

UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE
HUAMANGA

FACULTAD DE INGENIERÍA DE MINAS, GEOLOGÍA Y CIVIL

ESCUELA DE FORMACIÓN PROFESIONAL DE INGENIERÍA
DE SISTEMAS



INFORME DE PRÁCTICAS PRE PROFESIONALES

“INTEGRACIÓN DEL PROYECTO SAIKU ANALYTICS EN LA
HERRAMIENTA GIROSUITE DE LA EMPRESA TESLA
TECHNOLOGIES S.A.C, 2020”

PRESENTADOR POR:
HUAMANÍ FERNÁNDEZ, YOMAR

ASESOR:
MAG. MANUEL AVELINO LAGOS BARZOLA

2 DE DICIEMBRE DE 2020

Dedicatoria

*A mis amorosos y sabios padres, a mis hermanos de sangre y alma, a toda mi familia y
a mis grandes amigos de la vida que me tuvieron paciencia y me dieron su apoyo y
confianza.*

Índice general

Índice de figuras	VI
-------------------	----

Índice de cuadros	VII
-------------------	-----

1. Problemática y Objetivos	1
1.1. Problemática	1
1.2. Solución desarrolladora	3
1.3. Objetivo general	3
1.4. Objetivos específicos	3
2. Marco Teórico	4
2.1. Open Source	4
2.2. Legalidad del software	6
2.2.1. Licencia	6
2.2.2. Derecho de autor o Copyright	6
2.2.3. Patente	6
2.2.4. Licencia de Software	6
2.2.5. Licencia Open Source	7
2.2.5.1. Licencia Apache	7
2.2.5.2. Licencia MIT	7
2.3. Saiku Analytics	8
2.3.1. Pentaho	9
2.3.1.1. Web-based Components	9
2.3.1.2. Design Tools	10
2.4. SCRUM	11
2.4.1. Definición de Scrum	11
2.4.2. Teoría de Scrum	12
2.4.2.1. Transparencia	12
2.4.2.2. Inspección	12
2.4.2.3. Adaptación	13
2.4.3. Valores de Scrum	13
2.4.4. <i>Scrum Team</i>	14
2.4.4.1. <i>Developers</i>	14
2.4.4.2. <i>Product Owner</i>	15
2.4.4.3. <i>Scrum Master</i>	15
2.4.5. Eventos de Scrum	17
2.4.5.1. El <i>Sprint</i>	17
2.4.5.2. <i>Sprint Planning</i>	18

2.4.5.3.	<i>Daily Scrum</i>	19
2.4.5.4.	<i>Sprint Review</i>	20
2.4.5.5.	<i>Sprint Retrospective</i>	20
2.4.6.	Artefactos de Scrum	21
2.4.6.1.	<i>Product Backlog</i>	21
2.4.6.2.	<i>Sprint Backlog</i>	22
2.4.6.3.	<i>Increment</i>	23
2.4.7.	Cambios de la guía Scrum 2017 a la guía Scrum 2020	24
2.5.	Programación Extrema(XP)	25
2.5.1.	Valores, Principios y Prácticas	26
2.5.2.	Valores	27
2.5.2.1.	Comunicación	28
2.5.2.2.	Simplicidad	28
2.5.2.3.	Retroalimentación	28
2.5.2.4.	Coraje	29
2.5.2.5.	Respeto	30
2.5.3.	Principios	30
2.5.3.1.	Humanidad	30
2.5.3.2.	Económicos	31
2.5.3.3.	Beneficio mutuo	31
2.5.3.4.	Auto semejanza	32
2.5.3.5.	Mejora	32
2.5.3.6.	Diversidad	32
2.5.3.7.	Reflexión	33
2.5.3.8.	Flujo	34
2.5.3.9.	Oportunidad	34
2.5.3.10.	Redundancia	35
2.5.3.11.	Fracaso	35
2.5.3.12.	Calidad	36
2.5.3.13.	Pasos pequeños	36
2.5.3.14.	Aceptar responsabilidad	37
2.5.4.	Prácticas	37
2.5.4.1.	Prácticas primarias	37
2.5.4.2.	Prácticas corolarias	46
2.5.5.	Planificación	47
2.5.6.	Diseño	48
2.5.6.1.	Simplicidad	51
2.5.7.	Pruebas	52
2.6.	Manifiesto por el Desarrollo Ágil de Software	56
2.7.	Tecnologías BackEnd para la integración del proyecto Saiku Analytics	56
2.7.1.	Lenguaje de programación Java	56
2.7.2.	Java Specification Requests(JSRs)	57
2.7.2.1.	JSR-366	57
2.7.2.2.	JSR-47	57
2.7.2.3.	JSR-338	58
2.7.2.4.	JSR-346	58
2.7.2.5.	JSR-369	59
2.7.2.6.	JSR-370	59

2.7.2.7.	JSR-371	60
2.7.3.	<i>Spring Framework</i>	60
2.7.3.1.	<i>Spring security</i>	61
2.7.4.	Maven	61
2.7.5.	Mondrian	62
2.7.5.1.	<i>Mondrian y MDX</i>	64
2.7.6.	<i>Online Analytical Processing (OLAP)</i>	64
2.7.7.	Emunciate	65
2.8.	Tecnologías FrontEnd para la integración del proyecto Saiku Analytics	65
2.8.1.	ECMAScript5 (ES5)	65
2.8.2.	HTML5	65
2.8.3.	Javascript	66
2.8.4.	Css	67
2.8.5.	Backbone	67
2.8.6.	Node.js	67
2.8.7.	CCC-Charts	67
3.	Resultados	69
3.1.	Scrum en la práctica (1ra Iteración)	69
3.1.1.	<i>Produc BackLog</i>	69
3.1.2.	<i>Sprint Planning</i>	70
3.1.3.	<i>Sprint BackLog</i>	71
3.1.4.	<i>Daily Scrum</i>	72
3.1.5.	<i>Sprint Review</i>	73
3.1.6.	<i>Sprint Retrospective</i>	80
3.1.7.	<i>Increment</i>	81
4.	Conclusiones y Recomendaciones	82
4.1.	Conclusiones	82
4.2.	Recomendaciones	84
	Bibliografía	85

Índice de figuras

1.1. Total de requisitos y requisitos para atender reportes del 2016 al 2018	2
2.1. Scrum Framework	17
2.2. Valores, Principios y Prácticas	27
2.3. Historias de usuario en una pared	38
2.4. Espacio de trabajo de un equipo	39
2.5. Programación en pares	41
2.6. Tarjeta de historia de usuario	42
2.7. Cualquier diseño antiguo servirá	49
2.8. Se necesita algo de pensamiento o experiencia en diseño	50
2.9. Ninguna cantidad de pensamiento puro será suficiente	50
2.10. La experiencia no ayuda	51
2.11. Mucho que aprender de la experiencia	51
2.12. Las pruebas tardías y costosas dejan muchos defectos	53
2.13. Las pruebas frecuentes reducen los costos y los defectos	54
2.14. El ciclo de Stress	55
2.15. Manifiesto por el desarrollo ágil de software	56
2.16. Mondrian ejecutado en Pentaho	63
3.1. Product BackLog	69
3.2. Product BackLog	70
3.3. Sprint BackLog	72
3.4. Proyecto Saiku Analytics, código fuente discontinuado	73
3.5. Proyecto Rubik-report, código fuente	74
3.6. Proyecto Rubik-report integrado a GiroSuite	75
3.7. GiroSuite y Rubik-report desplegado en Wilfly 10.1	76
3.8. GiroSuite y Rubik-report desplegado en WebLogic 12.2.1.3	77
3.9. GiroSuite y Rubik-report desplegado en WebSphere 9.0	79
3.10. Resumen del Sprint	80
3.11. GiroSuite y Rubik-report desplegado en WebLogic 12.2.1.3	81
3.12. Rubik-report generando reportes dinámicos	81

Índice de cuadros

2.1. Algunos productos que usan Mondrian	62
3.1. Historias del primer Sprint	71

Resumen

Tesla Technologies S.A.C, es una empresa dedicada a aportar e implementar soluciones en materia de software de Riesgos, Seguridad de la información, Continuidad del negocio y Auditoría, con sede en Lima Perú y una sucursal en la ciudad de Santiago de Chile.

En el camino de atender las necesidades de sus clientes y apoyarlos en la creación de informes gerenciales Tesla Technologies S.A.C decide integrar funcionalidades dinámicas de otros proyectos ya consolidados para lograr crear una ventaja comparativa en sus aplicaciones web comerciales. Se inició con la búsqueda de productos de software que permitieran crear reportes de forma dinámica y puedan integrarse a las aplicaciones web ofrecidas por Tesla Technologies, se revisaron algunas herramientas del cuadrante de Gartner y del cuadrante Forrester siendo la mayoría de estas rechazadas por el alto costo de sus licencias e implementación.

Como resultado de la revisión a estas herramientas, fue el proyecto Saiku Analytics complemento de la suite Pentaho BI seleccionado para la integración con las aplicaciones web comerciales de Tesla Technologies S.A.C por su facilidad y compatibilidad.

Se analizaron las tecnologías, librerías de desarrollo y el impacto a generar con la integración del proyecto Saiku Analytics en las aplicaciones web comerciales de Tesla Technologies. Se procedió con la implementación de la integración del proyecto Saiku Analytics y ciertas modificaciones que luego de la integración fue llamado proyecto Rubik Report.

Finalmente luego del lanzamiento del módulo Rubik Report se procedió a crear y diseñar reportes dinámicos y se evaluaron el impacto y los resultados en las aplicaciones web comerciales de Tesla Technologies S.A.C.

Keywords— Licencias, Saiku Analytics, Aplicación web, Pentaho BI

Introducción

Una de las características importantes a considerar en un producto de software, es la flexibilidad o nivel de configuración del producto para responder a las existentes y nuevas necesidades de los usuarios. Aumentar el nivel de configuración de un producto de software tiene beneficios directos en el uso de recursos al atender nuevas necesidades de los usuarios y la oportunidad de convertir una necesidad en una ventaja comparativa. El presente informe aborda el proceso de integración del proyecto Saiku Analytics con los productos que ofrece Tesla Technologies S.A.C y lograr el objetivo de atender las necesidades de los clientes, quienes presentan informes gerenciales con más frecuencia e intentan disminuir su tiempo en la construcción de estos informes teniendo en cuenta aspectos técnicos importantes de compatibilidad y prioridad de los requerimientos de integración con la consigna que el impacto positivo sea alto y el impacto negativo sea bajo o en lo posible pueda ser mitigado. Las consideraciones técnicas incluyen aspectos de tipo y versión de los lenguajes de programación, frameworks de desarrollo frontend y backend, motores relacionales de bases de datos y motores OLAP.

El primer capítulo describe la problemática y define el objetivo general y los objetivos específicos, delimitados por el marco teórico desarrollado en el segundo capítulo con conceptos y fundamentos sobre las licencias, tecnologías usadas y limitaciones del proyecto Saiku Analytics para la integración con los productos ofrecidos por Tesla Technologies S.A.C.

En el tercer capítulo se desarrollan e implementan los requisitos de software para la integración del proyecto Saiku Analytics usando el marco de trabajo Scrum junto a la Programación Extrema(XP) como enfoque de desarrollo de software.

En aras de lograr los objetivos propuestos encontramos lugares comunes e inherentes en la implementación del proyecto revelando los beneficios, limitaciones, propuestas de mejora y nuevas lecciones aprendidas; plasmados en el cuarto capítulo donde indicamos nuestras conclusiones y recomendaciones obtenidas del presente proyecto.

Capítulo 1

Problemática y Objetivos

1.1. Problemática

Tesla Technologies S.A.C, es una empresa dedicada a construir e implementar productos de software en materia de Riesgos, Seguridad de la información, Continuidad del negocio y Auditoría, con sede principal en el Perú y con la consigna de consolidarse como una opción sólida en el mercado latinoamericano de productos digitales para el sector del gobierno corporativo de las empresas.

En el proceso orgánico y conciente de su crecimiento Tesla Technologies S.A.C tiene conocimiento de las limitaciones de los reportes *Ad Hoc* y el poco aporte de estos en la elaboración de informes gerenciales por parte de sus clientes. En consecuencia las limitaciones generaron un significativo incremento de nuevos requerimientos para crear reportes *Ad hoc* por cada cliente.

El incremento de los requerimientos para crear nuevos reportes tienen valores significativos en los periodos 2016, 2017 y 2018 como muestra la Figura 1.1, generando un desequilibrio en la carga operativa para el mantenimiento de reportes existentes y la atención de nuevos requerimientos, impactando en el cumplimiento de los objetivos estratégicos de la empresa.

Existieron intentos de darle dinamismo y flexibilidad a los reportes solicitados, sin embargo, los entregables mínimos no cumplieron con las expectativas de la gerencia y los clientes. La opción de desarrollar una nueva aplicación para atender esta necesidad implicaba desatender y colocar en riesgo el cumplimiento de objetivos estratégicos, razón fundamental para pensar en una solución diferente.

Dando una mirada a proyectos digitales externos con funcionalidades para una reportería dinámica, Tesla Technologies S.A.C toma la decisión de integrar sus productos con he-

herramientas externas evaluando las ventajas y desventajas de esta apuesta. En este nuevo camino es el proyecto Saiku Analytics complemento de la suite Pentaho BI, seleccionado para la integración con los productos de Tesla Technologies S.A.C.

Este proyecto de integración si bien no abarca todo el proceso de evaluación y selección de las herramientas de software del cuadrante de Gartner y del Wave Forrester para dar con la herramienta idónea, enfoca su mirada en consideraciones de licencia y en los puntos tecnológicos importantes de compatibilidad y desarrollo de requisitos críticos para lograr una integración exitosa.

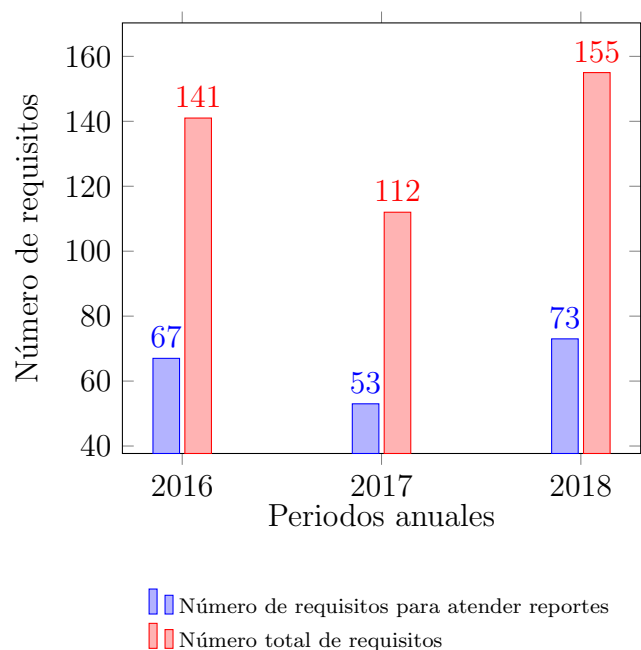


Figura 1.1: Total de requisitos y requisitos para atender reportes del 2016 al 2018

Fuente: Área de soporte de Tesla Technologies S.A.C

1.2. Solución desarrolladora

Desarrollar la integración de las funcionalidades del proyecto Open Source Saiku Analytics a la herramienta GiroSuite de la empresa Tesla Technologies, usando el marco de trabajo Scrum y los principios de Programación Extrema(XP), para apoyar a los usuarios a realizar la creación dinámica de reportes.

1.3. Objetivo general

Realizar la integración del proyecto Open Source Saiku Analytics a la herramienta GiroSuite de la empresa Tesla Technologies S.A.C, para la creación dinámica de reportes usando el marco de trabajo Scrum y los principios de la Programación Extrema(XP).

1.4. Objetivos específicos

- a) Establecer los artefactos Product BackLog y Sprint BackLog que permitan la integración del proyecto Saiku Analytics a la herramienta GiroSuite.
- b) Planificar las iteraciones y la entrega de los Sprints para controlar el avance de la integración del proyecto Saiku Analytics a la herramienta GiroSuite.
- c) Realizar la revisión de los Sprints para determinar mejoras y adaptaciones en las siguientes iteraciones de la integración del proyecto Saiku Analytics a la herramienta GiroSuite.
- d) Aplicar los principios y valores de la Programación Extrema(XP) en el desarrollo de los Sprints de la integración del proyecto Saiku Analytics a la herramienta GiroSuite.
- e) Realizar las pruebas unitarias, de integración y aceptación en el desarrollo y entrega de los Sprint de la integración del proyecto Saiku Analytics a la herramienta GiroSuite.

Capítulo 2

Marco Teórico

2.1. Open Source

Traducito literalmente como "fuente abierta", refiriéndose así a una característica del software donde el código fuente es de libre acceso, sin embargo, Open Source no solo significa eso. Los términos de distribución de un software Open Source deben cumplir con los siguientes criterios:

1. Distribución gratuita

La licencia no impedirá a ninguna parte vender o regalar el software como un componente de una distribución de software agregada que contenga programas de varias fuentes diferentes. La licencia no requerirá regalías u otros honorarios por dicha venta [DiBona et al., 2005].

2. Código fuente

El programa debe incluir el código fuente y debe permitir la distribución en código fuente y en forma compilada. Cuando alguna forma de un producto no se distribuye con el código fuente, debe haber un medio que esté bien publicitado para obtener el código fuente por un costo de reproducción no superior al razonable, preferiblemente, descargarlo a través de Internet sin cargo. El código fuente debe ser la forma preferida en la que un programador modificaría el programa. No se permite el código fuente deliberadamente ofuscado. No se permiten formas intermedias como la salida de un preprocesador o traductor [DiBona et al., 2005].

3. Trabajos derivados

La licencia debe permitir modificaciones y trabajos derivados, y debe permitir

que se distribuyan en los mismos términos que la licencia del software original [DiBona et al., 2005].

4. Integridad del autor del código fuente

La licencia puede restringir la distribución del código fuente en forma modificada solo si la licencia permite la distribución de .archivos de parche con el código fuente con el fin de modificar el programa en el momento de la compilación. La licencia debe permitir explícitamente la distribución de software creado a partir del código fuente modificado. La licencia puede requerir que las obras derivadas lleven un nombre o número de versión diferente al del software original [DiBona et al., 2005].

5. No discriminación contra persona y grupos

La licencia no debe discriminar a ninguna persona o grupo de personas [DiBona et al., 2005].

6. No discriminación en los lugares de actividad

La licencia no debe restringir a nadie el uso del programa en un campo específico de actividad. Por ejemplo, no puede restringir el uso del programa en una empresa o para la investigación genética [DiBona et al., 2005].

7. Licencia de distribución

Los derechos adjuntos al programa deben aplicarse a todos aquellos a quienes se redistribuye el programa sin la necesidad de que esas partes ejecuten una licencia adicional [DiBona et al., 2005].

8. La licencia no debe ser específica para un producto

Los derechos adjuntos al programa no deben depender de que el programa sea parte de una distribución de software en particular. Si el programa se extrae de esa distribución y se usa o distribuye dentro de los términos de la licencia del programa, todas las partes a quienes se redistribuye el programa deben tener los mismos derechos que los que se otorgan junto con la distribución original del software [DiBona et al., 2005].

9. La licencia no debe restringir a otro software

La licencia no debe imponer restricciones a otro software que se distribuya junto con el software con licencia. Por ejemplo, la licencia no debe insistir en que todos

los demás programas distribuidos en el mismo medio deben ser software de código abierto [DiBona et al., 2005].

10. La licencia debe ser de tecnología neutral

Ninguna disposición de la licencia puede basarse en una tecnología individual o estilo de interfaz [DiBona et al., 2005].

2.2. Legalidad del software

2.2.1. Licencia

Una licencia es la autorización que se concede para explotar con fines industriales o comerciales una patente, marca o derecho. Diccionario de la lengua española (23.^a ed.)

2.2.2. Derecho de autor o Copyright

Derecho que la ley reconoce al autor de una obra intelectual o artística para autorizar su reproducción y participar en los beneficios que esta genere. Diccionario de la lengua española (23.^a ed.)

2.2.3. Patente

Documento expedido por una autoridad en que se acredita una condición o un mérito o se da la autorización para hacer algo. Diccionario de la lengua española (23.^a ed.)

2.2.4. Licencia de Software

Una licencia de software es un contrato entre el licenciante (autor/titular de los derechos de explotación/distribución) y el licenciatarlo (usuario consumidor, profesional o empresa) del programa informático, para utilizarlo cumpliendo una serie de términos y condiciones establecidas dentro de sus cláusulas, es decir, es un conjunto de permisos que un licenciante otorga a un usuario en los que tiene la posibilidad de distribuir, usar o modificar el producto bajo una licencia determinada. Además se suelen definir los plazos de duración, el territorio donde se aplica la licencia (ya que la licencia se soporta en las leyes particulares de cada país o región), entre otros.

2.2.5. Licencia Open Source

Las licencias de código abierto son licencias que cumplen con la definición de código abierto; en resumen, permiten que el software se utilice, modifique y comparta libremente. Para ser aprobada por Open Source Initiative (también conocida como OSI), una licencia debe pasar por el proceso de revisión de licencias de Open Source Initiative.[[Chang et al., 2007](#)]

Existen 100 licencias registradas en la Open Source Initiative(OSI), de las cual el presente proyecto describía 2 de ellas por ser las más importante del proyecto Saiku Analytics.

2.2.5.1. Licencia Apache

La licencia de Apache solo requiere un reconocimiento en la “documentación del usuario final” o “en el software mismo”, no en “todos los materiales publicitarios”. La licencia de Apache no especifica la prominencia que debe darse a ese reconocimiento. La licencia de Apache es consistente con los Principios de código abierto porque no interfiere con la libertad de modificar o crear trabajos derivados de software de código abierto.[[Sinclair, 2010](#)] Se deben añadir dos archivos en el directorio principal de los paquetes de software redistribuidos.

1. LICENSE: Una copia de la licencia
2. NOTICE: Un documento de texto que incluye los “avisos” obligatorios del software presente en la distribución y una copia legible de estas notificaciones deben ser distribuidas como parte de los trabajos derivados, dentro de la forma de código fuente o documentación, o dentro de una pantalla generada por las obras derivadas (donde aparecen normalmente este tipo de notificaciones a terceros).

2.2.5.2. Licencia MIT

Licencia creada por los abogados del Instituto de Tecnología de Massachusetts (MIT) a partir de la licencia BSD. Limpiaron algo del lenguaje vago de la licencia BSD y facilitaron la lectura y la comprensión de su versión. En los Apéndices se muestra una copia de la versión actual de la licencia MIT.

La concesión de la licencia del MIT se extiende no solo al software en sí, sino a sus archivos de documentación asociados. No está claro si el MIT se ofrece aquí para proporcionar toda la documentación en su poder sobre el software o solo ciertos archivos que están asociados de alguna manera con el software.

La frase sin cargo significa que el licenciante (MIT en este caso) no cobrará regalías o derechos de licencia. Pero la palabra vender entre la lista de derechos otorgados significa que los licenciarios posteriores no están restringidos de ninguna manera para cobrar regalías o tarifas de licencia a sus clientes por versiones modificadas del software.

La licencia del MIT también sirve como plantilla de licencia. Es una licencia tan corta que solo es necesario cambiar el aviso de derechos de autor para completar la plantilla. Desafortunadamente, la frase este software y los archivos de documentación asociados no identifican claramente a qué software se aplica la licencia. La única forma de correlacionar un software en particular con una copia particular de la licencia del MIT es encontrar físicamente el texto de la licencia en el código fuente del software.[\[Rosen, 2005\]](#)

2.3. Saiku Analytics

Saiku fue fundada en 2008 por Tom Barber y Paul Stoellberger. Originalmente llamada *Pentaho Analysis Tool*, comenzó como un contenedor básico basado en GWT alrededor de la biblioteca OLAP4J. A lo largo de los años ha evolucionado, y después de una reescritura completa en 2010, renació como Saiku.

Saiku ofrece una solución de análisis basada en web fácil de usar que permite a los usuarios analizar rápida y fácilmente los datos corporativos y crear y compartir informes. La solución se conecta a una variedad de servidores OLAP, incluidos Mondrian, Microsoft Analysis Services, SAP BW y Oracle Hyperion y se puede implementar de manera rápida y rentable para permitir a los usuarios explorar datos en tiempo real.

Saiku es una popular herramienta de análisis gráfico de código abierto para Mondrian que se puede ejecutar de forma independiente o como un complemento de Pentaho [\[D. et al., 2013\]](#).

2.3.1. Pentaho

Son una plataforma integral que se utilizan para acceder, integrar, manipular, visualizar y analizar datos. Ya sea que los datos se almacenen en un archivo plano, una base de datos relacional, un clúster Hadoop, una base de datos NoSQL, una base de datos analítica, transmisiones de redes sociales, tiendas operativas o en la nube, los productos Pentaho pueden ayudarlo a descubrir, analizar y visualizar datos para encontrar las respuestas que necesita, incluso si no tiene experiencia en codificación. Los usuarios avanzados con experiencia en programación pueden utilizar la extensa API para personalizar informes, consultas y transformaciones para ampliar la funcionalidad.

Los productos de Pentaho incluyen tanto componentes basados en web como herramientas de diseño. Los componentes y herramientas que utiliza dependen de su flujo de trabajo y de lo que admita su entorno [[“Pentaho Products”, 2020](#)].

2.3.1.1. Web-based Components

Son componentes basados en la web de Pentaho creados para compartir soluciones de inteligencia empresarial mediante el análisis de datos la creación de informes y de paneles integrados.

a) Consola de usuario

La Consola de usuario de Pentaho (PUC) es un entorno de diseño para acceder a Analyzer, Reportes interactivos y Dashboard Designer. La consola de usuario de Pentaho también ofrece funciones de administración del sistema para configurar su servidor Pentaho [[“Pentaho Products”, 2020](#)].

b) Analizador

Analyzer lo ayuda a visualizar datos para tomar decisiones comerciales informadas. Puede crear visualizaciones geográficas, gráficas de dispersión, cuadrículas térmicas y de múltiples gráficas. También puede filtrar datos, agregar parámetros de consulta, configurar enlaces detallados, aplicar formato condicional y generar hipervínculos [[“Pentaho Products”, 2020](#)].

c) Informes interactivos

Interactive Reports es una interfaz de diseño que se utiliza para crear informes

operativos simples y bajo demanda sin depender de TI o desarrolladores de informes. Puede agregar elementos rápidamente a su informe y aplicarles el formato que desee [“Pentaho Products”, 2020].

d) **Diseñador de paneles**

Dashboard Designer se utiliza para elegir plantillas de diseño, temas y contenido para crear paneles visualmente atractivos que ayuden a los tomadores de decisiones a obtener conocimientos críticos de un vistazo. Puede combinar una amplia variedad de contenido, incluidos informes interactivos, visualizaciones del analizador y contenido colaborativo [“Pentaho Products”, 2020].

e) **CTools**

CTools es un marco impulsado por la comunidad para crear paneles mediante el uso de tecnologías web como JavaScript, CSS y HTML. Puede crear fácilmente cuadros de mando dinámicos para que los usuarios exploren y comprendan grandes cantidades de datos mediante una variedad de gráficos, tablas y otros componentes [“Pentaho Products”, 2020].

f) **Asistente de fuente de datos**

El Asistente de fuente de datos lo ayuda a definir una fuente de datos que contiene su información y lo guía a través de la creación de sus primeros modelos relacionales o multidimensionales para usar en la creación de informes y análisis [“Pentaho Products”, 2020].

g) **Editor de modelos de fuente de datos**

El Editor de modelos de origen de datos le ayuda a ajustar y perfeccionar los modelos de datos relacionales y multidimensionales. Puede arrastrar campos a sus ubicaciones apropiadas, mezclar y combinar campos de diferentes tablas, agregar campos a más de una categoría o eliminar un campo [“Pentaho Products”, 2020].

2.3.1.2. Design Tools

Las herramientas de diseño de Pentaho son para desarrollar y perfeccionar cómo se informan, modelan, transforman y almacenan los valores de sus datos.

1. Integración de datos

Pentaho Data Integration (PDI) brinda acceso a un motor de extracción, transformación y carga (ETL) que captura los datos correctos, los limpia y los almacena utilizando un formato uniforme que es accesible y relevante para los usuarios finales y las tecnologías de IoT [[“Pentaho Products”, 2020](#)].

2. Diseñador de informes

Report Designer se utiliza para generar informes detallados con píxeles perfectos utilizando prácticamente cualquier fuente de datos. Permite a los profesionales de inteligencia empresarial crear informes de calidad de impresión altamente detallados y basados en datos preparados adecuadamente [[“Pentaho Products”, 2020](#)].

3. Diseñador de agregaciones

Aggregation Designer proporciona una interfaz simple que le permite crear tablas agregadas a partir de niveles dentro de las dimensiones que especifique. Utilice esta herramienta para mejorar el rendimiento de sus cubos OLAP de Pentaho Analysis (Mondrian) [[“Pentaho Products”, 2020](#)].

4. Editor de metadatos

Metadata Editor simplifica la creación de informes. Utilice el editor de metadatos para crear modelos y dominios de metadatos de Pentaho. Un modelo de metadatos de Pentaho mapea la estructura física de su base de datos en un modelo de negocio lógico [[“Pentaho Products”, 2020](#)].

5. Schema Workbench

Schema Workbench le permite editar y crear modelos multidimensionales (Mondrian). Puede crear modelos de Mondrian gráficamente o definirlos codificando manualmente archivos XML [[“Pentaho Products”, 2020](#)].

2.4. SCRUM

2.4.1. Definición de Scrum

Scrum es un marco de trabajo liviano que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos [[Schwaber and Sutherland, 2020](#)].

El marco de trabajo Scrum es incompleto de manera intencional, solo define las partes necesarias para implementar la teoría de Scrum. Scrum se basa en la inteligencia colectiva de las personas que lo utilizan. En lugar de proporcionar a las personas instrucciones detalladas, las reglas de Scrum guían sus relaciones e interacciones.

En este marco de trabajo pueden emplearse varios procesos, técnicas y métodos. Scrum envuelve las prácticas existentes o las hace innecesarias. Scrum hace visible la eficacia relativa de las técnicas actuales de gestión, entorno y trabajo, de modo que se puedan realizar mejoras.

2.4.2. Teoría de Scrum

Scrum se basa en el empirismo y el pensamiento Lean. El empirismo afirma que el conocimiento proviene de la experiencia y de la toma de decisiones con base en lo observado. El pensamiento Lean reduce el desperdicio y se enfoca en lo esencial.

Scrum emplea un enfoque iterativo e Incremental para optimizar la previsibilidad y controlar el riesgo. Scrum involucra a grupos de personas que colectivamente tienen todas las habilidades y experiencia para hacer el trabajo y compartir o adquirir dichas habilidades según sea necesario. Scrum combina cuatro eventos formales para inspección y adaptación dentro de un evento contenedor, el Sprint. Estos eventos funcionan porque implementan los pilares empíricos de Scrum de transparencia, inspección y adaptación [Schwaber and Sutherland, 2020].

2.4.2.1. Transparencia

El proceso y el trabajo emergentes deben ser visibles tanto para quienes realizan el trabajo como para quienes lo reciben. Con Scrum, las decisiones importantes se basan en el estado percibido de sus tres artefactos formales. Los artefactos que tienen poca transparencia pueden llevar a decisiones que disminuyan el valor y aumenten el riesgo. La transparencia permite la inspección. La inspección sin transparencia es engañosa y derrochadora [Schwaber and Sutherland, 2020].

2.4.2.2. Inspección

Los artefactos de Scrum y el progreso hacia los objetivos acordados deben inspeccionarse con frecuencia y con diligencia para detectar variaciones o problemas potencialmente

indeseables. Para ayudar con la inspección, Scrum proporciona cadencia en forma de sus cinco eventos.

La inspección permite la adaptación. La inspección sin adaptación se considera inútil. Los eventos Scrum están diseñados para provocar cambios [Schwaber and Sutherland, 2020].

2.4.2.3. Adaptación

Si algún aspecto de un proceso se desvía fuera de los límites aceptables o si el producto resultante es inaceptable, el proceso que se aplica o los materiales que se producen deben ajustarse. El ajuste debe realizarse lo antes posible para minimizar una mayor desviación. La adaptación se vuelve más difícil cuando las personas involucradas no están empoderadas ni se autogestionan. Se espera que un Scrum Team se adapte en el momento en que aprenda algo nuevo a través de la inspección [Schwaber and Sutherland, 2020].

2.4.3. Valores de Scrum

El uso exitoso de Scrum depende de que las personas se vuelvan más competentes en vivir cinco valores: **Compromiso, Foco, Franqueza, Respeto y Coraje**. El Scrum Team se compromete a lograr sus objetivos y a apoyarse mutuamente. Su foco principal está en el trabajo del Sprint para lograr el mejor progreso posible hacia estos objetivos. El Scrum Team y sus interesados son francos sobre el trabajo y los desafíos. Los miembros del Scrum Team se respetan entre sí para ser personas capaces e independientes, y son respetados como tales por las personas con las que trabajan. Los miembros del Scrum Team tienen el coraje de hacer lo correcto, para trabajar en problemas difíciles [Schwaber and Sutherland, 2020].

Estos valores dan dirección al Scrum Team con respecto a su trabajo, acciones y comportamiento. Las decisiones que se tomen, los pasos que se den y la forma en que se use Scrum deben reforzar estos valores, no disminuirlos ni socavarlos. Los miembros del Scrum Team aprenden y exploran los valores mientras trabajan con los eventos y artefactos Scrum. Cuando el Scrum Team y las personas con las que trabajan incorporan estos valores, los pilares empíricos de Scrum de transparencia, inspección y adaptación cobran vida y generan confianza.

2.4.4. *Scrum Team*

La unidad fundamental de Scrum es un pequeño equipo de personas, un *Scrum Team*. El *Scrum Team* consta de un *Scrum Master*, un *Product Owner* y *Developers*. Dentro de un *Scrum Team*, no hay subequipos ni jerarquías. Es una unidad cohesionada de profesionales enfocados en un objetivo a la vez, el Objetivo del Producto. Los *Scrum Teams* son multifuncionales, lo que significa que los miembros tienen todas las habilidades necesarias para crear valor en cada *Sprint*. También se autogestionan, lo que significa que deciden internamente quién hace qué, cuándo, cómo y por qué [Schwaber and Sutherland, 2020]. El *Scrum Team* es lo suficientemente pequeño como para seguir siendo ágil y lo suficientemente grande como para completar un trabajo significativo dentro de un *Sprint*, generalmente 10 personas o menos. En general, hemos descubierto que los equipos más pequeños se comunican mejor y son más productivos. Si los *Scrum Teams* se vuelven demasiado grandes, deberían considerar reorganizarse en múltiples *Scrum Teams* cohesivos, cada uno enfocado en el mismo producto. Por lo tanto, deben compartir el mismo Objetivo del Producto, el *Product Backlog* y el *Product Owner*. El *Scrum Team* es responsable de todas las actividades relacionadas con el producto, desde la colaboración de los interesados, la verificación, el mantenimiento, la operación, la experimentación, la investigación y el desarrollo, y cualquier otra cosa que pueda ser necesaria. Están estructurados y empoderados por la organización para gestionar su propio trabajo. Trabajar en *Sprints* a un ritmo sostenible mejora el enfoque y la consistencia del *Scrum Team* [Schwaber and Sutherland, 2020]. Todo el *Scrum Team* es responsable de crear un *Increment* valioso y útil en cada *Sprint*. Scrum define tres responsabilidades específicas dentro del *Scrum Team*: los *Developers*, el *Product Owner* y el *Scrum Master*.

2.4.4.1. *Developers*

Son Las personas del *Scrum Team* que se comprometen a crear cualquier aspecto de un *Increment* utilizable en cada *Sprint* son *Developers*. Las habilidades específicas que necesitan los *Developers* suelen ser amplias y variarán según el ámbito de trabajo. Sin embargo, los *Developers* siempre son responsables de:

- Crear un plan para el *Sprint*, el *Sprint Backlog*;
- Inculcar calidad al adherirse a una Definición de Terminado;

- Adaptar su plan cada día hacia el Objetivo del *Sprint*; y,
- Responsabilizarse mutuamente como profesionales.

2.4.4.2. *Product Owner*

El *Product Owner* es responsable de maximizar el valor del producto resultante del trabajo del *Scrum Team*. La forma en que esto se hace puede variar ampliamente entre organizaciones, *Scrum Teams* e individuos.

El *Product Owner* también es responsable de la gestión efectiva del *Product Backlog*, lo que incluye:

- Desarrollar y comunicar explícitamente el Objetivo del Producto;
- Crear y comunicar claramente los elementos del *Product Backlog*;
- Ordenar los elementos del *Product Backlog*; y,
- Asegurarse de que el *Product Backlog* sea transparente, visible y se entienda.

El *Product Owner* puede realizar el trabajo anterior o puede delegar la responsabilidad en otros. Independientemente de ello, el *Product Owner* sigue siendo el responsable de que el trabajo se realice.

Para que los *Product Owners* tengan éxito, toda la organización debe respetar sus decisiones. Estas decisiones son visibles en el contenido y el orden del *Product Backlog*, y a través del *Increment* inspeccionable en la *Sprint Review*.

El *Product Owner* es una persona, no un comité. El *Product Owner* puede representar las necesidades de muchos interesados en el *Product Backlog*. Aquellos que quieran cambiar el *Product Backlog* pueden hacerlo intentando convencer al *Product Owner*.

2.4.4.3. *Scrum Master*

El Scrum Master es responsable de establecer Scrum como se define en la Guía de Scrum. Lo hace ayudando a todos a comprender la teoría y la práctica de Scrum, tanto dentro del *Scrum Team* como de la organización.

El *Scrum Master* es responsable de lograr la efectividad del *Scrum Team*. Lo hace apoyando al *Scrum Team* en la mejora de sus prácticas, dentro del marco de trabajo de

Scrum.

Los *Scrum Masters* son verdaderos líderes que sirven al *Scrum Team* y a la organización en general [Schwaber and Sutherland, 2020].

El *Scrum Master* sirve al *Scrum Team* de varias maneras, que incluyen:

- Guiar a los miembros del equipo en ser autogestionados y multifuncionales;
- Ayudar al *Scrum Team* a enfocarse en crear *Increments* de alto valor que cumplan con la “Definición de Terminado”;
- Procurar la eliminación de impedimentos para el progreso del *Scrum Team*; y,
- Asegurarse de que todos los eventos de Scrum se lleven a cabo y sean positivos, productivos y se mantengan dentro de los límites de tiempo recomendados en la guía de Scrum.

El *Scrum Master* sirve al *Product Owner* de varias maneras, que incluyen:

- Ayudar a encontrar técnicas para una definición efectiva de Objetivos del Producto y la gestión del *Product Backlog*;
- Ayudar al *Scrum Team* a comprender la necesidad de tener elementos del *Product Backlog* claros y concisos;
- Ayudar a establecer una planificación empírica de productos para un entorno complejo; y,
- Facilitar la colaboración de los interesados según se solicite o necesite.

El Scrum Master sirve a la organización de varias maneras, que incluyen:

- Liderar, capacitar y guiar a la organización en su adopción de Scrum;
- Planificar y asesorar implementaciones de Scrum dentro de la organización;
- Ayudar a los empleados y los interesados a comprender y aplicar un enfoque empírico para el trabajo complejo; y,
- Eliminar las barreras entre los interesados y los *Scrum Teams*.

2.4.5. Eventos de Scrum

El *Sprint* es un contenedor para todos los demás eventos. Cada evento en Scrum es una oportunidad formal para inspeccionar y adaptar los artefactos Scrum. Estos eventos están diseñados específicamente para habilitar la transparencia requerida. No operar cualquier evento según lo prescrito resulta en la pérdida de oportunidades para inspeccionar y adaptarse. Los eventos se utilizan en Scrum para crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum.

Lo óptimo es que todos los eventos se celebren al mismo tiempo y en el mismo lugar para reducir la complejidad [Schwaber and Sutherland, 2020], los eventos se pueden apreciar detalladamente en la Figura 2.1.

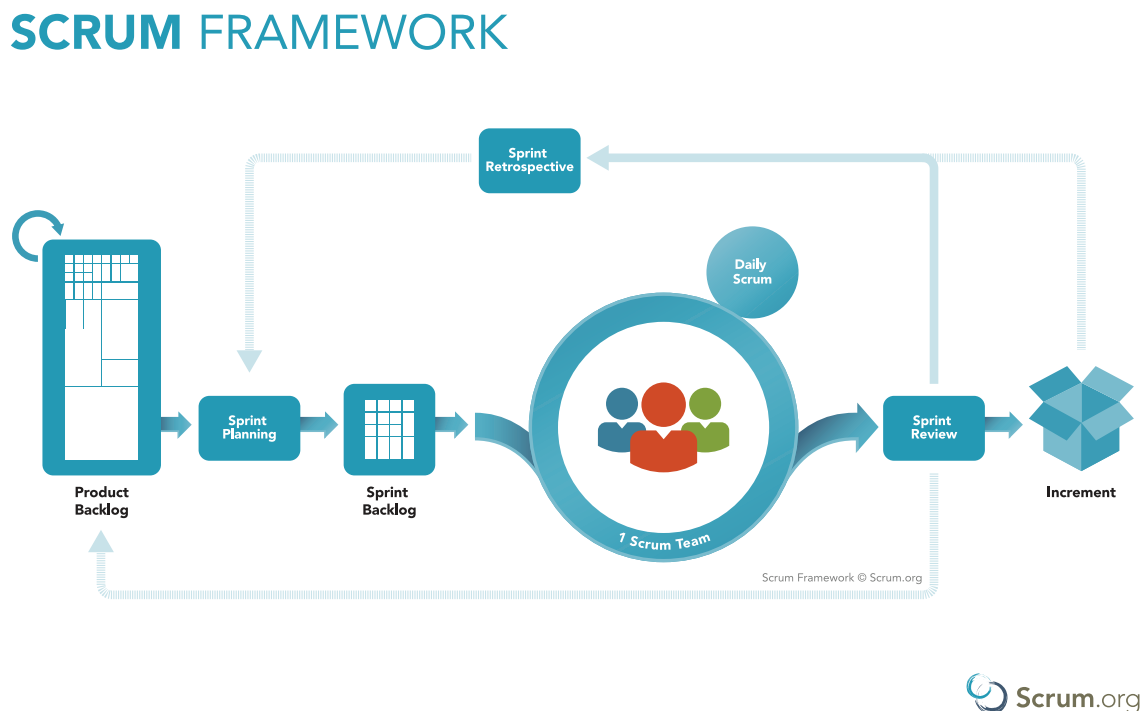


Figura 2.1: Scrum Framework

Nota: Extraído de “Scrum Framework Poster”, 2020, www.scrum.org

2.4.5.1. El *Sprint*

Los *Sprints* son el corazón de Scrum, donde las ideas se convierten en valor. Son eventos de duración fija de un mes o menos para crear consistencia. Un nuevo Sprint comienza inmediatamente después de la conclusión del Sprint anterior.

Todo el trabajo necesario para lograr el Objetivo del Producto, incluido la *Sprint Planning*, *Daily Scrums*, *Sprint Review* y *Sprint Retrospective*, ocurre dentro de los *Sprints* [Schwaber and Sutherland, 2020].

Durante el Sprint:

- No se realizan cambios que pongan en peligro el Objetivo del *Sprint*;
- La calidad no disminuye;
- El *Product Backlog* se refina según sea necesario; y,
- El alcance se puede aclarar y renegociar con el *Product Owner* a medida que se aprende más.

Los *Sprints* permiten la previsibilidad al garantizar la inspección y adaptación del progreso hacia un Objetivo del Producto al menos cada mes calendario. Cuando el horizonte de un *Sprint* es demasiado largo, el Objetivo del *Sprint* puede volverse inválido, la complejidad puede crecer y el riesgo puede aumentar. Se pueden emplear *Sprints* más cortos para generar más ciclos de aprendizaje y limitar el riesgo de costo y esfuerzo a un período de tiempo menor. Cada *Sprint* puede considerarse un proyecto corto.

Existen varias prácticas para pronosticar el progreso, como el trabajo pendiente (*burn-downs*), trabajo completado (*burn-ups*) o flujos acumulativos (*cumulative flows*). Si bien han demostrado su utilidad, no reemplazan la importancia del empirismo. En entornos complejos, se desconoce lo que sucederá. Solo lo que ya ha sucedido se puede utilizar para la toma de decisiones con miras al futuro.

Un *Sprint* podría cancelarse si el Objetivo del *Sprint* se vuelve obsoleto. Solo el *Product Owner* tiene la autoridad para cancelar el *Sprint*.

2.4.5.2. *Sprint Planning*

La *Sprint Planning* inicia el *Sprint* al establecer el trabajo que se realizará para el *Sprint*. El *Scrum Team* crea este plan resultante mediante trabajo colaborativo.

El *Product Owner* se asegura de que los asistentes estén preparados para discutir los elementos más importantes del *Product Backlog* y cómo se relacionan con el Objetivo del Producto. El *Scrum Team* también puede invitar a otras personas a asistir a la *Sprint Planning* para brindar asesoramiento [Schwaber and Sutherland, 2020].

El *Sprint Planning* aborda los siguientes temas:

- Tema uno: ¿Por qué es valioso este *Sprint*?

El *Product Owner* propone cómo el producto podría *Incrementar* su valor y utilidad en el *Sprint* actual. Luego, todo el *Scrum Team* colabora para definir un Objetivo del *Sprint* que comunica por qué el *Sprint* es valioso para los interesados. El Objetivo del *Sprint* debe completarse antes de que termine la *Sprint Planning*.

- Tema dos: ¿Qué se puede hacer en este *Sprint*?

A través de una conversación con el *Product Owner*, los *Developers* seleccionan elementos del *Product Backlog* para incluirlos en el *Sprint* actual. El *Scrum Team* puede refinar estos elementos durante este proceso, lo que aumenta la comprensión y la confianza.

Seleccionar cuánto se puede completar dentro de un *Sprint* puede ser un desafío. Sin embargo, cuanto más sepan los *Developers* sobre su desempeño pasado, su capacidad actual y su “Definición de Terminado”, más confiados estarán en sus pronósticos para el *Sprint*.

- Tema tres: ¿Cómo se realizará el trabajo elegido?

Para cada elemento del *Product Backlog* seleccionado, los *Developers* planifican el trabajo necesario para crear un *Increment* que cumpla con la “Definición de Terminado”. A menudo, esto se hace descomponiendo los elementos del *Product Backlog* en elementos de trabajo más pequeños de un día o menos. La forma de hacerlo queda a criterio exclusivo de los *Developers*. Nadie más les dice cómo convertir los elementos del *Product Backlog* en *Increments* de valor.

El Objetivo del *Sprint*, los elementos del *Product Backlog* seleccionados para el *Sprint*, más el plan para entregarlos se denominan juntos *Sprint Backlog*. El *Sprint Planning* tiene un límite de tiempo de máximo ocho horas para un *Sprint* de un mes. Para *Sprints* más cortos, el evento suele ser de menor duración.

2.4.5.3. *Daily Scrum*

El propósito del *Daily Scrum* es inspeccionar el progreso hacia el Objetivo del *Sprint* y adaptar el *Sprint Backlog* según sea necesario, ajustando el trabajo planificado en traste. El *Daily Scrum* es un evento de 15 minutos para los *Developers* del *Scrum Team*. Para reducir la complejidad, se lleva a cabo a la misma hora y en el mismo

lugar todos los días hábiles del *Sprint*. Si el *Product Owner* o *Scrum Master* están trabajando activamente en elementos del *Sprint Backlog*, participan como *Developers* [Schwaber and Sutherland, 2020]. Los *Developers* pueden seleccionar la estructura y las técnicas que deseen, siempre que su *Daily Scrum* se centre en el progreso hacia el Objetivo del *Sprint* y produzca un plan viable para el siguiente día de trabajo. Esto crea enfoque y mejora la autogestión.

Los *Daily Scrums* mejoran la comunicación, identifican impedimentos, promueven la toma rápida de decisiones y, en consecuencia, eliminan la necesidad de otras reuniones.

El *Daily Scrum* no es el único momento en el que los *Developers* pueden ajustar su plan. A menudo se reúnen durante el día para discusiones más detalladas sobre cómo adaptar o volver a planificar el resto del trabajo del *Sprint*.

2.4.5.4. *Sprint Review*

El propósito del *Sprint Review* es inspeccionar el resultado del *Sprint* y determinar futuras adaptaciones. El *Scrum Team* presenta los resultados de su trabajo a los interesados clave y se discute el progreso hacia el Objetivo del Producto.

Durante el evento, el *Scrum Team* y los interesados revisan lo que se logró en el *Sprint* y lo que ha cambiado en su entorno. Con base en esta información, los asistentes colaboran sobre qué hacer a continuación. El *Product Backlog* también se puede ajustar para satisfacer nuevas oportunidades. El *Sprint Review* es una sesión de trabajo y el *Scrum Team* debe evitar limitarla a una presentación.

La *Sprint Review* es el penúltimo evento del *Sprint* y tiene un límite de tiempo de máximo cuatro horas para un *Sprint* de un mes. Para *Sprints* más cortos, el evento suele ser de menor duración.

2.4.5.5. *Sprint Retrospective*

El propósito de la *Sprint Retrospective* es planificar formas de aumentar la calidad y la efectividad.

El *Scrum Team* inspecciona cómo fue el último *Sprint* con respecto a las personas, las interacciones, los procesos, las herramientas y su “Definición de Terminado”. Los elementos inspeccionados suelen variar según el ámbito del trabajo. Se identifican los supuestos que los llevaron por mal camino y se exploran sus orígenes. El *Scrum Team* analiza qué

salió bien durante el *Sprint*, qué problemas encontró y cómo se resolvieron (o no) esos problemas.

El *Scrum Team* identifica los cambios más útiles para mejorar su efectividad. Las mejoras más impactantes se abordan lo antes posible. Incluso se pueden agregar al *Sprint Backlog* para el próximo *Sprint* [Schwaber and Sutherland, 2020]. La *Sprint Retrospective* concluye el *Sprint*. Tiene un tiempo limitado a máximo tres horas para un *Sprint* de un mes. Para *Sprints* más cortos, el evento suele ser de menor duración.

2.4.6. Artefactos de Scrum

Los artefactos de Scrum representan trabajo o valor. Están diseñados para maximizar la transparencia de la información clave. Por lo tanto, todas las personas que los inspeccionan tienen la misma base de adaptación [Schwaber and Sutherland, 2020]. Cada artefacto contiene un compromiso para garantizar que proporcione información que mejore la transparencia y el enfoque frente al cual se pueda medir el progreso:

- Para el *Product Backlog*, es el Objetivo del Producto.
- Para el *Sprint Backlog*, es el Objetivo del *Sprint*.
- Para el *Increment* es la “Definición de Terminado”.

Estos compromisos existen para reforzar el empirismo y los valores de Scrum para el *Scrum Team* y sus interesados.

2.4.6.1. *Product Backlog*

El *Product Backlog* es una lista emergente y ordenada de lo que se necesita para mejorar el producto. Es la única fuente del trabajo realizado por el *Scrum Team*. Los elementos del *Product Backlog* que el *Scrum Team* puede dar por terminados dentro de un *Sprint* se consideran preparados para ser seleccionados en un evento de *Sprint Planning*. Suelen adquirir este grado de transparencia tras las actividades de refinamiento. El refinamiento del *Product Backlog* es el acto de dividir y definir aún más los elementos del *Product Backlog* en elementos más pequeños y precisos. Esta es una actividad continua para agregar detalles, como una descripción, orden y tamaño.

Los atributos suelen variar según el ámbito del trabajo.

Los *Developers* que realizarán el trabajo son responsables del dimensionamiento. El *Product Owner* puede influir en los *Developers* ayudándolos a entender y seleccionar sus mejores alternativas.

Compromiso: Objetivo del Producto

El Objetivo del Producto describe un estado futuro del producto que puede servir como un objetivo para que el *Scrum Team* planifique. El Objetivo del Producto está en el *Product Backlog*. El resto del *Product Backlog* emerge para definir "qué cumplirá con el Objetivo del Producto.

Un producto es un vehículo para entregar valor. Tiene un límite claro, personas interesadas conocidas, usuarios o clientes bien definidos. Un producto puede ser un servicio, un producto físico o algo más abstracto [Schwaber and Sutherland, 2020].

El Objetivo del Producto es el objetivo a largo plazo del *Scrum Team*. Ellos deben cumplir (o abandonar) un objetivo antes de asumir el siguiente.

2.4.6.2. *Sprint Backlog*

El *Sprint Backlog* se compone del Objetivo del *Sprint* (por qué), el conjunto de elementos del *Product Backlog* seleccionados para el *Sprint* (qué), así como un plan de acción para entregar el Incremento(cómo). El *Sprint Backlog* es un plan realizado por y para los *Developers*. Es una imagen muy visible y en tiempo real del trabajo que los *Developers* planean realizar durante el *Sprint* para lograr el Objetivo del *Sprint* [Schwaber and Sutherland, 2020].

En consecuencia, el *Sprint Backlog* se actualiza a lo largo del *Sprint* a medida que se aprende más. Debe tener suficientes detalles para que puedan inspeccionar su progreso en la *Daily Scrum*.

Compromiso: Objetivo del Sprint

El Objetivo del *Sprint* es el único propósito del *Sprint*. Si bien el Objetivo del *Sprint* es un compromiso de los *Developers*, proporciona flexibilidad en términos del trabajo exacto necesario para lograrlo. El Objetivo del *Sprint* también crea coherencia y enfoque, lo

que alienta al *Scrum Team* a trabajar en conjunto en lugar de en iniciativas separadas [Schwaber and Sutherland, 2020].

El Objetivo del *Sprint* se crea durante el evento *Sprint Planning* y se agrega al *Sprint Backlog*. Mientras los *Developers* trabajan durante el *Sprint*, tienen en mente el Objetivo del *Sprint*. Si el trabajo resulta ser diferente de lo que esperaban, colaboran con el *Product Owner* para negociar el alcance del *Sprint Backlog* dentro del *Sprint* sin afectar el Objetivo del *Sprint*.

2.4.6.3. *Increment*

Un *Increment* es un peldaño concreto hacia el Objetivo del Producto. Cada *Increment* se suma a todos los *Increments* anteriores y se verifica minuciosamente, lo que garantiza que todos los *Increments* funcionen juntos. Para proporcionar valor, el *Increment* debe ser utilizable. Se pueden crear múltiples *Increments* dentro de un *Sprint*. La suma de los *Increments* se presenta en la *Sprint Review* apoyando así el empirismo. Sin embargo, se puede entregar un *Increment* a los interesados antes del final del *Sprint*. La *Sprint Review* nunca debe considerarse una puerta para liberar valor [Schwaber and Sutherland, 2020]. El trabajo no puede considerarse parte de un *Increment* a menos que cumpla con la Definición de Terminado.

Compromiso: Definición de Terminado

La Definición de Terminado es una descripción formal del estado del *Increment* cuando cumple con las medidas de calidad requeridas para el producto.

En el momento en que un elemento del *Product Backlog* cumple con la Definición de Terminado, nace un *Increment* [Schwaber and Sutherland, 2020].

La Definición de Terminado crea transparencia al brindar a todos un entendimiento compartido de qué trabajo se completó como parte del *Increment*. Si un elemento del *Product Backlog* no cumple con la Definición de Terminado, no se puede publicar ni presentar en la *Sprint Review*. En su lugar, vuelve al *Product Backlog* para su consideración futura. Si la Definición de Terminado para un *Increment* es parte de los estándares de la organización, todos los *Scrum Teams* deben seguirla como mínimo. Si no es un estándar organizacional, el *Scrum Team* debe crear una Definición de Terminado apropiada para el producto.

Los *Developers* deben adherirse a la Definición de Terminado. Si hay varios Scrum Teams trabajando juntos en un producto, deben definir y cumplir mutuamente la misma Definición de Terminado.

2.4.7. Cambios de la guía Scrum 2017 a la guía Scrum 2020

- **Aún menos prescriptiva**

A lo largo de los años, la Guía Scrum comenzó a ser un poco más prescriptiva. La versión 2020 tenía como objetivo que Scrum volviera a ser un marco de trabajo mínimamente suficiente al eliminar o suavizar el lenguaje prescriptivo. Por ejemplo, eliminó las preguntas de la *Daily Scrum*, suavizó el lenguaje sobre los atributos de los *PBI*, suavizó el lenguaje sobre los elementos retro en el *Sprint Backlog*, acortó la sección de cancelación de *Sprint* y más [Schwaber and Sutherland, 2020].

- **Un equipo, enfocado en un producto**

El objetivo era eliminar el concepto de un equipo separado dentro de un equipo que ha llevado a un comportamiento de “proxy” o de “nosotros y ellos” entre el PO y el Equipo de Desarrollo. Ahora solo hay un *Scrum Team* enfocado en el mismo objetivo, con tres diferentes conjuntos de responsabilidades: PO, SM y *Developers* [Schwaber and Sutherland, 2020].

- **Introducción del Objetivo del Producto**

La Guía de Scrum del 2020 introduce el concepto de Objetivo del Producto para proporcionar enfoque al *Scrum Team* hacia un objetivo valioso más grande. Cada *Sprint* debería acercar el producto al Objetivo del Producto general [Schwaber and Sutherland, 2020].

- **Un hogar para el Objetivo del *Sprint*, la Definición de Terminado y el Objetivo del Producto**

Las Guías Scrum anteriores describían el Objetivo del *Sprint* y la Definición de Terminado sin realmente darles una identidad. No eran del todo artefactos, pero estaban algo unidos a los artefactos. Con la incorporación del Objetivo del Producto, la versión 2020 proporciona más claridad al respecto. Cada uno de los tres artefactos ahora contiene compromisos con ellos. Para el *Product Backlog* es el Objetivo del Producto, el *Sprint Backlog* tiene el Objetivo del *Sprint* y el *Increment* tiene la

Definición de Terminado(ahora sin las comillas). Existen para aportar transparencia y enfocarse en el progreso de cada artefacto[Schwaber and Sutherland, 2020].

- **Autogestión sobre autoorganización**

Las Guías Scrum anteriores se referían a los Equipos de Desarrollo como autoorganizados, eligiendo quién y cómo hacer el trabajo. Con un enfoque más en el *Scrum Team*, la versión 2020 enfatiza un *Scrum Team* autogestionado, eligiendo quién, cómo y en qué trabajar [Schwaber and Sutherland, 2020].

- **Tres temas del *Sprint Planning***

Además de los temas de la *Sprint Planning* de "Quéz Cómo", la Guía de Scrum 2020 pone énfasis en un tercer tema, "Por qué", en referencia al Objetivo del *Sprint* [Schwaber and Sutherland, 2020].

- **Simplificación general del lenguaje para una audiencia más amplia**

La Guía Scrum 2020 ha hecho hincapié en eliminar declaraciones redundantes y complejas, así como en eliminar cualquier inferencia restante al trabajo de TI (por ejemplo, pruebas, sistema, diseño, requisito, etc.). La Guía Scrum ahora tiene menos de 13 páginas [Schwaber and Sutherland, 2020].

2.5. Programación Extrema(XP)

Extreme Programming (XP) trata sobre un cambio social. Se trata de dejar de lado hábitos y patrones que eran adaptativos en el pasado, pero que ahora se interponen en nuestro camino para hacer nuestro mejor trabajo. Se trata de renunciar a las defensas que nos protegen pero que interfieren con nuestra productividad. Puede dejarnos sintiéndonos expuestos [Beck and Andres, 2004].

Se trata de ser abiertos sobre lo que somos capaces de hacer y luego hacerlo. Y permitir y esperar que otros hagan lo mismo. Se trata de superar nuestra seguridad adolescente de que

sé más que los demás y lo único que necesito es que me dejen solo para ser el mejor". Se trata de encontrar nuestro lugar adulto en el mundo más amplio, encontrar nuestro lugar en la comunidad, incluido el ámbito de los negocios y el trabajo. Se trata del proceso de convertirnos en lo mejor de nosotros mismos y en el proceso de convertirnos en lo

mejor de nosotros como desarrolladores. Y se trata de escribir un código excelente que sea realmente bueno para los negocios.

XP es un estilo de desarrollo de software que se centra en la excelente aplicación de técnicas de programación, comunicación clara y trabajo en equipo que nos permite lograr cosas que antes ni siquiera podíamos imaginar [Beck and Andres, 2004]. XP incluye:

- Una filosofía de desarrollo de software basada en los valores de comunicación, retroalimentación, sencillez, valentía y respeto.
- Un conjunto de prácticas de probada utilidad para mejorar el desarrollo de software. Las prácticas se complementan, amplificando sus efectos. Se eligen como expresiones de los valores.
- Un conjunto de principios complementarios, técnicas intelectuales para traducir los valores en práctica, útiles cuando no hay una práctica a la mano para su problema particular.
- Una comunidad que comparte estos valores y muchas de las mismas prácticas.

2.5.1. Valores, Principios y Prácticas

Iniciemos por las prácticas; las prácticas son las cosas que haces a diario. Especificar prácticas es útil porque son claras y objetivas. O escribes una prueba antes de cambiar el código o no lo haces. Las prácticas también son útiles porque brindan un lugar para comenzar. Puede comenzar a escribir pruebas antes de cambiar el código y beneficiarse al hacerlo, mucho antes de comprender el desarrollo de software de una manera más profunda [Beck and Andres, 2004].

Los valores son la raíz de las cosas que nos gustan y las que no nos gustan en una situación. Cuando un programador dice: "No quiero estimar mis tareas", generalmente no se refiere a la técnica. Él ya estima, pero no quiere revelar lo que realmente piensa por temor a proporcionar un punto fijo de juicio que se usará en su contra más adelante. ¡Mejor triplica esa estimación! Negarse a comunicar estimaciones revela algo mucho más profundo sobre cómo ve las fuerzas sociales en el desarrollo. Quizás no quiera rendir cuentas porque se le ha culpado injustamente en el pasado. En este caso, el programador valora la protección sobre la comunicación. Los valores son los criterios a gran escala que usamos para juzgar

lo que vemos, pensamos y hacemos. Las prácticas son evidencia de valores. Los valores se expresan a un nivel tan alto que podría hacer casi cualquier cosa en nombre de un valor [Beck and Andres, 2004].

Los valores y las prácticas son un océano aparte. Los valores son universales. Idealmente, mis valores mientras trabajo son exactamente los mismos que mis valores en el resto de mi vida. Las prácticas, sin embargo, están situadas intensamente. Si quiero comentarios sobre si estoy haciendo un buen trabajo programando, tiene sentido construir y probar continuamente mi software [Beck and Andres, 2004]. Cerrar la brecha entre los valores y las prácticas son principios (ver Figura 2.2). Los principios son pautas de por vida específicas del dominio. A continuación veremos los valores, principios y prácticas de XP.

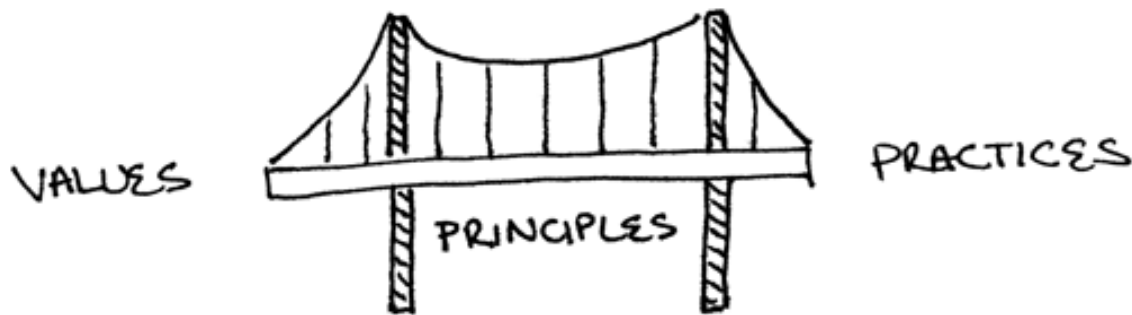


Figura 2.2: Valores, Principios y Prácticas

Extraído de “Extreme Programming Explained: Embrace Change, 2nd Edition” de Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

2.5.2. Valores

Como dijo Will Rogers, “no es lo que no sabes lo que te mete en problemas. Es lo que sabes que no es así”. El mayor problema que encuentro en lo que la gente “simplemente sab” sobre el desarrollo de software es que se centran en la acción individual. Lo que realmente importa no es cómo se comporta una persona determinada sino cómo se comportan los individuos como parte de un equipo y como parte de una organización [Beck and Andres, 2004].

Si todos los miembros del equipo deciden centrarse en lo que es importante para el equipo, ¿en qué deberían centrarse? XP adopta cinco valores para guiar el desarrollo: comunicación, simplicidad, retroalimentación, coraje y respeto.

2.5.2.1. Comunicación

Lo que más importa en el desarrollo de software en equipo es la comunicación. Cuando surgen problemas en el desarrollo, la mayoría de las veces alguien ya conoce la solución; pero ese conocimiento no llega a alguien con el poder de hacer el cambio. Esto ocurre internamente cuando ignoro mi intuición, pero los efectos se agravan cuando nos comunicamos entre personas. La comunicación es importante para crear un sentido de equipo y una cooperación eficaz [Beck and Andres, 2004].

Cuando encuentre un problema, pregúntese si el problema fue causado por una falta de comunicación. ¿Qué comunicación necesita ahora para abordar el problema? ¿Qué comunicación necesita para mantenerse fuera de este problema en el futuro?.

2.5.2.2. Simplicidad

La simplicidad es el más intensamente intelectual de los valores de XP. Hacer un sistema lo suficientemente simple como para resolver con elegancia solo los problemas de hoy es un trabajo duro. La solución simple de ayer puede estar bien hoy, o puede parecer simplista o compleja. Cuando necesite cambiar para recuperar la simplicidad, debe encontrar un camino desde donde está hasta donde quiere estar.

Los valores están destinados a equilibrarse y apoyarse mutuamente. Mejorar la comunicación ayuda a lograr la simplicidad al eliminar los requisitos innecesarios o diferibles de las preocupaciones actuales. Lograr la simplicidad le brinda mucho menos de qué comunicarse [Beck and Andres, 2004].

2.5.2.3. Retroalimentación

Ninguna dirección fija permanece válida por mucho tiempo; ya sea que estemos hablando de los detalles del desarrollo de software, los requisitos del sistema o la arquitectura del sistema. Las direcciones establecidas antes de la experiencia tienen una vida media especialmente corta. El cambio es inevitable, pero el cambio crea la necesidad de retroalimentación [Beck and Andres, 2004]. Al estar satisfechos con la mejora en lugar de esperar la perfección instantánea, utilizamos la retroalimentación para acercarnos cada vez más a nuestras metas. La retroalimentación viene de muchas formas:

- Opiniones sobre una idea, tuya o de tus compañeros.

- Cómo se ve el código cuando implementas la idea.
- Si las pruebas fueron fáciles de escribir.
- Si las pruebas se ejecutan.
- Cómo funciona la idea una vez que se ha implementado.

Los equipos de XP se esfuerzan por generar la mayor cantidad de comentarios que puedan manejar lo más rápido posible. Intentan acortar el ciclo de retroalimentación a minutos u horas en lugar de semanas o meses. Cuanto antes lo sepa, antes podrá adaptarse. Es posible que reciba demasiados comentarios. Si el equipo está ignorando comentarios importantes; necesita reducir la velocidad, por frustrante que sea, hasta que pueda responder a los comentarios.

2.5.2.4. Coraje

El coraje es una acción eficaz frente al miedo. Algunas personas se han opuesto al uso de la palabra “coraje”, reservándola para lo que hace un soldado que patrulla cuando atraviesa una puerta oscura. Sin pretender disminuir el tipo de coraje físico demostrado por el soldado, es cierto que las personas involucradas en el desarrollo de software sienten miedo. Es la forma en que manejan su miedo lo que dicta si están trabajando como una parte efectiva de un equipo.

A veces, el coraje se manifiesta como un sesgo hacia la acción. Si sabe cuál es el problema, haga algo al respecto. A veces, el coraje se manifiesta como paciencia. Si sabe que hay un problema, pero no sabe cuál es, se necesita valor para esperar a que el problema real surja de manera clara.

El coraje como valor primario sin valores compensatorios es peligroso. Hacer algo sin tener en cuenta las consecuencias no es un trabajo en equipo eficaz. Fomente el trabajo en equipo mirando los otros valores como guía sobre qué hacer cuando tiene miedo [Beck and Andres, 2004].

Si el coraje por sí solo es peligroso, junto con los otros valores es poderoso. El coraje para decir verdades, agradables o desagradables, fomenta la comunicación y la confianza. El valor de descartar las soluciones fallidas y buscar otras nuevas fomenta la sencillez. El coraje de buscar respuestas reales y concretas genera retroalimentación.

2.5.2.5. Respeto

Los cuatro valores anteriores apuntan a uno que se encuentra por debajo de la superficie de los otros cuatro: respeto. Si los miembros de un equipo no se preocupan por los demás y por lo que están haciendo, XP no funcionará. Si a los miembros de un equipo no les importa un proyecto, nada puede salvarlo.

Toda persona cuya vida se ve afectada por el desarrollo de software tiene el mismo valor que un ser humano. Nadie es intrínsecamente más valioso que nadie. Para que el desarrollo de software mejore simultáneamente en humanidad y productividad, las contribuciones de cada persona en el equipo deben ser respetadas. Yo soy importante y tú también [Beck and Andres, 2004].

2.5.3. Principios

Los valores son demasiado abstractos para guiar directamente el comportamiento. Los documentos largos están destinados a comunicarse, al igual que las conversaciones diarias. ¿Cuál es el más efectivo? La respuesta depende en parte del contexto y en parte de los principios intelectuales. En este caso, el principio de humanidad sugiere que la conversación satisface la necesidad humana básica de conexión y, por tanto, es la forma preferida de comunicación, en igualdad de condiciones. La comunicación escrita es inherentemente más derrochadora. Si bien la comunicación escrita le permite llegar a una gran audiencia, es una comunicación unidireccional. La conversación permite aclaraciones, comentarios inmediatos, lluvia de ideas y otras cosas que no puede hacer con un documento. La comunicación escrita tiende a ser tomada como un hecho o rechazada rotundamente, ninguna de las cuales es una invitación a una mayor comunicación.

2.5.3.1. Humanidad

La gente desarrolla software. Este simple e ineludible hecho invalida la mayoría de los consejos metodológicos disponibles. A menudo, el desarrollo de software no satisface las necesidades humanas, no reconoce la fragilidad humana y no aprovecha la fuerza humana. Actuar como si el software no fuera escrito por personas supone un alto costo para los participantes, su humanidad se destruye por un proceso inhumano que no reconoce sus necesidades. Esto tampoco es bueno para los negocios, con los costos y

la interrupción de la alta rotación y las oportunidades perdidas para la acción creativa [Beck and Andres, 2004]. ¿Qué necesitan las personas para ser buenos desarrolladores?

- **Seguridad básica** - estar libre de hambre, daño físico y amenazas a sus seres queridos. El miedo a perder el empleo amenaza esta necesidad.
- **Logro** - la oportunidad y la capacidad de contribuir a su sociedad.
- **Pertenencia** - la capacidad de identificarse con un grupo del que reciben validación y responsabilidad y contribuir a sus objetivos compartidos.
- **Crecimiento** - la oportunidad de ampliar sus habilidades y perspectiva.
- **Intimidad** - la capacidad de comprender y ser comprendido profundamente por los demás.

2.5.3.2. Económicos

Alguien tiene que pagar por todo esto. El desarrollo de software que no reconoce la economía corre el riesgo de la victoria vacía de un "éxito técnico". Asegúrese de que lo que está haciendo tenga valor comercial, cumpla con los objetivos comerciales y satisfaga las necesidades comerciales.

2.5.3.3. Beneficio mutuo

Cada actividad debe beneficiar a todos los interesados. El beneficio mutuo es el principio XP más importante y el más difícil de cumplir. Siempre hay soluciones para cualquier problema que le cueste a una persona mientras beneficia a otra. Cuando la situación es desesperada, estas soluciones parecen atractivas. Sin embargo, siempre son una pérdida neta, porque la mala voluntad que crean destruye las relaciones que debemos valorar. El negocio de las computadoras es realmente un negocio de personas y mantener las relaciones laborales es importante.

La extensa documentación interna del software es un ejemplo de una práctica que viola el beneficio mutuo. Se supone que debo ralentizar mi desarrollo considerablemente para que alguna persona desconocida en un futuro potencial tenga más facilidad para mantener este código. Puedo ver un posible beneficio para la persona futura en caso de que la documentación aún sea válida, pero ningún beneficio ahora.

XP resuelve el problema de la comunicación con el futuro de formas mutuamente beneficiosas:

- Escribo pruebas automatizadas que me ayudan a diseñar e implementar mejor hoy. Dejo estas pruebas para que las utilicen los futuros programadores. Esta práctica me beneficia ahora y a los mantenedores en el futuro.
- Refactorizo cuidadosamente para eliminar la complejidad accidental, dándome satisfacción y menos defectos y haciendo que el código sea más fácil de entender para quienes lo encuentren más adelante.
- Elijo nombres de un conjunto coherente y explícito de metáforas que acelera mi desarrollo y hace que el código sea más claro para los nuevos programadores.

2.5.3.4. Auto semejanza

Cuando la naturaleza encuentra una forma que funciona, la usa en todos los lugares que puede. El mismo principio se aplica al desarrollo de software: copiar la estructura de una solución en un nuevo contexto, incluso a diferentes escalas.

2.5.3.5. Mejora

En el desarrollo de software, “perfecto” es un verbo, no un adjetivo. No existe un proceso perfecto. No existe un diseño perfecto. No hay historias perfectas. Sin embargo, puede perfeccionar su proceso, su diseño y sus historias [Beck and Andres, 2004].

Si bien nuestra tecnología mejorada ha eliminado gradualmente el esfuerzo desperdiciado, nuestra mayor rigidez y estructuras sociales especializadas en las organizaciones de desarrollo son cada vez más un desperdicio. La clave para mejorar es conciliar los dos, utilizando la eficiencia tecnológica recién descubierta para permitir nuevas relaciones sociales más efectivas. Ponga a trabajar la mejora sin esperar la perfección. Encuentre un lugar de partida, comience y mejore desde allí.

2.5.3.6. Diversidad

Los equipos de desarrollo de software donde todos son iguales, aunque se sientan cómodos, no son efectivos. Los equipos deben reunir una variedad de habilidades, actitudes y perspectivas para ver los problemas y las trampas, pensar en múltiples formas de

resolver problemas e implementar las soluciones. Los equipos necesitan diversidad.

El conflicto es el compañero inevitable de la diversidad. No hay conflicto en el sentido de “nos odiamos y no podemos progresar”, sino en el sentido de “hay dos formas de resolver esto”. ¿Cómo eliges?

Dos ideas sobre un diseño presentan una oportunidad, no un problema. El principio de diversidad sugiere que los programadores deben trabajar juntos en el problema y se deben valorar ambas opiniones.

2.5.3.7. Reflexión

Los buenos equipos no solo hacen su trabajo, piensan en cómo están trabajando y por qué están trabajando. Analizan por qué tuvieron éxito o fracasaron. No intentan ocultar sus errores, sino exponerlos y aprender de ellos. Nadie tropieza con la excelencia.

Los ciclos trimestrales y semanales incluyen tiempo para la reflexión en equipo, al igual que la programación por parejas y la integración continua. Pero la reflexión no debe limitarse a las oportunidades “oficiales”. La conversación con un cónyuge o amigo, las vacaciones y la lectura y las actividades no relacionadas con el software brindan oportunidades individuales para pensar cómo y por qué está trabajando de la manera en que lo hace. Las comidas compartidas y las pausas para el café proporcionan un entorno informal para la reflexión compartida.

La reflexión no es un ejercicio puramente intelectual. Puede obtener información analizando datos, pero también puede aprender de su instinto. Las emociones “negativas” como el miedo, la ira y la ansiedad han proporcionado señales de que algo malo estaba a punto de suceder. Se necesita un esfuerzo para escuchar lo que sus emociones le dicen sobre su trabajo, pero los sentimientos templados por el intelecto son una fuente de conocimiento.

La reflexión puede llevarse demasiado lejos. El desarrollo de software tiene una larga tradición de personas tan ocupadas pensando en el desarrollo de software que no tienen tiempo para desarrollar software. La reflexión viene después de la acción. El aprendizaje es acción reflejada. Para maximizar los comentarios, la reflexión en los equipos de XP se mezcla con la práctica [Beck and Andres, 2004].

2.5.3.8. Flujo

El flujo en el desarrollo de software ofrece un flujo constante de software valioso al participar en todas las actividades de desarrollo simultáneamente. Las prácticas de XP están sesgadas hacia un flujo continuo de actividades en lugar de fases discretas.

Muchos equipos empeoran el problema al tender a responder al estrés haciendo que las partes de valor sean más grandes, desde implementar software con menos frecuencia hasta integrarlo con menos frecuencia. Menos retroalimentación empeora el problema, lo que lleva a una tendencia a fragmentos aún mayores. Cuantas más cosas se aplacen, cuanto mayor es la porción, mayor es el riesgo. En contraste, el principio de flujo sugiere que para mejorar, implemente incrementos más pequeños de valor cada vez con mayor frecuencia.

Algunas tendencias en el desarrollo de software se oponen al concepto de lotes más grandes. La construcción diaria, por ejemplo, está orientada al flujo. Sin embargo, las construcciones diarias son un pequeño paso en el camino hacia la fluidez. No es suficiente que el software se compile y se vincule todos los días; también debería funcionar correctamente todos los días o, mejor aún, varias veces al día [Beck and Andres, 2004].

2.5.3.9. Oportunidad

Aprenda a ver los problemas como oportunidades de cambio. Esto no quiere decir que no haya problemas en el desarrollo de software. Sin embargo, la actitud de “supervivencia” conduce a una resolución de problemas suficiente para salir adelante. Para alcanzar la excelencia, los problemas deben convertirse en oportunidades de aprendizaje y mejora, no solo en la supervivencia.

Es posible que no sepa qué hacer con un problema. Es posible que desee más tiempo para pensar qué hacer. A veces, el deseo de más tiempo es una máscara que se usa para protegerse del miedo a las consecuencias de ponerse en marcha. A veces, sin embargo, la paciencia resuelve un problema por sí sola.

La conversión de problemas en oportunidades se lleva a cabo a lo largo del proceso de desarrollo. Maximiza las fortalezas y minimiza las debilidades. ¿No puede hacer planes precisos a largo plazo? Bien, tenga un ciclo trimestral durante el cual refine sus planes a largo plazo. ¿Una persona sola comete demasiados errores? Bien, programe en parejas. Las prácticas son eficaces precisamente porque abordan los problemas persistentes de las

personas que desarrollan software en conjunto.

Cuando empiece a practicar XP, seguramente encontrará problemas. Parte de ser extremo es elegir conscientemente transformar cada problema en una oportunidad: una oportunidad para el crecimiento personal, profundizar las relaciones y mejorar el software [Beck and Andres, 2004].

2.5.3.10. Redundancia

Los problemas críticos y difíciles en el desarrollo de software deben resolverse de varias formas diferentes. Incluso si una solución falla por completo, las otras soluciones evitarán desastres. El costo de la redundancia está más que pagado por los ahorros de no tener el desastre.

Por ejemplo, los defectos corroen la confianza y la confianza es el gran eliminador de desperdicios. Los defectos son un problema crítico y difícil. Los defectos se abordan en XP mediante muchas de las prácticas: programación en pareja, integración continua, sentarse juntos, participación real del cliente e implementación diaria, por ejemplo. Incluso si su socio no detecta un error, alguien más sentado al otro lado de la habitación podría o podría ser detectado por la próxima integración. Algunas de estas prácticas son ciertamente redundantes y presentan algunos de los mismos defectos.

No se puede resolver el problema del defecto con una sola práctica. Es demasiado complejo, con demasiadas facetas y nunca se resolverá por completo. Lo que espera lograr son pocos defectos suficientes para mantener la confianza tanto dentro del equipo como con el cliente [Beck and Andres, 2004].

2.5.3.11. Fracaso

Si tiene problemas para tener éxito, falle. ¿No sabes cuál de las tres formas de implementar una historia? Pruébalo de tres formas. Incluso si todos fallan, seguramente aprenderá algo valioso.

¿No es un desperdicio el fracaso? No, no si imparte conocimientos. El conocimiento es valioso y, a veces, difícil de conseguir. La falla puede no ser un desperdicio evitable. Si supiera cuál es la mejor manera de implementar la historia, simplemente la implementaría de esa manera. Dado que aún no conoce la mejor forma, ¿cuál es la forma más barata de averiguarlo?

Esto no tiene la intención de excusar el fracaso cuando realmente sabía mejor. Sin embargo, cuando no sabe qué hacer, arriesgarse a fracasar puede ser el camino más corto y seguro hacia el éxito [Beck and Andres, 2004].

2.5.3.12. Calidad

Sacrificar la calidad no es eficaz como medio de control. La calidad no es una variable de control. Los proyectos no van más rápido al aceptar una calidad inferior. No van más despacio al exigir una mayor calidad. Mejorar la calidad a menudo resulta en una entrega más rápida; mientras que la reducción de los estándares de calidad a menudo resulta en una entrega tardía y menos predecible.

Si no puede controlar los proyectos controlando la calidad, ¿cómo puede controlarlos? El tiempo y el costo suelen ser fijos. XP elige el alcance como medio principal de planificación, seguimiento y dirección de proyectos. Dado que el alcance nunca se conoce con precisión de antemano, es una buena palanca. Los ciclos semanales y trimestrales proporcionan puntos explícitos para rastrear y elegir el alcance.

La preocupación por la calidad no es excusa para la inacción. Si no conoce una forma limpia de hacer un trabajo que debe hacerse, hágalo de la mejor manera posible. Si conoce un camino limpio pero tomaría demasiado tiempo, haga el trabajo tan bien como tenga tiempo por ahora. Decida terminar de hacerlo de la manera limpia más tarde. Esto ocurre a menudo durante la evolución arquitectónica, donde tienes que vivir con dos arquitecturas que resuelven el mismo problema mientras haces la transición de una a otra. Entonces, la transición en sí se convierte en una demostración de calidad: realizar un gran cambio de manera eficiente en pasos pequeños y seguros [Beck and Andres, 2004].

2.5.3.13. Pasos pequeños

Siempre es tentador hacer grandes cambios en grandes pasos. Después de todo, queda un largo camino por recorrer y poco tiempo para llegar. Un cambio momentáneo tomado de una vez es peligroso. Son las personas a las que se les pide que cambien. El cambio es inquietante. La gente cambia muy rápido.

Los pasos de bebé no justifican la estasis o el cambio glacial. En las condiciones adecuadas, las personas y los equipos pueden dar muchos pasos pequeños con tanta rapidez que parecen estar dando saltos.

Los pequeños pasos reconocen que la sobrecarga de los pequeños pasos es mucho menor que cuando un equipo retrocede inútilmente ante grandes cambios abortados. Los pequeños pasos se expresan en prácticas como la programación de prueba primero, que avanza una prueba a la vez, y la integración continua, que integra y prueba el valor de algunas horas de cambios a la vez [Beck and Andres, 2004].

2.5.3.14. Aceptar responsabilidad

No se puede asignar responsabilidad; solo se puede aceptar. Si alguien intenta responsabilizarte, solo tú puedes decidir si eres responsable o no. Las prácticas reflejan la responsabilidad aceptada, por ejemplo, sugerir que quien se inscribe para hacer un trabajo también la estima. De manera similar, la persona responsable de implementar una historia es en última instancia responsable del diseño, implementación y prueba de la historia. Con la responsabilidad viene la autoridad. Las desalineaciones distorsionan la comunicación del equipo. Cuando un experto en procesos puede decirme cómo trabajar, pero no comparte ese trabajo o sus consecuencias, la autoridad y la responsabilidad están desalineadas.

2.5.4. Prácticas

2.5.4.1. Prácticas primarias

1. *Sit Together*

“Sit Together” predice que cuanto más tiempo presencial tenga, más humano y productivo será el proyecto. Si tienes un proyecto de varios sitios y todo va bien, sigue haciendo lo que estás haciendo. Si tiene problemas, piense en formas de sentarse más juntos, incluso si eso significa viajar.

2. *Whole Team*

Incluir en el equipo a personas con todas las habilidades y perspectivas necesarias para que el proyecto tenga éxito. En realidad, esto no es más que la vieja idea de equipos multifuncionales. El nombre refleja el propósito de la práctica, una sensación de integridad en el equipo, la disponibilidad inmediata de todos los recursos necesarios para tener éxito. Cuando las interacciones intensas son necesarias para

la salud del proyecto, quienes interactúan deben identificarse principalmente con el equipo y no con sus funciones.

La gente necesita un sentido de “equipo”:

- Nosotros pertenecemos.
- Estamos en esto juntos.
- Apoyamos el trabajo, el crecimiento y el aprendizaje de los demás.

Lo que constituye un “equipo completo” es dinámico. Si un conjunto de habilidades o actitudes se vuelve importante, traiga al equipo a una persona con estas habilidades. Si alguien ya no es necesario, puede ir a otra parte.

3. *Informative Workspace*

Haga de su espacio de trabajo su trabajo. Un observador interesado debería poder entrar en el espacio del equipo y tener una idea general de cómo va el proyecto en quince segundos. Debería poder obtener más información sobre problemas reales o potenciales al mirar más de cerca. Muchos equipos implementan esta práctica en parte colocando tarjetas de historias en la pared. Ordenar las tarjetas espacialmente transmite información rápidamente. Si el área “Listo” no recopila tarjetas, ¿qué necesita mejorar el equipo en su planificación, estimación o ejecución? También me preguntaré qué necesitan los clientes para que el alcance deslizante tenga un impacto comercial mínimo. La Figura 2.3 muestra un muro de historias idealizado con historias ordenadas espacialmente.

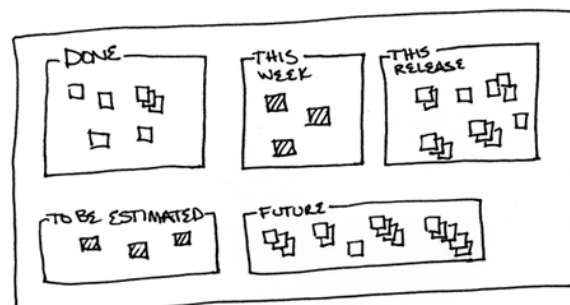


Figura 2.3: Historias de usuario en una pared

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

El espacio de trabajo (Figura 2.4) también debe cubrir otras necesidades humanas. El agua y los bocadillos brindan consuelo y fomentan las interacciones sociales positivas. La limpieza y el orden dejan la mente libre para pensar en los problemas en cuestión. Si bien la programación ocurre en un espacio público, las personas también necesitan privacidad, que puede proporcionarse mediante cubos separados o limitando las horas de trabajo.

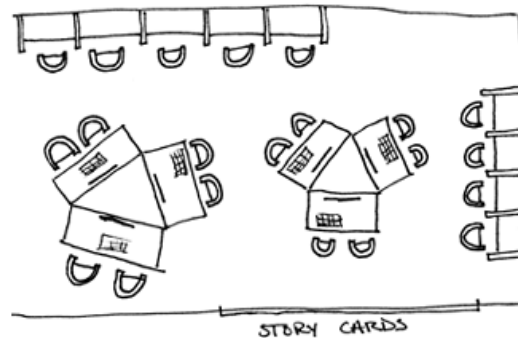


Figura 2.4: Espacio de trabajo de un equipo

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

4. *Energized Work*

Trabaje solo tantas horas como pueda ser productivo y solo tantas horas como pueda mantener. Quemarse improductivamente hoy y estropear el trabajo de los dos días siguientes no es bueno para usted ni para el equipo. Es fácil quitar valor a un proyecto de software; pero cuando estás cansado, es difícil reconocer que estás quitando valor[Beck and Andres, 2004].

Cuando esté enfermo, respétese a sí mismo y al resto de su equipo descansando y recuperándose. Cuidar de sí mismo es la forma más rápida de volver al trabajo lleno de energía. También protege al equipo de perder más productividad debido a una enfermedad. Llegar enfermo no demuestra compromiso con el trabajo, porque cuando lo haces no estás ayudando al equipo.

Puede realizar mejoras incrementales en las horas de trabajo. Permanezca en el trabajo la misma cantidad de tiempo, pero adminístrelo mejor. Declare un tramo de dos horas cada día como Código de tiempo. Apague los teléfonos y las notificaciones por correo electrónico, y programe durante dos horas. Eso puede ser una mejora

suficiente por ahora y puede preparar el escenario para menos horas de trabajo más adelante [Beck and Andres, 2004].

5. *Pair Programming*

Escriba todos los programas de producción con dos personas sentadas en una máquina. Configure la máquina para que los socios puedan sentarse cómodamente uno al lado del otro. Mueva el teclado y el mouse hacia adelante y hacia atrás para que se sienta cómodo mientras escribe. La programación en pareja es un diálogo entre dos personas que programan simultáneamente (y analizan, diseñan y prueban) y tratan de programar mejor. Programadores en pareja:

- Mantengan el uno al otro en la tarea.
- Haga una lluvia de ideas para mejorar el sistema.
- Aclarar ideas.
- Tome la iniciativa cuando su pareja esté estancada, reduciendo así la frustración.
- Háganse responsables unos a otros de las prácticas del equipo.

6. *Pairing and Personal Space*

Diferentes individuos y culturas se sienten cómodos con diferentes cantidades de espacio corporal. Emparejar con un italiano que se comunica mejor cuando está muy cerca es completamente diferente a emparejar con un danés al que le gustan unos pocos pies de espacio personal. Si no es consciente de la diferencia, puede resultar muy incómodo. Se debe respetar el espacio personal para que ambas partes funcionen bien.

La higiene personal y la salud son cuestiones importantes a la hora de emparejar. Cúbrase la boca cuando tosa. No venga a trabajar cuando esté enfermo. Evite las colonias fuertes que puedan afectar a su pareja.

Trabajar juntos de forma eficaz se siente bien. Para algunos, puede ser una nueva experiencia en el lugar de trabajo. Cuando los programadores no son lo suficientemente maduros emocionalmente como para separar la aprobación de la excitación, trabajar con una persona del sexo opuesto puede hacer surgir sentimientos sexuales

que no son lo mejor para el equipo. Si estos sentimientos surgen cuando se empareja, deje de emparejarse con la persona hasta que haya asumido la responsabilidad y se haya ocupado de sus sentimientos. Incluso si los sentimientos son mutuos, actuar en consecuencia dañará al equipo. Si desea tener una relación íntima, uno de ustedes debe dejar el equipo para que pueda construir una relación personal en un entorno personal sin confundir la comunicación del equipo con un subtexto sexual. Idealmente, las emociones en el trabajo estarán relacionadas con el trabajo [Beck and Andres, 2004].



Figura 2.5: Programación en pares

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

7. *Stories*

El desarrollo de software ha sido mal dirigido por la palabra “requisito”, definida en el diccionario como “algo obligatorio u obligatorio”. La palabra tiene una connotación de absolutismo y permanencia, inhibidores de la aceptación del cambio. Y la palabra “requisito” es simplemente incorrecta. De mil páginas de “requisitos”, si implementa un sistema con el 20 % o 10 % o incluso el 5 % correcto, probablemente obtendrá todos los beneficios comerciales previstos para todo el sistema. Entonces, ¿cuál fue el otro 80 %? No “requisitos”; en realidad no eran obligatorios ni obligatorios.

La estimación temprana es una diferencia clave entre las historias y otras prácticas de requisitos. La estimación brinda a las perspectivas comerciales y técnicas la oportunidad de interactuar, lo que crea valor temprano, cuando una idea tiene el mayor potencial. Cuando el equipo conoce el costo de las funciones, puede dividir, combinar o ampliar el alcance en función de lo que sabe sobre el valor de las funciones.

Dé a las historias nombres cortos además de una breve prosa o una descripción gráfica. Escriba las historias en fichas y colóquelas en una pared que pase con frecuencia. La Figura 2.6 es una tarjeta de muestra de una historia que deseo que implemente mi programa de escáner. Cada intento que he visto de computarizar historias no ha logrado proporcionar una fracción del valor de tener cartas reales en una pared real. Si necesita informar el progreso a otras partes de la organización en un formato familiar, traduzca las tarjetas a ese formato periódicamente [Beck and Andres, 2004].

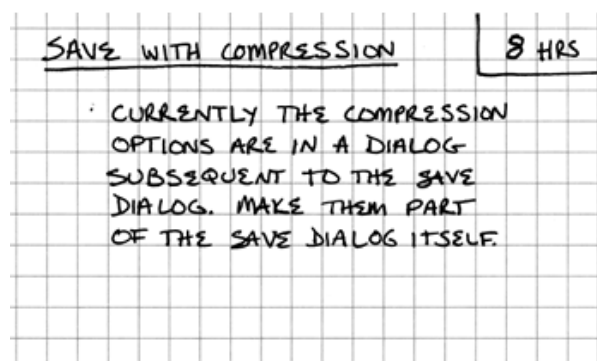


Figura 2.6: Tarjeta de historia de usuario

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

8. *Weekly Cycle*

Planifique el trabajo una semana a la vez. Tenga una reunión al comienzo de cada semana. Durante esta reunión:

- Revise el progreso hasta la fecha, incluyendo cómo el progreso real de la semana anterior coincidió con el progreso esperado.
- Haga que los clientes elijan historias de una semana para implementar esta semana.

- Divida las historias en tareas. Los miembros del equipo se registran para las tareas y las estiman.

Comience la semana escribiendo pruebas automatizadas que se ejecutarán cuando se completen las historias. Luego, pase el resto de la semana completando las historias y haciendo que las pruebas pasen. Un equipo orgulloso de su trabajo implementará completamente las historias, no solo hará el trabajo suficiente para que las pruebas pasen. El objetivo es tener software implementable al final de la semana que todos puedan celebrar como progreso.

9. *Quarterly Cycle*

Planifique el trabajo un cuarto a la vez. Una vez al trimestre, reflexione sobre el equipo, el proyecto, su progreso y su alineación con objetivos más amplios.

Durante la planificación trimestral:

- Identificar los cuellos de botella, especialmente los controlados fuera del equipo.
- Inicie las reparaciones.
- Planifique el tema o temas para el trimestre.
- Elija una cuarta parte de las historias para abordar esos temas.
- Concéntrese en el panorama general, donde el proyecto encaja dentro de la organización.

Una temporada es otra escala de tiempo natural y ampliamente compartida para usar en la organización del tiempo para un proyecto. El uso de un trimestre como horizonte de planificación se sincroniza muy bien con otras actividades comerciales que ocurren trimestralmente. Los trimestres también son un intervalo cómodo para la interacción con proveedores y clientes externos. La separación de "temas" de "historias" tiene como objetivo abordar la tendencia del equipo a concentrarse y entusiasmarse con los detalles de lo que están haciendo sin reflexionar sobre cómo las historias de esta semana encajan en el panorama general. Los temas también encajan bien en la planificación a mayor escala, como la elaboración de hojas de ruta de marketing [Beck and Andres, 2004].

10. *Slack*

En cualquier plan, incluya algunas tareas menores que pueden descartarse si se retrasa. Siempre puede agregar más historias más tarde y entregar más de lo prometido. Es importante en un ambiente de desconfianza y promesas incumplidas cumplir con sus compromisos. Algunos compromisos cumplidos contribuyen en gran medida a reconstruir las relaciones.

11. *Ten-Minute Build*

Cree automáticamente todo el sistema y ejecute todas las pruebas en diez minutos. Una compilación que demore más de diez minutos se usará con mucha menos frecuencia, perdiendo la oportunidad de recibir *feedback*. La construcción de diez minutos es ideal. ¿Qué haces en tu camino hacia ese ideal? La declaración de la práctica da tres pistas: construya automáticamente todo el sistema y ejecute todas las pruebas en diez minutos. Si su proceso no está automatizado, ese es el primer lugar para comenzar. Entonces es posible que pueda construir solo la parte del sistema que ha cambiado. Finalmente, es posible que pueda ejecutar solo pruebas que cubran la parte del sistema en riesgo debido a los cambios que realizó. Cualquier suposición sobre qué partes del sistema deben construirse y qué partes deben probarse presenta el riesgo de error. Si está equivocado, puede pasar por alto errores impredecibles con todos sus costos sociales y económicos. Sin embargo, poder probar parte del sistema es mucho mejor que no poder probar ninguno. Las compilaciones automatizadas son mucho más valiosas que las compilaciones que requieren intervención manual. A medida que aumenta el nivel de estrés general, las compilaciones manuales tienden a realizarse con menos frecuencia y menos bien, lo que genera más errores y más estrés. Las prácticas deberían reducir el estrés. Una construcción automatizada se convierte en un alivio del estrés en el momento decisivo. “¿Cometimos un error? Construyamos y veamos[Beck and Andres, 2004]”.

12. *Continuous Integration*

Integre y pruebe los cambios después de no más de un par de horas. La programación en equipo no es un problema de dividir y conquistar. Es un problema de dividir, conquistar e integrar. El paso de integración es impredecible, pero fácilmente puede llevar más tiempo que la programación original. Cuanto más espere para integrarse,

más cuesta y más impredecible se vuelve el costo. Integrar y construir un producto completo. Si el objetivo es grabar un CD, grabe un CD. Si el objetivo es implementar un sitio web, implemente un sitio web, incluso si es en un entorno de prueba. La integración continua debe ser lo suficientemente completa para que la primera implementación final del sistema no sea gran cosa [Beck and Andres, 2004].

13. *Test-First Programming*

La comunidad de XP no ha explorado mucho las alternativas a las pruebas para verificar el comportamiento del sistema. Se podrían utilizar herramientas como el análisis estático y la verificación de modelos al estilo de prueba primero. Empieza con una “prueba” que dice, por ejemplo, que no hay puntos muertos en el sistema. Después de cada cambio, verifica nuevamente que no haya puntos muertos. Las herramientas de análisis estático que he visto no están diseñadas para usarse de esta manera. Funcionan demasiado lentamente para formar parte del ciclo de programación minuto a minuto. Sin embargo, esto parece ser simplemente una cuestión de enfoque, no una limitación fundamental. En las pruebas continuas, las pruebas se ejecutan en cada cambio de programa, al igual que un compilador incremental se ejecuta en cada cambio en el código fuente. Los fallos de prueba se informan en el mismo formato que los errores del compilador. Las pruebas continuas reducen el tiempo para corregir errores al reducir el tiempo para descubrirlos. Sin embargo, las pruebas deben ejecutarse rápidamente. Las pruebas que escribe mientras codifica test-first tienen la limitación de que toman una microvista del programa: ¿estos dos objetos funcionan bien juntos? A medida que su experiencia crezca, podrá expresar más y más seguridad en estas pruebas. Debido a su alcance limitado, estas pruebas tienden a ejecutarse muy rápido. Puede ejecutar miles de ellos como parte de la compilación de diez minutos [Beck and Andres, 2004].

14. *Incremental Design*

Invierta en el diseño del sistema todos los días. Esfuércese por hacer que el diseño del sistema se adapte perfectamente a las necesidades del sistema ese día. Cuando su comprensión del mejor diseño posible salte adelante, trabaje de manera gradual pero persistente para volver a alinear el diseño con su comprensión. El consejo para los equipos de XP no es minimizar la inversión en diseño a corto plazo, sino

mantener la inversión en diseño en proporción a las necesidades del sistema hasta el momento. La cuestión no es si diseñar o no, la cuestión es cuándo diseñar. El diseño incremental sugiere que el momento más eficaz para diseñar es a la luz de la experiencia. Si los pasos pequeños y seguros son cómo diseñar, la siguiente pregunta es en qué parte del sistema se debe mejorar el diseño. La heurística simple que he encontrado útil es eliminar la duplicación. Si tengo la misma lógica en dos lugares, trabajo con el diseño para entender cómo puedo tener solo una copia. Los diseños sin duplicación tienden a ser fáciles de cambiar. No se encuentra en la situación en la que tenga que cambiar el código en varios lugares para agregar una función. A medida que más equipos invierten en el diseño diario, notan que los cambios que están realizando son similares independientemente del propósito del sistema. La refactorización es una disciplina de diseño que codifica estos patrones recurrentes de cambios. Estas refactorizaciones pueden ocurrir en cualquier nivel de escala. Pocas decisiones de diseño son difíciles de cambiar una vez tomadas. El resultado son sistemas que pueden comenzar con poco y crecer según sea necesario sin un costo exorbitante [Beck and Andres, 2004].

2.5.4.2. Prácticas corolarias

Las prácticas corolarias son difíciles o peligrosas de implementar antes de completar el trabajo preliminar de las prácticas primarias. Si se comienza a implementar a diario, por ejemplo, sin que la tasa de defectos se acerque a cero (con programación de pares, integración continua y programación de prueba primero); tendrás un desastre en tus manos. Confíe en su olfato sobre lo que necesita mejorar a continuación. Si alguna de las siguientes prácticas parece apropiada, pruébela. Podría funcionar o podría descubrir que tiene más trabajo por hacer antes de poder usarlo para mejorar su proceso de desarrollo.

- Real Customer Involvement
- Incremental Deployment
- Team Continuity
- Shrinking Teams
- Root-Cause Analysis

- Shared Code
- Code and Tests
- Single Code base
- Daily Deployment
- Negotiated Scope Contract
- Pay-Per-Use

2.5.5. Planificación

Planificar en XP es como ir de compras. Imagínese que entra en una tienda de comestibles con \$ 100 en el bolsillo. Los artículos en los estantes tienen un precio adjunto. Algunos elementos que necesita; otros no; y otros que desee, pero no se ajustan a su presupuesto. Si llega a la caja con \$ 101 en comida, tendrá que devolver algo. Su trabajo al comprar es gastar sus \$ 100 sabiamente, comprando lo que necesita y tanto como sea posible.

En XP, las compras son las historias. Los precios son las estimaciones adjuntas a las historias. El presupuesto es la cantidad de tiempo disponible. La fecha de implementación deseada generalmente se establece al principio de un proyecto, para que sepa cuánto tiene que gastar en historias. Si tiene doscientos pares de horas en su bolsillo y tiene cuatrocientos pares de horas de historias en su carrito, elija el conjunto de historias más valioso que sume hasta doscientos pares de horas. De lo contrario, todos saben que tiene más en el carrito de lo que puede pagar.

Parte de la planificación es decidir qué hacer a continuación entre todas las posibilidades. La planificación es complicada porque las estimaciones del costo y el valor de las historias son inciertas. La información en la que basa estas decisiones cambia. Usamos la retroalimentación para mejorar nuestras estimaciones y tomar decisiones lo más tarde posible para que se basen en la mejor información posible. Es por eso que la planificación es una actividad diaria, semanal y trimestral en XP. El plan puede cambiar para adaptarse a los hechos a medida que surgen. Los planes no son predicciones del futuro. En el mejor de los casos, expresan todo lo que sabe hoy sobre lo que podría suceder mañana. Su incertidumbre no niega su valor. Los planes le ayudan a coordinarse con otros equipos.

Los planes le brindan un lugar para comenzar. Los planes ayudan a todos en el equipo a tomar decisiones alineadas con los objetivos del equipo [Beck and Andres, 2004].

Para estimar una historia, imagina, dado todo lo que sabes sobre historias similares, cuántas horas o días le tomará a un par completar la historia. “Completo” significa listo para el despliegue; incluyendo todas las pruebas, implementación, refactorización y discusiones con los usuarios. A medida que aumente su conocimiento de historias similares, sus estimaciones mejorarán. Las estimaciones se basan en un par razonable que trabaja en la historia. Algunos pares pueden ser mejores y otros peores, pero si todos se turnan para estimar, todos deberían promediar al final.

Al principio, estas estimaciones pueden ser tremendamente erróneas. Las estimaciones basadas en la experiencia son más precisas. Es importante recibir comentarios lo antes posible para mejorar sus estimaciones. Si tiene un mes para planificar un proyecto en detalle, dedíquelo a desarrollar cuatro iteraciones de una semana mientras mejora sus estimaciones. Si tiene una semana para planificar un proyecto, realice cinco iteraciones de un día. Los ciclos de retroalimentación le brindan información y la experiencia para realizar estimaciones precisas. Obtenga esta experiencia lo antes posible para mejorar sus estimaciones.

2.5.6. Diseño

El diseño incremental es una forma de entregar funcionalidad temprana y continuar brindando funcionalidad semanalmente durante la vida del proyecto. XP lleva el incrementalismo del diseño al límite, lo que sugiere que los proyectos se ejecutan con mayor fluidez si el diseño es parte del trabajo diario [Beck and Andres, 2004].

En el mundo físico, las transformaciones individuales cuestan demasiado para que el diseño incremental funcione bien. Las etapas intermedias tienen muy poco valor o los cambios son demasiado costosos (las piezas se destruyen y es costoso duplicarlas en el proceso de reconstrucción). Incluso las estructuras físicas se someten a un diseño y construcción incrementales, y la experiencia de la estructura existente informa la siguiente etapa del diseño de formas que la especulación no puede [Beck and Andres, 2004].

Parte de lo que hace que el diseño incremental sea valioso en el software es que a menudo escribimos aplicaciones por primera vez. Incluso si esta es la enésima variación de un tema, siempre hay una mejor manera de diseñar el software. Como el diseño tiene

influencia y las ideas de diseño mejoran con la experiencia, la paciencia es una de las habilidades más valiosas que puede poseer un diseñador de software. Hay un arte en diseñar lo suficiente para obtener retroalimentación y luego usar esa retroalimentación para mejorar el diseño lo suficiente como para obtener la siguiente ronda de retroalimentación [Beck and Andres, 2004].

Los siguientes gráficos nos ayudarán a visualizar por sí mismo cuándo debe diseñar. El eje vertical del gráfico es la calidad del diseño. Hay una línea horizontal para la calidad mínima del diseño necesaria para el éxito. El diseño de software es curioso, ya que generalmente hay muchos diseños que son lo suficientemente buenos para que el software tenga éxito. La calidad del diseño no garantiza el éxito, pero el fracaso del diseño puede garantizar el fracaso [Beck and Andres, 2004].

Cada gráfico tendrá tres puntos, uno para cómo diseñarías “por instinto”, otro para cómo diseñarías si pensaras mucho en el diseño y otro para cómo diseñarías a la luz de la experiencia. La relación de los tres puntos y la ubicación del umbral de diseño mínimo indicará si diseñar por adelantado es una opción para usted, o si estaría mejor con un diseño incremental [Beck and Andres, 2004].

La figura 2.7 es un gráfico en el que el diseño puramente instintivo es suficiente. Cualquier diseño antiguo servirá. Puede seguir adelante y diseñar hoy y tener éxito.

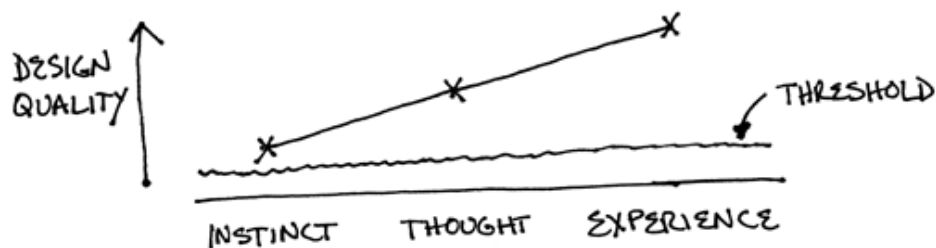


Figura 2.7: Cualquier diseño antiguo servirá

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

La Figura 2.8 es un escenario en el que la cuestión de cuándo diseñar no es tan clara. Pensar detenidamente conduciría a una respuesta suficientemente buena, pero la experiencia conduciría a una mejor respuesta. No tiene la opción de no diseñar en absoluto, porque el diseño inconsciente conducirá al fracaso. ¿Debería hacer la mayor parte de su inversión en diseño ahora o esperar hasta tener algo de experiencia?

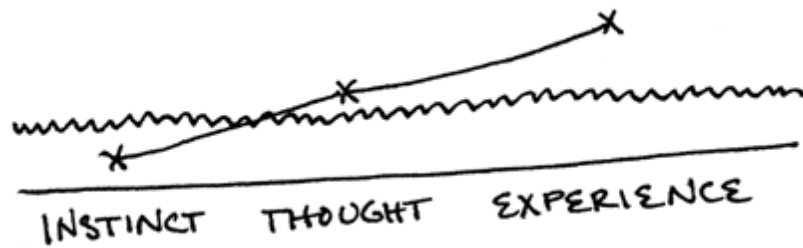


Figura 2.8: Se necesita algo de pensamiento o experiencia en diseño

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

La Figura 2.9 es un caso donde diseño incremental es inevitable. Ninguna cantidad de pensamiento sin experiencia resultará en un diseño que sea lo suficientemente bueno. Solo la experiencia dará como resultado la comprensión suficiente para producir un diseño suficientemente bueno.

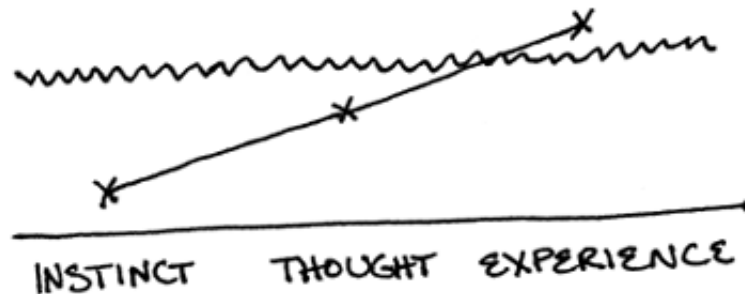


Figura 2.9: Ninguna cantidad de pensamiento puro será suficiente

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

Un factor a tener en cuenta para decidir cuándo diseñar es el valor disponible a través de las diferentes estrategias. Si el pensamiento puro crea la mayor parte del valor sin retroalimentación (Figura 2.10), diseñar antes tiene más sentido. Si la experiencia crea la mayor parte del valor (Figura 2.11), diseñar solo lo suficiente hoy para comenzar y luego diseñar principalmente a la luz de la experiencia tiene más sentido.

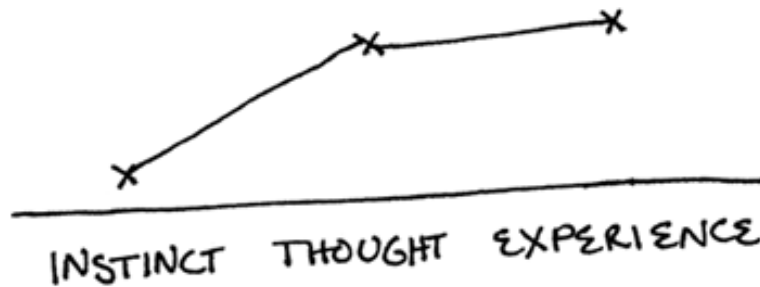


Figura 2.10: La experiencia no ayuda

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.



Figura 2.11: Mucho que aprender de la experiencia

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

2.5.6.1. Simplicidad

Los equipos de XP prefieren soluciones simples siempre que sea posible. A continuación, se muestran cuatro criterios utilizados para evaluar la simplicidad de un diseño.

- **Apropiado para la audiencia destinataria.** No importa lo brillante y elegante que sea una pieza de diseño; si las personas que necesitan trabajar con él no lo entienden, no es sencillo para ellos.
- **Comunicativo.** Cada idea que necesita ser comunicada está representada en el sistema. Como las palabras de un vocabulario, los elementos del sistema se comunican con los futuros lectores.
- **Factorizado.** La duplicación de lógica o estructura dificulta la comprensión y modificación del código.

- **Mínimo.** Dentro de las tres restricciones anteriores, el sistema debe tener la menor cantidad de elementos posible. Menos elementos significa menos para probar, documentar y comunicar.

2.5.7. Pruebas

Los defectos destruyen la confianza necesaria para un desarrollo de software eficaz. Los clientes deben poder confiar en el software. Los gerentes deben poder confiar en los informes de progreso. Los programadores deben poder confiar entre ellos. Los defectos destruyen esta confianza. Sin confianza, las personas pasan gran parte de su tiempo defendiéndose de la posibilidad de que otra persona haya cometido un error [Beck and Andres, 2004].

Aquí está el dilema en el desarrollo de software: los defectos son costosos, pero eliminarlos también es costoso. Sin embargo, la mayoría de los defectos terminan costando más de lo que hubiera costado prevenirlos. Los defectos son costosos cuando ocurren, tanto los costos directos de arreglar los defectos como los costos indirectos debido a relaciones dañadas, negocios perdidos y tiempo de desarrollo perdido. Las prácticas de XP tienen como objetivo comunicarse claramente para que no surjan defectos en primer lugar y, cuando lo hagan, asegurarse de que el equipo las utilice para aprender a evitar problemas similares en el futuro [Beck and Andres, 2004].

Siempre habrá defectos. Surgirán circunstancias inesperadas. En situaciones novedosas e imprevistas, es probable que el software haga algo que el autor no habría tenido la intención de haber sabido de antemano la situación.

Los niveles aceptables de defectos varían. Uno de los objetivos del desarrollo es reducir la aparición de defectos a un nivel económicamente sostenible. Este nivel es diferente para diferentes tipos de software.

Otro objetivo del desarrollo es reducir la aparición de defectos a un nivel en el que la confianza pueda crecer razonablemente en el equipo. La inversión en la reducción de defectos tiene sentido como inversión en el trabajo en equipo. Los errores introducidos por un programador dificultan que todos los demás hagan su trabajo. Cada error de un miembro del equipo que afecta a otro le cuesta tiempo, energía y confianza al equipo. El buen trabajo y el buen trabajo en equipo fortalecen la moral y la confianza. Si puede respetar y confiar en sus colegas, podrá ser más productivo y disfrutar más de su trabajo. Ocultar errores para protegerse, aunque a veces parece necesario, es una enorme pérdida

de tiempo y energía. La confianza da energía a los participantes. Nos sentimos bien cuando las cosas funcionan bien. Necesitamos estar seguros para experimentar y cometer errores. Necesitamos pruebas para responsabilizarnos de nuestra experimentación, de modo que podamos estar seguros de que no estamos haciendo daño [Beck and Andres, 2004].

El aumento del costo de los defectos (DCI) es el segundo principio que se aplica en XP para aumentar la rentabilidad de las pruebas. DCI es una de las pocas verdades verificadas empíricamente sobre el desarrollo de software: cuanto antes encuentre un defecto, más barato será repararlo. Si encuentra un defecto después de una década de implementación, tendrá que reconstruir una gran cantidad de historia y contexto para averiguar qué se suponía que debía hacer el código en primer lugar, cuáles de esas suposiciones son erróneas y qué debería corregirse. por lo que el resto del programa (presumiblemente correcto) permanece inalterado. Detecte el mismo defecto en el momento en que se crea y el costo de reparación será mínimo [Beck and Andres, 2004].

DCI implica que los estilos de desarrollo de software con ciclos de retroalimentación largos (Figura 2.12) serán costosos y tendrán muchos defectos residuales. El presupuesto para encontrar y reparar defectos es limitado. Cuanto mayor sea el costo de búsqueda y reparación de defectos, más defectos permanecerán en el código implementado.

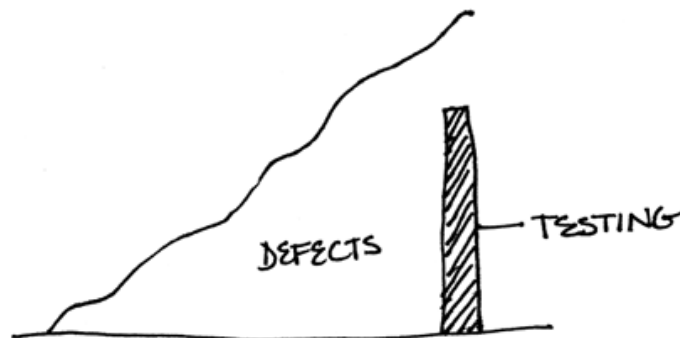


Figura 2.12: Las pruebas tardías y costosas dejan muchos defectos

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

XP usa DCI a la inversa para reducir tanto el costo de reparar defectos como el número de defectos implementados. Al incorporar las pruebas automatizadas al ciclo interno de la programación (Figura 2.13), XP intenta corregir los defectos de manera más rápida y económica. Esto les da a los equipos de XP la oportunidad de desarrollar software a bajo

costo con muy pocos defectos para los estándares de sus contemporáneos.



Figura 2.13: Las pruebas frecuentes reducen los costos y los defectos

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

Si los programadores escriben pruebas, es posible que aún se necesite otra perspectiva del sistema. Un programador o incluso una pareja aportan a su código y prueban un punto de vista singular sobre el funcionamiento del sistema, perdiendo parte del valor de la doble verificación. La verificación doble funciona mejor cuando dos procesos de pensamiento distintos llegan a la misma respuesta. Por eso es peligroso copiar los resultados de un cálculo como su valor esperado. Solo lo has pensado una vez. Es mucho mejor calcular un ejemplo a mano para tener una segunda perspectiva [Beck and Andres, 2004].

Para obtener todos los beneficios de la doble verificación, en XP hay dos conjuntos de pruebas: un conjunto está escrito desde la perspectiva de los programadores, probando los componentes del sistema de manera exhaustiva, y otro conjunto está escrito desde la perspectiva de los clientes o usuarios, probando el funcionamiento del sistema en su conjunto. Estas pruebas se comprueban entre sí. Si las pruebas de los programadores son perfectas, las pruebas del cliente no detectarán ningún error [Beck and Andres, 2004].

La inmediatez de las pruebas en XP también implica que las pruebas deben automatizarse. Con el tiempo, al mejorar el diseño y personalizar las herramientas de desarrollo, el equipo reduce el costo de automatizar las pruebas hasta el punto de que todas las pruebas están automatizadas. Las pruebas automatizadas rompen el ciclo de estrés (Figura 2.14).



Figura 2.14: El ciclo de Stress

Fuente: “Extreme Programming Explained: Embrace Change, 2nd Ed.” por Beck, Kent and Andres - Cynthia, 2004, Copyright by Addison-Wesley Professional.

En XP, las pruebas son tan importantes como la programación. Escribir y ejecutar las pruebas le da al equipo la oportunidad de hacer un trabajo del que puede estar orgulloso. La ejecución de pruebas le da al equipo una base válida para la confianza a medida que avanza rápidamente en direcciones imprevistas. Las pruebas aportan valor al desarrollo fortaleciendo las relaciones de confianza dentro del equipo y con los clientes.

2.6. Manifiesto por el Desarrollo Ágil de Software



Figura 2.15: Manifiesto por el desarrollo ágil de software

Fuente: “Manifiesto Ágil:” por Beck et al. ,2001 (<https://agilemanifesto.org/iso/es/manifesto.html>). Copyright 2001 por Beck et al.

2.7. Tecnologías BackEnd para la integración del proyecto Saiku Analytics

2.7.1. Lenguaje de programación Java

El lenguaje de programación Java TM es un lenguaje orientado a objetos de uso general, concurrente, fuertemente tipado y basado en clases. Normalmente se compila con el conjunto de instrucciones de código de bytes y el formato binario definido en la Especificación de máquina virtual de Java [“JavaTM Programming Language”, 2020].

2.7.2. Java Specification Requests(JSRs)

Las solicitudes de especificación de Java (JSR) son las descripciones reales de las especificaciones propuestas y finales para la plataforma Java [[“JSRs: Java Specification Requests”, 2020](#)].

2.7.2.1. JSR-366

Este JSR es para desarrollar Java EE 8, la próxima versión de Java Platform, Enterprise Edition. El enfoque principal de esta versión es el soporte para HTML5 y el estándar emergente HTTP 2.0; simplificación mejorada e integración de beans gestionados; e infraestructura mejorada para aplicaciones que se ejecutan en la nube. Desde su inicio, la plataforma Java EE ha tenido como objetivo liberar al desarrollador de las tareas de infraestructura comunes a través de su modelo basado en contenedores y la abstracción del acceso a los recursos. En lanzamientos recientes, la plataforma ha simplificado considerablemente las API para acceder a los servicios de contenedor al tiempo que amplía la gama de servicios disponibles. En esta versión, nuestro objetivo es continuar en la dirección de una simplificación mejorada, al tiempo que ampliamos el rango de la plataforma Java EE para abarcar tecnologías emergentes en el espacio web y en la tecnología en la nube [[“JSR 366: Java Platform, Enterprise Edition 8 \(Java EE 8\) Specification”, 2020](#)].

2.7.2.2. JSR-47

Una especificación para las API de registro dentro de la plataforma JavaTM. Estas API serán adecuadas para registrar eventos desde la plataforma Java y desde las aplicaciones Java [[“JSR 47: Logging API Specification”, 2020](#)].

Se prevé que:

- Será posible habilitar o deshabilitar el registro en tiempo de ejecución.
- Será posible controlar el registro con una granularidad bastante fina, de modo que el registro se pueda habilitar o deshabilitar para una funcionalidad específica.
- Las API de registro permitirán el registro de los servicios de registro en tiempo de ejecución, por lo que terceros pueden agregar nuevos servicios de registro.
- Será posible proporcionar servicios de puente que conecten las API de registro de Java a los servicios de registro existentes (por ejemplo, registros del sistema

operativo).

- Cuando corresponda, las API de registro también admitirán la visualización de mensajes de alta prioridad a los usuarios finales.

2.7.2.3. JSR-338

La API de persistencia de Java es la API de Java para la gestión de la persistencia y el mapeo de objetos / relacionales en entornos Java EE y Java SE. Proporciona una función de mapeo relacional / de objetos para el desarrollador de aplicaciones Java que utiliza un modelo de dominio Java para administrar una base de datos relacional. El propósito de la especificación Java Persistence 2.1 es ampliar la API de persistencia Java para incluir características adicionales solicitadas por la comunidad [[“JSR 338: Java™ Persistence 2.2”, 2020](#)].

2.7.2.4. JSR-346

Contexts and Dependency Injection for Java EE (CDI) 1.0 se introdujo como parte de la plataforma Java EE 6 y se ha convertido rápidamente en uno de los componentes más importantes y populares de la plataforma. CDI define un poderoso conjunto de servicios complementarios que ayudan a mejorar la estructura del código de la aplicación [[“JSR 346: Contexts and Dependency Injection for Java™ EE 1.1”, 2020](#)].

- Un ciclo de vida bien definido para objetos con estado vinculados a contextos de ciclo de vida, donde el conjunto de contextos es extensible
- Un mecanismo de inyección de dependencias sofisticado y con seguridad de tipos, que incluye
- la capacidad de seleccionar dependencias en el momento del desarrollo o la implementación, sin una configuración detallada
- Soporte para la modularidad de Java EE y la arquitectura de componentes de Java EE: la estructura modular de una aplicación Java EE se tiene en cuenta al resolver las dependencias entre los componentes de Java EE.
- Integración con el lenguaje de expresión unificado (EL), lo que permite que cualquier objeto contextual se utilice directamente dentro de una página JSF o JSP

- La capacidad de decorar objetos inyectados.
- La capacidad de asociar interceptores a objetos a través de enlaces de interceptores de tipo seguro.
- Un modelo de notificación de eventos
- Un contexto de conversación web además de los tres contextos web estándar definidos por la especificación Java Servlets.
- Un SPI que permite que las extensiones portátiles se integren limpiamente con el contenedor

2.7.2.5. JSR-369

El objetivo principal de este JSR es mostrar la compatibilidad con el próximo estándar IETF HTTP / 2 a los usuarios de la API de Servlet. Un objetivo secundario es actualizar la API de Servlet para lograr el cumplimiento de las nuevas funciones en HTTP 1.1, así como responder a las opiniones de la comunidad. HTTP / 2 todavía está en desarrollo activo en un grupo de trabajo (WG) de IETF, sin embargo, se espera que se complete mucho antes de la fecha de finalización de Java EE 8. Si bien en general es una mala idea rastrear un objetivo en movimiento como una especificación IETF desarrollada activamente, varios factores mitigan el riesgo en este caso [[“JSR 369: Java™ Servlet 4.0 Specification”, 2020](#)].

2.7.2.6. JSR-370

Los eventos enviados por el servidor (SSE) es una nueva tecnología definida como parte del conjunto de recomendaciones HTML5 para que un cliente (por ejemplo, un navegador) obtenga actualizaciones automáticamente de un servidor a través de HTTP. Se emplea comúnmente para transmisiones de datos de flujo unidireccional en las que un servidor actualiza a un cliente periódicamente o cada vez que ocurre un evento.

JAX-RS 2.0 introdujo la noción de procesamiento asíncrono tanto para el cliente como para las API del servidor. Sin embargo, el procesamiento asíncrono por sí solo no puede cumplir todas las promesas de una arquitectura moderna sin la ayuda de I/O sin bloqueo. Si solo está disponible el bloqueo de I/O, el procesamiento asíncrono simplemente empuja el problema de un hilo al siguiente; esto es similar a pedir prestado a una

persona para pagar a otra, el problema no se resuelve realmente, solo se aplaza. Por lo tanto, es necesario admitir I/O sin bloqueo para lograr un alto rendimiento y administrar de manera eficiente recursos como subprocesos.

[“[JSR 370: Java™ API for RESTful Web Services \(JAX-RS 2.1\) Specification](#)”, 2020]

2.7.2.7. JSR-371

Model-View-Controller (MVC) es un patrón común en los marcos web, donde se usa principalmente en aplicaciones basadas en HTML. El Modelo se refiere a los datos de la aplicación, la Vista a la presentación de datos de la aplicación y el Controlador a la parte del sistema responsable de administrar la entrada y producir representaciones. Los marcos de la interfaz de usuario web se pueden clasificar como basados en acciones o basados en componentes. En un marco basado en acciones, las solicitudes HTTP se enrutan a los controladores y se convierten en acciones mediante el código de la aplicación; en un marco basado en componentes, las solicitudes HTTP se agrupan y, por lo general, cada componente las maneja de manera independiente. El marco definido por este JSR entra en la categoría basada en acciones y, por lo tanto, no pretende ser un reemplazo del JSF basado en componentes, sino simplemente un enfoque diferente para crear aplicaciones web en la plataforma Java EE.

2.7.3. *Spring Framework*

Spring Framework proporciona un modelo integral de programación y configuración para aplicaciones empresariales modernas basadas en Java, en cualquier tipo de plataforma de implementación. Un elemento clave de Spring es el soporte de infraestructura a nivel de aplicación: *Spring* se enfoca en la “plomaría” de aplicaciones empresariales para que los equipos puedan enfocarse en la lógica de negocios a nivel de aplicación, sin vínculos innecesarios con entornos de implementación específicos [“[Spring Framework](#)”, 2020], Spring tiene las siguientes características.

- Tecnologías principales: inyección de dependencia, eventos, recursos, i18n, validación, enlace de datos, conversión de tipos, SpEL, AOP.
- Acceso a datos: transacciones, soporte DAO, JDBC, ORM, Marshalling XML.
- Marcos web Spring MVC y Spring WebFlux.

- Integración: remoto, JMS, JCA, JMX, correo electrónico, tareas, programación, caché.
- Lenguajes de programación: Kotlin, Groovy, lenguajes dinámicos.

2.7.3.1. *Spring security*

Spring Security es un marco de autenticación y control de acceso potente y altamente personalizable. Es el estándar de facto para proteger las aplicaciones basadas en Spring. Spring Security es un marco que se centra en proporcionar autenticación y autorización a las aplicaciones Java. Como todos los proyectos de Spring, el verdadero poder de Spring Security se encuentra en la facilidad con la que se puede extender para cumplir con los requisitos personalizados [[“Spring Security”, 2020](#)].

2.7.4. Maven

Apache Maven es una herramienta de comprensión y gestión de proyectos de software. Basado en el concepto de un modelo de objetos de proyecto (POM), Maven puede administrar la construcción, informes y documentación de un proyecto a partir de una pieza central de información [[“Welcome to Apache Maven”, 2020](#)].

2.7.5. Mondrian

Mondrian es un motor OLAP escrito en Java. Ejecuta consultas escritas en lenguaje MDX, lee datos de una base de datos relacional (RDBMS) y presenta los resultados en un formato multidimensional a través de una API de Java. Acepta consultas analíticas y las convierte en consultas relacionales, devolviendo los datos en una forma compatible con la analítica. Pero para que Mondrian sea útil para los usuarios comerciales, necesita algún tipo de interfaz y aplicación para ejecutarlo. Varios productos utilizan Mondrian como motor de análisis para informes y análisis, como se muestra en la tabla Uno de esos productos es Pentaho, un popular servidor de análisis de negocios de código abierto que incluye Mondrian y tiene una variedad de complementos para permitir a los usuarios usar directamente las capacidades de Mondrian. Pentaho es el distribuidor de código abierto de Mondrian más grande del mundo y lo utilizan miles de organizaciones. Pentaho también es uno de los principales patrocinadores y contribuyentes de Mondrian, lo que significa que Mondrian continuará trabajando con Pentaho en el futuro previsible, y las nuevas funciones de Mondrian se integrarán rápidamente en Pentaho [D. et al., 2013].

Name	Description
Pentaho Analyzer	—Pentaho’s enterprise analysis UI that provides interactive analysis with tables and graphs.
Pentaho Reporting	—A reporting tool that creates pixel-perfect reports using Mondrian data.
Community Dashboard Framework	—A popular open source dashboard framework for creating interactive dashboards.
Saiku	—A free open source analytics tool that provides interactive analysis with tables and graphs.Saiku is available as a Pentaho plugin or a standalone product.

Cuadro 2.1: Algunos productos que usan Mondrian

Fuente: “Mondrian in Action” por D., William Back and Goodman, Nicholas and Hyde, Julian, 2013, Copyright by Manning Publications Co.

La figura 2.16 muestra cómo Mondrian encaja en la arquitectura de Pentaho. Esta vista está muy simplificada, pero contiene las partes principales de un sistema que usa

Mondrian. Los usuarios interactúan utilizando herramientas basadas en la web. Mondrian acepta consultas de estas herramientas y luego usa coincidencias de esquema lógico para generar consultas SQL. Luego, Mondrian devuelve los resultados a los clientes para formatearlos y mostrarlos a los usuarios.

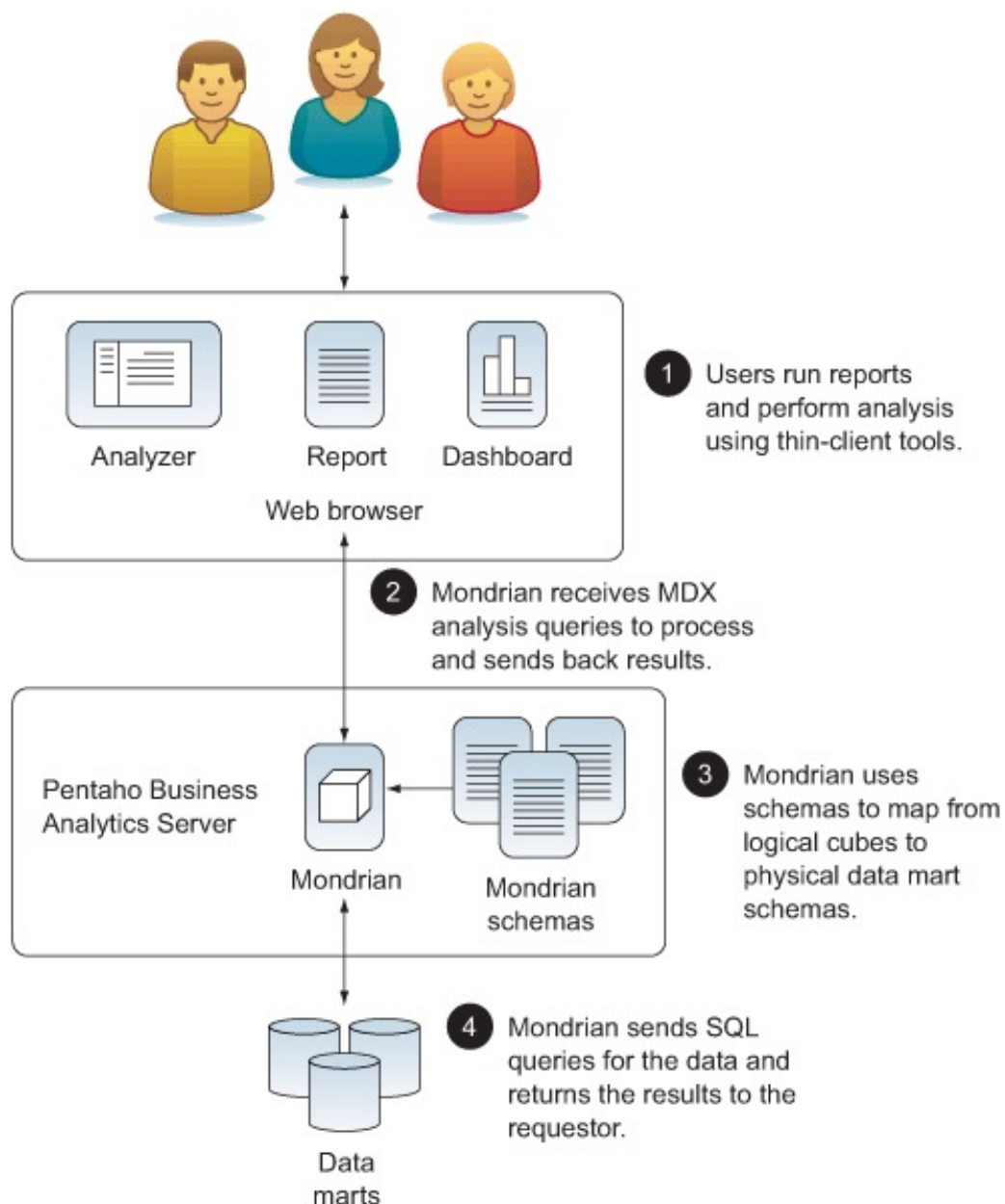


Figura 2.16: Mondrian ejecutado en Pentaho

Fuente: “Mondrian in Action” por D., William Back and Goodman, Nicholas and Hyde, Julian, 2013, Copyright by Manning Publications Co.

2.7.5.1. *Mondrian y MDX*

El lenguaje de consulta de Mondrian, MDX, proporciona una variedad de análisis avanzados basados en tiempo que se puede aprovechar inmediatamente sobre sus cubos existentes. MDX admite cosas como acumulaciones del “año hasta la fecha”, este trimestre frente al mismo trimestre del año pasado, aumento porcentual de este trimestre con respecto al trimestre anterior, y así sucesivamente. MDX significa Expresiones multidimensionales; Microsoft lo hizo popular como parte de sus servicios de análisis de SQL Server. Hasta el año 2000, no existía una forma coherente e independiente del proveedor de consultar cubos OLAP. A diferencia de las bases de datos relacionales que tenían un dialecto SQL similar entre los proveedores, los sistemas OLAP tenían API individuales y dispares. Aproximadamente al mismo tiempo, el dominio y liderazgo de mercado de Microsoft en el espacio del servidor OLAP convirtió a MDX en un estándar de facto, ya que la mayoría del mercado (ya SQL Server) ya conocía MDX. Hecho oficial, como parte de un estándar de múltiples proveedores (XML para análisis), MDX se ha convertido en el único lenguaje de consulta bien implementado para sistemas OLAP. Mondrian, como muchos otros sistemas OLAP, lo eligió por su compatibilidad y elocuencia [D. et al., 2013].

2.7.6. *Online Analytical Processing (OLAP)*

El procesamiento analítico en línea (OLAP) significa analizar grandes cantidades de datos en tiempo real. A diferencia del procesamiento de transacciones en línea (OLTP), donde las operaciones típicas leen y modifican registros individuales y pequeños, OLAP maneja datos de forma masiva y las operaciones generalmente son de solo lectura. El término “en línea” implica que, aunque se trate de grandes cantidades de datos (por lo general, muchos millones de registros que ocupan varios gigabytes), el sistema debe responder a las consultas lo suficientemente rápido como para permitir una exploración interactiva de los datos. Como veremos, eso presenta considerables desafíos técnicos.

OLAP emplea una técnica llamada Análisis multidimensional. Mientras que una base de datos relacional almacena todos los datos en forma de filas y columnas, un conjunto de datos multidimensional consta de ejes y celdas. Considere el conjunto de datos [Julian, 2006].

2.7.7. Enunciate

Enunciate es un motor de mejora para su api de servicio web java. Es simple: desarrolla su API de servicios web utilizando tecnologías estándar de Java y adjunta Enunciate a su proceso de compilación. De repente, su API de servicio web cuenta con algunas características bastante impresionantes [[“ENUNCIATE”, 2020](#)].

2.8. Tecnologías FrontEnd para la integración del proyecto Saiku Analytics

2.8.1. ECMAScript5 (ES5)

La especificación ECMAScript 5 contiene la siguiente descripción de su alcance: La quinta edición de ECMAScript (publicada como ECMA-262 5th edition) codifica las interpretaciones de facto de la especificación del lenguaje que se han vuelto comunes entre las implementaciones del navegador y agrega soporte para nuevas características que han surgido desde la publicación de la tercera edición. Tales características incluyen:

- Propiedades de acceso,
- Creación e inspección reflexiva de objetos,
- Control de programa de atributos de propiedad,
- Funciones adicionales de manipulación de matrices,
- Soporte para el formato de codificación de objetos JSON, y un modo estricto que proporciona una comprobación de errores mejorada y seguridad del programa [[“Chapter 25. New in ECMAScript 51”, 2020](#)].

2.8.2. HTML5

HTML5 es la última evolución del estándar que define HTML. El término representa dos conceptos diferentes. Es una nueva versión del lenguaje HTML, con nuevos elementos, atributos y comportamientos, y un conjunto más amplio de tecnologías que permite la construcción de sitios web y aplicaciones más diversos y poderosos. Este conjunto a veces

se llama HTML5 y amigos y, a menudo, se abrevia a HTML5. Diseñada para que la puedan utilizar todos los desarrolladores de Open Web, esta página de referencia enlaza con numerosos recursos sobre tecnologías HTML5, clasificados en varios grupos según su función [[“MDN contributors”, 2019](#)].

- **Semántica:** le permite describir con mayor precisión cuál es su contenido.
- **Conectividad:** le permite comunicarse con el servidor de formas nuevas e innovadoras.
- **Sin conexión y almacenamiento:** permite que las páginas web almacenen datos en el lado del cliente localmente y operen sin conexión de manera más eficiente.
- **Multimedia:** hacer ciudadanos de primera clase en vídeo y audio en la Web Abierta.
- **Gráficos y efectos 2D / 3D:** permiten una gama mucho más diversa de opciones de presentación.
- **Rendimiento e integración:** proporciona una mayor optimización de la velocidad y un mejor uso del hardware informático.
- **Acceso al dispositivo:** permite el uso de varios dispositivos de entrada y salida. Estilo: permitir que los autores escriban temas más sofisticados.

2.8.3. Javascript

Es un lenguaje de programación ligero, interpretado o compilado justo a tiempo con funciones de primera clase. Si bien es más conocido como el lenguaje de secuencias de comandos para páginas web, muchos entornos que no son de navegador también lo usan, como Node.js, Apache CouchDB y Adobe Acrobat. JavaScript es un lenguaje dinámico basado en prototipos, de múltiples paradigmas, de un solo subproceso, que admite estilos orientados a objetos, imperativos y declarativos (por ejemplo, programación funcional) [[“MDN contributors”, 2020b](#)].

2.8.4. Css

CSS (hojas de estilo en cascada) es un lenguaje declarativo que controla el aspecto de las páginas web en el navegador. El navegador aplica declaraciones de estilo CSS a los elementos seleccionados para mostrarlos correctamente. Una declaración de estilo contiene las propiedades y sus valores, que determinan el aspecto de una página web. CSS es una de las tres tecnologías web centrales, junto con HTML y JavaScript. CSS generalmente diseña elementos HTML, pero también se puede usar con otros lenguajes de marcado como SVG o XML [[“MDN contributors”, 2020a](#)].

2.8.5. Backbone

Backbone proporciona estructura a las aplicaciones web al proporcionar modelos con enlaces de valor clave y eventos personalizados, colecciones con una API rica de funciones enumerables, vistas con manejo de eventos declarativos y lo conecta todo a su API existente a través de una interfaz JSON RESTful [[“BackBone”, 2019](#)].

2.8.6. Node.js

Node.js es un entorno de ejecución de JavaScript de código abierto, multiplataforma. Ejecuta código JavaScript fuera de un navegador.

Node.js está diseñado sin subprocesos no significa que no pueda aprovechar múltiples núcleos en su entorno. Los procesos secundarios se pueden generar utilizando nuestra API `child-process.fork()` y están diseñados para que sea fácil comunicarse con ellos. Construido sobre esa misma interfaz está el módulo de clúster, que le permite compartir sockets entre procesos para permitir el equilibrio de carga en sus núcleos [[“About Node.js”, 2020](#)].

2.8.7. CCC-Charts

CCC es la biblioteca de gráficos de CTools, que se basa en Protovis, un conjunto de herramientas de visualización de código abierto y gratuito muy poderoso. El objetivo de CCC es proporcionar a los desarrolladores la ruta para incluir en sus cuadros de mando los tipos de gráficos básicos, sin perder el principio fundamental: Extensibilidad. Debería preferir CCC sobre otro tipo de gráficos debido a las propiedades CCC heredadas

de Protovis: los gráficos CCC se ven muy bien, son flexibles, permiten la interacción y mucho más[“CCC Playground”, 2020].

Capítulo 3

Resultados

3.1. Scrum en la práctica (1ra Iteración)

3.1.1. *Produc BackLog*

Las historias fueron trabajadas en las reuniones del *Scrum Team*, estuvieron presentes los *Developer*, el *Product Owner* y el *Scrum master* y de forma básica empezaron a ser redactadas en los tableros de nuestro *Workspace* como muestra la figura 3.1.

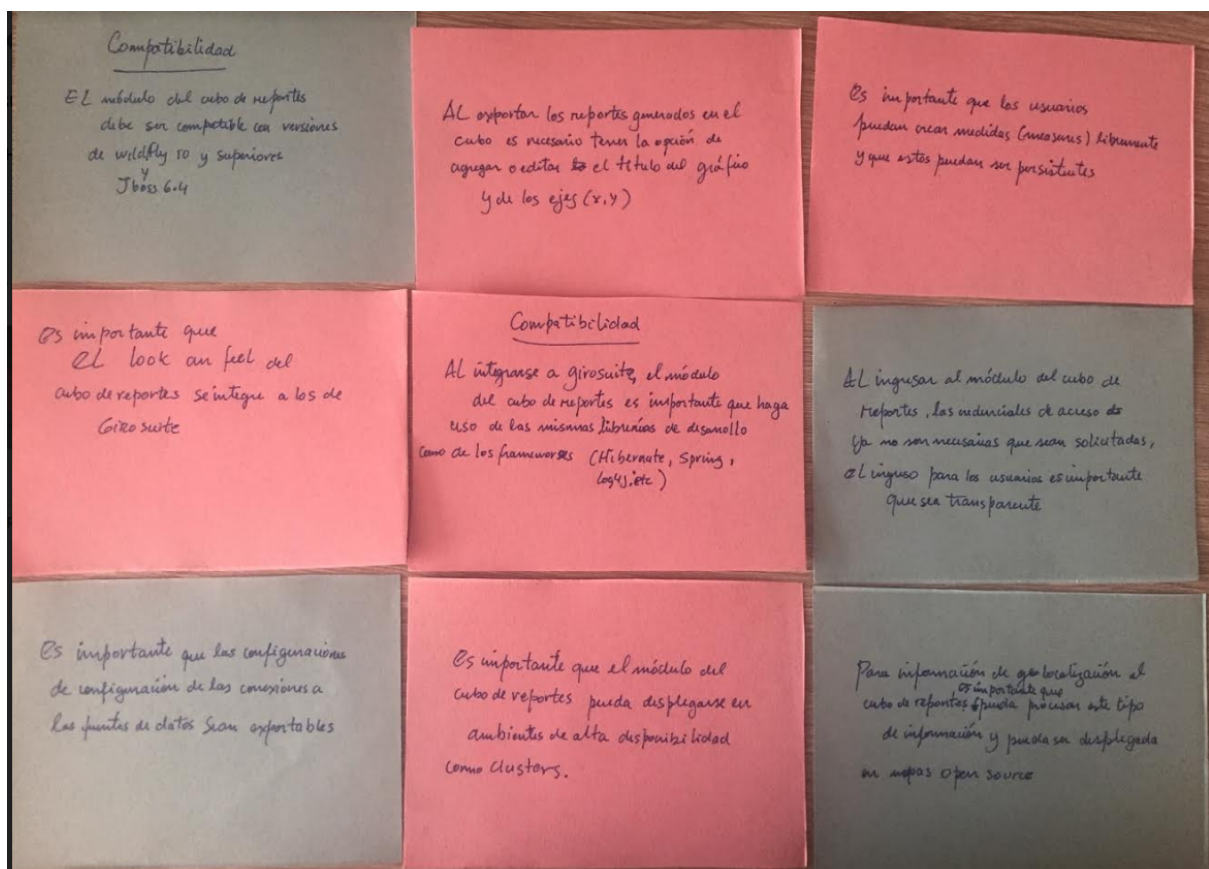


Figura 3.1: Product BackLog

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

En la primera versión del *Product Backlog* definieron 15 historias como línea base (ver figura 3.2) que trataron temas de compatibilidad, escalabilidad, integración y nuevas funcionalidades.

Backlog 15 incidencias Crear sprint ...

Compatibilidad: Verificar y corregir librerías de conflicto para las especificaciones del servidor Jbos EAP 6.4+	1.0	RUB-1	↑	-
Compatibilidad: Verificar y corregir las librerías de conflicto para las especificaciones del servidor WebLogic 1	1.0	RUB-2	↑	-
Compatibilidad: Verificar y corregir librerías de conflicto para las especificaciones del servidor WebSphere 8.0+	1.0	RUB-3	↑	-
Escalabilidad: La sesión del usuario administrada por spring security cambiar para que pueda estar centraliz	1.0	RUB-5	↑	-
Escalabilidad: Modificar la configuración para la cache de mondrian para tener persistencia con Redis o una l	1.0	RUB-6	↑	-
Integración: Integrar los componentes saiku-ui, saiku-core y saiku-war en el proceso de construcción del artefi	1.0	RUB-7	↑	-
Integración: Retirar la autenticación propia de saiku, y hacer uso de la autenticación de GiroSuite	1.0	RUB-10	↑	-
Mejora: Las configuraciones de conexión a las fuentes de datos deben ser exportables	1.1	RUB-4	↑	-
Mejora: La configuración de las fuentes de datos deben ser exportables e importables.	1.1	RUB-8	↑	-
Mejora: Tener la opción de poder crear medidas(measure) a demanda y que puedan ser persistentes	1.1	RUB-9	↑	-
Mejora: Agregar La opción de agregar y editar el título de los reportes y los ejes X,Y de los gráficos cuando :	1.1	RUB-11	↑	-
Mejora: Tener la opción de cambiar los colores de los gráficos construidos o cuando son exportados.	1.1	RUB-12	↑	-
Compatibilidad: Cambiar el look and feel para una identificación con la herramienta GiroSuite	1.1	RUB-13	↑	-
Mejora: Agregar geomondrian para el tratamientos de datos con geolocalización		RUB-14	↑	-
Mejora: Integración de la librería Echarts para ampliar el set de gráficos		RUB-15	↑	-

Figura 3.2: Product BackLog

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

Compromiso: Objetivo del Producto

“Integrar y mejorar las funcionalidades core de Saiku Analytics a la herramienta GiroSuite para darle potencia y dinámica a la generación de reportes”

3.1.2. *Sprint Planning*

Para establecer el trabajo que realizará el *Sprint*, el *Scrum Team* debatió la propuesta del *Product Owner* sobre integrar las funcionalidad core del proyecto Saiku Analytics manteniendo la compatibilidad del software GiroSuite con la infraestructura tecnológica de sus clientes; se priorizaron las siguientes historias.

En el primer *Sprint* se tomaron 7 historias, como se muestra en la tabla 3.1.

Tipo	Clave	Resumen	Estado
Historia	RUB-10	Integración: Retirar la autenticación propia de saiku, y hacer uso de la autenticación de GiroSuite	Tareas por hacer.
Historia	RUB-7	Integración: Integrar los componentes saiku-ui, saiku-core y saiku-war en el proceso de construcción del artefacto de GiroSuite.	Tareas por hacer
Historia	RUB-6	Escalabilidad: Modificar la configuración para la cache de mondrian para tener persistencia con Redis o una base de datos relacional.	Tareas por hacer
Historia	RUB-5	Escalabilidad: La session del usuario administrada por spring security cambiar para que pueda estar centralizada en la base de datos.	Tareas por hacer
Historia	RUB-3	Compatibilidad: Verificar y corregir librerías de conflicto para las especificaciones del servidor WebSphere 8.0+ y los archivos ibm-web-bnd.xml, ibm-web-ext.xml, deployment.xml y build.properties	Tareas por hacer
Historia	RUB-2	Compatibilidad: Verificar y corregir las librerías de conflicto para las especificaciones del servidor WebLogic 12c + y los archivos weblogic.xml y build.properties	Tareas por hacer
Historia	RUB-1	Compatibilidad: Verificar y corregir librerías de conflicto para las especificaciones del servidor Jbos EAP 6.4+ y los archivos jboss-deployment-structure.xml, application.xml y build.properties.	Tareas por hacer

Cuadro 3.1: Historias del primer Sprint

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

3.1.3. *Sprint BackLog*

Luego de la priorización de las historias adecuado al compromiso del *Product BackLog*, los *Developers* realizamos la estimación de cada historia haciendo uso de las *Story Points* teniendo ya en el tablero el primer Sprint BackLog.



Figura 3.3: Sprint BackLog

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

Compromiso: Objetivo del *Sprint*

“Integrar las funcionalidades core de Saiku Analytics y su compatibilidad al 100 % en la infraestructura de los clientes donde esta instala la herramienta GiroSuite”.

3.1.4. *Daily Scrum*

El *Daily Scrum* fue una constante revisión del avance del proyecto, hubo reuniones que pasaron los minutos recomendados y otras no se llevaron acabo, sin embargo estas conversaciones aportaron valor al momento tomar decisiones rápidas.

3.1.5. *Sprint Review*

Analizemos el sprint review de cada historia:

1. Integrar los componentes saiku-ui, saiku-core y saiku-webapp, saiku-service, saiku-query, saiku-repository en el proceso de construcción del artefacto de GiroSuite

- El proyecto Saiku Analytics a nivel de fuentes correctas y compilables ya están descontinuadas, y se tienen sub módulos que no son útiles al 100 % Figura 3.4.

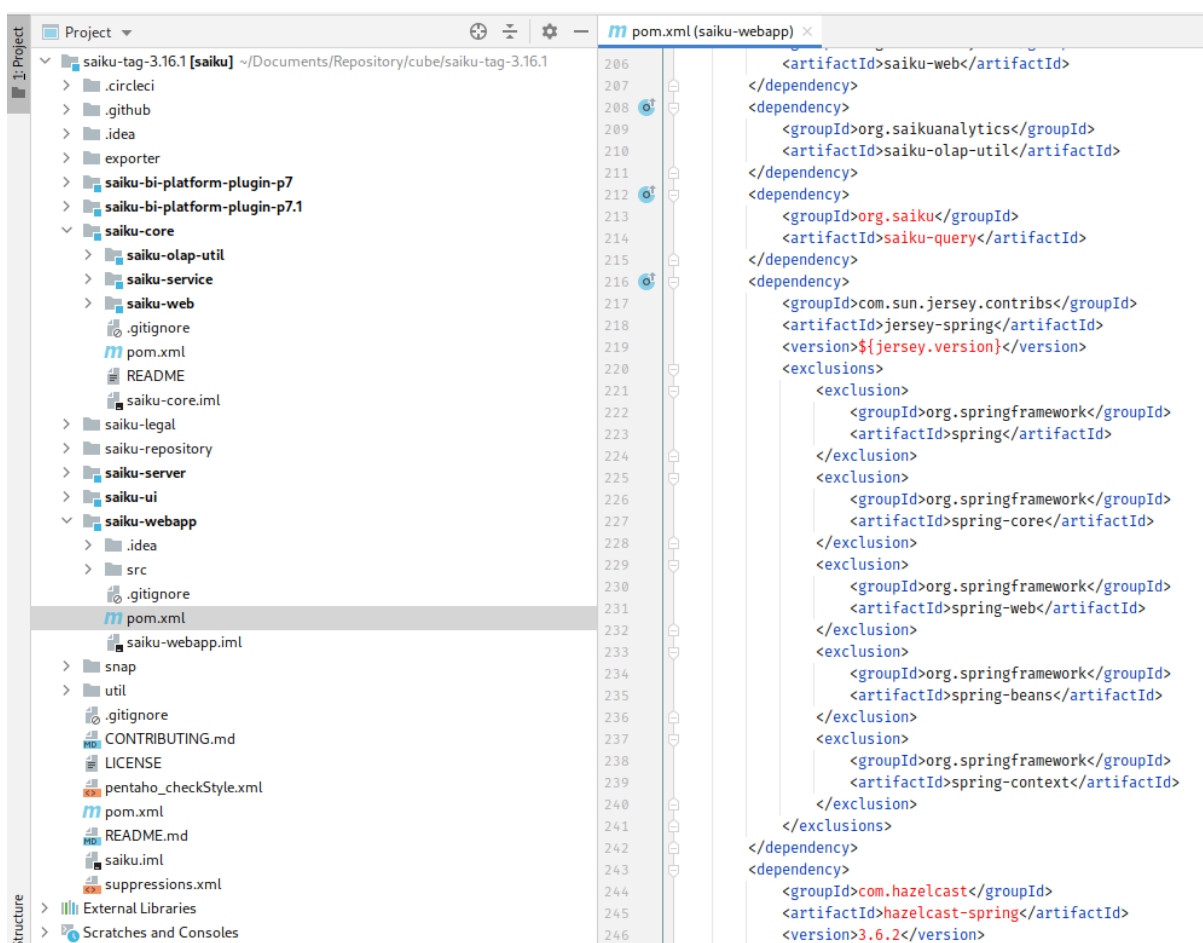


Figura 3.4: Proyecto Saiku Analytics, código fuente descontinuado

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

- El nuevo módulo resultante rubik-report tiene compatibilidad al 100 % con el producto GiroSuite.
- El nuevo módulo rubik-report si contiene el código fuente correcto de las funcionalidades mínimas del proyecto Saiku Analytics ver Figura 3.5.

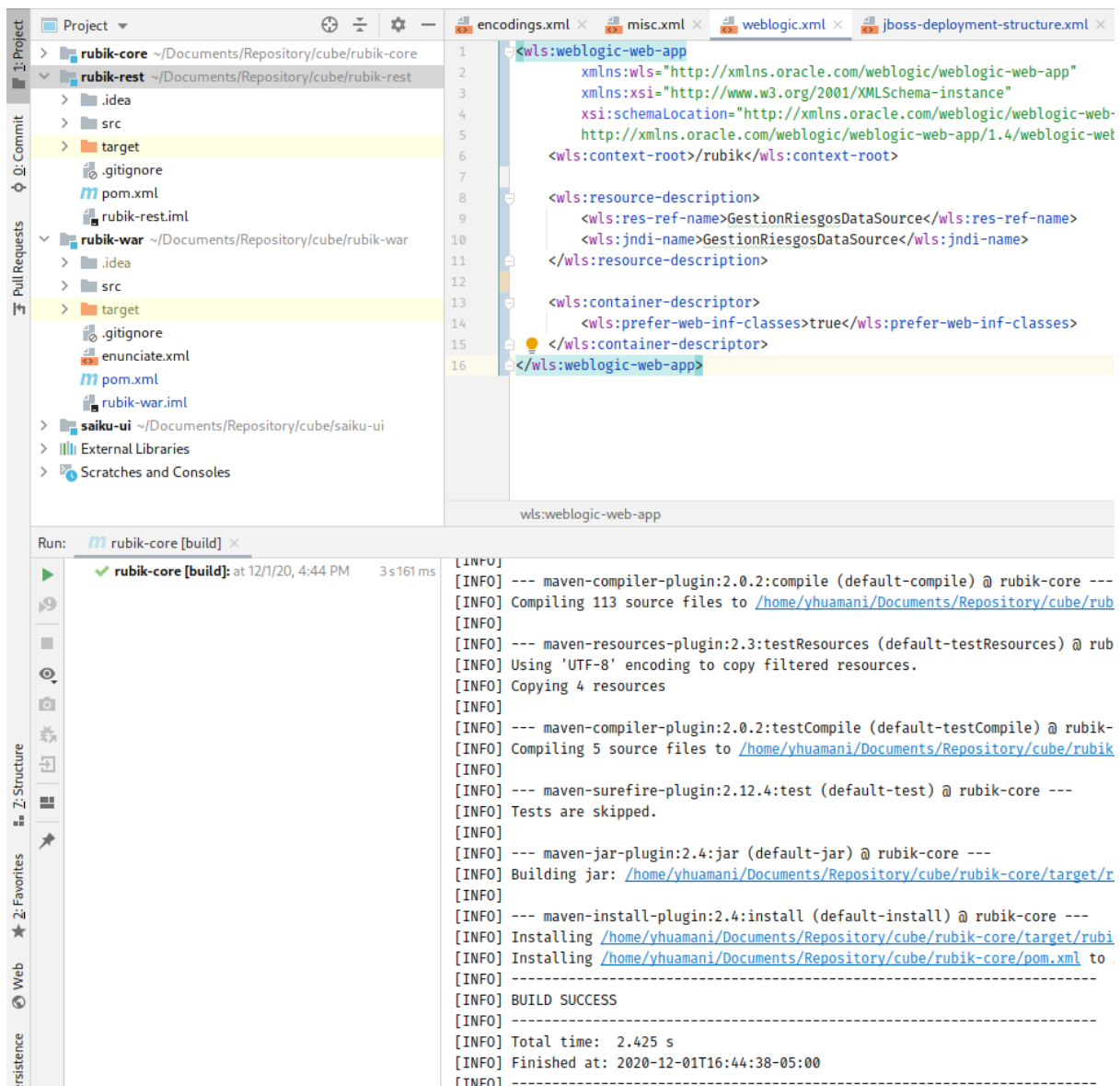


Figura 3.5: Proyecto Rubik-report, código fuente

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

- El equipo recomienda reducir el número de módulos pues 2 de ellos se encuentran ligados directamente como plugins de la Suite Pentaho Analytics.
- El equipo recomienda quitar todo aquello que no sea parte de la funcionalidad core y refactorizar el módulo rubik-query.

2. Retirar la autenticación propia de saiku, y hacer uso de la autenticación de GiroSuite

- Se eliminó la autenticación del usuario, fue integrado a la seguridad de la herramienta GiroSuite bajo el marco Spring Security.

- Con las modificaciones puede ingresar a la aplicación a través de la herramienta GiroSuite, se implementó, siendo transparente para el usuario ver Figura 3.6.

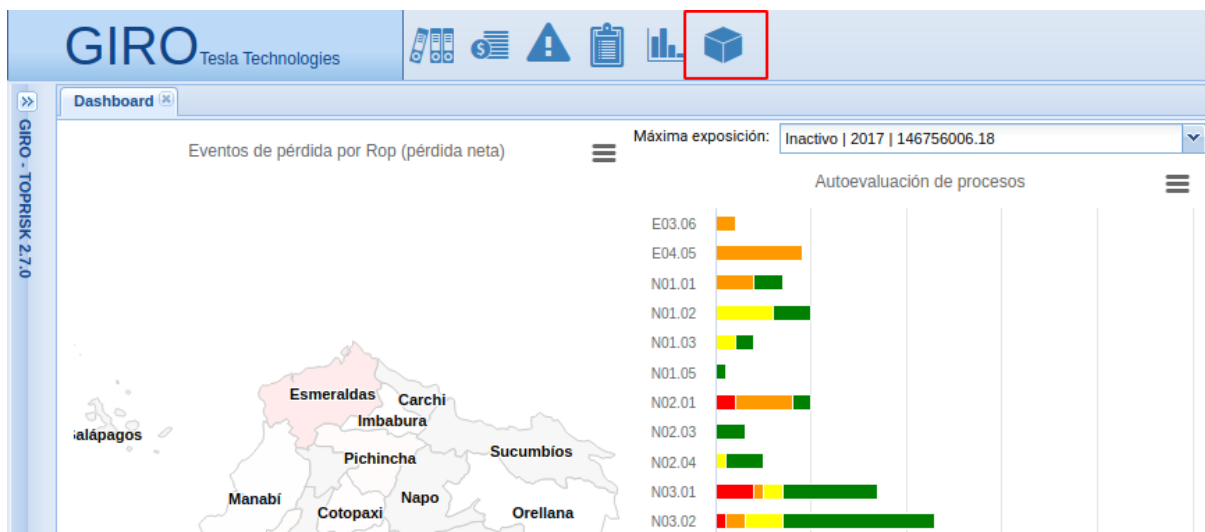


Figura 3.6: Proyecto Rubik-report integrado a GiroSuite

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

3. Verificar y corregir librerías de conflicto para las especificaciones del servidor Jbos EAP 6.4+ y loss archivos jboss-deployment-structure.xml, application.xml y build.properties

- En la verificación de los clientes se encontraron conflictos con las versiones Jboss EAP 6.4, 7.0 y sus versiones comunitarias Wildfly 10.0 Final, 10.1 y 11.0.
- Los conflictos son por configuraciones internas de los cliente como jasper reports 3.0, implementaciones del standard JPA 1.0, spring framework 2.5 y librerías de JAX-WS 1.0.
- El equipo recomienda mantenerse en la JRS-366(Jdk 8) y eso debe comunicarse al equipo de ventas para no modificar esta especificación.
- Actualmente GiroSuite ya integrado con Rubik Report puede desplegar en las versiones Jbos EAP 6.4, 7.0 y sus versiones comunitarias Wildfly 10.0 Final, 10.1 y 11.0, ver Figura 3.7.

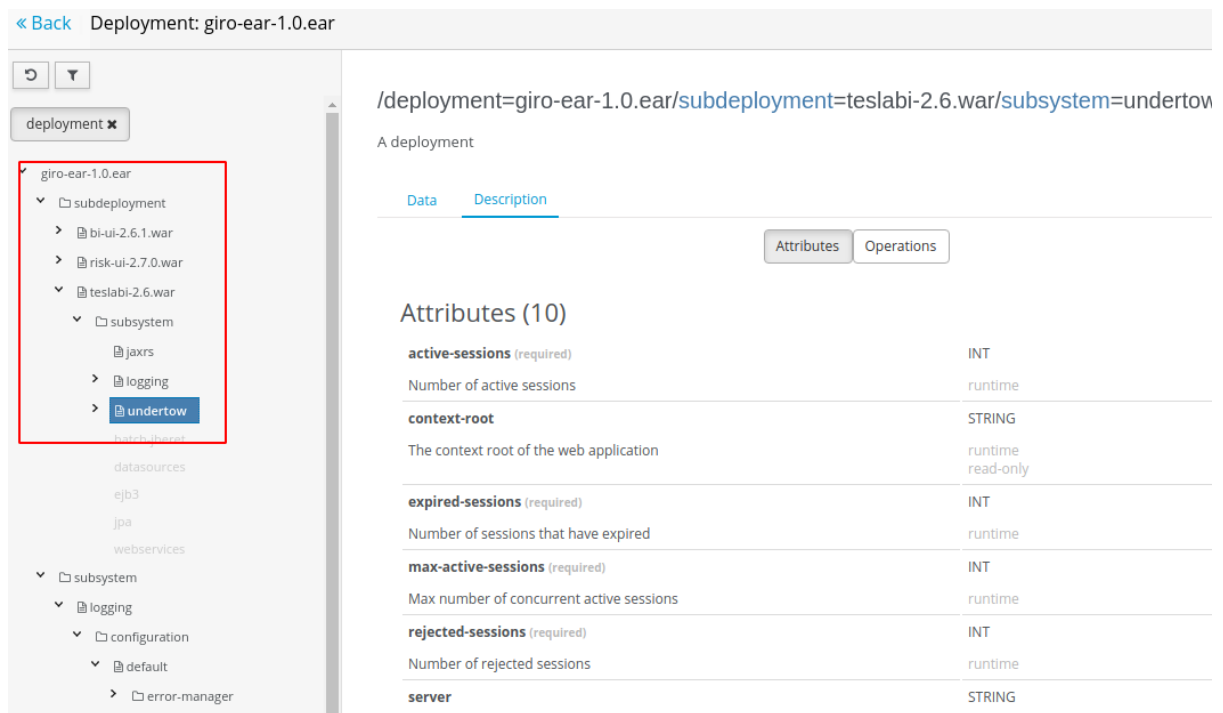


Figura 3.7: GiroSuite y Rubik-report desplegado en Wildfly 10.1

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

- El archivo application.xml fue modificado para contener los artefactos de rubik-report.
- Se debe verificar la configuración de Apache Jack Rabbit para el manejo de archivos y persistencia de los reportes pre-configurados.

4. Verificar y corregir las librerías de conflicto para las especificaciones del servidor WebLogic 12c + y los archivos weblogic.xml y build.properties

- En la verificación de los clientes se encontraron conflictos con las versiones WebLogic 12.2.1.3 y superiores.
- Los conflictos son por especificaciones de Weblogic y librerías internas fuertemente asociadas al propio servidor como jersey 1.17 y superiores.
- El equipo recomienda mantenerse en la JRS-366(Jdk 8) y eso debe comunicarse al equipo de ventas para no modificar esta especificación.
- Actualmente GiroSuite ya integrado con Rubik Report puede desplegar en las versiones de WebLogic desde 12.1.1 y superiores, ver Figura 3.8.

Change Center

View changes and restarts

Click the **Lock & Edit** button to modify, add or delete items in this domain.

Lock & Edit

Release Configuration

Domain Structure

- Resource Groups
 - Resource Group Templates
 - Machines
 - Virtual Hosts
 - Virtual Targets
 - Work Managers
 - Concurrent Templates
 - Resource Management
 - Startup and Shutdown Classes
- Deployments**
- Services
- Security Realms
- Interoperability
- Diagnostics

How do I...

- Install an enterprise application
- Configure an enterprise application
- Update (redeploy) an enterprise application
- Monitor the modules of an enterprise application
- Deploy EJB modules
- Install a Web application

System Status

Health of Running Servers as of: 8:48 PM

Failed (0)

Critical (0)

Overloaded (0)

Warning (0)

OK (15)

Summary of Deployments

Configuration Control Monitoring

This page displays the list of Java EE applications and standalone application modules installed to this domain.

You can update (redeploy) or delete installed applications and modules from the domain by selecting the checkbox next to the application or module.

To install a new application or module for deployment to targets in this domain, click **Install**.

Customize this table

Deployments

Install **Update** **Delete**

<input type="checkbox"/>	Name
<input type="checkbox"/>	AdminContratosWeb
<input type="checkbox"/>	BusIntegracionEar-1.0
<input type="checkbox"/>	CertificadoDiplomaIntegracion-ear-1.0
<input type="checkbox"/>	giro-ear-2.7.0
<input type="checkbox"/>	Modules
<input type="checkbox"/>	/bi-ui
<input type="checkbox"/>	/Giro
<input type="checkbox"/>	/teslabi
<input type="checkbox"/>	EJBs
<input type="checkbox"/>	None to display
<input type="checkbox"/>	Web Services
<input type="checkbox"/>	None to display
<input type="checkbox"/>	InscripcionBackService-1.0
<input type="checkbox"/>	LicenciaMedica-ear-1.0
<input type="checkbox"/>	MaestroPersonas-ear-1.0
<input type="checkbox"/>	MonitorNotificacionesAPP
<input type="checkbox"/>	motorB2bEar-0.0.1
<input type="checkbox"/>	mutual-circulares-servicios-1.0

Install **Update** **Delete**

Figura 3.8: GiroSuite y Rubik-report desplegado en WebLogic 12.2.1.3

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

- El archivo application.xml fue modificado para contener los artefactos de rubik-report.
- Se debe verificar la configuración de Apache Jack Rabbit para el manejo de archivos y persistencia de los reportes pre-configurados.

5. Verificar y corregir librerías de conflicto para las especificaciones del servidor WebSphere 8.0+ y los archivos ibm-web-bnd.xml, ibm-web-

ext.xml, deployment.xml y build.properties

- En la verificación de los clientes se encontraron conflictos con las versiones WebSphere 8.0, a partir de la versión 8.5 y superiores no existen problemas.
- Los conflictos son por especificaciones de WebSphere y librerías internas configuradas como jasper reports 4.6, implementaciones del standard JPA 2.0, spring framework 2.5.
- El equipo recomienda mantenerse en la JRS-366(Jdk 8) y eso debe comunicarse al equipo de ventas para no modificar esta especificación.
- Actualmente GiroSuite ya integrado con Rubik Report puede desplegar en las versiones de WebLogic desde 8.5 y superiores, ver Figura 3.9.

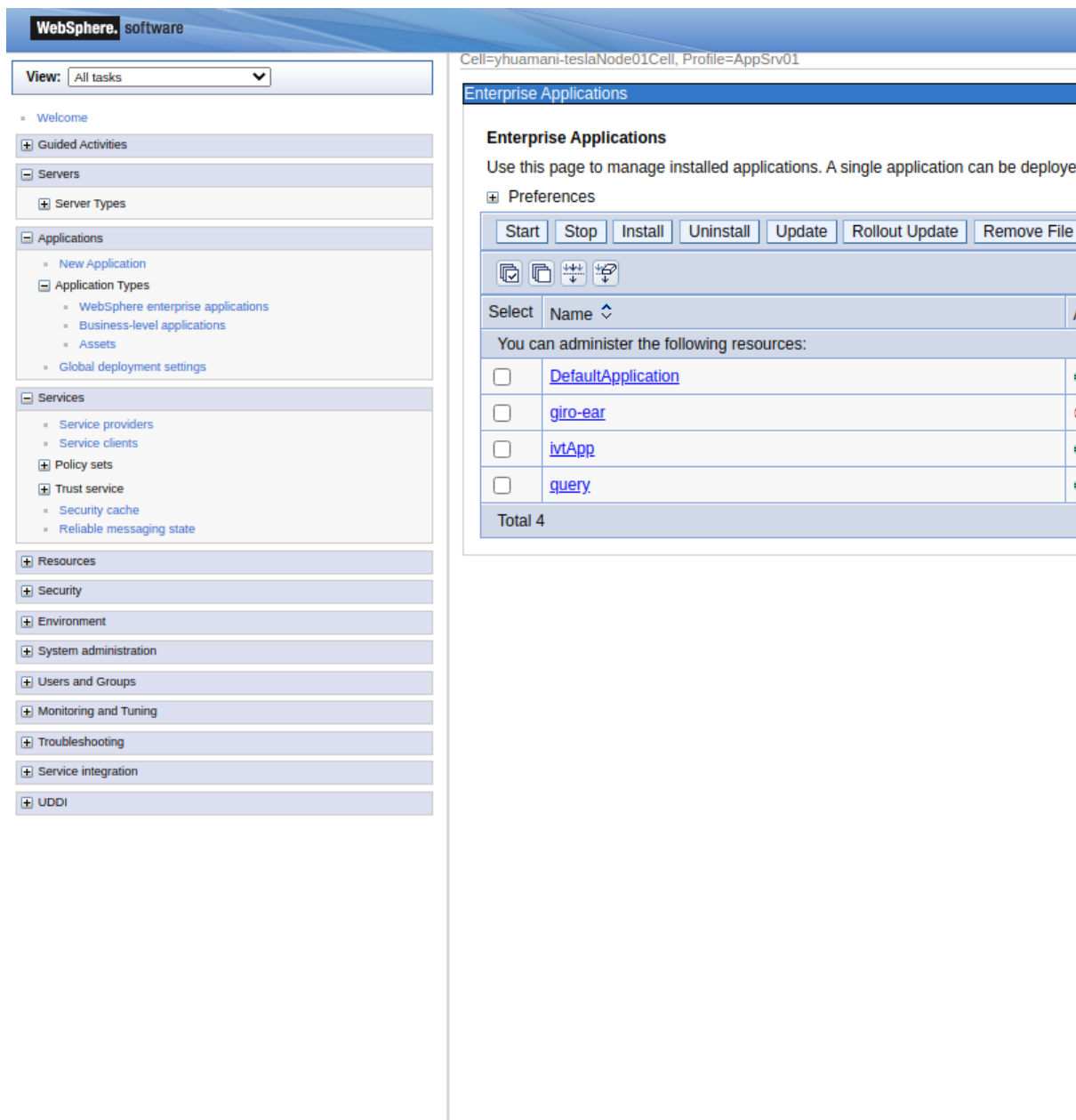


Figura 3.9: GiroSuite y Rubik-report desplegado en WebSphere 9.0

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

- Se debe verificar la configuración de Apache Jack Rabbit para el manejo de archivos y persistencia de los reportes pre-configurados.

6. La session del usuario administrada por spring security cambiar para que pueda estar centralizada en la base de datos

- El módulo Rubik Report ya es centralizado y administrado bajo spring security.
- El equipo de *Developers* recomienda evaluar el impacto de cambiar a JWT a la aplicación y su conexión con *Spring Security*.

7. Actualizar la versión de mondrian hacia la versión 4.0

- Se configuro la cache de Mondrian para mejorar el rendimiento y se cambio a Mondrian 4.0
- Se realizó la configuración de Measure Groups y Tablas agregadas para la construcción de los cubos OLAP.

Resumen de la ejecución del Sprint ver Figura 3.10.






Incidencias terminadas		Ver en el navegador de incidencias			
Clave	Resumen	Tipo de Incidencia	Prioridad	Estado	Story Points (- → 168)
RUB-1 *	Compatibilidad: Verificar y corregir librerías de conflicto para las especificaciones del servidor Jbos EAP 6.4+ y loss archivos jboss-deployment-structure.xml, application.xml y build.properties.	 Historia	↑ Medio	FINALIZADA	- → 24
RUB-2 *	Compatibilidad: Verificar y corregir las librerías de conflicto para las especificaciones del servidor WebLogic 12c + y los archivos weblogic.xml y build.properties	 Historia	↑ Medio	FINALIZADA	- → 24
RUB-3 *	Compatibilidad: Verificar y corregir librerías de conflicto para las especificaciones del servidor WebSphere 8.0+ y los archivos ibm-web-bnd.xml, ibm-web-ext.xml, deployment.xml y build.properties	 Historia	↑ Medio	FINALIZADA	- → 24
RUB-5 *	Escalabilidad: La session del usuario administrada por spring security cambiar para que pueda estar centralizada en la base de datos.	 Historia	↑ Medio	FINALIZADA	- → 16
RUB-6 *	Escalabilidad: Modificar la configuración para la cache de mondrian para tener persistencia con Redis o una base de datos relacional.	 Historia	↑ Medio	FINALIZADA	- → 40
RUB-7 *	Integración: Integrar los componentes saiku-ui, saiku-core y saiku-war en el proceso de construcción del artefacto de GiroSuite.	 Historia	↑ Medio	FINALIZADA	- → 32
RUB-10 *	Integración: Retirar la autenticación propia de saiku, y hacer uso de la autenticación de GiroSuite	 Historia	↑ Medio	FINALIZADA	- → 8

Figura 3.10: Resumen del Sprint

Fuente: Área de desarrollo de Tesla Technologies S.A.C, 2018

3.1.6. *Sprint Retrospective*

- Los plazos no deben ser necesariamente mensuales, cambiemos a seguimientos semanales, nos brindaron un mejor control para cumplir el objetivo del Sprint.
- Las nuevas características deben ser priorizadas con la experiencia de los *Developers* y apoyada con alguna metodología de priorización de historias.
- Se cumplieron todas las historias, sin embargo algunas estuvieron en riesgo de no cumplirse, nos centraremos en aquellas historias que tienen puntuación Alta.

- El equipo alcanzó el objetivo, continuemos mejorando

3.1.7. Increment

Como parte de evaluación, el *Product Owner* otorgo la aprobación con satisfacción luego de realizar sus pruebas de aceptación en la entrega del increment. En la figura 3.11 se muestra las pruebas de aceptación de un cliente y el pos de alcanzar el objetivo del producto ver figura 3.12.

Tipo	Funcionalidad	Datos de Prueba	Pre Condición	Prueba	Resultado Esperado	Responsable Mutua	Resultado Obtenido
Funcional	Login	Usuario: zbednarova Password:123456	Haber ingresado al sistema y Cubo de Reportes	Ingresar credenciales en la pantalla de inicio de sesión	Ingreso a la interfaz principal del sistema	Zuzana Bednarova	Ingreso a la interfaz principal del sistema, a través del URL http://172.30.10.191:8010/bi-ui/ al módulo de Cubo de Reportes
Funcional	Cubo contiene las dimensiones y medidas solicitadas	Usuario: zbednarova Password:123456	Haber ingresado al sistema y Cubo de Reportes	Ingresar a los cubos de reporte (1. Riesgos, 2. Controles, 3. Planes de Acción) revisando las dimensiones, aperturas y medidas solicitadas.	Los cubos de reporte (1. Riesgos, 2. Controles, 3. Planes de Acción) contienen todas las dimensiones, aperturas y medidas solicitadas.	Zuzana Bednarova	Se revisaron las dimensiones y medidas contenidas en los cubos de reporte de: 1. Riesgos, 2. Controles, 3. Planes de Acción; con algunas observaciones.
Funcional	Ejecutar consulta	Usuario: zbednarova Password:123456	Haber seleccionado las dimensiones y medidas para la consulta.	Ingresar a los cubos de reporte (1. Riesgos, 2. Controles, 3. Planes de Acción), seleccionar dimensiones y medidas, y ejecutar consulta.	Sistema muestra resultado de la consulta.	Zuzana Bednarova	Se pudo obtener el resultado de la consulta de los cubos de reporte (1. Riesgos, 2. Controles, 3. Planes de Acción).
Funcional	Descargar la consulta en archivo Excel	Usuario: zbednarova Password:123456	Haber generado correctamente la consulta.	Descargar la consulta generada en archivo Excel.	Archivo Excel descargado con datos correctos.	Zuzana Bednarova	Se pudo descargar la consulta, OK
Funcional	Descargar la consulta en archivo Excel	Usuario: zbednarova Password:123456	Haber generado correctamente la consulta.	Descargar la consulta generada en archivo formato "CSV".	Archivo formato "CSV" descargado con datos correctos.	Zuzana Bednarova	Se pudo descargar la consulta en archivo con formato csv.

Figura 3.11: GiroSuite y Rubik-report desplegado en WebLogic 12.2.1.3

Fuente: Área de soporte de Tesla Technologies S.A.C, 2018

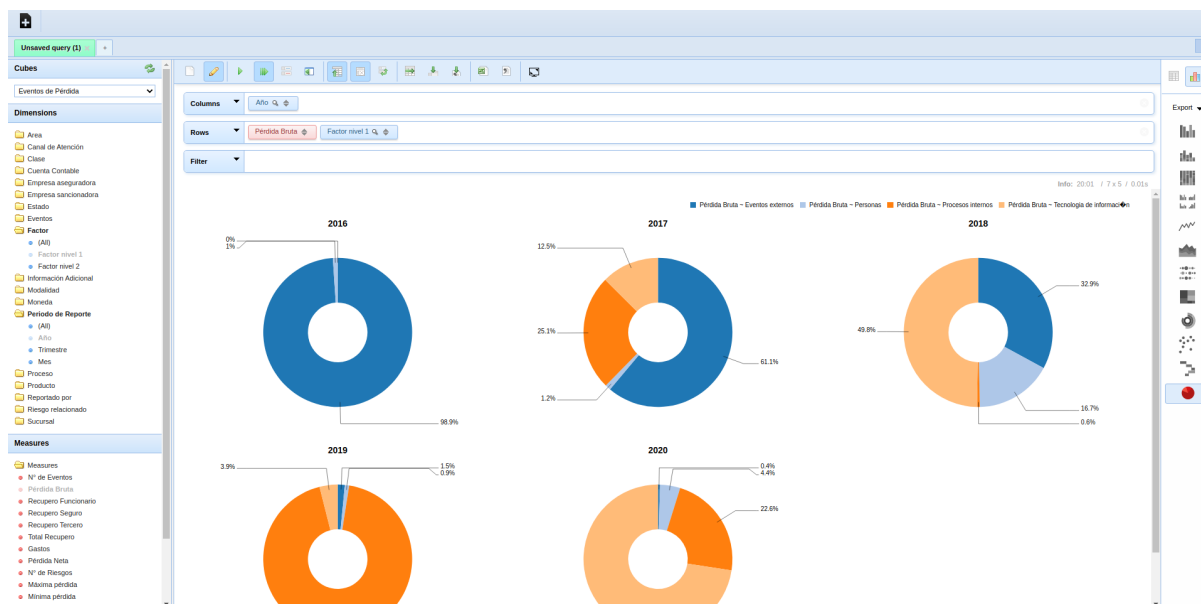


Figura 3.12: Rubik-report generando reportes dinámicos

Fuente: Área de soporte de Tesla Technologies S.A.C, 2018

Capítulo 4

Conclusiones y Recomendaciones

4.1. Conclusiones

Fue un reto para el *Scrum Team* y en particular para el *Product Owner* lograr el cumplimiento de los objetivos, sin embargo, tuvimos logros importantes y nuevas oportunidades de mejora. Líneas abajo explicamos las conclusiones y experiencias del proyecto.

- Se logro integrar las funcionalidades core del proyecto Saiku Analytics a la herramienta GiroSuite bajo el marco de trabajo Scrum conjuntamente con los valores, principios y prácticas de la Programación Extrema(XP). Las pruebas de aceptación también demostraron que los usuarios lograron generar dinámicamente sus reportes.
- Se crearon los artefactos *Product BackLog*, *Sprint BackLog* y el *Increment*. Los compromisos fueron muy importantes, durante todo el proyecto, mantuvo presente en todo el *Scrum Team* el valor aportado hacia los usuarios la integración del proyecto Saiku Analytics a la herramienta GiroSuite.
- La planificación y revisión de los Sprints aportaron un seguimiento detallado del avance del proyecto, además de contribuir a identificar oportunidades de cambio y mejora. Las nuevas oportunidades de cambio y mejora retroalimentaron al *Product BackLog* y el *Scrum Team* propició que sean atendidas en el siguiente *Sprint Planning*.
- Nuestras actividades de Sprint Review y Sprint Restrospective demostraron aportar y afinar el cumplimiento de los compromisos en las posteriores iteraciones. Se comprobó mediante la experiencia de participar en las sesiones que el tiempo recomendado, es, en su mayoría suficiente para ejecutar estas actividades.

- Se cumplió el objetivo del proyecto con el aporte fundamental de la puesta en práctica por el *Scrum Team* de los valores recomendados por el *Extreme Programming(XP)* hacemos énfasis en el respeto, la comunicación y el coraje. El *Scrum Team* resalta el desafío de la ejecución de estos valores en el proyecto.

Tener un equipo con conocimientos previos en las prácticas de la Programación Extrema(XP) apalanca el crecimiento hacia todos los *Developers*, y acelera la comprensión de los principios de la Programación Extrema(XP).

- Las pruebas unitarias en el desarrollo de las historias de usuario aportaron confianza a los *Developers* en el desarrollo de pasar una tarea “*In Process*” a “*Done*”. El ambiente de integración continua de la empresa Tesla Technologies S.A.C fue fundamental para las pruebas de integración.

Las pruebas de aceptación nos dió la confirmación y el indicador más importante del cumplimiento del compromiso del *Product Backlog* desde la primera iteración. En posteriores entregas nos confirmaron alcanzar al objetivo del producto.

4.2. Recomendaciones

Las recomendaciones del presente proyecto abordan aspectos académicos y prácticos para efectos de lograr un crecimiento colectivo e individual y la tarea de reconectar con principio de reciprocidad en nuestra cultura.

- Explorar nuevos proyectos de software open source expande la visión y alcance al ejecutar futuros productos digitales.
- No caer en el purismo de las metodologías tradicionales y las ágiles; seguir lo prescrito en la teoría limita la ejecución de proyectos en contextos particulares.
- Apostar por valores y principios e XP en un equipo de trabajo, sostienen a la continuidad de un proyecto.
- Poner en práctica los valores y principios de XP ayuda a desarrollar un estilo de trabajo basado en relaciones respetuosas y buenas en un equipo de trabajo.
- Contruir y participar en proyectos de software que contribuyan activa y positivamente al Perú 2y al mundo.

Bibliografía

- [“About Node.js”, 2020] “About Node.js” (2020). About node. <https://nodejs.org/en/about/>. Página web accedida el 30 de Noviembre del 2020.
- [“BackBone”, 2019] “BackBone” (2019). Backbone.js. <https://backbonejs.org/>. Página web accedida el 30 de Noviembre del 2020.
- [Beck and Andres, 2004] Beck, K. and Andres, C. (2004). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, USA.
- [“CCC Playground”, 2020] “CCC Playground” (2020). Ccc playground. <https://webdetails.github.io/ccc/>. Página web accedida el 30 de Noviembre del 2020.
- [Chang et al., 2007] Chang, V., Mills, H., and Newhouse, S. (2007). From open source to long-term sustainability: Review of business models and case studies. In *Proceedings of the UK e-Science All Hands Meeting 2007*. University of Edinburgh/University of Glasgow (acting through the NeSC).
- [“Chapter 25. New in ECMAScript 51”, 2020] “Chapter 25. New in ECMAScript 51” (2020). <http://speakingjs.com/es5/ch25.html>. Página web accedida el 30 de Noviembre del 2020.
- [D. et al., 2013] D., W. B., Goodman, N., and Hyde, J. (2013). *Mondrian in Action: Open Source Business Analytics*. Manning Publications Co., USA.
- [DiBona et al., 2005] DiBona, C., Stone, M., and Cooper, D. (2005). Open sources 2.0: The continuing evolution. pages 399–401.
- [“ENUNCIATE”, 2020] “ENUNCIATE” (2020). <https://enunciate.webcohesion.com/>. Página web accedida el 30 de Noviembre del 2020.
- [“Java™ Programming Language”, 2020] “Java™ Programming Language” (2020). <https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html>. Página web accedida el 30 de Noviembre del 2020.
- [“JSR 338: Java™ Persistence 2.2”, 2020] “JSR 338: Java™ Persistence 2.2” (2020). <https://jcp.org/en/jsr/detail?id=338>. Página web accedida el 30 de Noviembre del 2020.
- [“JSR 346: Contexts and Dependency Injection for Java™ EE 1.1”, 2020] “JSR 346: Contexts and Dependency Injection for Java™ EE 1.1” (2020). <https://jcp.org/en/jsr/detail?id=346>. Página web accedida el 30 de Noviembre del 2020.

- [“JSR 366: Java Platform, Enterprise Edition 8 (Java EE 8) Specification”, 2020] “JSR 366: Java Platform, Enterprise Edition 8 (Java EE 8) Specification” (2020). <https://jcp.org/en/jsr/detail?id=366>. Página web accedida el 30 de Noviembre del 2020.
- [“JSR 369: JavaTM Servlet 4.0 Specification”, 2020] “JSR 369: JavaTM Servlet 4.0 Specification” (2020). <https://jcp.org/en/jsr/detail?id=369>. Página web accedida el 30 de Noviembre del 2020.
- [“JSR 370: JavaTM API for RESTful Web Services (JAX-RS 2.1) Specification”, 2020] “JSR 370: JavaTM API for RESTful Web Services (JAX-RS 2.1) Specification” (2020). <https://jcp.org/en/jsr/detail?id=370>. Página web accedida el 30 de Noviembre del 2020.
- [“JSR 47: Logging API Specification”, 2020] “JSR 47: Logging API Specification” (2020). <https://jcp.org/en/jsr/detail?id=47>. Página web accedida el 30 de Noviembre del 2020.
- [“JSRs: Java Specification Requests”, 2020] “JSRs: Java Specification Requests” (2020). <https://jcp.org/en/jsr/overview>. Página web accedida el 30 de Noviembre del 2020.
- [Julian, 2006] Julian, H. (2006). Mondrian and olap. <https://mondrian.pentaho.com/documentation/olap.php>. Página web accedida el 30 de Noviembre del 2020.
- [“MDN contributors”, 2019] “MDN contributors” (2019). Html 5. <https://developer.mozilla.org/en-US/docs/Glossary/HTML5>. Página web accedida el 30 de Noviembre del 2020.
- [“MDN contributors”, 2020a] “MDN contributors” (2020a). Css. <https://developer.mozilla.org/en-US/docs/Glossary/CSS>. Página web accedida el 30 de Noviembre del 2020.
- [“MDN contributors”, 2020b] “MDN contributors” (2020b). Javascript (js). <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Página web accedida el 30 de Noviembre del 2020.
- [“Pentaho Products”, 2020] “Pentaho Products” (2020). <https://help.pentaho.com/Documentation/9.1/Products>. Página web accedida el 30 de Noviembre del 2020.
- [Rosen, 2005] Rosen, L. (2005). *Open source licensing*, volume 692. Prentice Hall.
- [Schwaber and Sutherland, 2020] Schwaber, K. and Sutherland, J. (2020). La guía de scrum. *Scrumguides. Org*, 1:21.
- [Sinclair, 2010] Sinclair, A. (2010). License profile: Apache license, version 2.0. *IFOSS L. Rev.*, 2:107.
- [“Spring Framework”, 2020] “Spring Framework” (2020). <https://spring.io/projects/spring-framework>. Página web accedida el 30 de Noviembre del 2020.
- [“Spring Security”, 2020] “Spring Security” (2020). <https://spring.io/projects/spring-security>. Página web accedida el 30 de Noviembre del 2020.

[“Welcome to Apache Maven”, 2020] “Welcome to Apache Maven” (2020). <https://maven.apache.org/>. Página web accedida el 30 de Noviembre del 2020.