SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Subsecretaría de Educación Superior
Tecnológico Nacional de México
Instituto Tecnológico de Orizaba

"Año del Centenario de la Promulgación de la Constitución Política de los Estados Unidos Mexicanos"

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

OPCIÓN I.- TESIS

TRABAJO PROFESIONAL

"Adaptación de la disciplina del Proceso Personal de Software (PSP) con el proceso ágil Scrum en el área de desarrollo de software de la Empresa TecnoMotum S. A. de C. V.".

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA:

I.S.C. Mauricio Leonardo Urbina Delgadillo

DIRECTOR DE TESIS:

M.C. María Antonieta Abud Figueroa

CODIRECTOR DE TESIS:

M.C. Silvestre Gustavo Sergio Peláez Camarena

ORIZABA, VERACRUZ, MÉXICO. MARZO 2017

Agradecimientos

A Dios por permitirme vivir esta vida regalándome la oportunidad de ser mejor día a día.

A mis padres por apoyarme en todo, por ser un gran ejemplo a seguir y por todos sus sacrificios.

A mis grandes amigos Moisés y Karen por darme siempre el ánimo para seguir adelante y nunca darme por vencido.

A mis amigos por todos aquellos momentos divertidos que pasamos juntos y porque a pesar de la distancia los siento cerca de mí

A mi directora de tesis Ma. Antonieta Abud Figueroa por ser mi guía durante el desarrollo de este proyecto.

A todos los maestros que creyeron en mí y compartieron sus conocimientos para hacer de mi aprendizaje más valioso.

Al Instituto Tecnológico de Orizaba-ITO junto con la División de Estudios de Posgrado e Investigación-DEPI por ofrecerme las instalaciones necesarias para realizar este trabajo de tesis.

Al Consejo Nacional de Ciencia y Tecnología, CONACyT, por el apoyo económico otorgado para la realización de mis estudios de maestría durante el periodo Enero 2015 – Enero 2017.

Contenido

Lista de Figuras

Lista de Tablas

Resumen

En la actualidad, la calidad del software es un factor de suma importancia para el cliente pues se sabe que un producto de calidad fortalece una relación de confianza entre empresa y cliente. Esta perspectiva del mundo empresarial genera competitividad entre las compañías por acaparar el mercado y atraer nuevos clientes a sus filas. Sin embargo, muchas de las empresas no logran satisfacer a sus clientes después de invertir esfuerzo, tiempo y dinero.

La combinación de métodos ágiles y modelos de disciplina establece una posible solución para crear un producto de calidad mediante la institucionalización de procesos en el desarrollo de software. Por esta razón en este trabajo se presenta una propuesta de combinación entre dos enfoques para el desarrollo de software, *Scrum* y *PSP* apoyados por la norma mexicana *MoProSoft* con el objetivo de mejorar la calidad del producto final estableciendo un proceso sencillo y bien definido para equipos de trabajo pequeños, especialmente para PyME.

Entre los principales beneficios que ofrece esta combinación se mencionan: mejora el rendimiento del ingeniero de software al establecer un proceso para el desarrollo de código, mejora la planificación de las actividades introduciendo una estimación basada en registros históricos, establece una fase para la gestión de riesgos para *Scrum*, apoya a la administración del proceso de desarrollo con pautas para elaborar la documentación correspondiente al proyecto, introduce técnicas para la detección y corrección de defecto reduciendo el tiempo de la fase de pruebas.

Abstract

At present, the quality of software has become a factor of importance for the customer, it is known that a quality product strength a relationship of trust between company and customer. This perspective of the business world generates competitiveness among companies by monopolizing the market and attracting new customers to their ranks. However, many companies fail to satisfy their customers after spend effort, time and money.

The combination of agile methods and discipline models establishes a possible solution to create a quality product through the institutionalization of processes in software development. For this reason, this paper presents a proposal for a combination of two approaches to software development, *Scrum* and *PSP* supported by the Mexican standard *MoProSof*t with the objective of improving the quality of the final product by introducing a simple process, defined for small work teams, especially for PyME's.

The main benefits offered by this combination are: improves the software engineer performance introducing a process for code development, improves the activities of the planning phase by introducing an estimate based on historical records, set up a phase for risk management for *Scrum*, supports the management of the development process with guidelines for make the documentation for the project, introduce techniques for defects detection and correction reducing the time of the test phase.

Introducción

En la actualidad, el desarrollo de software a un nivel empresarial se caracteriza por ser rápido, cambiante y complejo, debido a los sistemas que utilizan las grandes compañías, estos requieren una gran labor para su administración, desarrollo y mantenimiento. Los equipos de trabajo se enfrentan a calendarios de trabajo agresivos para crear un producto que satisfaga las necesidades del cliente. Por lo mismo, la demanda de la calidad del software es importante como factor imprescindible para construir una relación de confianza entre la empresa y el cliente.

Según la presentación realizada por Nielsen [1] el 70% de las fallas en el desarrollo de software se introducen en las fases de requerimientos, diseño del sistema, arquitectura y diseño de componentes; el 30% corresponde a las fases de código, pruebas e integración. Sin embargo, el 19.5% de estos defectos se encuentran en las fases de requerimientos, diseño del sistema, arquitectura, diseño de componentes, código y pruebas; aun con las mejoras que se realizan a los procesos de desarrollo el 50.5% de los defectos se encuentran en la fase de integración y un 21% en la fase de operación, es decir, por parte del usuario.

La combinación de métodos ágiles y modelos disciplinados es una solución cada vez más aceptada por los seguidores de ambos enfoques para contrarrestar las deficiencias que trae consigo el utilizar en solo método, proporcionado mejoras al desarrollar un producto de calidad y mejorar la gestión del proceso. Para esta investigación se tomaron en cuenta, *Scrum* como método de desarrollo ágil donde su filosofía impulsa y resalta el trabajo en equipo, el aprendizaje constante y una estructura de desarrollo dinámico; y *PSP* como modelo disciplinado enfocada al desarrollo de software personal que ayuda a mejorar la productividad a través de la medición y el análisis de datos de detallados sobre el proceso de producción de software.

Como resultado se tiene un modelo de combinación entre *Scrum* y *PSP* que se adapta al proceso de desarrollo a nivel empresarial. Gracias a la empresa TecnoMotum S.A. de C.V., que se dedica al desarrollo de soluciones basadas en telemetría, en su interés por cumplir con los más altos estándares y certificaciones de calidad en su proceso productivo, ayudó en esta propuesta proporcionando

información necesaria para genera un modelo sólido que ayude a su equipo de trabajo a mejorar la calidad del desarrollo y mejorar la gestión del proceso.

Este documento está organizado de la siguiente manera: en el Capítulo 1 se muestran los antecedentes abordando la calidad desde una perspectiva histórica y el marco teórico donde se define los conceptos básicos para esta investigación, además se definen el planteamiento del problema, así como los objetivos generales y específicos. En el Capítulo 2 se muestra el estado del arte utilizado y una tabla donde se resumen cada trabajo de referencia, por último, se tiene la propuesta de solución para el problema planteado. El desarrollo de esta investigación se encuentra en el Capítulo 3 donde se define una propuesta de un modelo que combina dos enfoques del desarrollo de software. Los resultados del caso de estudio de muestra en el Capítulo 4. Finalmente, en el Capítulo 5 se muestran las conclusiones y recomendación para trabajos futuros.

Capítulo 1 Antecedentes

En este capítulo, se da a conocer una perspectiva rápida de la ingeniería del software abordando el pasaje de la "Crisis del Software" y cómo ésta evolucionó hasta la era actual. También, se muestran y describen los conceptos básicos que el lector necesita para familiarizarse con el contexto del proyecto. Además, se presenta el planteamiento del problema que dio origen a este proyecto, el objetivo general, así como los objetivos específicos que se realizaron para llegar solución del problema, y la justificación del mismo.

1.1. La crisis del software

Las máquinas de cómputo más antiguas tenían los "programas" fijos y obligaban al operador a cambiar su diseño físico para alterar el "*programa*". Estas máquinas no eran "programadas" sino "*diseñadas*", todo era un proceso manual a partir de diagramas de flujo, notas de papel, cambiar cables. En 1944 la *Electronic Numerical Integrator and Computer* (ENIAC) fue puesta en operación, debido a sus capacidades limitadas y poca atención solo se prestaba al desarrollo de "*programas*" [2]. Con el tiempo, el hardware se hizo más barato, más fiable y más complejo, los "*programas*" que se ejecutaban eran más elaborados; se comenzaba a hablar sobre un nuevo conjunto de problemas, los problemas de *software* contra los problemas de *hardware*.

Durante la década de 1960, el uso del término software se vuelve más común y surge una división entre "*software*" y "*programa*" por primera vez con la importancia de la computación aplicada al procesamiento de datos empresariales [3]. Empezaron a ser habituales las computadoras y surgieron los primeros "*héroes de la programación*" contando con solo los manuales del sistema operativo y el lenguaje de programación ensamblador para desarrollar las primeras aplicaciones. Para finales del año 1960, el costo del hardware cayó exponencialmente, y continuó de esa manera, mientras que el software surgió de manera similar.

Margaret Hamilton adoptó el término "*Ingeniería de Software*", acuñado por Anthony Oettinger, estando con su equipo de desarrollo del Apolo 11 y, en 1968 se utilizó como título para la primera conferencia mundial de Ingeniería de Software, patrocinada por la OTAN [3]. En dicha conferencia se mencionó por primera vez el término "*crisis del software*", se definieron los problemas que

surgían en el desarrollo de sistemas de software indicando que las causas de la crisis estaban vinculadas con la complejidad del hardware y el proceso de desarrollo de software. El resultado fue informe que definió las normas para desarrollar software, es decir, los fundamentos de la Ingeniería de Software [4].

Recordando a Edsger Dijkstra, uno de los pioneros de la programación, dijo: "*La programación comenzó con un arte que se practicaba de forma intuitiva. En 1968, se comenzó a reconocer que los métodos de elaboración de programas hasta ahora era inadecuados…*", "*La principal causa de la crisis de software es que ¡las máquinas recibían órdenes de una magnitud más potente!*" es decir, "*Mientras no había máquinas, la programación no era un problema en lo absoluto, cuando tuvimos un par de ordenadores débiles, la programación se había convertido en un problema leve, y ahora que tenemos computadoras gigantescas, la programación se ha convertido en un problema de igual magnitud*" [5]. El comentario de Dijkstra no estaría tan lejos de la realidad durante los próximos años.

Hacia las décadas de los 70's y 80's las principales características de los proyectos de software eran: presupuestos rebasados, tiempo estimado de desarrollo fuera del plazo, pobre calidad de software, los requerimientos no se conocían y eran difíciles de administrar [4]. Algunos causaron daños a la propiedad y pocos causaron pérdidas de vidas, un famoso caso de este último es el incidente del ***Therac-25***. La crisis originalmente fue definida en términos de ***productividad***, pero evolucionaría para hacer un énfasis sobre la ***calidad***; inclusive varias personas lo utilizaron para referirse a su incapacidad para contratar programadores calificados.

El hallar una solución a la crisis de software desencadenó que las compañías e investigadores produjeran muchas herramientas de software. Cada nueva tecnología o práctica que apareció entre 1970 y 1990 fue trata como una ***bala de plata*** (del inglés, *silver bullet*) que sería la solución definitiva a la crisis que se tenía en la construcción de programas y sistemas. En 1986, Fred Brooks publicó un artículo titulado "*No Silver Bullet"* argumentado que ninguna tecnología, práctica o técnica de gestión que por sí sola prometiera una mejora en cuanto a la productividad, la fiabilidad y la simplicidad dentro del desarrollo de software [6], este artículo generó un debate continuo entre

defensores de lenguajes de programación, de procesos de desarrollo, empresarios e investigadores durante los siguientes años.

Así, con el paso del tiempo, casi todo el mundo aceptó el hecho de que nunca se encontraría una solución única a la temible crisis del software, por lo tanto, era necesario ser conscientes con el término ***bala de plata***. Este hito se tomó como una prueba de que la *Ingeniería de Software* estaba madurando y que los proyectos debían su éxito al trabajo duro, esfuerzo y compromiso por parte de todas las personas involucradas en el mismo.

En los albores de los años 90's, el surgimiento del Internet incrementó rápidamente la demanda internacional de sistemas que permitieran enviar, recibir y mostrar cualquier tipo de información (texto, colores, imágenes, fotos, animaciones) de maneras nunca antes vista y con tecnologías muy poco conocidas que debían trabajar sobre la gran *World Wide Web*.

Hacia el año 2000, la demanda del software era demasiada por empresas pequeñas, la necesidad de un software de bajo costo permitió el crecimiento de métodos rápidos y simples que ayudaran a desarrollar software ejecutable, desde los requerimientos hasta su despliegue, más rápido y más fácil. El uso de prototipos rápidos dio como resultados las metodologías ágiles, que simplificarían muchas áreas de la Ingeniería de Software, como la recolección de requisitos y pruebas de fiabilidad, principalmente para software pequeños que necesitaban un enfoque más simple y rápido para la gestión de su desarrollo y mantenimiento [4].

Desde el año 2010 hasta la actualidad, la Ingeniería de Software evolucionó de una manera casi inimaginable, recordando lo dicho por Dijkstra sobre la principal causa de la crisis: **"*la complejidad del software*"**. El desarrollo de software en el 2016 se agrupa en las siguientes categorías [4]:

- **Marcos de aplicación**: El desarrollo de software abrió camino en las siguientes áreas de aplicación: Web/Internet, Móvil, Redes, Cómputo en la nube, *Big Data*, *End-User Computing*.
- **Herramientas:** Las herramientas de trabajo que se utilizan para el desarrollo de software contemplan: Marcos de Trabajo, Servicios, Conjunto de Soluciones (*Software Stacks*),

Patrones, Reutilización, IDE's, Construcción automatizado y Pruebas automatizadas, Análisis estático, Ingeniería basada en modelos.

- **Métodos:** Series de procesos ya probados y comprobados para una mejor gestión del desarrollo de software, se contemplan los siguientes: Ágil, Lean, XP, DevOps, SAFe, *TSP*, CMMI, RUP, *PSP* y *Scrum* como las mejores prácticas.
- **Riesgos:** Gracias la globalización del software se contemplan los siguientes: *Hackers*, Vulnerabilidades, Protección de los datos, Seguridad en la integridad de los datos y Complejidad.

1.2. Marco teórico

1.2.1. Calidad del producto de Software

El software es el resultado de aplicar la Ingeniería de Software utilizando una metodología, pero no solo basta con seguir un flujo de instrucciones, es necesario que el software sea funcional, rápido, eficaz, adaptable a futuro y que cumpla las necesidades del cliente, es decir con calidad.

Para entender mejor lo escrito en el párrafo anterior, se presentan algunas definiciones de calidad:

- La RAE [7] la define como una "propiedad o un conjunto de propiedades esenciales a algo, que permiten juzgar su valor", esta descripción se orienta al mercado de productos o servicios.
- La ISO 9000 [8] señala: "el grado en el que un conjunto de características inherentes cumple con los requisitos*"* dando un énfasis para cumplir las necesidades del cliente.
- La IEEE [9] la define como: "El grado en que un sistema, componente o proceso cumple con las necesidades de cliente o usuario y sus expectativas."
- Pressman [10] afirma que es el resultado de una buena administración y correcta práctica de la Ingeniería de Software, utilizando métodos, técnicas, acciones de control de calidad y aseguramiento de la calidad, para satisfacer las necesidades del cliente o usuario.

De las definiciones anteriores es posible recuperar tres puntos importantes sobre la calidad: 1) la necesidad de cumplir los requisitos del cliente, 2) la reducción de costos y 3) la efectiva

asignación de recursos humanos. Estos puntos son esenciales para construir una definición de ***calidad en el ámbito de desarrollo de software***.

1.2.2. Calidad de proceso de software

En el desarrollo de software la relación entre calidad del proceso y la calidad del producto es muy compleja, ya que es difícil medir las características del software y las de proceso y, a su vez, determinar cómo influyen entre sí. Un proceso de software recopila las mejores prácticas y experiencias de casos de éxito dando importancia a los aspectos técnicos, y también a la gestión del proyecto.

Una definición propuesta por Derniame dice que la **calidad en el proceso de desarrollo de software** es una colección estructurada de buenas prácticas que describen las características de un proceso efectivo, esto implica: 1) la definición de estándares, 2) supervisar el proceso para asegurar que se sigan dichos estándares y 3) realizar informes del proceso para las personas interesadas [11], esto también se le conoce como **gestión de la calidad del proceso de desarrollo**.

1.2.3. Metodología de desarrollo de software

Un ***proceso de desarrollo*** tiene asociado un conjunto de actividades, acciones y tareas que se ejecutan para crear un producto de software [10] eficaz y eficiente que reúna las necesidades del cliente. Cuando se realiza una descripción sencilla de un proceso de desarrollo con un "*flujo*" en donde todas las actividades están estructuradas con respecto a secuencia y tiempo se le conoce como ***modelo de proceso de desarrollo*** [11].

Una ***metodología para desarrollar un software*** consiste en un proceso de software detallado y completo, se basa en la combinación de varios modelos de procesos genéricos definiendo con precisión los artefactos, roles y actividades, junto con las prácticas y técnicas para el equipo de desarrollo, y el uso de herramientas de apoyo [12].

Retomando lo mencionado por Pressman [10] la existencia de una metodología de desarrollo no garantiza que el software se entregue dentro del plazo, cumpla con las necesidades del cliente, o que tenga las características técnicas que, a largo del plazo, se convertirán en características de calidad.

1.2.4. Modelo y Estándar de Calidad

Dentro de las instituciones, empresas u organizaciones es necesario que utilicen modelos y/o estándares que guíen el desarrollo de un software por buen camino hacia la calidad total. En primer lugar, se muestran dos definiciones para ***Estándar de Calidad***:

- La especificación cuantitativa de un criterio (asignación numérica a un criterio de calidad), el estándar es un porcentaje determinado por la institución para considerarlo como un rango que se requiere de alcanzar para considerarlo con calidad [13].
- Aquello que permiten definir un conjunto de criterios que guían la forma en que se aplica la Ingeniería de Software, proporcionan medios para que todos los procesos se lleven a cabo de la misma forma para lograr productividad y calidad [14].

En segundo lugar, se presentan dos definiciones de un ***Modelo de calidad***:

- La Asociación Española para la Calidad [15] lo define como referencias que se utilizan para mejorar la gestión de un proyecto, se caracterizan por contener directrices y no requisitos para los sistemas.
- Según Piattini, García [14] son aquellos documentos que integran la mayor parte de las mejores prácticas, proponen temas de gestión e integran varias prácticas dirigidas a procesos clave y permiten medir los avances de la calidad.

A partir de las definiciones anteriores es posible decir que: ***un modelo y/o estándar de calidad*** son prácticas vinculadas a los procesos de gestión y desarrollo de proyectos. Lo que implica una buena planificación para lograr un impacto estratégico cumpliendo siempre con los objetivos fijados por la calidad del software. Es importante mencionar que ningún modelo o estándar tiene un estilo particular, ni recomienda implantar determinadas prácticas, ni el uso de metodologías para lograr su objetivo, es decir, cada empresa requiere adaptar un modelo y/o estándar a sus prioridades de trabajo y necesidades de desarrollo.

1.2.5. Norma de calidad *MoProSof*t

Es una norma mexicana NMX-I-059-NYCE-2011 (*MoProSof*t) enfocada a las organizaciones dedicadas al desarrollo y mantenimiento del software en la que se establecen los requisitos de conformidad para el método de verificación del modelo de procesos para la industria de software [16].

*MoProSof*t es un modelo de procesos diseñado en México como parte del programa PROSOFT de la Secretaría de Economía, su objetivo es incorporar las mejores prácticas en gestión de la Ingeniería de Software, además su incorporación permite elevar la capacidad de madurez en los procesos de desarrollo en producto y servicios de software con calidad.

En la *Figura 1.1* se muestra el diagrama de los procesos de la norma *MoProSof*t, cada categoría aborda varias tareas que se describen de manera breve a continuación [17]:

- **Alta Dirección**
  - **Gestión de Negoción:** Establece la razón de ser de la organización, sus objetivos y las condiciones para lograrlos, es necesario considerar las necesidades de los clientes, así como evaluar los resultados para estar en la posibilidad de proponer cambios que permitan la mejora continua. Adicionalmente, habilita a la organización para responder a un ambiente de cambio y a sus miembros para trabajar en función de los objetivos establecidos.
- **Gerencia**
  - **Gestión de Procesos:** Establece los procesos de la organización en función de los procesos requeridos identificados en el plan estratégico. Así como definir, planificar, e implantar las actividades de mejora en los mismos.
  - **Gestión de Proyectos:** El propósito es asegurar que los proyectos contribuyan al cumplimiento de los objetivos y estrategias de la organización.
  - **Gestión de Recursos:** El propósito es conseguir y dotar la organización de los recursos humanos, infraestructura, ambiente de trabajo y proveedores, así como crear y mantener la base de conocimiento de la organización.

- **Recursos Humanos y Ambiente de Trabajo:** Proporciona los recursos humanos adecuados para cumplir las responsabilidades asignadas a los roles dentro de la organización, así como la evaluación del ambiente de trabajo.
- **Bienes, Servicios e Infraestructura:** Proporciona proveedores de bienes, servicios, e infraestructura que satisfagan los requisitos de adquisición de los procesos y proyectos.
- **Conocimiento de la Organización:** Mantiene disponible y administra la base de conocimiento que contiene la información de los productos generados por la organización.

❑ **Operación**

❍ **Administración de Proyectos Específicos:** Establece y lleva a cabo un proceso disciplinado las actividades que permitan cumplir con los objetivos de un proyecto en tiempo y costo esperados.

❍ **Desarrollo y Mantenimiento de Software:** El propósito es la realización sistemática de las actividades de análisis, diseño, construcción, integración y pruebas de productos de software nuevo o modificado cumpliendo con los requerimientos especificados.

1.2.6. Marcos de trabajo para desarrollo de software

1.2.6.1. *Scrum*

Es un proceso de gestión de proyectos que toma su nombre y principios de estudios realizados por Hirotaka Takeuchi e Ikujijo Nonaka en los años 80, aunque surgió como modelo para el desarrollo de productos tecnológicos, sus principios eran válidos para trabajar con requisitos inestables, y que necesitan agilidad. Jeff Sutherland aplicó los principios de *Scrum* al desarrollo de software en 1993 en Easel Corporation y no fue hasta 1996 junto con Ken Schwaber que se presentaron como válidos para gestionar el desarrollo de software.

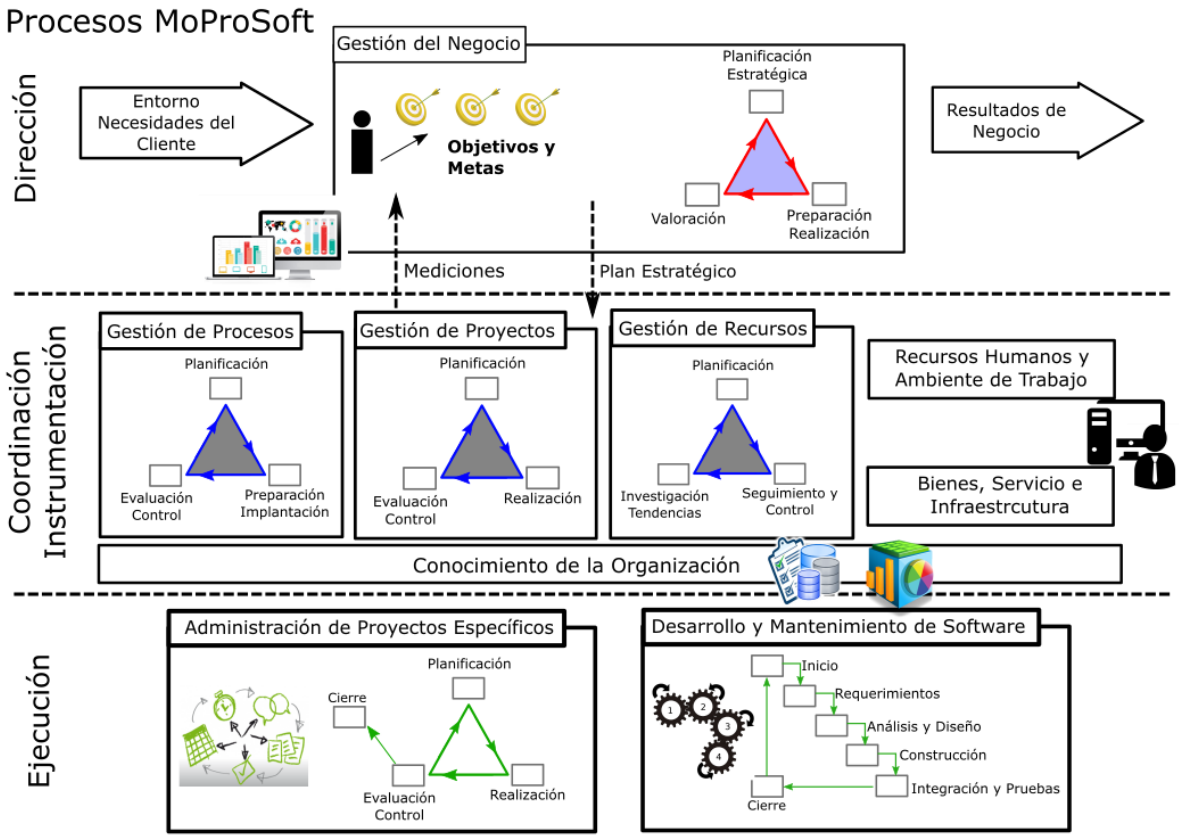


Figura 1.1 Procesos y Categorías de MoProSoft

Se considera una metodología de desarrollo ágil [18] por unos, y por otros como un marco de trabajo [19] [20] formado por un conjunto de prácticas y reglas, su propósito es gestionar el desarrollo de un software, prioriza el trabajo en equipo y su diseño le permite adaptarse a los cambios en los requerimientos. Está más orientado a las personas que a los procesos porque la gestión no se basa en el seguimiento de un plan sino en la adaptación continua a las circunstancias de un proyecto. El desarrollo se inicia desde la visión general del producto, como se muestra en la *Figura 1.2*, dando prioridad a las funcionalidades del negocio, que se llevan a cabo en un periodo breve de tiempo.

El resultado final, el producto del desarrollo de software, se realiza en entregas parciales y regulares llamados *Sprint* donde se van determinando las partes a desarrollar de acuerdo a criterios como la prioridad para el negocio y la cantidad de trabajo que se aborda durante dicha iteración.

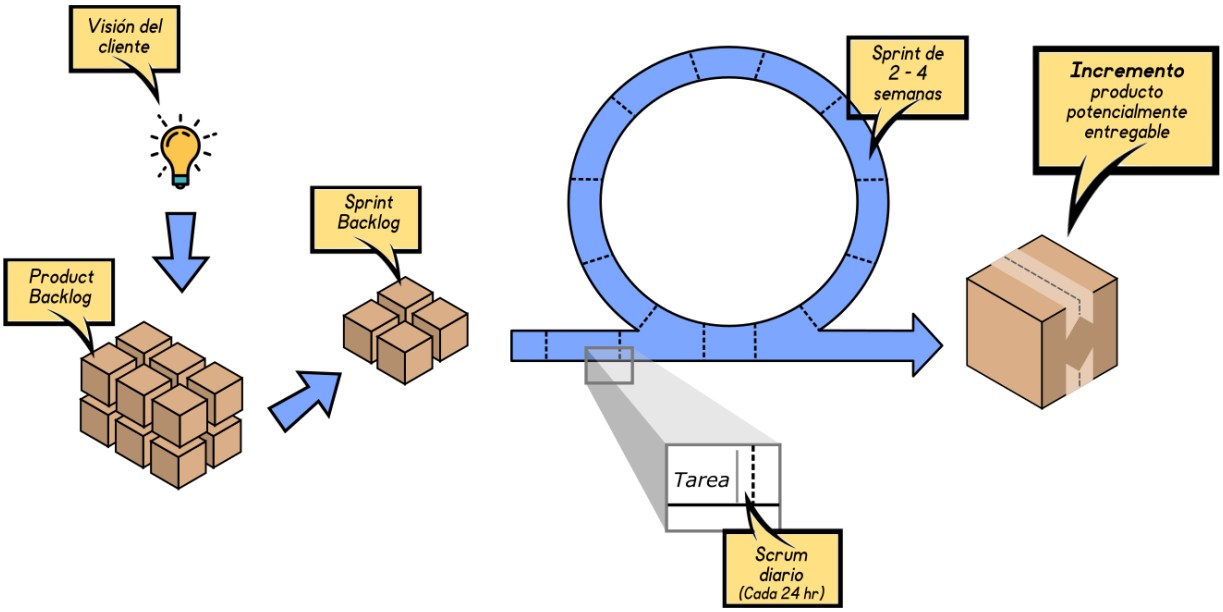


Figura 1.2 Visión general de Scrum

De manera breve se describirán los componentes y conceptos que empleados en *Scrum* [20]:

Las reuniones

- **Planificación del *Sprint*:** Con una duración entre máxima de 8 horas por *Sprint*, esta etapa de trabajo se determina cuál es el trabajo y los objetivos a realizar con esta iteración. Esta reunión se genera el *Sprint backlog* o lista de tareas que se van a realizar.
- **Seguimiento del *Sprint*:** Reunión corta diaria, de aproximadamente unos 15 minutos, para dar repaso sencillo del avance de cada tarea y la planeación del trabajo previsto para la siguiente etapa.
- **Revisión de *Sprint*:** Análisis y revisión del incremento generado. Esta reunión es de manera informal de los resultados, con una duración entre 3 a 4 horas máximo por mes, y está destinada a favorecer la retroalimentación y fomentar la colaboración.

Los elementos

- ***Product backlog*:** Lista de tareas que nunca se completa, es dinámico, siempre está en constante cambio para identificar que productos necesitan ser apropiados, competitivos y útiles. Es el inventario de características que el propietario del producto desea obtener, organizado por orden de prioridad.

- ***Sprint backlog*:** Lista de los trabajos que realiza el equipo durante el *Sprint* para entregar un incremento al cliente. El equipo asume el compromiso de la ejecución. Las tareas están asignadas a personas, y tienen estimados el tiempo y los recursos necesarios.
- **Incremento:** Resultado de cada *Sprint*, se trata de un producto completamente terminado y en condiciones de utilizase.

Los roles o responsabilidades

- **Propietario del Producto:** es responsable de maximizar el valor del producto y el trabajo del equipo de desarrollo, que varía dependiendo de las organizaciones, equipo *Scrum* y personas. Es solo una persona, y solo una, conocedora del entorno de negocio del cliente y de la visión el producto.
- **El equipo de desarrollo:** formado por profesionales que realzan el trabajo de la entrega de un incremento potencialmente liberable del producto. Los equipos están estructurados y capacitados para organizar y gestionar su propio trabajo, la sinergia resultante optimiza la eficiencia y eficacia general del equipo.
- ***Scrum* Master:** es el responsable de asegurar el funcionamiento de *Scrum*, dentro del proceso y la metodología que emplea la organización. Generalmente, es el líder del equipo de desarrollo, ayuda a los que están fuera del equipo a entender cuáles de las interacciones con el equipo son útiles y cuáles no.

1.2.6.2. *PSP*

Proceso Personal de Software (*Personal Software Process, PSP*) es un proceso de mejora individual que ayuda a controlar, gestionar, y mejorar la forma de trabajar. "Es un marco de trabajo estructurado de formularios, normas, y procedimientos para el desarrollo de software" [21] desarrollado por Watts S. Humphrey en 1993.

El *PSP* tiene como único propósito la ayuda para la mejora de las habilidades de Ingeniería de Software individuales. La estrategia de *PSP* es mejorar el rendimiento personal de la Ingeniería de Software, utilizando prácticas bien definidas para monitorizar y esforzarse en mejorar el desempeño personal, y ser responsable de la calidad de los productos que

desarrolla. En la *Figura 1.3* se observa el proceso evolutivo que sigue el *PSP* para el desarrollo de software.

PSP se basa en los siguientes principios de planificación y calidad [21]: 1) Cada desarrollador es diferente; para ser más efectivo, los desarrolladores tiene que planear su trabajo en base a su información personal, 2) Para la mejorar continua del desempeño, los desarrolladores tienen que definir y medir bien sus procesos, 3) Para producir productos de calidad, los desarrolladores tienen que sentirse responsable de la calidad de su trabajo, 3) El costo es menor si se encuentran y reparan defectos en fases tempranas que después, 4) Es más eficiente prevenir los defectos que buscarlos y repararlos, 5) El camino correcto el siempre el más rápido y barato de trabajar.

1.2.6.3. *TSP*

Proceso de Software en Equipo (*Team Software Process, TSP*) creado por Watts Humphery en 1996, con el objetivo de hacer frente al compromiso, el control, la calidad y los problemas del trabajo en equipo con que se enfrentan la mayoría de los equipos de desarrollo de software. *TSP* demostró ser eficaz para los equipos de software obteniendo el máximo rendimiento de las habilidades de sus miembros y cómo desarrollar productos de alta calidad en horarios previsibles y dentro del presupuesto [21].

El objetivo del *TSP* es construir y guiar a los equipos de desarrollo. Los equipos son necesarios para proyectos complejos de software. El desarrollo de sistemas es una tarea en equipo, y la efectividad del equipo determina la calidad del software. Los equipos de desarrollo tienen múltiples especialidades y todos los miembros trabajan en vista de un objetivo en común.

Los objetivos de *TSP* [21] son: 1) apoyar a los equipos a desarrollar productos de calidad dentro de los costos y tiempos establecidos, 2) tener equipos rápidos y confiables, y 3) optimizar el desempeño del equipo durante todo el proyecto. Usando *TSP*, los desarrolladores: a) siguen un proceso personal definido y medido, b) planifican el trabajo

antes de hacerlo, c) reúnen datos acerca del tiempo, tamaño y defecto, y d) utilizan estos datos para administrar el trabajo del personal y asegurar la calidad [22].

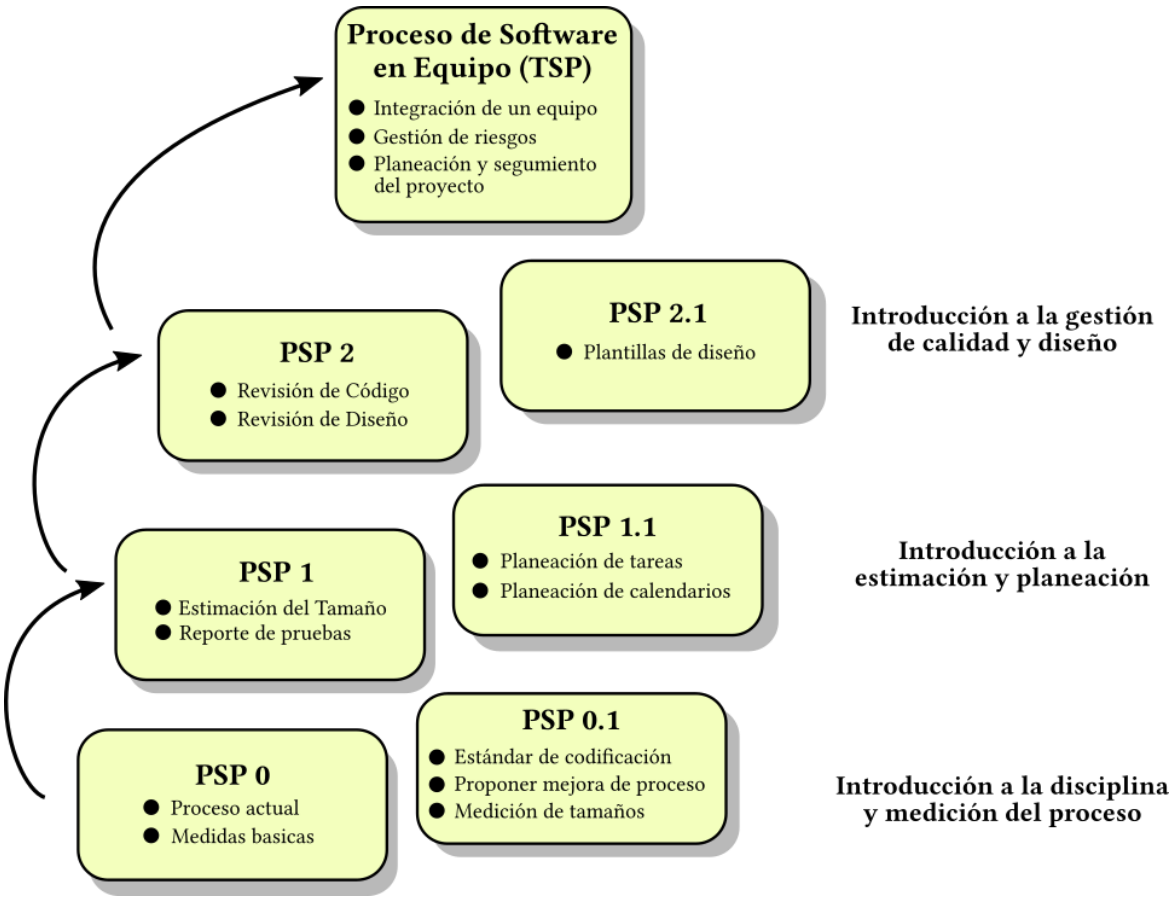


Figura 1.3 Niveles de madurez de PSP

1.2.7. Herramienta de gestión de proyectos

1.2.7.1. *JIRA*

Es una aplicación basada en Web para el seguimiento de errores, incidentes y para la gestión operativa de proyectos. La empresa *Atlassian* la creó inicialmente en el rubro del desarrollo de software como una herramienta de apoyo para gestión de requisitos, seguimiento de estatus y más tarde para el seguimiento de errores.

Este tipo de software integra muchos servicios para el desarrollo de software, principalmente para aquellos equipos que utilizan métodos ágiles; dispone de un sistema de alertas dentro de la plataforma Web, normalmente utilizando el correo electrónico para notificar a todas las personas afectadas en el transcurso de una incidencia.

La última versión estable conocida hasta ahora es la versión 7.1.0, dependiendo de las necesidades del consumidor *JIRA* dispone de dos versiones para su instalador: *Software Cloud* y *Software Server*. En la versión *Cloud* los datos y la configuración son administrados por la nube, mientras que la versión *Server* proporciona un instalador para cualquier computadora del consumidor y permite configurarlo de acuerdo a sus necesidades.

JIRA ofrece una versión de prueba de 7 días para la versión Cloud y 30 días para la versión *Server*, mientras que sus licencias de pago varían dependiendo de la cantidad de usuarios que van a utilizarla, la más popular es de US $10/mes para la versión *Cloud* con capacidad máxima para 10 Usuarios y US $10/pago único para la versión Server con la misma capacidad.

JIRA ofrece los cuadros de mando o *Dashboards* que permiten obtener información personalizada en tiempo real usando diferentes *gadgets*, toman como origen de datos un filtro de búsqueda y muestran información en forma determinada. La herramienta permita a cualquier usuario crear y administrar sus propios *Dashboard* con la posibilidad de compartirlos, en modo lectura, con otros usuarios.

La aplicación dispone de diferentes informes que aportan información adicional categorizada que ayuda a la toma de decisiones para su constante mejora y cuenta con varios *add-ons* en su mercado de componentes ya sea de pago o gratuitos, además se integrar con otra herramienta de *Atlassian* [23].

1.2.7.2. CONFLUENCE

Es un software colaborativo para equipos de desarrollo de software desarrollado por *Atlassian*, permite la centralización de información y conocimiento que necesitan para

desarrollar su trabajo de forma más rápida. El propósito declarado en la versión 1.0 fue "construir una aplicación acorde a los requisitos de un sistema de administración de conocimiento empresarial, sin perder la simplicidad esencial y potente del *wiki* en el proceso". En versiones más recientes, evolucionó a una herramienta de plataforma de colaboración entregada y adatada para trabajar con otros productos de *Atlassian*.

CONCLUENCE permite que cada equipo de desarrollo, departamento o proyecto dispongan de su propio espacio de trabajo personalizado y permisos, la información se mantiene accesible y ordenada con una jerarquía flexible. Tiene un editor flexible y enriquecido para exploradores Web, además proporciona al usuario plantillas predeterminadas para crear notas de reuniones, requerimientos de productos, lista de archivos, planes de proyecto, entre otras.

Permite una fácil colaboración entre equipos, gracias al espacio destinado para discusiones, toma de decisiones, asignación de tareas, el calendario, entre otras. Tiene una integración total con la herramienta *JIRA*, permite generar enlaces con las notas de requerimientos y la creación y asignación de incidencias en *JIRA*; además, para apoyar el desarrollo ágil proporciona espacio para retrospectivas una vez terminados un *Sprint* en *JIRA*.

CONCLUENCE ofrece una versión de prueba de 7 días para la versión Cloud y 30 días para la versión Server, mientras que sus licencias de pago varían dependiendo de la cantidad de usuarios que van a utilizarla, la más popular es de US $10/mes para la versión Cloud con capacidad máxima para 10 Usuarios y US $10/pago único para la versión Server con la misma capacidad [24].

1.3. Situación tecnológica, económica y operativa de la empresa

TecnoMotum es una empresa mexicana que se especializa en soluciones que permiten hacer mediciones a distancia para tomar decisiones en tiempo real. Estas soluciones se basan en telemetría creadas a través del centro de investigación y desarrollo (Centro de I+D), que cumple con los más altos estándares y certificaciones de calidad en su proceso productivo.

TecnoMotum es una empresa autorizada por la Dirección General de Seguridad Privada (DGSP) para la comercialización de productos y servicios en monitoreo de activos a través de tecnologías de rastreo satelital. La fiabilidad de sus productos está certificada por el Centro de Experimentación y Seguridad Vial de México (CESVI). Su misión es ofrecer servicios y soluciones tecnológicas de vanguardia, que generen información, conocimiento y valor agregado, que contribuyan al desarrollo y éxito de sus clientes.

Tiene a su disposición la tecnología para el ensamble de soluciones, así como un equipo de desarrolladores para necesidades específicas e integración con otros sistemas o aplicaciones para la administración de su empresa, ERP's, Bases de datos, entre otras. Además, su Centro de Investigación y Desarrollo (I+D) está certificado por *MoProSoft*, proporcionando garantía que justifica el alto nivel de madurez y capacidades de la organización.

1.4. Planteamiento del problema

Actualmente la mayoría de las empresas u organizaciones utilizan sistemas complejos para sus actividades cotidianas, por lo que es importante desarrollar software de bajo precio y en dentro plazo de tiempo. Sin embargo, en proyectos de mayor tamaño se pierde el control de la calidad, tanto en el producto como del proceso utilizado. Este problema surge principalmente de dos puntos de vista, el primero se refiere a la demanda del tipo de aplicaciones sumando la complejidad que estas representan a la hora del desarrollo, y el segundo estás relacionado con la competitividad de las empresas por entregar un producto en poco tiempo, de bajo costo y que cumpla con las características del cliente.

Gracias a los métodos ágiles como *Scrum*, *XP*, *Kamban*, *Crystal*, *Lean Development* (LD) por mencionar algunas, los equipos de desarrollo de software realizan entregas en poco tiempo y los proyectos se adaptan a las condiciones cambiantes del cliente. Por otra parte, los modelos de mejora de calidad para la gestión del proceso de desarrollo como *CMMI-DEV*, *Six Sigma* y *PSP/TSP* permiten tener una buena gestión del proceso utilizando guías para supervisar que los desarrolladores sigan el flujo establecido por la empresa y ayudan a mejorar calidad del producto final. Sintetizando las ventajas de cada enfoque es posible obtener un proceso hibrido que mezcle las mejores

herramientas y técnicas. Las grandes compañías y los grandes proyectos de software utilizan ambos enfoques en la medida de sus objetivos y su entorno.

Apoyándose en el principio de CMMI-DEV el cual dice: "*La calidad de un sistema es consecuencia de la calidad de los procesos empleados en su desarrollo y mantenimiento.*" la empresa TecnoMotum S.A. de C.V., que se dedica al desarrollo de soluciones basadas en telemetría, en su interés por cumplir con los más altos estándares y certificaciones de calidad en su proceso productivo, desea adoptar una combinación de *Scrum* y *PSP* en su centro de desarrollo de I+D adaptándolo a sus necesidades y forma de trabajo particulares.

1.5. Objetivos

1.5.1. Objetivo General

Proponer y diseñar un plan de trabajo para la adopción de *Scrum* y *PSP* dentro del área de desarrollo de software de la empresa TecnoMotum S.A de C.V. y realizan la implementación de *PSP*0 hasta *PSP* 1.1 en un equipo de desarrollo.

1.5.2. Objetivos Específicos

1) Revisar investigaciones relacionadas con la adaptación entre métodos agiles y métodos disciplinados para establecer una idea que permita combinar *Scrum* y *PSP*.
2) Conocer y analizar el proceso que se utiliza en el área de desarrollo de la empresa TecnoMotum para identificar el flujo de actividades de la metodología, lo roles que intervienen y los recursos disponibles para los equipos de desarrollo.
3) Estudiar los conceptos principales del *Personal Software Process* para identificar cuales aspectos son posibles de adaptar a las prácticas de desarrollo de *Scrum*.
4) Estudiar el estándar de calidad *MoProSoft* para localizar las actividades y documentos generados que apoyen a la combinación de *Scrum* y *PSP*.
5) Proponer y diseñar un modelo que permita combinar *Scrum* y *PSP* para mejorar la gestión del proceso de desarrollo de software y a los equipos de trabajo implementarlo en un plazo medio.

6) Seleccionar las herramientas a utilizar para la gestión del proceso de *Scrum* y *PSP*, y establecer los documentos y reportes serán elaborados a través de dichas herramientas.
7) Diseñar un caso de estudio que permita verificar y evaluar la efectividad del modelo propuesto dentro de un equipo de trabajo.
8) Analizar los resultados y realizar los ajustes necesarios al modelo propuesto para mejorar su integración con diferentes equipos.

1.6. Justificación

En la actualidad, debido a que las organizaciones dependen de un software para sus actividades de trabajo, la calidad es uno de los criterios más importantes de selección por parte de los clientes. Si una empresa entrega un Software que satisface las necesidades del cliente, es decir, un *producto de calidad* se establece un vínculo de confianza entre cliente y empresa.

La combinación de métodos ágiles y disciplinados es una solución cada vez más aceptada por los ingenieros de software, la mayoría de las empresas utilizan CMMI como modelo de madurez. El problema que enfrenta las PyMEs para implementar CMMI es la carencia de un proceso y la gran cantidad de áreas que define el modelo de madurez. *Scrum* es perfecto para cualquier tipo de empresas ya que permite agilizar el proceso de desarrollo sin sobrepasar los tiempos y recursos estipulados por los clientes sin olvidar la gestión y la calidad del producto.

Scrum se evalúa como un "muy buena" práctica para proyectos de tamaño pequeño y mediano con un equipo de trabajo relativamente pequeño (3 a 9) [9,10]. Por otro lado, *PSP* define prácticas sencillas y ligeras, considera una "excelente" práctica para proyectos de todo tipo de tamaño aplicable a nivel personal (1 a 3) [10] que favorecen la institucionalizan de procesos para este tipo de empresas. *Scrum* y *PSP* comparten características generales como son: equipos de trabajo pequeños, multidisciplinarios y auto-dirigidos para producir software; aquellos que utilizan *Scrum* usan las buenas prácticas para construir un producto adecuado y *PSP* ayuda a construir el producto correctamente.

Capítulo 2 Estado de la práctica

El objetivo de este capítulo es mostrar una colección y análisis de trabajos relacionados con el proyecto de innovación para determinar el flujo de trabajo de esta investigación, las técnicas y métodos que es posible utilizar en la construcción de un modelo hibrido entre *Scrum* y *PSP*.

2.1. Trabajos relacionados

En [25] se realizó un análisis entre los métodos ágiles (*XP*, *Adaptive Software Development*, *Crystal*, *DSDM*) y los métodos disciplinados (basados en riesgos, basados en planes, CMM, CMMI) se establecieron 7 áreas clave, para ambos enfoques, donde tienen ventaja para un desarrollo de software exitoso, con estas áreas (*Desarrolladores, Clientes, Requerimientos, Arquitectura, Ajustes, Tamaño y Objetivo principal)* se realiza una comparación para buscar un punto de equilibrio entre ambos enfoques de desarrollo. En resumen, los métodos ágiles tienen su punto fuerte en proporcionar resultados rápidos del software construido, adaptación contra el rápido cambio de los requerimientos y el diseño de la arquitectura; pero carecen de un soporte para predecir riesgos futuros. En contraste, los métodos disciplinados cuentan con planes para el ciclo de vida, proporcionan alta seguridad y confianza a los clientes; pero carecen de la habilidad para adaptarse a cambios rápidos en los requerimientos y arquitectura del sistema, elevando el tiempo, costo y uso de recursos. Se concluyó que ambos métodos, ágiles y disciplinados, tiene puntos fuertes para cada proyecto y para combinarlos es necesario realizar un análisis de riesgos para determinar cuáles son las características que determinan un mejor balance.

Paulk mencionó en [26] las diferencias que existen entre los métodos ágiles y los métodos disciplinados considerando como puntos de partida *El Manifiesto Ágil* y los métodos que describe CMM. Las metodologías ágiles defienden prácticas de ingeniera que son buenas, aunque algunas presentan una implementación extrema controversial y contraproducente fuera de un domino limitado. Paulk en su análisis hizo hincapié en la definición de los procesos de mejora basados en CMM contra las características que ofrecen los métodos ágiles resaltando: *personas por encima de los procesos, trabajo de desarrollo sobre la documentación, la colaboración de los clientes sobre los contratos y la respuesta rápida a cambios sobre la planificación.* Como conclusión, se dijo que

los métodos ágiles implican cierta disciplina, aunque las implementaciones difieren en formas extremas de las prácticas tradiciones, pero maximizando los beneficios de las buenas prácticas.

Sutherland, et al propusieron una combinación entre *Scrum* y CMMI en [27] para apoyar a las empresas a institucionalizar los procesos de desarrollo de software basado en 12 prácticas genéricas asociadas con los niveles de madurez 2 y 3 de CMMI. Los autores observaron que la principal crítica de CMM son los efectos del enfoque en la orientación de los procesos, la mayoría de las críticas negativas en contra de CMM se realizan por la existencia de implementaciones fallidas. CMMI nivel 5 proporciona una alta predictibilidad y mejora la ingeniería del producto para hacerlo escalable, facilita el mantenimiento, la adaptación y la fiabilidad. CMMI da una idea de cuáles son los procesos que se necesitan para mantener una disciplina madura capaz de predecir y mejorar el rendimiento de la organización y sus proyectos. Por otro lado, *Scrum* proporciona las guías para una gestión de proyectos eficiente que permita una gran flexibilidad y adaptabilidad. En conclusión, al combinar ambos enfoques surgió una "poción mágica", donde *Scrum* asegura que procesos se implementan eficientemente mientras se adoptan los cambios, y CMMI asegura que todos procesos relevantes sean considerados.

En [28] se propuso un modelo simple de madurez de nivel 4 para XP para indicar el riesgo asociado a un proyecto y mostrar las direcciones para una mejora. Un modelo de madurez para XP se presentó para un entorno industrial, pero se tiene que tener en cuenta algunos aspectos para su desarrollo: *el bajo costo de la introducción del modelo, visión clara entre la relación de nivel de madurez y calidad de los productos desarrollados, el modelo tiene que ser ligero como lo es XP*. XPMM (*eXtrem Programming Maturity Model*) es un modelo propuesto cercano a CMMI y *PSP*, consiste en 4 niveles: 1) No cumple con nada, 2) Inicial, 3) Avanzado, 4) Maduro. Para avanzar entre los niveles el equipo del proyecto tiene que cumplir todas las prácticas asignada a cada nivel. Lo resultados muestran que el nivel de madurez más alto esta relaciona con el rendimiento del equipo (sin horas extras, todo código paso las pruebas de unidad, el cliente está satisfecho). Además, el modelo ayudó a diferenciar un proyecto real *XP* de los pseudoproyectos que solo comparten la característica: "falta de documentación escrita".

Una investigación dio como resultado una propuesta para la modificación de *PSP* con el objetivo de aligerar el proceso de desarrollo de software y hacer más fácil su seguimiento, mantenido sus principios básicos. La nueva metodología mencionada en [29] es una ampliación con las prácticas eficaces de la programación extrema con el fin de apoyar una mejor planeación y control de la calidad el producto. El principal objetivo de metodología propuesta llamada *eXtrem Programming Personal* (PXP) es mejorar el rendimiento y la calidad de los ingenieros mediante la automatización de las actividades diarias del desarrollador y retrospectivas regulares. La metodología es formada por 6 principios básicos derivados de *PSP* pero reduciendo la cantidad de documentación y esfuerzo de mantenimiento, con un proceso iterativo y aplicando prácticas que permiten al desarrollador ser más flexible y responsable de los cambios. PXP se conforma de 14 prácticas de desarrollo y organización, seis prácticas de *PSP* se preservan y otras seis prácticas provienen desde la programación extrema. PXP está conformado por ciclo de vida de 7 fases: *Requerimientos, planeación, iteración inicial, diseño, implementación, pruebas al sistema, retrospectiva*. Los resultados se comparan con un desarrollo ad-hoc, los resultados se inclinaron a favor en PXP gracias a su planificación solo se necesitan la información necesaria para trabajar y con una disminución del 50% de defectos encontrados en comparación con el desarrollo ad-hoc.

En [30] propusieron una representación de *PSP* dentro del núcleo de SEMAT basados en combinaciones de *Scrum*, XP con *PSP*. Gracias a que SEMAT presenta un nuevo lenguaje en común en el cual es posible describir cualquier práctica de software, sin importar el método de desarrollo que se utilice. *PSP* es representado con el alpha "*way of working*", los sietes niveles de *PSP* se representan como estados de una sub-alpha llamada "*PSP Compliance*" esta representación describe las actividades que son necesarias para el equipo de desarrollo. Por otro lado, la representación de *Scrum* dentro del núcleo de SEMAT es muy sencilla y clara, utilizando de nuevo las alphas para representar las actividades principales (*Backlog*, incremento, *Sprint backlog* y al equipo de desarrollo). En conclusión, la representación emplea el núcleo SEMAT para facilitar la realización de las prácticas *PSP* en diferentes métodos de software, el utilizar *Scrum* donde la capacidad de adaptación es parecida, la representación define las actividades a realizar por el equipo para cumplir los niveles de *PSP* en lugar de definir herramientas para cada nivel.

El objetivo de [31] se presentó en la correspondencia entre CMMI y *Scrum*, mostrando diferencias entre ellos e identificando como las pequeñas y medias organizaciones están adoptando prácticas completarías en sus proyectos para hacer estos enfoques más dóciles. Resulta útil para organizaciones que tiene un proceso basado en métodos disciplinarios como CMMI y planean mejorar la agilidad de los procesos. Es posible que CMMI y *Scrum* se complementen una a la otra creando una sinergia que beneficia a la organización que los utiliza. En conclusión, la idea principal es la integración de las siguientes actividades: *Nivel del Proceso de la Organización, Administración de las Actividades del Proyecto. Gestión de Rasgos; Soporte a la Gestión de Configuración de Procesos, Aseguramiento de la Calidad.*

Se muestra en [32] la posibilidad de un mapeo entre el tercer nivel de CMMI y tres prácticas como *Scrum*, Kanban y XP. El planteamiento parte de otros autores que realizaron análisis para determinar la compatibilidad entre métodos ágiles y CMMI. Un mapeo extenso y detallado es mostrado donde se compaginan las prácticas de CMMI Nivel 3 y cómo es posible implementar *Scrum*, Kanban y XP a cada una de ellas. Las áreas utilizadas de CMMI son: *Desarrollo de requerimientos, Solución Técnica, Integración del Producto, Verificación, Validación, Concentra en el proceso de la organización, Definición del proceso de la organización, Entrenamiento de la Organización, Gestión de la integración del proyecto, Gestión de riesgos, Resolución y Análisis de decisiones.* Los resultados muestran una cobertura, de CMMI Nivel 3, con *Scrum* del 44% involucrado en las áreas de administración y procesos de la organización, con XP con el 45% en los detalles e implementación de procesos y con Kanban con el 6% en el aspecto de control y decisiones.

En [33] propone una estrategia para implementar la administración de proyectos ágiles en compañías las cuales busquen la cumplir con CMMI haciendo uso de las mejores prácticas de Administración de Proyectos Ágiles como *Scrum*, *Feature Driven Development* (FFD), *Lean*, *Kanban*, *Crystal* y *Extrem Programming* (XP). Las iniciativas de transición a métodos ágiles fallan debido a que los procesos normativos se impulsan por grupos de mejora externos. Los empleados apoyan las iniciativas, pero se resisten al cambio por las creencias de que el proceso no encaja, el cambio es obligado y sin consultarlo antes. Basándose en este escenario, es posible definir una estrategia ágil para ayudar a las empresas a implementar prácticas de administración de proyectos de una manera

organizacional que ésta mejor estructurada y madura. El primer paso para la estrategia es una correlación de conceptos de cada área de proceso en un contexto de gestión de proyectos ágiles. En total, esta propuesta definió 84 maneras para implementar, de una forma ágil, la práctica de gestión de proyectos y 38 productos de trabajo como parte de la estrategia para implementación de la administración de proyectos ágiles para empresas que buscan CMMI.

En [34] se describió una experiencia de una empresa mexicana de desarrollo de software que integra métodos ágiles dentro del proceso de desarrollo CMMI-DEV Nivel 5. El objetivo del caso de estudio fue documentar los efectos, cuantitativos y cualitativos, reflejados en el desempeño de sus proyectos. Los resultados pretender proporcionar un ejemplo de los beneficios de incorporar técnicas ágiles con CMMI-DEV y que sirva como referencia a empresas latinas. La motivación del caso de estudio fue conocer las técnicas ágiles que integró Praxis (organización reconocida en el área de Tecnologías de la Información) con CMMI-DEV para establecer una nueva forma de construir software para mejorar la productividad. Los resultados muestran que los efectos de la integración de técnicas ágiles con CMMI-DEV tuvieron un impacto sobre los integrantes de los equipos de trabajo, el cliente y la misma organización. Algunos beneficios obtenidos fueron: *Enfoque en producto y no en documentación, Selección de las tareas a realizar, descripción de tareas, mayor control de actividades, corrección oportuna de defectos, aumento de productividad, conservar la calidad, mejora profesional de los integrantes del equipo de trabajo, mejoras personales de los integrantes, mejoras en el ambiente de trabajo, recertificación en Nivel 5 de CMMI-DEV y satisfacción del cliente*.

2.2. Análisis comparativo

De los trabajos mencionados con anterioridad se presenta en la *Tabla 2.1* una comparativa que permite una rápida y clara perspectiva de las investigaciones realizadas por diferentes autores.

Tabla 2.1 Comparativa de Trabajos Relacionados

Objetivo	Métodos Utilizados	Proceso Metodológico	Resultados
Get Ready For Agile Methods, With Care [25]			
Sintetizar las características de los métodos ágiles y disciplinados para proporcionar a los desarrollares de software un amplio espectro de herramientas y opciones.	*Ágiles:* XP, Adaptive Software Development, Crystal, DSDM. *Disciplinados:* basados en riesgos, basados en planes, CMM, CMMI.	Una comparación entre áreas clave de ambos enfoques. Estableció 7 áreas familiares (*Desarrolladores*, *Clientes*, *Requerimientos*, *Arquitectura*, *Ajustes*, *Tamaño* y *Objetivo principal*) en donde cada método tiene ventajas en determinados proyectos.	Por sugerencia del autor, realizar análisis de riesgos basado en las características del proyecto utilizando las 7 áreas descritas para determinar un balance entre los enfoques.
Agile Methodologies And Process Discipline [26]			
Dar una perspectiva sobre la compatibilidad de los métodos ágiles con los métodos disciplinados.	*Ágiles:* Principios del manifestó ágil y XP. *Disciplinados:* CMM y SW-CMM.	Comparación entre características de ambos enfoques: *Velocidad* contra *Disciplina*, *Individuos* contra *Procesos*, *Desarrollo* contra *Documentación*, *Respuesta a cambios* contra *Planeación*, *Colaboración de Clientes* contra *Contratos*; principalmente de XP y CMM y SW-CMM.	Por sugerencia del autor, recolectar información sobre el tipo de proyecto, generar un análisis considerando la naturaleza del proyecto para la integración de prácticas de ambos enfoques.
***Scrum* and CMMI Level 5: The Magic Potion for Code Warriors [27]**			
Establecer una combinación entre *Scrum* y CMMI para mejorar la adaptabilidad y predictibilidad en las compañías.	*Ágil: Scrum.* *Disciplinado:* CMMI nivel 2 y 3.	A través de 12 Prácticas Genéricas asociadas con los niveles de madurez 2 y 3 de CMMI se establece la disciplina dentro de los proyectos de una empresa. Por medio de una lista de	Combinación sin nombre mencionado de CMMI con *Scrum* y Lean, ofrecen disciplina y agilidad para la institucionalización de procesos dentro de una empresa. Evaluados en dos

Objetivo	Métodos Utilizados	Proceso Metodológico	Resultados
		críticas hacia CMMI se obtiene el apoyo de *Scrum* para resolver el problema.	casos de estudio con un alto desempeño en el proceso de desarrollo.
Toward Maturity Model For Extreme Programming [28]			
Proponer un modelo de madurez de nivel 4 para XP.	*Ágil:* XP. *Disciplinados:* CMMI.	Estableciendo un análisis para relacionar diferentes prácticas de XP con Prácticas Especificas de CMMI nivel 2.	eXtrem Programming Maturity Model (XPMM) consta de cuatro niveles: *No cumple con nada*, *Inicial*, *Avanzado*, *Maduro*. Un proceso incremental definido por un conjunto de prácticas obligatorias de XP para cada nivel del proceso.
Personal Extreme Programming – An Agile Process for Autonomous Developers [29]			
Presentar una nueva metodología extendida que provee prácticas eficientes para el desarrollo desde XP para ayudar a un mejor control de la planeación y control de calidad, manteniendo los principios básicos de *PSP*.	*Ágiles:* XP. *Disciplinados: PSP*.	Modificación de *PSP* para aligerar y hacer más fácil el proceso de desarrollo manteniendo sus principios de básicos. Tomando seis prácticas de *PSP* y seis prácticas de XP se estructuró una metodología que apunta hacia una mejor planeación y control de la calidad del desarrollo de software.	Personal eXtrem Programming (PXP) es un proceso iterativo y comprende un par de iteraciones y ciclos controlados. Conformado por siete fases: *Requerimientos*, *Planificación*, *Iteración inicial*, *Diseño*, *Implementación*, *Pruebas de Sistema*, *Retrospectiva*.
***PSP* Implementations for agile methods: a SEMAT-based approach [30]**			
Realizar una integración limpia entre *Scrum*, *PSP* y el núcleo SEMAT; para que desarrolladores tenga una perspectiva clara del proceso y reconozcan las dificultades de los mismos.	*Ágiles: Scrum.* *Disciplinados: PSP* y SEMAT.	Basado en un análisis de las adaptaciones *Scrum-PSP* y PXP mostradas anteriormente se propuso una adaptación que reutilice las prácticas de *PSP* en cualquier método de desarrollo de software. Se utilizó la notación de SEMAT para expresar en un lenguaje común las prácticas de *PSP*.	Integración con el núcleo SEMAT, muestra a *PSP* como una alpha "*way of working*". Los 7 niveles de *PSP* se representa como estados de un sub-alpha "*PSP Compliance*". Esta representación da a conocer a todo el equipo en qué nivel *PSP* se encuentran. Igualmente se adapta *Scrum* utilizando el Backlog, el incremento y el *Sprint* como alphas dentro de SEMAT.

Objetivo	Métodos Utilizados	Proceso Metodológico	Resultados
Research on Combining *Scrum* with CMMI in Small and Medium Organizations [31]			
Realizar una correlación entre *CMMI* y *Scrum*, mostrando diferencias entre ellos e identificando como las PyMEs están adoptando prácticas en sus proyectos para hacer estos enfoques más dóciles.	*Ágiles: Scrum* *Disciplinados:* CMMI	La integración de las siguientes actividades: *Nivel del Proceso de la Organización*, *Administración de las Actividades del Proyecto*. *Gestión de Riesgos*, *Soporte a la Gestión de Configuración de Procesos, Aseguramiento de la Calidad* y *Construir un Nuevo Ciclo de Vida de Software basado en Scrum*.	Es posible que CMMI y *Scrum* se contemplen una a la otra creando una sinergia que beneficia a la organización que los utiliza. Resultando útil para organizaciones que tiene un proceso basado en métodos disciplinarios y planean mejorar la agilidad de los procesos.
The projection of the specific practices of the third level of CMMI model in agile methods: *Scrum*, XP and Kanban [32]			
Proyectar entre CMMI 3 y tres prácticas ágiles: *Scrum*, Kanban y XP. Su planteamiento parte de otros autores que realizaron análisis para determinar la compatibilidad entre métodos ágiles y CMMI.	*Ágiles: Scrum*, Kanban, XP *Disciplinados:* CMMI	Mapeo extenso y detallado de áreas de CMMI: *Desarrollo de requerimientos, Solución Técnica, Integración del Producto, Verificación, Validación, Concentra en el proceso de la organización, Definición del proceso de la organización, Entrenamiento de la Organización, Gestión de la integración del proyecto, Gestión de riesgos, Resolución y Análisis de decisiones*.	Se muestran una cobertura, de CMMI Niv. 3, con *Scrum* del 44% involucrado en las áreas de administración y procesos de la organización, con XP con el 45% en los detalles e implementación de procesos y con Kanban con el 6% en el aspecto de control y decisiones.
An agile strategy for implementing CMMI project management practices in software organizations. [33]			
Las iniciativas de transición a métodos ágiles fallan debido a que los empleados creen que el proceso no encaja, el cambio es obligado y sin consultarlo antes.	*Ágiles:* Prácticas ágiles *Disciplinados:* CMMI	En total, esta propuesta definió 84 maneras para implementar, de una forma ágil, la práctica de gestión de proyectos y 38 productos de trabajo como parte de la estrategia para implementación de la	Para guiar a las empresas dentro este tipo de escenarios, este trabajo presentó una estratégica que apoya de manera gradual y disciplinada la puesta en marcha de la gestión de proyectos ágiles basado en marcos de trabajo y

Objetivo	Métodos Utilizados	Proceso Metodológico	Resultados
Basándose en este escenario de define una estrategia ágil que pueda ayudar a las empresas a implementar prácticas de administración de proyectos.		administración de proyectos ágiles para empresas que buscan CMMI.	métodos previamente validados y en su creciente uso ara la comunidad de desarrollo de software.
Integrating Agile Methods into a Level 5 CMMI-DEV Organization: a Case Study [34]			
Describir la experiencia de una empresa mexicana de desarrollo de software que integra métodos ágiles dentro del proceso de desarrollo CMMI-DEV Niv. 5. El objetivo del caso de estudio fue documentar los efectos, cuantitativos y cualitativos, reflejados en el desempeño de sus proyectos.	*Ágiles:* Pocos conceptos de *Scrum* *Disciplinados:* CMMI-DEV Nivel 5	La motivación fue conocer las técnicas ágiles que integró Praxis (organización en el área de Tecnologías de la Información) con CMMI-DEV para establecer una nueva forma de construir software y mejorar la productividad. Las unidades analizadas fueron: *Técnicas ágiles integradas con CMMI-DEV, Métricas de proyectos para determinar beneficios, Adaptación del grupo de trabajo*.	Los efectos de la integración tuvieron un impacto sobre los integrantes de los equipos, el cliente y la misma organización. Algunos beneficios obtenidos fueron: *Enfoque en producto y no en documentación, mayor control de actividades, corrección oportuna de defectos, aumento de productividad, conservar la calidad, mejoras en el ambiente de trabajo, satisfacción del cliente*.

En conclusión, la adaptación entre procesos ágiles y disciplinados es posible, pero depende mucho de las características de cada proyecto, el conocimiento del equipo de desarrollo, los recursos y tiempo disponibles; por lo tanto, es aconsejable realizar un análisis de características y riesgos lo más detallado posible para integrar los métodos necesarios. Las debilidades de un método se apoyan en las fortalezas del otro para generar un marco de trabajo sólido que mejore la calidad, control y seguimiento del proceso de desarrollo del software.

2.3. Propuesta de solución

Como solución al problema planteado se propuso el diseño de una estrategia para la implementación de conceptos *PSP* dentro del contexto industrial de software, se tiene contemplado abordar los niveles de *PSP0* hasta *PSP1.1*. Para desarrollar la estrategia fue necesario utilizar los siguientes recursos:

- El proceso de desarrollo ágil *Scrum* como marco de trabajo que guía al equipo de desarrollo de software en la construcción de un sistema de calidad y dar seguimiento al proceso utilizado en la construcción del mismo. Además, ofrece ayuda a los ingenieros a enfrentar problemas complejos adaptativos, al mismo tiempo que se entrega un producto con el máximo valor productivo posible. *Scrum* se caracteriza por ser ligero, fácil de entender, iterativo e incremental; dentro se aplican de manera regular un conjunto de mejores prácticas para trabajar en equipo y obtener los mejores resultados posibles de un proyecto.
- *PSP Dashboard* como herramienta de apoyo, software de código abierto, la versión utilizada es la *2.2 Student*. Es un software compuesto por un panel de control para escritorio donde se gestiona el proceso de desarrollo de las actividades y una parte basada en Web donde se registran todos datos que generan las actividades realizadas. También proporciona funcionalidad para la recopilación de datos, planeación, seguimiento, análisis de datos y exportación de datos. Construido para la plataforma de Java y se ejecuta sobre cualquier Sistema Operativo que la soporte (Windows, Unix, Linux, entre otros).
- *JIRA* y *CONFLUENCE* de *Atlassian* como herramientas de seguimiento de proyectos, la combinación de estas dos herramientas hace más fácil la administración de métodos ágiles para software, sus características se mencionaron anteriormente, aunque cabe hacer hincapié en su facilidad para la recolección de requerimientos y panel de control para *Scrum*.

Por último, es importante mencionar que la estrategia consta de: 1) El diseño de un modelo que combine *Scrum* con *PSP*, 2) La documentación pertinente, es decir, una guía que permita a los ingenieros en adoptar el modelo desarrollador en el punto anterior.

2.3.1. Justificación de la solución seleccionada

Tomando en cuenta el contexto de la tesis, se eligió una combinación entre *Scrum* y *PSP* para generar una estrategia para proveer mejor calidad del proceso utilizado para crear software y, por ende, elevar la calidad del producto final. La justificación para las herramientas seleccionadas es la siguiente:

En primer lugar, se encuentra *Scrum* un modelo de desarrollo de software que proporciona una guía a equipos de trabajo para adaptarse a los cambios constantes que surgen durante el desarrollo de un sistema, gracias a que es iterativo e incremental, desde la primera iteración de tiene un producto funcional conforme los requerimientos del cliente. Ofrece ventajas a *PSP* con respecto a la adaptación, estimación "empírica" del esfuerzo y registros de tiempos en bitácoras; también se menciona la facilidad con la que compaginan los roles de *Scrum* y *PSP*, de este modo los nuevos roles de *PSP* son perfectamente acoplables sin requerir mayor esfuerzo por parte de los integrantes del equipo.

En segundo lugar, la herramienta de apoyo *PSP* Dashboard ofrece un contexto dedicado a *PSP* para el ingeniero de manera individual protegiendo la privacidad de los datos en cierta medida. Como se mencionó en apartados anteriores su interoperabilidad se basa en la máquina virtual de Java. Lo más importante de la herramienta es la estadística que tiene integrada en su funcionalidad, haciendo más fácil el análisis de los datos recopilados en cada una de las fases de *PSP*.

Por último, se tienen las herramientas de seguimiento de proyectos *JIRA* y *CONFLUENCE* que proveen varias funcionalidades para el trabajo en equipo especialmente en el contexto ágil. Todas las herramientas utilizadas para el desarrollo de la solución ofrecen licencias de código abierto a excepción de *JIRA* y *CONFLUENCE* que tiene licencia de paga pero que son parte de los recursos proporcionadas por la empresa para la realización de esta tesis.

2.3.2. Proceso de desarrollo de la estrategia

En la figura 3.1 se muestra el flujo de proceso que se seguirá en la construcción del modelo *Scrum-PSP* y sólo se presentan los hitos más importantes del desarrollo, para agregar valora al modelo se utilizó la norma NMX-I-059-NYCE-2011 (*MoProSoft*) como estándar de calidad para el producto final de software.

Es importante mencionar que los hitos definición del método de desarrollo y diseño del método *Scrum-PSP* tiene prioridad y agrupan más actividades que se describen en los apartados próximos siguientes. En el hito aplicando *Scrum-PSP* a un caso de estudio está agrupando la capacitación a los integrantes del proyecto de software por medio de recursos de autoaprendizaje.

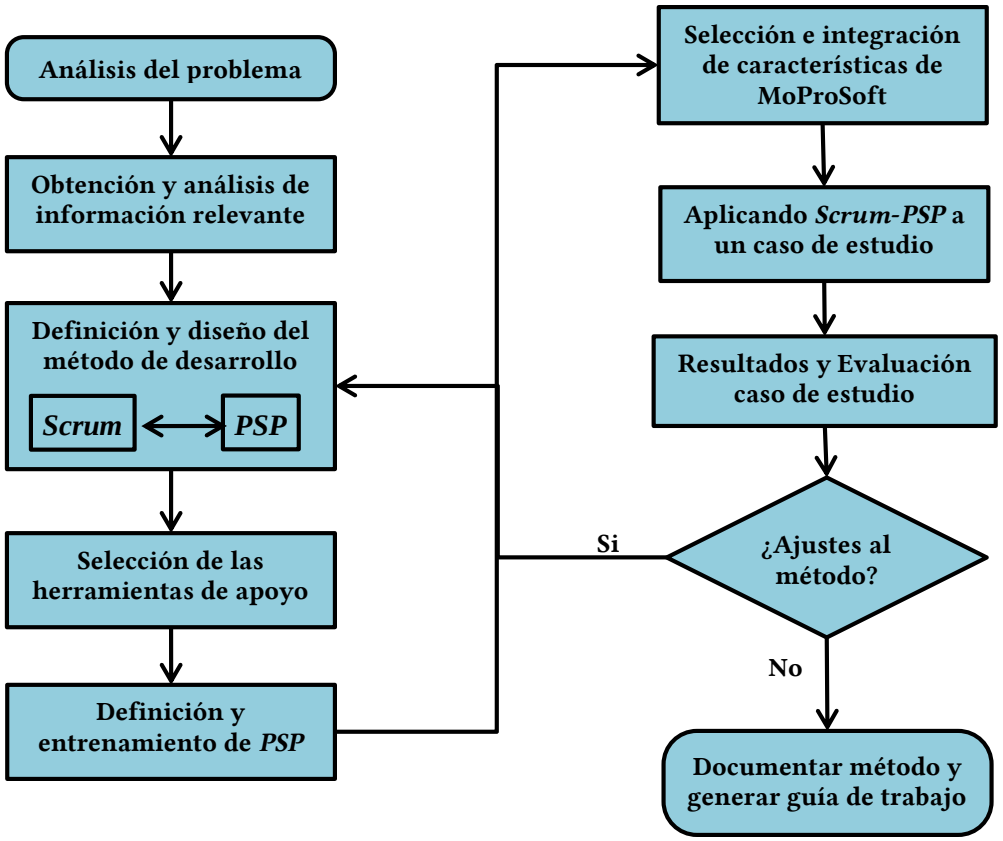


Figura 2.1 Método para desarrollar el modelo hibrido

Capítulo 3 Aplicación de la metodología

En el presente capítulo se presenta el desarrollo de una estrategia factible para la combinación del proceso disciplinado *PSP* y el método ágil *Scrum* utilizando las herramientas y métodos descritos en el apartado de la propuesta. El objetivo principal de este capítulo es proporcionar la información necesaria para el entendimiento del modelo *Scrum-PSP*, debido a que algunos recursos se proporcionan por parte de la empresa TecnoMotum S.A de C.V. alguna información se censura para respetar la confidencialidad de los datos.

3.1. Recopilación de la información relevante

Para conocer mejor el panorama que la empresa utiliza para el desarrollo de software se recopiló toda información que sea relevante, para afrontar con éxito la identificación y el análisis de los procesos críticos se indican las acciones a seguir:

- Establecer las estrategias de recolección de datos.
- Conocer el flujo del proceso y documentación del mismo.
- Determinar las áreas involucradas en la mejora de procesos.
- Determinar las personas involucradas en el proceso de desarrollo de software.
- Conocer el (los) estándar(es) de calidad que actualmente se utilizan en la empresa.
- Conocer la(s) metodología(s) para el desarrollo y mantenimiento de software.

Para obtener datos e información se utilizaron técnicas básicas de recolección las cuales se describen a continuación:

- **Entrevistas con involucrados:** Necesarias para tener una perspectiva de los procesos desde el punto de vista de los involucrados (Analista, Encargado de la Calidad, Líder de proyecto, Desarrollador), las entrevistas se abordan de manera informal.
- **Revisión a los flujos del proceso:** Con esta actividad se pretende identificar todos los elementos clave del desarrollo y es posible que se arrojen a luz límites más apropiados límites más apropiados a la hora de definir el proceso y descubrir posibles oportunidades de mejora.

- **Análisis de los documentos:** Con la identificación de los documentos que circulan y se generan durante el proceso se completa una idea global del proceso, para que sea más fácil delimitarlo.

3.1.1. Entrevistas a los involucrados

El análisis e identificación de los procesos comenzó con una pequeña reunión con el **Gerente de Ingeniería de Software** en la cual se concluyeron los siguientes puntos: 1) *Establecimiento del plan de trabajo:* se acordaron varias entrevista con las partes involucradas en el proceso de desarrollo de software para el análisis de la información generada por los mismos, 2) *Visión global de los objetivos del proyecto de implementación:* se acordaron cuáles eran los objetivos y resultados que se requieren para la implementación de *PSP* dentro de la empresa. En la reunión con el **Analista/Encargado** de la calidad se observó parcialmente el flujo de trabajo para el desarrollo de software, partiendo de la identificación de requerimientos, utilización de las herramientas de gestión de proyectos, distribución de las tareas para cada integrante del área; también se observaron los estándares de calidad que utilizan, los indicadores de productividad que se generan con la información recolectada de las herramientas de apoyo.

Así mismo, la entrevista realizada con el **Líder de proyecto** tuvo el objetivo principal de conocer cómo se lleva a cabo la metodología *Scrum* para el desarrollo del software. Además, cómo es la organización por parte del equipo del trabajo para la metodología, los recursos y técnicas que ocupan para trabajar en la construcción de software. En la sección 3.3 se explica con más detalle el proceso de implementación de la metodología *Scrum*.

Las entrevistas con los **Desarrolladores** revelaron una perspectiva de las tareas enfocadas a las actividades del desarrollo de software como son: 1) Realización de las tareas programadas con la metodología *Scrum*, 2) Entorno y ambiente de trabajo, 3) Recursos y herramientas para el apoyo de la construcción del software, 4) técnicas y estándares de codificación.

En resumen, las entrevistas proporcionaron información sobre los siguientes puntos:

- Definición global de los procesos para el desarrollo de software, definición funcional, alcance e identificación de los responsables de los mismos.
- Destinatarios y objetivos parciales de cada proceso, identificación de las expectativas del proceso, así como objetivos y flujos de entrada y salida.
- Componentes del proceso: determinación de los elementos que intervienen, de los recursos y de las actividades de cada proceso e interacciones entre ellos.
- Identificación de los problemas que impiden alcanzar los objetivos de la implementación de los procesos.
- Las técnicas, los recursos y estándares que utiliza cada involucrado para realizar las actividades asignadas a su rol de trabajo.

3.1.2. Análisis de documentos

Durante las entrevistas se proporcionaron documentos existentes sobre los diferentes procedimientos y protocolos de calidad, diagramas de procesos, plantillas, documentos de solicitud, manuales para el uso de las herramientas utilizadas por los ingenieros, por mencionar los más importantes. Gracias a la herramienta de apoyo para la gestión de los proyectos, se tienen espacios de trabajo donde reside todo el conocimiento del cada proyecto que lleva el equipo, incluyendo algunos diagramas de modelado y bases de datos, procesos de instalación de *plugins* necesarios para la codificación, notas sobre el análisis de los requerimientos y las reuniones de retroalimentación. Como parte del análisis de documentos se revisaron de manera rápida archivos de codificación de un proyecto, en donde se detectaron los lenguajes de programación utilizados y los estilos de codificación, así como el repositorio para las versiones y la distribución de los archivos durante el proyecto.

3.2. *Scrum* en TecnoMotum

En este apartado se presenta información resumida sobre el método *Scrum* que se utiliza en TecnoMotum S.A de C.V., con el fin de proporcionar al lector la información necesaria para familiarizarse con el lenguaje se definen de manera abstracta aquellos conceptos que sean

necesarios, si desea más información se recomienda consultar los artículos mencionados en la sección de referencias.

Recordando información descrita en secciones anteriores *Scrum* se define como un método de desarrollo muy simple, que requiere trabajo duro por la adaptación continua a las circunstancias de la evolución del proyecto. Es ágil porque es un modo de desarrollo de carácter adaptable, orientado a las personas antes que a los procesos, es iterativo e incremental. Es un proceso porque se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente en equipo, y obtener el mejor resultado posible de un proyecto. Está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, los requisitos son cambiantes, donde la innovación, la flexibilidad y la productividad son fundamentales.

Scrum comprende 3 fases para su ciclo de vida (*Figura 3.1*) que a continuación se describen de manera breve:

- **Pre-juego**: el principal objetivo, es la administración de las características del producto final, es aquí donde el cliente propone los requerimientos, definidos en un lenguaje natural y aceptados como parte de las caracterices del software; independientemente como se obtengan estos requerimientos tiene que ser entendidos por todos en un lenguaje sencillo y que no contengan, en la medida posible, ninguna ambigüedad. Esta fase incluye la descripción de la implementación del ***Product Backlog*** con un diseño de alto nivel (*High-level design*) de la arquitectura del sistema.
- **Desarrollo**: El equipo de desarrollo por medio de juntas se organiza para determinar el ***Sprint Backlog***, se reparten los roles, las responsabilidades, así como se realizan estimaciones en esfuerzo y el objetivo del ***Sprint***. Las fases del ciclo de vida del software se abordan, los ingenieros de software utilizan sus habilidades y técnicas para llevar a cabo las tareas asignadas, con la constante interacción a las variables de tiempo, requisitos, calidad, costo y competencia. Existen varias iteraciones de *Sprints* que tienen darse para que el sistema evolucione.

- **Post-juego**: El cierre del desarrollo, la preparación para la liberación del producto. Es en esta fase donde se realizan las últimas pruebas al sistema completo, la integración con otros sistemas con los que va a interactuar y si necesita configuraciones especiales para funcionar, i.e. Internet, almacenamiento en la nube, y algo importante es que tiene que incluir toda la documentación completa del proceso utilizado, así como los manuales apropiados del sistema al usuario o usuarios.

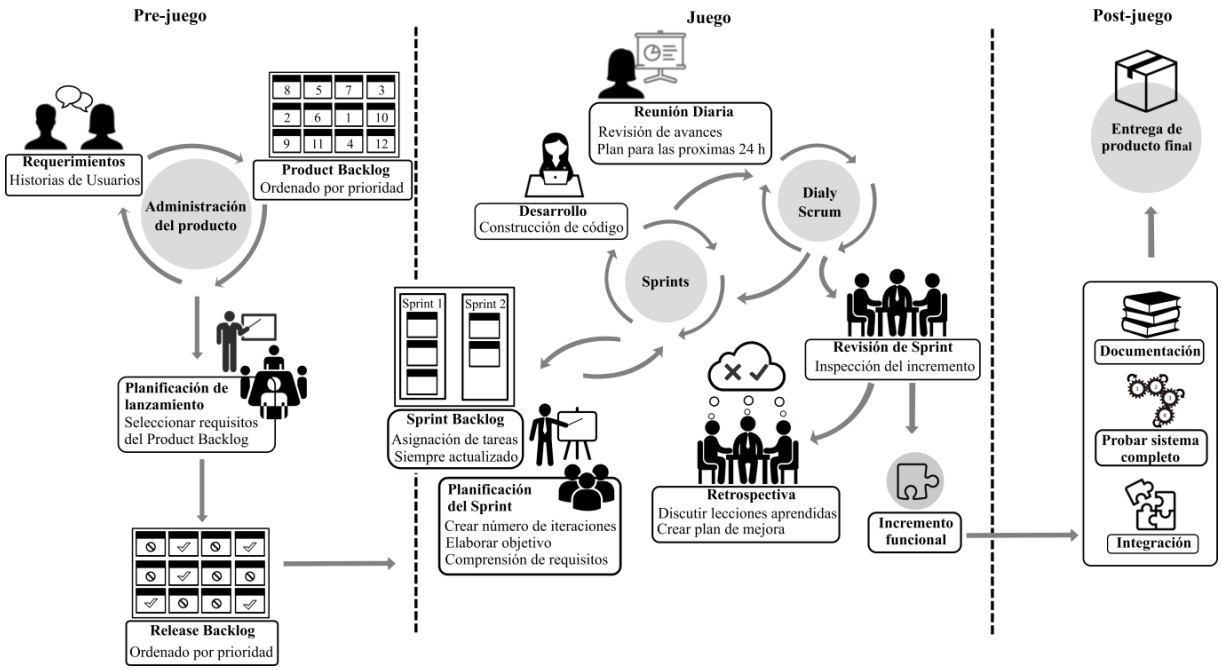


Figura 3.1 Ciclo de vida de Scrum

Explicar de manera completa el método *Scrum* dentro de la empresa es muy largo y complejo, por lo tanto, a continuación, solo se describen ciertas etapas relevantes del proceso junto con la combinación de las herramientas de apoyo de *Atlassian*

3.2.1. *Herramientas de desarrollo utilizadas*

Los equipos de desarrollo, así como todo el personal que labora en la empresa, utilizan dos herramientas para la gestión de los proyectos: *CONFLUENCE* y *JIRA* ambas de *Atlassian*. La primera (*Figura 3.2*) es un software para trabajo colaborativo que permite conectar a equipos de trabajo, crear y compartir el contenido y conocimiento que necesitan para desarrollar su trabajo mejor y con mayor rapidez. Entre los usos más importantes sobresalen los siguientes:

- Cada proyecto dispone de su propio espacio para mantener la información accesible y ordenada con una jerarquía flexible.
- Dispone de un editor para crear todo tipo de contenido: actas de reuniones, requerimientos, listas de archivos, planes de proyecto, entre otros.
- El conocimiento está centralizado lo que permite a los usuarios encontrar información sin importar dónde se encuentren.
- Total integración con *JIRA* para fomentar las transparencias entre las áreas gestionadas con *JIRA* y el contenido en *CONFLUENCE*.

La segunda (*Figura 3.3*) es un software para el seguimiento de errores, incidentes y gestión operativa de proyectos de desarrollo de software, flexible y permite coordinar procesos semiestructurados hasta llegar a transformarse en un motor de procesos modelable de acuerdo a los procesos de la organización. Se basa en una plataforma colaborativa que emplea tableros, paneles (*Dashboards*) y soporta metodologías ágiles de desarrollo de proyectos permitiendo llevar un control y seguimiento de los avances de los proyectos, de las tareas asignadas a cada desarrollador y generar reportes para la alta gerencia.

Otras herramientas utilizadas por los integrantes de los equipos de desarrollo son: el entorno de desarrollo *NetBeans IDE* para las cuestiones de codificación y *SubVersion* como repositorio para las versiones de los módulos diseñados.

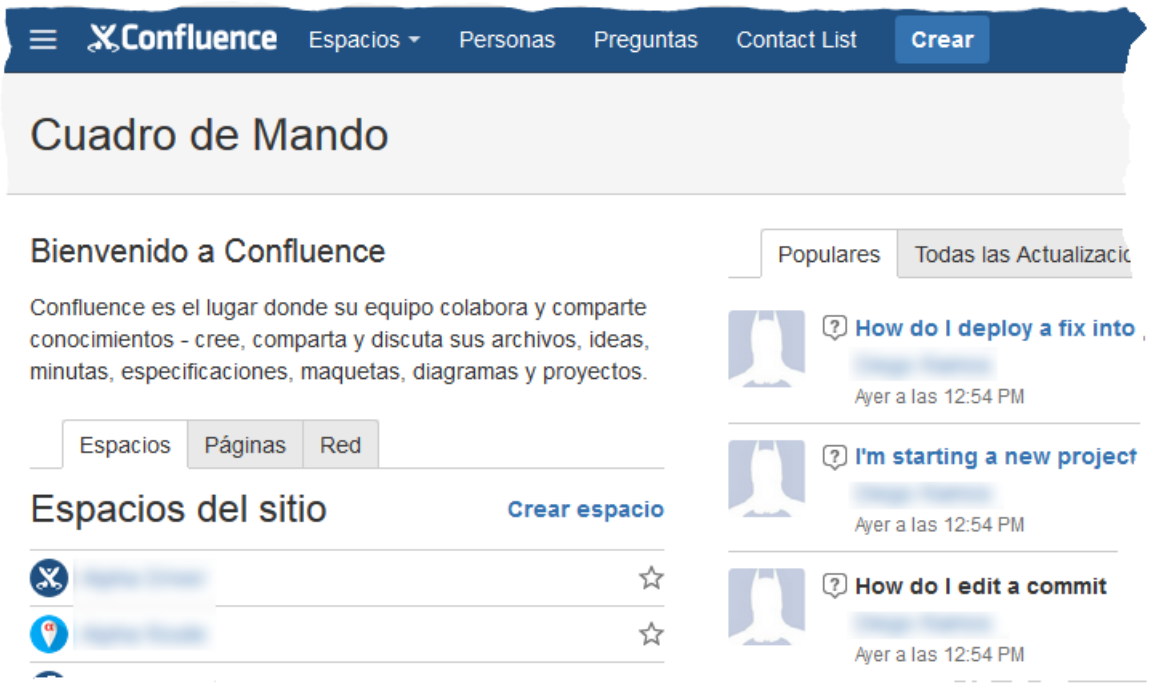


Figura 3.2 Pantalla principal de Confluence (Fragmento)

Figura 3.3 Pantalla principal de JIRA (Fragmento)

3.2.2. Roles en *Scrum*

El grado de funcionamiento de *Scrum* depende de tres condiciones:

- Características del entorno adecuadas al desarrollo ágil.
- Conocimiento de la metodología de trabajo por todas las personas implicadas.
- Asignación de las responsabilidades: del producto, del desarrollo y del funcionamiento del *Scrum*.

En la *Figura 3.4* se muestra la distribución de los roles de *Scrum* en relación con los cargos del personal dentro de la empresa, a continuación, se da una breve descripción del rol que juega dentro de *Scrum* y las funciones que realizan cada uno para lograr la construcción y mantenimiento de software.

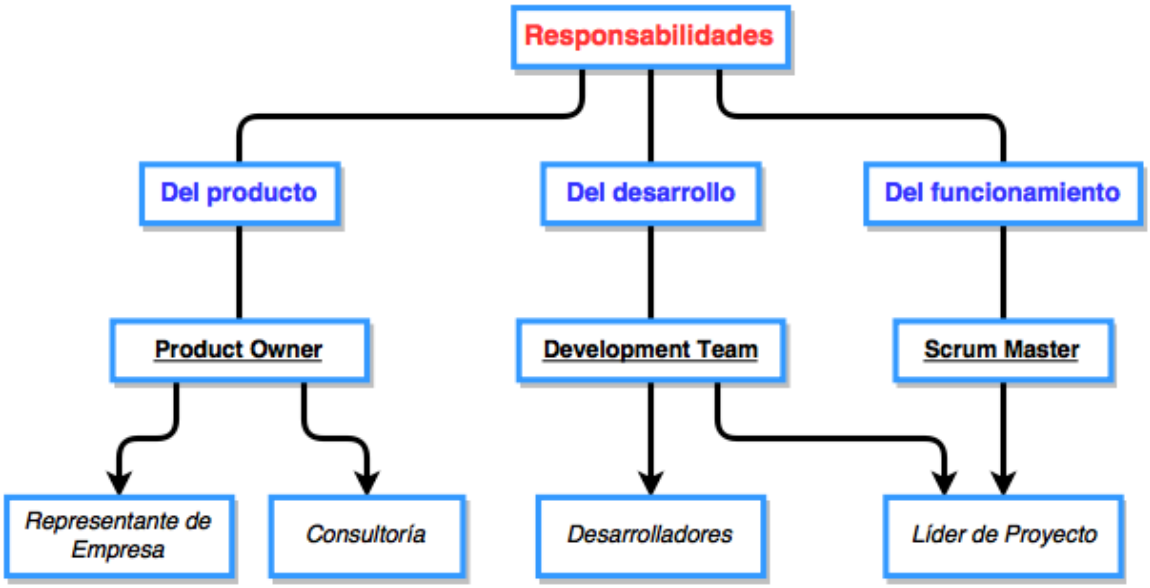


Figura 3.4 Distribución de roles de Scrum

***Product Owner*:** En el proyecto hay una persona que es conocedora del entorno del negocio de cliente y la visión del producto, representa a todos los interesados en el producto final. Es responsable de obtener los resultados de mayor valor posible para los usuarios o clientes. Este rol se encuentra distribuido, esto es debido a que la empresa ofrece servicios de telemática a diferentes empresas, en los siguientes involucrados:

- ***Representante de la Empresa:*** se considera necesario para entender el concepto de la adquisición de requisitos, es posible que sean diferentes personas que están en interacción con los servicios que ofrece la empresa, generalmente esta persona representa las necesidades de los usuarios finales, este rol representa al cliente de la empresa.
- ***Consultoría:*** *Cliente directo* para el área de desarrollo de software, área encargada del proceso de adquisición de requerimientos, donde se recolectan todos los datos necesarios que ayuden a identificar y resolver mejor el problema presentando por *El Cliente*. Identifica las peticiones y las clasifica en dos categorías: **Hardware:** Requerimiento o *Bug* y **Software:** Requerimiento o Corrección de error.

***Development Team*:** Responsable del desarrollo del producto, equipo multidisciplinar que cubre todas las habilidades necesarias para generar el resultado esperado, se auto-gestiona y auto-organiza, y dispone de atribuciones suficientes para tomar decisiones sobre cómo realizar su trabajo. Habitualmente el equipo es conformado por 4±1 desarrolladores incluyendo al líder del proyecto, cada integrante es responsable de las actividades que se asignan en las reuniones de planificación del *Sprint*, el registro de su progreso apoyándose con una herramienta *JIRA*, así como la construcción del programa o modulo para cumplir con el objetivo de sus actividades.

***Scrum Master*:** Es el responsable de garantizar el funcionamiento de los procesos y metodología que emplea la organización, algunos autores consideran posible que esta función sea realizada por varias personas dependiendo de las prácticas de trabajo de la organización. Este rol es llevado a cabo por el Líder el Proyecto, en algunas ocasiones se apoya de un integrante del equipo si la carga de trabajo es demasiada.

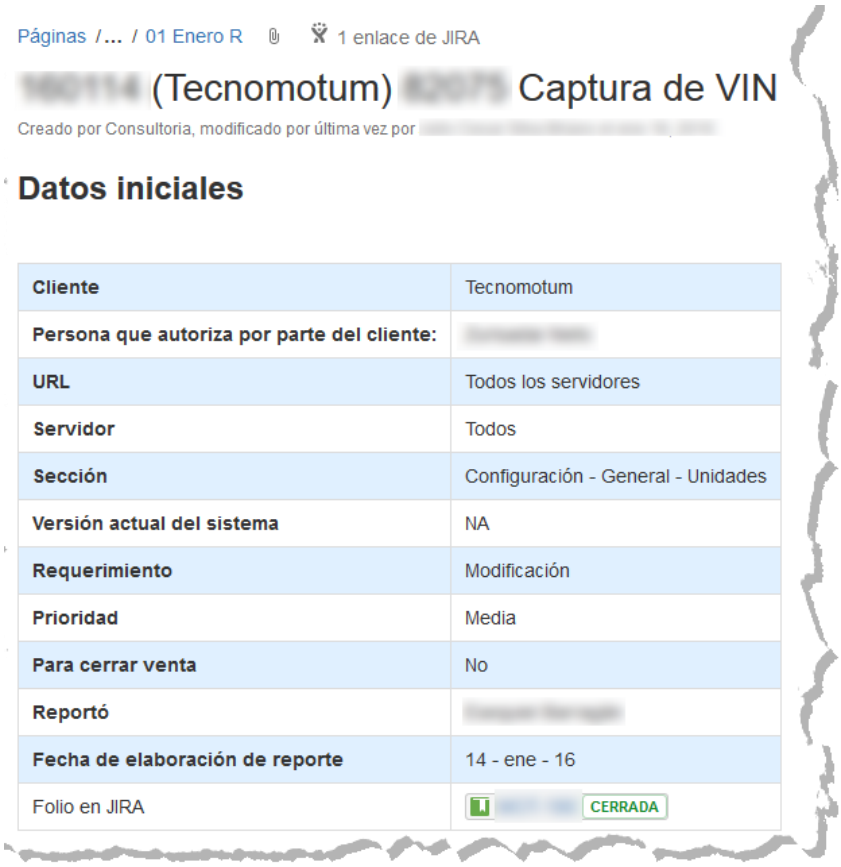
3.2.3. Obtención de requisitos del cliente

La adquisición de las necesidades es responsabilidad de Consultaría y Centro de Soporte, estas son las áreas encargadas de interactuar con el Cliente con el propósito de definir el problema que presenta el software y, si es posible, proponer una posible solución. Para esta actividad se utiliza la siguiente clasificación: ***Hardware*** y ***Software*** (*requerimiento* o *bug*).

El Cliente informa a cualquiera de las áreas antes mencionadas el problema que presenta el Sistema, estableciendo una comunicación se obtienen los datos que ayuden a definir claramente el problema. Una vez recolectado la información se identifica si el problema o necesidad solicitada es para el área de ***Ingeniería de Hardware*** o ***Ingeniería de Software*** para solucionarla.

Las peticiones se registran en *Confluence* siguiendo una plantilla para cada tipo de requerimiento, en la *Figura 3.5* se muestra un fragmento de un requerimiento asignado al área de Ingeniería de Software. Antes de enviarlas y asignarlas al líder de proyecto, el Analista tiene que verificar que los datos sean completos y claros para resolver el problema. La ***incidencia*** se define como cualquier actividad que una persona tiene que realizar (*no está relacionado con la palabra de error únicamente*).

Existen varios tipos de incidencias en *JIRA*, pero los más utilizados son **Épica:** Incidencia para una historia de usuario grande que tiene que desglosarse, **Historia:** Incidencia para una historia de usuario, **Tarea:** una simple tarea que necesita realizarse. *JIRA* presenta un formato parecido a una tarjeta historia de usuario (*Figura 3.6*) lo que proporciona una vista familiar al usuario que está inmerso en terrenos ágiles y familiar para aquellos que son nuevos en estos terrenos. Una vez en este punto, la incidencia pasa por una serie de análisis y autorizaciones por parte de la Gerencia para su desarrollo. En la ayuda proporciona por *JIRA* se muestran todos lo tipo de incidencias que maneja, consulte la referencia [23].



Páginas / ... / 01 Enero R 1 enlace de JIRA

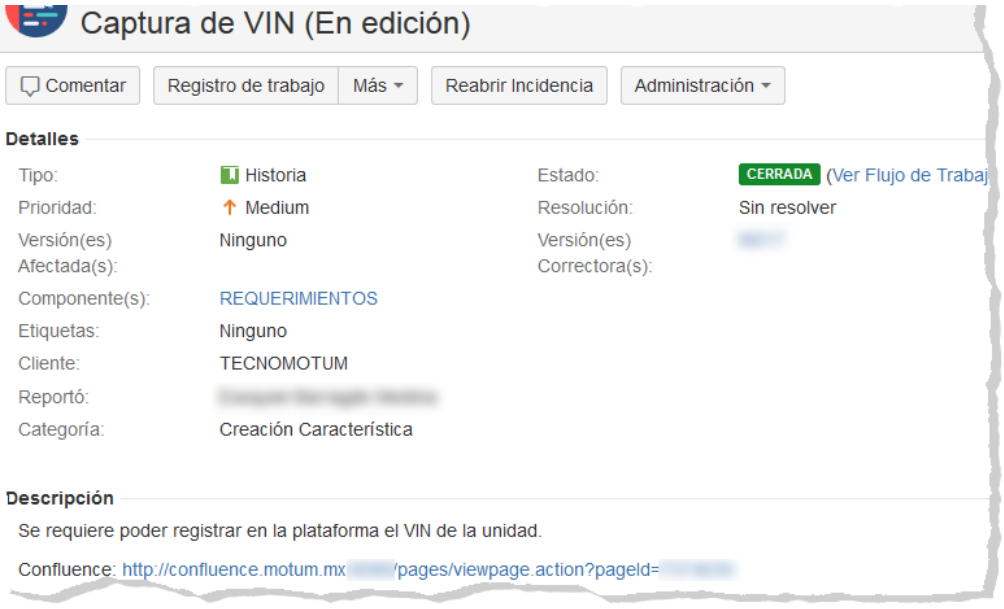
[redacted] (Tecnomotum) [redacted] Captura de VIN

Creado por Consultoria, modificado por última vez por [redacted]

Datos iniciales

Cliente	Tecnomotum
Persona que autoriza por parte del cliente:	[redacted]
URL	Todos los servidores
Servidor	Todos
Sección	Configuración - General - Unidades
Versión actual del sistema	NA
Requerimiento	Modificación
Prioridad	Media
Para cerrar venta	No
Reportó	[redacted]
Fecha de elaboración de reporte	14 - ene - 16
Folio en JIRA	[redacted] CERRADA

Figura 3.5 Requerimiento en Confluence (Fragmento)



Captura de VIN (En edición)

Comentar | Registro de trabajo | Más | Reabrir Incidencia | Administración

Detalles

Tipo:	Historia	Estado:	CERRADA (Ver Flujo de Trabaj
Prioridad:	Medium	Resolución:	Sin resolver
Versión(es) Afectada(s):	Ninguno	Versión(es) Correctora(s):	[redacted]
Componente(s):	REQUERIMIENTOS		
Etiquetas:	Ninguno		
Cliente:	TECNOMOTUM		
Reportó:	[redacted]		
Categoría:	Creación Característica		

Descripción

Se requiere poder registrar en la plataforma el VIN de la unidad.

Confluence: http://confluence.motum.mx[redacted]/pages/viewpage.action?pageId=[redacted]

Figura 3.6 Incidencia tipo Historia en JIRA (Fragmento)

3.2.4. Backlog en *JIRA*

3.2.4.1. Product Backlog

Una vez que se tenga la aprobación para realizar el requerimiento, el siguiente paso es agregarlo al *Product Backlog*. Normalmente se dividen en objetivos expresados como historia de usuario cada una es independiente y aporta valor de negocios incremental, aunque no hace falta utilizar formatos estrictos cualquier sintaxis donde la acción sea clara y el beneficio buscado se atendido por todos en el equipo es suficiente.

La terminología que *JIRA* utiliza en este punto es algo confusa, especialmente si se está inmerso en el mundo ágil o en *Scrum*. Por esta razón se muestra una relación entre la terminología de *JIRA* y *Scrum*:

- ***Backlog*** como lo llama *JIRA* se refiere al **Product Backlog** de *Scrum*.
- Una incidencia, comúnmente de tipo **Historia,** son **historias de usuario** en *Scrum*.

En el *Backlog* se visualizan todas las incidencias con excepción las incidencias de tipo *Sub-tarea*. Por ejemplo, las incidencias de tipo **Épicas** son historias de usuario que por su gran tamaño es necesario dividirlas en tareas con un menor tamaño con el fin de gestionarlas ágilmente; sigue el mismo formato que una historia de usuario, pero su alcance es mayor. Estas incidencias se visualizan en panel a la izquierda del *Backlog* y agrupan incidencias de tipo **Historia**, en la *Figura 3.7* se muestra un ejemplo de esta distribución.

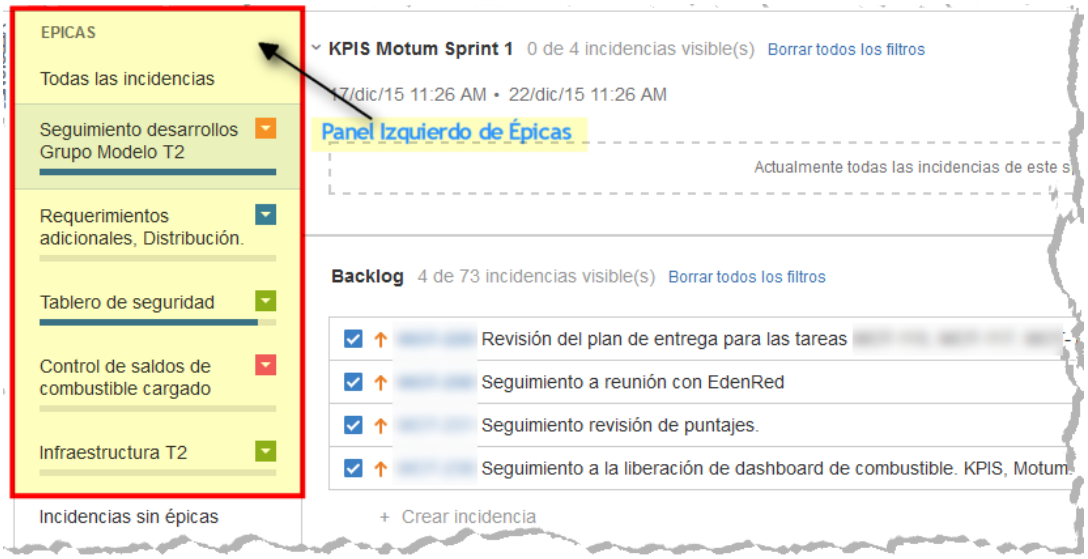


Figura 3.7 Distribución de Epicas en Backlog (Fragmento)

La estructura del *Backlog* para las incidencias de tipo **Historia**, **Tarea y Error** se visualizan en la parte central como se observa en la *Figura 3.8*. Las incidencias de tipo **Sub-tarea** no se muestran en el *Backlog* debido a que su jerarquía es menor y solo se visualizan accediendo a la ficha de la incidencia que las contiene.

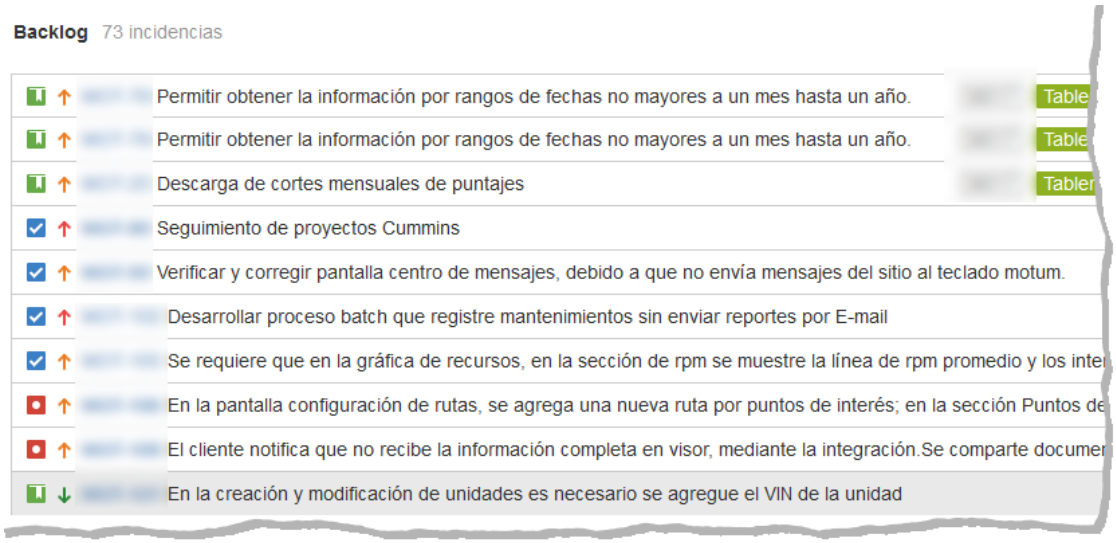


Figura 3.8 Backlog en JIRA (Fragmento)

3.2.4.2. *Sprint* Backlog

Como siguiente paso se crea el *Sprint Backlog*, este elemento es la lista de tareas que descompone las funcionalidades del Producto Backlog en las tareas necesarias para construir un incremento. Esta lista se elabora en una reunión de panificación (*Sprint Planning*) como plan para completar los objetivos/requisitos seleccionados para la iteración, permite ver las tareas donde el equipo está teniendo problemas y no avanza, con lo que permite que se tomen decisiones al respecto.

Dentro de la *JIRA* el *Sprint Backlog* y el *Producto Backlog* comparten el mismo espacio, para crear un *Sprint* solo se tiene que crear el espacio dentro de Backlog, nombrarlo y mover las incidencias que formaran parte del *Sprint* actual, en la *Figura 3.9* se muestra el *Backlog* con un *Sprint* creado.

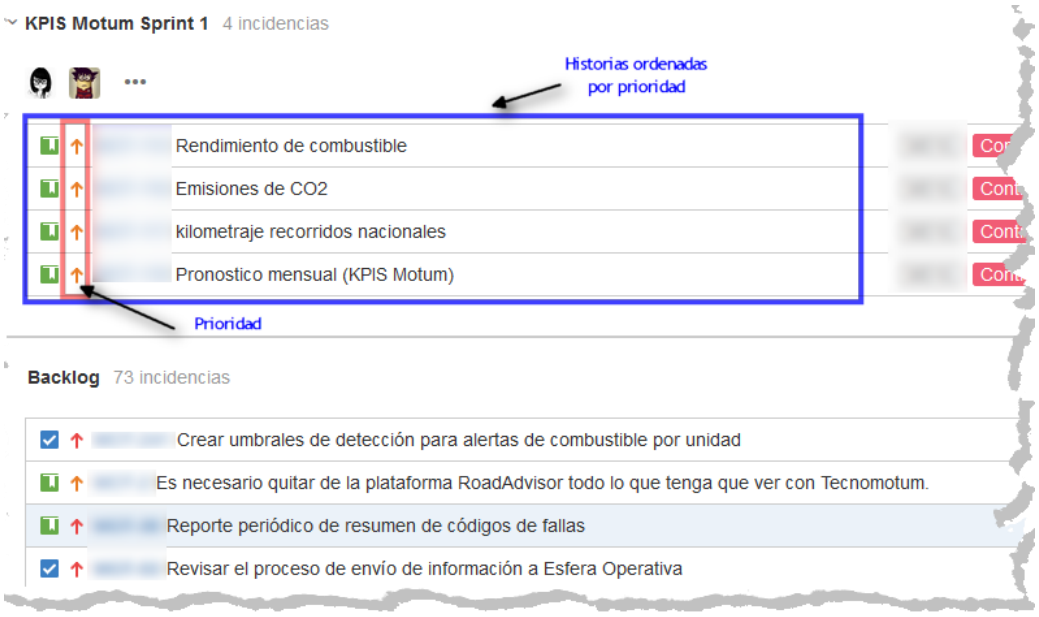


Figura 3.9 Sprint Backlog en JIRA (Fragmento)

Además, *JIRA* ofrece una pizarra virtual (*Figura 3.10*) donde se muestran todas las tareas del *Sprint*, esta pizarra contiene información de la lista de tareas, persona responsable, estado en el que se encuentra y tiempo de trabajo restante para completarla. La pizarra funciona de la misma manera que una pizarra en el mundo real, se tiene

columnas que representan los estados de las actividades e historias de usuario que representa las tarjetas físicas de los requisitos, cuando una tarea pasa por los estados definidos cambia de lugar en la pizarra.

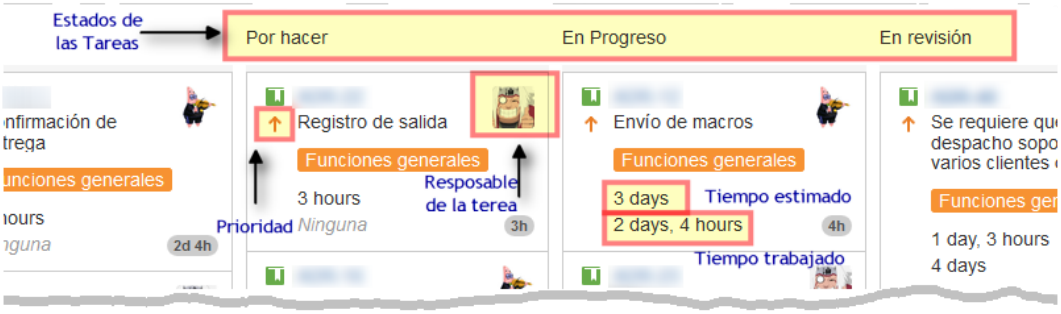


Figura 3.10 Pizarra del Sprint en JIRA (Fragmento)

3.2.5. El incremento

Idealmente en el desarrollo ágil cada funcionalidad del *Backlog* se refiere a características entregables, una historia contemplada en el *Backlog* requiere documentación, procesos de validación y verificación documentados, niveles de independencia que implican procesos con terceros. En la parte operativa, el lanzamiento del incremento se vuelve un poco complejo, debido a que se necesitan autorización de parte de la alta gerencia o datos de suma importancia por parte del cliente que son críticos para terminar una actividad de un desarrollador.

3.2.6. El *Sprint* y las reuniones

El corazón de *Scrum* es el *Sprint*, una iteración que al menos tiene una duración de un mes como máximo, y potencialmente se librera una parte del producto final, funcionando correctamente según los requisitos descritos en el *Product Backlog*. El *Sprint* contiene y consiste de las siguientes reuniones: Planificación del *Sprint* (*Sprint Planning*), Monitorización del *Sprint* (*Daily Scrum*), Revisión del *Sprint* (*Sprint Review*) y la Retrospectiva del *Sprint* (*Sprint Retrospective*).

3.2.6.1. La planificación del *Sprint*

En esta reunión se determinan cuáles son las funcionalidades que se van a incorporar en el próximo *Sprint*, el *Scrum Manager* se asegura que el evento se lleve a cabo, que los asistentes comprendan su propósito y enseña al equipo a mantenerlo dentro del tiempo estimado. Antes de cada reunión el equipo realiza una lista de requerimientos y, según su criterio, determinan si es necesario desglosarlas. La planificación se realiza en una sola reunión dividida en dos sesiones de alrededor de 3 horas aproximadamente, si durante este proceso se encuentran tareas que necesiten dividirse se realiza y se estiman individualmente.

Al iniciar la reunión de planificación se tiene que contar con el *Product Backlog*, la estimación del esfuerzo se realiza por cada *Historia de usuario*, revisando generalidades de funcionalidad y realizando anotaciones o comentarios en el espacio de trabajo para el proyecto proporcionado por Confluence. En la *Figura 3.11* se muestra un fragmento de este espacio y en el panel izquierdo se tiene un árbol de todas las páginas con las que cuenta.

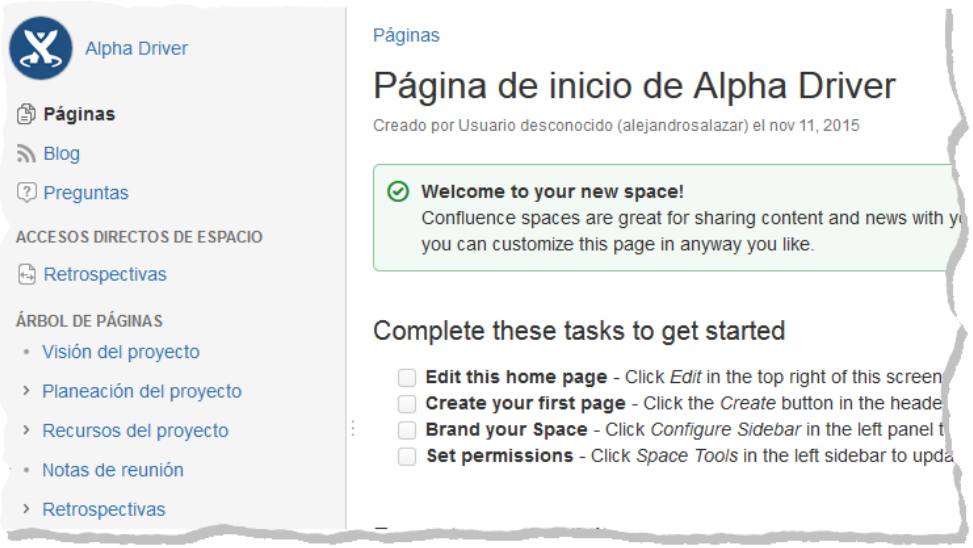


Figura 3.11 Espacio de trabajo en Confluence (Fragmento)

La estimación se realiza utilizando un protocolo llamado Estimación Póquer (*Póker Planning*), que resulta útil ya que el equipo realiza dicha estimación con un criterio de "juicio de expertos", cada participante dispone de 8 cartas con números necesarios para

representar el esfuerzo de cada tarea. En resumen, el proceso de estimación es sencillo, cada integrante elije una carta que representa el esfuerzo de la Historia en base a las características que se proporcionan durante el análisis de los requerimientos, se tiene la oportunidad de hacer preguntas para despejar las dudas. Simultáneamente todos muestran la tarjeta elegida cada persona ofrece su justificación para la estimación seleccionada y la discusión continua hasta que se llegue a un consenso.

3.2.6.2. Monitorización del *Sprint*

Es una reunión diaria breve en la que todos los miembros platican las tareas en las que se encuentran trabajando, si encontraron o prevén encontrarse con algún impedimento y actualizan sobre el *Sprint Backlog* las tareas ya terminadas o los tiempos de trabajo que les quedan. Se utiliza una sala de juntas para esta reunión, aunque debido otras actividades que desempeña cada integrante del equipo no sé cuenta con la disponibilidad necesaria, en estos casos se utilizan herramientas alternas como el *chat, email* y Confluence se utilizan para dar a conocer las conclusiones de la revisión. Las herramientas de gestión ofrecen una comunicación rápida y un panorama amplio del *Sprint* actual, proporcionando una lista de tareas en su estado actual, con el esfuerzo pendiente para cada tarea, así como los gráficos de horas que se llevan y restan para cada Historia.

3.2.6.3. Retrospectiva del *Sprint*

Es una oportunidad para que el equipo desarrollador inspeccione y cree un plan con el objetivo de mejorar de manera continua su productividad y la calidad del producto, se produce después del cierre de un *Sprint*. En esta reunión, el *Scrum Master* y el Equipo se aseguran que el evento se lleve a cabo dentro de los márgenes de tiempo disponibles para cada integrante, el propósito de la retrospectiva es revisar el estado del último *Sprint* respecto a las personas, relaciones de procesos y herramientas, se identifican los elementos principales que salieron bien y las mejoras potenciales, además de crear un plan para mejorar el trabajo en el siguiente *Sprint*.

Para esta retrospectiva se tiene un espacio de trabajo para cada proyecto utilizado dentro de Confluence, en ella se anotan todos los problemas, contratiempos y observaciones que surgieron durante el *Sprint*, en la *Figura 3.12* se muestra un fragmento de una retrospectiva, con datos como fecha, participantes, numero del *Sprint*. También se establece comparativa entre lo realizado de manera correcta contra las acciones que se necesitan mejorar.

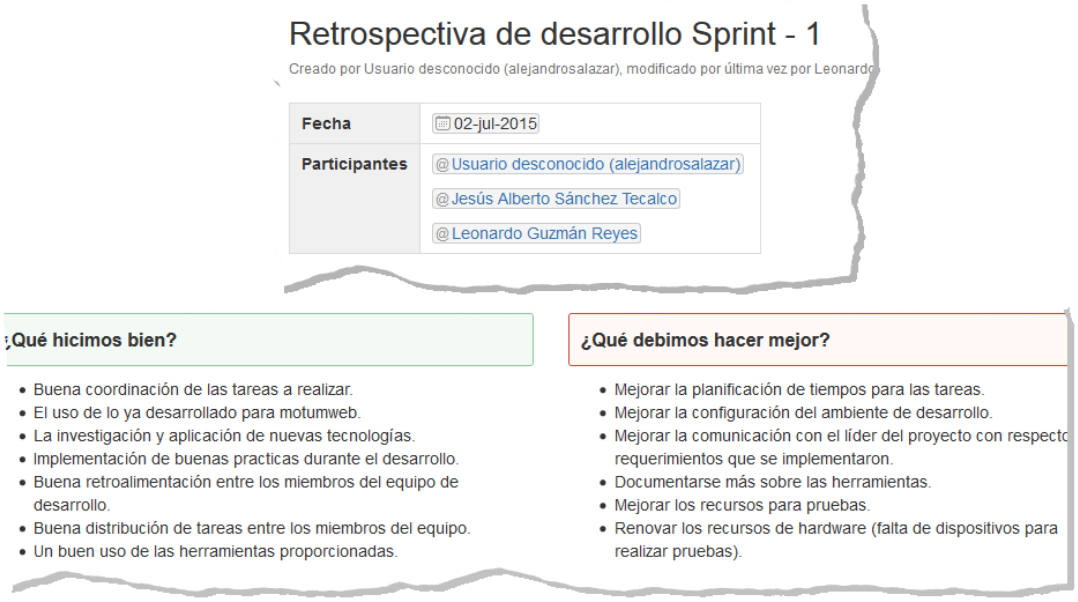


Figura 3.12 Retrospectiva en Confluence (Fragmento)

3.3. Análisis de características entre *Scrum* y *PSP*

Este análisis tiene como propósito generar una comparación entre las características de ambos enfoques para determinar las fortalezas y debilidades, y realizar una mejor integración para mejorar el proceso de desarrollo del software (*Tabla 3.1*).

Tabla 3.1 Análisis comparativo entre características de Scrum y PSP

Factores	*Scrum*	*PSP*
Tamaño de Equipo	Equipo de trabajo pequeño (7 ±2).	Se enfoca en el trabajo personal.
Enfoque	Agilidad y adaptación a cambios. Producción rápida.	Mejorar de habilidades personales del desarrollador. Estimación basada en datos históricos. Predictibilidad.
Conocimiento	Empírico y Tácito.	Teórico, basado en registros.
Prácticas en el Desarrollo	Gestión del proceso centrado en los requerimientos del cliente. Priorización de los requerimientos y estimación del tiempo. Descomposición de tareas.	Establece un flujo de trabajo para el ingeniero de manera personal. Define guías (scripts) para la administración del proceso. Centrado en el Proceso.
Mecanismo de Operación	Iterativo e incremental. Tres fases: Pre-juego, Juego y Post-juego. Planeación de *Sprint*, Revisión Diaria del *Sprint*, Revisión del *Sprint* y Retrospectiva del *Sprint*.	Iterativo y escalonad. Tres principales procesos: Planificación, Desarrollo y Postmortem. Revisiones de Diseño y Código. Enfocado a la Introducción y Eliminación de Defectos.
Ambiente de Trabajo	Cambiante y rápido.	Estable, pocas modificaciones.
Tipo de Proyecto	Para proyectos cortos y medianos. Poca necesidad de documentación.	Para cualquier tipo de proyectos. Se enfoca en el trabajo personal.
Calidad	Implícita, a través de la definición de *Terminado*.	Explicita, la define y administra cuantitativamente.
Gestión de Riesgos	No definido.	Definido y administrado por el desarrollador.
Cultura	Colaboración en equipo. Permite una mejora continua para todos los integrantes del equipo. Equipos auto-dirigidos.	Disciplina y el respeto al proceso de trabajo. Mejora continua de las habilidades personales del desarrollador. Proceso auto-dirigido.

Las principales características de *Scrum* son: su proceso iterativo e incremental, adaptación a los cambios y la estimación del esfuerzo de las actividades; características consideradas puntos críticos para formar parte del esqueleto de la integración. Sin embargo, *Scrum* no proporciona un apoyo a la calidad y gestión de riesgos de la forma que *PSP* la aborda. *Scrum* y *PSP* tienen similitudes en cuento a la mejora continua por parte de su enfoque y cultura, además del tamaño de equipo y

proyectos, el proceso que siguen para desarrollar las actividades de codificación es flexible gracias al proceso iterativo de ambos.

3.4. Consideraciones para adaptar Scrum y PSP

Para realizar una combinación sólida y exitosa entre ambos enfoques de trabajo se tiene que tomar en cuenta ciertas reglas constantes por parte de *Scrum* y *PSP*. *Scrum* considera las siguientes "*reglas*" para una buena implementación de su proceso:

- **Una gestión regular de las expectativas del cliente**, indicando el valor que aporta cada requisito del proyecto y cuando espera que esté completo. El cliente comprueba de manera regular si se están cumpliendo estas expectativas.
- **Flexibilidad y adaptación**, de manera regular el cliente redirige el proyecto en función de nuevas prioridades, de los cambios en el mercado, de la velocidad del desarrollo, entre otros. Al final de cada iteración el cliente aprovecha la parte del producto completada hasta ese momento para hacer pruebas y tomar decisiones en función del resultado.
- **Alineamiento entre cliente y equipo**, los resultados y esfuerzos del proyecto se miden en forma de objetivo y requisitos entregados al negocio. Todos los participantes en el proyecto conocen cual es el objetivo a conseguir.
- **Equipo multidisciplinar**, formado por todas las personas con las especialidades necesarias para llevar a cabo el proyecto.
- **Estimación de esfuerzo conjunta**, al inicio de cada iteración los miembros del equipo estiman de manera conjunta el esfuerzo necesario para completar requisitos y tareas.

Por otro lado, para tener una base sólida existen 5 expectativas de *PSP* se tiene que cumplir en cualquier implementación mencionadas por Humphrey:

- **Definir las métricas y/o características de calidad** que se aplican al proyecto y su producto.
- **La recolección de los datos de todas las actividades** realizadas en los proyectos para generar la base de conocimientos que permitirá hacer las estimaciones a futuro.

- Es obligatorio que **todos los integrantes del equipo dispongan de los recursos** con libertad y facilidad que les ayuden a realizar sus actividades.
- Cada integrante del equipo tiene que **seguir el flujo del proceso con disciplina**.
- **Realizar la medición del producto final** y por ende que cumpla con la meta establecida.

PSP define 7 niveles de madurez a lo largo de su ciclo de vida, como se mostró en capítulos anteriores (*Figura 1.3*) los primeros dos niveles *PSP*0 y *PSP*0.1 son críticos para la implementación de la disciplina y son el punto de partida para el proceso personal. Para esta integración se toma como base el flujo de trabajo del nivel *PSP*0, con un ajuste en las actividades de complicación y pruebas que dependen de los lenguajes de programación relacionados con el entorno de la actividad.

Por último, se eligió *MoProSof*t como norma de calidad para el proceso de desarrollo centrado principalmente en la categoría de Operación, realizando adaptaciones para las sub-categorías: Administración de Proyectos Específicos y Desarrollo, y Mantenimiento de Software.

Una vez seleccionado las características para la integración de *Scrum*, *PSP* y *MoProSof*t, se procede con el diseño del nuevo método el cual se define en dos capas: la primera define el ciclo de vida de *Scrum–PSP* y la segunda contiene las iteraciones de *PSP* que cada desarrollador del equipo necesita para realizar sus actividades.

3.5. Mixing *Scrum-PSP*: Agilidad y Disciplina trabajando en equipo

En este apartado se presenta, a forma de resumen, la propuesta del modelo resultante de la combinación de los conceptos de *Scrum* y *PSP*. Para que este modelo tenga una base sólida se agregó ciertas tareas y documentos de la norma *MoProSoft*, así como ciertos roles fueron ajustados a *Scrum* para no perder agilidad sin olvidar la disciplina que marca *PSP*.

Mixing Scrum-PSP (MSP) es la combinación de dos enfoques para el desarrollo de proyectos de software, con el principal objetivo de combinar la capacidad de adaptación con la disciplina para mejorar el proceso de desarrollo de proyectos de software. MSP consiste en una serie de prácticas

combinadas entre *Scrum* y *PSP* para guiar el trabajo que se lleva en el desarrollo de un producto de software.

MSP está pensado para PyME's donde las organizaciones tienen una plantilla laborar no mayor a los 100 empleados aproximadamente, y equipos de trabajo son formados por 5 a 7 integrantes en promedio. Para generar un modelo sólido, MSP adapta algunas actividades establecidas en la norma mexicana ***MoProSoft*** dentro de la categoría de **Operación** (OP) en los procesos de **Administración de Proyectos Específicos** (APE) y **Desarrollo y Mantenimiento de Software** (DMS), utilizando los niveles de **Realizado** y **Gestionado**.

3.5.1. Roles en Mixing *Scrum-PSP*

En cada etapa están definidos los roles responsables para la ejecución de las prácticas, los roles se asignan al personal de acuerdo a sus habilidades y capacitación. Los roles del proceso MSP se muestran agrupados en la *Tabla 3.2*.

Tabla 3.2 Compaginación de Roles (Scrum-PSP-MoProsoft)

Rol	Abreviatura	Capacitación / Conocimiento
Responsable de Gestión de Proyectos	RGPY	Conocimiento de las actividades necesarias para llevar a cabo la gestión de proyectos.
Responsable de la Administración del Proyecto Específico	RAPE	Capacidad de liderazgo con experiencia en la toma de decisiones, planificación estratégica, manejo de personal, delegación y supervisión, gestión de recursos y desarrollo de software.
Responsable del Desarrollo y Mantenimiento de Software	RDMS	Conocimiento y experiencia en el desarrollo y mantenimiento de software.
Equipo Trabajo	ET	Conocimiento y experiencia de acuerdo a su rol.
Desarrollador	DS	Conocimiento y/o experiencia en las fases de planificación, análisis, diseño, codificación, y pruebas unitarias.
Scrum Master	SM	Capacidad para liderar y enseñar el marco de trabajo de *Scrum* a la organización y todos los participantes, servicio y facilitación para la realización de los eventos de *Scrum*.
Product Owner	POW	Conocimiento en el negocio del cliente y la visión del producto.
Analista	ANL	Conocimiento y/o experiencia en la obtención, especificación y análisis de los requisitos.

Rol	Abreviatura	Capacitación / Conocimiento
Responsable de Calidad	RC	Conocimiento y/o experiencia en estándares de calidad, técnicas de revisión y desarrollo de software.
Responsable de Pruebas	RPR	Conocimiento y experiencia en la planificación y realización de pruebas de integración y de sistema.

Además, el proceso contempla roles auxiliares de tal forma que la falta de éstos no afecte la adopción de MSP. En la *Tabla 3.3* se muestran dichos roles con su capacitación y/o conocimiento.

Tabla 3.3 Roles Auxiliares (Scrum-PSP-MoProsoft)

Rol Auxiliar	Abreviatura	Capacitación/Conocimiento
Arquitecto	AR	Conocimiento de la plataforma tecnológica objetivo, conocimiento de los recursos existentes a ser reutilizados, visión global del negocio y de las soluciones de arquitectura que garantizan la evolución del sistema.
Diseñador de Interfaz Usuario	DIU	Conocimiento en el diseño de la estructura de los componentes de software, interfaces de usuario y criterios ergonómicos.
Responsable de Manuales	RMA	Conocimiento en las técnicas de redacción y experiencia en el desarrollo de software.
Soporte Técnico	ST	Conocimiento de la plataforma objetivo y de los lineamientos existentes en la empresa cliente para el despliegue de componentes y la operación de sistema.

3.5.2. Estructura del Modelo

MSP proporcionar un modelo de procesos iterativos, incrementales y escalonados, es decir, las prácticas se introducen una a la vez dependiendo de la capacidad de la(s) persona(s) que lo utilizan. Compuesto por dos capas de procesos, la primera llamada ***Ciclo de Vida MSP*** define la secuencia de las etapas y flujo de actividades que se tienen que realizar para la gestión del proyecto. La segunda capa es ***Iteración de Desarrollo***, donde se define el flujo de actividades a nivel personal por medio de las guías proporcionadas por *PSP*. En la *Figura 3.13* se muestra de manera resumida el modelo *Mixing Scrum-PSP*.

3.5.2.1. Ciclo de vida MPS

Al igual que *Scrum*, este modelo define 3 fases para la administración de un proyecto con la diferencia de proporcionar guías para la fase de desarrollo y una nueva integración de actividades para gestionar la calidad del proceso. Las fases son: **Preparación**, **Desarrollo** y **Entrega**. Cada fase proporciona una serie de actividades que se atienden de manera puntual y ordenada, cada una define *sub-actividades* sencillas a realizar por todos los involucrados en el proyecto de software. Cada actividad menciona un producto de entrada y genera un producto de salida.

Primera Etapa: *Preparación*

Fase de definiciones por parte del equipo de trabajo, en donde se detalla la *Descripción del proyecto*; además se realiza la elaboración de calendarios y se estable un protocolo de entrega. Por lo general durante esta etapa se producen gran número de inexactitudes con las estimaciones, pues se realizan a "juicio de expertos". Conforme se apliquen las técnicas que define la iteración de trabajo PSP las estimaciones serán más exactas gracias a registros históricos de cada actividad.

A continuación, se describen de manera resumida las principales actividades del flujo de trabajo de la fase de ***preparación*** mostrado en la *Figura 3.14*; en la sección de ***Anexos*** es posible apreciar el detalle de la secuencia de trabajo.

A1. Revisar la «Descripción del proyecto». El equipo de trabajo realizar una reunión para conocer y elaborar las necesidades del negocio, objetivos, alcance, descripción del producto y entregables, así como sus restricciones y limitantes. Con cada iteración la «Descripción del proyecto» será actualizada de acuerdo a las necesidades del *Product Owner* y del *Sistema de Software.* No se exige un nivel de detalle para la descripción, el contexto práctico determinará si es general o detallado.

A2. Ingeniería de requisitos. Conjunto de tareas que permiten una definición y gestión de los requisitos del proyecto de manera sistemática. Se considerar dentro de esta etapa el proceso de comunicación entre los clientes, usuarios finales y desarrolladores del software. Durante las siguientes iteraciones de trabajo se presentan las solicitudes de cambios por parte del *Product Owner*, toda petición de cambio debe se analiza, definida y detallada para conocer su factibilidad e impacto en el desarrollo y sistema.

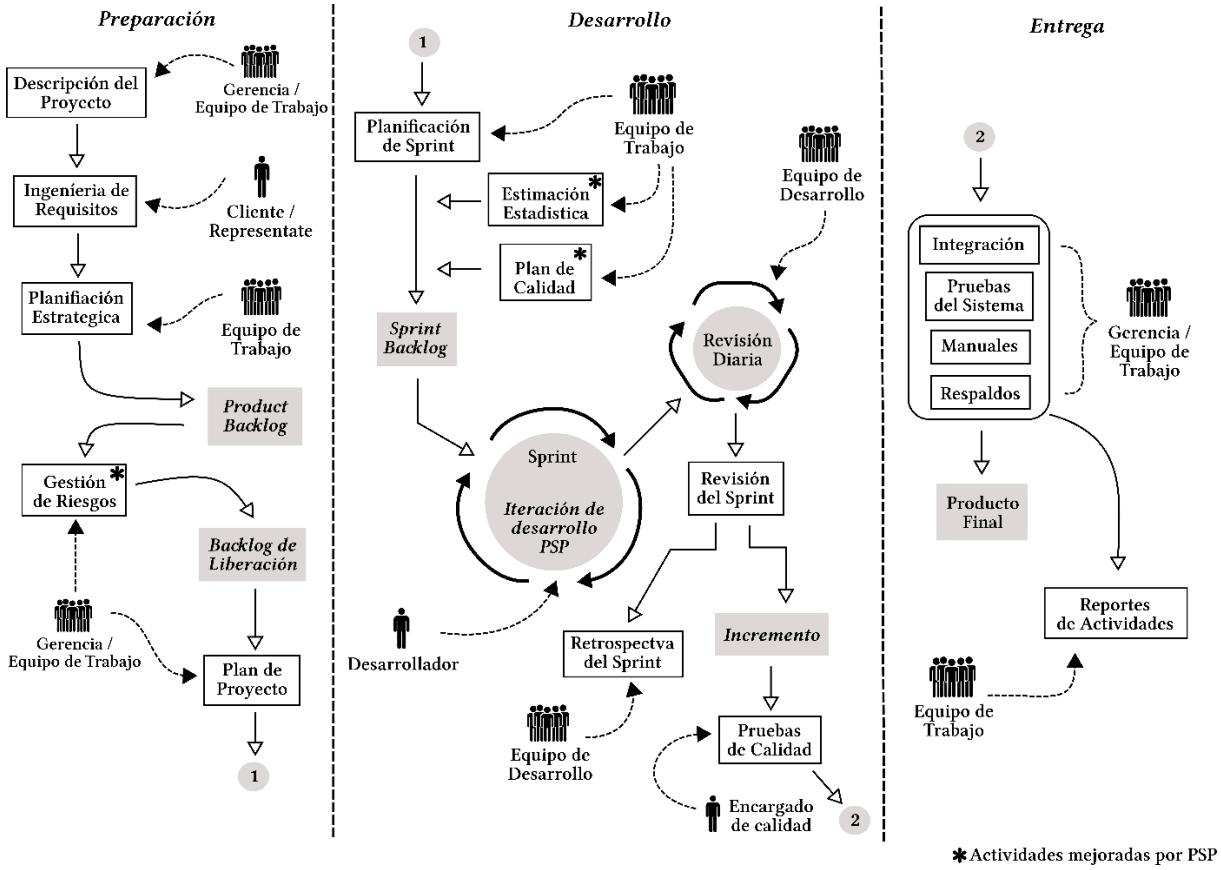


Figura 3.13 Modelo Mixing Scrum-PSP

A3. Planificación estratégica de recursos. Los procesos de planificación ayudan a identificar el tipo y las cantidades de materiales, recursos humanos y tecnología para ejecutar cada requerimiento del proyecto.

A4. Gestión de riesgos. Incluye los procesos para llevar a cabo la gestión y control de riesgos para reducir la probabilidad y el impacto de los eventos negativos, y aumentar

la probabilidad e impacto de eventos positivos. Al finalizar esta actividad se genera la información necesaria para la sección «Plan de manejo de riesgos» del documento «Plan del Proyecto y Desarrollo».

A5. Priorización de requisitos. Una vez completada la gestión de riesgos se tiene que generar el «*Backlog de Liberación*» donde se contemplan los requisitos más importantes para generar la primera versión del software. Utilizar el «*Calendario de trabajo*», «*Product Backlog*» y «Plan de manejo de riesgos» para tener un mejor panorama a la hora de seleccionar los requisitos.

A6. Generar/Actualizar el «Plan del proyecto y desarrollo». Aquellas descripciones elaboradas en las actividades anteriores se integran parar formar el documento llamado «Plan de proyecto y desarrollo».

Segunda Etapa: *Fase de Desarrollo*

En esta fase se desarrollan las actividades planificadas del «*Backlog de Liberación*», compagina flujo de trabajo y actividades del *Sprint* junto con actividades de *MoProSoft*, para mejorar el proceso de desarrollo del sistema. Además de construir los componentes necesarios del sistema, se realizan diferentes revisiones a las actividades efectuadas por el *Equipo de Trabajo*. Gracias a la adaptación con *PSP* los desarrolladores tienen guías para desempeñar mejor sus actividades dentro del *Sprint*.

A continuación, se describe de manera resumida las actividades que componen a esta etapa, y en la *Figura 3.15* se muestra el flujo de trabajo, dirigirse a la sección de ***Anexos*** para apreciar el detalle de la secuencia de trabajo:

A7. Planificación del *Sprint.* Tiene como finalidad realizar una reunión en la que participan el *Product Owner*, el *Scrum Master* y el Equipo de desarrollo, con la intención de organizar y priorizar los requerimientos que se van a trabajar.

A8. Gestionar pruebas del sistema. La gestión de pruebas es un proceso que proporciona información sobre el correcto funcionamiento del software, se enfoca en la lógica interna y las funciones externas del software. Entre sus objetivos se encuentra: la

detección de defectos, integración adecuada de los componentes, la implementación correcta de los requisitos, y la corrección de los defectos encontrados.

A9. Generar/actualizar los manuales del software. Actividad donde se elaboran los manuales de usuario, operación y mantenimiento; en caso de que ya existan se tiene que actualizar. Esta actividad será realizada durante todo el *Sprint* y la *fase de entrega*.

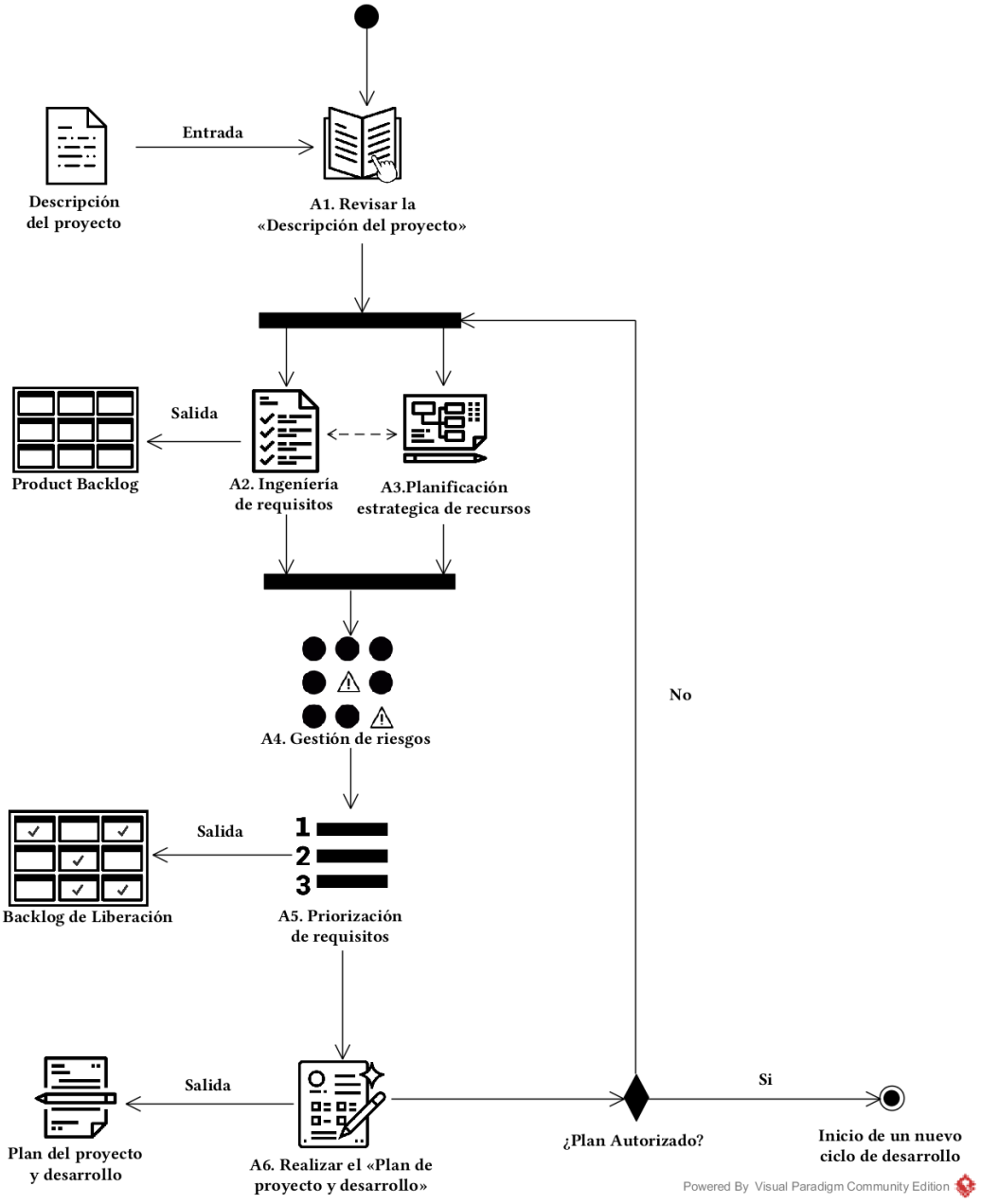


Figura 3.14 Fase de Preparación

A10. Iteración de trabajo PSP. Conjunto de actividades que cada desarrollador utiliza para construir los programas o módulos correspondientes a las actividades asignadas. El marco de trabajo define tres fases: *Planeación*, *Desarrollo* y *Post-mortem*. El proceso de interacción de ésta actividad es de manera iterativa y escalonada.

A11. Realizar revisiones diarias. Las inspecciones ayudan tener el conocimiento sobre el estado de las actividades y la carga de trabajo de cada integrante del equipo, así como los impedimentos o complicaciones para terminar las actividades en tiempo y forma.

A12. Revisión del *Sprint*. En esta reunión, máximo de 4 horas, se presenta un producto funcional; el *Product Owner* junto con los interesados analizan la entrega y escuchan al equipo de desarrollo el resumen de las actividades realizadas.

A13. Generar Incremento. Producto del *Sprint* que contiene todos los elementos seleccionados de «Sprint Backlog». El nuevo incremento debe de estar "*Terminado*" conforme a las condiciones estipuladas al inicio del proyecto.

A14. Realizar pruebas al «Incremento». Las pruebas permiten evaluar la calidad del incremento, es decir, si el producto generado cumple con los criterios de la empresa, el cliente y la finalidad para la cual fue desarrollado.

A15. Retrospectiva del Sprint. Última reunión del ciclo de desarrollo de *Scrum*, actividad donde el equipo de desarrollo debatirá temas relacionados con actividades realizadas, complicaciones el desarrollo, flujo de trabajo y los cambios que se podrían hacer para el próximo *Sprint*.

A16. Generar/actualizar la «Configuración del Software». Integrar cada la salida de las actividades realizadas anteriormente, en el documento llamado «Configuración del Software».

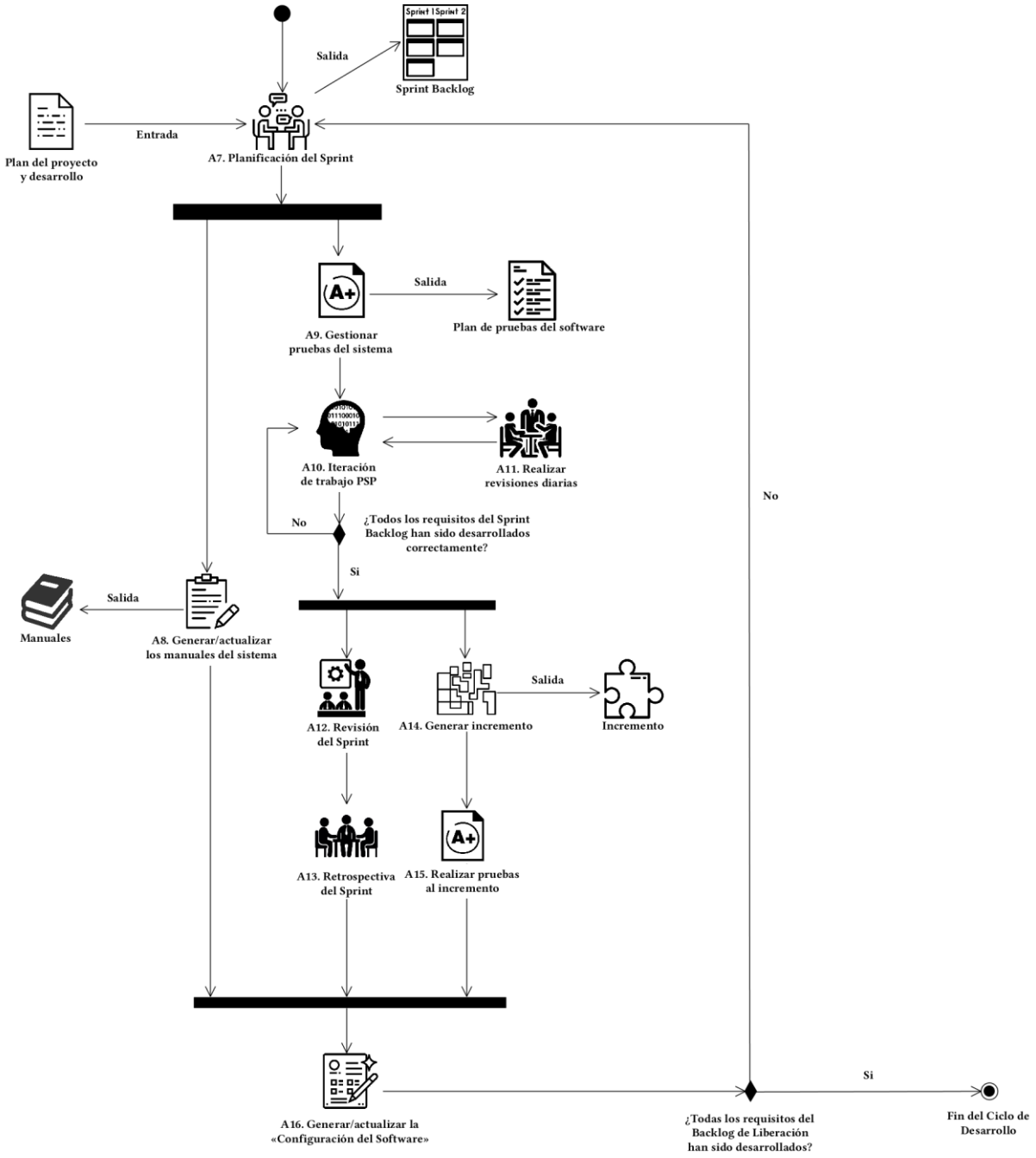


Figura 3.15 Fase de Desarrollo

Tercera Etapa: *Fase de Entrega*

Última fase del ciclo de vida de MSP, en esta se realizan la integración del nuevo incremento con el previamente desarrollado para verificar su funcionamiento correctamente dentro del ambiente del sistema realizando las pruebas necesarias. También se realiza una evaluación conforme al plan del proyecto para determinar la consistencia contra los objetivos establecidos desde el inicio del proyecto. Además, se realiza actualiza los reportes de actividades al desarrollo del proyecto y generara los respaldos de los datos e información en los diferentes repositorios y control de versiones.

En la *figura 3.16.* de muestra el flujo de trabajo y a continuación se describen las principales actividades de esta fase, ver en la sección de ***Anexos*** para conocer mejor el detalle de la secuencia de trabajo:

A17. Integrar nuevo «Incremento». Realizar la integración y comprobar que funcionan conforme la especificación establecida por el *Product Owner*. Además, se evalúa el cumplimiento con respecto al «Plan del Proyecto y Desarrollo».

A18. Realizar pruebas al Software. Las pruebas permiten evaluar la calidad del incremento, es decir, si el producto generado cumple con los criterios de la empresa, el cliente y la finalidad para la cual fue desarrollado.

A19. Conformar los productos para la entrega. En esta actividad se agrupar todos los entregables que fueron escritos en el «Protocolo de entrega».

A20. Presentar los informes de actividades. Se elaboran los informes que se darán a conocer a los diferentes grupos de interesados, el nivel de detalle dependerá el tipo de persona al que se presentaran los reportes.

A21. Realizar los respaldos de información. Realizar el Respaldo del Repositorio de acuerdo a la Estrategia de Control de Versiones.

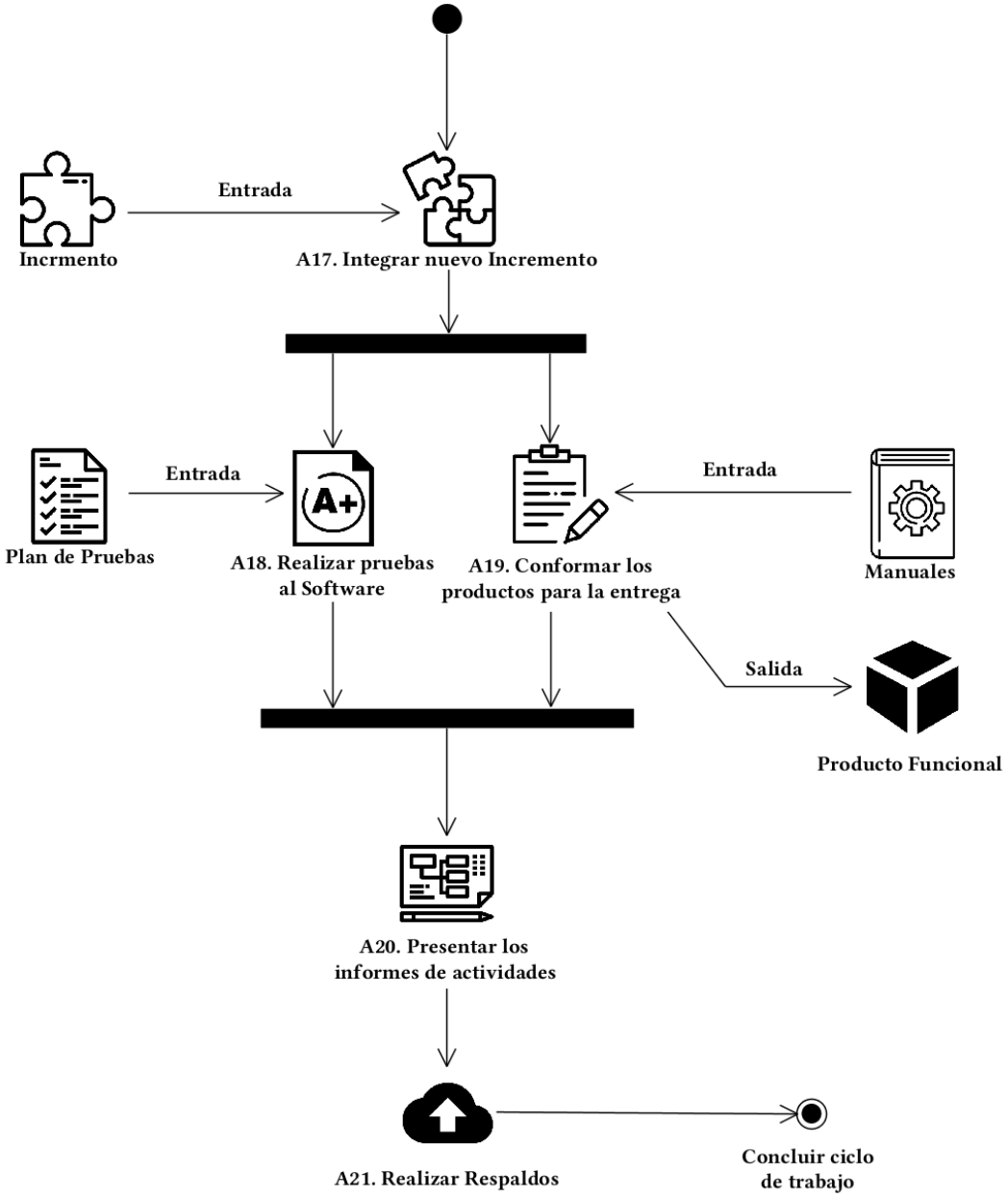


Figura 3.16 Fase de Entrega

3.5.2.2. Iteraciones *PSP*

La segunda capa de este modelo se aplica en el *Sprint*, tiene como actores principales al Equipo de Desarrollo conformado por el *Scrum* Master y los Desarrolladores. Los roles como Analista, Arquitecto, Diseñador de Interfaz de Usuario, Responsable de Manuales, Soporte Técnico sirven como roles de apoyo al Equipo de Desarrollo.

El objetivo de la iteración de trabajo PSP es proporcionar un actividades y técnicas para mejorar las habilidades de los programadores y mejorar los productos (programas, módulos o códigos) que construyen. Una iteración de trabajo consiste en la realización de una tarea definida en el *Sprint Backlog* donde existe la posibilidad de desarrollar uno o varios módulos/programas como parte del objetivo final de la tarea.

Las actividades que define PSP son adaptadas para su integración en equipos de desarrollo que utilizan *Scrum* proporcionando la disciplina suficiente al proceso dependiendo de ciertos criterios como son: tiempo, complejidad, esfuerzo, prioridad e integrantes del equipo. El marco de trabajo *PSP* proporciona una serie de Scripts (Guías) para apoyar al *Desarrollador* a establecer y mejorar su proceso de desarrollo, los Scripts de cada nivel de *PSP* se muestran en el apartado de Anexos y son propiedad del *Software Enginnering Institute* (SEI).

La *Figura 3.17* muestra el flujo de trabajo de la ***Iteración de Trabajo PSP***, así como las actividades a realizar por los desarrolladores. Para la definición de la iteración de trabajo se utiliza las actividades definidas en los niveles 0 y 1.1 A continuación, se describe el concepto de cada actividad que define la iteración de trabajo PSP:

- **Planificación Personal:** Se produce un plan detallado para trabajar el desarrollo del programa definido por los requisitos de la tarea, los formatos para escribir el plan de trabajo no son difíciles, pero requieren toda la atención del desarrollador. El plan consiste en la obtención y definición de los requisitos y recursos que involucran la construcción de los módulos o programas escritos en un documento de manera clara.

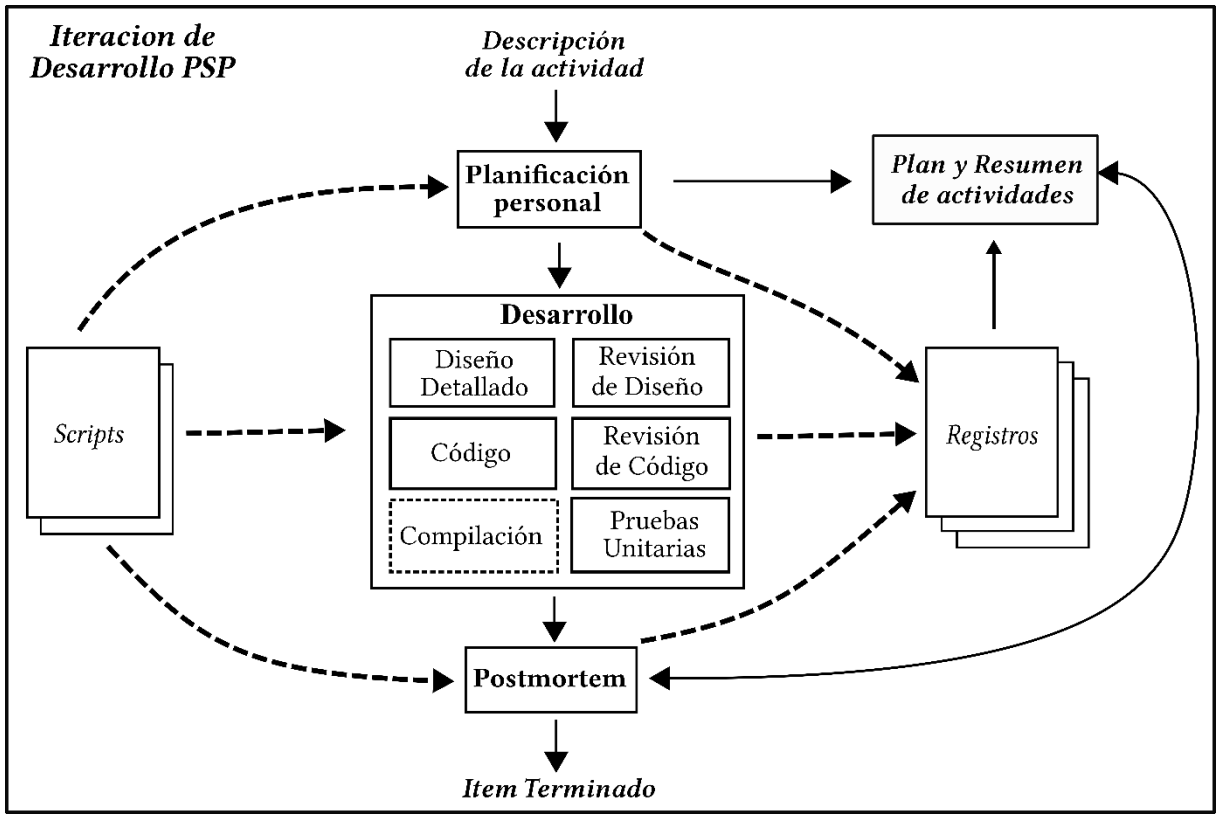


Figura 3.17 Iteración de Desarrollo PSP

- **Diseño Detallado:** Se realiza un diseño, dependiendo de la complejidad de la tarea, para las especificaciones del módulo o programas definidas en los requisitos de la tarea. El diseño es independiente del paradigma de programación y el programador es libre de utilizar las herramientas de modelado que desee.
- **Código:** Se produce la transformación e implementación del diseño a sentencias de lenguajes de programación.
- **Compilación:** Se traducen las sentencias del lenguaje de programación a código ejecutable. La mayoría de los *defectos de sintaxis* serán removidos durante esta fase. Esta fase es *opcional*, determinada por el entorno de desarrollo y el lenguaje de programación.
- **Pruebas Unitarias:** Cada desarrollador realiza pruebas unitarias al módulo o programa para verificar que cumpla con los requisitos establecidos en la planificación, no se establece un número límite para las pruebas, herramientas o

técnicas para realizarlas; sin embargo, se tiene que llevar un registro de las pruebas realizadas.

- **Postmortem:** Puede considerarse una retrospectiva personal para analizar los datos generados por el proceso. Estos datos incluyen valores sobre el tiempo estimado y el tiempo real utilizado, la calidad y productividad del programador. También se asegura la consistencia y la falta de los datos que puedan generar errores en los registros. Además, la información proporcionada por esta fase permite dar bases a las retrospectivas del Sprint.

En el apartado de ***Anexos*** se muestra la secuencia de actividades y sub-actividades que se tiene que realizar en cada nivel de la iteración de *PSP*.

Capítulo 4 Resultados

En el presente capítulo se dan a conocer los resultados obtenidos de la adaptación de la propuesta llamada *Mixing Scrum-PSP*, el cual se generó con el propósito definir un buen proceso para un equipo de desarrollo y como consecuencia de éste se espera mejorar la calidad del proceso de desarrollo del software. Empezando por la parte de la gerencia se tienen reuniones donde se explicaron los fundamentos del modelo y sus principales ventajas, a partir de esto se obtuvo una retroalimentación para ajustar la propuesta anteriormente obtenida. En segundo lugar, se plantea un caso de estudio dentro de un equipo de la empresa *TecnoMotum* con el objetivo de verificar y validar el modelo propuesto.

4.1. Boceto del modelo y aprobación

En conjunto con el personal de la empresa *TecnoMotum* involucrado en el desarrollo de esa investigación se realizó un análisis de los datos e información del proceso que el equipo de desarrollo utilizaba con el objetivo de identificar oportunidades para la implementación de *PSP*.

Derivado del análisis se procedió a realizar varios intentos para generar una propuesta de modelo para combinar *Scrum* y *PSP*, el boceto inicial contemplaba todas las actividades definidas por PSP sin ninguna adaptación al proceso. Este primer borrado se muestra en la *Figura 4.1* el cual presenta las tres fases que define *Scrum* y ciertas actividades principales que se definen a continuación:

- **Fase de pre-juego:** En un principio esta fase contemplaba las actividades tradicionales de inicialización y planeación, las cuales aplican un enfoque ligero para asegurar que se obtengan las propiedades del producto de manera rápida.
- **Fase de desarrollo:** Contemplaba la construcción de la funcionalidad de la nueva versión con respecto a las variables de tiempo, requisitos, costo y competencia. La evolución del sistema a través de múltiples iteraciones de desarrollo.
- **Fase de post-juego:** Preparación para el lanzamiento de la versión, incluyendo la documentación final y pruebas antes del lanzamiento de la versión.

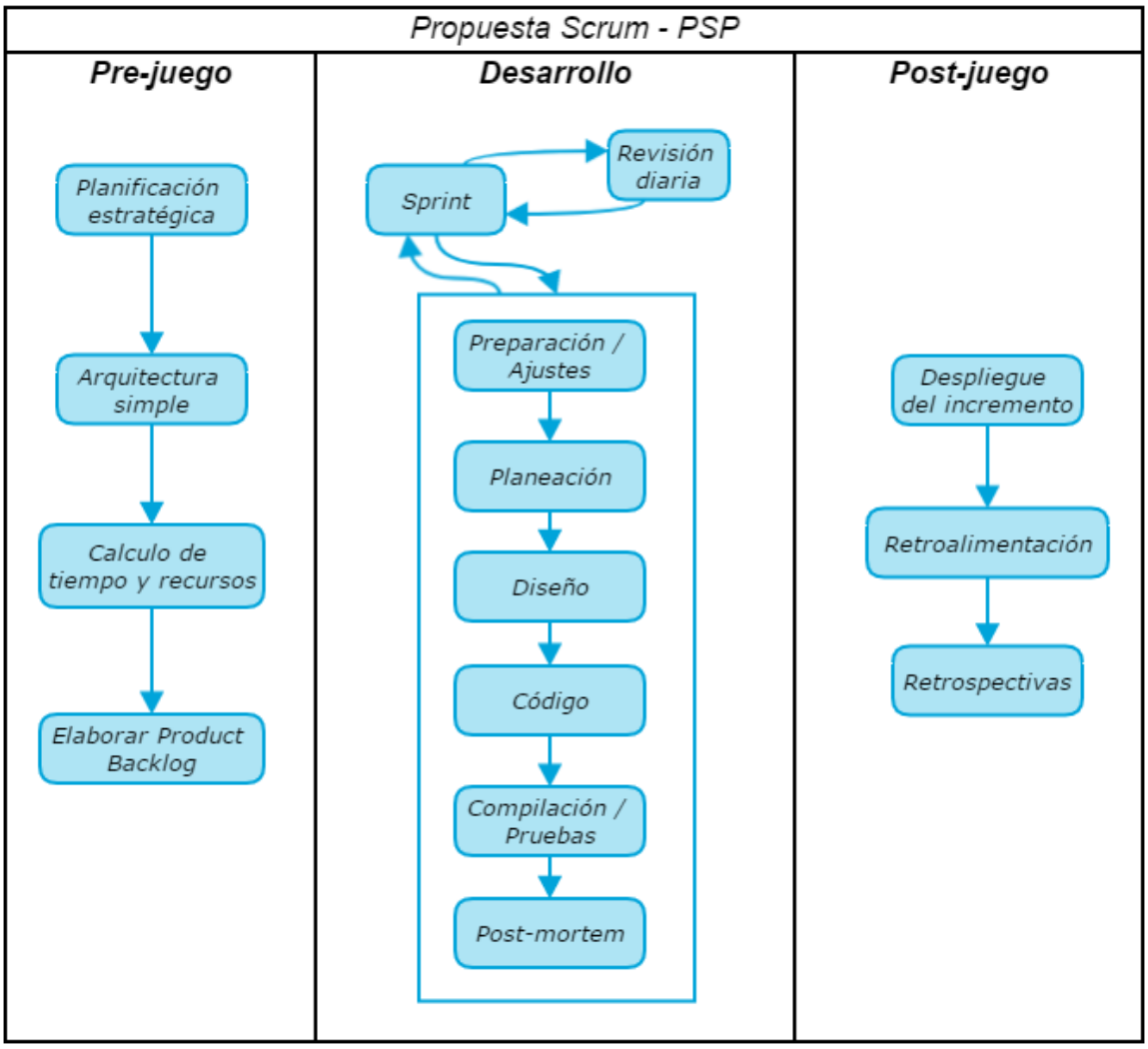


Figura 4.1 *Boceto de Scrum – PSP*

Una vez terminado el boceto del modelo se presentó al *Jefe del Área de desarrollo*, dos *líderes de proyectos* y al *analista de sistemas*; cada uno de ellos presentó diversas observaciones para mejorar la propuesta e integrar mejores técnicas al proceso. A continuación, se enlistan de manera general las observaciones mencionadas durante la reunión:

- ***Integrar actividades de MoProSoft al modelo.*** Petición realizada por parte del *Jefe del Área de desarrollo*, en de su interés retomar las actividades de la norma antes mencionada y facilitar la implementación de la misma para los equipos de trabajo. Para atender esta observación se adaptaron ciertas actividades de los niveles ***Realizado*** y ***Gestionado***, de la

Categoría de **Operación** (OPE) en los procesos de **Administración de Proyectos Específicos** (APE) y **Desarrollo y Mantenimiento de Software** (DMS).

- ***Establecer una compaginación entre roles.*** Realizar un definición sencilla y clara de los roles que es posible utilizar en esta propuesta. Para atender esta petición de cambio se realizó un mapeo sencillo y consistente entre los roles de los tres modelos contra los puestos de trabajo que establece la empresa, acotando el conocimiento básico que requiere tener cada uno para realizar sus funciones.
- ***Mejorar la integración de las fases de PSP con Scrum.*** En un principio se pensó en utilizar el flujo de trabajo que define *PSP* en el nivel *2.1* aunque no se llegará a implementar hasta dicho nivel. Sin embargo, dado que el tiempo de capacitación estaba condicionado por la cantidad de trabajo del equipo se decidió que las fases fueran más rápidas y sencillas de asimilar por los desarrolladores, por lo tanto, se ajustaron las actividades a las definidas en el nivel 1.1.
- ***Resaltar las ventajas entre ambos enfoques.*** De manera puntual se integraron al nuevo modelo mejorado las actividades de *Gestión de Riesgos*, *Estimación* y *Plan de Calidad*, cada una descrita de manera clara y sencilla.
- ***Resaltar los artefactos generados por MoProSoft.*** Cada actividad genera una serie de artefactos o parte de un artefacto, ya sea un documento elaborado en un procesador de texto o bien almacenado en un espacio propio dentro de las herramientas de trabajo que utiliza el equipo.

Gracias a las observaciones realizadas se realizaron los ajustes necesarios dando como resultado un modelo más estable (*Figura 3.13*); una vez más se verificó la estructura y se esperó la aprobación del *Jefe del Área de desarrollo*. De igual manera se logró convencer al líder de equipo que formara parte del caso de estudio y sus respectivos integrantes con la propuesta generada, sobre todo en la métrica principal de *PSP*, las *Líneas de Código* (LOC), que dada su subjetividad en la medición se pensaba que era el principal impedimento para su adaptación.

4.2. Caso de estudio: *Prueba piloto en TecnoMotum*

En este apartado se presenta el desarrollo del caso de estudio realizado en la empresa *TecnoMotum*, para verificar y comprobar la efectividad del modelo *Scrum* y *PSP* dentro de un equipo de desarrollo de software. Es de suma importancia mencionar que cierta información es considerada, por el *Jefe del Área de Software*, confidencial para la empresa ya que se exponen datos personales, código que contiene reglas propias de la empresa, contraseña o técnicas que no puede ser difundidas; por lo tanto, cierta información no será presentada o en el caso estará censurada.

4.2.1. Introducción

En la empresa *TecnoMotum S.A de C.V.*, específicamente en el área de desarrollo de software se realiza el desarrollo y mantenimiento de varios sistemas de telemetría y posicionamiento global para satisfacer las necesidades de los clientes. Los equipos de desarrollo gestionan el desarrollo de los proyectos gracias a ciertas herramientas que los apoyan en el seguimiento de las tareas de sus integrantes, un equipo en particular utiliza el método de ágil *Scrum* para realizar la administración de los proyectos. Gracias a Scrum se proporciona un cierto grado de calidad al producto final, sin embargo, a la hora de "programar" las actividades o módulos asignados a cada integrante del equipo no se considera la disciplina adecuada.

4.2.2. Propósito

Implementar, documentar y difundir las lecciones aprendidas durante el proceso de combinación de dos modelos para el desarrollo de software, *Scrum* y *PSP*, a través de la introducción de conceptos y actividades disciplinarias que permitan mejorar las habilidades de los ingenieros de software.

4.2.3.Pregunta de reflexión

¿Cuál es el beneficio que se obtiene al realizar una combinación entre un enfoque ágil con un enfoque disciplinado para el desarrollo de software que permita mejora la calidad del producto final, mejorando la gestión del proceso de construcción?

4.2.4. Unidad de análisis

Equipo de desarrollo de la empresa *TecnoMotum*, asignado a Leonardo Guzmán Reyes como líder del equipo, tiene a su responsabilidad el proyecto llamado *Alpha Driver* del cual se está próximo a integrar un nuevo alcance llamado *Bitácora de operador*.

4.2.5. Métodos y herramientas de recolección y análisis de la información

Uso de métodos cuantitativos proporcionados por la herramienta de trabajo *PSP Dashboard* con el objetivo de conocer ciertas métricas personales que indiquen el rendimiento de los involucrados en la prueba piloto. Del mismo modo, para la recolección de los datos se utilizan las herramientas *PSP Dashboard* y *JIRA*; la primera ofrece el registro automático del tiempo utilizado en el desarrollo de las actividades personales, la segunda ofrece un control general en el desarrollo del proyecto.

La información recolectada se analizó en base a conjunto de categorías y criterios definidas con el apoyo de los involucrados en la prueba piloto. Para este trabajo se utilizó los cálculos estadísticos proporcionada por *PSP Dashboard*. Se analizaron las variables de tiempo, esfuerzo, complejidad, lenguaje de programación y líneas de código para conocer la relación que existe entre ellas.

4.2.6. Primera etapa

Existen varios factores involucrados en el caso de estudio: (a) En ciertos requerimientos no es posible definirlos a principio del caso de estudio y tienen una alta probabilidad de ser cambiados durante el proceso, (b) Es posible que el tiempo planeado y la prioridad de ciertas actividades cambie por indicaciones de jefes o actividades clasificadas como urgentes, desplazando el caso de estudio, (c) El equipo de desarrollo está formado por cuatro miembros, y dos miembros involucrados en el proyecto, todos los integrantes del equipo de desarrollo recibieron un introducción a los principios *PSP* siguiendo los lineamientos del material proporcionado por Instituto de Ingeniería de Software (*SEI*), (e) El equipo de desarrollo tiene experiencia en la práctica de *Scrum*, el gerente de operaciones de la misma empresa se considera el *Product Backlog*, (f) Antes de comenzar con el caso de estudio el equipo de desarrollo termina sus actividades pendientes, tomando un tiempo de aproximadamente 80 hrs. (2 semanas) para iniciar con el proyecto que será tomado en cuenta para esta caso de estudio.

Durante la primera fase de este caso, el equipo de desarrollo trabajó en conjunto con todos los involucrados para la definición de los requisitos. El desarrollo del sistema comenzó con el análisis de estándares y reglamentos, para después realizar la definición del sistema. Las juntas con el *Product Owner* eran esporádicas, debido a otras responsabilidades que tiene a su cargo, pero largas (alrededor de 5 horas al día). La definición de las historias de usuario fue lenta pero exitosa y se crearon 12 épicas para la primera versión del sistema, basados en conocimientos y en la técnica conocida como *póker planning* fueron elegidas 5 épicas para desarrollar un producto funcional, las cuales fueron descompuestas en tareas y actividades repartidas a cada desarrollador el equipo; a partir de este punto se planean al menos tres Sprint para terminar el producto.

4.2.7. Resumen del caso de estudio

Durante el desarrollo del caso de estudio se observó una buena planeación y administración del proyecto por parte de todos los integrantes del equipo, la definición de los requerimientos fue la

mejora actividad realizada, así como la definición del proyecto y un alcance razonable. Sin embargo, debido a factores que no fueron previstos y distractores originados por peticiones con mayor prioridad, las fechas programadas en el calendario del equipo fueron cambiadas varias veces. Debido a la incertidumbre y al tiempo límite que se tiene para esta investigación se optó por realizar un ajuste drástico al caso de estudio.

Un primer cambio se realizó en la manera de observar el proceso de desarrollo del software, el equipo ya tenía la experiencia en *Scrum* por lo que se decidió realizar una comparación entre las actividades realizadas por equipo de desarrollo contra las actividades definidas en el modelo propuesto. Igualmente, para las capacitaciones programadas se tuvo igualmente el problema con el tiempo, por lo tanto, solo consiguió introducir los conceptos más importantes de la propuesta, así como los principios principales del marco de trabajo *PSP*.

Para esta comparación se establecieron varios criterios para calificar la relación existente entre las actividades realizadas con el equipo y las actividades definidas el modelo propuesto, con ayuda del equipo de desarrollo, se optó por utilizar una simple lista de verificación para evaluar las actividades y los artefactos obtenidos por el caso de estudio.

La escala de numeración que se utilizó para evaluar los artefactos comprende los rangos de 0 a 1, donde **cero** significa que el artefacto no cumple con las características u no fue contemplado por proceso de desarrollo del equipo y **uno** significa que el artefacto cumple con las características necesarias. El modelo propuesto define siete artefactos, cada uno establece diversas "*puntos"* que tienen que redactarse para formar un artefacto completo. En la Tabla 4.1 se muestra los siete artefactos y sus propiedades.

Tabla 4.1. *Artefactos establecidos por Mixing Scrum-PSP*

Artefacto	Formado por
Descripción del proyecto	*Propósito*
	Objetivos
	Alcance del proyecto
	Entregables
	Supuestos
	Restricciones
Plan del Proyecto y Desarrollo	*Tipo de proyecto*
	Descripción del producto
	Calendario de trabajo
	Equipo de trabajo
	Costos estimados
	Plan de manejo de riesgos
	Protocolo de entrega
Configuración de Software	*Especificación de requerimientos*
	Análisis y Diseño
	Software
	Plan de calidad
	Plan de pruebas del software
	Manuales
Product Backlog	*Lista de historias de usuario*
	Definición de completo
	Priorización de requisitos
Backlog de liberación	*Selección de requisitos más urgentes*
	Estimación de costos o esfuerzo
Sprint backlog	*Descomposición de requisitos en tareas*
	Repartir tareas a cada integrante
	Progreso de las tareas
Incremento	*Software funcional*

Para que un artefacto se considerada completo, cada *punto* que lo conforma debe estar escrito y documentado y, lo más importante, contemplado por el equipo de desarrollo durante su flujo de trabajo. Por ejemplo, para que el artefacto **Backlog de liberación** se consideren completo tiene que estar formado por la *selección de requisitos más urgente* y *la estimación de costos o esfuerzo*. Del mismo modo, para calificar las actividades se utiliza la escala de 0 a 1, donde **uno** equivale a la contemplación y desarrollo de la actividad, y **cero** significa que la actividad no fue desarrollada ni contemplada por el equipo de desarrollo.

De manera resumida, se presenta los resultados para cada fase del modelo propuesto, recalcando que debido a factores de tiempo la última fase, la **fase de entrega**, no se logró concluir por completo. Sin embargo, se realizar una avaluación de acuerdo con la planificación que se tenían en ese momento. En la *Tabla 4.2* muestra los valores categóricos utilizados para calificar la exactitud entre las actividades del equipo contra las del modelo propuesto.

Tabla 4.2 Criterio obtenidos para calificar las actividades del modelo

Valor	Categoría	Descripción
C	Completa	La actividad tiene entre un 100% a 90% de exactitud en relación a las actividades y documentos realizados por el equipo.
P	Parcial	La actividad tiene entre un 89% y 50% de exactitud en relación a las actividades y documentos realizados por el equipo.
D	Deficiente	La actividad tiene menos del 50% de exactitud en relación a las actividades y documentos realizados por el equipo.
NC	No contemplada	La activad no se contempla por el equipo de trabajo.

En la **fase de preparación** se observó que el equipo de desarrollo realiza un buen análisis sobre el proyecto, dedicando alrededor de unas 8 horas por semana aproximadamente, resultando en una excelente investigación de normas, recursos, tecnologías y requisitos. En la *Tabla 4.3* se observa la calificación para cada actividad de acuerdo a la escala de valores anteriormente descrita. Cabe mencionar que, durante el desarrollo de esta fase, la actividad de ***Gestión de requisitos*** no fue contemplada por el equipo de desarrollo, pero se genera los artefactos necesarios para realizarla.

Tabla 4.3 *Calificación de las actividades de la fase de preparación*

Actividades	Calificación
A1. Revisar la «Descripción del proyecto».	C
A2. Ingeniería de requisitos.	C
A3. Planificación estratégica de recursos.	P
A4. Gestión de Riesgos.	NC
A5. Priorización de Requisitos.	C
A6. Generar o Actualizar el Plan del Proyecto.	D

La **fase de desarrollo** presentó varios impedimentos que obligaron a modificar el desarrollo del caso de estudio y la prueba piloto. En primer lugar, se menciona que el equipo lleva una buena gestión de las actividades nativas de *Scrum* como son la planificación de sprint, las revisiones diarias y las retrospectivas, gracias a su experiencia y a las herramientas de gestión de proyecto que se maneja dentro de la empresa. Sin embargo, las actividades donde interviene la generación de documentos se mantiene deficiente o sus en el peor de los casos no se contempla. La *Tabla 4.4* muestra la calificación que recibieron las actividades.

Tabla 4.4 *Calificación de las actividades de la fase de preparación*

Actividades	Calificación
A7. Planificación del Sprint	P
A8. Generar/actualizar los manuales del sistema	NC
A9. Gestionar pruebas del sistema	D
A10. Iteración de trabajo *PSP*.	D
A11. Realizar revisiones diarias	C
A12. Revisión del *Sprint*.	P
A13. Retrospectiva del Sprint	P
A14. Obtener incremento	D
A15. Realizar pruebas al incremento	NC
A16. Generar/actualizar la “Configuración del Software”.	NC

Hacia la última fase, en conjunto con los desarrolladores y el líder del proyecto se realizó un análisis al calendario del *Sprint* para determinar una calificación a las actividades que no se lograron realizar, otorgando una calificación provisional en relación a la planificación realizada y los documentos que el equipo de desarrollo se comprometió a entregar al finalizar el *Sprint*. Como se muestra en la *Tabla 4.5* las calificaciones para las actividades de esta fase presentan un asterisco, el cual indica un valor provisional.

Tabla 4.5 *Calificación de las actividades de la fase de entrega*

Actividades	Calificación
A17. Integrar nuevo «Incremento».	P*
A18. Realizar pruebas al Software.	NC*
A19. Conformar los productos para la entrega.	D*
A20. Presentar los informes de actividades.	NC*
A21. Realizar los respaldos de información.	P*

En la sección de ***Anexos*** en el apartado "*Cálculos de caso de estudio"* se presenta a detalle el proceso con el cual se obtuvieron las calificaciones para cada una de las fases descritas con anterioridad.

Para la iteración de PSP se presentan los datos obtenidos para tres programadores que fueron introducidos en los conceptos de cada nivel, teniendo un alcance de enseñanza para PSP fue hasta el nivel 0.1. La capacitación se realizó siguiendo el mismo flujo de trabajo que proporciona el material de enseñanza del Instituto de Ingeniería de Software (*SEI*).

Mediante la observación del desarrollo de cada actividad definida en la iteración de trabajo PSP, se obtuvo la definición de dos principales flujos de trabajo que son mostrados en la Figura 4.2 y la Figura 4.3. Se eligieron dos actividades de cada programador donde construyeran modulo del incremento funcional, para actividad se dio la indicación de dividirlas en pequeños programas a desarrollar y medir el tiempo invertido en cada fase, y los defectos introducidos y reparados.

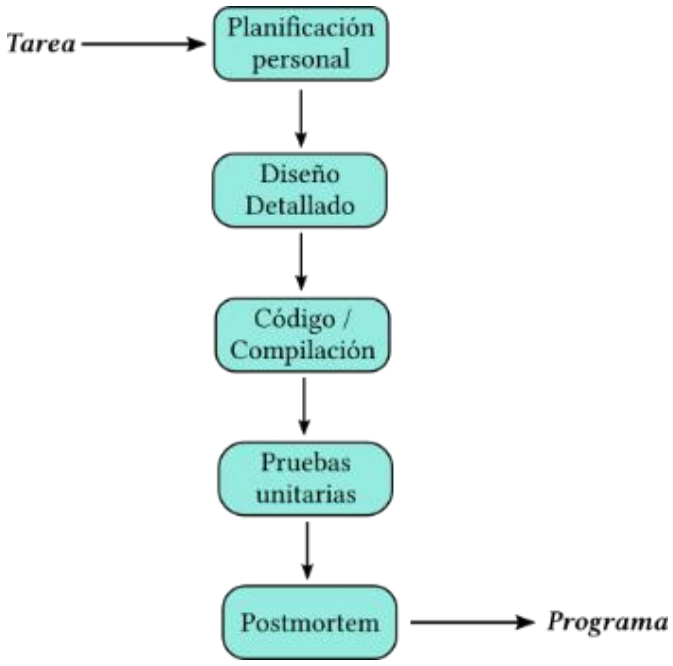


Figura 4.2. *Flujo de trabajo No. 1*

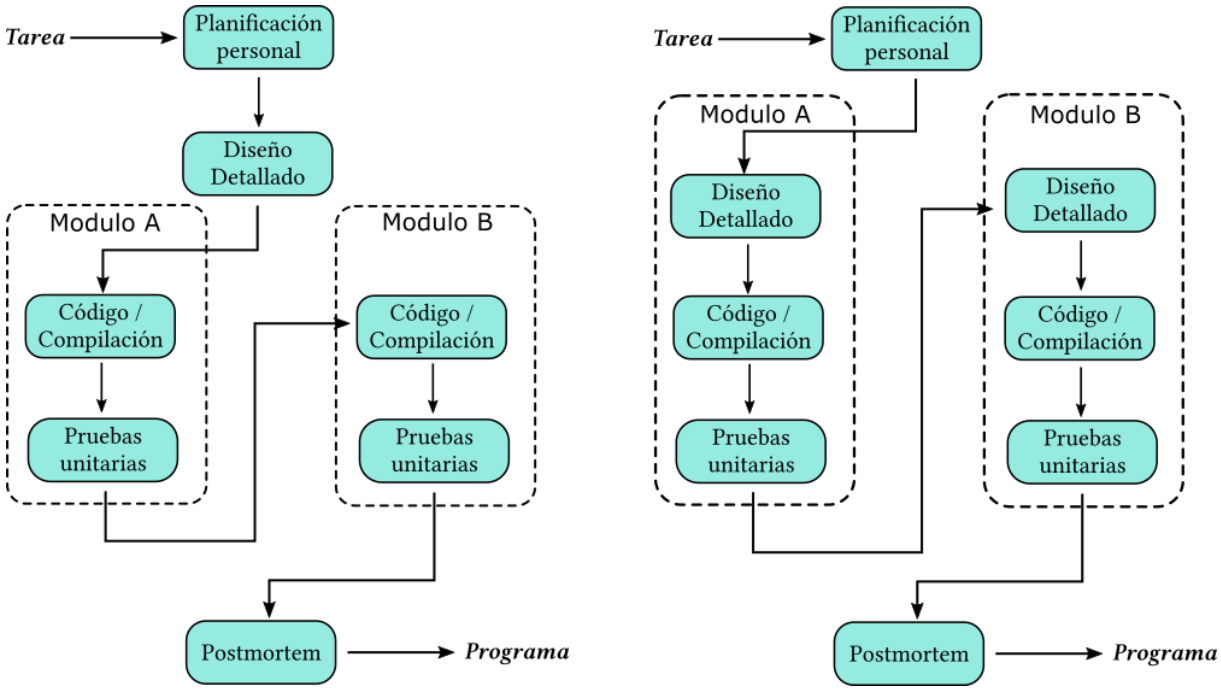


Figura 4.3. *Flujo de trabajo No. 2*

En la *Figura 4.4* se muestra el tiempo que cada integrante utilizó para las diferentes fases de *PSP*, hasta el nivel 0.1. Cada integrante tiene diferentes valores de tiempo para cada una de las fases, esto apoya el primer principio de *PSP*, *"cada desarrollador es diferente; para ser más efectivo, los desarrolladores tienen que planear su trabajo en base a su información personal"*. Sin embargo, uno de los principios de PSP es destruir efectivamente el tiempo estimado del desarrollo en cada una de las fases.

Conocer los tiempos permitió a los programadores tener una perspectiva del tiempo utilizado en el desarrollo de sus actividades, marca el inicio de cualquier módulo o programa a partir de obtener toda la información relevante al tema y considerar todos los inconvenientes que se puedan tener durante la construcción del módulo. Uno de los principales problemas que se tuvieron en la aplicación de PSP fue el cambio de contexto entre el entorno de desarrollo y la herramienta de registro de datos, aunque el registro del tiempo es automático se tiene que interactuar con los controles del cronometro, por lo tanto, es posible que existan discrepancias con los tiempos registrados.

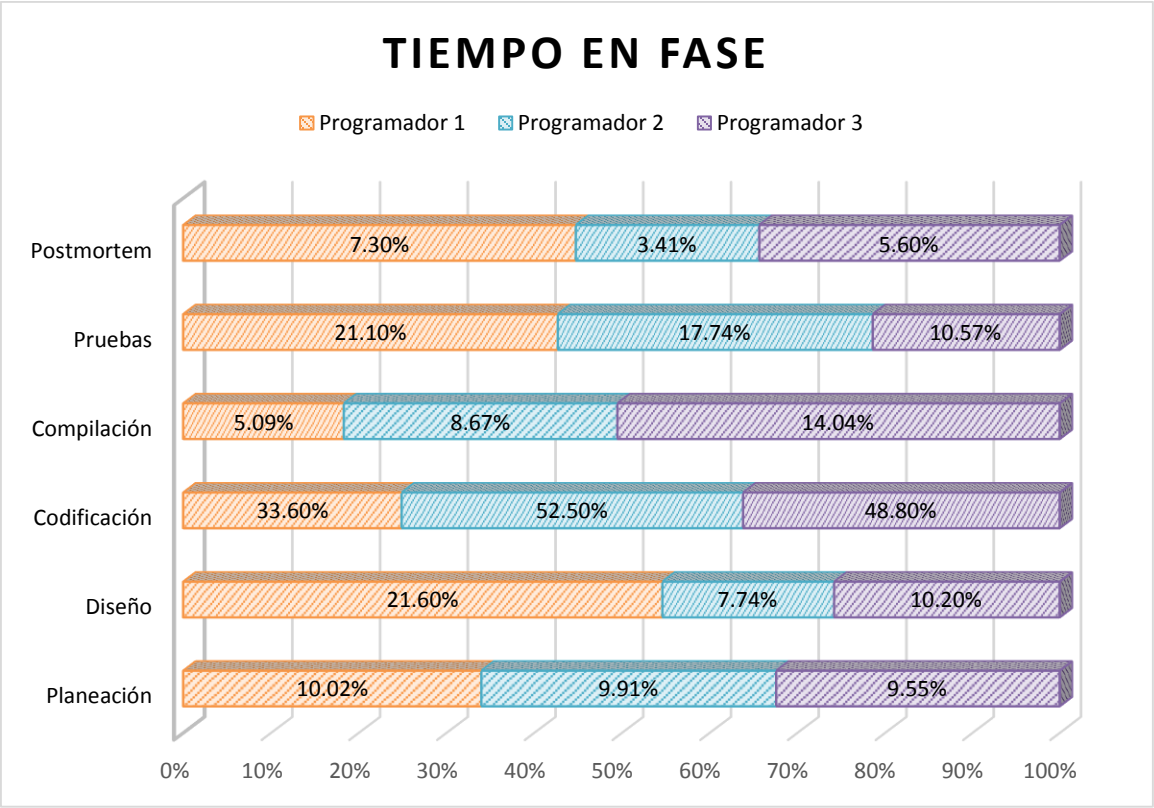


Figura 4.4 *Tiempo utilizado por usuario en fases*

Durante la capacitación de los programadores el término "defecto" fue entendió de manera subjetiva por lo que cada programador registro los defectos que, para sí mismos, los consideran como tal. Debido a esto se presentan ciertas discrepancias en cuanto al registro de los defectos.

En la *Figura 4.5* muestra la relación que existe entre los defectos introducidos y los defectos removidos por fase; una de las reglas que menciona *PSP* es que, por lo general, el número de defectos introducidos tiene que ser el mismo que se eliminan. Los programadores introdujeron la mayoría de los defectos en la fase de codificación y los eliminan durante la misma fase y la fase compilación, derivado a que la idea de programar y probar aún sigue aplicando al flujo de trabajo de los programadores.

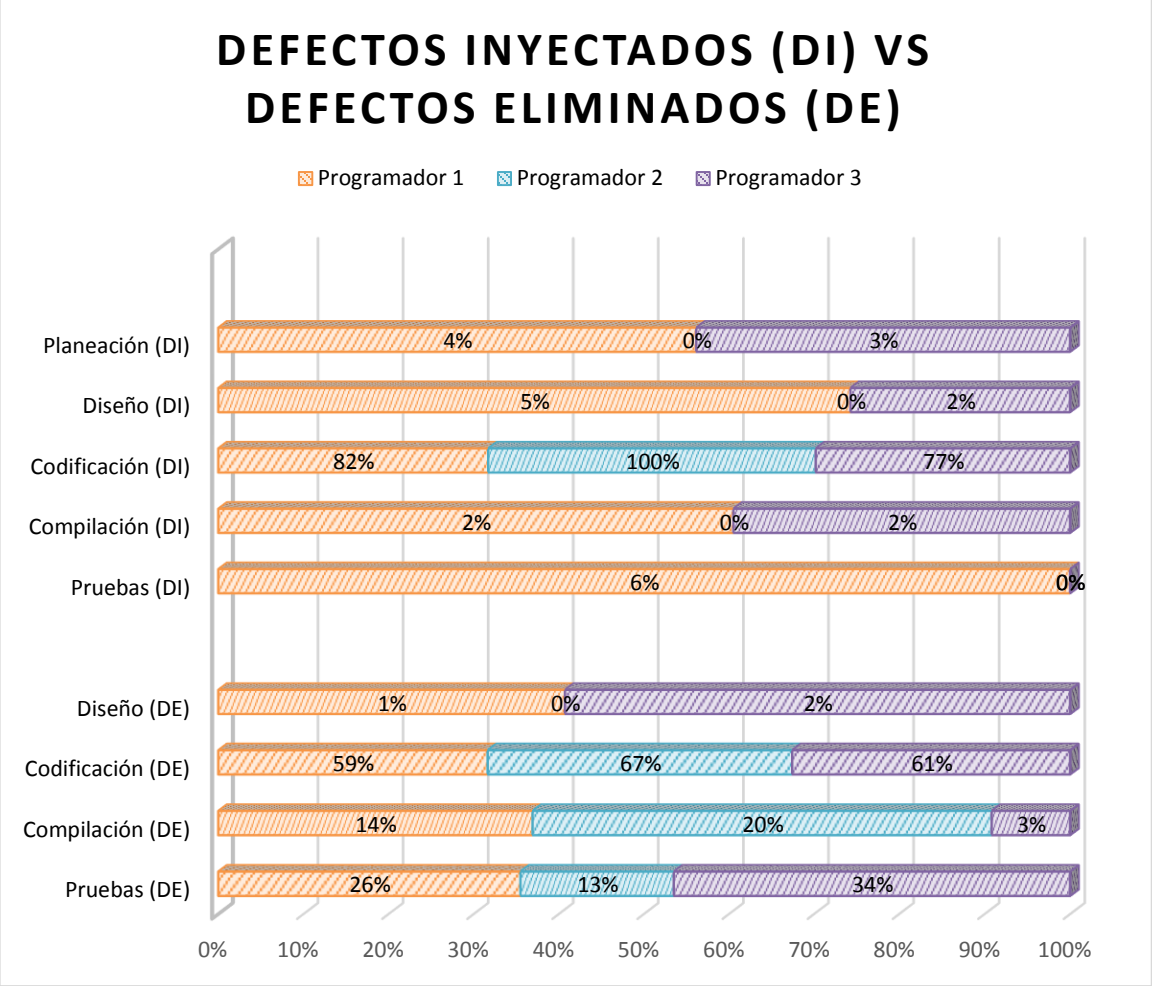


Figura 4.5 *Defectos Inyectados VS Defectos Eliminados*

Como observaciones finales de esta prueba piloto se menciona que la capacitación para los niveles 0 y 0.1 de *PSP* no fueron continuas, por lo que existe un desfase en la comprensión de ciertos conceptos básicos en especial en lo que se refiere al registro de defectos. Sin embargo, los programadores se mostraron abiertos a utilizar nuevas técnicas para mejorar sus habilidades de trabajo.

Un inconveniente principal es el cambio de contexto entre herramientas, ya que los programadores en ciertas ocasiones olvidan registrar los defectos y tiempos. Derivado de este problema, es posible que los datos analizados presenten inconsistencias.

Capítulo 5 Conclusiones y Recomendaciones

5.1. Conclusiones

Con este proyecto de tesis se plantea la posibilidad de utilizar un modelo de desarrollo para software basado en *Scrum*, *Personal Software Process* y *MoProSoft* dentro de un proyecto real que mejora la calidad del proceso de desarrollo de software. Es posible que dentro de las actividades realizadas para el desarrollo del modelo propuesto en este trabajo de investigación se adquirieran los conocimientos sobre los modelos de mejora de calidad, lo complicado y complejo que son a la hora de adaptar a ciertas empresas, y sobre todos que las partes interesadas como la gerencia y equipo de trabajo tenga el interés de utilizar en beneficio propio.

Uno de los principales obstáculos que se presentó durante la prueba piloto, fue la resistencia al cambio de las personas, pues se considera como punto crítico para una adaptación exitosa para cualquier modelo de mejora que se pretenda implementar en una organización. En cuanto al tema de la capacitación, es necesario replantar la manera de tener sesiones de aprendizaje más largas de las acordadas en el caso de estudio ya que no se cubre, al menos, un nivel de *PSP*; esto genera problemas de entendimiento en conceptos básicos. Además, un factor no se tomó en cuenta fue la rotación del personal, pues esto altera por completo el diseño establecido para la capacitación.

Por otro lado, la herramienta utilizada para el registro de los datos de *PSP*, *Process Dashboard*, presenta ventajas a la hora de registrar los tiempos de desarrollo de manera automática, el conteo de líneas de código definido previamente por plantillas, y el análisis de los datos generados por el proceso representados en gráficas. Sin embargo, el cambio de contexto que se realiza para registrar la información de cada actividad es un indicador negativo que resta interés a los desarrolladores.

Como puntos positivos de menciona que el modelo es factible para mejorar el proceso de desarrollo del software dentro de un equipo ya que proporciona un flujo flexible y gracias a la documentación generada a lo largo del proyecto reduce los tiempos al final de desarrollo permitiendo tener actualizado la gestión del proyecto. El introducir los conceptos de PSP dentro del equipo es

beneficioso ya que se genera disciplina al flujo de proceso y ayuda a equilibrar las habilidades de los desarrolladores de software. Como punto negativo no fue posible establecer una relación entre la mejora del proceso de desarrollo contra la calidad del producto final, pues no fue posible medir el software.

5.2. Recomendaciones

Como recomendaciones se encuentra realizar más casos de estudio para conocer la relación entre mejorar el proceso de desarrollo contra la calidad del producto, así como la definición de otros casos de estudios en diferentes ambientes, es decir, comparar resultados de un equipo que no utilice el modelo descrito en este trabajo para tener una perspectiva de la mejora de la calidad, o realizar un caso práctico dentro de un ambiente académico donde no se tenga los factores de presión mencionados anteriormente.

Separar la capacitación de *Scrum* y *PSP*, ya que los conceptos de este último requieren más tiempo y concentración por parte de la persona. Se aconseja realizar un taller de aprendizaje utilizando el material que proporcionar el Instituto de Ingeniería de Software (SEI) junto con la herramienta *PSP Dashboard* para que se familiaricen los desarrolladores con la herramienta. Además, es recomendable contar un mecanismo de evaluación que indique al desarrollador avanzar al siguiente nivel de *PSP*.

Debido a la estructura y contexto en el que se planteó el modelo en sus inicios es posible que se tengan que realizar modificaciones o ajustes para aquellos equipos de desarrollo que se dedican a dar manteamiento al software, integrando otros modelos o normas que ayuden desde esta perspectiva. Es factible enriquecer las actividades que establece *Scrum* con las técnicas y prácticas que ofrece *Team Software Process* (*TSP*) u otras técnicas que ayuden a obtener métricas cuantitativas para medir la calidad del proceso y la calidad del producto.

Anexos

Estructura detalla del ciclo de vida MPS

En esta sección se muestra un fragmento la guía de ***Mixing Scrum – PSP*** realizada para la empresa ***TecnoMotum S.A de C.V.***; la guía completa se proporciona en disco adjunto a este trabajo con el nombre de "*Guia-ScrumPSP.docx*".

"… este modelo define 3 fases: **Preparación**, **Desarrollo** y **Entrega**; para el apoyo de la gestión del proyecto de software. Cada fase proporciona una serie de actividades y sub-actividades que se deben atender de manera puntual y ordenada, cada actividad necesita de una entrada o varias entradas durante el desarrollo del proceso para generar una salida o varias salidas …"

Primera Etapa: Preparación

"Fase de definiciones por parte del equipo de trabajo, en donde se detalla o actualiza la «Descripción del proyecto»; además se realiza la elaboración de calendarios y se estable un protocolo de entrega. Por lo general durante esta etapa se producen gran número de inexactitudes con las estimaciones, pues se realizan a "juicio de expertos". Conforme se apliquen las técnicas y mejoras que define la iteración de trabajo PSP las estimaciones serán predecible gracias a registros históricos de cada actividad…"

A1. Revisar la «Descripción del proyecto».

A1.1. Examinar/actualizar la «Descripción del proyecto». Revisar la descripción que se tiene del proyecto y, si es necesario, actualizarlo con las últimas modificaciones realizadas en iteraciones anteriores; la descripción debe ser escrita en un lenguaje natural donde cualquier persona lo entienda. En caso de que el documento no exista se deberá elaborar, tomando el tiempo necesario, las fuentes de información para su contenido y a los involucrados del proyecto.

A1.2. Elaborar la «Descripción del producto». Conforme a la «Descripción del proyecto» generar la «Descripción del producto» la cual contiene una descripción de las características del software, su alcance y restricciones.

A1.3. Establecer el «Protocolo de entrega». Definir de manera clara y precisa, cuáles serán los entregables del proyecto, el responsable de entregarlos, la fecha de entrega, medio y mecanismo de entrega. Toda la información es anexada al documento «Plan del proyecto y desarrollo» en la sección «Protocolo de entrega».

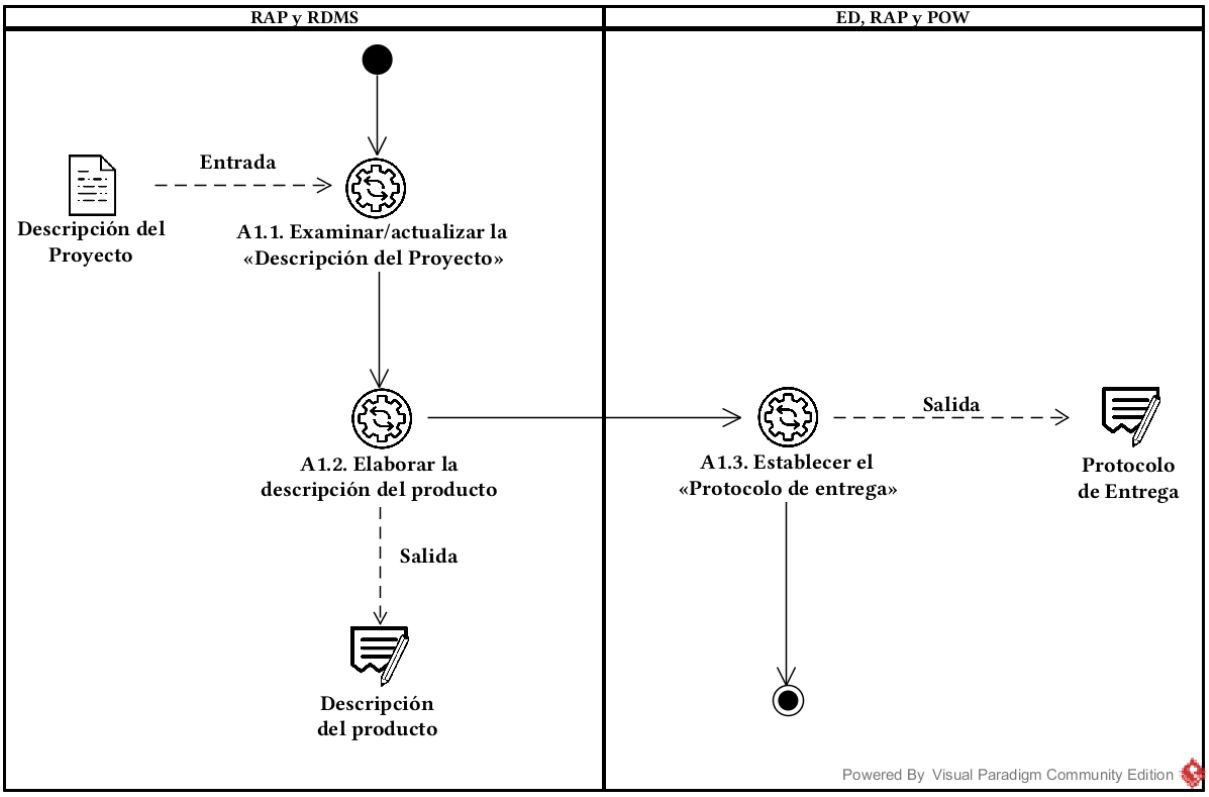


Ilustración 1. *Revisar la «Descripción del proyecto»*

A2. Ingeniería de requisitos.

A2.1. Captura de requerimientos. Actividad en la que una persona o grupo especializado extrae, de cualquier fuente de información disponible (*clientes, usuarios, sistemas previos, documentos, entre otros.*), las necesidades del cliente y las características que el sistema tiene que cumplir.

A2.2. Definición de requisitos. Las historias de usuario son útiles para este tipo de tarea, pues es posible describir la funcionalidad de manera corta y breve desde una perspectiva del usuario. En este punto es posible detallar cada historia de usuario e incluso utilizar la estructura de descomposición del trabajo con el fin de establecer actividades más sencillas de realizar.

A2.3. Verificación y validación de requisitos. Se tiene que verificar la claridad de redacción de cada requisito y su consistencia con respecto a la «Descripción del producto», que sean completos, sin ambigüedad y conforme al estándar de documentación que se utiliza en la empresa. Además, se tiene que validar que los requisitos sean factibles para su producción cumpliendo con las necesidades y expectativas acordadas con el *Product Owner*.

A2.4. Elaborar/actualizar el «Product Backlog». Se comienza la creación de la lista de requerimientos del proyecto, conocida como *Product Backlog*. Es posible realizarlo paralelamente con la actividad de *Planificación estratégica de recursos*. Para las próximas iteraciones este artefacto será actualizado de acuerdo con las actividades finalizadas y cambios provenientes de los

interesados. Toda la información generada será parte de la «Especificación de requerimientos» del documento «Configuración del Software».

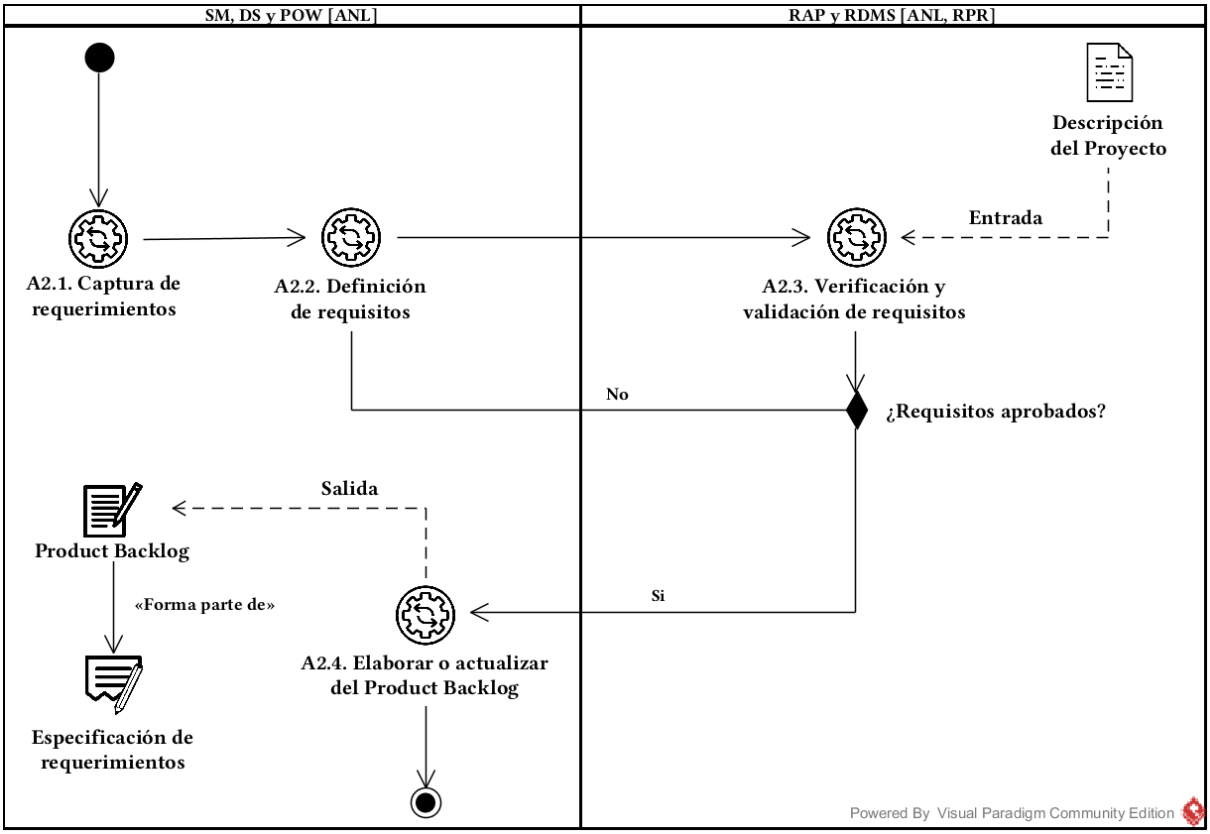


Ilustración 2. *Ingeniería de requisitos*

A3. Planificación estratégica de recursos.

A3.1. Identificar los recursos de trabajo. Actividad para confirmar la disponibilidad de los recursos y conseguir el equipo necesario para completar las actividades del proyecto. Además, se tienen que identificar el conocimiento y/o nivel de habilidad sobre la lógica del negocio y las tecnologías para desarrolla el proyecto. Realizar capacitaciones para aquellos miembros del equipo que lo necesiten.

A3.2. Formar el equipo de trabajo. Seleccionar e identificar a las personas y sus roles que participaran dentro del proyecto, así como las responsabilidades y las habilidades requeridas por cada integrante del equipo de trabajo. Es posible que cualquier integrante del equipo se asignado a más de un rol, incluso como rol auxiliar. Como salida se elabora la sección «Equipo de trabajo» del «Plan del proyecto y desarrollo».

A3.3. Estimar costos generales. Estimar la duración y los costos para el desarrollo del proyecto considerando información histórica o experiencia. Para las estimaciones considerar la complejidad,

tamaño y grado de incertidumbre por cada requerimiento. Al finalizar esta información es concentrada en el apartado «Estimación de Costos» del documento «Plan del proyecto y desarrollo».

A3.4. Generar un calendario de trabajo. Consiste en analizar las secuencias de actividades, los responsables, las duraciones, la disponibilidad de los recursos y las restricciones de tiempo. El beneficio es tener una relación lógica de fechas planificadas para completar cada actividad. Anexar esta información al «Plan del proyecto y desarrollo» en la sección «Calendario de trabajo».

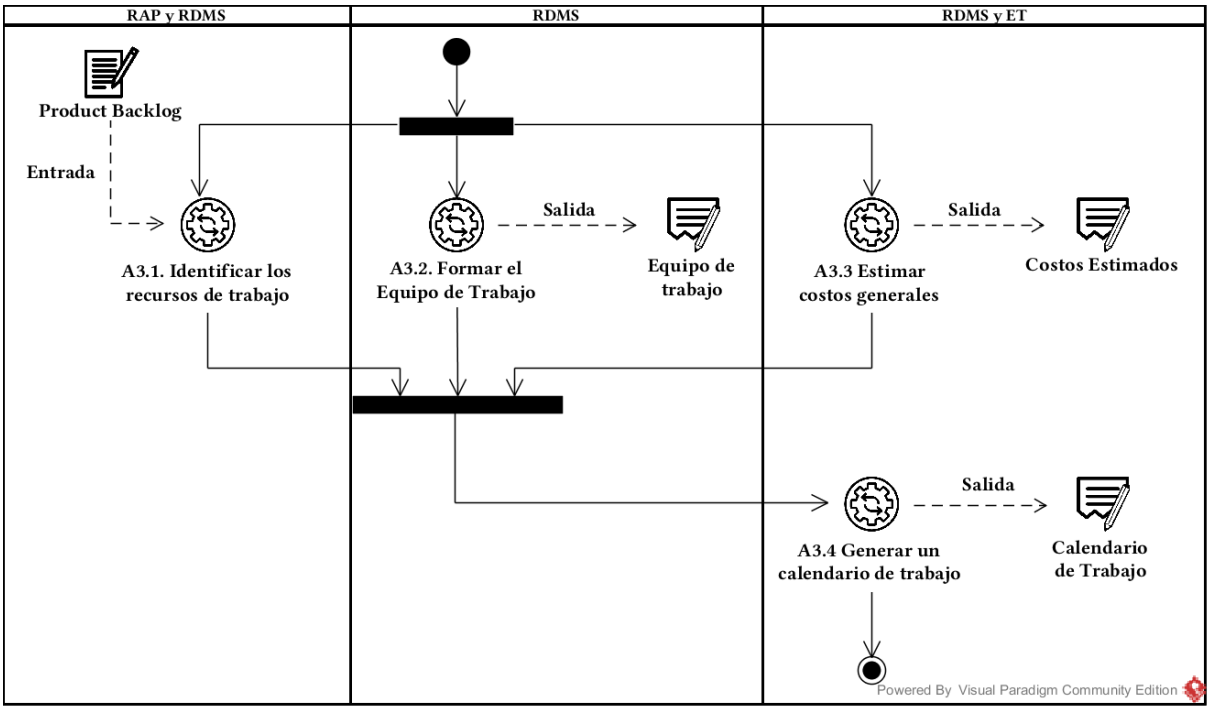


Ilustración 3. *Planificación estratégica de recursos*

A4. Gestión de riesgos.

A4.1. Identificar los posibles riesgos. Consiste en determinar el nivel y tipo de riesgo que pueden afectar al proyecto y conocer sus características. La clave de la identificación recae en el conocimiento y capacidad que confiere al equipo para anticipar eventos caóticos. Es un proceso iterativo debido a que pueden evolucionar o describirse nuevos riesgos conforme el proyecto avanza.

A4.2. Realizar un análisis de riesgos. Los análisis se realizan sobre los riesgos para conocer los efectos que tendrán en caso de presentarse. La ventaja radica en la generación de información cuantitativa o cualitativa, dependiendo del tipo de análisis utilizado, para determinar el plazo de respuesta y la tolerancia al riesgo por parte de la organización, asociado a las restricciones de costo, tiempos y alcance.

A4.3. Elaborar planes de contingencia. Actividad de desarrollar opciones y acciones para mejorar las oportunidades y reducir las amenazas a los objetivos del proyecto. El punto clave radica en el abordaje de riesgos en función de su prioridad, introduciendo recursos y actividades dentro del «Calendario de trabajo» y en el «Plan del proyecto y desarrollo» según las necesidades.

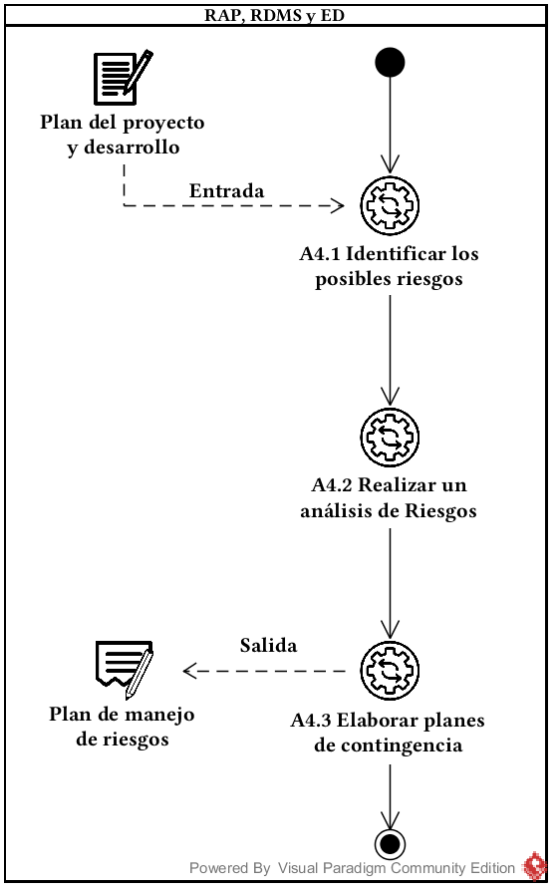


Ilustración 4. *Gestión de riesgos*

A5. Priorización de requisitos.

A5.1. Seleccionar los requisitos. Seleccionar los requisitos que conformaran la versión actual del software, es decir el incremento, los criterios para priorizar los requisitos pueden ser de negocios, costos, técnicos o recursos.

A5.2. Realizar estimaciones de esfuerzo. Utilizar la técnica *Planning Poker* para generar una estimación de esfuerzo para cada requisito considerado anteriormente. Reforzar las estimaciones con el método PROBE proporcionado por PSP. Actualizar las secciones «Estimación de Costos» y «Especificación de requerimientos» con las estimaciones realizadas.

A5.3. Generar el «Backlog de Liberación». Elaborar el concentrado de los requisitos ordenados por prioridad, utilizando cualquier tecnología disponible para el equipo de trabajo. Si los requerimientos del «Product Backlog» son pocos y el equipo de trabajo considera agregarlos a la versión actual, el «Product Backlog» se convierte en el «Backlog de Liberación».

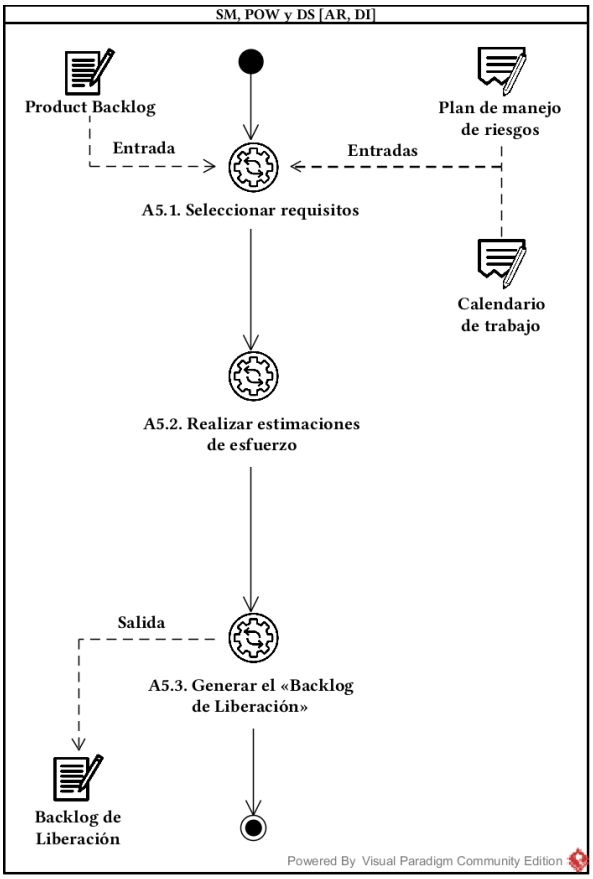


Ilustración 5. *Priorización de requisitos*

A6. Generar/Actualizar el «Plan del proyecto y desarrollo».

A6.1. Conformar el «Plan del proyecto y desarrollo». Juntar la información de las secciones «Descripción del producto», «Calendario de trabajo», «Equipo de trabajo», «Costos estimados», «Plan de manejo de riesgos», «Protocolo de entrega» antes de iniciar un nuevo ciclo.

A6.2. Verificar el «Plan del proyecto y desarrollo». Verificar que todos los elementos del plan son viables y consistentes, y que la redacción sea conforme a los estándares de la empresa. Todos los defectos encontrados tienen que ser informados a los responsables para su corrección.

A6.3. Validar el «Plan del proyecto y desarrollo». La validación es de acuerdo con la «Descripción del Proyecto» donde se describen las necesidades y expectativas del cliente. Todos los defectos encontrados tienen que ser informados a los responsables para su corrección.

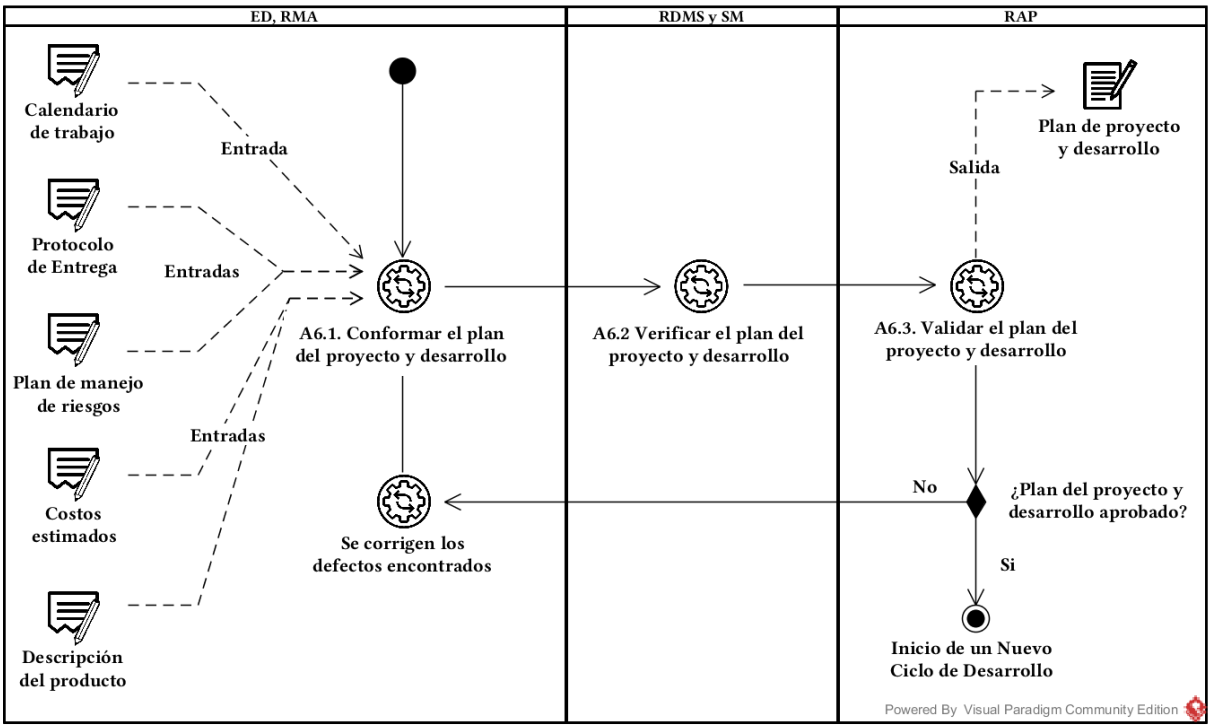


Ilustración 6. *Generar/Actualizar el «Plan del proyecto y desarrollo»*

Segunda Etapa: Fase de Desarrollo

"… se desarrollan las actividades planificadas del «Backlog de Liberación», compagina flujo de trabajo y actividades del *Sprint* junto con actividades de *MoProSoft*, para mejorar el proceso de desarrollo del sistema…"

A7. Planificación del *Sprint*.

A7.1. Revisar el «Plan del proyecto y desarrollo». Revisar con los miembros del equipo de trabajo el «Plan del proyecto y desarrollo» para recordar los tiempos de ejecución de cada actividad, lograr un entendimiento común y obtener su compromiso con el proyecto. Cada miembro del equipo tiene que registrar las actividades realizadas, fechas de inicio y fin, duración y mediciones requeridas.

A7.2. Recibir y analizar solicitudes de cambio. Se reciben las peticiones de cambios por parte del *Product Owner* y se analizan a detalle para identificar el impacto al desarrollo actual. Se tomarán en cuenta aquellos cambios en los requerimientos actuales del «Backlog de Liberación» o sobre los requerimientos del «Sprint Backlog» que aún no han sido desarrollados. Aquellas peticiones inesperadas se tendrán que analizar y discutir por el equipo de trabajo para llegar a una decisión que beneficie a ambas partes.

A7.3. Generar el «*Sprint Backlog*». Una vez que se tengan ordenas las funcionalidades se reparten en diferentes *Sprints*, recordando que cada uno tiene duración máxima es de 30 días. Si los requisitos son pocos y no sobrepasan de la duración máxima el «Backlog de Liberación» pasa a convertirse en el «Sprint Backlog».

A7.4. Distribuir las tareas al equipo de desarrollo. Junto con el *Scrum Master* asignar las tareas a cada integrante del equipo de desarrollo a realizar durante el *Sprint.* Además, la distribución de la información necesaria a los desarrolladores y la asignación de recursos para llevar acabo dichas tareas. Es posibles que las tareas asignadas se detallen por parte de los responsables de cada actividad, en cualquier caso, se debe actualizar la «Especificación de Requerimientos».

A7.5. Definir y elaborar un «Plan de calidad». Seleccionar las normas o reglas de calidad que se utilizarán durante el desarrollo del *Sprint*. Se contemplan reglas para la documentación de manuales de usuario, manuales de operación, manuales de mantenimiento, para la codificación de los módulos, para el registro de actividades, entre otras. Lo importante es que el equipo de desarrollo las defina de manera sencilla y las lleve a cabo durante todo el desarrollo. Anexar la información generada en la sección «Plan de Calidad» del documento «Configuración de Software»

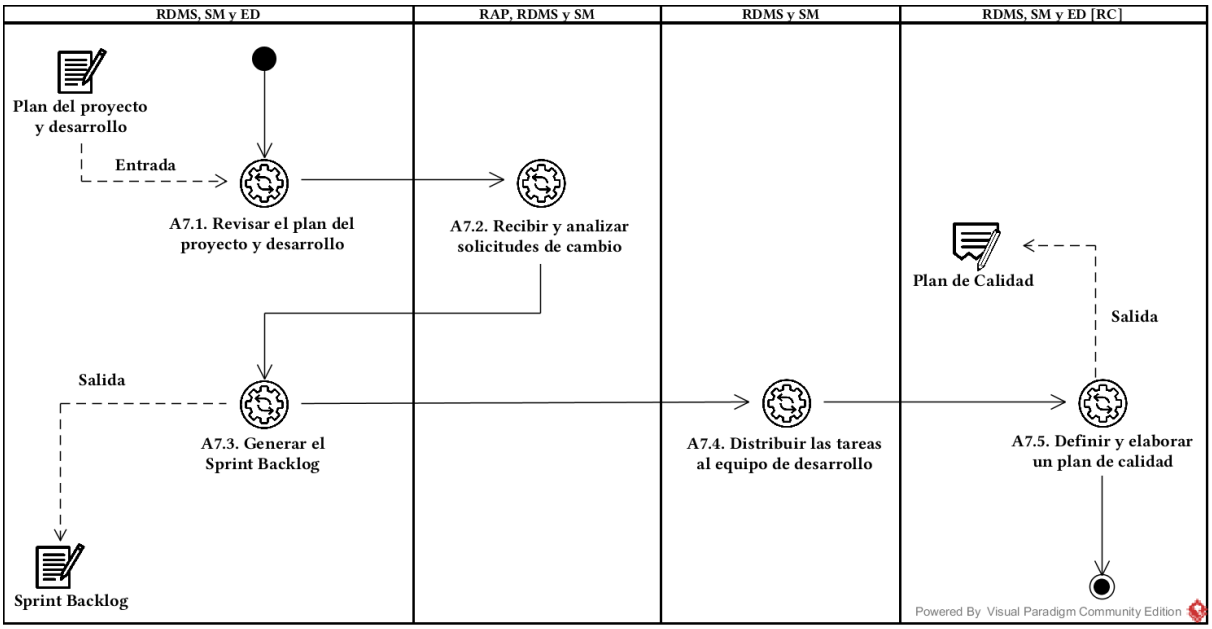


Ilustración 7. *Planificación del Sprint*

A8. Generar/actualizar los manuales del software.

A8.1. Redactar los manuales. Tomando como base la información contenida en el «Plan del proyecto y mantenimiento», «Configuración de software» y el «Incremento» se escribirán los

diferentes manuales. En iteraciones siguientes a la primera se tiene que actualizar conforme a cambios y modificaciones realizados en fase anteriores.

A8.2. Revisar los manuales. Verificar y validar la consistencia de la información y su redacción que sea conforme a los lineamientos establecidos por el equipo de desarrollo y las normas que utiliza la empresa.

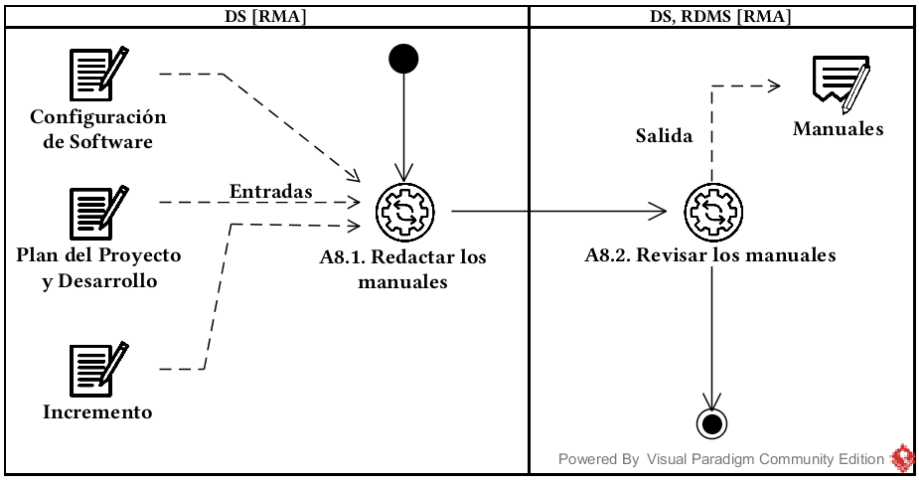


Ilustración 8. *Generar/actualizar los manuales del software*

A9. Gestionar pruebas del sistema.

A9.1. Revisar los requerimientos y el diseño. El equipo de desarrollo y el responsable de pruebas tienen que estudiar y revisar los requerimientos para identificar el tipo de pruebas que se deberán realizar para cada requerimiento, la prioridad y el enfoque para cada prueba, y elegir el entorno de pruebas. La posibilidad de automatizar las pruebas se considera también en esta actividad.

A9.2. Planificar las pruebas. El equipo de desarrollo y el responsable de pruebas determinarán la estrategia para los tipos de pruebas, como serán realizadas y quien será el responsable de ejecutarlas, la selección de las herramientas, una estimación de esfuerzo y determinar las responsabilidades.

A9.3. Diseñar las pruebas. Esta actividad involucra la creación de casos de pruebas o procesos de automatización, la revisión de las guías para los casos ejecutar, creación de los datos de prueba o entornos de pruebas automáticos si están disponibles.

A9.4. Ejecutar las pruebas. El equipo de desarrollo ejecuta las pruebas de acuerdo a la planificación y los casos de pruebas preparados. Los defectos encontrados son registrados y reportados para su corrección y devuelto para probar su desempeño.

A9.5. Reportar los resultados. Es la recopilación de los resultados realizados de las pruebas, con estos datos se generar informes que serán analizados y presentados para comunicar los avances a los responsables del proyecto, además es una evidencia de un correcto desarrollo de los requerimientos del software. La información generada en esta actividad es registrada en la sección «Bitácora de pruebas» del documento «Configuración del software».

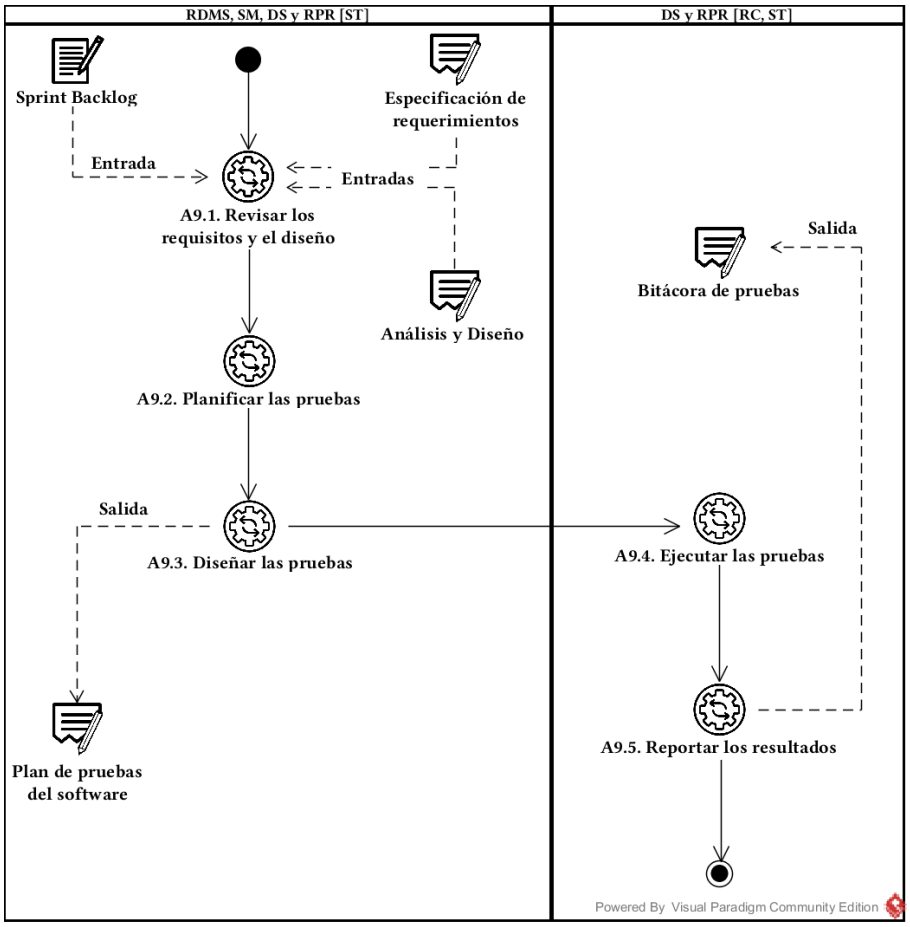


Ilustración 9. *Gestionar pruebas del sistema*

A10. Iteración de trabajo PSP.

A10.1. Planeación. De manera personal y conforme a las actividades asignadas en fases anteriores se elabora un plan que consiste en la obtención y definición detallada de los requerimientos para el programa, escritos en documento claramente y sin ambigüedades, y una buena estimación de tiempo para completar el desarrollo del programa. Los formatos para escribir el plan de trabajo no son difíciles, pero requieren toda la atención del desarrollador.

A10.2. Desarrollo. Es aquí donde se realizar el diseño, código, revisiones y pruebas de cada actividad asignada a cada integrante del equipo de desarrollo. El desarrollador debe utilizar todas

las herramientas a su disposición para realizar esta actividad. La métrica básica que establece PSP son las líneas de código en base al tiempo, esfuerzo, y complejidad de la actividad. La información obtenida en esta actividad debe ser anexa a la sección «Análisis y Diseño» de documento «Configuración del Software».

A10.3. Post-mortem. Actividad donde se analizan y verifican la consistencia de los datos generados por el proyecto. También se actualizan los valores sobre el tiempo estimado y el tiempo real utilizado, calidad y productividad. Además, la información proporcionada por los productos personales ayuda a estableces bases y una mejora del proceso en las retrospectivas del Sprint.

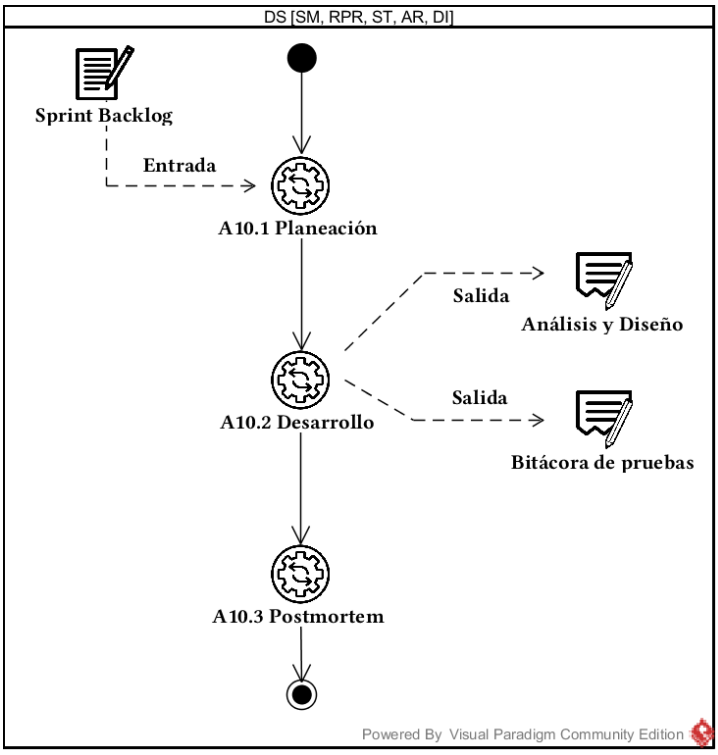


Ilustración 10. *Iteración de trabajo PSP*

A11. Realizar revisiones diarias.

A11.1. Inspeccionar trabajo realizado. Revisar el avance del trabajo realizado desde la última revisión, el *Scrum Master* se asegura que los requerimientos se elaboren conforme a lo estipulado en el «Plan del proyecto y desarrollo» y «Configuración de software. Se prepara un plan para las actividades de las siguientes 24hrs. Cualquier impedimento o contratiempo tiene que ser informado a todo el equipo y se abordara conforme al plan de riesgos y los planes de contingencia.

A11.2. Revisar peticiones de cambios. Se realiza una revisión rápida sobre las solicitudes de cambios por parte del *Product Owner*, el Scrum Master y equipo de desarrollo debe analizar si los cambios son posibles aceptar en el *Sprint* actual o en el siguiente.

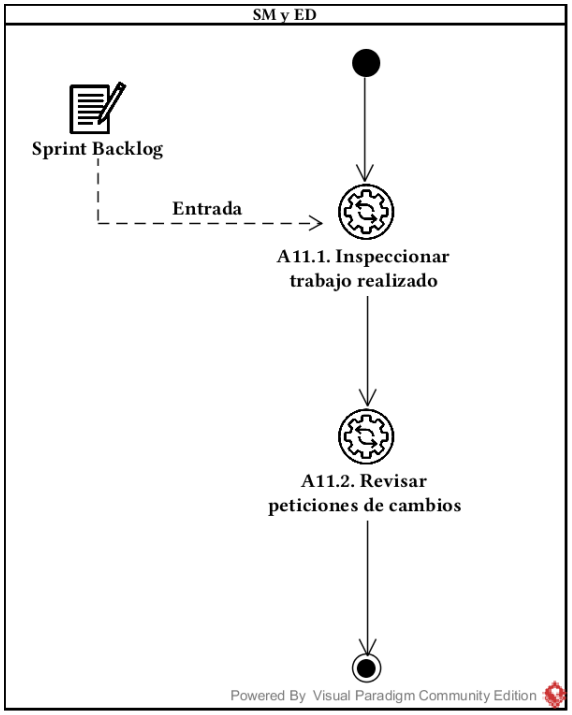


Ilustración 11. *Realizar revisiones diarias*

A12. Revisión del *Sprint*.

A12.1. Revisar las actividades realizadas. El equipo de trabajo presenta un resumen de las actividades realizadas basándose en el «Sprint Backlog», puntualizando los requerimientos cubiertos, cuáles fueron los inconvenientes y las ventajas que se tuvo en el ciclo de trabajo.

A12.2. Ejecutar la demostración del «Incremento». Se muestra las funcionalidades realizadas al *Product Owner*, la presentación se realizará desde un servidor lo más parecido al de producción. Solo aquellas funcionalidades que estén completas se presentan, los artefactos que no sean funcionales no se presentan para no confundir a los grupos interesados.

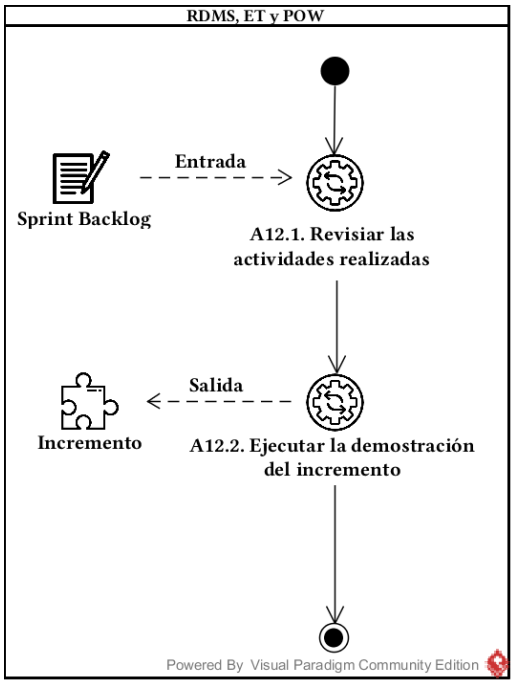


Ilustración 12. *Revisión del Sprint*

A13. Retrospectiva del Sprint.

A13.1. Inspeccionar el último Sprint. Conforme a los datos generados en la iteración de desarrollo PSP y aquellos que puedan ser compartidos al equipo, se realizar una revisión en cuanto a personas, relaciones, herramientas y el proceso de trabajo. Las verificaciones se realizarán tomando en cuenta el «Plan del proyecto y desarrollo» y la «Configuración de Software».

A13.2. Identificar lo bueno y lo malo. Ordenar los elementos más importantes que se realizaron correctamente y lo que se realizaron de manera incorrecta. Elaborar una comparación con respecto a las acciones negativas y positivas, y encontrar las mejores vías de apoyo para introducir las mejoras al proceso de desarrollo.

A13.3. Crear un plan de mejora. Elaborar un plan con las mejoras identificadas para corregir las acciones negativas y añadirlo al «Plan de calidad».

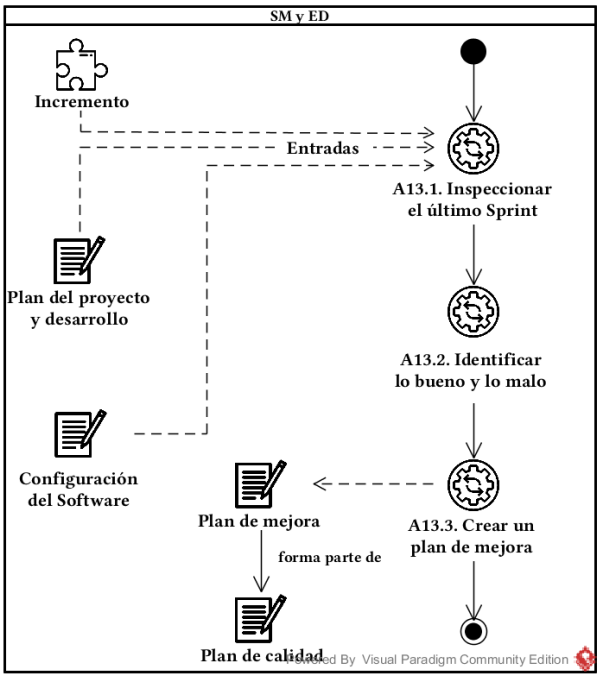


Ilustración 13. *Retrospectiva del Sprint*

A14. Obtener Incremento.

A14.1. Actualizar los Backlog del desarrollo. Al finalizar la demostración del incremento, se actualiza el «Backlog de Liberación» con las actividades terminas. Se revisan las fechas y tiempos para la próxima entrega del producto, estas tienen que quedar registradas en documento formal y ser aprobadas por el *Product Owner*.

A14.2. Actualizar los manuales del sistema. Actualizar la información contenida en los manuales y revisar su consistencia contra la funcionalidad del software y los estándares de calidad descritos en el «Plan de calidad».

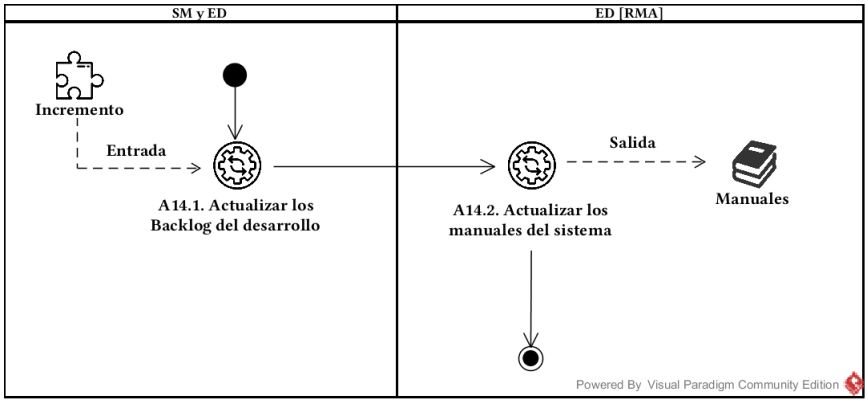


Ilustración 14. *Obtener Incremento*

A15. Realizar pruebas al «Incremento».

A15.1. Ejecutar las pruebas. Siguiendo con los lineamientos establecidos en el «Plan de pruebas del software» se debe de realizar las pruebas al incremento generado.

A15.2. Registrar y corregir los defectos. Aquellos defectos encontrados por las pruebas deben ser registrados en la «Bitácora de pruebas» para su respectivo análisis y su posible solución. Si el defecto es lo bastante simple y rápido de cambiar se realiza antes de finalizar el Sprint, por el contrario, si el defecto es complejo y no se encuentra precedentes para su corrección se tendrá que realizar en otro Sprint. Al final es el *Scrum Master* quien toma la decisión.

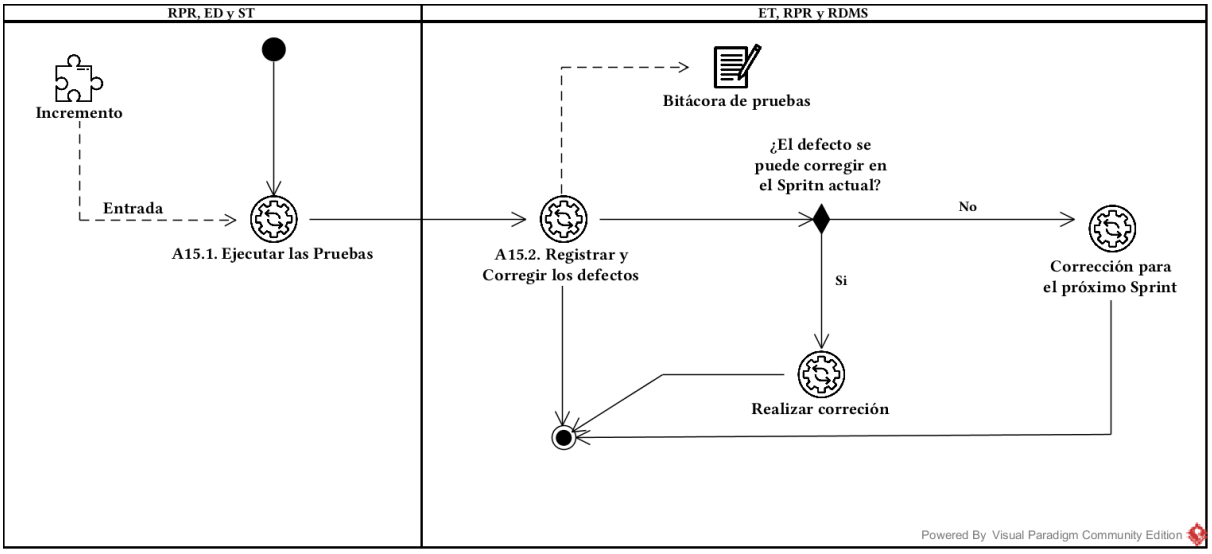


Ilustración 15. *Realizar pruebas al «Incremento»*

A16. Generar/actualizar la «Configuración del Software».

A16.1. Conformar la «Configuración del Software». A partir de los elementos definidos en las tareas anteriores y antes de iniciar un nuevo ciclo se tiene que generar el documento mencionado; si se encuentra en ciclos posteriores se actualiza.

A16.2. Revisar la «Configuración del Software». Verificar y validar que el contenido del documento sea consistente y que la redacción sea conforme a los estándares de la empresa. Todos los defectos encontrados tienen que ser informados a los responsables y ser corregidos antes de iniciar la siguiente etapa.

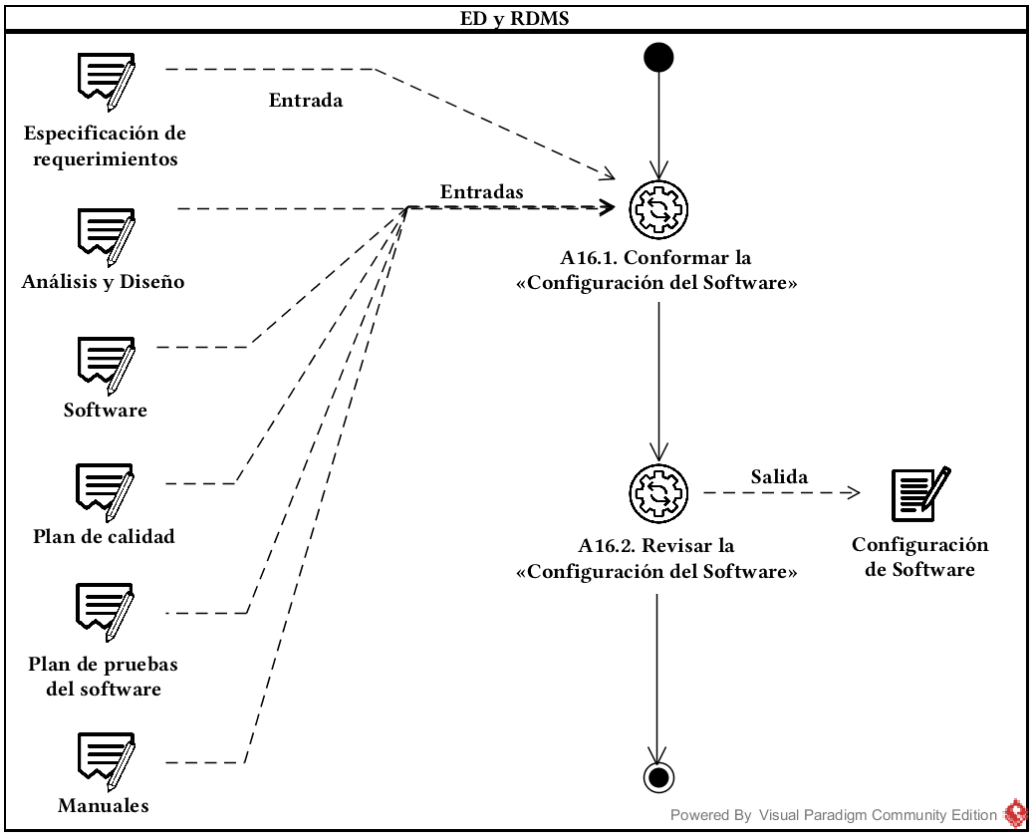


***Ilustración 16**. Generar/actualizar la «Configuración del Software»*

Tercera Etapa: Fase de Entrega

"… se realizan la integración del nuevo incremento con el previamente desarrollado para verificar su funcionamiento correctamente dentro del ambiente del sistema realizando las pruebas necesarias. También se realiza una evaluación conforme al plan del proyecto para determinar la consistencia contra los objetivos establecidos desde el inicio del proyecto. Además, se realiza actualiza los reportes de actividades al desarrollo del proyecto y generara los respaldos de los datos e información en los diferentes repositorios y control de versiones…"

A17. Integrar nuevo «Incremento».

A17.1. Realizar Integración. Verificar que todas las unidades funcionales están listas para su integración, donde se revisan los datos de entrada, el control de procesos, la integridad de los mensajes, validar datos de salida y protección de los datos de prueba. Crear el procedimiento de generación del programa a distribuir de acuerdo a la plataforma objetivo e integrar todas las unidades funcionales de acuerdo al procedimiento definido.

A17.2. Evaluar el cumplimiento del «Plan del proyecto y desarrollo». Evaluar el cumplimiento del «Plan del proyecto y desarrollo», con respecto al alcance, objetivos, costo, calendario, equipo de trabajo, proceso y, en su caso se establecen acciones correctivas.

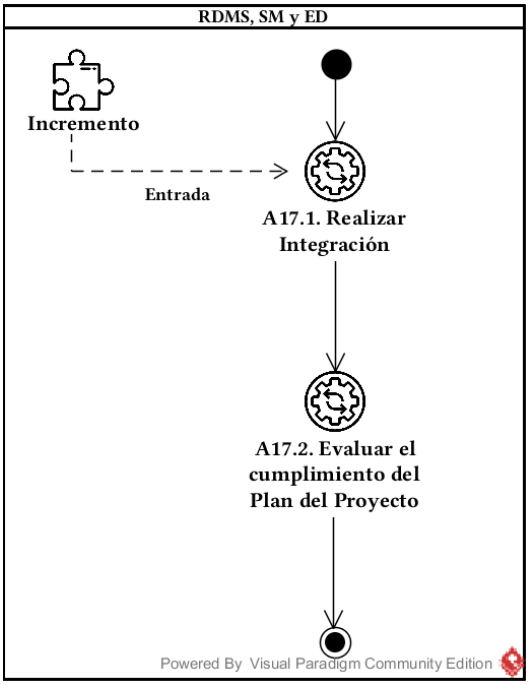


Ilustración 17. *Integrar nuevo «Incremento»*

A18. Realizar pruebas al Software.

A18.1. Ejecutar las pruebas. Siguiendo con los lineamientos establecidos en el «Plan de pruebas del software» se debe de realizar las pruebas al incremento generado.

A18.2. Registrar y corregir los defectos. Aquellos defectos encontrados por las pruebas deben ser registrados en la «Bitácora de pruebas» para su respectivo análisis y su posible solución. La corrección de los defectos encontrados es responsabilidad del equipo de desarrollo.

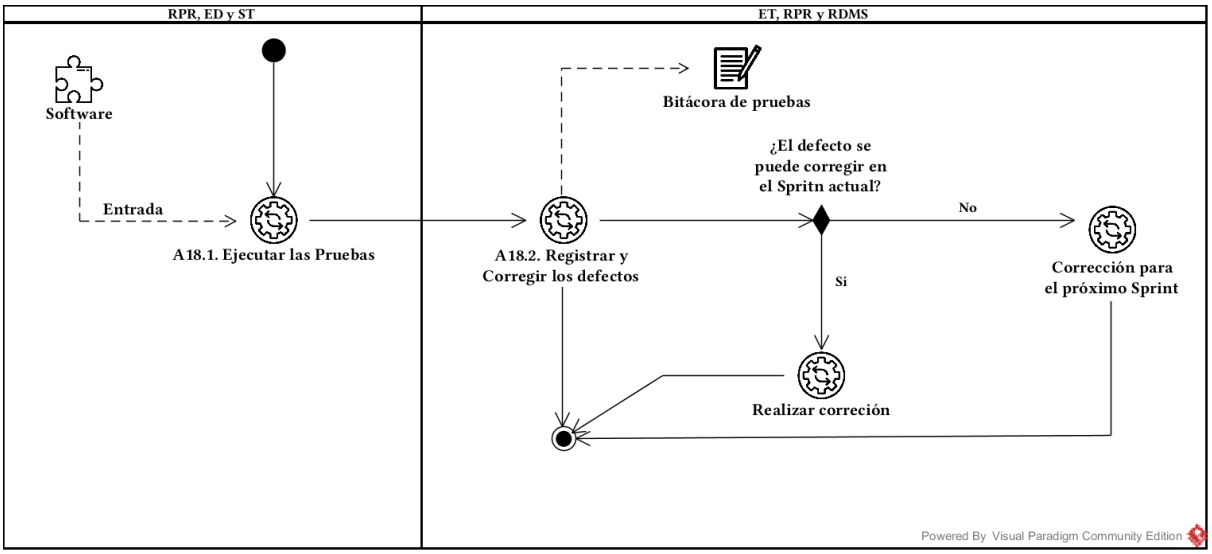


***Ilustración 18**. Realizar pruebas al Software*

A19. Conformar los productos para la entrega.

A19.1. Elaborar versiones finales de los manuales. Actualizar los manuales para su incorporación en la configuración del software. Los manuales ya han sido verificados desde la fase de desarrollo, se pude realizar otra verificación para encontrar detalles que se pasaran por alto en etapa anterior o nuevos cambios agregados por esta actividad.

A19.2. Realizar la reunión de entrega. De manera planteada al principio de la definición del proyecto se entrega al cliente lo estipulado en el «Protocolo de entrega».

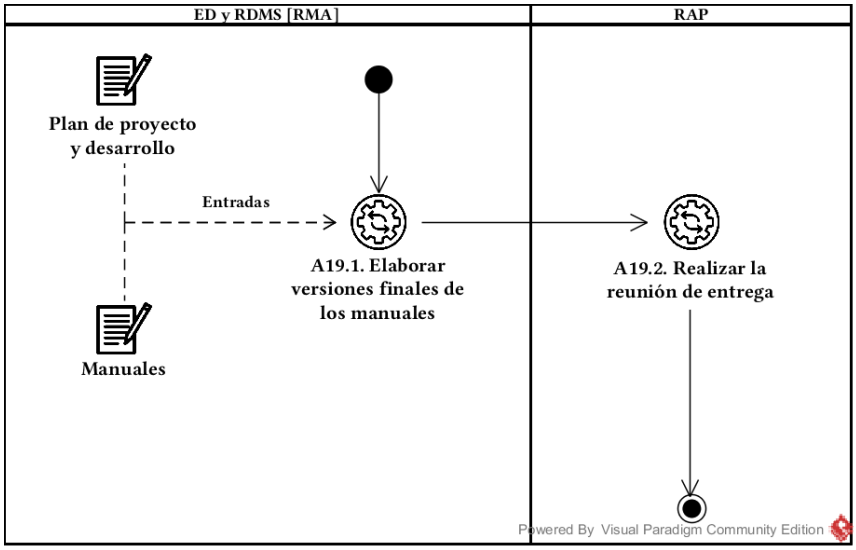


***Ilustración 19.** Conformar los productos para la entrega*

A20. Presentar los informes de actividades.

A20.1. Revisar consistencia de datos. Verificar la consistencia de los reportes de actividades en base a las fechas de inicio y fin, y los responsables de cada actividad. En esta actividad se registran o actualizan aquellas actividades pendientes por los integrantes del equipo de desarrollo.

A20.2. Elaborar los reportes. Seleccionar la información necesaria para realizar los reportes que se entregaran a la gerencia para mantenerlos informados sobre el estado del proyecto.

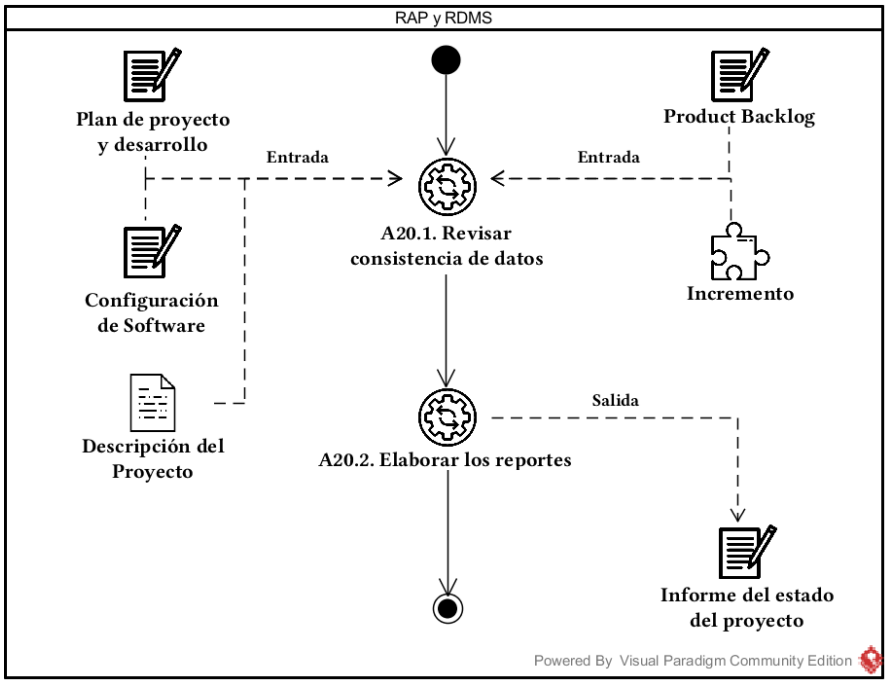


Ilustración 20. *Presentar los informes de actividades*

A21. Realizar los respaldos de información.

A21.1. Realizar el respaldo del Software. El nuevo software generado se guarda o mantiene seguro en un repositorio, del cual se puede generar nuevas versiones sin afectar el original. Este respaldo también incluye la documentación generada por el proceso de desarrollo.

A21.2. Incorporar el nuevo conocimiento a la organización. Incorporar el conocimiento del proyecto a la Base de Conocimiento de la organización o similares, esto se realiza con el fin de enriquecer a los demás equipos con posibles mejoras, conocimiento tácito y solución a problemas.

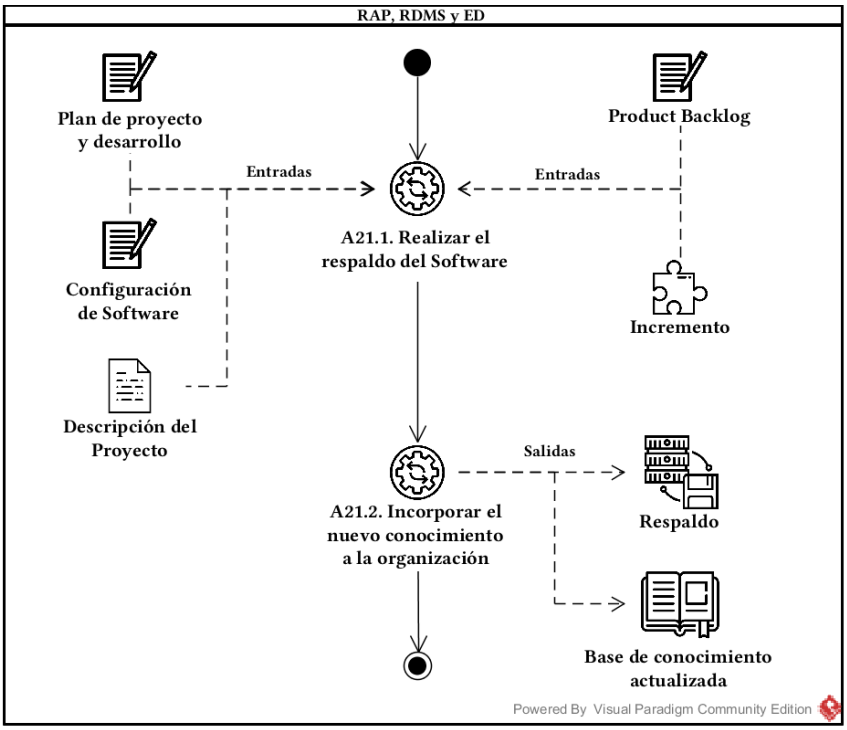


***Ilustración 21**. Realizar los respaldos de información*

Iteración de Trabajo PSP

Flujo de trabajo

"…es la secuencia de actividades para adoptar el flujo de trabajo de PSP, cada nivel agrega nuevas actividades al proceso y establece técnicas de medición y control de las actividades realizadas por los programadores…"

Nivel 0

La ***planeación*** comienza con obtención de los requisitos de la tarea y la definición clara y sin ambigüedades, para después realizar una estimación empírica del tiempo que se necesita para terminar la tarea asignada. Como se muestra en la *Ilustración 22* el registro del tiempo es automático gracias a la herramienta *PSP Dashboard*.

El ***desarrollo*** (*Ilustración 22*) comienza la elaboración de un *diseño* que cumpla con los requisitos obtenidos en la *planeación* y registra cualquier defecto encontrado durante estas actividades conforme la lista de defectos que proporciona *PSP* o si el desarrollador lo prefiere elaborar su propia lista. De la misma manera, al realizar la traducción del diseño a líneas de *código* se tiene que registrar cualquier defecto encontrado. Una vez terminado la escritura del

código se procede a realizar la *compilación*, es aquí donde se corrigen la mayoría de los defectos encontrados en las actividades anteriores, por lo general son defectos de sintaxis. De igual manera, al realizar las pruebas necesarias al código se tienen que corregir y registrar los defectos eliminados. Al finalizar esta actividad se tiene un programa completamente probado.

El ***Post-mortem*** es la última actividad, su objetivo principal es verificar la *consistencia de los datos*. Se comienza por revisar que la mayoría de los defectos estén registrados y verificar que el número de defectos encontrados y eliminados sea razonable, de lo contrario registrar aquellos datos olvidados. Como se muestra en la *Ilustración 22*, la finalizar se tienen el programa en ejecución y el plan de taras completado.

Nivel 0.1

"… se tienen una perspectiva de los conceptos que define el flujo de trabajo PSP, como se manejó en capítulos anteriores cada nivel solo agrega nuevas actividades a las ya establecidas..."

En la ***planeación*** (*Ilustración 23*) se agrega el estándar de codificación y conteo que en actividades posteriores se utilizara para medir el programa desarrollado. Después de terminar la definición de requisitos se tiene que *estimar el tamaño*, en líneas de código, del programa a realizar. Posteriormente al realizar la estimación del tiempo, la herramienta *PSP Dashboard distribuye el tiempo sobre las fases correspondientes*.

La *Ilustración 23* muestra la implementación del *código* se *realiza utilizando el estándar de codificación* que se estableció en la planificación. Las demás actividades siguen sin cambio para el ***desarrollo***, al finalizar estas actividades se obtiene un *programa probado conforme a un estándar*.

En el ***Post-mortem*** *(Ilustración 23)* se agrega la actividad de tamaño de LOC y consiste en contar las líneas de código que se utilizaron en el desarrollo del programa. *PSP Dashboard* ofrece una función muy útil para realizar este cálculo.

Nivel 1.0

"...se añaden dos nuevos documentos, la *plantilla de estimación de tamaño* y los *datos históricos.* El primero es un formulario que sirve para realizar estimaciones de tamaño con ayuda del método *PROBE*, el segundo son datos que se han registrado desde el inicio del proceso y sirven para realizar una mejor estimación de tamaño y tiempo..."

Durante la ***planeación*** *(Ilustración 24)*, una vez que se realiza la definición de requisitos se elabora un diseño conceptual del módulo o programa a desarrollar, utilizando las plantillas que proporciona *PSP Dashboard* es posible realizar los cálculos del método PROBE *para estimar*

el tamaño del programa. A continuación, utilizar el mismo método de estimación a *para calcular el tiempo de desarrollo*, de *manera automática el tiempo estimado se distribuye sobre las fases del proceso*.

En la actividad de ***desarrollo*** se agrega al finalizar las pruebas, la *elaboración de un informe de las pruebas* realizadas al programa que se tiene registradas en la ***bitácora de pruebas*** *(Ilustración 24)*.

En el *Post-mortem*, específicamente después de la actividad de revisar la *consistencia de datos*, se tiene *determinar el tamaño base, agregado, eliminado, modificado y reusado del programa* gracias a las funcionalidades de PSP Dashboard es posible comparar dos archivos alojados en repositorios *(Ilustración 24)*.

Nivel 1.1

"…PSP establecen que se utilice un plan de planeación de actividades para aquellas que duren varios días; sin embargo, en esta adaptación ese plan está controlado por la planeación del Sprint, que no sobre pasa los 15 días. Por lo tanto, este nivel no tiene ningún cambio en cuanto a las actividades del proceso (*Ilustración 25*) …"

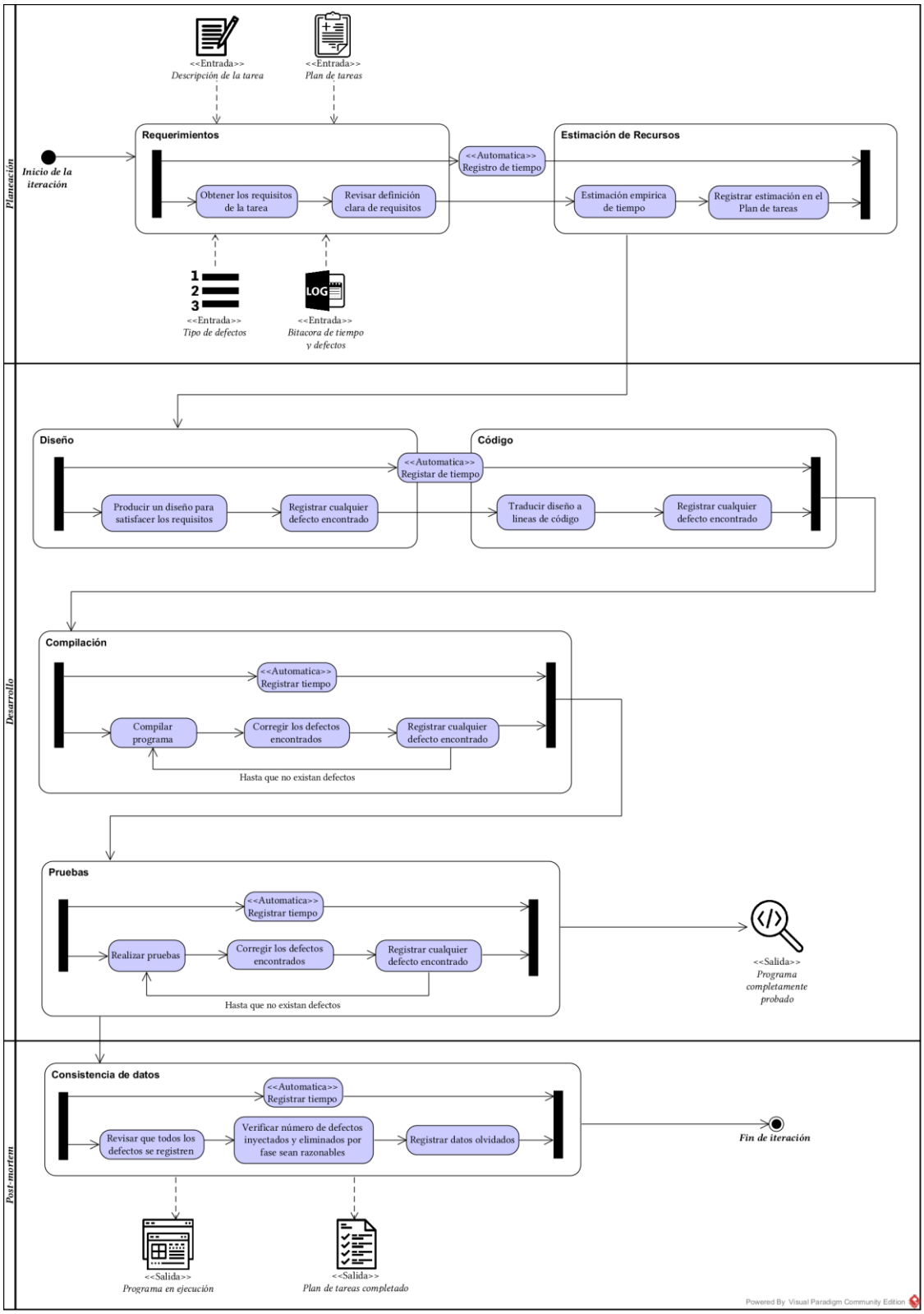


Ilustración 22. *Nivel de 0 de PSP*

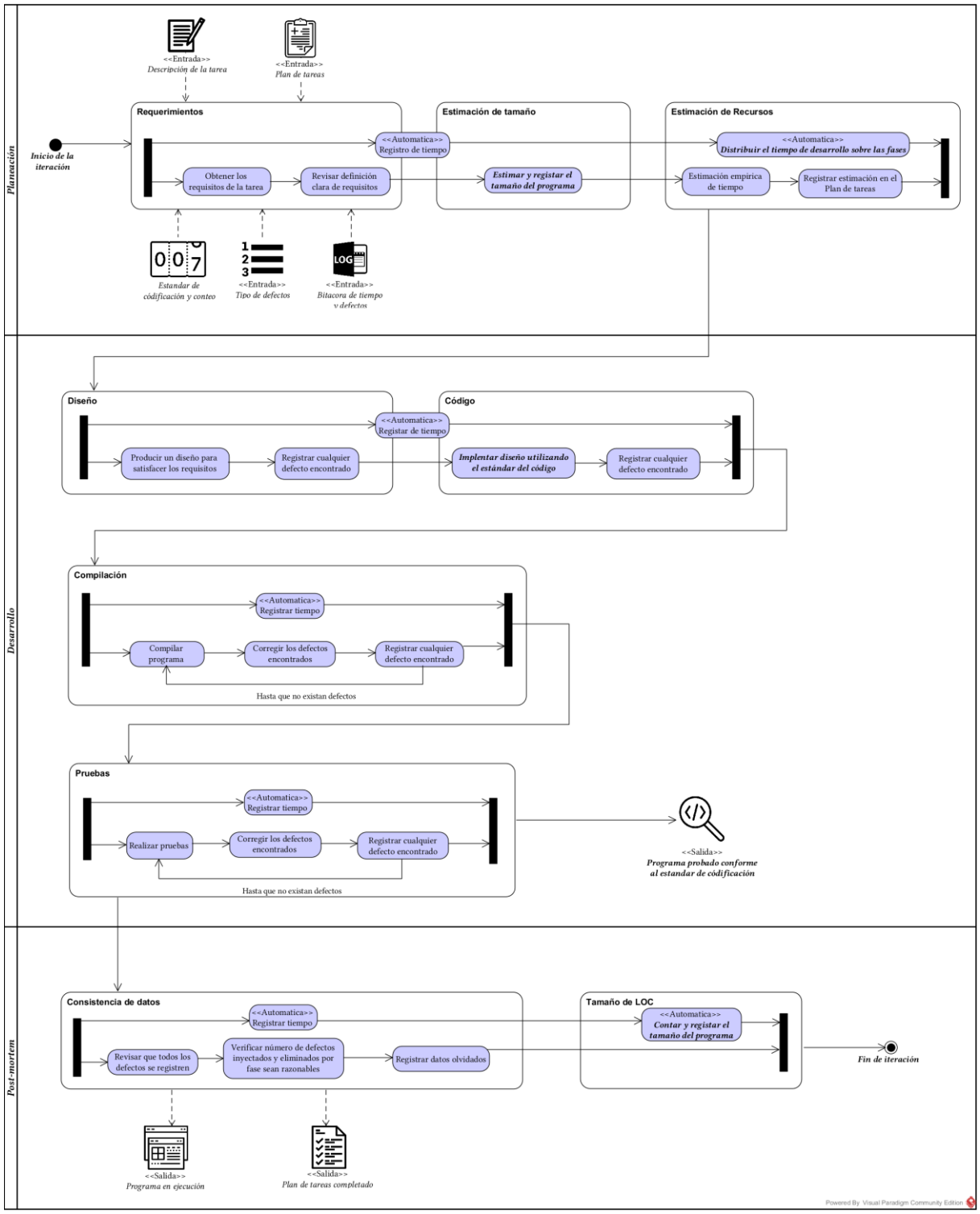


Ilustración 23. *Nivel de 0.1 de PSP*

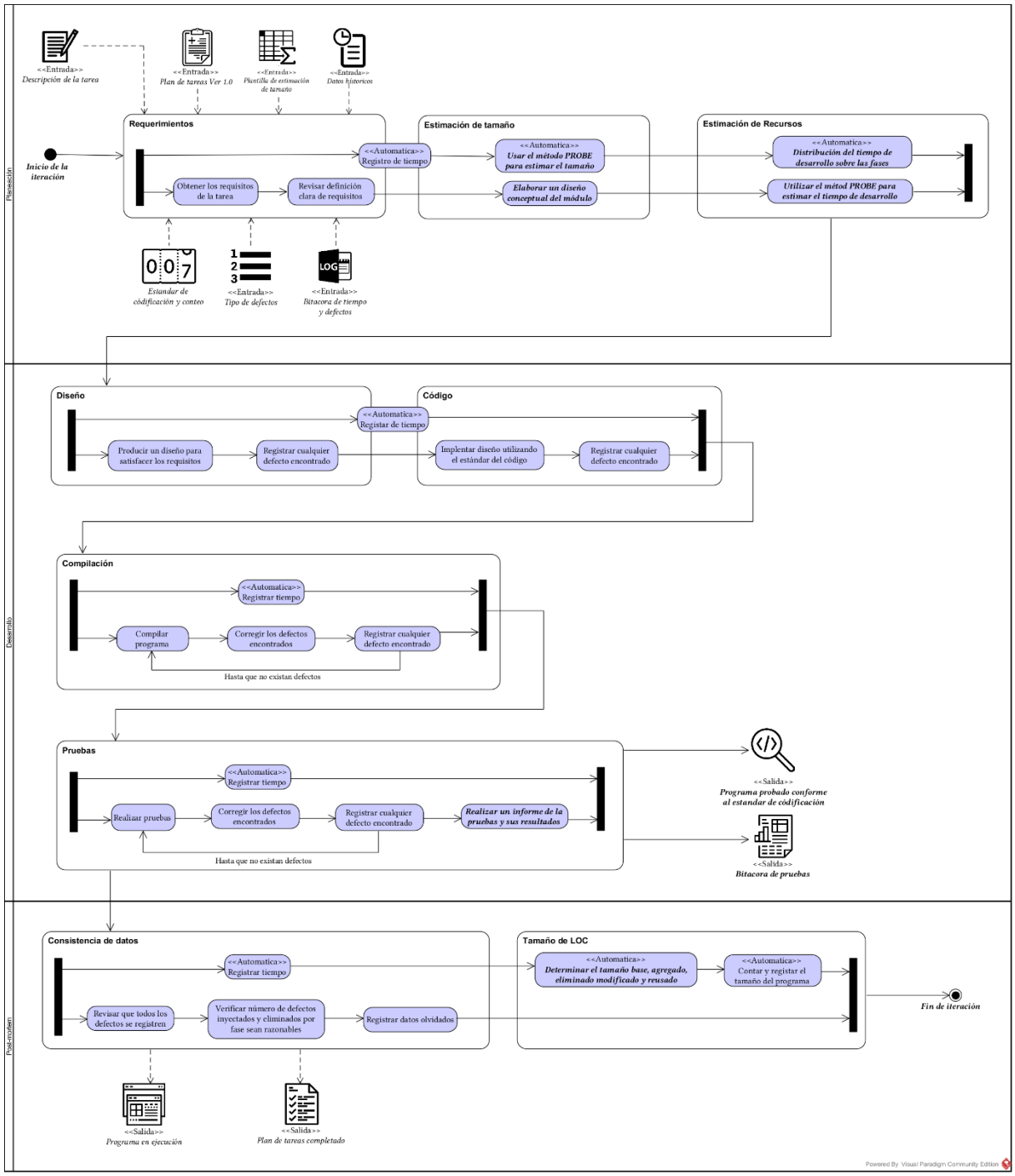


Ilustración 24. *Nivel de 1.0 de PSP*

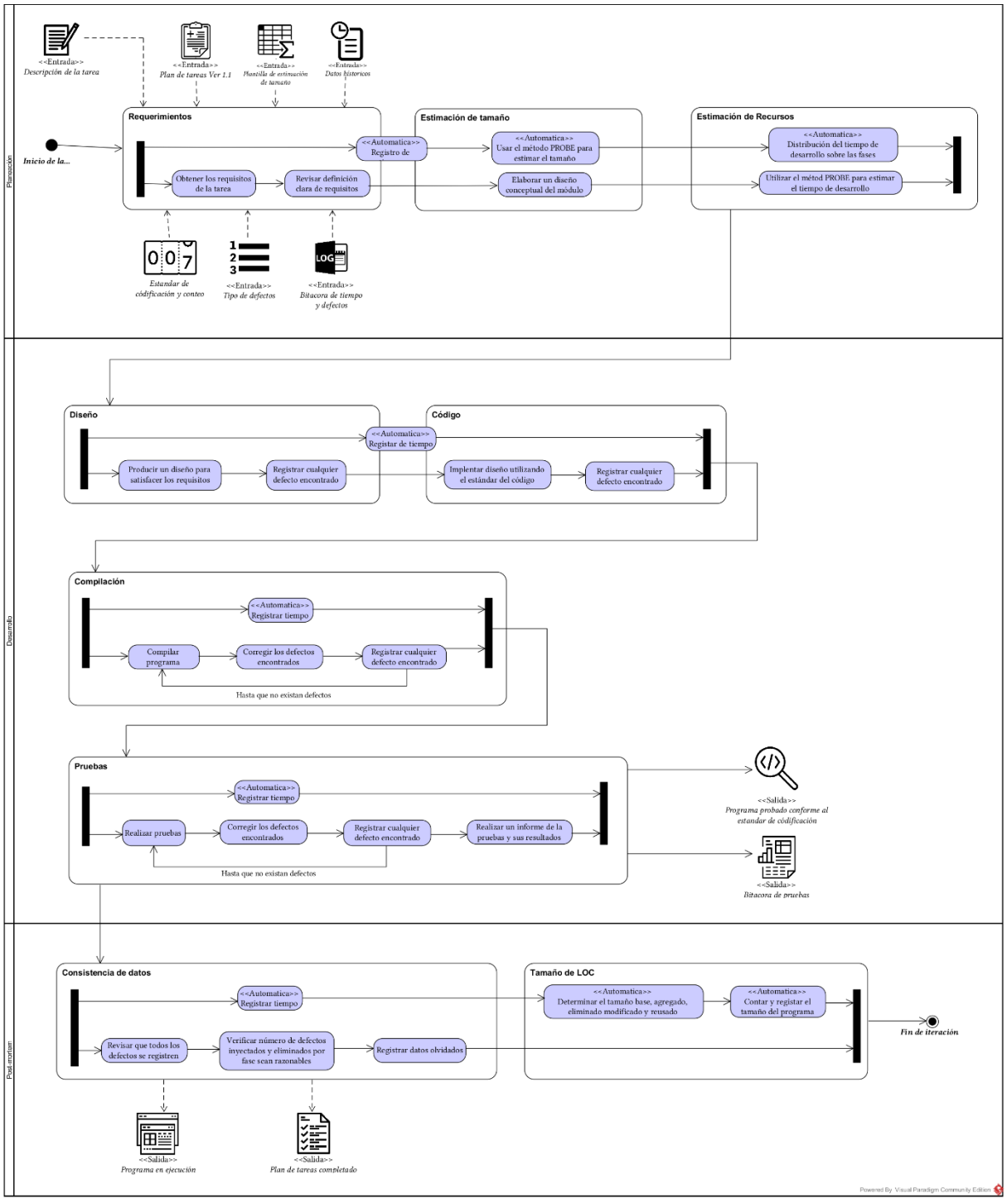


Ilustración 25. *Nivel de 1.1 de PSP*

Cálculos para el caso de estudio

En esta sección se explica el proceso de calificación para el caso de estudio descrito en este documento para el *ciclo de vida de MSP*.

En primer lugar, se revisó el contenido de cada artefacto que el equipo de desarrollo elaboraba y se realizó una comparación contra los artefactos que se establecieron en el modelo propuesto, cada artefacto está compuesto por diferentes secciones, se revisó que el contenido de cada sección fuera lo más completo a la definición descrita en modelo propuesto.

En la *Tabla A-1* se muestra que las secciones que presentan ● tiene una compatibilidad buena con la definición del modelo, por lo cual reciben un valor de *1*; aquellos que presentan ◑ tiene una compatibilidad regular y reciben un valor de *0.5*; por último, aquellos que presentan una calificación ○ no se realizaron de manera adecuada y reciben un valor de *0*. La última columna refleja la calificación final del artefacto elaborado por el equipo de desarrollo, el cual se calcula por promedio de los demás valores.

Como siguiente paso, se realizó una compaginación entre la actividades realizadas y contempladas por el equipo de desarrollo y las definidas en el modelo propuesto, para obtener la calificación que se muestra en la *Tabla A-2* en la columna "*calificación de actividad"* se utilizó el mismo cálculo con el que se obtuvieron los resultados de los artefactos.

Cabe mencionar que también evaluaron las entradas y salidas que se generan en cada actividad, para las primeras el conjunto de artefactos que forman una entrada se consideró como un solo objeto; por ejemplo, para la actividad "*A.5 Priorización de requisitos"* se tiene como entrada el **Product Backlog**, *Plan de manejo de riesgo* y *Calendario de trabajo* cada una de estas entradas tiene una calificación de acuerdo a la *Tabla A-1* para obtener su valor como "*entrada"* se realiza un promedio de la calificaciones que obtuvieron.

Por último, para obtener la evaluación final de las fases se juntaron los porcentajes de las calificaciones de actividades y artefactos, y con ellos se obtuvo un promedio. En la *Tabla A-3* en la columna promedio final se muestra el porcentaje de compatibilidad de cada fase del modelo con las actividades realizadas por el equipo de desarrollo.

Tabla A 1. *Calificación para los artefactos*

Artefacto	Secciones	Calificación	Porcentaje total
Descripción del proyecto	Propósito	●	75%
	Objetivos	●	
	Alcance del proyecto	●	
	Entregables	◑	
	Supuestos	◑	
	Restricciones	◑	
Plan del Proyecto y Desarrollo	Tipo de proyecto	●	86%
	Descripción del producto	●	
	Calendario de trabajo	●	
	Equipo de trabajo	●	
	Costos estimados	●	
	Plan de manejo de riesgos	○	
	Protocolo de entrega	●	
Configuración de Software	Especificación de requerimientos	●	50%
	Análisis y Diseño	◑	
	Software	●	
	Plan de calidad	◑	
	Plan de pruebas del software	○	
	Manuales	○	
Product Backlog	Lista de historias de usuario	●	100%
	Definición de completo	●	
	Priorización de requisitos	●	
Backlog de liberación	Selección de requisitos más urgentes	●	100%
	Estimación de costos o esfuerzo	●	
Sprint Backlog	Descomposición de requisitos en tareas	●	100%
	Repartir tareas a cada integrante	●	
	Progreso de las tareas	●	
Incremento	Software funcional	●	50%
	Documentación	○	

Tabla A 2. *Calificación para las actividades en conjunto con los artefactos*

Actividad padre	Actividades hijas	Calificación de actividad	Artefacto de entrada	Artefacto de salida	Calificación del artefacto
A1. Revisar la «Descripción del proyecto»	A1.1. Examinar/actualizar la «Descripción del Proyecto»	●	**Descripcion del proyecto**		75%
	A1.2. Elaborar la descripción del producto	●		*Descripción del producto*	100%
	A1.3. Establecer el «Protocolo de entrega»	●		*Protocolo de entrega*	100%
A2. Ingeniería de requisitos	A2.1. Captura de requerimientos	●			
	A2.2. Definición de requisitos	●			
	A2.3. Verificación y validación de requisitos	◑	**Descripcion del proyecto**		75%
	A2.4. Elaborar o actualizar del Product Backlog	●		**Product Backlog**	100%
A3.Planificación estratégica de recursos	A3.1. Identificar los recursos de trabajo	◑	**Product Backlog**		100%
	A3.2. Formar el Equipo de Trabajo	●		*Equipo de trabajo*	100%
	A3.3 Estimar costos generales	●		*Costos estimados*	100%
	A3.4 Generar un calendario de trabajo	◑		*Calendario de trabajo*	100%
A4. Gestión de riesgos	A4.1 Identificar los posibles riesgos	○	**Plan de proyecto y desarrollo**		86%
	A4.2 Realizar un análisis de Riesgos	○			
	A4.3 Elaborar planes de contingencia	○		*Plan de manejo de riesgos*	0%
A5. Priorización de requisitos	A5.1. Seleccionar requisitos	●	**Product Backlog,** *Plan de manejo de riesgo, Calendario de trabajo*		66.67%
	A5.2. Realizar estimaciones de esfuerzo	●			
	A5.2. Generar el «Backlog de Liberación»	●		**Backlog de liberación**	100%
A6. Realizar el «Plan de proyecto y desarrollo»	A6.1. Conformar el plan del proyecto y desarrollo	◑	*Calendario, Protocolo de entrega, Plan de manejo de riesgos, Costos estimados, Descripcion del producto*		80%
	A6.2 Verificar el plan del proyecto y desarrollo	○			
	A6.3. Validar el plan del proyecto y desarrollo	○		**Plan de proyecto y desarrollo**	80%
A7. Planificación del Sprint	A7.1. Revisar el plan del proyecto y desarrollo	●	**Plan de proyecto y desarrollo**		80%
	A7.2. Recibir y analizar solicitudes de cambio	●			
	A7.3. Generar el Sprint Backlog	●		**Sprint backlog**	100%
	A7.4. Distribuir las tareas al equipo de desarrollo	●			
	A7.5. Definir y elaborar un plan de calidad	○		*Plan de calidad*	50%
A8. Generar/actualizar los manuales del sistema	A8.1. Redactar los manuales	○	**Plan del Proyecto y Desarrollo, Configuración de Software, Incremento**		57.33%
	A8.2. Revisar los manuales	○		**Manuales**	0%
A9. Gestionar pruebas del sistema	A9.1. Revisar los requisitos y el diseño	◑	**Sprint backlog**, *Esp. Requerimientos, Analisis y diseño*		83.33%
	A9.2. Planificar las pruebas	○		*Plan de pruebas del software*	0%
	A9.3. Diseñar las pruebas	○			
	A9.4. Ejecutar las pruebas	◑			
	A9.5. Reportar los resultados	○		*Bitacora de pruebas*	0.00%
A10. Iteración de trabajo PSP	A10.1 Planeación	◑	**Sprint backlog**		100%
	A10.2 Desarrollo	◑		*Analisis y diseño*	0.00%
	A10.3 Postmortem	◑		*Bitacora de pruebas*	0.00%
A11. Realizar revisiones diarias	A11.1. Inspeccionar trabajo realizado	●	**Sprint backlog**		100%
	A11.2. Revisar peticiones de cambios	●			
A12. Revisión del Sprint	A12.1. Revisiar las actividades realizadas	●	**Sprint backlog**		100%
	A12.2. Ejecutar la demostración del incremento	●		**Incremento**	50%
A13. Retrospectiva del Sprint	A13.1. Inspeccionar el último Sprint	●	**Incremento, Plan de proyecto y desarrollo , Configuración del Software**		59.33%
	A13.2. Identificar lo bueno y lo malo	●			
	A13.3. Crear un plan de mejora	●		Plan de mejora	100%
A14. Obtener Incremento	A14.1. Actualizar los Backlog del desarrollo	●	**Incremento**		50%
	A14.2. Actualizar los manuales del sistema	○		**Manuales**	0%
A15. Realizar pruebas al «Incremento».	A15.1. Ejecutar las Pruebas	○	**Incremento,** *Plan de pruebas del software*		25%
	A15.2. Registrar y Corregir los defectos	○		*Bitacora de pruebas*	0%
A16. Generar/actualizar la «Configuración del Software»	A16.1. Conformar la «Configuración del Software»	○	*Especificación de requerimientos, Análisis y Diseño, Software, Plan de calidad, Plan de pruebas del software, Manuales*		50%
	A16.2. Revisar la «Configuración del Software»	○		**Configuración de Software**	50%
A17. Integrar nuevo «Incremento».	A17.1. Realizar Integración	●	**Incremento**		50%
	A20.2. Evaluar el cumplimiento del Plan del Proyecto	○			
A18. Realizar pruebas al Software.	A18.1. Ejecutar las pruebas.	○	*Software, Plan de pruebas de software*		50%
	A18.2. Registrar y corregir los defectos.	○		*Bitacora de pruebas*	0%
A19. Conformar los productos para la entrega.	A19.1. Elaborar versiones finales de los manuales.	○	**Incremento, Manuales**		25%
	A19.2. Realizar la reunión de entrega.	●		**Manuales**	0%
A20. Presentar los informes de actividades.	A20.1. Revisar consistencia de datos.	○	**Descripción del proyecto, Plan del Proyecto y Desarrollo, Configuración de Software, Product Backlog, Incremento**		72%
	A20.2. Elaborar los reportes.	○		**Informe del estado del proyecto**	0%
A21. Realizar los respaldos de información.	A21.1. Realizar el respaldo del Software.	●	**Descripción del proyecto, Plan del Proyecto y Desarrollo, Configuración de Software, Product Backlog, Incremento**		72%
	A21.2. Incorporar el nuevo conocimiento a la organización.	○		**Base de conocimiento, Respaldo**	50%

Tabla A 3. *Calificación total por fases*

Actividad padre	Calificación por actividades	Calificación por artefacto	Promedio final
A1. Revisar la «Descripción del proyecto»	100%	91.67%	95.83%
A2. Ingeniería de requisitos	87.50%	87.50%	87.50%
A3.Planificación estratégica de recursos	75%	100%	87.50%
A4. Gestión de riesgos	0%	43%	0%
A5. Priorización de requisitos	100%	83.33%	91.67%
A6. Realizar el «Plan de proyecto y desarrollo»	16.66%	80%	48.33%
A7. Planificación del Sprint	80%	76.67%	78.33%
A8. Generar/actualizar los manuales del sistema	0%	28.67%	14.33%
A9. Gestionar pruebas del sistema	20%	27.78%	23.89%
A10. Iteración de trabajo PSP	50%	33.33%	41.67%
A11. Realizar revisiones diarias	100%	100%	100%
A12. Revisión del Sprint	100%	75%	87.50%
A13. Retrospectiva del Sprint	100%	79.67%	89.83%
A14. Obtener Incremento	50%	25%	37.50%
A15. Realizar pruebas al «Incremento».	0%	12.50%	6.25%
A16. Generar/actualizar la «Configuración del Software»	0%	50%	25%
A17. Integrar nuevo «Incremento».	50%	50%	50%
A18. Realizar pruebas al Software.	0%	50%	25%
A19. Conformar los productos para la entrega.	50%	12.50%	31.25%
A20. Presentar los informes de actividades.	0%	36.10%	18.05%
A21. Realizar los respaldos de información.	50%	61.10%	55.55%

Material de apoyo

Para realizar la capacitación se utilizó el material que proporciona el Instituto de Ingeniería de Software (SEI) en colaboración con la Universidad de Carnegie Mellon, el cual se puede descargar del siguiente enlace: *http://www.sei.cmu.edu/tsp/tools/academic/index.cfm*.

Para el registrar los datos del marco de trabajo se utlizó el software *PSP Dashboard* de *Tuma Solutions*, en su versión *2.2 student*, el cual se puede descargar del siguiente enlace: http://www.processdash.com/pspForEng.

Productos académicos

En revista indizada en latindex y periódica

Mauricio Leonardo Urbina Delgadillo, M. Antonieta A. Figueroa, G. Peláez Camarena, G. Alor Hernández, A. Ivonne Sánchez García. "*Mixing Scrum-PSP: Combinación de Scrum y PSP para mejorar la calidad del proceso de software"*. Research in Computing Science, Vol. 126, pp. 19-29, ISSN 1870-4069. Revista editada por el Centro de Investigación en Computación del IPN, México 2016.

Mauricio Leonardo Urbina Delgadillo, María Antonieta Abud Figueroa, Gustavo Peláez Camarena, Giner Alor Hernández, Alma Ivonne Sánchez García. *"Propuesta de un modelo de integración de PSP y Scrum para mejorar la calidad del proceso de desarrollo en una MiPyME*". Research in Computing Science, Vol. 120, pp. 147–157, ISSN 1870-4069. Revista editada por el Centro de Investigación en Computación del IPN, México 2016.

En congreso internacional

Mauricio Leonardo Urbina Delgadillo, María Antonieta Abud Figueroa, Gustavo Peláez Camarena, Giner Alor Hernández, Alma Ivonne Sánchez García. *"Propuesta de un modelo de integración de PSP y Scrum para mejorar la calidad del proceso de desarrollo en una MiPyME*", 16th edición del International Congress on Computer Science (CORE 2016).

Mauricio Leonardo Urbina Delgadillo, M. Antonieta A. Figueroa, G. Peláez Camarena, G. Alor Hernández, A. Ivonne Sánchez García. "*Mixing Scrum-PSP: Combinación de Scrum y PSP para mejorar la calidad del proceso de software",* X Congreso Internacional en Tecnologías Inteligentes y de la Información (CITII 2016).

Bibliografía

[1] Nielsen, P.: Struggles at the Frontier of Software Engineering. , *TSP* Symposium, "Going beyond agile", D.F., México (2016).

[2] E. Karch, "*The Software Crisis: A Brief Look at How Rework Shaped the Evolution of Software Methodolgies*", Karchworld Identity, 2011. Disponible: https://blogs.msdn.microsoft.com/karchworld_identity/2011/04/04/the-software-crisis-a-brief-look-at-how-rework-shaped-the-evolution-of-software-methodolgies/.

[3] "*Ingeniería del Software | Historia de la Informática*", Histinf.blogs.upv.es, 2010. Disponible: http://histinf.blogs.upv.es/2010/12/28/ingenieria-del-software/.

[4] J. Over, "*Back to The Future*", International *TSP* Symposium, Mexico City, Mexico, 2016.

[5] E. Dijkstra, "*The humble programmer*", Communications of the ACM, vol. 15, no. 10, pp. 859-866, 1972.

[6] F. Brooks, "*No Silver Bullet Essence and Accidents of Software Engineering*", Computer, vol. 20, no. 4, pp. 10-19, 1987.

[7] Diccionario de Real Academia Española, s.v. "Calidad", http://buscon.rae.es/drae/srv/search?val=calidad.

[8] Norma Internacional ISO, 9000, 2005.

[9] Standards Coordinating Committee of the Computer Society of the IEEE, *IEEE Standard Glossary of Software Engineering Terminology*. New York: IEEE Standards Board, 1990, p. 60.

[10] R. Pressman, V. Campos Olguín and J. Enríquez Brito, *Ingeniería del software*. México: McGraw-Hill, 2010.

[11] I. Sommerville and M. Alfonso Galipienso, *Ingeniería del software*. Madrid: Pearson Addison-Wesley, 2005.

[12] P. Letelier, "*Proceso de desarrollo de software*", Universidad Politécnica de Valencia., Valencia. Disponible en: http://ldc.usb.ve/~abianc/materias/ci4712/ProcesoSW-Letelier.pdf.

[13] C. Carrillo Pardo, J. González Rebanal, D. Gómez Martínez, L. Silva García, M. Castilla Álvarez and P. López Martínez, *Auxiliares de Enfermería Del Servicio de Salud de Castilla Y Leon*. Sevilla, España: EDITORIAL MAD, S.L., 2005, p. 406.

[14] Piattini, Mario, García Félix O, "*Calidad en el desarrollo y mantenimiento del software*", RA-MA Editorial, Madrid, 2003.

[15] Asociación Española para la Calidad, s.v., "*Modelos de calidad*", http://www.aec.es/web/guest/centro-conocimiento/modelos-de-calidad.

[16] "*Nyce*", Moprosoft.com.mx, 2016. Disponible: http://www.moprosoft.com.mx/contenido.aspx?id_pagina=8.

[17] Secretaría de Economía, "*Modelo de Procesos para la Industria de Software, MoProSoft, Por Niveles de Capacidad de Procesos*", Hanna Oktaba, México, 2005.

[18] *Scrum* MEXICO. *Scrum & Agile*. (2014). Disponible en: http://*Scrum*.org.mx/*Scrum*-agile/.

[19] Palacio, J. and Ruata, C. *Scrum Manager Gestión de Proyectos*. 4th ed. SafeCreative, pp.57-87. (2010). Disponible en: http://www.safecreative.org/work/1012268137397

[20] Palaco, J. *Flexibilidad con Scrum*. 2nd ed. pp.87, 125-176. (2007). Disponible en: http://www.safecreative.org/work/0710210187520

[21] W. Humphrey, *PSP*. Upper Saddle River, NJ: Addison-Wesley, 2005.

[22] F. Scalone, "*Estudio Comparativo De Los Modelos Y Estándares De Calidad Del Software"*, Licenciatura, Universidad Tecnológica Nacional Facultad Regional Buenos Aires, 2006.

[23] "*JIRA Software - Issue & Project Tracking for Software Teams | Atlassian*", *Atlassian*, 2016. [Online]. Disponible en: https://www.atlassian.com/software/jira.

[24] "*Confluence - Team Collaboration Software | Atlassian*", *Atlassian*, 2016. Disponible en: https://www.atlassian.com/software/confluence.

[25] Boehm, B. 2002. "Get Ready For Agile Methods, With Care". *Computer* 35 (1): 64-69. doi:10.1109/2.976920.

[26] Paulk, Mark C. 2002. "Agile Methodologies And Process Discipline". *CROSSTALK - The Journal Of Defense Software Engineering* 15 (10): 15-18. http://www.crosstalkonline.org/back-issues/.

[27] Sutherland, Jeff, Carsten Ruseng Jakobsen, and Kent Johnson. 2007. "*Scrum* And CMMI Level 5: The Magic Potion For Code Warriors". *AGILE 2007*. doi:10.1109/agile.2007.52.

[28] Nawrocki, J., B. Walter, and A. Wojciechowski. "Toward Maturity Model For Extreme Programming". *Proceedings 27Th EUROMICRO Conference. 2001: A Net Odyssey*. doi:10.1109/euromic.2001.952459.

[29] Dzhurov, Y. et al.: Personal Extreme Programming – An Agile Process for Autonomous Developers. Proceedings of the International Conference on Software, Services & Sematic Technologies. (2009).

[30] Brown, D… 2014. "*PSP* Implementations for agile methods: a SEMAT-based approach". *Software Engineering: Methods, Modeling, and Teaching*. 3, 41-45 (2014).

[31] Lina, Z.Dan, S.: Research on Combining *Scrum* with CMMI in Small and Medium Organizations. 2012 International Conference on Computer Science and Electronics Engineering. (2012).

[32] Bougroun, Z., Zeaaraoui, A., Bouchentouf, T.: The projection of the specific practices of the third level of CMMI model in agile methods: *Scrum*, XP and Kanban. 2014 Third IEEE International Colloquium in Information Science and Technology (CIST). (2014).

[33] Soares, F., de Lemos Meira, S.: An agile strategy for implementing CMMI project management practices in software organizations. 2015 10th Iberian Conference on Information Systems and Technologies (CISTI). (2015).

[34] Arauz Ortiz, G., Morales Trujillo, M., Oktaba, H., Ramirez Hernandez, E.: Integrating Agile Methods into a Level 5 CMMI-DEV Organization: a Case Study. IEEE Latin America Transactions. 14, 1440-1446 (2016).