UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Herramienta Para el Análisis de Dependencias de Software en Proyecto de Integración Continua

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

José de Jesús Herrera Ledón

ASESORA DE INFORME

M. en I. Norma Elva Chávez

Ciudad Universitaria, Cd. Mx., 2019

Índice

INTRODUCCIÓN Y OBJETIVO

La vida solo pude entenderse
yendo hacia atrás, pero debe
vivirse hacia adelante.
-Soren Kierkegaard

Introducción

Es imposible saber cómo será recordado el siglo XXI, el ser humano ha evolucionado para utilizar herramientas y moldear su realidad, ha sido un largo camino desde las primitivas herramientas de piedra al dominio de la electricidad. La actual explosión tecnológica aun no nos garantiza si resolveremos antes de que termine el siglo los graves problemas que atañen a la humanidad, las primeras dos décadas del siglo XXI se han caracterizado por una ola de innovación tecnológica que no se había visto desde la segunda revolución industrial en el siglo XIX, por tal motivo el parlamento europeo reconoció en el año 2006 una tercera revolución industrial en el mundo. Dicha declaración fue la aceptación del fenómeno tecnológico, social y económico que vivimos hoy en día.

Zygmunt Bauman, filósofo polaco, acuño el termino de modernidad liquida para describir los tiempos modernos, según él la época moderna se caracteriza por su fluidez, el cambio continuo, la flexibilidad y la adaptación. Bauman también describía a la época moderna como un periodo de tiempo donde se enaltece al individuo, cada vez más convencido de que su vida estaba ligada al cien por ciento a sus decisiones y que poco o nada influye su entorno.

Mucho ha cambiado desde que nuestra especie comenzó a moldear su realidad, cada salto tecnológico modifica enormemente a la sociedad, su calidad de vida, la manera en la que entiende la realidad y la forma en la que modifica su entorno.

Puede entenderse a la tecnología cómo un medio que nos permite modificar nuestra realidad y a la innovación como el paso siguiente a una tecnología predecesora o al uso disruptivo de una tecnología existente.

Precisamente es la innovación el motor de este tren de cambios, posiblemente la definición de Bauman de una modernidad liquida es otra forma de describir a una sociedad que convive y se desenvuelve en un flujo de innovaciones continuas.

La continua evolución de la tecnología es un medio para la mejora en la calidad de vida de los individuos, es incluso un factor primordial que re dibuja las fronteras políticas del mundo, como en la guerra de las Islas Malvinas en 1982, una batalla por el control de un grupo de islas en el océano atlántico disputadas entre Inglaterra y Argentina, donde muchos expertos atribuyen la victoria de Inglaterra a su poder tecnológico, el avión Harrier de despegue vertical fue fundamental para determinar el destino de la guerra. También la tecnología es víctima de los regionalismos cómo lo es el famoso efecto galápagos en Japón, una nación que continúa innovando incansable y lamentablemente no ha logrado exportar muchas de sus innovaciones.

El estilo de vida occidental esta sin duda fundido con las tecnologías de la información, nunca nuestra especie había tenido acceso a tanta información, nunca habíamos estado tan conectados y al mismo tiempo tan distantes. Hacer el uso correcto de la tecnología es el reto de nuestros tiempos, es por eso por lo que científicos e ingenieros al rededor del mundo han unido esfuerzos con diferentes disciplinas como la física, psicología, agronomía, geografía, economía, administración y medicina entre otras muchas disciplinas con la finalidad de dar solución a los problemas cotidianos y a la generación de herramientas que se crean su propio nicho en el mercado solucionando problemas bien conocidos y otros que no sabíamos que teníamos.

Hablar de innovación en el área de las tecnologías de la información es común en ciudades como San Francisco, Seattle, New York y Guadalajara, donde existen comunidades activas de desarrolladores de software, científicos de datos, emprendedores e

inversionistas habidos de nuevos desafíos, es común escuchar en los cafés de las ciudades diversas conversaciones sobre nuevas tecnologías e innovaciones, se habla con naturalidad de lenguajes de programación, marcos de referencia o productos novedosos que mejoran la calidad, la usabilidad y velocidad de producción. Pero detengámonos un momento en revisar las ideas que nos permiten generar, como si fuera un castillo, ladrillo a ladrillo una nueva y excitante herramienta o producto.

Es precisamente el impulso de innovar lo que nos ha hecho construir herramientas y metodologías que aceleren la innovación. Es esta idea, herramientas para facilitar la innovación lo que se abordará en este reporte, la capacidad de utilizar y reutilizar lo hecho en el pasado, regresando a la analogía del castillo, mejores ladrillos y muros más altos, cómo el muro que se construye sobre el anterior.

Es este principio, la reutilización de código, uno de los valores intrínsecos más poderosos de la programación orientada a objetos, un modelo que describe precisamente el concepto de construir ladrillo a ladrillo para crear muros y más tarde muros mejores sobre el anterior. Reusando el conocimiento adquirido posteriormente.

Evolución de los paradigmas de programación.

Un paradigma en programación es el modelo que sigue un lenguaje de programación al ser creado y con el cual se entiende intenta resolver un problema. Partiendo de esa base todo lo programado con ese modelo tiende a usar el mismo principio.

Pero es imperativo entender que antes de la existencia de las computadoras multipropósito era imposible pensar en paradigmas que pudieran aplicarse como lo hacemos hoy en día fuera del marco teórico. Esto claro no indica que un paradigma, es decir, un modelo no existiera, si no que era impensable aplicarlo sin la tecnología adecuada.

No fue hasta la aparición de la primera computadora multipropósito, diseñada por Konrad Zuse en 1936, que se empezó a debatir la aplicación de modelos de programación. Sin embargo, estábamos muy lejos todavía de los paradigmas abstractos que dominan nuestros equipos hoy en día.

La Z1 de Konrad fue una máquina que operaba de manera mecánica, leyendo tarjetas perforadas, este tipo de programación es conocida como programación secuencial.

Las computadoras debieron evolucionar antes de que un nuevo paradigma se aplicara fuera del campo teórico, con el tiempo los sistemas mecánicos fueron evolucionando a sistemas más complejos, utilizando propiedades de la electricidad como los tubos al vacío, diodos y finalmente transistores, en aquella época las máquinas más modernas eran programadas en ensamblador, un lenguaje que traducía instrucciones del lenguaje humano a lenguaje binario, sin embargo no fue hasta el año de 1966 cuando Corrado Böhm, matemático, publicó *"Flow diagrams, turing machines and languages with only two formation rules"* que nació el paradigma de la programación estructurada.

Cabe destacar que para esta época Alan Turing, uno de los mayores genios en ciencias computacionales (tal vez el mayor) ya había teorizado a profundidad problemas relacionados con la complejidad en tiempo de ejecución, con el concepto y máquinas de

turing, un emocionante campo de investigación se había creado para futuros programadores.

Pocos años después en 1969, Dennis Ritchie crea el lenguaje C y ese mismo año junto a Brian Kernighan crean el sistema operativo UNIX.

A partir de este momento nuevos campos de investigación se abrieron para que científicos, matemáticos en su mayoría, crearan todo tipo de soluciones a problemas cada vez más novedosos, no es casualidad que muchos de los conceptos usados hoy en día para describir los diferentes paradigmas de programación estén profundamente ligados a conceptos matemáticos, como lo son los de estados deterministas y no deterministas. La clasificación más aceptada divide a los diferentes paradigmas en tres ramas, la rama imperativa, la declarativa y la simbólica.

Los lenguajes de programación más comunes en la industria pertenecen a la rama imperativa y declarativa. Es en la rama imperativa donde encontramos a la programación orientada a objetos y procesal, en el declarativo a los paradigmas funcional y lógico.

La POO o programación orientada a objetos se implementa en lenguajes como C++, java o Python, las principales características del modelo en el paradigma orientado objetos son precisamente la creación abstracta de objetos, dichas abstracciones contienen atributos intrínsecos del objeto, los objetos pueden heredar sus atributos a objetos hijos, otra de sus características es el polimorfismo, que no es otra cosa que la manera en que un objeto puede ser descrito por los atributos de sus padres, del mismo modo que puede llamarse simplemente árbol a una particular especie de árboles.

La encapsulación de datos es un atributo ligado a la misma capacidad de la herencia, donde un objeto tiene la capacidad de ocultar atributos para que no puedan ser modificados desde el exterior.

En el campo declarativo podemos encontrar lenguajes como SQL o MySQL dentro del paradigma lógico, pero también herramientas procesales como CSS o Python, considerados lenguajes interpretados, esto quiere decir que se aplican las instrucciones línea por línea.

La rama de la programación simbólica es la más novedosa y está altamente ligada al aprendizaje de las maquinas, se entiende como la manera en que un sistema experto o una inteligencia artificial puede utilizar líneas de código para crear nuevos símbolos que representen una operación o una acción. La programación simbólica tiene su propio nicho ya que es muy complicado encasillarla en cualquier otro paradigma. Si su finalidad es precisamente la de crear nuevos símbolos de representaciones abstractas y por la naturaleza heurística del aprendizaje máquina podría terminar imitando cualquier otro paradigma.

Programación orientada a objetos (POO)

El desarrollo de software en la actualidad está fuertemente ligado al paradigma de la programación orientada a objetos, en el caso de Java, Python y C++, también se conocen como objetos basados en clases, otros lenguajes como ECMA (java script) se basan en prototipos.

Con el tiempo la analogía que más me ha ayudado a describir la POO basada en clases es la siguiente: Imaginemos un conjunto de casas todas iguales en la misma calle, cada casa podría describirse como un objeto con atributos propios, sin embargo, todas las casas idénticas se han construido siguiendo un mismo plano, en este caso la clase sería el plano.

Es así como una clase sirve para definir objetos y cada objeto, aunque construidos igual, son independientes uno del otro y conservan su singularidad.

Como ya se ha descrito en el capítulo anterior, la POO se define por una serie de atributos como herencia, polimorfismo y encapsulación de datos.

Herencia es la capacidad de los objetos a heredar sus atributos a otros objetos, es decir que un objeto contenga todos los atributos del objeto que le ha heredado más los atributos de este nuevo objeto contenga. Retomando la analogía de las casas, si alguien quisiera agregar un atributo extra, un piso extra, por ejemplo, simple mente tendría que dibujar el piso adicional y conservar los atributos de la casa inicial. En el caso de la herencia en objetos se dice que la casa con el piso extra ha heredado los atributos de la casa original y ha agregado los suyos propios.

La POO se caracteriza por su dinamismo, el polimorfismo se podría considerar como uno de los conceptos más abstractos y singularmente uno de los más poderosos.

El polimorfismo se podría describir como la capacidad de los objetos a adaptarse y utilizar sus propiedades más generales. Utilizaré como ejemplo una nueva analogía, por ejemplo, un robot que cuya única tarea es lavar automóviles. A este robot se le da la tarea de lavar un vehículo Honda Civic, el robot podría confundirse ya que él solo sabe lavar autos, en una capa más abstracta un Honda Civic tiene todos los atributos de un automóvil, forma, desplazamiento, llantas, etc. Se podría decir que un Honda Civil es una subclase de automóvil. Una vez que el Honda es identificado como una abstracción proveniente de automóvil el robot no tendrá ningún problema en completar su tarea.

Así un objeto B que hereda propiedades de un objeto A puede ser definido como A o B dependiendo del problema que se desee resolver independientemente de los atributos adicionales que contenga B.

El caso de la encapsulación de datos es una característica especial que en lenguajes como Java da a los objetos la seguridad de que sus atributos serán usados o modificados por los objectos correctos, es decir que objetos externos podrán o no accesar a valores o atributos dependiendo del nivel de encapsulamiento que se le asigne. Existen cinco niveles de encapsulamiento, un nivel estándar que protege los atributos a nivel de paquete, abierto, donde los atributos pueden ser modificados desde cualquier punto, el nivel protegido limita

el acceso a los atributos al mismo objeto y a los que le heredan y el nivel cerrado que restringe al mismo objeto el acceso a estos atributos.

Es precisamente el poder de los objetos y la reutilización de código lo que le ha ganado a POO su popularidad. Estos atributos principales se han desarrollado cada vez más y más hasta lograr una gran escalabilidad, cumpliendo con el concepto de resolver cada problema una sola vez.

Este concepto adoptado en entornos empresariales de manera excelente, lo que les ha dado la capacidad de crear productos cada vez más rápido y con un alto nivel de cohesión.

Uso de objetos

Siendo la reutilización de código uno de los atributos más excepcionales de la programación orientada a objetos *("escribe una vez, corre donde sea")*, lenguajes como Java han mejorado de manera continua la manera en la que estos objetos pueden ser reutilizados y desde hace varios años la versión empresarial, J2EE, ha evolucionado hasta la generación del concepto de características y conectores.

Empresas de software han encontrado en estos atributos una gran motivación para crear sets de productos relacionados con un código base común.

Plug-in and Features (conectores y caracteristicas) son conceptos de objetos que como el nombre los describe están diseñados para utilizarse al conectar, sin más configuraciones. Los features son objetos que se entienden como atributos de un producto y del mismo modo está diseñado para ser utilizado sin más configuraciones en un objeto.

Un *plug-in* empaqueta una solución en un módulo, esta solución no es exclusiva de Java, otros lenguajes como C++ y Python también implementan el concepto de paquetes.

Estos módulos se crean con la finalidad de solucionar un problema común una sola vez y aplicarlos en las situaciones donde se requiera, esto les da la habilidad a los creadores de software de trabajar en más de un producto simultáneamente con atributos similares y así mejorar la velocidad de producción.

Un producto puede tener varios proveedores de soluciones y a su vez una solución puede aplicarse a varios productos. Esto garantiza la integridad del set de productos y la efectividad al tener una sola solución para un problema específico.

Esta manera de pensar permite manejar el tiempo de desarrollo de manera dinámica, cuando se definen los requerimientos para el siguiente lanzamiento del set de productos un solo equipo crea la solución nativa y el resto de los productos que lo requieren lo consumen. Esta etapa puede llegar a ser complicada por los diferentes factores a considerar, como la jerarquía entre productos, que producto requiere esta solución mejor integrada y la carga de trabajo en los equipos de desarrollo.

Justificación

Este reporte pretende describir una solución a un problema puntual dentro de la organización de dependencias en un set de productos empresariales.

Así como un proveedor puede generar una solución para varios productos, este proveedor puede generar varias versiones de la misma solución. A su vez el consumidor de la solución necesita estar altamente cohesionado con otros productos y por lo tanto utilizará la misma solución en la misma versión.

En otras palabras, si el producto final fuera una computadora, el fabricante necesitaría comprar cada pieza que integra la maquina con diferentes proveedores y estos proveedores a su vez le venden piezas a otros consumidores.

En este ejemplo todos los proveedores trabajan continuamente en nuevas versiones de su producto y tienen fechas específicas de lanzamiento empatadas con su mayor consumidor.

Sin embargo el consumidor ha disidido utilizar una versión específica del producto y no utilizar los últimos lanzamientos, pero sigue dispuesto a pagar por el soporte técnico necesario. Este comportamiento se continúa a lo largo de los años y ese consumidor se convierte en el único. Sin embargo el modelo de integración continua le fuerza a seguir desarrollando y dando soporte a sus productos.

Imaginemos este escenario con muchos proveedores y un solo cliente que les consume a todos. Este cliente además ha decidido usar una versión específica en más del 90% de sus proveedores. Pareciera un cliente muy incómodo, pero ese cliente a su vez ha generado una herramienta muy ligada a un solo cliente. Llevando su solución al mismo problema que a sus proveedores.

Ahora imaginemos que dicho cliente y proveedores en realidad forman parte de la misma empresa y de hecho son productos generados en un ambiente diseñado, en principio, para generar muchos y diferentes productos desde una misma base.

El problema que se aborda en este reporte es un análisis parcial de la problemática en un área de negocios de la empresa en la que se desarrollan más de 30 herramientas de software que se conjugan en un solo producto y al mismo tiempo se consumen entre ellos.

Dicho problema se complica cuando el número de dependencias entre productos es alto y el número de soluciones existe en diferentes versiones y productos.

Y agrava cuando estas dependencias se reflejan en los estados de cuenta de los productos e inicia un drama en la toma de decisiones, que impactará a cientos de desarrolladores y gerentes en tres continentes.

Objetivo

Crear una herramienta para graficar las diferentes relaciones entre productos e identificar el nombre, la versión y la fuente de una de las características y conectores que se consume en el producto principal.

1 DESCRIPCIÓN DE LA EMPRESA

Nada grande se ha hecho en el mundo sin una gran pasión.
-Georg Wilhelm Friedrich Hegel

Durante los más de cinco años que he laborado en la ciudad de Guadalajara he tenido la oportunidad de trabajar primeramente como emprendedor y posteriormente en compañías internacionales, la primera empresa de la que hablaré y a la que llamo "el gigante azul" es en la cual elaboré el proyecto reportado en este trabajo.

En Febrero de 2017, después de dejar al gigante azul, trabajé como consultor en un corporativo internacional con origen en la india, en la que me desarrolle como desarrollador de software y posteriormente líder técnico en pruebas de concepto para aprendizaje maquina en una empresa de servicios de paquetería internacional.

1.1 El “Gigante Azul”

Hay pocas empresas en el mundo con tanta historia y con tantas miras a futuro en el ámbito de las tecnologías de la información como donde laboré durante el periodo 2015 – 2016 , esta empresa tiene presencia en 170 países y cuenta con alrededor de 386,000 empleados y aun cuando la competencia con otras empresas como Amazon y Google le han relegado en ciertas áreas como la nube, sigue siendo conocida en la industria cómo “el gigante azul”, durante los ya casi cuatro años que he vivido en la ciudad de Guadalajara he podido ver de primera mano el cambio dramático que esta empresa ha realizado en su plan de negocios y la importancia de emplear todos sus esfuerzos en 5 áreas prioritarias conocidas cómo CAMSS (Cloud, Analytics, Mobile, Security and Social). Lamentablemente la gama de productos en los que me inicié trabajando en esta empresa eran difíciles de clasificar dentro de estas cinco áreas y más adelante abundaré en ese tema.

Siendo honesto antes de trabajar en “el gigante azul” de la tecnología había tenido un contacto escaso con sus productos, principalmente por su naturaleza privativa, en algunos casos conocí dichos productos antes de pasar a ser propiedad de la compañía como fue el caso de Racional R, software dedicado al diseño y la ingeniería de software que al pasar a manos del gigante azul cambió su nombre a Racional SA y del que me hice responsable, junto con otros productos, durante más de año y medio.

Cabe destacar que el gigante azul promueve entre sus empleados, desde hace muchos años, la creación de patentes, entre las más conocidas destacan los códigos de barras y últimamente las relacionadas con aprendizaje maquina, es tal el impulso que la empresa da a esta cultura de las patentes que para 2016 habría superado las 8000 patentes sólo en 2016 según sus propios comunicados de prensa.

Entre muchas de las virtudes del gigante azul están la invención en investigación relacionada con la salud y la industria alimenticia, uno de sus proyectos más ambiciosos y populares se hizo mundialmente famoso por sus resultados, la manipulación de átomos, un proyecto sin duda alguna excitante y con grandes aplicaciones dentro del sector de almacenamiento de datos y fue ampliamente difundida por un vídeo conocido como “The boy and his atom” (”el niño y su átomo”).

Se caracteriza también por ser una empresa socialmente responsable, una cualidad de la que pude ser testigo en persona y que gracias a esta política empresarial logramos apoyar a mucha gente necesitada, tuve la oportunidad de organizar un evento de voluntariado en el albergue Fray Antonio Alcalde a principios de 2016, patrocinado por la empresa y el club de empleados. Posteriormente dado el éxito del evento la empresa hizo un donativo de dos mil dólares al albergue y el club de empleados otros quince mil pesos, con este dinero el albergue logró reparar el motor de su frigorífico y realizar reparaciones importantes en las instalaciones y compraron medicamentos, mis compañeros y yo, unas 35 personas,

limpiamos a conciencia dicho frigorífico de dimensiones aproximadas a treinta y seis metros cuadrados.

Mis actividades dentro de la empresa eran en un principio de manera externa, en la figura de contratista externo por la empresa HITSS, inicié en actividades de soporte nivel 3, el más alto en la escala de resolución de problemas dentro de los productos y se enfocaba a la solución de defectos encontrados por usuarios, realizando modificaciones en código para defectos generales y particulares, estos últimos fueron en mi experiencia los más complicados por la dificultad para reproducir los defectos y el largo peregrinar de los procesos de aprobación para obtener información posiblemente confidencial del cliente.

El equipo al que me integré en el área de soporte estaba dirigido a una sección del core de ALM llamada JF, una solución que integran todos los productos que forman CLM y SSE, JF se definiría como el servidor y manejador de las relaciones en la base de datos de cada objeto que exista en el proyecto, así pues gestionaba la consolidación, almacenamiento y relaciones entre otros objetos, creando así tablas de objetos serializados llamados meta modelos, que se pueden describir como la representación de objetos serializados y las relaciones con otros objetos serializados que en su conjunto componen un a un objeto mayor, esto de manera escalonada, es decir, un objeto que está compuesto por otros muchos objetos a manera de muñeca rusa.

Seis meses después me integre directamente a la empresa en una posición diferente, ahora mi rol dentro del proyecto estaría ligado al desarrollo de herramientas internas y a las responsabilidades ligadas a la construcción y lanzamiento o build and release engineering para ALM.

Mis primeras obligaciones fueron aprender la arquitectura general del set de productos que integran el Rational ALM, crear herramientas para el equipo de ingeniería de construcción y lanzamiento, una de estas herramientas estaba enfocada al análisis de las

relaciones entre los productos y como se consumían entre ellos, esto debía desplegarse en una gráfica relacional de tipo globos, donde cada cuerda describía la relación y puntas de flecha el sentido de la misma,es decir, si se consumía o se aportaba a otros productos. Otro de los proyectos estaba enfocado a la preservación automática de los componentes del producto en cada hito que podían ser más de 5 al mismo tiempo con árboles de dependencias de más de 25 nodos repartidos en varios niveles. Esta última tarea se hacía manualmente, tomaba varias horas y su solución es el tema principal que se aborda en este reporte.

ALM es un set de varios productos individuales que se integran en uno solo, aunque podría resumirse en dos productos finales que trabajan juntos, CLM y SSE. CLM contiene el software relacionado con la parte funcional y SSE que se encarga de la gestión de negocio.

Los productos racional iniciaron enfocados al manejo de proyectos de software en lenguaje Java y contiene varias herramientas que permiten implementar metodologías AGILE, versionamiento de código y liberación continua, el continuo desarrollo y soporte a los usuarios. Sin embargo al estar ALM y sus derivados enfocados al cien por ciento a las tecnologías java enfrenta actualmente varios retos complicados en el mercado, uno de ellos es el alto coste de las licencias y capacitación para las empresas que adoptan inicialmente ALM, el costo de las licencias limita su uso a grandes empresas, por eso mismo el desarrollo constante estaba enfocado a cambios destinados a satisfacer las necesidades de los clientes con mayor cantidad de licencias.

A mediados del año 2016 la empresa comenzó un proceso de transición de los productos a dos empresas indias, en un plan de negocios donde ambas empresas se adquirirían al personal y se encargarían del soporte y desarrollo de los productos y el gigante azul mantendría la parte de facturación y conservaría las patentes, en un plan a 4 años en la que yo llamaría una adquisición suave. Finalmente se nos informó de las decisiones tomadas por las empresas y como nos repartirían entre ambas empresas al finalizar el mes de Septiembre.

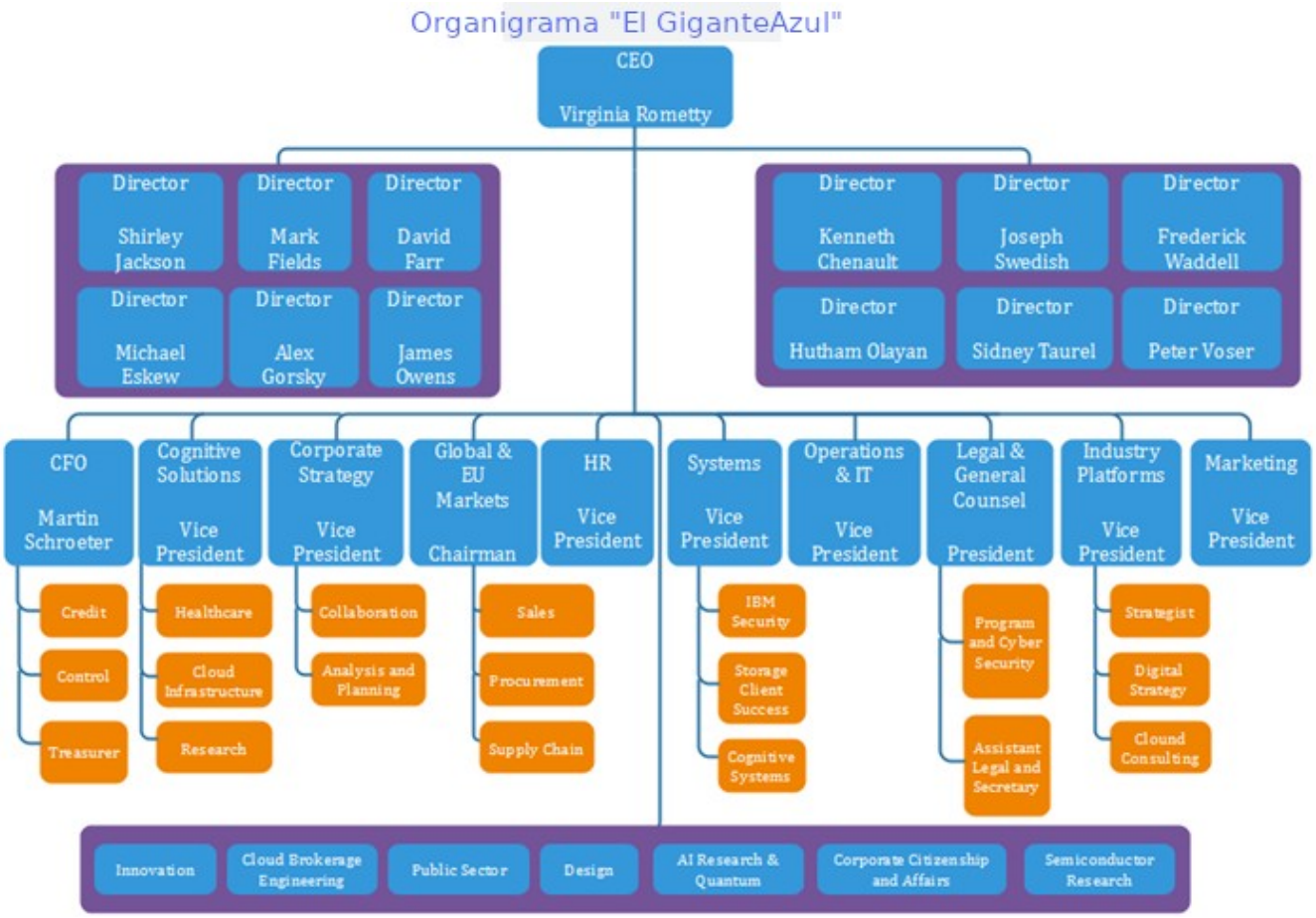
1.1.2 Misión

La misión corporativa de la empresa es ser líder en la creación, desarrollo y manufactura en la industria de las más avanzadas tecnologías de la información, incluyendo sistemas computacionales, software, redes, equipos de almacenamiento y micro electrónica.

1.1.3 Visión

La visión corporativa es ser la empresa más exitosa e importante en tecnologías de la información. Ser exitosa ayudando a sus clientes a aplicar tecnología para resolver sus problemas

1.1.4 Organigrama



1.2 Historia

El gigante azul es una empresa creada en el año de 1911 como (C-T-R) por Charles F. Flint, más tarde en 1914 se incorporaría Thomas J. W. Sr a la edad de 40 años como gerente general, ese mismo año implemento muchas de las prácticas que caracterizaron a la empresa como los amplios incentivos a los equipos de ventas y el famoso traje negro que portaban. En menos de un año Watson fue nombrado presidente de C-T-R y en los tres años siguientes la empresa expandió sus operaciones a Europa, Sud Africa, Asia y Australia.

En aquellos años C-T-R se caracterizaba por sus máquinas de escribir y contabilidad, el clasificador vertical y el tabulador, una maquina programable por tarjetas perforadas.

En 1915 la célebre frase de Thomas W. "THINK" se convirtió en el eslogan de la compañía, dos años más tarde C-T-R fue lanzada en la bolsa de valores de Canadá con el nombre por el que es conocida hoy en día.

En 1920 la empresa lanza al mercado la primera máquina eléctrica contable y para 1924 el nombre C-T-R es cambiado formalmente.

Durante la época denominada la Gran Depresión economía de Estados Unidos en los años 30 y a falta de demanda de productos la empresa cambió el enfoque de su modelo de ventas y enfocó todos sus esfuerzos en aprovechar la crisis y formar una división de investigación y desarrollo en Nueva York. Dicha inversión de tiempo, recursos y talento

fueron dieron frutos en los años 40, donde la empresa se convierte en el mayor proveedor de tecnología para el gobierno de Estados Unidos en materia de tecnología, durante el periodo de la segunda guerra mundial la empresa también participó en la producción de dispositivos que mejoraron la precisión de las bombas aéreas, rifles, piezas de motor y muchos más suministros de la primer guerra de proporciones industriales.

En esta época mientras se desarrollaba la idea de la primera computadora multi-proposito en Inglaterra por Alan Turing, en estados unidos el gigante azul desarrollaba la calculadora de control automático secuencial, para 1944 está máquina era capaz de ejecutar cálculos de gran tamaño automáticamente, la Mark I era una máquina de cinco toneladas capaz de hacer sumas en menos de un segundo, una multiplicación en seis y una división en 12 segundos, tiempos impensables en la actualidad pero que en aquella época se consideró el tope más alto en tecnología.

Fue hasta el año de 1949 que se desarrolló la primera calculadora electrónica programable por tarjetas perforadas.

La innovación en el mundo de la electrónica alcanzaron una nueva escala al implementarse tubos de vacío o bulbos para almacenar datos en memoria, en el año de 1952 la empresa lanza el modelo 701, la primer computadora basada en bulbos, capaz de ejecutar 17,000 instrucciones por segundo (las computadoras actuales rondan los 177,730 millones de instrucciones por segundo) .Ese mismo año Thomas Watson Sr. cede la dirección a su hijo Thomas Watson Jr.

En 1957 el modelo 305 innova con la primera memoria de acceso aleatorio llamada RAMAC o Random Access Method of Accounting and Control, dicha memoria funcionaba utilizando cintas magnéticas y se considera el primer sistema de almacenamiento de disco.

Este sistema de instrucciones sin embargo seguía siendo muy complicado de programar, en 1957 se pública FORTRAN (FORmula TRANSlation), el lenguaje de alto nivel patentado por el gigante azul y que satisfacía la necesidad del mercado de facilitar el tiempo de programación de los costosos equipos.

En los años 60 el gigante azul dominaba prácticamente todo el mercado de computadoras para uso empresarial, esta situación y el liderazgo de Thomas W. Jr. cambiaron radicalmente el modelo de negocios de la empresa, se estableció el modelo llamado "unbundled" que consiste en vender hardware, servicios técnicos y software en paquetes, es decir se pagaba por cada uno y se incluía en atractivos paquetes, esta estrategia le dio un empuje económico a la empresa sin precedentes.

A principios de los 70 Tomas W. Jr. deja la presidencia y la deja en manos de T. Vincent Learson y el a su vez en 1973 la dejaría a Frank T. Cary. En estos años las maquinas del gigante estaban profundamente arraigados en todas las industrias, desde cajas registradoras en tiendas de abarrotes y bancos.

Fue en esta misma época otra empresa tecnológica, Microsoft, iniciaría operaciones en la industria de software. En la década de los 1980 la nueva época de la computación iniciaba, el gigante azul creo su propia computadora personal (PC), con tan solo 16 Kb de memoria, un periferico para discos floppy y un procesador intel. Microsoft inició las negociaciones para proveer el sistema operativo, al mismo tiempo compraron el sistema operativo a otra compañía llamada Seattle Computer Products, el sistema operativo 86-DOS posteriormente fue llamado PC-DOS.

Conforme la PC se hacía más popular el mercado también cambió, la empresa llamada Columbia Data Products logró clonár exitosamente la BIOS de la PC, esto inició la producción de nuevos modelos de PC fura de la empresa y el principal beneficiado fue el proveedor del SO, Microsoft. El aliado formado por 32 personas ahora competía con el gigante.

En 1984 Macintosh anuncia su primera computadora personal y declara formalmente su competencia con el gigante azul. Ese mismo año el gigante responde con una infinidad de innovaciones, por muchas razones este año es un gran referente en la innovación tecnológica.

Por su parte el gigante contestó con cintas magnéticas con el doble de velocidad, una impresora de impacto, la tarjeta de crédito, su primer software de presentaciones Trintex y claro inició un programa de donación de equipos a escuelas y cursos gratuitos.

En la década de 1990 las computadoras personales eran comunes. El gigante enfocó, tal vez muy tarde, sus esfuerzos en los usuarios de equipos personales pero sobre todo se afianzó en el ramo empresarial que ya dominaba, esta década se consolidarían por sus productos empresariales y la producción de servidores de alta gama.

Lamentablemente no cambiar el modelo de negocios a tiempo llevó a la compañía a uno de sus momentos más obscuros al perder más de 8 mil millones de dólares en 1993, tal estrago causó la división de la compañía en negocios independientes y a un cambio de visión completamente enfocado al campo empresarial.

En 1997 para mostrar el potencial y los campos de desarrollo en los que se enfocarían en años posteriores el gigante azul hace gala de su proyecto Deep B, un desarrollo de machine learning que ese mismo año venció al campeón mundial de ajedrez Garry Kasparov.

Los avances en la investigación de la compañía eran prometedores, sin embargo gran parte de la década de los 90 las estrategias de negocio se enfocaron a la recuperación y reestructuración económica de la empresa.

Durante la primera década del siglo XXI la empresa conservó su presencia en el terreno empresarial, creando los servidores UNIX más veloces del mercado y las computadoras corporativas TinkP para uso empresarial. La principal meta de la compañía se ha vuelto fomentar en sus empleados el desarrollo de patentes, en el año 2002 se registraron casi 7000 patentes.

La serie de servidores más popular de eSeries, los Zseries son lanzados en 2003, las versiones más actuales de estos servidores se producen actualmente en la ciudad de Guadalajara México y uno de los mayores consumidores de los Zseries es Amazon en su división AWS.

1.3 Perfil actual

Actualmente la estrategia de la empresa es enfocar todos sus esfuerzos en áreas estratégicas denominadas CAMSS, compuesta por tecnologías en la nube, análisis de datos, dispositivos móviles, redes sociales y seguridad.

La competencia en estas áreas en el sector tecnológico es enorme, en el área de tecnologías en la nube su mayor competidor es Amazon, que también es uno de sus mayores clientes de servidores, con la aparición de docker, kubernetes y el concepto de micro servicios las tecnologías en la nube son parte de la tendencia actual y futura del desarrollo de software.

Uno de los productos estrella de la empresa es la inteligencia artificial utilizada para análisis de datos Watson, una de la herramientas empresariales más populares en el mercado.

En el campo de dispositivos móviles el producto principal es la herramienta de desarrollo de aplicaciones multiplataforma Blue M, crear aplicaciones en blueM no requiere conocimiento de ningún lenguaje de programación por lo tanto es increíblemente sencillo crear una aplicación móvil. Sin embargo la gran desventaja competitiva de blueM es su naturaleza privativa, el coste beneficio puede ser muy desigual para la pequeña y mediana industria generadora de aplicaciones.

Las ultimas ramas como redes sociales o seguridad no son menos importantes pero son sub productos que se agregan a las herramientas anteriores, por ejemplo contratando módulos extra de seguridad en un servicio de cloud o contratar la funcionalidad de red social en una aplicación móvil.

1.4 Descripción de sus productos y negocios.

La manera en la que la empresa desarrolla sus productos sigue un proceso de ingeniería y economía bastante complejo, tomando el ejemplo de una empresa de software prototipo que cuente con un departamento de desarrollo de software, finanzas, recursos humanos y legal, la relación entre dichos departamentos en un entorno ideal deberían hacer funcional a la empresa.

En empresas tan grandes como el gigante azul, se ha dividido en medida de lo posible cada departamento en mini departamentos, cada uno recibe un presupuesto anual y debe ejercerlo de la manera más eficiente posible, este presupuesto se utiliza para pagar a otros departamentos por sus servicios, por ejemplo el departamento de software le pagaría cierta cantidad a recursos humanos por lograr la contratación de un excelente ingeniero.

La problemática de mantener una gran cantidad de diferentes productos radica en la dependencia entre ellos, es decir, cuando un proyecto consume una característica de otro y este a su vez depende de otra característica de algún otro producto.

Un ejemplo sencillo sería pensar en tres diferentes herramientas, un manejador de base de datos diseñado para guardar objetos, una herramienta de java script personalizada y un generador de blogs de Internet.

En este ejemplo simple, el blog depende de un manejador de base de datos y la herramienta personalizada de java script, así que el equipo desarrollador del blog le paga al equipo del manejador de bases de datos y de java script por usar sur herramientas, incluso les pagan el tiempo invertido por la implementación de una característica específica, todo dentro de la misma empresa.

Este ejemplo explica de manera muy general la relación que tienen los productos entre ellos, creando un ecosistema que permite regular el número de productos, la inversión de tiempo y dinero. Esta estrategia genera un modo de soporte para tecnologías que siguen siendo utilizadas, si un producto se queda sin fondos es abandonado, eso significa que no necesariamente ya no es usado si no que ya no son requeridos cambios en el y se consideraría terminado y por otro lado mientras más demanda tenga un producto tiene más recursos para crecer el equipo de desarrollo o consumir tiempo y características de otros productos.

1.5 Descripción del producto específico

El set de productos en los que trabajé se engloba en dos paquetes llamados CLM y SSE, ambos muy relacionados y con pocas diferencias en lo general.

El ecosistema de productos relacionados en la generación de CLM y SSE suma 35 proyectos diferentes relacionados en una estructura de red, algunos proyectos se relacionan hasta con 15 proyectos distintos.

Estas relaciones son producto de un arduo trabajo de arquitectura de software en programación orientada a objetos y sus características a gran escala, una visión de la ideología OAOO (*Once And Only Once*) "una vez y sólo una vez" y DRY (*Dont Repeat Yourself*)

"no te repitas a ti mismo", conceptos que aplicados a una lógica de negocio resultan ser muy convenientes en cuestiones de ahorrar tiempo y dinero.

CLM y SSE son un conjunto de herramientas para manejar equipos y proyectos utilizando versionamiento de código y herramientas de SCRUM, SCRUM son un conjunto de practicas y roles en un equipo para desarrollo de proyectos "ágiles".

Las características generales de un proyecto que utiliza metodologías SCRUM es la división de tareas en equipos independientes, flexibles y auto dirigidos. Cada día los equipos realizan juntas para resolver problemas, reportar bloqueos o dificultades y medir la capacidad del equipo para completar sus metas. Dichas prácticas no son únicas del desarrollo de software y su eficacia radica en la comunicación efectiva, las habilidades de liderazgo y la motivación de los equipos.

CLM inició como una herramienta que facilita las actividades SCRUM en proyectos de software en lenguaje Java, sin embargo el cliente con mayor número de licencias de la herramienta hoy en día utiliza las herramientas para el desarrollo de diseño de piezas automotrices y sistemas embebidos.

Las herramientas para el desarrollo de un proyecto de software en CLM siguen un ciclo de vida llamado iteración que inicia a partir de la iteración anterior, se definen las metas por equipo y se crean espacios de comunicación para cada una, otros espacios de comunicación sirven para el reporte colectivo de un equipo especifico y las áreas generales para recopilar el estado de todos los equipos.

Las herramientas de comunicación sustentan la implementación de SCRUM, sin embargo el poder de estas prácticas en el desarrollo de software se hacen visibles con la integración de RTC. RTC es un entorno de desarrollo de software basado en Eclipse que integra versionamiento de código ligado a las líneas de tiempo de discusión o tareas. Éste concepto es útil ya que se puede seguir los avances en tiempo real de las metas.

Una tarea consta de una descripción de la tarea a realizar, el estado de la tarea, una línea de tiempo que contiene comentarios asociados a la tarea y el código especifico.

Los estados posibles de una tarea pueden variar dependiendo de la configuración del proyecto, los estados por defecto son una descripción clara del estado de la tarea, sea que aún no se ha iniciado, si esta en progreso, se ha bloqueado o se ha pospuesto y si ya ha sido terminada.

Al marcar la tarea como realizada el código asociado se considera listo para ser integrado en el proyecto y al mismo tiempo el equipo puede ir midiendo el avance de las tareas realizadas en la iteración. Una característica interesante de ligar el código con las tareas es que al pertenecer una tarea a una iteración concreta se puede crear código para integrarlo varias iteraciones en el futuro.

Otra de las filosofías implementadas en CLM es la entrega continua, este concepto se describe como la integración continua del código, la compilación y ejecución de pruebas automatizadas, cabe destacar que CLM se construye utilizando las mismas herramientas de CLM, así como toda la familia de productos Racional. CLM implementa integración continua dos veces cada 24 horas, esto implica que todos los productos asociados son integrados, copilados y probados automáticamente en un proceso que toma al rededor 6 horas y en que se detectan posibles problemas en pruebas o tiempo de compilación continuamente, a este proceso se le llama también construcción y lanzamiento (construir y liberar).

Si un error se detecta durante este proceso las herramientas de integración sirven para encontrar la pieza de código involucrada, el autor y la tarea relacionada.

2 ANTECEDENTES

La desaparición de un sentido
de responsabilidad es la consecuencia
de mayor alcance de la sumisión
a la autoridad.
-Stanley Milgram

2.2 Problemática del producto

Uno de los principales retos en el proceso de desarrollo de software es el tiempo que se invierte en la creación de nuevos features (características), en proyectos en constante desarrollo se aspira a lanzar versiones nuevas en fechas programadas, en el proyecto que es el caso de estudio de este trabajo se implementa estrategias de desarrollo de productos de software conocida como lanzamiento continuo e integración continua.

Lanzamiento continua es la estrategia de desarrollo de software que impulsa la creación continua de nuevas características, reparación de errores, todos los proyectos de software exitosos y en continuo crecimiento deberían aspirar a la mejora continua que ofrece una estrategia como entrega continua, esto implica una integración continua de nuevas líneas de código y una construcción del producto final la mayor cantidad de veces posible dentro de un criterio razonable entre cada iteración, esto implica compilación y pruebas automatizadas.

Las características que ofrece un proceso de entrega continua o continuous delivery son:

- Bajo riesgo en la entrega de producto
- Lanzamiento rápido al mercado
- Mejor calidad
- Menor costo

El éxito de la estrategia se sustenta en tres conceptos principales: administración, integración y pruebas.

Uno de los aspectos más importantes de la administración es el diseño de patrones de implementación basados en entregas continuas, una tarea que normalmente se sustenta en los requerimientos deseados y se proyecta a través de los arquitectos de software, en esta etapa de diseño se debe mantener en todo momento el concepto de escalabilidad reutilización de código y en algunos casos la integración de productos ya existentes. Es precisamente esta característica del producto en el que trabajé el tema principal de este reporte, la complejidad del desarrollo de un producto se incrementa a medida que se agregan otros productos de los que se depende directamente y estos a su vez replican el mismo modelo de dependencia a otros productos, con el paso del tiempo se vuelve muy complicado establecer con exactitud de donde proviene una característica integrada des de otro producto.

Este problema podría ser trivial si se cumple el objetivo final, un software estable, escalable y seguro, sin embargo, se había convertido en un verdadero problema desde el punto de vista de negocios.

2.3 Problemática del entorno empresarial

Las empresas con un volumen considerable de productos de software que implementan arquitecturas de software dependientes de otros productos necesitan forzosamente una estrategia para justificar y mantener productos que no se ofrecen directamente al usuario final pero que resuelven problemas puntuales que otros productos que sí logran generar recursos consumen. Sin embargo, como se habló en el capítulo anterior era muy complicado detectar de donde venia ciertas dependencias originalmente, esto acarreaba un problema administrativo grave porque no se podía garantizar que los equipos que desarrollaban o mantenían ciertas características comunes a muchos productos recibieran los recursos suficientes.

El ciclo de vida de un producto se ligaba directamente al modelo capitalista, cada año cada producto recibía una cantidad de recursos económicos virtuales, una cantidad en papel del presupuesto anual en dólares, con ese presupuesto debían pagar los servicios de desarrolladores, ingenieros de pruebas, arquitectos de software, constructores, ingenieros de soporte, gerentes, renta de los lugares físicos, renta de servidores, licencias de software y el pago proporcional a otros productos por el uso de características integradas.

De esta manera se buscaba que los productos más exitosos recibieran un mayor presupuesto anual y que los proyectos de los que eran clientes se mantuvieran sanos y fueran sustentables.

Sin embargo, cada producto buscaba su propio nicho en el mercado, aunque con fines diferentes los productos más grandes y con alta visibilidad en el mercado consumían muchas características de otros productos de alta visibilidad, el mismo mercado les exigía un acoplamiento ideal.

Esto propició un problema que con el paso de los años se agravo, el producto *"A"* de alta visibilidad recibió el requerimiento de incorporar una característica del producto *"B"*, también de alta visibilidad, que les permitiría trabajar con un acoplamiento ideal. Una vez el producto *"A"* se acopló a la característica de *"B"* en la versión *v.1* se identificó que la característica era en realidad desarrollada por el producto *"C"* y que actualmente se encontraba en su versión *v.3,* sin embargo, el acoplamiento exigía utilizar forzosamente la característica de "B" así que se documentaba la referencia directa a "B".

En el mejor de los casos "A" y "B" adoptaban la última versión desarrollada por "C" y ambos se referenciaban a "C", pero esto raramente pasaba, con el paso del tiempo algunas características se quedaban en una versión de manera estática, esto era grave porque precisamente descubría una debilidad en el modelo de integración continua, los tres productos al estar bajo este modelo debían seguir desarrollando sus productos, muchas veces esto se guiaba a los requerimientos de los clientes.

En estas circunstancias el producto "C" al no estar referenciado directamente por "A" solo podía recibir recursos de "B" si no se hacía un seguimiento detallado del origen de la característica.

En la gran mayoría de los casos el sistema funcionaba muy bien, la comunicación entre los diferentes equipos era fundamental para poder llevar acabo la documentación y las relaciones entre los productos, en este caso todo este proceso se gestionaba en dos áreas, una de ellas poco común y que nace de la necesidad de solventar estos y otros problemas de sustentabilidad en el modelo de entrega continua, estas áreas son: Arquitectura e ingeniería de construcción de software.

2.3.1 Ingeniería de construcción y lanzamiento de software

El área de build and release engineering es un departamento que se creó con la finalidad de gestionar la implementación del modelo de continuous delivery, generando, monitoreando y gestionando automatizaciones para las integraciones, compilaciones, pruebas y crear las versiones estables y finales en cada entrega.

Otro de los aspectos importantes que se realizan este departamento es la documentación de dependencias entre productos para generar los recursos adecuados para cada producto y sus dependencias.

Esta última tarea le valía al departamento de build and release engineering una importancia significativa, ya que no solamente se gestionaba la calidad del producto y se implementaban todas las partes que integran el modelo de continuous delivery, sino que también se influía en la toma de decisiones gerenciales para el manejo del presupuesto y estrategias al mapear las dependencias adecuadas entre productos.

El principal problema del área de build and release engineering fue uno de los principales problemas que aquejan a los proyectos que alcanzan con el tiempo cierto tamaño, los procesos descontinuados o antiguos, también llamados legacy. Una de las creencias no escritas de sistemas legacy es “si funciona no lo muevas”.

Esto es paradójicamente una buena estrategia de sentido común y por otra parte una advertencia mal entendida que al saberse inamovible poco se puede hacer, olvidándose poco a poco su función, proceso y el problema original por el que fue creado, con el paso del tiempo los involucrados, como es natural, continúan rotando hasta que no queda nadie de los involucrados en el proceso inicial.

Esto también se reflejaba en los procesos de referencias entre productos, solo unos pocos conocían la función de las referencias y al existir un ecosistema de más de 50 productos relacionados entre ellos con miles de características cada uno la tarea se había vuelto titánica.

Al no tener la certeza de las implicaciones de abandonar las características de un producto o los orígenes reales de las características adoptadas los productos “grandes” decidían mantener a todos los productos del ecosistema, esto sin duda repercutía en el presupuesto del mismo proyecto y de los productos a los que se debía canalizar más presupuesto.

Administrativamente era insostenible, cuando me uní al proyecto el número de desarrolladores en los proyectos principales era muy bajo.

3 DEFINICIÓN DEL PROBLEMA

Hay Muchos mundos y
muchos sistemas de
universos que existen
todos al mismo tiempo,
todos ellos perecederos.
-Anaximandro

El área de ingeniero ingeniería de construcción y lanzamiento es la encargada de generar el producto final para los clientes. Entre sus tareas esta realizar un seguimiento de los requerimientos planeados en cada versión, colectar el código fuente específico para la versión, vigilar y detectar la compilación del producto, ejecutar pruebas automatizados para estas versiones del producto, proveer un diagnóstico rápido y preciso del estado del producto diariamente.

Ingeniería de construcción y lanzamiento es un vínculo entre el área de gerencia con el área de desarrollo de producto, la principal tarea es mantener informada a la gerencia para que esta a su vez pueda tomar decisiones informadas y mantener a los desarrolladores en contacto con otros equipos y productos en el momento oportuno para garantizar la salud del producto.

El reto principal es conocer las diferentes herramientas y equipos de desarrollo que conforman uno o varios productos para encontrar y corregir problemas en un modelo de entrega continua.

3.1 Entrega continua

Entrega continua (continuous delivery) es un modelo de trabajo en desarrollo de software donde se estipulan meas concretas en tiempos cortos, en el caso concreto de la empresa el modelo de integración continua era una de las piezas clave en la metodología ágil.

Cada día el equipo de desarrollo tenía una reunión de no más de 15 minutos donde cada uno de los miembros hablaba del estado de sus tareas actuales, si iban conforme a lo planeado se establecía en estado verde o por el contrario podía tener un bloqueo o impedimento, en cuyo caso se debía tratar en una junta posterior con las personas interesadas en ese tema particular. Estas reuniones eran parte de un marco de trabajo ágil llamada SCRUM.

La manera en la que las diferentes tareas se integraban al proyecto era utilizando una herramienta de software que administraba las versiones mediante un complejo sistema de planeación, es decir, si una tarea se planeaba para la versión 2.0 entonces se creaba un entorno específico para el desarrollo de la versión 2.0 con las condiciones iniciales del ultimo lanzamiento de la versión 1, también se generaban tareas de integración que serían las que mantendrían el seguimiento general de las tareas que debían integrarse y las tareas específicas harían el seguimiento por versiones de dicho código.

La herramienta que utilizábamos para esta tarea era también el producto en el que trabajábamos por lo tanto teníamos una buena idea de primera mano sobre el uso y la experiencia de uso de esta. La integración continua era muy sencilla cuando se desglosaba en pequeñas partes y se permitía realizar pruebas generales en un build antes de integrar a producción.

3.2 La complejidad de diseño

Por lo general un producto de software empresarial está formado por varias herramientas integradas que resuelven varias necesidades. En el caso del producto el que trabajábamos esta herramienta era la combinación de más de 50 diferentes herramientas, muchas de ellas aun en producción y generando versiones nuevas continuamente.

Parte de las tareas del equipo de construcción y lanzamiento de la empresa era asistir a las reuniones diarias de cada producto para presentar un informe relacionado con los resultados generales del código entregado a producción al final del día anterior.

La tarea principal era crear una versión del producto final dos veces al día, esto implicaba recabar el código de cada herramienta, compilarlas en un orden especifico, ejecutar pruebas automáticas y publicar internamente el resultado. Si existía alguna anomalía debía reportarse directamente con la persona responsable de la falla y con los dueños de los productos para que estuvieran enterados.

Integrar más de 50 productos en uno solo era una tarea de ingeniería aplicada con mucha complejidad y que se había logrado hacer funcionar correctamente con el esfuerzo de muchas personas.

Las relaciones entre productos solían ser complicadas ya que una herramienta A dependía de otras 3 de un producto X y este a su vez de dos herramientas del producto Y, el producto B de una de Y y cuarto de Z. Al final A y B se integraban en una herramienta concreta.

La relación anterior se había vuelto compleja por la cantidad de diferentes herramientas entrelazadas. Y aún más complejo cuando estas herramientas se entrelazaban en versiones diferentes, es decir, si regresamos al ejemplo anterior donde la herramienta A depende de X y Y,

Ahora sea la herramienta A en su versión 1.1 dependiendo de la herramienta X v3.1 y Y v2.6 pero B depende de Y v2.2 y Z v1.3.

La complejidad se convierte en un verdadero problema cuando las dependencias de A y B a Y se diferencian por la versión de Y. En algunos casos estaba bien justificada dicha discrepancia, pero en algunos casos simplemente no estaba justificado que una herramienta utilizara una versión reciente de otra solo para satisfacer las necesidades de una tercera.

3.3 El negocio e integración continua

El problema de la complejidad de dependencias escalaba a medida que pasaba el tiempo, el modelo de integración continua en cada equipo les empujaba a tener versiones nuevas continuamente y a actualizar características y enchufes continuamente, reparar el código con errores y remplazar módulos completos en algunas situaciones muy especiales.

El modelo de negocios de la empresa se basa en un comercio interno muy interesante que le garantiza la salud económica a muchos productos, este modelo consiste en el pago interno por servicios.

Muchas herramientas tienen un mercado limitado, al agruparlas en un producto final la cantidad potencial de clientes aumenta y la entrada económica para re inversión en desarrollo y las ganancias de la empresa mejoran. Era así como el modelo de negocios del producto repartía las ganancias de una licencia con los diferentes equipos relacionados, en este caso con las 50 herramientas involucradas. Esto a su vez elevaba el coste de las licencias y por el otro garantizaba que todos los equipos siguieran produciendo versiones nuevas y sobre todo dando soporte al código.

La herramienta llegó a tener momentos muy prósperos donde el reparto de ganancias a los equipos era proporcional a su participación en cada una de las herramientas, sin

embargo, el mercado cambia constantemente y la aparición de herramientas similares por otras compañías y de código abierto rompieron de tajo con el dominio indiscutible del mercado de una solución empresarial tan sólida.

Como era de esperarse, siempre que terminan los tiempos de prosperidad se pone mucha más atención a las fugas de capital y sobre todo en empresas altamente organizadas se debían entregar resultados concretos para mejorar la administración.

3.4 Herramientas de análisis en la mejora de producto

Cuando inicie a trabajar en el equipo de ingeniería de construcción y liberación tenía apenas seis meses trabajando en la empresa como desarrollador de software en el área de soporte nivel tres para la herramienta base del producto, donde se manejaba la base de datos que controlaba cada característica, cada dato y cada etiqueta relacionada con el vercionamiento de código.

Era una herramienta fascinante, aprendí el concepto de un meta-meta módelo, un concepto abstracto con el que operaban todas las herramientas que podían modificarse por los usuarios.

Esos seis meses había trabajado para una empresa consultora externa y para el final de ese periodo me ofrecieron una plaza estable en la empresa, pero en un proyecto distinto, ahora sería parte del equipo de ingeniería de construcción y liberación, un equipo conformado por ingenieros con mucha experiencia y que habían participado en varios proyectos diferentes, mi tarea era generar herramientas para analizar la composición del producto final a detalle.

Aunque los componentes y productos necesarios para generar el producto final de inicio eran transparentes, el mapa estaba incompleto, ya que solo se conocían las características y

enchufes que venían desde la última capa de productos relacionados en muchos casos esas características y enchufes venían de un nivel aún más profundo.

Mi tarea era hacer un mapa de cómo estaban relacionándose las diferentes herramientas y al mismo tiempo identificara cada uno por versión en cada diferente producto.

La meta era que este mapeo les diera una mejor perspectiva a los arquitectos del producto para mejorar la relación entre productos y al mismo tiempo dar una visión más clara a la gerencia de cómo estaba compuesto el negocio y validar o no la estrategia que se estaba adoptando respecto al manejo de los recursos destinados para cada equipo.

El problema entonces se planteaba como un reto para la sostenibilidad administrativa de los productos, la detección puntual de los proveedores de características era prioritario para realizar una estrategia que definiera el camino de todo el ecosistema de productos.

3.5 Estrategia

Los cuatro productos principales y con mayor cantidad de dependencias se proporcionan a los clientes en dos diferentes presentaciones, una relacionada con el ciclo de vida colaborativo y otra más robusta que englobaba el manejo completo del ciclo de vida de una aplicación, es decir, una parte parcial que permitía generar repositorios y versiones de código ligado a alguna estrategia de SCRUM y la más completa con herramientas adicionales para la detección de nuevas características y manejo del negocio de desarrollo de aplicaciones en Java e integraciones con otras tecnologías de la empresa bajo licencia.

Antes de proponer una solución debía informarme del proceso completo de construcción del producto, sobre todo cómo funcionaban los conectores que permitían compartir características entre productos, esta técnica de escalamiento de código es común hoy en día con diferentes herramientas como Maven o Greadle. La herramienta de la empresa

funcionaba muy similar, con la particularidad de ser software propietario escrito en C sin más documentación que el mismo código.

Un colega que seguía de cerca mi investigación me hizo una recomendación muy acertada que me hizo cambiar la estrategia de inmediato, me comento que dicho software escrito en C había sido escrito hace varios años atrás sin recibir mantenimiento y que había sido escrito por un grupo de desarrolladores durante 2 años, me aconsejo que tomara una ruta diferente para encontrar las dependencias de los productos utilizando solo los archivos de definición que mostraban dependencias directas a otros productos.

Los archivos de definición de dependencias eran una solución que permitía acoplar productos de manera sencilla pero que también era uno de los motivos por los que registrar las dependencias exactas era tan complicado, sin embargo esos archivos de dependencias contenían información relacionada con el resultado de construcción de ese producto y por ende a los enlaces de resultados de los productos a los que dependía, otra información valiosa era el nombre de los enchufes y características que se utilizaban y una marca interna para conocer si eran creados en ese producto o eran consumidos.

En ese momento inicie el desarrollo de una solución que navegará entre todos los enlaces relacionados con la construcción principal, listar las características y enchufes en cada uno y analizar la fuente original. Una vez inicié con esta tarea le reporte a mi gerente lo que estaba haciendo y le pareció que mientras recorría todos los productos para analizar las dependencias también tuviera la capacidad activar el mecanismo de preservación de los builds, esto se hacía en cada ralease de manera manual cambiando el valor de una etiqueta en XML dentro de la definición del resultado de construcción, esta etiqueta le indicaba al proceso orquestado escrito en C no borrar las construcciones, estas medidas de borrado se deben a la capacidad limitada de los discos y el enorme tamaño de las construcciones.

Al investigar las posibles soluciones para cambiar dicha etiqueta dentro del archivo de resultados de construcción (build results) y hacer las pruebas necesarias para avanzar entre los diferentes productos descubrí que no era posible cambiar el etiquetado a través de una dirección HTTP a un servidor HTML y que algunas de esas direcciones estaban restringidas y no tenía los privilegios suficientes para visualizar el contenido de los resultados.

En ese momento busque la asesoría de uno de los miembros más experimentados del equipo en Canadá, gracias a la información de infraestructura proporcionada por esta persona me fue posible mapear cada dominio de producto con su localización física real dentro de los diferentes servidores, ahora podía orquestar un análisis desde dentro de cualquiera de los servidores a los que tuviera acceso con el usuario funcional de construcción.

El usuario funcional de construcción era una cuenta con privilegios de escritura en los directorios de cada servidor donde se generaba un construcción, esta cuenta era compartida para todo el grupo de construcciones para generar automatizaciones aún que en la mayoría de los casos se utilizaba para entrar a los servidores vía SSH parar alguna anomalía con los servidores de construcción y para preservar las construcciones de productos en cada lanzamiento.

Un módulo adicional en el software me permitió leer en todo momento la última versión del mapa de servidores HTML y las direcciones físicas de los servidores y sus directorios, para permanecer con la versión actual de los construcciónes en todo momento. Una vez mapeadas las direcciones físicas inicie con las pruebas de software tocando cada uno de los productos, mapeando sus componentes, características y conectores, clasificando las dependencias externas y nativas. En estas primeras pruebas no se intentó la modificación de las etiquetas de preservación sin embargo si se debía tener acceso al archivo, solo como prueba de que era accesible, este modo se agregó dentro de las configuraciones del software con la etiqueta *-e o —effort*.

4 METODOLOGÍA

La prueba de todo conocimiento
es el experimento.
El experimento es el único juez
de la verdad científica.
-Richard Feynman

4.1 Antecedentes

La meta por alcanzar sin duda es ahorrar dinero a la compañía, sin embargo, el problema requiere una evaluación del estado actual que ayude en la toma de decisiones a nivel gerencial.

El valor de los resultados debe ser correctamente expresado en gráficas y el formato debe facilitar que pueda ser compartido con los directivos y gerentes del producto, con esto en mente se pensó en una metodología que abarcara dos ejes importantes: Analizar el estado de las dependencias entre los productos y al mismo tiempo generar graficas fáciles de entender para los involucrados en el proyecto.

La manera de construir los productos implementa herramientas desarrolladas internamente que a partir de un archivo de definiciones mapea las dependencias de cada producto y los productos que consume, ese archivo es la primera línea de entrada para saber que dependencias tiene cada producto y poder analizar cada uno hasta su raíz. La siguiente figura muestra este proceso.

Éste archivo de propiedades se define como el punto de montaje para la creación de builds en los dos productos principales. Lo que se obtiene de estos archivos es un mapeo de relaciones entre los productos que deben ser analizados para iniciar la investigación y mapear las relaciones.

Sin embargo, aún que se definían los nombres de los productos no se tiene más información relevante relacionada con la versión especifica que consume una versión en concreto, para resolver este problema se analiza el build result.

El build result es un reporte sobre el estado de un build, en primer instante reporta si el build ha sido o no exitoso, posteriormente un estado por cada componente que lo integra el y los builds que ha consumido para crearse.

Esta es la pista correcta para encontrar las versiones utilizadas para cada build ya que casi todos los productos se desarrollan continuamente y generan dos o tres versiones cada 24 horas. En acuerdos anteriores y para salvar espacio en servidores solamente 5 builds no preservados se mantienen en disco, y se van eliminando conforme se genera un uno nuevo.

Aquí nace otra necesidad para el negocio ya que cada vez que se define a una versión como release todas las dependencias deben preservarse, en ese momento el proceso era manual y se requería la intervención de un ingeniero en Canadá que pasaba alrededor de 3 días aplicando las modificaciones para preservar los buids.

Como ya se ha mencionado anteriormente el proceso de build da como resultado un paquete de software listo para instalar y un archivo build result, es en este archivo de build result donde se obtienen las versiones de las dependencias directas, el build result es la representación en html del verdadero archivo de estado del build en formato xml, uno de los tags internos hace referencia a la preservación del build. Una vez que el proceso de build inicia lo primero que hace es verificar la pila de builds, si la pila ya está en su límite, borra el más antiguo y posteriormente inicia la construcción del nuevo, esta pila de builds solo toma en cuenta aquellos builds que no están etiquetados para ser preservados.

Los buid results son consultados a través de un servidor web, es por eso por lo que parte de las mejoras a este proceso fue agregar la funcionalidad de cambiar el estado de la etiqueta con una casilla de selección, sin embargo, seguía siendo una tarea muy tardada e innecesariamente manual.

El proceso de preservación de builds y el mapeo de dependencias están ahora ligados por un procedimiento en común: la obtención de las dependencias de manera recursiva. Sin embargo, intentar preservar los archivos utilizando el servicio de hiper texto seguro (https) no era posible, los builds están alojados en servidores diferentes y por lo tanto sistemas de autenticación de usuarios distintos. Otra característica importante son los parámetros de seguridad configurados en la red para evitar este tipo de accesos automatizados desde la red interna.

Los servidores físicos compartían una cuenta de usuario común ligada al proceso de los builds. Ya que el firewall permitía conexiones seguras utilizando el protocolo secure shell (ssh) se planteó la posibilidad de hacer los cambios automáticos a través de este servicio.

El siguiente paso fue mapear las direcciones de dominio web con las de los servidores físicos, para este paso fue necesario contactar al administrador de los servidores web e iniciar una captura de la correspondencia a cada uno.

Ejemplo:

https://servidorA.aurora.com/ProductNameX/buid : /cartagena/ProductNameX/build

Donde *https://servidorA.aurora.com/ProductNameX/buid* es la dirección al servicio http en un servidor web y */cartagena/ProductNameX/build* la dirección dentro del servidor donde se guardaba el archivo html.

4.1.1 Servidor WEB (HTTP)

HTTP corresponde a las siglas de Hypertext Transfer Protocol o protocolo de transferencia de hipertexto, en los principios del internet y antes del estallido de frameworks de desarrollo web, las escasas páginas web que existían pertenecían a el ejército de estados unidos y algunos centros de investigación que compartían información de manera remota. Con el paso de los años se fueron elaborando diferentes metodologías y

estándares para la transmisión de información, estos estándares se catalogaron como protocolos de comunicación, es decir, si se quería transferir información y consumir un servicio entre dos máquinas, ambas debían seguir unos protocolos específicos.

En los años 80 se habían creado ya varias soluciones o protocolos por diferentes fabricantes, desde el nivel físico hasta las aplicaciones, como páginas web o clientes de música como Spotify. A esta pila de soluciones divididas por capas de abstracción se le llamó modelo OSI.

El modelo más moderno y utilizado es el modelo TCP/IP, el modelo OSI o Open System Interconnection (Modelo de Interconexión de Sistemas) es usado muchas veces para ejemplificar el nivel en la capa de abstracción para un protocolo, este protocolo también es conocido como DoD, el protocolo de internet del departamento de defensa de estados unidos ya que la mayor parte de su desarrollo fue elaborado en DARPA (Agencia de Proyectos de Investigacion Avanzada de Defensa) desde los años 60. El modelo OSI consta de siete niveles divididos en dos categorías. La primera categoría agrupa las capas de medios o media layers y esta a su vez se compone de las capas física, enlace de datos y determinación de ruta. Estas capas se relacionan con la comunicación a nivel de routers y servidores, en resumen, se encargan de que los paquetes lleguen y sean devueltos de la mejor manera posible.

La segunda clasificación, llamada capas de cliente se compone de capas enfocadas a preservar los bits de los mensajes hasta la presentación de los datos. Las cuatro capas son: transporte, sesión, presentación y aplicación.

Es esta última capa la que nos interesa para describir el servicio de HTTP ya que pertenece a la capa de aplicación y trabaja con transacciones de petición y respuesta, una respuesta contiene un código de servidor de la respuesta, la URL y la versión de HTTP con la que se esta comunicando.

Las peticiones contienen un método de petición, los más comunes son get, post, put y delete, seguidos de la URL y la versión HTTP con la que se comunica el cliente.

4.1.2 Protocolo SSH

El protocolo SSH viene de las siglas de Secure Shell, o shell seguro. Un shell es una interfaz de usuario, este concepto engloba desde el escritorio de un sistema operativo hasta la línea de comandos. En el caso de SSH la interfaz principal se presenta en la línea de comandos y se utiliza principalmente para crear canales seguros en una arquitectura cliente servidor.

SSH v2 genera encriptación basada en llaves públicas y privadas, por lo que podrían no necesitar una contraseña per si un archivo relacionado con la llave de conexión, en un ejemplo donde maquina A y maquina B desea comunicarse usando un protocolo SSH, a genera una llave privada y otra pública, comparte su llave pública con B, incluso podría crear una contraseña para esa llave pública. A comparte la llave pública con otras máquinas como C y D, y todas podrían acceder a A.

El método de seguridad de las llaves primarias está basado en el protocolo RSA (siglas de los nombres de sus creadores, Rivest, Shamir y Adleman) que al multiplicar un par de números primos es capaz de crear una llave privada, compartiendo uno de los números como llave pública. Su eficacia raca en crear, validar llaves y la dificultad de crear una falsa autorización ya que la factorización de números muy grandes. Aunque la factorización no es sumamente complicada si lleva mucho tiempo, las técnicas de cómputo clásico para la factorización de números primos se lleva acabo identificando la periodicidad en series de valores, esto para reducir el número de valores posibles en la factorización.

SSH permite autenticación en máquinas remotas por terminal, *tunneling*, una técnica muy utilizada para enviar datos privados en redes públicas, su implementación más conocida es en el protocolo HTTPS, también se implementa en *forwarding*, una técnica de mapeo que permite enmascarar los servicios de un servidor a través de una maquina diferente, muchas veces se utiliza para dar acceso a IPs externas en un servicio específico en nubes privadas. Otra implementación común es la transferencia de archivos mediante SCP (Protocolo de copia segura), una manera mucho más segura de envío de archivos que FTP.

4.2 Metodología implementada

Para obtener un análisis de las dependencias entre productos y las gráficas relacionadas se desarrollaron dos programas. El primer programa tenía como objetivo extraer la información de dependencias en cada build result y la posibilidad de activar la bandera de preservación.

4.2.1 Resultado de construcción

Los buid results son archivos xml con una presentación web que permite visualizar los componentes que integraron al build en la primera columna y cada columna representa el estado de la operación realizada en dicho componente, la mayoría de las operaciones estaban relacionadas con pruebas, sin embargo, la más importante era la compilación del componente.

4.2.2 Errores en el resultado de construcción

En este punto la primera estrategia planeada fue identificar cómo funcionaba actualmente el proceso de preservación. En la presentación de resultados del build result se había incorporado un botón que al activarlo cambiaba cuerpo de la petición web del

botón de preservación y replicarla, sin embargo, los firewalls de la empresa estaban configurados para detectar y evitar el envió automatizado de formularios y la gerencia no quería correr el riesgo de lanzar alarmas innecesarias. Esta limitante no parecía ser un impedimento sin embargo uno de los miembros de infraestructura del equipo aprovecho la tarea para que se creara un registro de los servidores web y su dominio ya que no se habían documentado antes para nuestro equipo.

Existían varios servidores de administración con directorios conectados remotamente a los servidores web y a los archivos que se desplegaban en el servidor web. Estos servidores fueron muy importantes en la identificación de las rutas web de cada servidor.

Me proporcionaron un archivo CSV con dos columnas: Acrónimo del Producto y Servidor. Éste archivo es el punto de entrada para la modificación de los archivos xml, el siguiente paso era terminar el mapeo del archivo a los dominios necesarios. Ambas rutas tenían un punto raíz correspondiente a su sistema de archivos en el caso del servidor web y otro correspondiente al dominio y el resto de la ruta era exactamente la misma.

Una vez que se revisaron varios casos donde los patrones fueron identificados realice un programa que leyera las dependencias de un build result y a través de expresiones regulares los identificara con el acrónimo del producto y el build result de dicha dependencia a través del dominio web.

En este punto el software podía recorrer un build result y los build result relacionados, extrayendo información de los nombres del producto y los dominios relacionados con cada uno, el siguiente paso fue mapear usando diccionarios para completar el archivo CSV original y agregar la tercera columna, para dar como resultado las columnas de Acrónimo Producto, Servidor WEB, DNS.

Ahora que se tenía la capacidad de moverse entre los build results y los build results que consumía el siguiente paso era utilizar la tabla CSV ahora con los dominios para utilizarlos y preservarlos en el servidor web, es decir una vez que se seleccione un build result para

preservar esta acción debe ejecutarse en cascada hasta el último nivel de dependencia alcanzable.

En este punto la información de cada build ya era guardada en una estructura de datos que contenía información de cada build result. La información de los builds se guardaba en un diccionario y posteriormente se guardaba y versionaba en un archivo JSON.

En los primeros intentos por recorrer los builds existieron diversos errores relacionados con expresiones regulares ya que las direcciones a builds relacionados no apuntaban directamente al buid result si no a un archivo en la misma ruta que describía el nombre del build y un par de líneas adicionales que no eran relevantes para el proceso.

Así que se realizaron modificaciones para apuntar correctamente a los buid results de cada producto utilizando expresiones regulares. Me vi en la necesidad de desarrollar mi propia herramienta para limpiar y extraer información de URLs y directorios ya que la empresa no permitía el uso de librerías ni herramientas no autorizadas, eso incluía las librerías de Python como urllib y urlib2.

4.3 Desarrollo de sofware para preservación y mapeo.

Una vez se tuvo la idea general de lo que se requería y se había analizado el problema era momento de listar los requerimientos del software y los diferenter módulos que debían programarce.

Los requerimientos generales eran :

- Obtener los nombres y versiones de los build results que componían al producto principal.
- Guardar los features y plugins ligados a cada dependencia externa.
- Obtener las rutas de archivos en el servidor mapeando las rutas de los servidores web.
- Cambiar la bandera de preservación en los builds dependientes.
- Agregar los productos al mapa generador de gráficas.
- Repetir el proceso en cada producto hasta que las dependencias sean inalcanzables o no se tenga ninguna.

Para dichas tareas se desarrollaron tres diferente módulos, uno relacionado con la exploración de dependencias, otro para la preservación de builds y un último relacionado con la generación de gráficas mostrando dependencias.

4.3.1 Módulo de exploración en dependencias.

Una URL o Uniform Resource Location (Localizador de Recursos Uniforme), por sus siglas en inglés, es una cadena de caracteres estandarizado que permiten encontrar recursos en internet, al principio de una URL suele existir un acronimo al servicio que se decea aceder, seguido por el nombre del host o servidor, aveces el puerto y una ruta especifica. Una URL típica para el procololo https sería *https://www.google.com*.

Las rutas de las que dependería la construcción del producto principal se habían automatizado para utilizar la última construcción exitosa de la dependencia, una vez que identificaba cuales serían las rutas a las dependencias lo primero que creaba era el archivo que asociaba la dependencia con el repositorio de donde lo consumiría.

La estructura de estas rutas consistian en un listado de productos y las rutas de los repositorios en siguiente formato:

src.JAVA = repository.https://constelacion.software.compania.com/Java71-I20160101_0000/cic-metadata/

Se procuraba que el listado de dependencias se mantuviera en menos de 20 diferentes dependencias pero una de pendencia podía contener otras 10 dependencias y estas a su vez otras 5.

Cabe destacar que solo el producto principal utilizaba las versiones más actualizadas de una dependencia, en algunos casos las dependencias anidadas habían adoptado una sola versión de una sub dependencia y no era extraño que el producto y una de sus dependencias dependieran de un mismo producto pero no en la misma versión.

El análisis de URL's se utilizó para :

- Obtener la ruta del repositorio limpia.

- Obtener el nombre de la dependencia.
- Obtener la versión de la dependencia.
- Obtener la ruta dentro de los servidores mapeando los dominios.

Las herramienta principales para este módulo fueron expresiones regulares y diccionarios. Los datos obtenidos para los análisis de cada construcción se guardaban en un archivo en formato JSON para ser consumidos en los siguientes pasos.

4.3.2 Módulo de preservación de repositorios.

El módulo de preservación de repositorios esta dedicado a modificar los archivos de preservación. La construcción de un producto requería descargar todos los componentes de sus dependencias, una vez descargados iniciaba la compilación, pruebas y generaba un reporte de resultados en formato XML, entre ellos Junit, Qualifyer Scan, CHKPII, Copyright, API Usage y APP Scan Source.

Encabezado Build Result Fuente: Reporte de resultados generado por herramienta RTC

El archivo de resultados integraba una etiqueta que le decía al orquestador que no debía borrar la construcción. El orquestador es la herramienta que se utiliza para programar automáticamente las construcciones, aveces dos veces al día, con la finalidad de no saturar los servidores de compilación se le definía una pila máxima de construcciones preservados,

lo que le daba la libertad de borrar las construcciones más antiguos que no estuvieran etiquetados para ser preservados.

Haciendo uso de los resultados obtenidos por el módulo de exploración se accedía mediante una cuenta funcional a los servidores contenedores de los repositorios utilizando instrucciones SSH (Secure SHell), un servicio que permite conectar dispositivos de manera segura utilizando llaves simétricas para el cifrado.

La acciones de preservación consistían en:

- Obtener las rutas de los repositorios.
- Localizar y modificar el etiquetado del build result.
- Crear un reporte.

4.3.3 Módulo de generación de gráficas.

El requerimiento principal de la herramienta es generar una grafica que indique las diferentes dependencias entre productos, este era un requerimiento de la gerencia y de los dueños de producto.

La persona con más experiencia en el equipo me había indicado que la empresa contaba con la licencia de una herramienta que permitía la generación de gráficas en el formato que se requería.

La herramienta podía ser utilizada como servicio y podía recibir en sus parámetros la ruta a un archivo que contuviera el código que interpretaría para generar las gráficas.

En este momento una de las partes más complejas parecía estar resuelta, ya que graficar las dependencias de manera correcta implicaba resolver los grafos generados y colocar los globos de cada producto en el espacio correcto para que existiera la menor cantidad de cruces en las relaciones.

El módulo entonces se encargaría de las siguientes tareas:

- Encontrar las relaciones entre productos
- Generar el código adecuado para ser graficado, destacando las relaciones a productos principales.
- Desplegar la gráfica de las dependencias dentro del build result.

Una vez más utilicé expresiones regulares y herramientas de formato de python para generar el meta lenguaje de la herramienta generadora de graficas y la opción de ver solo las referencias a los productos más importantes.

5 RESULTADOS

No hay ningún viento favorable
para quien no sabe a donde
se dirige.
-Artur Schopenhauer

Los resultados de la actividad reportada son los siguientes:

- Automatización de la preservación de productos.
- Generación de gráficas de dependencias entre productos.

Las preservaciones automatizas son invocadas todas a la vez, esta tarea se realizaba manualmente y solamente en versiones finales, la tarea de preservación le tomaba entre 8 a 12 horas a uno de los ingenieros en Canadá. También se eliminó la incertidumbre del error humano. El reporte ayuda al colaborador en Canadá a dedicar tiempo sólo a aquellos casos donde el programa no pudo acceder o no pudo cambiar los archivos necesarios en la preservación, estos casos no ocurrieron mientras labore en la empresa.

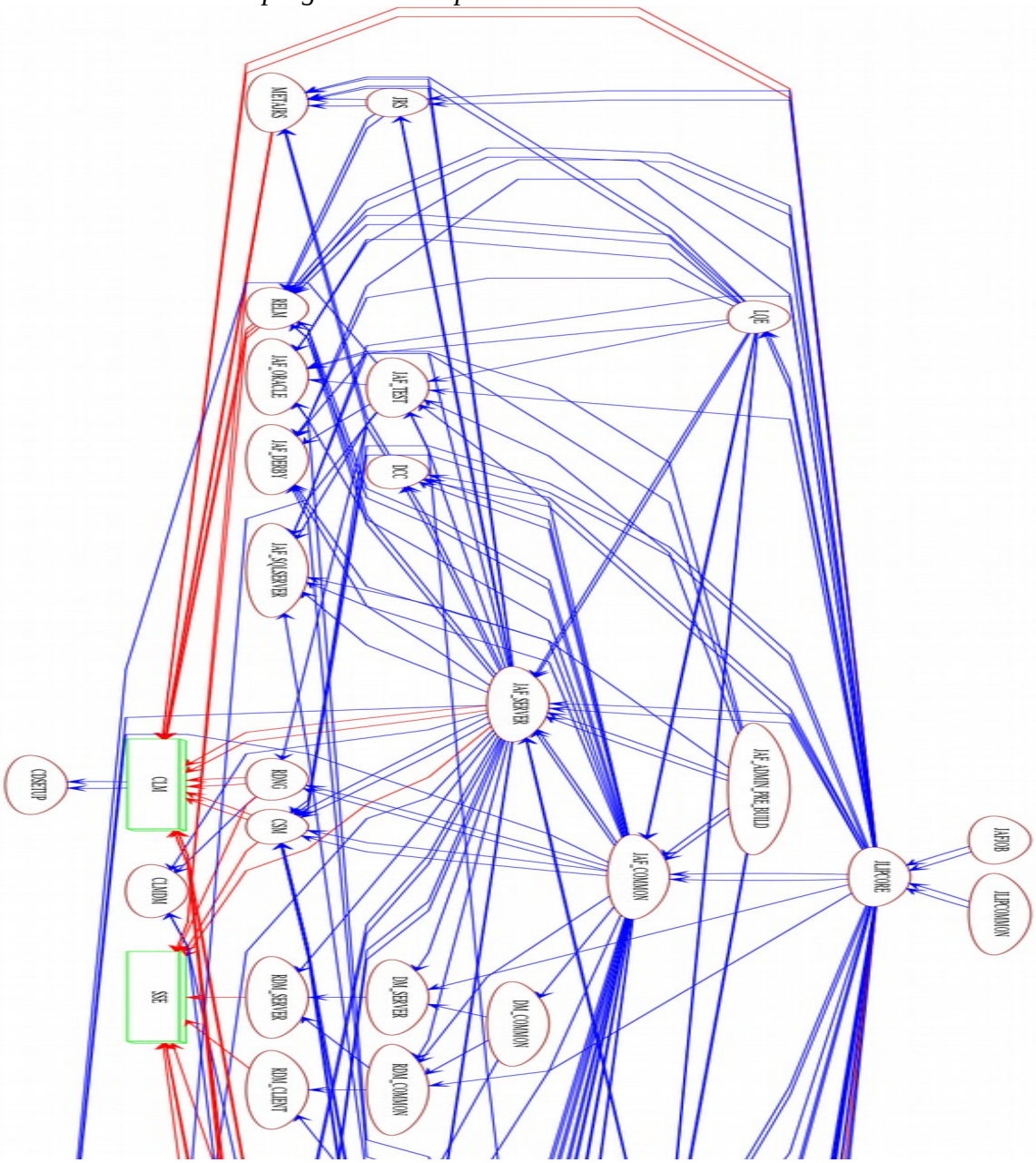
La generación de gráficas de dependencias entre productos se presentaba en cada reporte de creación de los productos principales, se utilizó como referencia para los arquitectos y en la toma de decisiones para la futura venta de los productos a otra compañía.

Las gráficas le dieron una herramienta extra a la gerencia en estados unidos para la toma de decisiones entorno a como debían crear la estrategia para la transición entre compañías, los equipos involucrados, productos a los que no se seguiría dando mantenimiento, los que se eliminarían y los que absorberían las responsabilidades de los que serían eliminados.

Las gráficas de los productos se pueden ver en dos versiones, la general que muestra sólo los productos principales y la completa que muestra el mapa de dependencias completo.

Grafica completa de relaciones entre productos.

Fuente: Resultado del programa de mapeo



Grafica de productos principales.

Fuente: Programa generador de gráficas de dependencias

Gráfica de productos principales
version 2

6 CONCLUSIONES

La verdadera razón no está libre de
todo compromiso con la locura;
por el contrario,
debe seguir los caminos que ésta le señala.
-Michael Foucault

Las herramientas de análisis en el ámbito corporativo se utilizan principalmente para la toma de decisiones, a veces, como en el caso de este reporte la información obtenida se utiliza a discreción de quien la solicita e incluso el propósito puede ser diferente al que se menciona en un principio para la elaboración del análisis. Si bien es cierto, era una pieza de información para los equipos de desarrollo, arquitectos y gerentes fue una gran sorpresa que su principal función fuera, en parte, un mapa para una transición y venta a una compañía distinta.

Algunos meses después de terminar esta herramienta se nos informó de la venta y del estado que presentaban los productos económicamente, parecía que la complejidad en la integración de productos había traspasado el campo técnico y se estaba reflejando directamente desde el punto de vista comercial. Una de las ventajas principales del modelo de integración continua en productos de software es precisamente detener las operaciones cuando económicamente no son viables, sin embargo se habían preservado equipos en productos con bajas ventas y poca integración y al mismo tiempo se había permitido que los clientes principales propusieran funcionalidades personalizadas dentro del producto principal. Los altos costos y la falta de nuevos clientes sumado a herramientas similares gratuitas y en auge llevaron al producto en su compleja estructura a un punto de quiebre en el que fue más sencillo vender la herramienta y al personal a una empresa diferente.

Esta herramienta fue mi contribución a facilitar la comprensión para los actuales y nuevos desarrolladores, arquitectos y gerentes en su nueva etapa y años posteriores.

REFERENCIAS

CAMSS
Public Comunities (2017)
https://www.ibm.com/developerworks/community/groups/community/cams/

Patentes
IBM Sala de prensa (2019)
http://www-03.ibm.com/press/mx/es/pressrelease/51450.wss

Portada-Contra_HR (2019)
https://www.ibm.com/expressadvantage/mx/pdf/Folleto_Conozca_IBM.pdf

El niño y su átomo
A Boy And His Atom: The World's Smallest Movie (2019)
https://www.youtube.com/watch?v=oSCX78-8-q0

HCL Technologies (2019)
https://en.wikipedia.org/wiki/HCL_Technologies

Connectors
JEE Connector Architecture White Paper (2019)
http://www.oracle.com/technetwork/java/javaee/overview/connector-jsp-135375.html

Paradigmas de Programación
http://www.iue.tuwien.ac.at/phd/heinzl/node32.html
Böhm, Jacopini. "Flow diagrams, turing machines and languages with only two formation rules" Comm. ACM, 9(5):366-371, May 1966

Michael A. Covington (2010-08-23). "CSCI/ARTI 4540/6540: First Lecture on Symbolic Programming and LISP" (PDF). University of Georgia. Archived from the original (PDF) on 2012-03-07. Retrieved 2014-12-29.

Wolfram Language Notes for Programming Language Experts
http://www.wolfram.com/language/for-experts/

Programación Orientada a Objetos
Taivalsaari, Antero. "Section 1.1". *Classes vs. Prototypes: Some Philosophical and Historical Observations*. p. 14. CiteSeerX 10.1.1.56.4713

Continuous delivery
Jez Humble. What is Contoinous Delivery (2019)

https://continuousdelivery.com/

“Compañia” Misión y Visión
Jessica Lombardo. IBM’s Mission Statement & Vision Statement (An Analysis) (2018)
http://panmore.com/ibm-vision-statement-mission-statement-analysis-recommendations