UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Servicio de consultoría para una empresa de software en el área de desarrollo de aplicaciones

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniera Mecatrónica

P R E S E N T A

Karen Gabriela Gutiérrez Felipe

ASESOR DE INFORME

Dr. Edmundo Gabriel Rocha Cózatl

Ciudad Universitaria, Cd. Mx., 2019

Agradecimientos

Este trabajo refleja el fruto de un esfuerzo constante a lo largo de mi vida, el cual no sería el mismo sin el apoyo de mi familia, amigos, maestros y colegas.

Quiero agradecer a mi alma máter, la Universidad Nacional Autónoma de México, por brindarme sus aulas para estudiar una ingeniería que me cautivó desde el primer día, por concederme maestros tan excepcionales que se caracterizan no sólo por tener excelentes conocimientos en su área, sino porque enseñan con una pasión que se contagia en cada clase y que además poseen una gran calidad humana. A ellos, gracias.

Gracias a mi asesor de informe, el Dr. Edmundo Rocha, por su amable disposición de guiarme en la elaboración de este informe, y por ser un excelente profesor.

Agradezco a mi tutor de carrera y maestro, Yair Bautista, por escucharme y aconsejarme en el ámbito académico y profesional y a mis sinodales, por su contribución de mejoras y correcciones a este reporte.

Agradezco a mis padres, Magdalena y Ricardo, por estar presentes en todas las etapas de mi vida brindándome amor y apoyo, por enseñarme los valores que ahora tengo y que me permitieron llegar hasta aquí de una manera íntegra. Gracias por ser un ejemplo de vida admirable, por preocuparse siempre por mí, por darme consejos, palabras de aliento y confiar en mis sueños. Quiero que sepan que este logro también es suyo y que valoro mucho todo el esfuerzo y sacrificios que hicieron para que yo lo lograra.

Agradezco a mis hermanos, Paola y Ricardo, por su apoyo incondicional, sus palabras de aliento y por compartir experiencias que me han hecho crecer en todos los sentidos de la vida. Gracias por hacerme ver el lado divertido de la vida.

Gracias a mis abuelitas y abuelitos por confiar en mí y alentarme con su amor durante mi formación.

Gracias Jaime, por apoyarme y aconsejarme cuando el camino no parece claro, por explicarme aquellos temas de la carrera que no lograba comprender tan fácilmente y por creer en mí en todo momento.

A mis amigos de la Facultad, por las experiencias enriquecedoras que vivimos a lo largo de la carrera. Gracias Yael, Magui, Alan y Sergio.

Un agradecimiento para Microsoft por abrirme las puertas al ambiente laboral, por invertir en mi aprendizaje y darme la oportunidad de crecer a pasos agigantados en el mundo del software. Gracias a Marcelo, Jordan y Edna por guiarme durante mis primeros proyectos.

Índice

Introducción y Objetivo

En este informe se presentan en detalle las actividades que, como consultora en el dominio de aplicaciones de software, desempeñé durante más de 6 meses para Microsoft, una empresa ampliamente reconocida a nivel mundial por su alto grado de desarrollo tecnológico en materia de software.

El principal objetivo de este informe es describir las actividades del servicio de consultoría para implementar y asegurar la calidad de los productos que se entregan al cliente y las tecnologías usadas para este fin.

Se hace énfasis en el proceso de desarrollo y las etapas del proyecto, para algunos proyectos del sector público en el área de administración tributaria en los que tuve la oportunidad de contribuir.

El servicio de consultoría en Microsoft se divide por dominios según el tipo de tecnologías de Microsoft que se implementan en un proyecto de software, por lo cual se tienen consultores para cada área, ya sea de análisis de datos, infraestructura de software, inteligencia artificial, productividad en negocios, desarrollo de aplicaciones, etc.

En mi área (*Modern Apps Domain*) se implementan soluciones personalizadas a los clientes según las necesidades de su industria y su mercado. Entre las principales actividades de mi rol se encuentran las de entender las necesidades del cliente para transformarlas en requerimientos técnicos que, posteriormente, se utilizan para desarrollar aplicaciones web o móviles que satisfagan y solucionen las necesidades del cliente.

Ejecutando estas actividades he podido poner en práctica los conocimientos y herramientas que adquirí en la Facultad de Ingeniería, principalmente en las áreas de programación, metodologías y análisis de información, así como las habilidades que desarrollé trabajando en equipos multidisciplinarios y la capacidad de aprender continuamente de forma autodidacta.

I. Antecedentes

Microsoft, fundada en 1975[1], es una empresa líder en el desarrollo de software de uso personal y empresarial, dispositivos electrónicos y servicios. Algunos de los productos sobresalientes de esta compañía son su sistema Operativo Windows, la paquetería de aplicaciones de escritorio Microsoft Office, sus navegadores Internet Explorer y Edge, su línea de videojuegos Xbox y recientemente su línea de computadores personales Surface y dispositivos diversos de hardware.

Algunos de estos productos, si no es que la mayoría, ahora son parte de nuestra vida diaria y nos facilitan el trabajo diario, nos proveen entretenimiento y nos proporcionan alternativas de comunicación personal y profesional. Haciendo uso del internet de las cosas, la inteligencia artificial y la nube, esta empresa está proveyendo soluciones a problemas reales que se presentan en nuestro mundo.

I.I. Organización del negocio en Microsoft

La organización de negocio en Microsoft se vuelve compleja si se toma en cuenta que opera en 120 países del mundo, proveyendo servicios de nube y locales, servicios de entretenimiento, inteligencia artificial, servicios personalizados, etc.

De acuerdo a la página oficial de Microsoft[2], la compañía está organizada de manera general en dos categorías, Grupos de Ingeniería y Funciones de Negocio, la primera trabaja específicamente en la investigación y desarrollo de los productos bajo la marca Microsoft, mientras que la segunda se encarga de su distribución y asuntos de negocio propios de la empresa.

En la Figura 1, se muestran las áreas que corresponden a estas categorías. Dentro de las Funciones de Negocio se encuentra la Organización Mundial de Negocios, que incluye empresas del sector privado, sector público, pequeños y medianos mercados, servicios, desarrolladores y socios.

Dentro del área de Servicios se encuentra la unidad de Servicios de Consultoría de Microsoft (MCS), que es a la que pertenece mi rol.

Con el área de Servicios, Microsoft, va más allá proveyendo soluciones personalizadas a las empresas, tanto públicas como privadas, con el fin de que puedan desarrollarse al máximo haciendo uso de las nuevas tecnologías. Tal como lo indica la misión que tiene la empresa[3]:

“Empoderar a cada persona y organización en el mundo para hacer más”

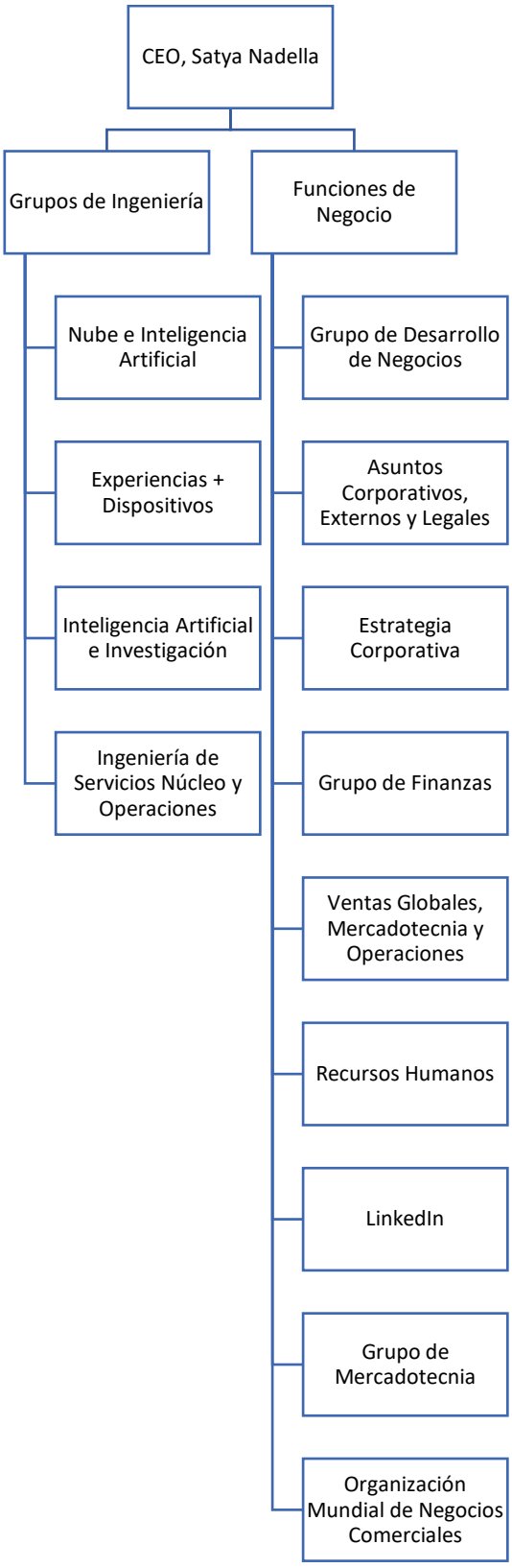


Figura 1. Diagrama de Organización del Negocio en Microsoft.

I.II. El rol del consultor en Microsoft

De acuerdo a la página oficial de carreras en Microsoft[4], el consultor en el dominio de aplicaciones modernas trabaja en la entrega de aplicaciones que ayudan a organizaciones a interactuar con los clientes a través de múltiples canales y en cualquier dispositivo, el rol requiere un conocimiento tecnológico amplio y profundo y la capacidad de diseñar y desarrollar soluciones integradas utilizando los últimos productos y tecnologías Microsoft.

Responsabilidades:

- Trabajar con clientes y socios de entrega para ayudar a definir y entregar soluciones de aplicaciones modernas de vanguardia
- Impulsar actividades para implementar la solución que proporciona el valor comercial esperado para el cliente y llevar el proyecto a su objetivo de manera efectiva (visualizar, planificar, desarrollar, estabilizar e implementar la solución).
- Garantizar la calidad de los productos propios y de los socios.
- Actuar como asesor de soluciones de confianza para clientes en diversos compromisos que pueden variar desde una breve tarea de consultoría hasta un gran proyecto de implementación.
- Actuar como experto técnico para cuestiones amplias y complejas.
- Trabajar de manera independiente y como miembro de un equipo ágil.
- Impacto en el negocio a través de la excelencia y previsibilidad de la entrega, la creación y reutilización de propiedad intelectual y el intercambio de experiencias
- Analizar y proporcionar información estratégica sobre los riesgos, beneficios y oportunidades de varias soluciones según sea necesario.
- Trabajar de manera proactiva con el equipo de compromiso y los socios y buscar oportunidades de venta ascendente para Microsoft

Conjunto de habilidades técnicas que requiere el puesto

- Experiencia en el uso de aplicaciones de Microsoft.
- Experiencia en la industria de tecnologías de la información, trabajando con desarrollo de software, entrega de soluciones.
- Tener conocimiento de tecnología en la nube como Azure Office 365 o Dynamics
- Lenguajes de desarrollo, por ejemplo, C #, VB.NET, JavaScript, HTML5,
- Conocimiento en patrones de desarrollo que utilizan tecnologías C # .NET hasta la última versión (por ejemplo, WCF, WF, MVC);
- Conocimiento en Metodologías y gestión del ciclo de vida de la aplicación, como Agile, Scrum o TFS.
- Conocimiento de DevOps

Conjunto de habilidades blandas que requiere el puesto

- Capacidad de resolución de problemas comerciales y tener la facilidad de explicar beneficios al cliente desde el valor comercial hasta aspectos profundamente técnicos.
- Pasión por trabajar con el cliente
- Pasión por aprender
- Ser proactivo, colaborativo y seguro bajo presión.
- Liderazgo técnico de equipos de desarrollo

II. Contexto de la participación profesional

Microsoft ha trabajado desde el 2013, con una entidad de administración de impuestos del gobierno Mexicano, en un sistema completamente automatizado para el cobro y administración de los impuestos recaudados en el país[5].

A través de los servicios de Azure, Microsoft ha desarrollado un portal web mediante el cual esta entidad (que para fines prácticos llamaremos Empresa 1) promueve el cumplimiento voluntario de los contribuyentes a través de procesos simples. Ahora los contribuyentes mexicanos pueden presentar su declaración de impuestos de manera rápida y confiable.

Los cambios en la legislación tributaria y la forma de cobro de impuestos que evoluciona diariamente exigen que el sistema de recaudación también sea renovado cada periodo fiscal con el fin de mantenerlo actualizado y que el cobro de impuestos sea cada vez más rápido y seguro.

Detrás de estas actualizaciones y mejoras del sistema hay todo un equipo de desarrollo que se encarga de mantener el control en el procesamiento de la información fiscal de cada contribuyente. He tenido la oportunidad de trabajar con el equipo de consultores que se encarga de desarrollar estas mejoras e implementar nuevas funcionalidades al portal de empleados[1] para ejercicio fiscal 2018.

Dentro del sector público y área tributaria, Microsoft también ha implementado un sistema de facturación en la nube para procesar los comprobantes fiscales de los contribuyentes peruanos. En este proyecto que se lleva a cabo con una segunda institución de administración de impuestos (Empresa 2), he trabajado en el área de pruebas y aseguramiento de calidad de la solución.

En el entorno de operación de los organismos de tributación, siempre ha sido de suma importancia la presentación de las declaraciones de impuestos de los contribuyentes ya que estas se convierten en una herramienta de fiscalización que permite a la autoridad supervisar a las empresas que cumplen sus obligaciones fiscales. Por lo cual es necesario se lleven a cabo los ajustes necesarios al flujo y funcionales del aplicativo de la declaración anual de personas físicas y morales, integrando las reformas económicas que se llevan a cabo en los países.

Algunos de estos cambios, de manera general, pueden ser:

- Integrar las nuevas reglas de negocio en los formularios que se presentan en el portal de declaraciones.
- Incorporar mensajes o formularios[2] que mejoran la experiencia con el usuario

[1] Sitio web publicado para dar a los empleados de ventanilla de las oficinas locales una herramienta para consultar y generar declaraciones en apoyo al contribuyente.

[2] Plantilla de llenado y calculo que un contribuyente llena al presentar una declaración, cada formulario está asociado a un régimen en particular

- Generar archivos de consultas, constancias de pagos o copias certificadas de documentos fiscales.
- Actualizar información relevante para el negocio del cliente en su propio portal.
- Validar archivos recibidos y canalizarlos al sistema para que puedan ser procesados, etc.

Llegando a este punto y por cuestiones confidencialidad es importante mencionar que no tocaré con más profundidad las necesidades del cliente ni los requerimientos técnicos que forman parte de la lógica de negocio de los proyectos y se hará énfasis sólo en el papel que ejecuté como consultora en estos casos.

Para realizar los cambios anteriormente mencionados y gestionar nuevas peticiones en los proyectos, seguimos un proceso para asegurar la calidad en la entrega del producto. Este proceso es mejor conocido como ciclo de vida del proyecto, el cual se ilustra en la Figura 2.

A continuación, explicaré de forma general las fases de un proyecto en el que se desarrolla un aplicativo para un cliente en el área de consultoría, así como las partes que intervienen en cada fase.

Figura 2. Diagrama de fases de un proyecto de aplicaciones.

Descripción de las etapas del proyecto:

Visión y alcance:

Durante esta etapa se realizan reuniones para estructurar o alinear el enfoque del negocio y definir el alcance del proyecto que hará la realidad la visión.

También se elabora una carta de inicio del proyecto que establecerá formalmente el inicio de actividades y es firmada por ambas partes.

En esta etapa generalmente está involucrado el gerente del proyecto, arquitecto de software, personal de ventas y en ocasiones consultores experimentados de parte del proveedor y los gerentes y representantes de parte del cliente.

Planeación y diseño

En este punto se establece el diseño conceptual de la solución mediante diagramas y definiciones de negocio, se establece la arquitectura del aplicativo tomando en cuenta las tecnologías con las que el sistema va a interactuar, seguridad, disponibilidad y escalabilidad.

- Para los proyectos que utilizan servicios en la nube, se hace un análisis del consumo y licenciamiento de estos productos, tanto en la etapa de desarrollo como en la de producción.
- Se analiza el espacio de almacenamiento que será necesario durante el desarrollo y producción, así como el modelo y flujo de almacenamiento de datos.
- Se representa también el flujo de interacción entre los distintos componentes tecnológicos que conforman la solución, la división por capas, los protocolos involucrados y los balances.

Se analizan riesgos asociados al proyecto.

Se realiza un plan de trabajo detallado y durante todo el proyecto se da seguimiento al plan original y se incorporan cambios.

Esta etapa está directamente relacionada al trabajo del gerente de proyecto y al arquitecto de proyecto.

Construcción

Se desarrolla la solución de acuerdo con los diagramas que se generaron previamente en la arquitectura. Se habilitan ambientes integrados en la nube para el desarrollo y el trabajo en equipo de la solución.

En esta fase interviene el esfuerzo de los consultores, quienes trabajamos directamente con el código fuente y termina con las pruebas de aceptación UAT por sus siglas en inglés (*User Acceptance Testing*), que indican que ya no existen errores que impliquen cambios en el código.

Durante esta fase generalmente también se realiza documentación sobre la solución, como:

- Manuales de instalación
- Manual de compilación
- Diagramas de diseño de componentes de la solución (clases y servicios) y diagramas de secuencia, entre otros.

Pruebas y estabilización

En esta fase, los consultores y/o personal de calidad aseguramos el correcto funcionamiento del entregable de acuerdo con las especificaciones establecidas previamente, mediante:

- Ejecución de pruebas funcionales e integración
- Pruebas UAT
- Pruebas de vulnerabilidad
- Pruebas de estrés

Durante esta etapa, continuamos con el desarrollo y se corrigen errores de código y diseño.

La definición y ejecución de estas pruebas se explicará más adelante.

Despliegue e implementación

Se preparan y se envían paquetes al ambiente de producción, que es el ambiente en el que la aplicación estará funcionando y tendrá contacto directo con el usuario final.

También se hace entrega de la documentación de la solución al equipo de producción, que es el encargado de dar mantenimiento al aplicativo y corregir posibles errores que no fueron identificados durante el desarrollo.

En esta etapa está involucrado el equipo de producción y algunos miembros clave del equipo de desarrollo que tienen conocimiento del código y cómo se relacionan sus componentes. Es común, que los consultores resolvamos ciertas inconsistencias durante este proceso.

Para que se pueda dar la transición entre un sistema implementado y el nuevo, generalmente se inhabilita la página web por unas horas en lo que se despliega la solución en la web, es decir se hace la transición.

Cierre

Finalmente se valida que no haya problemas severos en el ambiente productivo. Con lo cual el cliente firma la carta de aceptación del proyecto, se entrega la documentación requerida y posteriormente se firma el cierre del proyecto.

Esta etapa corresponde a los gerentes de proyecto.

Estas etapas no son completamente secuenciales, ya que en ocasiones unas dependen de otras y son cíclicas entre ellas; como lo son el proceso de desarrollo, despliegue y pruebas que, en ciertos momentos del proyecto son dependientes unos de otros. En la Figura 3 se muestra un ejemplo de cronograma donde se integran estas actividades.

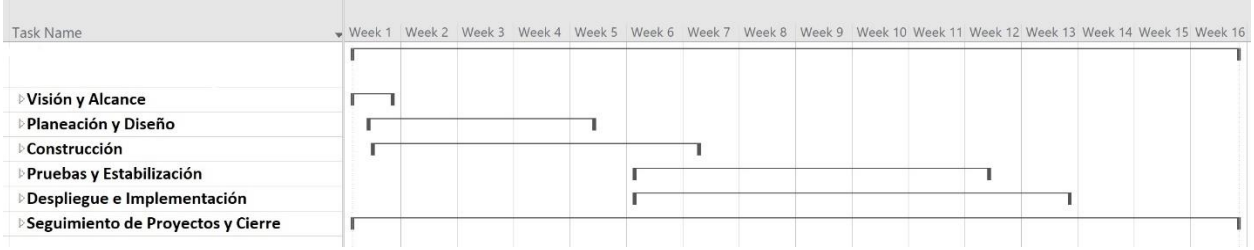


Figura 3. Ejemplo de cronograma de las fases de un proyecto.

Las actividades de consultoría están presentes en las diferentes etapas del proyecto y el trabajo en equipo es de relevante importancia para que éste sea exitoso en tiempo y forma.

En los dos proyectos en los que contribuí, interactué de maneras diferentes con las personas que están involucradas en los roles mencionados anteriormente. En los diagramas de Venn de las Figuras 4 y 5 se muestra mi relación como consultora con los diferentes miembros del equipo desde dos perspectivas diferentes.

Para el proyecto con la Empresa 1 el contacto que tuve fue más cercano al equipo de desarrollo, ya que mi participación fue más bien técnica y limitada a los miembros del equipo de Microsoft y colaboradores de otras empresas de consultoría. Con ellos la relación fue de intercambio de información entre los componentes del desarrollo, ya que la solución se trabajaba de manera paralela, es decir, los componentes en los que trabajábamos dependían unos de otros.

El líder del desarrollo al ser un consultor más experimentado es el que conoce mejor las reglas de negocio que aplican para cada componente. Es la persona que tiene más contacto con el cliente y al que recurría cuando una especificación técnica no estaba clara.

Otra de las personas clave con las que tuve interacción fue el agente de pruebas, quien, al detectar un error, característica o cambio en el funcionamiento de la aplicación web, lo canalizaba directamente a mí o al consultor que hubiera desarrollado esa parte de la solución.

La interacción con el arquitecto se limita a los problemas de compatibilidad en el desarrollo entre los componentes del sistema, pero en la mayoría de los casos era quien tomaba la decisión final sobre la mejor solución técnica.

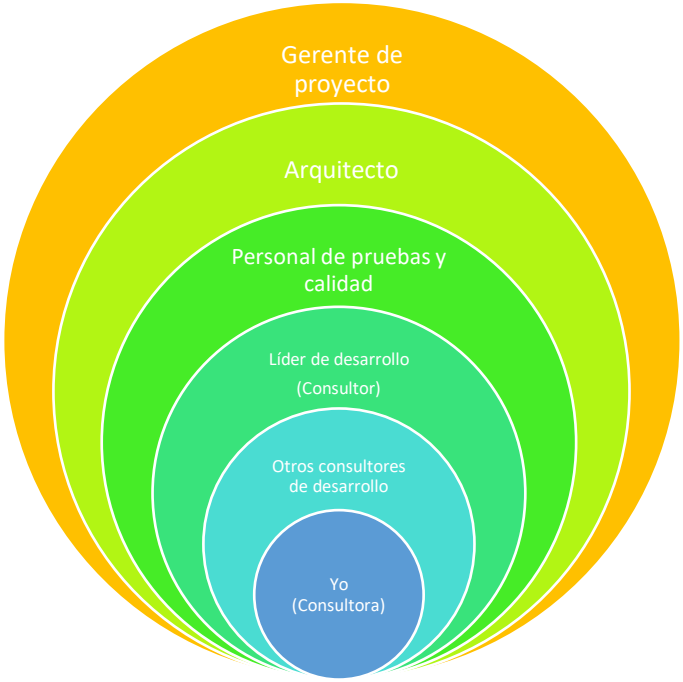


Figura 4. Diagrama de Venn que muestra el nivel de relación que mi puesto tuvo con los miembros del equipo durante el proyecto con la Empresa 1.

Para el proyecto con la Empresa 2 el escenario en el que me desenvolví fue completamente diferente, en este caso los alcances del aplicativo eran solamente a nivel de "*back end*", lo que significa que la solución que se desarrollaba tenía toda la lógica de negocio, pero no era visible para el usuario final. Debido a que la empresa mencionada tenía ya su propio portal web y se encargaba de gestionarlo, no era necesario que Microsoft desarrollara uno. Lo realmente importante e innovador para la Empresa 2 era tener su sistema de facturación y recaudación de impuestos funcionando en la nube, permitiéndose así liberar espacio físico de sus servidores, aumentar su capacidad de respuesta, y mantener su información más segura y disponible incluso en situaciones de contingencia.

Mi integración en este proyecto fue cubriendo el rol de "*tester*" realizando actividades desde el área de calidad y pruebas. En este caso la coordinación e interacción en equipo exigían mayor trabajo de los miembros debido a dos situaciones principalmente:

1. El proyecto se ubicaba en un país latinoamericano, donde por cuestiones administrativas, los consultores en el área de *Apps* (de ahora en adelante usaré este término para referirme al área que trabaja con Aplicaciones) eran escasos y se recurrió al equipo de consultores de "*Global Delivery*" que trabaja desde la India para desarrollar la solución e implementar en ella todos los requerimientos técnicos que pedía el cliente.
2. La dependencia tenía su propio equipo de pruebas que trabajaba de forma local, que se aseguraba de que se cumplían sus reglas de negocio al recibir comprobantes electrónicos.

Mi puesto interactuaba con ambas partes ya que entre mis actividades estaba la de encontrar errores de implementación en la solución y reportarlas al equipo de desarrollo, así como comprobar los bugs del cliente y traducirlo a lenguaje técnico en inglés para que el equipo de desarrollo corrigiera el error.

La interacción con el área de desarrollo siempre fue remota, mediante chat, videollamadas, por correo o por notas e historias de caso que creaba utilizando Azure DevOps[6] (herramienta de Azure para facilitar el trabajo y operaciones en ambientes de desarrollo), sin embargo, era un método muy efectivo para la solución de errores que se presentaban.

Más adelante haré énfasis en el tipo de pruebas y actividades específicas que mantenía con ambos equipos, tanto del lado del cliente como del de Microsoft

En la Figura 5 muestro en un diagrama de burbujas el nivel de relación de mi puesto con otros miembros del equipo, donde la burbuja de "Otros consultores" representa el grupo de individuos con los que tuve mayor grado de relación y la burbuja de Gerente del proyecto fue la persona con la que interactué menos.

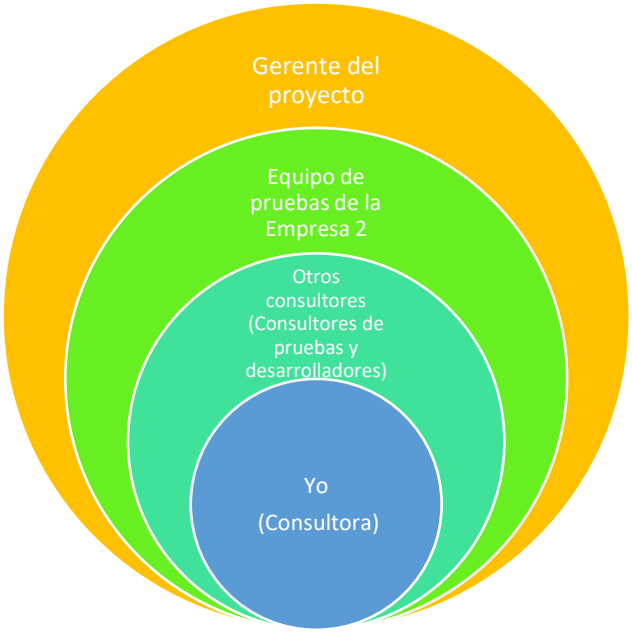


Figura 5. Diagrama de Venn que muestra el nivel de relación que mi puesto tuvo con los miembros del equipo durante el proyecto con la Empresa 2.

III. Metodología utilizada

III.I. Metodología usada en el área de desarrollo

Durante el tiempo que trabajé en el proyecto de la Empresa 1 me enfoqué en el área de desarrollo de aplicaciones web con el marco de ASP.NET.

El entorno .NET es una plataforma de desarrollo con herramientas, lenguajes de programación y bibliotecas que permiten construir diferentes tipos de aplicaciones[7], en este caso aplicaciones web con programación orientada a objetos.

Considero significativo mencionar que, durante mis estudios en las materias de programación (Computación para ingenieros, Técnicas de programación, Temas selectos de programación y Diseño mecatrónico) en la Facultad de Ingeniería, aprendí y trabajé en proyectos utilizando la plataforma .NET y desarrollé experiencia utilizando algunos de sus componentes:

- **Metodologías de programación orientada a objetos con el lenguaje de programación C#:** desde la definición del problema, identificación de objetos y clases, manejo de herencias, determinación de métodos, funciones, objetos dinámicos, constructores y operadores.
- **Metodologías de programación estructurada**
- **Librerías base** para trabajar con cadenas, fechas, archivos de entrada y salida, etc.

Así mismo, utilicé conocimientos de programación en HTML para hacer la interfaz gráfica de algunas páginas web durante mis estudios.

En este punto es importante especificar que, en informática, una de las buenas prácticas de programación es dividir el código en capas según su funcionalidad, entre más capas existan se vuelve más segura la aplicación, las principales capas son las siguientes:

- Capa de presentación o programación de **"*Front End*"** se utiliza para dar estilo y formato a la interfaz de la aplicación, puede ser estática con contenido fijo o dinámica en la que el usuario puede interactuar con la información, puede ver contenido multimedia, formularios, ingresar datos, desplazarse entre menús, etc., según sea el objetivo de la página. Esta capa se comunica con la capa de *Back End* donde se procesa información.
- Capa de negocio o programación de **"*Back End*"** se refiere al código en el que se encuentra toda la lógica de negocio de la aplicación, los cálculos, procesamiento de datos, interacción con otros servicios etc.
- Capa de **datos**, es la encargada del almacenamiento de datos, a esta capa le llegan solicitudes de parte del *back end* para extraer información, modificarla o almacenarla[8].

En la Figura 6 se observa el flujo que siguen estas capas y sus conexiones.

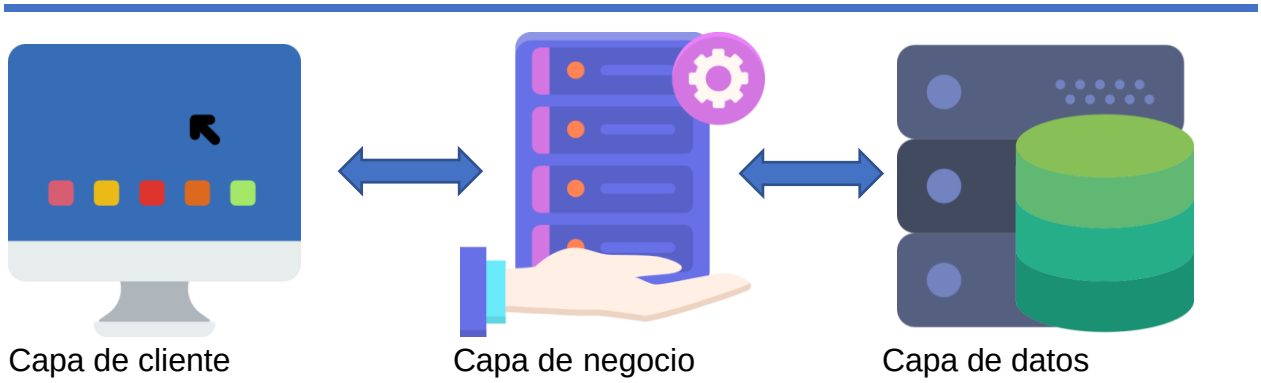


Figura 6 Capas de programación en una solución de software.

III .I.I. Métodos y arquitectura de desarrollo

La programación de "*Front end"* fue mi principal actividad durante este proyecto. Trabajé en la interfaz de algunos catálogos con opciones de borrado, modificación e inserción de parámetros.

Para ello, utilicé tres lenguajes comúnmente empleados para el desarrollo de *Front end* en aplicaciones web:

1. **HTML** (*HyperText Markup Language*) es un lenguaje de programación simple con base en etiquetas que se utiliza para estructurar el contenido de la página web. Las etiquetas son comandos en una página que son leídas por el navegador para ejecutar tareas como escribir el encabezado de la página, iniciar un nuevo párrafo, escribir el contenido de la página, etc. Estas etiquetas se escriben entre los signos (<) y (>)[9].

   En la Figura 7 se muestra como ejemplo el código de una página web simple con título "*Mi first Web Page*" y en el cuerpo de la página se muestra la frase *"Hello World!"*

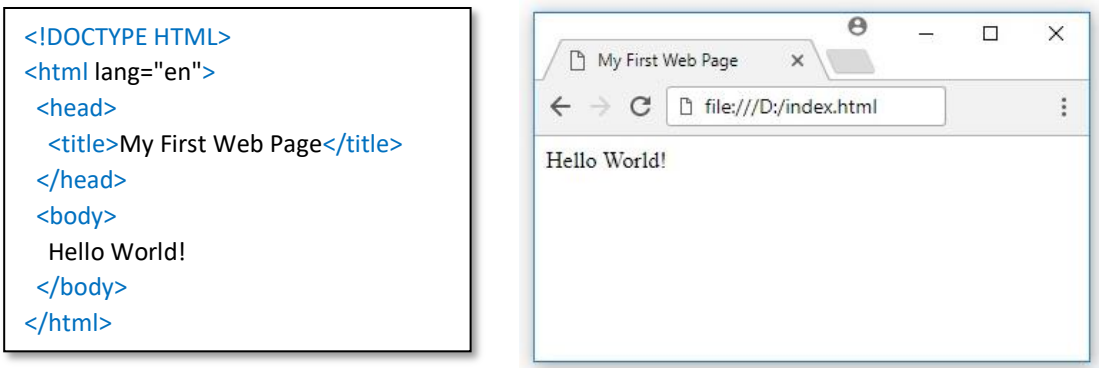


Figura 7. Ejemplo de una página web simple codificada en lenguaje HTML y su código fuente.

2. **CSS** (*Cascading Style Sheets*) u hojas de estilo en cascada en español, es un lenguaje diseñado para darle formato de estilo a la página web y lograr una buena presentación al usuario, definiendo colores, tamaños, fuentes de texto, etc.

   El código utilizando CSS consta de dos partes: un selector que indica qué tipo de elemento será afectado y las declaraciones donde se indica qué propiedades de ese elemento están siendo modificadas[10].

   En la Figura 8 se muestra un ejemplo básico de cómo mediante los comandos de CSS se puede tener control sobre la presentación de la página, en este ejemplo se trabaja específicamente sobre el título de la página y el color del fondo.

```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: green;}
h1 {color: white; text-align: center;}
p {font-family: arial; font-size: 20px;}
</style>
</head>
<body>
<h1>Mi primer ejemplo de CSS</h1>
<p>Esto es un párrafo dentro del
cuerpo de la página</p>
</body>
</html>
```

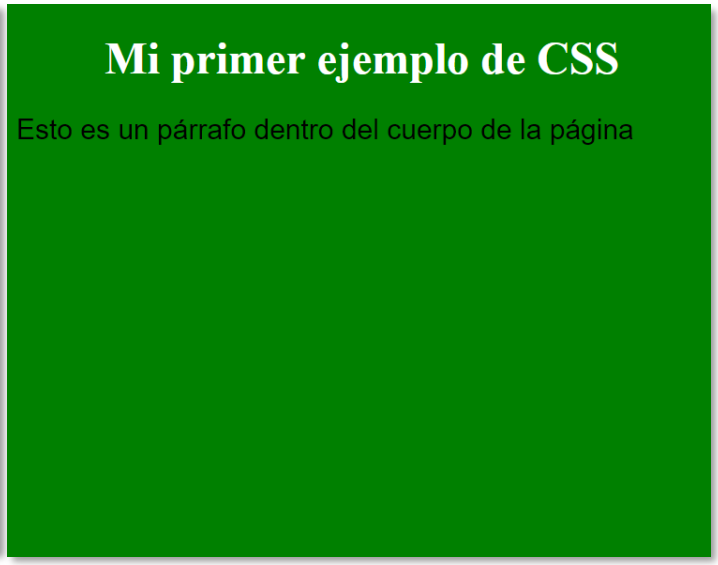


Figura 8. Ejemplo de una página web simple codificada en lenguaje CSS y HTML y su código fuente.

3. **JavaScript** (JS) este lenguaje de programación mejora la experiencia del usuario en páginas HTML, nos permite incluir elementos dinámicos y dotarlos de funcionalidad en nuestra página web. Elementos como botones, videos y animaciones, pueden crearse utilizando los comandos de JavaScript[11].

   En el código de la Figura 9 se crea un botón y se le añade la función de mostrar la fecha y hora cuando es presionado una vez.

```
<!DOCTYPE html>
<html>
<body>
<h2>Un botón en JavaScript</h2>
<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()">
Soy un botón, si haces click te mostraré la fecha y hora</button>
<p id="demo"></p>
</body>
</html>
```

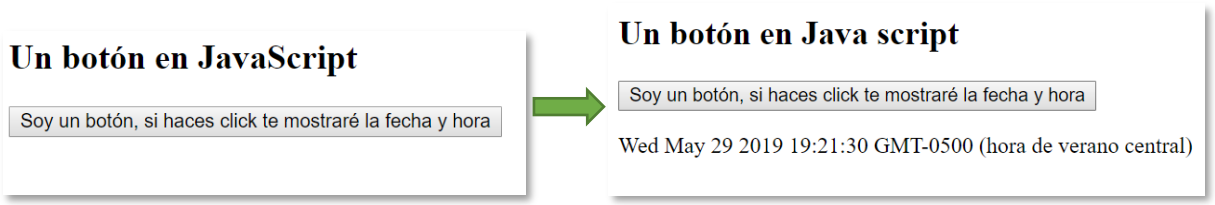


Figura 9. Ejemplo de una página web simple codificada en lenguaje JavaScript y HTML y su código fuente.

Cabe señalar que estos leguajes de programación se complementan para crear una experiencia amigable con el usuario y cada uno aporta una funcionalidad diferente a la interfaz. En esta tarea yo los utilicé en conjunto para desarrollar la interfaz gráfica de una serie de catálogos en forma de tablas que contenían información relevante para el cliente.

La información que contenían estos catálogos era procesada en el "*Back end*" y finalmente almacenada en bases de datos de SQL y en tablas de *Azure Storage*.

Dichos catálogos tenían las siguientes características:

- Contenían diferente cantidad de columnas y tipos de datos
- Algunos catálogos debían tener la opción de ser filtrados por el valor de alguna de las columnas.
- En la interfaz, se debía acceder a cada catálogo mediante un menú desplegable y solo se mostraban si el usuario tenía los permisos para verlos.
- Todos los catálogos debían permitir al usuario modificar registros, registrar nuevos, eliminarlos e inmediatamente ver los cambios en el portal web. Los cambios en los catálogos debían verse reflejados en ambas bases de datos (Tablas de Azure y SQL).
- Cuando se querían ingresar nuevos valores, se debía mostrar una ventana emergente con un formulario para que el usuario ingresara nuevos valores. En dicho formulario cada campo tenía sus propias validaciones para no permitir el ingreso de datos erróneos al sistema
- Cuando se quería editar un registro, se debía mostrar una ventana emergente con un formulario y algunos campos debían ser bloqueados para que el usuario no pudiera

editarlos. Al igual que en la ventana de inserción, aquí cada campo tenía sus reglas de validaciones

- En todo momento se mostraban cuadros de diálogo y ventanas emergentes que ayudaban al usuario a saber si la operación se realizaba correctamente o había un error con la información que se estaba ingresando.

Por la similitud de las funcionalidades que cada catálogo debía cumplir, opté por utilizar un patrón de arquitectura de software, que previamente había aprendido durante mi carrera en la materia de Técnicas de programación, llamado MCV[12] (*Model-View-Controller*) en ASP.NET que me fue de gran utilidad para generalizar las funcionalidades implementadas a los catálogos.

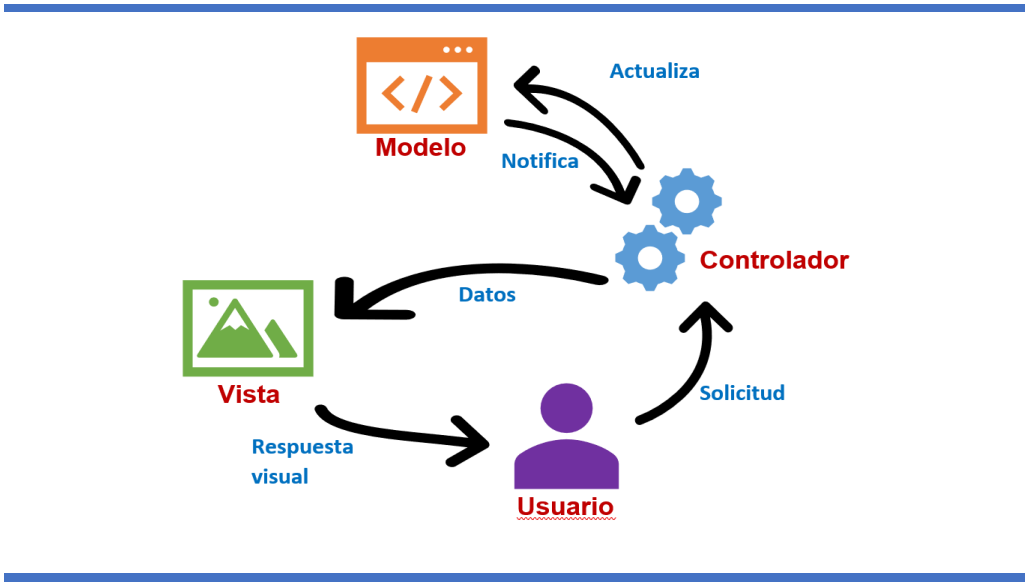


Figura 10. Diagrama de flujo del patrón de arquitectura MVC, Modelo-Vista-Controlador.

El patrón Modelo-Vista-Controlador, ilustrado en la Figura 10, es un estilo de programación en el que se separa la lógica de una aplicación en tres componentes distintos y cada uno tiene las siguientes características:

- Modelo
  - Representa toda lógica de negocio e implementación que se deba realizar.
  - Accede a la capa de almacenamiento de datos.
  - Mantiene comunicación directa con el controlador, el controlador le indica al modelo las operaciones que debe ejecutar y éste notifica el estado de la operación.

    - En ASP.NET se utilizan los archivos de tipo *ViewModel* para contener los datos que se utilizarán en la vista. Con los datos del modelo el controlador llena estas instancias.
- Vista
    - Contiene la estructura de la información que se muestra al cliente en la interfaz de usuario, así como los elementos con los que interactúa.
    - ASP.NET provee un motor de programación muy poderoso llamado Razor que permite incrustar código C# en HTML, habilitando el uso de lógica relativa a la presentación del contenido.
- Controlador
    - Es el encargado de manejar la interacción con el usuario, envía información de las solicitudes del usuario al modelo y luego actualiza la vista.
    - Elige el modelo o modelos con los que debe trabajar y también se encarga de elegir cuál vista es la que se va a presentar.

Para el caso específico de los catálogos utilicé este patrón del lado del *front end*, estandarizando un modelo con dos *scripts* principalmente; uno de ellos era usado por todos los catálogos y creaba el modal para inserción de nuevos registros, modal de edición y modal de eliminación de un registro, estos métodos funcionaban de manera genérica junto con un script donde se encontraba la configuración de cada catálogo El otro script era utilizado de manera genérica para generar la vista de los catálogos en el portal web.

Ambos scripts funcionaban en conjunto con un archivo de tipo controlador que recibía acciones solicitadas por el usuario gestionando los eventos que llegaban a través de "*handlers*" y accediendo al modelo para responder a las peticiones del usuario.

Para obtener la información requerida recurrí al uso de pequeñas aplicaciones APIs en la capa de *Back end* que se encargaban de hacer las funciones básicas que se esperaba que se pudieran hacer en los catálogos: crear, editar, eliminar, filtrar y obtener la información de las bases de datos.

Las peticiones de tipo http y de AJAX me permitieron realizar la conexión entre capas, llamando a las APIs y de esta manera habilitar el paso de información entre ellas mediante métodos de GET, POST, DELETE y PUT[3], los cuales se describen en la Tabla 1.

Método	
GET	Recupera información de un recurso almacenado
POST	Envía entidades específicas que se agregarán al recurso

[3] Métodos de petición HTTP para indicar la acción que se desea realizar para un recurso determinado

PUT	Reemplaza todos los campos de una entidad
DELETE	Borra entidades o recursos específicos.

Tabla 1. Métodos de petición HTTP.

La información que era enviada y recibida con los métodos mencionados debía ser consistente a todos los niveles de la programación, por lo que en cada parte del código fue necesario poner filtros y validaciones de la información que se ingresaba al sistema. El formato que me resultó muy útil para gestionar esta información fue el formato JSON[13] y los métodos para serializar y deserializar este tipo de archivos.

III. I. II. Integración de la solución

Participé activamente en el desarrollo de parte de *front end* de la solución. Sin embargo, para desplegar la solución completa en el ambiente de Desarrollo de Azure era necesario empaquetar tanto el código programado en *Front end* como el de *Back end* de todo el proyecto, para lo que fue necesario unir el código en el que varios consultores habíamos trabajado.

El reto al unir todas las partes de la solución es que el código se mantenga funcionando y sea consistente, evitando problemas por versionamiento diferente en los componentes o bibliotecas utilizadas y que los cambios hechos en algún documento no interfirieran con el funcionamiento de código de otros colegas.

Para esta tarea utilizamos GIT[14], una herramienta muy potente y confiable que se puede añadir a Visual Studio. Es un servicio para trabajo en equipo que funciona como versionador de archivos, de forma que siempre tendremos versiones de aquellos archivos que hayan sido modificados. Nos permite regresar a versiones anteriores en cualquier momento, comparar versiones, ver el autor de los cambios, así como la fecha y hora en que se produjo el cambio.

Esta herramienta, entre otras es compatible con el servicio de Azure DevOps, servicio anteriormente conocido como VSTS (por sus siglas en inglés, Visual Studio Team Services)[15].

Para aclarar cómo funciona un versionador, utilizaré el ejemplo de un desarrollador que trabaja en una página web, cuando el desarrollador termina el índice de la página guarda su progreso y crea una versión de su trabajo en ese punto, posteriormente se le pide añadir una página extra en la que debe agregar una descripción sobre un elemento del índice, cuando ha finalizado este trabajo, él guarda su trabajo y crea otra versión, finalmente le piden cambiar la interfaz gráfica de la segunda página que creó y de igual manera actualiza sus cambios, en cada nueva versión el desarrollador es cuidadoso y añade una pequeña descripción sobre los cambios que realizó en cada punto, de manera que si cometió alguna equivocación, el servidor falla o le piden hacer alguna corrección, el desarrollador puede ir al historial de versiones y elegir la versión desde la que desea trabajar o desplegar nuevamente al servidor. En la Figura 11 se ilustra el procesamiento de versionamiento del ejemplo anterior.

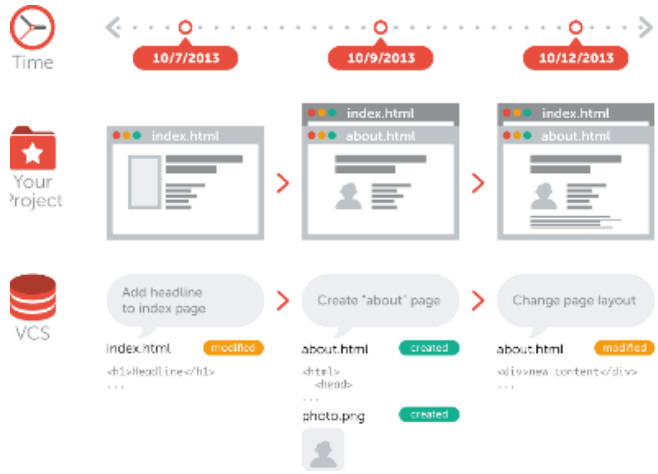


Figura 11. Versiones en el desarrollo de un producto de software[4]

Existen dos sistemas de versionamiento ampliamente utilizados en el área de desarrollo, los sistemas centralizados y los sistemas distribuidos[16].

En los sistemas centralizados (CVCS) se tiene un servidor central que almacena todos los archivos versionados, desde el cual los miembros del equipo descargan los archivos que necesitan y trabajan de forma local con ellos, ellos mismos son los encargados de sincronizar sus cambios con el servidor. La desventaja de este sistema es que, si se presenta una falla en el servidor central, entonces no se podrán guardar cambios ni extraer información de versiones anteriores hasta que la falla sea solucionada. Si el disco duro del servidor sufre daños irreparables y no se tiene copia de seguridad de la información, se corre el riesgo de perder toda la información del proyecto y sus versiones (Véase Figura 12).

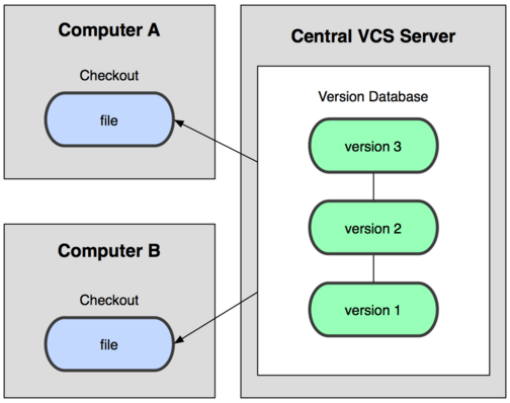


Figura 12. Diagrama que muestra el sistema centralizado de versionamiento.

[4] Araujo, J. (2018). Ilustración de un sistema de control de versiones. Recuperado de https://joaquinaraujo.github.io/git-github/

Por otro lado, en los sistemas de control de versionamiento distribuido (DCVS) todos los miembros del equipo replican completamente el repositorio principal en una copia local, por lo que si el servidor falla cualquiera de las copias puede usarse para restaurarlo. Una vez que se ha replicado el repositorio todos los miembros del equipo pueden trabajar sin necesidad de conectarse a la red y mantener un seguimiento de versiones de manera local, hasta que decidan publicar los cambios en el repositorio principal, la Figura 13 muestra un esquema de este sistema.

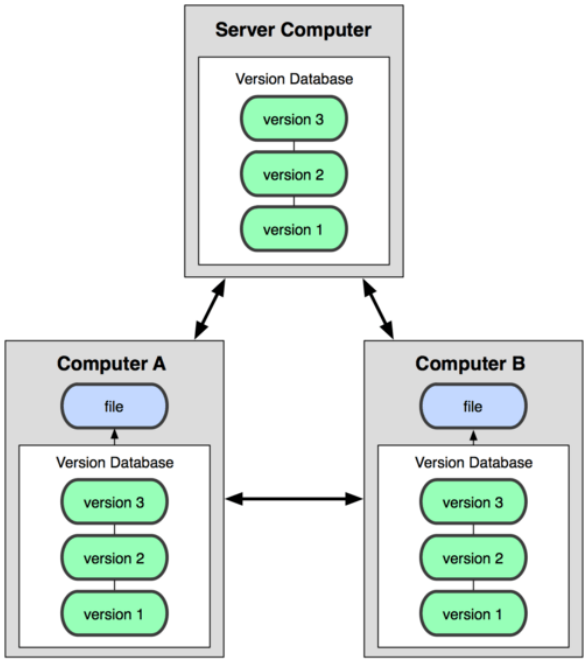


Figura 13. Diagrama de control de versiones distribuido.

El sistema que utilizamos y que utiliza Git es el DVCS, con el cual podemos asegurar nuestro trabajo, compartir código fácilmente y comparar antiguas versiones con las nuevas.

En proyectos grandes como en el que trabajé, cuyos repositorios guardan gran cantidad de archivos, coordinar y mantener un control de versiones se vuelve complicado si se trabaja linealmente, es decir, sobre una sola rama, es por esto que usamos un sistema de ramificaciones cuya función era dividir los flujos de trabajo, donde cada área de la solución se trabajaba sobre una rama, creando sus propias versiones y actualizando la rama principal después de haber concluido el desarrollo de una tarea.

Es así como de acuerdo con la funcionalidad que cumple el código en que trabajamos creamos ramas a partir de la rama principal o "*master*" que es la fuente más fidedigna y actualizada de todo el proyecto. Esta rama es muy importante y debemos ser cuidadosos con los cambios que subimos a ella, ya que el código que contiene debe mantenerse en funcionamiento y está lista para ser desplegada y ejecutar la solución completa.

Comprobar la funcionalidad del código desarrollado en cada rama antes de fusionarlas o hacer "*merge*" con la rama "*master"* es una de las formas en que mantenemos la rama master depurada.

La acción que se hace en Git para fusionar ramas se llama "*merge*" y durante cada *merge* Git ofrece una interfaz para resolver los conflictos que se puedan presentar al unir pedazos de código entre los repositorios de cada rama. En la Figura 14 se ejemplifica el flujo de versiones en 6 ramas, donde 5 de ellas emergen de la rama *master*.

Figura 14.Diagrama que muestra el historial de versiones entre ramas de un proyecto.

Durante el proyecto seguí buenas prácticas de versionamiento:

- Confirmar (*commit*) mis cambios con regularidad en mi copia local documentando lo que representaba cada *commit*, esto me permitía compartir cambios de forma frecuente con mis compañeros y con la descripción sabía a qué se refería cada cambio.
- Trabajar sobre una rama exclusiva para la funcionalidad que yo añadía a la solución.
- Cuando realizaba un "*merge*" con la rama de *master*, primero actualizaba mi rama para obtener los cambios más recientes de la rama principal, de esta manera si ocurrían problemas de sincronización se quedaban en mi rama y no afectaban al repositorio más importante, una vez que hacía esta sincronización hacía un *merge* con la rama master sin problemas.
- Para aumentar la seguridad sobre los cambios que se subían a la rama de *master* del servidor, se creó una regla de solicitud llamada "*pull request"*, donde un consultor más experimentado revisaba mis cambios antes de integrarlos a la rama master del servidor.
- Actualizaba constante mi rama *master* local directamente del servidor. Así estaba enterada de los cambios que realizaban otros miembros del equipo y analizaba el impacto de sus cambios en la funcionalidad de mi código.

III. I. III. Revisión y depurado de la solución

Cuando se tuvo un avance aproximado del 60% equivalente a una primera iteración completa en el desarrollo de la solución, empezamos a desplegarla en el ambiente de desarrollo. De esta manera el área de calidad podía empezar a realizar pruebas de funcionalidad a cada parte del código.

En el caso de *Front end* las pruebas consistían en comprobar que todas las interfaces mostraran la información correcta al usuario, que los elementos dinámicos con los que el usuario podía interactuar funcionaran correctamente y que las especificaciones de seguridad que restringían las acciones de usuario estuvieran implementadas correctamente. Si había algún error de funcionamiento, sintaxis, ambigüedad etc., el encargado me lo reportaba utilizando la aplicación de Azure DevOps. Una vez que yo recibía la notificación corregía el error y en el próximo despliegue mandaba los nuevos cambios.

Por mi parte, de manera recurrente también participé en la programación de pruebas unitarias[5] que utilicé para comprobar que la información que se procesaba hacia y desde los catálogos fuera consistente y adecuada en todo el proceso.

Las pruebas unitarias las hacíamos durante el proceso de desarrollo y se caracterizaban por ser automatizables, rápidas e independientes, es decir, que no dependían de otras pruebas o "*tests*" unitarios. Estas sirven para asegurar que cada parte del código funcione bien por separado por medio de comparaciones ante diferentes escenarios, si el resultado de una comparación es negativo, entonces el resultado de la prueba es una falla y se tendrá que revisar el código programado. (Véase Figura 15).

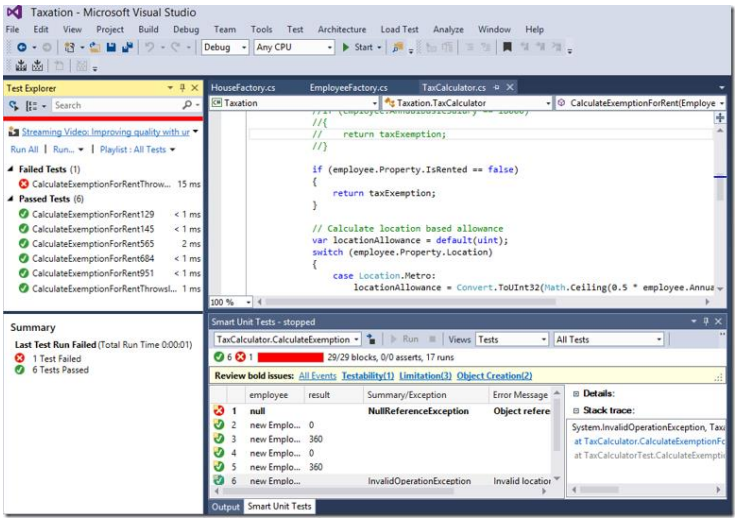


Figura 15. Ejemplo de ejecución de siete pruebas unitarias en el IDE de Visual Studio, donde seis son exitosas y una falla..

III.II. Metodología usada en el área de calidad

Mi participación en proyecto con la Empresa 2 fue diferente respecto al anterior ya que me dediqué a comprobar que la solución implementada por el equipo de desarrollo cumpliera las especificaciones técnicas que previamente fueron acordadas con el cliente.

[5] Pruebas que se realizan para saber si una unidad o módulo de código funciona como se espera.

De manera general hay dos tipos de pruebas: pruebas de caja negra y de caja blanca.

En las pruebas de caja negra, se ingresan datos a la prueba deseando obtener un resultado esperado. Este tipo de pruebas se concentra en el resultado mas no en el proceso que sigue. Por otro lado, las pruebas de caja blanca o transparentes se concentran en el mecanismo interno y son usadas para verificar que el producto se comporta de la forma en que se ha especificado[17].

A lo largo del proyecto se llevan a cabo diferentes pruebas a la solución dependiendo del avance que se lleve y tienen la secuencia:

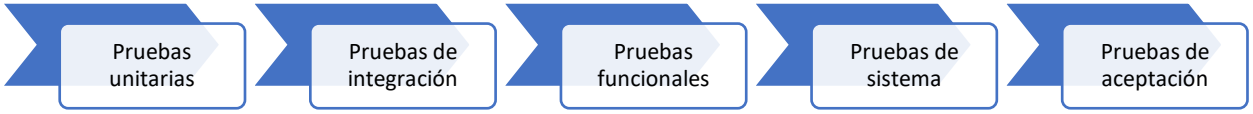


Figura 16. Secuencia de pruebas durante la vida del proyecto.

En la tabla 2 se describen los tipos de pruebas de software[18] que se utilizaron para asegurar la calidad del sistema.

Tipos de pruebas	Descripción	Encargado de hacerlas
Pruebas unitarias	Sirven para revisar si un módulo específico de código funciona como se espera.	Desarrollador
Pruebas de integración	Sirve para revisar si uno**,** dos o más módulos de código funcionan adecuadamente de manera simultánea.	Probador de software
Pruebas funcionales	Sirven para validar si el sistema completo se comporta como se espera. Las pruebas funcionales no tienen que ver con el código interno.	Probador de software
Pruebas de Sistema	Pruebas de carga: sirven para evaluar el comportamiento de la solución ante condiciones desfavorables de consultas. Pruebas de regresión: después de la modificación del sistema, componentes o grupo de unidades, sirve para asegurar que la solución aún funciona correctamente.	Probador de software

	Pruebas de usabilidad: Prueba que la interfaz sea amigable con el cliente, prueba la facilidad de uso de la aplicación. Pruebas de rendimiento: prueba la velocidad y efectividad del sistema, asegurando así tener respuesta dentro del tiempo especificado.	
Pruebas de aceptación	También conocidas como pruebas de UAT (por si siglas en inglés, *User Acceptance Testing*), son pruebas formales con respecto las necesidades del usuario, requerimientos y procesos de negocio, realizadas para determinar si satisface los criterios de aceptación que permita al cliente determinar si se acepta o no el sistema.	Cliente

Tabla 2. Tipos de pruebas de software.

En este proyecto mi integración se dio durante la etapa de pruebas, es decir, cuando el desarrollo ya estaba en una fase avanzada y era necesario comprobar que funcionaba de acuerdo a las necesidades del cliente, por lo que trabajé con pruebas funcionales y pruebas de rendimiento o "performance", con un equipo de consultores que únicamente se dedicaba a la realización de pruebas.

III.II.I. Pruebas funcionales

Las pruebas funcionales con las que trabajé fueron las necesarias para asegurar que el sistema funcionara de forma correcta, específicamente verificando que se implementara una serie de validaciones en el servicio de recepción de documentos fiscales.

Estas validaciones eran aplicadas a los comprobantes digitales fiscales que, de acuerdo con el estándar internacional UBL 2.1 de la norma ISO/IEC 19845:2015[19] se recibían en formato .xml con la estructura y reglas establecidas en formato .xsd del mismo estándar.

Existen doce tipos de comprobantes fiscales que de acuerdo con las normas fiscales de aquel país son aceptados y requerían ser validados al pasar por un sistema de recepción de archivos, cada uno de ellos contenía información estructurada y a cada campo se le aplicaban validaciones para saber si eran documentos verídicos y sustentados.

Además de nuestro equipo de pruebas, el equipo de informática de parte del cliente también realizaba sus propias pruebas y se encargaba de entregarnos los archivos (modificados arbitrariamente para que fallaran con alguna validación en el servicio de recepción) con los que trabajaríamos aplicándole las pruebas.

Cada validación tenía un código de error asociado y para comprobar que se ejecutaban correctamente hacíamos pruebas masivas de funcionalidad con documentos de ejemplo, donde se esperaba que los documentos fallaran con algún código de error.

Al término de la prueba obteníamos un documento que nos mostraba el código de error con el que fallaba la recepción del archivo y lo comparábamos con el error que debía salir, si ambos códigos de error eran iguales entonces el análisis terminaba en ese punto y se corroboraba el buen funcionamiento de esa validación, si no eran idénticos entonces analizábamos por qué habían fallado. A este proceso de análisis le denominamos hacer "triaje".

El triaje fue el proceso que demoraba más tiempo debido a la cantidad de archivos que eran enviados en las pruebas. En cada prueba lazábamos hasta 5,000 archivos con los que se esperaba comprobar cientos de validaciones en los doce tipos de comprobantes fiscales los cuales se probaban en dos servicios web de recepción diferentes.

Las validaciones se procesaban en niveles jerárquicos, las más importantes se ejecutaban primero, de tal manera que los documentos podían presentar fallas de tipo "fatal error" o advertencias.

- Si un documento fallaba con "fatal error", el sistema arrojaba el código de la primera validación que se había incumplido y el archivo no podía ser recibido y se mandaba mensaje al usuario
- Si un documento fallaba con advertencia, el sistema podía recibir el archivo, pero lanzaba un mensaje de advertencia al usuario

Durante las pruebas existían varias razones por las que un archivo podía fallar:

- La validación se estaba implementando de manera incorrecta a nivel de código
- La validación no estaba clara o se tradujo mal al equipo de desarrollo
- El error que presentaba el archivo caía en más de una validación y el resultado de salida regresaba sólo el primer "fatal error" encontrado o se lanzaban más errores de advertencia que los esperados.
- El error esperado por el cliente era incorrecto
- Hubo un problema con el servidor o las credenciales de autenticación del archivo no eran válidas

La meta al ejecutar las pruebas funcionales era tener un resultado transparente, es decir, que todos los archivos enviados a los receptores arrojaran los códigos de error esperados. El proceso que seguí para identificar y canalizar las fallas se describe en el diagrama de la Figura 17

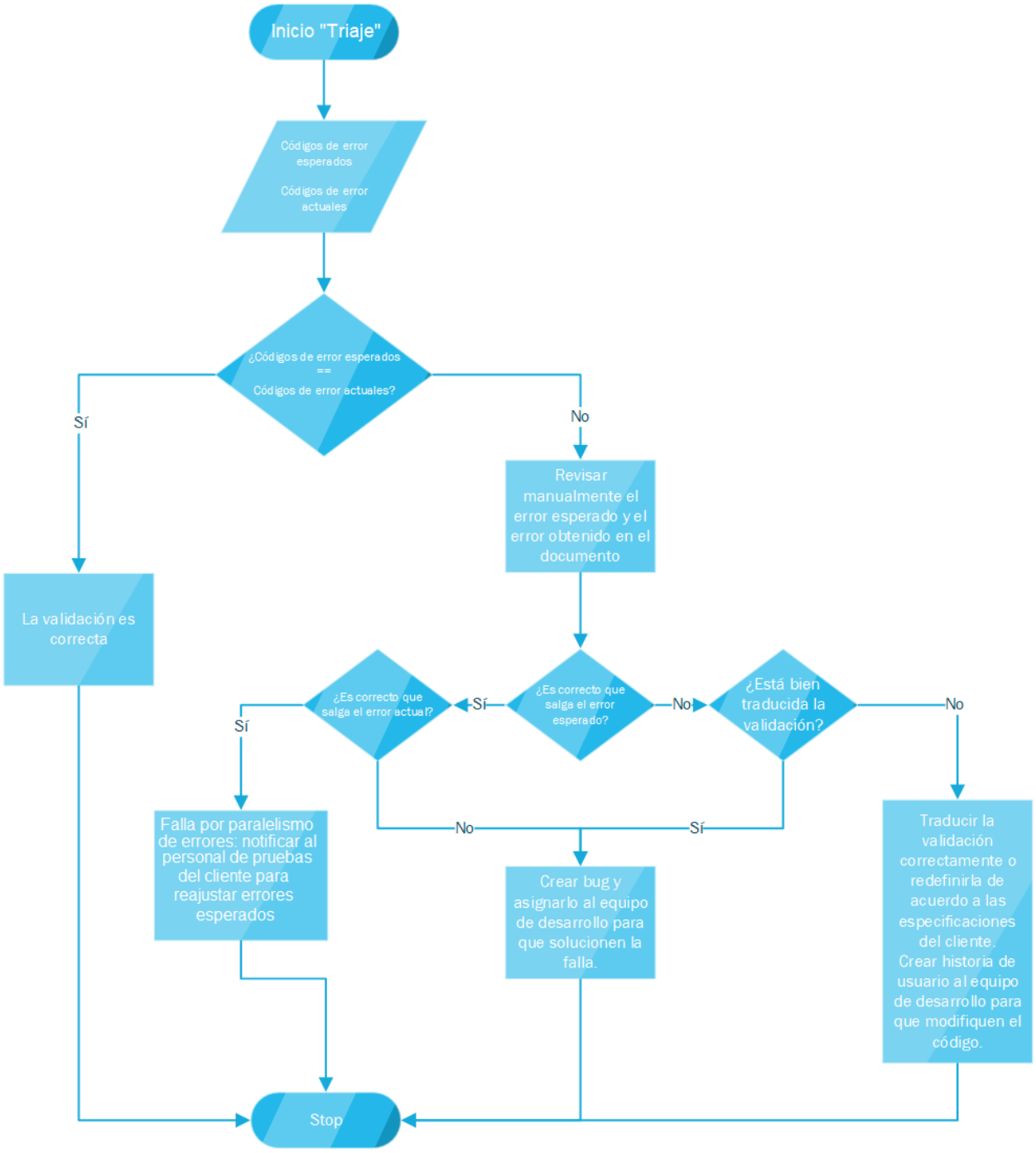


Figura 17. Diagrama de flujo del proceso de triaje.

En el diagrama anterior se muestran las acciones que tomaba durante el proceso de triaje, dos de estas acciones fueron la creación de reportes de *bugs* e historias de usuario. La diferencia entre un bug y una historia de usuario es que el bug es un defecto específico en el código y un solo programador es necesario para resolverlo, mientras que la historia de usuario es un elemento de trabajo que define aplicaciones, requerimientos y elementos que los equipos planean crear (Kathryn & Steve, 2018). Las historias de usuario que definía estaban basadas en cambios de la

funcionalidad de la solución o cambios en las especificaciones iniciales de desarrollo, eran más complejas que un bug y requerían colaboración de varios miembros del equipo para su solución.

Para levantar estos reportes y asignarlas a la persona correcta utilicé una metodología de trabajo llamada Agile y la plataforma de Azure DevOps que provee herramientas para el trabajo en equipo con Agile.

Agile[20] es una familia de procesos de ingeniería que tiene el objetivo de agilizar la entrega de productos de software de calidad con un enfoque comercial que alinea las necesidades del cliente y los objetivos de la empresa. En Azure DevOps los procesos de Agile están orientados a la autoorganización, el trabajo en equipo, la inspección y procesos de adaptación frecuentes que son necesarias para un proyecto exitoso.

Haciendo uso de Azure *Boards*, yo rápidamente podía registrar nuevos bugs o historias de usuario, consultar su estado, modificar o asignarlos a un miembro del equipo, consultar el estado en el que se encontraba o rastrear bugs o historias de usuario para ver si ya existían y no repetir el reporte de una falla si ya existía una parecida. En el Anexo 1, se muestra la interfaz para creación y edición de una historia de usuario con descripción detallada de sus componentes, cuya interfaz es idéntica a la de manejo de bugs.

Estos elementos podían tener cuatro estados diferentes según evolucionaba la solución de la falla:

- Nuevo: el primer estado con el que yo creaba el reporte de falla o historia de usuario era nuevo.
- Activo: una vez que el equipo de desarrollo lo tomaba y analizaba ellos se encargaban de pasarlo a estado activo, y se quedaba con ese estado mientras hacían los cambios necesarios para solucionar el error.
- Resuelto: cuando el encargado de resolver el problema los resolvía, cambiaba el estado a resuelto y lo asignaba nuevamente a mí para su revisión.
- Cerrado: era el último estado que tenía un reporte de falla y yo lo fijaba una vez que había hecho triaje de la falla y verificaba que estaba resuelta.

Un quinto estado podía ser fijado si se cometía un error al registrar una falsa falla que era el de "removido" que servía para desechar el reporte y como etiqueta para que el equipo no lo tomara como un pendiente más. El proceso completo se ilustra en la Figura 18.

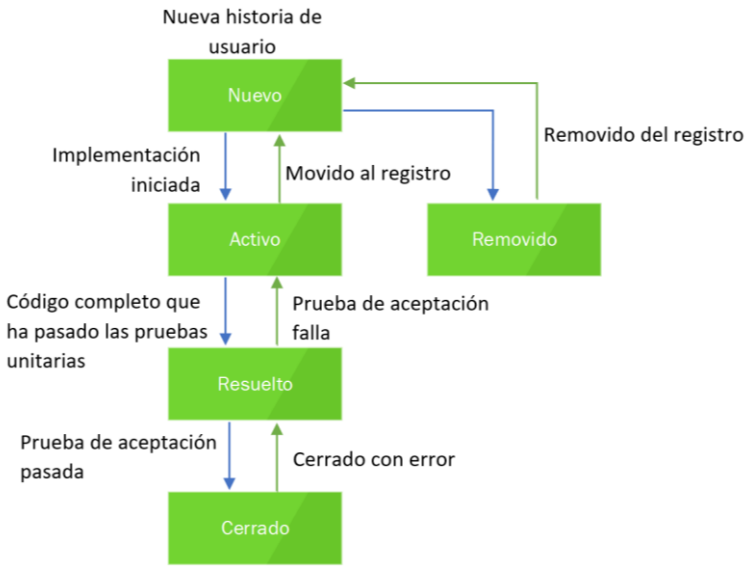


Figura 18. Diagrama del ciclo de vida de una historia de usuario.

El proceso que seguía para verificar la resolución y cerrar bugs era hacer el reenvío del archivo que había fallado al receptor y analizar si esta vez se comportaba correctamente, si así era entonces cambiaba el estado del bug a cerrado y si no lo regresaba a estado de activo con nuevos comentarios para que el desarrollador comprendiera mejor el caso y lo resolviera correctamente.

Como en este proceso no utilizaba la herramienta de pruebas masivas, enviaba los documentos .xml individuales utilizando una herramienta llamada SoapUI. SoapUI es una herramienta versátil para probar, simular y generar código de servicios web de tipo SOAP o REST[21].

Con SoapUI yo lograba vincularme al servicio web de la aplicación. Ejecutaba una solicitud lanzando los archivos de los comprobantes electrónicos de tipo .xml al *endpoint* del servicio web en cuestión, este servicio respondía con un archivo .xml cuyo contenido me resultaba útil para saber si el documento contenía errores en las validaciones. (Véase Figura 19).

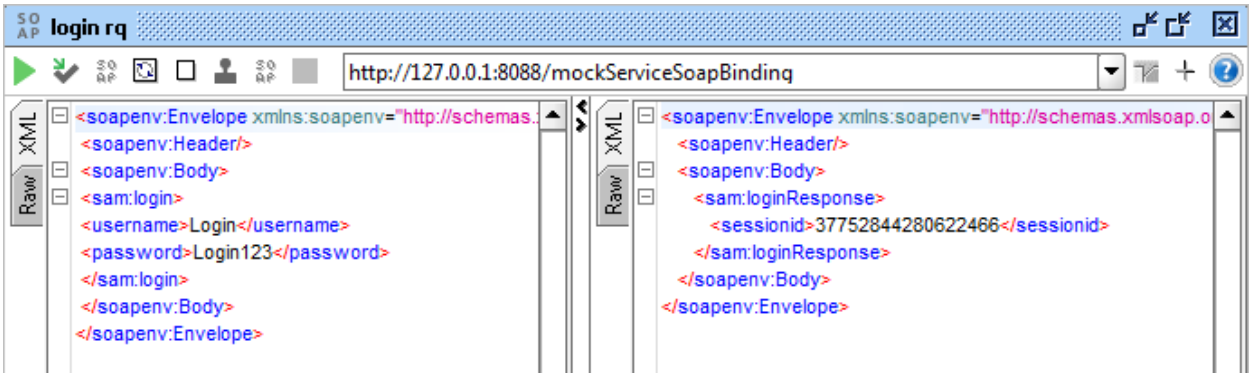


Figura 19. Ejemplo de una solicitud a un servicio web y su respuesta con la interfaz de SoapUI.

El proceso de triaje era repetitivo ya que se aplicaba a cada archivo enviado que tuviera fallas hasta que se encontraban la causa y se cerraran todos los casos.

Entre mis responsabilidades también estaba la de revisar los bugs que creaban los miembros del equipo de pruebas por parte del cliente. Puesto que algunos reportes no correspondían a fallas de funcionalidad, sino a problemas de entendimiento de las especificaciones o funcionalidades de la solución, primero hacía el triaje del caso y si eran observaciones correctas entonces los hacía llegar a un desarrollador. Para hacerlos llegar al equipo de desarrollo hacía su traducción a un lenguaje de inglés técnico.

III.II.II. Pruebas de rendimiento

Otra de mis tareas fue ayudar en la documentación de pruebas de rendimiento.

Las pruebas de rendimiento se basaban en comprobar el tiempo de procesamiento al realizar solicitudes masivas a cinco componentes de la solución con diferentes escenarios de carga y tiempo, los cuales generalmente eran pensados para situaciones adversas como horas pico.

Se documentaron las pruebas en los cinco componentes, demostrando la rapidez de respuesta del sistema al:

- Recibir, procesar y almacenar pruebas en los receptores
- a la carga inicial de una página de consultas
- Hacer consultas de documentos XML de tamaños variables.

Por ejemplo, si en las especificaciones estaba el envío de "x" cantidad de archivos durante "x" tiempo las pruebas se corrían y se comprobaba que el sistema era capaz de procesar esa carga en un tiempo menor al esperado por el cliente.

Para esto se corrieron cinco pruebas con aproximadamente cuatro escenarios distintos usando un componente de Visual Studio llamado *Web Performance And Load Testing Tools* [22] que además reportaba un análisis completo y detallado de los resultados de las pruebas y los resultados se vinculaban al portal de DevOps. (Véase Figura 20)

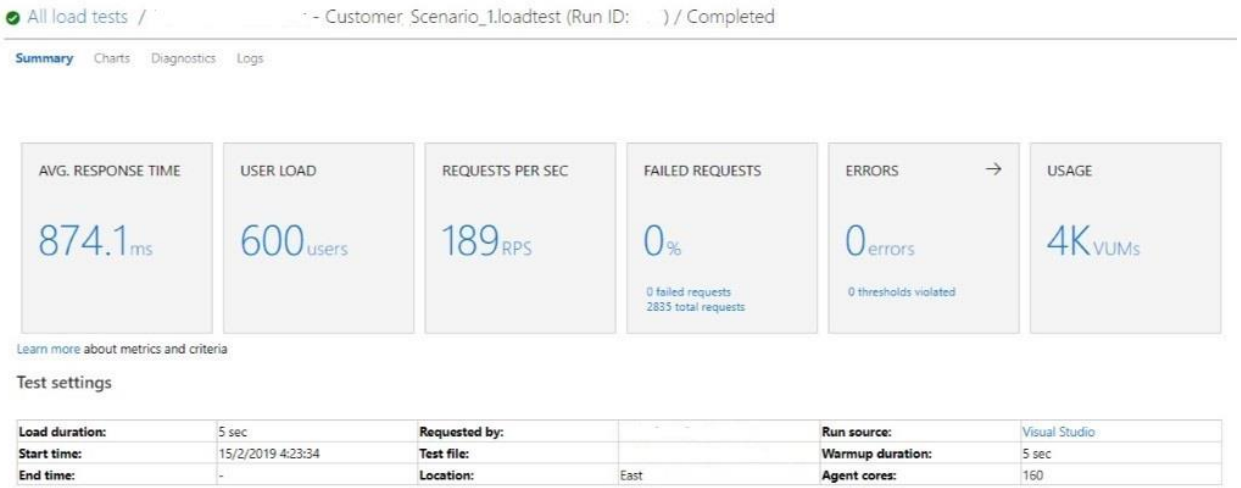


Figura 20. Interfaz que muestra el resultado de un escenario de una prueba de performance en el portal de Azure DevOps.

Además, documenté y reporté algunos casos en los que la prueba fallaba. Sin embargo, reportar estos casos era un poco más complicado que el triaje convencional, ya que al tratarse de pruebas que incurrían en el procesamiento de archivos a través de diferentes componentes del sistema, era necesario hacer un rastreo del documento específico que había fallado.

Afortunadamente para esta tarea existe una herramienta versátil que provee Azure DevOps conocida como *Application Insights*. Ésta es un servicio de *Application Performance Monitor* (APM)[23] extensible para desarrolladores web. Sirve para supervisar aplicaciones web en tiempo real y detectar anomalías de rendimiento. Incluye funcionalidades de análisis y telemetría de lo que pasa en las aplicaciones suscritas en todo momento

Específicamente provee una funcionalidad de rastreo mediante *logs* o trazas. Los *logs* son registros que el desarrollador implementa en la programación a nivel de código como método de rastreo de errores que suelen contener mensajes sobre la razón de la falla. Cuando se ejecutan ciertas funciones en los componentes, los logs que contienen mensajes e información útil sobre el proceso en ejecución, se guardan en una base de datos de *Application Insights* y ayudan a detectar fallas en el sistema.

De esta manera pude aplicar filtros y consultas en lenguaje *Kusto Query*[24] para encontrar el origen de la falla, reportarlo y darle seguimiento a la solución. (Véase Figura 21)

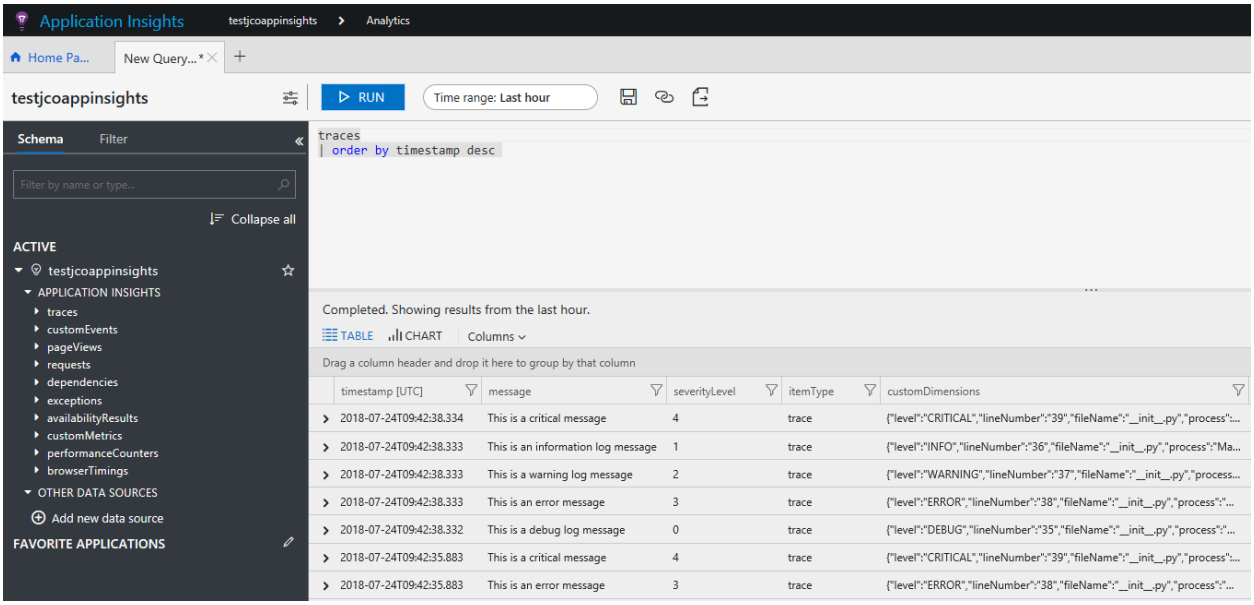


Figura 21. Ejemplo de una consulta de registros en el portal de DevOps usando Application Insights..[6]

[6] Corioland, J. (2018). Imagen ilustrativa de una consulta en Application Insights. Recuperado de https://blog.jcorioland.io/archives/2018/07/24/how-to-azure-application-insights-python-flask.html

III.III. Aprendizaje

Como parte de mis actividades de consultoría, mantenerme al día de las nuevas tecnologías es fundamental para el desarrollo y entrega de soluciones competitivas en el mercado.

Es por eso que durante los meses en los que he trabajado para Microsoft también he dedicado tiempo para aprender y repasar desde los conceptos más básicos de programación hasta conceptos de arquitectura de software en la nube.

He estudiado de los conceptos fundamentales de la nube, los servicios que ofrece Azure como *PaaS*, almacenamiento en la nube con bases de datos relacionales y no relacionales, integración y seguridad en la nube, servicios cognitivos, *bots*, internet de las cosas, *machine learning*, aplicaciones web entre otros temas.

Gracias a este tiempo de estudio tuve la oportunidad de presentar y aprobar tres exámenes de certificación:

- 98-361 *Software Development Fundamentals*
- AZ 70-535 *Architecting Microsoft Azure Solutions*
- AZ 203 *Developing Solutions for Microsoft Azure*
- AZ 400 *Microsoft Azure DevOps Solutions*

Pero no sólo he aprendido cosas técnicas, también he reforzado mis habilidades blandas como la buena comunicación y organización al asistir a juntas técnicas o de negocio y resolviendo conflictos con el cliente.

El trabajo en equipo de manera presencial o remota, con colegas nacionales y extranjeros me ha permitido desarrollar más mi capacidad de hacer relaciones interpersonales de trabajo y he notado una creciente capacidad de creatividad.

El hecho de intercambiar papeles dentro del mismo rol de consultor en diferentes proyectos también me ha hecho tener más perspectivas sobre mi trabajo y aumentar mi capacidad crítica sobre las situaciones que suceden dentro de los proyectos.

IV. Resultados

IV.I. Resultados en Desarrollo

En el proyecto con la Empresa 1 logré crear una interfaz gráfica en el portal de empleado mediante la cual algunos empleados con niveles de autorización permitidos tenían acceso inmediato a catálogos con información de negocio que era importante para ellos. Contribuí así a la automatización del proceso de despliegue, almacenamiento y edición de información que de otra manera sería un trámite lento dentro de la misma institución, provocando demoras en la implementación de cambios en sus reglas de negocio, que se traducen en tiempo de demora del sistema para los trámites de recaudación de impuestos de un país completo.

La interfaz, después del proceso de pruebas de usabilidad y correcciones de *bugs*, resultó amigable con el usuario, puesto que en cada catálogo incluí filtros de búsqueda y paginación que facilitaban y ahorraban tiempo para encontrar información específica dentro de los registros que solían ser sustanciosos. Además, cada catálogo incluía mensajes claros sobre los procesos que se estaban ejecutando y advertencias cuando era necesario, la navegación por los menús y modales era fluida y cumplía con el principio del mínimo esfuerzo, es decir, dándole al usuario la facilidad de llegar a la información que necesitaba con la menor cantidad de clics.

Al ser una solución en la nube también se aseguró que la información estuviera disponible en todo momento y ante alguna contingencia los sistemas que tenían dependencia de los datos siguieran funcionando adecuadamente.

IV.II. Resultados en el área de calidad

Por otro lado, para el proyecto con la Empresa 2, se realizaron varias iteraciones de pruebas masivas, en cada iteración identifiqué errores y los reporté para que fueran atendidos por el área de desarrollo, con estas acciones fueron disminuyendo los reportes de fallas hasta llegar a la meta de cero bugs. La Figura 22 muestra el cierre de bugs durante el proyecto y es prueba fehaciente de la culminación de pruebas de funcionalidad.

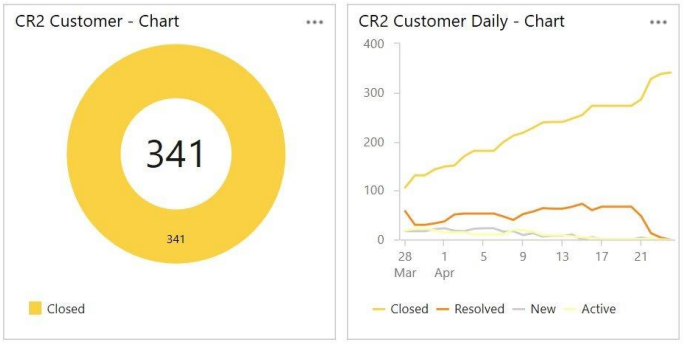


Figura 22. Gráficas que muestran el reporte de fallas y su estado durante las pruebas funcionales.

Al llegar a la meta se firmó por ambas partes el documento de cierre de las pruebas funcionales correspondientes al entregable de recepción y validación de comprobantes en la plataforma de nube.

Posteriormente se cerraron también las pruebas de rendimiento al validarse que en los tres receptores de la solución se recibían, procesaban y almacenaban comprobantes en tiempos (segundos) menores a los establecidos por el cliente en condiciones adversas, es decir, sobrecarga de comprobantes como en escenarios de horas pico.

Una vez que el cliente firmó la conformidad con todos los entregables incluyendo documentación, el proyecto entró a la fase de producción, en la que actualmente se encuentra. Lo que significa que la solución ya está siendo utilizada por millones de usuarios que generan y emiten comprobantes de pago a la entidad y tardan apenas unos segundos para recibir un comprobante de recepción que da fe y garantía a los adquirientes de un bien o servicio de que su comprobante de pago ha sido recibido por la Empresa 2.

V. Conclusiones

El mundo cambia rápidamente y la tecnología con él a pasos agigantados. Cada vez los usuarios buscamos rapidez y eficacia en todos los sistemas con los que interactuamos, buscamos soluciones automatizadas para evitar procesos repetitivos y tediosos, buscamos estar conectados todo el tiempo y en cualquier parte del mundo.

Las empresas y negocios deben adaptarse a estos cambios, cambiando su mentalidad y forma de trabajar para crecer a la par de la sociedad y mantener su nivel de competitividad en el mercado. Pero no sólo las empresas del sector privado están implementando acciones como mudar sus sistemas tecnológicos a la nube, los gobiernos también se están dando cuenta del poder de las tecnologías para manejar sus recursos de manera efectiva.

Y prueba de esto son los casos las Empresas 1 y 2 que mediante sus plataformas en la nube buscan acercarse a sus usuarios con la mentalidad de simplificar los trámites fiscales a sus usuarios y empleados.

Ahora sus sistemas no sólo cuentan con almacenamiento masivo, si no que gracias a que están alojados en la nube tienen ventajas de adaptabilidad a lo largo del tiempo, tienen mejor persistencia de información, pueden ser escalados fácilmente sin necesidad de añadir o adquirir servidores físicos y mejores niveles de respuesta en sus sistemas.

Pero además cuentan con herramientas de monitoreo y análisis que les permiten seguir mejorando sus aplicaciones con el tiempo.

En mi paso a través de estos dos proyectos logré aplicar mis conocimientos en programación orientada a objetos en lenguaje C# y habilidades de investigación de otras tecnologías de software como la nube de Azure, que adquirí durante mis estudios de licenciatura para desarrollar una interfaz pensando en las necesidades del cliente y del usuario final. Esta interfaz desarrollada a lo largo de un mes y medio ahorrará tiempo de latencia de implementación de parámetros en los procesos de recaudación de impuestos.

Así mismo, a lo largo de dos meses contribuí al cierre de 341 *bugs* y documentación de pruebas con el fin de asegurar la calidad del producto que se entregaba a la Empresa 2.

Para ambos proyectos además de conocimientos técnicos, las habilidades blandas como exponer temas y el trabajo en equipo fueron útiles al solucionar los problemas que se presentaban durante el desarrollo y las pruebas.

La actividad que me pareció más importante al iniciar cada proyecto fue la de leer y entender la documentación de necesidades y especificaciones del proyecto, puesto que entendiendo eso podía empezar a hacer mi trabajo porque sabía lo que se necesitaba y con un poco de ayuda técnica podía implementar un producto que realmente reflejara una solución a las necesidades del cliente.

Debido a que fueron mis primeros proyectos en el área laboral cada uno resultó ser un reto debido a la infinidad de temas nuevos que tuve que aprender, pero eso no fue impedimento para buscar llegar a la mejor solución. Como consultora la parte más importante de mi trabajo es la búsqueda

del aprendizaje constante y la adaptación a las nuevas tecnologías, de esta manera siempre, aunque no tenga la solución inmediata la tendré.

Finalmente quiero hacer énfasis en un aspecto en el que caí en cuenta: el trabajo en equipo es el principal apoyo en situaciones cruciales del proyecto y es la clave del éxito de cualquier enmienda. Una de las armas más valiosas que tenemos para lograr nuestros objetivos es la comunicación, sobre todo en proyectos grandes como en los que trabajé pues nos permite tener un mejor control en la sincronización de tareas, podemos resolver más fácilmente algunas tareas y asegurar el entendimiento de los problemas.

VI. Referencias

1. Musolf, N. (2009). *Build For Success, The History Of Microsoft.* Mankato, Minessota: Creative Education.
2. Microsoft Coroporation. (30 de junio de 2019). *Microsoft Business Organization*. Recuperado el 7 enero de 2020 de Facts About Microsoft: https://news.microsoft.com/facts-about-microsoft/#BusinessOrganization
3. Microsoft Corporation. (2019). *Our company*. Recuperado el 20 de diciembre de 2019 Microsoft About: https://www.microsoft.com/en-us/about
4. Microsoft Corporation. (20 de diciembre de 2019). *Modern Applications Consultant*. Recuperado el 7 enero de 2020 Microsoft Careers: https://careers.microsoft.com/us/en/job/686117/Modern-Applications-Consultant
5. Microsoft New Center LATAM. (12 de Junio de 2014). *Es el SAT caso de éxito en la adopción de tecnologías en la nube con Microsoft Azure.* Recuperado el 20 de marzo de 2019 de Microsoft News Center Latinoamérica: https://news.microsoft.com/es-xl/es-el-sat-caso-de-exito-en-la-adopcion-de-tecnologias-en-la-nube-con-microsoft-azure/
6. Microsoft Corporation. (s.f.). *¿Qué es DevOps?* Recuperado el 20 de marzo de 2019 de Microosft Azure: https://azure.microsoft.com/es-es/overview/what-is-devops/
7. Microsoft Corporation. (29 de Marzo de 2017). *Información general acerca de .NET Framework*. Recuperado el 20 de diciembre de 2020 de Microsoft .Net: https://docs.microsoft.com/es-es/dotnet/framework/get-started/overview
8. Pluralsight LLC. (28 de Enero de 2015). *What's the Difference Between the Front-End and Back-End?* Recuperado el 4 de junio de 2019 de Pluralsight: https://azure.microsoft.com/es-es/overview/what-is-devops/
9. Mozilla. (5 de Mayo de 2019). *HTML*. Recuperado el 4 de junio de 2019 de MDN web docs: https://developer.mozilla.org/es/docs/Web/HTML
10. W3schools. (2019). *CSS Tutorial*. Recuperado el 4 de junio de 2019 de w3schools: https://www.w3schools.com/css/default.asp
11. Grados, J. (12 de abril de 2017). *¿Qué es JavaSript?* Recuperado el 20 diciembre de 2019 Devcode: https://devcode.la/blog/que-es-javascript/
12. Smith, S. (1 de julio de 2018). *Overview of ASP.NET Core MVC.* Recuperado el 12 de marzo de 2019 de ASP.NET Core: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.2
13. Json.org. (s.f.). *Introducción a JSON*. Recuperado el 12 de marzo de 2019 de https://www.json.org/json-es.html

14. Chacon, S., & Ben, S. (2019). *Pro GIT* (2da ed.). Mountain View, CA: Apress. Recuperado el 12 de marzo de 2019 de https://git-scm.com/book/en/v2

15. Sherer, T., et al (07 de noviembre de 2019). *Visual Studio Team Services is now Azure DevOps Services*. Recuperado el 12 de marzo de 2019 de Microsoft Docuementation: https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops#visual-studio-team-services-is-now--devopazures-services

16. Acens Technologies. (23 de Julio de 2015). *Control de versiones: Git y GitHub*. Recuperado el 12 de marzo de 2019 de de Acens: https://www.acens.com/comunicacion/white-papers/control-versiones-git-github/

17. Terrera, G. (26 de Febrero de 2017). *Pruebas de caja negra y un enfoque práctico*. Recuperado el 12 de marzo de 2019 de Testing Baires: https://testingbaires.com/2017/02/26/pruebas-caja-negra-enfoque-practico/

18. ISTQB. (2016). *Glossary*. Recuperado el 12 de marzo de 2019 de International Software Testing Qualifications Board: http://glossary.istqb.org/

19. OASIS Standard. (4 de Noviembre de 2013). *[UBL-2.1] Universal Business Language Version 2.1*. Recuperado el 10 de junio de 2019, de UBL: http://ubl.xml.org/news/general-availability-of-ubl-21-oasis-standard

20. Kathryn, E., et al (18 de Noviembre de 2018). *Agile Glossary*. Recuperado el 4 de junio de 2019 de Azure DevOps Services: https://docs.microsoft.com/es-mx/azure/devops/boards/work-items/agile-glossary?view=azure-devops

21. SoapUI. (2019). *SoapUI Projects*. Recuperado el 4 de junio de 2019 de SoapUI: https://www.soapui.org/soapui-projects/soapui-projects.html#2-Project-Types

22. Microsoft Corporation. (12 de junio de 2018). *Load test your app in the cloud using Visual Studio and Azure DevOps*. Recuperado el 20 de diciembre de 2019 de Microsoft Docuementation: https://docs.microsoft.com/en-us/azure/devops/test/load-test/getting-started-with-performance-testing?view=azure-devops

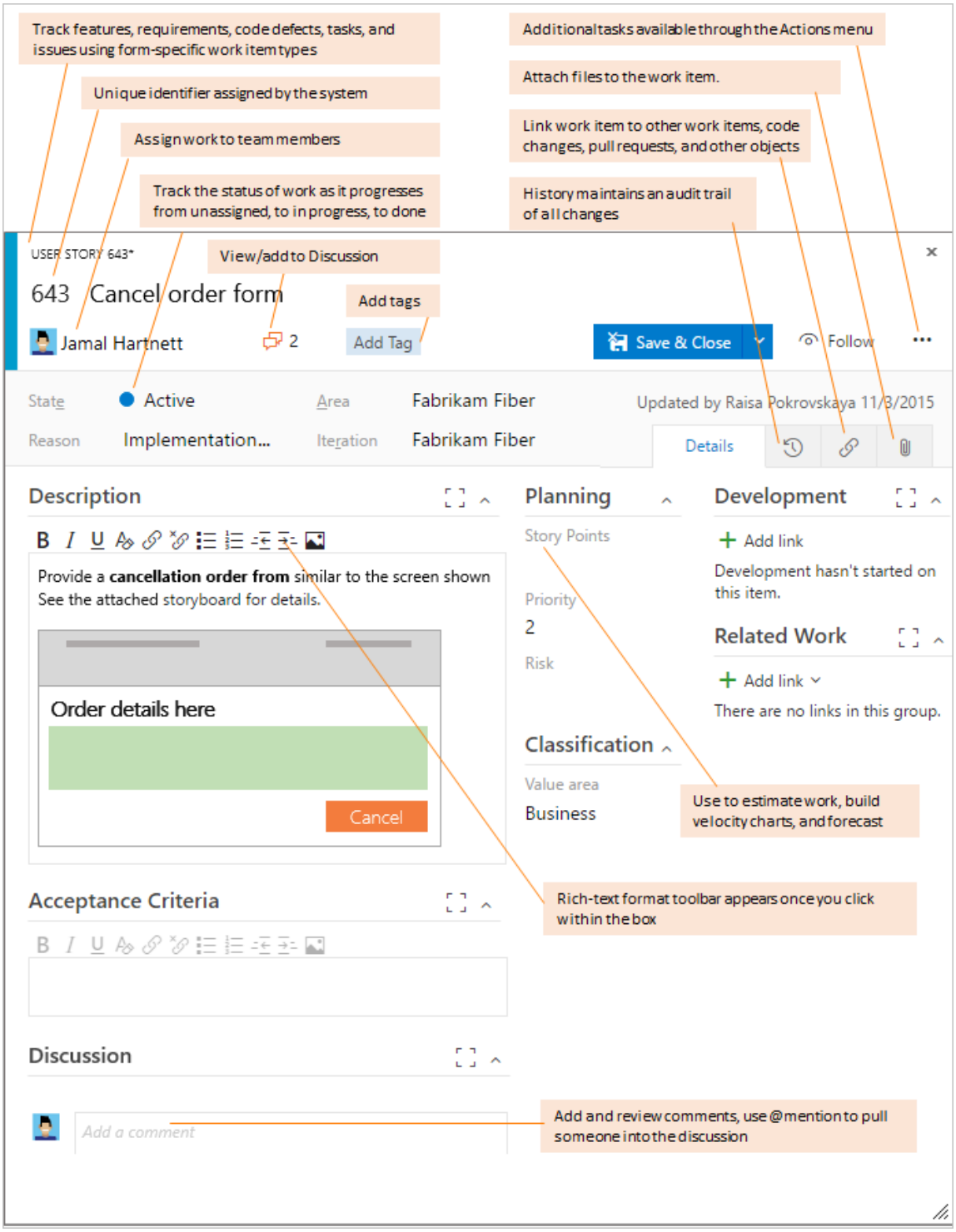
23. Microsoft Corporation. (10 de junio de 2019). *Azure Monitor overview*. Recuperado el 20 de diciembre de 2019 de Microsoft Docuementation: https://docs.microsoft.com/en-us/azure/azure-monitor/overview

24. Microsoft Corporation. (3 de junio de 2019). *Query language*. Recuperado el 20 de diciembre de 2019 de Microsoft Docuementation: https://docs.microsoft.com/en-us/azure/kusto/query/

Anexos

Anexo 1.

Vista descriptiva de una historia de usuario dentro del Portal Azure DevOps que sigue la metodología “Agile”



Glosario

Término	Descripción
AJAX	Ajax (*Asynchronous JavaScript and XML*) se refiere a un grupo de tecnologías que se utilizan para desarrollar aplicaciones web.
API	Siglas en inglés de *Application Programming Interface*. Es un conjunto de funciones y métodos que se usan para conectar diferentes elementos de software.
APM	Siglas en inglés de *Application Performance Management*, sirve para monitorear aplicaciones web.
Application Insights	Es un servicio de Azure usado para para monitorear tu aplicación web en vivo. Automáticamente detectará anomalías de rendimiento.
ASP.NET	IDE de aplicaciones web desarrollado por Microsoft.
Azure	Servicios de almacenamiento y administración masivo de datos y soluciones en la nube registrada por Microsoft.
Azure Boards	Es una herramienta de Azure con la que se puede iniciar de forma rápida y sencilla el seguimiento de tareas, características y errores asociados con su proyecto.
Azure DevOps	Es la herramienta de la nube de Microsoft que incluye prácticas principales, como planeamiento y seguimiento, desarrollo, compilación y pruebas, entrega, supervisión y operaciones El término DevOps, compuesto por *dev* (desarrollo) y *ops* (operaciones), da nombre a una práctica de desarrollo de software que unifica el desarrollo y las operaciones de TI.
Bug	Término usado en informática para referirse a un error de programación.
Catálogo	Término utilizado en el contexto de la solución del proyecto para la Empresa 1 que se refiere a bases de datos utilizadas por el cliente que contienen información relativa a su negocio y necesita ser actualizada periódicamente.
Endpoint	Es el punto de conexión de un servicio web. Indica la ubicación física del servicio

GIT	Es un sistema de control de versiones que sirve para trabajar en equipo de una forma simple y optima en desarrollo de software.
Global Delivery	Unidad de negocios de Microsoft con base en la India, responsable de apoyar en las actividades de desarrollo en el área de servicios.
Handler	Es un manejador responsable de cumplir con las solicitudes de un navegador.
HTTP	Siglas en inglés de *HyperText Transfer Protocol*. Es un protocolo utilizado para solicitar y transmitir archivos a través de Internet, especialmente páginas web y componentes de páginas web, está orientado a transacciones y opera a través de un esquema petición-respuesta, entre un cliente y un servidor.
Machine Learning	Es una rama de la inteligencia artificial encargada de hacer que que los sistemas informáticos pueden aprender de manera autónoma, identificar patrones y tomar decisiones con mínima intervención humana.
PaaS	Siglas en inglés de *Platform as a Service* es un entorno que permite el manejo y desarrollo completo de aplicaciones en la nube con servicios de administración e infraestructura en la nube. Su principal ventaja es que el proveedor se encarga de la seguridad de la infraestructura, las licencias, herramientas de desarrollo y almacenamiento y el cliente solo se preocupa por desarrollar y administrar su aplicación.
REST	Siglas en inglés de *Representational State Transfer*. Es un estilo de arquitectura de software que establece un estándar para la interacción entre sistemas web mediante solicitudes de tipo HTTP.
Script	Archivo que contiene un conjunto de comandos/órdenes que se ejecutan por lotes.
SOAP	Siglas de *Simple Object Access Protocol*. Es un protocolo de interacción entre servicios web mediante mensajes en formato XML.
Triaje	Proceso de análisis de las fallas encontradas en el código de un producto de software.
UBL	Siglas en inglés de *Universal Business Language*, define al conjunto de librerías que

	establecen un estándar electrónico de reglas de negocio en documentos de tipo XML, tal como ordenes de adquisición y facturas.
XML	Siglas en inglés de *eXtensible Markup Language* sirve como un medio de almacenamiento de información estructurada.

Índice de figuras

Índice de tablas