Aseguramiento de calidad sobre el sistema Pharmacy to Home(P2H)

Daniel Steve Blandón Sánchez

Informe de práctica para optar al título de Ingeniero de Sistemas

Asesora
Sandra Patricia Zabala Orrego, Especialista (Esp) en Gerencia de Proyectos

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería de Sistemas
Medellín, Antioquia, Colombia
2023

TABLA DE CONTENIDO

LISTA DE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

IEEE	Institute of Electrical and Electronics Engineers
ISTQB	International Software Testing Qualifications Board.
E2E	End to End
CP	Caso de prueba
TC	Test Case
HU	Historia de usuario
P2H	Pharmacy to Home
UdeA	Universidad de Antioquia

RESUMEN

En este documento se expondrán los elementos fundamentales en las pruebas funcionales de software implementadas en la compañía Omnivida SAS durante el periodo de prácticas, partiendo desde la definición, diseño, desarrollo y ejecución de casos de pruebas y escenarios, construidos a partir de las historias de usuario con una sintaxis Gherkin que permite la fácil interpretación y comprensión por parte de todo el equipo y no solo el personal con conocimiento técnico. La documentación y el reporte de hallazgos se realizó en la plataforma Azure DevOps, donde se facilitó el seguimiento y trazabilidad del proyecto, para así brindar calidad al software y agilizar su desarrollo detectando fallas en etapas tempranas. También se exploró la automatización de pruebas, técnica que aporta agilismo al proceso de repetir pruebas y regresión, dicha automatización llevada a cabo con Cypress en pruebas end-to-end donde se prueba tanto front-end como back-end y se puede hacer aserciones no solo del diseño en el front sino también de las respuestas en el back permitiendo una mayor cobertura de los distintos flujos de un proceso dado. Finalmente se dará a conocer la importancia que tienen las pruebas de software y todo lo que pueden aportar para garantizar la calidad no solo de los productos sino también de los procesos.

Palabras clave — **Calidad de software, Pruebas de software, Pruebas funcionales, Casos de prueba, Automatización de pruebas, Cypress, End-to-end, Azure DevOps.**

ABSTRACT

In this document, the fundamental elements of software functional tests that were implemented at the company Omnivida SAS during the internship will be exposed, starting from the definition, design, development and execution of the test cases and scenarios; they were built from the user story using the Gherkin syntax, which allows the easiest interpretation and comprehension by the entire team instead of only by staff with technical knowledge. The documentation and reporting of bugs was done on the platform Azure DevOps, where the monitoring and traceability of the project were facilitated, in order to provide quality to the software and speed up its development by detecting failures in early stages. Test automation was also explored; it is a technique that increases the expedience of the process of repeating tests and regression. Said automation was carried out using Cypress in end-to-end tests where both front-end and back-end are tested and assertions can be made, not only about the design in the front-end, but also about the responses from the back-end, allowing a greater coverage of the different flows of a given process. Finally, the importance of software testing and all that it can contribute to quality assurance of not only of products, but also of processes will be shown.

Keywords — **Quality Assurance, QA, QA Automation, Testing, Test Case, Cypress, E2E, Azure DevOps.**

I. INTRODUCCIÓN

En la construcción de software, el aseguramiento de la calidad hace parte importante del proceso, ya que necesariamente son llevadas a cabo a lo largo del ciclo de vida; en el presente documento se mostrarán las pruebas realizadas sobre el sistema Pharmacy to Home(P2H), producto de la compañía Omnivida SAS, esta Empresa se dedica a la consultoría y apoyo en diversos procesos en el área de la salud. En sus inicios P2H era un proyecto bajo un desarrollo tercerizado, hasta que la compañía decide crear un equipo propio de desarrollo para continuar con su construcción.

Se presentará la creación y metodología usada para implementar el proceso de pruebas de software, (principalmente funcionales y asociadas al cambio) en un equipo donde no eran llevadas a cabo inicialmente, todo esto con el fin de mejorar la calidad del producto, apoyar diferentes procesos y detectar fallos en etapas tempranas del desarrollo, los cuales son más fáciles de corregir permitiendo ahorrar tiempo y costos; Demostrado así como las pruebas brindan seguridad en la construcción del producto correcto.

Se definió e implemento una metodología de pruebas que será útil en otros proyectos llevados por la compañía, se adapta a las metodologías usadas en desarrollo, a las entregas continuas de nuevas funcionalidades, la regresión cuando hay nuevos despliegues, considera la deuda técnica y realiza pruebas de aceptación con cliente. Todo esto con el fin de garantizar la calidad de los productos este a su vez generó conciencia dentro del equipo respecto a la importancia de las pruebas de software.

II. PLANTEAMIENTO DEL PROBLEMA

La compañía Omnivida SAS fue fundada en el 2017, se dedica a la consultoría y apoyo para el mejoramiento de procesos en el área de la salud. Durante la contingencia por la pandemia en el año 2020, se creó una plataforma para el dispendio de medicamentos a domicilio, la cual permitió reducir el contacto entre los pacientes y el personal de farmacia; por ende, el riesgo de contagio entre las partes, siempre pensando en el cuidado y salud de las personas.

El proyecto consistió en el de desarrollo de un software para solicitar los medicamentos a domicilio (Pharmacy to home o P2H) para los clientes Sura y HelPharma, dicho proyecto era desarrollado por terceros, pero no se llegaba a un nivel de satisfacción con los tiempos de entrega y la calidad del producto. En la compañía no se contaba con un equipo de desarrollo dedicado y debido a la necesidad de apropiarse de sus productos y construirlos por sí mismos, se fue creando poco a poco un equipo para suplir las necesidades de algunos desarrollos tanto internos como externos, uno de ellos, la aplicación P2H. Se decide comenzar a realizar pruebas de software para garantizar la calidad del producto e instaurar una metodología que sea aplicable en todos los demás proyectos comenzando con una cultura de pruebas.

En el desarrollo de software un ciclo de vida básico puede ser definición de requisitos, diseño, implementación, pruebas e implantación, pero es un modelo viejo que se ha ido mejorando conforme avanza el tiempo para suplir las necesidades de los clientes, optimizando los tiempos de entrega, pero siempre pensando en la calidad el producto. Por ello es importante e indispensable el aseguramiento de la calidad, ya que es por medio de esta que se puede comprobar y dar cierta confianza en los productos desarrollados.

III. OBJETIVOS

A. Objetivo general

Realizar pruebas funcionales a P2H, no solo para las funcionalidades nuevas sino para las ya existentes, en búsqueda de defectos con el fin de garantizar la calidad y correcto funcionamiento del producto.

B. Objetivos específicos

- Crear casos de prueba para las funcionalidades requeridas en base a los requerimientos e historias de usuario.
- Probar funcionalidades nuevas y reportar los bugs encontrados.
- Poner al día la deuda técnica en pruebas pendientes de desarrollos previos.
- Aportar valor a los productos con el mejoramiento de su calidad.
- Generar conciencia de lo importante que son las pruebas de software en el área de TI y el equipo de desarrollo

IV. MARCO TEÓRICO

Las pruebas de software son una manera de asegurar la calidad del producto y reducir el riesgo de fallas **[1]**, para así aumentar la confianza del cliente. Para la implementación de la calidad de software es necesario tener en cuenta los principios y referentes de calidad como la IEEE 1012 **[2],** la ISO **[3]** o sistemas de gestión de calidad por organizaciones internacionales como la ISTQB, donde se tiene un marco de referencia con los lineamientos y ayuda a identificar el alcance e impacto que tiene el área de pruebas en las empresas.

Las pruebas son un conjunto de actividades que pueden planearse por adelantado y realizarse de manera sistemática **[4]**, dentro de ellas hay actividades como lo son la planificación de la prueba, análisis y diseño de pruebas, implementación y ejecución de pruebas, evaluación de criterio de salida y generación de informes; y actividades de cierre de pruebas **[5]**, en la **Figura 1** se puede ver cómo están relacionadas estas actividades, todas ellas ocurren de manera concurrente con las fases del proceso de desarrollo. Dado que en la actualidad se usan metodologías ágiles para el desarrollo de software, en la **Figura 2** se aprecia que se hacen entregas parciales de las funcionalidades en un proceso iterativo e incremental **[6]**, se solía pensar que las pruebas solo se hacían al final, pero hay diferentes tipos de pruebas y estas se ejecutan cada una en diferentes etapas del proceso, mejorando considerablemente la calidad del producto por la detección de errores en etapas tempranas del proyecto.

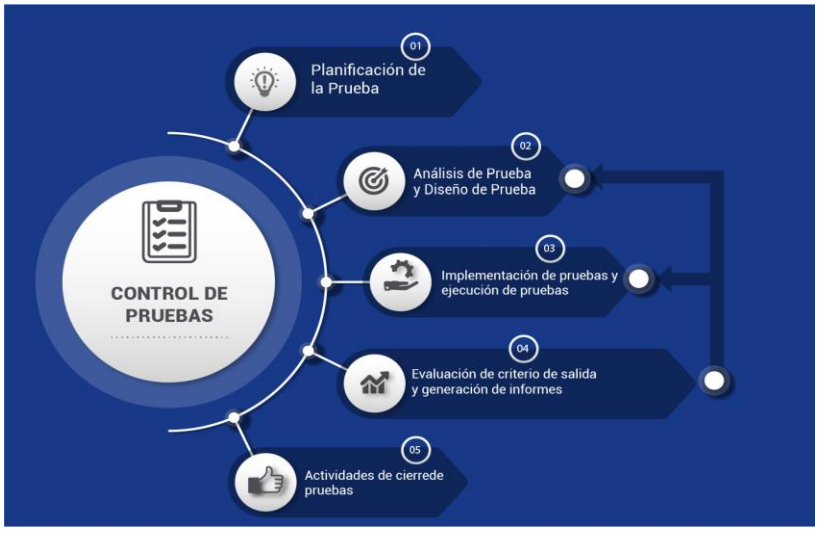

Fig. 1 Actividades de pruebas de software.

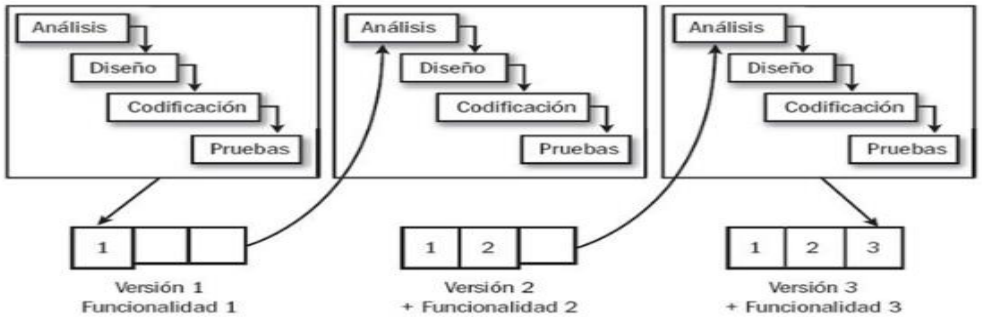

Fig. 2 Ciclo de vida iterativo incremental.

Dentro de las pruebas de software se encuentran diferentes tipos de pruebas y niveles de las mismas, los niveles son componentes, integración, sistema y aceptación, mientras que en los tipos se tiene pruebas funcionales, pruebas no funcionales, pruebas estructurales y pruebas asociadas al cambio **[5]**. Estas pruebas tienen lugar en diferentes momentos del desarrollo y tienen su objetivo y respectivo nivel o niveles en los que impactan. Las pruebas funcionales tienen como propósito verificar los requisitos funcionales, la verificación y validación van de la mano y son definidas en el estándar de la IEEE 1012, donde estos conceptos responden a las preguntas *¿estamos construyendo el producto correctamente?* y *¿estamos construyendo el producto correcto?* respectivamente, es decir, la verificación muestra que el producto está funcionando de paso a paso, mientras que la validación es la evaluación final de todo **[2]**. Las pruebas funcionales usualmente utilizan métodos de caja negra o caja blanca **[7]**, la diferencia consiste en el conocimiento del código, ya que para caja negra se realiza el proceso con entradas y salidas sin ver que pasa dentro, caso contrario en caja blanca donde si conocemos el código, dichas pruebas funcionales son aplicables en todos los niveles de pruebas, su ejecución se hace a partir de casos de pruebas (CPs), que describen el flujo de un proceso, las entradas y la salida esperada, tras su ejecución se tiene un resultado, el cual se compara con el esperado para indicar si se cumple la prueba y por ende el requerimiento.

La construcción de los casos de prueba (CP) va de la mano con los requerimientos, se tienen distintas formas para construir CP, una de ellas es convertir las historias de usuario (HU) en CP usando una sintaxis Gherkin (Given, When, Then), lo cual permite una mayor claridad ya que usa

un lenguaje natural y común para todo el equipo, incluso entendible para aquellos sin un dominio técnico del sistema.

Otro tipo de prueba que es de interés son las pruebas asociadas al cambio, como su nombre indica son pruebas tras realizar cambios, esto puede ser corrección de errores o cambios por nuevas funcionalidades, cuando es por corregir un defecto, el software debe probarse nuevamente para confirmar que ha sido corregido con éxito, esto se denomina re-test y se prueba la funcionalidad sobre la cual fueron aplicados los cambios, mientras que otra prueba asociada a cambios es la regresión, son la prueba de un software ya probado, tras su modificación, su propósito es identificar defectos como resultados del cambio, estos defectos pueden estar en el software objeto de las pruebas, o en cualquier otro componente de software asociado o no asociado. Conforme crece el programa sujeto a pruebas, las pruebas de regresión van aumentando y las tareas de pruebas para garantizar la calidad también lo hacen, es por ello por lo que se presenta la automatización como respuesta a grandes sets de pruebas que cumplen la característica de ser repetitivas y simples pero que con la gran cantidad de ellas cada vez requieren más tiempo.

La automatización de pruebas **[8]**, es una técnica usada para agilizar el proceso de las pruebas asociadas al cambio, aunque no todas las pruebas se pueden automatizar, se pretende reducir el tiempo que se invierte en las pruebas manuales de esta etapa y usar herramientas que hagan este proceso de manera autónoma ejecutando un script. En la automatización se suelen hacer pruebas E2E, donde se prueba tanto el front-end como el back-end, verifica el funcionamiento de toda una aplicación, también se conocen como pruebas de extremo a extremo, porque prueban de principio a fin el flujo de un proceso desde la interfaz del usuario. En la **Figura 3** podemos ver los 3 ciclos de vida que son el de desarrollo, pruebas manuales y automatizadas, todos con las mismas etapas, siendo el diseño del desarrollo la entrada para los análisis de los otros dos ciclos.

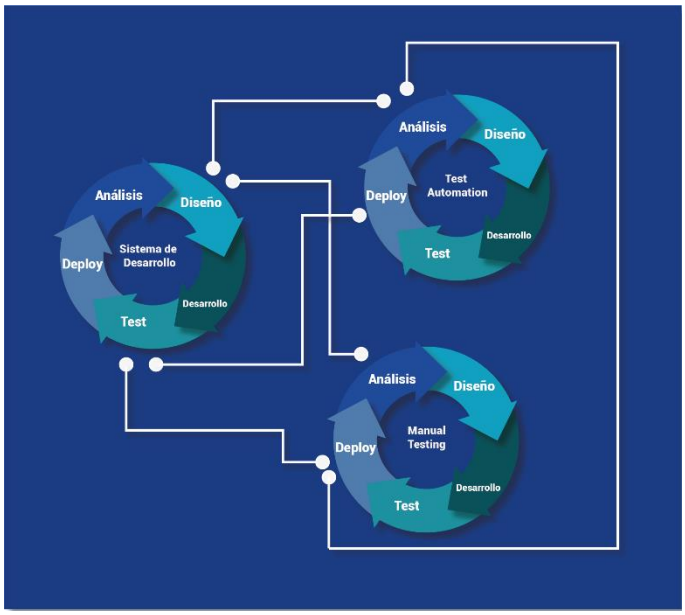

Fig. 3 Ciclo de desarrollo, pruebas manuales y automatizadas ***[8]***

V. METODOLOGÍA

Inicialmente para llevar a cabo este proyecto se dio un tiempo de capacitación y documentación, tanto de P2H como del contexto, también de algunos conceptos vistos en calidad y pruebas de software, herramientas útiles que se usaron para el desarrollo de la práctica en la creación de las pruebas funcionales de caja negra.

Se realizaron reuniones semanales con el equipo para revisar los avances y asignar nuevas tareas, se implementó un backlog y un check list en los que se disponen las tareas con todo el equipo para una mejor trazabilidad. Todos los documentos necesarios se comparten por Microsoft teams y por este mismo se hacen notificaciones de despliegues, teniendo una comunicación directa con desarrollo lo cual aporta flexibilidad y agilidad en el proceso. por otra parte, se acompañó todo el proceso desde el análisis y se hicieron reuniones para la revisión y el refinamiento de los requerimientos entre personal de análisis, diseño y pruebas.

En un nivel personal para el trabajo autónomo, se trabajó bajo una combinación entre Kanban y SCRUM **[9]**, dada la comunicación directa con el equipo de desarrollo muchas de las tareas eran añadidas por demanda, entonces se requería cierto agilismo, así como hacer seguimiento y poder lidiar con las tareas previas y cambiar prioridades si es el caso.

Kanban utiliza columnas para representar el flujo del trabajo; para este caso tenemos las tareas de diseñar CP, ejecución de pruebas, reporte de bugs, re-test, pruebas de regresión y aceptación; pero no todas las tareas necesitan su propia columna, como lo es el caso de reporte de bugs, re-test que son dos tareas dependientes una de otra. También proporciona ayuda con la carga de trabajo y una correcta distribución de esfuerzo ya que cuenta con los límites Kanban, estos delimitan la cantidad de tareas máximas para cada columna o etapa del flujo, otro nombre con que se le conoce es WIP **[9]** (work in progress).

A continuación, se presentan y describen las columnas:

> **To-do:** Límite Kanban 5. Es la entrada al proceso, es donde diseñan los CP para una funcionalidad teniendo en cuenta la historia de usuario para los escenarios de prueba, se implementa la sintaxis Gherkin para una transformación de HU a CP más ágil. Una vez diseñado, se crean los test case en la herramienta Azure DevOps y se mueve a la siguiente etapa del flujo que sería doing.
>
> **Doing**: Límite Kanban 5. En esta columna se concentra el flujo de trabajo, se tienen los CP listos para ejecutar, sí se ejecutan sin fallos se continúa con el flujo hacia la columna Done, en el caso contrario se deberá reportar el bug detallando el CP, las variables (sí son necesarias), la fecha de ejecución, el valor esperado, el valor obtenido y la versión del sistema sobre la que fue ejecutado, luego de crear el reporte del bug en Azure DevOps se mueve a la columna Review.
>
> **Review**: Los ítems acá presentes se encuentran a la espera de solucionar el bug. Una vez sea corregido se ejecutará nuevamente la prueba para saber si se continúa con el flujo, de fallar nuevamente el ítem será retenido y el bug no se podrá cerrar en Azure DevOps hasta

ser corregido. no hay un límite Kanban presente dado que es una tarea que depende de desarrollo principalmente y solo se espera poder hacer el re-test para saber cómo continuar.

Done: en esta etapa las HU se encuentran probadas y sin bugs hallados, de estas funcionalidades se deben seleccionar algunos CP para conformar un conjunto de pruebas de regresión, con el fin de validar constantemente que no se presenten fallos con nuevas implementaciones y versiones del sistema. No hay límite Kanban y son candidatas a una posible automatización usando Cypress con pruebas E2E en el proceso de regresión.

Closed: finalmente las funcionalidades que llegan a esta etapa pasarán a revisión con cliente para pruebas de aceptación, no tiene límite Kanban.

La arquitectura de P2H se encuentra en la **Figura 4,** donde se aprecia que se encuentra desplegado en la nube de Azure, todo por medio de contenedores de Docker, se cuenta con un backoffice para la gestión y manejo de interno como administrador y una parte para el paciente, ambos separados en el front en diferentes contenedores, pero bajo el mismo framework de vue.js, mientras que el back-end si es uno solo y está en node js, la base de datos está en una MV con SQL server.

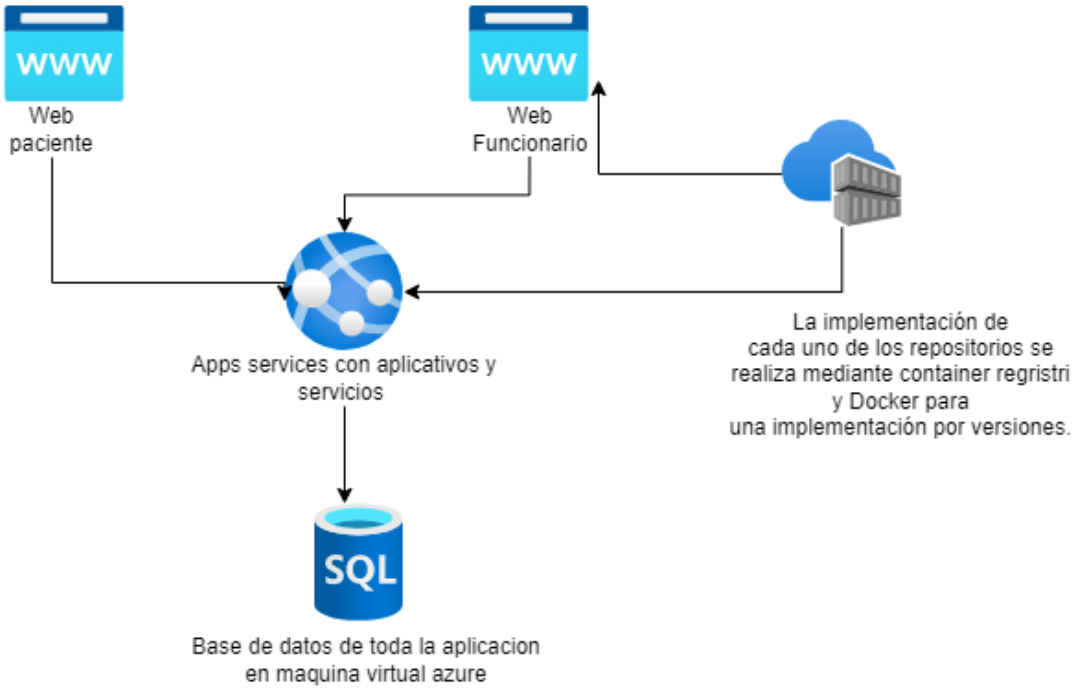

Fig. 4 Arquitectura P2H

VI RESULTADOS

Se hicieron pruebas a funcionalidades de P2H como: tipificación de usuarios, registro de usuarios, login, agendamiento cita reclamación. solicitud domicilio, pedidos manuales, pedidos por procesar y cargue de órdenes masivo, dentro de las cuales se han revisado requerimientos funcionales y no funcionales, añadido flujos omitidos en un principio o consideraciones de seguridad, en el diseño revisado la usabilidad y responsive de las vistas. También hizo pruebas funcionales y asociadas a cambios, para las funcionalidades nuevas y ya existentes, estas últimas entran como deuda técnica de pruebas, adicionalmente se realizó acompañamiento en otros proyectos nuevos de la compañía como lo es un PRM (Patient Relationship Management), donde se planea integrar todas las soluciones brindadas por la compañía, finalmente se comenzó con la automatización de algunas pruebas.

Los CP presentados a continuación en el documento corresponden al login, cuya vista se puede observar en la **Figura 5**, funcionalidad de la cual es posible mostrar información no sensible, además de la complejidad del flujo la hace lo suficientemente corta para presentarla, también es por esto que cumple con las características de una prueba candidata a automatización. La vista solo está compuesta por 2 campos, número de identificación y contraseña, los flujos presentes están descritos en la HU que se ve en la **Figura 6**, los cuales corresponden al flujo normal, y dos flujos de error, uno por usuario no registrado y el otro por campo faltante.

Fig. 5 Vista login P2H

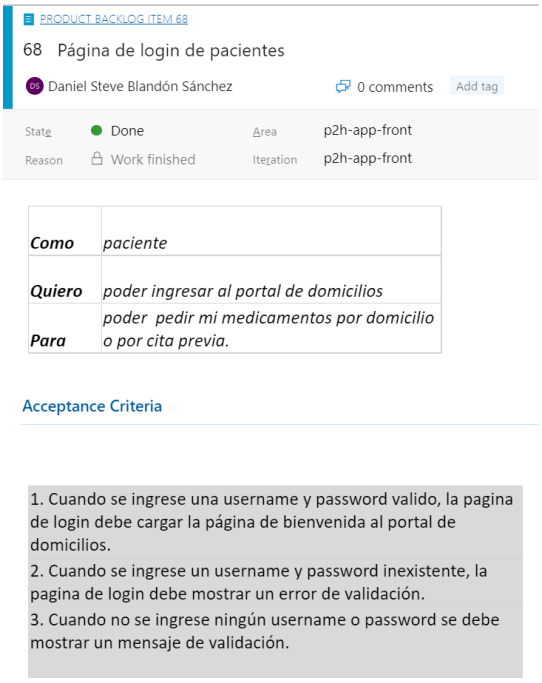

Fig. 6 Historia de usuario del login en Azure DevOps

Los CP se construyen a partir de las HU y son una forma de documentar y dejar la evidencia de las pruebas realizadas, la estructura implementada consistió en dos secciones, una de detalle que describe todo lo del CP y otra de resultados para la evidencia de su ejecución. para la primera parte contamos con un identificador, un resumen de que se desea realizar, una prioridad para definir su importancia e impacto, las precondiciones que definen todo aquello que debe ser considerado antes de ejecutar el caso de prueba, los datos de entrada que contemplan variables a usar, el paso a paso del proceso y un resultado esperado de la ejecución; mientras que en los resultados tenemos el resultado obtenido, el estado que nos indica si hay alguna inconsistencia entre el resultado esperado y el obtenido, el ambiente/versión para saber en cual versión y bajo qué sistema fue ejecutada la prueba y una fecha de ejecución para llevar una trazabilidad. La documentación de estos CP es llevada en Excel y se puede apreciar en la **Figura 7** y **Figura 8**, luego es implementada en Azure DevOps cuyos pasos se ven en la **Figura 9** y el resumen en la **Figura 10**, se ha mejorado y detallado los CP, actualmente se hace su resumen utilizando una sintaxis Gherkin para tratar de ser más claro en cada CP, los bugs se reportan y documentan también en Azure como se muestra en la **Figura 11,** adicionalmente se hace uso de la lista de chequeo para tener la trazabilidad de estos; dicha lista

está en Excel y es compartida con todos los miembros del equipo y se puede ver un fragmento en la **Figura 12**.

ID	Resumen	Prioridad	Precondiciones	Datos de entrada	Pasos	Resultado Esperado	Resultado Obtenido	Estado	Ambiente/versión	Fecha ejecución
CP001	Historial de citas busqueda por fecha	Media	*Usuario registrado y previamente logueado *BD con registros de citas del Usuario	*Fecha_inicial *Fecha_final	*Ingresar a la url o hacer click en 'Mis citas' *Ingresar los campos de busqueda por fecha *Click en el botón	Listar las citas en la tabla debajo que cumplen con la condicion de que su fecha se encuentre entre "Fecha_inici	img	OK		

Fig. 7 Caso de prueba en Excel.

Resumen	Prioridad	Precondiciones	Datos de entrada	Pasos	Resultado Esperado
Scenario: Iniciar Sesión en la plataforma empleando un Numero de documento y Contraseña correctos	Media	Usuario y Contraseña inscritos a través del proceso de registro y auprobado desde el Módulo Administrativo (BackOffice)	*User* : *Password* :	Ingresar a la url:	
Given: Un usuario se dirige a la página de Login y llena los campos con su número de documento y contraseña				Escribir en el campo del numero documento *User*	Se debe mostrar el *numero documento* en el campo de numero de documento
When: El usuario clica sobre el botón Iniciar Sesión				Escribir en el campo de la contraseña *Password*	la *Password* ingresada debe mostrarse de forma oculta y contar con la opción que el usuario la vea
Then: El usuario puede ver la pantalla de Home				Hacer click en el boton iniciar Sesión	vista Home

Fig. 8 Mejora caso de prueba en Excel con Gherkin.

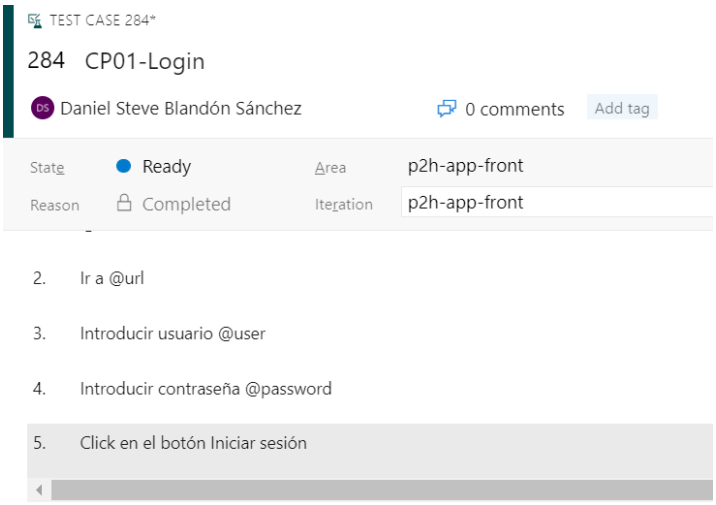

Fig. 9 Pasos caso de prueba en Azure DevOps.

Fig. 10 Resumen caso de prueba en Azure DevOps.

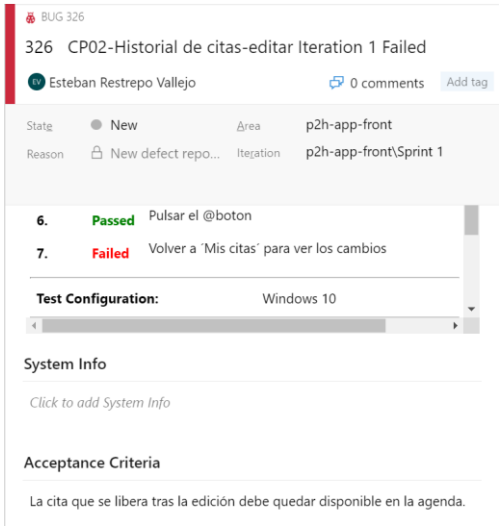

Fig. 11 Bug en Azure DevOps.

Fig. 11

#	document o V	Descripción /HU/BUG	Tipo	Pruebas - Daniel -Juliái		Realizado QA - Johanna - Daniel	
				Check	Comentarios	Check	Comentarios
23	Nuevo	Edición cita reclamación: - Cuando se **edita** una **cita**, la **cita anterior** debe quedar **disponible** para su asignacion nuevamente.(se soluciono cuando se cancela, hace falta al editar)	Back - Fron	Pdte		Pdte	

Fig. 12 Check list en Excel.

Finalmente, para la parte de automatización se utilizó la herramienta de Cypress, la Figura 13 se muestra la interfaz gráfica que provee, en Cypress se puede automatizar pruebas E2E, esto es muy

útil no solo por probar de principio a fin un flujo, sino también porque nos permite agilizar los procesos de retesting y regresión, que conforme crecen los proyectos crecen las pruebas y toma más tiempo poder validar todo un conjunto de pruebas.

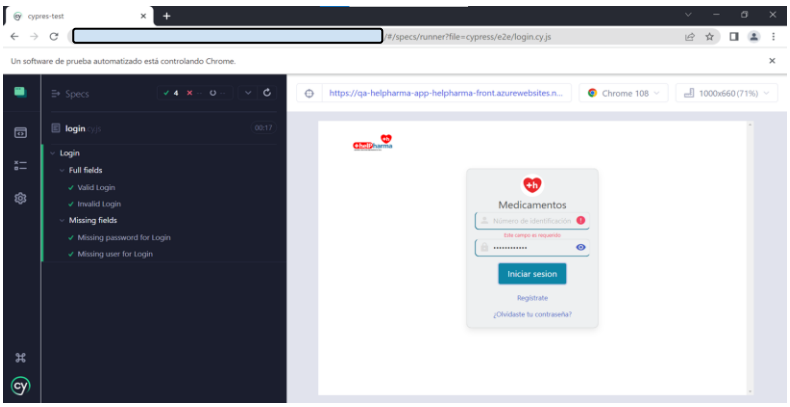

Fig. 13 Interfaz gráfica Cypress.

Un ejemplo de una prueba que se puede automatizar simplemente es un login, siempre se visita la misma página, en la **Figura 14** vemos la función que se ejecuta antes de cada prueba, en la cual solo se tiene una instrucción para visitar una ruta especifica, luego se puede crear la prueba como en la que se validan errores de campos faltantes, en la **Figura 15** se ven algunos métodos como que tenga un valor determinado o sea visible y contenga cierto mensaje particular. También se tiene otra pueda donde se valida una correcta redirección tras un login exitoso, el cual es representado en la **Figura 16**, con la línea donde dice que la URL ahora debe incluir cierta ruta. Aunque hay más consideraciones este es un ejemplo de lo que nos podemos ahorrar con la automatización, ya que podemos elegir y controlar el entorno y navegador sobre el cual se ejecuta nuestra prueba, y hacer seguimiento paso a paso lo cual nos sirve a su vez para hacer un debugging en caso de ser necesario.

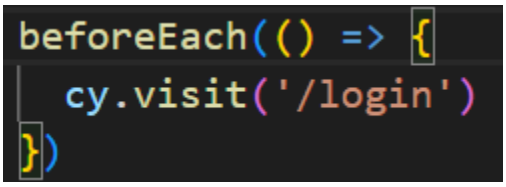

Fig. 14 Cypress beforeEach.

```
it('Missing password for Login', () => {
  cy.get('input').eq(0).type('[redacted]').should('have.value', '[redacted]')
  cy.get('button').eq(0).click()
  cy.get(':nth-child(2) > .field > .help').should('be.visible').and('contain','Este campo es requerido')
})
```

Fig. 15 Cypress campos faltantes.

```
it('Valid Login', () => {
  cy.get('input').eq(0).type('[redacted]').should('have.value', '[redacted]')
  cy.get('input').eq(1).type('[redacted]').should('have.value', '[redacted]')
  cy.get('button').eq(0).click()
  cy.url().should('include', '/home')
})
```

Fig. 16 Cypress redirección.

El tiempo que toma ejecutar las pruebas del login va alrededor de 15-20 segundos(en la Figura 13 se puede ver el tiempo que tardo el Script y es de 17 segundos), siendo 4 pruebas diferentes, mientras que estas pruebas manuales pueden tomar cada una entre 1-2 minutos aproximadamente, siendo un total mínimo de 4 minutos y máximo de 8 minutos, con un promedio de ejecución de 6 minutos para las pruebas manuales y de 17 segundos en pruebas automatizadas tenemos una relación de $\frac{17}{360}$ lo que equivale a 0,047 que en porcentaje es 4,7% del tiempo total, es decir las pruebas automatizadas del login se ejecutan en un 95,3% menos del tiempo total, este porcentaje de tiempo en segundos puede ser poco para considerar, pero cuando el proyecto es robusto y se deben probar muchas cosas un ahorro del tiempo del 95% será muy significativo.

Durante este periodo de práctica se reportaron y corrigieron fallos, algunos leves tales como errores del diseño, pero otros más significativos como errores en las peticiones, mal funcionamiento de un proceso o hasta flujos no contemplados, estos fallos fueron detectados en etapas tempranas del desarrollo lo cual fue de gran importancia ya que pudo evitar costos en tiempo con reprocesos.

VII. CONCLUSIONES

- Las pruebas son un proceso necesario en el desarrollo de software, ayuda a detectar errores en etapas tempranas del desarrollo, optimizando los costos que implica hacer correcciones en otras etapas.
- Las pruebas y los ciclos de entrega temprana ayudan a mantener un desarrollo ágil, además de darnos una ventaja en tiempo para revisar y corregir los hallazgos
- Hacer la entrega de funcionalidades al cliente, apoyado con una serie de pruebas de aceptación provee mayor confianza y seguridad de que se desarrolló el producto correcto.
- La comunicación con el área de desarrollo es vital, dado que de esta depende un buen flujo de trabajo para ambas partes.
- Las herramientas provisionadas por Azure son muy completas, ayudando al manejo del trabajo en equipo y la trazabilidad.
- La experiencia en el diseño de CP hace el proceso más ágil, intuitivo y completo.
- La automatización de pruebas es una herramienta muy útil conforme aumentan las funcionalidades y las pruebas, dado que ahorra tiempo en algunas pruebas manuales y al repetir pruebas ya sea por regresión o comprobación es más optimo en tiempo.

REFERENCIAS

[1] R. S. Pressman, B. R. Maxim, and Antonio Medellín Serna Luis, Ingeniería del software: Un Enfoque práctico. México etc.: McGraw-Hill, 2021.

[2] IEEE Standard for System and Software Verification and Validation, IEEE Standard 1012, 2012.

[3] ISO/IEC/IEEE Software engineering — Guidelines for the application of ISO 9001:2015 to computer software, ISO/IEC/IEEE Standard 90003:2018

[4] ISTQB, "ISTQB Glossary," ISTQB. [Online]. Available: https://glossary.istqb.org/en/search/.

[5] B. Moreno, "Curso de Fundamentos de Pruebas de software," Platzi [Online]. Available: https://platzi.com/cursos/pruebas-software/.

[6] G. Hernandez Sola, "¿La integración continua está muerta?," Scrum.org, 15-Oct-2022. [Online]. Available: https://www.scrum.org/resources/blog/la-integracion-continua-esta-muerta.

[7] J. M. Sánchez Peño, "Pruebas de software. Fundamentos y Técnicas," [trabajos de grado]. Madrid (España): Universidad Politécnica de Madrid, 2015. [Online]. Available: https://oa.upm.es/40012/.

[8] J. Fuentes Mora, "Curso de Introducción a la automatización de pruebas," Platzi [Online]. Available: https://platzi.com/cursos/automatizacion-pruebas/.

[9] M. Iqbal, "How scrum with kanban works," Scrum.org, 18-Jul-2022. [Online]. Available: https://www.scrum.org/resources/blog/how-scrum-kanban-works.