Kanban y Scrumban orientados a Proyectos de Tecnología de la Información

Proyecto de Grado | Ingeniería en Sistemas

Javier E. Salvay | 30.540.046
jsalvay046@alumnos.iua.edu.ar | javier.salvay@gmail.com

Versión: 3.9

Esta página ha sido dejada en blanco intencionalmente

HISTÓRICO DE CAMBIOS HISTORIA DE REVISIÓN

VERSIÓN	FECHA	DESCRIPCIÓN DEL CAMBIO
Inicial	Octubre-26-2015	Primer Draft.
1.0	Noviembre-18-2015	Introducción
1.1	Enero-05-2016	Tecnología de la Información
1.2	Marzo-18-2016	Fases de Desarrollo
1.3	Marzo-25-2016	Paradigmas
1.4	Mayo-18-2016	Manifiesto Ágil
1.5	Agosto-08-2016	Kanban y Scrumban
2.0	Agosto-10-2016	AUP
2.1	Agosto-24-2016	Aplicación Normas APA
2.2	Noviembre-08-2016	Herramientas Ágiles
2.3	Enero-06-2017	Cuadro Comparativo
2.4	Marzo-11-2017	Proyecto a implementar “Wifi Social”
3.0	Junio-02-2017	Verificación final de normas APA.
3.1	Julio-16-2017	Requerimientos Iniciales
3.2	Julio-27-2017	Kanban
3.3	Agosto-17-2017	Scrumban
3.9	Agosto-18-2017	Mejoras Potenciales y Conclusión

TABLA DE CONTENIDOS

ANEXOS

CUADROS

GRÁFICOS

INTRODUCCIÓN

Se presenta el siguiente proyecto como modelo para solucionar una problemática en el desarrollo y la gestión de proyectos informáticos. Si bien dicha problemática será explayada más adelante, básicamente la presente propuesta formula una solución para los problemas comunes en el proceso de desarrollo y gestión de un proyecto de Tecnología de la Información, también llamados proyectos TI (Information Technology).

Esencialmente, esta propuesta se basa en el análisis de diferentes metodologías de trabajo Ágiles, con especial foco en Kanban y Scrumban como modelos Ágiles, según sea el campo de TI en la cual se lleve a cabo su implementación.

A su vez, cabe aclarar que esta propuesta se encuentra orientada al análisis de estas metodologías para su aplicación a lo largo de todo el ciclo de vida de un proyecto de desarrollo de software. Incluyendo la definición de requerimientos, análisis, diseño, su posterior desarrollo de código para generar los entregables, sus etapas de control de calidad de software (comúnmente llamado testing o QA), su implementación en el ambiente productivo y; su soporte y mantenimiento operativo.

A los efectos de entender la idea en su conjunto, es necesario recorrer los procedimientos, técnicas y herramientas que se disponen para el desarrollo y gestión de proyectos de Tecnología de la Información.

Dado que la implementación de estos modelos Ágiles (Kanban y Scrumban), requieren de una transición de paradigma conjunta de todo el equipo de trabajo y, por consiguiente, de una adaptación progresiva para obtener beneficios concretos, las organizaciones pueden encontrarse con diferentes barreras que impidan la implementación de las mismas. Es por eso que, asimismo, es función de esta propuesta la de llevar a cabo un análisis en detalle sobre la problemática y presentar soluciones, con sus diferentes teorías y procedimientos asociados.

El trabajo se encuentra dividido en once (12) capítulos:

1. Capítulo I - Situación Problemática: se describe la situación del problema, el objeto de estudio, la delimitación del problema; así como el objetivo general, los objetivos específicos, la propuesta a justificar y los aportes teóricos y prácticos.
2. Capítulo II - Proyectos de Tecnología de la Información: se presentan los siguientes aspectos: concepto de proyecto de Tecnología de la Información, distinción entre una metodología de administración de proyectos y una metodología de desarrollo de software, entre otros.
3. Capítulo III - Metodologías de Desarrollo para Proyectos TI: presenta un marco teórico sobre las metodologías de desarrollo para proyectos TI, describiendo la historia de las metodologías de desarrollo de Software, las metodologías tradicionales y sus ventajas y desventajas.
4. Capítulo IV - Metodologías Ágiles: se presenta en este capítulo una amplia descripción de las metodologías Ágiles, resaltando Kanban y Scrumban, ya que constituyen el centro de esta investigación. También, se realiza una comparación de las metodologías Ágiles con las metodologías tradicionales.
5. Capítulo V - Herramientas para la Gestión de Metodologías Ágiles: se presentan diferentes herramientas para la gestión de dichas metodologías Ágiles, con foco predominante en la herramienta JIRA de la empresa Atlassian. También, se realiza un cuadro comparativo analizando cada una de las herramientas más utilizadas con sus ventajas y desventajas.
6. Capítulo VI - Proyecto Práctico Real: se presenta el proyecto real en el cual se basará el trabajo y experiencia práctica.
7. Capítulo VII - Ejemplificación de Implementación con Enfoque Kanban: se realiza la ejemplificación de Kanban dentro de un proyecto de IT, utilizando la herramienta JIRA.

8. Capítulo VIII - Ejemplificación de Implementación con Enfoque Scrumban: utilizando el mismo proyecto aplicado en el capítulo anterior, se realiza la ejemplificación de Scrumban utilizando la herramienta JIRA.
9. Capítulo IX - Mejoras Potenciales: se desarrolla un análisis en el que se identifican mejoras potenciales para ambos enfoques de la metodología ágil.
10. Capítulo X - Conclusiones: se concluye dicho documento con una conclusión final y general sobre el proyecto y la aplicación de ambas orientaciones ágiles. También, se incluyen referencias bibliográficas, glosario y acrónimos.

CAPÍTULO I

SITUACIÓN PROBLEMÁTICA

Hasta hace algunas décadas, un proyecto de Sistemas de Información (o de Tecnología de la Información o TI) mal desarrollado tenía un impacto menor en las organizaciones y en los negocios. Hoy en día, muchos proyectos de TI impactan directamente en la supervivencia de un número de organizaciones que sigue en aumento. Una demora puede significar la pérdida de una posición competitiva, o hasta posponer por meses el lanzamiento de un producto o servicio.

Proyectos de TI con una gestión equivocada, inadecuada o sin un proceso de desarrollo coherente, deterioran las relaciones con los clientes y con los proveedores, provocan pérdida de ingresos e incluso pueden causar penalidades por incumplimiento de regulaciones contractuales, legales o tributarias. Existen algunos proyectos también, cuyo destino final termina siendo su cancelación antes del tiempo previsto, e incluso antes de ser completados. Esto genera que muchas veces se pierdan todos los recursos invertidos, provocando un daño grave a la reputación de la Gerencia de Sistemas o en muchos otros casos, al prestigio de la organización.

Según Brooks (1987) “No existe ningún desarrollo, ya sea en tecnologías como en management, que por sí solo prometa siquiera una mejora de un orden de magnitud en la próxima década en lo referido a productividad, confiabilidad, simplicidad.”

En base a lo expresado por Brooks, numerosos estudios y la experiencia misma a lo largo de estas décadas, han demostrado que las principales causas de fracaso no se encuentran en los aspectos tecnológicos, sino en los factores en el desarrollo y la gestión de proyectos de TI. Sus principales causas se deben a que son demasiado grandes, excesivamente ambiciosos, disponen de un uso inadecuado de recursos, porque no toman en cuenta los aspectos de su entorno, o simplemente porque carecen de un proceso de desarrollo y organización de recursos que facilitan la comunicación y participación de sus partes.

Pese a esta problemática, en la actualidad existen diferentes metodologías de trabajo para el desarrollo y gestión de proyectos de TI. Dentro de todas estas metodologías, las que han adquirido un papel importante en el último tiempo son las metodologías ágiles. Las mismas, surgen como una alternativa a las metodologías tradicionales. Son una forma de reacción a ellas, principalmente debido al hecho de que las metodologías tradicionales no han podido suplir las necesidades (y porque no llamarlos problemas en algunos casos), que persiguen al desarrollo de proyectos de software desde sus comienzos.

Problema

Tal como se comentaba anteriormente, existen diferentes problemas referidos al desarrollo de un proyecto de TI. Algunos de ellos se mencionan a continuación:

1. El uso ineficiente de recursos como así también, de sus tiempos y costos estimados.
2. Carencia de un proceso de gestión y administración para el desarrollo del proyecto.
3. Ineficiencias identificadas dentro del mismo como lo pueden ser: la falta de herramientas orientada a coordinar y mantener una comunicación consistente y constante en el equipo de trabajo.

Estos problemas pueden impactar considerablemente en la entrega del producto y, por consiguiente, puede llegar a causar el no cumpliendo del objetivo propuesto.

De esta forma se puede decir que el problema a solucionar es la ineficiencia en la gestión y administración de recursos, tiempos, costos a lo largo del proceso de desarrollo de un proyecto TI.

Objeto de estudio

El objeto de estudio para este trabajo es el análisis de los enfoques Kanban y Scrumban orientados a los proyectos de Tecnología de Información, como así también, su aplicación en las organizaciones a partir de marcos de referencia y casos prácticos para los diferentes tipos de proyectos TI.

De dicha manera, se buscará evidenciar que es posible llevar a cabo una correcta gestión y administración en un proyecto TI, mediante diferentes técnicas y herramientas que conforman un entorno de trabajo flexible y al mismo tiempo robusto.

Campo de acción

Teniendo la posibilidad de aplicar marcos de referencia de TI en cuanto a metodologías ágiles, el campo de acción se desarrollará sobre un proyecto de software donde se cubrirán las diferentes etapas del ciclo de vida. Desde sus requerimientos, análisis, diseño, desarrollo y pruebas, hasta su implementación y mantenimiento en el entorno productivo.

A su vez, se describirán los requerimientos, procesos y documentos a utilizar durante la gestión y desarrollo del proyecto, los cuales pueden ser útiles como guía y/o referencia para el desarrollo de la herramienta en futuras implementaciones.

Por último, cabe aclarar que, de todas las alternativas de enfoques ágiles existentes, se seleccionarán 2 de ellos (Kanban y Scrumban), a los efectos de desarrollar diversas pruebas para el presente trabajo y demostrar el valor agregado que pueden brindar las mismas en las organizaciones.

Objetivos

A continuación, se detallan los objetivos del proyecto, describiendo brevemente los resultados esperados a su conclusión.

Objetivo general

Implementar las metodologías ágiles, sus marcos de referencia y los beneficios de las mismas, en lo que respecta a otras metodologías tradicionales en el mercado.

Objetivos específicos

Se describen los siguientes objetivos específicos de este proyecto:

- Presentar el funcionamiento de los enfoques ágiles: Kanban y Scrumban, conjuntamente con sus beneficios a corto y largo plazo.
- Proveer información sobre la implementación de las orientaciones ágiles: Kanban y Scrumban, en cada una de las etapas del ciclo de vida del proyecto.
- Evaluar en detalle las diferentes alternativas a estos dos enfoques ágiles en particular.
- Implementar ambos enfoques en herramientas online para efectuar pruebas.

Propuesta a justificar

La justificación de esta propuesta está basada en la implementación/aplicación de las orientaciones ágiles: Kanban y Scrumban para demostrar la factibilidad de utilizar los mismos como marco de referencia en TI; de tal forma que facilite y mejore la gestión, administración y comunicación entre los recursos, así también como los tiempos y costos dentro de un proyecto TI.

Delimitación del proyecto

El proyecto se limitará a la investigación de los enfoques ágiles Kanban y Scrumban, aplicados a proyectos de Tecnología de la Información, contemplando las diferentes etapas del ciclo de vida del mismo: (a) análisis, (b) diseño, (c) su posterior desarrollo de código para generar los entregables, (d) sus etapas de control de calidad

de software (comúnmente llamado de testing o QA), (e) su implementación en el ambiente productivo y (f) su soporte y mantenimiento operativo.

El marco práctico del mismo se realizará en una herramienta online que se detallarán más adelante.

Aporte teórico

El proyecto presenta diversos tipos de aportes teóricos. El principal de ellos es la información clara, dinámica que será brindada con respecto a lo que se está trabajando, permitiendo el control y la toma de decisión en sus diferentes niveles dentro del proyecto y/o la organización.

Otro aporte presentado es la forma de trabajo, la cual no se limita al equipo de trabajo del proveedor, sino también a la participación del cliente. La cual es muy importante y necesaria para que se obtengan los resultados que éste último persigue.

Este proyecto también servirá como un documento instructivo con explicaciones detalladas que permitirá a los responsables del área de Sistemas, utilizarlo como guía.

Finalmente, se incluye dentro de los aportes teóricos un estudio de la viabilidad económica de la implementación de dicha propuesta.

Aporte práctico

Los beneficios que el proyecto brindará son:

- Ayudar a las organizaciones, desde un punto de vista práctico de gestión, a llevar a cabo sus proyectos dentro del área de Tecnología de la Información (TI).
- Ayudar, guiar y proveer información en tiempo real a los responsables y/o gerentes que se encuentren a cargo del proyecto, de forma de asistirlos en la toma de decisiones.

- Mejora concreta en los flujos comunicacionales dentro del proyecto, entre miembros de equipo o colegas y sus superiores.
- Evaluación de la inversión a realizar, con los costos correspondientes de la implementación del enfoque ágil seleccionado.
- Reducción considerable de los tiempos empleados para realizar auditorías internas.
- Reducción de los tiempos para proveer reportes sobre los estados actuales del proyecto, como así también su planificación futura.
- Mejora en el proceso de estimaciones de trabajo y relevamiento de recursos disponibles.

CAPÍTULO II

PROYECTOS DE TECNOLOGÍA DE LA INFORMACIÓN

A lo largo de este capítulo se abordarán temas relacionados con las metodologías de Gestión de Proyectos de Tecnologías de la Información (también denominado Administración de Proyectos), sus etapas, y otras consideraciones para comprender el objetivo de esta disciplina. También se presentarán las fases de una metodología de desarrollo de Software. Finalmente, se presentará una comparación de ambas metodologías.

Concepto de Proyecto de Tecnología de la Información (PTI)

Un proyecto según el IESA (2005) es:

> Cualquier trabajo finito, complejo y no repetitivo sea de diseño, construcción u otro, el cual contiene un conjunto de actividades formalmente organizadas a las cuales se les han establecido fechas de inicio y terminación, y consumen recursos humanos, materiales, equipos, tiempo y dinero. (p.2).

Ahora bien, las Tecnologías de la Información (TI) se definen según Arias (2009) como: "la plataforma e infraestructura tecnológica que mantiene las operaciones de los servicios de información de una compañía." En este sentido, señala este autor que la Tecnología incluye: (a) la planificación estratégica de sistemas de información, (b) la implantación de tecnologías y paquetes, (c) distribución de datos y procesos, (d) auditoría de sistemas, (e) diseño de modelos de datos, (f) ingeniería de información con tecnología y (g) herramientas y técnicas de desarrollo acelerado. (p. 13).

Como se puede apreciar en la definición anterior, las Tecnologías de la Información abarcan una cantidad amplia de trabajos, que no se limitan exclusivamente al desarrollo de software.

Por lo antes expuesto, se define a un Proyecto de Tecnología de la Información (PTI) como cualquier trabajo, basado en la plataforma e infraestructura tecnológica, que mantiene las operaciones de los servicios de información de una compañía, el cual contiene un conjunto de actividades formalmente organizadas, a las cuales se les han establecido fechas de inicio y terminación, y consumen recursos humanos, materiales, equipos, tiempo y dinero.

Metodologías de Gestión de Proyectos de TI

La Gestión de Proyectos se define según el IESA (2005) como:

> La aplicación de conocimientos, habilidades, técnicas y herramientas con la intención de satisfacer o superar los requerimientos del dueño o del inversionista. A su vez, es la encargada de visualizar y establecer prioridades del proyecto, ubicarlas en un espacio de tiempo determinado y asignar el tipo y número de recursos necesarios para satisfacer esas prioridades. (p. 5).

En el caso de los proyectos de TI, los recursos a gestionar incluirán Software, Hardware y competencia de personas, entre otros.

En cuanto a las metodologías de gestión de proyectos, se definirán brevemente tres de las cuales han tenido mayor impacto en el mercado: PRINCE, CCPM y PMBOK. Siendo esta última, un estándar muy importante para la gestión de proyectos en general.

Proyectos en Ambientes Controlados (PRINCE, por sus siglas en inglés: PRojects IN Controlled Environments)

La metodología PRINCE (PRojects IN Controlled Environments), surge en 1989, como una publicación de La Agencia Central de Informática y Telecomunicaciones del Gobierno del Reino Unido, presentándose como un estándar para todos los proyectos de sistemas de información del gobierno, con la finalidad de reducir las excedencias de costos y tiempo. Posteriormente en 1996, se edita la segunda

revisión para considerar la aplicación de la metodología a cualquier tipo de proyecto. La versión más reciente de dicha metodología fue publicada en el año 2009, con el fin de proveer a los Gerentes de Proyectos (Project Managers) un conjunto de herramientas mejor determinadas a cumplir los proyectos en tiempo, presupuesto y con la calidad apropiada. (Haughey, 2012).

Dirección de Proyectos con Cadena Crítica (CCPM, por sus siglas en inglés: Critical Chain Project Management)

La metodología CCPM (Dirección de Proyectos con Cadena) fue desarrollada por el Dr. Eliyahu M. Goldratt en 1997, se basa en métodos y algoritmos extraídos de su Teoría de las Restricciones presentada en su novela "La Meta" en 1984. Esta metodología se basa en la gestión de "buffers" o amortiguadores, que se refieren a los tiempos de protección y/o tolerancias que se asignan a una determinada tarea. A su vez, se aborda un tema muy importante que es la limitación de recursos compartidos en multiproyectos y los problemas asociados a esta práctica.

Guía de los Fundamentos para la Dirección de Proyectos (PMBOK, por sus siglas en inglés: Project Management Body of Knowledge)

Finalmente, vale la pena destacar en este apartado, a PMBOK: en 1987, se publica por primera vez la Guía de los Fundamentos para la Dirección de Proyectos (PMBOK) por el PMI. El PMBOK surge inicialmente como un reporte o intento por documentar y homologar las prácticas y su información, en relación a la administración de proyectos aceptados. En 1996, es publicada una revisión de la primera publicación. En el año 2000, se presenta la segunda edición, que demoró un total de 13 años y que denota cambios en todos sus contenidos. Luego, en 2004 se realiza la publicación de la tercera edición. Ya para el año 2008, se presenta la cuarta edición y en el año 2013 se presenta la quinta y última versión disponible. Para el año 2018, se estima que será publicada la sexta versión del PMBOK, que contempla numerosos capítulos que han

sido reescritos y se encuentran alineados a los tiempos actuales y la intensa velocidad de cambio en los mercados.

Este cuerpo de conocimientos es referencia primordial para todos los vinculados al mundo de los proyectos actualmente y se ha convertido en un estándar global para la industria. (Haughey, 2012).

PMBOK se encuentra orientado a una gestión predictiva de los proyectos, esto implica que la necesidad/solución, el alcance y la planificación (p.ej. costo y duración de cada una de las tareas a realizar), se establece en las fases iniciales.

Parte de las herramientas que ofrece PMBOK pueden ser utilizadas en combinación con metodologías ágiles y flexibles.

Las características de un proyecto según el PMBOK son: (a) un proyecto intenta dar solución a un problema (cubrir una necesidad); (b) es temporal; (c) es único en el tiempo y no repetible bajo las mismas circunstancias; (d) conlleva incertidumbre y (e) consume recursos: tiempo, dinero, materiales y trabajo.

Para cumplir con todas las fases de un proyecto, se deben ejecutar los siguientes procesos generales:

1. Identificar el problema o la oportunidad.
2. Identificar y definir la solución idónea.
3. Identificar las tareas y los recursos necesarios.
4. Preparar el calendario y la obtención de recursos.
5. Estimar el costo del proyecto y preparar un presupuesto acorde.
6. Analizar los riesgos y establecer relaciones con los stakeholders (toda persona que tenga un interés directo o indirecto en el proyecto): Gestión del riesgo periódico.
7. Mantener el control y la comunicación en el nivel adecuado durante la ejecución: Reuniones periódicas para detectar y comunicar riesgo, problemas o desvíos.
8. Gestionar un cierre satisfactorio: (a) Punch list: listado de tareas para poder acabar el proyecto y (b) los miembros del equipo tienden a dispersarse dado que el proyecto se encuentra casi cerrado.

No obstante, un proyecto puede ser visto desde otras perspectivas, como, por ejemplo, desde el punto de vista de las **relaciones interpersonales**:

1. Motivar al equipo: Crear el clima adecuado.
2. Invertir tiempo en explicar cómo cada función contribuye al proyecto: (a) invertir tiempo en las reuniones para destacar contribuciones positivas de los miembros; (b) confiar en el trabajo delegado; (c) asignar objetivos a los individuos y permitir que ellos escojan el camino; (d) reconocer los esfuerzos que van más allá de lo solicitado y (e) predicar con el ejemplo.
3. Gestionar la diversidad: (a) identificar posibles objetivos personales para minimizarlos o convertirlos en objetivos grupales y (b) buscar la cohesión del grupo (armonizar costumbres, culturas, etc.).

En definitiva, el grado de madurez de la organización y los procedimientos internos establecidos pueden contribuir al éxito o fracaso del proyecto:

1. Si la organización trabaja habitualmente en proyectos, se disponen de pautas ya definidas.
2. Vías de comunicación formales: si son muy rígidas pueden entorpecer el trabajo.
3. Vías de comunicación informales (amistades, conocidos, etc.): si son muy frecuentes puede producir desinformación.

Finalmente, PMBOK establece que para poder considerar que un proyecto ha sido exitoso, se deben cumplir las siguientes expectativas:

- Nivel I. Alcanzar los objetivos del proyecto.
- Nivel II. Eficiencia del proyecto.
    - Nivel de interrupción del trabajo del cliente.
    - Eficiencia en el uso de los recursos.
    - Crecimiento del número de miembros del equipo.
    - Gestión de conflictos.
- Nivel III. Utilidad para el usuario/cliente final.
    - ¿Ha sido solucionado el problema inicial?
    - ¿Se han incrementado los beneficios o se ha producido ahorro real?
    - ¿El usuario se encuentra actualmente usando el producto?
- Nivel IV. Mejora organizacional: Aprender sobre la experiencia.

Fases de la Gestión de Proyectos

En la perspectiva tradicional, es posible distinguir cinco (5) componentes en el desarrollo de un proyecto (4 fases, más el control) (Project Management, 2005). Las fases se detallan a continuación:

1. Iniciación de proyecto: en esta fase se identifica la necesidad y se evalúa la posibilidad de llevar a cabo el proyecto, para ello es necesario: (a) entender el problema o la oportunidad, (b) identificar la solución más óptima, (c) desarrollar la solución y la elaboración de un plan y (d) lanzamiento del proyecto.
2. Planificación de proyecto: se desarrolla una solución en un detalle mayor, definiendo las tareas que se llevarán a cabo, el calendario y la estimación de costos en tiempo y dinero.
3. Producción de proyecto o ejecución: se lleva a cabo la monitorización del proyecto y se realizan los ajustes necesarios a la planificación.
4. Finalización de proyecto o cierre: Se comprueba si el proyecto satisface la necesidad a cubrir. Es importante validar que los criterios de finalización establecidos al inicio del proyecto se encuentran cubiertos.

Por otra parte, la supervisión y control del proyecto consiste en detectar desviaciones respecto a la planificación inicial y realizar estimaciones sobre cuál será la desviación al final del proyecto.

Por regla general, en cualquier proyecto se deben controlar las siguientes dimensiones: (a) calendario, (b) costos, (c) funcionalidades y (d) calidad.

Con el objetivo de ejercer las tareas de control se requiere recopilar, de forma periódica, la siguiente información. (ver gráfico 1).

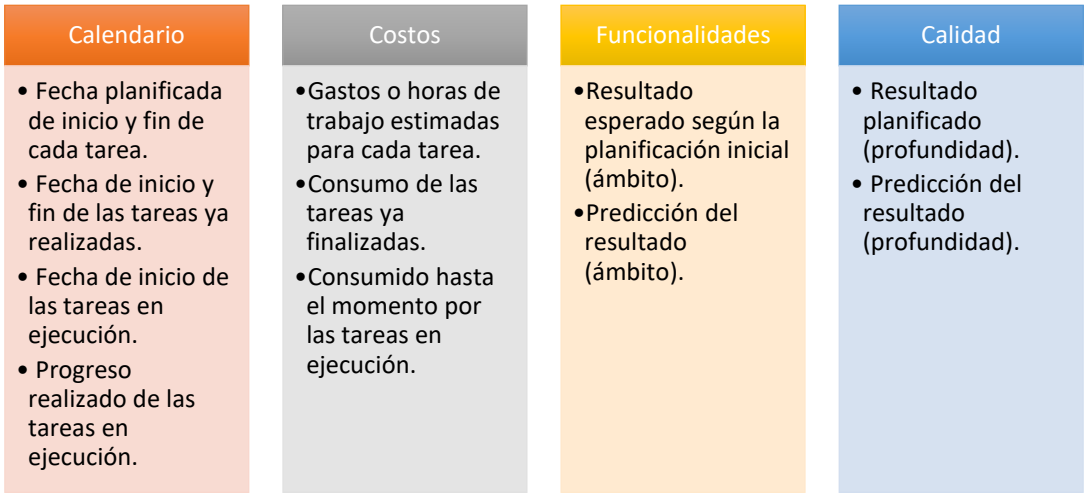


Gráfico 1. **Información para el control de proyectos.**

Finalmente, cabe destacar que la documentación resultante de la planificación y del control, puede servir como base histórica para futuros proyectos con características similares.

Fases del Desarrollo de Software

Las fases del desarrollo de Software dependen de la metodología de desarrollo que se utilice. Se puede tomar como ejemplo una metodología estructurada según Senn (1995), compuesta por 6 fases (ver gráfico 2), que se detallan seguidamente.

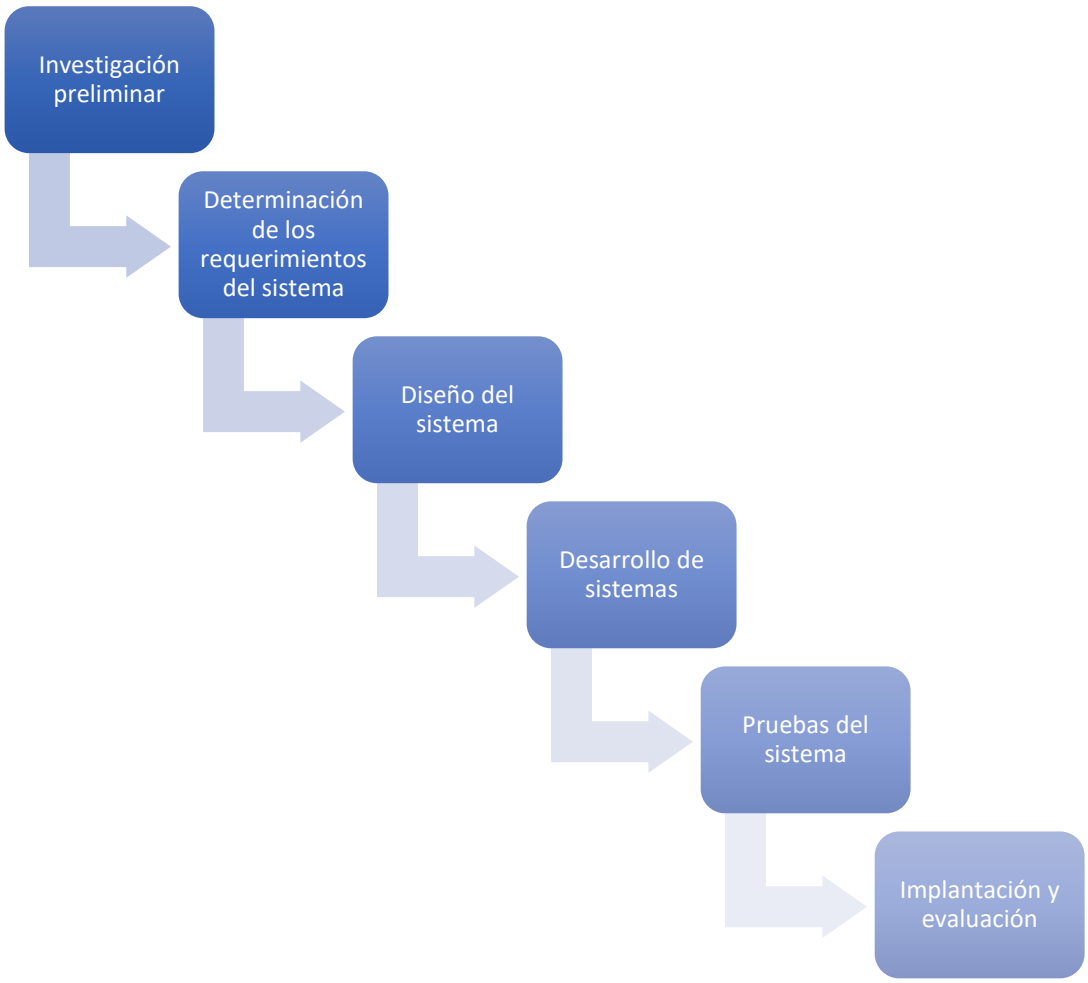


Gráfico 2. **Fases del desarrollo de Software**

Las fases del desarrollo de Software según Senn (1995) se detallan seguidamente.

1. Investigación preliminar: esta fase comienza con la formulación de una solicitud, ya sea por parte de un usuario o de un gerente de un departamento que haya detectado una necesidad de mejoramiento de un sistema o de automatización de una serie de actividades. Esta fase consta de tres partes: (a) aclaración de la solicitud: se determina con precisión, qué es lo que realmente el usuario necesita mejorar o automatizar, (b) estudio de factibilidad: en esta parte se determina si lo que el usuario solicita es factible, tomando en cuenta la factibilidad técnica,

económica y operacional y (c) aprobación de la solicitud: una vez que se aprueba el proyecto, se estima su costo y el tiempo necesario para hacerlo; así como también la necesidad de personal para realizarlo.

2. Determinación de los requerimientos del sistema: en esta fase se estudian los procesos de la empresa, para dar respuesta a las siguientes preguntas claves: ¿Qué es lo que se hace?, (b) ¿Cómo se hace?, (c) ¿Con qué frecuencia se presenta?, (d) ¿Qué tan grande es el volumen de transacciones o de decisiones?, (e) ¿Cuál es el grado de eficiencia con el que se efectúan las tareas?, (f) ¿Existe algún problema?, (g) Si existe un problema, ¿qué tan serio es? y (h) si existe un problema, ¿cuál es la causa que lo origina? Se utilizan como técnicas de recolección de datos, la entrevista y la encuesta. Así mismo, se requiere del estudio de manuales y reportes, la observación directa de las actividades que se realizan y en algunos casos, formas y documentos para comprender mejor el proceso en su totalidad.
3. Diseño del sistema: en esta etapa, el analista usa la información recolectada anteriormente para realizar el diseño lógico del sistema de información. "Los especialistas en sistemas se refieren, con frecuencia, a esta etapa como diseño lógico en contraste con la de desarrollo de software, a la que denominan diseño físico". Dicho diseño es realizado por el analista y contempla entre otras cosas: (a) procedimientos precisos para la captura de datos, a fin de que los datos que van a entrar al sistema, sean correctos; (b) diseño de formas y pantallas, para lo cual se deben usar técnicas que garanticen una alta calidad; (c) diseño de bases de datos, donde se guardarán la mayor parte de los datos necesarios para quienes toman decisiones dentro de la empresa; (d) diseño de la salida de información de las bases de datos, esta puede ser en pantalla o impresa, según como se requiera satisfacer las necesidades de información y (e) diseño de procedimientos de control y respaldo para proteger al sistema y a los datos. Los documentos que contengan las especificaciones de diseño se representarán por medio de diagramas de flujo, tablas, símbolos especiales, árboles, graficas, entre otros. Los diseñadores son los

responsables de dar a los programadores, las especificaciones del sistema de información completas y claramente delineadas.

4. Desarrollo de sistemas: en esta etapa el analista trabaja junto con el programador para desarrollar cualquier sistema que se necesite, basándose en el diseño realizado previamente. Los programadores llevan a cabo las siguientes funciones: (a) codifican los módulos correspondientes, (b) realizan pruebas integrales a cada módulo para garantizar su correcto funcionamiento, (c) redactan la documentación del sistema, elaborando el manual del usuario que sirve al usuario final (aquellos usuarios que usarán el sistema constantemente), para aprender a manejarlo y el manual del sistema, en el cual se explica la forma de programar e implementar los módulos. Esta documentación es de vital importancia para probar el sistema y posteriormente para su mantenimiento, una vez que éste haya sido implantado.
5. Pruebas del sistema: el objetivo de esta etapa es detectar y corregir los errores que se cometieron en el proceso de desarrollo del sistema, en cuanto al funcionamiento esperado por el usuario. Estas pruebas consisten en hacer funcionar al sistema como si estuviera realizando las operaciones cotidianas para lo cual fue desarrollado. En este sentido, se introducen entradas de conjunto de datos para su procesamiento y después se examinan sus salidas o resultados con aquellas salidas o resultados o esperados. Muchas veces se permite a los usuarios finales utilizar el sistema como ellos lo usarían sin limitarlos. Es decir, dejarlos utilizar el sistema en forma libre para así, poder detectar fallas o errores no encontrados en el proceso de desarrollo del sistema.
6. Implantación y evaluación: durante esta etapa se procede a instalar y verificar el nuevo sistema, capacitar a los usuarios que lo usarán y realizar la conversión o migración del viejo sistema al nuevo, verificando que los usuarios no encuentren inconvenientes en el uso del mismo. Posteriormente, se debe cubrir la fase de evaluación, en la cual se identifican las fortalezas y debilidades del sistema de información.

Muchas metodologías varían en las fases que involucran y pueden realizar procesos de retroalimentación o iterativos, hasta lograr tener un producto de software terminado. Sea cual sea la metodología, finalmente se busca producir una aplicación lista para su uso.

Éxito y fracaso en Proyectos de Desarrollo de Software

Sin considerar el tamaño del proyecto, su alcance o duración, existen 5 máximas o indicadores de satisfacción del cliente en lo que respecta al desarrollo del proyecto (Boyd, 2001): (a) entregar el producto que el cliente desea o necesita, (b) entregar la calidad de manera acorde con el precio, (c) entregar el producto en el espacio de tiempo que el cliente desea o necesita, (d) entregar el nivel de retroalimentación que el cliente desea y (e) contar con un sistema de soporte o resolución de conflictos, para el cliente conjuntamente con el equipo de desarrollo.

Se enumeran a continuación, los pasos básicos que se consideran esenciales para lograr una administración eficiente de proyectos (Toledo, 2002):

1. Nunca iniciar un proyecto sin un objetivo bien definido.
2. Fragmentar el proyecto.
3. Invertir tiempo en la planificación.
4. Involucrar al equipo de trabajo en la planificación y el control.
5. Fomentar la cohesión del equipo de trabajo.
6. Prevenir problemas.
7. Antes de ejecutar, establecer líneas de base.
8. Mantener claro el objetivo principal del proyecto.
9. Establecer un proceso para monitorear y controlar.
10. Atender los puntos críticos primordialmente.
11. Tomarse el tiempo necesario para cerrar el proyecto.
12. Utilizar una metodología para todos los proyectos.

Comparación entre Metodología de Gestión de Proyectos y Metodología de Desarrollo de Software

Según Caballero (2006), es importante establecer una distinción entre una metodología de Administración (Gestión) de Proyectos y una metodología de Desarrollo de Software, ya que una organización debe contar con una metodología de Gestión de Proyectos consistente a cualquiera que sea la naturaleza del proyecto que se desarrolla. En este caso, la mayoría de los proyectos de TI implicarán el uso de metodologías de desarrollo.

Las diferencias entre ambos conceptos se muestran en el siguiente cuadro:

Cuadro 1. Metodologías de Gestión de Proyectos Vs Metodología de Desarrollo de Aplicaciones

Metodología de Gestión de Proyectos	Metodología de Desarrollo de Aplicaciones
Establece que los proyectos deben ser divididos en fases y antes de iniciar con cada una de estas fases, debe existir un plan.	Establece cuáles son las fases y qué actividades involucra.
Define roles y responsabilidades de los participantes en el proyecto. Ejemplos de roles en un proyecto son el ingeniero de sistemas, el analista de negocios o el coordinador de pruebas. Un rol es un conjunto de actividades y responsabilidades asignada a una persona o un grupo.	Define cuáles son los roles y responsabilidades que corresponden a cada fase. Por ejemplo, algunos roles es Scrum serían: (a) **Usuario o Cliente**: es el destinatario final del producto y es quien acompaña el progreso del desarrollo y (b) **Manager:** toma las decisiones finales participando en la selección de los objetivos y de los requisitos.
Establece que un presupuesto debe ser definido y administrado.	Define qué medidas deben emplearse para determinar el desarrollo en la organización.

CAPÍTULO III

METODOLOGÍAS DE DESARROLLO PARA PROYECTOS TI

Definición de metodología

Según INTECO (2009) “Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo.” (p. 39).

Ventajas del uso de una metodología

Según INTECO (2009), las ventajas del uso de una metodología son las siguientes:

1. Desde el punto de vista de gestión: facilitar la tarea de planificación, la tarea del control y seguimiento de un proyecto, mejorar la relación costo/beneficio, optimizar el uso de recursos disponibles, facilitar la evaluación de resultados y cumplimiento de los objetivos y facilitar la comunicación efectiva dentro del equipo.
2. Desde el punto de vista de los ingenieros de sistemas: ayudar a la comprensión del problema, optimizar el conjunto y cada una de las fases del proceso de desarrollo, facilitar el mantenimiento del producto final y permitir la reutilización de componentes del producto.
3. Desde el punto de vista del cliente o usuario: garantía de un determinado nivel de calidad en el producto final, confianza en los plazos de tiempo fijados en la definición del proyecto y apoyo en la definición del ciclo de vida que más se adecue a las condiciones y características del desarrollo.

Historia de las metodologías de desarrollo de Software

La historia de las metodologías de desarrollo de software, está íntimamente ligada a la evolución de la computación, en cuanto a hardware y a la forma de manejar los datos por dicho hardware. En este sentido, se presenta a continuación un recorrido por los principales hitos que se han producido en cada década, desde de 1940, cuando apareció el ordenador digital.

Década de 1940

Se señala en INTECO (2009) que la historia de la ingeniería de sistemas y de software está entrelazada con las historias contrapuestas de hardware y software. Cuando el ordenador moderno apareció por primera vez en 1941, las instrucciones para hacerlo funcionar estaban conectadas dentro de la máquina. Las personas relacionadas con la ingeniería, rápidamente se dieron cuenta de que este diseño no era flexible e idearon la arquitectura de programa almacenado o arquitectura Von Neumann. De esta forma, la primera división entre “hardware” y “software” empezó con la abstracción usada para tratar la complejidad de la computación.

En esta década se usó el lenguaje máquina o código binario. Sus instrucciones son cadenas binarias (cadenas o series de caracteres de dígitos 0 y 1) que especifican una operación y, las posiciones (dirección) de memoria implicadas en la operación se denominan instrucciones de máquina o código máquina.

Una de las principales ventajas del lenguaje máquina era la posibilidad de cargar (transferir un programa a la memoria) sin necesidad de traducción posterior, lo que supone una velocidad de ejecución superior a cualquier otro lenguaje de programación.

Entre las desventajas del lenguaje máquina, se pueden destacar: (a) dificultad y lentitud en la codificación, (b) poca fiabilidad, (c) gran dificultad para verificar y poner a punto los programas y, (d) los programas sólo son ejecutables en el mismo procesador (CPU).

Década de 1950

Algunos avances importantes en la década de 1950 fueron la aparición de los lenguajes de programación ensambladores y de alto nivel, estos constituyeron pasos importantes en la abstracción.

Los lenguajes de programación ensambladores, se crearon a principios de la década de los 50 con el fin de facilitar la labor de los programadores. Estos lenguajes son más fáciles de utilizar que los lenguajes máquina, pero al igual que ellos, dependen de la máquina en particular. La computadora sigue utilizando el lenguaje máquina para procesar los datos, pero los programas ensambladores traducen antes los símbolos de código de operación especificados a sus equivalentes en el lenguaje máquina.

Los lenguajes, principales de programación de alto nivel, como Fortran y Algol se lanzaron a finales de los 50s para tratar con problemas científicos y algorítmicos respectivamente. Algunas características de este tipo de lenguaje se mencionan a continuación: (a) los programas son portables o transportables: un programa escrito en lenguaje de alto nivel es independiente de la máquina (las instrucciones no dependen del diseño del hardware o de una computadora en particular); (b) los programas escritos en lenguaje de alto nivel pueden ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras y, (c) los lenguajes son más fáciles de aprender, ya que las instrucciones enviadas para que el ordenador ejecute ciertas órdenes, son similares al lenguaje humano.

Por otra parte, según Jibaja (2011) desde que se empezó a trabajar sobre el desarrollo de programas, se siguieron ciertos métodos que permitían llevar a producir un buen proyecto. Estas metodologías aplicadas eran simples, solo se preocupaban por los procesos, pero no por los datos. Por lo tanto, los métodos eran desarrollados y enfocados hacia los procesos, y consistían en c*odificar y corregir* (Code-and-Fix). Este tipo de metodología tuvo su auge en la década de 1960, por lo cual será detallada en el siguiente apartado.

Década de 1960

El modelo de procesos predominaba en los años sesenta y consistía en codificar y corregir (Code-and-Fix). Si al terminar la codificación, se descubría que el diseño era incorrecto, la solución era desecharlo y volver a empezar. Este modelo implementaba el código y luego se pensaba en los requisitos, diseño, validación y mantenimiento. Los principales problemas de este modelo son: (a) los arreglos son realmente costosos, ya que después de tantas correcciones, el código tenía una estructura precaria e incorrecta; (b) el software no se ajusta a las necesidades del usuario, por lo que es rechazado y/o su reconstrucción es muy cara; y (c) el código es difícil de reparar por la falta de documentación que detalle el objeto del mismo, provocando subjetividades al momento de entender con qué objetivo se codificó de cierta forma.

En cuanto a los lenguajes de programación de alto nivel, se destaca la creación del lenguaje COBOL en 1960, enfocado a los negocios. Este lenguaje se popularizó rápidamente y sufrió diversas actualizaciones, siendo un predecesor importante para el manejo de las bases de datos actuales.

Otro lenguaje de programación que se destacó en esta década fue el lenguaje Simula, creado en 1967 por Kristen Nygaard y Ole-Johan Dahl, del Centro Noruego de Computación en Oslo. El objetivo inicial era definir un lenguaje de propósito específico para aplicaciones de simulación. Se introdujeron los primeros conceptos dentro del paradigma de la programación orientada a objetos, tales como: clase, objeto, herencia, polimorfismo, entre otros.

A su vez, como un hito importante en lo que respecta a sistemas operativos, a mediados de esta década se destaca el comienzo del sistema operativo Unix, que inició como un esfuerzo conjunto entre el Instituto de Tecnología de Massachusetts con los laboratorios AT&T Bell y General Electric, para desarrollar un sistema operativo experimental de tiempo compartido, llamado Multics que posteriormente llevaría un rotundo cambio estructural y el cambio de nombre a Unix.

Década de 1970

En la década de los setenta, empezaron a tomar importancia los datos, y para solucionar sistemas complejos, se comenzó con el análisis por partes o etapas. Desde este punto, se introdujeron conceptos como la planificación y administración. El modelo en cascada surge como respuesta al modelo de procesos. Este modelo tiene más disciplina y se basa en el análisis, diseño, pruebas y mantenimiento.

David Parnas introdujo el concepto clave de la modularidad y encapsulación en 1972, para ayudar a los programadores a tratar con la complejidad de los sistemas de software.

A mediados de los setenta, la microcomputadora fue introducida, haciendo económico a los aficionados obtener una computadora y escribir software para él. Esto, condujo sucesivamente al famoso ordenador personal o PC y a Microsoft Windows. A finales de los 70s y principios de los 80s, se vieron varios lenguajes de programación orientados a objetos inspirados en Simula, incluyendo C++, Smalltalk y Objective C.

Por otra parte, en 1976 en EEUU, se da el nacimiento de la Técnica de Análisis y Diseño Estructurado (SADT, por sus siglas en inglés: Structured Analysis and Design Technique). Esta técnica fue creada por la sociedad americana Sorteen en colaboración con la ITT, por D.T. Ross. Según (De Amescua, 1990), “SADT abrió un nuevo camino en las áreas de análisis de problemas, definición de requisitos y especificación funcional, ya que permite una expresión rigurosa de ideas de alto nivel, que previamente habían parecido demasiado indeterminadas para tratar técnicamente” (p. 49). El método fue usado por grandes grupos industriales, tales como: Thomson, Philips, Aérospatiale, Toshiba, entre otros y organismos nacionales o internacionales, como por ejemplo la Agencia Espacial Europea. Posteriormente, en 1982, SADT fue introducido en Francia, por IGL Technology, formando a una gran cantidad de ingenieros para la especificación de sistemas.

SADT incluyo los siguientes modelos: (a) modelo del entorno actual, (b) modelo de operaciones actuales, (c) modelo del nuevo sistema, (d) modelos funcionales

y de diseño, (e) modelo del plan de test, (f) nuevo modelo del entorno, (g) nuevo modelo de operaciones, (h) modelo de conversión e (i) modelo de desarrollo del sistema.

Década de 1980

La década de los ochenta es la época marcada por las metodologías dirigidas a los datos, cuya importancia fue tomando cuerpo en las organizaciones. Se empiezan a estudiar los objetos en sí, como unidades de información. También, es importante destacar que el ciclo de vida de Desarrollo de Software (SDLC), empezó a aparecer a mediados de esta década como un consenso para la construcción centralizada de software. En esta década, se plantearon las siguientes metodologías:

1. Metodología de Análisis y Diseño de Sistemas Estructurado (SSADM por sus siglas en inglés: Structured Systems Analysis and Design Methodology): señala (De Amescua, 1990) que esta metodología fue creada para los proyectos del gobierno británico, en el año 1980, por la empresa consultora Learmonth y Burchett Management Systems (LBMS) para la Central Computer and Telecommunications Agency (CCTA), la cual era responsable entre otras funciones, de la formación de informática y telecomunicaciones para el Servicio Civil del Reino Unido. Esta metodología consta de seis (6) fases: (a) análisis: busca construir un modelo lógico del sistema actual, documentar los problemas del sistema actual y los requerimientos para el nuevo sistema; (b) especificación de requerimientos: busca construir un modelo lógico del sistema requerido junto con documentación detallada; (c) selección de opción del sistema: es en esta tercera fase que se identifican y documentan los requerimientos opcionales para el nuevo sistema; (d) diseño lógico de datos: se busca completar una especificación de diseño lógico de datos detallada; (e) diseño lógico de procesos: se completan un conjunto de diseños lógicos de procesos detallados y (f) diseño físico: se traslada el diseño lógico de

datos a las especificaciones de ficheros o bases de datos y los diseños lógicos de procesos a las especificaciones de programa.

2. Ingeniería de la Información (IE/IEM): En 1981, Clive Finkelstein escribió una serie de artículos sobre la metodología. En ese mismo año colaboró con James Martin en un libro. A partir de entonces, aparecen diferentes versiones de IE, las cuales son muy similares en el contenido, pero con algunas pequeñas líneas diferentes. La razón de esto, es que James Martin fundó una serie de compañías independientes basadas en la metodología de IE. La principal base filosófica de IE es la creencia de que los datos son el corazón de un sistema de información y que, además, éstos son mucho más estables que los procesos que actúan sobre los datos. IE también considera los procesos en detalle, pero siendo los datos la base del sistema de información. La metodología está compuesta de cuatro fases: (a) planificación: el objetivo es construir una arquitectura de información y una estrategia que soporte los objetivos de la organización; (b) análisis: el objetivo es entender las áreas de negocio y determinar los requisitos del sistema; (c) diseño: el objetivo de esta fase es establecer el comportamiento de los sistemas de la manera que el usuario lo espera y que se pueda llevar a cabo con la tecnología existente y, (d) construcción: el objetivo en esta fase es construir los sistemas tal como han sido especificados en las fases previas. (De Amescua, 1990).
3. Modelo de espiral: este modelo fue propuesto originalmente por BOEHM en 1976, es un modelo de proceso de software evolutivo donde se conjuga la naturaleza de construcción de prototipos con los aspectos controlados y sistemáticos del Modelo Lineal y Secuencial. Proporciona el potencial para el desarrollo rápido de versiones incrementales del software, que no se basa en fases claramente definidas y separadas para crear un sistema. Este modelo será ampliado posteriormente.

Década de 1990

Para los años noventa, se busca dar respuesta al entorno siempre cambiante y en rápida evolución, en el cual se desarrollan los programas informáticos. Esto da lugar a trabajar en ciclos cortos de desarrollo (como mini-proyectos), que implementan una parte de las funcionalidades, pero sin perder el rumbo general del proyecto global. Asimismo, las metodologías de desarrollo comienzan a interesarse no sólo en lograr que el proyecto sea puesto en funcionamiento, sino en minimizar costos durante su desarrollo y sobre todo durante su mantenimiento.

Por otra parte, el software “Open Source” comienza a aparecer a principios de esta década, con el lanzamiento de Linux y otros Softwares introduciendo el “bazar” o el estilo descentralizado de construcción de software. Después, aparecieron Internet y la World Wide Web a mediados de los noventa, cambiando fuertemente de nuevo a la ingeniería de sistemas y de software. Los sistemas distribuidos ganaron dominio como forma de diseñar sistemas y el lenguaje de programación Java se introdujo como otro paso en la abstracción, teniendo su propia máquina virtual.

Asimismo, en esta década se produce otro hito importante que afecta a las metodologías de desarrollo de forma muy directa. Este hito es resultante de la reunión de diversos programadores que colaboraron y escribieron el manifiesto ágil, que favoreció al disponer de procesos más ligeros para crear software más barato y en menos tiempo. (INTECO, 2009).

Algunas de las metodologías surgidas en esta década son:

1. Rapid Application Development (RAD) desde 1991: Según INTECO (2009), la metodología de desarrollo rápido de aplicaciones (RAD) se desarrolló para responder a la necesidad de entregar sistemas muy rápido. El método comprende el desarrollo interactivo, la construcción de prototipos y el uso de utilidades CASE (Computer Aided Software Engineering). Tradicionalmente, el desarrollo rápido de aplicaciones tiende a englobar también la usabilidad, utilidad y la rapidez de ejecución. Si bien este enfoque no es apropiado para todos los proyectos, ya que se

deben tomar en cuenta algunos aspectos en particular, a continuación se definen las características de proyectos en el que RAD resultaría efectivo de aplicar: (a) proyectos donde convergir tempranamente en un diseño aceptable para el cliente y posible para los desarrolladores; (b) limitar la exposición del proyecto a las fuerzas de cambio y (c) ahorrar tiempo de desarrollo, posiblemente a expensas de dinero o de calidad del producto. Hoy en día, se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario, tales como Glade, o entornos de desarrollo integrado completos. Algunas de las plataformas más renombradas son Visual Studio, Lazarus, Gambas, Delphi, Foxpro, Anjuta, Game Maker, Velneo o Clarion.

2. Dynamic Systems Development Method (DSDM): el método de desarrollo de sistemas dinámico (DSDM) es una metodología de desarrollo de software originalmente basado en la metodología RAD. Éste es un enfoque iterativo e incremental que enfatiza la participación continua del usuario de manera directa. Fue desarrollado en Reino Unido por el DSDM Consortium, una asociación de vendedores y expertos en el campo de la ingeniería creado con el objetivo de "desarrollar y promover conjuntamente un marco de trabajo RAD independiente", combinando sus propias experiencias.
3. Rational Unified Process (RUP) desde 1999: el proceso unificado Rational (RUP), es un marco de trabajo de proceso de desarrollo de software iterativo, creado por Rational Software Corporation, Actualmente es propiedad de IBM y el producto incluye una base de conocimiento con artefactos de ejemplo y descripciones detalladas para diversos tipos de actividades. RUP resultó de la combinación de varias metodologías y se vio influenciado por métodos previos, como el modelo en espiral. Se consideraron para el desarrollo de la metodología los siguientes puntos: (a) énfasis en disminuir el fallo en proyectos, buscando detectar los defectos en las fases iniciales, intentando reducir el número de cambios y realizando un análisis y diseño exhaustivo; (b) aplicar el desarrollo orientado a objetos y las tecnologías GUI; (c) un deseo de elevar el modelado de sistemas a la práctica del desarrollo y, (d) adoptar los principios de calidad, que se venían aplicando a las manufacturas en

general, al desarrollo del Software, de forma tal que el control de calidad se realice en todos los aspectos de la producción y no al final de cada iteración. (INTECO, 2009).

Década de 2000 hasta a la fecha

Los nuevos métodos van buscando minimizar riesgos y, debido a que los errores más perjudiciales se producen en los primeros pasos, desde la fase más general del estudio se comienzan a analizar los posibles riesgos que se pueden presentar al continuar con las siguientes fases del desarrollo. De forma tal, que se puedan tomar decisiones acertadas en etapas tempranas del proyecto. Algunas de las metodologías surgidas a partir del año 2000 son:

1. Extreme Programming (XP) desde 1999: La programación extrema (Extreme Programming XP) es una metodología ágil[1], formulada por Kent Beck y dirigida a equipos de desarrollo de software de pequeños a medianos, donde los requisitos son vagos o cambian rápidamente. El nombre de XP se debe a que se toman los principios y prácticas de sentido común y se llevan al extremo. En el siguiente capítulo se encontrarán más detalles de esta metodología.
2. Enterprise Unified Process (EUP) extensiones RUP desde 2002: La metodología Enterprise Unified Process (EUP) se basa en la extensión de la metodología Rational Unified Process, RUP. Esta extensión se presenta en dos fases más y una sección de disciplinas de soporte, que agrega cuatro disciplinas más a las que ya cuenta la metodología RUP. La diferencia teórica entre RUP y EUP es que la metodología RUP se enfoca únicamente en el ciclo de vida del desarrollo de software, en cambio el EUP cubre el ciclo de vida de la tecnología de información, es decir, abarca una visión más amplia que el desarrollo de software enfocándose en las etapas siguientes a la elaboración del software. Etapas como, por ejemplo, la inserción del nuevo sistema en una empresa donde se cuenta con otro sistema

[1] Antes del 2001 la denominación para las metodologías Ágiles era metodologías ligeras.

antiguo. Para esto se deberá de decidir si los dos sistemas van en paralelo o si uno se retira por completo o en otro caso, si se complementan.

3. Constructionist Design Methodology (CDM) desde 2004: La Metodología de diseño construccionista (MDL), fue desarrollada por la inteligencia artificial (AI), por el investigador Kristinn R. Thorisson y sus estudiantes en las Universidades de Columbia y de Reykjavik, para su uso en el desarrollo de la robótica cognitiva, humanoides de comunicación y los sistemas de AI Amplio (Broad IA): estos son sistemas sensibles al contexto en el que pueden imitar la actividad humana o la toma de decisiones.

Línea de vida de las Metodologías de desarrollo de Software

A continuación, se presentan de forma resumida, los principales hitos de la historia de las metodologías de desarrollo (ver gráfico 3).

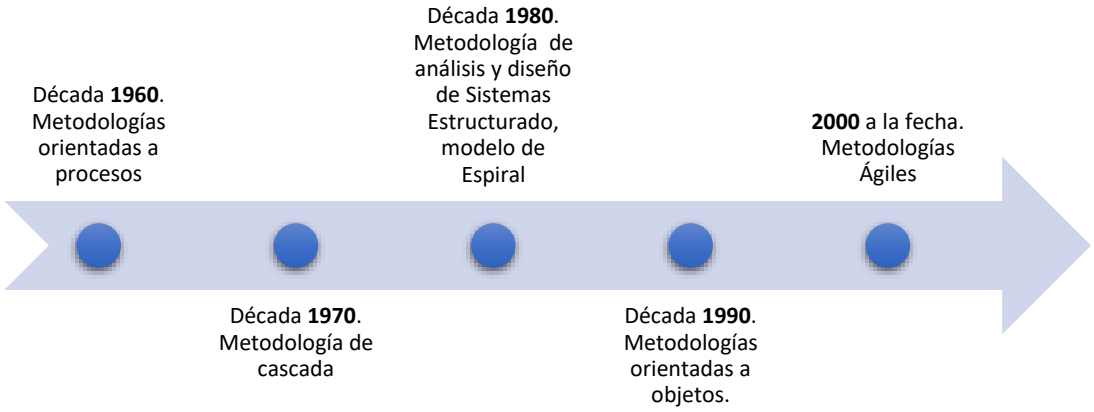


Gráfico 3. **Línea de vida de las metodologías de desarrollo de Software**

Paradigmas de las Metodologías de Desarrollo

Un paradigma indica un método de realizar cómputos y la manera en que se deben estructurar y organizar las tareas que debe llevar a cabo un programa. El paradigma influye en las metodologías de desarrollo, ya que éstas deben considerarlo

para la determinación de sus técnicas y herramientas. Se considerarán en esta investigación: el paradigma tradicional, el paradigma orientado a objetos y paradigma de desarrollo ágil.

Paradigma Tradicional

Es uno de los paradigmas más antiguo, se inventó durante la creación del método estructurado que concibe que cuando se elige un proyecto, el método puede variar en la cantidad de etapas.

Si se aplica este paradigma, unos de los principales problemas, es que las etapas realizadas no son independientes de las siguientes, creando una dependencia estructural y en el caso de un error, se atrasa todo el proyecto. Es por eso que, se tienen que tener pautas bien definidas, y que éstas no incurran en modificaciones, ya que implicaría que el software no cumpla con su ciclo de vida. También, un punto muy importante es el hecho de que, se debe tener en cuenta que el cliente no se vea afectado por la impaciencia.

Paradigma Orientado a Objetos

Estos modelos son proyectados para implementar directamente un enfoque orientado a objetos, donde se refiere al concepto de clase, el análisis de requisitos y el diseño. Este paradigma posee dos características principales, las cuales son: (a) permite la re-utilización de software y (b) facilita el desarrollo de herramientas informáticas de apoyo al desarrollo.

A continuación, se muestran algunas de las metodologías orientadas a objetos más populares (Meyer, 1999).

- Metodología Coud-Yourdon: surgió inicialmente de un esfuerzo por trasladar la orientación a objetos, a las ideas procedentes del análisis estructurado. Contiene cinco fases: (a) búsqueda de clases y objetos, partiendo del dominio de la aplicación y analizando responsabilidades del sistema; (b) identificación de estructuras:

buscando las relaciones de generalización-especialización y de todo-parte; (c) búsqueda de "sujetos": grupos de clases objetos; (d) definición de atributos y (e) definición de servicios.

- Metodología OMT (Técnica de Modelado de Objetos): esta metodología "combina conceptos de la tecnología de objetos con otros pertenecientes al modelado de entidad-relación. La metodología incluye un modelo estático, basado en los conceptos de clase, atributo, operación, relación y agregación, y un modelo dinámico basado en diagramas evento-estado, que describen de forma abstracta el comportamiento que se pretenda que tenga el sistema" (p. 869).
- Metodología OOIE (Ingeniería de la Información Orientada a Objetos): fue creado por Martin-Odell, consta de dos partes: (a) análisis de estructura de objetos: identifica los tipos de objetos y sus relaciones de composición y herencia y (b) análisis de comportamiento de objetos: define el modelo dinámico, considerando los estados de los objetos y los eventos que puedan modificar estos estados.
- Metodología OOSE (Ingeniería de Software Orientado a Objetos): se basa en la utilización de casos de uso (escenarios) para extraer las clases. Distingue cinco (5) modelos de casos de uso: modelo de objetos del dominio, modelo de análisis (los casos de uso estructurados mediante el análisis), modelo de diseño, modelo de implementación y modelo de comprobación.

Paradigma de Desarrollo Ágil

Es un paradigma de las Metodologías de Desarrollo basado en procesos Ágiles. Estos intentan evitar los tediosos caminos de las metodologías tradicionales, enfocándose en las personas y en los resultados. Se utiliza un enfoque basado en el Valor para construir software, colaborando con el cliente e incorporando los cambios continuamente. Es importante destacar que este paradigma será ampliado en el próximo capítulo, ya que constituye el tema central de este proyecto.

Metodologías Tradicionales

Cascada (Waterfall)

Este modelo según Sommerville (2001), fue el primer modelo de desarrollo de Software que se publicó y establece una secuencia de fases que deben concluirse totalmente para avanzar. Se establecen las siguientes cinco (5) fases:

1. Análisis y definición de requerimientos: se definen a partir de las consultas con los usuarios, los servicios, las restricciones y metas del sistema.
2. Diseño de sistemas y de software: en esta fase se dividen los requerimientos en sistemas de hardware o de software. Establece una arquitectura completa del sistema. El diseño de software identifica y describe las abstracciones fundamentales del sistema de software y sus relaciones.
3. Implementación y prueba de unidades: durante esta etapa, el diseño de software se lleva a cabo como un conjunto o unidades de programas. La prueba de unidades implica verificar que cada una cumpla su especificación.
4. Integración y prueba del sistema: los programas o las unidades individuales de programas se integran y son validadas como un sistema completo e integrado, para asegurar que se cumplan los requerimientos del software. Después de estas pruebas, el sistema de software es entregado al usuario.
5. Operación y mantenimiento: por lo general (aunque no necesariamente), ésta es la fase más grande del ciclo de vida. El sistema se implementa, se instala y se dispone para uso práctico. El mantenimiento implica corregir errores no descubiertos en las etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema y también, resaltar los servicios del sistema una vez que se descubren nuevos requerimientos. (ver gráfico 4).

Gráfico 4. **Fases de la metodología de cascada**

La falta de flexibilidad en un modelo de cascada puro ha sido fuente de crítica de los defensores de modelos más flexibles.

Modelo de Espiral

Según INTECO (2009), la principal característica del modelo en espiral es la gestión de riesgos de forma periódica en el ciclo de desarrollo. Este modelo fue creado en 1985 por Barry Boehm, combinando algunos aspectos claves de las metodologías del modelo de cascada y del desarrollo rápido de aplicaciones, pero también, dando énfasis en un área que, para muchos, no jugó el papel que requiere en otros modelos: un análisis iterativo y minucioso de los riesgos, especialmente en el caso de sistema complejos de gran escala. Las actividades de este modelo se conforman en una espiral, cada bucle representa un conjunto de actividades. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgos, comenzando por el bucle anterior.

El espiral se visualiza como un proceso que pasa a través de algunas interacciones con el diagrama de los cuatro cuadrantes representativos de las siguientes

actividades: (a) determinar objetivos, (b) evaluar riesgos, (c) desarrollar y probar y (d) planificar (ver gráfico 5).

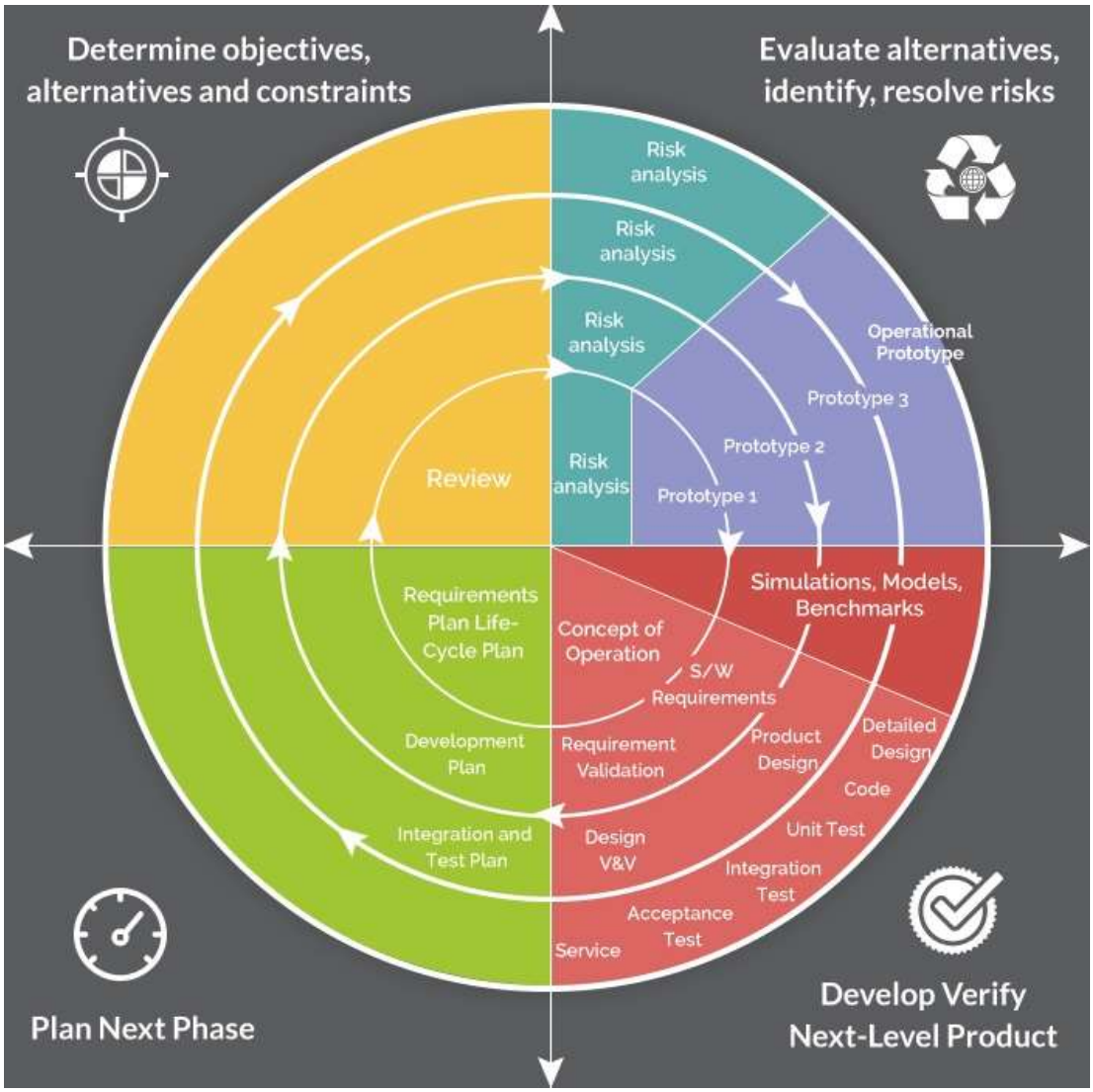


Gráfico 5. **Actividades del modelo de espiral**

El Modelo de espiral establece énfasis en los riesgos, haciendo hincapié en las condiciones de las opciones y limitaciones para facilitar la reutilización de software. En este sentido, un riesgo es la probabilidad de que una amenaza se convierta en un problema o desastre. Es decir, se denomina riesgo a todo lo que pueda salir mal

en un proyecto de desarrollo de software. Por ejemplo, si se requiere utilizar un lenguaje de programación en particular para desarrollar un sistema operativo, un posible riesgo es que los compiladores a utilizar no produzcan un código objeto eficiente, lo cual puede traer diferentes tipos de problemas a futuro como lo puede ser un bajo performance. Los riesgos originan problemas en el proyecto, como el exceso de los costos. Es así que, la disminución de los riesgos es una actividad fundamental. El modelo en espiral tiene algunas limitaciones, entre las que destacan:

El énfasis se sitúa en el análisis de riesgo, y por lo tanto requiere que los clientes acepten este análisis y que actúen en consecuencia. Para ello, es necesaria una determinada confianza en los desarrolladores, así como la predisposición a que el costo sea mayor para solventar algunos inconvenientes, por lo cual este modelo se utiliza frecuentemente en desarrollo interno de software a gran escala. Es por eso que, no debería utilizarse este modelo si la implementación del riesgo de análisis afectará de forma esencial a los beneficios del proyecto.

Los desarrolladores de software necesitan identificar de forma explícita los riesgos y analizarlos de forma exhaustiva para que este modelo funcione.

La primera fase es la búsqueda de un plan para conseguir los objetivos con las limitaciones del proyecto. De esta forma, se buscan y se eliminan todos los riesgos potenciales por medio de un minucioso análisis, y de ser necesario se incluye el desarrollo de un prototipo. Si es imposible descartar algunos riesgos, el cliente necesitará decidir que es conveniente realizar: si se definen los pasos a seguir para reducir los riesgos y luego del análisis de estos riesgos, se planean estrategias alternativas; o los mismos se ignoran, aceptando sus posibles consecuencias.

Dependiendo del resultado de la evaluación de riesgos, se elige un modelo para el desarrollo, que puede ser cualquiera de los otros existentes, como: formal, evolutivo, cascada, etc. Así, por ejemplo, si los riesgos de la interfaz de usuario son dominantes, un modelo de desarrollo apropiado podría ser la construcción de prototipos evolutivos. Por último, se evalúan los resultados y se inicia el diseño de la siguiente fase.

Desarrollo Iterativo

Según INTECO (2009), el desarrollo iterativo es un modelo derivado del ciclo de vida en Cascada. Consiste en la iteración de varios ciclos de vida en Cascada (ver gráfico 6).

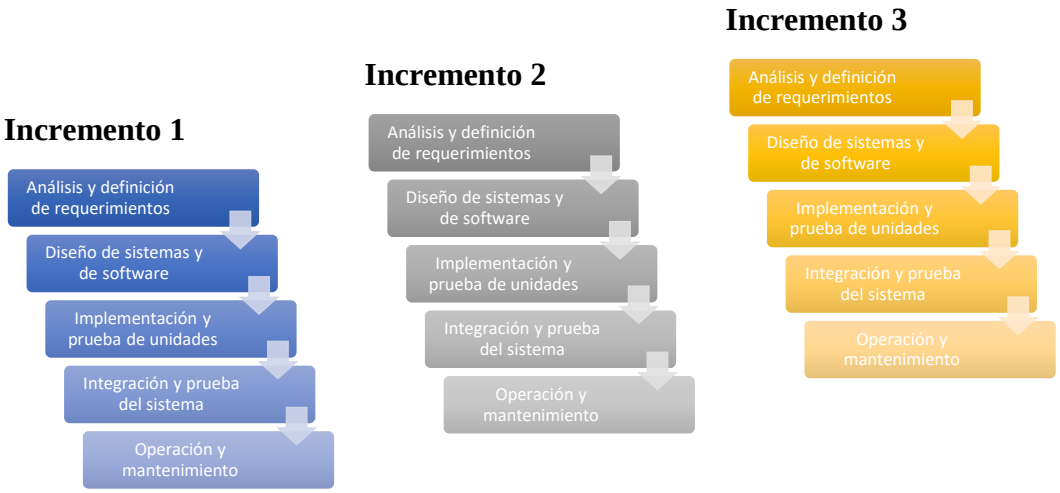


Gráfico 6. **Desarrollo Iterativo**

Se recomienda la construcción de secciones reducidas de software, que irán ganando en tamaño para facilitar así, la detección de problemas de importancia antes de que sea demasiado tarde. Los procesos iterativos pueden ayudar a desvelar metas del diseño, en caso de clientes que no disponen de una definición clara de lo que necesiten o consideran necesario.

Esta metodología ha sido fundamental para el avance en las metodologías ágiles, ya que permite realizar entregas parciales de funcionalidades, al cliente.

Las iteraciones se pueden entender como mini proyectos: en todas las iteraciones se repite un proceso de trabajo similar (de ahí el nombre “iterativo”), para proporcionar un resultado completo sobre producto final, de manera que el cliente pueda obtener los beneficios del proyecto de forma incremental. De esta forma, cada requisito se debe completar en una única iteración: el equipo debe realizar todas las

tareas necesarias para completarlo (incluyendo pruebas y documentación) y también, que esté preparado para ser entregado al cliente con el mínimo esfuerzo necesario. De esta manera, no se deja para el final del proyecto, ninguna actividad arriesgada relacionada con la entrega de requisitos.

En cada iteración, el equipo evoluciona el producto (hace una entrega incremental), a partir de los resultados completados en las iteraciones anteriores, agregando nuevos objetivos/requisitos o mejorando los que ya fueron completados.

Ventajas y Desventajas de las Metodologías Tradicionales

A continuación, se muestra un cuadro comparativo con las ventajas y desventajas de las metodologías tradicionales presentadas anteriormente (ver cuadro 2).

Cuadro 2. Ventaja y desventajas de las Metodologías Tradicionales

Metodología Cascada	Modelo de espiral	Desarrollo iterativo
Ventajas: 1. Puede ser apropiado, en general, para proyectos estables (especialmente los proyectos con requisitos no cambiantes). 2. Funciona bien para proyectos pequeños, donde los requisitos están bien entendidos. 3. Es un modelo en el que todo está bien organizado y no se mezclan las fases. 4. Es simple y fácil de usar. Debido a la rigidez del modelo es fácil de gestionar, ya que cada fase tiene entregables específicos y un proceso de revisión. 5. Las fases son procesadas y completadas de una vez. **Desventajas:** 1. Un proyecto rara vez sigue una secuencia lineal, esto crea una mala implementación del modelo. 2. Difícilmente un cliente va a establecer al principio, todos los requisitos necesarios de forma correcta, por lo que es posible que exista un atraso en el desarrollo. 3. Los resultados y/o mejoras no son visibles progresivamente, el producto se ve cuando ya está finalizado, lo cual provoca una gran inseguridad por parte del cliente. .	**Ventajas** 1. Reduce riesgos del proyecto. 2. Incorpora objetivos de calidad. 3. Integra el desarrollo con el mantenimiento. 4. Es posible tener en cuenta mejoras y nuevos requerimientos sin romper con el modelo, ya que el ciclo de vida no es rígido ni estático. 5. Se produce software en etapas tempranas del ciclo de vida y suele ser adecuado para proyectos largos de misión crítica. **Desventajas:** 1. Es un modelo que genera mucho trabajo adicional. 2. Al ser el análisis de riesgos una de las tareas principales, exige un alto nivel de experiencia y cierta habilidad en los analistas de riesgos.	**Ventajas** 1. Una de las principales ventajas que ofrece este modelo es que no hace falta que los requisitos estén totalmente definidos al inicio del desarrollo, sino que se pueden ir refinando en cada una de las iteraciones. 2. Igual que otros modelos similares tiene las ventajas propias de realizar el desarrollo en pequeños ciclos, lo que permite gestionar mejor los riesgos, las entregas, entre otras cosas. **Desventajas:** 1. Pueden surgir problemas relacionados con la arquitectura. al no tener los requisitos totalmente definidos.

CAPÍTULO IV

METODOLOGÍAS ÁGILES

El marco teórico de este proyecto tiene su base sobre las metodologías ágiles. Puntualmente en dos metodologías: (a) Lean Software Development y (b) Scrumban: que combina el enfoque Scrum y el enfoque Kanban de la metodología Lean.

Como fue señalado a lo largo del presente documento, existen diferentes metodologías ágiles que han podido ayudar a la gran problemática de Gestión de Proyectos a lo largo de estos últimos años. Sin embargo, estas metodologías adoptadas dependen en gran medida de muchas variables que pueden impactar directamente en su implementación. La forma en la que se implementa cada una de ellas, es la clave para poder garantizar mejoras notables en el corto plazo y la sustentabilidad necesaria para el largo plazo.

El Manifiesto Ágil

Según INTECO (2009), las metodologías ágiles son una familia de metodologías, no un enfoque individual de desarrollo de software. En 2001, 17 figuras destacadas en el campo del desarrollo ágil (llamado entonces, metodologías de peso ligero), se juntaron en la estación de esquí Snowbird, en Utah, para tratar el tema de la unificación de sus metodologías. Crearon el manifestó ágil, ampliamente considerado como la definición canónica del desarrollo ágil.

Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente, que responda a los cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de

desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

Tras esta reunión se creó The Agile Alliance, una organización sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida fue el Manifiesto Ágil, documento que resume la filosofía "ágil".

Extreme Programming (XP)

La programación extrema (Extreme Programming XP), es una metodología ágil[2], formulada por Kent Beck y dirigida a equipos de desarrollo de software de pequeño a mediano porte, donde los requisitos son vagos o cambian rápidamente. El nombre de XP se debe a que se toman los principios y prácticas de sentido común y se llevan al extremo. (Beck, 1999).

Se puede considerar a la programación extrema, como la adopción de las mejores metodologías de desarrollo, de acuerdo a lo que se pretende llevar a cabo con el proyecto y aplicarlo de manera dinámica durante el ciclo de vida del software.

Para XP es importante la capacidad de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto, en vez de intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después, en controlar los cambios en los requisitos.

Por otra parte, vale la pena destacar que XP está centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo. XP se basa en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los

[2] Antes del 2001, la denominación para las metodologías Ágiles era metodologías ligeras.

participantes, simplicidad en las soluciones implementadas y coraje o valentía para enfrentar los cambios.

Elementos de la metodología

Los elementos de la metodología XP son: (a) las historias de usuario y (b) los distintos actores que asumen una serie de roles dentro del ciclo de desarrollo. Observe a continuación, un gráfico con la información detallada de estos elementos (ver gráfico 7).

Historias de usuarios

- Es la técnica utilizada para especificar los requisitos del software.
- Se trata de tarjetas de papel en las cuales, el cliente describe brevemente las características que debe poseer el sistema, sean requisitos funcionales o no funcionales.
- Estas historias de usuarios utilizan un formato determinado para describir de manera objetiva y concreta la necesidad del cliente y poseen un valor correspondiente al peso que simbolizan para el negocio.
- Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarlas en unas semanas.
- Las historias de usuario se descomponen en tareas de programación y se asignan a los programadores para ser implementadas durante una iteración.

Roles

- **Programador**: el programador desarrolla el código del sistema y escribe las pruebas unitarias.
- **Cliente**: escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración, centrándose en aquellas historias de usuarios que agregan mayor valor al negocio.
- **Encargado de pruebas (tester**): ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para las pruebas.
- **Encargado de seguimiento (tracker**): proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- **Entrenador (coach)**: es el responsable del proceso global. Debe proveer guías al equipo, de forma que se apliquen las prácticas XP y de que se siga el proceso correctamente.
- **Consultor**: es un miembro externo del equipo, con un conocimiento específico en algún tema en particular y puede ser necesario en caso de que surjan problemas específicos en el proyecto.
- **Gestor (big boss)**: es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente, creando las condiciones adecuadas. Su labor esencial es de coordinación.

Gráfico 7. **Elementos de XP**

Proceso XP

A continuación, se presenta el ciclo de desarrollo en XP indicado por Pressman (2006) (ver gráfico 8).

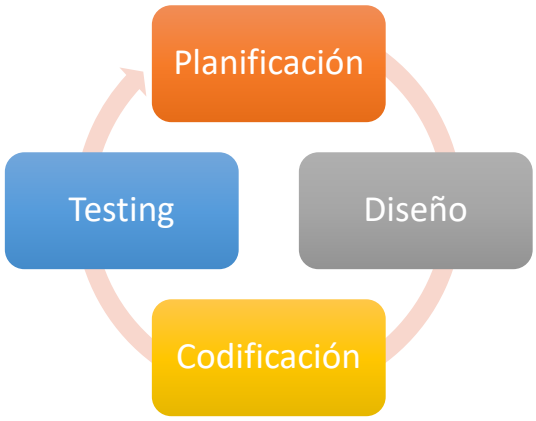


Gráfico 8. **Proceso XP (Pressman, 2006)**

En la fase de planificación, se crean un conjunto de historias de usuario escritas por el cliente con una representación numérica del valor asociado al negocio. Este valor describe el peso de esa especificación en representación a otras especificaciones descriptas en historias de usuario. Se seleccionan las historias de usuario que se pretenden desarrollar en la iteración y se continua a la siguiente fase. En la fase de diseño, se crea una representación simple de las historias de usuario seleccionadas para ser trabajadas en la iteración y se verifica que cada historia de usuario disponga de un test de aceptación. Este test validará que el comportamiento de la historia de usuario corresponde al esperado. La siguiente fase es la fase de codificación, en la cual se recomienda comenzar codificando casos de pruebas o test unitarios que validarán el código desarrollado (este punto se explicará con mayor detalle más adelante). Este desarrollo o codificación es recomendada que sea realizada en pares, utilizando una sola estación de trabajo. Como última fase del proceso, se encuentra la fase de testing, en la cual se realizan diferentes pruebas de manera manual y también puede ser automática, que ayudan a proveer una rápida retroalimentación.

En todas las iteraciones que se realizan en XP, tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que

el estimado, ya que se puede incurrir en pérdida de la calidad del software, o no se cumplirán los plazos. De la misma forma, el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse de que el sistema aporte el mayor valor posible al negocio.

Prácticas de XP

La principal suposición que se realiza en XP, es la posibilidad de disminuir la mítica curva exponencial del costo de cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas:

- **El juego de la planificación**. Hay una comunicación frecuente entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.
- **Entregas pequeñas**. Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión constituye un resultado de valor para el negocio. Una entrega no debería tardar más de 3 meses.
- **Metáfora**. El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema. Por ejemplo, para un sistema de cálculo de prestaciones sociales se podría establecer la siguiente metáfora: “el sistema es una hoja de cálculo”. En este sentido, la comunicación se establece en función de la metáfora definida, compartiendo el mismo vocabulario durante el desarrollo del sistema.

- **Diseño simple.** Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
- **Pruebas.** Como se indicó en el proceso de XP, la producción de código está dirigida por las pruebas unitarias. Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.
- **Refactorización (refactoring).** Es una actividad constante de reestructuración del código, con el objetivo de evitar duplicación del código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar cambios posteriores. Se mejora la estructura interna del código, sin alterar su comportamiento externo.
- **Programación en parejas**. Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas, tales como: menor tasa de errores, mejor diseño, mayor satisfacción de los programadores, entre otros. Esta práctica se explicará más en profundidad, en los apartados sucesivos.
- **Propiedad colectiva del código**. Cualquier programador puede cambiar cualquier parte del código, en cualquier momento.
- **Integración continúa.** Cada pieza de código es integrada en el sistema una vez que esté lista. De esta forma, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
- **40 horas por semana**. Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.
- **Cliente in-situ**. El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor al negocio y los programadores pueden resolver de manera más inmediata, cualquier duda asociada. La comunicación oral es más efectiva que la escrita.

- **Estándares de programación**. XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada, debido a que se apoyan unas en otras. La mayoría de las prácticas propuestas por XP no son novedosas, sino que en alguna forma ya habían sido propuestas en Ingeniería de Sistemas y de Software, e incluso demostrado su valor en la práctica. Por ejemplo, las prácticas comunes con Scrum. El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo. Un ejemplo de una práctica existente previamente y adaptada para su uso en XP es la integración continua, ya que ésta fue utilizada por IBM para el desarrollo del OS/360 en los años 60.

Valores de XP

Según Beck, (1999) los valores básicos de XP son: (a) comunicación, (b) simplicidad, (c) retroalimentación y (d) coraje.

1. Comunicación: Este es el primer valor de XP. La mayoría de los problemas con los proyectos se relacionan con la comunicación, algunos ejemplos son: (a) un programador no comunica acerca de un cambio fundamental en el diseño; (b) un programador no realiza la pregunta correcta, por lo que se omite una decisión de dominio crítico; (c) un administrador no realiza la pregunta correcta a un programador, y está mal informado del progreso del proyecto. Hay muchas circunstancias que conducen a interrupciones en la comunicación, por ejemplo: (a) cuando un programador comunica malas noticias a un administrador, y el gerente,

de cierta forma castiga al programador, (b) cuando un cliente comunica a un programador algo importante, pero el programador parece ignorar la información. XP tiene como objetivo mantener el flujo de las comunicaciones adecuadas mediante el empleo de muchas prácticas, que no pueden llevarse a cabo sin comunicarse correctamente. Son prácticas que tienen sentido a corto plazo, como la unidad de pruebas, la programación en parejas, y la estimación de tareas. Estas prácticas mejoran la comunicación entre los programadores, los clientes y los gestores involucrados con el desarrollo del sistema. El trabajo del entrenador XP es notar cuando las personas no se están comunicando y reintroducirlos. (Beck, 1999).

2. Simplicidad: en XP “se apuesta a que es mejor hacer una cosa simple hoy y pagar un poco más mañana para cambiarlo si lo necesita, que no hacer una cosa más complicada hoy que no se puede utilizar de todos modos.” En este sentido, el entrenador debe guiar al equipo para buscar la solución más simple posible, que funcione correctamente y se pueda adaptar a los cambios requeridos. Por otra parte, se establece una relación de mutuo apoyo entre la simplicidad y la comunicación. Cuanto más se comunica, más claro se puede ver exactamente lo que hay que hacer y se tiene más confianza sobre lo que realmente no necesita hacerse. Cuanto más simple sea el sistema es, cuanto menos se tienen que comunicar, lo que conduce a una comunicación más completa, especialmente si se puede simplificar el sistema lo suficiente como para requerir un menor número de programadores.
3. Retroalimentación (feedback): la Retroalimentación permite obtener información concreta sobre el estado actual del sistema, funcionando en diferentes escalas de tiempo: (a) retroalimentación en la escala de minutos y días: en esta escala los programadores escriben pruebas unitarias para toda la lógica en el sistema con posibilidades de falla, obteniendo minuto a minuto retroalimentación concreta sobre el estado de su sistema; (b) retroalimentación en la escala de semanas y

meses: en esta escala, los clientes y los encargados de pruebas realizan las pruebas funcionales para todas las historias (piense en "casos de uso simplificados"), implementadas por el sistema. De esta forma, se obtiene información concreta sobre el estado actual de su sistema, que será usada por los clientes cada dos o tres semanas para verificar si la velocidad general del equipo coincide con el plan, y para realizar los ajustes correspondientes. El sistema se implementa en producción tan pronto como es funcional, permitiendo verificar lo que el sistema es realmente en su campo de acción, y descubrir lo que mejor se puede aprovechar de él. Una de las estrategias para implementar esta “producción temprana” consiste en que, en el proceso de planificación, el equipo desarrolle e implemente las historias más valiosas en producción, tan pronto como sea posible. Esto les da a los programadores información concreta acerca de la calidad de sus decisiones. Finalmente, la retroalimentación concreta trabaja en conjunto con la comunicación y la simplicidad. Cuantos más comentarios se dispongan, más fácil es comunicar. Si alguien tiene una objeción en relación a una cierta porción de código o módulo en particular que fue desarrollada, y entrega un caso de prueba que lo invalida, sin duda que tiene un valor superior a mil horas de discusión.

4. Coraje o valentía: este valor está asociado con decisiones que se deben tomar durante el ciclo del desarrollo de Software. Por ejemplo, es necesario tener valentía para deshacerse del código, si éste al final del día está un poco fuera de control. Posiblemente se puedan salvar los casos de prueba, si la interfaz que se ha diseñado es adecuada, pero también puede suceder que se requiera empezar de nuevo, o hasta también evaluar nuevas alternativas de diseño. Otro ejemplo de este valor es que alguien del equipo tenga una idea fuera del padrón o un tanto arriesgada, pero que la misma pueda reducir la complejidad de todo el sistema y, por lo tanto, se tenga el coraje para implementarla en producción. La comunicación apoya al coraje, ya que abre la posibilidad de asumir riesgos más altos, haciendo experimentos que

tenga una recompensa mayor. La simplicidad apoya al coraje, ya que puede se puede ser mucho más valiente con un sistema simple que con uno complejo. Por último, la retroalimentación concreta apoya al coraje, porque se dispone de una mayor seguridad al intentar la reconstrucción del código.

Principios básicos de XP

Los principios sirven de puente entre los valores y las prácticas, permitiendo tomar decisiones en un momento determinado. Los principios básicos en XP, son catorce (14) según Beck (1999) y se listan a continuación.

1. Humanidad: el software lo desarrollan personas.
2. Economía: asegura el valor económico de lo que haces.
3. Beneficio mutuo: busca prácticas que beneficien a todos, ahora.
4. Auto-similitud: aplica patrones conocidos a múltiples problemas.
5. Mejora: excelencia en el desarrollo a través de la mejora. Busca perfeccionar, no busca lo perfecto.
6. Diversidad: dos ideas sobre un diseño presentan una oportunidad, no un conflicto.
7. Reflexión: los buenos equipos piensan cómo y por qué hacen el trabajo, no esconden errores.
8. Flujo: entrega continua de valor.
9. Oportunidad: los problemas son oportunidades de cambio.
10. Redundancia: los elementos complejos deberían estar duplicados, las partes más complicadas se deben mirar dos veces.
11. Fallo: un fallo no es un desperdicio, si es que el mismo sirve para aprender algo.
12. Calidad: es necesario garantizar una excelente calidad durante el desarrollo del proyecto, para evitar pérdidas de tiempo y dinero, y el riesgo de que el proyecto fracase.

13. Pequeños pasos: ¿qué es lo menos que puede ser realizado, que sea reconocible y que vaya en la dirección correcta?
14. Responsabilidad aceptada: el que acepta algo, se hace responsable de ello.

Actividades de XP

Según Beck (1999) XP describe cuatro (4) actividades que se llevan a cabo dentro del proceso de desarrollo de software. Estas son: (a) codificar, (b) probar, (c) escuchar y (d) diseñar.

1. Codificar: el desarrollo de código o la codificación, es la única actividad de la cual no se puede prescindir. El código es la mejor manera para comunicarse de manera clara y concisa, y generar aprendizaje, ya que el código no puede ser influenciado por el poder y la lógica de la retórica. El código fuente, se puede utilizar para diversos propósitos dentro de la ingeniería de Software. Por ejemplo, puede usarse para comunicar, ya que expresa la intención táctica, describe los algoritmos, apuntando a los puntos de posible expansión futura y la contracción. De esta forma, un programador que trate con un problema de programación complejo y encuentre difícil explicar la solución al resto, podría codificarlo y usar el código para demostrar lo que quería decir. El código, también se puede utilizar para expresar las pruebas, ensayos que tengan como objetivo probar el funcionamiento del sistema y proporcionar una valiosa especificación operativa del sistema, en todos los niveles.
2. Probar: en XP se afirma que no se puede estar seguro de que un módulo funciona, si éste no es probado. De igual manera, no se puede estar seguro si lo que se ha desarrollado, es el objetivo al cual se quería llegar. Por ello, XP usa dos tipos de prueba: (a) pruebas unitarias: son pruebas automatizadas que prueban el código. El programador intentará escribir todas las pruebas en las que pueda pensar para

validar el código que está escribiendo. Si todas las pruebas se ejecutan satisfactoriamente entonces el código está completo; y (b) pruebas de aceptación: éstas son basadas en los requisitos dados por el cliente, para convencerse de que el sistema en su conjunto, funciona de la manera en la que se espera que trabaje.

3. Escuchar: los programadores no saben necesariamente todo sobre sobre el negocio del sistema que están desarrollando. La función del sistema está determinada por el lado del negocio y todas sus reglas internas. Para que los programadores entiendan cual debe ser la funcionalidad del sistema, deben escuchar al negocio y por supuesto, también escuchar las necesidades de los clientes. Al mismo tiempo, tienen que intentar entender la problemática del negocio y proveer a los clientes retroalimentación sobre el problema, para mejorar el propio entendimiento del cliente sobre el problema.
4. Diseñar: desde el punto de vista de la simplicidad, se puede decir (erróneamente) que el desarrollo de sistemas puede prescindir de diseñar, dejando solo actividades como codificar, probar y escuchar. Si estas tres actividades se desarrollan de forma correcta, el resultado debería ser un sistema que funcione satisfactoriamente. Sin embargo, esto no ocurre en la práctica de esa forma. Si bien en un principio, es posible continuar sin diseñar, en algún momento dado, el sistema se vuelve muy complejo y las dependencias dentro del mismo dejan de estar claras, por lo cual se presentan fallas. Es por eso que el diseño es fundamental para documentar y entender todos los componentes del sistema, creando una estructura de diseño que organice la lógica del mismo. Crear diseños consistentes evitan perdidas de dependencias dentro de un sistema.

Kanban

La palabra Kanban significa “tablero” o “tarjeta visual” en japonés. Éste se refiere a un sistema de tarjetas que ayuda a visualizar el estado en el que está cada

actividad o tarea. Fue concebido para las líneas de producción de Toyota en la década del setenta y hace algunos años, fue adoptado en el mercado de TI con el fin de aplicarlo para el desarrollo de software.

Su objetivo es organizar y gestionar de manera general, la forma en la que se van completando las tareas. En los últimos años, se ha utilizado en la gestión de proyectos de desarrollo de Software, asociado a la metodología Lean y ha logrado un amplio grado de aceptación y utilización dentro del mercado.

Este sistema de tarjetas japonés, es básicamente un tablero que refleja los estados de las actividades o tareas que se deben realizar en un flujo de trabajo determinado. Dichas actividades se visualizan en el tablero mediante tarjetas y a su vez, las mismas personas que trabajan en el proyecto, moverán y trasladarán dichas tarjetas a lo largo del flujo de trabajo que se encuentra representado en el tablero. Al poder visualizar el flujo de trabajo, se muestran rápidamente los logros y problemas del proceso, identificando diferentes riesgos, problemas que pueden generar cuellos de botellas en el flujo de ejecución. Con esta herramienta los equipos logran visualizar rápidamente el estado general de actividad y permite enfocarse en terminar las tareas que tienen asignadas y no acumular tareas ya iniciadas. (Riquelme, 2011).

Kanban no es un método de Administración de Proyectos, ni tampoco un Software de manejo de Ciclo de Vida de Desarrollo. Es un enfoque para cambiar la administración, un marco de trabajo (o framework) para catalizar el cambio en una organización.

Éste utiliza la limitación del “Trabajo en Curso” o “Work In Progress” (WIP), como mecanismo de control para demostrar cuantas actividades por estado pueden ser trabajadas y de esta forma, incentivar las discusiones de cambio. Estas discusiones sobre mejoras son objetivas, gracias a que la visualización, mediciones y claridad de las políticas y los modelos de Lean, Theory of Constraints (Teoría de Restricciones), y las enseñanzas de W. Edwards Deming, le permite al equipo analizar científicamente

sus problemas y proponer soluciones. Cabe aclarar que este límite de WIP debe ser respetado, ya que sino las tareas pueden estancarse y esto puede complicar el flujo de trabajo en su totalidad.

Kanban tiene un enfoque más evolutivo que revolucionario para asumir el cambio, ya que comienza con un proceso que ya existe, mapeando el flujo, visualizándolo, y limitando el WIP para crear un sistema Pull. De esta manera, se mantienen intactos los roles existentes, responsabilidades, cargos, y prácticas. Los únicos cambios son aquellos que se realizan en la interfaz con quienes los integrantes se relacionan en el trabajo, por ejemplo: gerencias y operaciones. Cualquier otro cambio necesario, será específicamente realizado a los fines de evitar impactos en la jerarquía social, o también para evitar invocar una respuesta emocional defensiva por parte de los integrantes afectada. Es por eso que se afirma que Kanban provoca un cambio evolutivo en los equipos, presumiendo (no a simple vista, sino gradualmente), una Optimización de Procesos (Kaizen). Así, mientras la organización madura en sus capacidades, esto se transforma en grandes cambios administrados (Kaikaku). Se ha observado que eventos Kaizen progresivos llevan a una madurez organizacional mejorada, y a provocar niveles de cambio de Kaikaku más dramáticos. Kanban está diseñado como un enfoque que permitirá personalizar y evolucionar un proceso existente, sin importar donde se encuentre el proceso.

Kanban conlleva 5 propiedades principales para catalizar el comportamiento emergente de la evolución: (a) visualizar el Workflow, (b) limitar el Work-In-Progress, (c) medir el Flujo, (d) explicitar las Políticas de los Procesos y (e) usar Modelos para Evaluar Oportunidades de Mejoras. Estas propiedades representan las 5 prácticas que deben estar presentes para que el enfoque Kanban funcione.

El proceso del equipo para el desarrollo de software y la administración de proyectos siempre será único, y con el tiempo es casi inevitable que sea adaptado a la medida del equipo, optimizado para darle valor al flujo de trabajo. Dicho proceso

involucra medición de riesgo, capacidades y habilidades del equipo, demanda del cliente, identificación y corrección de cuellos de botella, como así también, variaciones de distintas índoles que pueden afectar al equipo y a sus miembros.

Kanban impulsa mecanismos para simplificar la coordinación de los elementos de todo un sistema. Por ejemplo, la combinación de la visualización y de la limitación del WIP (sistema Pull), permiten una interfaz simplificada con las gerencias. Como resultado, la mayoría de las organizaciones que adoptan Kanban no necesitan el concepto de "Product Owner" (Dueño del Producto), y pueden fácilmente enfrentar múltiples canales de entrada, encolando requerimientos. Las "Stand-up Meetings" (Reuniones de Pie) diarias se ha demostrado que son muy efectivas. La razón es porque el equipo confía implícitamente en que el trabajo que se muestra en la visualización se está realizando. No es necesario usar estas reuniones para reforzar el compromiso personal, por lo que las reuniones se enfocan en el trabajo, y no en la gente. Los equipos iterarán sobre los tickets de trabajo, en vez de los miembros del equipo, obviando las típicas tres preguntas sobre el estado de las actividades. Equipos más maduros reducen la discusión sólo al trabajo que está bloqueado o tiene algún problema, enfocándose sólo en las excepciones, en vez de hacerlo en el trabajo que está avanzando con normalidad.

En Kanban se observan dos (2) tipos de distribuciones de equipos, que conforman diferentes estrategias (ambas válidas), para organizar su trabajo: (a) equipos pequeños, que aprovechan la ventaja de pocos integrantes para eliminar costos de coordinación, lo cual genera buenos resultados en su trabajo en conjunto. (b) equipos grandes, compuestos de 20, 30, e incluso 50 personas. Al disponer de equipos con gran cantidad de miembros, la visualización del workflow generalmente implica múltiples filas (o swimlanes) que representan los distintos flujos de desarrollo presentes en el mismo proyecto. Esta sumatoria de swimlanes conforman el proyecto en su totalidad y a su vez, cada una de ellas consigue ordenar el tablero de forma limpia y consistente

para el corriente uso de todos sus integrantes. Muchos de estos integrantes, pueden ser asignados a un swimlane como miembros permanentes, mientras que otros miembros (generalmente de habilidades especiales o miembros con gran experiencia), pueden cambiar entre distintos swimlanes de forma de apoyar a otros equipos en situaciones en la que sea requerido. Como resultado, se obtiene un uso más efectivo y eficiente de los recursos disponibles, y como consecuencia, las entregas se realizan cada vez más rápidas. (Riquelme, 2011).

Según Pérez (2011), el enfoque Kanban no prescribe roles. Tener un papel asignado y las tareas asociadas a dicho papel crean una identidad en el individuo. Por lo tanto, pedir que adopten un nuevo papel o un nuevo puesto de trabajo, puede ser entendido como un ataque a su identidad. Es decir, esto podría generar una notable resistencia al cambio. Kanban trata de evitar esa resistencia emocional, entiende que la ausencia de papeles es una ventaja para el equipo.

Visualizar el trabajo y las fases del ciclo de producción o flujo de trabajo

Kanban se basa en el desarrollo incremental, dividiendo el trabajo en partes. Uno de los principales aportes es que utiliza técnicas visuales para ver la situación de cada tarea, y se representan en tableros llenos de post-it. Los post-it suelen tener información variada, si bien, aparte de la descripción, necesitan tener la estimación de la duración de la tarea. (ver gráfico 9).

El flujo de trabajo o las fases del ciclo de producción se deben decidir según el caso. El tablero tiene tantas columnas como estados (o fases) por los que puede pasar la tarea (ejemplo: en espera de ser desarrollada, en análisis, en diseño, etc.).

Gráfico 9. **Tablero Kanban**

El objetivo de este tipo de visualización es que quede claro el trabajo a realizar, en qué está trabajando cada persona, que todo el mundo tenga algo para hacer y también, tener clara la prioridad de las tareas.

Determinar el límite del trabajo en curso (Work In Progress)

Quizás una de las principales ideas de Kanban es que el trabajo en curso (Work In Progress) debe estar limitado. Es decir, que el número de tareas que se pueden realizar en cada fase debe ser algo conocido por todo el equipo. Independientemente de si es un proyecto grande o pequeño, simple o complejo, hay una cantidad de trabajo óptima que se puede realizar sin sacrificar eficiencia. Por ejemplo, puede ser que realizar diez tareas a la vez nos lleve una semana, pero hacer dos cosas a la vez nos lleve sólo unas horas, lo que nos permite hacer quince tareas en la semana.

En Kanban es necesario definir cuantas tareas, como máximo, pueden realizarse en cada fase del ciclo de trabajo (ejemplo, como máximo 4 tareas en desarrollo, como máximo 1 en pruebas, etc.). A ese número de tareas, se le llama límite del “work in

progress". A esto, se agrega otra idea muy razonable, que es que, para empezar con una nueva tarea, alguna otra tarea previa debe haber finalizado.

Por ejemplo, en la anterior figura de ejemplo el número límite del "work in progress" para las pruebas es 1. Es decir, solo una tarea por vez puede ser trabajada en esa columna.

Medir el tiempo en completar una tarea (Lead Time y Cycle Time)

El tiempo que el equipo tarda en completar cada actividad del tablero Kanban se lo denomina "lead time". Éste, cuenta desde que se realiza el pedido, hasta que se hace la entrega. La métrica más distinguida del enfoque Kanban es el "lead time", en español "tiempo o plazo de entrega", que representa el tiempo desde que ha ingresado el trabajo a la fila, hasta el momento en el mismo que sale de la fila terminado. Regularmente este indicador se suele utilizar o combinar también, con otro indicador que se denomina "cycle time", que en español significa tiempo de ciclo, y se refiere al tiempo completo del ciclo de trabajo. Es decir, mide el tiempo desde que el trabajo comienza a ser trabajo, hasta el momento en el que se completa y se entrega.

En palabras más simples, podemos definir que el "lead time" mide lo que los clientes perciben, ven o esperan, y con el "cycle time", se calcula el rendimiento del proceso. A continuación, una imagen que simboliza lo aquí expuesto:

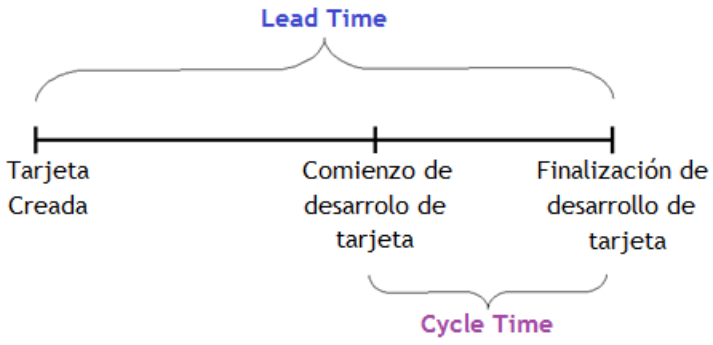


Gráfico 10. **Lead Time y Cycle Time**

Tablero Kanban físico

Si bien los tableros Kanban comúnmente son utilizados por los equipos en formato digital (en pantallas con aplicaciones online), muchos equipos optan por la utilización de tableros físicos. El tablero físico consiste en una pizarra con tarjetas (Kanban) pegadas, con un significado compartido por los miembros de un proyecto. Cada parte del tablero (sintaxis) tiene su propio significado (semántica), las cuales se manejan con ciertos gestos que son soportados por el tablero y los miembros del equipo.

Estos tableros físicos Kanban cumplen con todos los requerimientos básicos como: (a) facilidad de uso, (b) flexibilidad, (c) visibilidad, (d) concurrencia de múltiples usuarios (incluso un equipo entero). Pero también, al ser básicamente un tablero con papeles, y dados los requerimientos más modernos, tiene las siguientes desventajas: (a) portabilidad, (b) dificultad para generar reportes (en general), (c) incapacidad de generar reportes de forma digital, (d) persistencia de la información, (e) historia, (f) cálculo de métricas relevantes: tiempo promedio de una tarea en comenzar, tiempo de ciclo, tiempo de plazo, etc.

Se presenta seguidamente un ejemplo de tablero físico que fue publicado en el año 2010 por la empresa Tupalo, un startup de Viena que. (ver gráfico 10)

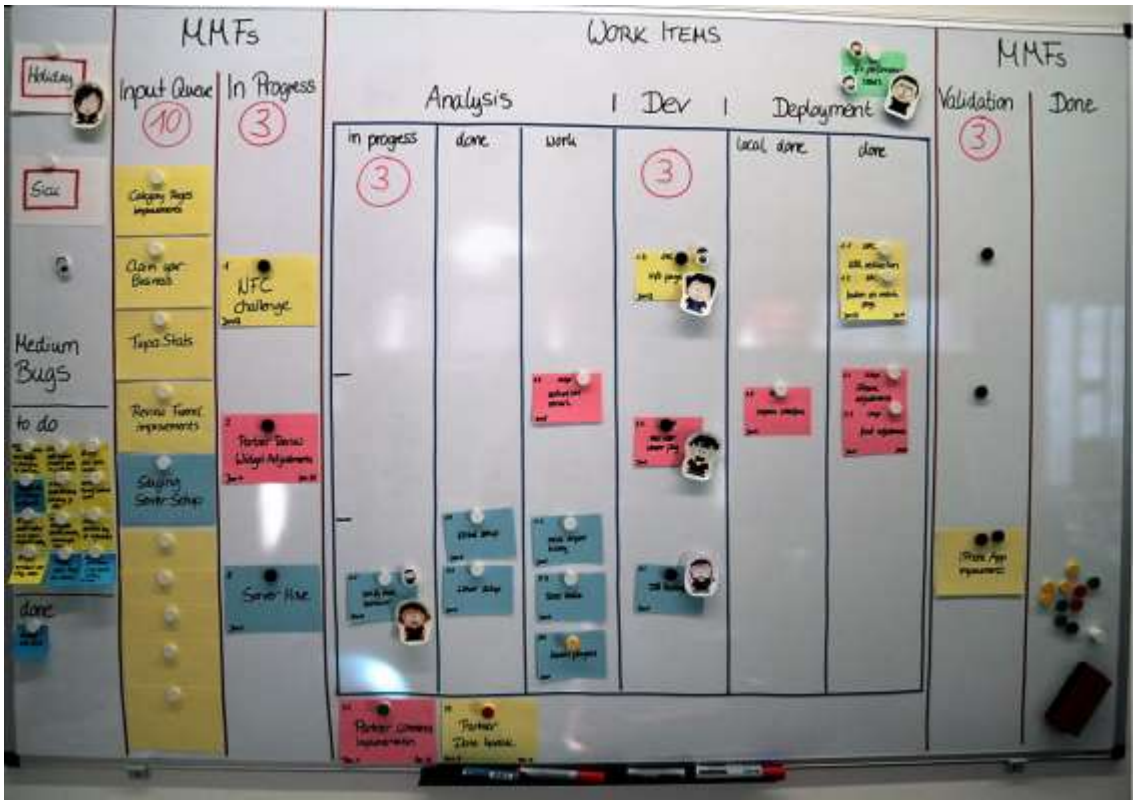


Gráfico 11. **Tablero Kanban en la empresa Tupalo**

A continuación, se detallan las características del tablero, en cuanto a sintaxis, semántica y gestos.

1. Sintaxis: 10 columnas, 8 filas, hojas del mismo tipo, texto en las hojas pegadas.
2. Semántica: (a) cada columna representa un estado: “MMFs Input Queue”, “MMFs In Progress”, “Analysis In Progress”, “Analysis Done”, “Analysis Work”, “Dev”, “Deployment Local Done”, “Deployment Done”, “MMFs Validation”, “MMFs Done”; (b) los papeles son de distintos colores: amarillo, azul o rosado; y (c) los títulos de las filas y columnas son etiquetas con un tipo de letra específico.
3. Gestos: (a) agregar, eliminar y modificar kanbans, filas y columnas; (b) se pueden mover kanbans entre columnas, no entre filas. Cada actividad parte con una fila

asignada y (c) se puede agregar, eliminar y modificar la información en cualquiera de los campos existentes.

Tableros Kanban Virtuales

Los Tableros Kanban Virtuales nacen de la necesidad de modernizar un simple sistema de administración para, por ejemplo, coordinar equipos que no trabajan en conjunto, poder hacer mediciones sobre la información contenida, poder acceder a la información desde cualquier lugar, obtener la historia de un proyecto, entre otros aspectos.

Los Tableros Kanban Virtuales vienen a solucionar muchas de las desventajas de los tableros físicos, pero dependiendo de la solución que se elija, tienen sus propias limitaciones en comparación con los mismos tableros físicos. La mayor ventaja de los tableros físicos, es que su sintaxis (estructura) es independiente de su semántica (significado). Los tableros son lo suficientemente expresivos para que se acomoden a los proyectos, y no los proyectos acomodarse a los tableros. Si cambia la sintaxis, cambia el significado del tablero.

Las herramientas disponibles para tableros virtuales imponen una semántica y definen una sintaxis, evitando que se puedan realizar algunas “jugadas” o personalizaciones, que si se pueden hacer en un tablero físico. No todos los sistemas de tableros Kanban soportan todas las posibles combinaciones de movimientos más utilizadas en los tableros físicos, como, por ejemplo: - Swimlanes (filas) - Atributos en las tarjetas - Ubicación personalizada de partes y componentes del tablero, etc.

En esta documentación se presentará una de las herramientas más utilizadas para tableros Kanban virtuales, denominada JIRA.

Metodología Lean Software Development

La metodología Lean Software Development está dirigida especialmente al desarrollo de sistemas cuyas características varían constantemente. Fue definida por Bob Charette's, a principios de los 90, a partir de su experiencia en proyectos industriales, constituyendo una adaptación para el desarrollo software de las lecciones aprendidas en la industria, en particular, en el sistema de producción automovilista japonesa de Toyota. La metodología establece que todo cambio en el desarrollo software conlleva riesgos, pero si se manejan adecuadamente pueden convertirse en oportunidades que mejoren la productividad del cliente.

Posteriormente, en 2003, se presentó el trabajo de los hermanos Tom y Mary Poppendieck, denominado "Lean Software Development: An Agile Toolkit" y su avance en publicaciones posteriores, entre las que se incluye "Leading Lean Software Development: Results are Not the Point".

A continuación, se expone una descripción de la metodología Lean Software Development detallando sus principios, las personas y los equipos de trabajo, la mejora continua, eliminar el desperdicio, Heijunka – producción nivelada, Just-in-Time y Jidoka.

Principios Lean

La metodología Lean Software Development consta de siete principios dirigidos a gestionar los cambios:

1. Eliminación de todo aquello que no aporte valor al negocio: hacer desaparecer del proceso y el producto todo aquello que no aporta valor al cliente.
2. Conocimiento incremental: según Poppendieck (2003), el desarrollo de Software conlleva un proceso de aprendizaje constante, similar al proceso de crear una receta de cocina y luego ir variando ésta al momento de crear los platos. Por lo tanto, es

necesario crear las condiciones para que el equipo de desarrollo pueda fomentar la creatividad e incrementar constantemente su aprendizaje.

3. Toma de decisiones tan tarde como sea posible: dada la frecuente incertidumbre que rodea la toma de requisitos, lo más aconsejable es retrasar las decisiones tratando de tomarlas con la mayor cantidad de información posible, y siempre adoptando una aptitud previsora ante la certeza del cambio.
4. Liberar funcionalidad tan pronto como sea posible: consecuencia de lo anterior, es necesario disponer de medios que permitan, una vez tomada una decisión, materializarla, sin sacrificar la calidad.
5. Poder del equipo: se basa en el empoderamiento (empowerment), que consiste en dotar del liderazgo suficiente para tomar decisiones, a aquellos en los que reside el conocimiento y realizan el trabajo, evitando aceptar o aprobar algo en otras instancias, que obstaculizan el flujo normal de las actividades.
6. Construcción del producto con integridad: se deben garantizar en la construcción del producto, dos tipos de integridad: (a) integridad conceptual: se refiere a que el producto debe responder a una necesidad del cliente, éste debe percibir el producto como algo coherente, donde los requisitos a los que da solución se observan como un todo cohesionado armónicamente y (b) integridad técnica: la arquitectura planteada para el producto debe ser coherente, usable, que responde a su cometido y a la que se puede dar mantenimiento, adaptar y ampliar.
7. Perspectiva global del proyecto: se debe evitar la tendencia a realizar mejoras locales a favor de un enfoque global. En este sentido señala Poppendieck (2003), que “uno de los problemas más difíciles con el desarrollo de productos, es que los expertos en cualquier área (por ejemplo, base de datos o interfaz gráfica de usuario), tienen una tendencia a maximizar el rendimiento de la parte del producto que representa su propia especialidad, en lugar de centrarse en el rendimiento general del sistema”, esto ocasiona problemas en cuanto a la optimización del Software.

Una situación similar se presenta cuando se fusionan dos organizaciones, ya que cada empresa tiende a maximizar su propio rendimiento, en vez de pensar en un enfoque global. Por lo expuesto, este principio consiste en implementar prácticas que eviten la suboptimización.

Las personas y los equipos de trabajo

Poppendieck (2003) destaca que el componente humano es la piedra angular del éxito en la aplicación de Lean. En este ámbito, aborda una serie de factores que se exponen a continuación.

1. Autodeterminación: basar la transformación en transferir las prácticas en lugar de los principios sobre los que se fundamentan es un error, aunque se consiga realizar la implantación del cambio, es difícil mantener esta transformación. En este sentido, la prioridad es conseguir que las personas crean en el cambio y participen de él.
2. Motivación: el primer elemento motivacional es dar un propósito al trabajo de las personas, por encima de un simple conjunto de tareas, por ello deben entender el propósito de su trabajo, de una forma clara, convincente y alcanzable. Cuestiones como facilitar la comunicación entre el equipo y el cliente, dejar al equipo alcanzar sus propios compromisos, ayudan a dar sentido al trabajo que se está realizando. En estas circunstancias, la gestión tiende a convertirse principalmente en un facilitador, detectando las necesidades del equipo para realizar correctamente su trabajo. Los sentidos de pertenencia a un grupo en particular, de confianza, de competencia y de progreso deben estar presentes para conseguir un equipo motivado. Es destacable la referencia en el apartado motivacional que realiza Poppendieck, considerando que, para una conciliación de la vida familiar y laboral, es recomendable la moderación frente al heroísmo, reservando los esfuerzos adicionales para las emergencias.

3. Liderazgo: se buscan líderes más que simples gestores, que hagan frente al cambio, marcando el camino a seguir, alineando y motivando al equipo. Un liderazgo basado en el respeto del equipo hacia el líder, por su profundo conocimiento del cliente y de los aspectos técnicos, más allá de una autoridad concedida.
4. Experiencia: facilitar que los equipos adquieran y compartan su experiencia, que experimenten de manera autónoma, tolerando los errores durante el proceso de aprendizaje y fomentando la transmisión del conocimiento, especialmente el tácito. Por otra parte, la Ingeniería de Sistemas necesita lidiar con una multitud de áreas de conocimiento especializado, tanto a nivel técnico como a nivel de negocio. Este último es tan o más importante como el primero, dado que, para conseguir dar valor al cliente, es necesario conocer primero su negocio en general y sus necesidades concretas en particular. Por esta cuestión, es importante facilitar la creación de comunidades de expertos, especialmente en aquellas áreas críticas para el éxito de la organización. Estas comunidades, además, deben jugar un papel importante en cuanto a la decisión de los estándares a seguir en el área de experiencia que cubren.

Uno de los factores más importantes a tener en cuenta es la necesidad de crear equipos de trabajo, en lugar de grupos. El desarrollo de sistemas implica la resolución de problemas a diario, decisiones complejas que afectan más allá del trabajo de la persona que las realiza. De esta forma, disponer de un conjunto de personas cohesionadas cuya experiencia y conocimientos aporte nuevas perspectivas a los problemas, es una base fundamental para obtener buenos resultados. Para crear estos equipos es necesario establecerles desafíos y metas comunes. Los miembros que los componen, dependen unos de otros y por tanto debe existir el compromiso del trabajo conjunto para el logro de la meta u objetivo común.

Mejora continua- Kaizen

El Kaizen es un término japonés que se refiere a la práctica de la mejora continua de los procesos por medio de pequeños cambios incrementales. El kaizen reconoce en su filosofía que cualquier empresa tiene problemas, y estos deben ser detectados, eliminados y prevenidos. Fundamentalmente, se pretende alcanzar los objetivos mediante tres herramientas fundamentales: (a) la estandarización de los procesos y su constante mejora, (b) la búsqueda permanente de desperdicios y su eliminación, y (c) la organización, orden y limpieza. Estas herramientas constituyen la base para la reducción de costos y tiempos de ciclos, la mejora en la calidad y niveles de seguridad, y un mayor cumplimiento en los niveles de satisfacción para los clientes. De esta manera, la mejora continua en el kaizen persigue mejorar la calidad, los costos, la logística, la satisfacción del cliente, la seguridad y los productos, teniendo siempre en consideración la mejora en los estándares, la continua eliminación de desperdicios (mudas o despilfarros) y la mejor organización, orden y limpieza de los elementos y espacios de la empresa.

F. Huda y D. Preston (1992), publican la primera referencia encontrada que reconoce la aplicabilidad de Kaizen a la Ingeniería de Sistemas y de Software, entendido como la mejora continua dentro de un entorno estable de producción.

El papel del componente humano es crítico en el desarrollo de sistemas y Kaizen reconoce esta circunstancia, obteniendo provecho de ella: cambia la perspectiva de la resolución de problemas y lo centra en las personas, buscando soluciones en lugar de culpables y fomentando la búsqueda de los intereses colectivos, en lugar de intereses individuales.

F. Huda y D. Preston (1992), señalan en su artículo que se pueden extrapolar algunas de las experiencias Kaizen, realizadas para empresas de manufactura y a otras disciplinas para resolver algunos de los problemas de la industria del Software. Las conclusiones más relevantes que destacan son las siguientes: (a) el control del proceso

se sugiere como un medio para la construcción de la calidad, mediante el uso de métodos estadísticos que deben ser accesibles a todos los niveles de la jerarquía de la organización. Además, se resalta la importancia del compromiso de gestión a este cambio fundamental en las prácticas de trabajo y (b) se identifica la necesidad de capacitar a las personas y fomentar su participación en la organización, conjuntamente con la necesidad de un alto compromiso por parte de ellas. Asimismo, se precisa enfocar la atención al componente motivacional de las personas que realizan el trabajo.

Poppendieck (2003), propone el uso de eventos Kaizen para la resolución de problemas. Dado un problema crítico, bien definido, los eventos Kaizen son una herramienta que reúne en un equipo, a los representantes de las distintas áreas involucradas en el problema para su resolución. Durante un espacio de tiempo de no más de una semana, trabajarán intensivamente en realizar las modificaciones necesarias en los procesos para solucionarlo. Tras el evento Kaizen, el equipo se disuelve y sus miembros vuelven a realizar sus labores habituales, y el proceso queda implantado con el cambio.

La proactividad en la búsqueda de la mejora continua a través de todos los aspectos de la vida, sin prejuicios ni culpables, es crítica en la filosofía Kaizen: todo el personal involucrado en el proyecto (desarrolladores, clientes, usuarios, etc.), deben sentirse en confianza para expresar su punto de vista y poder participar en el proceso de mejora continua, disponiendo de los canales adecuados para comunicarse.

Finalmente, señala el autor que, es sumamente importante el alineamiento de toda la organización hacia esta filosofía, en lo que respecta a la disposición de los medios para esta experimentación constructiva y sin prejuicios, y la constancia de que no todos los experimentos pueden ser un éxito. (Moreno, 2010).

Eliminar el desperdicio

El primer paso para comenzar a aplicar la filosofía Lean es aprender a reconocer los desperdicios o “mudas”. Una “muda” en su concepto más amplio es: cualquier otra cosa distinta a la cantidad mínima de equipos, materiales, partes, espacio y tiempo del trabajador, que son absolutamente necesarios para dar valor al producto.

A continuación, se comparan 7 tipos de desperdicios en el ámbito de la Ingeniería de Sistemas, identificadas por Shingeo Shingo en manufactura (ver cuadro 3).

Cuadro 3. Los 7 desperdicios en el ámbito de la Ingeniería de Sistemas

Contexto de la manufactura	Contexto de la Ingeniería de Sistemas
Inventario	Trabajo sin terminar
Sobreproducción	Características “extras”
Sobre-procesamiento	Reaprendizaje
Transporte innecesario	Cambio de persona asignada a una tarea
Movimientos innecesarios	Cambio de tarea
Esperas	Retrasos
Defectos	Defectos

A continuación, se detallan cada una de las “mudas” en el ámbito de la Ingeniería de Sistemas.

1. Trabajo sin terminar: Es una de las fuentes de desperdicio más peligrosas, ya que un trabajo no se considera como terminado hasta que el sistema no se encuentra en producción y, si no llega a estarlo, puede llegar a suponer la pérdida de toda la inversión realizada. En este sentido, realizando una analogía con una empresa de manufactura, mientras el sistema está en desarrollo se considera parte del inventario de productos que oferta la empresa y hay una “muda” relacionada con el exceso de

inventario, esto se refiere al almacenamiento excesivo de materia prima, producto en proceso o producto terminado, causando mayores plazos de entrega, costos de almacenamiento, etc. La forma de evitar este exceso de inventario es dividir el trabajo en lotes pequeños o iteraciones. Algunos ejemplos de trabajo sin terminar son: (a) documentación sin codificar: documentos de requisitos y de diseño técnico que aún no se comienzan a implementar; (b) código sin consolidar: este concepto guarda relación con las herramientas de control de versiones: deben minimizarse situaciones como código sin subir al repositorio o desarrollos paralelos sobre un mismo recurso que requieren posteriormente proceso de mezcla; (c) código sin probar; (d) código sin documentar (en la medida que requiere) y (e) código sin desplegar: los resultados necesitan ser implementados en producción, evitando posibles demoras.

2. Características “Extras”: Cada línea de código debe responder a una característica que se sabe con certeza que es necesaria y en ese momento, no solo teniendo en cuenta el costo de su desarrollo, sino también el de su mantenimiento en cuanto a tiempo de compilación, posibles errores y complejidad agregada al código en su conjunto. Por otra parte, la inversión realizada en frameworks, en busca del beneficio de las economías de escala, también puede suponer desperdicio. Si no se acaban utilizando, este tipo de adquisiciones (ej. compra a un tercero de una librería para la representación gráfica) o desarrollos dentro de la propia organización, no se amortizan en un escenario de baja reutilización.
3. Re-aprendizaje: el reaprendizaje, por una incorrecta gestión del conocimiento se deriva en rehacer trabajo. Las siguientes cuestiones inciden en el costo agregado por el reaprendizaje: (a) código sin documentar: la documentación justa y necesaria en el momento del desarrollo evita el reaprendizaje en una modificación posterior del mismo; (b) planificación deficiente: si la asignación de tareas a cada miembro del equipo de trabajo no toma en consideración su conocimiento previo, se dan

situaciones de reaprendizaje. Por ejemplo, asignando a un programador A, una tarea codificada por un programador B y viceversa, ambos deberán aprender que realizó su compañero con el costo que eso conlleva, lo cual puede redundar en altos costos; y (c) calidad deficiente: si un error llega hasta el entorno de producción, puede incrementar la dificultad para localizar su causa e implicar el reaprendizaje, incluso si el desarrollador que aborda la corrección es el mismo que realizó el código que lo provoca; (d) tareas en paralelo, o excesivo cambio de tarea: cada vez que se retoma una tarea, existe un costo relacionado al tiempo para entrar en contexto; (e) comunicación / gestión del conocimiento deficiente: si bien cada vez existen más herramientas para facilitar estas cuestiones como, sistemas para compartir el conocimiento, mejores herramientas de búsqueda de información, wikis, etc. La captura efectiva del conocimiento en una organización tiene una dificultad implícita, que pone en evidencia el fuerte contraste del punto de vista occidental frente al oriental. Así mismo, mientras el primero considera que este conocimiento es algo que puede y debe ser escrito, el último considera el conocimiento tácito viene de la experiencia y no del estudio, algo no fácilmente almacenable o procesable por un ordenador y difícil de formalizar y transmitir.

4. Cambio de persona asignada a una tarea: La asignación de personas a más de un proyecto o tareas paralelamente induce este tipo de desperdicio, dado que necesitarán emplear un tiempo para contextualizarse y concentrarse en una tarea antes de comenzar a trabajar en ella, además de las posibles pérdidas por interrupciones en el trabajo (por ejemplo, la resolución de dudas). Como norma general, la forma más eficiente de realizar varias tareas que comparten el mismo equipo de trabajo y que deben finalizar dentro de un mismo plazo, es realizarlas secuencialmente y nunca en paralelo, para evitar desperdicios. Por otra parte, es necesario evitar las reasignaciones de tareas incompletas. El paso de persona en persona provoca pérdida de tiempo y conocimiento. De no ser posible, se necesita

potenciar la comunicación que favorezca y agilice la transmisión del conocimiento tácito. Existen una serie de consideraciones a tener en cuenta para reducir el impacto de los cambios de asignación, aparte de tratar de minimizar estos cambios en la medida de lo posible [26]: (a) formar equipos multidisciplinares que permitan la formación dentro del propio equipo; (b) comunicación fluida: fomentar la comunicación cara a cara, la observación directa, la interacción con prototipos y (c) mostrar avances del trabajo que se está realizando para obtener feedback tan pronto como sea posible.

5. Cambio de tarea: disponer de un equipo específico para dar mantenimiento correctivo reduce el número de cambio de tareas, así como otras estrategias basadas en agrupar este tipo de actuaciones y realizarlas minimizando las interrupciones al resto de desarrollos.
6. Retrasos: es uno de los mayores ladrones de tiempo en el desarrollo de sistemas/software. Estas esperas, además, provocan la reducción del plazo efectivo para producir y entregar el valor al cliente. Para combatir este tipo de pérdidas, es necesario retrasar las decisiones tanto como sea posible, hasta tener la certeza de que son correctas y que aportarán valor al cliente [22]. Algunos ejemplos de retrasos son: (a) retraso del comienzo del proyecto, (b) retrasos en la asignación de recursos al proyecto, (c) retrasos en revisiones y validaciones, (d) retrasos en las pruebas y (e) retrasos en las implantaciones.
7. Defectos: cuanto más tempranamente se identifiquen los defectos, menor será el desperdicio. La práctica recomendada para paliar este tipo de desperdicio es probar inmediatamente, integrar el código frecuentemente y actualizar el sistema en producción a la mayor brevedad posible.

Heijunka – producción nivelada

Heijunka es un término japonés que significa “nivelar”, se usa a escala industrial para nivelar la producción, incrementando el rendimiento global. Las prácticas Heijunka que se implementan son las siguientes: (a) ejecución del trabajo utilizando el sistema pull del almacén; (b) gestión de la carga de trabajo de una sola cola: las peticiones llegan a un almacén en el que se gestionan mediante una única cola y son procesadas de manera secuencial; (c) gestión visual de la carga de trabajo: mediante la gestión visual, los desarrolladores pueden ver el trabajo pendiente en cada una de las operaciones, pudiendo ellos mismos detectar y corregir de manera autónoma, un desequilibrio entre la carga de trabajo y la capacidad en alguna de ellas; y (d) control autónomo de la misma.

Una cuestión importante a tener en cuenta, es que cada uno de los miembros del equipo sólo atiende una única petición a la vez (flujo unitario).

Just-in-Time

Se entiende por Just in Time (JIT) el conjunto de principios y técnicas que permiten a una empresa la producción y entrega de productos en pequeñas cantidades, con plazos de entrega reducidos, y para dar respuesta a necesidades específicas de los clientes, esto es, entregar el producto correcto, en la cantidad correcta y en el plazo correcto.

Reducir los tiempos de producción o tiempo de ciclo (o como se definió anteriormente en inglés, “cycle time”), es una ventaja competitiva crítica en los proyectos de sistemas, por lo que cada vez existen mayores exigencias por parte de los clientes, en cuanto a plazos y también, por la competencia con otras empresas que evolucionan sus propios productos a contrarreloj, algo especialmente intenso en productos web y aplicaciones para móviles. Poppendieck (2003) propone la estimación

del costo del retraso como herramienta, evaluando en términos económicos el importe a invertir en la mejora del tiempo de ciclo y la ventaja económica que repercute de la misma.

Existen cuatro estrategias para reducir los tiempos de ciclo en desarrollo sistemas: tiempo de aprendizaje, tiempo debido al desperdicio, tiempos de espera y tiempos de repetición.

Reducir el tiempo de aprendizaje, capturando el conocimiento del producto. La correcta documentación y el acceso ágil a la misma, mejoran este factor. Una de las formas mayormente recomendadas es mediante el paquete de "Bienvenida al Proyecto", que es orientado a los miembros que se incorporan al equipo. De la misma manera, la estandarización de herramientas y procedimientos a nivel de organización facilita la reutilización del material destinado al aprendizaje.

Reducir el tiempo debido al desperdicio, es una cuestión que se tratará con detalle en el apartado "eliminar el desperdicio".

Reducir los tiempos de espera vinculados al cliente (por ejemplo, la demora en la toma de decisiones, en la resolución de dudas sobre requisitos o el retraso en la aceptación), o vinculadas a la tecnología utilizada (por ejemplo, espera de la ejecución del proceso de compilación y despliegue en el entorno de un sistema). Es por eso que, algunas fórmulas para reducir estos tiempos son: equipos pequeños con responsabilidad compartida, involucrados con el proyecto y con un contacto fluido con el cliente, una comprensión mayor de los requisitos del proyecto y por sobre todo, de los objetivos que persiguen. Ésta, simboliza una clara idea en la línea de las metodologías ágiles.

Reducir el tiempo de set-up: por ejemplo, para la realización de un test de sistema sobre una versión del software, es necesario esperar a que se realice su despliegue sobre el entorno de pruebas.

Reducir los tiempos de repetición: mediante la reutilización (en el amplio sentido) de documentación, código, etc., pasando del paradigma de la codificación, a la del ensamblaje de componentes. Esta es una técnica con otras ventajas adicionales, como mayor calidad y funcionalidades más completas.

Para una mejor gestión de reutilización, se debe crear una infraestructura global de reutilización, conformando dos equipos:

1. Equipo de reutilización de entornos. Éste es responsable de la provisión y reutilización del hardware y plataformas (sistemas operativos, entornos de desarrollo, etc.).
2. Equipo de reutilización de componentes, que es el responsable de la provisión y reutilización de widgets, librerías, plantillas, etc.

Ambos equipos tienen los siguientes objetivos sobre sus áreas de responsabilidad: (a) selección de los reutilizables necesarios para cada proyecto, (b) producción, adquisición, evolución: de aquellos elementos reutilizables de los que no se dispone en la organización o cuya funcionalidad es insuficiente, (c) documentación: práctica que facilite el uso y la transmisión del conocimiento. (d) soporte y mantenimiento: apoyo a los equipos de desarrollo que hacen uso de esta infraestructura, solucionando dudas, realizando correcciones sobre errores detectados en los reutilizables, etc. (e) difusión: de aquella información relevante para los usuarios de los proyectos reutilizables (mejoras, parches, etc.).

Aplicación de la teoría de colas

El uso de la teoría de colas es propuesto por Poppendieck para reducir el tiempo de ciclo. Típicamente, existen departamentos dentro de la organización con tendencia a que se conviertan en cuellos de botella y comiencen a formarse colas de trabajo pendiente.

Una de las maneras de reducir el tamaño de las colas, es reducir el tamaño de los “paquetes” de trabajo, evitando las esperas asociadas a la acumulación del conjunto mínimo de tareas para realizar. Si tomamos, por ejemplo, el departamento de testing, una posible estrategia es solicitar la realización de las pruebas por áreas funcionales, en vez de todo el sistema de una vez.

Otras de las optimizaciones de las colas es eliminar la variabilidad (aumento en el tiempo de servicio para la atención de un cliente de la cola), en este sentido si se reduce el tamaño de los paquetes, se disminuye también la variabilidad. La otra estrategia aplicable es el procesado en paralelo, incrementar el número de agentes que pueden procesar el paquete en la cola, lo que, además, añade tolerancia a fallos en el sistema de producción. De esta forma, si un agente se detiene, la cola no tiene por qué hacerlo.

Por otra parte, en general se recomienda mover la variabilidad hacia el final del proceso, si éste es iterativo (como es el caso de las distintas modalidades de desarrollo iterativo). Es necesario prestar especial atención a la hora de seguir este tipo de recomendaciones, debido a que no son de aplicación inmediata. Así, por ejemplo, si las pruebas de aceptación se consideran el final del proceso en un desarrollo en cascada, en un proceso iterativo preceden la siguiente iteración, de forma que un cuello de botella en esta actividad nos detendría el proceso completo.

En general, el exceso de carga de trabajo, complica de sobremanera la reducción del tiempo de ciclo. En la Ingeniería de Sistemas y de Software, esto no es una excepción. Es por eso que, volviendo al ejemplo, si el equipo de pruebas sufre exceso de trabajo y el tamaño de la cola de atención asociada al mismo aumenta, también se genera un incremento en el tiempo en que el feedback (respuesta de la atención), se envía al equipo de desarrollo que solicitó las pruebas. Si persiste esta dinámica, los equipos de desarrollo crearán paquetes de pruebas de mayor tamaño y se producirá una reducción de la intensidad con las que se realizan las pruebas (equipo de testing). Este

espiral de sucesos se encadena de forma tal que puede producir un fuerte impacto en el tiempo de ciclo, que se degrada exponencialmente y con mayor intensidad. El mismo se aprecia en el siguiente gráfico (ver gráfico 11).

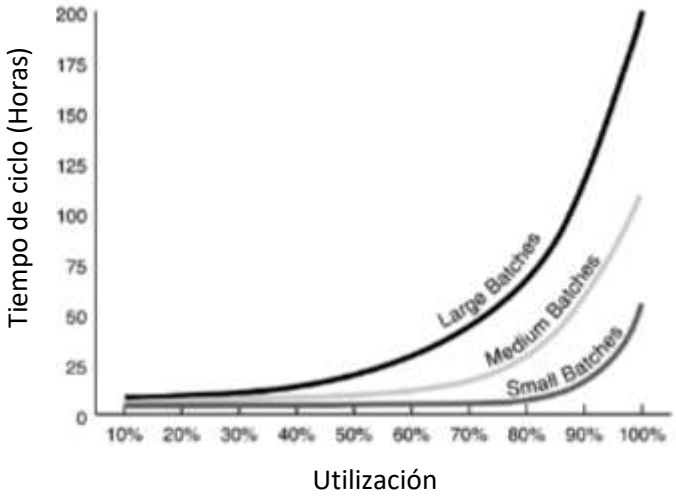


Gráfico 12. **Impacto del tamaño del lote en el tiempo de ciclo**

Por último, Poppendieck hace una referencia a la teoría de restricciones y el uso iterativo de la estrategia de localizar y solucionar el cuello de botella que limita la capacidad de producción, para mejorar gradualmente el desempeño del sistema.

Jidoka

Jidoka es un término japonés que se traduce como “autonomación”, que significa “automatización con un toque humano”. La idea es que los trabajadores no tengan que supervisar constantemente el funcionamiento de las máquinas, sino dotar de mecanismos que hagan este control automático y que el trabajador sólo tenga que intervenir cuando algo va mal, disponiendo de medios para solucionar los problemas en el momento que se producen y evitando que los defectos se propaguen hacia abajo, en el proceso productivo.

En esencia Jidoka, se compone de dos partes:

1. Un mecanismo de detección de problemas (anormalidades o defectos): esta detención podrá realizarse: (a) manualmente: por el trabajador al detectar el error y (b) mediante paradas automáticas: la automatización de la detección y detención, en caso de error, de una máquina o proceso, sin necesidad de intervención humana.
2. Un mecanismo para interrumpir el proceso cuando se detecta uno de estos problemas: se debe garantizar que todo vuelva a la normalidad, identificando y corrigiendo la causa raíz del error y que, con ello, el proceso pueda continuar con calidad.

Los ejemplos más extendidos de la aplicación de Jidoka a la producción de sistemas y se software tratan sobre la automatización de pruebas y sobre el concepto de integración continua. Desde el enfoque Jidoka, las pruebas deberían ser automáticas y los errores no deberían propagarse nunca hasta el final del proceso de desarrollo. Algunos ejemplos de Jidoka son: (a) el marco de trabajo xUnit y (b) la metodología Test Driven Development, facilitando la automatización de las pruebas, que quedan integradas dentro del propio proceso de diseño. Por lo cual no es necesario que ningún grupo de personas quede destinado exclusivamente a labores de inspección, sino que pueden dedicarse a tareas más productivas.

Poka-yoke

Poka-yoke son mecanismos de calidad preventiva, desarrollados para evitar los errores humanos que deriven en condiciones inadecuadas de operación y, por lo tanto, fuentes potenciales de errores. Los siguientes, son ejemplos de errores humanos que se pretenden evitar: errores por olvidos, desconocimiento o inexperiencia, de identificación, voluntario, por despiste, por lentitud, falta de estándares, por sorpresa o intencionales.

Se pueden eliminar los efectos en dos posibles estados:

• Predicción: antes de que ocurran, se trata de diseñar mecanismos que avisen al integrante del equipo cuándo se va a cometer un error para que lo evite (alarma). Es decir, evitar que se pare el flujo del proceso cuando se ha hecho algo mal (parada), o que simplemente se eviten incorporen nuevos elementos que hagan imposible o difícil un determinado error (control).

• Detección: una vez ocurridos los errores, se trata de diseñar mecanismos que avisen cuando se ha fabricado un producto defectuoso (alarma), que se detenga el flujo del proceso si esto ocurre (parada), o que simplemente que se evite que ese producto defectuoso pase a la siguiente etapa (control). Muchas de estas técnicas hacen posible la inspección al 100% incorporando mecanismos económicos.

El concepto de poka-yoke, ha tenido una amplia aplicación en la Ingeniería de Sistemas y de Software, tanto para la prevención como para la detección de problemas. Ya en 1993 en “Writing solid code” de Steve Maguire, se presentan una serie de técnicas para resolver dos preguntas: “¿Cómo puedo detectar automáticamente este error?” y “¿Cómo puedo prevenir este error?

Scrum

Scrum es una de las metodologías ágiles más utilizadas en el mercado, se caracteriza por ser incremental e iterativa. No se trata de un concepto nuevo, sino que ya en 1987, Ikujiro Nonaka y Hirotaka Takeuchi acuñaron a este término para denominar un nuevo tipo de proceso de desarrollo de productos. La palabra “Scrum” proviene del deporte rugby y es básicamente, una estrategia utilizada en la que todos los integrantes del equipo actúan juntos para avanzar la pelota y ganar posición en el campo de juego. Se eligió este nombre por las similitudes que se consideraba que existían entre el juego del rugby y el tipo de proceso que estaba siendo propuesto: adaptable, rápido, auto-organizable y con pocos descansos.

En 1996, Jeff Sutherland y Kent Schwaber formalizaron este enfoque como un proceso para la gestión y control del producto, que trata de eliminar la complejidad en estas áreas, para centrarse en la construcción de software que satisfaga las necesidades del negocio. Es simple y escalable, ya que se aplica o combina, fácilmente, con otras prácticas ingenieriles, metodologías de desarrollo o estándares ya existentes en la organización.

Scrum se concentra, principalmente, a nivel de las personas y equipos de desarrollo que construyen el producto. Su objetivo es que los miembros del equipo trabajen juntos y de forma eficiente, obteniendo productos complejos y sofisticados. Se puede entender Scrum como un tipo de ingeniería social que pretende conseguir la satisfacción de todos los que participan en el desarrollo, fomentando la cooperación y cohesión grupal a través de la auto-organización. Los equipos se guían por su conocimiento y experiencia más que por planes del proyecto formalmente definidos. La planificación detallada se realiza sobre cortos espacios de tiempo, lo que permite una constante retroalimentación, que proporciona inspecciones simples y un ciclo de vida adaptable.

Roles en Scrum

A continuación, se muestran los distintos roles que se manejan con la metodología Scrum (ver gráfico 12).

Roles

- **Product Owner**: es la persona que toma las decisiones, conoce el negocio del cliente y su visión del producto. Se encarga de escribir las ideas del cliente, las ordena por prioridad y las coloca en lo que se denomina "Product Backlog". Debe asistir, al menos, a las reuniones de planificación y de revisión de cada sprint (iteración), al igual que encontrarse en continuo contacto con el equipo para proporcionar detalles sobre las historias de usuario y constante retroalimentación que dirija el desarrollo del sprint.
- **ScrumMaster (Facilitador)**: es el encargado de eliminar todos los riesgos e inconvenientes que impidan que el proceso fluya. Para ello, el ScrumMaster interactua con el equipo, con el cliente y con los gestores para facilitar y ayudar a eliminar dichos riesgos o problemas. A su vez, es el encargado de comprobar que el modelo y la metodología funcione, asegurando que los valores, prácticas y reglas son seguidos por el resto del equipo.
- **Scrum Team (equipo de desarrollo)**: suele ser un equipo pequeño de unas 5 a 9 personas que tienen autoridad para organizar y tomar decisiones para conseguir su objetivo (entrega final del producto). Está involucrado en la estimación del esfuerzo de las tareas del Backlog.
- **Usuario o Cliente**: es el destinatario final del producto y es quien acompaña el progreso del desarrollo. También, es quien puede aportar ideas, sugerencias o necesidades. Su participación es importantísima e imprescindible en esta metodología.
- **Stakeholders:** las personas a las que el proyecto les producirá un beneficio. Participan durante las revisiones del Sprint.
- **Manager:** es quien toma las decisiones finales participando en la selección de los objetivos y de los requisitos.

Gráfico 13. **Roles en Scrum**

Elementos de la metodología Scrum

A continuación, se presentan los elementos de Scrum (ver gráfico 13).

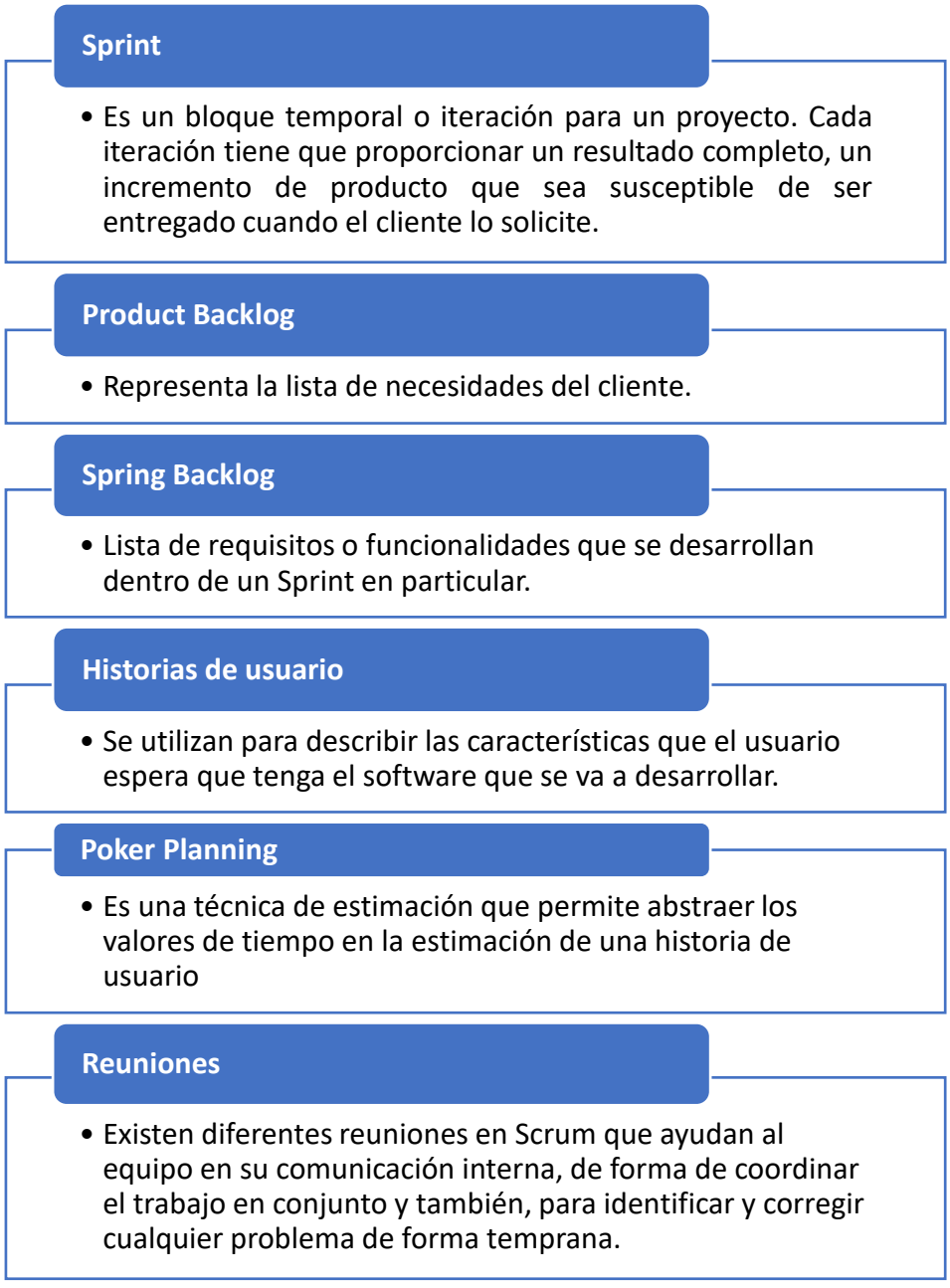


Gráfico 14. **Elementos de Scrum**

Sprint

Como anteriormente se definió, los Sprints son bloques temporales, cortos, fijos y bien definidos en los cuales, el equipo de trabajo desarrolla determinadas tareas con el fin de obtener un resultado que contribuya a la totalidad del producto final. Este resultado se logra entre todo el equipo del proyecto que colabora estrecha y comprometidamente para llegar al final del sprint, con una porción incremental del producto final que se desea entregar al cliente.

El tiempo que dura un sprint es definido por el equipo, quién será el ejecutor dentro de este espacio de tiempo. Comúnmente, el tiempo establecido por los equipos es entre 1 semana y 4 semanas, siendo 2 semanas el valor más utilizado para los Sprints. Es decir que, por ejemplo, el equipo que defina un valor de 2 semanas para cada sprint, tendrá 2 sprints por cada mes del proyecto. Por consiguiente, si el proyecto está planificado para tener una duración de 6 meses, el total de sprints a utilizar para ese proyecto será de 24.

Product Backlog

Representa la lista de necesidades del cliente. Es decir, todos y cada uno de los requisitos que luego de haber sido desarrollados por el equipo de trabajo, conllevan al producto completo en su totalidad y por consiguiente a la terminación del proyecto. Es comúnmente visto también, como una “bolsa de tareas a desarrollar” y puede sufrir diversos cambios en los requisitos que lo integran. Esto es ya que, dichos requisitos necesitan mantenerse actualizados en el Product Backlog, alineándose así con la necesidad del producto. El principal responsable por el Product Backlog es el Product Owner y éste es quien reordena los requisitos según su prioridad, puede crear nuevos requisitos y modificar o eliminar los ya existentes en este backlog.

Sprint Backlog

Lista de requisitos o funcionalidades priorizadas y ordenadas que se desarrollan dentro de un Sprint. En otras palabras, es la lista de requisitos a desarrollar en un sprint en particular y por los cuales el equipo se ha comprometido a disponer de los mismos completados al final del sprint. Estos requisitos son seleccionados desde el Product Backlog y generalmente se encuentran en la parte superior de la lista, ya que fueron priorizados por el Product Owner y son categorizados como los más importantes a desarrollar.

Historias de usuario

Una historia de usuario consiste en frases descriptivas sobre la funcionalidad del objeto a desarrollar en un proyecto, desde el punto de vista del usuario. El título de la misma debe ser corto y conciso (cuánto más conciso mejor). La historia de usuario también incluye los criterios de aceptación expresados desde el punto de vista del usuario. En toda historia de usuario se necesita especificar motivo claro, conjuntamente con la funcionalidad esperada.

- Las historias de usuario son el elemento base que utiliza Scrum para describir las características que el usuario espera que tenga el software que se va a desarrollar. Permiten incorporar tanto cuestiones relacionadas con las funciones del sistema, como también cualquier otro aspecto del mismo (restricciones, rendimiento, etc.). Las historias de usuario se presentan desde la perspectiva del usuario. Es decir, no se describen utilizando una terminología técnica, sino que se escriben utilizando un lenguaje cercano al dominio de la aplicación que se está desarrollando, de forma que sea comprensible por los clientes y por los desarrolladores. Éstas se realizan bajo un mismo esqueleto que centra el foco de las características del producto: (a) primero se determina quién propone la

historia de usuario, ergo, cual es el actor que escribe esa historia de usuario. Por ejemplo, si el proyecto en cuestión se trata de un sistema de ventas y control de stock en un negocio en particular, se dispondría de diferentes actores que podrían tener diversas necesidades, y que las mismas se convertirían en historias de usuario. (b) luego se describe la característica que se cubre con la historia de usuario y (c) finalmente se especifica la razón por la que dicha característica es necesaria.

- Existe un modelo en común para definir historias de usuario, este es el siguiente: *Yo como <usuario>, necesito/deseo/quiero <funcionalidad> para <beneficio de negocio>*.
- El proceso comienza de forma conjunta con el cliente, definiendo de manera sencilla y clara, las características de usuario que van a guiar el proceso de desarrollo. Posteriormente, cada historia de usuario se refina de manera que se identifican las tareas que son necesarias para llevar a cabo el desarrollo de la historia de usuario. En un principio, no es necesario detallar de forma completa todas las historias de usuario, sino que sólo las que tienen un mayor nivel de prioridad por parte del Product Owner. Esto, sin duda, permite que el proceso de desarrollo pueda adaptarse a posibles posteriores modificaciones, de forma que se vuelva más flexible. El resultado de esta fase es lo que se denomina en Scrum “Product Backlog” y contiene una lista de todas las historias de usuario priorizadas. Este enfoque basado en quién, qué y por qué, facilita la tarea de priorización de las historias de usuario, lo que permite realizar la planificación inicial del proyecto de forma más rápida, concreta y efectiva. Como puede deducirse de lo anteriormente expuesto, la obtención del Product Backlog se realiza con la ayuda del cliente y product owner, por lo que puede considerarse que se realiza en directo y, por lo tanto, las posibles dudas en lo que respecta a las historias de usuario pueden resolverse en ese mismo instante. Una vez

identificadas y priorizadas las historias de usuario del Product Backlog, se realiza la separación de historias de usuario en cada uno de los sprints. Es decir, que se distribuyen las historias de usuario a lo largo de los sprints. El objetivo es mover las historias con mayor prioridad para el Product Owner, directamente al "Sprint Backlog". Luego de que las historias de usuarios se encuentran seleccionadas para el primer sprint, se procede al desglose de cada historia de usuario en tareas. Las mismas son la suma de todos los esfuerzos necesarios por parte de cada integrante del equipo (incluyendo diseño, desarrollo, pruebas, integración, etc.) para completarla. Estas historias de usuario seleccionadas para el Sprint Backlog, se congelan de forma que durante dicho período no puedan producirse cambios sobre los aspectos que se encuentran en fase de desarrollo.

- Ejemplos de historias de usuarios: a continuación, se presenta un ejemplo de historia de usuario para responder a comentarios en un blog (ver gráfico 15).

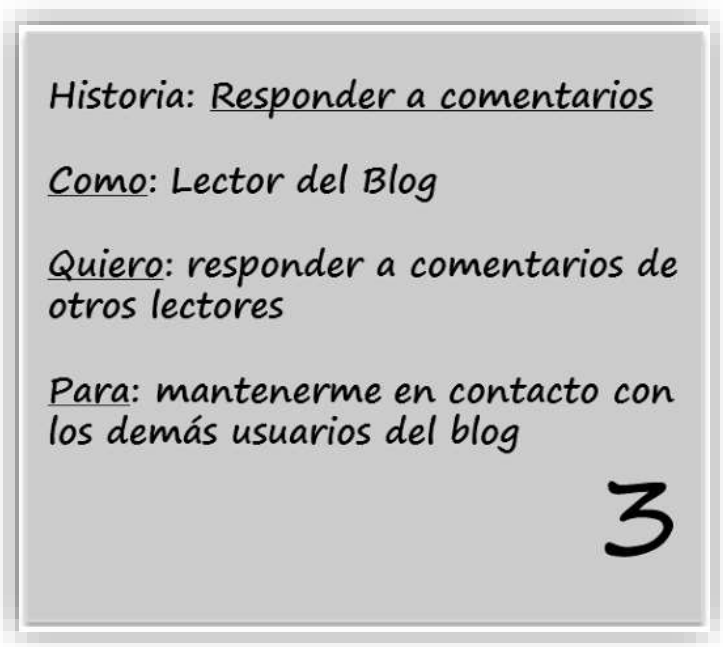


Gráfico 15. **Ejemplo de historia de usuario**

Estimación de Puntos (Poker Planning)

Esta es una técnica que se utiliza para estimar el esfuerzo de desarrollar cada una de las historias de usuario de un sprint en particular, abstrayendo al equipo de pensar en minutos, horas, días o cualquier unidad de tiempo. Es decir, se intenta utilizar valores que son abstractos para el equipo inicialmente, pero que con su uso y práctica permitirán sincronizar a todo el equipo, no solo a lo que refiere a tiempos, sino también al conocimiento que implica desarrollar un tipo de actividad. Requiere la participación de todas las personas comprometidas en el desarrollo de un producto software. Se denomina Poker Planning porque requiere el uso de un juego de cartas especial para realizar esta planificación. Las cartas se encuentran numeradas con una serie de Fibonacci desde 0 hasta 100, más la utilización de algunas cartas extras. Las mismas son: “0”, “½”, “1”, “2”, “3”, “5”, “8”, “13”, “20”, “40”, “100”, “?”, “dibujo de una taza de café”. Las cartas numeradas desde el 0 al 100 son utilizadas en la estimación de las historias de usuario, mientras que la carta “?” es utilizada para los momentos en que todavía existen dudas sobre la historia de usuario que se está estimando y, por ende, es necesario que se proveen o se utilicen otros términos para complementar el objetivo de la historia de usuario. Por último, la carta “dibujo de taza de café” se utiliza para sugerir un intervalo durante la estimación.

La estimación con Poker Planning involucra el siguiente proceso: (a) se reúne a todo el equipo alrededor de una mesa, preferentemente en una sala de reunión donde no haya interrupciones y se reparte el juego de cartas numeradas a cada uno de los integrantes del equipo por igual. Así, cada uno tiene las cartas desde el 0 al 100, más la carta del dibujo del café, más la carta del símbolo de interrogación. (b) el Product Owner lee la primera (y más prioritaria) historia de usuario a todo el equipo; (c) si alguien tiene dudas sobre el objetivo o algún punto sobre esa historia de usuario, es planteada en ese mismo momento. El equipo y el mismo Product Owner ayudarán a

resolver las dudas. Esto sirve para que, todo el equipo tenga en claro lo que es necesario desarrollar en esa historia de usuario en particular; (d) cada miembro del equipo (incluido el dueño de producto), selecciona una carta, la ubica boca abajo en la mesa y cuando todos los integrantes del equipo ya han seleccionado una carta, todos al mismo tiempo viran sus cartas boca arriba. (e) una vez todos han mostrado su carta (su estimación), hay dos opciones: descartar las estimaciones mínimas y máximas y quedarse con la estimación medía más repetida o buscar la unanimidad. Es decir, los que han diferido de la mayoría explican sus motivos y el resto explican los suyos. El objetivo es llegar a una estimación unánime. Este proceso, (f) se repite con cada historia de usuario presente en el product backlog.

A continuación, se muestra un juego de cartas utilizado durante el póker planning (ver gráfico 14).

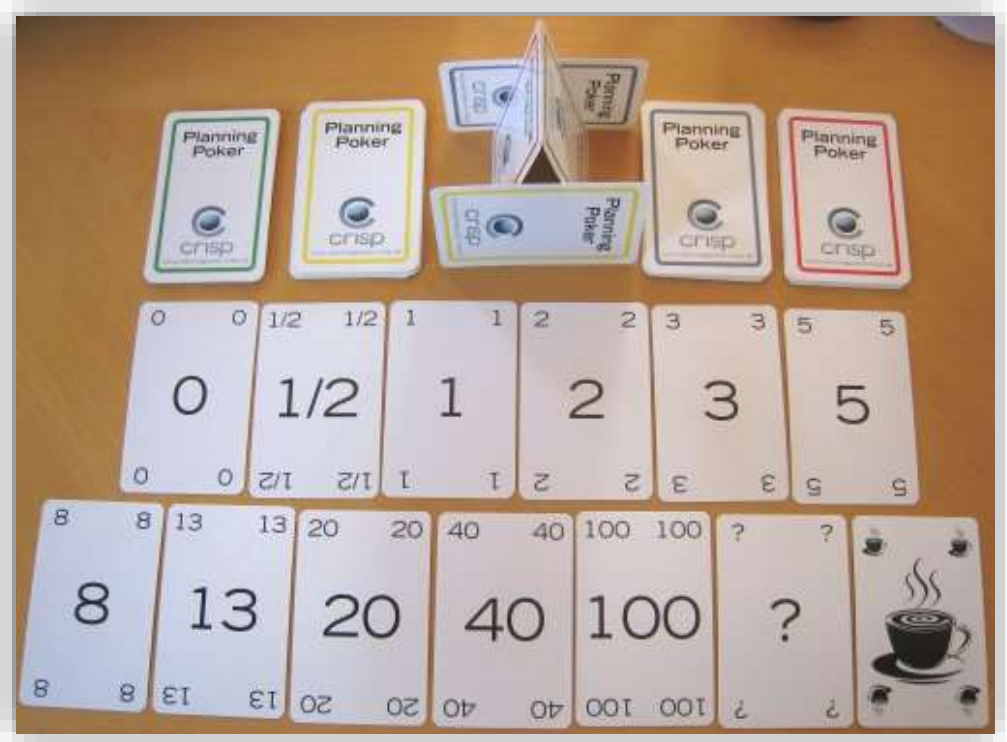


Gráfico 16. **Cartas utilizadas en Poker Planning**

Reuniones en Scrum

Como anteriormente se ha descripto, el sprint es el ritmo de los ciclos de Scrum. Está delimitado por la reunión de planificación del sprint y la reunión de retrospectiva. Entre esta delimitación de reuniones, existen otros dos (2) diversos tipos de reuniones que se realizan en diferentes periodicidades: (a) las llamadas reuniones “stand-ups”, que son reuniones diarias que se realizan durante el desarrollo de todo el sprint y (b) las reuniones de “revisión del sprint”: que, por lo general, se realizan una sola vez por sprint, casi al final del mismo. De esta forma, en contraposición con las metodologías convencionales, el proceso de desarrollo adquiere flexibilidad, agilidad y capacidad para adaptarse a su entorno.

A continuación, se describen de manera detallada cada una de las reuniones que se llevan a cabo usando la metodología Scrum (Pérez, 2011).

1. Reunión de Planificación de Sprint: se lleva a cabo al inicio del ciclo Sprint. La periodicidad de la reunión depende del tiempo en que el Sprint perdure. Como dijimos anteriormente, comúnmente los Sprints son establecidos entre 1 y 4 semanas de duración. Por lo cual, la reunión de planificación de Sprint se realizará al iniciar un nuevo Sprint. Los objetivos de esta reunión son: (a) describir en forma general el Objetivo del Sprint. Es decir, definir en pocas palabras el trabajo en el que el equipo se enfocará durante el Sprint; (b) conjuntamente con todo el equipo completo, preparar el Sprint Backlog, con su descomposición de tareas y estimación de tiempos. La planificación de las tareas a realizar en la iteración se divide en dos partes: (a) primera parte de la reunión, que se realiza en un tiempo máximo de 4 horas, durante las cuales el cliente presenta al equipo la lista de requisitos o funcionalidades priorizada del producto o proyecto, define la meta de la iteración (de manera que ayude a tomar decisiones durante su ejecución) y propone los requisitos más prioritarios a desarrollar en ella. Seguidamente, el equipo examina la lista, pregunta al cliente las dudas que le surgen y selecciona los

requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita y (b) segunda parte de la reunión, que se realiza en un tiempo máximo de 4 horas, durante las cuales el equipo planifica la iteración, dado que ha adquirido un compromiso, es el responsable de organizar su trabajo y es quien mejor conoce cómo realizarlo. El equipo lleva a cabo las siguientes actividades: (a) define las tareas necesarias para poder completar cada requisito, creando la lista de tareas de la iteración; (b) realiza una estimación conjunta del esfuerzo necesario para realizar cada tarea y (c) se asignan las tareas que cada miembro del equipo puede realizar.

1. Reunión Stand-up (diaria): es una reunión operativa, informal y ágil con el equipo de desarrollo, de un máximo de quince minutos y generalmente con sus integrantes de pie. En ella, a cada integrante del equipo se le hacen tres preguntas: (a) ¿Qué tareas ha hecho desde la última reunión? Es decir, tareas realizadas en un día; (b) ¿Qué tareas va a realizar hoy? (c) ¿Qué ayuda necesita para poder realizar este trabajo? Es decir, identificación de obstáculos o riesgos que impiden o pueden impedir el normal avance. Como apoyo a la reunión, el equipo cuenta con la lista de tareas de la iteración, donde se actualiza el estado y el esfuerzo pendiente para cada tarea, así como con el gráfico de horas pendientes en la iteración. Finalmente, se actualiza el grafico burndown con el trabajo realizado.
2. Reunión de Revisión de Sprint: es la reunión donde el equipo presenta el trabajo completado (funcionalidad o el módulo listo) al Product Owner. A su vez, se revisa el trabajo que no fue completado para identificar las causas en la reunión de Retrospectiva de Sprint. Es común que la reunión sea liderada por el product Owner y también, que se utilice este espacio para poder revisar indicadores y métricas de la evolución de desarrollo del producto en general a lo largo de los Sprints. Se asignan 4 horas como máximo para esta reunión.

3. Reunión de Retrospectiva: esta reunión se lleva a cabo cuando se ha terminado el sprint en curso y se ha realizado la revisión del mismo. Es decir, la reunión sucede entre cada sprint, para evaluar las acciones pasadas. Es por eso que permite aprender de los conocimientos y experiencias adquiridas hasta el momento y mejorar de forma continua el proceso de desarrollo. Se revisarán con el equipo, los objetivos marcados inicialmente en el Sprint Backlog concluido, se aplicarán los cambios y ajustes si son necesarios, y se marcarán los aspectos positivos (para repetirlos) y los aspectos negativos (para evitar que se repitan) del Sprint, que servirán de retroalimentación para el siguiente sprint. Los elementos que servirán de entrada y guiarán la planificación de un nuevo sprint serán el Product Backlog, las capacidades o habilidades del equipo, las condiciones que en ese momento se den del negocio, el avance tecnológico que se haya podido producir y el incremento que se pretenda hacer al producto que se está desarrollando. Las preguntas que se hacen durante esta reunión son: (a) ¿qué hicimos bien? (b) ¿cuáles cosas hay que mejorar? (c) ¿que hicimos mal?

Scrum aboga por la incorporación del Product Owner como un miembro más del equipo de desarrollo. De este modo, no se considera el proceso de definición de requisitos como un fin dentro del desarrollo del proyecto, sino que los requisitos aparecen implícitamente dentro del contenido de las historias de usuario.

Proceso de desarrollo Scrum

La clave de este enfoque es tener equipos conformados por pocas personas, pasando cortos tiempos desarrollando o construyendo algo pequeño. Esto es lo opuesto a lo que muchas metodologías tradicionales describen: grupo numeroso, construyendo algo grande, en grandes intervalos de tiempo. El objetivo principal es proveer algo que sea inmediatamente valorado por el cliente.

Según Peralta (2003), Scrum consta de tres fases: Pregame, Development y Postgame.

1. La fase de Pregame incluye dos subfases: (a) Planning: consiste en la definición del sistema que será construido. Para esto se crea la lista Product Backlog a partir del conocimiento que actualmente se tiene del sistema. En ella se expresan los requerimientos priorizados y a partir de ella se estima el esfuerzo requerido. La lista de Product Backlog es actualizada constantemente con ítems nuevos y más detallados; y con cambios en la prioridad de los ítems. (b) Architecture / High level Design: el diseño de alto nivel del sistema se planifica a partir de los elementos existentes en la Product Backlog List. En caso de que el producto a construir sea una mejora a un sistema ya existente, se identifican los cambios necesarios para implementar los elementos que aparecen en la lista Product Backlog y el impacto que pueden tener estos cambios. Se realiza una reunión de Revisión de Diseño, a los efectos de examinar los objetivos de la implementación y tomar decisiones a partir de la revisión. Luego de esto, se preparan planes preliminares sobre el contenido de cada versión de publicación (comúnmente llamada release).
2. La fase de Development (también llamada Game Phase): es la parte ágil de Scrum: En esta fase se espera que ocurran cosas impredecibles. Para evitar el caos, Scrum define prácticas para observar y controlar las variables técnicas y del entorno. Este control se realiza durante los sprints. Dentro de variables de entorno encontramos: tiempo, calidad, requerimientos, recursos, tecnologías y herramientas de implementación. En lugar de tenerlas en consideración al comienzo del desarrollo, Scrum propone controlarlas constantemente para poder adaptarse a los cambios de forma flexible.
3. Fase de PostGame: contiene el cierre del release. Para ingresar a esta fase se debe llegar a un acuerdo respecto a las variables del entorno. Es decir, si las mismas fueron completadas satisfactoriamente o no. Si estas variables de entorno, fueron

completadas correctamente, el sistema está listo para ser liberado (publicado) y es en esta etapa en la que se realiza integración, pruebas del sistema y documentación.

Según Pérez (2011), en Scrum, un proyecto se ejecuta en bloques temporales (sprints), en la que cada sprint debe proporcionar un resultado completo, un incremento del producto que sea susceptible de ser entregado con el mínimo esfuerzo, cuando el cliente lo solicite (ver gráfico 16).

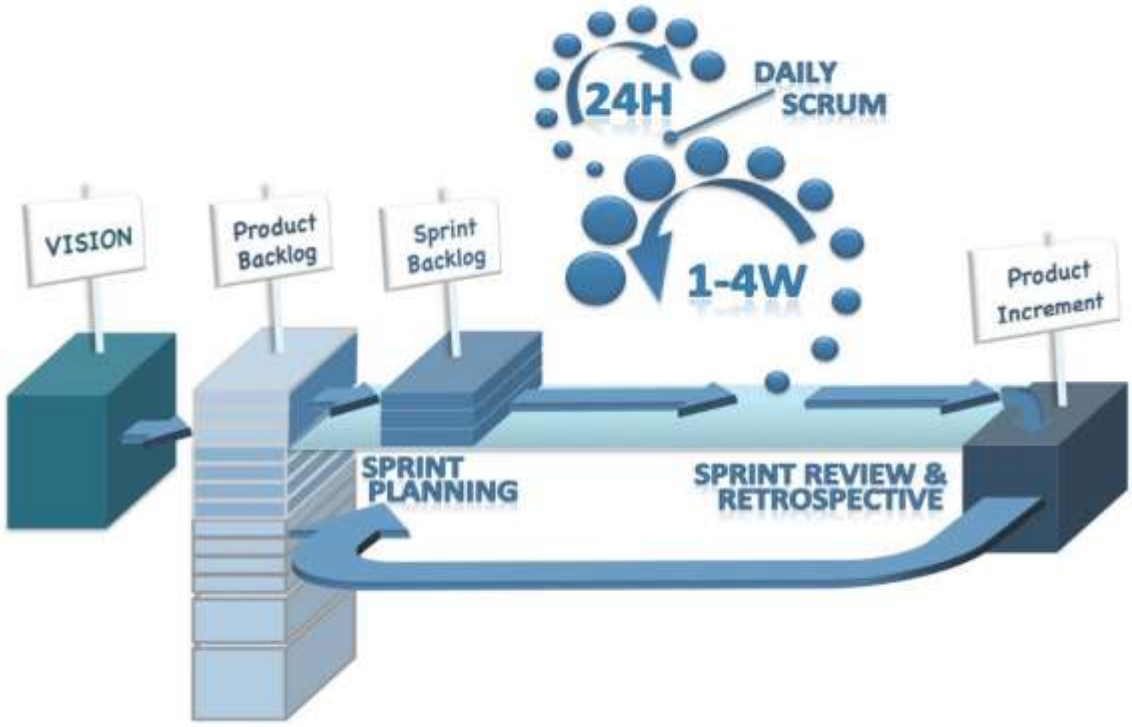


Gráfico 17. **Ciclo de Scrum Simplificado – Sprint**

A su vez, a continuación, se describe el proceso de Scrum de una forma más detallada (ver gráfico 17).

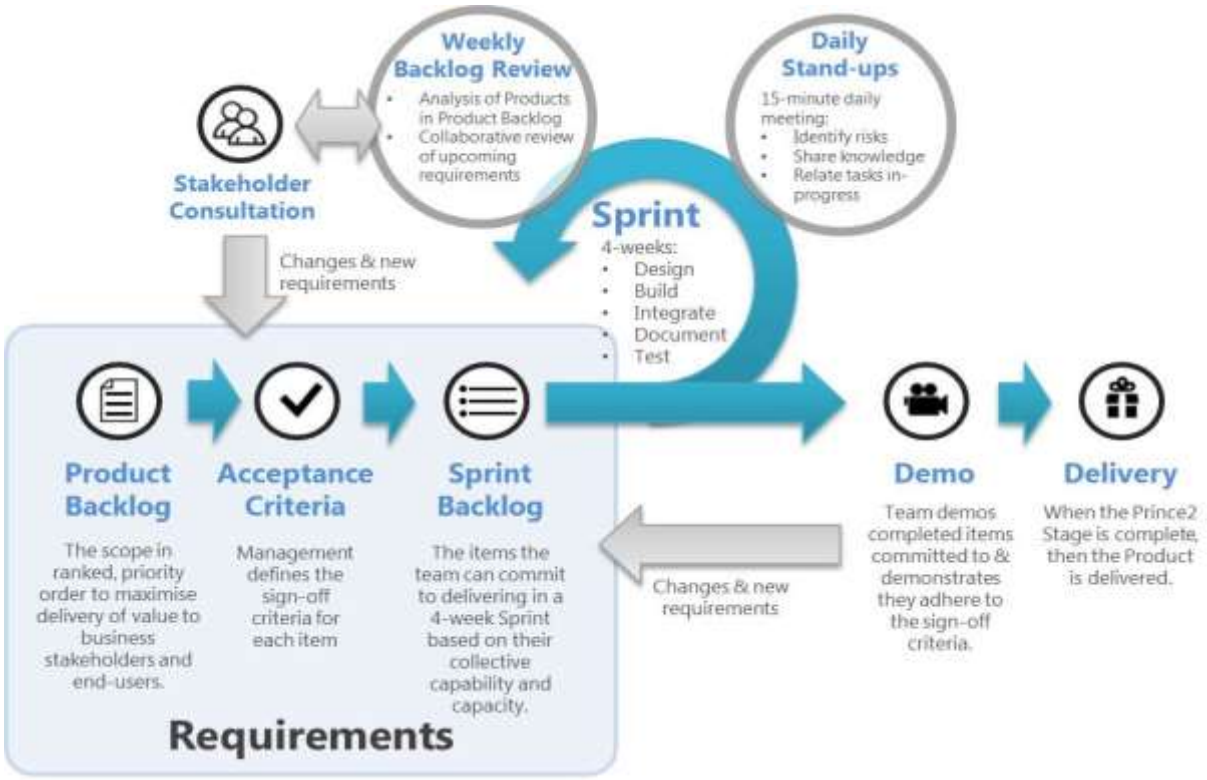


Gráfico 18. **Ciclo de Scrum Detallado – Sprint**

Valores y Principios en Scrum

Los valores y principios que deben seguir todos los integrantes del equipo, cuando se utiliza la metodología Scrum son, según Rodríguez (2008):

1. Compromiso: los miembros del equipo deben estar dispuestos a comprometerse con el objetivo de cada sprint y del proyecto en general. Scrum proporciona al equipo toda la autoridad que necesiten para obtener a cambio su compromiso. En este sentido, el equipo se tiene que comprometer a hacer su trabajo y cada miembro debe concentrar todos sus esfuerzos y habilidades en cumplir con el trabajo que se han comprometido a realizar, sin desviarse y realizando todas sus labores con

esmero. En Scrum se les pide a los participantes que adquieran tres compromisos: (a) el compromiso de entregar trabajo de calidad (Planificación del Sprint), (b) el compromiso de mejorar (Retrospectiva) y (c) el compromiso de ayudar (Scrum Diario). Estos compromisos son el núcleo de una red de los mismos, que permiten que Scrum funcione. Cuando los compromisos de unos con otros, el espíritu de Scrum y el marco son débiles, la entropía, el decaimiento, el cinismo y el olvido comienzan a aparecer, generando problemas exponenciales y visibles.

2. Franqueza: todos los aspectos del proyecto son visibles para todo el equipo, por lo que un valor fundamental es también la franqueza. Se pretende que no haya problemas ocultos, asuntos u obstáculos que puedan poner en peligro el proyecto.
3. Respeto: los miembros del equipo están avalados por sus conocimientos y experiencias. El respeto es un valor fundamental en y con cada uno de ellos.
4. Coraje: cada miembro del equipo tiene que tener el coraje suficiente para comprometerse, actuar, ser transparente en el desarrollo, respetar y hacer que lo respeten.

Scrumban

Scrumban (o Scrum-ban) es una metodología derivada de los enfoques Scrum y Kanban (Pérez, 2011). Esta metodología, por cierto, híbrida, contempla componentes y conceptos de ambas que se complementan entre sí, para lograr una mejor optimización del proceso de desarrollo. El término "Scrumban" fue utilizado por primera vez por Ladas (2008), en su publicación "Scrumban-Essays on Kanban System for Lean Software Development".

En la actualidad, muchas organizaciones definen a Scrumban como un enfoque avanzado y orientado a la mejora del proceso de desarrollo, ya que permite adoptar una combinación de reglas que ambas metodologías por separado no permiten. Existen dos (2) líneas de pensamiento en relación a Scrumban como enfoque híbrido: (a) se destacan algunos elementos de Scrum que son aplicados directamente al enfoque de Kanban, donde el proceso es mayormente inclinado hacia Kanban y (b) algunos elementos de Kanban que son aplicados al enfoque de Scrum, donde el proceso es mayormente inclinado hacia Scrum.

Además, desde una perspectiva de implementación, Scrumban permite un grado de flexibilización mayor, para partir desde una base simple y de forma gradual, llegar a una base compleja.

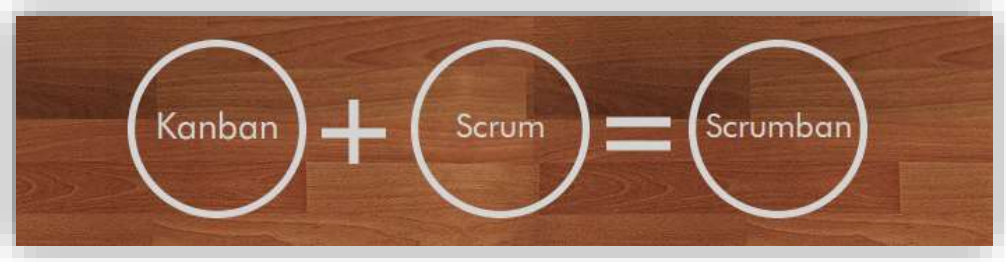


Gráfico 19. **Kanban y Scrum combinados**

Éste, es un modelo de desarrollo especialmente adecuado para proyectos de mantenimiento o proyectos en los que las historias de usuarios, varían con frecuencia

de forma muy dinámica. También, es reconocida por ser un enfoque adecuado para aquellos proyectos, en los cuales surjan errores de programación inesperados durante todo el ciclo de desarrollo del producto. Para todos estos casos, los sprints del enfoque Scrum independiente no son factibles. Esto es, dado que los errores/impedimentos que surgirán a lo largo de las actividades, son difíciles de determinar y, por lo tanto, no es posible estimar el tiempo que conlleva cada historia de usuario (user story). Por lo expuesto, resulta más beneficioso adoptar un flujo de trabajo continuo propio del enfoque Kanban, el cual es adecuado a este tipo de escenarios.

Características de Scrum

- Roles: Cliente, equipo (con los diferentes perfiles que se necesiten).
- Reuniones: reunión diaria.
- Herramientas: tablero.

Características de Kanban

- Flujo visual.
- Hacer lo que sea necesario, cuando sea necesario y solo la cantidad necesaria.
- Limitar la cantidad de trabajo (WIP)
- Optimización del proceso.

Características de Scrumban

Ahmad Khan (2014) define las principales actividades que se realizan en Scrumban:

1. Visualizar el flujo de trabajo: esta es una de las herramientas más importantes tomadas de Kanban y que se aplica a Scrumban, la cual se refiere a literalmente visualizar cada historia de usuario en cada una de las etapas (o columnas) del flujo

del proceso. Esto infiere que cada ítem en el tablero es observado de su comienzo en el sprint backlog (en un tablero Scrumban, primera columna a la izquierda), hasta su etapa final “Completado” o “Done” (por lo general, la última columna a la derecha. La visualización ayuda al equipo de trabajo, a identificar los cuellos de botellas en el flujo del proceso. A su vez, esta observación del tablero completo ayuda a la sincronización del equipo, ya que ayuda a saber en qué está trabajando cada integrante.

2. Cola de trabajo: como fue definido anteriormente, una de las características de Scrum, es que las historias de usuario priorizadas y seleccionadas para ser trabajadas dentro de un sprint en particular, son un compromiso de entrega por parte del equipo hacia el cliente. Esto quiere decir que una vez que el sprint es iniciado, no son aceptados cambios en su contenido (es decir, en sus historias de usuario). En Scrumban, esto no sucede de esta forma, ya que se utilizan las denominadas colas de trabajo de Kanban, que permiten que los sprints sean alterados cuando sea requerido, sin producir grandes impactos.
3. Limitar el trabajo en progreso (WIP): otro de los aspectos de Scrumban es el de aplicar límites al trabajo en los puntos de progreso de todas las etapas, basado en la capacidad del equipo. De esta forma, al revés de ingresar más trabajo al flujo del proceso, el equipo se enfoca en localizar el cuello de botella (o el WIP limitado) en alguna de las fases, para ayudar a resolverlo y estar en condiciones nuevamente retomar nuevas actividades.
4. Reglas explícitas: en Scrum, los equipos son auto-organizados, trabajan y se coordinan a sí mismos con reglas implícitas. Sin embargo, en la práctica existen siempre diferencias entre cómo un equipo debe organizarse y cómo están funcionando en la realidad. Es por eso que en Scrumban, las reglas (o políticas) del equipo se hacen efectivamente explícitas, para que todos en el equipo estén facultados para auto-organizarse, con el fin de lograr flujos de trabajos más

dinámicos y efectivos. Así, esto permite que los integrantes del equipo puedan tomar decisiones rápidas sin poner mucho esfuerzo en el pensamiento, e incluso reducir la posibilidad de decidir incorrectamente y también, de ceder a las peticiones especiales bajo estrés. Estas reglas explícitas tratan de hacer frente a situaciones recurrentes, en las que alguien necesita tomar una decisión sobre cómo proceder o qué hacer, si surge una situación de algún tipo en particular.

Reuniones en Scrumban

Las reuniones de Scrum son uno de los elementos que se mantienen sin grandes cambios en Scrumban. El único cambio que predomina entre cada enfoque es la periodicidad en la cual las reuniones son realizadas.

1. Reuniones de Planificación: a diferencia de Scrum, Scrumban tiene reuniones de planificación más cortas, con el fin de actualizar el backlog cuando sea necesario. El equipo siempre debe planificar para el período más corto por delante. Tener reuniones de planificación más largas no tiene sentido en el caso de que las prioridades cambien a menudo. Esto reduce significativamente el tiempo en las que las reuniones de planificación se llevan a cabo (Ladas, 2008).
2. Reunión Stand-up (diaria): como en el enfoque Scrum, esta reunión diaria es de carácter operativa y ayuda a coordinar las actividades diarias y también, a remover cualquier impedimento que se presente durante el flujo.
3. Reunión de Revisión de Sprint: se consideran las mismas características que en el enfoque Scrum, para las estas reuniones de revisiones.
4. Reunión de Retrospectiva: la periodicidad de esta reunión puede diferir en cada equipo/proyecto a donde se implementa el enfoque Scrumban. Sin embargo, en Scrumban se hace especial hincapié en los cuellos de botellas presentados durante el trabajo pasado, de forma de entender las razones de los mismos y poder anticiparse en futuras reincidencias.

Dynamic Systems Development Method (DSDM)

El método de desarrollo de sistemas dinámico (DSDM) es una metodología de desarrollo de software originalmente basado en la metodología RAD. DSDM es un enfoque iterativo e incremental que enfatiza la participación continua del usuario.

DSDM fue desarrollado en Reino Unido en los años 90s por el DSDM Consortium, una asociación de vendedores y expertos en el campo de la ingeniería creado con el objetivo de "desarrollar y promover conjuntamente un marco de trabajo RAD independiente" combinando sus propias experiencias. El DSDM Consortium es una organización sin ánimo de lucro que tiene la propiedad y se encarga de la administración del marco de trabajo DSDM. La primera versión se completó en enero de 1995 y se publicó en febrero de 1995.

Su objetivo es entregar sistemas software en tiempo y presupuesto, ajustándose a los cambios de requisitos durante el proceso de desarrollo. DSDM es uno de los métodos ágiles para el desarrollo de software, y forma parte de la Alianza Ágil.

Como extensión del desarrollo rápido de aplicaciones, DSDM se centra en proyectos de sistemas de información que se caracterizan por planificaciones y presupuestos estrictos.

DSDM trata las características más comunes de los proyectos de sistemas de información, incluyendo presupuestos sobrepasados, plazos de entrega desaparecidos y falta de participación del usuario y compromiso de la alta gerencia.

Fases de DSDM

DSDM consiste en tres fases:

1. Fase pre-proyecto: Se define el alcance global, quiénes son los departamentos y personas implicadas, los compromisos de las distintas partes y quién o quienes financiarán el proyecto.

2. Fase de ciclo de vida del proyecto: La fase de ciclo de vida del proyecto está subdividida en 5 etapas:
    a. Estudio de la viabilidad (Feasability study): se estudia la adecuación de DSDM al proyecto y se identifican los riesgos del mismo. En esta fase se realiza un informe de viabilidad, un prototipo de viabilidad y un plan general del proyecto (plan de desarrollo + registro de riesgos);
    b. Estudio del negocio (Business study): si DSDM se ha considerado adecuado para el proyecto, el siguiente paso consiste en realizar un análisis más en profundidad del proceso o procesos de negocio que se van a informatizar. La participación e implicación del usuario resulta fundamental en esta fase, si en la misma no se consigue, habría que replantear la realización del proyecto siguiendo DSDM o con cualquier otra metodología. En esta fase, se obtiene un modelo de procesos identificando los usuarios clave en cada uno de ellos, un catálogo de requisitos priorizado (algo importante para una metodología iterativa e incremental), la arquitectura del sistema y el plan de prototipado;
    c. Iteración del modelo funcional (Functional Model Iteration): se divide en 4 fases: Identificación del prototipo funcional, donde se definen las funcionalidades que cubrirá el prototipo y se elabora un modelo funcional del mismo. Definición del calendario, basado en el plan de trabajo para la realización de este modelado funcional. Obtención del prototipo funcional y revisión del prototipo funcional, donde se determina el grado de aceptación del prototipo desarrollado, mediante pruebas realizadas por el usuario y/o la revisión de documentación. Para este último, es sumamente importante la obtención del feedback del usuario para que las especificaciones del producto a obtener con esta iteración, se acerquen lo máximo posible a las necesidades del usuario;

d. Iteración del diseño y de la construcción (Design and Build Iteration). Se divide en 4 fases: Identificación del prototipo de diseño, donde se determinan los requisitos funcionales y no funcionalidades que cubrirá el prototipo. Definición del calendario, basado en el plan de trabajo para la construcción de este prototipo. Construcción del prototipo de diseño, donde efectivamente se construye un producto utilizable para los usuarios. Tratándose, por tanto, de un producto finalista en el sentido de que, ya es posible su usabilidad para realizar el trabajo cotidiano sobre las funcionalidades implementadas. Y, por último, la revisión del prototipo de diseño.

e. Implementación (Implementation). Se divide en 4 fases: Aprobación del usuario: el usuario realiza la aprobación del producto a entregar. Formación, donde se forma a los usuarios finales de la aplicación. Implementación: se realiza la instalación del producto en las instalaciones del cliente. Revisión de negocio, donde se revisa la adecuación del sistema a las necesidades del negocio y a los objetivos iniciales que se habían establecido para el mismo. En función de lo que se decida en esta última fase, se continuará a la fase de Post-proyecto, o a una de las fases anteriores del ciclo de vida. Si falta o se detecta algún nuevo aspecto funcional relevante, es necesario volver a la fase de Estudio del Negocio. Sin embargo, si no es relevante, se vuelve a la fase de Iteración del modelo funcional y si, falta o se detecta algún nuevo aspecto técnico, se vuelve a la fase de Iteración del diseño y la construcción.

3. Fase post-proyecto: Tiene como objetivo la continuidad del sistema en el sentido de que siga siendo útil a las necesidades de los usuarios, comprende el mantenimiento directo del sistema que se construye.

En algunas circunstancias, hay posibilidades de integrar prácticas de otras metodologías y modelos, tales como RUP, XP y PRINCE2 como complemento a

DSDM. Otro método ágil que tiene alguna similitud en el proceso y concepto de DSDM, es Scrum.

El enfoque DSDM

En DSDM hay nueve procesos subyacentes que consisten en cuatro fundamentos y cinco puntos de partida.

- La participación del usuario es la clave principal para llevar a cabo un proyecto eficiente y efectivo, donde ambos, usuarios y desarrolladores compartan un sitio de trabajo, de tal forma que las decisiones se puedan hacer de la forma más exacta.

- Se le deben otorgar poderes al equipo del proyecto para tomar decisiones que son importantes para el progreso del proyecto, sin tener que esperar a aprobaciones de más alto nivel.

- Poner foco en la entrega frecuente de productos, con el supuesto de que entregar algo suficientemente bueno pronto es siempre mejor que entregar todo perfecto al final. Entregando el producto con frecuencia desde una etapa temprana del proyecto, el producto se puede probar y revisar, y el registro de pruebas y el documento de revisión se pueden tener en cuenta en la siguiente iteración o fase.

- El principal criterio de aceptación de un entregable es que entregue un sistema que trate las necesidades del negocio actuales. Entregar un sistema perfecto que trate todas las necesidades del negocio posibles, es menos importante que centrarse en las funcionalidades críticas.

- El desarrollo es iterativo e incremental y conducido por la retroalimentación del usuario para converger en una solución de negocio efectiva.

- Todos los cambios durante el desarrollo son reversibles.

- Se debe crear una línea base del alcance y los requisitos a alto nivel, antes de que el proyecto empiece.

- Las pruebas se llevan a cabo a lo largo de todo el ciclo de vida del proyecto.

- La comunicación y la cooperación entre todas las personas involucradas en el negocio son necesarias para ser eficientes y efectivos.

Supuestos adicionales:

- Ningún sistema es construido perfectamente al primer intento. En pocas palabras, el 80% de los beneficios del negocio vienen de un 20% de los requisitos del diseño. Por lo tanto, DSDM empieza implementando este 20% crítico primero; esto puede producir un sistema que proporcione funcionalidad suficiente para satisfacer al usuario final y el 80% restante se puede añadir posteriormente en siguientes iteraciones. Esto mitiga el riesgo del proyecto de salirse de los plazos o el presupuesto.

- La entrega de los proyectos debe ser en tiempo, en presupuesto y con buena calidad.

- Cada paso del desarrollo sólo necesita completarse hasta que empiece el siguiente paso. Esto permite que la nueva iteración del proyecto comience sin un retraso innecesario. Los cambios en el diseño pueden coincidir con los cambios en la demanda de los usuarios finales, ya que cada iteración del sistema es incremental.

- Se incorporan técnicas de gestión de proyectos y desarrollo.

- DSDM se puede aplicar en proyectos nuevos o para ampliar sistemas actuales.

- La evaluación de riesgos debe centrarse en la funcionalidad del negocio que se va a entregar, no en el proceso de desarrollo o sus artefactos (tales como documentos de requisitos o diseño).

- La gestión premia la entrega de productos, más que la compleción de tareas.

- La estimación debe basarse en la funcionalidad del negocio, en vez de las líneas de código.

Factores de éxito críticos de DSDM

Se han identificado una serie de factores que tienen gran importancia para asegurar proyectos con éxito. Estos son:

- La aceptación de DSDM por parte de la gerencia y otros empleados. Esto asegura que los diferentes actores del proyecto están motivados desde el principio y que permanece a lo largo del proyecto.

- El compromiso de la gestión de asegurar la participación de usuario final. En otras palabras, el enfoque de prototipos requiere una participación fuerte y dedicada del usuario final para probar y juzgar los prototipos funcionales.

- El equipo tiene que estar formado por miembros habilidosos, conformando una cohesión estable. Un tema importante es el otorgamiento de poderes al equipo del proyecto. Esto quiere decir que el equipo tiene que poseer el poder y posibilidad de tomar decisiones importantes relacionadas con el proyecto, sin tener que escribir propuestas formales a la alta gerencia, lo que puede llevar mucho tiempo. Para que el equipo del proyecto pueda ejecutar un proyecto con éxito, necesitan también la tecnología correcta para llevar a cabo el proyecto. Esto implica un entorno de desarrollo, herramientas de gestión de proyectos, etc.

A continuación, se visualiza el ciclo de vida de DSDM (ver gráfico 19).

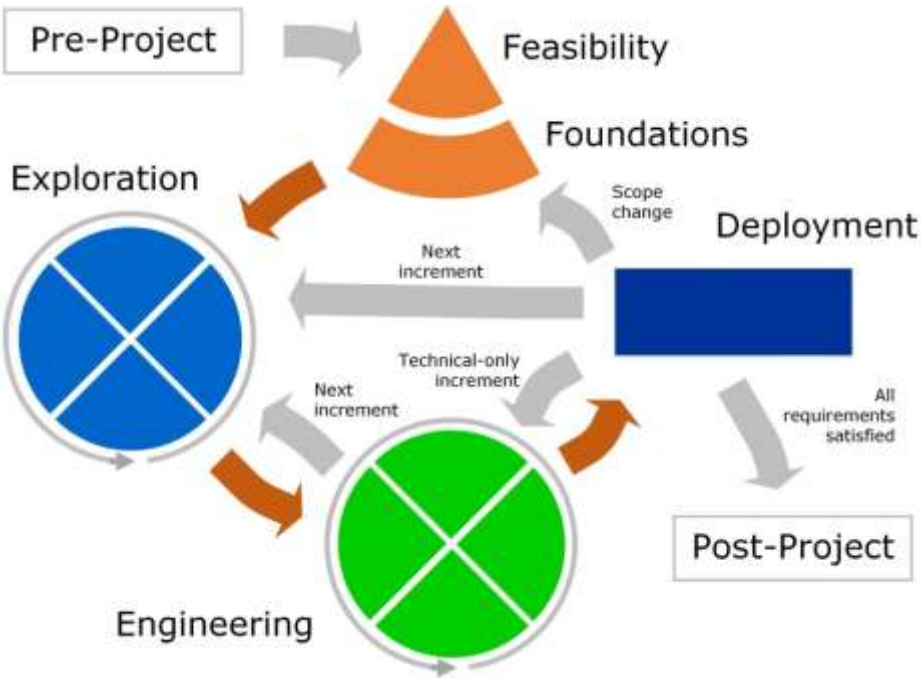


Gráfico 20. **Ciclo de vida de DSDM**

Crystal Clear

Crystal Clear fue propulsada por Alistair Cockburn, uno de los padres del Manifiesto Ágil, que consideraba que la metodología debe adaptarse a las personas que componen el equipo, utilizando políticas diferentes para equipos diferentes. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores: Crystal Clear (3 a 8 miembros), Crystal Yellow (10 a 20 miembros), Crystal Orange (25 a 50 miembros), Crystal Red (50 a 100 miembros) y Crystal Blue (para más de 100 miembros). (Rodríguez, 2008).

Crystal Clear está dirigida a la comunicación de equipos pequeños que desarrollan Software cuya criticidad no es elevada. Tiene asociadas siete propiedades que se describen a continuación:

1. Entrega frecuente: consiste en entregar incrementos de software a los clientes con frecuencia, no solamente en compilar el código. La frecuencia dependerá del tipo de proyecto. La misma puede ser diaria, semanal o mensual.
2. Comunicación osmótica: el término osmótica se refiere a la información que fluye de forma dinámica en un entorno compartido por los miembros del equipo. Por lo general es necesario que todos los integrantes se encuentren en el mismo cuarto de forma que la comunicación se produzca prácticamente por ósmosis.
3. Mejora reflexiva: se refiere a disponer de un pequeño tiempo (unas pocas horas cada o una vez al mes), para pensar bien qué se está haciendo, comparar notas, reflexionar, discutir.
4. Seguridad personal: dialogar con los compañeros cuando algo molesta dentro del grupo, de forma de incentivar una cohesión grupal y trabajo en equipo.
5. Foco: entender las actividades que se está realizando y tener la tranquilidad y el tiempo para hacerlo.
6. Fácil acceso a usuarios expertos: disponer de una comunicación directa con expertos desarrolladores, de forma de que puedan ayudar con consultas y o dudas que puedan generar desvíos o impedimentos.

Estrategias que se aplican en Cristal Clear

Crystal Clear no requiere ninguna estrategia o técnica, pero siempre es útil tener unas cuantas a mano para empezar. Las estrategias comunes a otras Metodologías Ágiles, son:

1. Exploración de 360°: verificar tomando una muestra de valores del negocio, los requerimientos, el modelo de dominio, la tecnología, el plan del proyecto y el proceso.
2. Victoria temprana: es preferible buscar pequeños triunfos iniciales que aspirar a una gran victoria tardía.

3. Esqueleto ambulante: hace mención a transacción que deben ser simple pero completas.
4. Re-arquitectura incremental: se ha demostrado que no es conveniente interrumpir el desarrollo para corregir la arquitectura. Es por eso que se recomienda, en lo posible a que la arquitectura evolucione en etapas, manteniendo el sistema en ejecución mientras se la modifica gradualmente.
5. Radiadores de información: se refiere a dejar la información del proyecto en algún lugar visible, donde todo el equipo pueda observar. Tiene que ser comprensible para el observador casual, entendida de un vistazo y renovada periódicamente en función de cualquier cambio que se realice en el proyecto.

Agile Unified Process (AUP)

El proceso unificado ágil (AUP) es una versión simplificada de RUP desarrollada por Scott Ambler, entre 2002 y 2006. Describe un enfoque simple, fácil de entender, del desarrollo de software de aplicación de negocios usando técnicas y conceptos ágiles. AUP aplica técnicas ágiles, incluyendo desarrollo orientado a pruebas, modelado ágil, gestión de cambios ágil y refactorización de bases de datos para mejorar la productividad. (Patiño, 2013).

Fases de AUP

En Patiño (2013) se señala que la metodología AUP, al igual que RUP, tiene cuatro fases consecutivas:

1. Inicio: el objetivo es obtener una comprensión común entre el cliente y el equipo de desarrollo sobre alcance del nuevo sistema y definir una o varias arquitecturas candidatas para el mismo.

2. Elaboración: el equipo de desarrollo debe profundizar la comprensión de los requisitos del sistema y probar la arquitectura del mismo.
3. Construcción: construir software operativo de forma incremental, que cumpla con las necesidades de prioridad más altas de las personas involucradas en el negocio y también, que éste sea validado en el ambiente de desarrollo.
4. Transición: el objetivo es validar y desplegar el sistema en el entorno de producción.

Disciplinas de AUP

La metodología AUP según Patiño (2013) tiene siete disciplinas, de las cuales, las primeras cuatro son orientadas al desarrollo y las restantes se ejecutan en paralelo con éstas. A continuación, se definen:

1. Modelado: esta disciplina tiene como objetivo entender el negocio de la organización, tratar el dominio del problema e identificar una solución viable para este último.
2. Implementación: en esta disciplina se transforma el modelo en código ejecutable y se realiza un nivel básico de pruebas, en particular pruebas unitarias.
3. Pruebas: el objetivo de esta disciplina es realizar una evaluación objetiva para asegurar calidad. Esto incluye encontrar defectos, validar que el sistema funciona como es esperado y verificar que se cumplen los requisitos.
4. Despliegue: consiste en planificar el despliegue del sistema y ejecutar el plan para que el sistema esté a disposición de los usuarios finales.
5. Gestión de configuración: consiste en la gestión de acceso a los artefactos del proyecto. Esto no sólo incluye el seguimiento de las versiones de los artefactos, sino también controlar y gestionar los cambios en ellos.
6. Gestión de proyecto: consiste en la dirección de las actividades que tienen lugar dentro del proyecto. Esto incluye gestionar riesgos, dirigir a las personas y

coordinar a las personas y sistemas fuera del alcance del proyecto, para asegurar que se entrega a tiempo y dentro del presupuesto.

7. Entorno: esta disciplina proporciona soporte al resto del esfuerzo, asegurando que el proceso, la orientación (estándares y guías) y las herramientas adecuadas (software y hardware), están disponibles para el equipo cuando sean necesarias.

Filosofías de AUP

AUP se basa en los siguientes seis principios:

1. Los integrantes del equipo saben lo que están haciendo. La gente no va a leer documentación del proceso detallada, pero quieren algo de orientación a alto nivel y/o formación de vez en cuando. El producto AUP proporciona enlaces a muchos de los detalles, pero no fuerza a dichos integrantes a leer documentación del proceso.
2. Simplicidad. Todo está descrito de forma concisa.
3. Agilidad. AUP se ajusta a los valores y principios de desarrollo de software ágil y a la Alianza Ágil.
4. Foco en las actividades de alto valor. El foco está en las actividades que realmente cuentan, no en todas las posibles situaciones o escenarios que pudieran pasar en un proyecto.
5. Independencia de herramientas. Se puede utilizar cualquier conjunto de herramientas. La recomendación es que se usen las herramientas que mejor se adapten al trabajo, que son con frecuencia herramientas simples.
6. Es necesario adaptar AUP para cumplir con las necesidades propias, de forma tal que se deben seguir los principios que mejor se adapten a cada equipo y proyecto.

A continuación, se muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto

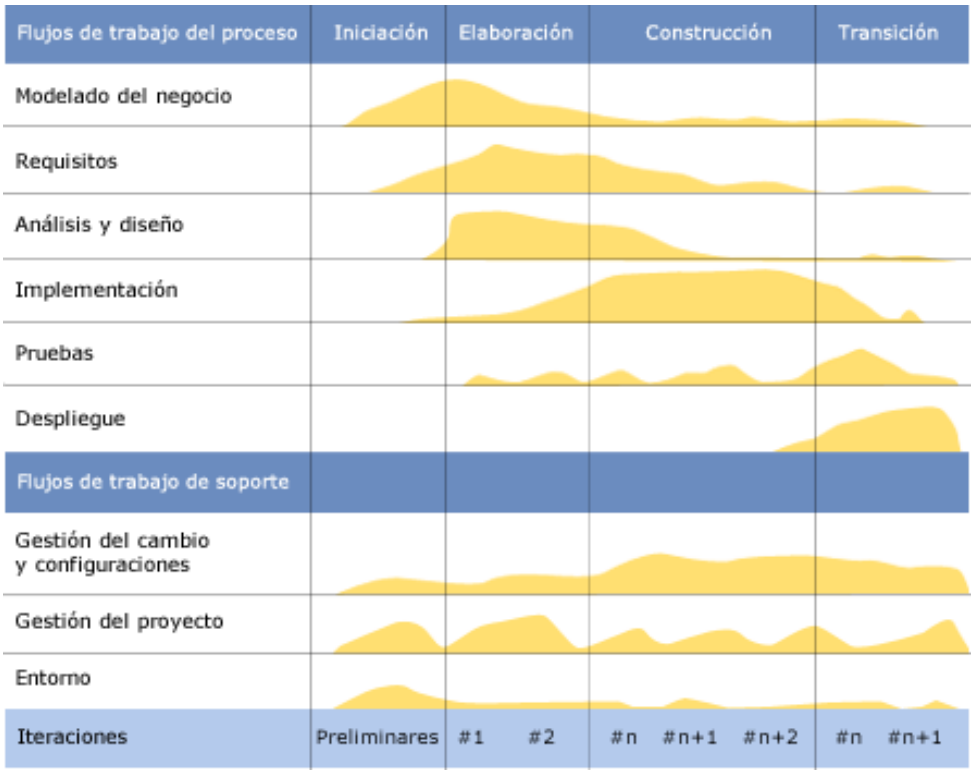


Gráfico 21. **Esfuerzo de actividades según cada fase**

Modelos Prácticos para Metodologías Ágiles

Son varias las prácticas que se utilizan en el desarrollo ágil. Aunque algunas de ellas se consideran metodologías en sí mismas, son simplemente prácticas usadas en diferentes metodologías.

A continuación, se describen algunas de las más extendidas: (a) el desarrollo orientado a pruebas (TDD por sus siglas en inglés Test Driven Development), (b) integración continua y (c) programación por pares.

Test Driven Development (TDD)

El desarrollo orientado a pruebas (TDD por sus siglas en inglés Test Driven Development), es un método de diseño de software que está relacionado con los primeros conceptos de pruebas de programación de Extreme Programming, en 1999, pero más recientemente se está creando un interés general mayor en sí mismo. Éste es una técnica de desarrollo de software que usa iteraciones de desarrollo cortas basadas en casos de prueba escritos previamente, que definen las mejoras deseadas o nuevas funcionalidades. Un concepto clave de este modelo es que las pruebas se escriben antes de que se escriba el código, para que éste cumpla con las pruebas. Cada iteración produce una porción de código necesario para pasar la prueba de la iteración. Finalmente, el programador o el equipo, refactoriza el código para acomodar y afinar los cambios.

Este modelo supone que los desarrolladores crean sus propias pruebas unitarias automatizadas para definir los requisitos del código, antes de escribir el código en sí mismo. Dichas pruebas contienen afirmaciones que son verdaderas o falsas y tienen como objetivo modelar al código de forma tal, que el código sea claro y funcione según lo esperado. Para realizar esta labor, los desarrolladores utilizan herramientas que asisten durante la creación y ejecución automáticamente de los conjuntos de casos de pruebas.

Una ventaja de esta forma de programación es evitar escribir código innecesario. Se intenta escribir el mínimo código posible y al escribir las pruebas antes que el código, conlleva a dos beneficios principales: (a) ayuda a asegurar que la aplicación se escribe para poder ser probada, ya que los desarrolladores deben considerar cómo probar la aplicación desde el principio, en vez de preocuparse por ello luego y (b) asegura que se escriben pruebas para cada característica.

El ciclo de las pruebas según Beck (2002) consta de cinco pasos:

1. Añadir una prueba: en el desarrollo orientado a pruebas, cada nueva característica empieza con la escritura de una prueba. Esta prueba deberá fallar inevitablemente porque se escribe antes de que se haya implementado la característica. Para escribir una prueba, el desarrollador debe entender claramente la especificación y los requisitos de la característica. El desarrollador puede llevar a cabo esto a través de casos de uso e historias de usuario que cubran los requisitos y las condiciones de excepción. No hace falta que sea una nueva funcionalidad, puede implicar también una variación o modificación de una prueba existente.
2. Ejecutar las pruebas y comprobar que la última que se ha añadido falla: esto valida que las pruebas están funcionando correctamente y que las nuevas pruebas no pasan erróneamente, sin la necesidad de código nuevo.
3. Escribir una porción de código, realizar cambios en la implementación: el siguiente paso es escribir una porción de código que haga que se puedan pasar las pruebas. El código escrito en esta etapa no será perfecto y puede, por ejemplo, pasar la prueba de una forma poco elegante. Esto es aceptable porque en pasos sucesivos se mejorará y perfeccionará. Es importante resaltar que el código escrito sólo se diseña para pasar la prueba; no se debería predecir más funcionalidad.
4. Ejecutar las pruebas automatizadas y ver que tienen éxito: si todas las pruebas ahora pasan, el programador puede estar seguro de que el código cumple todos los requisitos probados. Este es un buen punto para empezar el paso final del ciclo.
5. Refactorizar el código para mejorar su diseño: en este punto, se puede limpiar el código si es necesario. Volviendo a ejecutar las pruebas, el desarrollador puede estar seguro de que la refactorización no ha dañado ninguna funcionalidad existente.

Este ciclo se repite con cada una de las nuevas pruebas.

Ventajas y desventajas del TDD

Entre las ventajas que se desprenden del uso de esta práctica, se encuentran las siguientes:

1. Al escribir primero los casos de prueba, se definen de manera formal los requisitos que se espera que cumpla la aplicación. Los casos de prueba sirven como documentación del sistema.
2. Al escribir una prueba unitaria, se piensa en la forma correcta de utilizar un módulo que aún no existe.
3. Las pruebas permiten perder el miedo a realizar modificaciones en el código, ya que, tras realizar modificaciones se volverán a ejecutar los casos de pruebas para comprobar si se ha cometido algún error.

   Entre las desventajas del TDD se describen:

   - TDD es difícil de usar en situaciones donde hacen falta todas las pruebas funcionales para determinar éxito o fracaso. Ejemplos de esto son interfaces de usuario, programas que trabajan con bases de datos, y algunos que dependen de configuraciones de red específicas.
   - El soporte de la gestión es esencial. Sin la creencia de toda la organización, de que TDD va a mejorar el producto, la gestión sentirá que se pierde tiempo escribiendo pruebas. Esto es porque históricamente, las pruebas siempre se han visto en un nivel o posición más baja que los desarrolladores o arquitectos.

Integración Continua

La integración continua es un conjunto de prácticas de ingeniería de sistemas y de software que aumentan la velocidad de entrega de software, disminuyendo los

tiempos de integración (entendiendo por integración, la compilación y ejecución de pruebas en todo un proyecto).

La integración continua es un proceso que permite comprobar continuamente que todos los cambios que lleva cada uno de los desarrolladores, no producen problemas de integración con el código del resto del equipo. Los entornos de integración continua construyen el software desde el repositorio de fuentes y lo despliegan en un entorno de integración sobre el cual, se realizan las diversas pruebas necesarias. El concepto de integración continua hacer referencia a que la integración del código se debe realizar de forma incremental y continua.

Cuando se necesita realizar un cambio en el código, el desarrollador selecciona el código actual que se encuentra en el repositorio y realiza una copia para su entorno de trabajo, de forma de poder realizar el cambio necesario. Es común que durante el tiempo que este desarrollador se encuentre trabajando sobre esa copia de código, otros desarrolladores (miembros del equipo), realicen actualizaciones al código que se encuentra en el repositorio. Esto genera que al momento de que el desarrollador, quiera subir su código modificado, deberá primero actualizarlo, para reflejar los cambios que se han producido en el repositorio desde que tomó su copia. Cuantos más cambios haya en el repositorio, más trabajo debe hacer el desarrollador antes de subir sus propios cambios. Teniendo en cuenta que esta situación puede suceder en diversos casos, el repositorio puede convertirse en un repositorio muy diferente al de la línea base del desarrollador. Estos casos son llamados “infierno de integración”, donde el tiempo que usan para integrar es mayor al tiempo que ha llevado realizar los cambios originales. En el peor de los casos, los cambios puede que sean descartados por el desarrollador y el trabajo se deberá realizar nuevamente.

La integración continua en sí misma, hace referencia a la práctica de la integración frecuente del código. Si bien el término frecuente es abierto a interpretación, con frecuencia se interpreta como “muchas veces al día”.

Algunas prácticas importantes para la integración son:

1. Mantener un repositorio de código: esta práctica recomienda el uso de un sistema de control de revisiones de código fuente del proyecto. Todos los artefactos que son necesarios para construir el proyecto se colocan en el repositorio. A su vez, existe la premisa de que: el sistema debe ser construido a partir de un check-out nuevo y no debe requerir dependencias adicionales.
2. Automatizar la construcción: el sistema debe ser construible usando un comando único. La automatización de la construcción debería incluir la integración, que con frecuencia incluye el despliegue en un entorno similar al de producción. En muchos casos, el script de construcción no sólo compila binarios, sino que también genera documentación, páginas web, estadísticas y distribución.
3. Hacer las pruebas propias de la construcción: este punto refiere exclusivamente al desarrollo orientado a pruebas, que fue descripto en el modelo anterior (TDD).
4. Una vez que el código está construido, se deben pasar todas las pruebas para confirmar que se comporta como el desarrollador esperaba que lo hiciera.
5. Mantener la construcción rápida: la construcción del código debe suceder de forma rápida y dinámica, de tal manera que, si hay un problema con la integración, éste sea identificado rápidamente.
6. Probar en un clon del entorno de producción: es importante disponer de un ambiente para pruebas igual al ambiente productivo. De esta forma, se puede evitar conducir a fallas cuando se despliegan en el entorno de producción.
7. Hacer fácil conseguir los últimos entregables: mantener disponibles las construcciones para las personas involucradas en el negocio y los técnicos de pruebas, puede reducir la cantidad de re-trabajo necesaria cuando se reconstruye una característica que no cumplía los requisitos. Adicionalmente, las pruebas tempranas reducen las opciones de que los defectos sobrevivan hasta el despliegue.

Encontrar incidencias también de forma temprana, en algunos casos, reduce la cantidad de trabajo necesario para resolverlas.

8. Todos los integrantes del equipo pueden ver los resultados de la última construcción.
9. Despliegue automático.

Ventajas y desventajas de la integración continua

La integración continua tiene muchas ventajas, entre ellas:

1. Reducción del tiempo de integración: los problemas de integración se detectan y arreglan continuamente. Si la integración se realiza de manera constante, no deberían existir grandes problemas al momento del último día antes de la fecha de liberación o release.
2. Cuando las pruebas unitarias fallan, o se descubre un defecto, los desarrolladores pueden revertir el código base a un estado libre de defectos, sin necesidad de depurar código.
3. Avisos tempranos de código no operativo/incompatible.
4. Avisos tempranos de cambios conflictivos.
5. Pruebas unitarias inmediatas de todos los cambios.
6. Disponibilidad constante de una construcción actual para propósitos de pruebas, demo o liberación.
7. El impacto inmediato de subir código incompleto o no operativo actúa como incentivo para los desarrolladores, para aprender a trabajar de forma más incremental con ciclos de retroalimentación más cortos.

Las desventajas que se pueden observar con el uso de esta práctica son las siguientes: (a) sobrecarga por el mantenimiento del sistema, (b) necesidad potencial de un servidor dedicado a la construcción del software y (c) el impacto inmediato al subir

código erróneo provoca que los desarrolladores no hagan tantos commits como sería conveniente como copia de seguridad.

Programación por pares (Pair Programming)

La programación por pares es una de las prácticas más utilizadas en metodologías ágiles, sobre todo en sistemas de alta complejidad que establece que la producción de código se realice con trabajo en parejas de programadores.

La utilización de Pair Programming en un proyecto, no implica que todas las líneas de código sean escritas por una pareja, ya que mientras una persona está escribiendo el código, la otra puede estar verificando errores, pensando en alternativas mejores, identificando casos de prueba, pensando en aspectos de diseño, etc. El objetivo principal de esta técnica es disminuir la tasa de errores, mejorar el diseño y aspectos como la satisfacción de los programadores. (Rodríguez, 2008).

La persona que teclea es el denominado “driver” (conductor-controlador). La persona que revisa el código recibe el nombre de “observer” o “navigator”. Los dos programadores cambian los roles con frecuencia (posiblemente cada 30 minutos). (INTECO, 2009).

Mientras revisa, el observador también considera la dirección del trabajo, generando ideas para mejoras y problemas futuros posibles a arreglar. Esto libera al driver para centrar toda su atención en los aspectos “tácticos” de completar su tarea actual, usando al observador como una red y guía segura.

Ventajas de la programación por pares

Entre las ventajas que puede ofrecer una práctica como la programación por pares según INTECO (2009) encontramos:

1. Calidad de diseño: las parejas típicamente consideran más alternativas de diseño que los programadores que trabajan solos, y llegan a diseños más simples, más fáciles de mantener. Así también, encuentran defectos de diseño muy pronto, por ello se obtienen programas más cortos, con mejores diseños y se disminuye la cantidad de errores de lógica.
2. Disminución del costo del desarrollo: siendo los defectos una parte particularmente costosa del desarrollo de software, especialmente si se encuentran tarde, la gran reducción en la tasa de defectos debido a la programación por pares puede reducir significativamente los costos del desarrollo de software.
3. Aprendizaje y formación: el conocimiento se transfiere fácilmente entre programadores: comparten conocimiento del sistema, y aprenden técnicas de programación unos de otros a medida que trabajan.
4. Superar problemas difíciles: los pares a menudo encuentran que problemas "imposibles" se convierten en más sencillos o incluso más rápidos, o al menos posibles de resolver cuando trabajan juntos.
5. Moral mejorada: los programadores informan que obtienen una mayor alegría en su trabajo y mayor confianza en que su código se encuentre correcto.
6. Disminución del riesgo de gestión: ya que el conocimiento del sistema se comparte entre varios programadores, hay menos riesgos que gestionar si un programador abandona el equipo.
7. Incremento de la disciplina y mejor gestión del tiempo: es menos probable que los programadores pierdan tiempo navegando en la web o en mails personales cuando trabajan con un compañero.
8. Flujo elástico: la programación por pares conduce a un diferente tipo de flujo que la programación individual. Es más rápido y más fuerte o elástico a las interrupciones, ya que cuando uno trata la interrupción el otro puede seguir trabajando.

9. Menos interrupciones: la gente es más reacia a interrumpir a una pareja que a una persona que trabaja sola.
10. Menos puestos de trabajo requeridos.

Desventajas de la programación por pares

Las principales desventajas de la programación por pares se listan a continuación:

1. Preferencia de trabajo: muchos ingenieros prefieren trabajar solos.
2. Intimidación: un desarrollador con menos experiencia puede sentirse intimidado cuando su compañero sea un desarrollador con más experiencia y como resultado puede que participe menos.
3. Costo de tutoría: los desarrolladores con experiencia pueden encontrar tedioso enseñar a un desarrollador con menos experiencia. Desarrolladores con experiencia que trabajen solos pueden ser capaces de producir código limpio y exacto desde el principio, y los beneficios de los pares pueden no valer el costo que supone un desarrollador adicional en algunas situaciones. Esto puede aplicar especialmente cuando se producen las partes más triviales del sistema.
4. Egos y conflictos potenciales: conflictos personales que pueden resultar en que uno o los dos desarrolladores se sientan incómodos. Las diferencias en el estilo de codificación pueden llevar a conflicto.
5. Hábitos personales molestos: las personas se pueden sentir molestas por otros miembros del equipo debido a objeciones con algunos de sus hábitos.
6. Costo: ya que hay dos personas a las que pagar, el beneficio de la programación por pares no empieza hasta que la eficiencia es por lo menos, el doble.

Ventajas y Desventajas de Metodologías Ágiles

Rodríguez (2008), señala que las ventajas principales de las metodologías ágiles son las siguientes:

1. Capacidad de respuesta a cambios a lo largo del desarrollo, ya que no los perciben como un obstáculo sino como una oportunidad para mejorar el sistema e incrementar la satisfacción del cliente, considerando la gestión de cambios como un aspecto característico del propio proceso de desarrollo software.
2. Entrega continúa y en plazos breves de software funcional, lo que permite al cliente verificar in situ el desarrollo del proyecto, ir conociendo la funcionalidad del producto progresivamente y comprobando si satisface sus necesidades, mejorando de esta forma su satisfacción. Además, el desarrollo en ciclos de corta duración favorece a que los riesgos y dificultades se repartan a lo largo del desarrollo del producto, principalmente al comienzo del mismo y permite ir aprendiendo de estos riesgos y dificultades.
3. Trabajo conjunto entre el cliente y el equipo de desarrollo con una comunicación directa que pretende mitigar malentendidos, que constituyen una de las principales fuentes de errores en productos de software, y exceso de documentación improductiva.
4. Importancia de la simplicidad, eliminando el trabajo innecesario que no aporta valor al negocio.
5. Atención continua a la excelencia técnica y al buen diseño, para mantener una alta calidad de los productos.
6. Mejora continua de los procesos y el equipo de desarrollo, entendiendo que el éxito, depende de tres factores: éxito técnico, éxito personal y éxito organizacional. (ver gráfico 21).

Gráfico 22. **Factores de Éxito**

Por otro lado, algunas de las desventajas de las metodologías ágiles se describen a continuación:

1. Falta de documentación del diseño: si bien Agile define un conjunto mínimo de documentación necesaria. Algunos equipos que ya han trabajado con metodologías y procesos de desarrollos menos flexibles y más ortodoxos, pueden entender que Agile no dispone de la documentación mínima necesaria y que es el código (junto con sus comentarios), lo que se denota como documentación formal del proyecto.
2. Problemas derivados de la comunicación oral: una de las características muy denotadas de las metodologías ágiles (en especial en el modelo Scrum), es su ventaja comunicacional que ayuda a crear un ambiente dinámico entre los integrantes del grupo de trabajo. La mayoría de la comunicación se realiza de forma oral (reuniones) y eso contribuye a agilizar el proceso de desarrollo de manera significativa. Sin embargo, esta ventaja puede convertirse rápidamente en una desventaja, si los integrantes del equipo no se conocen lo suficiente, ya que se pueden generar ambigüedades que impidan que este ambiente dinámico se desarrolle y en el peor de los casos, que cree problemas en el código desarrollado.

3. Fuerte dependencia de las personas: como se describe en el punto anterior, al tener una fuerte dependencia en el factor humano, esto puede conllevar a otros problemas relacionados.
4. Falta de reusabilidad derivada de la falta de documentación: al no requerir de documentación sobre cada detalle del proyecto y al depender del factor humano, esto puede causar que la reusabilidad de componentes no sea la esperada.
5. Problemas derivados del fracaso de los proyectos ágiles: si un proyecto ágil fracasa, no existe (o muy poca) documentación detallada en los cuales se detalle lo sucedido; lo mismo ocurre con el diseño. La comprensión del sistema se queda en las mentes de los desarrolladores.

Comparación de Metodologías Ágiles versus Metodologías Tradicionales

A continuación, se muestra una comparación entre las Metodologías Ágiles y las Metodologías tradicionales (ver cuadro 4).

Cuadro 4. Comparación de Metodologías Ágiles y Metodologías Tradicionales

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo).	Impuestas externamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

CAPÍTULO V

HERRAMIENTAS PARA LA GESTIÓN DE METODOLOGÍAS ÁGILES

A los fines de presentar como se comportan los enfoques Kanban y Scrumban, resulta necesario definir cómo interactúan equipos y terminales en este entorno de trabajo ágil. Para brindar soporte a la información y proveer un contexto de comunicación efectivo se utiliza un esquema Cliente / Servidor, el cual se detalla seguidamente (ver gráfico 22).

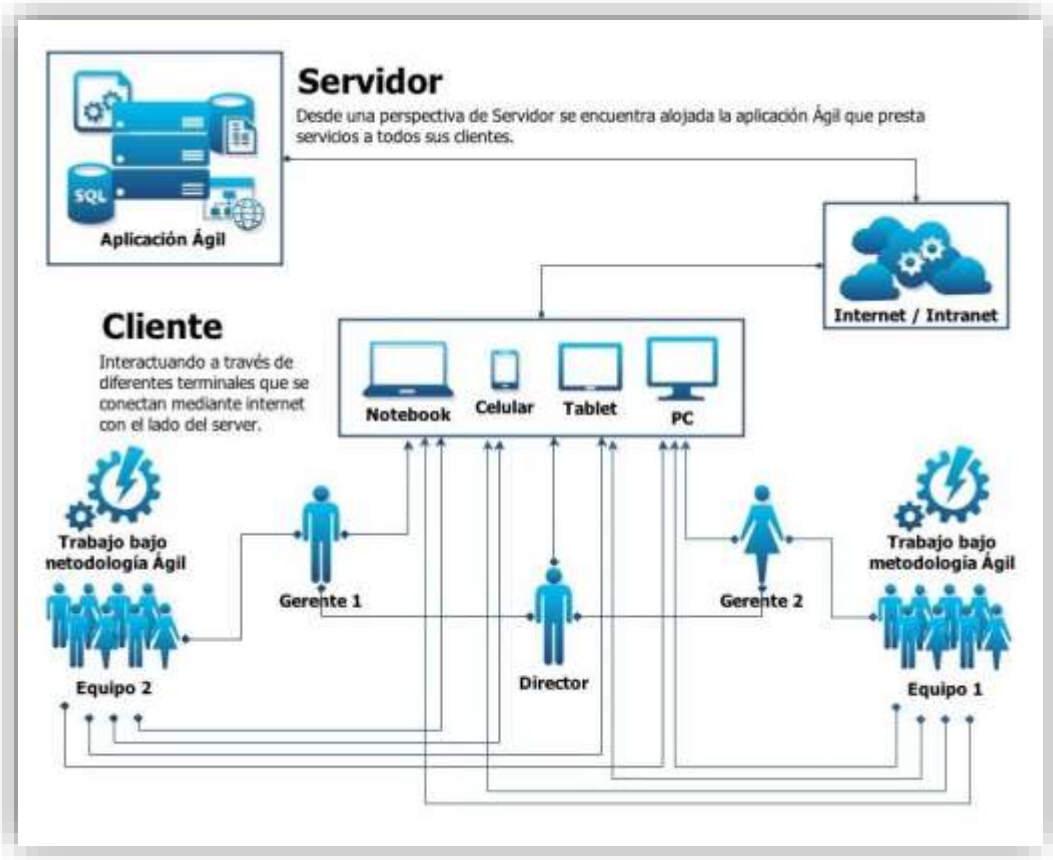


Gráfico 23. **Arquitectura Cliente / Servidor utilizada en las Herramientas de Gestión de Metodologías Ágiles**

Pueden observarse en el gráfico, dos (2) equipos (a modo ejemplo) con sus respectivos gerentes y su director. La interfaz cliente desde la cual se produce la comunicación con la parte servidora. La misma, una interfaz web que puede ser accedida desde una gama de dispositivos móviles y no móviles. Como se puede inferir, esta interfaz formará parte del lado del cliente, al cual el usuario se conectará para actualizar información con respecto al estado de sus actividades y/o para obtener cualquier tipo de información referente al proyecto asignado.

Por otro lado, dicha aplicación que sirve de plataforma de comunicación en tiempo real, se encuentra alojada en el lado del servidor, Cliente y Servidor se intercomunican de manera recíproca, dinámica e interactiva a través de internet o también puede ser una red interna, intranet o VPN.

A continuación, se describen las herramientas más reconocidas en el mercado, para la gestión de Herramientas para Metodologías Ágiles.

JIRA Software (Agile)

Inicialmente, JIRA surgió como un gestor de incidencias y era el producto insignia de la compañía Atlassian, fundada por Mike Cannon-Brookes y Scott Farquhar en el año 2002, en Australia. El nombre “JIRA” deriva de la traducción en japonés de Godzilla: “Gojira”. Los fundadores y desarrolladores querían que el término estuviera relacionado con Bugzilla, entonces para concluir llamaron a este de Gojira, JIRA.

Luego de varios años, Atlassian decidió apostar al segmento de gestión operativa de proyectos y posteriormente, agregó algunas características importantes para áreas no técnicas, como son la administración de tareas. Con el pasar del tiempo y los cambios tecnológicos en cuanto metodologías de trabajo, Atlassian agregó diversas características a su producto JIRA, para transformarlo también en un gestor de procesos que permitía planificación, ejecución y acompañamiento de actividades, con características muy interesantes para la organización de flujos de trabajo.

Como resultado de todo este avance en su software, hoy en día JIRA es un producto cuyo uso está extendido en todo tipo de compañías a lo largo del mundo, tanto tecnológicas como cualquier otra que requiera de una gestión de proyectos, procesos, incidencias, defectos y/o actividades. En este sentido, JIRA también es un facilitador de comunicaciones y de seguimiento de trabajo fundamental para agilizar los procesos en un flujo de trabajo.

Según Caballero (2012), algunas características fundamentales de JIRA son sus amplias posibilidades de configuración y su esencia de software de código abierto. Esto nos permite tanto ampliarla como realizar integraciones con otros sistemas.

Con el auge de las metodologías ágiles, Atlassian implementó su versión de agile llamada “Greenhopper”. Éste era un plugin que permitía definir y realizar el seguimiento a proyectos basados en el marco de trabajo Scrum. Proporcionaba funcionalidades tales como: creación de historias de usuario, estimación de las historias en la planificación de la iteración, visualización de la actividad del equipo, reporte de progreso, entre otros. Dicho plugin, ofrecía un “Scrumboard”, en el cual se podían planificar las iteraciones (Sprint) y visualizar distintos reportes personalizables. (PMOinformatica.com, 2012)

Luego, Atlassian re-estructuró la herramienta para hacerla totalmente compatible para equipos trabajando en metodologías Ágiles (en diferentes modelos y enfoques) y también, cambió su nombre para JIRA Software. El mismo no solo dispone de soporte para trabajar con Scrum, sino que tiene integrado el tablero Kanban como parte de su software. La herramienta permite definir y hacer seguimiento a proyectos basados en cualquier de los marcos de trabajo Scrum, Kanban y Scrumban, proporcionando funcionalidades como: Creación de historias de usuario, estimación de las historias en la planificación de la iteración, visualización de la actividad del equipo, reporte de progreso, entre otros.

JIRA Software continúa ofreciendo un “Scrum-board” mejorado, con nuevas características, en el cual se pueden planificar iteraciones (Sprint) y visualizar distintos reportes personalizables a medida. También, ofrece un “Kanban-board” que permite personalizar las etapas de desarrollo según sea requerido, basado en el proyecto, equipo y sus definiciones. (Javiergarzas.com, 2014)

Algunos equipos han adoptado enfoques de metodologías de trabajo mixtas para soportar su naturaleza de proyecto y trabajo. Por ejemplo, puede suceder que un equipo trabaje sobre un enfoque Scrumban, en el cual utilice los Sprints y roles de Scrum para combinarlos con el WIP y los ciclos de tiempo del enfoque Kanban. Estas características mixtas son fácilmente combinables con JIRA Software.

Entre otras características de JIRA Software a nivel general, dispone de notificaciones vía correo electrónico, posibilidad de adjuntar documentación, filtros, sistema de búsqueda basado en lenguaje natural, extensible y de fácil integración con casi todos los sistemas o bases de datos.

A nivel integraciones con otros sistemas, JIRA permite realizar integraciones mediante su back-end, con plataformas de versionamiento de código y también, plataformas de testing automático. De esta forma y como ejemplo, el solo hecho de cambiarle el estado a una actividad en particular (o arrastrarla de una columna a otra en un Scrum o Kanban board), permite que la herramienta dispare automáticamente una compilación de código, generando una nueva versión del sistema y posteriormente, ejecutar automáticamente sus casos de pruebas a diferentes niveles de complejidad. Esto sin duda, convierte a JIRA en una herramienta muy poderosa para aquellos equipos que requieren un nivel de automatización muy importante en su proceso de desarrollo.

Existen 2 tipos de licencias de JIRA Software que dependen de la naturaleza del equipo y su infraestructura.

1. JIRA Software Clouding: dispone la instancia de trabajo en el servidor clouding de la misma empresa, Atlassian. Lo cual permite tener alta disponibilidad y actualizaciones constantes del software. Su desventaja es que, si se dispone de un proyecto JIRA detalladamente implementado con características de automatización, este puede verse afectado con la actualización de una nueva versión de JIRA Clouding.
2. JIRA Software Server: permite instalar JIRA Software en un servidor privado, con actualización incluidas durante un año. La principal ventaja de esta licencia es que permite disponer de un servidor privado y esto permite un nivel de automatización mucho mayor a la licencia de Clouding. A su vez, permite integraciones con numerosas aplicaciones que automatizan de manera rápida y efectiva, los diferentes procesos del flujo de trabajo.

Como destaque final sobre esta herramienta, dado su flexibilidad adaptación a diferentes tipos de enfoques agile, ésta será la herramienta seleccionada para llevar a cabo la parte práctica en este proyecto.

Trello

Trello es el gestor de proyectos que más similitudes tiene con JIRA y también, es el más ajustado a lo que sería un tablero en Kanban típico. Inicialmente fue desarrollador por la empresa Fog Creek Software y luego se tornó una empresa propia, bajo el modelo de negocio Freemium (que permite una versión gratuita y otra paga). Posteriormente y hasta la actualidad, fue adquirida por Atlassian. Se puede definir a Trello como una herramienta sencillamente útil, flexible y visual, que es muy utilizada por aquellas personas y equipos, que comienzan a organizar sus actividades de forma colaborativa y digital.

Trello tiene su propia aplicación para móviles y tablets que permite realizar las mismas acciones que son realizadas desde su aplicación web. Entre muchas de sus características simples, los miembros del equipo consiguen generar columnas donde pueden reflejar su propio flujo de trabajo, sus ideas, listas de tareas por hacer, tareas en progreso y tareas finalmente realizadas.

Si bien, esta herramienta no es un gestor con grandes recursos, es sencillo y con todo lo necesario para desempeñar una buena gestión de proyectos, aplicando una metodología ágil o no. Permite cargar archivos desde el escritorio y se integra con otras herramientas de trabajo colaborativo como Google Drive, Dropbox, Box y Microsoft OneDrive. A su vez, las características más importantes, y al mismo tiempo sencillas de utilizar, son los listados de tareas (check-lists), etiquetas con colores y fechas de vencimiento y ejecución para las tareas en el tablero que permiten una organización realmente simple y efectiva, sin complicaciones. Por último, vale la pena describir que Trello dispone de características colaborativas que permiten incorporar comentarios y archivos adjuntos en cada una de esas tareas, como así también, la posibilidad de hacer comentarios en cada uno de los documentos o archivos que cada miembro adjunte al panel.

Rally Dev

Rally Dev surge exclusivamente como una herramienta para facilitar la gestión de proyectos mediante metodologías ágiles en el año 2001 por Ryan Martens, con especial foco en el gerenciamiento de los ciclos de vida de las aplicaciones y el gerenciamiento de portfolios de proyectos.

El sitio PMOinformática.com (2012), señala que Rally Dev ofrece características enfocadas a proyectos ágiles utilizando Scrum, tales como personalizar el propio dashboard, priorizar la lista de características / objetivos (Backlog), planificar las iteraciones, incluir historias de usuario a las iteraciones y gestionar la ejecución de

las iteraciones. La herramienta también brinda una versión denominada “Community Edition” que es gratuita, mientras que las versiones “Enterprise” y “Unlimited” poseen costo por la licencia según la cantidad de usuarios que sean requeridos.

Rally está ganando en popularidad e intenta quitarle el puesto a JIRA, sobre todo cuando en el año 2015, fue adquirida por CA Technologies, quien ha invertido recursos para mejorar y aumentar la cantidad de características del producto. En un artículo de la página JavierGarzas.com (2016) se presenta una comparación de Rally Software versus JIRA Software, estableciendo los siguientes puntos:

1. La herramienta Rally Software es más completa que JIRA en cuanto a las opciones de personalización, debido a que sus características de administración son mayores. Siempre hay que tener cuidado, con estas opciones y adaptar la herramienta a las necesidades particulares de cada situación, ya que puede que una acción simple, al personalizarla, se vuelva complicada para los miembros del equipo.
2. En cuanto a la gestión del proyecto, Rally es más completa ya que nos permite gestionar el Roadmap, permitiendo crear releases mediante la página “TimeBoxes”, que visualiza tanto las releases como las iteraciones.
3. En cuanto a las pruebas de aceptación, Rally permite un mayor soporte mediante la creación de casos de prueba directamente en la historia de usuario, y eso permite evolucionar las pruebas de aceptación al mismo tiempo que se va codificando; sin embargo, en JIRA es necesario utilizar complementos adicionales (llamados add-ins), para poder llevar a cabo dichas pruebas. Existen diversos complementos adicionales como Zephyr, que es una herramienta de terceros (no Atlassian), que tiene un costo mensual.
4. JIRA permite la personalización de los flujos de trabajo; sin embargo, no permite definir roles del equipo entre los usuarios de la herramienta, mientras que Rally permite asignar los siguientes roles Ágiles: Scrum Master, Product Owner y el equipo técnico.

Rational Team Concert

Rational Team Concert surge en el año 2008 de la mano de Rational Software (quien es una marca de IBM), como una herramienta de colaboración para equipos de desarrollo de software. La página de IBM (2017) promueve la herramienta Rational Team Concert IBM Rational Team Concert indicando que “es una solución de gestión del ciclo de vida del software que permite colaboración contextual, en tiempo real, para equipos distribuidos”. Basado en la plataforma de IBM Rational Jazz, Rational Team Concert proporciona una configuración del proceso, un marco de guía y obligatoriedad que da soporte a todo el entorno de entrega de software”.

La herramienta es propietaria con modelos de precios flexibles y software de servidor de forma gratuita. También, ofrece soporte para diversas plataformas y licencias basadas en rol en un solo lanzamiento, lo que permite la implementación de componentes individuales ahora y en el futuro. Además, Rational Team Concert proporciona beneficios de gestión de cambios de colaboración. Estos beneficios están disponibles de forma separada y se pueden integrar con sistemas de control de código fuente conocidos.

Rational Team Concert facilita la colaboración de los equipos y permite una entrega de software más rápida.

Kanbanize

Kanbanize es una herramienta para gestionar los tableros Kanban de los proyectos, tiene una versión online gratuita muy potente y otras versiones pagas con distintas funcionalidades agregadas (Garzás, 2013).

Desde Kanbanize se pueden ejecutar las funciones básicas de un enfoque Kanban, tales como: crear columnas, asignarles un WIP, crear tarjetas, cambiar el

estado de cada tarjeta, asignar prioridades a las tarjetas, asignar una fecha, entre otras. Además, permite crear y gestionar varios tableros.

Por otra parte, los miembros del equipo pueden dejar comentarios en las tarjetas y visualizar un log de actividad. Otro aspecto interesante, es que se puede visualizar el "Cycle Time" de cada tarea, incluida en la propia tarjeta.

Kanbanize fomenta la colaboración entre los miembros del equipo, ya que además de tener la posibilidad de publicar comentarios en cada tarjeta, hay chats para discutir aspectos de cada uno de los tableros Kanban.

El administrador del equipo puede crear usuarios, que serán los miembros del equipo. Estos miembros podrán seleccionar en una tarea del tablero y asignársela o, al crear la tarjeta se podrá asignar la tarea a un miembro concreto del equipo. Además, a través de notificaciones, estos usuarios pueden ver de una forma rápida los últimos cambios en el proyecto: si se ha creado una tarea o si se ha modificado el estado de alguna tarea, entre otras.

Otras herramientas no tan conocidas

Existen otras herramientas que son propias para el uso de metodologías ágiles y que no son reconocidas como las anteriormente mencionadas. A continuación, se describen brevemente cada una:

Redmine

Utilizada principalmente para metodologías de trabajo que involucran un enfoque Kanban ortodoxo. Especialmente aplicable en equipos donde se realizan actividades que involucran tickets de resoluciones de problemas. No dispone de flexibilidad en lo que respecta a su uso y configuración de opciones.

ScrumDo

Herramienta enfocada a Scrum, muy centrada en su simplicidad para gestionar historias de usuarios y listas de tareas, incluyendo iteraciones, gráficos burndown chart y también soporte para estimaciones siguiendo Poker Planning.

SprintoMeter

Especialmente aplicada para enfoques Scrum y eXtreme Programming para gestionar, medir y dar seguimiento a diferentes proyectos. Esta herramienta posee un foco importante en el intercambio de datos, por lo cual, entre otras cosas, permite la exportación de datos a informes Excel con gráficos burndown chart 3D.

Kungai

Orientada a enfoques Scrum. Dispone de panel de control interactivo del sprint, dashboard o cuadro de mando del proyecto y entre otras de sus facilidades, soporta estimación con Poker Planning.

Pango Scrum

Herramienta principalmente aplicada a Scrum, con una interfaz sencilla que permite escribir, estimar y priorizar los ítems del backlog y ofrece diversas características que facilitan la planificación de los Sprints y todas las reuniones scrum.

IceScrum

Herramienta versátil que puede ser utilizada para enfoques Scrum y Kanban e incluye facilidad para crear, consultar y estimar historias de usuario, interactuar de

forma sencilla con estas historias de usuario a través del Backlog y los Sprints dentro del proyecto.

Cuadro Comparativo: Ventajas y Desventajas

Cuadro 5. Ventaja y desventajas de Herramientas más reconocidas en el mercado, utilizadas en la Gestión de Metodologías Ágiles.

JIRA Software (Agile)	Trello	Rally Dev	Rational Team Concert	Kanbanize
Ventajas: 1. Ofrece dos (2) modalidades de uso, permitiendo alojar en servidores propios (JIRA Server), o utilizar los servidores de Atlassian (JIRA Clouding). 2. Compatible con Kanban, Scrum y Scrumban. 3. Posibilidad de automatización de actividades. 4. Permite integración con sistemas externos. 5. Diferentes add-ins para las diferentes necesidades. **Desventajas:** 1. Relativa complejidad para realizar la configuración de un proyecto e implementar todas sus reglas. 2. Migraciones de proyectos de una instancia de JIRA a otra no es fácil de realizar	**Ventajas** 1. Compatible con Kanban. 2. Sencillo de configurar y utilizar. 3. Gratuito con poderosas herramientas. **Desventajas:** 1. No posee add-ins. 2. No posee features Scrum. 3. Su nivel de personalización es bajo. 4. No posee ningún módulo de métricas e informes.	**Ventajas** 1. Compatible con Kanban, Scrum y Scrumban. 2. Permite dividir y diferir historias de usuario con sus métricas correctas. 3. Permite realizar seguimiento de los riesgos del proyecto. 4. Incorpora objetivos de calidad como KPIs, métricas e informes diversos. 5. Permite incorporación de SLAs. **Desventajas:** 1. No posee add-ins. 2. No posee features Kanban.	**Ventajas** 1. Compatible con Scrum. 2. Reduce riesgos del proyecto. 3. Incorpora objetivos de calidad, como métricas y KPIs. **Desventajas:** 1. No posee add-ins. 2.	**Ventajas** 1. Compatible con Kanban. 2. Una de las principales ventajas que ofrece este modelo es que no hace falta que los requisitos estén totalmente definidos al inicio del desarrollo, sino que se pueden ir refinando en cada una de las iteraciones. **Desventajas:** 1. No posee add-ins. 2. No posee features Scrum. 3. Posee módulo de métricas e informes muy básico.

CAPÍTULO VI

PROYECTO PRÁCTICO A IMPLEMENTAR

Introducción

Con el objetivo de realizar este trabajo de investigación sobre un ejemplo práctico con parámetros reales y actuales, se presenta el siguiente proyecto de carácter interno, bajo un ambiente real, con recursos humanos y materiales reales para demostrar el uso de ambos enfoques ágiles en la práctica.

Dicho proyecto a desarrollar se denomina “Wifi Social” y a seguir, se provee un contexto introductorio en términos generales del mismo.

Descripción del proyecto

El proyecto Wifi Social se refiere al desarrollo de un sistema web con integración de scripting para routers Mikrotik, en el cual se utiliza una red inalámbrica existente, para integrarla con software que es capaz de administrar y potenciar las características de dicha red wifi de diversas maneras. A su vez, dicho producto tiene como objetivo final el de ayudar al crecimiento de cualquier negocio desde una perspectiva de comunicación y fidelización clientes, marketing y visibilidad digital, seguridad y monitoreo.

Objetivos específicos

- Incrementar ventas, entendiendo lo que los clientes buscan.
- Aumentar visibilidad digital del establecimiento.
- Mejorar la seguridad de la red wifi, incluyendo monitoreo de su actividad.
- Atraer más clientes mediante promociones.

- Fidelizar clientes del establecimiento mediante descuentos.

Características

- Integración con redes sociales, como Facebook, Google, Instagram, Twitter entre otras.
- Personalización de las pantallas que el cliente verá en su dispositivo al conectarse a la red wifi.
- Recopilación, análisis y segmentación de diferentes informaciones estadísticas en forma de informes y gráficos que serán visualizadas en un panel de control.
- Envío automático de e-mails y de notificaciones push mediante navegador y SMS.
- Compatibilidad con gama de equipos de bajo costo.
- Soporte y monitoreo por parte de un equipo especializado.

Proceso de flujo del producto

1. Utilizando la wifi existente del establecimiento a donde se desea implementar el producto, se conecta un router mediante cable de red, que dispone de la funcionalidad de Wifi Social. Al conectarlo, automáticamente el router Wifi Social se auto configura y ofrece una nueva red Wifi de forma abierta, que dispone el nombre del establecimiento.
2. Con el objetivo de obtener internet gratis, los clientes captan esta red wifi y se conecta a ella sin la necesidad de contraseña.
3. El sistema identifica el cliente conectado y antes de brindarle internet, le pide que inicie sesión con alguna de sus redes sociales de preferencia. A su vez, si lo prefiere, puede iniciar sesión registrando su email.
4. Al iniciar sesión, el usuario observa publicidad exclusiva del establecimiento y luego ingresa un código de acceso que solo es visible desde dentro del establecimiento.
5. Una vez que el código correcto es ingresado, el sistema le otorga internet al cliente.

6. Por otro lado, el sistema Wifi Social, obtiene datos de los clientes conectados para entender sus hábitos de visitas al establecimiento, ofrecerles diferentes promociones y mantener comunicación con el mismo.

Arquitectura y componentes

Las tecnologías utilizadas para el desarrollo de dicho proyecto son las siguientes: a) lenguaje PHP para el desarrollo del sistema web que se mostrará en el dispositivo móvil del usuario que se conecte a la red wifi social. El mismo lenguaje también será utilizado para el desarrollo del panel de control que visualizará el dueño o gerente del establecimiento, con el objetivo de administrar la red Wifi Social. b) la arquitectura utilizada para almacenar sus datos será MySQL. c) Se utilizará RouterOS scripting (propiedad de Mikrotik), para la interacción entre el hardware y el servidor de aprovisionamiento. Este será el encargado de decidir mediante una serie de políticas, si otorgará o no acceso a internet al usuario. d) Se utilizará un marco de trabajo llamado CodeIgniter, que proveerá una moldura base en la cual se desarrollará cada una de las características de la aplicación web y que permitirá también, acelerar la construcción de todo el sistema.

A continuación, se visualiza el proceso de conexión con todos sus componentes:

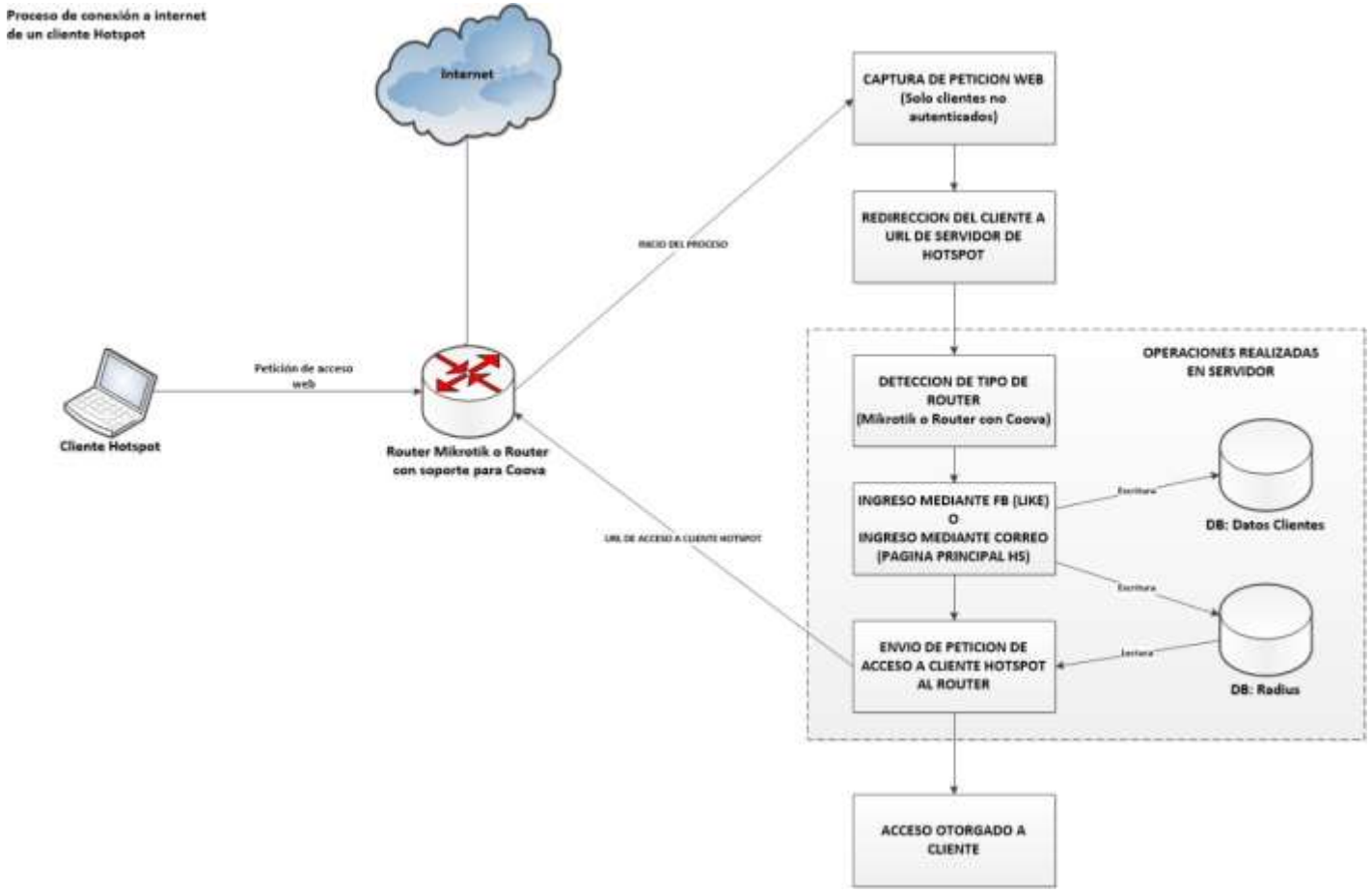


Gráfico 24. **Arquitectura del producto Wifi Social**

Dicha arquitectura se base en que el dispositivo móvil del cliente interactuará directamente con el router Mikrotik y éste iniciará comunicación mediante una petición al servidor Radius (de las siglas en inglés que significan Remote Authentication Dial-In User Service), quien será el encargado de verificar sus políticas de conexión y comunicarle al router Mikrotik si puede o no autorizar al dispositivo del cliente a utilizar internet. Al mismo tiempo, dispondremos de una segunda base de datos donde se almacenarán todos los datos del cliente, para reconocerlo y aprovisionarlo futuramente de manera automática, sin necesidad de iniciar sesión nuevamente.

Recursos Humanos

Teniendo en cuenta la necesidad y prioridad del proyecto a desarrollar, se asignarán diferentes perfiles profesionales requeridos, que serán seleccionados según los requerimientos del proyecto y participarán en el mismo en tiempo completo.

Perfiles Profesionales

Dichos perfiles profesionales se detallan a continuación:

- Un (1) líder de equipo, quien estará a cargo del proyecto desde su inicio a fin. El mismo será quien comunique a los interesados del proyecto, acerca del avance del mismo. A su vez, trabajará conjuntamente con el equipo para analizar, mitigar y resolver cualquier riesgo o problema que se presente. Este perfil será uno de los ejes principales para que la metodología ágil se aplique de forma efectiva durante el ciclo de vida de todo el proyecto.
- Un (1) desarrollador senior, con amplia experiencia en el desarrollo de este tipo de proyectos con las tecnologías anteriormente indicadas. El mismo tendrá un rol de líder técnico, para identificar, desarrollar y guiar la construcción del sistema desde una perspectiva tecnológica.

- Un (1) desarrollador semi-senior, con experiencia en el desarrollo de sistemas web, utilizando las mismas tecnologías que serán empleadas para el desarrollo de este proyecto.
- Un (1) analista de calidad y probador de calidad, quién tendrá como responsabilidad, crear, ejecutar y producir informes sobre el plan de control de calidad (comúnmente llamado, plan de testing). Él será el encargado de velar por la calidad del producto, para alcanzar estándares de calidad aceptables y para que el producto atienda la necesidad específica.

Equipo de Trabajo

Este es, probablemente uno de los apartados más importantes de este capítulo ya que se definen quienes serán los miembros del equipo que actuarán desde principio hasta el fin del proyecto, logrando la entrega del producto al cliente, su implementación y puesta a punto. Se pretende definir un equipo de trabajo con una alta cohesión grupal, que permita lograr el objetivo del proyecto y, asimismo, que pueda mitigar cualquier riesgo y también, resolver cualquier problema que se presente durante el ciclo de desarrollo del mismo.

Seguidamente, se observa un cuadro donde se definen las personas que integrarán el equipo de trabajo, por cada perfil profesional requerido:

Cuadro 6. Equipo de trabajo para Proyecto Wifi Social.

Perfil Profesional	Miembro de equipo seleccionado
Líder de equipo: Javier Esteban Salvay (argentino, localizado en São Luis, Maranhão, Brasil).	7 años de experiencia en gestión de proyectos IT, con experiencia en la implementación de metodologías ágiles. Además de experiencia en calidad de software. Nivel de español nativo, portugués e inglés avanzado.

Desarrollador Senior: Agustín Rafael González (argentino, localizado en Capital Federal, Buenos Aires, Argentina).	7 años de experiencia en desarrollo web con integración de dispositivos de red. Experiencia de 2 años con el uso de metodología ágil Kanban. Nivel de español nativo, portugués básico e inglés avanzado.
Desarrollador Semi-Senior: Paulo Prazeres Duarte (brasilero, localizado en Rio de Janeiro, Rio de Janeiro, Brasil).	3 años de experiencia en desarrollo de aplicaciones web con PHP, MySQL y CodeIgniter. Experiencia básica con metodología ágil Scrum. Nivel de español avanzado, portugués nativo e inglés intermedio.
Analista de Calidad: Gustavo Costa Masiero (brasilero, localizado en Rio de Janeiro, Rio de Janeiro, Brasil).	4 años de experiencia como analista de testing para diferentes proyectos de aplicaciones en diversas tecnologías. Desarrollando testing funcional, de componentes, de regresión y no funcional. Amplia experiencia con el uso de metodologías ágiles (Scrum y Kanban). Nivel de español intermedio, portugués nativo e inglés básico.

Como puede observarse en el cuadro arriba indicado, los miembros del equipo se encuentran en diferentes localizaciones, por lo que es de suma importancia la constante comunicación de cada uno de ellos con su líder y sus colegas de grupo.

Idioma

El idioma utilizado para comunicarse en todos los ámbitos de dicho proyecto será el español y oportunamente, los miembros del equipo en Brasil disponen de un nivel de español acorde al requerido para participar y llevar adelante el proyecto. Sin embargo, no es posible garantizar que no sean necesarias aclaraciones, explicaciones detalladas o indicaciones extras en algún momento en particular del proyecto. Es por eso que cada miembro entiende que se deberá disponer de entereza, paciencia e

integridad para entender y mantener el buen ambiente de trabajo, así como también, la cohesión grupal para llegar al objetivo final.

Frente a este punto tan importante, se dilucida que en aquellos casos que sean necesarias aclaraciones o revisión de algún punto en específico, el líder de equipo podrá comunicarse con los miembros del equipo de Brasil, por separado y en portugués, para alcanzar el entendimiento completo de cualquier tema que necesite ser re-aclarado.

Experiencia como equipo

Un punto muy importante que merece ser citado, es el simple hecho de que los miembros del equipo seleccionados ya disponen de experiencia previa trabajando como grupo laboral en otros proyectos de tecnología. En algunos de ellos, las fechas de entrega eran programadas por terceras personas que no participaban activamente en el proyecto. Como resultante, dichas fechas ya se definían como implícitas en el proyecto, sin posibilidad de cambio alguno, con poco margen para errores. Como otra de las experiencias que se pueden mencionar, dichos profesionales también han trabajado juntos como equipo en conjunto, en proyectos de operaciones y soporte a producción con acuerdos de nivel de servicio con valores demasiado escuetos (en inglés, SLA, Service Level Agreement), que aumentaban considerablemente el grado de presión, y muchas veces de estrés en momentos específicos.

Todo lo expuesto, permite puntualizar que previamente al inicio de este proyecto denominado Wifi Social, existe un alto grado de sinergia entre los miembros que integrarán el equipo del proyecto. Esto contribuirá sin duda, a acelerar el desarrollo del trabajo en términos generales.

Hardware requerido

En lo que respecta al equipamiento necesario para llevar a cabo el proyecto, se define como una necesidad imperativa, la utilización de tecnologías de nube (en inglés, clouding) para apoyar todo el ciclo de desarrollo en si, con productos Amazon AWS.

Esto define un marco extenso de aplicación que es desde su programación inicial hasta la implementación y puesta a punto en producción.

Para precisar con mayor claridad el hardware requerido, se describen a continuación los siguientes puntos:

1. Servidor de desarrollo y testing: se adquiere un servidor AWS EC2 (Amazon Elastic Compute Cloud), donde se instalará y configurará un servidor/contenedor de aplicaciones PHP y un servidor MySQL. Dicho ambiente será utilizado para realizar el desarrollo de la aplicación web y también realizar las pruebas de calidad primarias. A su vez, el mismo servidor tendrá 2 dominios publicados: a) http://wifi-development.harvay.com/, donde se publicará la última versión disponible por parte de los desarrolladores. b) http://wifi-testing.harvay.com/, donde se publicará la última versión disponible estable, sobre la cual se ejecutarán los casos de prueba para evaluar la calidad de la versión desarrollada.
2. Servidor de Pre-Producción y Producción: se adquiere una segunda instancia AWS EC2 (Amazon Elastic Compute Cloud), en la cual se instalará y configurará otro servidor/contenedor de aplicaciones PHP y un servidor MySQL conjuntamente. El objetivo será disponer de un entorno de Pre-Producción para realizar diversas pruebas de mayor especificación, ya integrando los equipos Mikrotik, para efectuar lo que comúnmente se denominan ensayos de campo, en los que se ejecutan diferentes pruebas a lo largo de todo el flujo del proceso del producto. En este ambiente, también se experimentarán versiones candidatas que luego podrán ser liberadas en el ambiente productivo. Conjuntamente con este entorno anteriormente mencionado, dicha instancia será, además, utilizada como entorno productivo (o ambiente de Producción), donde se realizarán implementaciones y puestas a punto con versiones beta y versiones finales. Por último, se registrarán 2 dominios hacia este servidor: a) http://wifi-pre.harvay.com/, donde se publicará la versión de pre-producción. b) http://wifi.harvay.com/, donde se publicará la versión de producción.

3. Servidor Radius: se adquiere una tercera instancia AWS EC2 (Amazon Elastic Compute Cloud), quien será la encargada de autenticar y autorizar el acceso a los dispositivos clientes a la red de internet. Este mismo servidor autorizará a los dispositivos de los 4 ambientes (Desarrollo, Testing, Pre-Producción y Producción) para que puedan hacer uso del producto.

A continuación, se observa un diagrama con las tres (3) instancias AWS EC2y su comunicación entre si:

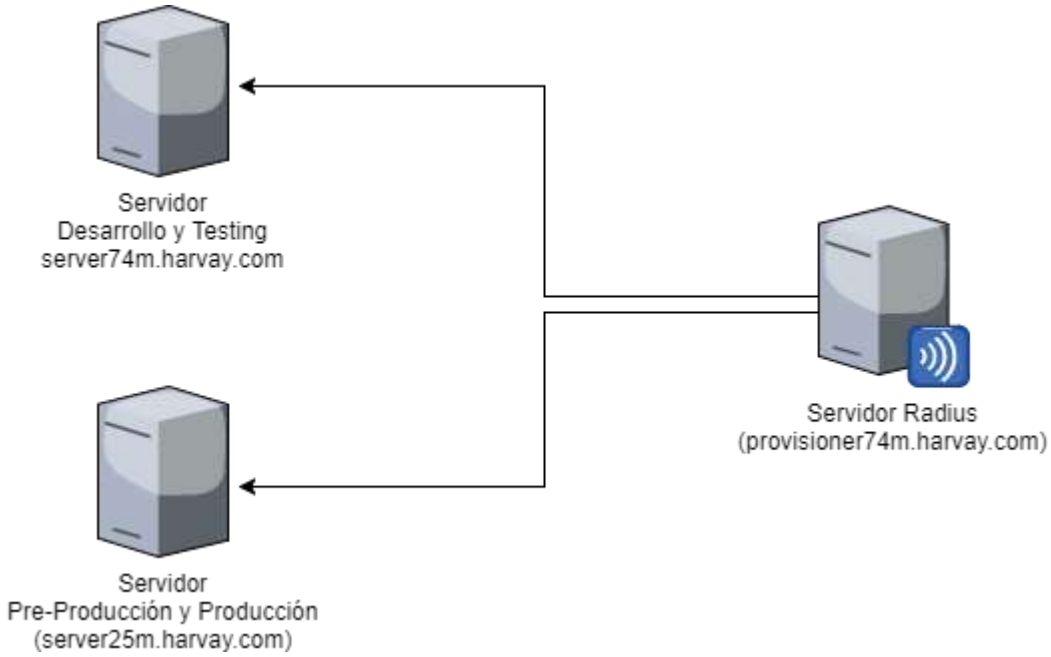


Gráfico 25. **Diagrama simplificado hardware adquirido para Proyecto Wifi Social**

Software requerido

Seguidamente, se enumeran y se describen las herramientas que el equipo utilizará para llevar a cabo dicho trabajo, conjuntamente con las necesidades que atenderán cada herramienta. Éstas, son popularmente reconocidas dentro del área de sistemas/software y facilitarán la comunicación, el sincronismo del equipo y la transparencia del esfuerzo entre los integrantes e interesados en dicho proyecto.

JIRA

Como anteriormente se ha mencionado, la herramienta elegida para gerenciar este proyecto es JIRA, de la empresa Atlassian. Se dispone de una licencia para el uso de este proyecto, la cual ya se encuentra instalada y disponible en un servidor de la empresa. Se puede acceder desde la siguiente URL: http://jira.harvay.com:8080

Como punto de partida, JIRA tendrá un tablero Kanban que será configurado por el líder del equipo y en el mismo se disponibilizarán los requerimientos necesarios para desarrollar el proyecto. Se pretende que el equipo de trabajo analice cada uno de los requerimientos que le serán provistos y que proceda a su construcción y desarrollo de forma eficiente y sincronizada, actualizando el avance día a día en JIRA.

La complejidad de uso de esta herramienta es de nivel básico, ya que supone una utilización direccionada solo al tablero de trabajo Kanban. A su vez, cabe mencionar que el idioma adoptado por la empresa al instalar JIRA fue el idioma inglés. por lo tanto, las opciones de menú, botones en formularios e informaciones generales se visualizarán en este idioma. Sin embargo, como se especificó en puntos anteriores, los requerimientos del proyecto y comunicación del equipo y los interesados se deberá realizar en español.

Skype

Skype será la herramienta para comunicarse internamente entre los miembros del equipo, por medio de un chat grupal, en idioma español. A su vez, se utilizará la característica de llamada y video llamada, para cualquier reunión que sea necesaria durante el período en que el proyecto es desarrollado.

Correo Electrónico

El correo electrónico será utilizado para comunicarse con los interesados en el proyecto, para: a) entregas de informes sobre el proyecto, b) escalamientos formales necesarios, c) quejas puntales y específicas. d) minutas de reuniones.

Estimación de entrega del proyecto

Como se indicó anteriormente en el marco teórico de este documento, las metodologías tradicionales suponen un particularidad distintiva y muy importante, en cuanto a la estimación de la entrega final del proyecto. Dicha estimación es definida al inicio del mismo y se refiere específicamente a la estimación de la fecha de entrega del producto completamente desarrollado y listo para su uso, donde se analiza en forma detallada, cada uno de los requerimientos ya definidos, de forma de proveer a los interesados, una fecha de entrega del proyecto. No obstante, y, por el contrario, siendo este un proyecto ágil de carácter interno, todos los interesados y sus participantes ya poseen conocimientos sobre metodologías ágiles y uno de los puntos más significativos es que, ya han trabajado con dichas metodologías anteriormente.

Por consiguiente, cada uno de los integrantes de este proyecto práctico entienden que conjeturar una estimación de tiempo de entrega del producto al inicio (o lo que frecuentemente se denomina como “definir al inicio, la fecha de entrega”), no es factible de calcular de manera sencilla, ya que es necesario invertir una cantidad de esfuerzo considerable para evaluar detalladamente cada uno de los requerimientos, y además, no existe ningún tipo de garantía de que la estimación proporcionada sea correcta y consistente en el tiempo en el que el proyecto transcurre. Asimismo, muchos de estos requerimientos, todavía ni siquiera se encuentran completamente definidos por los equipos responsables que delimitan el alcance final, y muchos otros, definidos, pero con una alta probabilidad de cambio durante el ciclo de desarrollo, debido a diversos factores externos.

De cualquier manera, con todo lo expuesto no se intenta evitar el hecho de proveer una estimación de fecha de entrega del producto. Por el contrario, la fecha de entrega es vital para que el proyecto tenga un sentido como proyecto en si y que, al mismo tiempo, disponga de un ciclo de desarrollo progresivo y acorde para concretar la meta final. La esencia de definir una fecha de entrega en las metodologías ágiles, es hacerlo durante el proceso de desarrollo de forma natural, mediante diversas técnicas estadísticas que tienen en cuenta factores internos, como la velocidad del equipo y otros

externos. De esta forma, y en comparación con las metodologías tradicionales, se puede invertir menor esfuerzo en estimar una fecha final de entrega al inicio del proyecto y canalizar dicho esfuerzo inicial en el desarrollo de funcionalidades que agreguen valor al objetivo final.

Modelo de documentación del trabajo práctico

Con la finalidad de exponer el desarrollo práctico del proyecto en este documento, resulta necesario definir un modelo de documentación que será utilizado para visualizar las implementaciones de los enfoques Kanban y Scrumban. La variable tiempo ejercerá un fuerte papel, ya que se empleará para distinguir las jornadas laborales del equipo a lo largo del desarrollo. Dicha distinción permitirá analizar, comparar y evaluar cada jornada dentro de cada enfoque y así, obtener conclusiones concretas.

En relación a la identificación de cada integrante del equipo en el tablero de la herramienta, se le pedirá a cada uno de los miembros que incorporé a su perfil de JIRA, una foto que permita identificarlo de manera rápida y clara. Esto ayudará a acelerar el entendimiento de lo que sucede en cada retrato del tablero.

La interacción entre los integrantes del proyecto en este entorno de trabajo ágil, no solo se manifestará sobre el tablero digital de la herramienta JIRA, sino que paralelamente, se referenciarán los hechos más notables por cada jornada. Conjuntamente, se detallarán las reuniones organizadas por el equipo, mediante la minuta correspondiente a cada encuentro.

CAPÍTULO VII

IMPLEMENTACIÓN DE ENFOQUE KANBAN

Inicialmente se pretende utilizar Kanban, ya que es un enfoque claro, simple y, sobre todo, flexible para diversos tipos de proyectos de desarrollo de sistemas. Estos, son factores clave a la hora de seleccionar una metodología de trabajo en la cual, el proyecto sea encuadrado de manera sencilla y a la vez, satisfactoria para todos los participantes. Sin embargo, la elección de este marco no se traduce en la persistencia del mismo, de forma invariable hasta el fin del proyecto. Es decir, no significa que no se permitan realizar adecuaciones, ajustes y cambios cuando sean identificados y necesarios a lo largo del proyecto. Como se indicó en la teoría que respalda este trabajo práctico, las metodologías ágiles permiten una implementación y adecuación progresiva, siempre y cuando cada ajuste, tenga un sentido común y lógico dentro del entorno.

Es por ello que, es parte del alcance de este trabajo de tesis, el de documentar dichas adecuaciones cuando sean identificadas e implementadas.

A continuación, se da inicio al proyecto anteriormente definido, dentro de un marco ágil con enfoque hacia Kanban.

Kick-Off - Primera Reunión

Como primer punto, resulta de importancia afirmar que el inicio oficial del proyecto estaba planificado para la primera semana de Julio de 2017, específicamente el día lunes 3 de Julio, pero por algunos inconvenientes de logística y tiempos del lado del cliente, es inevitable comenzar al día siguiente, martes 4 de Julio.

No obstante, el día lunes 3 de Julio de 2017 el líder del equipo convoca a una reunión llamada kick-off (que en español significa “saque inicial”), con el fin de exponer detalles sobre el proyecto a desarrollar, como su descripción, objetivos, software y hardware a utilizar, enfoque metodológico, riesgos de alto nivel ya

identificados y; expectativas del cliente en cuanto a diseño y requerimientos no funcionales entre otros de los puntos. Se posee información sobre los primeros requerimientos, que si bien, aún no están definidos por completo, ya se dispone de una idea bastante avanzada (y documentada) y, por lo tanto, resulta más que conveniente analizarlos entre todos los integrantes. El equipo de trabajo comienza a adentrarse dentro del proyecto, conociendo sus requerimientos y su ambiente en términos generales.

La reunión procede durante una (1) hora y treinta (30) minutos, se produce en un ambiente relajado y "descontracturado". Se suscita un debate sobre las tecnologías ya previamente seleccionadas y sobre las diferentes opciones de desarrollo e implementación que podrían emplearse. Surgen diversos puntos a tener en cuenta de acuerdo a experiencias pasadas en proyectos similares y, además, pedidos al cliente que son anotados por parte del líder del equipo. Muchos de estos pedidos se encuentran relacionados a algunos de los requisitos. De todas maneras, el equipo tiene siempre presente de que estos requerimientos no están definidos en un cien por ciento y que, de alguna forma, son "borradores" que podrían sufrir cambios en los próximos días.

El punto de mayor consideración relevado durante el encuentro, es en relación a la priorización de los requerimientos. Esto refiere concretamente a que, no existe ningún rol definido dentro del equipo, ni de los participantes en general, que realice la organización y priorización de los requisitos. Como se deduce, es de suma importancia facilitar la definición de este rol antes de dar inicio al desarrollo del producto, para que exista una persona responsable por aclarar, explicar y priorizar requerimientos. Así, el equipo sabrá a que deberá dedicar su esfuerzo de desarrollo en todo momento.

Desde la perspectiva del tablero de Kanban, los requerimientos que se encuentren ordenados al comienzo de la pila (o la "columna"), serán aquellos que ya han sido descriptos, definidos, ordenados, priorizados y que, por ende, se encuentran disponibles para su desarrollo técnico. Por último, se entiende también, que dichos requisitos están íntegramente alineados con los departamentos de negocios y comercial del cliente y su entorno.

Request Manager, nuevo rol

A pedido específico del equipo de trabajo, se crea un nuevo rol para describir, definir, priorizar y ordenar los requerimientos en el tablero Kanban. Dicho rol es denominado “Gerente de Requerimiento” (del inglés, “Request Manager”). El mismo surgirá desde el equipo del cliente y tendrá algunas similitudes con el rol de Product Owner (enfoque Scrum).

Seguidamente, se definen otras responsabilidades que asumirá el perfil, además de las mencionadas anteriormente: a) comprensión de las necesidades del cliente. b) gestión activa con las partes interesadas del proyecto. c) sincronización continua sobre los ajustes y/o cambios en los requisitos. d) análisis, entendimiento y mapeo de los procesos de negocio que podrían impactar al producto. e) comunicación y escalamiento de problemas y riesgos a los departamentos pertinentes.

Es decir, será el nexo responsable de la comunicación constante con el departamento de negocio y departamento comercial del cliente, para que, en todo momento, se entienda cuál es la característica del producto que posee mayor prioridad a lo largo del desarrollo del proyecto.

En el siguiente cuadro, se define la persona que ocupará el rol de Request Manager:

Cuadro 7. Perfil Profesional para Request Manager.

Perfil Profesional	Miembro de equipo seleccionado
Request Manager: Lucas Rodrigo Benitez (argentino, localizado en Córdoba Capital, Córdoba, Argentina).	5 años de experiencia en gestión de proyectos IT, con experiencia en la implementación de metodologías ágiles. Nivel de español nativo, portugués intermedio e inglés avanzado.

Puesta a punto de JIRA

A los fines de configurar la herramienta para que atienda a los requerimientos del proyecto, se realiza una configuración y puesta a punto básica, utilizando parámetros que la propia herramienta ya dispone de forma predeterminada. Esto, sin duda agiliza su configuración y puesta a punto para iniciar el proyecto sin dedicar esfuerzo inicial a la configuración de dicha herramienta. Posteriormente se analizarán e implementarán cambios dentro de la misma, en función de las necesidades del proyecto y sus participantes, para trabajar de la forma más cómoda posible, en el ambiente más consistente posible.

Seguidamente, se detallan las configuraciones realizadas:

Proyecto en JIRA: Wifi Social

Se crea un proyecto denominado “Wifi Social”, con las siglas “WS” como identificador del mismo dentro de la herramienta. Esto quiere decir, que cada uno de los ítems que se añadan, poseerán una nomenclatura con el formato “WS-XX”, siendo “XX” un número correlativo que identificará a un requerimiento, defecto, actividad o tarea relativa al proyecto, que deberá ser tratado por el equipo de desarrollo.

A su vez, el departamento de diseño, envía el logo del producto que será incorporado para su individualización en relación a otros proyectos en la herramienta.

Se proporciona los accesos a los diferentes participantes de este proyecto. Una vez confirmados los mismo, se configuran los privilegios para cada usuario en función de su rol y se establecen los paneles de control para cada uno.

Flujo de Trabajo (JIRA)

El flujo de trabajo (en inglés, “workflow”), que será utilizado en el proyecto, es el que dicha herramienta ofrece de forma predeterminada. El mismo es simple, potente y flexible para el trabajo en grupo.

Se identifican cuatro (4) estados que se utilizarán en el proyecto. Los mismos se muestran a continuación en la siguiente imagen:

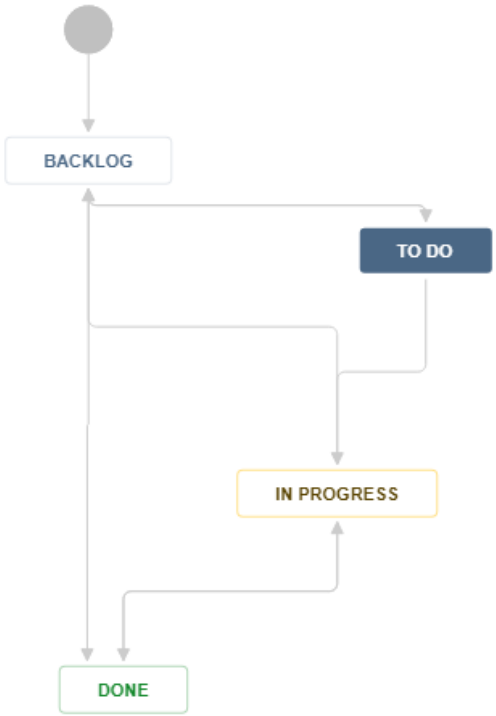


Gráfico 26. **Flujo de trabajo del proyecto JIRA**

Cada estado corresponde a una fase dentro del proceso de desarrollo:

1. Backlog: o también llamada “pila de trabajo”. En este estado se ubicarán todos los requerimientos que todavía no se encuentren listos para trabajar por parte del equipo. Es decir, requerimientos que todavía están siendo trabajados por el Request Manager y el cliente, que pueden sufrir cambios en su alcance o su orden de priorización. Este estado resulta interesante para el equipo ya que puede adelantarse de cierta forma, conociendo los requerimientos que le llegarán en el futuro. Sin embargo, no se les debe dedicar esfuerzo de desarrollo.
2. To Do: en español significa “Para hacer”. Este estado es donde se ubican todos aquellos requerimientos que se encuentran listos para ser trabajados por el equipo. Los mismos ya se encuentran definidos y priorizados. Sin duda, el equipo puede trabajarlos.

3. In Progress: en español, “en progreso”. Se refiere a todos los requerimientos que están siendo trabajados actualmente por parte del equipo de desarrollo.
4. Done: en español, “Hecho”. Son todos aquellos requerimientos que ya se encuentren desarrollados en su integridad, es decir, completados. Se utiliza este estado para demostrar que han sido trabajados en su totalidad y se encuentran listos para ser mostrados al Request Manager y cliente para su aprobación.

Prioridades en JIRA

Es importante remarcar las prioridades que se utilizarán durante el proyecto. Cada prioridad es representada por un nombre de prioridad y un ícono. A continuación, se listan dichos nombres con sus íconos, desde la más importante a la menos importante:

Gráfico 27. **Prioridades utilizadas en el proyecto “Wifi Social”**

Tablero Kanban

Se dispone de un tablero Kanban con cuatro (4) columnas que visualizan el flujo de trabajo por el que transcurrirán los requerimientos a desarrollar. Cabe aclarar que, desde este punto, dichos requerimientos serán llamados “tarjetas” o también, en inglés, “cards”.

Dicho tablero digital se podrá visualizar en la siguiente dirección web, solo por miembros del proyecto, donde los mismos podrán interactuar con las tarjetas.

En la siguiente imagen, se muestra la configuración de dicho tablero:

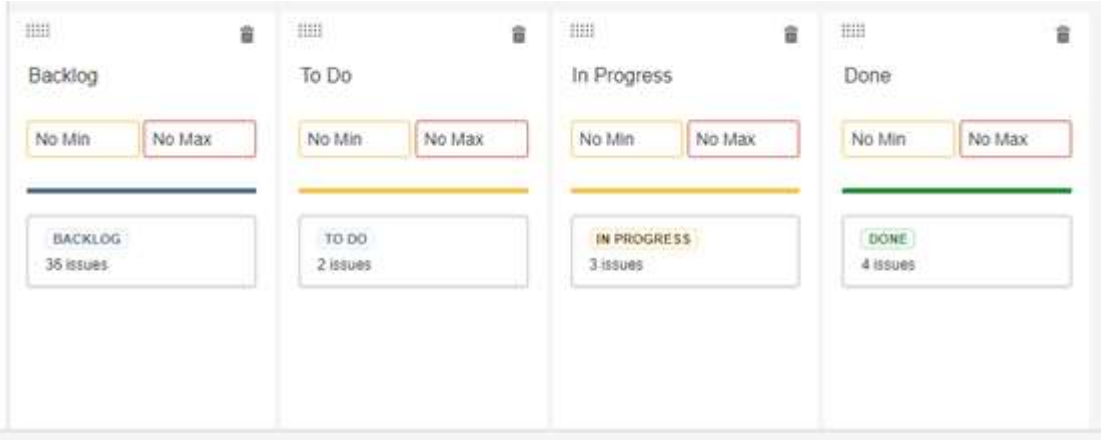


Gráfico 28. **Configuración de tablero Kanban (JIRA)**

Una vez más, la configuración establecida es la que ofrece JIRA de forma predeterminada con un flujo de trabajo básico, que utiliza un estado por cada una de sus columnas. Es decir, las tarjetas que posean el estado “Backlog”, se ubicarán en la columna “Backlog”. Las tarjetas con el estado “To Do”, se ubicarán en la columna “To Do”. Y lo mismo sucederá con los otros dos (2) estados “In Progress” y “Done”.

Perfiles de los participantes en JIRA

A continuación, se visualizan los perfiles JIRA de cada uno de los participantes que interactuarán en el tablero de Kanban en dicha herramienta:

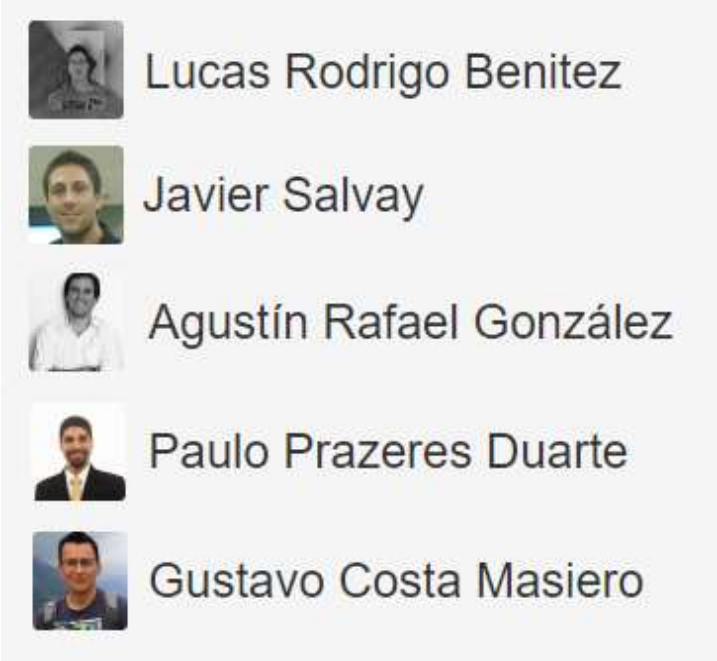


Gráfico 29. **Participantes del proyecto "Wifi Social"**

Requerimientos iniciales

El día lunes 3 de Julio de 2017, el Request Manager completa la definición de las primeras cards, que son posicionadas en el tablero Kanban para su futuro análisis por parte del equipo. Las mismas, ya se encuentran ordenadas según su prioridad.

Se muestra una imagen con dicho tablero y los denominados requerimientos iniciales:

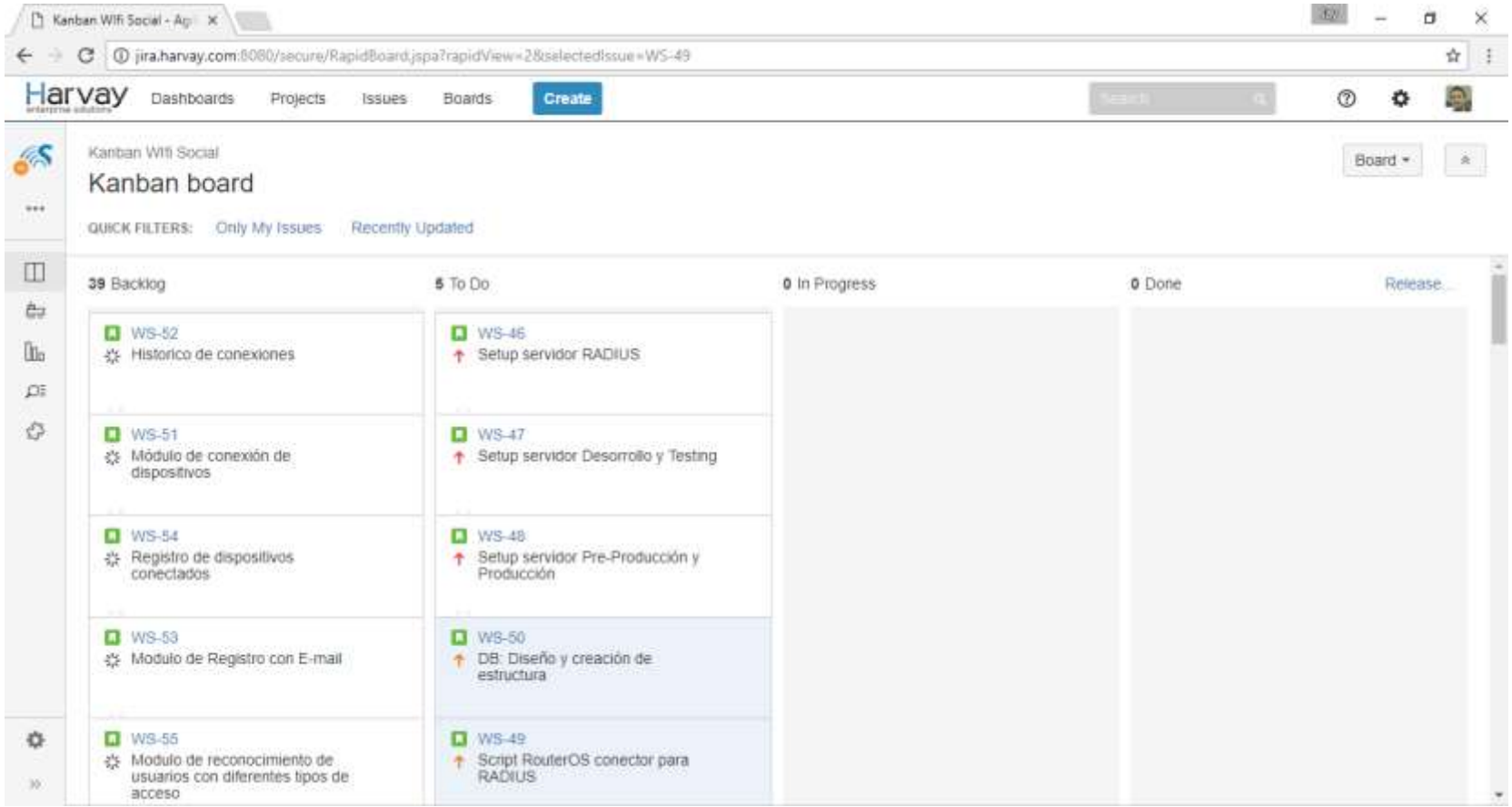


Gráfico 30. **Requerimientos iniciales en el Kanban**

El Request Manager “mueve” dos (2) requerimientos con los identificadores WS-49 y WS-50 (marcados en la imagen en color celeste), desde la columna “Backlog” hacia la columna “To Do”, asignándoles sus respectivas prioridades. Nótese que las prioridades de los diferentes cards son representadas en flechas de diferentes colores. Este es el “guiño verde” (ergo, la señal que indica que pueden proceder), con el desarrollo de los mismos.

Asimismo, el líder del proyecto ya ha creado 3 cards (WS-46, WS-47 y WS48), que se refieren a la preparación de los diferentes ambientes. Y es por eso que, el Request Manager ha enviado los 2 cards mencionados anteriormente hacia el final de la columna, teniendo el entendimiento de que es necesario configurar primero los ambientes, para luego comenzar a trabajar en el desarrollo puro del producto.

Primera semana de proyecto

Con el objetivo de estructurar el desarrollo y entendimiento de este documento, a continuación, se identifican cada uno de los días de trabajo con su respectivo detalle de la jornada. Se pretende exponer el desarrollo del proyecto y la aplicación del enfoque Kanban en función del tiempo transcurrido.

Como todo primer día dentro de un nuevo proyecto, el mismo supone una ambientación progresiva por parte del equipo y un análisis sobre las actividades a desarrollar.

Día 1 - 4 de Julio de 2017

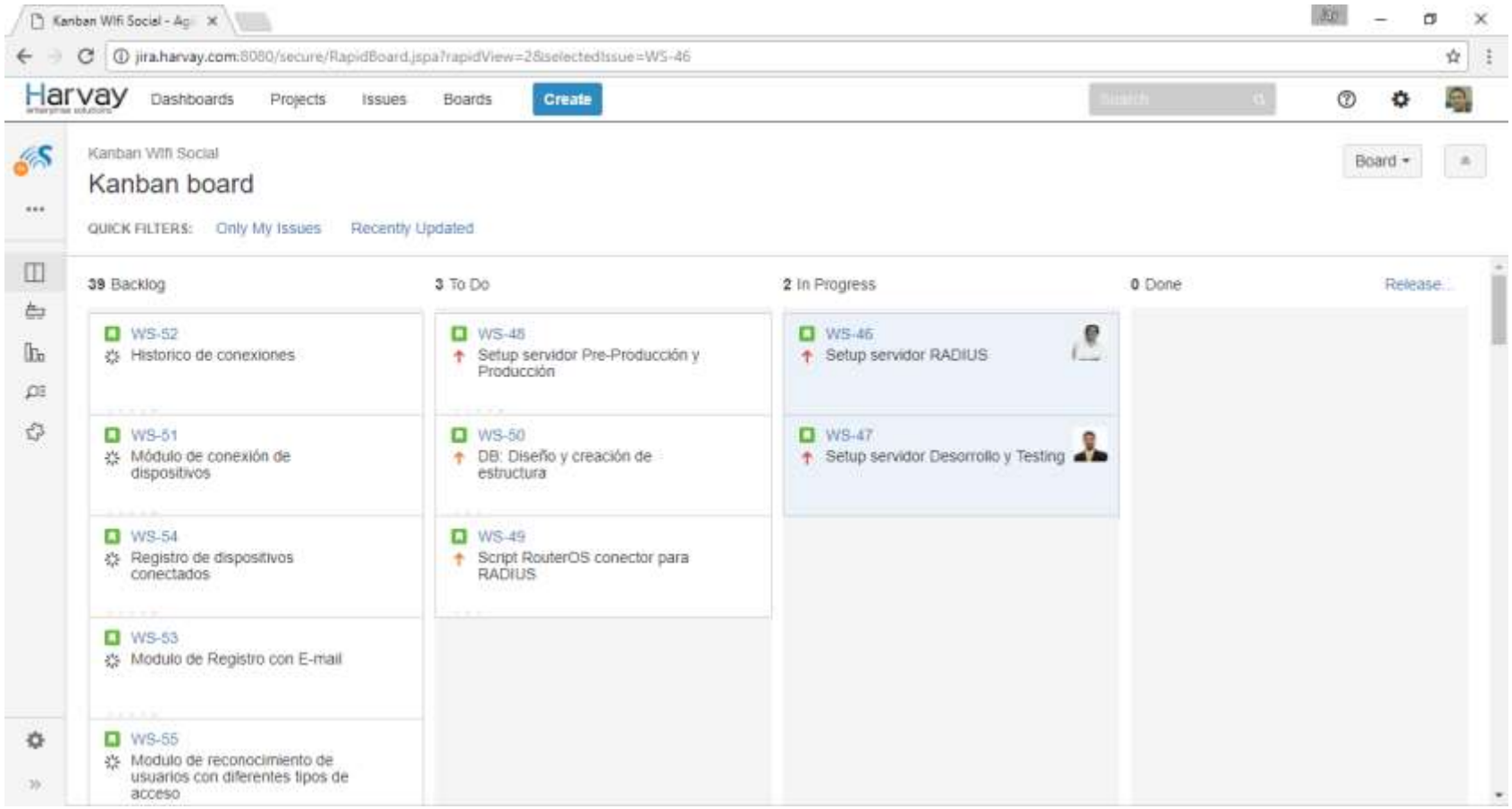


Gráfico 31. **Tablero Kanban Día 1 (semana 1)**

Detalle Jornada: Día 1 - 4 de Julio de 2017

Se inicia el desarrollo en dos (2) tarjetas relacionadas al setup (en español, comúnmente denominado “configuración”), del servidor Radius (WS-46), y paralelamente, al setup del servidor para ambas instancias de desarrollo y testing (WS-47). Se observa que Agustín (desarrollador senior) y Paulo (desarrollador semi-senior) se encargan de dichas actividades (ver fotos de los integrantes en cada una de las tarjetas).

Durante la jornada laboral, Paulo le consulta a Agustín sobre algunos detalles comunicacionales entre el servidor Radius y las instancias de Desarrollo y Testing para que se encuentren encriptadas a la hora de procesar transacciones de aprovisionamiento. Al final del día, ambas actividades se encuentran completas con su correspondiente documentación que se almacena en un archivo de texto compartido en la nube. Javier (líder de equipo), verifica la documentación para analizar si la misma está completa y si puede ser utilizada como guía para el setup de las próximas instancias.

Gustavo (analista de calidad), analiza la documentación compartida por Lucas (request manager), de forma de entender y emprender el proceso de construcción del plan de calidad. La finalidad del mismo es, cubrir todas las áreas importantes del producto con el registro de casos de pruebas que posteriormente serán verificados sobre las versiones del producto. Cabe destacar que este plan de calidad o de testing, es un documento de alto valor que contendrá toda la información a tener en cuenta, en relación a la estrategia y ejecución de casos de pruebas a lo largo de todo el proyecto.

Asimismo, todo el equipo concibe la importancia de que la documentación referente al desarrollo y al proyecto en si, tanto en términos generarles como específicos, debe ser registrada y compartida entre sus integrantes en todo momento, desde el inicio hasta el fin. Este es un eje fundamental para convertir el desarrollo del proyecto en un éxito, ya que se pretende crear gradualmente, una base de datos de conocimiento en la cual se pueda encontrar cualquier tipo de información que ayude al desarrollo del producto. Así, se estará contribuyendo no solo al proceso de desarrollo,

sino a la puesta a punto y primordialmente al mantenimiento posterior en el ambiente productivo.

Día 2 - 5 de Julio de 2017

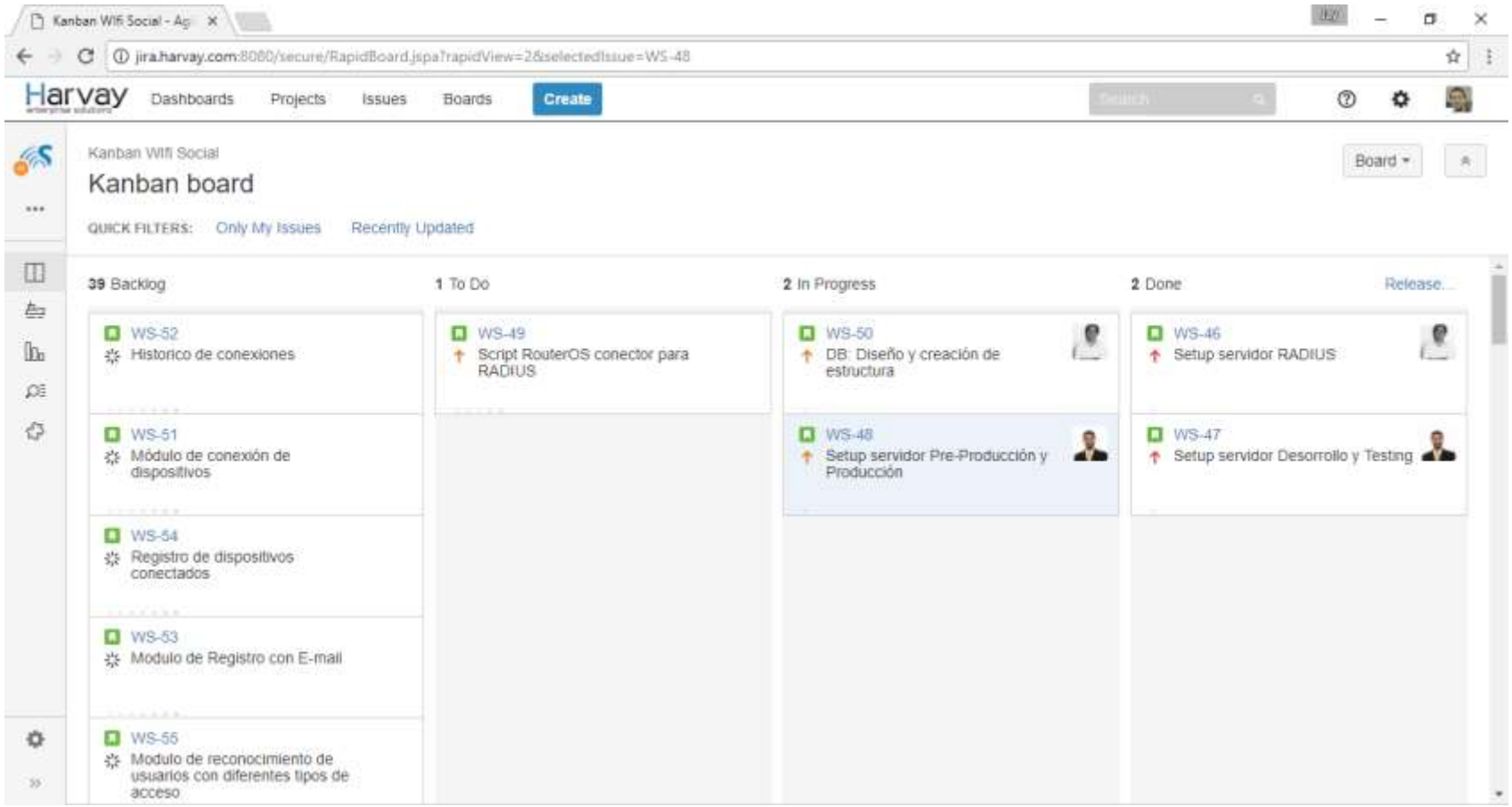
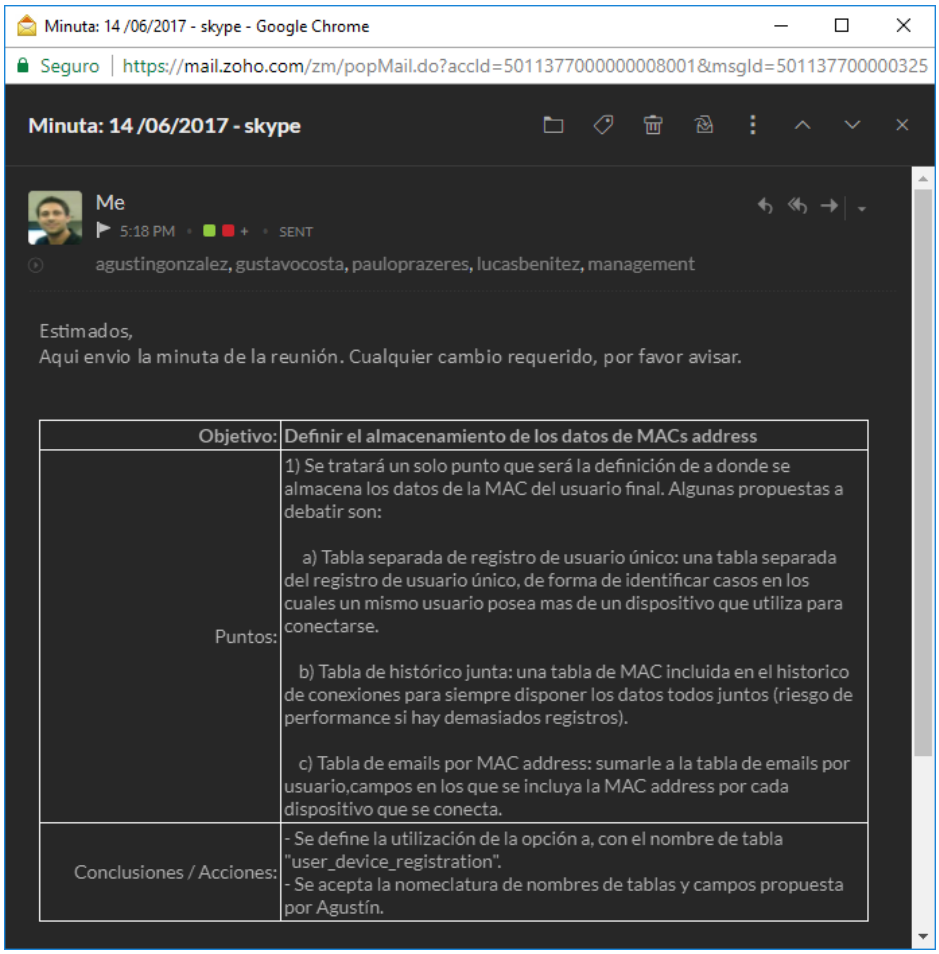


Gráfico 32. **Tablero Kanban Día 2 (semana 1)**

Detalle Jornada: Día 2 - 5 de Julio de 2017

La jornada lleva a un gran desafío, ya que se pretende trabajar en la creación y desarrollo de la base de datos para el sistema (WS-50). Agustín comienza a trabajar en el diseño de la misma, mientras que Paulo realiza la configuración del servidor pendiente, en el que se situarán las instancias de Pre-Producción y Producción (WS-48). Este último miembro del equipo, utiliza la guía de setup anteriormente creada en la WS-47 (verificada por Javier) y luego de seguirla, encuentra oportunidades de mejoras que se refieren a automatizaciones de scripts, que, sin duda, acelerarán el setup a futuro en otras instancias que se deseen configurar.

Por otro lado, Agustín identifica diferentes riesgos que hacen que el equipo tenga que reunirse para analizar algunos factores de cada uno de los ambientes relacionados al performance de la base de datos. A continuación, se documenta la minuta de la reunión:



Minuta: 14 /06/2017 - skype - Google Chrome

Seguro | https://mail.zoho.com/zm/popMail.do?accId=501137700000008001&msgId=501137700000325

Minuta: 14 /06/2017 - skype

Me
5:18 PM • SENT
agustingonzalez, gustavocosta, pauloprazeres, lucasbenitez, management

Estimados,
Aqui envio la minuta de la reunión. Cualquier cambio requerido, por favor avisar.

Objetivo:	**Definir el almacenamiento de los datos de MACs address**
Puntos:	1) Se tratará un solo punto que será la definición de a donde se almacena los datos de la MAC del usuario final. Algunas propuestas a debatir son: a) Tabla separada de registro de usuario único: una tabla separada del registro de usuario único, de forma de identificar casos en los cuales un mismo usuario posea mas de un dispositivo que utiliza para conectarse. b) Tabla de histórico junta: una tabla de MAC incluida en el historico de conexiones para siempre disponer los datos todos juntos (riesgo de performance si hay demasiados registros). c) Tabla de emails por MAC address: sumarle a la tabla de emails por usuario,campos en los que se incluya la MAC address por cada dispositivo que se conecta.
Conclusiones / Acciones:	- Se define la utilización de la opción a, con el nombre de tabla "user_device_registration". - Se acepta la nomeclatura de nombres de tablas y campos propuesta por Agustín.

Gráfico 33. **Minuta de reunión**

Día 3 - 6 de Julio de 2017

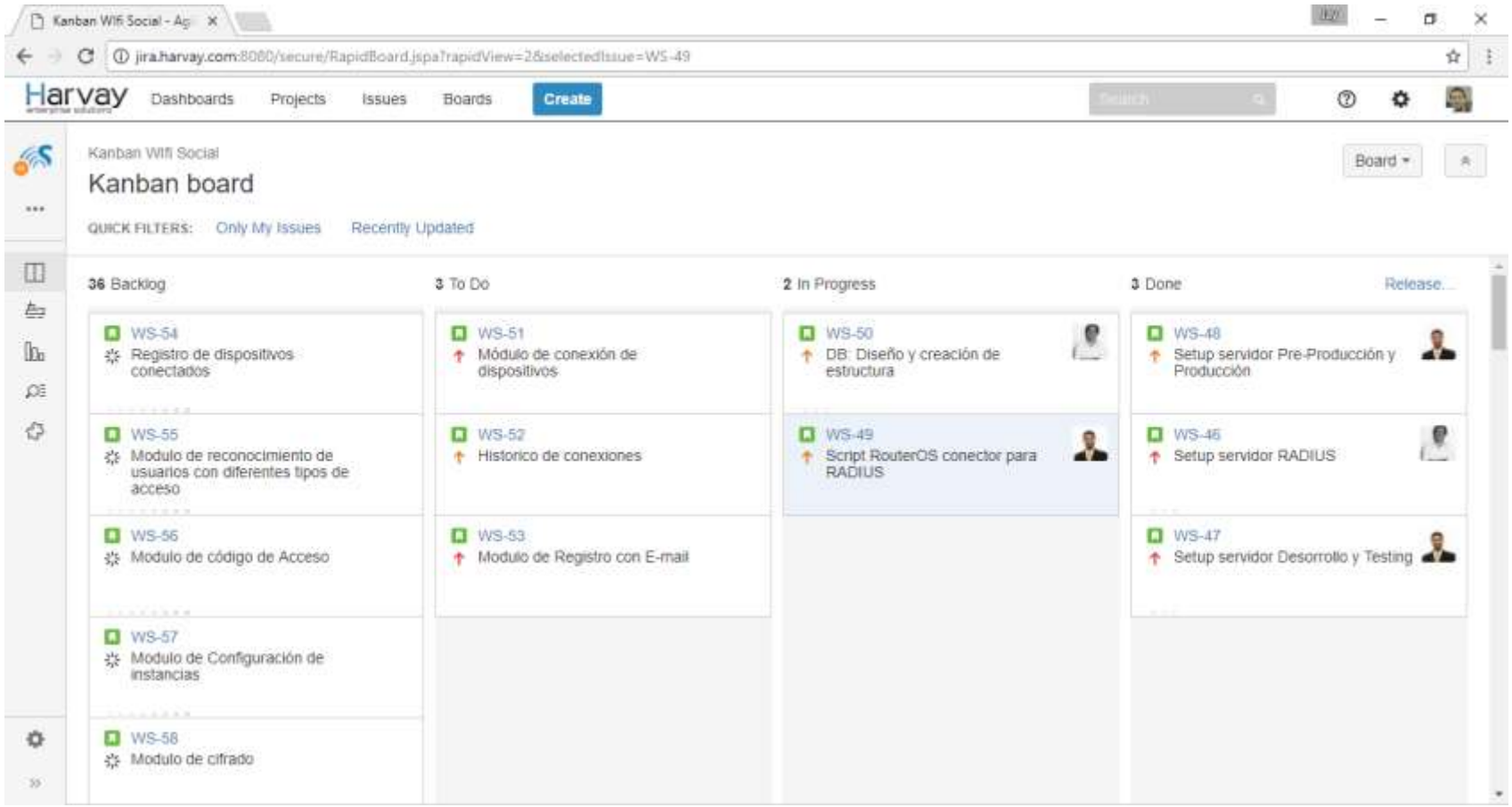


Gráfico 34. **Tablero Kanban Día 3 (semana 1)**

Detalle Jornada: Día 3 - 6 de Julio de 2017

Cabe destacar que si bien no se están efectuando y comunicando estimaciones formales por cada card al request manager (Lucas), el equipo si debate, y muchas veces discute los tiempos de cada actividad, de modo de poder organizar los esfuerzos conjuntos durante toda la jornada. Es por eso que, a raíz de una subestimación sobre la tarjeta “DB: Diseño y creación de estructura” (WS-50), el equipo entiende que se encuentra atrasado con la entrega de este card, y también entiende que el mismo es vital para la construcción de las bases del sistema.

Consecuentemente, Javier le pide a Paulo pausar las actividades sobre la card WS-46, para ayudar a Agustín con el desarrollo y documentación de la tarjeta referida a la base de datos. Desde un punto de vista técnico, dicha tarjeta requiere un análisis pertinente y minucioso para construir los cimientos a donde se montará todo el sistema. Ambos desarrolladores consiguen acelerar dicha construcción casi en su totalidad, pero no se observan cambios en el tablero Kanban.

Representación de esfuerzo testing

Gustavo continúa trabajando en la creación de todo el plan de calidad (también denominado en inglés, “test plan”), para el producto e identifica un punto interesante que es compartido y debatido con Javier. Específicamente, surge la necesidad de mostrar este esfuerzo dentro del tablero, de forma de también alinear al request manager desde una perspectiva de calidad del producto.

En este punto, cabe destacar que si bien este trabajo relacionado a calidad, no se encuentra representado en el tablero Kanban de forma directa (mediante una tarjeta específica), el mismo se localiza fragmentado de forma implícita en cada una de las tarjetas que involucran desarrollo. Por lo tanto, se define que resulta efectivo y práctico, el hecho de crear una tarjeta dentro del tablero Kanban, que represente puntualmente el análisis detallado y por, sobre todo, la creación de todos los casos de prueba, que serán futuramente ejecutados sobre los componentes desarrollados. Se envía un email para comunicar y alinear al resto de los participantes del proyecto.

Día 4 - 7 de Julio de 2017

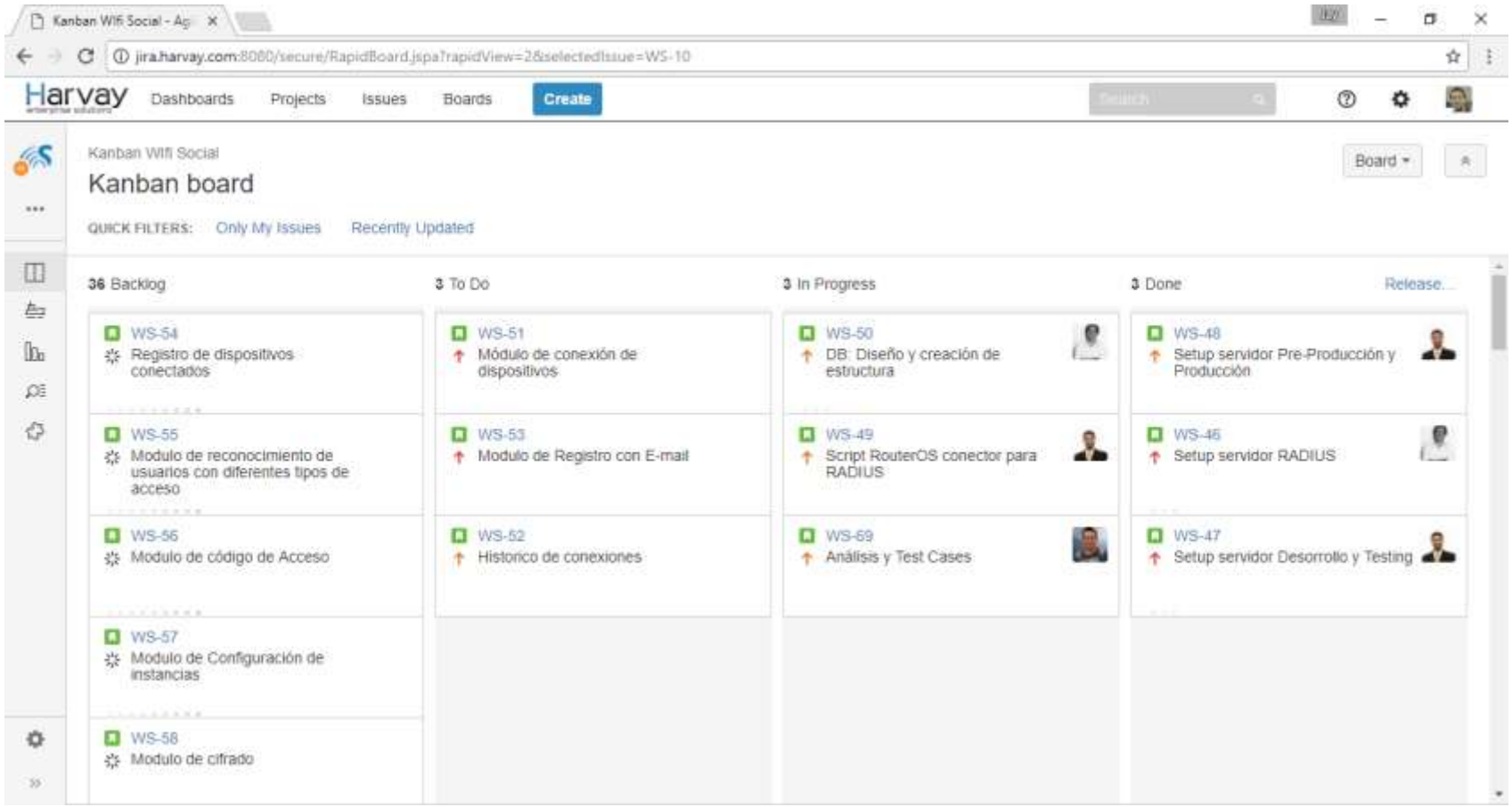


Gráfico 35. **Tablero Kanban Día 4 (semana 1)**

Detalle Jornada: Día 4 - 7 de Julio de 2017

En el cuarto día de la primera semana, cada miembro del equipo continúa trabajando en sus actividades. La más importante de todas es aquella relacionada con la base de datos (WS-50), que necesita ser finalizada cuanto antes para comenzar a desarrollar el sistema sobre una base de datos sólida. Luego de realizar un trabajo en conjunto entre Agustín y Paulo, realizando reuniones técnicas y también sesiones de pantalla compartida por Skype, se define un modelo y al final de la jornada, se concluye el desarrollo de dicho requerimiento.

A seguir, Paulo continua su actividad pausada en la tarjeta del script (WS-49), y si bien, la misma no es completada en el mismo día, con la ayuda de Agustín, logran debatir algunas ideas que permiten realizar un amplio progreso en la definición del concepto del script, para que, de esta forma, se simplifique el trabajo de desarrollo de Paulo al día siguiente.

Como fue determinado por Gustavo y Javier, se crea la tarjeta WS-69, que se refiere a todo el esfuerzo dedicado al análisis y creación del plan de testing. En ella se define toda la información que se realizará durante el desarrollo de la misma, estableciendo el alcance y los objetivos del plan.

Un punto interesante a tener en cuenta es sobre la tarjeta WS-50, que, al ser completada, Gustavo ya dispone de “luz verde” para dar inicio a la validación con el uso de los casos de prueba creados hasta el momento y también, una validación general del plan de testing. Así, se pretende realizar un triple cruce de validaciones entre: a) la estructura de la base de datos generada, b) los casos de prueba ya generados en el plan de testing, y c) la documentación de diseño provista originalmente por el cliente. Esta actividad, deberá ser realizada una vez que el plan de testing sea definido por completo.

Día 5 – 10 de Julio de 2017

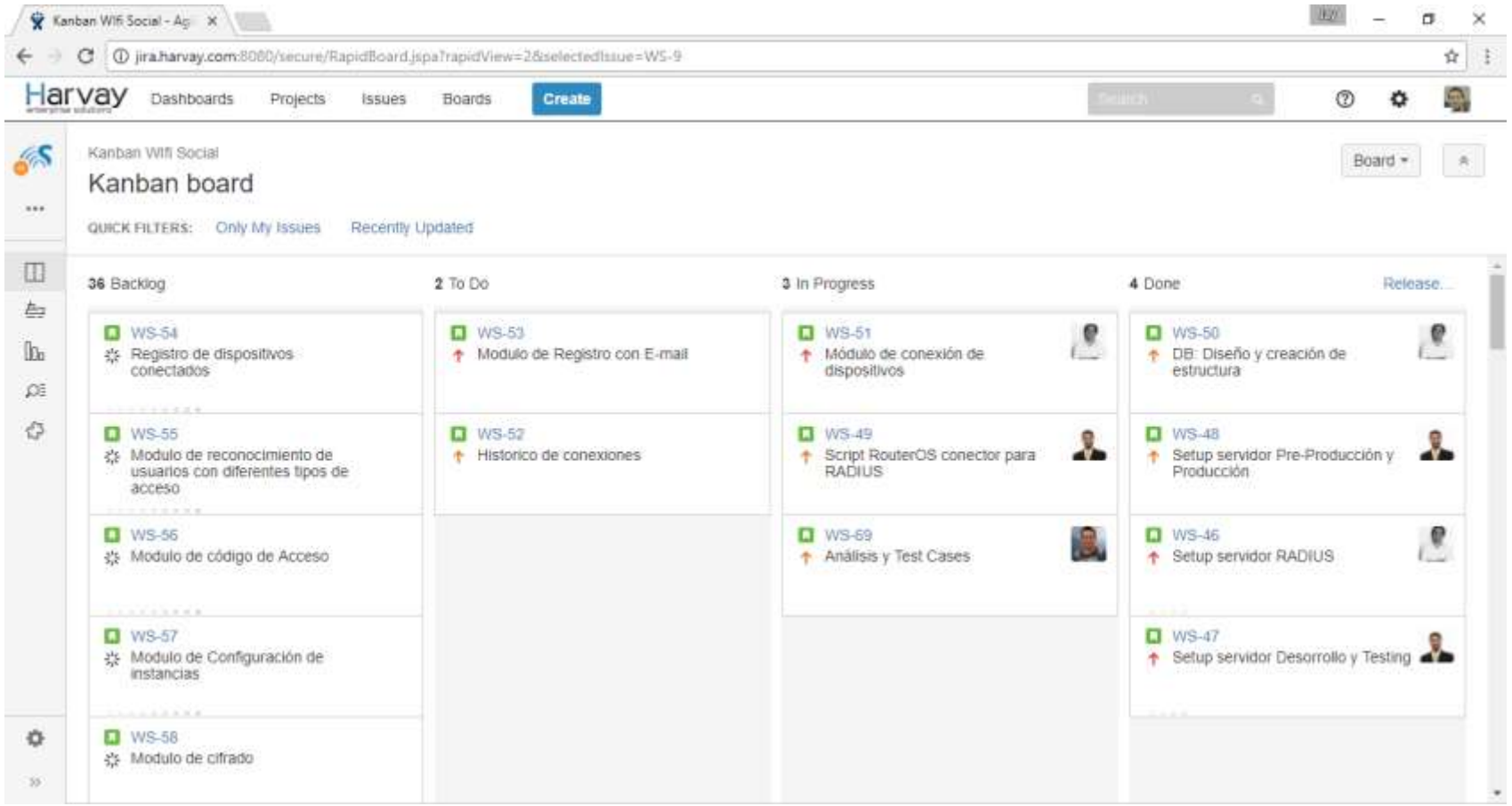


Gráfico 36. **Tablero Kanban Día 5 (semana 1)**

Detalle Jornada: Día 5 – 10 de Julio de 2017

El último día de la primera semana comienza de forma productiva, ya que Paulo rápidamente finaliza el card WS-49 y está listo para tomar el próximo desarrollo. En este caso, la prioridad que sigue en la lista de "To Do" es la tarjeta WS-53 "Módulo de registro con E-mail".

Por otro lado, Agustín comienza a definir y desarrollar el módulo de conexión de dispositivos (WS-51), el cual será vital para el funcionamiento del producto. Este módulo concentra todo el handshaking (es decir, el proceso automatizado que establece la forma en la cual se comunicarán las entidades de red), entre el servidor RADIUS, el router Mikrotik y el sistema siendo desarrollado.

Gustavo completa el plan de testing con los casos de pruebas y el mismo es entregado a Javier para una validación, conjuntamente con sus baterías de casos de pruebas ya definidos en un modelo en particular, utilizando hojas de cálculos. Luego de realizar una revisión, Javier identifica algunos cambios menores que necesitan ser aclarados y/o validados antes de entregar dicho plan de testing al request manager. Cabe aclarar que, en el mismo día, dicho documento queda corregido y validado en un cien por ciento.

El formato seleccionado para documentar dichos casos de pruebas, es en formato de columnas, dentro de una plantilla de hoja de cálculo. En la misma se han creado sesenta (60) casos de pruebas iniciales, que servirán para validar las diferentes versiones del producto en diferentes instancias del proceso de desarrollo.

A continuación, se muestra una imagen parcial de la planilla de casos de pruebas, con dos (2) casos de pruebas para dispositivos móviles con sistema operativo Windows Phone:

Project Name / Module:	Test Designed by:	Gustavo	Test Executed by:	N/A	Test Designed date:	06/28/2017
WiFi Social / Login	Release Version:	5.0	Test Priority:	Alta	Test Execution date:	TBD
	Ambiente :	N/A		Comentarios Generales: PEQL		

Test Suite	Test Case	Test Title	Test Summary	Pre-condition	Test Steps	Test Data (especifico)	Expec
WS-Windows Phone	WS0001	Registro y conexión con Facebook	Registrarse y conectarse en la aplicación a través de Facebook	1- Disponer de cuenta de Facebook. 2- Dicha cuenta de Facebook no dispone de la aplicación de WifiSocial activa.	1- Conectarse a la wifi. 2- Clickear en "Conectate con Facebook". 3- Completar datos de cuenta de Facebook e "Iniciar Sesion". 4- Clickear botón "Continuar como" (plataforma Facebook). 5- Clickear botón "Continuar" (plataforma WifiSocial). 6- Ingresar el código de acceso y luego clickear en "Conectar ahora". 7- Ingresar al navegador y acceder a www.lavoz.com.ar.		1- La wifi se encuentra abierta y la conexió segundos, el mismo dispositivo muestra au en la que se observa el logo, los botones d condiciones. Estos últimos ya se encuentra 2- Lleva a la página de login de Facebook. 3- Se observa la pantalla en la que Faceboo del perfil y cuales se compartirán. En la opc detalles. 4- Se muestra el carrusel de publicidad con esperar hasta que dicho botón se active. Lu botón, pero activo con el label "Continuar" 5- Se muestra la pantalla de solicitud de co 6- Desaparece el campo de texto y el botón con la palabra "Cargando". Esperar a que se desaparecerá. 7- Se muestra la web solicitada y se compru
WS-Windows Phone	WS0002	Login con Facebook	Loguearse en la aplicación con Facebook	1- Estar registrado con cuenta de Facebook.	1- Conectarse a la wifi. 2- Clickear en "Conectate con Facebook". 3- Completar datos de cuenta de Facebook e "Iniciar Sesion" (plataforma Facebook). 4- Clickear boton "Continuar" (plataforma WifiSocial). 5- Ingresar el código de acceso y luego clickear en "Conectar ahora". 6- Ingresar al navegador y acceder a www.lavoz.com.ar.		1- La wifi se encuentra abierta y la conexió segundos, el mismo dispositivo muestra au en la que se observa el logo, los botones d condiciones. Estos últimos ya se encuentra 2- Lleva a la página de login de Facebook. 3- Se muestra el carrusel de publicidad con esperar hasta que dicho botón se active. Lu botón, pero activo con el label "Continuar" 4- Se muestra la pantalla de solicitud de co 5- Desaparece el campo de texto y el botón con la palabra "Cargando". Esperar a que se desaparecerá. 6- Se muestra la web solicitada y se compru
WS-	WS0003	Login Facebook SIN publicidad	Loguearse con	1- Estar registrado con	1- Conectarse a la wifi.		1- La wifi se encuentra abierta y la conexió

Gráfico 37. **Ejemplo de casos de prueba**

Sobre esta planilla de calidad, se pretende trabajar a lo largo de todo el proyecto, de forma de incorporar la mayor cantidad de casos de prueba posible, que permitan eliminar la incertidumbre en cuanto a la calidad del producto. Es decir, mientras mayor cantidad de casos de prueba puedan ser consumados para validar cada una de las versiones, menor cantidad de problemas y/o defectos surgirán durante el uso del producto en el ambiente productivo. Y, por consiguiente, esto generará un aumento en la confianza de los usuarios finales e igualmente, un aumento en el porcentaje de cobertura de la calidad del software. En conclusión, lo expuesto, es la expectativa plena del cliente en cuanto a este proyecto.

Reunión semanal (primera semana)

Durante la misma quinta jornada laboral, Javier convoca a todo el equipo para realizar una reunión semanal sobre lo sucedido durante el periodo, de forma de analizar, corregir y evaluar todo el esfuerzo invertido.

A continuación, se exhiben los puntos concretos que fueron tratados, con sus correspondientes acciones:

Pedido de mayor detalle en requerimientos

Ambos desarrolladores expresan que los requerimientos que han encontrado en JIRA durante este período, se encuentran definidos de forma escueta y demasiado sucinta, dejando muchos detalles a libre elección, lo cual puede llevar a subjetivismos que pueden provocar grandes problemas en el futuro.

Rápidamente, Javier levanta este riesgo y contacta a Lucas para organizar una reunión durante la semana siguiente, de forma de corregir esto a la brevedad. Paulo propone utilizar un modelo de formulario dentro de JIRA, en el cual existan ciertos parámetros que sean obligatorios, para crear un ítem en la herramienta.

Esto es algo que Javier llevará a la mesa de discusión con Lucas, para intentar negociar un ajuste que controle este aspecto en la herramienta.

Despliegue de actividades por tarjeta

De acuerdo a lo expuesto por el equipo, existen tarjetas que pueden requerir el esfuerzo de más de un integrante del equipo. Un ejemplo claro que es expuesto por Gustavo, es el hecho de que al disponer de casos de pruebas que aplican para casi cualquier componente a desarrollar, es necesario disponer de una tarea que se encuentre asociada a ese requerimiento.

Agustín y Paulo coinciden con la implementación de esta idea, ya que comentan que en el ítem WS-50 (“DB: Diseño y creación de estructura”), fue un aporte conjunto para poder finalizarlo. Por último, Javier también coincide, ya que todos los puntos expuestos son válidos y, también, desde una perspectiva de métricas y estadísticas, esto generaría una posibilidad de crear una matriz de trazabilidad para el proyecto. Es decir, partiendo desde un requerimiento en particular, se podría saber cuánto fue el esfuerzo total del equipo y también, cuantos defectos fueron encontrados asociados a ese requerimiento.

Asimismo, podrían generarse otros indicadores como las áreas del sistema que presentan mayores defectos o problemas. Javier expresa que este es un cambio que puede afectar el uso de JIRA por parte del cliente, y es por eso que se decide primero dialogar con el request manager para pedir autorización y si fuese el caso, negociar este punto.

Estado de “Ready for Validation”

Se debate la necesidad de incluir una nueva columna o fase dentro del tablero Kanban, que posibilite la organización de ítems que deben pasar por una validación. Este punto es ampliamente aceptado por parte de todos los integrantes y no produce ningún impacto al cliente. Por lo tanto, Javier toma la acción de realizar el cambio en JIRA y comunicar al request manager.

Definición de un valor WIP

Como anteriormente ha sido descripto en el marco teórico, este punto es muy importante, ya que ayudará a identificar aquellos puntos en donde el proceso de desarrollo se ralentiza por alguna causa determinada. Esta acción de identificar estos puntos dentro del proceso es comúnmente denominada “identificación de cuellos de botella”. El equipo debate sobre el valor para cada fase en el tablero Kanban y establece lo siguiente: a) “Backlog”: sin WIP. b) “To Do”: 4 WIP. c) “In Progress”: 2 WIP. d) “Ready for Validation” (nueva fase): 2 WIP. e) “Done”: sin WIP. Los mismos serán implementados y comunicados para el request manager por parte de Javier.

Todos los puntos arriba mencionados resultaron en acciones concretas propiamente comunicadas a todos los interesados, a través de la siguiente imagen que corresponde a la minuta enviada por email:

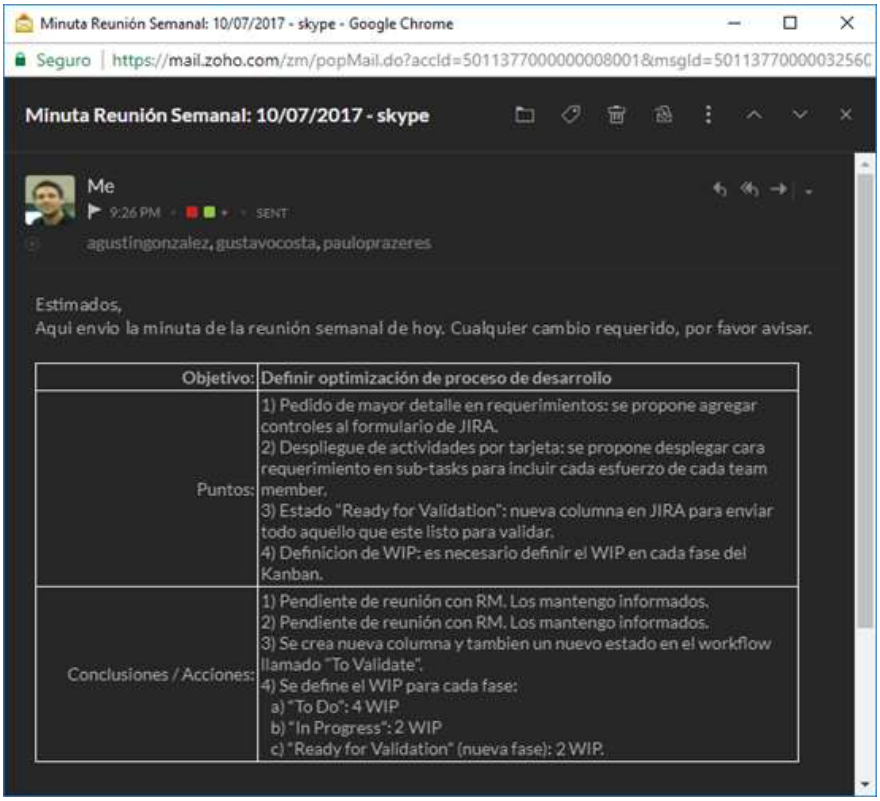


Gráfico 38. **Minuta de reunión semanal**

Segunda semana de proyecto

Inicia la segunda semana con un pedido importante por parte del request manager al líder de proyecto, para chequear todos los ítems que fueron completados en la semana anterior. Javier le comenta a Lucas que, estos ítems pueden ser verificados en cada una de sus instancias en los servidores, ya que se refieren exclusivamente a la preparación de ambientes y también, a la creación de la base de datos en si, no habiendo ninguna característica visual para mostrarle al cliente, pero si es viable observar el esfuerzo invertido en dichas instancias y en la documentación generada, que será un activo clave a futuro.

Por otro lado, todo el equipo se encuentra motivado y expectante sobre los cambios debatidos en la última reunión, de modo de poder realizar un buen progreso en el desarrollo. A continuación, se detalla cada uno de los cambios que estaban programados para ser realizados al comienzo de esta semana.

Nueva columna en tablero Kanban

Conforme a lo expresado y decidido por parte del equipo en la reunión semanal de la semana pasada, se realiza la inclusión de una nueva fase al tablero Kanban, denominado: “Ready to Validate”. El fin de esta fase o columna dentro del tablero es organizar aquellos ítems que se encuentran desarrollados en su totalidad y que necesitan pasar por un proceso de verificación para validar si cumplen con todos los objetivos. A su vez, se crea un nuevo estado denominado “To Validate”, que coindice con dicha fase del proceso. A continuación, se observa una imagen que muestra la configuración del tablero Kanban actualizada:

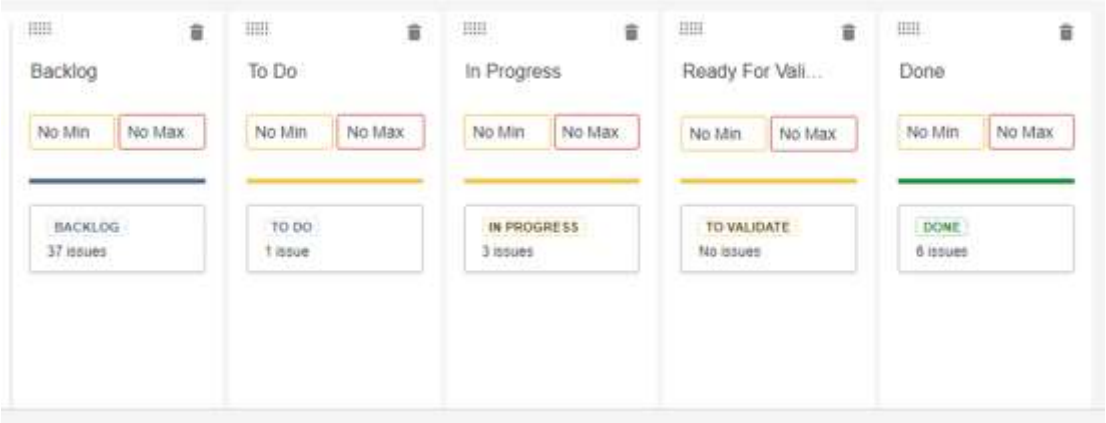


Gráfico 39. **Configuración de nueva fase en tablero Kanban JIRA**

Las validaciones a realizar pueden ser categorizadas en diferentes tipos:

- Revisión de Pares (también llamada en inglés, “Peer Review”): se realiza entre colegas de manera informal y pretende encontrar problemas de forma conjunta.
- Inspecciones Técnicas: en las que se realiza un análisis y verificación minuciosa para encontrar problemas de código. Este tipo de revisión puede llegar a realizarse en módulos o porciones de código totalmente vitales y prioritarias en un sistema.
- Ejecución de casos de prueba: ejecución de casos de prueba para constatar el nivel de calidad del requerimiento o ítem. Dentro de los tipos de ejecuciones de casos de pruebas, se pueden contemplar: a) testing funcional, testing no funcional, testing de regresión, testing complementario, testing sanitario (o en inglés denominado “sanitary testing” o “smooth testing”).
- Validación de defectos: validación de defecto previamente encontrado. También puede conllevar a la ejecución de casos de pruebas que permitan tener la completa cobertura del defecto encontrado, de forma de evitar que el mismo aparezca en el futuro.
- Validaciones de documentos: en los cuales se validan los contenidos en función de documentación previamente creada y dispuesta como base.

Definición de WIP

Luego de utilizar el flujo de trabajo que JIRA integra de forma predeterminada, se identifica esta oportunidad de mejora durante la reunión semanal del equipo. Se debate sobre la posibilidad de aplicar un valor de WIP para cada una de las fases y se definen los siguientes valores:

1. "Backlog": no se determina ningún WIP, ya que aquí se sitúan todos los requerimientos creados por el cliente, que muchas veces pueden no estar ni siquiera definidos.
2. "To Do": se define un WIP con valor 4.
3. "In Progress": se establece un WIP de 2 unidades.
4. "Ready for Validation": se limita el WIP en esta fase a un valor de 2.
5. "Done": no se determina WIP, ya que en esta fase se acumularán todos los ítems ya completados.

A continuación, se muestra una imagen de configuración de dicho WIP por cada una de las fases del tablero Kanban:

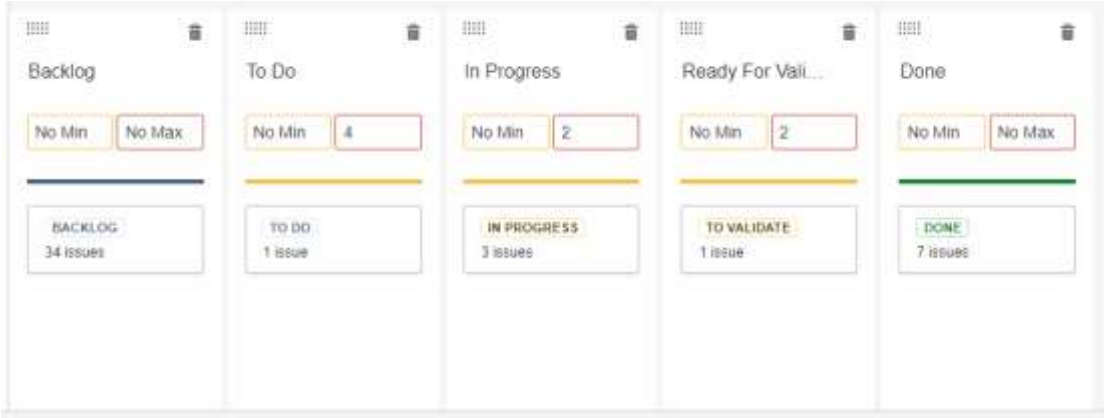


Gráfico 40. **Configuración de WIP en tablero Kanban JIRA**

Se observa en los casilleros rojos, debajo de cada nombre de fase, la definición del tamaño del WIP. Se comprende también, que dicho WIP podrá cambiar con el correr del proyecto, según la necesidad del equipo.

Día 1 - 11 de Julio de 2017

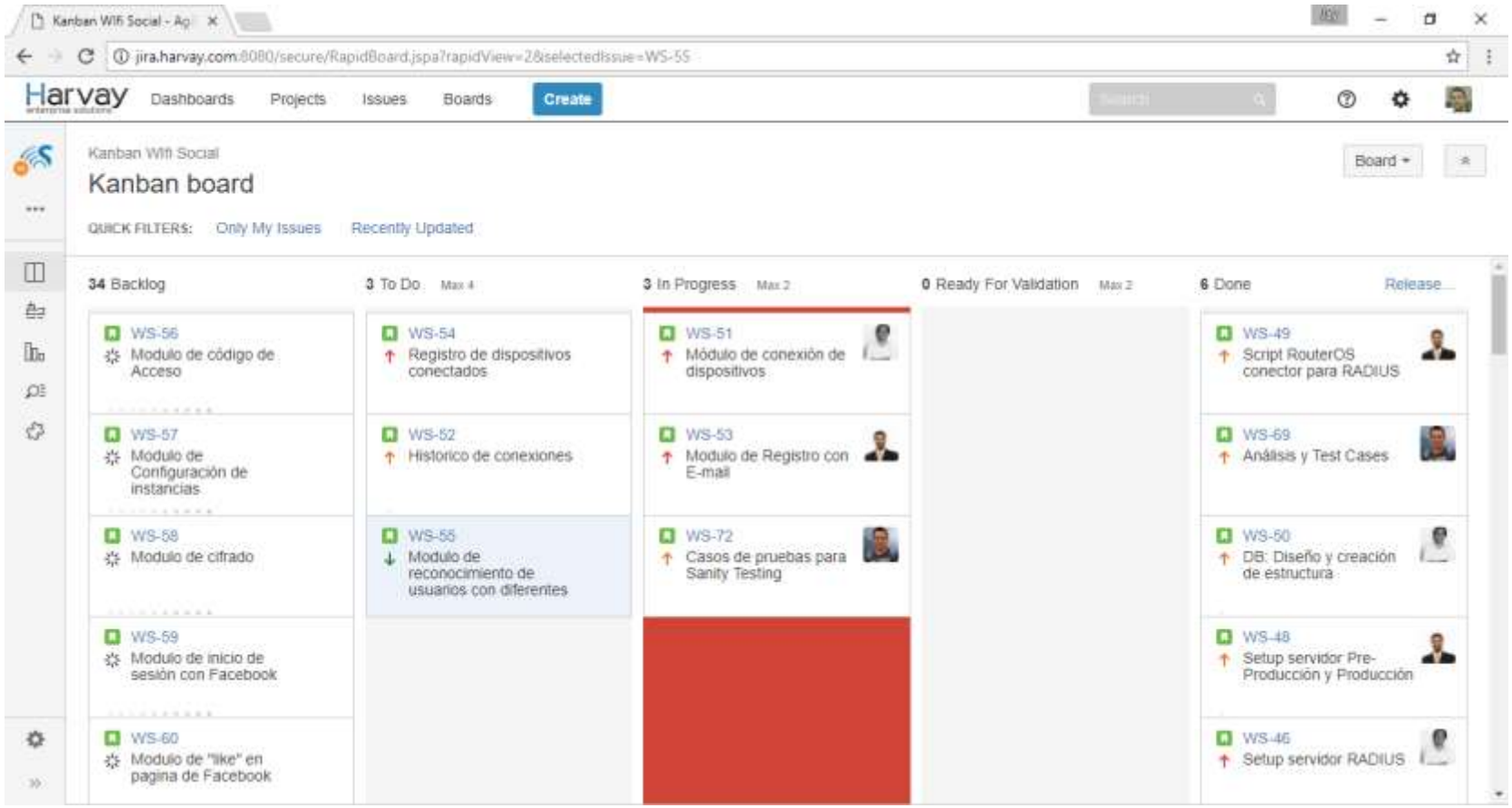


Gráfico 41. **Tablero Kanban Día 1 (semana 2)**

Detalle Jornada: Día 1 - 11 de Julio de 2017

Empieza el primer día de la segunda semana, con dos (2) de las optimizaciones del proceso realizadas y otras dos (2) pendientes de aprobación desde el lado del cliente. Cada uno de los miembros del equipo continua con el trabajo pendiente de la semana pasada. En el caso de Agustín, continua con el WS-51 y pretende completarlo al día siguiente. Por otro lado, Paulo inicia su trabajo en el ítem WS-53, llevándolo desde la columna "To Do" hacia la columna "In Progress" y luego de unas horas de análisis y de dialogar con Agustín, identifican que existen componentes de WS-53 que son dependientes de WS-51. Con lo cual, será necesario completar primero WS-51, para que luego de validarlo, se proceda con el desarrollo de WS-53. La situación es informada al líder del proyecto (Javier) y se decide pausar dicho requerimiento (llevándolo a la columna "To Do") y comenzar a trabajar algún otro que no disponga de ninguna relación con WS-51. Luego de chequear, se determina que el requerimiento WS-55 (de baja prioridad), no posee ninguna dependencia con WS-51, por lo cual se puede iniciar su proceso de desarrollo.

Redefinición inmediata de WIP

Gustavo crea una nueva tarjeta llamada "Casos de pruebas para Sanity Testing" (WS-72) y la envía para la columna "In Progress". Automáticamente JIRA marca dicha columna en color rojo, denotando de que se ha superado el WIP definido para esa fase.

Gustavo entra en contacto con Javier para explicarle la situación y se redefine que el WIP para la columna "In Progress" necesita ser aumentado a 3, ya que son 3 miembros los que pueden disponer de tarjetas en dicha fase.

A su vez, Javier indica que la fila "Ready for Validation" no debería tener un límite WIP, ya que está es una pila en la que se sitúan todos los ítems listos para validarlos. Es decir, dichos ítems se encuentran en espera de que alguien los tome para realizar esta verificación de calidad. Por lo tanto, no es conveniente realizar una limitación de WIP en fases que sirven como pilas.

Finalmente, ambos cambios son realizados de forma inmediata y se comunica a todo el equipo mediante e-mail, reforzando con un chat por Skype.

Luego de esta redefinición, la columna vuelve al color original y Gustavo continua con su trabajo sobre la tarjeta WS-72. Cabe aclarar que, esta card corresponde a la creación de más casos de pruebas que serán utilizados para verificar la calidad del producto, al momento de liberar una nueva versión. Estos son denominados casos de pruebas sanitarios, es decir, casos de pruebas esenciales que validan funcionalidades y características claves en el producto, de modo de identificar rápidamente si algún componente no está funcionando como se espera. Evidentemente, esto ayudará a que rápidamente se entienda si la versión del producto liberada, se encuentra acorde a la expectativa o si, por el contrario, necesita corregir defectos. La expectativa en cuanto a tiempo de ejecución de esta batería de casos de pruebas sanitarios, es de un máximo de 30 minutos por cada versión que se evalué. Futuramente, se pretende automatizar dichos casos de pruebas para que esta verificación pueda ser realizada de forma automática, rutinaria y expeditiva.

Javier dispone todavía de 2 puntos pendientes para tratar con el request manager: a) Especificación detallada de requerimientos y b) despliegue de actividades por card. Es por eso que se comunica con el mismo (Lucas) y se marca un encuentro por Skype para el día 13 de Julio a las 16 horas.

Día 2 - 12 de Julio de 2017

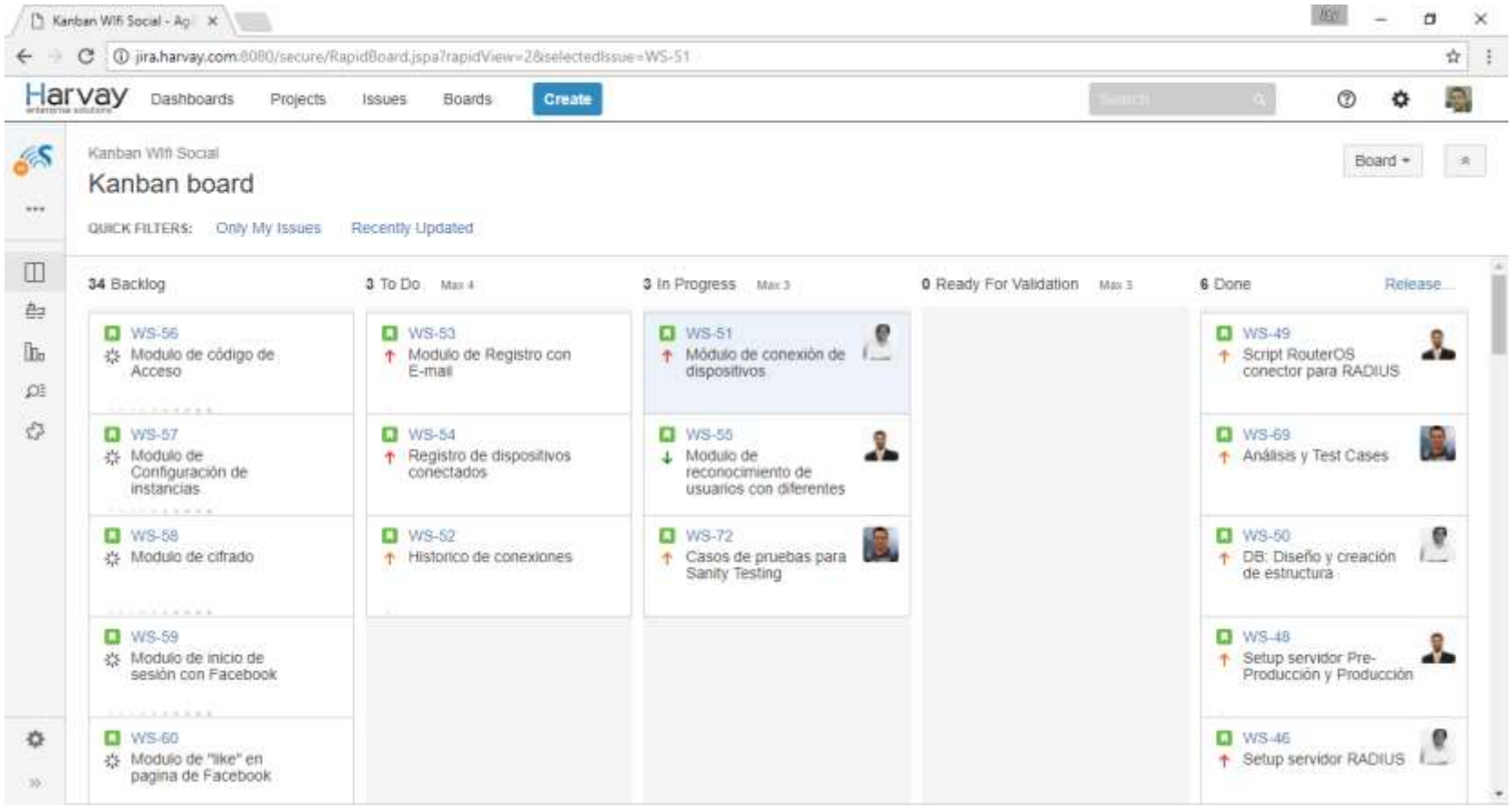


Gráfico 42. **Tablero Kanban Día 2 (semana 2)**

Detalle Jornada: Día 2 - 12 de Julio de 2017

Por un lado, a Agustín le surgen algunas preguntas que deben ser realizadas al request manager, de forma de evitar subjetividades durante el desarrollo del req. WS-51. Es por eso que envía email con todas las preguntas y Javier refuerza dicho email con un contexto propio para el análisis completo de la situación. Lucas responde luego de unas horas y provee toda la información objetiva y requerida para poder continuar. Todavía se disponen de algunas horas necesarias para completar la finalización del requerimiento, pero Agustín estima que al final de la jornada lo tendrá listo para enviarlo a validar.

Por el otro lado, Paulo llega a un punto en el cual, necesita entender algunos puntos técnicos en relación a lo que será necesario más adelante para el reconocimiento de los usuarios (WS-55). Por eso lo contacta a Agustín para analizarlo y evaluarlo juntos. Deciden realizar una reunión por Skype, y luego de debatir durante unos minutos y examinar los tipos de acceso, se establecen guías en las cuales Paulo se basará para construir este módulo. Paulo consigue completar el requerimiento y enviarlo para validación.

Llega el primer ítem a la fila de validación, Gustavo realiza una pausa en la creación de los casos de pruebas sanitarios y decide validar el módulo de reconocimiento de usuarios con diferentes tipos de acceso (WS-55). Durante la revisión, encuentra algunos aspectos que necesitan ser discutidos con Paulo para entender si hay un error en el comportamiento o no. Javier decide participar de dicha reunión, para entender la dinámica utilizada Gustavo para llevar a cabo las pruebas de calidad. Finaliza la misma reunión con una aclaración en el código y una adaptación que deberá ser realizada en el requerimiento, para que todo tenga una congruencia y sentido. Javier envía un email a Lucas para comunicar el cambio sobre este requisito y queda expectante por cualquier duda o pregunta sobre el mismo. Se realizan las modificaciones en el código y en el requerimiento, y se da por completada la actividad.

Día 3 - 13 de Julio de 2017

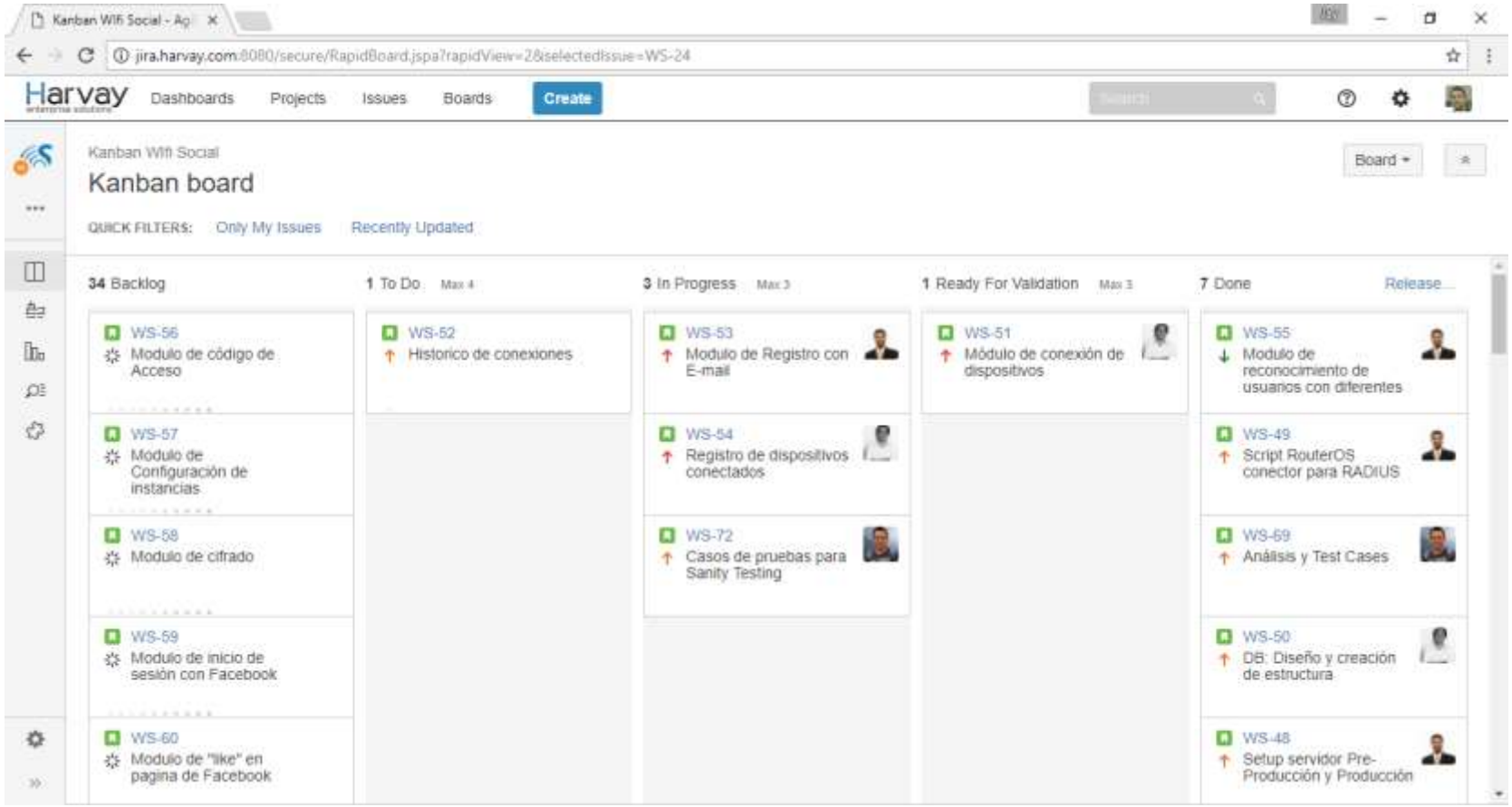


Gráfico 43. **Tablero Kanban Día 3 (semana 2)**

Detalle Jornada: Día 3 - 13 de Julio de 2017

La tercera jornada de trabajo acontece sin grandes preocupaciones y con grandes avances de los módulos más importantes. Ambos desarrolladores se auto-asignan nuevos requerimientos (WS-53 y WS-54), pero luego de analizar el contenido de los 2 ítems en JIRA, identifican que falta información clave para proceder con la creación de ambos requisitos. Cabe destacar que este problema ya había sido reportado al líder del equipo durante la reunión de la semana pasada, sin embargo, la reunión con el request manager para tratar estos temas, está citada para la jornada de hoy a las 16 horas.

En este sentido, una vez que Javier es informado sobre este inconveniente nuevamente, él sugiere enviar un email a Lucas (con copia a todo el equipo), realizando las preguntas pertinentes a cada ítem de JIRA, de modo que, o el mismo Lucas responda antes de las 16 horas o bien, que pueda responder durante la reunión de forma oral. En este último caso, Javier separará la reunión en dos partes: a) reunión para dialogar todo lo referente a gerenciamiento del proyecto, y b) reunión para dialogar todos los puntos técnica.

Desde un aspecto de calidad del sistema, Gustavo visualiza el ítem WS-51 ya disponible para realizar su validación, pero determina que, de modo de no interrumpir nuevamente el desarrollo de los casos de prueba sanitarios, decide continuarlos hasta su fin, para lugar tomar esta validación en particular.

Reunión con Request Manager

Durante la misma jornada a las 16 horas, Javier inicia una conversación con el request manager (Lucas) mediante Skype y detalla los 2 puntos pendientes para proceder con la actualización en la herramienta JIRA. A su vez, propone las ideas debatidas con el equipo de trabajo. A continuación, se detalla cada uno de los puntos discutidos en la reunión, que sin duda son parte de una optimización del proceso de trabajo completo:

Despliegue de actividades por tarjeta

Durante la reunión se explica al request manager los beneficios de desglosar cada requerimiento en actividades unitarias. Es decir, a raíz de que un requerimiento puede demandar esfuerzo de más de un miembro del equipo, se expone la necesidad de individualizar cada uno de esos esfuerzos en actividades visibles, en el de cada uno de los requerimientos se desglosen en actividades unitarias.

Para ello, se crearán actividades denominadas “Sub-Task” dentro de cada requerimiento, de forma de poder desglosar cada card en diferentes tareas (llamadas sub-tareas en JIRA), en función del esfuerzo necesario para su construcción y desarrollo de cada requerimiento en sí.

Por su parte, Lucas valida si podría existir algún tipo de confusión por parte del cliente en el tablero, a lo que Javier le comenta que se creará un filtro en el cual mostrará solo la vista del cliente, para que puedan disponer de la misma vista que tienen actualmente. Lucas acepta la idea y aprueba el cambio.

Especificación detallada de requerimientos

Javier trae como ejemplo a la mesa de discusión, el último email enviado por el equipo en relación a algunas preguntas que son necesarias para entender con exactitud, el alcance y especificación del requerimiento.

A su vez, se le comenta a Lucas, que han encontrado requerimientos en la columna de “To Do”, en la cual no se informa con detalle lo que necesita ser realizado. Y que, como consecuencia, esto acarrea demoras en el desarrollo y posibles subjetividades intrínsecas por parte del equipo.

Lucas, por su parte, comenta que dados los tiempos en los cuales fue iniciado el proyecto, no se dispuso de tiempo suficiente para especificar algunos detalles que ahora, entiende que son fundamentales para el desarrollo, pero que se están completando poco a poco en JIRA. Javier ofrece un punto importante a tener en cuenta que es la modificación del formulario en JIRA, que permita pedir algunos datos de

forma obligatoria, a fin de evitar faltantes de información. Lucas acepta realizar esta prueba, pero afirma que esto depende de muchos factores, ya que a veces, se agregan ítems a JIRA sin disponer de todas las informaciones y luego se van pidiendo en función de la prioridad de cada uno. Esto es entendible y es allí donde Javier ofrece la opción de agregar estos cambios obligatorios solo al momento de transicionar el requisito desde “Backlog” hacia “To Do”. Esto es aceptado por Lucas.

Luego de acordar este punto y, de forma de acelerar con las respuestas a las dudas y preguntas por parte del equipo sobre los ítems WS-53 y WS-54, Javier le pide al equipo entrar en la reunión, para debatir y realizar las preguntas de forma oral. El equipo se conecta y realiza todas las preguntas necesarias para continuar con el trabajo. Lucas por su parte, responde a cada una de las preguntas con exactitud.

Luego de 2 horas de reunión, el equipo completa la llamada de Skype con grandes pasos ya acordados y con mucho por resolver.

Día 4 – 14 de Julio de 2017

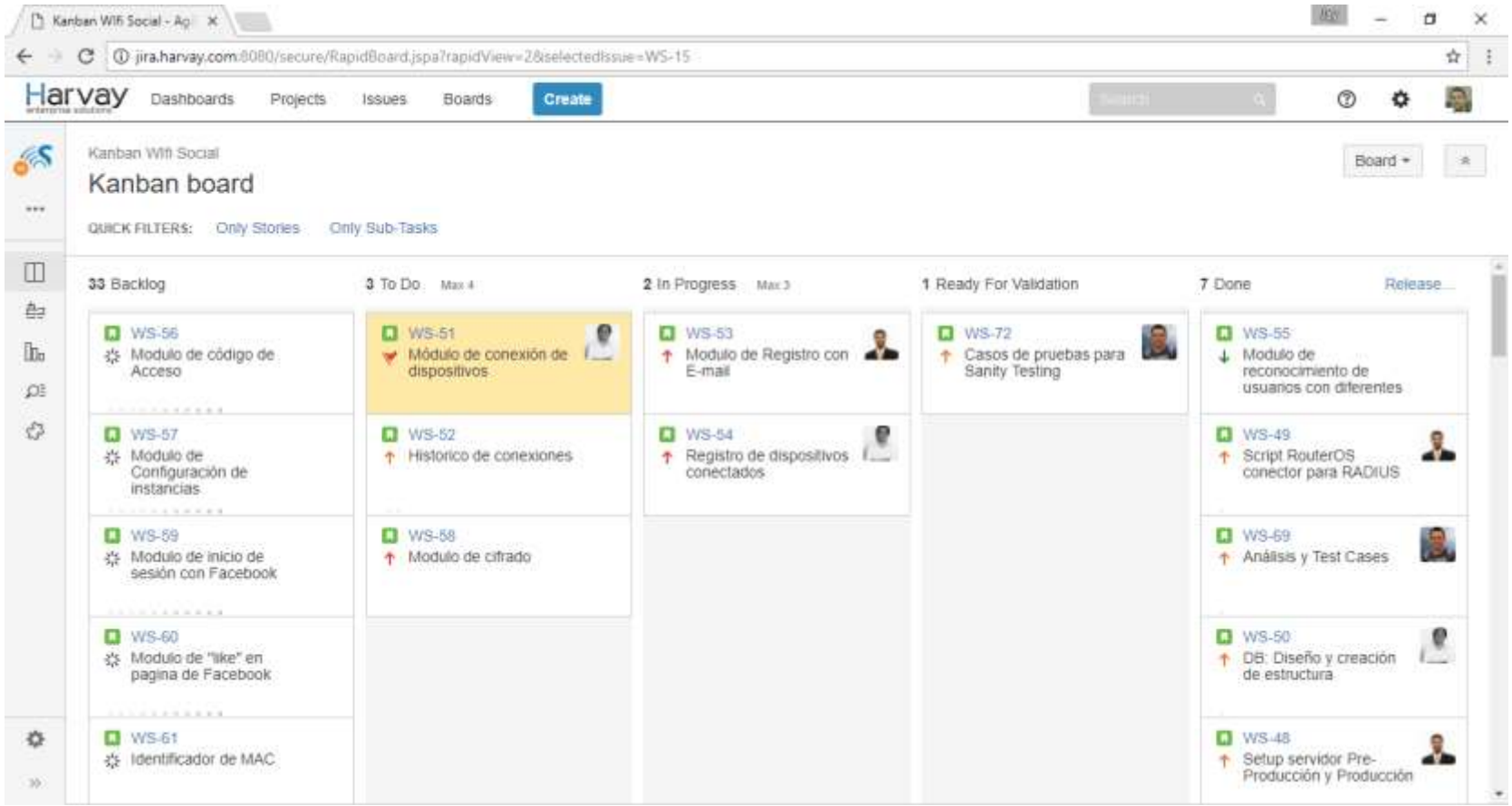


Gráfico 44. **Tablero Kanban Día 4 (semana 2)**

Detalle Jornada: Día 4 – 14 de Julio de 2017

Javier realiza la implementación de una de las mejoras definida y aprobadas por parte del request manager, denominada “Especificación detallada de requerimientos”, y se entiende que, desde ahora en adelante, todo aquel requerimiento que sea transicionado para la columna de “To Do”, deberá disponer todos los campos obligatoriamente requeridos por JIRA para poder continuar.

Luego de obtener las clarificaciones por parte del request manager, ambos desarrolladores continúan con su trabajo (WS-53 y WS-54) y estiman completarlo al día siguiente.

En lo que respecta al área de calidad, Gustavo completa los casos de pruebas y a su vez, actualiza el plan de testing con esta nueva información (WS-72). Transiciona el ítem para la fase de “Ready for Validation”, para que Javier pueda realizar una revisión general sobre la cobertura de calidad hacia el producto.

Luego de enviar el ítem WS-72 a fila de validación, comienza a verificar el ítem WS-51, que es fundamental para el producto y también para sus dependencias. En este punto, cabe destacar que muchos de los casos de pruebas establecidos para el testing sanitario se encuentran exclusivamente definidos para este requerimiento, que resulta ser uno de los núcleos fundamentales.

Por primera vez, se realiza la ejecución de una batería de casos de pruebas sanitarios sobre uno de los módulos de núcleo (WS-51), ya desarrollados y listo para validar. Dicha ejecución se realiza de forma exitosa. Eso no quiere decir que la misma se encuentre libre de bugs, pero si quiere decir, que los aspectos críticos del sistema, funcionan de manera correcta.

Posterior a esta validación, se realizan otras pruebas, aplicando casos funcionales y no funcionales, donde se encuentra un defecto funcional que no corresponde con el comportamiento esperado del módulo. Es por eso que la tarjeta se transiciona para la columna de “To Do” con una bandera roja, especificando que ese ítem requiere atención por parte del equipo.

Día 5 - 17 de Julio de 2017

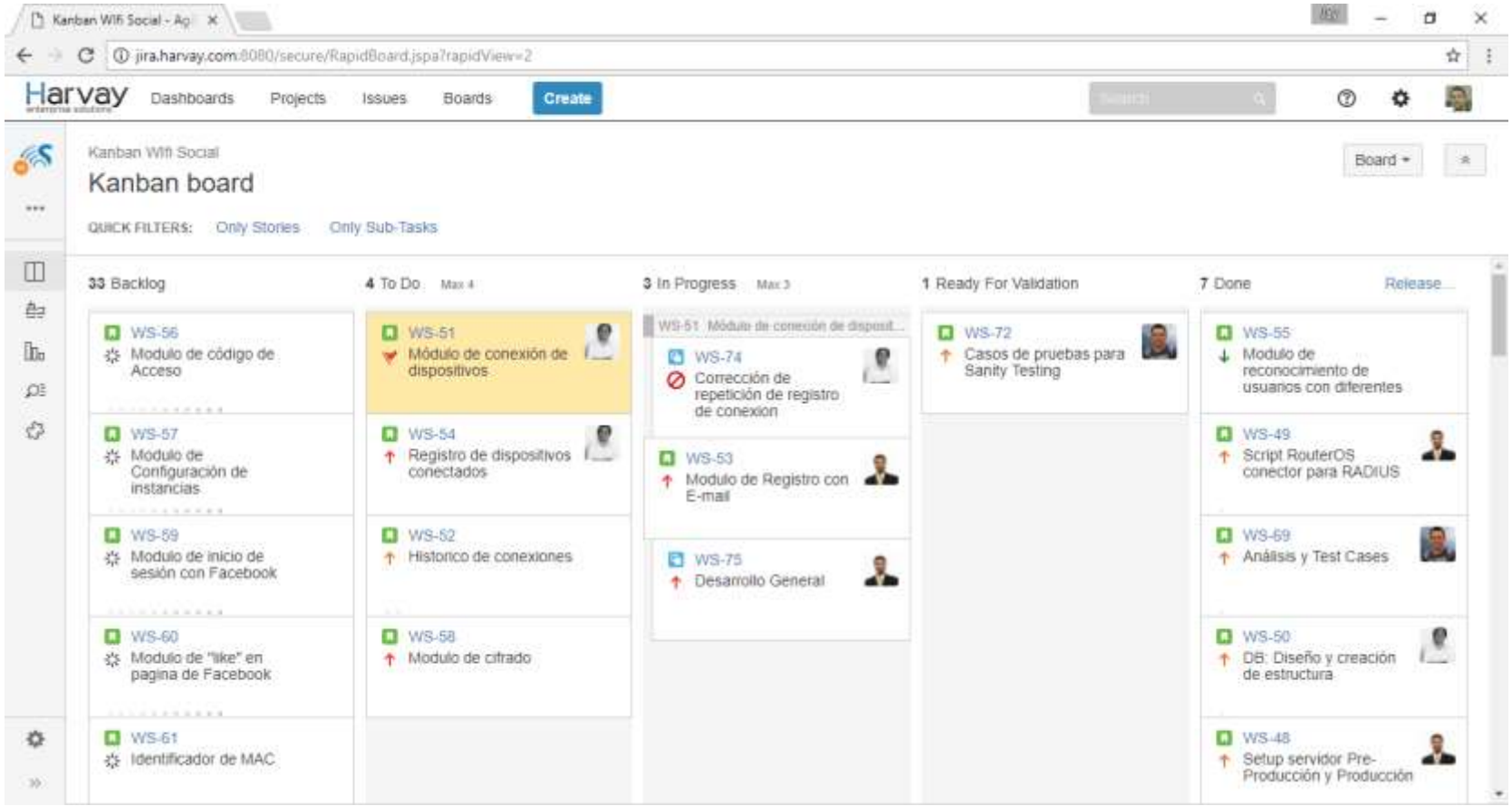


Gráfico 45. **Tablero Kanban Día 5 (semana 2)**

Detalle Jornada: Día 5 - 17 de Julio de 2017

Javier realiza la implementación la segunda mejora definida y aprobada por el request manager, denominada "Despliegue de actividades por tarjeta" y se visualiza en la última jornada de la segunda semana. Asimismo, se crearon dos (2) filtros: a) “Only Stories”, que significa en inglés “solo historias”. Este se refiere a la visualización solo de los ítems que son requerimientos por parte del request manager. b) “Only Sub-Tasks”, que significa en inglés “solo subtareas”. Este segundo, se traduce como el filtro para visualizar solo el esfuerzo del equipo en cada requerimiento. Ambos filtros se pueden activar con un clic en la sección “Quick Filters”, en la parte superior izquierda, debajo del título “Kanban board”.

El equipo ya comienza a crear sub-tareas dentro de cada requerimiento, lo cual lleva a entender con exactitud el esfuerzo de cada uno. Posteriormente, la idea es estimar dichas tareas de forma de obtener el esfuerzo total por cada requerimiento.

Se crea una subtarea para la corrección del defecto encontrado por Gustavo en la card WS-51. Dicha subtarea tiene la identificación WS-74 y es trabajada con urgencia por Agustín. A su vez, como se observa en el Kanban, el requerimiento WS-54 que estaba comenzando a ser desarrollado por Agustín, se pausa y mueve a la columna de “To Do”, des-priorizando el mismo para atender el defecto en cuestión. Este punto muestra como correctamente se utiliza el límite WIP definido. Tiene sentido denotar que, la corrección está estimada para ser completada en 2 horas, y es por eso que, en el trascurso de la jornada se completa y se entrega, a fin de realizar una nueva ejecución del ciclo de casos de pruebas.

Por otra parte, Paulo crea una subtarea (WS-75), para reflejar su esfuerzo de desarrollo para el requerimiento WS-53, que pretende tenerlo listo en 2 días.

Reunión semanal (segunda semana)

Durante la misma quinta jornada laboral, Javier convoca a todo el equipo para realizar una reunión semanal sobre lo sucedido durante el periodo, de forma de analizar, corregir y evaluar todo el esfuerzo invertido.

A continuación, se exhiben los puntos concretos que fueron tratados:

Pedido de mayor detalle en requerimientos

Agustín afirma que luego de verificar algunas tarjetas en la columna de "To Do", él observa un cambio positivo en relación a días anteriores, en los que algunos requerimientos disponían de solo 3 líneas de explicación y un diseño adjunto. Ahora, disponen de mayor detalle y se entiende mejor a dónde quiere llegar dicho requerimiento.

Sin embargo, entiende que puede mejorar aún más al utilizar algunas técnicas que utilizan algunos Product Owners (en español, dueño del producto), en el enfoque Scrum. Él propone utilizar el formato de historias de usuario (en inglés, "User Story") del enfoque Scrum.

El resto del equipo está de acuerdo con la propuesta de Agustín y Javier entiende que es el paso correcto para crear una dinámica de trabajo consistente a lo largo del tiempo. Pretende dialogar con Lucas para ver la posibilidad de incluir este tipo de técnicas de redacción de requerimientos. Javier lleva la acción de negociar este punto.

Despliegue de actividades por tarjeta

Sin duda, esto ha traído mayor transparencia y ha ayudado a la comunicación entre los miembros del equipo. Javier resalta los beneficiosos de disponer de este desglose de tareas y comunica que pretende implementar un sistema de estimaciones a futuro, donde sea más simple entender cuál es la velocidad de desarrollo del equipo. No hay acción sobre este punto.

Reunión diaria de sincronización

Javier propone una reunión diaria de 15 minutos en la cual se verifique el tablero Kanban y se respondan las preguntas de la reunión “Stand-up” del enfoque Scrum. El equipo cree que eso ayudará a generar un alto grado de sincronismo entre ellos. Todos están de acuerdo. Se establece que esta reunión no necesitará minuta por e-mail, ya que es una reunión a modo de sincronización. Javier lleva la acción de enviar un evento para el calendario de cada miembro, de forma de dejar la cita marcada para cada día.

Definición de WIP

Todo el equipo está de acuerdo que la limitación del WIP provee mayor claridad al tablero y esto ayuda a organizarse en todo momento. No obstante, Gustavo trae a colación un punto muy importante que es el hecho de validar y redefinir el límite de WIP una vez más, a consecuencia del despliegue de actividades de cada tarjeta. Es decir, al realizar este desglose en subtareas, cada requerimiento dispondrá de actividades que también se visualizarán en el tablero o board, con lo cual, esto causará que el límite de WIP sea excedido en algunas fases determinadas. Gustavo indica que en el caso de la columna “In Progress” con un WIP actual de 3, si los dos (2) desarrolladores y el mismo Gustavo se encuentran realizando alguna actividad en particular, cada uno con sus sub-tareas para cada tarjeta, esto generará un total de 6 ítems en esa fase.

Como consecuencia, el límite WIP se verá afectado y JIRA marcara dicha columna en rojo (como anteriormente se pudo visualizar). Es por eso que Gustavo propone cambiar el WIP a por lo menos un valor de 7 u 8. Javier acepta su propuesta y ejemplifica otro caso en el cual no solo los 2 desarrolladores y el analista de testing podrían encontrarse trabajando en el tablero, sino también el mismo líder o el request manager. Por lo cual, el WIP debe aumentar.

Luego de debatir en lo que respecta a cada una de las fases del proceso en el tablero, se llega a la conclusión de que solo es necesario modificar la fase de “In Progress”, estableciendo valor de 8. Javier lleva la acción de actualizar este parámetro.

Mayor detalle del proceso en Kanban

Este es uno de los puntos que Javier argumenta con el objetivo de detallar aún más, el flujo de trabajo dentro del tablero. Esto traerá incluso más transparencia para el cliente y a su vez, se podrá identificar con mayor exactitud a donde se produce un cuello de botella en particular (en inglés, comúnmente denominado “roadblock”). Se debaten entre todos los miembros, las fases que se desearían incluir y se definen las siguientes: a) “Backlog” (existente), b) “To Do” (existente), c) “In Progress” (existente), d) “Ready for Validation” (existente), e) “In Pre-Production” (nueva), f) “Done” (existente).

Como puede observarse, la nueva fase a ser incorporada es “In Pre-Production” que tiene como objetivo agrupar aquellos ítems que son liberados en el ambiente de Pre-Producción, para así, el request manager pueda realizar sus pruebas también. Javier lleva la acción de implementar esta fase en el Kanban con su correspondiente estado y transición de fase.

Demostración del entregable

En la misma minuta que es enviada a todo el equipo, incluyendo el request manager, se especifica que la demostración del producto bajo desarrollo, puede ser comprobada en el ambiente de Pre-Producción. No obstante, existe un defecto conocido (WS-74), que, si bien ya ha sido corregido, su corrección no ha sido todavía implementada en el ambiente de Pre-Producción y por eso, ser obviado durante la demostración. Javier lleva la acción de comunicar al nivel gerencial sobre este punto.

Por último, todo lo expuesto arriba es enviado en formato minuta por e-mail:

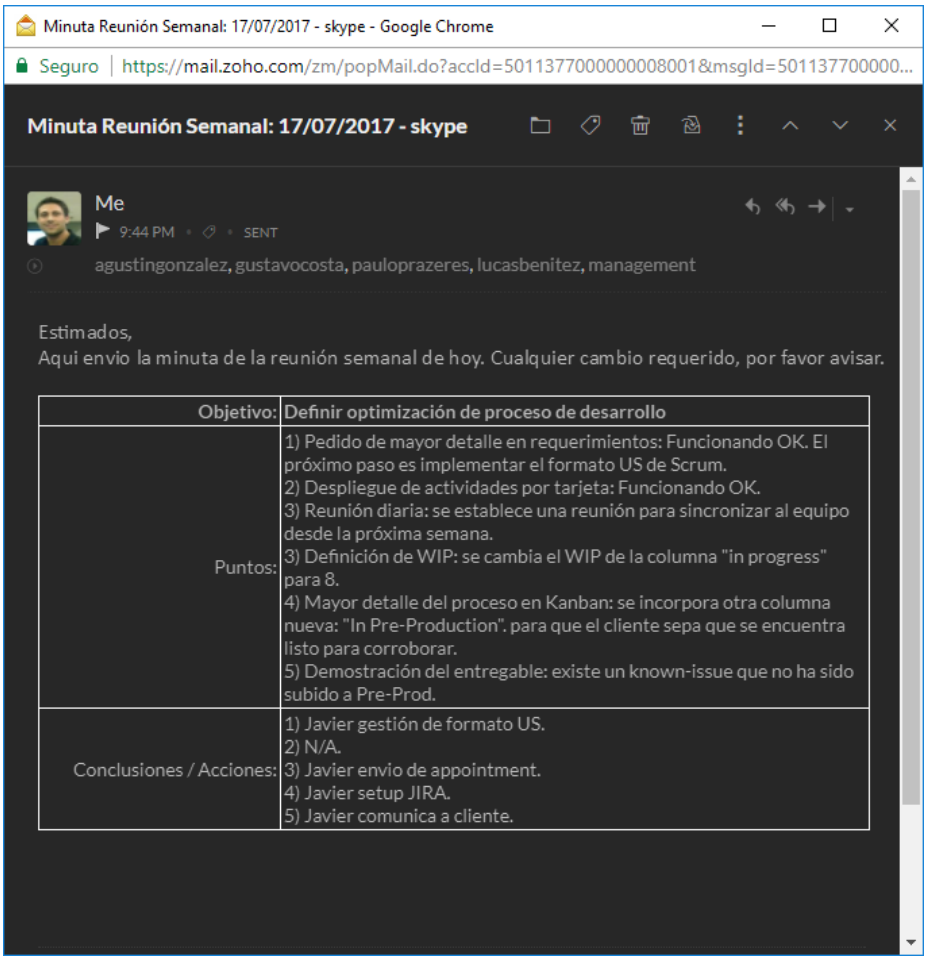


Gráfico 46. **Minuta de reunión semanal**

Análisis de indicadores Kanban

Existen diferentes indicadores que pueden ayudar a interpretar el accionar del equipo dentro de un marco de tiempo específico del proyecto, o, porque no, dentro del marco total del proyecto. JIRA dispone de alrededor de 20 indicadores gráficos provistos por la herramienta de forma predeterminada, que permiten realizar un análisis detallado. Es posible obtener diversas informaciones en sus diferentes grados organizacionales. Ergo, es posible proveer informaciones a los tres niveles más importantes en una organización: a) direccionales, b) gerenciales y c) operativos.

El alcance del módulo de indicadores de JIRA es amplio y va desde obtener datos como “la cantidad de productos trabajados por el equipo”, hasta datos como “la cantidad de bugs corregidos por cada desarrollador”, pasando por datos como “cantidad de fallas relacionadas a cada área del sistema” o “comparación bugs reportados vs bugs resueltos”.

El objetivo de este apartado es exponer el gráfico que nos ayuda en mayor medida a entender lo sucedido en estas dos semanas de trabajo. A continuación, se observa el Diagrama de Flujo Acumulativo del proyecto:

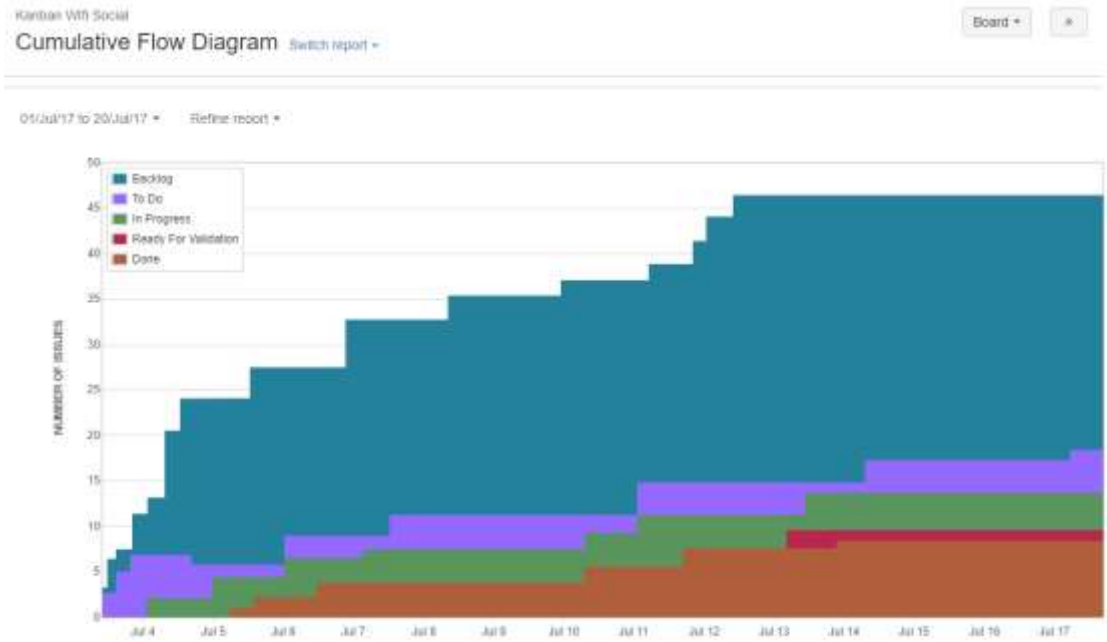


Gráfico 47. **Diagrama de Flujo Acumulativo**

En él, se observan todas las fases de desarrollo presentes en el tablero Kanban que muestran su evolución en función de los días trabajados.

De este gráfico pueden obtenerse información relevante al accionar del equipo en estas 2 semanas. Por un lado, el eje “y” muestra la cantidad de ítems en JIRA (es decir, requerimientos) son visualizados. Por otro, el eje “x” muestra las fechas desde el inicio hasta la finalización de la segunda semana.

A continuación, y a los fines de proveer una explicación más concreta y específica, se muestra el mismo gráfico con algunas indicaciones incorporadas dentro del mismo:

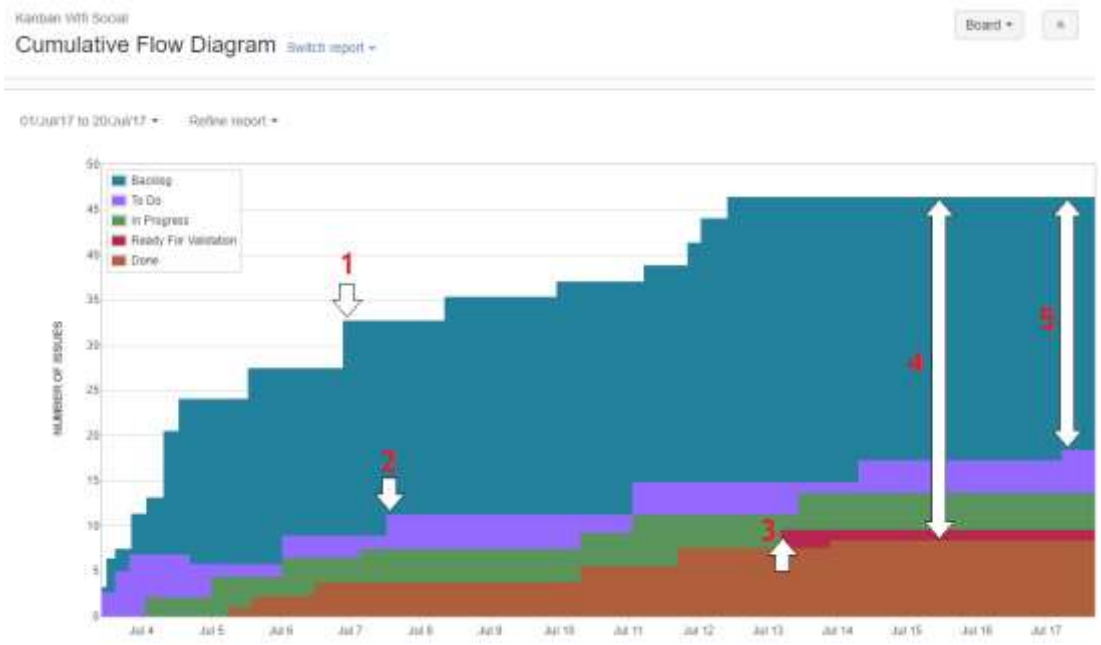


Gráfico 48. **Diagrama de Flujo Acumulativo explicado**

Siendo que cada punto simboliza:

1. Día 7 de Julio 2017: la cantidad de ítems que fueron creados y que se encuentran en el Backlog. Su valor es 33 ítems.
2. Día 7 de Julio 2017: la cantidad de ítems que fueron priorizados y enviados a la columna de “To Do” dentro del tablero ese día. Su valor es 11 ítems.
3. Día 13 de Julio 2017: la cantidad de ítems que fueron enviados a la columna “Ready to Validate” ese día. Cabe destacar que ese mismo día, se enviaron dos ítems para esta columna del tablero y más tarde (ese mismo día), se retiró uno de esos ítems, dejando solo uno en la fila de validación. Su valor es 9 ítems.
4. Día 15 de Julio 2017: la cantidad de ítems que todavía están pendientes de ser desarrollados y completados. Su valor es 38 ítems.

5. Día 17 de Julio 2017: la cantidad de ítems en el backlog, o lo que se define mejor como el tamaño del backlog. Su valor es 28 ítems.

Lead Time y Cycle Time

A continuación, se muestra el mismo gráfico con los indicadores referenciando al Lead y Cycle Time:

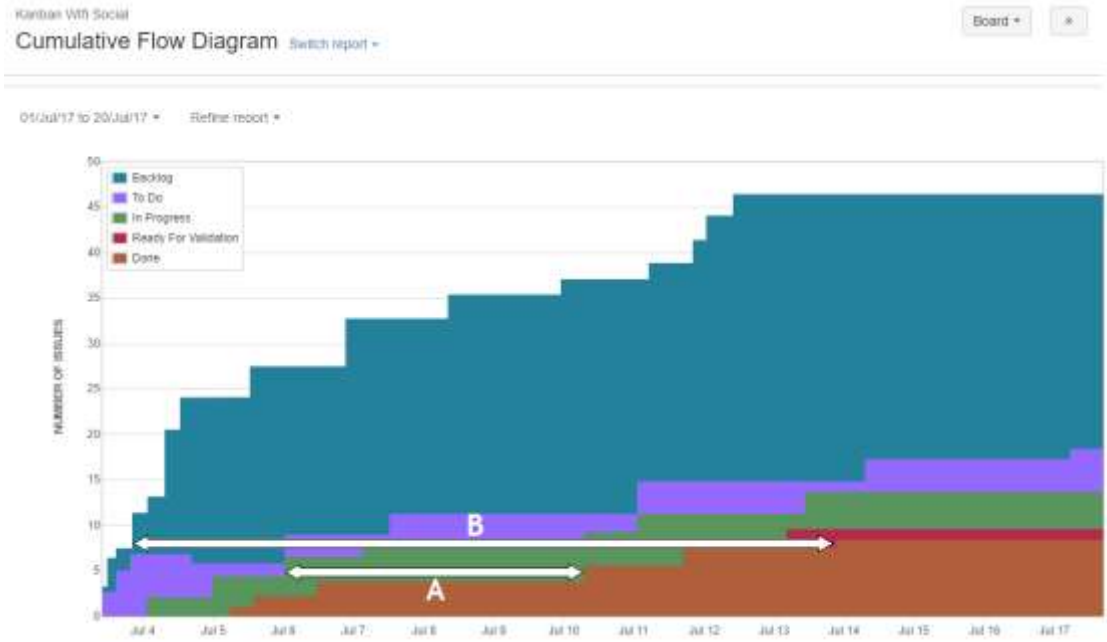


Gráfico 49. **Identificación del Lead Time y Cycle Time**

Siendo que, dichos indicadores se referencian como:

A. Cycle Time o el tiempo de ciclo, que comprende desde que el ítem comienza a ser desarrollado (“In Progress”), hasta que es finalizado (“Done”). Su valor es: 4 días.
B. Lead Time, o tiempo de entrega, comprende desde que el ítem ingresa al tablero Kanban (“Backlog”), es decir, hasta que sale del tablero Kanban (es decir, llega a “Done”). Su valor es 9 días.

CAPÍTULO VIII

IMPLEMENTACIÓN DE ENFOQUE SCRUMBAN

Fueron completadas las dos (2) primeras semanas de desarrollo dentro del proyecto, con importantes avances dentro del mismo. El equipo ya entiende la dinámica de trabajo y ha logrado una interesante cohesión grupal entre sus miembros, lo que contribuye a avanzar con los requerimientos dentro de una atmosfera adecuada, con una importante base en lo que refiere al soporte grupal. Asimismo, los estándares de calidad utilizados durante el desarrollo son acordes a las expectativas del cliente y el equipo entiende que este es uno de los factores de éxito que definen un proyecto de TI. Por lo cual, se concibe una estrategia implícita e interna de mejora continua entre los miembros del equipo, en lo que respecta a la calidad del código desarrollado.

A raíz de la última reunión realizada por el equipo, al final de la segunda semana, se definieron puntos en los que se puede avanzar para mejorar la velocidad del equipo y su comunicación, tanto interna como externa. El líder del equipo considera que una migración desde un enfoque ágil Kanban hacia un enfoque ágil Scrumban, sin duda podrá ayudar a la organización del trabajo y comunicación de todos los participantes del proyecto desde diferentes perspectivas. Es por ello que, luego de realizar una reunión con el request manager para debatir sobre los puntos anteriormente discutidos en la reunión interna, se presenta un acuerdo para emprender la incorporación de forma gradual, de diferentes adecuaciones y ajustes compatibles con la naturaleza del proyecto, provenientes de este enfoque Scrumban señalado.

Como consecuencia y a seguir, se realiza una redefinición de la metodología de trabajo del equipo, migrando a un marco ágil con orientación hacia Scrumban.

Redefinición de la metodología de trabajo

De acuerdo a la reunión entre el request manager y el líder del equipo, fueron definidos los siguientes cuatro (4) puntos, que se integran al enfoque actual de trabajo

(Kanban) y que transforman dicha orientación en una orientación Scrumban propiamente dicha.

Cabe aclarar también, que estos puntos poseen diferentes características que deben ser aplicadas en diferentes momentos del desarrollo del proyecto. Es por ello que, se pretende realizar una incorporación de cada una de las características de forma gradual a lo largo de cada una de las semanas de proyecto.

A continuación, se listan y se procede con una breve descripción de cada uno:

1. Roles scrum: si bien el equipo ya posee miembros con diferentes perfiles profesionales que llevan a cabo un rol técnico en particular durante el desarrollo, el enfoque Scrum integra roles específicos que van más allá de lo técnico. Dichos roles, se concentran en la metodología de trabajo del equipo para contribuir al marco de trabajo (o framework) en general. Lo acordado con el request manager es la inclusión de todos los roles Scrum en la metodología de trabajo diaria.
2. Historias de usuario: conforme fue definido teóricamente y también debatido por el equipo, la necesidad de integrar historias de usuarios como requerimientos, es una contribución importante a la definición de los mismo, para marcar el comienzo de una estandarización en lo que respecta a lo que se desea realizar. Lo acordado con el request manager es la inclusión de historias de usuario para nuevos requerimientos que sean incluidos en la herramienta JIRA.
3. Reuniones scrum: este punto es uno de los mayores colaboradores para optimizar la comunicación en el equipo de trabajo y también, con el cliente. El simple hecho de marcar reuniones con períodos fijos y constantes, genera un compromiso por parte de quien realiza el trabajo y, por otro lado, una expectativa por parte del cliente de ver ese trabajo terminado. A su vez, al ser reuniones consistentes en el tiempo, esto permite realizar una comparación entre cada período, denotando el esfuerzo invertido a lo largo del proyecto. Lo acordado con el request manager es la inclusión gradual de las reuniones utilizadas en Scrum.
4. Sprints (iteraciones): la implementación de iteraciones como períodos consolidados de desarrollo dentro del proyecto, sin duda puede ayudar a concentrar

el esfuerzo de cada requerimiento que es desarrollado. Previo al consenso del equipo de trabajo, se acuerda con el request manager la implementación de iteraciones (en Scrum, llamadas Sprints).

Sprint 1 (tercera semana de proyecto)

Se inicia la tercera semana de proyecto con un cambio de enfoque de trabajo para potenciar aún más, la productividad y eficiencia de los miembros del equipo. A continuación, se destacan en detalle, cada una de las cuatro (4) implementaciones anteriormente citadas:

Implementación de roles scrum

Product Owner

Hasta el momento, el rol similar al Product Owner lo llevaba a cabo Lucas, siendo el Request Manager del proyecto. En este sentido y luego de definir con él mismo, resulta conveniente que Lucas lleve a cabo un rol de PO, para integrar algunas responsabilidades extras que provienen del Product Owner de Scrum, en su día a día.

Scrum Master

Actualmente llevado a cabo por Javier como líder de proyecto, pero desde ahora definido como Scrum Master para integrar este nuevo enfoque de trabajo. Tal como sucede con Lucas, Javier también tiene algunas responsabilidades extras, pero de cierta, forma él ya las realizaba de forma implícita a lo largo de estas semanas de desarrollo. Por lo tanto, no tendrá cambios abruptos en lo que respecta a sus responsabilidad y actividades de su día a día.

Equipo Scrum

El equipo de trabajo continúa llevando a cabo los mismos roles técnicos que utilizaban anteriormente en Kanban. No existe un rol específico en Scrum que necesite ser desplegado o diferenciado dentro de su equipo de trabajo. Por lo cual no se ven cambios de responsabilidades ni de actividades en el mismo.

Implementación de historias de usuario

Como fue debatido dentro de una de las reuniones del equipo, y aceptado por el Request Manager (desde ahora llamado Product Owner), fue aprobada la utilización de historias de usuarios para nuevos requerimientos dentro del Backlog. También, se le pidió al PO actualizar aquellos requerimientos que ya se encuentran priorizados (listos para ser trabajados), de forma de revisar nuevamente dichas informaciones, bajo el nuevo marco de historias de usuario. Si bien la labor del PO en cuanto a proveer mayor información para cada requerimiento, ya estaba siendo realizada (como parte de su compromiso anterior), aquí se incorpora un nuevo modelo que contribuirá a la comunicación entre las partes, para saber que es necesario desarrollar a cada momento, eliminando aún más, las ambigüedades o subjetividades durante su análisis.

En este sentido, es notable aclarar que la incorporación de este formato de “User Stories” en cada requerimiento, será una base para la futura implementación de las estimaciones en cada tarjeta, y que, sin duda, éste será el factor fundamental para conocer el valor de velocidad de desarrollo del equipo.

Implementación de reuniones scrum

Como se expuso en los puntos generales de la redefinición de la metodología de trabajo, se pretende que dichas reuniones scrum sean integradas al proceso de manera gradual y consistente, en función de los tiempos y en los momentos correctos que el proyecto lo permita. Existe una necesidad precisa de implementar dichas

reuniones en el contexto correcto, a fin de aprovechar y enfatizar la importancia de cada una de las reuniones.

Implementación de sprints (iteraciones)

Resulta conveniente esclarecer que, en base a lo expuesto en el punto anterior, la identificación de la velocidad de desarrollo, conjuntamente con la implementación de los sprints, ayudará a mejorar el proceso de estimación y desarrollo en general. Y, por consiguiente, esto contribuirá de forma macro para estimar la cantidad de sprints necesarios para completar el proyecto. En otras palabras, esto quiere decir que, conociendo la velocidad promedio de trabajo del equipo y sabiendo también, la cantidad de requerimientos pendientes a desarrollar, se podrá estimar la cantidad de sprints necesarios para completar el proyecto en su totalidad.

El scrum master ya conoce la dinámica del equipo de trabajo y es por eso que define la periodicidad de los sprints de 1 semana de duración. Este valor es definido de esta forma porque, basándose en las 2 semanas de trabajo anteriores, se considera que la mayoría de los requerimientos en general pueden ser construidos e implementados en el lapso de 1 semana. Sin embargo, se prevé una reunión stand up con el equipo en la cual se dialogará sobre este valor en concreto.

A continuación, se visualiza la configuración del primer sprint, conforme a las fechas que se venían utilizando. En la imagen se observa en inglés, la aclaración de la herramienta, que informa que el sprint tendrá 5 días hábiles para trabajar, comenzando en la fecha 18 de Julio de 2017 y terminando el 25 de Junio del 2017. Es decir, con su último día hábil laboral el día 24 de Julio del 2017:

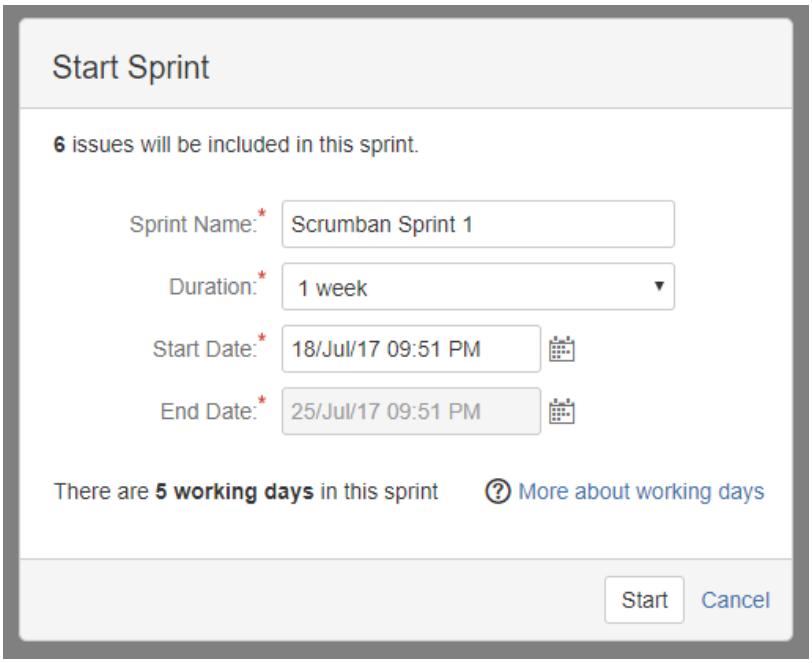


Gráfico 50. **Creación de sprint en JIRA**

Configuración inicial de tablero scrumboard

Fases del proceso de desarrollo

El scrum master realiza la configuración del tablero Scrumban, incorporando algunas características propias del tablero de Kanban y también, modificaciones de acuerdo al nuevo enfoque adoptado.

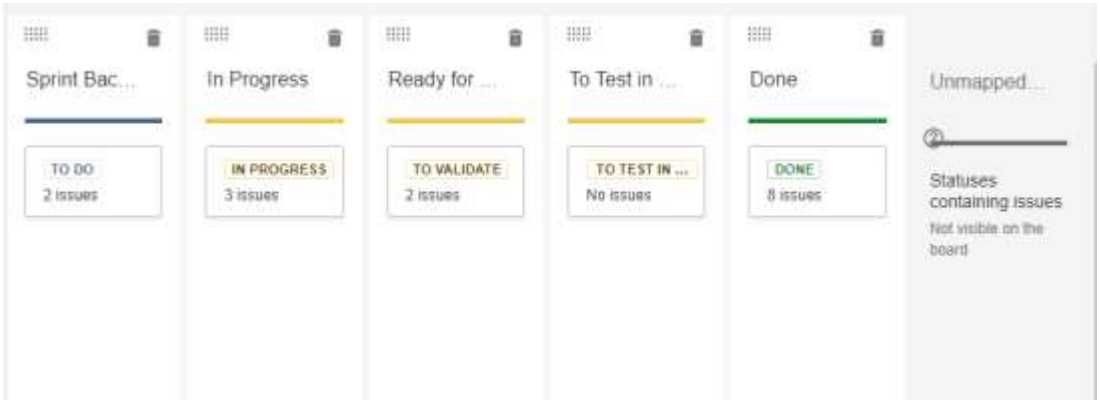


Gráfico 51. **Fases agregadas al tablero Scrumban**

Se adicionan las columnas o fases utilizadas en el tablero Kanban y se incluye la nueva fase identificada en la última reunión “To Test in Pre-Production”.

Adicionalmente, se realiza un pequeño ajuste en el nombre de la columna “To Do”, que desde este momento es llamada “Sprint Backlog”, en referencia al Backlog de la iteración actual y se elimina el WIP que estaba definido en un valor de 4, ya que en el sprint se podrían tratar más de 4 requerimientos.

Filtros del tablero

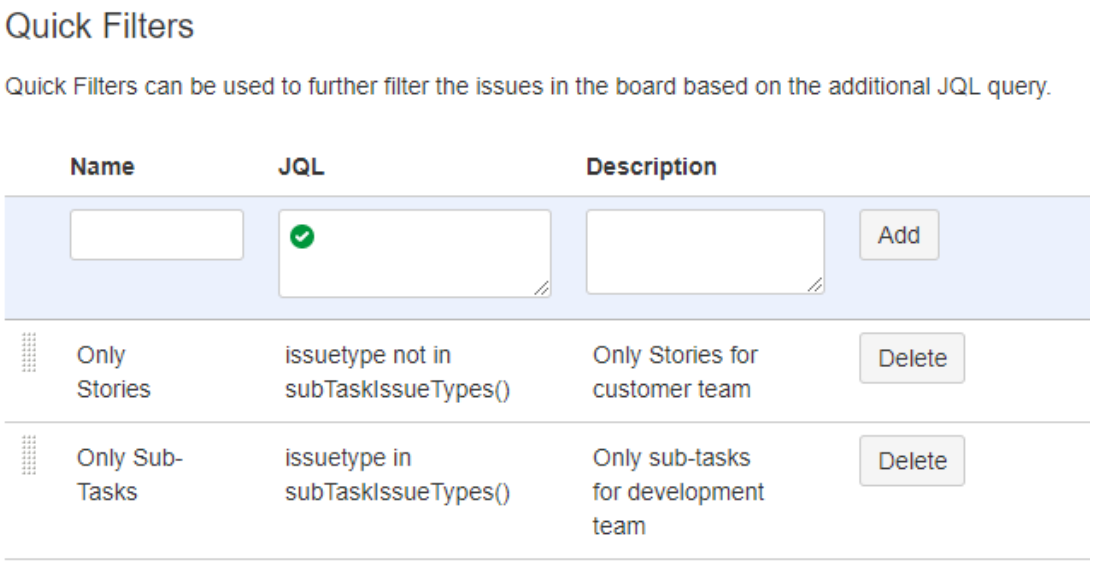


Gráfico 52. **Filtros reconfigurados en el nuevo tablero**

Se configuran los mismos filtros que anteriormente fueron utilizados en Kanban para ser utilizados en este nuevo tablero. Esto tiene como objetivo que los diferentes niveles de participantes dentro del proyecto (operativo, gerentes y directores), puedan disponer de una visión de lo que está sucediendo en el proyecto con las diferentes granularidades.

Días hábiles y zona horaria

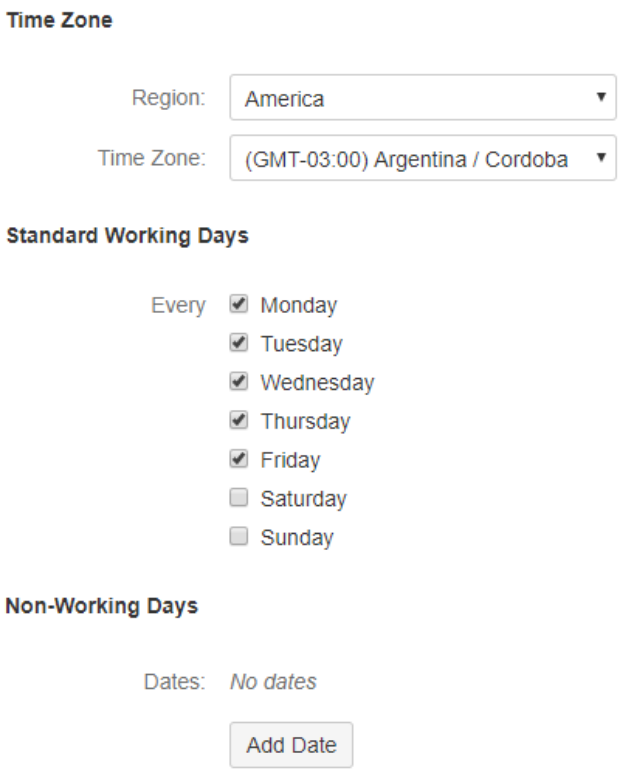


Gráfico 53. **Días hábiles de trabajo**

Se configuran los días laborales hábiles dentro del tablero Scrum y la zona horaria para calcular el termino de los días y el sprint.

Requerimientos próximos a desarrollar

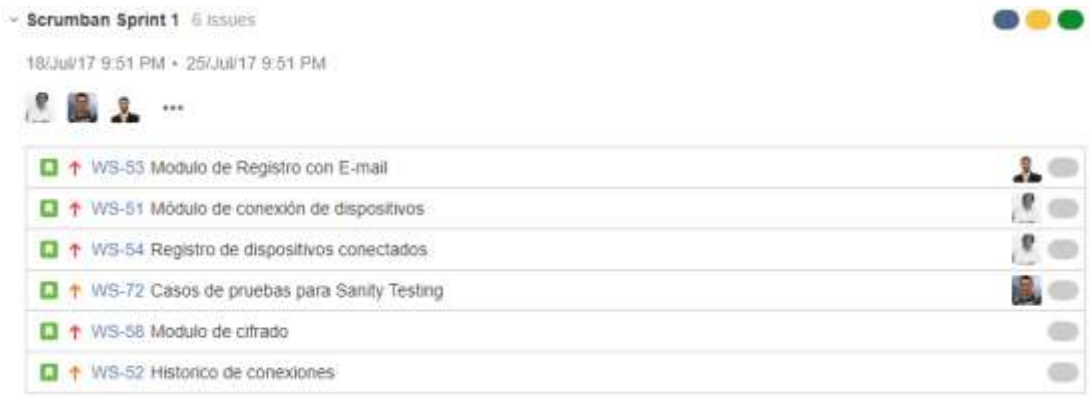


Gráfico 54. **Requerimientos en el sprint backlog**

Por último y en relación a los requerimientos a desarrollar, se incorporan a este nuevo tablero, aquellos requerimientos que ya estaban siendo trabajados la semana pasada y, además, se suman aquellos que se encontraban en la lista de “To Do” en el tablero Kanban. Siendo estos últimos los priorizados por el Product Owner.

Tablero Scrum inicial

A continuación, se muestra el tablero ya configurado para ser utilizado con el enfoque de Scrumban, conjuntamente con todos sus requerimientos y sub-tareas creadas anteriormente:

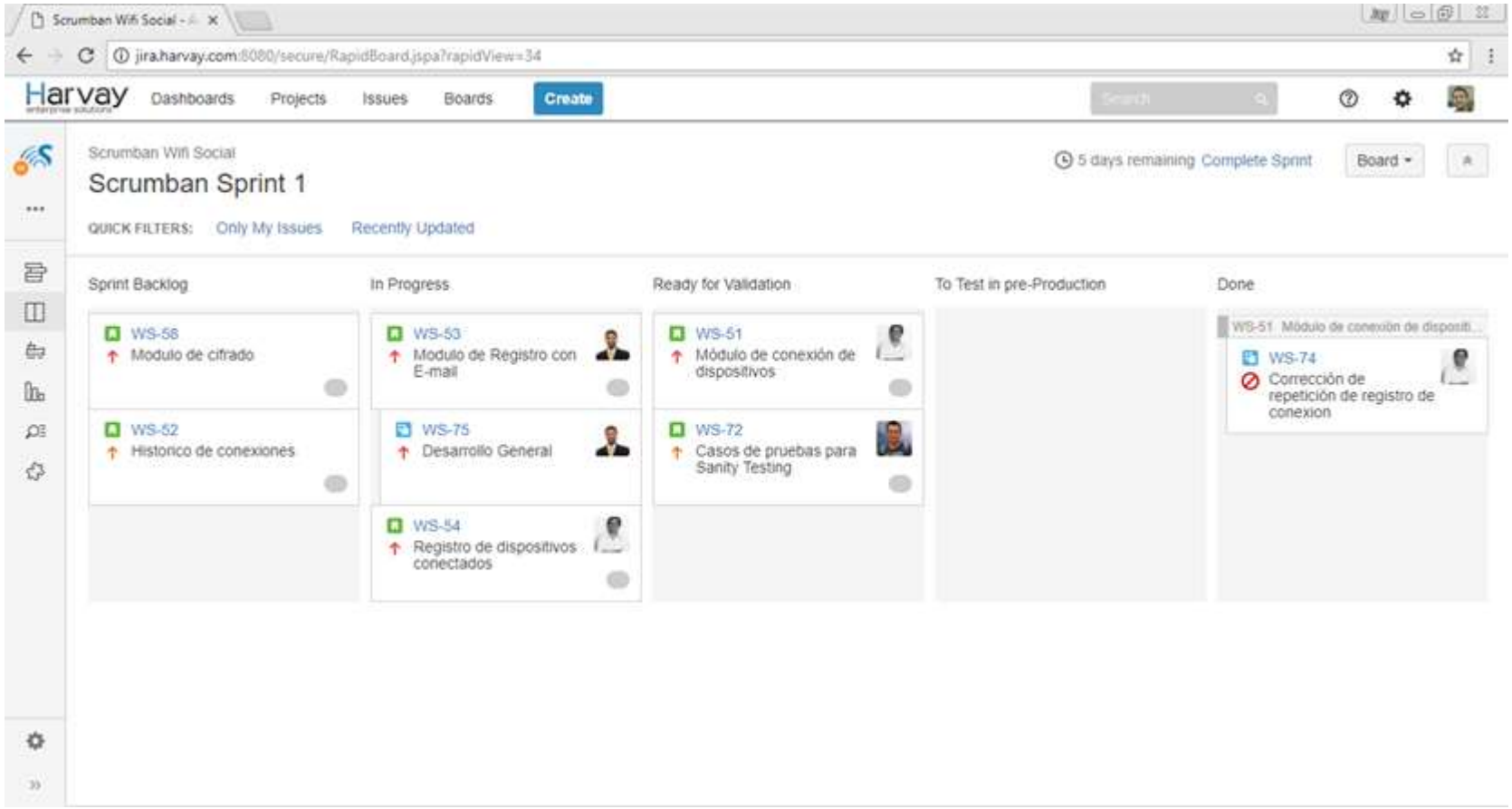


Gráfico 55. **Configuración de tablero scrumboard inicial**

Día 1 - 18 de Julio de 2017

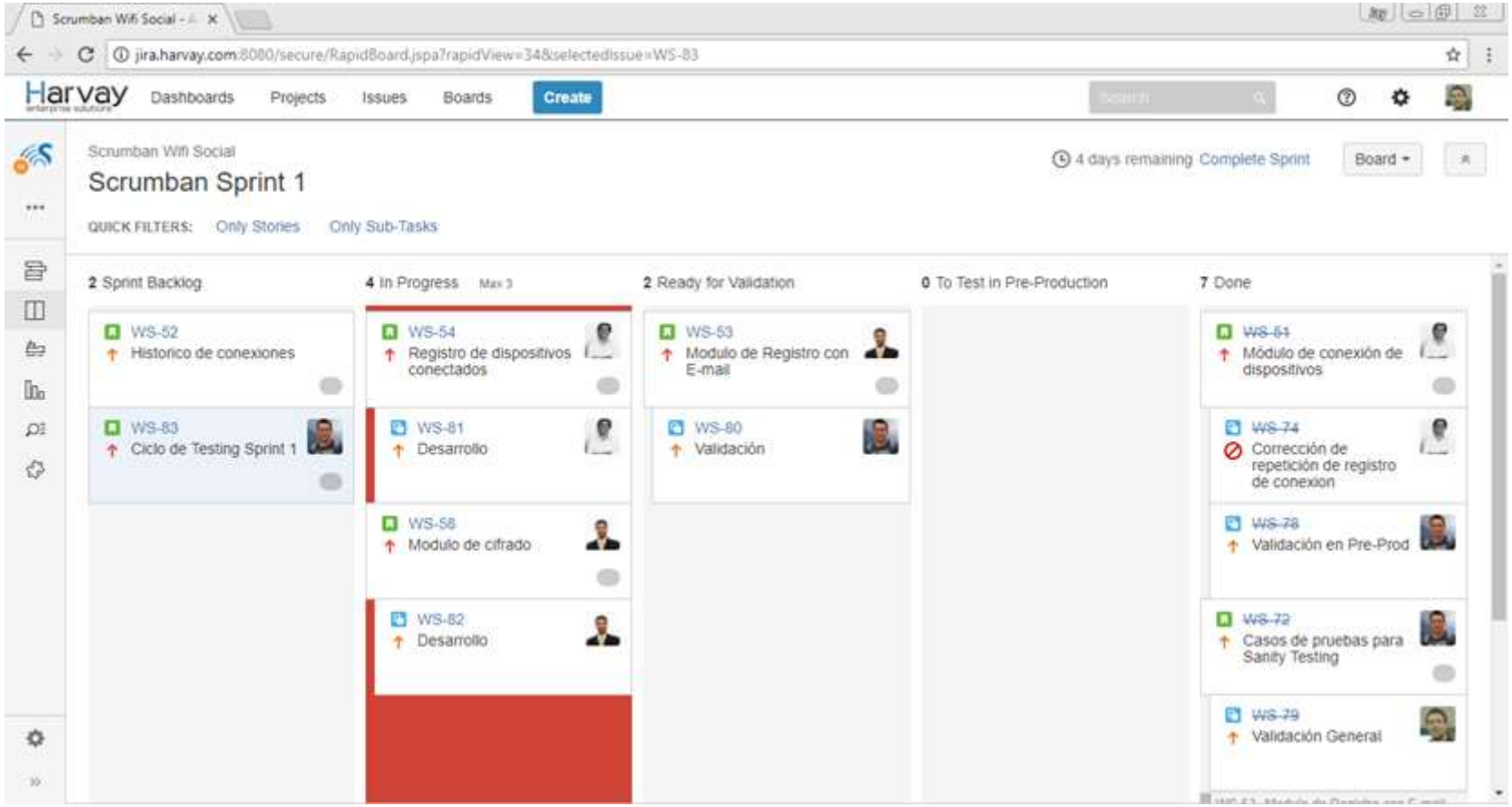


Gráfico 56. **Tablero Kanban Día 1 (sprint 1 - semana 3)**

Implementación de reunión stand-up

Al iniciar el primer día del primer sprint, todos los miembros del equipo tienen conocimiento de la nueva reunión en su calendario denominada “stand-up” (en español, “de pie”). Todo el equipo se conecta a Skype y luego de que Javier provee una breve introducción sobre el objetivo y la manera en que la reunión se debería desarrollar, el equipo comienza a sincronizarse uno a la vez, sobre tres (3) puntos esenciales: a) lo trabajado en el último día, b) lo que se trabajará en el día actual y c), si existe algún impedimento o problema que no esté permitiendo continuar con alguna tarea.

Todo el equipo provee su estado sobre su trabajo al resto y luego, se pasa al siguiente punto de la llamada de Skype, que es la planificación del sprint.

Implementación de reunión de planificación

En este segundo segmento de la llamada, y previamente al inicio de la reunión de planificación del sprint como tal, resulta importante debatir con el equipo, la duración de cada uno de los sprint. Si bien, este valor anteriormente fue definido por el Scrum Master, el mismo requiere una aprobación final por parte del equipo, ya que son quienes trabajarán dentro de ese marco y, por consiguiente, deberán cumplir con su compromiso en cada iteración.

Duración del sprint

Javier comienza exponiendo las razones por las cual él considera que el sprint debería ser definido en 1 semana de trabajo. El equipo comienza un debate en relación a que, si el sprint debería ser fijado en una (1) semana o en dos (2) semanas. Luego de 20 minutos cada uno expresarse en función de sus experiencias previas, el equipo decide iniciar la implementación de este nuevo enfoque Scrumban, con sprints de una (1) semana de duración.

Planificación del sprint

Luego de decidir la duración de las iteraciones, resulta oportuno identificar que va a ser desarrollado dentro de esa semana de trabajo en sí. Para ello, el equipo analiza los requerimientos que se encuentran en el backlog del sprint y también, en el backlog del proyecto. Se observa que el sprint backlog dispone de un requerimiento que no se dispone la suficiente experiencia para deducir y estimar cuanto podría dudar, esta es la card WS-58 – "Módulo de cifrado".

Como consecuencia de esta incertidumbre, se resuelve definir el compromiso del sprint con seis (6) historias de usuario a ser trabajadas y que, en algunos casos, ya se encuentran siendo desarrolladas desde la semana pasada. Esto incluye los mismos requerimientos que habían sido visualizados en la columna "To Do" del tablero Kanban, utilizado la semana anterior. Asimismo, el equipo considera que el nuevo enfoque Scrumban de la metodología de trabajo, necesitará inicialmente un tiempo de aprendizaje práctico, por lo que no agregar muchas historias de usuario al sprint, puede ser un diferencial de éxito al final de este primer sprint.

Detalle Jornada: Día 1 - 18 de Julio de 2017

Inicia el desarrollo del primer día del sprint luego de haber realizado la configuración del tablero y también, luego de haber sincronizado a todo el equipo en relación a los ajustes definidos. Javier comienza su verificación pendiente sobre WS-72, mediante la actividad creada para su correspondiente validación: WS-79. Logra realizar dicha validación sin encontrar problemas, por lo cual, transfiere ambas US (en inglés, "user stories") para la columna "Done".

Sobre el requerimiento WS-53, que estaba siendo desarrollado por Paulo en la actividad WS-75, la misma es completada luego de que se realice una revisión de a pares (en inglés, "peer review"), conjuntamente con Agustín para verificar si esta todo de acuerdo a la expectativa del requisito. Este chequeo de pares es realizado en la actividad WS-77 y luego de esto, se considera el requerimiento listo para enviarlo al

área de testing. Gustavo crea una actividad para esta validación (WS-80), y la deja lista para trabajarla luego de completar su ciclo de testing correspondiente a la actividad WS-51.

Sobre la tarjeta WS-51, se había aplicado una corrección al código y faltaba que dicha corrección sea validada por Gustavo, esto fue realizado exitosamente en la actividad WS-78 y ambas tarjetas se envían para la columna de Done.

Se inicia el desarrollo en 2 nuevas historias de usuario que se encontraban en el "Sprint Backlog". Estos son WS-54 y WS-58. Para ambos requerimientos, se ha agrega su correspondiente actividad de desarrollo (WS-81 y WS-82). Sin embargo, al final del día, se observa que el límite WIP ha sido superado y esto lleva a que, al siguiente día, se debata si esto es por un error de organización o si el límite WIP debería ser actualizado nuevamente.

Día 2 - 19 de Julio de 2017

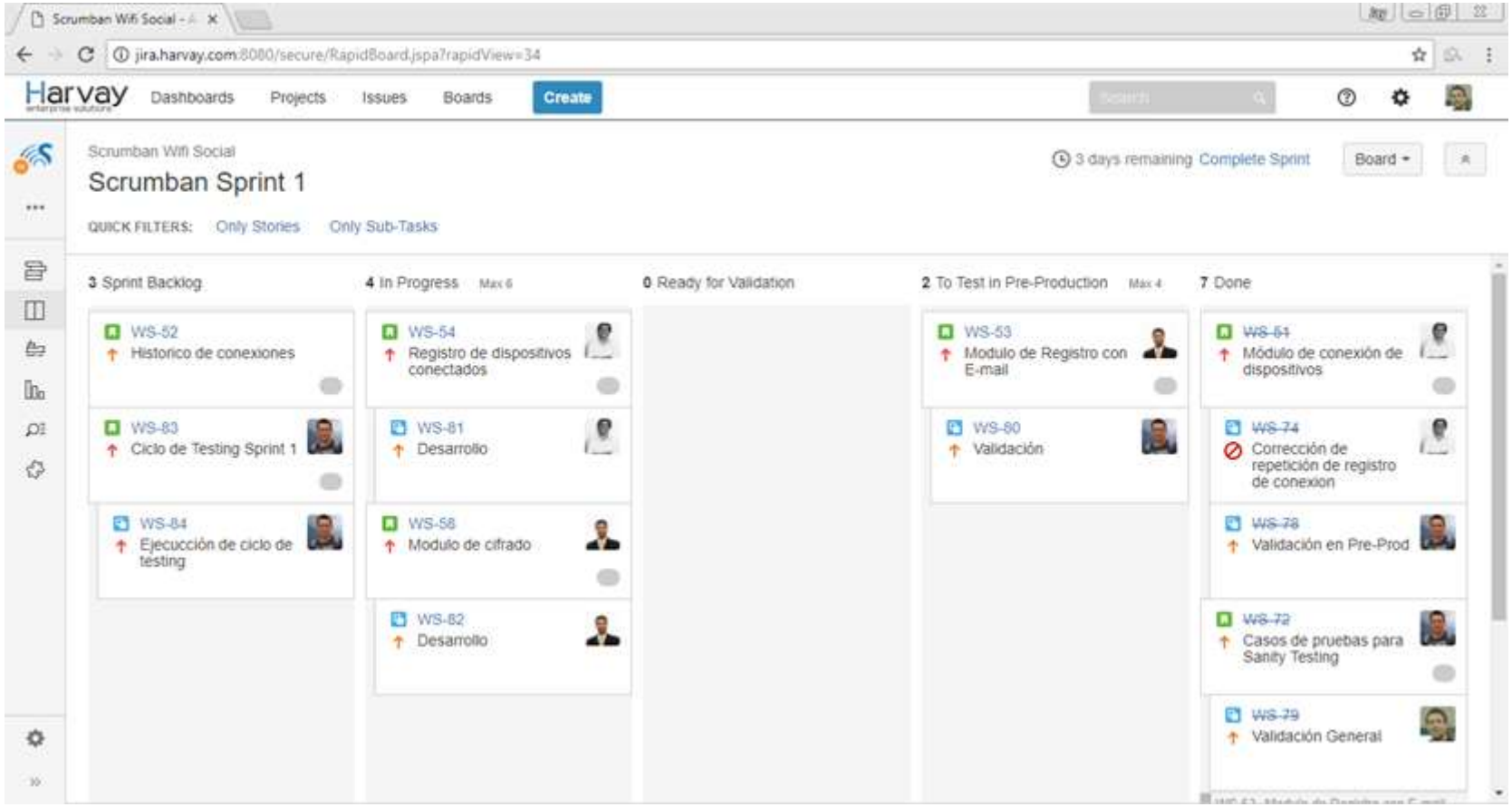


Gráfico 57. **Tablero Kanban Día 2 (sprint 1 - semana 3)**

Redefinición de límite de WIP

Durante la reunión stand-up que se realiza por medio de Skype, se discute inicialmente la definición del valor límite de WIP, para entender si este necesita ser actualizado o si se agregó más trabajo del que se puede realizar. Posteriormente al debate entre los integrantes del equipo, se entiende que luego de implementar las sub-tareas o; actividades para cada historia de usuario, se observa mayor cantidad de ítems en el tablero y específicamente en la fase “In Progress”, se podrían llegar a acumular hasta 6 ítems dentro de la columna. Esto sucede, porque tanto cada uno de los desarrolladores, como el analista de testing, podrían estar trabajando en actividades de diferentes historias de usuario, con lo cual, supone tres (3) historias de usuario con una (1) actividad o tarea por cada una, sumando un total de 6 ítems en la susodicha columna. Por consiguiente, el equipo acuerda y define que el límite WIP para la columna “In Progress”, debería tener un valor de 6.

En lo que respecta a las otras fases del proceso, la única otra columna que se considera que debería actualiza su límite WIP, es la columna de “To Test in Pre-Production”, en la cual se entiende que puede existir dos (2) historias de usuario, con cada una, una tarea relacionada a testing. Aquí vale la pena aclarar un punto importante en relación a las actividades de testing. Como ocurrió anteriormente, algunos esfuerzos del área de testing, también pueden ser ejecutados por parte del Scrum Master, en casos de suma urgencia en los cuales es necesario duplicar el esfuerzo de calidad de software. Por todo esto, el valor limitante considerado para la fase de “To Test in Pre-Production”, es de 4 ítems.

Detalle Jornada: Día 2 - 19 de Julio de 2017

Ambos desarrolladores continúan trabajando en los requerimientos: a) WS-54: con la actividad WS-81 (Agustín) y b) WS-58: con la tarea WS-82 (Paulo). Al final del día, Gustavo finaliza la actividad WS-80 de la historia de usuario WS-53, para continuar con la tarea WS-84, como parte del ciclo de testing de este sprint.

Día 3 - 20 de Julio de 2017

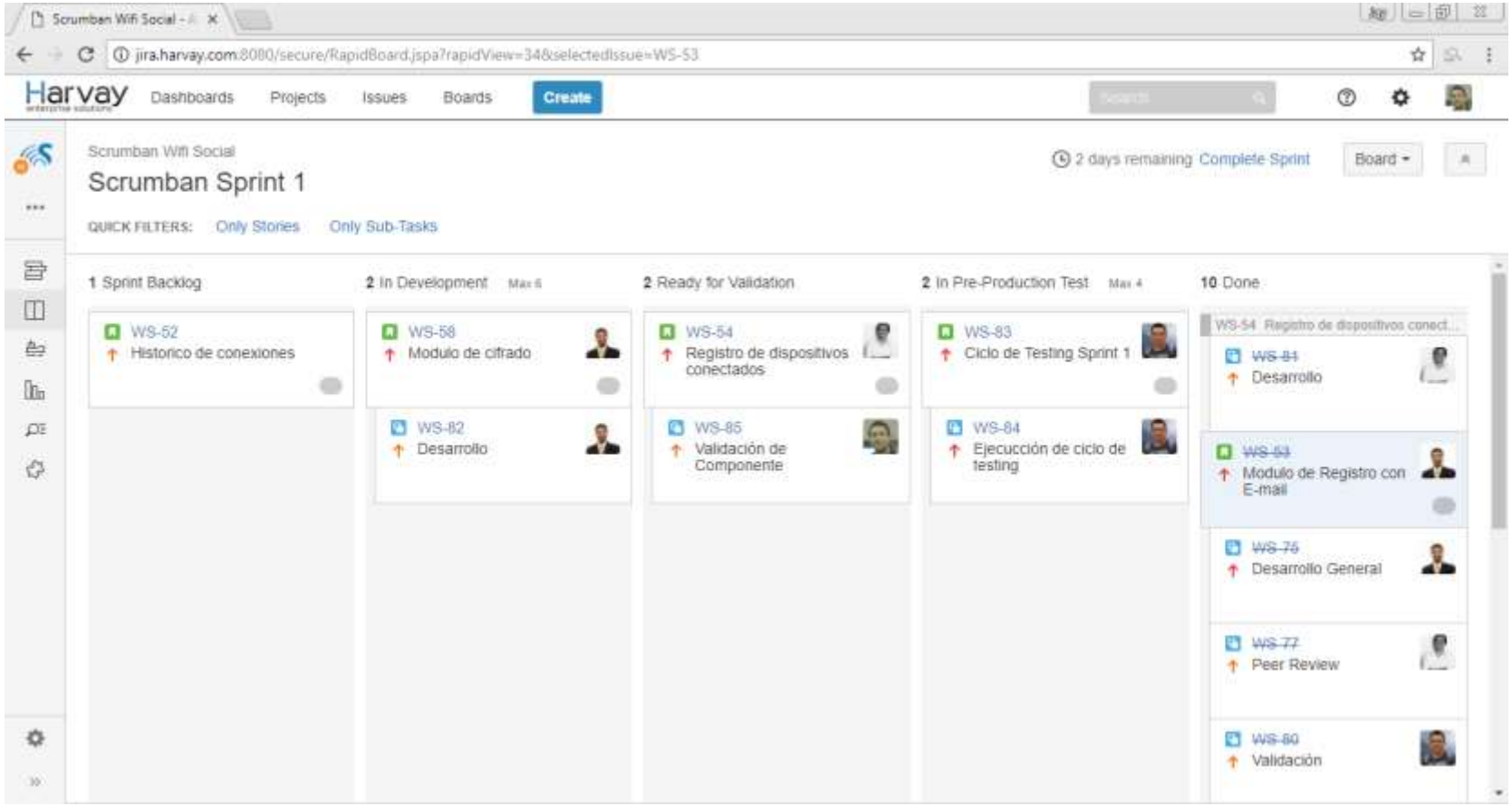


Gráfico 58. **Tablero Kanban Día 3 (sprint 1 - semana 3)**

Cambio de nombres en columnas tablero

Durante la reunión stand-up, parte del equipo recomienda renombrar algunas fases del tablero Scrumban, para evitar confusión a la hora de transicionar las actividades. Los cambios en los que todos aceptan en realizar son los siguientes:

- "In Progress": de forma de evitar confusiones entre los desarrolladores y el analista de testing, es necesario diferenciar esta columna. Es por ello que el nuevo nombre de la misma es: "In Development", haciendo referencia a "en desarrollo" en español.
- "To Test in Pre-Production": el mismo caso sucede aquí con el analista de testing al realizar validaciones en el ambiente de Pre-Producción, previo al envió del ítem a la columna "Done". Es por eso que el nuevo nombre de la fase será "In Pre-Production Test", haciendo alusión a que el ítem se encuentra en testing en el ambiente de Pre-Producción.

A continuación, se muestra una imagen del setup realizado en el tablero Scrumban:

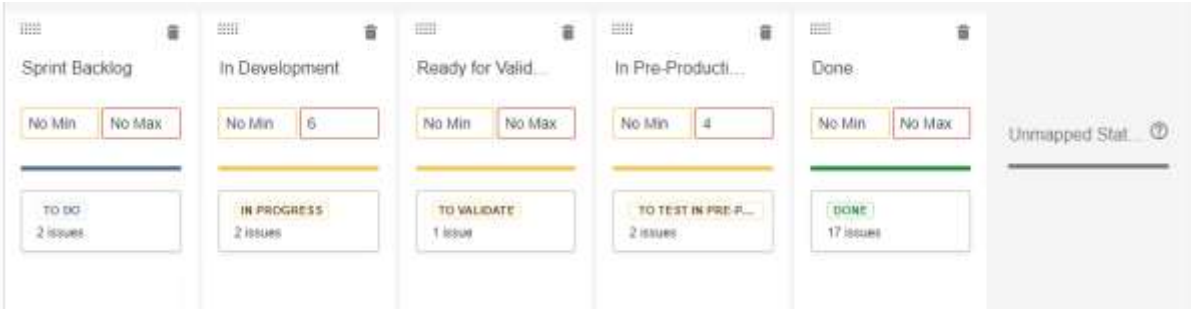


Gráfico 59. **Cambio de nombre en las fases del Scrumban board**

En lo que respecta al límite WIP, no es necesario realizar cambios, ya que este ajuste es un mero esclarecimiento para que la organización del equipo trabaje más fluidamente.

Detalle Jornada: Día 3 - 20 de Julio de 2017

Durante el día 3 del sprint, el equipo comienza a entender que le quedan 2 días para trabajar todo lo pendiente y comprometido del sprint. Agustín, por un lado, logra completar la actividad WS-81, relacionada al "registro de dispositivos conectados" (WS-54) y esto lo libera

para ayudar a Paulo en la actividad WS-82, correspondiente a la historia de usuario WS-58, que se sitúa en la fase de “In Development”.

Por otro lado, Javier identifica un escenario en el que puede ayudar a acelerar el trabajo, tomando una actividad de validación creada por Gustavo. Esta es la actividad WS-85 que fue creada para validar la user story WS-54 (que recién ha terminado de desarrollar Agustín). Luego de realizar una sincronización entre ambos, Gustavo le indica a Javier que ya se dispone de una batería de casos de pruebas creadas exclusivamente para validar ese requerimiento. Javier ejecuta cada uno de los casos de pruebas y al final del día, todo está de acuerdo a lo esperado por parte del cliente.

En paralelo, Gustavo comienza a realizar la ejecución del ciclo completo de testing (WS-84), sobre la última versión liberada a Pre-Producción.

Día 4 - 21 de Julio de 2017

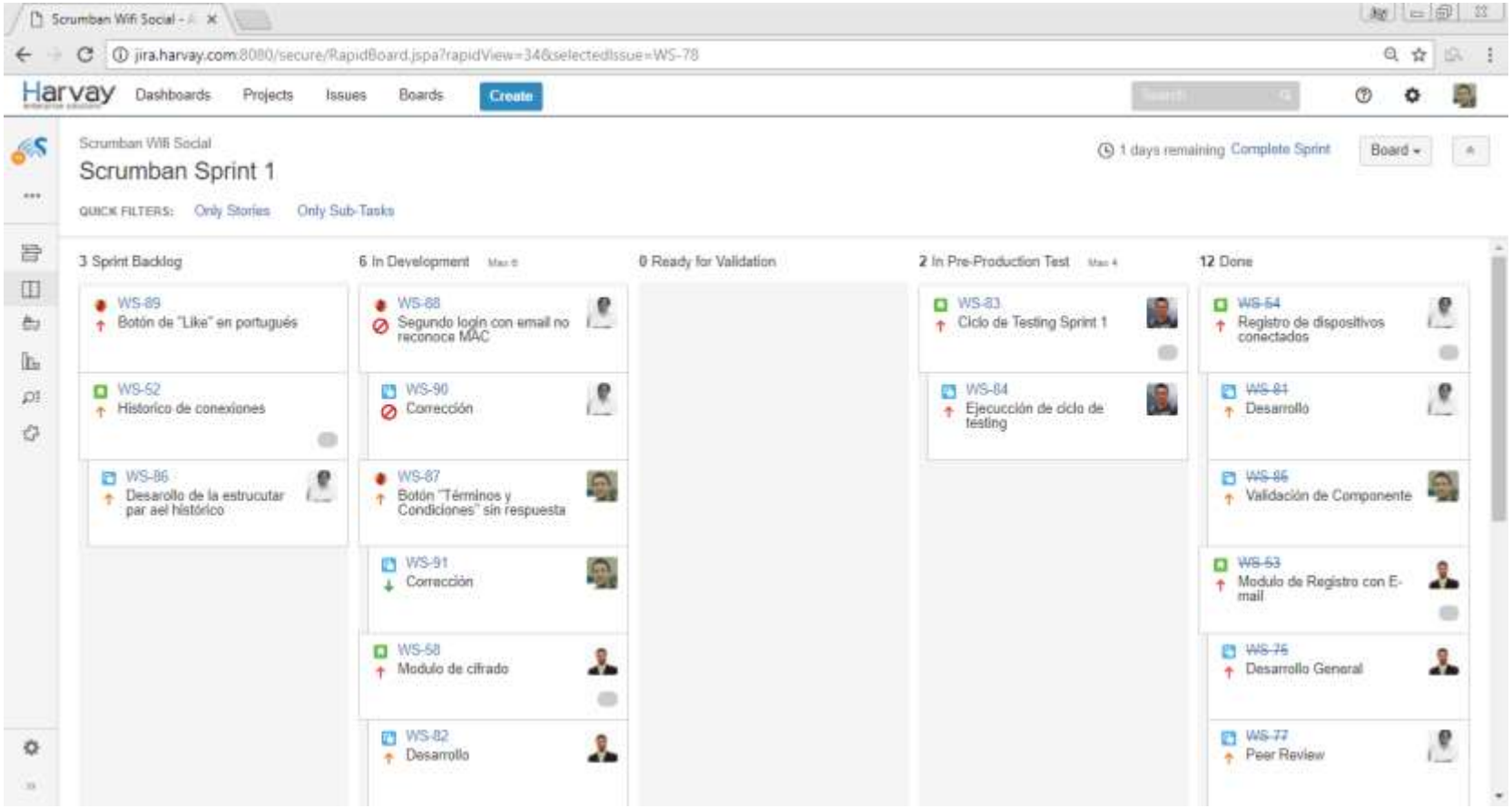


Gráfico 60. **Tablero Kanban Día 4 (sprint 1 - semana 3)**

Detalle Jornada: Día 4 - 21 de Julio de 2017

Esta anteúltima jornada antes de completar el primer sprint y, fruto de la ejecución de casos de pruebas (WS-84), por parte de Gustavo, son reportados 3 defectos (en inglés, “bug”), de diferentes prioridades: a) WS-88: muy alta, b) WS-89: alta, c) WS-87: media. Estos bugs son reportados a mitad de la jornada, lo que complica aún más la corrección para completarla durante el mismo día laboral.

Es por eso que se llama a una reunión de carácter urgente para evaluar las acciones de cada miembro a fin de resolver dichos bugs. Las acciones que se toman son las siguientes:

1. Agustín, pospone el desarrollo de la historia de usuario WS-52 (tarea creada por el con el id: WS-86 y toma el bug más prioritario de todos, que es el WS-88. Al momento, Agustín crea una actividad para trabajarlo (WS-90) y comienza su depuración (comúnmente llamada también, en inglés, “debugging”).
2. Paulo por otro lado, casi termina el módulo de cifrado y es por eso que, al ser un módulo que necesita la incorporación de un módulo de terceros, es mejor mantener el foco en el mismo de forma de no cambiar de actividad y crear una pérdida de foco y, por consiguiente, tiempo.
3. Javier: luego de verificar y validar en torno al defecto WS-87, encuentra la falla, crea una tarea para trabajarla (WS-91) y comienza su corrección. De esta forma, solo queda un (1) defecto pendiente y un requerimiento. No obstante, Javier entiende que pueden surgir nuevos bugs, ya que la ejecución del ciclo de testing no ha acabado. Es por ello que levanta el riesgo al Product Owner, de forma de alertar con un posible escenario, en el cual no se pueda completar con todas las historias de usuarios comprometidas.
4. Gustavo: continúa realizando la ejecución del ciclo de testing y estima que al día siguiente puede completar la ejecución. Por otro lado, al ejecutar diferentes casos de pruebas e intentar validar dicha información contra los requerimientos, encuentra mucha información que se encuentra descentralizada, lo cual general un problema para compartir conocimiento. Por lo cual, resurge un tema muy importante a tratar en la próxima reunión stand-up. La idea de Gustavo es implementar una herramienta que provea información de forma instantánea, buscando por palabras claves.

Día 5 - 24 de Julio de 2017

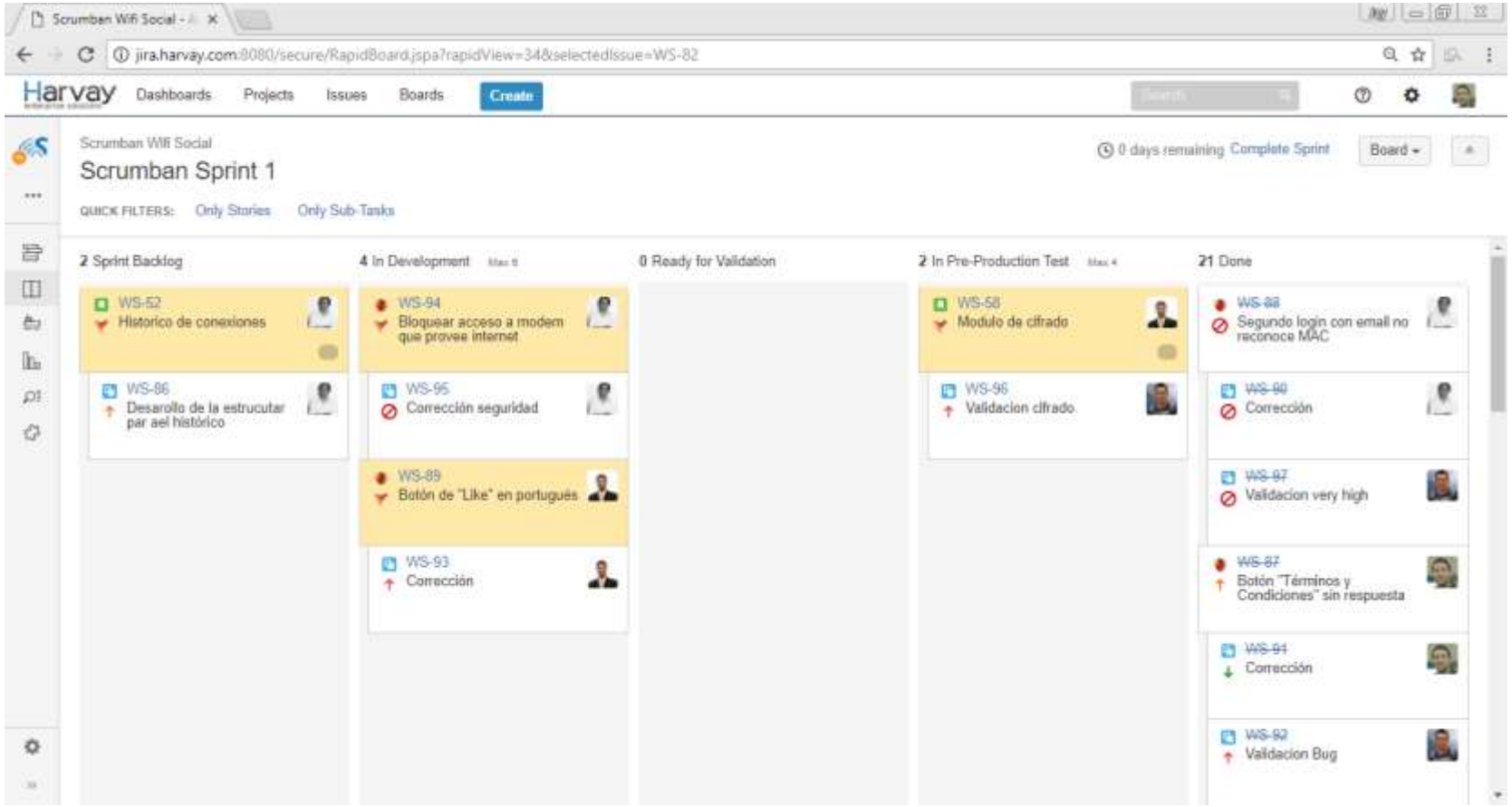


Gráfico 61. **Tablero Kanban Día 5 (sprint 1 - semana 3)**

Detalle Jornada: Día 5 - 24 de Julio de 2017

El último día del sprint se produce con algunos inconvenientes debido a los bugs que fueron encontrados en el ciclo de testing del sistema. Sin embargo, el equipo está dispuesto a demostrar que puede esforzarse y corregir dichos bugs, a fin de conseguir cumplir su compromiso.

Por un lado, Javier ya ha comunicado al product owner sobre la situación del proyecto en este último día del primer sprint. Por otro lado, Lucas advierte que de cierta forma podía suponer que esto ocurriría, por el hecho de que era el primer ciclo de testing completo que se realizaba sobre la primera versión del producto. En este sentido, el product owner recomienda al scrum master que el equipo se focalice en los defectos de mayor prioridad, a fin de liberar de una versión con las características básicas más importantes y que al final del día le informe cuales son los ítems que serán diferidos hacia el segundo sprint. Cabe aclarar que el término diferir en Scrumban, se refiere al acto de enviar ítems (sean requerimientos, defectos o tareas), desde un sprint hacia otro, debido a que el sprints origen ha culminado. Es decir, el trabajo se difiere de un período de trabajo hacia otro.

El equipo, por otro lado, intenta acelerar la marcha sin cometer errores. Gustavo completa el ciclo de testing sobre la primera versión del producto, en el ambiente de Pre-Producción y encuentra un cuarto defecto, que es reportado en el tablero bajo la identificación WS-94. El mismo es de carácter urgente ya que se trata de la seguridad del producto.

Agustín, logra corregir uno de los bugs que se encontraban en desarrollo (WS-88), que es de prioridad muy alta, bajo la actividad WS-90. Logra corregir el defecto y lo envía rápidamente para validación. Es allí donde Gustavo, comienza su labor de verificación (WS-97) sobre este mismo defecto (WS-88) y encuentra que el mismo se ha corregido de forma exitosa. Es por ello, que transfiere dichos ítems para la columna “Done”.

Luego de esto, Agustín continúa trabajando sobre el último de los defectos reportados por Gustavo (que es el último más prioritario de la lista): WS-94. Crea una

actividad para corregirlo (WS-95) y comienza su labor con pocas horas de completar la jornada.

En paralelo, Pablo, que estaba trabajando sobre el módulo de cifrado (WS-58), completa su desarrollo (WS-82) y envía dicha historia de usuario para la columna “Ready for Validation”. Ágilmente, prosigue con otro bug que fue reportado por Gustavo en relación al idioma de un botón (WS-89). Crea una actividad para su desarrollo (WS-93) y comienza su trabajo, también con pocas horas pendientes para finalizar la jornada.

Javier, quien había comenzado a corregir el bug WS-87, corrige dicho bug (WS-91) y envía dicho bug a ser validado. Gustavo, luego de completar la validación sobre el bug WS-88, valida de forma satisfactoria el bug corregido por Javier.

Por último, luego de haber validado varios bugs, Gustavo comienza a validar el módulo de cifrado (WS-58), mediante la actividad WS-96, pero no logra terminar su verificación a tiempo.

Con la última jornada del sprint casi completándose, Javier marca los siguientes ítems para ser diferidos al siguiente sprint, incluyendo sus sub-tareas: a) WS-52, b) WS-94, c) WS-89 y d) WS-58. Todos ellos, marcados también, en el tablero Scrumban con una bandera roja con color de fondo amarillo.

Incorporación de herramienta para soporte de documentación

Cabe aclarar que, antes de completarse este primer sprint y, además, fruto de un debate con el equipo de trabajo durante varias reuniones, el scrum master realiza la instalación y puesta a punto de una herramienta para documentar cualquier tipo de información relevante al proyecto de desarrollo. Esta plataforma se denomina “Doku Wiki” y posee un alto grado de personalización y practicidad. Asimismo, es de simple instalación, utilización e integración con otras herramientas, y provee una centralización de información dinámica en formato wiki, es decir, en forma de Wikipedia.

A lo largo de diversas reuniones, el equipo coincide en la implementación de dicha herramienta y Javier realiza la instalación y puesta a punto de la misma en un servidor. La misma se puede visualizarse en la URL: http://wiki.harvay.com y requiere un usuario y contraseña. Luego de realizar dicho setup, Javier migra toda la documentación del proyecto, que hasta ahora se encontraba en documentos de procesador de texto, documentos de texto plano y e-mails y los localiza dentro de una estructura en la herramienta Doku wiki.

A continuación, se visualiza la pantalla inicial de dicha herramienta

Gráfico 62. **Integración de "Wiki" del proyecto (Doku Wiki)**

Análisis sobre la finalización de sprint y sus indicadores

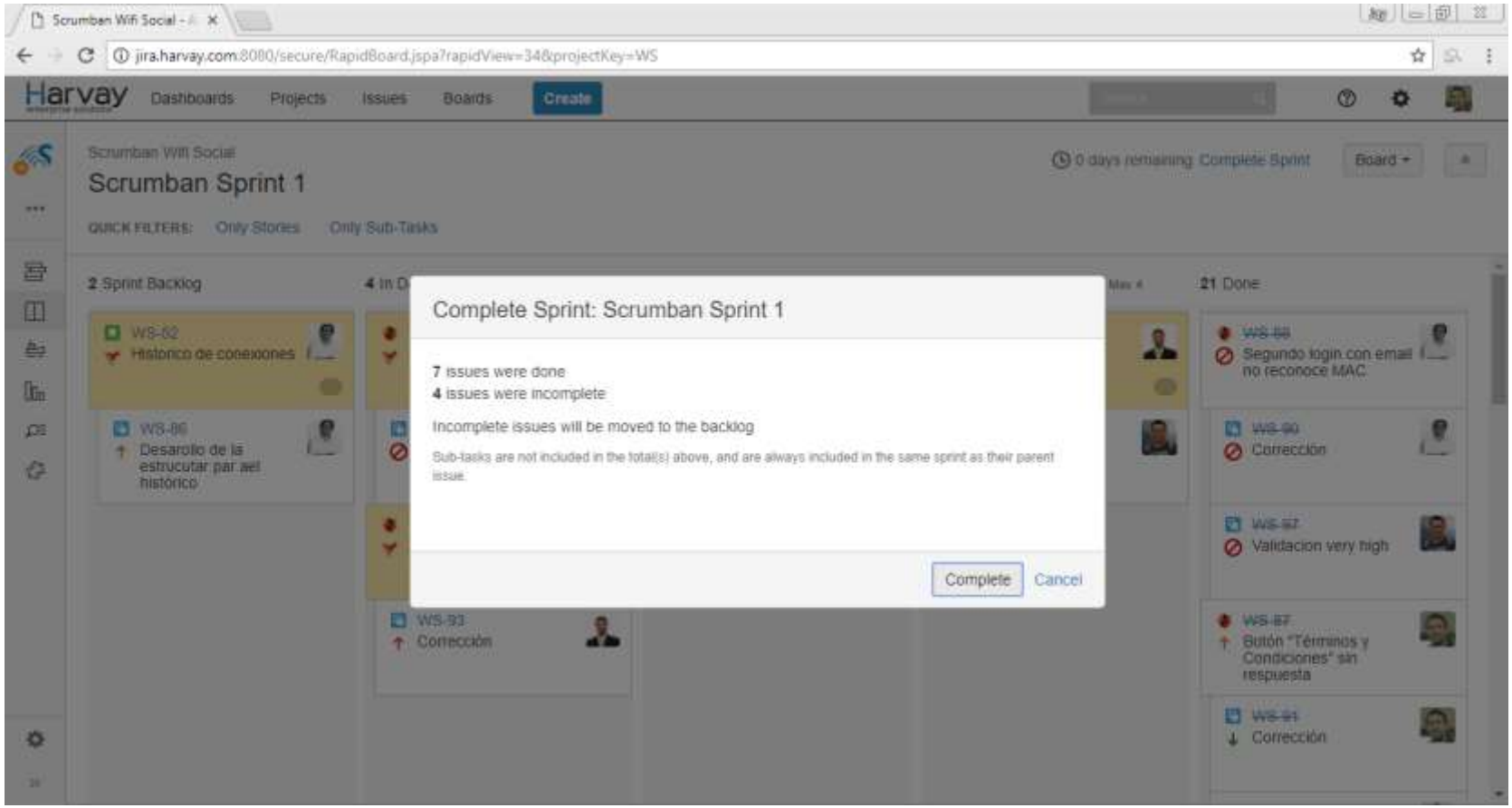


Gráfico 63. **Historias de usuario siendo diferidas**

Al cliquear en el link “Complete Sprint” en la parte superior derecha, se muestra este mensaje en el cual se informa:

- Siete (7) ítems fueron completados. Esto incluye cinco (5) historias de usuario y dos (2) defectos.
- Cuatro (4) ítems no fueron completados y necesitan ser enviados al backlog, para que posteriormente se los envíe para el siguiente sprint. Estos cuatro (4) ítems se dividen en dos (2) historias de usuario y dos (2) defectos pendientes de resolver.

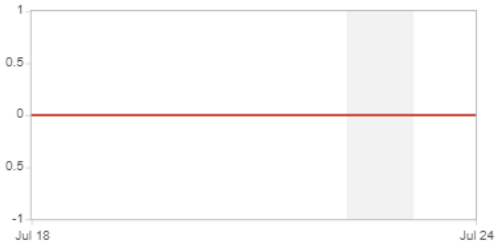
Los indicadores utilizados en Scrum y también, en Scrumban son basados en la utilización de puntos para la estimación de cada una de las historias de usuario en las que se trabaja. Estos puntos, generalmente son definidos durante el juego “poker planning”. Sin embargo, en este primer sprint completado, no se dispone de dichos puntos, ya que todavía no ha sido implementado el juego poker planning en el equipo. Por ello, en el gráfico siguiente, se muestra un gráfico sin movimientos, ni información relevante a esta tercera semana.

No obstante, se puede visualizar en el mismo reporte, los siete (7) ítems anteriormente denotados, conjuntamente con los otros cuatro (4) ítems que no fue posible completar y que, por ende, fueron diferidos. Se puede observar, también, los tipos de ítems, sus prioridades y el dato los puntos de cada ítem.

La expectativa para los próximos sprints es implementar diferentes características de Scrum, dentro de este enfoque Scrumban, de forma de poder disponer de diferentes indicadores que permitan realizar un análisis de mayor profundidad.

Scrumban Sprint 1

Closed Sprint, ended by Javier Salvay 18/Jul/17 9:51 PM - 24/Jul/17 10:47 PM



Status Report

Completed Issues

Key	Summary	Issue Type	Priority	Status	Story Points (-)
WS-51 *	Módulo de conexión de dispositivos	Story	High	DONE	-
WS-53 *	Modulo de Registro con E-mail	Story	High	DONE	-
WS-54 *	Registro de dispositivos conectados	Story	High	DONE	-
WS-72 *	Casos de pruebas para Sanity Testing	Story	Medium	DONE	-
WS-83 *	Ciclo de Testing Sprint 1	Story	High	DONE	-
WS-87 *	Botón "Términos y Condiciones" sin respuesta	Bug	Medium	DONE	-
WS-88 *	Segundo login con email no reconoce MAC	Bug	Highest	DONE	-

Issues Not Completed

Key	Summary	Issue Type	Priority	Status	Story Points (-)
WS-52 *	Historico de conexiones	Story	Medium	TO DO	-
WS-58 *	Modulo de cifrado	Story	High	TO TEST IN PRE-PRO...	-
WS-89 *	Botón de "Like" en portugués	Bug	High	IN PROGRESS	-
WS-94 *	Bloquear acceso a modem que provee internet	Bug	Highest	IN PROGRESS	-

Gráfico 64. **Reporte de finalización de sprint 1**

CAPÍTULO IX

MEJORAS POTENCIALES

Se identifican diversos puntos relacionados a mejoras potenciales que son significativas para ambos enfoques y que ayudarán al equipo en su desempeño para cumplir con su objetivo final en el proyecto. A continuación, se especifican las mejoras identificadas para cada uno de los enfoques, aplicadas a este proyecto de sistemas que se ha documentado.

Asimismo, es importante destacar que estas mejoras potenciales pueden ser incorporadas en ambas orientaciones ágiles, pudiendo ser implementadas en forma separada o combinándose unas con otras para un mayor beneficio.

Mejoras potenciales identificadas en enfoque kanban

A continuación, se desarrolla cada uno de los puntos de mejoras que son reconocidos como aptos para potencializar la utilización de este enfoque ágil en el presente proyecto:

Utilización de indicadores kanban

La utilización de indicadores dentro de Kanban ayuda a identificar, por un lado, las diferentes situaciones que pueden complicar al proyecto, como lo son los riesgos y problemas o inconvenientes que se pueden producir de un momento a otro. Es claro que un seguimiento constante de los riesgos del proyecto, puede ser vital al momento de individualizar de forma anticipada, un posible problema futuro.

Durante la implementación práctica de Kanban, se han documentado los dos (2) de los indicadores más importantes de dicho enfoque, como lo son el tiempo de entrega y el ciclo de entrega. Sin embargo, existen otros indicadores que también pueden ser de utilidad para identificar diferentes situaciones, desde diferentes perspectivas. A

continuación, se nombran y se describen brevemente algunos indicadores que también podrían formar parte del enfoque Kanban en este proyecto:

1. Tiempo en filas de espera: en casi toda la mayoría de los tableros Kanban de proyectos de sistemas de información, existen filas de espera en las cuales, los ítems quedan estacionados allí durante un tiempo, esperando el momento en el cual, algún integrante del equipo tome ese ítem, lo trabaje y continúe el flujo del proceso de desarrollo. A lo largo del proyecto presente en este documento, se identifica una fase que corresponde a esta categoría: “Ready for Validation”. Dicha categoría es una fila, a la espera de que alguien se asigne el ítem que llega a la misma y lo transicione a la próxima columna (en este caso es “In Pre-Production Test”), para efectuar su labor de validación dentro del ambiente de Pre-Producción. El objetivo de este indicador es medir el tiempo promedio que los ítems pasan estacionados en dicha fila de espera. Si dicho tiempo, representa un gran porcentaje dentro de la composición del cycle time, esto quiere decir que es necesario realizar algún ajuste para disminuir dicho tiempo de espera de los ítems que pasan por allí.
2. Distribución del esfuerzo: ¿qué sucedería si los indicadores lead time y cycle time fueran realmente excelentes, pero el equipo solo estuviera corrigiendo defectos y no trabajara en el desarrollo de nuevas características del producto? Esto no es realmente aceptable, ya que se estaría trabajando solo en actividades que no agregan valor al producto (defectos encontrados, por ejemplo). Si bien, la mejor situación (por cierto, utópica), sería solo trabajar en actividades que agreguen valor al producto, como lo son las nuevas funcionalidades, esto no sucede en el mundo real. Es por eso que, siempre es bueno tener un nivel equilibrado entre las actividades que agregan valor al producto, y aquellas actividades que no agregan valor en forma directa, pero son necesarias para que dicho producto funcione de manera correcta. De esto se trata este índice, de controlar que el porcentaje de las actividades que agregan valor siempre sea superior (por más que sea poca la diferencia), a las actividades que no agregan valor de forma directa.

Concluyentemente, este indicador podría beneficiar a la toma de decisiones en este proyecto.

3. Número de defectos recurrentes: los defectos recurrentes representan un malestar importante para aquellos clientes que encuentran el mismo defecto una y otra vez. Este indicador calcula la cantidad de defectos recurrentes en cada área o funcionalidad del sistema, de forma de entender a donde disponemos de un problema recurrente. Al identificar dicha recurrencia, el equipo puede investigar a fondo, para encontrar la causa raíz que lleva a que este defecto aparezca una y otra vez. Este tipo de indicador, es muy útil para proyectos largos, que se prolongan por mucho tiempo y, proyectos de gran escala, que disponen de gran cantidad de funcionalidades desarrolladas. Si bien, en este proyecto no hubo ninguna recurrencia de defectos, sería interesante disponer de este indicador de forma automática, para tener el control de algo tan sensible como los son los defectos recurrentes.

Diferenciación de tarjetas con diversos colores

Este es uno de los aspectos más simples de implementar en cuanto a las mejoras potenciales al enfoque. La diferenciación de colores en las tarjetas Kanban, es una mejora rápida de implementar debido a su simplicidad. Esta supone la categorización de cada uno de los tipos de ítems del tablero y la posterior asignación de un color en particular, para cada tipo de ítem. Esto hace que los participantes en el tablero, rápidamente puedan diferenciar cada tipo de tarjeta y, por ende, de actividad.

Incorporación de swimlanes o filas

El uso de swimlanes, niveles, o filas dentro del tablero, facilitan la organización del esfuerzo de trabajo en filas diferentes, que poseen diferente priorización y capacidad. Esto supone que cada fila tendrá una cantidad de participantes diferentes, una de la otra, con exclusividad de trabajo solo para los ítems de su fila en particular.

Sin embargo, los participantes de cada una de las filas, pertenecerán al mismo equipo de trabajo.

Una de las configuraciones mayormente empleadas por los equipos es la división de 2 swimlanes o filas: a) defectos e ítems urgentes y b) nuevas funcionalidades y todos los demás ítems. Siendo que los primeros, suponen la priorización de ítems urgentes que necesitan ser trabajados al momento en que llegan y luego, los defectos del producto que son identificados durante la ejecución de los ciclos pruebas de calidad.

En lo que respecta a la fila de las nuevas funcionalidades y los ítems restantes, aquí se dispone de la capacidad suficiente para trabajar todas las nuevas funcionalidades que componen al producto. En cuanto a la asignación de capacidad para cada una de las filas, existen diferentes estrategias. Una de ellas es asignar a todo el equipo al swimlane de nuevas funcionalidades y, en el caso de surgir algún ítem urgente o defecto prioritario en la otra fila, el miembro del equipo libre o más próximo a completar alguna actividad, es asignado a trabajar en la otra fila. Otra estrategia es dividir al equipo para alocar parte en una fila y parte en otra.

La utilización de swimlanes o filas dentro de las metodologías ágiles, es una práctica muy utilizada con beneficios importantes en equipos multidisciplinares.

Ciertamente, cada uno de los puntos expuestos y ejemplificados en este apartado, pueden ser implementados en este proyecto denominado “Wifi Social”, a fin de obtener mayor provecho de Kanban.

Mejoras potenciales identificadas en enfoque Scrumban

En lo que respecta a las mejoras potenciales identificadas sobre el enfoque Scrumban, se definen algunos puntos que ya fueron anteriormente detallados y debatidos por el equipo, en algunas de las reuniones documentadas. No obstante, por

diferentes cuestiones relacionadas a los ritmos del presente proyecto, no han podido ser integrados en dicha metodología.

A seguir, se enuncian de forma ordenada de acuerdo a su simplicidad de implementación:

Análisis de duración de sprint

A raíz del ritmo de trabajo observado durante las tres (3) semanas de desarrollo que fueron documentadas, se puede aseverar que la duración de los sprints necesita un nuevo análisis por parte del equipo, a fin de determinar si se debe continuar con sprints de una (1) semana de duración o, si es necesario incrementarlo a dos (2) semanas de duración.

Desde una perspectiva de planificación en cuanto a los ciclos de testing y la correspondiente corrección de los defectos que surjan de él, se concibe la necesidad de utilizar sprints de dos (2) semanas. Esto es para que se disponga de suficiente tiempo para ejecutar un ciclo o dos ciclos de testing completos a fin de validar y corregir defectos que puedan surgir.

Utilización de poker planning para estimar

El juego de poker planning es también, una mejora potencial que puede ser implementada de forma sencilla en este proyecto. Esta afirmación se debe a que el equipo ya posee experiencia previa sobre la dinámica de trabajo con dicho juego de estimación.

A su vez, las estimaciones generadas mediante el juego de poker planning no solo permite obtener beneficios directos, sino también beneficios indirectos asociados a la velocidad del equipo y a su estimación a lo largo de los sprints del proyecto.

Incorporación de swimlanes o filas

Al igual que en el enfoque Kanban, también se resalta la utilización de swimlanes en este Scrumban, de forma de organizar al equipo de una forma más eficiente.

Incorporación de reunión retrospectiva (retrospective)

La incorporación de la reunión retrospectiva al finalizar cada sprint es algo importante a tener en cuenta y sin duda, es una mejora potencial a implementar en este proyecto. Es necesario entender el desempeño del equipo en cada sprint y también, entender como calificar cada sprint hasta el final del proyecto.

Incorporación de reunión de revisión (sprint review)

Este tipo de reunión es fundamental para organizar a todos los participantes del equipo y poder coordinar la revisión del producto en un momento específico del sprint.

De acuerdo al marco teórico, esta reunión debería suceder en el momento final del sprint y sería siempre programada por el scrum master y el product owner. El ambiente que se utilizaría para realizar la demostración de las funcionalidades desarrolladas del producto (comúnmente llamada también “demo”), sería el ambiente de Pre-Producción.

CAPÍTULO XII

CONCLUSIÓN

Las metodologías ágiles se encuentran en continuo crecimiento en el mercado de tecnología de la información. Su capacidad de dinamismo y versatilidad frente a requerimientos cambiantes son, sin duda, uno de los factores diferenciales con respecto a otras metodologías existentes, de gran auge en el pasado.

Puntualmente, Kanban y Scrumban son enfoques ágiles que permiten efectuar su implementación de forma gradual, incorporando uno a uno sus componentes, en diferentes momentos del proyecto, y permitiendo una posterior adaptación de los mismos, de acuerdo a la realidad concreta del proyecto. Sin embargo, cabe destacar que estos dos enfoques, conjuntamente con todas las metodologías ágiles en sí, exponen algunos inconvenientes y restricciones también. De hecho, no es posible concebir una metodología de trabajo universal que pueda ser aplicada exitosamente en todos los proyectos de tecnología de la información. Toda metodología necesita una adaptación, un "tayloring", según el contexto donde necesite ser aplicada y, este contexto como tal, puede verse afectado por varios factores como los son: la experiencia del equipo en cuanto a las llamadas "habilidades ágiles", la experiencia técnica de cada miembro, el uso de tecnologías que no tengan un ciclo rápido de realimentación o que no soporten fácilmente el cambio y, por, sobre todo, la cultura organizacional en la cual el proyecto es aplicado.

Por otro lado, al final del camino es necesario concentrarse en que el proyecto tenga un desenlace exitoso y no, en la aplicación ortodoxa de una metodología en particular, ya que cada proyecto es diferente, al igual que su proceso de adaptación. En este mismo sentido, existe una fuerte corriente de profesionales "agilistas" que promueven el concepto de los métodos ágiles como un conjunto de herramientas que deberían estar disponibles para su uso en situaciones apropiadas, pero no como un método ortodoxo que debería valer y ser aplicado en todas las situaciones sin falta.

Como resultante de todo esto, se puede ratificar que es necesario buscar la combinación de componentes ágiles que resulten efectivos dentro de la naturaleza del proyecto, de forma de fusionarlos, y así trabajar con aquellos que contribuyen con beneficios puntuales para el equipo de trabajo y para el éxito del proyecto.

REFERENCIAS BIBLIOGRÁFICAS

Agile Spain www.agile-spain.com

Alianza ágil www.agilealliance.org

Ahmad Khan, Z, (2014), Scrumban - Adaptive Agile Development Process. Using Scrumban to improve software development process. Tesis de maestría. Helsinki Metropolia University of Applied Sciences. Finlandia. Disponible: https://www.theseus.fi/bitstream/handle/10024/77014/Khan_Zahoor.pdf?sequence=1

Arias, *L. (2009). Gestión de proyectos de tecnologías de información en la gerencia DIS*. Trabajo especial de Grado para la Especialización en Gerencia de Proyectos no publicado. Venezuela: Universidad Simón Bolívar. Disponible: http://159.90.80.55/tesis/000150367.pdf. [Consulta: 2016, febrero 18].

Beck, K. (1999). *Extreme Programming Explained*. Disponible: http://software2012team23.googlecode.com/git-history/5127389d21813c2bd955c53999f66cede994578b/docs/literature/Extreme_Programming_Explained_Kent_Beck_1999.pdf. [Consulta: 2016, marzo 18].

Bruce I. Blum. *Software Engineering: A Holistic View*

Caballero, O. (2006). *Tecnologías de información y herramientas para la administración de proyectos de software*. Articulo revista digital universitaria. Disponible: https://confluence.excentia.es/display/JIRADOC/Manual+de+JIRA. México.

Calero, A (2012). *Manual de Jira*. Disponible: https://confluence.excentia.es/display/JIRADOC/Manual+de+usuario. [Consulta: 2016, febrero 26].

De Amescua, A. (1990). Diseño de una metodología case de desarrollo de software. Tesis Doctoral. Universidad Politécnica de Madrid. Disponible: http://oa.upm.es/34996/7/AMESCUA_SECO_ANTONIO.pdf

DIXON, M. (2000). *Project management body of knowledge*. Disponible: http:// www.apm.org.uk. [Consulta: 2016, febrero 17].

Dorothy Graham, Erik Van Veenendaal, Isabel Evans y Rex Black, "Foundations of Software"
Duvall, P. (2007). *Continuous Integration. Improving Software Quality and Reducing Risk*.
Fabók, Z (2011). *XP with Kanban instead of Scrum*. Disponible: http://www.zsoltfabok.com/blog/2011/02/xp-with-kanban-instead-of-scrum/
Garzás, J (2013). *Kanbanize: una herramienta para gestionar los tableros Kanban de tus proyectos.* Disponible: http://www.javiergarzas.com/2013/10/herramienta-kanbanize.html
Haughey (2012). *Manual de administración de proyectos*. http://www.liderdeproyecto.com/manual/index.html
http://docplayer.es/1449365-Metodologia-agil-scrumban-en-el-proceso-de-desarrollo-y-mantenimiento-de-software-de-la-norma-moprosoft.html
http://modeloespiral.blogspot.com/2009/08/definicion.html
http://www.quimbiotec.gob.ve/sistem/auditoria/pdf/ciudadano/mtrigasTFC0612memoria.pdf
IBM (2016). Rational Team Concert. Disponible: http://www-03.ibm.com/software/products/es/rtc.
IEEE www.ieee.org/portal/site
INTECO (Instituto Nacional de Tecnologías de la Comunicación). Ministerio de Industria, Turismo y Comercio. Gobierno de España. (2009). *Ingeniería del Software: Metodologías y Ciclos de Vida*. Disponible: https://www.google.co.ve/?gfe_rd=cr&ei=e-zEVvShDIbd8gfs4JngBA#q=INTECO++Ingenier%C3%ADa+del+Software:+Metodolog%C3%ADas+y+Ciclos+de+Vida+. [Consulta: 2016, febrero 17].
Jacobson, I, Grady Booch y James Rumbaugh, (1999). *The Unified Software Development Process*.
Jibaja, F. (2011). Metodologías para el desarrollo del software. Disponible: http://www.academia.edu/9953322/Metodologias_para_el_desarrollo_de_software.

Ken Schwaber, Mike Beedle, (2008). *Agile Software Development with SCRUM.*
Kent Beck, Martin Fowler, (2000). *Planning Extreme Programming*
Kent Beck, *Test-Driven Development by Example*
Kniberg, H.y Skarin, M (2010). Kanban *and Scrum - making the most of both.* Disponible: http://www.agileinnovation.eu/wordpress/wp-content/uploads/2010/09/KanbanAndScrum_MakingTheMostOfBoth.pdf . [Consulta: 2016, febrero 17].
Ladas (2008). Scrumban *Essays on Kanban systems for Lean Software Development.* Disponible: https://books.google.co.ve/books?hl=es&lr=&id=SQFdAgAAQBAJ&oi=fnd&pg=PA7&dq=Scrumban-Essays+on+Kanban+Systems+for+Lean+Software+Development&ots=c89XRBXGNg&sig=I59JUW5uUe2KDdWgc-LuJnkQKd8#v=onepage&q=Scrumban-Essays%20on%20Kanban%20Systems%20for%20Lean%20Software%20Development&f=false
Lawrence-Pfleeger y Shari, (1998). *Software Engineering: Theory and Practice.*
Manifiesto ágil. www.agilemanifesto.org
Manual de usuario Jira. https://confluence.excentia.es/display/JIRADOC/Manual+de+usuario
Mitchel H. Levine. (2000). *Analyzing the Deliverables Produced in the Software Development Life Cycle.*
NEVILLE, T. (2005). Project management and software development methodology. Retrieved October 19, 2005 from http://pm.ittoolbox.com/documents/document.asp?i=1215
Paradigmas de programación. http://www.infor.uva.es/~cvaca/asigs/docpar/intro.pdf
Pérez, M (2011). *Guía comparativa de metodologías Ágiles*. Tesis de Grado en Ingeniería Informática de servicios y aplicaciones no publicado. España: Universidad de Valladolid. Disponible:

https://uvadoc.uva.es/bitstream/10324/1495/1/TFG-B.117.pdf [Consulta: 2016, febrero 17].

Pierre Bourque y Robert Dupuis. (2004). *Guide to the Software Engineering Body of Knowledge.*

PMBOK Project Management Institute. 5ta edición. [Documento en línea]. Disponible: www.pmi.org. [Consulta: 2005, febrero 18].

PMOinformatica.com (2012). *Herramientas de software para gestión de proyectos de desarrollo ágil*. Disponible: http://www.pmoinformatica.com/2012/08/herramientas-de-software-para-gestion.html. [Consulta: 2016, febrero 27].

Pressman, R. (2002). *Ingeniería de Software. Un enfoque práctico.* 5ta edición. España: McGraw-Hill/Interamericana de España.

Robert C. M. *Agile Software Development, Principles, Patterns, and Practices*

RODRÍGUEZ, J. (2002). Administración de proyectos de desarrollo de sistemas de información. Retrieved October 19, 2005 from http://www.monografias.com/trabajos15/sist-informacion/sist-informacion.shtml

Rodríguez, P. (2008). *Estudio de la aplicación de metodologías Ágiles para la evolución de productos de software*. Tesis de Master en Tecnologías de la información no publicado. España: Universidad Politécnica de Madrid. Disponible: http://oa.upm.es/1939/1/TESIS_MASTER_PILAR_RODRIGUEZ_GONZALEZ.pdf. [Consulta: 2016, marzo 5].

Ron Burback, *Software Engineering Methodology*. (1998)

Schenone, M. (2004). *Diseño de una Metodología Ágil de Desarrollo de Software*. Tesis de Grado en Ingeniería en Informática no publicado. Argentina: Universidad de Buenos Aires. Disponible: http://materias.fi.uba.ar/7500/schenone-tesisdegradoingenieriainformatica.pdf. [Consulta: 2016, febrero 17].

SENN, J. (1995). "ANÁLISIS Y DISEÑO DE SISTEMAS DE INFORMACION". Segunda e7dición. Mc Graw-Hill. México, 1995.

Sitio web de la Organización Internacional para la Estandarización www.iso.org
Sommerville, I. (2001). *Software Engineering*. 6ta Edición.
Swebok www.swebok.org
Testing - ISTQB® Certification" (2007)
Tong Ka lok, Kent, "Essential Skills for Agile Development" Sitios web
Vliet, H. (2002). *Software Engineering. Principles and Practice.* Tercera edición,
http://www.javiergarzas.com/2014/07/jira-agile-scrum-1.html
Moreno, M. (2010). http://bibing.us.es/proyectos/abreproy/70201/fichero/04+-+Lean+aplicado+a+la+Ingenieria+del+Software.pdf
http://www.leanroots.com/heijunka.html
(2003). Lean Software Development: An Agile Toolkit Lean Software Development: An Agile Toolkit. Disponible: http://200.17.137.109:8081/novobsi/Members/teresa/optativa-fabrica-de-sw-organizacoes-ageis/artigos/Addison%20Wesley%20-%20Lean%20Software%20Development%20-%20An%20Agile%20Toolkit.pdf.
Leading Lean Software Development: Results are Not the Point
http://api.eoi.es/api_v1_dev.php/fedora/asset/eoi:80094/EOI_LeanManufacturing_2013.pdf
repositorio.uchile.cl/handle/2250/104052
Blanco, S. (2008). Marble-Station. Disponible: https://www.marblestation.com/?p=66. Blog de apuntes.
https://wbsburgos.files.wordpress.com/2011/07/presetnacion-pmi.pdf
http://fi.ort.edu.uy/innovaportal/file/2021/1/scrum.pdf. *Material para la cátedra de Ingeniería de Software.*

GLOSARIO

- Quality Assurance: aseguramiento de la calidad del software (SQA): es un conjunto de actividades planificadas y ejecutadas sistemáticamente que apunta a asegurar que el software que se está construyendo es de alta calidad.

- Ciclo de desarrollo de software: tiempo entre la iniciación del desarrollo del software y la entrega del mismo.

- Marco de proceso común: es un proceso genérico que define las actividades requeridas para ejecutar un proyecto software. Puede ser adaptado para un proyecto específico.

- Métodos de ingeniería del software: los métodos de ingeniería del software proporcionan el "cómo" para la construcción del software. Abarcan las tareas técnicas requeridas para realizar y documentar el análisis de requisitos, el diseño, la construcción de programas, las pruebas y el mantenimiento.

- Modelos de proceso Ágiles: los modelos de proceso Ágiles son enfoques de desarrollo donde los requisitos del cliente se cumplen temprano en el ciclo de vida de desarrollo del software a través de continuas entregas de software. En estos modelos, los cambios en los requisitos son bienvenidos.

- Modelo de proceso de software: un modelo de proceso de software es una estrategia abstracta para la construcción y mantenimiento de un producto software. Ayuda a los jefes de proyecto en la planificación y ejecución de un proyecto. También se le llama modelo de ciclo de vida o paradigma de ingeniería del software.

- Modelo de proceso incremental: el modelo de proceso es un enfoque en el que el producto se entrega al cliente incrementalmente sobre un periodo de tiempo planificado.

- Proceso: secuencia de pasos para realizar alguna actividad e incluye la descripción de entradas, salidas, procedimientos, herramientas, responsabilidades y criterios de salida.

- Productos de trabajo: elementos que se crean durante el curso del proceso software, tales como documentos, software y manuales.

- **Refactorización**: es una actividad de examinar la estructura del software para eliminar redundancias, funcionalidad no utilizada y rejuvenecer objetos obsoletos mientras se mantiene el comportamiento observable. Esto asegura que la estructura del software permanece simple y fácil de modificar.
- **Framework**: es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.
- **Gráfico Burndown:** es una representación gráfica del trabajo pendiente a hacer versus el tiempo disponible para hacerlo. Es útil para predecir cuando el todo trabajo será completado.

ACRÓNIMOS

- **CAD**: Computer-Aided Design (Diseño asistido por computador)
- **CASE**: Computer Aided Software Engineeirng (Ingeniería de software asistida por computador)
- **CPF**: Common Process Framework (Marco de proceso común)
- **DSDM**: Dynamic Systems Development Method (Método de desarrollo de sistemas dinámico)
- **IEC**: International Electrotechnical Commission
- **IEEE**: Institute of Electrical and Electronics Engineers
- **ISO**: International Organization for Standardization
- **UP**: Unified Process (Proceso Unificado)
- **RAD**: Rapid Application Development (Desarrollo Rápido de Aplicaciones)
- **RUP**: Rational Unified Process (Proceso Unificado Rational)
- **SDLC**: Software Development Life Cycle (Ciclo de vida de desarrollo de software)
- **TDD**: Test Driven Development (Desarrollo orientado a pruebas)
- **UML**: Unified Modeling Language
- **XP**: Extreme Programming (Programación extrema)
- **PO**: Product Owner (Dueño del Producto)
- **SM**: Scrum Master (Maestro del Scrum)

ANEXO A. INSTALACIÓN DE JIRA

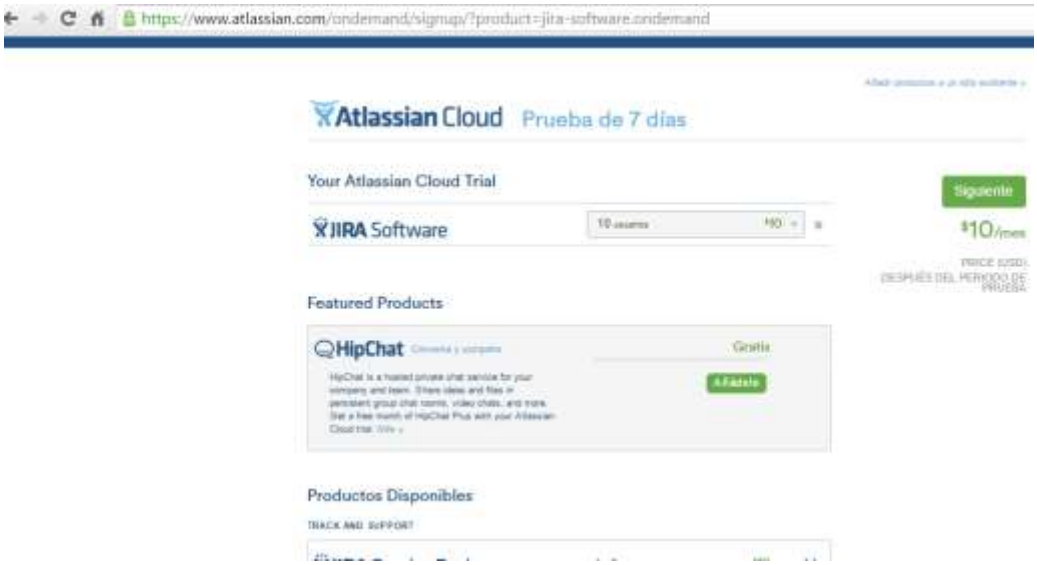
Se disponen de varias opciones para la instalación de Jira como herramienta de gestión para metodologías ágiles, se documentan en este anexo dos formas de obtener una instalación gratuita: (a) versión Jira Clouding: es una versión de prueba Online de 7 días y (b) versión JIRA Server: es una versión de prueba para descargar de 30 días.

Versión JIRA Clouding

1. Para instalar una versión gratuita de prueba de Jira Online, se debe entrar a la siguiente dirección: https://es.atlassian.com/software/jira. Aparecerá entonces una pantalla similar a la siguiente:

2. Se debe seleccionar la combinación de productos que se desea instalar. En nuestro caso instalaremos una versión de prueba gratuita por 7 días (opción Tu propia combinación).

3. Seguidamente se debe seleccionar el número de usuarios. En nuestro caso asumiremos el mínimo de usuarios: 10.



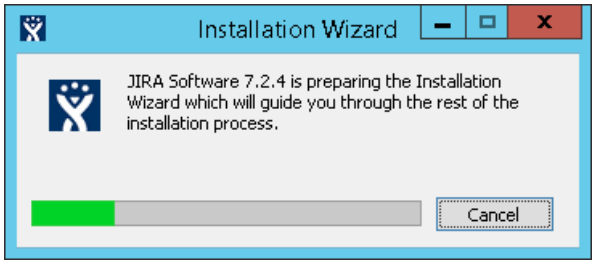
4. La versión de prueba del Jira se ejecuta desde la Web, por lo cual debemos elegir una dirección Web para el sitio; además de colocar nuestros datos personales como usuario.

Versión JIRA Server

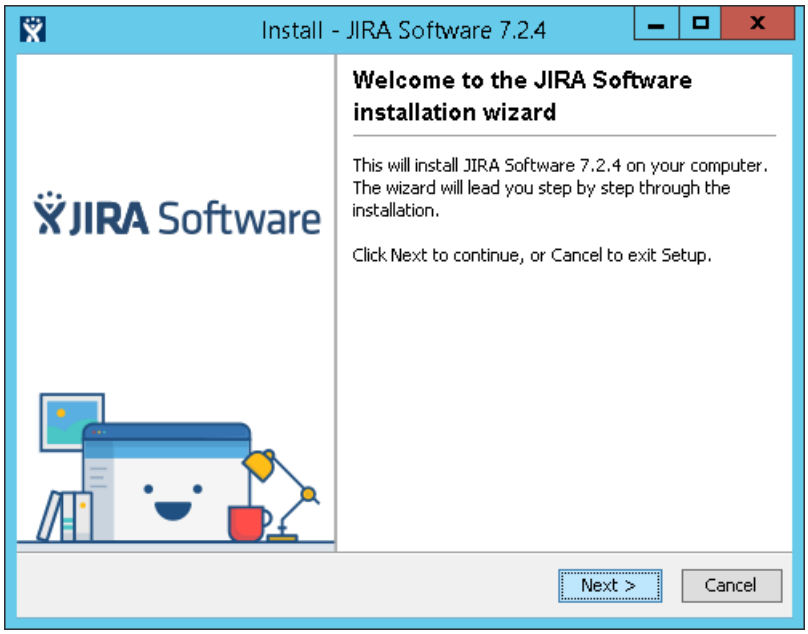
1. Para instalar una versión gratuita de prueba de Jira para descargar en un servidor local, se debe entrar a la siguiente dirección: https://es.atlassian.com/software/jira/download. En nuestro caso descargaremos una versión JIRA para Windows (64 bits). Aparecerá entonces una pantalla similar a la siguiente:

2. Al hacer Clic en el botón Descargar, se descargará un archivo en la carpeta de Descargas.

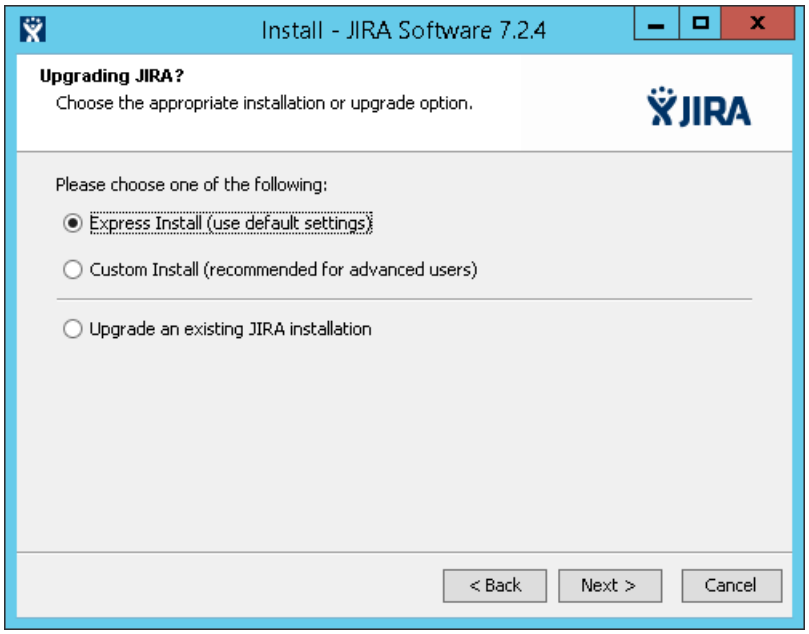
3. Se ejecuta el programa de instalación, preparando todo el asistente:



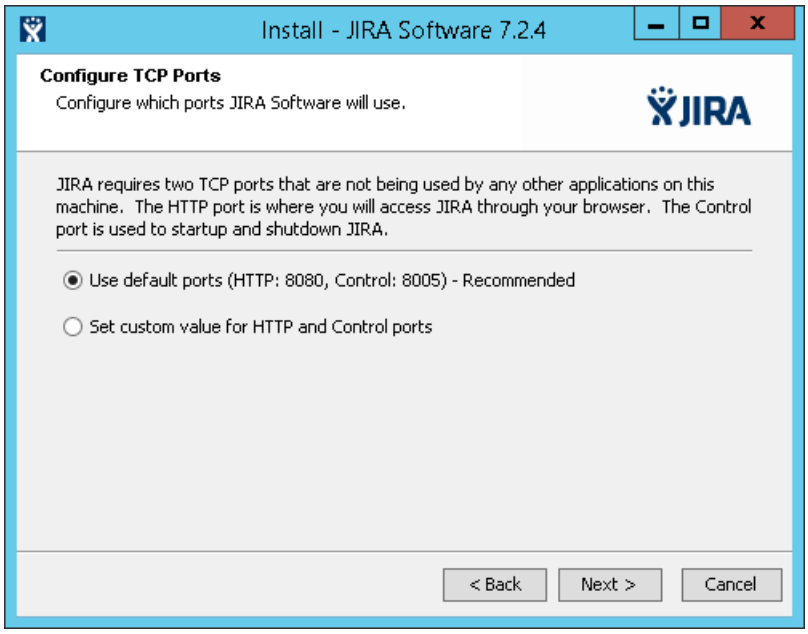
4. Se ejecuta el asistente para la instalación de JIRA y muestra una pantalla de bienvenida. Se debe hacer clic sobre el botón “Next”:



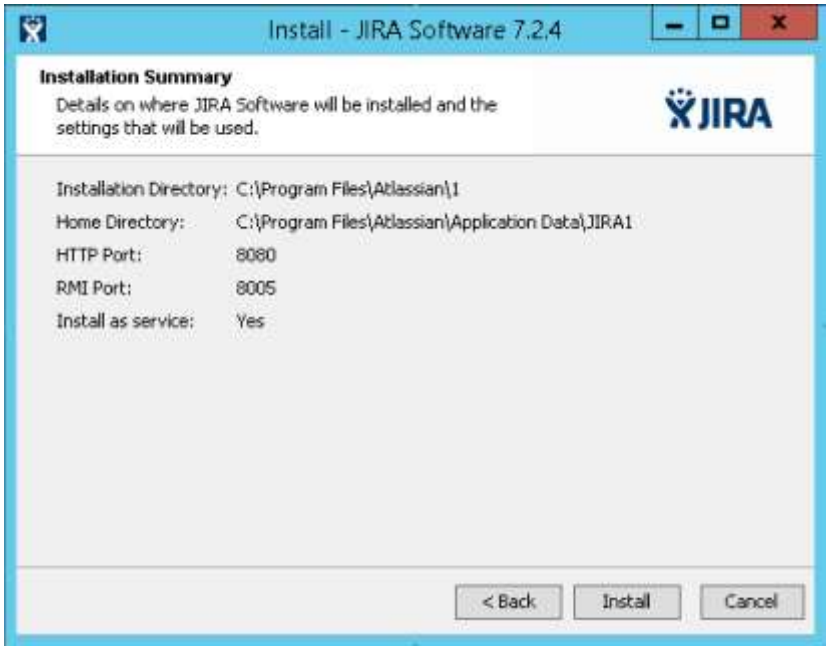
5. Seguidamente, se debe seleccionar el tipo de instalación que se desea realizar. En nuestro caso seleccionaremos la versión “Express”.



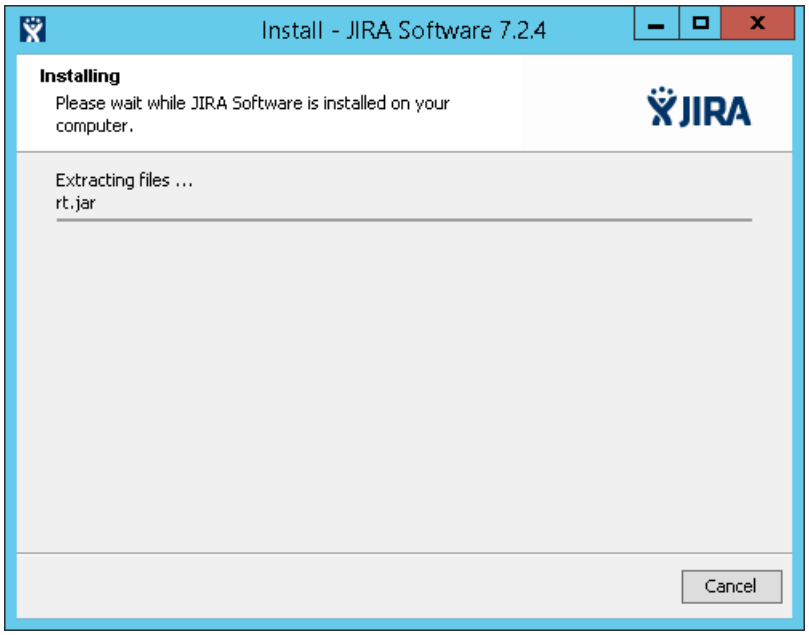
6. Para completar el proceso de instalación de JIRA se requiere tener disponibles dos puertos, por omisión se utilizan los puertos 8080 y 8005; pero en caso de que dichos puertos se encuentren ocupados con otras aplicaciones (por ejemplo, Skype), es necesario seleccionar otros puertos. Es nuestro caso, como este es un servidor Windows Server 2012 R2 dedicado a nuestra instancia, dejaremos los puertos por default.



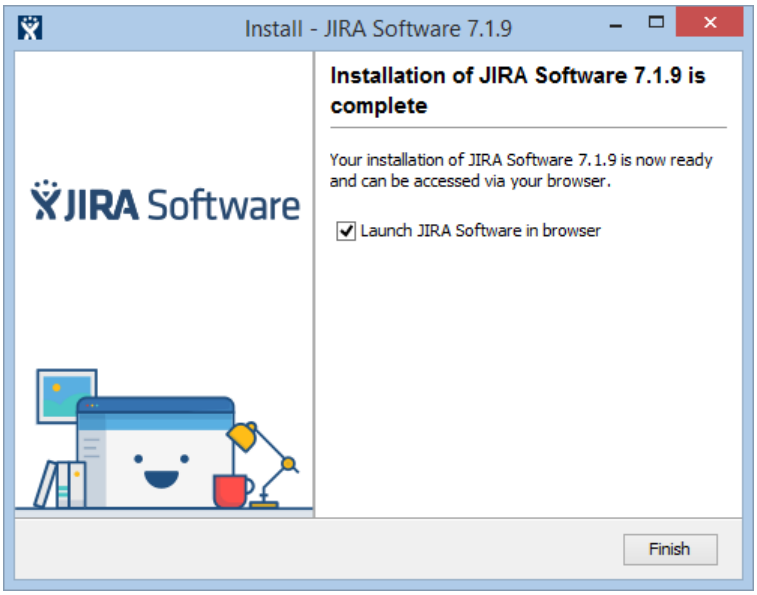
7. Luego el asistente nos mostrará un resumen de la configuración que hemos indicado para la instalación, de forma tal que se pueda corregir cualquier error cometido. En caso de que la información sea correcta, procedemos clicar el botón “Install”, tal como se muestra a continuación.



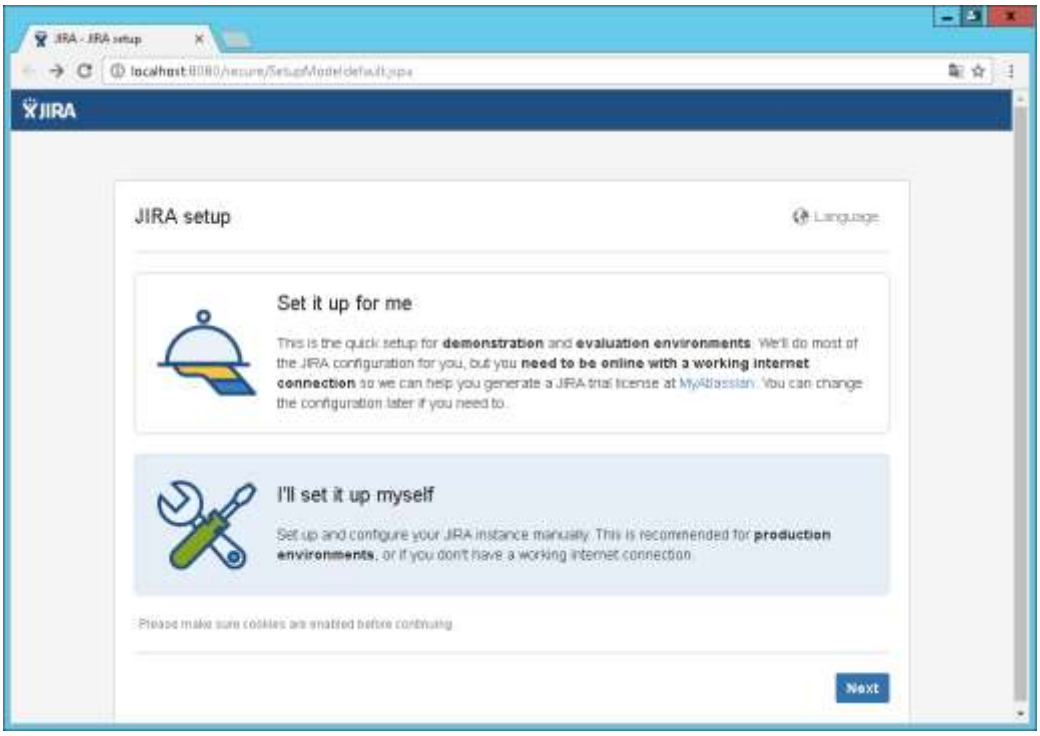
8. Se presenta una barra de estado con el avance de la instalación, tal como se muestra a continuación.



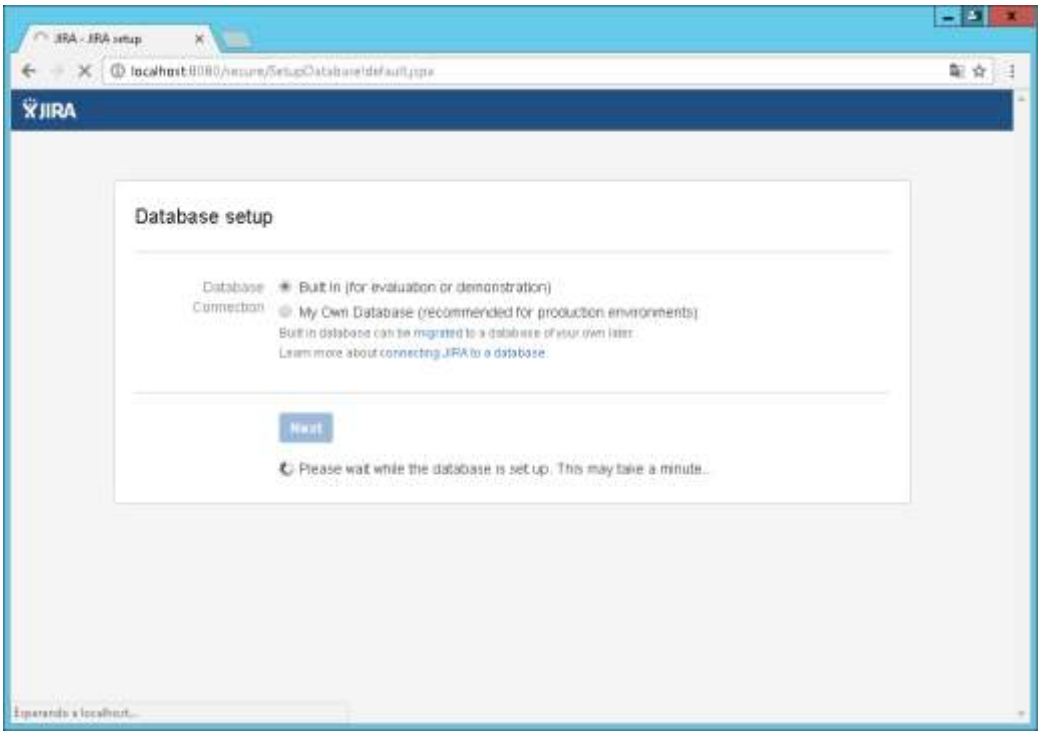
9. Una vez que se terminan de copiar los archivos en la terminal, se debe clicar en el botón “Finish” en la pantalla, tal como se muestra a continuación.



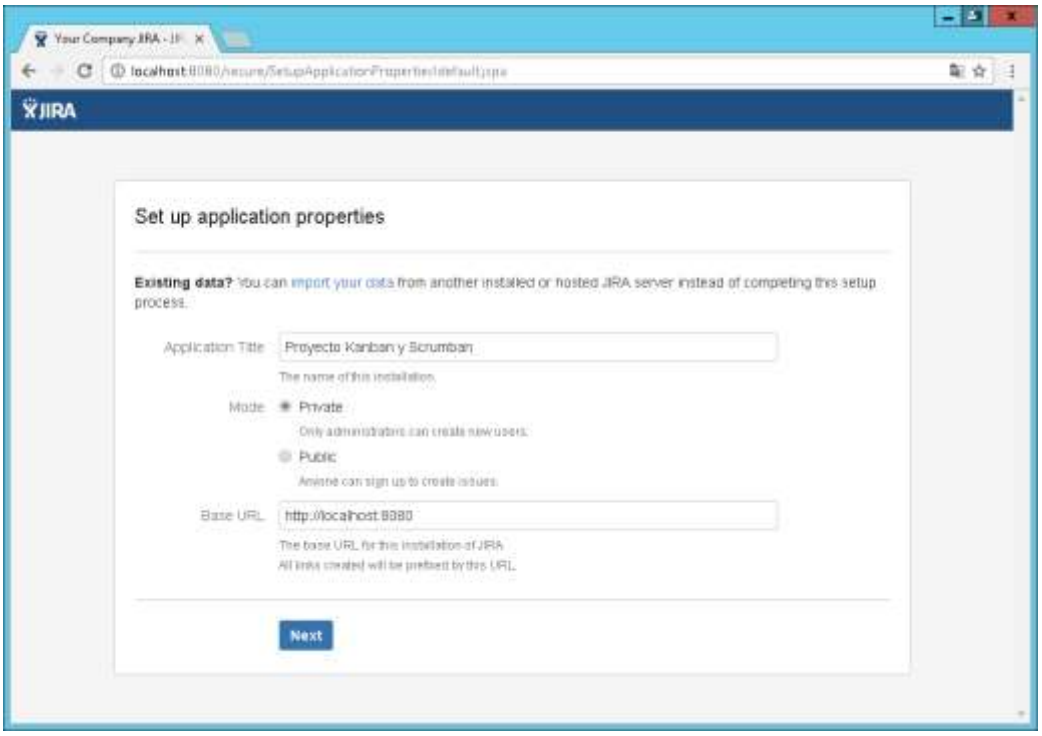
10. Automáticamente se abrirá nuestro navegador mostrando la página inicial de configuración de nuestro JIRA. En el caso de que el navegador no se inicie automáticamente o que la siguiente página no aparezca, ingresar manualmente a http://localhost:8080/.



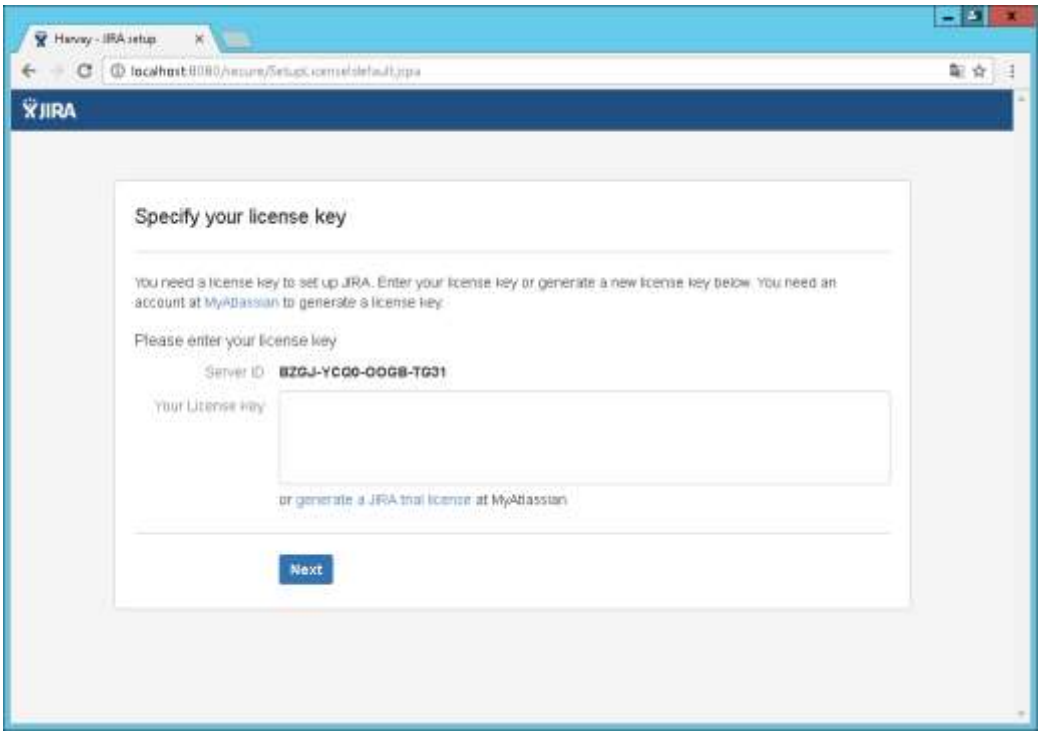
11. En este punto, tenemos 2 opciones para configurar nuestro JIRA. En el caso de seleccionar “Set it up for me”, JIRA realizará el proceso automáticamente guiando al usuario por simples pasos. En caso de que seleccionemos la opción de configuración manual (“I’ll set it up myself”), debemos seleccionar la base de datos que se utilizará. En nuestro caso, seleccionamos la conexión con la base de datos que se encuentra construida dentro de la misma instancia de JIRA y cliqueamos en “Next”. Este paso puede demorar algunos minutos, ya que se estará configurando todas las variables que serán utilizadas posteriormente en la plataforma web.



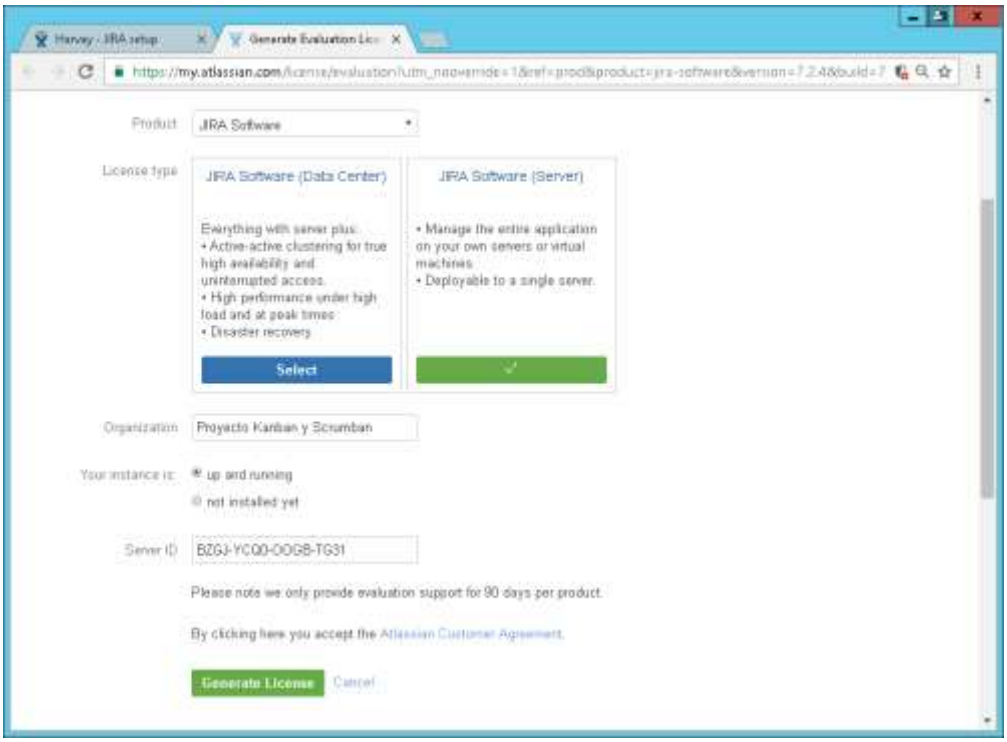
12. . Luego de que la instancia se ha conectado a la base de datos, deberemos indicar el título o nombre de la aplicación JIRA. En nuestro caso, nuestro nombre será “Proyecto Kanban y Scrumban”. Las 2 siguientes opciones las dejamos en su valor predeterminado. Las mismas se refieren al modo de acceso (nosotros mismos autorizaremos el acceso a los usuarios) y la URL base de la instancia.



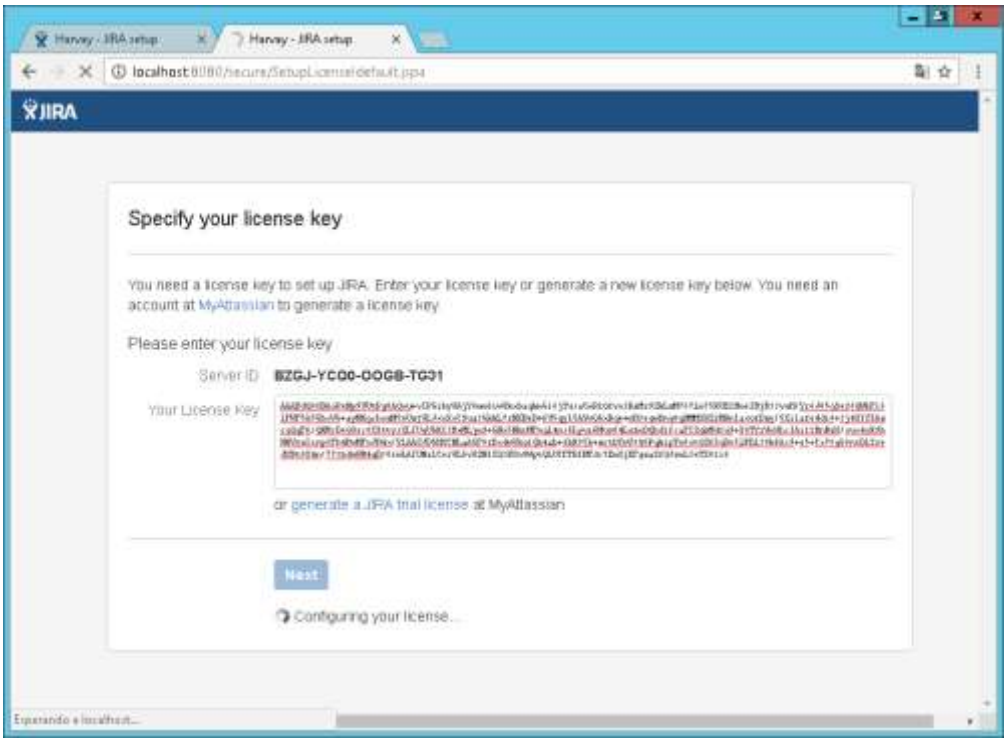
13. Por último, se debe generar una llave de licencia para registrar la instalación de JIRA. Para ello, debemos clicar en el link “generate a JIRA trial license”, que se encuentra debajo del cuadro de texto grande (campo “You Licencse key”):



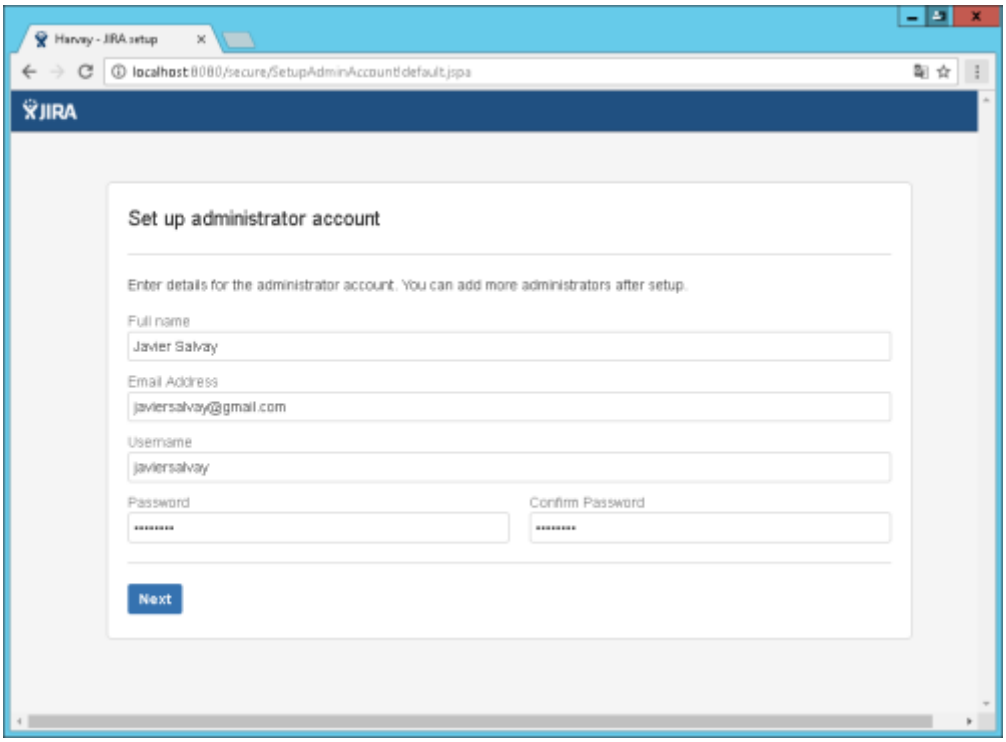
14. Una vez que cliqueamos en el link, se abre una nueva pestaña mostrando un formulario para generar una nueva licencia. Este formulario lo completaremos con nuestros datos de la siguiente manera y cliquearemos en “Generate License”:



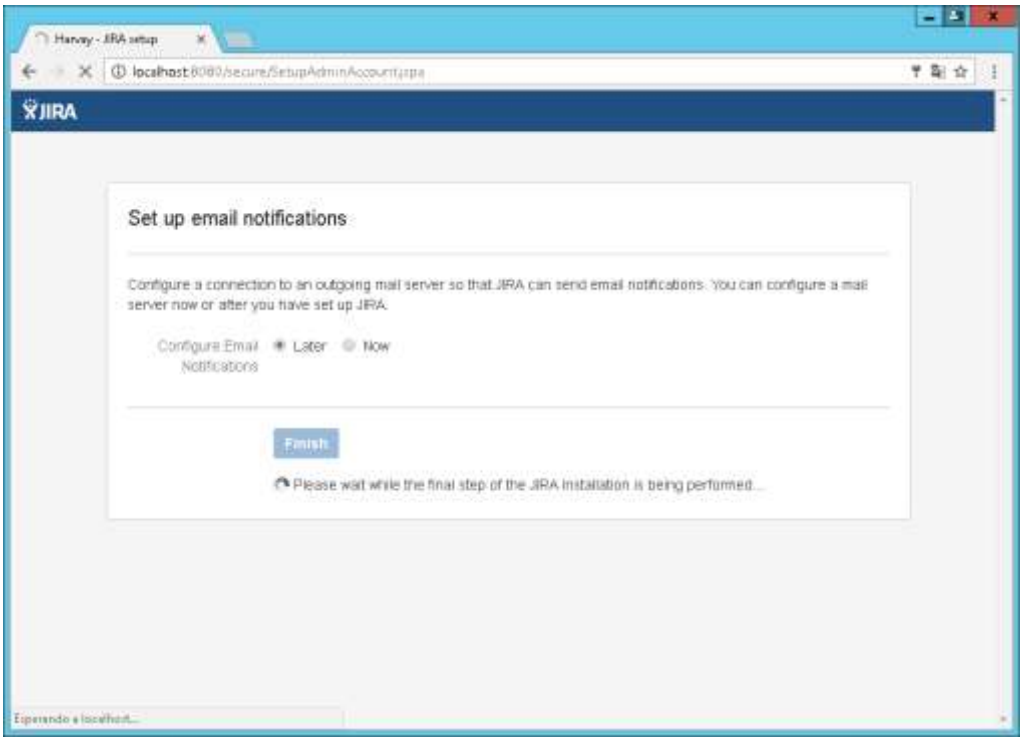
15. Luego de esperar unos momentos, en la misma pestaña se abrirá la misma página para insertar la licencia, pero esta vez, con la licencia pegada dentro del campo “Your License key:”. Cliquear en el botón “Next” y esperar que se procese el pedido.



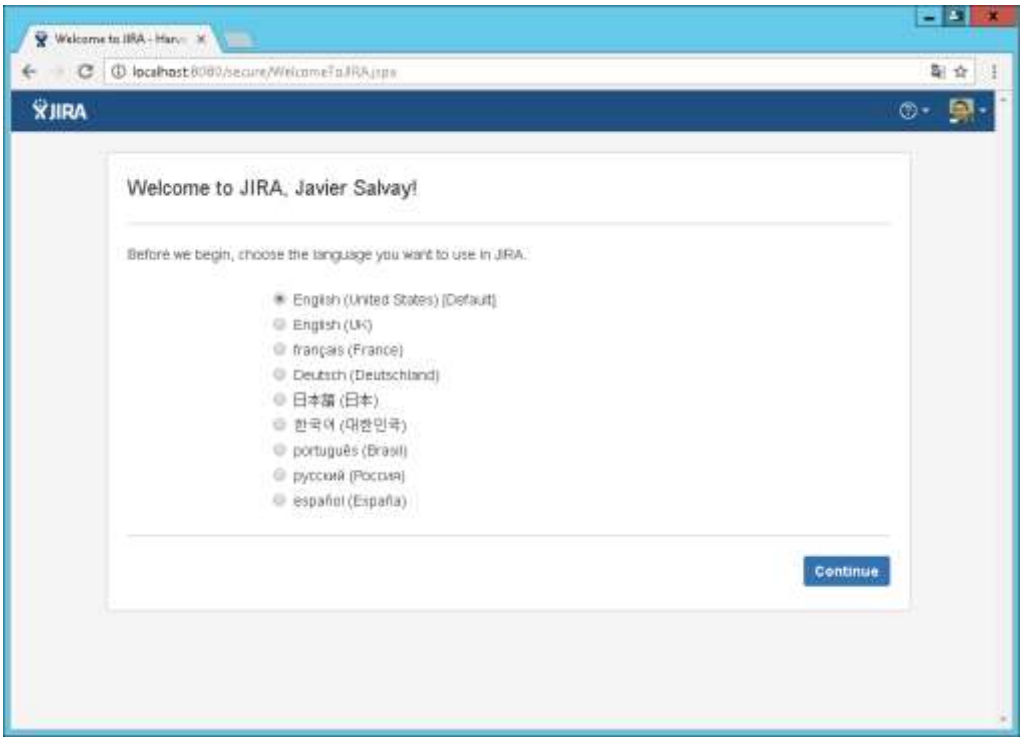
16. Una vez el pedido ha sido procesado, se continua con las configuraciones de la cuenta administradora. Completar el formulario con los datos del administrador y cliquear en el botón “Next”:



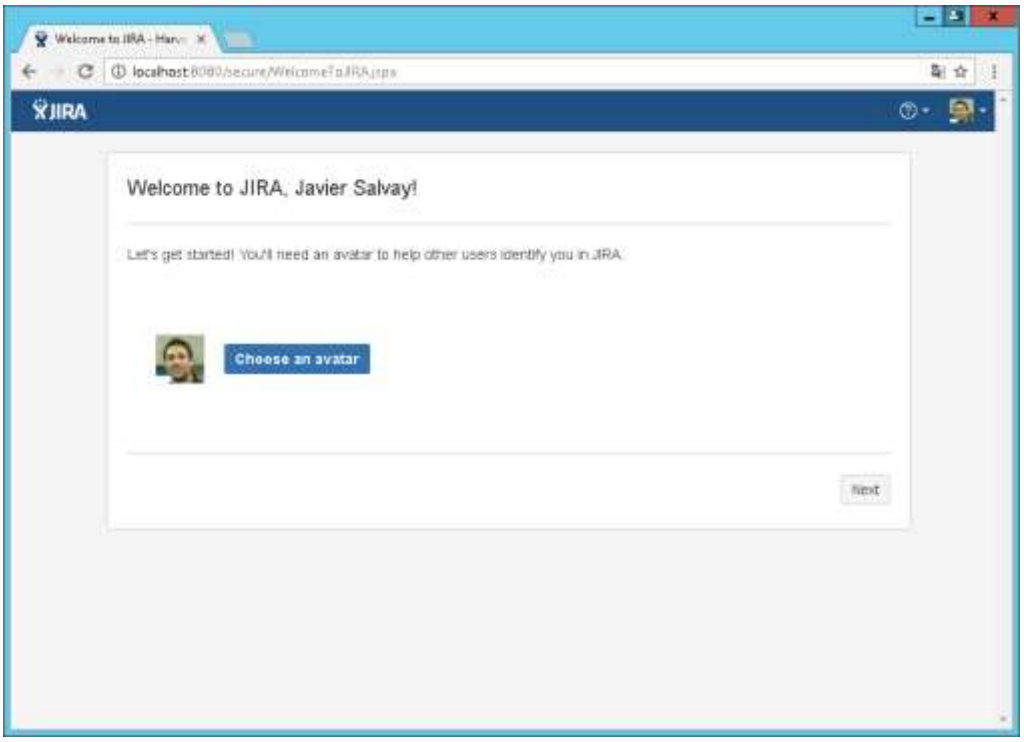
17. En este paso, se pedirá configurar el sistema de emails para enviar notificaciones. En nuestro caso, no lo utilizaremos inicialmente, por lo que dejaremos seleccionado “Later” y cliquearemos en el botón “Finish”.



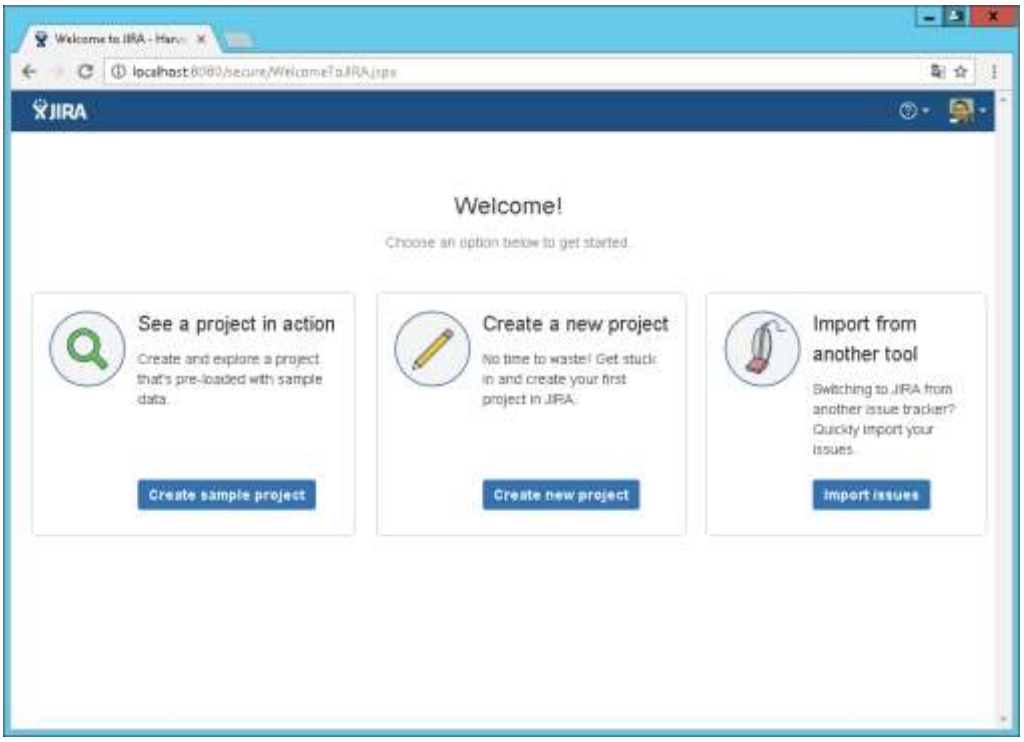
18. Una vez que la instancia ha sido instalada, JIRA nos dará la bienvenida y aparecerá la siguiente pantalla en la misma pestaña. Dejaremos seteado el idioma inglés y cliquearemos en el botón “Continue”:



19. Como último paso, nos pedirá nuestro avatar (imagen para nuestro usuario). Seleccionamos una imagen y cliqueamos en el botón “Next”:



20. JIRA mostrará que está listo para ser utilizado cuando muestre esta pantalla o similar:



Finalmente, se podrá realizar la creación de un nuevo proyecto, para su utilización.

ANEXO B. INSTALACIÓN DE HERRAMIENTA PARA SOPORTE A DOCUMENTACIÓN

Una de las herramientas para soporte de documentación compatible con metodologías ágiles es Dokuwiki, esta herramienta permite crear documentación de cualquier tipo dentro de los grupos de desarrollo. Esta información se guarda en archivos planos, dentro de un servidor local. A continuación, se detallan los pasos que deben seguirse para la instalación.

Requerimientos

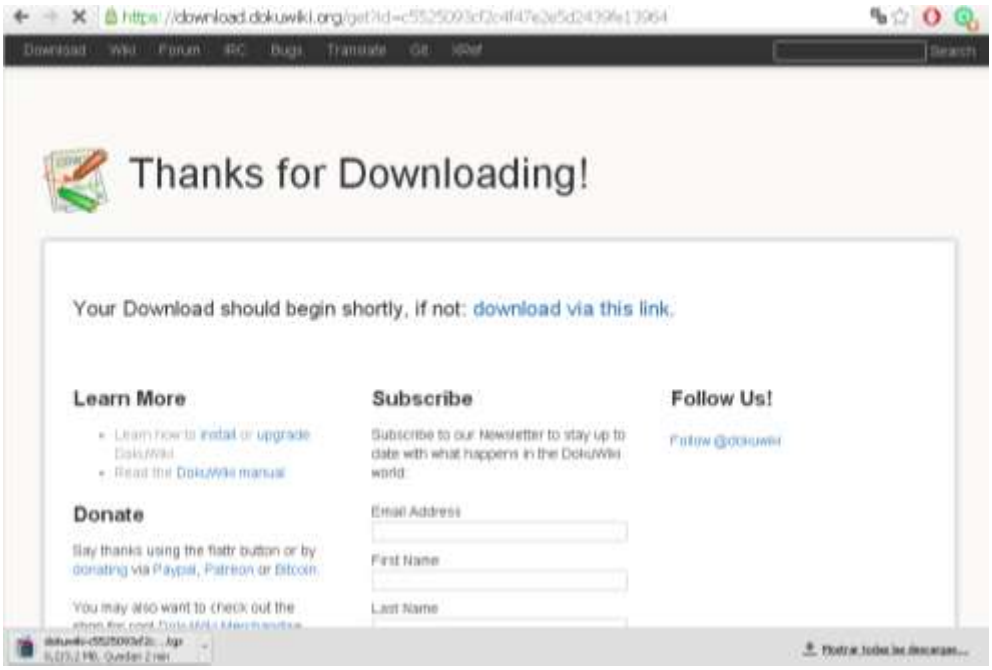
La presente guía tiene en cuenta de que es necesario:

- Sistemas operativos: Windows, Ubuntu, Solaris, Arch Linux, CentOS, Debian, Fedora, FreeBSD, Gentoo, MacOSX, OpenSolaris, openSUSE, QNAP NAS.
- Contenedor de aplicaciones PHP (webserver): se dispone instalado y funcionando un contenedor de aplicaciones PHP para alojar y publicar dicha “Doku Wiki” en una intranet, VPN, red semi-publica o internet. Se recomienda Apache, pero también soporta IIS, litespeed, lighttpd, nginx y Abyss.
- Versión de PHP: versión 5.6 o superior.
- Base de Datos: si bien, esta aplicación no necesita base de datos (ya que utiliza textos planos como base), es necesario al menos 5 gigabytes de almacenamiento disponible en el disco rígido del servidor que hospedará dicha aplicación.
- Navegador de Internet: soporta cualquier versión actual de Firefox, Chrome, Safari, Microsoft Edge u Opera y también, Internet Explorer 8 o superior.

En la presente guía, a modo de mostrar paso a paso su instalación, se utilizará un equipo de escritorio con Windows 8.1 y se instalar la aplicación WAMP, que ya dispone de todos los paquetes necesarios y pre-configurados para que Doku Wiki funcione. Estos mismos pasos pueden ser realizados en otros sistemas operativos, de acuerdo con su uso y requisitos.

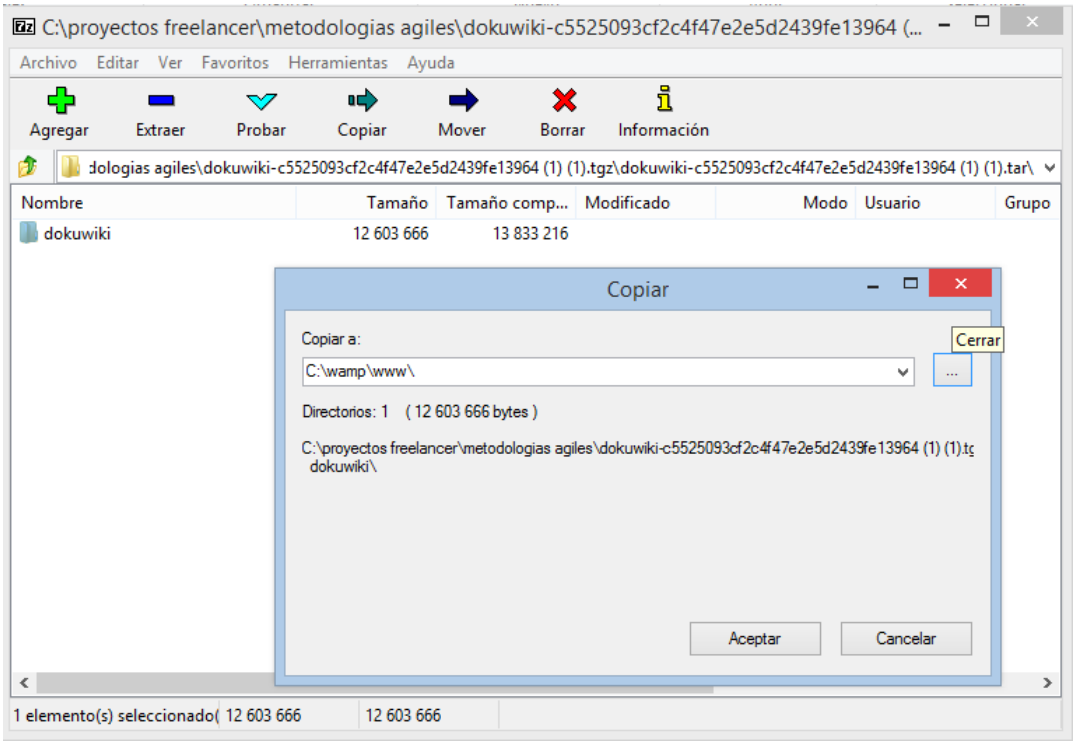
1. Para instalar una versión gratuita de Dokuwiki, se debe ingresar a la siguiente dirección: https://download.dokuwiki.org. Aparecerá entonces una pantalla similar a la siguiente:

2. Es necesario descargar el archivo de instalación de la herramienta. Al cliquea en “Download”, se mostrará la siguiente pantalla:



3. Podemos abrir el archivo descargado para proceder a iniciar la instalación, como se muestra a continuación.

4. Se descomprime o extrae el archivo comprimido en una carpeta nueva (notase que el archivo descargado tiene un nombre bastante largo). Esta es la carpeta que dispone de todos los archivos necesarios para que la wiki sea instale y se active para su uso. Una vez descomprimido el archivo, se selecciona la carpeta "dokuwiki", dentro de la raíz principal del directorio que utilizamos para nuestro servidor local: /www (comúnmente utilizado por contenedores de aplicaciones). En el presente caso, su directorio e ubica en: C:\wamp\www, tal como se muestra a continuación.



5. Es necesario que el servidor WAMP se encuentre activo antes de proceder. Se abre el navegador (Chrome, en este caso ejemplo), y se ingresa a la URL: http://localhost/dokuwiki/install.php y se configuran los datos del servidor a donde se realiza dicha instalación. Los siguientes datos deben indicarse: (a) nombre del wiki, (b) usuario y contraseña, (c) habilitamos ACL (Control de acceso para

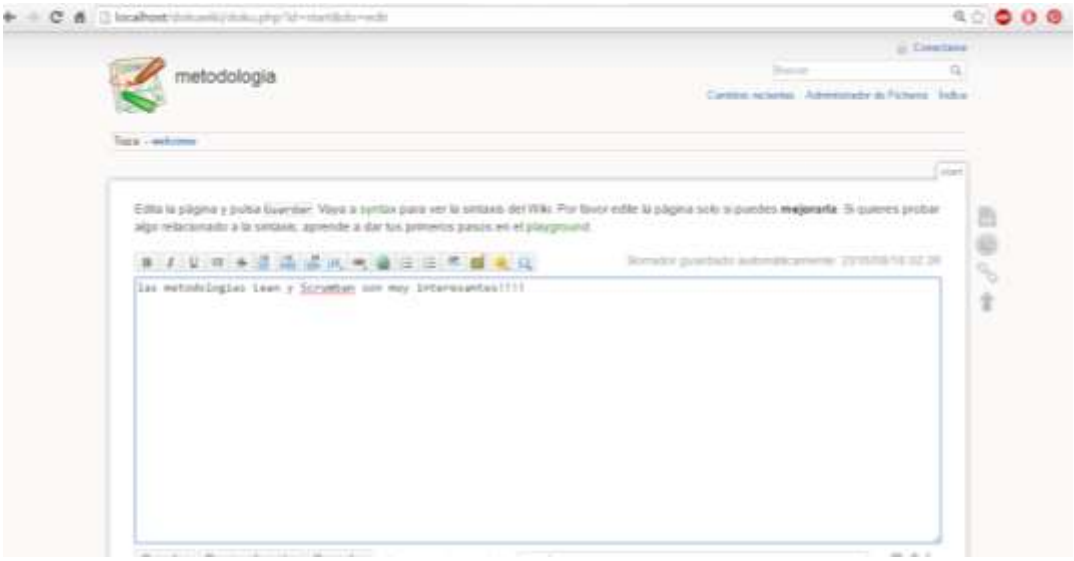
usuarios), y (d) se selecciona el grado de interactividad que tendrá wiki. Luego, se guardan los cambios (botón “Guardar), tal como se muestra a continuación.

6. Luego de realizar este conjunto de pasos, ya se dispone de la aplicación lista para ser utilizada y se puede proceder a crear la página de inicio. Para ello debe ingresar al link: “Visite su nuevo Dokuwiki”:

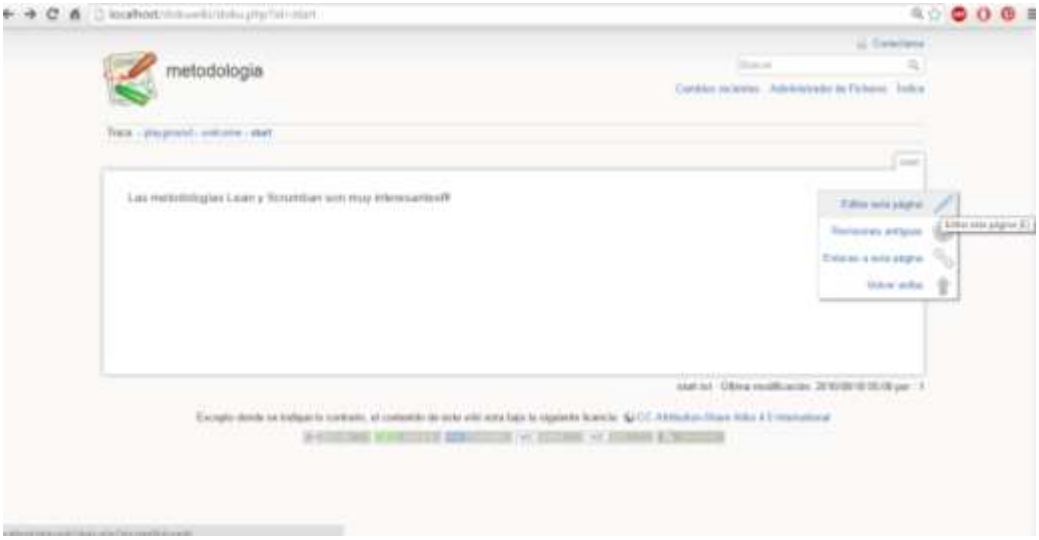
7. Se presenta un mensaje de bienvenida y un conjunto de opciones para configurar la nueva Dokuwiki. De forma de crear la página de inicio, se cliquea en el link: “Start”, en la opción “Create your first pages”, tal como se muestra a continuación:

8. Al seleccionar la creación de la página de inicio, se presenta un mensaje indicando que se debe usar el botón “Crea esta página”, que está ubicado a la derecha de la pantalla, tal como se muestra a continuación.

9. Seguidamente es necesario editar la página de inicio, para presentar la información que se desea, tal como se muestra a continuación:



10. Por último, se deben guardar los cambios realizados en la página de inicio, mediante el botón “Guardar” posicionado en la parte inferior izquierda de la pantalla. Cabe destacar que podemos modificar dicha página en cualquier momento. Se muestra en la siguiente imagen, como se hace visible el botón de “Editar esta página” en la parte derecha de la pantalla.



11. De la misma forma que se procede en esta guía, puede hacerse con un servidor productivo que se encuentre publicado en alguna dirección URL o en alguna IP que pueda ser accedida por el equipo de trabajo.

DATOS PERSONALES DEL ALUMNO

Nombres: **Javier Esteban**

Apellido: **Salvay**

E-mail Académico: **jsalvay046@alumnos.iua.edu.ar**

E-mail Personal: **javier.salvay@gmail.com**

Teléfono: **+55 (98) 9210 1758**

Dirección: **Rua Grajaú 4, Condominio Studio 1, Departamento 1006, São Luís, Maranhão, Brasil**

Firma: