Universidad Técnica Federico Santa María
Departamento de Informática
Valparaíso - Chile

Diseño e implementación de una metodología de desarrollo ágil en una PYME TIC carente de metodología de desarrollo

Daniel Héctor Santibáñez Vera

Memoria para optar al título de Ingeniero Civil Informático

Profesor Guía: Luis Hevia

Abril 2017

DEDICATORIA

A mi madre, pareja e hija que tuvieron la paciencia para verme finalizar este trabajo y a todos aquellos, amigos y compañeros que me recordaron día a día que esto debía ser terminado.

Resumen

Resumen— En este trabajo se diseñó e implementó una metodología de desarrollo de software incluyendo la adopción herramientas que son usadas para soportar su funcionamiento, tanto herramientas para definir y mantener la metodología como aquellas necesarias para ejecutar las actividades descritas. La metodología se diseñó a partir de la observación de cómo se ejecuta un proyecto de desarrollo en una PYME para luego hacer mejoras sobre el mismo y se propone como una metodología inspirada en principios y prácticas ágiles y *lean*, que otras PYMES TICs en situaciones similares pueden usar como una forma de disminuir la incertidumbre y hacer eficiente uso de sus recursos.

Se concluye que cada PYME es distinta y que la metodología diseñada se utilice como una herramienta que puede ser modificada de acuerdo con la situación particular y que en el caso de la empresa en cuestión lográ buenos resultados. También se indica que la adopción de una nueva "forma de hacer" debe estar acompañada de un correcto liderazgo que incentive e inspire a los equipos que harán uso de ella. La metodología trata de implementar un "hacer agil" y no un "ser ágil" que se identifica como un fin más deseable en donde siquiera una metodología definida o estandard es necesaria.

Palabras Claves— ADAPTE, EPF, Ágil, Lean, SCRUM, Liderazgo, PYMEs, TIC

Abstract

Abstract— This thesis present the design and implementation of a software development methodology and the adoption of tools that support its operation and execution. Tools are between those of purpose of definition and maintenance as well as those to execute the activities of the methodology. The methodology was designed after the observation of several in project executions of software development projects in a SME after that improvement was made on it for a resulting methodology, inspired in agile and lean software development principles and practices, that other and similar SMEs could take advantage of and use as a way to reduce the uncertainty and optimize resources.

It's come to the conclusion that every SME is different and the methodology should be used as a tool that could be changed in order to fit the current organization state. In the case presented the methodology gives good results. Also is worth mention that every initiative adopting new "ways of doing" has to be guided with a leadership style that inspire and motivates to the teams that embrace the initiative. A last thought explain that this methodology try to adopt a "do agile" and not necessarily a "be agile", which could be a better goal because transform a organitation in a flexible entity not requiring a stNDrd or official methodology at all in many cases.

Keywords— ADAPTE, EPF, Agile, SCRUM, Leadership, SMEs, ICT

INTRODUCCIÓN

La metodología descrita en este trabajo fue desarrollada dentro de una unión temporal de la empresa con la Universidad de Chile en proyecto de investigación ADAPTE en donde en forma conjunta se diseñó y documentó la metodología. Existiendo previamente una metodología inicial con sus actividades y herramientas de soporte diseñadas a partir de la *praxis* en una serie de proyectos, se evaluaron las recomendaciones de la universidad en cuanto a qué elementos podrían ser mejorados en la metodología, desde actividades hasta las herramientas a adoptar o desechar de acuerdo a la experiencia de otras empresas también participes del proyecto. También se incorporó la posibilidad de realizar análisis de consistencia y tailoring al proceso resultante como una forma de flexibilizar y adaptar mejor la metodología al contexto de un proyecto particular usando las herramientas generadas en el proyecto de investigación.

Comienza con el Capítulo 1 en donde se define el problema a resolver caracterizando a las PYMES TIC y los problemas a los que se enfrentan al embarcarse en proyectos de desarrollo de software. En el Capítulo 2 se describe el estado del arte del uso y adopción de metodologias en las PYMES TIC, uso de herramientas y el papel del liderazgo en esta mezcla. En el Capítulo 3 se desarrolla la propuesta de solución al problema descrito en el Capítulo 1. En el Capítulo 4 se trata la validación de la solución, se entregan detalles prácticos de su uso en algunas categoria de proyectos y se discute acerca del papel del liderazgo en la adopción de metodologías. El Capítulo 5 incluye las conclusiones del trabajo realizado.

Al final de este escrito se incorporan referencias bibliográficas y detalles de las actividades de la metodologóia que se pueden ver en la definición en *SPEM* de la misma en la versión digita de esta memoria.

Al final de este escrito se incorpora un anexo que ejemplifica la navegación de la metodología en su formato como sitio web desde las fases hasta los *ouputs* de las tareas. También se ejemplifica cómo generar el sitio web para publicar la metodología usando la herramienta EPF Composer.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

1.1 Contexto

Las pequeñas y micro empresas TIC [EstatutoPYMES] (PYME, en su extensión máxima) suelen comenzar operaciones sin tener metodologías claras de desarrollo y aunque éstas existen, su implantación toma tiempo y recursos, que en tales organizaciones escasean. La motivación para comenzar pronto las operaciones lleva a operar en el caos e improvisación, con las siguientes consecuencias:

- Aparece el concepto del personaje héroe
- Las operaciones se vuelven poco productivas
- La metodología, si se instaura, cambia constantemente
- Se pierde el conocimiento
- Baja el nivel motivacional y autoestima de los equipos
- Se basa en el "Echandole pa´ adelante programming" [Villena12]
- Fracaso en los emprendimientos y proyectos

Luego cuando se hace necesario implantar una metodología, como RUP, SCRUM[1], TUTELKAN u OpenUP se vuelve caro, lento e ineficiente debido a que tales labores se deben realizar en paralelo con el resto de las operaciones ya en curso. Además, adoptar una metodología de desarrollo de software no es un proceso fácil, pues se debe buscar de entre las metodologías existentes y adaptarlas a las necesidades de la organización.

1.2 Identificación de Problemas

Sabiendo las PYME's a los problemas que se enfrentan si comienzan sus operaciones sin una metodología, pueden decidir comenzar a operar con una metodología de desarrollo de software y en este caso son muchos los desafíos y preguntas a las que se enfrentan:

- ¿Con qué conocimientos contamos?
- ¿Qué metodología de desarrollo, de las que existen, selecciono?
- Si ya he seleccionado una metodología ¿Cómo adopto, implanto y adapto?
- ¿Qué herramientas debo considerar para soportar el uso de la metodología?
- ¿Existen experiencias de PYME's que hayan enfrentado el mismo desafío, dónde están y qué se puede aprender de ellas?

Si esas preguntas son respondidas de alguna forma, la metodología de desarrollo seleccionada debe ser usada adecuadamente por los equipos de desarrollo, respondiendo al siempre cambiante y exigente ecosistema tecnológico en la que las PYME's TIC se desenvuelven, es decir:

- Demanda por menores *time-to-market*

- Demanda por mayores niveles de calidad
- Emergencia de nuevas tecnologías
- Emergencia de nuevos paradigmas tecnológicos
- Demanda por mayor innovación

Según un estudio reciente [Bastarrica16] el 80% de las PyME's que desarrollan software en el país declaran tener definido su proceso de desarrollo aunque sólo el 50% dice aplicarlo. Éste último dato se debe a que las metodologías seleccionadas no son las adecuadas para la realidad de la empresa, no son suficientemente flexibles o son muy pesadas, en referencia a la cantidad de esfuerzo que se requiere para seguirlas.

Existe, además, un importante aspecto interno de las PYME's TIC que adoptan alguna metodología de desarrollo, a saber, el papel del liderazgo que soporta:

- La iniciativa de adopción e implantación
- La gestión del cambio debido a la adopción de la metodología
- La continuación del uso de la metodología
- La mejora continua en el tiempo

Los elementos anteriores representan un aspecto muy importante para éxito de las operaciones de la organización. Un pobre o inadecuado liderazgo pueden tener como consecuencia la declinación en el uso de la metodología y en el mediano plazo la pérdida de lo invertido.

Una síntesis de los problemas identificados se elabora en el diagrama de Ishikawa en la Figura 1.1.

1 Notar que SCRUM no es una metodología es un **framework** de desarrollo de software (Ver Capítulo 2)

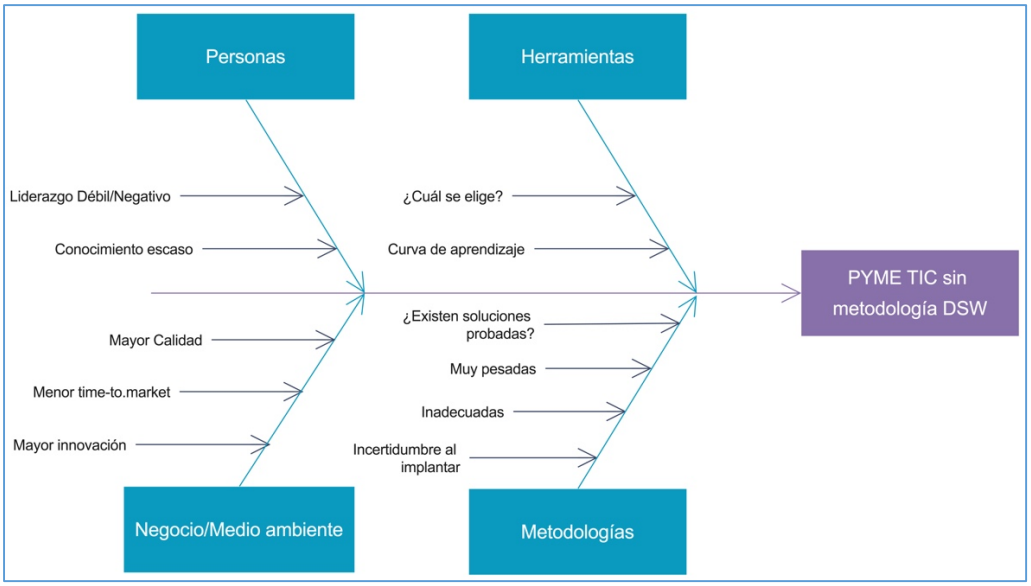

Figura 1.1: Análisis de causa-efecto de Ishikawa (Fuente: elaboración propia)

1.3 Objetivos de una Solución

1.3.1 Objetivo General

Diseñar e implementar una metodología de desarrollo ágil, junto a las herramientas que lo soportan, que permita un productivo operar de una PYME TIC.

1.3.2 Objetivos Específicos

1. Analizar el estado actual de una PYME TIC sin metodología de desarrollo de software
2. Diseñar una metodología utilizando principios y prácticas lean y ágiles
3. Proporcionar herramientas existentes que soporten la metodología diseñada
4. Validar uso efectivo de la metodología

1.4 Alcance de la Solución

La metodología de desarrollo de software propuesta está diseñada para ser aplicada en los equipos de desarrollo de software de una organización, sin perjuicio que debido a las flexibles prácticas que posee, pueda ser adaptada para ser aplicada en otras áreas en donde

se practiquen disciplinas distintas, por ejemplo: Área de Soporte y Mantenimiento de Hardware o DevOps[2].

2 DevOps: conjunto de procesos y técnicas que se utilizan en el desarrollo y operaciones de sistemas.

CAPÍTULO 2: MARCO CONCEPTUAL

Los procesos, metodologías y frameworks de desarrollo de software están en constante evolución tratando de sobreponerse a la llamada crisis del software que ya suma más de 40 años [SoftwareCrisis]. Desde los años 90 se han introducido una nueva casta de metodologías y *frameworks* de desarrollo: los llamados procesos, *framework* o metodologías **ágiles y *lean*** [Poppendieck03]. Si bien en los detalles difieren, todas comparten un núcleo común fundamentado en que los equipos se concentran en entregar software funcionando de forma continua y frecuente, experimentando, siempre mejorando y a la vez que desarrollando soluciones y satisfaciendo a sus clientes. Son formas de desarrollar software livianas y efectivas en donde además se promueve el trabajo en equipo, la autogestión, una visión compartida y el cuidado de la motivación a través de la realización de la importancia del trabajo que se realiza y el respeto hacia las personas que conforman el equipo y los clientes.

Procesos y metodologías de épocas anteriores han sido principalmente **prescriptivas**, sin embargo las nuevas "formas" de desarrollar software son más bien **descriptivas** y **adaptativas** [D'Amico12] formando frameworks que permiten gran flexibilidad al momento de poner en práctica la disciplina. Funcionan como herramientas y no como guías paso a paso que se deben seguir rigurosamente, y como herramientas pueden ser usadas correcta o incorrectamente entonces no sirviendo como una solución segura aplicable a cada caso, es decir, "No hay bala de plata" [Brooks95].

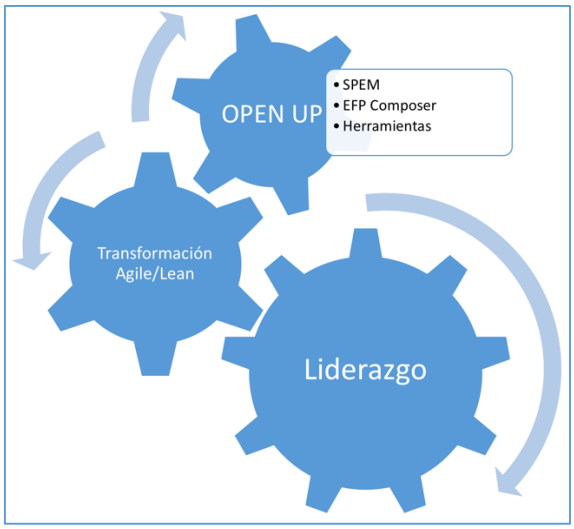

Figura 2.1: Fuente: Elaboración propia

Modernas prácticas específicas de desarrollo como *continuous integration/delivery*, visualización del flujo de trabajo con *Kanban*, tener al cliente cerca, *TDD*[3] y documentar requerimientos con historias de usuario emergen como una forma de dotar de flexibilidad a los procesos de desarrollo de software. Estas nuevas metodologías, frameworks y

3 **T**est **D**riven **D**evelopment, Desarrollo guiado por pruebas

mindsets fueron abarcando cada vez más aspectos del desarrollo de software, desde técnicos hasta un mindset de mejora continua, flexibilidad y enfoque a un objetivo aplicable a cualquier industria. Desde *XP* que se centra en prácticas específicas en el desarrollo, como *Pair Programming* y *TDD*; SCRUM [Kniberg15] que entrega un marco de trabajo flexible y de alto nivel con un conjunto de roles, artefactos y reglas de relación entre ellos [Schwaber16]; hasta el movimiento del desarrollo de software agile o lean que se refiere más bien a un *mindset* y principios que se aplican en el ejercicio de la disciplina [Fichtner].

Una evidencia del alcance amplio que poseen las nuevas formas (o nuevos sistemas de trabajo en general) para desarrollar software es que pueden servir como frameworks para actualizar metodologías tradicionales. Uno de los procesos de desarrollo prescriptivos más conocidos es *RUP*[4] que ha servido de base para la enseñanza de la disciplina y como proceso standard en la industria y que a la luz de la emergencia de las nuevas formas ha visto desarrollado una "actualización ágil" de si misma llamada *OpenUP* [OpenUP], donado por un conjunto de empresasa a la fundación Eclipse, hereda de su antepasado el carácter iterativo e incremental mientras incorpora prácticas de estas nuevas formas de desarrollar software y que sirve de punto de partida para la metodología delineada en este trabajo. Si bien *OpenUP* es agnóstico del uso de cualquier herramienta, es muy recomendable el uso de aquellas que "faciliten la comunicación, automatización y colaboración" [García11].

Para ayudar a definir y documentar una metodología existen especificaciones como *BPNM* y *SPEM*[5] [SPEM]. Éste último es un meta-modelo de especificación de procesos que hereda su nomenclatura de UML 2.0, permite ser usado en editores como *EPF Composer* [Balduino07], parte del proyecto ***E**clipse **P**rocess **F**ramework* de la fundación Eclipse, creado como un *spin-off* del *IDE Eclipse*. Permite además ser ejecutado en algunas herramientas para el análisis de procesos como *AVISPA*[6] [Hurtado11]. Proyectos de investigación como *ADAPTE*[7] hacen uso de *SPEM* para dotar de *tailoring* al proceso documentado e investigar la adecuación o adaptación de procesos a diferentes contextos de proyectos.

La piedra angular de toda transformación hacia un *mindset* ágil en la disciplina es el liderazgo que permite inspirar a equipos empoderados que se auto-organizan para practicar la excelencia técnica en un ambiente de decisiones e interacciones colaborativas [Highsmith2013][Cantor01][Mahanti06][Koutsoumpos14].

La Figura 2.1 refleja cómo el liderazgo mueve a los equipos y organizaciones hacia la adopción de nuevas y mejores formas para desarrollar software que decantan en el diseño, documentación e implantación de metodologías y herramientas.

4 RUP: **R**ational **U**nified **P**rocess

5 SPEM: **S**oftware **P**rocess **E**ngineering **M**odel

6 AVISPA: **A**nalysis and **Vi**sualization for **S**oftware **P**rocess **A**ssessment

7 ADAPTE: **A**daptable **D**omain and **P**rocess **T**echnology **E**ngineering

CAPÍTULO 3: SOLUCION PROPUESTA

La solución se basa en el diseño y documentación de una metodología de desarrollo de software cuyas características la hacen adhoc a una PYME TIC, debido a que es una metodología inspirada en *OpenUP*, que incorpora prácticas ágiles y *lean*, de fácil adopción e implementación[8] y que incluye una serie de herramientas que en la práctica facilitan la ejecución y el seguimiento de las tareas, artefactos y piezas de código de los proyectos. La descripción de la metodología toma inspiración de literatura similar como "*SCRUM and XP from the trenches*" [Kniberg15] y la "Guía SCRUM" [Schwaber16].

Descripción General

La metodología se conforma por un ciclo de desarrollo de 4 fases, cada una de las cuales tiene un conjunto secuencial de actividades y tareas que permiten lograr el objetivo de cada fase y aportar al logro de los objetivos del proyecto.

Todas las actividades se componen a su vez por tareas. Las tareas se describen a partir de los siguientes elementos:

1. Artefactos de entrada: son los artefactos necesarios que requiere la tarea y que serán transformados en los artefactos de salida por las acciones de los roles participantes. Por ejemplo, artefactos de entrada pueden ser correos electrónicos, piezas de software o documentos de diseño
2. Roles: es el conjunto de identificadores técnicos que representan las personas que participan en la ejecución de la tarea, por ejemplo, en una tarea particular las personas pueden tener los roles de programadores, analistas de *testing* o *SCRUM masters*/jefes de proyecto
3. Descripción: texto que describe el (los) objetivo(s) de la tarea, los pasos necesarios para realizarla y/o recomendaciones (mejores prácticas) al ser ejecutada por los roles participantes
4. Artefactos de salida: son los artefactos resultantes de la ejecución de la tarea y que sirven de entrada a tareas sucesoras. Por ejemplo, artefactos de salida pueden ser correos electrónicos, piezas de software o documentos de diseño

Adicionalmente, todo el proceso esta soportado por una serie de herramientas que están seleccionadas para facilitar la ejecución y seguimiento de las tareas por parte del equipo y los stakeholders (Figura 3.2). Éste conjunto de herramientas no quiere ser, de manera alguna, la única opción dentro del amplio abanico de herramientas para similares propósitos que existen en el mercado. Tampoco las herramientas son propiedad de una fase en particuar ellas se utilizan durante todo el desarrollo. Recuérdese la célebre frase en el ámbito del desarrollo de hardware "no hay bala de plata" [Brooks95], que en este caso

8 La experiencia actual del autor permite aseverar que la adopción e implementación es fácil, relativa a lo que ocurre con metodologías más pesadas.

se interpreta como que no existe una sola herramienta (o conjunto) que resuelva de forma final el problema en cuestión. Queda entonces a criterio del equipo de desarrollo seleccionar las que sean de su convencimiento y preferencia particular siempre que les permitan lograr sus objetivos de forma productiva. En esta memoria se entrega un conjunto de tales herramientas cuya utilidad, a la luz de los resultados empíricos, ha sido productiva.

La estrategia de descripción de la metodología, en este capítulo, será de alto nivel entregando los detalles de roles y artefactos de entrada/salida de cada tarea junto con el modelo navegable que se puede encontrar en la versión digital de la memoria en el *plug-in* para el *EPF Composer* 8si se quiere experimentar con el editor e incorporar mejoras.

Fases

La metodología se desenvuelve desde 4 fases en forma secuencial (ver Figura 3.1):

1. Inicio
2. Elaboración
3. Construcción
4. Transición

Si bien la metodología a primera vista no presenta un proceso iterativo e incremental (lo cual debe estar en el núcleo de cualquier metodología o proceso *agile/lean*) debido a que las 4 fases por si solas no se ejecutan iterativamente, dentro de la ejecución de cada una existe retroalimentación y un marcado carácter iterativo en donde la fase que destaca en este contexto es la fase de Construcción a la que se le dará mayor énfasis descriptivo en este capítulo.

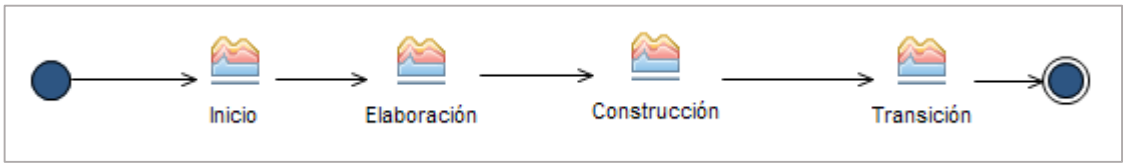

Figura 3.1: Fases de la metodología propuesta (Fuente: elaboración propia).

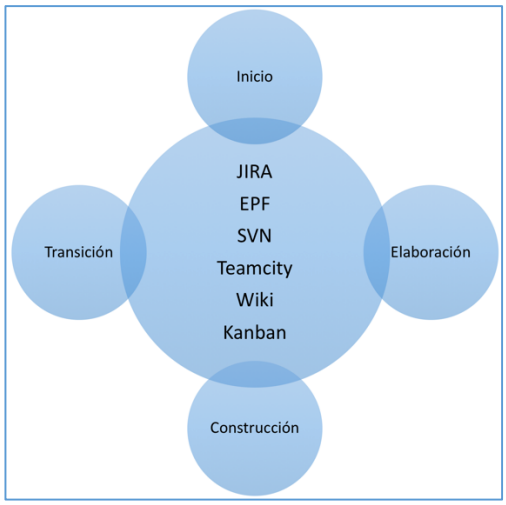

Figura 3.2: Fases y herramientas utilizadas (Fuente: elaboración propia)

1. Inicio

La fase de inicio (ver Figura 3.3) tiene como objetivo presentar el caso técnica y comercialmente al cliente y determinar su cancelación o aceptación. Contiene, principalmente, las actividades y tareas que permitirán configurar y desarrollar los artefactos iniciales del proyecto (creación de repositorios, documentos de análisis y diseño preliminares, y documentos de planificación).

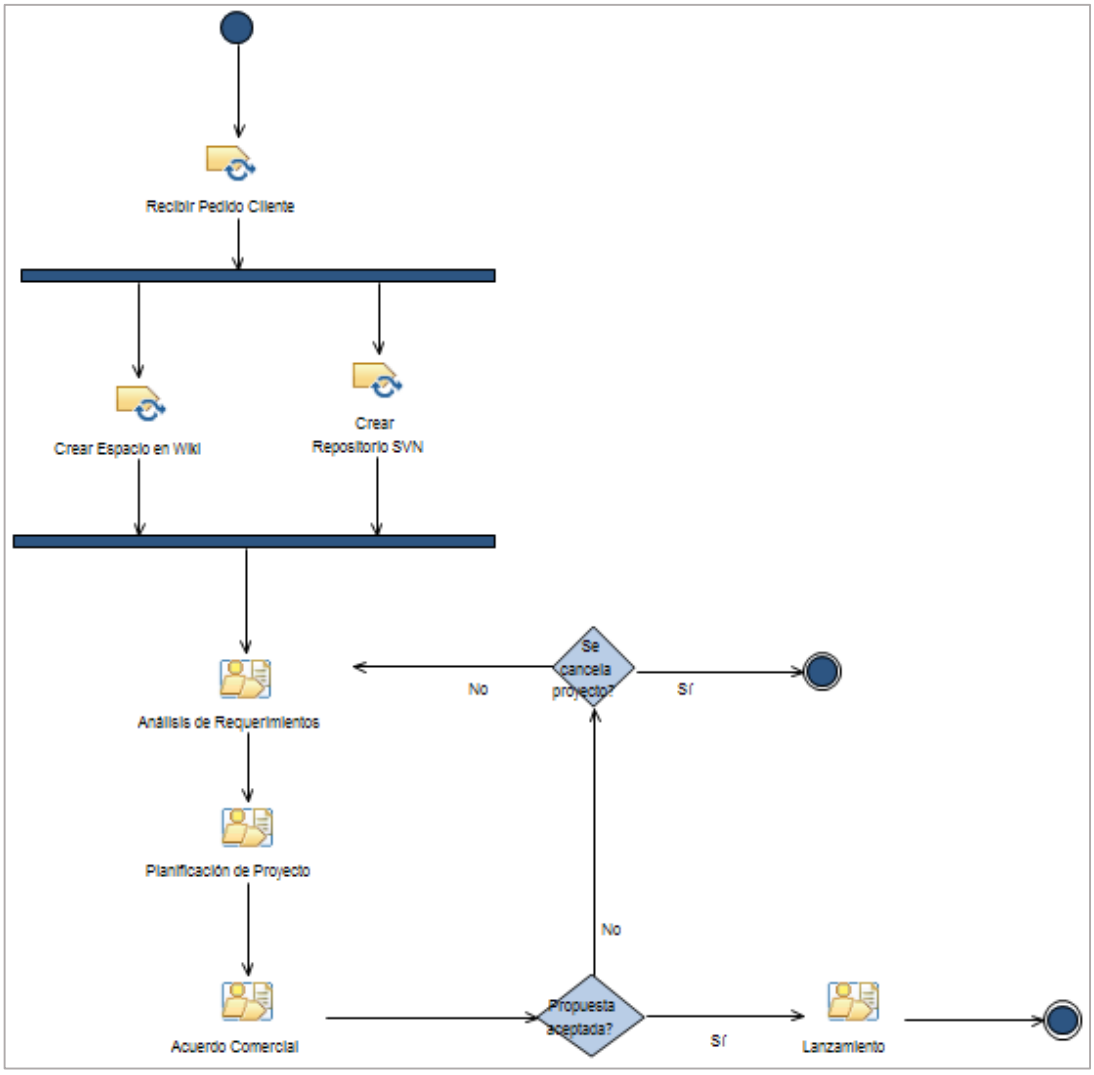

Figura 3.3: Fase de Inicio (Fuente: elaboración propia).

2. Elaboración

En esta fase (ver Figura 3.4) el objetivo es principalmente realizar la validación de la arquitectura de la solución en caso que sea necesario hacerlo debido a la inexperiencia del equipo de desarrollo en alguna tecnología imprescindible para desarrollar el proyecto. Se considera que el prototipo de arquitectura a validar debe contener sólo los aspectos necesarios para eliminar la incertidumbre que existe con respecto a ella, por lo tanto se trata del desarrollo de un prototipo desechable. Dependiendo del resultado del desarrollo del prototipo se pude seguir adelante con el proyecto o incluso tomar la decisión de cancelar el mismo debido a las incertidumbres que existen y el riesgo que ello ocasiona. Además si el equipo de desarrollo tiene baja incertidumbre en las tecnologías debido por ejemplo a su experiencia previa, es factible no ejecutar la fase y pasar de inmediato a la fase de **Construcción**.

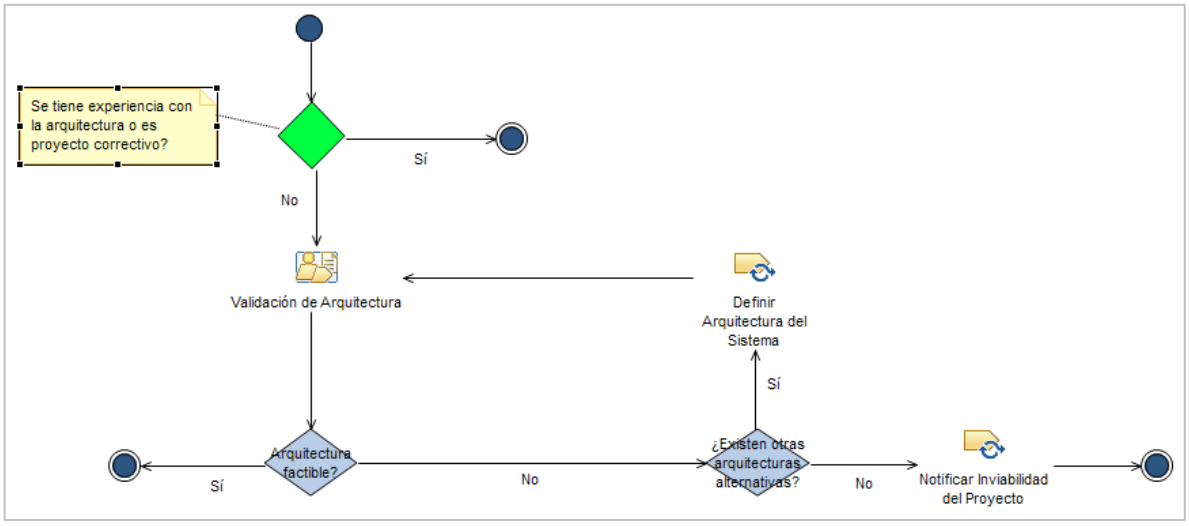

Figura 3.4: Fase de Elaboración (Fuente: elaboración propia).

3. Construcción

En esta fase (ver Figura 3.5) el objetivo es la construcción directa de las piezas de código que conformarán el sistema o producto que tiene como fin el proyecto en cuestión. Principalmente se realizan actividades y tareas clásicas de bajo nivel, es decir, programación y *testing*, aunque también existen tareas de planificación, diseño y documentación.

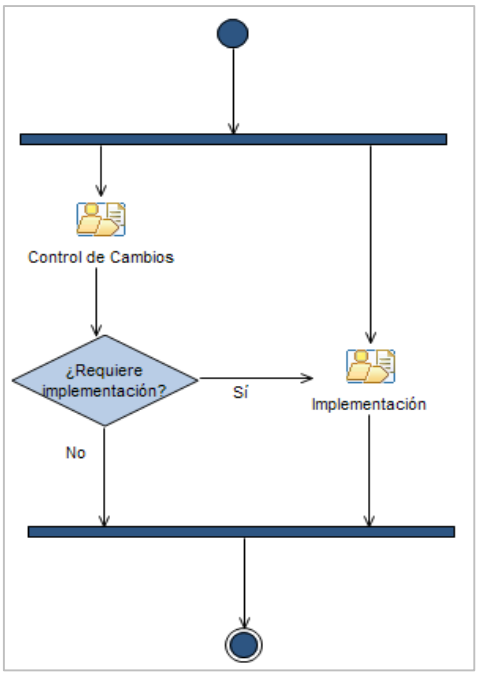

Figura 3.5: Actividades de Fase de Construcción (Fuente: elaboración propia).

Esta fase está compuesta por 2 actividades **Control de Cambios** e **Implementación**.

3.1 Implementación

Esta actividad (ver Figura 3.6) pretende construir las piezas de software que se requieren para satisfacer los requerimientos del sistema o producto a ser desarrollado en el proyecto. Comienza con una **Planificación de Entregas Internas**, es decir, dentro del equipo de desarrollo, de las historias de usuario a construir y la secuencia en se abordará la construcción de cada una. A partir de la planificación de entregas internas se puede iniciar de forma coordinada y en paralelo, las actividades de **Diseño de Pruebas** y **Desarrollo de Solución**.

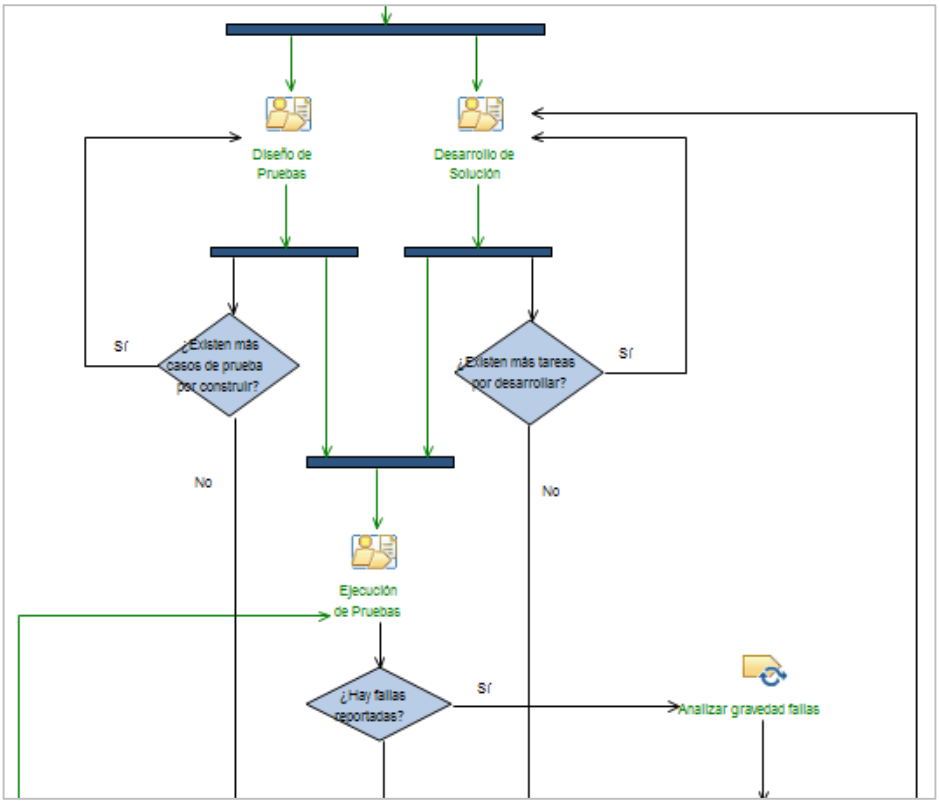

Figura 3.6: Extracto de diagrama de Implementación mostrando actividades de Diseño de pruebas, Desarrollo de Solución y Ejecución de Pruebas (Fuente: elaboración propia)

3.1.1 Diseño de Pruebas

El diseño de pruebas (ver Figura 3.7) se documenta en el Sistema de Gestión de Proyectos (JIRA, ver sección **Herramientas de Soporte a la Metodología**), abordando cada historia de usuario en el orden en que fue determinado en la planificación interna. El sistema de gestión permite, en este contexto, ser utilizado como sistema de administración de casos de pruebas que permite documentar el diseño y los resultados de la ejecución en cada ciclo.

Dependiendo de la práctica actual se puede diseñar las pruebas de forma tradicional, es decir, emitiendo un enunciado, pasos de ejecución y detallando los datos de prueba a utilizar, o programar las pruebas si el proyecto se ejecuta en un ambiente dotado con ejecución automática de pruebas o *Test Driven Development*[9](*TDD*).

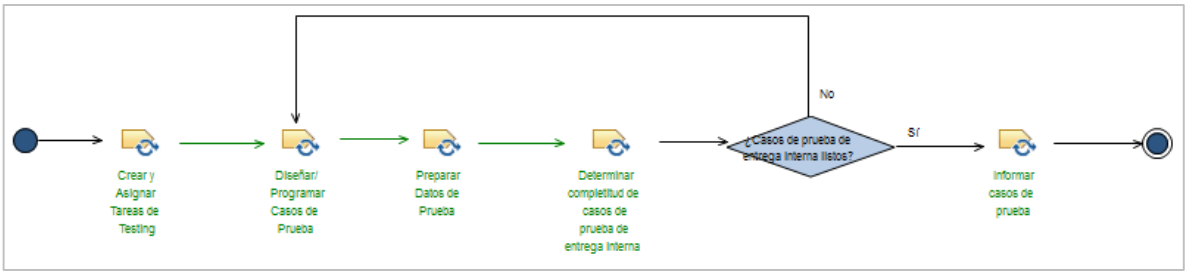

Figura 3.7: Diagrama de actividad de Diseño de Pruebas (Fuente: elaboración propia)

3.1.2 Desarrollo de Solución

En esta actividad (ver Figura 3.8) se asignan las tareas (**Asignar Tareas Estimadas**) definidas anteriormente en la **Fase de Inicio** y se prepara el ambiente de desarrollo para iniciar la construcción a bajo nivel del sistema. La asignación de tareas se puede realizar de forma tradicional, es decir, asignando a una persona en particular o permitiendo "asignar al equipo" en cuyo caso se deja a iniciativa del equipo decidir quién abordará el desarrollo de una historia de usuario en cada momento.

Una vez asignadas las tareas se procede a **Armar Ambiente Desarrollo** que permitirá a los programadores construir y probar localmente las historias abordadas. Es una tarea que generalmente se efectúa una sola vez, ya que luego del *setup* inicial del ambiente no será necesario realizar el *setup* nuevamente aunque podrían existir actualizaciones marginales al mismo.

Luego de ajustar el ambiente se actualiza el diseño preliminar (**Elaborar Diseño Detallado**) y las historias de usuario (**Refinar Requerimientos**) elaboradas en la **Fase de Inicio** para agregar mayor detalle que permita abordar de mejor manera el desarrollo de cada historia de usuario en la tarea **Implementar Solución**. Notar que la tarea **Refinar Requerimientos** se ejecuta de forma paralela a las anteriores dos, esto ocurre para permitir aumentar el entendimiento de los requerimientos aun cuando ya se haya comenzado con la implementación dotando de flexibilidad y evolución a la metodología. Una vez implementada el conjunto de historias que se determinó deben ser abordadas en la actual

9 Desarrollo Guiado por Pruebas, es una práctica de desarrollo en que la construcción del sistema comienza desde la elaboración de una prueba unitaria que se debe aprobar, luego se programan los componentes (clases en orientación a objetos) necesarios para que su ejecución permita aprobar la prueba, luego se repite el proceso. En este ciclo se va, de forma iterativa e incremental, desarrollando el sistema ajustado a los requerimientos.

iteración, se elaboran pruebas unitarias en **Crear Pruebas Unitarias** y luego se ejecutan localmente en el ambiente de desarrollo en **Ejecutar Pruebas Unitarias**. Habiendo pasado exitosamente las pruebas unitarias se incorporan las piezas de software al repositorio del proyecto en **Subir Implementación a Repositorio** desde donde se gatilla el *build* del sistema o producto en el servidor de integración continua en **Integrar Pieza de Software** donde se vuelven a ejecutar las pruebas unitarias elaboradas y se incorporan definitivamente al proceso de integración. Obtenida una integración exitosa se debe determinar si se ha completado una entrega interna (**Determinar completitud de entrega interna**) en cuyo caso se informa al equipo (**Informar Entrega Interna**) que junto a la notificación del fin del diseño de casos de prueba permite iniciar la ejecución de éstos últimos en la actividad **Ejecución de Pruebas**.

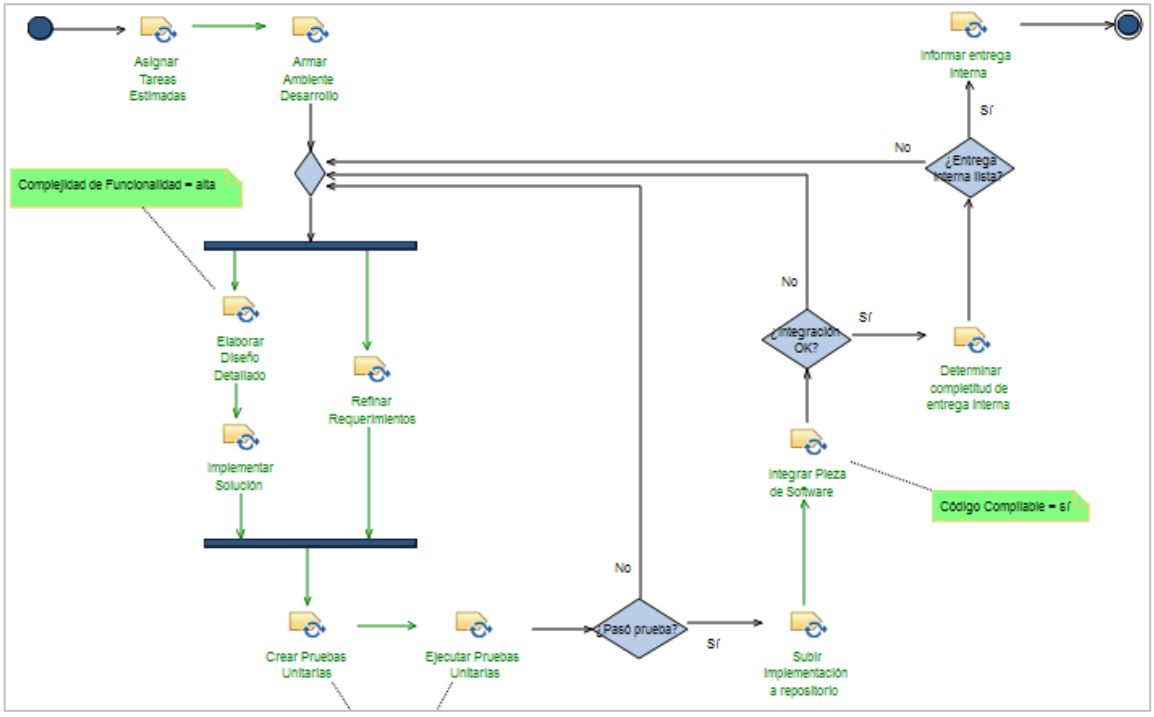

Figura 3.8: Diagrama de Desarrollo de Solución (Fuente: elaboración propia)

3.1.3 Ejecución de Pruebas

Las actividades de ejecución de pruebas (ver Figura 3.9) se inician en el armado de un ambiente de ejecución (**Armar ambiente de pruebas**) si se está en presencia de la primera entrega interna, similarmente a la actividad de **Desarrollo de Solución**, el ajuste del ambiente se realizará una sola vez requiriendo actualizaciones posteriores en cada ciclo.

Una vez ajustado el ambiente se ejecutan los casos de prueba, diseñados anteriormente, en **Ejecutar casos de prueba** para las historias de usuario de la entrega interna. La ejecución de cada caso de prueba dentro del ciclo es documentada en el sistema de administración

de casos de prueba, haya finalizado en la detección de fallas o no. Al finalizar la ejecución y si existen fallas detectadas se reportan al equipo para que inicie su análisis en **Analizar gravedad fallas** y si es necesaria su corrección, en vista de su gravedad, prioridad y estado del proyecto, se vuelven a ejecutar las tareas de la actividad **Desarrollo de Solución** en el contexto de una corrección de falla, es decir, en donde la asignación de tareas será reemplazada por la asignación de la corrección de la falla y/o que se requerirá solo de una actualización del ambiente de desarrollo y/o que no será necesario elaborar un diseño o refinar requerimientos.

La ejecución de ciclos de prueba para entregas internas se suceden hasta que un último ciclo asociado a la última entrega interna, es decir, que finaliza con las historias de la iteración termina sin fallas, entonces se informa del resultado de esta última ejecución al resto del equipo en **Informar resultado último ciclo iteración**. Puede ocurrir que el ciclo anterior finalice con fallas, pero de menor importancia, en tal caso no se informará de resultado exitoso de este último ciclo y se tomará la decisión de liberar la versión actual.

Llegado el momento en que se tiene una versión estable se debe analizar la situación de ésta. Si fue generada como un desarrollo de mantenimiento (sistema o producto ya en estado de mantenimiento), por ejemplo para nuevas funcionalidades pequeñas, esta versión se encontrará en un *branch* dentro del sistema de control de versiones y por lo tanto deberá ser incorporada al *trunk*, al hacerlo se deberán ejecutar nuevamente las tareas en la actividad **Ejecución de Pruebas** para verificar que la mezcla de las dos versiones, *branch* y *trunk*, no quiebre la estabilidad del sistema. Si no existe tal situación entonces se ejecuta la actividad **Liberación**.

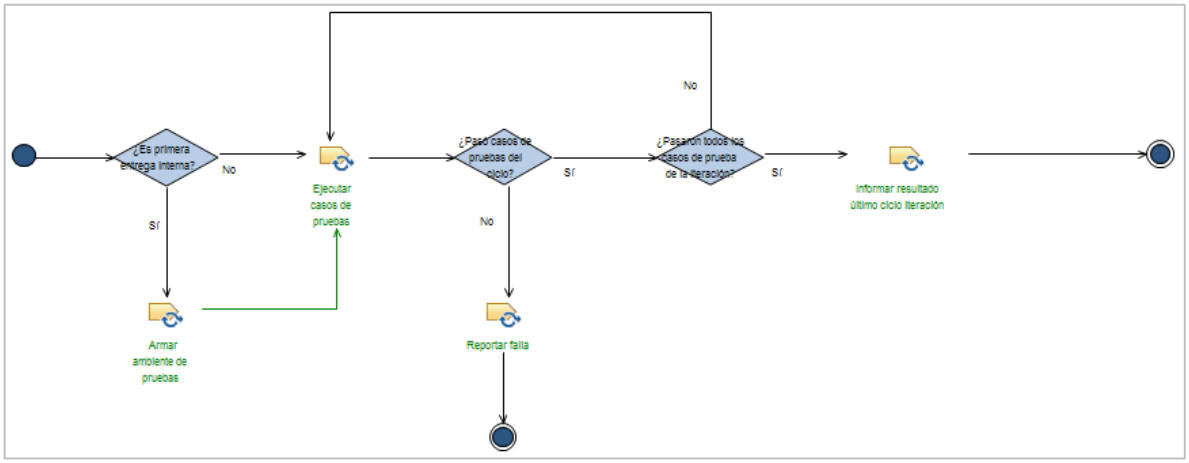

Figura 3.9: Diagrama de Ejecución de Pruebas (Fuente: elaboración propia)

3.1.4 Liberación

Al llegar a la etapa de liberar la versión a producción (ver Figura 3.10) se debe determinar la información necesaria para permitir hacer un seguimiento de la versión (**Generar documentación de Versión del Sistema**). Principalmente lo que se determina es el número de versión, pasos para instalarla y los requerimientos implementados en ella. Luego se debe generar el instalable de la versión que se obtiene del servidor de integración continua y que debe ser igual al *build* que se informó en el último ciclo de la iteración. En este mismo servidor se debe indicar que el *build* en cuestión representa la versión que se liberará (**Generar tag de versión**) y queda destacado del resto de los *builds* generados durante la implementación. A continuación se ejecuta la liberación (actualización) del sistema o producto en producción notificando del fin la tarea (**Notificar liberación**). Finalmente se registra la documentación generada en la wiki del proyecto (**Registrar versión y documentación en Wiki**) y si la liberación contempla la instalación de base de datos se registra en el sistema de control de versiones un backup de esa versión (**Generar y registrar backup BD versión**).

Otra situación, similar a la que ocurre al final de la actividad **Ejecución de Pruebas,** es que se trate de una liberación intermedia entre una iteración y otra debido a la corrección de fallas detectada en la iteración primera, ya en producción, en cuyo caso una vez liberada la versión se debe mezclar ésta con el *trunk* en el sistema de control de versiones en donde se encuentra implementándose la siguiente iteración.

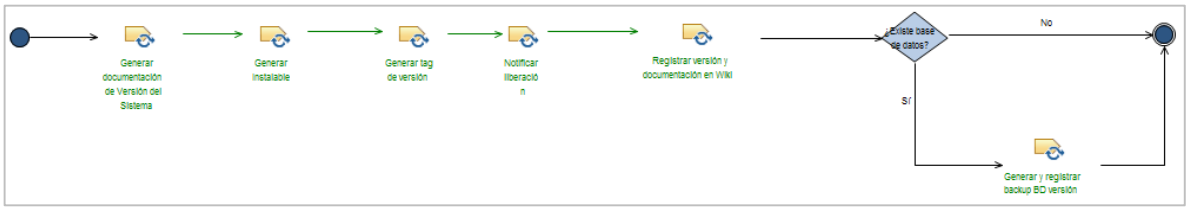

Figura 3.10: Diagrama de Liberación (Fuente: elaboración propia)

Para dotar de flexibilidad y agilidad a la metodología y abrazar el cambio que está siempre presente, existe la actividad **Control de Cambios** que permite abordar de forma controlada los cambios de requerimientos que existen debido a la caducidad de los mismos.

3.2 Control de Cambios

El control de cambios (ver Figura 3.11) se gatilla desde la recepción de solicitud del cliente por un cambio en los requerimientos (**Recibir Solicitud de Cambio**) estando la implementación ya iniciada en cualquier iteración. La solicitud es analizada en las tareas de las actividades (anteriormente discutidas) **Análisis de Requerimientos** y **Planificación de Proyecto**. Con la información entregada por las actividades anteriores se puede analizar el impacto técnico que tiene en la ejecución del proyecto la solicitud actual (**Analizar Impacto**

Solicitud de Cambio). Tras ejecutar la tarea anterior se debe analizar el estado de la construcción del sistema y pueden ocurrir los siguientes escenarios:

- Se determina que el impacto es bajo:
    - Si la solicitud es urgente entonces se puede abordar el cambio dentro de la iteración actual solo si la próxima liberación de ella es lejana en tiempo, en cuyo caso se procede a incorporar el (los) cambio(s) al plan de iteraciones elaborado en la **Fase de Inicio** (**Agregar Iteración a Plan de Iteraciones**)
    - Si la solicitud no es urgente o la próxima liberación es cercana en tiempo entonces se abordará en iteraciones adicionales al final de lo actualmente planificado
- Se determina que el impacto es alto:
    - Se requiere de un acuerdo comercial (**Acuerdo Comercial**) para autorizar la incorporación del cambio en el proyecto, si el acuerdo es positivo entonces se requerirá implementación y se incorporará en iteraciones adicionales al final de lo actualmente planificado

Decidir si la próxima liberación es lejana o cerca en tiempo tal que pueda ser abordada una solicitud de cambio o no, queda a discreción del equipo de desarrollo. Elementos de decisión pueden ser la posibilidad de postergar alguna historia de usuario a favor de otra que se vuelve más compleja o de lograr algún acuerdo con el cliente en cuanto a la fecha de liberación de una iteración.

Si el cambio es aceptado entonces es efectivamente abordado en la actividad **Implementación**.

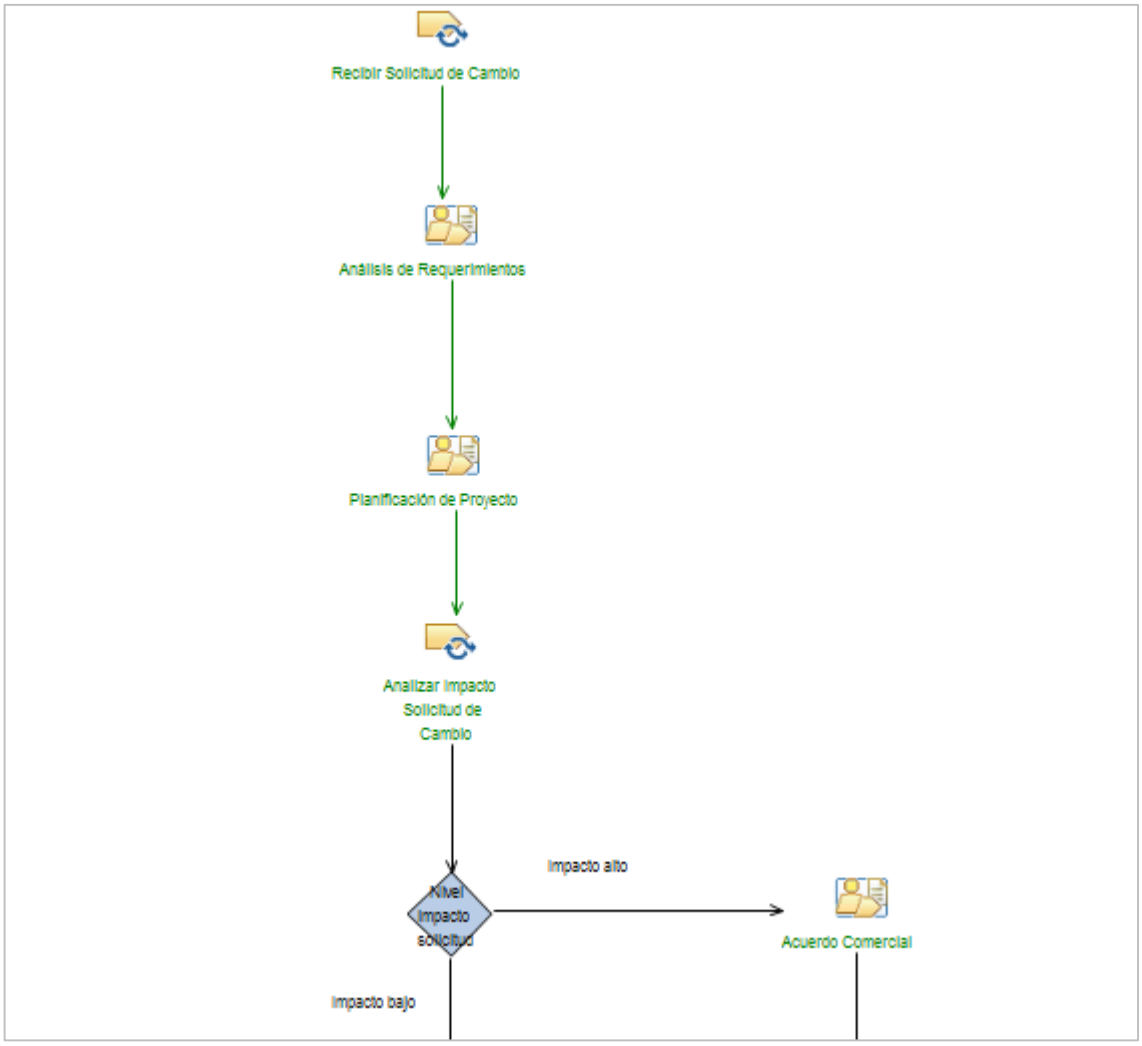

Figura 3.11: Extracto de Diagrama Control de Cambios (Fuente: elaboración propia)

4 Transición

En esta fase (ver Figura 3.12) el objetivo es lograr una completa aceptación, por parte del cliente, del sistema desarrollado. Al tratar de cumplir con lo anterior es posible que se deba pulir el sistema lo cual que puede significar: corregir una falla reportada en producción, ya sea por el cliente o por el equipo de desarrollo, o implementar una mejora propuesta por los mismos *stakeholders*.

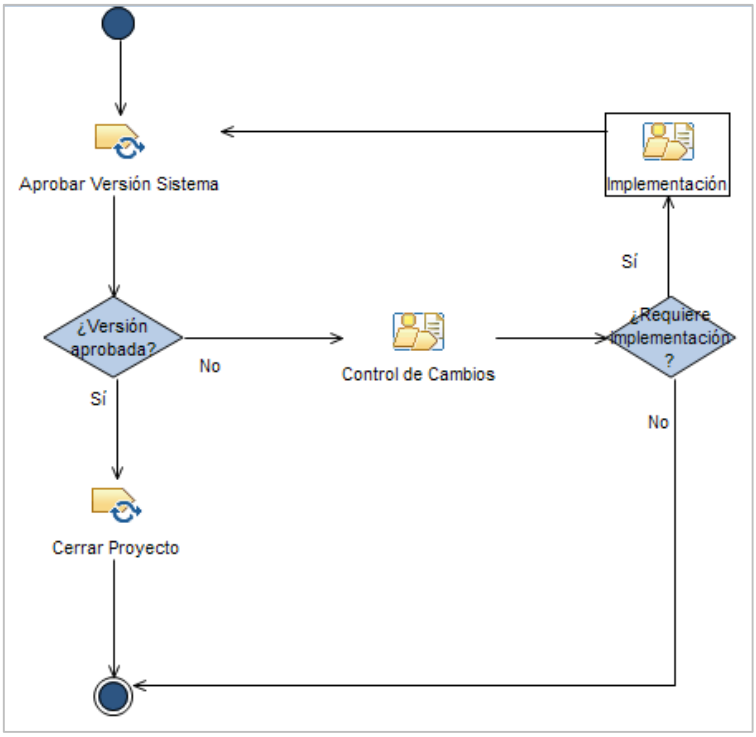

Figura 3.12: Actividades de Fase de Transición (Fuente: elaboración propia)

Luego de lograr el objetivo, el sistema pasa a una categoría de “En Mantenimiento” en donde se realizará el soporte del sistema a largo plazo hasta el fin de su vida útil. El mantenimiento es aconsejable se realice usando una metodología independiente ya que en esta etapa se enfrentan otros tipos de desarrollos y problemas.

5 Herramientas de Soporte a la Metodología

La metodología presentada puede ser mejor ejecutada y permite una comodidad para el equipo de desarrollo si se complementa con una serie de herramientas para cada aspecto presente dentro de la ejecución. La práctica de la metodología ha llevado a identificar las siguientes herramientas (Ver Tabla 3.1), a las que se ha hecho referencia en la descripción de las actividades anteriores, con las que se han obtenido buenos resultados:

Herramienta	Aspecto de la metodología (Uso dado)
Wiki	Gestión de conocimiento y documentación del proyecto [MediaWiki]
Teamcity	Servidor de integración continua (contiene y permite gestión de *builds* del sistema) [Teamcity]
SVN	Sistema de control de versiones (*SCM*[10]) [SVN]
JIRA	Sistema de gestión de proyectos Sistema de administración de casos de prueba Kanban digital [JIRA]
Kanban (tablero físico/digital)	Visualización de flujo de trabajo (historias de usuario)
Eclipse Process Framework	Editor de procesos en meta-modelo *SPEM* Validación de consistencia

Tabla 3.1: Lista de herramientas utilizadas en la práctica

Como se indicó anteriormente, la lista presentada ha funcionado empíricamente y como toda herramienta el objetivo de su uso es facilitar la ejecución de tareas por parte del equipo de desarrollo, si otro equipo se embarca en la implementación de esta metodología, se incentiva a que busque las herramientas que mejor permiten la ejecución de las tareas según la experiencia del mismo, tratando de abarcar los aspectos de la metodología descritos.

10 *SCM*: ***S**ystem **C**onfiguration **M**anagement*

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

La génesis de la metodología presentada ocurrió como resultado de una cooperación entre la Universidad de Chile y la empresa como parte del proyecto ADAPTE, cuyo objetivo era investigar el uso de la adaptación de procesos en el desarrollo de software en las empresas de software chilenas. Dado que era necesario que las empresas participantes tuviesen ya definido y documentado un proceso o metodología de desarrollo, la universidad ofreció sus servicios para ayudar a las empresas a cumplir el requisito. La empresa aceptó la sugerencia y se definió y documentó una primera versión de la metodología presentada en donde el autor de esta memoria participó activamente de las actividades de documentación de la metodología. La metodología ya era usada anteriormente, es decir, estaba definida conceptualmente pero carecía de la documentación necesaria para formalizar su existencia. Dado lo anterior, el uso de la metodología era común en los proyectos de la empresa. La oportunidad de definir y documentar permitió explicitar el cómo se desarrollaba el software, detectar inconsistencias y oportunidades de mejora.

Elementos Destacables

La metodología se usó en una docena de proyectos desde el 2012 al 2015 (ver Tabla 4.1, 4.2 y 4.3 con número, categoría y tipos de proyectos) y si bien fueron proyectos en su mayoría exitosos (considerando aspectos comerciales y técnicos) elementos importantes que destacaron en las ejecuciones de la metodología son:

- El ajuste inicial del servidor de integración continua tomaba un tiempo no despreciable y aunque se realizaba solo una vez por proyecto técnicas como *Pipeline as code* pueden hacerlo más liviano y efectivo
- El uso de la wiki es muy útil, sin embargo, si no se tiene el cuidado de incorporar la categoría a cada artículo[11] creado, éstos se pueden volver rápidamente difíciles de encontrar
- El uso de *TDD* se volvió casi imposible debido a la falta de conocimiento y tiempo para aprender la práctica
- En relación al punto anterior, sí se logró adoptar la implementación de pruebas automatizadas que se ejecutan justo después de incorporar código fuente en el *SCM*
- Se hizo necesario incorporar detalles acerca de cómo ejecutar las tareas más importantes, por lo tanto, se incorporaron guías, recomendaciones y formatos de artefactos (disponibles en metodología en formato *SPEM* en la versión digital de esta memoria)
- El registro de tiempo trabajado por persona no fue útil ya que la métrica que importaba era el tiempo medio de entrega de funcionalidades y valor al cliente

11 Un artículo en la wiki es cualquier documento en ella, por ejemplo una historia de usuario o la Visión del Sistema

- No se logró incorporar un conjunto importante de métricas y *dashboards* en donde realizar análisis a modo de *software development analytics*, principalmente a causa del poco tiempo con que contaba el equipo

Tabla 4.1: Categoría de Proyectos según Duración

Duración		
<2 mes	2 a 6 meses	>6 meses
Pequeño	Mediano	Grande

Tabla 4.2: Cantidad de Proyectos por Tipo durante años 2012-2014

	Tipo	
	Mantenimiento	Nuevos Desarrollos
# de proyectos	7	5

Tabla 4.3: Cantidad de Proyectos por Tipo durante años 2012-2014

	Categoría		
	Pequeño	Mediano	Grande
# de proyectos	7	2	3

Factor Humano y liderazgo

La **metodología** al ser una herramienta en sí misma para permitir al equipo desarrollar software se convierte en un elemento que puede mutar de acuerdo a las necesidades de las personas que la utilizan y también de acuerdo a los niveles de motivación y compromiso de las mismas.

En la empresa y durante cierto periodo se produjo un cambio de liderazgo con respecto al que existía cuando la metodología fue definida, documentada y mejorada, éste liderazgo de marcada negatividad fue socavando el compromiso y motivación del equipo de desarrollo a través de las constantes solicitudes “urgentes”, cambios de contexto frecuentes, reuniones ineficientes e instauración de situaciones conflictivas de forma premeditada. La consecuencia lógica terminó siendo el abandono del uso de la metodología, la adopción del caos, los procesos no repetibles y el quiebre del equipo de desarrollo con varias deserciones.

CONCLUSIONES

Tener una base desde donde comenzar cuando se hacen esfuerzos para hacer del desarrollo de software una actividad confiable, predecible y repetible es una opción muy deseable en la industria, en particular en las PYMES en donde los recursos son relativamente más escasos que en otra categoria de empresas. Casos de éxito, recomendaciones, descripción de metodologías y buenas prácticas es información de gran valor en estos casos, sin embargo es muy importante indicar que cada proyecto es una instancia particular y muy distinta cada vez y que se debe considerar cualquier modificación o *tailoring* a la forma de desarrollo que sea necesaria para lograr los objetivos del proyecto tanto del proceso como de las herramientas que lo apoyan. La metodologia es una herramienta y sirve a los objetivos del equipo de desarrollo, no al revés.

El liderazgo que incentiva, gatilla y sostiene los cambios y la búsqueda de la mejora continua es imprescindible. Las transformaciones ágiles o adopciones de nuevas formas de “hacer” no son posibles sin el adecuado liderazgo de los *sponsors* que empoderan a los equipos y les permiten la autonomía necesaria para lograr grandes resultados.

Existen dos aspectos importantes en el enfoque de la agilidad en el desarrollo de software: “hacer ágil” y “ser ágil”. En este trabajo se ha descrito una metodología que se acerca al “hacer ágil”, es decir, aplicar en algunas fase del desarrollo de software prácticas ágiles que la experiencia indica que han entregado buenos resultados, pero se debe tener en cuenta que la real transformación ocurre cuando se “es ágil” lo que requiere un cambio de *mindset* total en la organización. Las organizaciones que logran llegar al “ser ágiles” suelen no necesitar una metodología definida para toda la organizaciñon ya que sus equipos empoderados pueden definir lo que mejor aplique al proyecto en curso y cambiar de forma rápida y flexible si fuese necesario.

REFERENCIAS BIBLIOGRÁFICAS

- [Balduino07] Balduino R. (2007) *Introduction to OpenUP (Open Unified Process)*
- [Bastarrica16] Bastarrica, M. C., Marques, M., Ochoa, S., Perovich, D., Quispe, A., Robbes, R., Simmonds, J. (2016). *La industria del software en Chile: Desafíos y oportunidades para generar un impacto país*, Bits de Ciencia, N° 13, 34-39. Departamento de Ciencias de la Computación de la Facultad de Ciencias
- [Brooks95] Brooks F. (1995 – 20ava. edición aniversario). *The Mythical Man-Month*. Addison-Wesley
- [Cantor01] Cantor, M. (2001). *Software Leadership: A Guide to Successful Software Development*
- [D'Amico12] D'Amico V. *Waterfall*. 5/6/2012. *Development Is Prescriptive; Agile Development Is Adaptive*. Obtenido desde http://brainslink.com/2012/06/waterfall-development-is-prescriptive-agile-development-is-adaptive/. Último acceso 16/03/2017
- [EstatutoPYMES] *Estatuto de las PYMES*. Obtenida desde http://www.bcn.cl/leyfacil/recurso/estatuto-de-las-pymes. Último acceso 19/06/2016
- [Fichtner] Fichtner A. *Agile Vs. Lean: Yeah Yeah, What's the Difference?*. Obtenida desde https://hackerchick.com/agile-vs-lean-yeah-yeah-whats-the-difference/. Último acceso 16/03/2017
- [García11] García F., Vizcaíno A., Ebert C. (2011). *Process Management Tools*. IEEE edición Abril 2011, 15-18. Obtenida desde https://vector.com/portal/medien/vector_consulting/publications/Ebert_ProcessTools_IEEESoftware_2011V28N2.pdf. Último acceso 16/03/2017
- [Highsmith2013] Highsmith J. (2013) *Adaptive Leadership, Acelerating Enterprise Agility*. Addison-Wesley Professional. Obtenida desde https://assets.thoughtworks.com/articles/adaptive-leadership-accelerating-enterprise-agility-jim-highsmith-thoughtworks.pdf. Último acceso 15/03/2017
- [Hurtado11] Hurtado A., Bastarrica M., Bergel A. (2011) *Analyzing Software Process Models with AVISPA*
- [JIRA] https://www.atlassian.com/software/jira
- [Kniberg15] Kniberg H. (2015). *SCRUM and XP from the trenches*. C4Media
- [Koutsoumpos14] Koutsoumpos V., Marinelarena I. (2014). *Agile Methodologies and Software Process Improvement Maturity Models, Current State of Practice in Small and Medium Enterprises*. Dept. of Software Engineering (DIPT), Blekinge Institute of Technology
- [Mahanti06] Mahanti A. (2006). Challenges in Enterprise Adoption of Agile Methods – A Survey. Journal of Computing and Information Technology. Edición 14, 197-206
- [MediaWiki] https://www.mediawiki.org/wiki/MediaWiki

- [MicroFocus] *Top 5 Software Development Process Challenges*. (2016). White Paper de Micro Focus. Obtenido desde https://www.microfocus.com/media/white-paper/WP_Top-5-Software-Development-Process-Challenges_Final.pdf. Último acceso 15/03/2017
- [OpenUP] *OpenUP*. http://epf.eclipse.org/wikis/openup/. Último acceso el 27/06/2016
- [Piquer16] Piquer, J. M. (2016) ¿Existe una industria TI en Chile?, Bits de Ciencia, N° 13, 28-30
- [Poppendieck03] Poppendieck, M., Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*, Adison Wesley
- [Riquelme16] Riquelme, M. E. (2016). Radiografía a la industria del software en Chile y sus desafíos, Bits de Ciencia, N° 13, 31-33
- [Schwaber16] Schwaber K., Sutherland J. (2016). *The Scrum Guide*.
- [SoftwareCrisis] *Software Crisis*. 23/03/2015. Obtenida desde https://www.ukessays.com/essays/computer-science/software-crisis.php. Último acceso 16/03/2017
- [SPEM] *Software & Systems Process Engineering Meta-Model Specification*. (2008) Object Management Group. Obtenido desde http://www.omg.org. Último acceso 16/03/2017
- [SVN] https://subversion.apache.org/
- [Teamcity] https://www.jetbrains.com/teamcity/
- [Villena12] Villena, A. (2012), ¿Qué metodología será más adecuada para mi proyecto software? Obtenida desde http://es.slideshare.net/leansight/qu-metodologa-ser-ms-adecuada-para-mi-proyecto-software-13905273. Último acceso 27/06/2016

Anexo

Navegación de metodología en sitio web

En la versión digital de esta memoria se encuentra el directorio `SitioWeb/` en el cual se puede encontrar una versión en formato de sitio web estático de la metodología. Éste sitio web es exportado desde la herramienta *EPF Composer* y es recomendable realizar esta acción para comunicar la metodología a los stakeholders de la organización ya que presentarlo desde la herramienta puede llegar a ser complejo debido a lo técnico del ambiente de edición. Indicaciones acerca de cómo exportar y editar se darán en la siguiente sección en este anexo.

El sitio web muestra 2 zonas como se ve en la Figura 1.

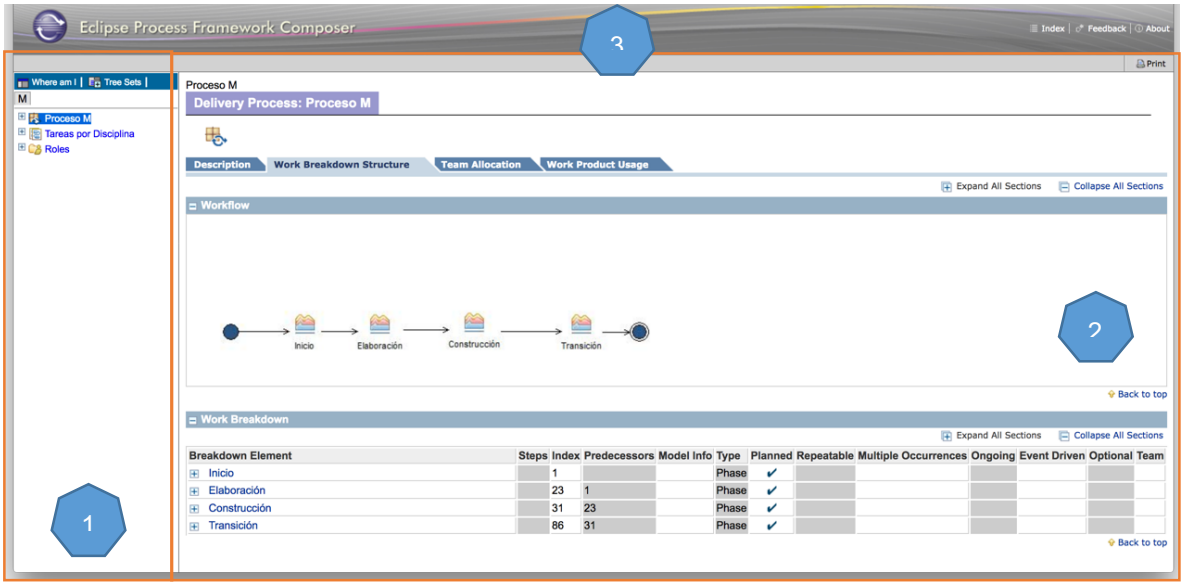

Figura 1 (Fuente: elaboración propia)

La sección 1 es un árbol de navegación que permite llegar rápidamente a los elementos (o ramas) de la metodología: el Proceso, las Tareas por Disciplina y los Roles. Al expandir éstos últimos se pueden ver todas las tareas en las que cada rol está involucrado, entonces sirviendo como un elemento de análisis para la carga de tareas de cada rol.

La sección 2 es la zona principal que despliega todos los detalles de los elementos siendo vistos actualmente: vista del workflow y las actividades/tareas en forma de *breakdown structure* (*Work Breakdown*).

La sección 3 es el banner principal que tiene un título y links como *Index*, *Feedback* y *About*.

Todas las secciones anteriores son configurables desde la herramienta *EPF Composer* en la cual se puede ajustar el despliegue de más o menos información.

En lo que se sigue se obviará la sección 1 y el banner principal para concentrarse en la sección 2 y los elementos que se presentan allí.

Al hacer clic en una de las fases, por ejemplo, **Inicio**, se despliega la siguiente pantalla:

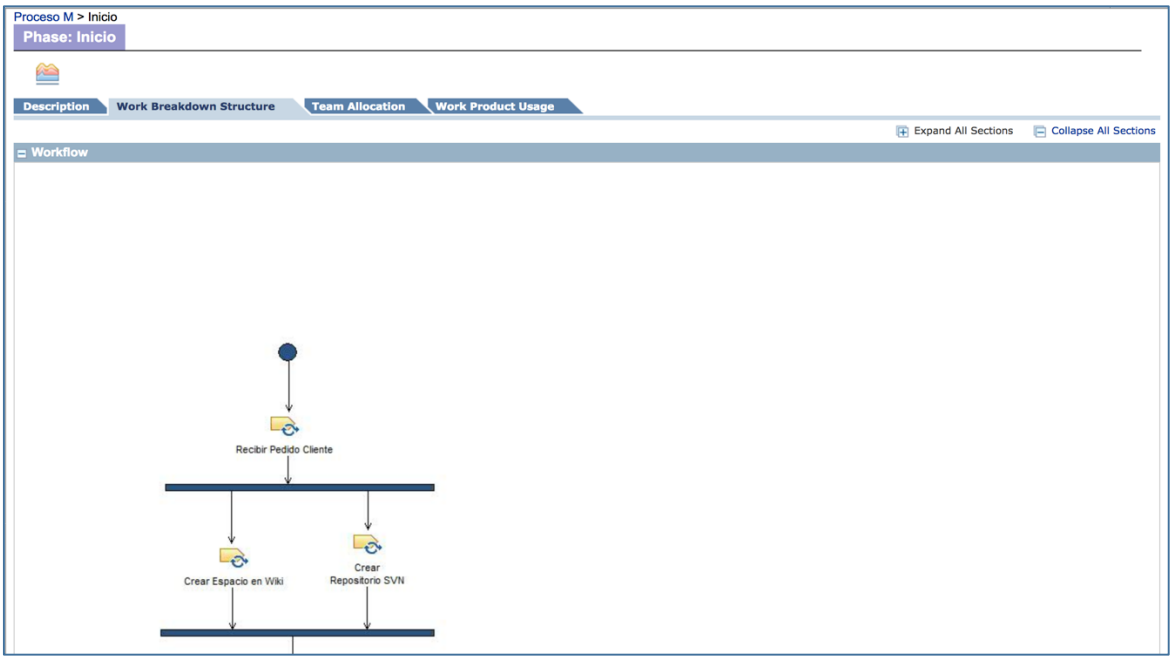

Figura 2 (Fuente: elaboración propia)

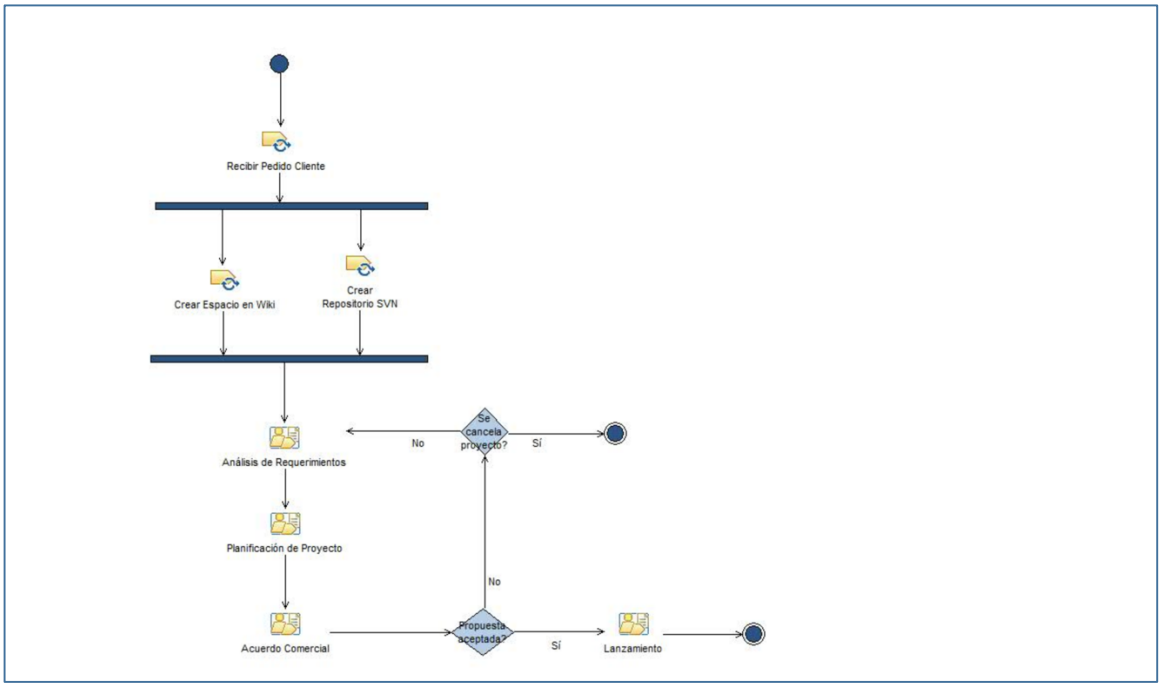

Figura 3 (Fuente: elaboración propia)

Como muestra la Figura 2 y 3 se despliegan las actividades y tareas de la fase. El despliegue se realiza en 3 zonas:

1. Zona de *workflow* (ver Figuras 2 y 3)
2. Zona de detalle de tareas con rol que la ejecuta, *inputs* requeridos y *ouputs* que genera (ver Figura 4)
3. Zona con *breakdown structure* (ver Figura 5)

Figura 4 (Fuente: elaboración propia)

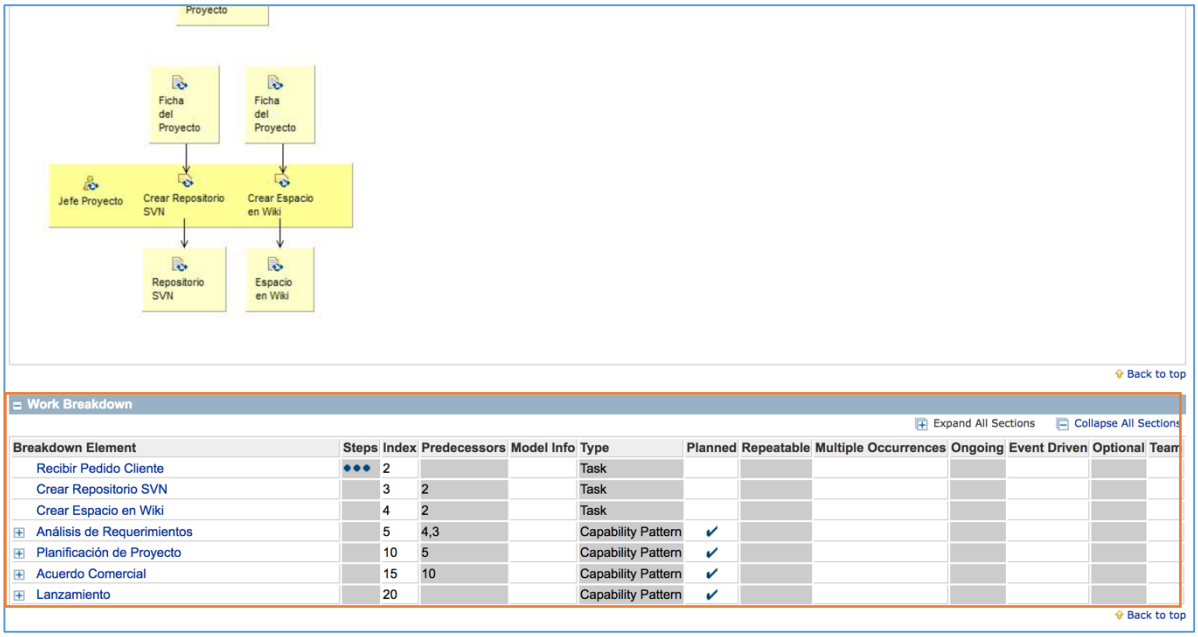

Figura 5: Resalta la zona *breakdown structure* (Fuente: elaboración propia)

Notar que no se detallan las actividades en la zona de tareas, ya que éstas tienen, a su vez, su propia sección de *workflow* y despliegue como el que se describe para una fase. Por ejemplo, si se hace clic sobre la actividad **Analisis de Requerimientos** se despliega la siguiente pantalla (ver Figuras 6 y 7):

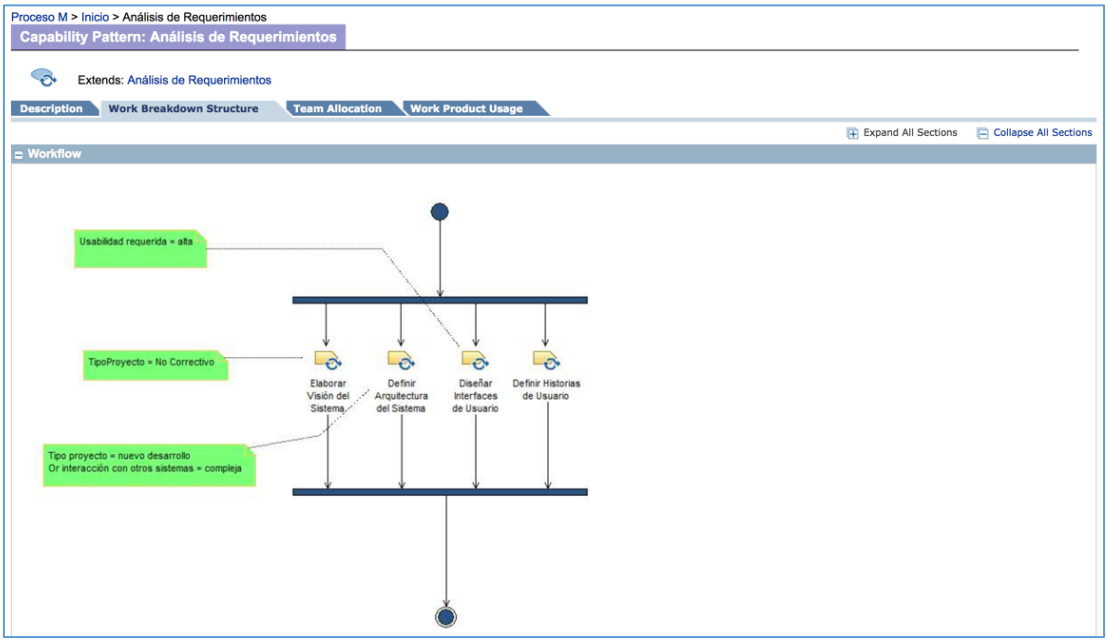

Figura 6 (Fuente: elaboración propia)

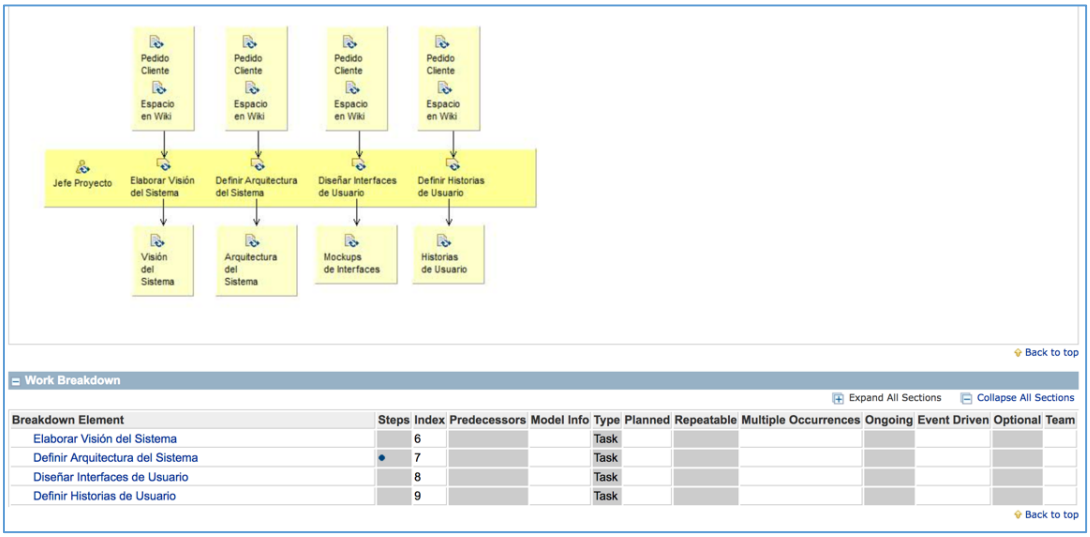

Figura 7: continuación de Figura 6 (Fuente: elaboración propia)

Una actividad en *SPEM* se modela como un *Capability Pattern* que es un elemento que se puede reutilizar dentro de la descripción del proceso, como ocurre con la actividad **Analisis de Requerimientos** que es utilzada tanto dentro de la fase Inicio como dentro de la actividad **Control de Cambios** en la fase **Construcción**.

Al hacer clic en una tarea dentro de la sección de *worflow*, por ejemplo, **Elaborar Vision del Sistema** se puede ver el detalle de la tarea en 4 dimensiones (ver Figura 8):

- Descripción (si existe)
- Roles involucrados en su ejecución
- Inputs requeridos
- Outputs esperados

Además de otros atributos que no fueron usados para el diseño de la metodología que se describe en esta memoria.

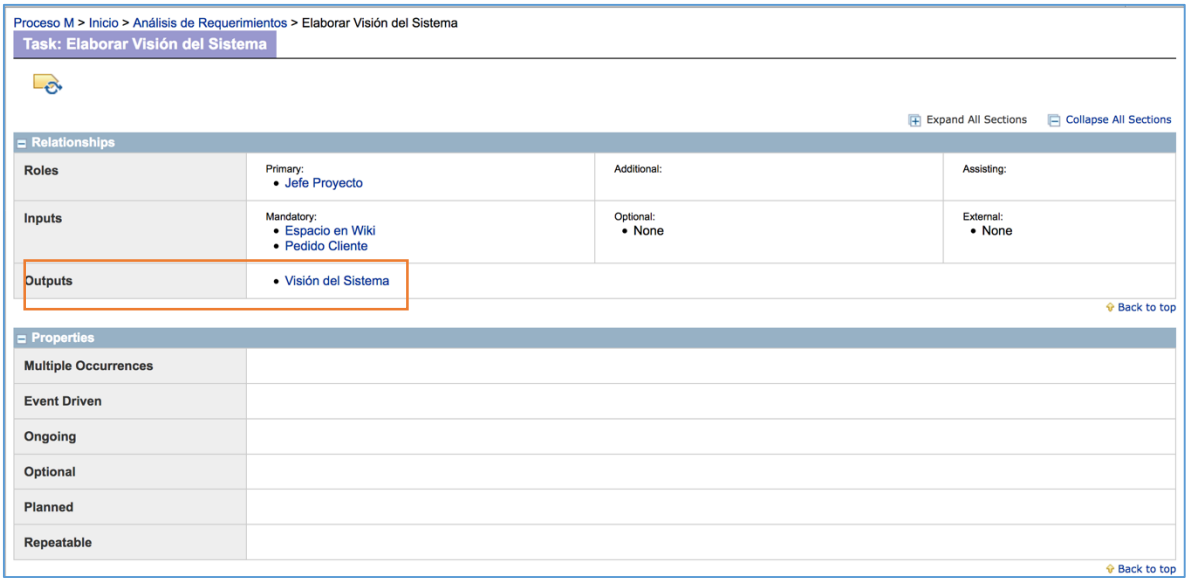

Figura 8: Destacando link hacia el *output* **Visión del Sistema** (Fuente: elaboración propia)

Si se quiere ver el detalle de cualquiera de las 4 dimensiones solo se debe hacer clic sobre uno de ellos, por ejemplo al hacerlo en el *output* **Visión del Sistema** se despliega la siguente pantalla:

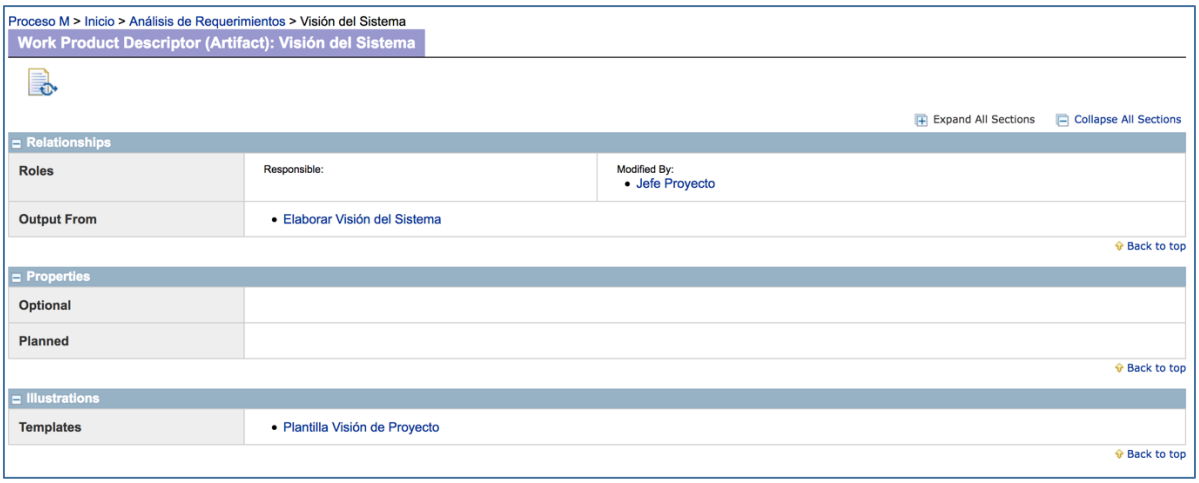

Figura 9 (Fuente: elaboración propia)

Como muestra la Figura 9, se despliegan datos e informaciones asociadas al *output* o *work product* **Vision del Sistema**:

- Descripción (si existe)
- Roles involucrados (*Roles*)
- Tareas de las que es una salida o resultado (*Output From*)
- Ayudas o indicaciones para construir el output. En este caso como ayuda se adjunta un *template* (*Templates*) que es una información común cuando los *outputs* tiene la forma de documento

Otro tipo de ayudas pueden ser guías (*Guidelines*) acerca de como ejecutar una tarea, por ejemplo, la tarea **Implementar Solución** dentro de la actividad **Desarrollo de Solución** a su vez dentro de la actividad **Implementación** de la fase **Construcción** (Ver Figuras 10 a 14 para ver la secuencia de navegación para llegar a la tarea **Implementar Solución**)

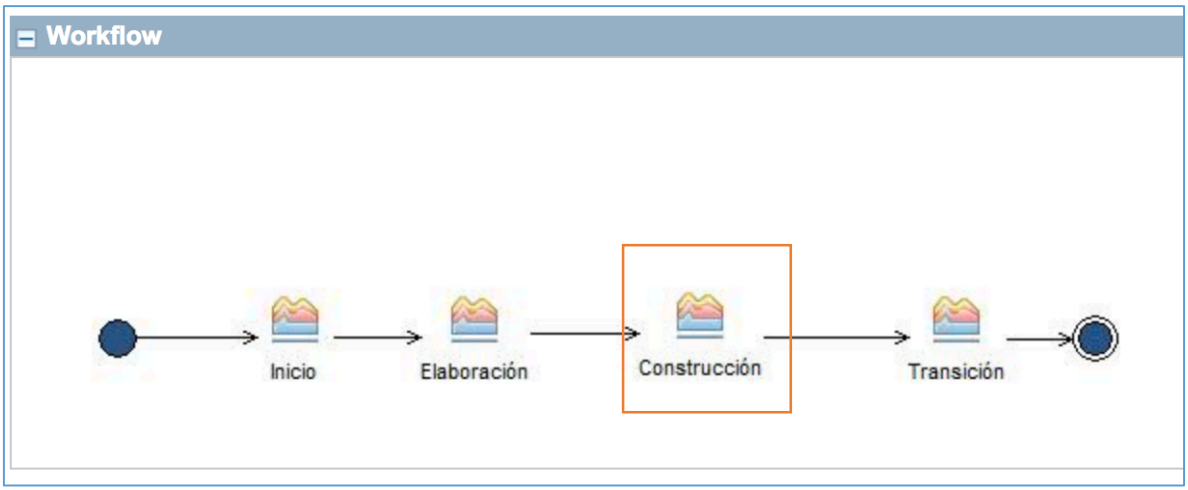

Figura 10 (Fuente: elaboración propia)

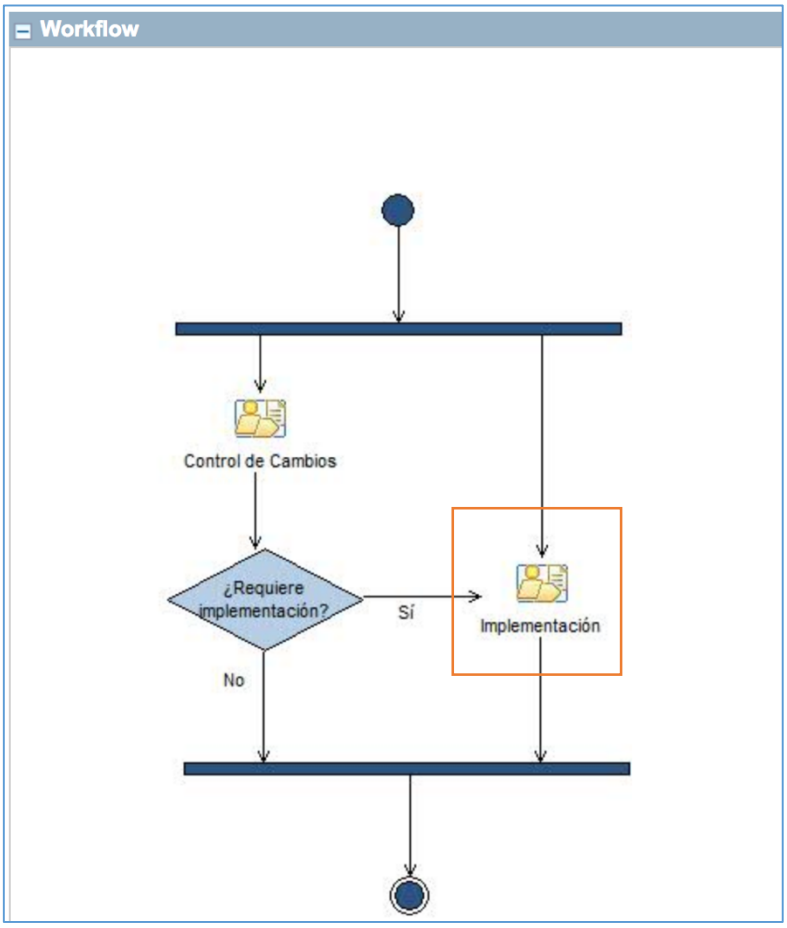

Figura 11 (Fuente: elaboración propia)

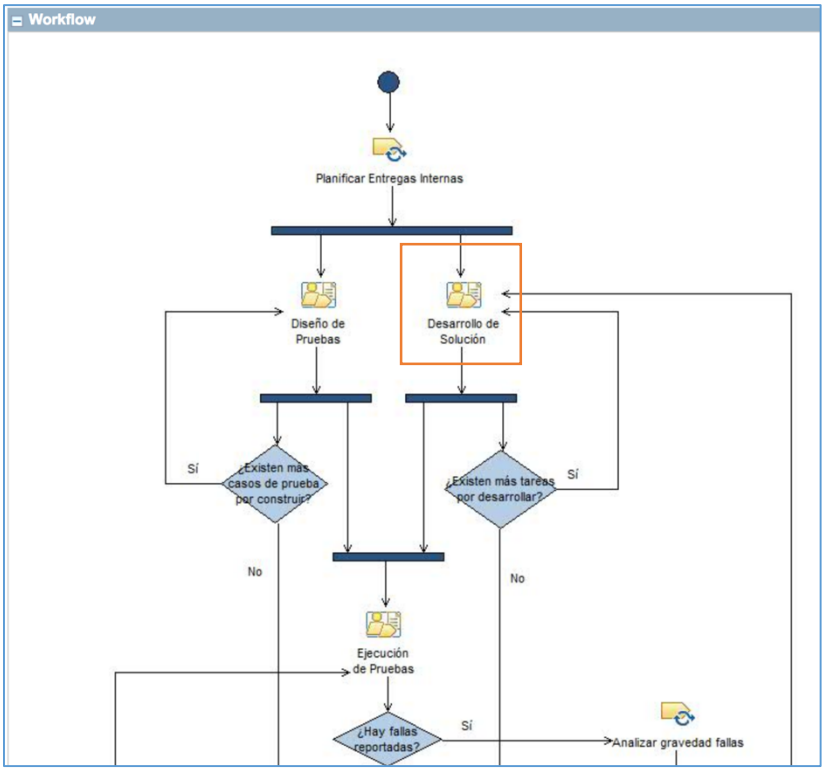

Figura 12: (Fuente: elaboración propia)

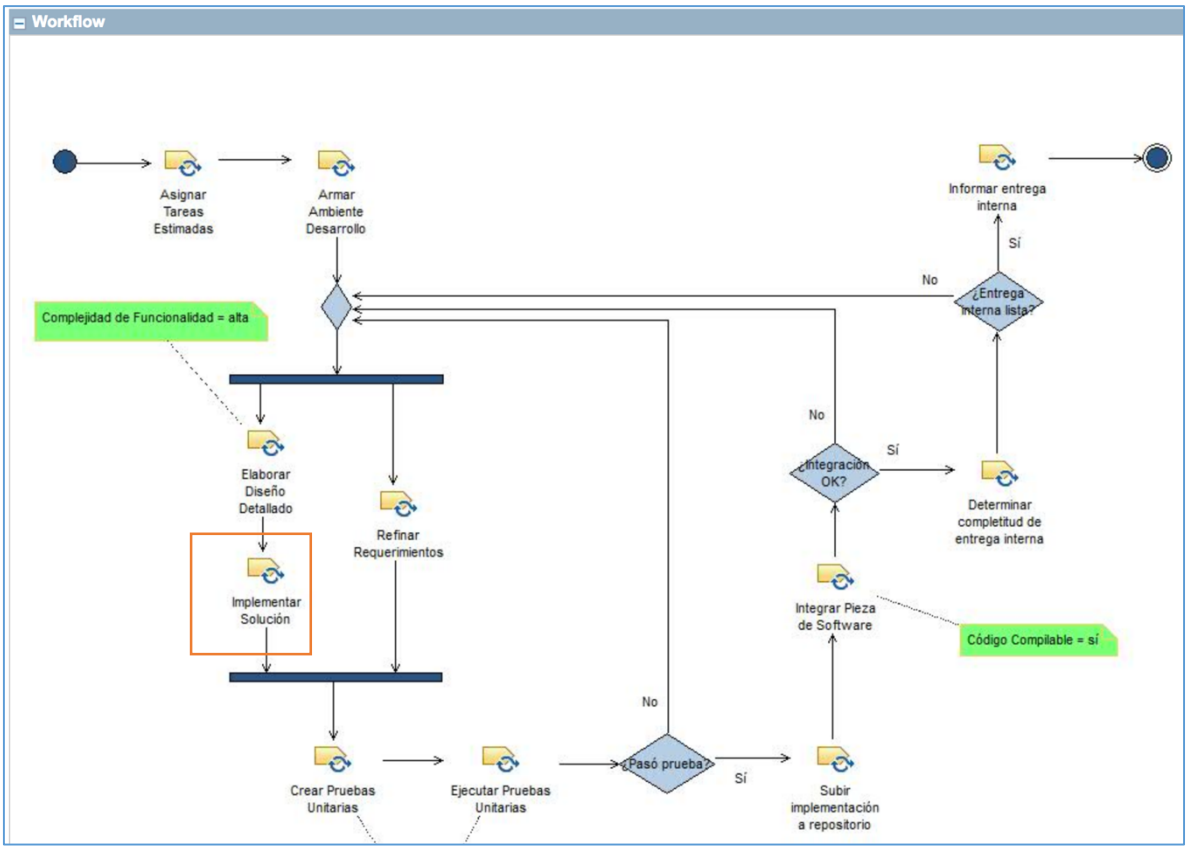

Figura 13: (Fuente: elaboración propia)

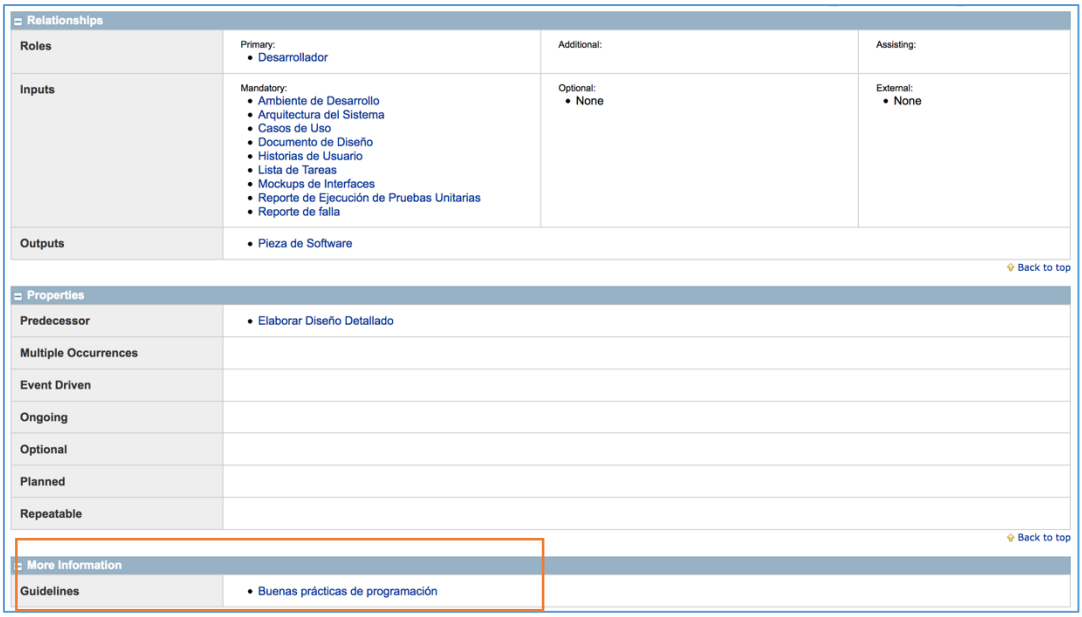

Figura 14: Tarea **Implementar Solución** (Fuente: elaboración propia)

La Figura 14 presenta la información de la tarea **Implementar Solución** y la sección de interés en este caso, *Guidelines*. Al hacer clic en la guia **Buenas prácticas de programación** se presenta una descripción y un link (ver Figura 15) hacia prácticas que de describen, en este caso, en el sitio web de Uncle Bob Martin[12] (Ver Figura 16).

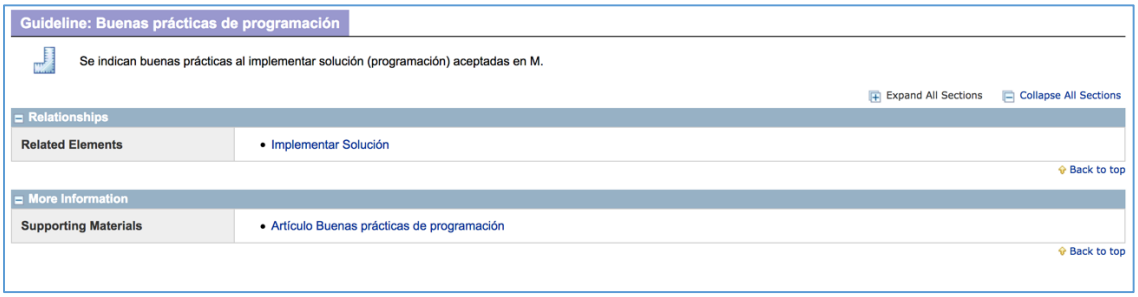

Figura 15 (Fuente: elaboración propia)

Figura 16 (Fuente: elaboración propia)

En resumen el ejemplo anterior muestran la navegación *top-bottom* que se realiza al analizar la metodología. Se comienza por una fase, luego por actividades (pueden ser tareas), luego por tareas las cuales pueden tener guías de ayuda para ser ejecutadas, hasta los *work products* u *outputs* que pueden tener plantillas que ayudan en su construcción (ver Figura 17).

12 Robert C. Martin autor de "Clean Code: A Handbook of Agile Software Craftsmanship", Prentice Hall, sitio web http://blog.cleancoder.com/, ultimo acceso 3/4/2017

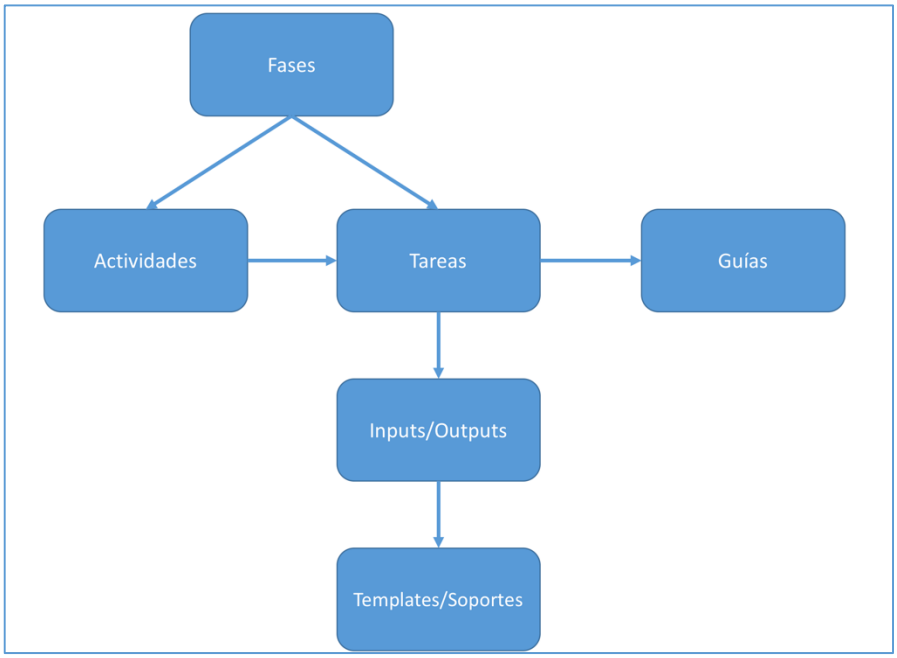

Figura 17 (Fuente: elaboración propia)

Generación sitio web de la metodología

Para generar la versión como sitio web de la metodología importar al EPF Composer el directorio `Mlibrary/`, luego ir al menú Configuration>Publish y seguir los pasos que allí se detallan. Ver Figuras 18 a 24 desplegando la secuencia de opciones para exportar en sitio web la metodología.

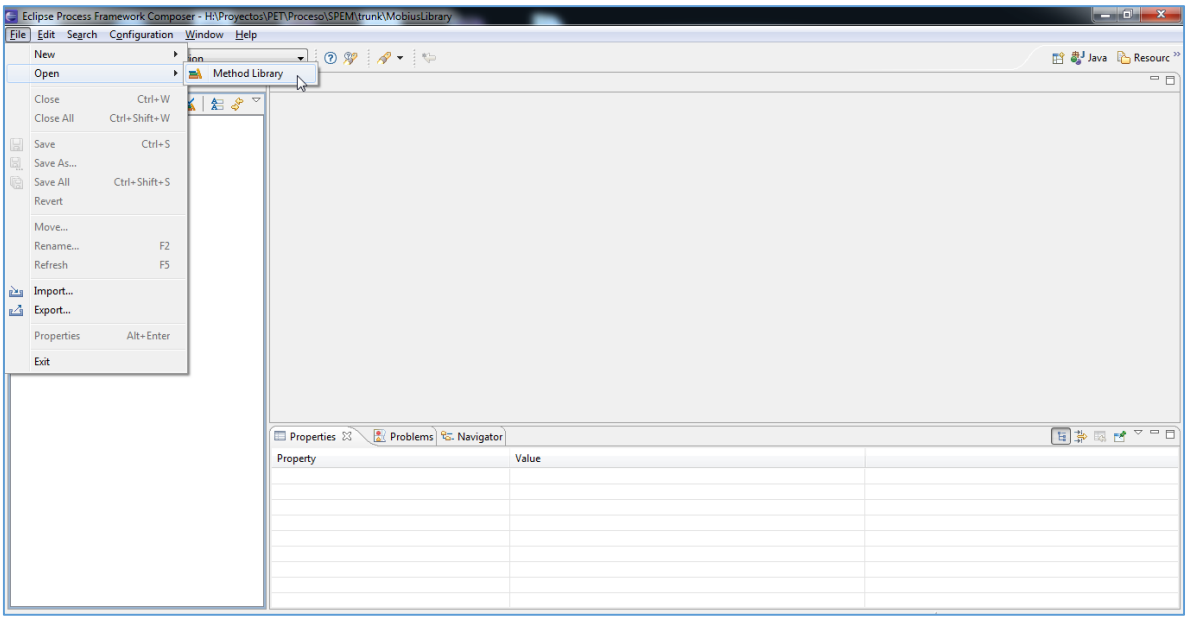

Figura 18: Exportar a sitio web paso 1 (Fuente: elaboración propia)

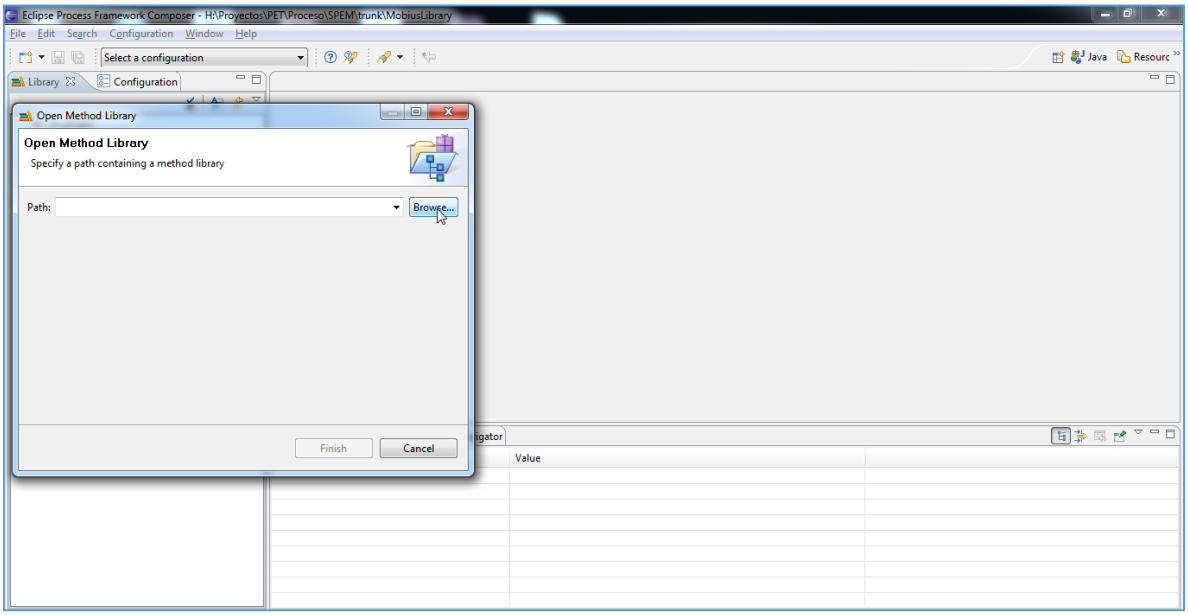

Figura 19: Exportar a sitio web paso 2 (Fuente: elaboración propia)

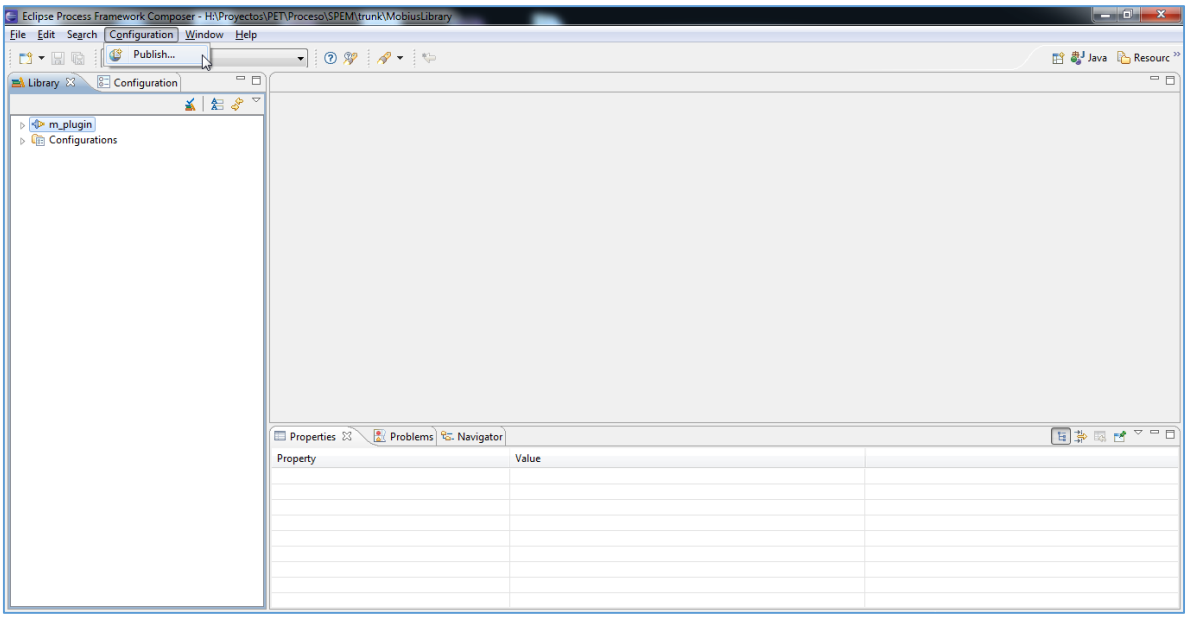

Figura 20: Exportar a sitio web paso 3 (Fuente: elaboración propia)

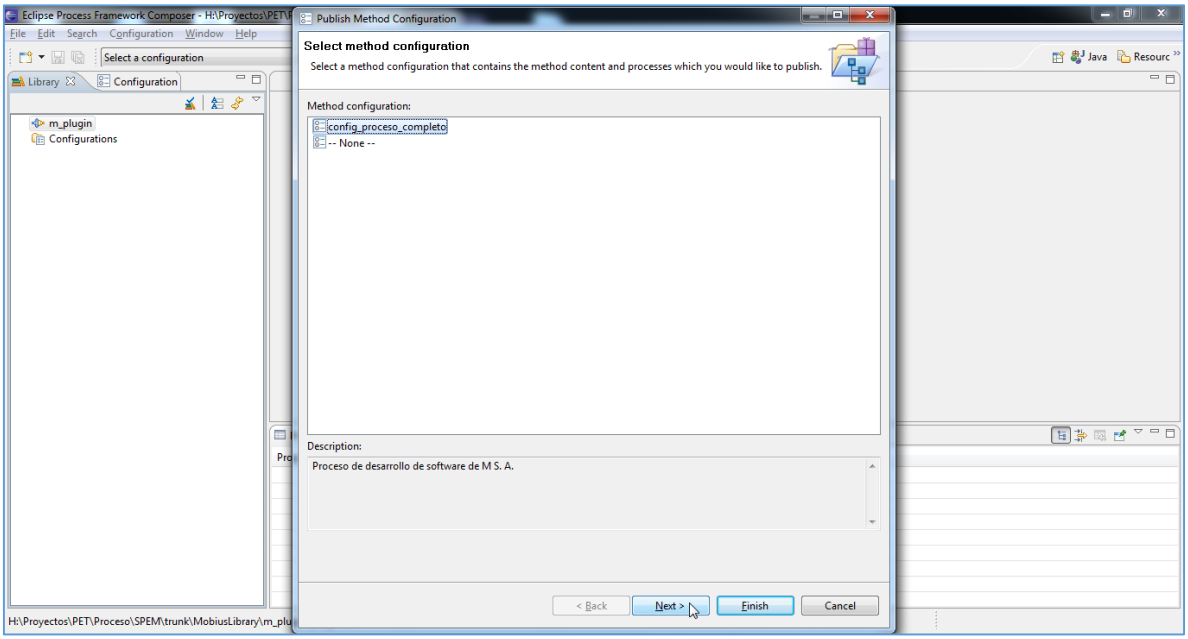

Figura 21: Exportar a sitio web paso 4 (Fuente: elaboración propia)

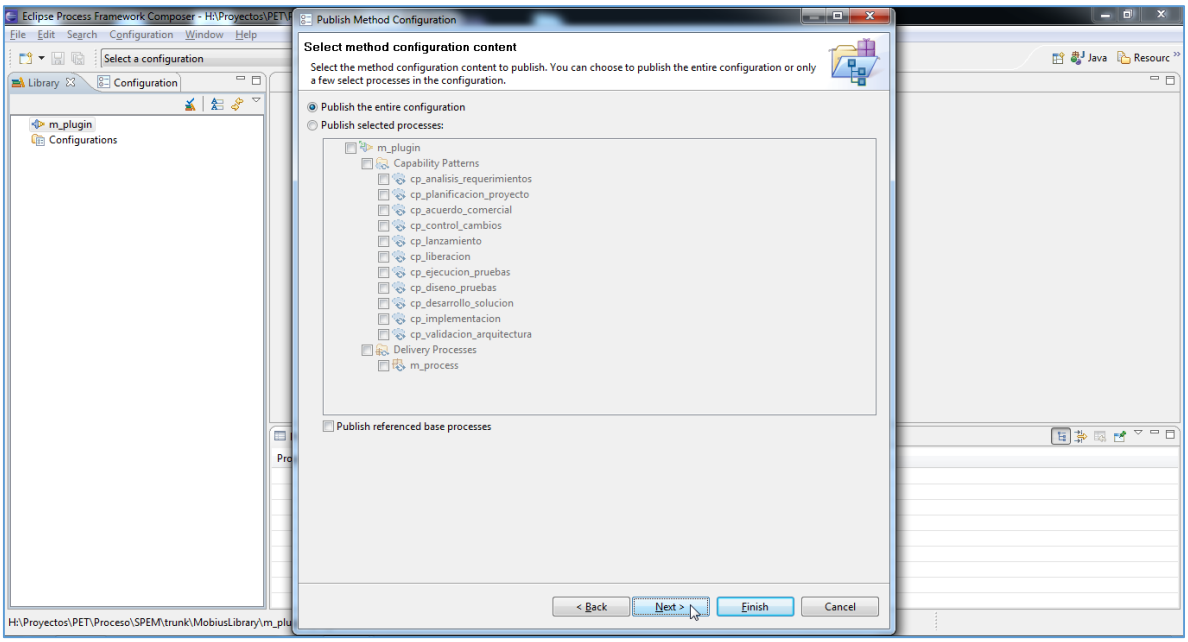

Figura 22: Exportar a sitio web paso 5 (Fuente: elaboración propia)

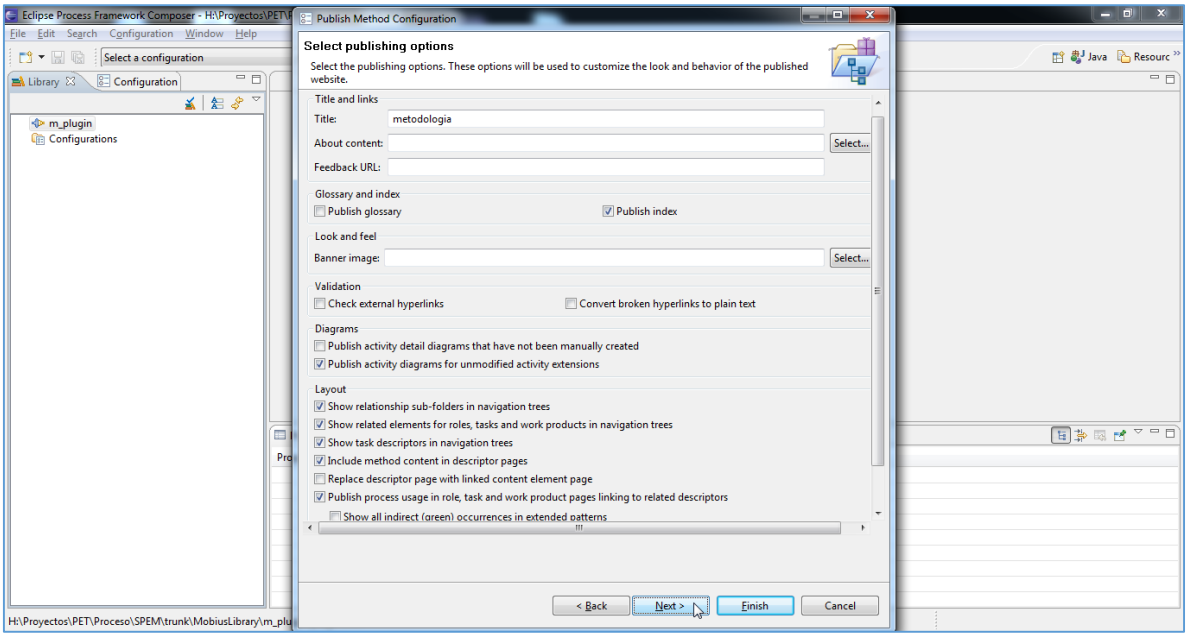

Figura 23: Exportar a sitio web paso 6 (Fuente: elaboración propia)

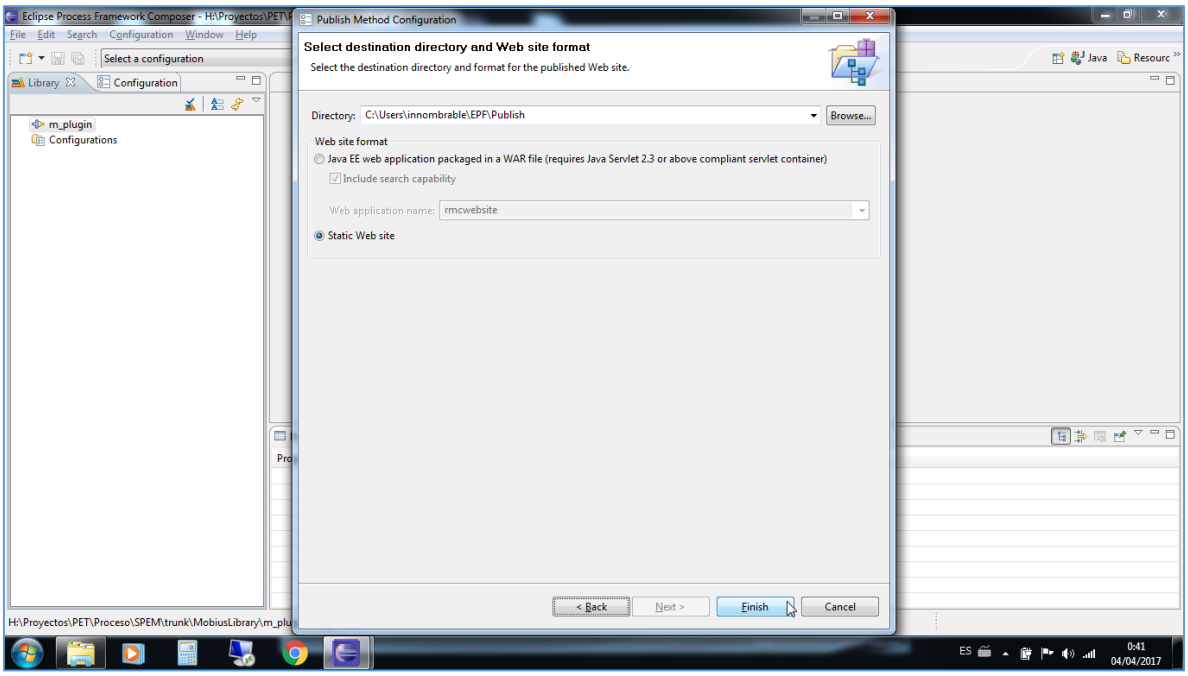

Figura 24: Exportar a sitio web paso 7 (Fuente: elaboración propia)

Para mas detalles acerca del funcionamiento de la herramienta *EPF Composer* ver [OpenUP] y visitar https://eclipse.org/epf/general/getting_started.php