2

[2] https://engrxiv.org/index
[3] https://arxiv.org/

Mendeley		9	7	6	0	1	6	1
Biblioteca Electrónica de Ciencia y Tecnología	Engineering engrxiv archive	0	0	0	0	0	0	0
	ArXiv	1	1	0	1	0	0	1
Total		240	177	12	13	152	12	165

Tabla 2. Cantidad de resultados gestionados según plataforma

A continuación se procede con un análisis de los resultados, destacando particularidades de cada uno y su relación con el tema de investigación.

CAPITULO 4

Análisis de Resultados

Análisis de resultados

A partir del conjunto de producciones seleccionadas en la etapa anterior de la metodología, a continuación se presenta un análisis de las mismas según dos perspectivas. La primera de ellas, está relacionada al análisis general del conjunto de trabajos, permitiendo así instruir al lector en como está determinada la muestra. La segunda perspectiva, tiene un enfoque teórico-técnico, en donde se detallaran las técnicas analizadas en cada producción y se pondrán en tensión el conjunto de trabajos identificando similitudes o diferencias entre los mismos.

Al finalizar dichos análisis, se realizará una reflexión general, destacando emergentes y atributos observados.

Análisis general del conjunto

Para conocer características del conjunto se han tomado principalmente tres dimensiones las cuales son, (1) distribución de los trabajos según el año de publicación, (2) tipo de investigación y enfoque utilizado, y por último (3) si realizan algún aporte, ya sea desde la perspectiva de ofrecer una nueva técnica de testing o metodología.

En lo que refiere a la primera dimensión, los estudios seleccionados fueron publicados entre el 2018 y el 2022, en la Figura 4 se puede ver una distribución de los trabajos según el año de publicación. En dicha figura, se observa que el año 2020 se destaca con el mayor número de trabajos dentro del conjunto con un total de 7 producciones (58,33%), luego le sigue año 2018 con un total de 4 producciones (33,33%) y finalmente el año 2019 con una única producción dentro del conjunto (8,34%). En lo que respecta a los años 2021 y 2022 se observa que durante éste período no se ha ubicado ningún trabajo del conjunto, al analizar la situación se concluyó que la misma respondería principalmente al tamaño del conjunto, el cual es afectado por las siguientes variables: a) criterios de selección de trabajos mediante el empleo de los criterios prestablecidos y b) el valor de la información, ya que las investigaciones más recientes, completas, multifocales y referidas a técnicas avanzadas de testing en contextos ágiles son producciones que tienen un valor económico, por ser las que se encuentran más actualizadas, y que por cuestiones de presupuesto fueron inaccesibles para el marco de este trabajo.

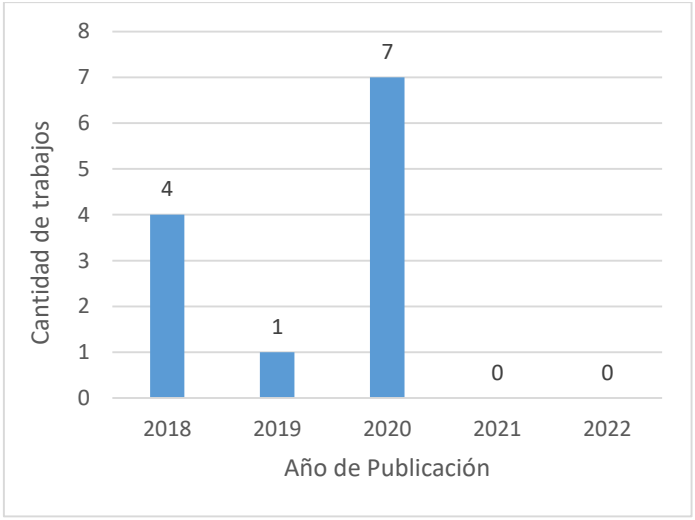

Figura 4. Distribución de los trabajos seleccionados según año de publicación

Por otro lado, si analizamos al conjunto de trabajos seleccionados desde el punto de vista referido a que tipo de trabajo son y que enfoque utilizan, se puede decir que 10 producciones (83,33%) corresponden a una investigación aplicada (práctica o empírica) que, como dice Vargas Cordero (2009), es un tipo de investigación que utiliza determinados conocimientos para aplicarlos a un problema específico que ocurre en la práctica y de esta forma generar nuevo conocimiento o simplemente resolver la cuestión. El resto de las producciones (16,67%) son trabajos de indole académica que, en este caso especifíco, abordan un tema desde lo teórico pero no presentan resultados prácticos de sus conclusiones en un proyecto de la industria.

Siendo más específico y abordando el analisis pero desde el punto de vista del enfoque con el cual están realizadas las producciones, puede decirse que 5 de ellas (41,67%) aplican un enfoque experimental. En Huynh, Le-Trinh, Ha, & Man (2020) los autores proponen un enfoque práctico desarrollado por ellos para la aplicación de una metodología de testing basado en contexto (Context-driven testing) y lo ponen en prueba en dos proyectos llevados a cabo por una software Company llamada Enclave ubicada en Vietnam. En Faizullah & Almutairi (2018) los autores presentan una técnica diseñada por ellos para la priorización de las pruebas dentro de un test de regresión y analizan los resultados de la misma en una serie de productos que presentan diferente complejidades, número de versiones y tamaños de la suite de prueba. En Hallmann (2020) el autor propone un modelo propio para mejorar el proceso cognitivo de la comprensión de historias de usuario y luego lo valida contra un set de 74 historias de usuario vinculadas a un proyecto relacionado al sector automotriz de Alemania. Otro de los trabajos que aplica este enfoque es el de Li, Wang, Yang, & Wang, (2020) en donde los autores proporcionan un framework para seleccionar un subconjunto de casos de prueba adecuados (que cubran completamente todo el código modificado y el código afectado por dichos cambios) para ser ejecutados en el proceso de integración continua y así reducir el costo de la prueba tanto como sea posible sin sacrificar la calidad del software. Dicho framework es evaluado en 18 proyectos de código abierto de las comunidades de Eclipse y Apache. Por último, dentro de aquellos trabajos con un enfoque experimental se encuentra el de Romano, Scanniello, Baldassarre, Fucci, & Caivano, (2020) donde los autores validan los resultados de un experimento anterior en el que alumnos universitarios de tercer año ha tenido que aplicar una metodología de desarrollo orientada al testing (TDD), replicando dicho experimento pero con una población diferente (alumnos universitarios de segundo año) para así aportar con resultados concluyentes al empleo de esta metodología.

En relación con los trabajos con un enfoque de estudio de caso, dentro del conjunto existen 2 producciones (16,67%). La primera de ellas es la de Sophocleous & Kapitsaki (2020) en donde los autores, a partir de un relevamiento de las tendencias en técnicas de pruebas, demuestran que aplicando una combinación de ellas en un proyecto específico (en particular, una aplicación web en el dominio de la salud en la República de Chipre) reducen el número de defectos identificados por el cliente. Por último, en Sánchez & Lago, (2019) los autores realizaron una investigación en el proceso de pruebas sobre los productos de software que son creados por el Departamento de Señales Digitales de la Facultad de Ciencias y Tecnologías Computacionales (CITEC) ubicado en Cuba exponiendo la falta de un proceso que les permita realizar pruebas en el código usando técnicas de Caja Blanca, lo cual provoca que los desarrolladores y testers de software deban realizar estas técnicas de forma manual y debido a la complejidad de estas se termine omitiéndolas. Para suplir esta falta y alcanzar mejores niveles de calidad, analizaron diferentes herramientas de la industria del testing pero decidieron diseñar e implementar una propia, ajustándola a las necesidades específicas de su proceso de pruebas y enfocándola específicamente en la realización de pruebas de Mutación, es decir una herramienta que testea los test unitarios.

En lo que respecta a otras 2 producciones del conjunto (16,67%), las mismas se realizaron bajo un enfoque de trabajo de investigación. En Yu, Alegroth, Chatzipetrou, & Gorschekb (2019) se realiza un análisis del estado del arte relacionado a la utilización de un entorno de integración continua para las

pruebas de requerimientos no funcionales a través de una rigurosa selección basada en un set inicial de 747 producciones. Por otro lado en Jammalamadaka (2018) partiendo de que las pruebas de regresión son extremadamente costosas (principalmente en términos de tiempo de ejecución) y que no pueden ser ignoradas completamente sin la consecuencia de reducir o comprometer la calidad del producto, el autor realiza una investigación en este ámbito para analizar aquellas técnicas que puedan ser utilizadas para seleccionar un caso de prueba y priorizarlo dentro de una suite de tests de regresión.

Concluyendo el análisis del grupo según esta dimensión, queda por mencionar que existen dos trabajos (16,67%) que han realizado encuestas para sus análisis. En Maqbool, Abbas, Rehman, & Rehman (2018) los autores exploran en 21 organizaciones de IT ubicadas en Pakistán las tendencias de las prácticas de testing de software y como resultado encuentran que la educación y el entrenamiento en dichas prácticas no es lo suficientemente competitivo, según su análisis la industria en esa región debe realizar importantes mejoras en relación a este tema y esperan que así sea de la mano de académicos e impulsores de la industria. Por otro lado en Binamungu, Embury, & Konstantinou (2020) interesados por la metodología BDD, decidieron encuestar a practicantes de esta metodología acerca de sus opiniones sobre los criterios de calidad que son importantes para llevarla a cabo. En el trabajo se informan y analizan los resultados de dicha encuesta y además se presenta un enfoque para definir la calidad de una suite BDD.

Finalmente, en lo que respecta a la última producción del conjunto (8,33%), se puede ubicar que la misma realiza una revisión sistemática de literatura. En Mascheroni & Irrazábal (2018) los autores analizan aquellas propuestas, técnicas, enfoques, métodos, herramientas y soluciones presentes en la bibliografía con el fin de validar si las pruebas continuas son el componente faltante en aquellos proyectos que poseen entrega continua. Un resumen de todo lo mencionado anteriormente puede encontrarse en la Figura 5, en donde se observa la distribución de los trabajos según el enfoque de los mismos.

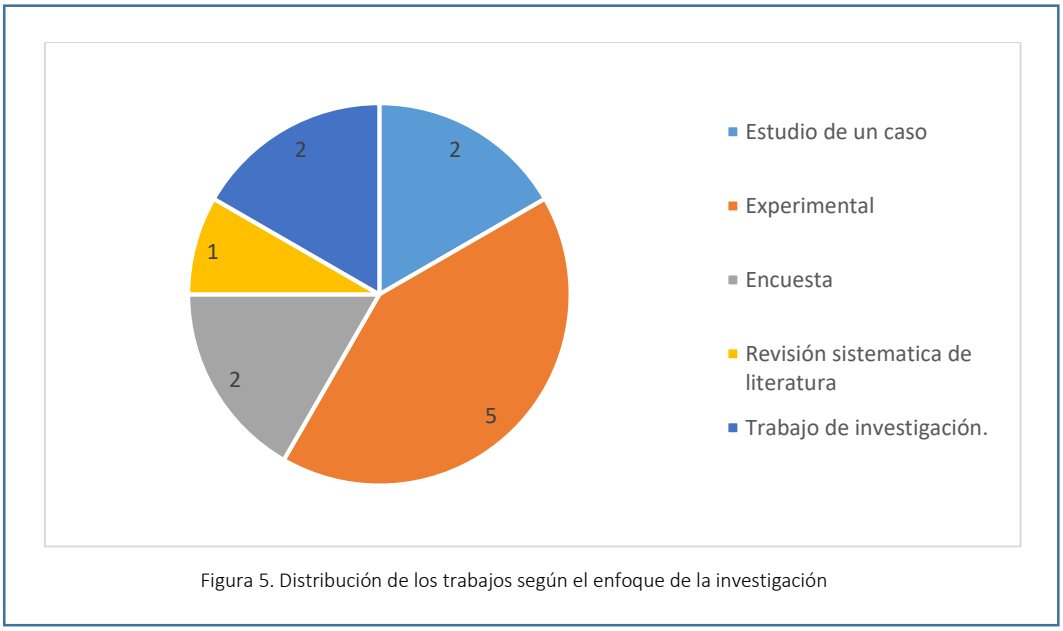

Figura 5. Distribución de los trabajos según el enfoque de la investigación

Hasta aquí, se ha analizado al conjunto de producciones seleccionadas de acuerdo a su distribución en el periodo de tiempo 2018 – 2022, y de acuerdo al enfoque seguido por los autores a la hora de abordar su tema de investigación. Resta por analizar al conjunto desde la dimensión de aquellos trabajos donde se ofrezca una nueva técnica de testing o metodología.

Según esta dimensión, se puede decir que 7 de las producciones (58,33%) plantean la necesidad de algo propio, ya sea una metodología o técnica de testing y el resto de los trabajos (41,67%) no lo hace.

Análisis del contenido y sus relaciones

Habiendo realizado un análisis sobre el conjunto de trabajos seleccionados en la sección anterior, a continuación se presenta un análisis detallado de cada uno de los mismos, profundizando sobre aquellas técnicas de testing al cual refieren, metodologías que evalúan o llevan a cabo para complementar el proceso de aseguramiento de la calidad y además entablando similitudes o diferencias con otros trabajos dentro del conjunto de manera tal de obtener diferentes puntos de vista sobre un mismo tema.

Adentrándonos en el análisis, se puede segmentar al conjunto en cinco categorías generales (en las cuales un trabajo puede pertenecer a más de una categoría específica). Dichas categorías son: técnicas de caja blanca vs caja negra; pruebas de regresión (regression test); integración continua; técnicas de testing sobre los requerimientos y finalmente metodologías.

Técnicas de caja blanca vs caja negra

Dentro de la primera categoría, es importante destacar los aportes de tres trabajos puntualmente. En los resultados del relevamiento realizado en Sophocleous & Kapitsaki (2020) se observa que la mayoría de los participantes emplean para asegurar la calidad técnicas de caja negra (87,3%), otros (11,1 %) una combinación entre pruebas de caja blanca y caja negra y sólo un (1,2 %) realiza pruebas de caja blanca exclusivamente. Los autores agregan que estos resultados son esperados debido a que su relevamiento fue dirigido a los miembros del equipo de control de calidad, lo cual es llamativo según este enunciado, pareciera que es esperable y aceptable que el equipo de aseguramiento de calidad sólo realice testing empleando este tipo de técnicas.

En Maqbool, Abbas, Rehman, & Rehman (2018) se puede encontrar una similitud con el trabajo anterior, ya que según los autores, muchas organizaciones adoptan comúnmente técnicas de caja negra para realizar sus tareas de testing (mencionan como ejemplo una técnica en particular llamada análisis de flujo de datos) en lugar de emplear técnicas de caja blanca donde según ellos, técnicas como el análisis simbólico o testing de mutación (que clasifican dentro de esta categoría) son raramente utilizadas por empresas debido a la falta de adopción y *expertise* necesario para aplicarlas.

En contra de las posturas mencionadas anteriormente, se encuentran los autores Sánchez & Lago, (2019) que destacan la importancia de las técnicas de testing de caja blanca ya que permiten derivar casos de prueba en donde se garantice que todas las rutas dentro de un módulo en particular se ejecute al menos una vez (concepto de cobertura de código). De esta manera como se mencionó anteriormente, los autores desarrollaron una herramienta propia que permite automatizar pruebas de mutación (la cual fue sometida a pruebas de aceptación por parte de los stakeholders) y que en conjunto con otras técnicas de caja blanca como pruebas unitarias y pruebas del camino básico, han podido detectar un 25% más de fallos, aumentando así la fiabilidad en los resultados del testing y reducir los tiempos de prueba agilizando el proceso de desarrollo.

Habiendo expuesto los argumentos y trabajos que abordan principalmente la temática de esta primer categoría, surge como emergente el cuestionamiento sobre los motivos que inducen a los equipos de aseguramiento de calidad a emplear mayoritariamente técnicas de caja negra en lugar de aplicar un mix que incluya dichas técnicas de caja negra pero que sean complementadas con otras como las de caja blanca. Tal vez, dichos motivos estén relacionados con una cuestión de tiempo para realizar las pruebas dentro del

sprint, en donde si la relación entre tester vs desarrollador se encuentra desproporcionada y existen errores en producción que el ingeniero de calidad debe analizar e intentar reproducir y a su vez, existe una gran número de bugs reportados que debe llevarse la trazabilidad y el estado actual, indudablemente el equipo de calidad se encontrará “apagando incendios” y ejecutando aquellas pruebas más fáciles de observar que planteando escenarios de prueba más complejos.

Pruebas de regresión

En segundo lugar, en lo que refiere a la categoría relacionada con las pruebas de regresión existen 3 trabajos que lo abordan como tema principal. En Jammalamadaka, (2018) el autor plantea que el testing de regresión es una de las fases en el desarrollo del software más importantes, costosas y que no puede ser ignorada, reducida o minimizada sin comprometer a la calidad del producto. Además otro factor importante es que mientras más casos de prueba se incluyan dentro de esta etapa, más costosa será en términos de tiempo y más aún dentro del ámbito de un sprint donde el tiempo juega un papel fundamental. Debido a estos factores, en dicho trabajo se hace una clasificación para la optimización de la suite de casos de pruebas en tres categorías, las cuales son: i) reducción de los casos de prueba; ii) selección de casos de prueba y iii) priorización de los casos de prueba y para cada una de ellas luego se plantean técnicas acompañadas con diferentes métricas para medir la eficiencia de las mismas. Finalmente, se concluye que muchos investigadores han propuesto una gran variedad de técnicas para las sesiones de test de regresión y que el desafío se encuentra en seleccionar aquella que sea más eficiente en términos de efectividad-costo.

Por otra parte, un punto muy similar con el anterior puede encontrarse en Faizullah & Almutairi (2018) donde se enuncia que precisamente debido a las restricciones de tiempo dentro de entornos de desarrollo ágiles es necesaria una priorización de las pruebas de regresión. Los autores plantean una técnica para seleccionar casos de prueba basada en la información del próximo sprint, en donde en equipos pequeños, cada miembro sabrá exactamente que módulos/partes del software se modificarán y esto ayudará a identificar en equipo, el mejor conjunto posible de casos de prueba que serán utilizados para las pruebas de regresión en el siguiente sprint. Finalmente, luego de analizar los resultados de la aplicación de dicha técnica en diferentes productos, con diferentes complejidades, los autores concluyen que dicha técnica aporta a la efectividad en la detección de fallas y permite la reducción de tiempo (y por ende costo) de las pruebas de regresión. Se destaca la necesidad de automatizar casos de prueba para la obtención de resultados más rápidos aunque se necesite personal experimentado.

Por último, en Li, Wang, Yang, & Wang (2020) también se plantea que con el crecimiento de los sistemas, ejecutar todos los casos de prueba en un test de regresión para verificar la calidad del nuevo código no resulta económico en determinado punto. Es por esto que los autores desarrollaron un framework para seleccionar un subconjunto de casos de prueba basado en un análisis estático del código que tiene por objetivo seleccionar un subconjunto de pruebas preciso para cubrir completamente todo el código modificado y afectado por los cambios y luego dos enfoques de selección de casos de prueba basados a nivel de clase y otro a nivel de métodos. En sus resultados encuentran que la selección a nivel de método suele ser más efectivo que la selección de casos de prueba a nivel de clase y es importante destacar que articulan todo su análisis y selección de casos de prueba con la premisa de que ya se encuentran automatizados y disponibles para ser ejecutados en un proceso de integración continua.

Como resultado del análisis de esta categoría y de aquellos trabajos que lo abordan como tema principal, se puede leer como emergente que todos los autores dentro del conjunto de trabajos seleccionados, están de acuerdo en que las pruebas de regresión son costosas en término de tiempo. Si bien estas pruebas pueden realizarse en forma manual, la automatización de las mismas aporta a que la ejecución sea más

rápida y con menos errores, sin embargo, dicha automatización no resuelve uno de los problemas de fondo el cual trata que mientras más grande sea el sistema y por ende más tiempo (sprints) se lo esté desarrollando, se necesitará más tiempo dentro de cada sprint para ejecutar todas las pruebas de regresión y es por esto que los autores trabajaron en una priorización de las mismas a partir de sus puntos de vista singulares. Por otro lado, es importante destacar que sólo uno de los tres trabajos dentro del conjunto parte del concepto de tener una suite automática de casos de prueba, lo cual produce un disparador hacia la pregunta ¿qué cantidad de proyectos realmente tiene una suite automática de casos de prueba actualizada y funcionando en proyectos guiados por metodologías agiles? pregunta que excede al ámbito de este trabajo pero que sin embargo es importante considerar ya que dimensiona los procesos de testing dentro de metodologías ágiles.

Por último, una observación que se le puede realizar a los trabajos es que ninguno de ellos tiene en consideración aquellos errores encontrados en un ambiente productivo y la consecuente reflexión de ¿por qué el set de casos de prueba de regresión no detectó dicha falla? Dicho feedback, el cual podría ser costoso en términos de confiabilidad del usuario, imagen de la empresa, costo monetario, entre otros, no es tenido en cuenta para la selección de casos de prueba a incluir en los test de regresión y quedan por fuera del alcance de las técnicas presentadas lo cual deposita toda la responsabilidad en el equipo que lleva adelante el proceso de testing.

Integración Continua

En esta categoría se logra encontrar tres trabajos que abordan dicho tema principalmente. El primero de ellos es el de Yu, Alegroth, Chatzipetrou, & Gorschekb (2019) en donde se menciona que la integración continua (CI) es ampliamente adoptada para mejorar la calidad del software a través de la verificación y validación de cada cambio de código mediante el uso de herramientas y tecnologías de automatización. Sin embargo, los autores mencionan que los ambientes de CI existentes principalmente se enfocan en los requerimientos funcionales del software dejando de lado los requerimientos no funcionales (NFR) debiendo ser testeados en forma manual debido a la complejidad inherente a ellos y es por esto que los autores proponen un marco de CI enfocado específicamente hacia estos requerimientos junto con un mapeo de herramientas de la industria que podrían ser utilizadas. Resulta interesante observar que los autores realizaron una comparación de los NFRs extraídos de su conjunto de producciones seleccionadas con las características de calidad del producto software mencionados en las ISO 25010 y si bien pudieron mapear la mayoría de las características, mencionan que la “portabilidad” fue el único NFR que no pudieron (Ver. Figura 1).

Las conclusiones presentadas en dicha producción, se encuentran relacionadas a que la CI se encuentra infrautilizada para el testo de NFR, posiblemente debido a que las herramientas carecen de soporte para este tipo de testing y además resultan complejas de integrar y mantener. Por otro lado, también se menciona que algunos NFR son difíciles de automatizar (como por ejemplo la usabilidad) y otros en cambio, se desconocen implementaciones prácticas para el testeo debido a que no se encontraron trabajos que los mencionen (como por ejemplo portabilidad, compatibilidad). Por último, los autores mencionan que existe un ratio muy bajo de CI-NFR entre implementaciones en la industria vs estudios académicos siendo estos últimos los de mayor proporción.

Otro de los trabajos, Mascheroni & Irrazábal (2018) relaciona el proceso de integración continua con los conceptos de *“Continuos Delivery (CD)”* y *“Continuos Testing (CT)”* en donde el primero se trata de una disciplina del desarrollo del software en la que se construye con una calidad tal que permita lanzarse a producción en cualquier momento y el segundo es el proceso de ejecución de pruebas automatizadas como parte del proceso de entrega del software, de manera tal de obtener información inmediata. Los autores

al procesar los resultados de su búsqueda sistemática de literatura encontraron que el término CT fue evolucionando a través de los años (al principio estaba relacionado a la ejecución constante de unit test en la computadora del desarrollador y hoy en día tiene un concepto más amplio en donde no se limita sólo a los unit test sino que comprende también a todo caso de prueba que pueda ser automatizable) y que existe un consenso en la idea de ejecutar test automáticos que validen lo más rápido posible la existencia de fallas en el producto, por lo que los términos CD y CT están directamente relacionados.

Por otro lado, en este trabajo se puede encontrar un análisis de técnicas, enfoques y herramientas utilizadas para resolver problemas de CD, los cuales están relacionados con el consumo del tiempo (hay que considerar que cualquier cambio que se incorpore al repositorio del código debe ser testeado lo más pronto posible y que grandes suites de pruebas conllevan una gran consumo de tiempo) al unit testing y al testing funcional. Dentro de los enfoques que abordan esta problemática se encuentran los de agrupamiento de escenarios de prueba, priorización de escenarios, ejecución en paralelo de test automáticos, ejecución de test en un servidor de integración, entre otros.

Finalmente, el otro trabajo que aborda esta categoría es el de Li, Wang, Yang, & Wang (2020) en donde los autores destacan que la integración continua es un diferenciador en el desarrollo ágil ya que proporciona un rápido feedback sobre los cambios de código que han sido subidos al repositorio, pero es importante una priorización de los escenarios de prueba ya que de modo contrario existirán restricciones de tiempo y el proceso no será eficiente.

Una vez presentados los tres trabajos cuyo principal tema de investigación es el de integración continua, se puede decir que todos los trabajos están de acuerdo en que este proceso brinda importantes mejoras en el proceso de desarrollo. En la mayoría de los trabajos se destaca que es importante, no sólo quedarse con el proceso implementado, sino que se requiere sprint tras sprint analizar cuál será el conjunto de pruebas que deberá ejecutarse ya que una ejecución de la suite completa produciría cuellos de botella y no sería lo suficientemente ágil. Por otro lado, resultó sumamente interesante el trabajo que aborda el testing de requerimientos no funcionales y como pueden ser incluidos dentro de este proceso de integración.

Técnicas de testing sobre los requerimientos

En esta categoría se catalogó a una sola producción que es la de Hallmann (2020), allí el autor menciona que las historias de usuario son ampliamente utilizadas para comunicar requerimientos en el desarrollo de software guiado bajo metodologías ágiles. Sin embargo, a pesar de su amplia utilización, es muy común que las mismas contengan errores y no estén con el suficiente nivel de detalle, lo cual repercute en el equipo a la hora de estimar esfuerzo y estén de acuerdo con los criterios de aceptación ya que cada persona completa esa ambigüedad de requerimientos faltantes a su manera, haciendo que aparezca el re trabajo de la tarea y discusiones sobre el alcance. Para evitar esto, los autores proponen un modelo en el cual diferentes factores son medidos con un conjunto de métricas estipuladas y esto proporciona una noción acerca de la calidad de la historia de usuario que se está analizando.

Al realizar un análisis sobre dicho trabajo, se puede mencionar que resulta interesante tener un modelo que evalúe la calidad de las historias de usuario, las cuales serán tomadas como uno de los inputs en el proceso de desarrollo del software y que mejorando esta entrada se podrían llegar a tener mejores salidas en el proceso de desarrollo. Sin embargo, también es necesario destacar que los autores reconocen la necesidad de probar el modelo en diferentes organizaciones, proyectos, equipos de trabajo ya que si bien sus resultados son prometedores pueden estar sesgados a la forma de trabajo y las condiciones propias del proyecto en el cual fue aplicado este modelo.

Metodologías

En esta última categoría, existen tres trabajos que abordan esta categoría de manera principal. Uno de ellos es el de Binamungu, Embury, & Konstantinou (2020) en donde se analiza la metodología BDD desde la perspectiva del testing considerándola a nivel de suite de escenarios y no a nivel de escenarios puntuales (que es donde según los autores la mayoría de la literatura se concentra). En este trabajo se plantea la problemática de que a pesar que un escenario esté bien definido y tenga calidad, al momento de incrementar el conjunto de escenarios surgen nuevos desafíos que afectan a la calidad del conjunto y algunos atributos son puestos en jaque, tales como: mantenibilidad, reutilización, acoplamiento, cohesión, legibilidad, claridad entre otros.

Dicho esto, los autores proponen cuatro principios para evaluar la calidad de las suites BDD, los cuales son: i) principio de conservación de los pasos, ii) principio de conservación del vocabulario del dominio, iii) principio de eliminación del vocabulario técnico y finalmente iv) principio de conservación de la abstracción apropiada. A través de una encuesta a practicantes de esta metodología logran confirmarlos y descubrir nuevos, tales como la legibilidad y claridad de las especificaciones resultantes. El trabajo concluye abordando los resultados de su encuesta y proporcionando un enfoque para definir la calidad de una suite BDD.

Otro de los trabajos presentes en esta categoría es el de Romano, Scanniello, Baldassarre, Fucci, & Caivano (2020) en donde se aborda la metodología TDD (Test-Driven Development). Los autores mencionan que dicha metodología permite mejorar la calidad (interna y externa) del software así como también la productividad de los desarrolladores, sin embargo, luego de realizar una investigación y al observar que en los trabajos detectados no se presentaban resultados concluyentes de dichas afirmaciones, deciden replicar un experimento particular que permite comparar TDD con una metodología ágil tradicional.

En el experimento base, se había obtenido como resultado que los desarrolladores con un nivel de expertise básico, parecía gustarle menos implementar código bajo esta modalidad y que eran expuestos a mayores niveles de frustración al tener que dedicar más tiempo a escribir pruebas unitarias, en lugar de concentrarse exclusivamente en la implementación de la funcionalidad. En la replicación del experimento base y variando únicamente el contexto y número de participantes, los autores obtuvieron resultados disimiles ya que no se observó como emergente dichas características negativas por lo que los autores concluyen que el funcionamiento de esta metodología depende del grado de experiencia de los desarrolladores.

Por último, se encuentra el trabajo de Huynh, Le-Trinh, Ha, & Man (2020) si bien se menciona que investigadores, expertos e ingenieros de software continúan estudiando diferentes técnicas de testing que puedan ser adaptadas al desarrollo ágil actual como por ejemplo TDD, BDD y CDT (Context-driven testing), los autores proponen un enfoque práctico para la aplicación este último, el cual llaman CDTiP. Este enfoque permite que el equipo de calidad elija sus técnicas de prueba, documentos y objetivos de prueba basados en la situación particular del proyecto. Los autores proponen que en lugar de aplicar lo que se considera la mejor práctica, debido a que el tiempo disponible para probar una funcionalidad generalmente está limitado, el equipo de calidad debe aplicar diferentes prácticas (reduciendo tiempo y costo) que funcionen bien, aplicando su criterio, habilidades técnicas y la comprensión del escenario actual (el cual recomiendan que sea representado a través de herramientas como XMind Zen, One2Explore para graficar ideas y conceptos y de esta forma, obtener una ayuda rápida para sintetizar ideas que necesitan ser testeadas y tenidas en consideración en el proceso de ejecución de pruebas).

A través de un análisis en dos casos de estudio, se experimentó en términos de cobertura de test, detección de errores y esfuerzo de testeo la efectividad de este enfoque, comparando su rendimiento contra los métodos tradicionales de prueba. Para esto, se tomó una misma pieza de software, la cual fue testeada por dos equipos de QA (Quality Assurance) con la misma cantidad de integrantes y experiencia dentro del rubro y la misma cantidad de tiempo y documentación disponible. Como resultado del estudio empírico, el equipo que aplicó el enfoque CDTiP obtuvo mejores resultados en comparación con el método de prueba existente. Según los autores, el ejercicio de realizar mapas que representen diferentes contextos junto con el uso de herramientas para la gestión de las actividades de testing, permiten una mayor cobertura de los escenarios, disminuir el tiempo de diseño de los casos de prueba, cubrir contextos no solicitados y encontrar un mayor número de defectos mediante el uso de CDTiP, aportando eficiencia al proyecto en el esfuerzo, tiempo y capacidad de detección de errores que ayudan a mejorar la calidad del producto.

Analizando estas tres producciones, se puede decir en primer lugar que afortunadamente el proceso de pruebas está tomando una mayor relevancia dentro del espacio del desarrollo de software y esto posibilita que se diseñen metodologías que brindan un marco al proceso. Sin embargo, es importante destacar que en las tres producciones se presentaron desventajas, necesidad de soporte de herramientas y condiciones especiales para la implementación de dichas metodologías y es interesante mencionar que difícilmente se encuentren observaciones con el mismo rigor en trabajos que traten de la metodología scrum, a pesar de que cada organización lo adapta a su contexto empresarial. Dicho esto, pareciera ser que tomará algún tiempo hasta que se popularicen enfoques orientados a las pruebas y que se comprenda que dichas metodologías también pueden adaptarse a las particularidades de cada proyecto – empresa.

Reflexiones generales de la etapa de Análisis

En líneas generales, el conjunto de trabajos fue heterogéneo en términos de enfoque de la investigación, es decir se encontraron trabajos con un enfoque experimental, otros tenían enfoque de estudio de un caso, de revisión sistemática de literatura, entre otros.

Por otro lado, en cuanto al contenido, también fue heterogéneo, debiéndose clasificar los trabajos en distintos tópicos para poder abordarlos de forma ordenada (como ser: técnicas de caja blanca vs caja negra, pruebas de regresión, integración continua, técnicas de testing sobre los requerimientos y finalmente metodologías). Se considera además que dichos tópicos son los más importantes y representativos al momento de hablar de pruebas dentro de metodologías ágiles.

Profundizando en el contenido del conjunto, se observó que muchas de las técnicas descriptas pueden ser aplicadas tanto a aplicaciones web como móviles lo cual es un punto interesante ya que habla de la versatilidad de dichas técnicas, sin embargo también es necesario mencionar que los trabajos exponen que dependiendo del expertise de quien las ejecuta se podrán obtener mejores o peores resultados. Desafortunadamente dentro del conjunto de producciones seleccionadas en base a los criterios de búsqueda establecidos, no se encontraron trabajos que contengan técnicas de testing en big data, técnicas de testing para datawarehouse, técnicas de testing de accesibilidad de las plataformas, ni tampoco técnicas impulsadas o asistidas mediante inteligencia artificial en donde posiblemente se obtengan mejores rendimientos que en una técnica manual de priorización de casos de prueba de regresión.

A su vez, otra cuestión que emerge del conjunto analizado es la necesidad de diseñar y construir herramientas ad-hoc para determinado proyecto-empresa debido a que, por más que exista un gran número de herramientas comerciales, éstas no alcanzan a cumplir todos los requerimientos específicos del contexto en el cual se van a utilizar. Dicha cuestión, plantea el interrogante de si se seguirán desarrollando

este tipo de herramientas ad-hoc o bien existirán en el futuro herramientas comerciales genéricas que permitan instanciar modelos particulares y así ser adaptadas fácilmente a cada organización y proyecto en el cual desarrolla sus actividades.

Finalmente es importante destacar que la automatización de casos de prueba estuvo implícita en los temas tratados en los trabajos del conjunto (por ejemplo en aquellos trabajos relacionados a integración continua), pero ninguno de ellos desarrolló técnicas de automatización, tanto desde la perspectiva funcional detallando que automatizar, como la perspectiva de como desarrollar test automáticos dentro de un sprint.

CAPITULO 5

Conclusiones y trabajos futuros

Conclusiones y trabajos futuros

A modo de cierre, la investigación realizada permitió relevar y analizar aquellas técnicas de testing aplicadas en metodologías ágiles alcanzándose el objetivo definido al inicio del trabajo. Se empleó una metodología de cinco pasos para hallar los trabajos a evaluar y posteriormente se realizó un análisis de los mismos a partir de diferentes puntos de vista y agrupándolos según diferentes criterios, presentándose al final de cada punto de vista una reflexión sobre los mismos.

Una conclusión inicial que se desprende de este análisis es que el conjunto de producciones abordado en este estudio exhibió una diversidad significativa en cuanto a las técnicas de testing empleadas. Al evaluar el conjunto de producciones en relación con las características de calidad del producto de software, conforme a la norma ISO 25010 (ver Figura 1), se observa que las características que predominaron en las discusiones fueron principalmente la Fiabilidad y la Adecuación Funcional del software. En contraste, las características de portabilidad y compatibilidad no fueron abordadas por ninguna de las producciones dentro de este conjunto.

Se concluye entonces que para obtener un software de calidad, es necesario evaluarlo desde diversas perspectivas, ya que representa un desafío que abarca múltiples dimensiones y características. Según las producciones analizadas, se destaca que mayormente se prefiere evaluar la calidad a través del empleo de técnicas de caja negra en lugar de técnicas de caja blanca o mixta, debido al escaso tiempo dentro del sprint para diseñar y ejecutar las pruebas. En este contexto, resulta fundamental incorporar prácticas como la integración continua, la automatización de casos de prueba y la selección y priorización de escenarios de prueba que conforman a los test de regresión, de manera tal de optimizar los beneficios de su ejecución. A su vez, es esencial mantener una perspectiva integral, ya que si bien un test individual puede cumplir con las mejores prácticas, su verdadero valor se revela al integrarse a un conjunto más amplio sin comprometer los atributos de mantenibilidad, reutilización, legibilidad, entre otros, que caracterizan al conjunto en su totalidad.

Finalmente, en respuesta a la pregunta sobre como evaluar la calidad de un software, la respuesta no es simple. Aunque el profesional de aseguramiento de la calidad dispone de los recursos presentados en este informe, la evaluación debe tener en cuenta que este proceso es continuo y requiere una adaptación constante de estrategias, ya sea en el ámbito del testing o de la gestión para el ámbito específico en el cual el proyecto de desarrollo de software se lleva a cabo.

Antes de concluir este trabajo final integrador, no quiero dejar de mencionar las valiosas experiencias y conocimientos que he adquirido durante su desarrollo y como este contribuyó a mi crecimiento y desarrollo profesional, proporcionándome nuevas perspectivas sobre situaciones descritas en diversas producciones y vividas de manera particular en el proyecto en el cual me desempeño.

Un ejemplo concreto que ilustra esta situación se relaciona con la aplicación de técnicas de priorización de casos de prueba de regresión. Los trabajos que abordan esta técnica lo hacen de manera general, ofreciendo soluciones genéricas aplicables en diversos contextos. Sin embargo, en mi entorno laboral específico, la aplicación de esta técnica resulta imperativa debido a la naturaleza del proyecto en el que participo. Nos enfrentamos a la constante presión del mercado y a situaciones imprevistas que requieren movilizar el código desde ambientes de prueba a entornos de producción en plazos extremadamente cortos. Esta dinámica interrumpe la planificación del análisis de calidad de los cambios, lo que nos lleva a realizar con los otros miembros del equipo de QAs, una priorización de casos de prueba de regresión para asegurar la calidad del producto que se lanzará al mercado.

Durante la investigación, al analizar los documentos que tratan esta técnica y considerar sus enfoques y supuestos, identifiqué áreas de mejora para nuestro proceso de priorización de casos de prueba de regresión. Uno de los aspectos clave de mejora se centró en un documento que describe los cambios de

versión, detallando las funcionalidades añadidas o afectadas por algún cambio y proporcionando información sobre los errores solucionados. Transformé este documento para que de una formalidad sin uso interno pasara a ser un recurso de alta calidad y utilidad para el equipo de QA. Actualmente, refleja con precisión las modificaciones en el código y se lo utiliza como entrada para el proceso de priorización. Este documento, junto con otras variables relevantes, nos permite definir de manera metodológica los escenarios de prueba que deben incluirse en el conjunto de pruebas de regresión. De esta manera, hemos transformado un proceso manual que dependía de la memoria del equipo en un enfoque más estructurado y eficiente.

Por último, otro ejemplo particular y relevante para resaltar es que, gracias a este trabajo, tuve la oportunidad de conocer la técnica de testing basada en contexto, explorar la técnica de testing de requerimientos y comprender cómo la aplicación de mapas mentales aporta beneficios al equipo al alinear a todos sus miembros. Afortunadamente, al aplicar el uso de mapas mentales en algunas historias de usuario, observé que el testing llevado a cabo por el desarrollador era más completo y contemplaba diferentes escenarios y contextos que, de no haber implementado estas técnicas, hubieran pasado desapercibidos. Como resultado, esta mejora permitió que el equipo de QA disponga de un poco más de tiempo y pueda dedicárselo a otras tareas.

Para finalizar y considerando la evolución constante de la tecnología, un aspecto relevante para investigaciones futuras podría ser el desarrollo de una metodología de testing que facilite la integración de diversas técnicas, contextos de aplicación y directrices generales. Esta metodología estaría diseñada para orientar a aquellas personas que carecen de experiencia en el área, permitiéndoles incorporarse de manera rápida y eficiente a proyectos que ya se encuentran en producción y puedan desempeñar sus funciones sin depender de la existencia de documentación del sistema (la cual raramente existe) o de reuniones informales con otros miembros del equipo de QA. En segundo lugar, sería interesante el desarrollo de una IA que asista al tester en su labor, ya sea detectando errores (como por ejemplo validaciones a nivel de pixel entre la versión anterior del sistema y la versión actual, donde las diferencias detectadas entre ambas versiones sería reportado como un error) o bien, como asistente en la automatización de casos de prueba (similar a GitHub Copilot), o como asistente en la priorización de escenarios de pruebas de regresión.

REFERENCIAS BIBLIOGRAFICAS

Referencias Bibliográficas

Referencias Bibliográficas

Albin, S. T. (2003). *The Art of Software Architecture: Design Methods And Techniques* (1 ed.). John Wiley & Sons.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001). *Manifesto for Agile Software Development*. Obtenido de http://agilemanifesto.org/

Binamungu, L. P., Embury, S., & Konstantinou, N. (2020). Characterising the Quality of Behaviour Driven Development Specifications. (V. Stray, R. Hoda, M. Paasivaara, & P. Kruchten, Edits.) *Agile Processes in Software Engineering and Extreme Programming: 21st International Conference on Agile Software Development, XP 2020, 383*, 87-102. doi:https://doi.org/10.1007/978-3-030-49392-9_6

Crispin, L., & Gregory, J. (2014). *Agile Testing. A practical guide for testers and agile teams.* Addison Wesley Longman (A Mike Cohn signature book).

Dijkstra, E. W. (1969). Notes on Structures Programming. 15-64. Obtenido de https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF

Faizullah, S., & Almutairi, S. (01 de 2018). Considerations for Regression Testing Process in Agile Development Environments. *International Journals of Advanced Research in Computer Science and Software Engineering, 8*, 153. doi:http://dx.doi.org/10.23956/ijarcsse.v8i1.565

Hallmann, D. (2020). "I Don't Understand!": Toward a Model to Evaluate the Role of User Story Quality. (V. Stray, R. Hoda, M. Paasivaara, & P. Kruchten, Edits.) *Agile Processes in Software Engineering and Extreme Programming: 21st International Conference on Agile Software Development, XP 2020, 383*, 103-112. doi:https://doi.org/10.1007/978-3-030-49392-9_7

Huynh, T., Le-Trinh, P., Ha, N.-H., & Man, N. (2020). An Effective Approach for Context Driven Testing in Practice — A Case Study. *International Journal of Software Engineering and Knowledge Engineering, 30*, 1245-1262. doi:10.1142/S0218194020500333

IEEE Computer Society. (1990). IEEE Standard Glossary of Software Engineering Terminology. doi:10.1109/IEEESTD.1990.101064

IEEE Computer Society. (1998). IEEE Standard for a Software Quality Metrics Methodology. doi:10.1109/IEEESTD.1998.243394

ISO. (23 de 05 de 2022). *ISO25000*. Obtenido de https://iso25000.com/index.php/normas-iso-25000/iso-25010

ISO/IEC 25010. (23 de 05 de 2022). *ISO/IEC 25010*. Obtenido de https://iso25000.com/index.php/normas-iso-25000/iso-25010

Jammalamadaka, K. (2018). A SURVEY ON TEST CASE SELECTION AND PRIORITIZATION TECHNIQUES. *International Journal of Research in Engineering and Technology, 07*(12). doi:https://doi.org/10.15623/ijret.2018.0712009

Kaner, C. (1983). *Defining Exploratory Testing*. Obtenido de https://kaner.com/?p=46

Laing, S., & Greaves, K. (2015). *Growing Agile: A Coach's Guide to Agile Testing* (1 ed.). Growing Agile.

Li, Y., Wang, J., Yang, Y., & Wang, Q. (2020). An Extensive Study of Class-level and Method-level Test Case Selection for Continuous Integration. *The Journal of Systems and Software, 167*. doi:https://doi.org/10.1016/j.jss.2020.110614

Maqbool, B., Abbas, M., Rehman, F., & Rehman, S. (2018). Implementation of Scrum in Pakistan's IT Industry. doi:10.1145/3180374.3181336.

Mascheroni, M., & Irrazábal, E. (2018). Continuous Testing and Solutions for Testing Problems in Continuous Delivery: A Systematic Literature Review. *Computación y Sistemas, 22*(3), 1009-1038. doi:doi: 10.13053/CyS-22-3-2794

Medina-Lopez, C., Marin-Garcia, J. A., & Alfalla-Luque, R. (2010). Una propuesta metodológica para la realización de búsquedas sistemáticas de bibliografía (A methodological proposal for the systematic literature review). *Working Papers on Operations Management, 1*(2), 13-30. doi:https://doi.org/10.4995/wpom.v1i2.786

Pressman, R. S. (2010). *Ingeniería del Software. Un enfoque práctico* (7 ed.). McGraw-Hill.

Razak, R. A., & Fahrurazi, F. R. (2011). Agile Testing with Selenium. *Malaysian Conference in Software Engineering*, (págs. 217-219). doi:10.1109/MySEC.2011.6140672

Romano, S., Scanniello, G., Baldassarre, M., Fucci, D., & Caivano, D. (2020). Results from a replicated experiment on the affective reactions of novice developers when applying test-driven development. (V. Stray, R. Hoda, M. Paasivaara, & P. Kruchten, Edits.) *Agile Processes in Software Engineering and Extreme Programming: 21st International Conference on Agile Software Development, XP 2020, 383*, 223-239. doi:https://doi.org/10.1007/978-3-030-49392-9_15

Sánchez, Á. N., & Lago, C. N. (2019). Pruebas de mutación, control sobre variaciones en el código fuente. *I+D Tecnológico, 15*(2), 38-45. doi:https://doi.org/10.33412/idt.v15.2.2228

Sharmaa, G., Sharmaa, S., & Gujrala, S. (2015). A Novel Way of Assessing Software Bug Severity Using Dictionary of Critical Terms. *Procedia Computer Science, 70*, 632-639. doi:10.1016/j.procs.2015.10.059

Somerville, I. (2005). *Ingeniería del Software* (7 ed.). Madrid: Pearson Addison Wesley.

Sophocleous, R., & Kapitsaki, G. (2020). Examining the Current State of System Testing Methodologies in Quality Assurance. (V. Stray, R. Hoda, M. Paasivaara, & P. Kruchten, Edits.) *Agile Processes in Software Engineering and Extreme Programming: 21st International Conference on Agile Software Development, XP 2020, 383*, 240-249. doi:https://doi.org/10.1007/978-3-030-49392-9_16

Vargas Cordero, Z. R. (2009). LA INVESTIGACIÓN APLICADA: UNA FORMA DE CONOCER LAS REALIDADES CON EVIDENCIA CIENTÍFICA. *Revista Educacion, 33*(1), 155-165.

Yu, L., Alegroth, E., Chatzipetrou, P., & Gorschekb, T. (2019). Utilising CI environment for efficient and effective testing of NFRs. *Information and Software Technology, 117*. doi:https://doi.org/10.1016/j.infsof.2019.106199