Universidad de Pamplona

Facultad de Ingenierías y Arquitectura

Maestría en Gestión de Proyectos Informáticos

Modelo de despliegue de procesos de software en ambientes colaborativos y distribuidos típicos del desarrollo de Software Libre

T E S I S

Presentada como requisito para obtener el título de:

Magister en Gestión de Proyectos Informáticos

PRESENTA:

Luis Alberto Esteban Villamizar

DIRECTOR:
Fabian Alonso Lara Vargas

Pamplona, Colombia, 2019

Dedicatoria

Mi tesis de trabajo de grado, la dedico con todo mi amor y cariño a mi esposa Rosalba Mendoza Suárez, quien por medio de su sacrificio en el compartir con la familia mientras curse los estudios de maestría, y sus frecuentes llamados de atención para la culminación de este trabajo, llevaron a feliz término este proyecto.

A mi hijo Daniel Alberto Esteban Mendoza, mi inspiración en todas y cada una de las actividades que desarrollo, para que Dios lo bendiga en un largo camino personal y académico.

A mis padres que cumplieron sus bodas de oro matrimoniales, y que gracias a sus sacrificios, me ofrecieron estudios y ejemplo de vida, con lo cual he podido superar todas las adversidades. Dios les retribuya todo su amor y dedicación a la familia y nos permita a nosotros honrar sus valores de humildad, honradez y servicio a los demás.

Agradecimiento

El autor expresa sus agradecimientos al director de la tesis, profesor Fabian Alonso Lara Vargas, por su tutoría en el desarrollo del presente trabajo.

A los expertos que respondieron el llamado a evaluar el modelo, quienes dedicaron su valioso tiempo para leer el resumen del modelo y diligenciar el instrumento.

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea el resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Comité de evaluación

Jurado Calificador:

William Mauricio Rojas Contreras: _____

José Orlando Maldonado Bautista: _____

Edgar Alexis Albornoz Espinel: _____

Director de tesis:

_____

Fabian Alonso Lara Vargas

Resumen

Este trabajo define un modelo de despliegue de procesos adaptado a las necesidades de una comunidad de desarrollo de software libre y open source (Free/Libre and Open Source Software FLOSS), en la cual los miembros están distribuidos geográficamente y trabajan de manera colaborativa sin un contrato de compromisos para con el proyecto. Este modelo fue validado mediante juicio de expertos.

Un Modelo de este tipo permite mejorar la interacción en una comunidad que desarrolla software, lo que podría llevar a la obtención de procesos mas usables para quienes por primera vez se vinculan a una comunidad de desarrollo FLOSS, teniendo claro su rol desde el principio y el proceso asignado a cada rol.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Generalidades

Este documento está organizado en cuatro capítulos, en el primero se presentan las generalidades del proyecto incluyendo los objetivos, la definición del problema y la metodología de investigación utilizada. El segundo capítulo describe los principales conceptos que estructuran el objeto de estudio y presenta un breve estado del arte relacionado con los principales trabajos de investigación que abordan de una u otra manera el objeto de estudio.

El tercer capítulo presenta el modelo propuesto para el despliegue de procesos software en comunidades de desarrollo FLOSS[1], junto a los resultados del proceso de validación, realizado mediante el diseño y aplicación de un instrumento a expertos seleccionados para la evaluación del modelo aquí propuesto.

Finalmente el capitulo cuarto presenta las conclusiones y resultados obtenidos durante el desarrollo del proceso de investigación, y como anexos se incluye un resumen del modelo presentado a los expertos para su valoración y el instrumento aplicado.

1.1. Presentación

En este capitulo se describen los alcances del proyecto en términos del problema que aborda, su justificación, objetivos y metodología de investigación seguida para el logro de los objetivos planteados.

[1] También conocido como FOSS o FLOSS, siglas de Programas Libres y de Código Abierto - Sigla en Español PLiCA poco usada. Esta abreviatura abarca los conceptos de programas libres y programas de código abierto, que para este trabajo son sinónimos, pero que tienen diferencias en sus aspectos filosóficos que destaca la Free Software Foundation el adjetivo libre se enfoca en las libertades filosóficas que les otorga a los usuarios mientras que el software de código abierto se enfoca en las ventajas de su modelo de desarrollo.

1.2. Objetivos

1.2.1. Objetivo General

Construir un modelo que facilite el despliegue de un proceso de software dentro de una comunidad de desarrollo FLOSS, adecuada a las necesidades de desarrollo de software web y en las cuales los miembros de la comunidad no interactúan cara a cara, sino a través de herramientas de trabajo colaborativo, y principalmente de manera voluntaria entendido como que no tienen un contrato de compromiso para con el proyecto.

1.2.2. Objetivos específicos

- Realizar el estado del arte sobre el despliegue de procesos de software en ambientes colaborativos y distribuidos propios de las comunidades de desarrollo de software libre, que permita determinar las componentes de un modelo de despliegue usando como criterios la simplicidad del proceso, la facilidad de aprendizaje y su adaptabilidad, que motiven la vinculación y participación activa de miembros que no interactúan personalmente sino a través de herramientas colaborativas digitales y que ademas no tienen un contrato remunerado de compromiso para con el proyecto.
- Modelar el despliegue de procesos de software en estos ambientes colaborativos para estas comunidades, mediante el análisis documental obtenido en el estado del arte, que permita evaluar y mejorar la interacción entre los miembros de una comunidad de una manera mas eficiente con el objetivo de motivar la vinculación de nuevos miembros, sin necesidad de experiencias previas.
- Validar el modelo propuesto mediante la valoración de expertos, bajo esquemas de consenso y con criterios preestablecidos de valoración.

1.2.3. Acotaciones

El diseño de procesos de software es previo a su despliegue, este proyecto hará recomendaciones para el diseño de procesos, pero su principal producto se centra en la documentación consensuada del proceso dentro de una comunidad FLOSS y la aplicación (despliegue) de dichos proceso al interior de una comunidad.

1.3. Planteamiento del problema

El Instituto de Ingeniería del Software de la Universidad Carnegie Mellon University en los Estados Unidos (SEI)[1] es probablemente el organismo de mayor representatividad

[1]El Software Engineering Institute (SEI) es un centro de investigación y desarrollo financiado por el gobierno federal (FFRDC) patrocinado por el Departamento de Defensa (DoD). Es operado por la Carnegie

en el mundo científico de la Ingeniería del Software, y dentro de sus áreas de trabajo[1] se encuentran los procesos de software, cuyo producto más representativo podría ser los modelos de madurez y principalmente el CMMI[2](Team, 2002a,b).

Dentro de su estructura orgánica se encuentra la comisión de investigación en Procesos (IPRC) que reúne a 27 líderes del mundo académico y la industria para estudiar las consecuencias de escenarios futuros plausibles para las investigaciones sobre los procesos de software. En el año 2006 se proyectó un marco de trabajo para la investigación en esta área conocido como el IPRC Framework[3] (Forrester et al., 2006), documento que se ocupa de cómo las comunidades representadas por los miembros IPRC, deben invertir en la investigación sobre el proceso de software durante la siguiente década de la publicación[4]. Al proporcionar temas de investigación organizados de manera sistemática y preguntas de interés particular, el marco sirve como una herramienta de enfoque para la industria, los investigadores en la determinación de las cuestiones más fructíferos para abordar en sus programas de investigación de sus propio proceso de desarrollo.

En términos generales dicho framework propone organizar la investigación en cuatro grandes temas: tema Q[5] Relación entre los procesos y las cualidades del producto, Tema E[6]

Mellon University.

[1]Todo sus áreas en relación a la Ingeniería del software: Medición y Análisis, Rendimiento y Confiabilidad, Computación Móvil generalizada, Procesos y Mejora del Rendimiento, Gestión de riesgos,Seguridad y Supervivencia, Red inteligente,Arquitectura de software,Líneas de Producto Software, Sistema de sistemas, Sistemas Ultra y de Gran Escala.

[2]Del Ingles Capability Maturity Model Integration. Es un modelo de evaluación de los procesos de una organización dedicada al desarrollo de software ya sea como proceso misional o de apoyo a los procesos de cualquier organización.

[3]The International Process Research Consortium: A Process Research Framework.

[4]Si bien es cierto que ya esta terminando la década desde la publicación del framework, es probable que se demore unos años mas recogiendo informes que podría dar luces sobre los resultados de este proceso.

[5]Q de Qualities. Este tema hace énfasis en la perspectiva del producto. Tiene que ver con la comprensión de si y cómo las características de un proceso pueden afectar producto software deseado. Cualidades o características tales como la seguridad, facilidad de uso y facilidad de mantenimiento.

[6]E de Engineering. Este tema hace énfasis en la perspectiva del proceso. Tiene que ver con la forma de definir y construir procesos y comprender su desempeño.

Ingeniería de procesos, tema P[1] procesos de gestión de proyectos, y el tema D[2] despliegue de procesos(Forrester et al., 2006).

Este trabajo de investigación se centra en el despliegue inicial de un proceso de software, lo que incluye algunos aspectos de la Ingeniería de procesos software en ambientes distribuidos y colaborativos como los exigidos para el desarrollo de aplicaciones, bajo los principios de FLOSS. Particularmente se requiere que estos procesos se puedan describir con una adecuada notación como SPEM2(Object Managemen Group, 2008), centrado en la descripción de roles a través de las actividades bajo responsabilidad del rol, apoyados en artefactos[3] bien definidos y herramientas[4] que ayudan a la generación de dichos artefactos.

El software libre según Stallman et al. (1999), como cualquier otro tipo de aplicaciones requiere de procesos de software adecuadamente diseñados y desplegados para motivar la voluntaria participación de los interesados en el desarrollo desde diversos roles, necesarios y suficientes para cada uno de los proyectos. Es por esta razón que las comunidades FLOSS definen de manera simplificada sus procesos, evitando la desmotivación de un participante que se pudiere dar por un proceso con excesivo trabajo innecesario de acuerdo a su propio criterio. Es por esto que los conceptos de agilidad como los referidos por Sillitti and Succi (2005) deben ser tenidos en cuenta al momento de diseñar y desplegar un proceso de software en un ambiente de desarrollo FLOSS donde la colaboración es voluntaria y los colaboradores no se conocen y probablemente no puedan interactuar cara a cara.

Sin embargo se han realizado diversos estudios que demuestran la poca, escasa o nula documentación del proceso seguido al interior de una comunidad de software libre (Barbosa and Alves, 2011; González-Barahona and Koch, 2005; Muruzábal and Castillo Acuña, 2014) y la informalidad de los procesos como la captura de requerimientos que en muchas ocasiones se hacen por medio de foros o intercambio de correos entre los participantes. Esto puede ser un factor positivo para quienes participan desde la creación del proyecto o han sido activos participantes en diversos proyectos, pero puede ser un factor negativo

[1] P de Project. Este tema hace énfasis en la perspectiva de la organización del proyecto. Tiene que ver con valores de los actores que impulsan las estructuras organizativas utilizadas para llevar a cabo el trabajo del proyecto. Estos valores pueden ser políticos, económicos o sociales. Pueden ser impulsados por las preocupaciones particulares de un dominio de negocio, las condiciones del mercado, o la naturaleza de los entornos competitivos.

[2] D de Deployment. Este tema hace hincapié en la perspectiva de la gente. Tiene que ver con la obtención de los procesos adecuados y desplegados de manera efectiva en las estructuras organizativas adecuadas para que las personas estén habilitadas de manera más eficiente para satisfacer las necesidades de una empresa.

[3] En los procesos de desarrollo de software existen variedad de artefactos entre los cuales se pueden citar: Descripción de requerimientos, Diseños, código fuente, casos de prueba, manuales de usuario, etc. y otros artefactos de gestión como planes de diferente índole, cronogramas, listados de riesgos, presupuestos, etc.

[4] Herramientas tanto para la gestión de proyectos como gestores de actividades, manejo de versiones, seguimiento de bugs, etc, y de soporte a actividades técnicas como entornos integrados de desarrollo, editores, compiladores, generadores de casos de pruebas, etc.

para quienes por primera vez quieren hacerse participes dentro de una comunidades.

De igual manera el despliegue del proceso, básicamente es impuesto por el líder del proyecto o por la forja[1] en la cual se desarrolla el proyecto, y los interesados en participar de manera voluntaria sin ningún tipo de remuneración económica, sin contrato de compromiso, tan solo tienen la oportunidad de seleccionar uno de los roles definidos por la comunidad como formas de colaboración.

En este contexto es importante entonces, plantear como pregunta de investigación ¿De que manera desplegar un proceso de desarrollo de software simplificado, que facilite la evolución exitosa de un proyecto FLOSS, en el cual los interesados están distribuidos geográficamente y participan de manera voluntaria sin ningún tipo de contratación ni compromisos, y en la mayoría de los casos no interactúan físicamente sino a través de medios digitales?

1.4. Metodología

Los procesos de investigación requieren organizarse en términos de actividades tendientes a obtener el producto de investigación. Estas actividades dependen del enfoque, alcance, diseño, fuentes e instrumentos del trabajo investigativo.

1.4.1. Enfoque

De acuerdo a Hernando Ramírez and Anne Marie (2012) uno de los criterios que se pueden usar para determinar la conveniencia de un enfoque cualitativo o cuantitativo puede estar en la forma como se aborda el proceso en términos de estrategia o táctica: La estrategia (Cualitativo) es un proceso regulable, conjunto de las reglas que aseguran una decisión óptima en cada momento, mientras que la táctica (Cuantitativa) es el arte que enseña a poner en orden las cosas. Método o sistema para ejecutar o conseguir algo.

Es por lo tanto que la presente investigación tiene un enfoque cuantitativo en el sentido utiliza una táctica predefinida que da un orden cronológico a las etapas a seguir en la investigación. Por otro lado esta investigación se centra en la validez de un modelo de despliegue y su confiabilidad mediante la valoración por parte de expertos y el tratamiento de estas valoraciones es una característica de los enfoques cuantitativos.

[1] Una forja es un portal web que sirve de albergue o plataforma de desarrollo colaborativo. Se enfoca en permitir la cooperación entre desarrolladores para la difusión de software y el soporte al usuario. En este tipo de espacios en la nueve se albergan múltiples proyectos de software, y permiten que los desarrolladores se registren voluntariamente en un determinado proyecto para poder contribuir.

1.4.2. Alcance

Esta investigación define un alcance exploratorio, pues antecede en el contexto a cualquier otro tipo de investigación que se ha realizado en la maestría[1] respecto al despliegue de procesos de software, de igual manera intenta innovar en la forma conveniente de despliegues de procesos, identificando los elementos mínimos y suficientes, necesarios en el despliegue de procesos de desarrollo FLOSS.

1.4.3. Diseño de investigación

Según Dávila (1995), en la investigación se tienen diseños tácticos y diseños estratégicos; cuando el diseño es táctico, este se presenta en etapas, existe un orden en el tiempo, una jerarquía; hay un criterio lógico, entre premisas y conclusiones.

Es por esta razón, dentro de esta investigación con enfoque cuantitativo se presenta un diseño en forma de etapas predefinidas antes de iniciar el proceso investigativo que incluye: revisión de conocimientos y antecedentes, replanteamiento del problema, construcción del marco teórico, planteamiento de hipótesis, aplicación de técnicas e instrumentos en la comprobación de hipótesis, validación de resultados.

Una investigación puede ser formulada como una expresión lógica de la forma: $(h_1, h_2, h_3...h_k) \rightarrow T$, donde T es la tesis que se desea demostrar y para lo cual es suficiente y necesario demostrar la veracidad de cada una de las hipótesis h_i las cuales deben ser sustentadas y/o demostradas usando diversos métodos y técnicas de investigación.

Estas hipótesis de investigación son el fundamento o base, sobre la cual se sostiene la tesis, y por tal motivo de su validez depende la validez del resultado de investigación.

Este trabajo de investigación define como hipótesis:

1. Los procesos de desarrollo software, carecen de rigor procedimental debido a su carácter colaborativo[2].(Variable: nivel de rigor)

2. Las herramientas de comunicación son indispensables en las comunidades FLOSS dado su carácter distribuido[3] (Variable: Nivel de dependencia del proceso en herramientas)

3. Es posible formalizar el proceso de desarrollo en una comunidad FLOSS, de manera que motive la participación del voluntariado novato en este tipo de comunidades.(Variable nivel de formalización del proceso software)

Sobre estas hipótesis y su validez, se soporta la siguiente tesis:

Se puede diseñar un modelo de despliegue de procesos software, aplicables a comunidades FLOSS que facilite la incorporación de voluntarios novatos.

Para soportar la validez de las hipótesis fue necesario, recurrir a diversos métodos y técnicas de investigación y consulta bibliográfica de la siguiente manera:

[1] Maestría en Gestión de proyectos informáticos de la Universidad de Pamplona, para lo cual este trabajo se desarrollo como requisito para obtar por el grado de Magister.

[2] Los miembros del equipo participan voluntariamente sin remuneraciones económicas ni compromisos contractuales.

[3] Los miembros no se encuentran en un mismo espacio físico.

Métodos y herramientas utilizadas para la Hipótesis 1: Argumentación mediante consulta bibliográfica, o investigación documental, principalmente de artículos y trabajos de exploración sobre algunas comunidades de desarrollo de software libre.

Métodos y herramientas utilizadas para la Hipótesis 2: Argumentación mediante consulta bibliográfica, o investigación documental, principalmente de artículos y trabajos de exploración sobre las herramientas de soporte a procesos de software en algunas comunidades FLOSS.

Métodos y herramientas utilizadas para la Hipótesis 3: validación del modelo propuesto mediante juicio de expertos, a los cuales se aplicará instrumentos de valoración del modelo.

1.4.4. Fuentes de información

Las principales fuentes de información en esta investigación se centran en artículos de investigación relacionados, trabajos de pregrado en los que se estudian los procesos de comunidades FLOSS particulares, lo que permitirá establecer el modelo a validar.

En este proceso de validación se tendrán como principales fuentes de información un conjunto de expertos, conocedores del movimiento FLOSS.

1.4.5. Población y muestra

Cada comunidad FLOSS define su proceso de desarrollo y por lo tanto este trabajo considera como población, todas las comunidades FLOSS, aunque este número es indeterminado, se disponen meta repositorios[1] con información del comportamiento de algunas de las principales comunidades.

El muestreo sobre esta población no sigue una formula determinista, por lo tanto se toma a conveniencia, dependiendo de estudios previos sobre comunidades puntuales encontrados en la literatura. Desde un segundo punto de vista la población puede considerar a los expertos que han participado en comunidades FLOSS, cuyo número puede ser estimado en millones de personas en todo el mundo, pero que por conveniencia de esta investigación se esperan seleccionar por lo menos 7 expertos que conozcan de comunidades FLOSS, y por lo tanto puedan evaluar el modelo propuesto como producto de la investigación.

1.4.6. Instrumentos

El análisis documental se constituye en el principal instrumento de recolección de datos para el soporte de las hipótesis 1 y 2 definidas en el diseño de investigación.

Un segundo tipo de instrumentos se hace necesario para demostrar la hipótesis 3 que incluye la encuesta a los expertos seleccionados.

[1] Repositorios tipo forjas como sourceforce, los repositorios de manejo de versiones como GitHub y principalmente los metarepositorios de comunidades FLOSS como FLOSSmole y FLOSSMetrics.

1.4.7. Tipos de análisis

Dos tipos de análisis se realizarán dentro de la demostración de las hipótesis 1 y 2: número de roles por comunidades de desarrollo de software libre y tipos de herramientas colaborativas utilizadas en las comunidades

Otro tipo de análisis estadístico se realizará en la validación de modelo propuesto, básicamente se centrará en análisis de dispersión de las respuestas de los expertos seleccionados, en busca de un consenso de opinión.

1.5. Contribuciones

La principal contribución de este trabajo es la comunicación simple de procesos en comunidades FLOSS, de tal manera que la vinculación de miembros a un proyecto sea efectiva, dada la comprensión de las tareas que un miembro debe desarrollar según su rol dentro del proyecto. Se esperaría que una comunidad que explique de manera clara sus procesos de desarrollo, incrementaría la participación de nuevos miembros poco expertos, pero con interés de aportar

Capítulo 2

Marco teórico

Este trabajo tiene como objeto de estudio del despliegue de procesos software, aplicado en ambientes FLOSS, en los cuales existen dos desafíos constantes para cualquier comunidad: la distribución geográfica de los participantes y la colaboración voluntaria. Es así como en esta sección se describen de manera general los conceptos mas importantes alrededor de procesos software, y sus particularidades en el contexto de las comunidades FLOSS.

2.1. Los proyectos de software libre

La noción hacia el concepto de software libre viene de la traducción "free software" pero ha sido mal interpretada ya que en inglés la palabra "free" puede tomar dos significados: gratis o libre, en nuestro contexto típicamente las personas lo entienden como gratis.

La idea de software libre va más allá de utilizar una herramienta informática bajo una licencia de uso sin pagar dinero por ello, según la Free Software Fundation (FSF) el software libre le da al usuario la libertad de poder ejecutar, estudiar, compartir y modificar el software, estas son las cuatro libertades que un producto software debe tener para que sea reconocido como libre, esto quiere decir que el software es una elección de libertad y no de precio, es por esto que el movimiento recibió el nombre de software libre ya que el usuario es libre

Estas libertades han sido descritas así:

Libertad 0: Ejecutar el programa como se desea, con cualquier propósito

Libertad 1: Estudiar cómo funciona el programa y cambiarlo para que haga lo que usted quiera. El acceso al código fuente es una condición necesaria para ello.

Libertad 2: Redistribuir copias para ayudar a otras personas.

Libertad 3: Distribuir copias de sus versiones modificadas a terceros. Esto le permite ofrecer a toda la comunidad beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

Por otro lado surgió como consecuencia directa de las libertades 1 y 3 un nuevo movimiento denominado "Open Source" (código abierto) que está muy relacionado con el

software libre, y es así que el proyecto GNU expone que un sector de la comunidad de software libre decidió tomar un nuevo rumbo para promover el "Open Source".

Dentro de este movimiento cabe destacar la participación de Erick Raymond y Bruce Perens, este grupo de personas ve los beneficios para el software que se producen cuando el código está disponible para todos, luego el concepto filosófico de "Open Source" se basa en la idea de mejorar el software en un sentido esencialmente práctico, dejando de un lado las ideas filosóficas de las libertades del software libre y definiendo como principales premisas (Stallman et al., 1999):

- Libre redistribución: el software debe poder ser regalado o vendido libremente.
- Código fuente: debe estar incluido u obtenerse libremente.
- Trabajos derivados: la redistribución de modificaciones debe estar permitida.
- Integridad del código fuente del autor: las licencias pueden requerir que las modificaciones sean redistribuidas sólo como parches.
- Sin discriminación de personas o grupos: nadie puede dejarse fuera.
- Sin discriminación de áreas de iniciativa: los usuarios comerciales no pueden ser excluidos.
- Distribución de la licencia: deben aplicarse los mismos derechos a todo el que reciba el programa.
- La licencia no debe ser específica de un producto: el programa no puede licenciarse solo como parte de una distribución mayor.
- La licencia no debe restringir otro software: la licencia no puede obligar a que algún otro software que sea distribuido con el software abierto deba también ser de código abierto.
- La licencia debe ser tecnológicamente neutral: no debe requerirse la aceptación de la licencia por medio de un acceso por clic de ratón o de otra forma específica del medio de soporte del software.

Para entender las diferencias entre estos dos movimientos se debe entender sus fundamentos filosóficos: uno se basa en los ideales de libertad y el otro en los ideales de llevar lo más óptimo posible el producto software, es decir el software libre se centra en principios morales y éticos (la libertad del usuario) y el Open Source se centra meramente en aspectos técnicos.

Para este trabajo de investigación estos dos movimientos no tienen ninguna diferencia en cuanto a que cualquiera sea el enfoque estas comunidades desarrollan software utilizando un proceso de desarrollo (objeto de estudio de esta investigación), y el código fuente debe estar disponible para poder llevar a cabo el proceso.

2.2. El proceso de software

En un contexto generalizado existen muchos conceptos respecto a lo que significa proceso:

- Un proceso (del latín processus) es un conjunto de actividades o eventos que se realizan o suceden (alternativa o simultáneamente) con un fin determinado
- Conjunto de actividades que, realizadas en forma secuencial, permiten transformar uno o más insumos en un producto o servicio.
- Es una secuencia temporal de ejecuciones de instrucciones que corresponde a la ejecución de un programa secuencial.
- Es cualquier operación o secuencia de operaciones que involucren un cambio de energía, estado, composición, dimensión, u otras propiedades que pueden referirse a un dato
- "... Proceso es la serie de pasos utilizados para producir un resultado deseado"... según el Concise American Heritage Dictionary, 1987

En el contexto del desarrollo de software un proceso de software puede ser definido como:

- Conjunto de actividades, métodos, prácticas y transformaciones que las personas usan para desarrollar y mantener software y los productos de trabajo asociados (planes de proyecto, diseño de documentos, código, pruebas, manuales de usuario, etc.)
- Proceso o conjunto de procesos usados por una organización o proyecto, para planificar, gestionar, ejecutar, monitorizar, controlar y mejorar sus actividades software relacionadas.
- Conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necearías para concebir, desarrollar empaquetar y mantener un producto software
- Conjunto parcialmente ordenado de actividades llevadas a cabo para gestionar, desarrollar y mantener sistemas software
- El proceso software define como se organiza, gestiona, mide, soporta y mejora el desarrollo, independientemente de las técnicas y métodos usados

Los cuatro principales temas indispensables para la definición y despliegue de un proceso software son: Los modelos de ciclo de vida del software, los estándares para la definición de procesos, las metodologías de desarrollo y los modelos de mejoras de procesos.

Mientras que los modelos de ciclo de vida solo explican de manera general un conjunto de faces en el desarrollo de software, las metodologías van mas allá explicando no solo las fases sino actividades dentro de las fases, los roles necesarios para la ejecución de las

actividades, los artefactos generados en cada actividad y las herramientas que se deben utilizar en cada una de esas actividades.

Por otro lado algunos estándares como los de la IEEE 12207 y 1074 brindan una subdivisión del proceso global de desarrollo software[1], lo que hacen de estos estándares ideales para garantizar la identificación de actividades propias del proceso.

Finalmente es necesario el estudio de modelos de mejoras de procesos ya que una vez definidos un proceso dentro de una organización puntual, se hace necesario el despliegue de dicho proceso (puesta en funcionamiento) y su adecuada adaptación según las necesidades de cambio sugeridas por una adecuada evaluación de la eficacia y eficiencia del proceso

2.2.1. Modelos clásicos de ciclo de vida

Los modelos de ciclo de vida, inicialmente (en los años 70)considerados metodologías para el desarrollo de software, se constituyen hoy en día en elementos fundamentales para el diseño de procesos, pues estos modelos muestran de manera global la forma de organizar las actividades (no definidas por el modelo) dentro de un proyecto de desarrollo de software. A diferencia de las metodologías de software, los modelos clásicos de ciclo de vida no pretenden definir los detalles de un proyecto de desarrollo y tan solo definen las grandes fases y las condiciones para la transición entre fases.

2.2.1.1. Modelo en Cascada

El modelo en cascada [2] se define como una secuencia de fases en la que al final de cada una de ellas se verifica si reúne la documentación para garantizar el inicio de la siguiente fase. De esta forma cualquier error en una fase anterior se puede propagar de manera acumulando hasta el final del proyecto.

Sin embargo este modelo es conceptualmente útil para pequeños proyectos donde se puede reconsiderar la posibilidad de retroalimentación como muestra la figura 2.1. Esta idea de secuencialidad en grandes fases del desarrollo esta hoy en día re-valorada por conceptos iterativos e incrementales, sin embargo todo proyecto aunque no tenga fases dependientes, también tiene secuencias o dependencias en actividades y por lo tanto se puede ver pequeñas cascadas en proyectos iterativos o incrementales.

Dentro de las diferentes variantes del modelo en cascada se puede resaltar tres: con retroalimentación 2.2, por subproyectos 2.3 y por etapas 2.4.

En el modelo de cascada con retroalimentación, se da la posibilidad de un análisis de problemas al final de cada fase, de tal manera que al identificarlas, se reinicie la fase respectiva donde fue detectado el problema.

El modelo de cascada por subproyectos presenta una manera de paralelismo en actividades, cuando un producto software se puede construir con módulos altamente independientes, y es aunque las fases iniciales constituyen una cascada, el desarrollo de

[1] En subprocesos sin dar un secuenciamiento especial, pero definiendo el alcance de cada uno.

[2] comenzó a diseñarse 1966 y tuvo auge hasta alrededor de 1970.

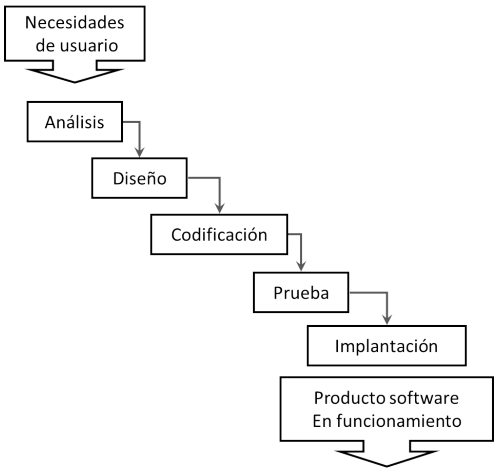

Figura 2.1: Modelo clásico en cascada

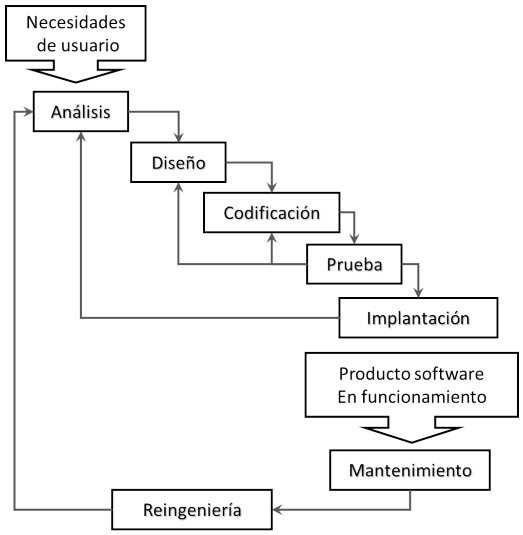

Figura 2.2: Modelo general de cascada con retroalimentación

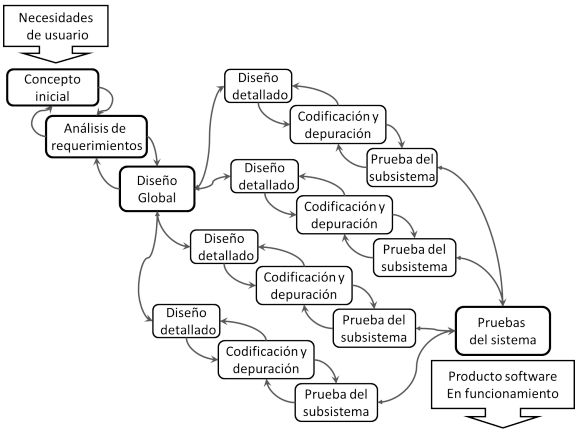

Figura 2.3: Modelo variante de cascada con subroyectos

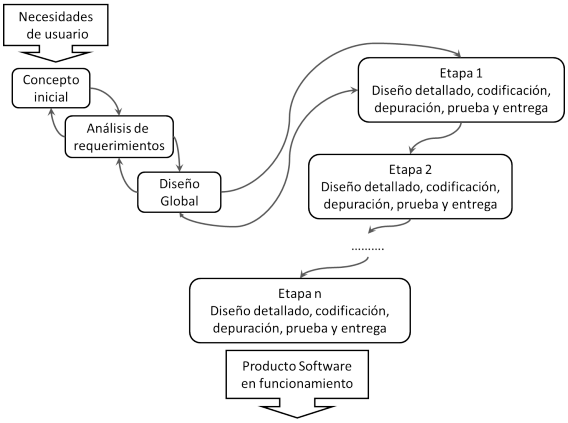

Figura 2.4: Modelo variante de cascada por etapas

cada modulo puede ser entendido como cascadas independientes pero paralelas(ver figura 2.3).

El modelo de cascada por etapas, mantiene la secuencia de fases en la construcción de módulos, cuando estos tienen alta dependencia unos de otros, por lo cual las cascadas para cada modulo deben ser adecuadamente secuenciadas en función de relación de dependencia entre sus módulos (ver figura 2.4).

2.2.1.2. Modelo en V

El modelo en V corresponde a una cascada, en las que una fase temprana de un proyecto anticipan la planificación de una fase final, y cuando es ejecutada la fase final los artefactos generados en la fase inicial correspondiente, se utilizan para hacer procesos de validación

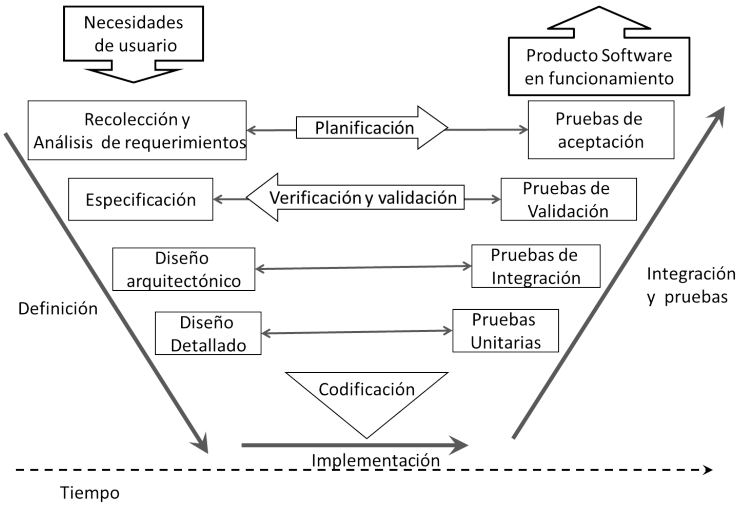

Figura 2.5: Modelo en V

y verificación (ver figura 2.5).

2.2.1.3. Modelo en espiral

El modelo en espiral fue la base para la concepción de los actualmente muy conocidos y recomendados procesos iterativos e incrementales, sin embargo se debe resaltar que el aporte mas importante de este modelo corresponde a la importancia que le da a las actividades de gestión dentro del proyecto, casi en igual o mayor proporción que las actividades técnicas típicamente asociadas a un proyecto de software como diseño, codificación y pruebas(ver figura 2.6).

Actualmente es reconocida la importancia de la gestión de proyectos software, para garantizar el éxito de un producto y es por esto que son relevantes las actividades de gestión tales como manejo de riesgos, gestión de alcance, tiempo, costo, calidad, comunicaciones, adquisiciones e incluso la administración de personal ya sea del proyecto o de personas interesadas, entre otras temáticas de gestión, para lo cual el ingeniero de software que dirija un proyecto debe conocer y dominar la aplicación de diversas técnicas y herramientas en cada uno de estos temas, típicamente abordados por diversas metodologías de gestión de proyectos como PMBOK(PMI, 2014, 2017), Scrum(Schwaber Ken, 2017), Prince2(Keith and Lawrence, 2015), entre otras.

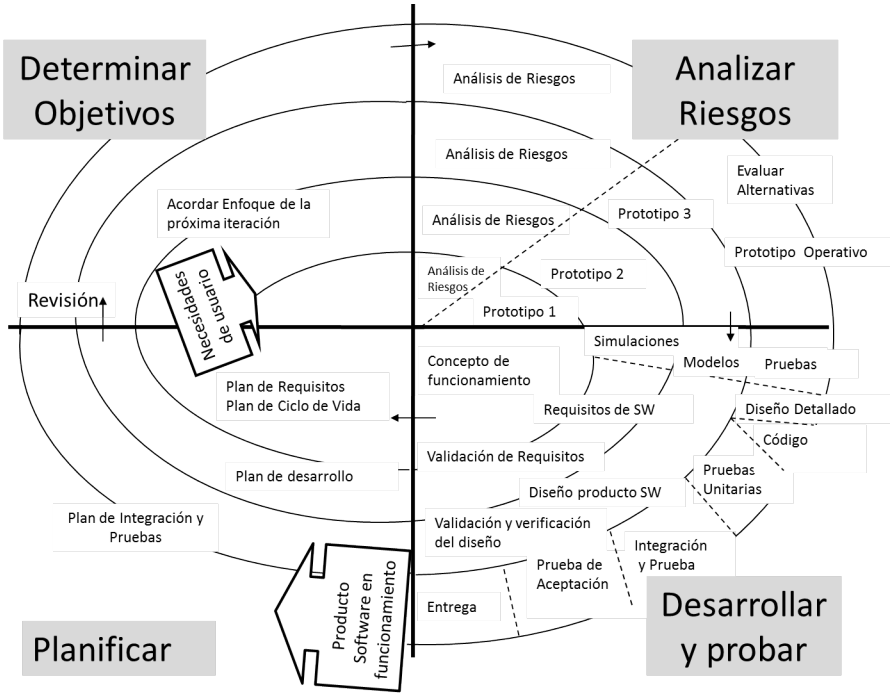

Figura 2.6: Modelo en Espiral

2.2.1.4. Modelo de Prototipado evolutivo

El prototipado de software se ha visto desde dos perspectivas: como modelo de desarrollo y como técnica de captura de requerimiento. Como técnica de captura de requerimientos el código utilizado para generar el prototipo se vuelve desechable, es decir que este código no se utiliza en la construcción del producto final, a pesar de esto el prototipo cumple su finalidad al ser visto por el usuario quien puede garantizar la adecuada interpretación de su requerimiento por parte de los ingenieros de software. Visto como modelo de desarrollo el prototipado evolutivo, esto corresponde de manera muy cercana al concepto de desarrollo iterativo, en el cual el primer prototipo de un producto software se considera un borrador que se valida, se refactoriza y se sigue agregando funcionalidades sobre el mismo borrador, hasta que se el borrador se convierte en el producto final(ver figura 2.7).

2.2.1.5. Modelo Iterativo

La palabra iteración es sinónima de repetición o reiteración, y en el contexto de software hace referencia a la repetición de procesos como requerimientos, diseño, codificación y pruebas en el desarrollo de un producto software.

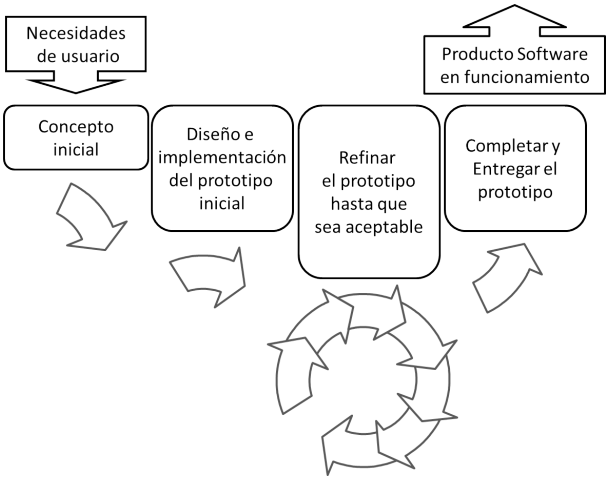

Figura 2.7: Modelo de Prototipado Evolutivo

En cada una de las iteraciones en un proceso de desarrollo software, se va construyendo un borrador, se valida, y luego se sigue agregando detalles de funcionalidad y mejora del producto, sin perder la visión global del producto final, es decir que las actividades que se repiten se aplican al producto global. Es importante resaltar que al comenzar un proceso iterativo no hay certeza absoluta sobre el resultado deseado, sino que se va construyendo a medida que se transcurren las iteraciones y se va viendo el producto. Esto implica que un producto desarrollado de manera exclusiva con el modelo iterativo solo estará listo para entrega en la ultima iteración ejecutada(ver figura 2.8).

Pese a que en los modelos iterativos se repiten las actividades como requerimientos, diseño, codificación y pruebas, es importante notar que los esfuerzos invertidos en cada una de estas actividades varia de una iteración a otra, por lo que las primeras iteraciones necesitan dedicar mayor esfuerzo en actividades de requerimientos y diseño, mientras que las finales requieren mas esfuerzo en pruebas.

También es importante aclarar que los modelos iterativos no solamente se repiten los procesos técnicos del desarrollo por lo tanto se deben repetir en cada iteración actividades de gestión como planificación, gestión de riesgos, tiempo, comunicaciones, calidad, etc.

2.2.1.6. Modelo Incremental

En un proceso incremental se debe tener una idea completa del producto final desde el inicio del proyecto por lo que al comenzar hay certeza absoluta sobre el resultado final deseado, sin embargo en cada incremento se debe tener parte del producto final terminado, lo que implica que esa parte (típicamente un modulo) debe estar en condiciones de funcionamiento (listo para ser desplegado si el cliente lo requiere).

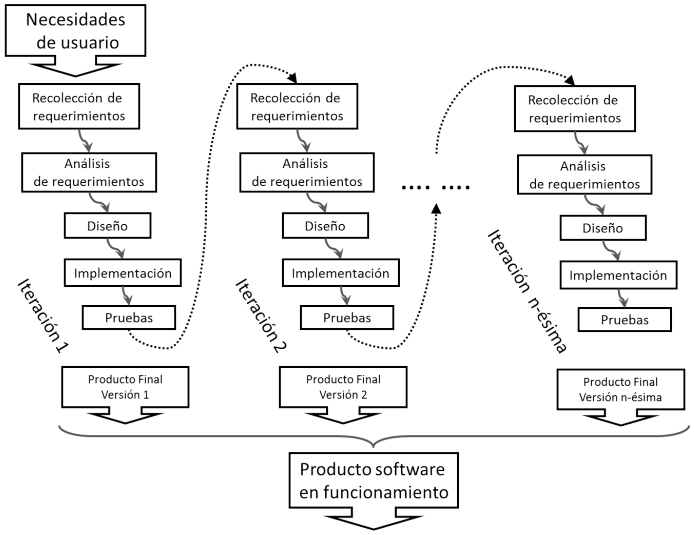

Figura 2.8: Modelo Iterativo

Al igual que los modelos iterativos, en estos modelos también se repiten las actividades en cada uno de los incrementos(requerimientos, diseño, codificación, pruebas, etc), sin embargo estas actividades están enfocadas casi exclusivamente en el incremento y no en todo el producto de manera global. A diferencia de los modelos iterativos en los modelos incrementales las entregas parciales de subproductos validados y funcionales son indispensables, por lo que cada incremento termina con una entrega de una parte del producto final en funcionamiento(ver figura 2.9).

Actualmente muchas de las metodologías de desarrollo software, combinan modelos iterativos e incrementales, con lo cual hacen coincidir una entrega con una o varias iteraciones, sin dejar de dedicar esfuerzos en cada una las iteraciones al producto global, pero haciendo los ajustes que se consideren pertinentes para mejorar el producto final, y por ello lo imaginado al inicio de los procesos, no siempre concuerda con el resultado finalmente obtenido.

2.2.2. Estándares para definición de procesos

Los estándares aquí estudiados, no representan metodologías de desarrollo de software, pero son importantes en la medida que definen los subprocesos típicos de un proceso de desarrollo de software, en algunos casos llegando hasta la definición de actividades propias de un subproceso. Es así que estos conceptos son de importancia relevante para el diseño y adaptación de un proceso de desarrollo en una organización y particularmente en una comunidad de desarrollo FLOSS.

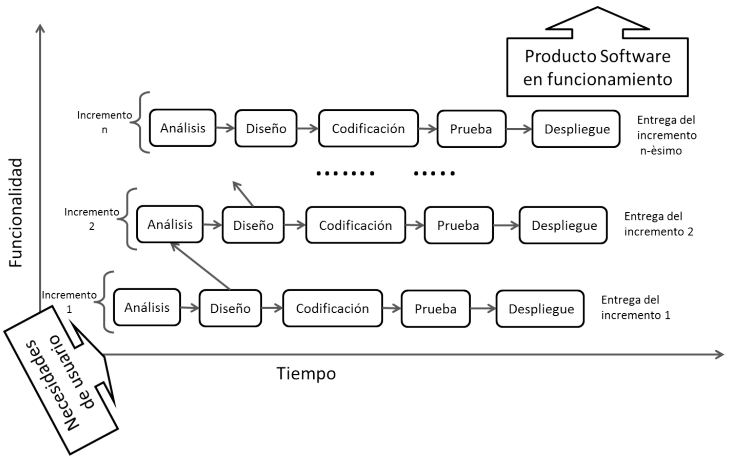

Figura 2.9: Modelo Incremental

2.2.2.1. IEEE 12207

El estándar ISO/IEC 12207 [1], define los procesos de ciclo de vida del software de una organización productora de software, desde que surge una idea o necesidad hasta que el producto software se construye, su puesta en funcionamiento y se decide retirar de su uso típicamente por obsolescencia de la tecnología(Mutafelija and Stromberg, 2008). Organiza los proceso del ciclo de vida en tres grupos: Principales, de soporte y organizacionales como se muestra en la siguiente figura 2.10:

2.2.2.2. IEEE 1074

El estándar de procesos para el desarrollo de un ciclo de vida IEEE 1074[2], define los lineamientos para el diseño de un proceso de desarrollo dentro de una organización, personalizando los procesos definidos por la norma, a las necesidades de un proyecto en particular. Este estándar no recomienda ningún ciclo de vida específico, por lo cual parte de sus procesos incluye la selección de un modelo de ciclo de vida apropiado para un proyecto, y permite el mapeo de las actividades entre el modelo de ciclo de vida seleccionado y los procesos del estándar (ver figura 2.11), incluyendo aspectos mas allá del desarrollo de un producto software como la instalación, operación y soporte, mantenimiento, retiro o jubilación del producto software y capacitación a los ingenieros en el uso de los procesos (IEEE, 2006).

[1] Information Technology / Software Life Cycle Processes.

[2] Estándar IEEE 1074 Developing Software Life Cycle Process.

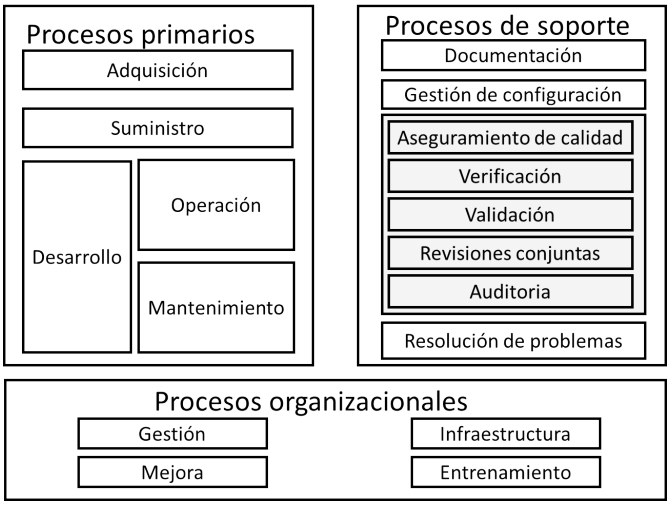

Figura 2.10: Procesos según IEEE12207

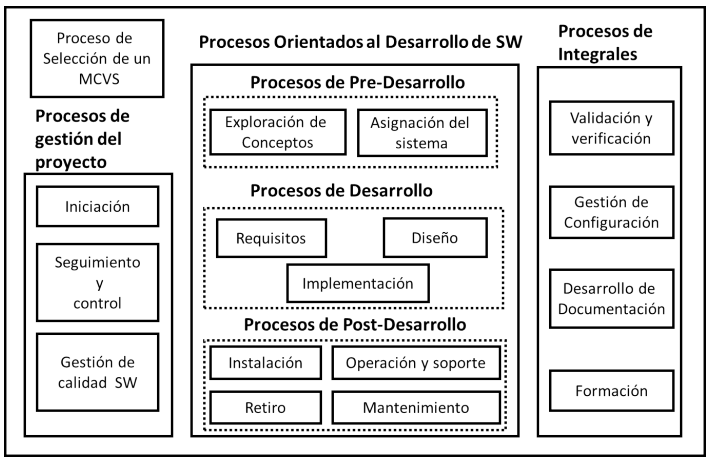

Figura 2.11: Procesos de la IEEE1074

2.2.3. Modelos de Mejora de Procesos

Gran parte de los esfuerzos de investigación en el área de despliegue de procesos software, tienen que ver con la mejora de procesos y por lo tanto con los modelos de mejora aquí descritos. Estos modelos, definen estrategias para que una organización a partir del proceso adoptado para el desarrollo de software, permitan el mejoramiento continuo de sus procesos, basado en la hipótesis de muchos modelos de calidad: Ün mejor proceso, genera un mejor producto".

Estos modelos no se constituyen por si solos en procesos de desarrollo software, pues cada organización según sus necesidades, debe partir de un proceso base, el cual debe ser continuamente evaluado y adaptado de acuerdo a los resultados obtenidos en procesos de medición de la capacidad del proceso (madurez del proceso) y de las lecciones aprendidas en su utilización dentro de proyectos puntuales.

Existen a grandes rasgos tres tipos de modelos de mejora de proceso, el primero orientado a los procesos de una organización para el desarrollo de software como CMMI(Team, 2002a,b), ISO 15504ISO/IEC (2008), el segundo orientado a los procesos que un grupo de personas utiliza dentro de un proyecto particular TSP Humphrey (2000) y a nivel individual de un ingeniero de software que desempeña un rol dentro de un proyecto PSP Humphrey (2002).

2.2.3.1. ISO/IEC 15504

El ISO/IEC 15504 ISO/IEC (2008), también conocido como SPICE[1], abreviatura de la expresión en español, «Determinación de la Capacidad de Mejora del Proceso de Software» es un modelo para la mejora y evaluación de los procesos de desarrollo, mantenimiento de sistemas de información y productos de software.

Tiene una estructura basada en dos dimensiones: de proceso (el modelo de procesos que usa la organización) y de capacidad (o madurez) del proceso (niveles de capacidad o madurez y los atributos de los procesos) (ver figura 2.12). Los niveles de capacidad o madurez para cada uno de los procesos de desarrollo utilizados por una organización pueden tener una valoración desde 0 – Incompletos: representa que dicho proceso no es capaz de conseguir sus objetivos; hasta 5 – En optimización: representa que el proceso no solo es capaz de alcanzar sus objetivos sino que está continuamente mejorando (madurando).

Cada nivel de valoración de un proceso, se enfoca particularmente en unos atributos del proceso: Nivel 5 - en Optimización (Cambio de los procesos y Mejora continua), Nivel 4 – Predecible (Medición de los procesos y Control de los procesos), Nivel 3 – Establecido

[1]Software Process Improvement Capability Etermination, abreviado de SPICE.

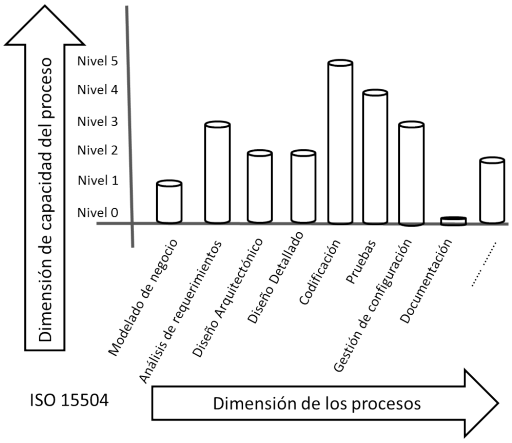

Figura 2.12: Dimensiones del modelo ISO 15504

(Definición de los procesos y Recursos de los procesos), Nivel 2 – Gestionado (Gestión del proceso y de los productos), Nivel 1 – Realizado(Ejecución del proceso). Es así como cada proceso madura al incrementar su capacidad de lograr el objetivo para el cual se definió dicho proceso, hasta lograr un nivel optimizado o mas bien un estado de mejoramiento continuo.

2.2.3.2. CMMI

El modelo Integrado de Madurez de Capacidades [1] CMMI[2] es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de software. Este modelo actualmente administrado por el Instituto CMMI, una subsidiaria de ISACA[3], se desarrolló en la Universidad Carnegie Mellon (CMU), a partir de varios modelos de madurez desarrollados por el SEI[4].

[1]La versión 1.3 se publicó en 2010 mientras que la 2.0 en 2018. CMMI está registrada en la Oficina de Patentes y Marcas de Estados Unidos por la Universidad Carnegie Mellon (CMU).

[2]Capability Maturity Model Integration.

[3]ISACA es el acrónimo de Information Systems Audit and Control Association, asociación internacional que patrocina y apoya el desarrollo de metodologías y certificaciones para actividades de auditoría y control en sistemas de información.

[4]CMMI integró los modelos (CMM-SW, SE-CMM, IPD-CMM, P-CMM, SA-CMM y S3M), CMM-SW: desarrollado inicialmente para los procesos relativos al desarrollo e implementación de software por la Universidad Carnegie-Mellon para el Software Engineering Institute (SEI). SE-CMM:El Modelo de Madurez de Capacidades en la Ingeniería de Sistemas fue publicado por el SEI en noviembre de 1995. Está dedicado a las actividades de ingeniería de sistemas, que en el contexto americano se diferencia

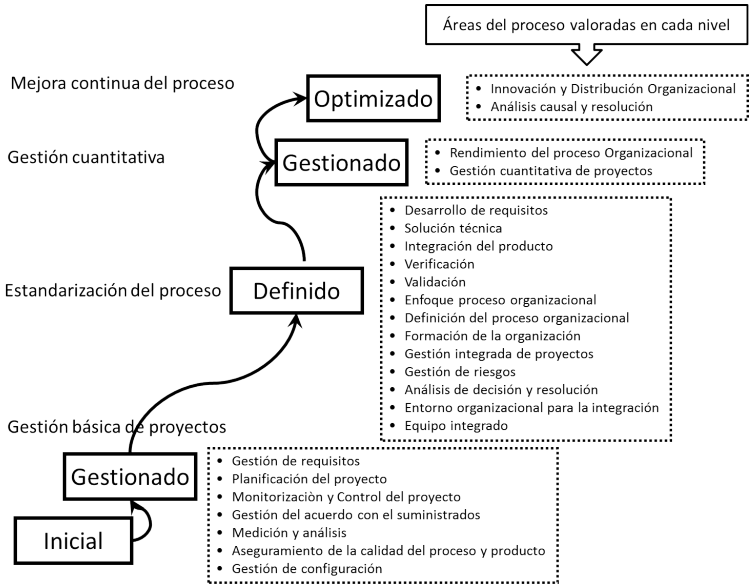

Figura 2.13: Modelo CMMI por etapas

El Modelo CMMI fue presentado inicialmente en dos modelos, uno por etapas(Team, 2002b) como lo muestra la figura2.13 y en el cual se valora el proceso global de una organización, buscando en cada nivel el énfasis en ciertas áreas del proceso. El otro modelo continuo (Team, 2002a) en la cual se valora el nivel de capacidad de manera separada cada uno de los procesos de acuerdo a la figura 2.14.

2.2.3.3. TSP

El Team Software Process TSP (Humphrey, 2000) ofrece una guía de trabajo para equipos de gerentes e ingenieros de software con el objetivo de mejorar los niveles de calidad y productividad de un equipo en un proyecto de desarrollo de software. El TSP se enfoca en la construcción de un equipo[1] desarrollador de software, estableciendo objetivos del equipo, distribuyendo los roles, y otras actividades de trabajo en equipo.

El TSP comienza con un proceso de cuatro días llamado lanzamiento o despegue

claramente de la ingeniería del software. IPD-CMM: El Modelo de Madurez de Capacidades para el Desarrollo Integrado de Productos propuesto como un borrador por el SEI en 1997. P-CMM: Modelo de Madurez de Capacidades para Recursos Humanos. SA-CMM: Modelo de Madurez de Capacidades para la Adquisición de Software. Y S3M: Modelo de Madurez de Capacidades para el mantenimiento del software.

[1]El TSP está diseñado para equipos de 2 a 20 miembros.

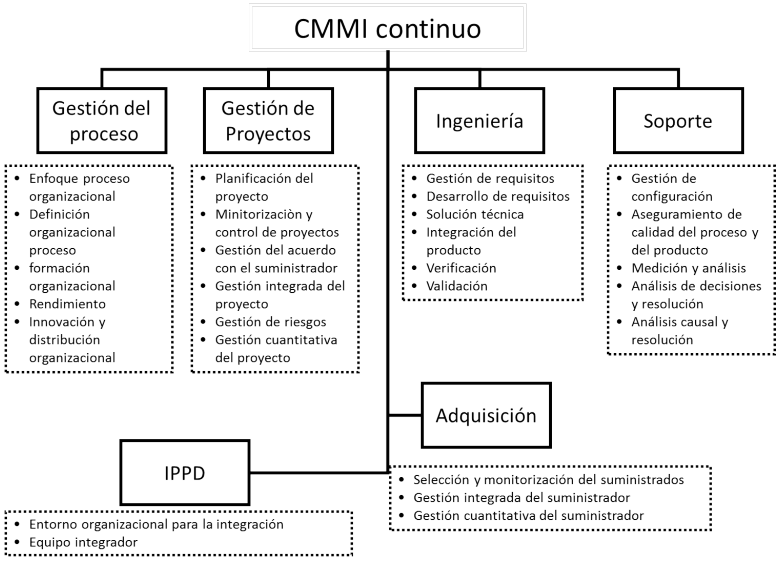

Figura 2.14: Modelo CMMI Continuo

del equipo para iniciar el proceso de construcción de los equipos y durante éste tiempo junto con sus directores establecen metas, definen roles, evalúan riesgos y producen un plan de equipo para su participación en un proyecto particular. Una vez que se lanza el equipo TSP, se enfoca en garantizar que todos los miembros del equipo sigan el plan. Esto incluye los siguientes temas: liderazgo del equipo, disciplina de proceso,seguimiento de problemas,comunicación,informes de gestión, mantenimiento del plan, estimación de la finalización del proyecto, re-equilibrar la carga de trabajo del equipo, relanzar el proyecto y gestión de calidad TSP.

Por otro lado TSP basa la gestión de calidad [1], en la gestión del defecto y para ello, los equipos deben establecer objetivos de calidad, definir y aplicar medidas de calidad, establecer planes para lograr estos objetivos, medir el progreso respecto a los planes y tomar medidas correctivas cuando los objetivos no se cumplen. Por lo tanto la gestión de calidad en TSP incluye principalmente el plan de calidad, la identificación de problemas de calidad y como encontrarlos y prevenirlos.

Dependiendo de la metodología de desarrollo utilizada en un proyecto particular, los equipos TSP se relanzan periódicamente[2] (ver figura 2.15), por lo que cada fase, ciclo o iteración se puede planificar en función del conocimiento y las lecciones aprendidas

[1]TSP Quality Management: la mayoría de las organizaciones comprenden que la calidad es importante, pero pocos equipos saben cómo gestionar la calidad de sus productos, pues no existen métodos generales para prevenir los defectos. Los ingenieros de software cometen errores, y estos errores son la fuente de defectos software. En el TSP, el principal énfasis de calidad está en la administración defecto.

[2]De acuerdo a lo iterativo o incremental del proceso de desarrollo utilizado.

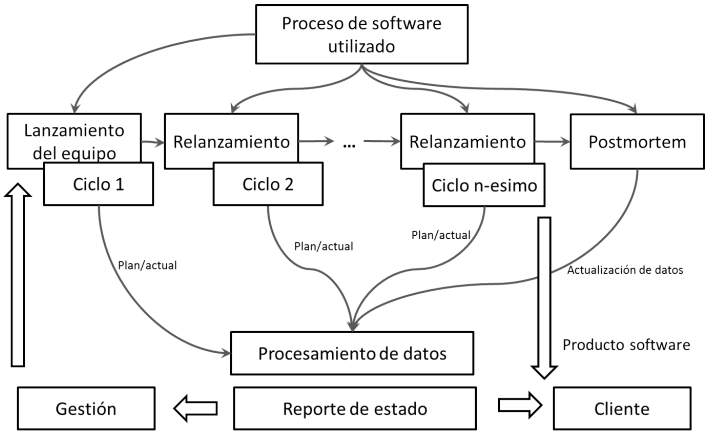

Figura 2.15: Flujo iterativo de TSP

adquiridas en el ciclo inmediatamente anterior. En el re-lanzamiento es necesario actualizar los planes detallados [1] de los ingenieros de software, y revisan el plan general del proyecto si es necesario.

2.2.3.4. PSP

El proceso personal de software(Humphrey, 2002), PSP [2], es un conjunto de prácticas para el manejo de tiempo y mejora de la productividad individual de los ingenieros, en tareas de desarrollo y mantenimiento de productos software, mediante el seguimiento de lo planificado frente al desempeño real. Esto permite que los desarrolladores mejoren sus habilidades de estimación y planificación, se fijen objetivos alcanzables, administren la calidad de los procesos individuales y como consecuencia de lo anterior se esperaría reducir la cantidad de defectos en sus productos.

Uno de los aspectos fundamentales de PSP es el uso de datos históricos[3] para analizar y mejorar el desempeño del proceso individual.

La recolección de datos para PSP es soportada por cuatro elementos importantes:

[1] Estos planes por lo general son precisos durante unos pocos meses.

[2] Personal Software Process.

[3] El termino histórico no se refiere a datos antiguos, de proyectos viejos, sino que hace referencia a datos de un ciclo o iteración previa de desarrollo.

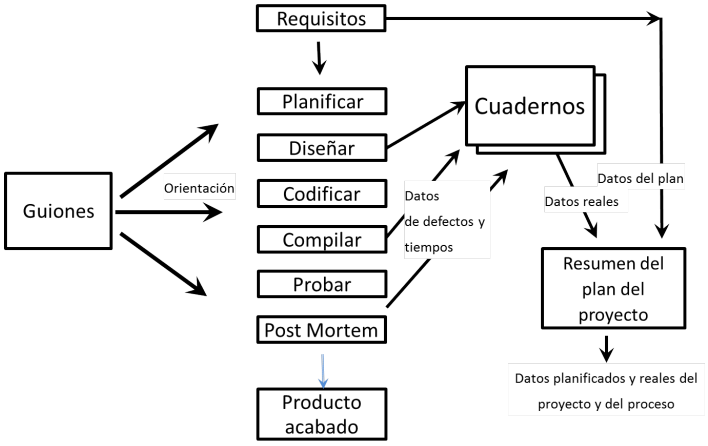

Figura 2.16: Flujo de trabajo con PSP

guiones[1], métricas (tamaño[2], esfuerzo[3], calidad[4] y agenda[5] principalmente), estándares[6] y formatos[7].

La calidad de un proceso personal, se valora en niveles dependiendo de las prácticas adoptadas por el ingeniero para su proceso de desarrollo como se muestra en la figura 2.17.

2.2.4. Metodologías de desarrollo de software

A diferencia de los modelos de ciclo de vida que solo definen las fases de un proyecto, las metodologías brindan detalles adicionales que dan respuesta al cómo, quién, cuándo y con qué se deben realizar cada una de las actividades definidas por un proceso de desarrollo software. En una metodología los modelos están implícitos en el proceso, mientras que cada metodología define sus propias prácticas y principios adecuadamente coordinando

[1]Los guiones de PSP describen el proceso personal en términos de pasos o actividades y sirven para definir las mediciones que se deben realizar.

[2]El tamaño típicamente medido en líneas de código (LOC) o artefactos de software equivalentes (proxies).

[3]El tiempo requerido en una tarea, típicamente en minutos dependiendo de la granularidad de las actividades.

[4]La cantidad de defectos encontrados en un artefacto o producto software.

[5]La medición de avance del proyecto, mediante la comparación de lo planificado contra lo realmente ocurrido y registrado.

[6]Estándares de codificación, técnicas y herramientas para pruebas, documentación, entre otros.

[7]Plantillas para el registro de datos de medición.

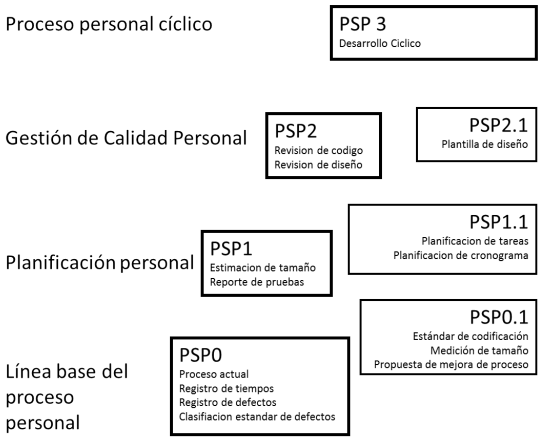

Figura 2.17: Niveles de Calidad del proceso Personal

mediante los conceptos adjuntos a cada una de las cuatro componentes de una metodología: Proceso, Personal, Producto y Tecnología, como se puede ver en la figura 2.18. En términos generalas existen tantas metodologías de desarrollo software como organizaciones, que ya sea por razón social o por necesidad interna tenga que desarrollar aplicaciones, pues cada cual adopta las que considere mejores prácticas y/o experiencias que se pueden encontrar en la literatura, por lo tanto no existe una metodología dominante incluso en las comunidades FLOSS. Es así como en este documento se referencias algunas metodologías ampliamente conocidas y que terminan sirviendo para adaptar un proceso de manera puntual a una organización específica.

A partir del año 2000 se acuño en el contexto de desarrollo software el concepto de método ágil, como una reacción a las formas tradicionales de producción software, admitiendo la necesidad de una alternativa al desarrollo orientado a la documentación y centrados en procesos complejos o pesados. Los métodos ágiles de desarrollo software nacieron con baja incredibilidad, pero con el tiempo han logrado una amplia aceptación, logrando la atención de los académicos y empresarios para su adopción como metodologías de desarrollo. Estas tendencias se conocen también con el nombre de métodos ligeros o livianos.

Algunas metodologías encajan dentro del concepto de ágil: programación extrema XP de Ken Keck, Crystal-family de Alistair Cockburn, desarrollo de software adaptativo ASD de Highsmith, Scrum de Schwaber y Beedle, desarrollo manejado por características FDD de Jeff De Luca y Peter Coad, método de desarrollo dinámico de sistemas DSDM, entre otros.

Estos métodos reconocen la importancia de los valores conocidos como “el manifiesto ágil” (Agile Manifesto, 2001): Los métodos ágiles valoran los individuos y su interacción más que los procesos y herramientas; el software funcionando (producto) más que la extensa documentación comprensiva; la colaboración del cliente más que la negociación

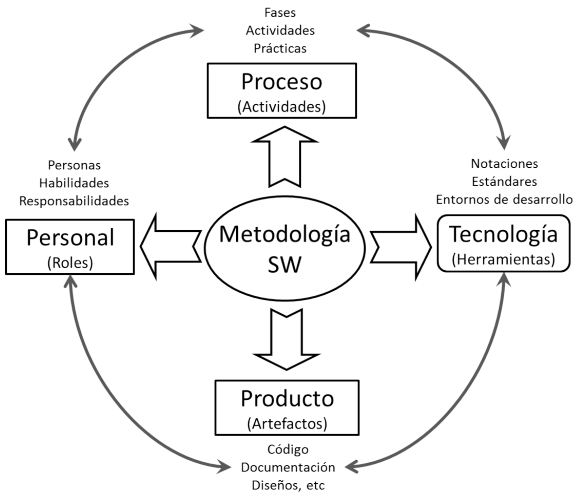

Figura 2.18: Componentes de una metodología de desarrollo de Software

contratada; responder al cambio más que seguir un plan estricto.

De igual forma en Sillitti and Succi (2005) se define la agilidad con dos caracterizaciones: Agilidad es la habilidad para crear y responder a cambios de acuerdo a los beneficios en un ambiente de negocios turbulento. Y la agilidad como habilidad para balancear flexibilidad y estabilidad. En este sentido la agilidad de los métodos de desarrollo software se puede sintetizar en la capacidad para responder al cambio y para simplificar los proceso de desarrollo.

2.2.4.1. Open Unified Process OpenUP

OpenUP (Cossentino et al., 2014) es una metodología de desarrollo software que aplica enfoques iterativos e incrementales, utiliza una filosofía ágil que se enfoca en la naturaleza de colaboración para el desarrollo de software. Está basada en RUP[1] (Rational Unified Process). Dentro de los principios metodológicos de OpenUP se encuentran:

- Colaboración entre los miembros del equipo para sincronizar intereses y compartir conocimiento, lo que promueve prácticas crean equipos en un ambiente saludable, facilitan la colaboración y principalmente permite el desarrollo de un conocimiento compartido del proyecto.

[1]El Proceso Unificado de Rational o RUP (por sus siglas en inglés de Rational Unified Process) es una metodología para el desarrollo software definida por la empresa Rational Software, actualmente propiedad de IBM, y se promovió junto con el Lenguaje Unificado de Modelado (UML), como los estándares más utilizados para el análisis, diseño, implementación y documentación de software orientados a objetos.

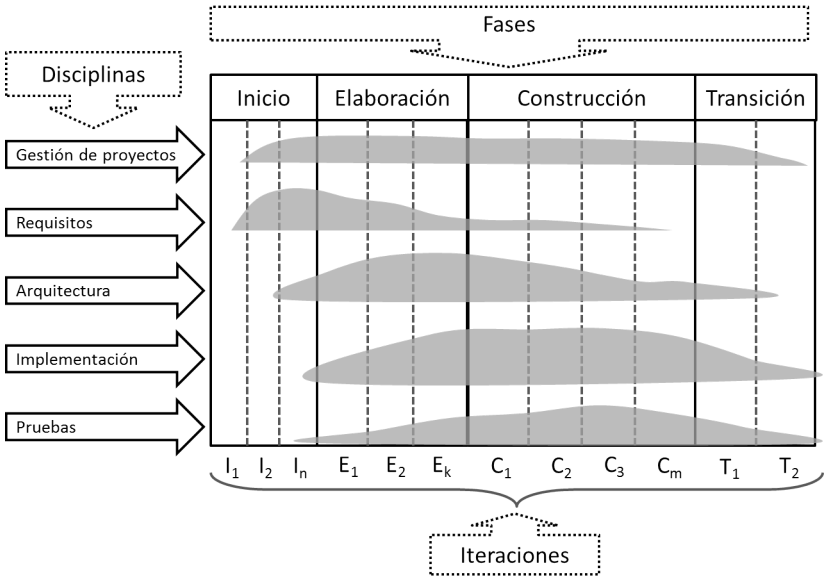

Figura 2.19: Proceso software con OpenUP

- Equilibrar las prioridades para maximizar el beneficio obtenido para lo usuarios del software, cumpliendo con los requisitos y restricciones del proyecto.
- Centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo.
- Desarrollo evolutivo con retroalimentación temprana y continua de los participantes del proyecto, permitiendo mostrar incrementos progresivos en la funcionalidad.

El proceso en OpenUP como se muestra en la figura 2.19, esta organizado en 4 fases y cada fase en iteraciones y cada iteración en flujos o disciplinas de trabajo en lo que se lleva a cabo las actividades propias del desarrollo.

Dentro de los roles en openUP se encuentran: Jefe de proyecto, Usuarios del software, Arquitecto, Analistas/desarrolladores, Diseñadores e ingenieros de pruebas.

2.2.4.2. SCRUM

Scrum (Schwaber Ken, 2017) es una metodología de dirección de proyectos[1] de desarrollo, más que una metodología de desarrollo software, por lo que no describe técnicas,

[1]Aplica un conjunto de buenas prácticas para descomponer el trabajo y colaborar en equipo para obtener el mejor resultado posible.

o herramientas para especificar, diseñar, codificar y probar software. Sin embargo define un proceso que particularmente se puede aplicar al desarrollo de productos software, basado en los siguientes principios:

1. El proceso de desarrollo es incremental, en lugar de una planificación detallada y completa del producto. Cada incremento se organiza en bloques temporales llamados Sprint que son cortos[1] y fijos [2].
2. La calidad de un producto depende la capacidad de un equipo[3] para trabajar de manera auto organizada. El equipo se reúne diariamente para sincronizar el trabajo y se realizan las adaptaciones necesarias, dando la autoridad necesaria al equipo para poder cumplir los requisitos.
3. Dar prioridad a lo que tiene más valor para el cliente.
4. Hacer entregas periódicas al cliente, de subproductos totalmente funcionales, que por medio de pruebas de aceptación se permita tomar las decisiones necesarias en relación a lo observado.
5. El espacio de trabajo debe permitir el encuentro de todos los miembros del equipo y su comunicación verbal entre ellos y con los involucrados en el proyecto.

Scrum dentro de su agilidad, define tan solo 3 roles considerados esenciales: Scrum Master[4], el Product Owner[5], y el Team (equipo)[6].

Como se observa en la figura 2.20, el proceso de Scrum inicia con la definición de un primer conjunto de características deseable denominado Product Backlog[7], según la prioridad dada a los elementos del Product Backlog y a su estimación de esfuerzo, se

[1] Entre 15 y 30 días.

[2] Cada Sprint tiene la misma duración durante todo el proyecto, en cada uno de estos periodos el equipo crea un incremento de software potencialmente utilizable por el cliente.

[3] Equipos pequeños entre 5 y 9 integrantes.

[4] No es un director de proyecto, este no es requerido dado que el equipo es auto dirigido, por lo tanto este rol tiene como responsabilidad facilitar la aplicación de la metodología Scrum, gestionar cambios y principalmente asistir al equipo de desarrollo.

[5] Representa a los stakeholders del proyecto (interesados externos o internos) y es el principal conocedor de las necesidades de los usuarios del software.

[6] Grupo de ingenieros que ejecuta el trabajo técnico propio de un desarrollo software y demás elementos relacionados con él.

[7] Product Backlog traduce Lista de retraso del producto, se refiere a las funcionalidades que no sean desarrollado en un momento determinado del proceso. Este listado esta constituido por un conjunto de requisitos de alto nivel priorizados que definen el trabajo a realizar.

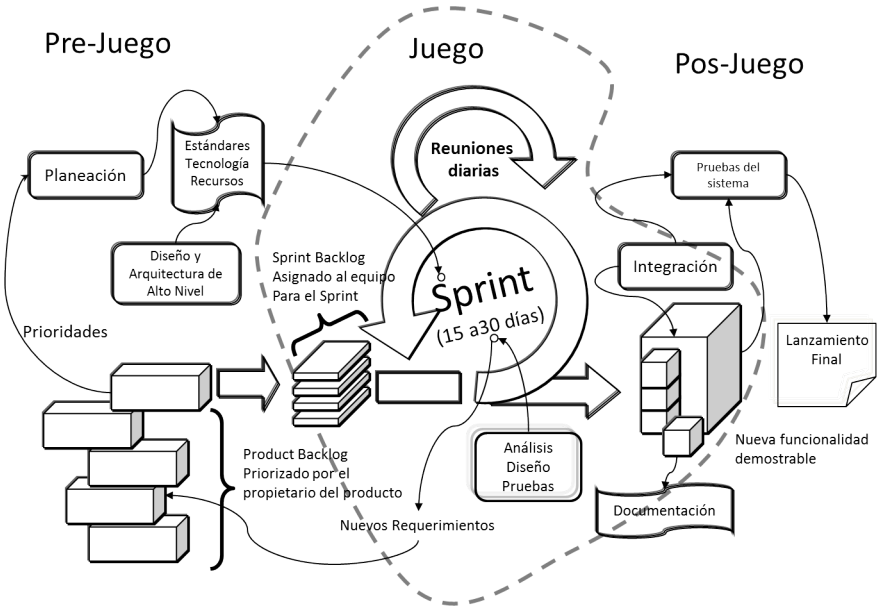

Figura 2.20: Proceso software con scrum

seleccionan los items que aproximadamente se pueden desarrollar en el periodo de tiempo definido para los Sprint [1] y esto se constituye en la meta del Sprint[2].

Este proceso se repite hasta lograr un product backlog vacío, lo que significa haber desarrollado a conformidad del cliente, todos y cada uno de los requisitos del producto software.

2.2.4.3. XP

Programación Extrema XP[3] es una metodología ágil y por lo tanto se centra en la comunicación efectiva y las relaciones interpersonales como clave para el éxito en un

[1]La cantidad de items que forman parte del Sprint se determinan durante la reunión de planificación del Sprint Planning, en la cual, el Product Owner identifica los elementos del Product Backlog que según su criterio desea ver terminados al final del Sprint y los da a conocer al equipo buscando la claridad y estimación adecuada de magnitud hasta conseguir consenso.

[2]Durante el Sprint no se puede modificar el Sprint Backlog lo que implica congelamiento de requisitos en ese periodo y por lo tanto, los nuevos requisitos o los incompletos al final de un Sprint, pasarán al Product Backlog, para que puedan ser tenidos en cuenta en un futuro Sprint.

[3]Del ingles Extreme Programming, nombre asignado pues sus principios y prácticas aunque son de sentido común, son llevadas al extremo.

proyecto de desarrollo software, se basa en realimentación continua entre el cliente y el equipo de desarrollo, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios[1].

Sus aspectos claves se resumen en el concepto de historia de usuario[2] y 14 prácticas altamente relacionadas como se muestra en la figura2.21: Entregas pequeñas[3] El juego de la planificación[4], Metáfora[5], Diseño simple[6], Pruebas[7], Refactorización[8], Programación en parejas[9], Propiedad colectiva del código[10], Integración continua[11], 40 horas por semana[12],

[1]Esta metodología es particularmente ideal para proyectos con requisitos imprecisos y poco estables, en lo cual encajan una buena parte de los proyectos de desarrollo software.

[2]Es una representación de un requisito escrito brevemente de una o dos frases y en lenguaje común del usuario, estas guían el proceso de desarrollo y deben ser útiles para el cliente o usuario, independientes unas de otras, negociables para llegar a una prueba de validación, estimable y verificable.

[3]Producir de manera frecuente versiones operativas del producto software a desarrollar. Una entrega no debería tardar más 3 meses.

[4]A partir de estimaciones el equipo planifica las entregas y cada una de sus iteraciones, teniendo en cuenta que los clientes incide sobre las prioridades el ámbito y tiempo de las entregas, pues esta practica se ve como un juego de dos equipos el de clientes o usuarios y el de desarrolladores. El equipo cliente define prioridades de cada historia de usuario, los programadores estiman esfuerzo, se ordenan las historias de usuario según prioridad y esfuerzo y se llega a un consenso en el contenido de la entrega y/o iteración.

[5]Una metáfora es una historia compartida claramente entendida tanto para el cliente como para los desarrolladores, que describe cómo debería funcionar el sistema. Estas sirven para como vocabulario comúnmente entendido para hablar sobre el dominio del producto software.

[6]Se diseña la solución más simple que pueda funcionar y ser implementada por lo que la complejidad innecesaria y el código extra debe ser removido inmediatamente.

[7]Primero se diseña la prueba unitaria y luego se implementa el código, y estas pruebas unitarias automatizadas son ejecutadas constantemente ante cada modificación del código.

[8]Refactoring, la constante re-estructuración para, mejorar su legibilidad, simplificarlo, remover duplicados y hacerlo más fácil de mantener.

[9]La codificación se realiza por dos personas trabajando en una sola máquina, esta práctica probablemente es la mas controvertida y por lo tanto ha generado muchas investigaciones que analizan su eficiencia y efectividad en la producción de software.

[10]Cualquier desarrollador puede cambiar cualquier parte del código en cualquier momento, siempre y cuando las modificaciones superen las pruebas unitarias automatizadas.

[11]Cada artefacto de código es integrado una vez que esté listo cuando todas las pruebas son ejecutadas y aprobadas para que el nuevo código sea incorporado definitivamente.

[12]No se trabajan horas extras, pues el desarrollo es considerado mas una actividad creativa que mecánica de fuerza.

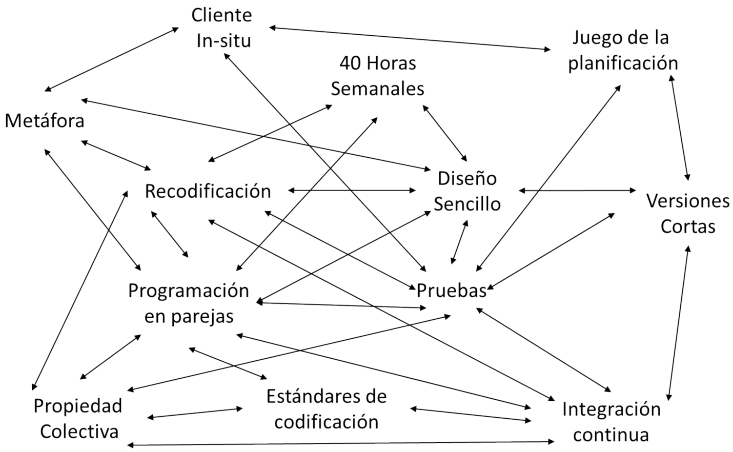

Figura 2.21: Practicas de XP

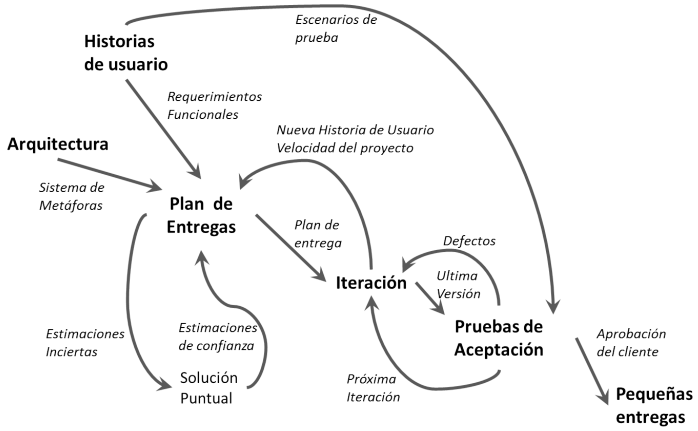

Figura 2.22: Proceso con XP

Cliente in-situ [1], Estándares de programación[2].

El proceso como se muestra en la figura 2.22 esta constituido por entregas planificadas según prioridad de las historias de usuario y cada una de estas entregas se dividen en iteraciones dentro de las cuales se realizan las típicas actividades técnicas del desarrollo y se aplican las prácticas.

[1]El cliente está presente y disponible para el equipo, pues la comunicación oral es más efectiva que la escrita, la cual toma mucho tiempo en generarse y puede ser mal interpretada.

[2]Es clave para la comunicación entre programadores y mantienen el código legible para todos, facilitando los cambios.

XP define como principales roles: Programador[1], Cliente[2], Gestor[3], Ingeniero de pruebas[4], Encargado de seguimiento[5], Entrenador[6], Consultor [7].

2.2.4.4. Otras metodologías

Si bien es cierto que XP y principalmente Scrum son las metodologías ágiles mas conocidas y probablemente mas usadas, también es importante reconocer que existe otras que encajan dentro de este movimiento.

- **Desarrollo orientado a funcionalidades FDD** (Feature-Driven Development) se caracteriza por la planificación y el diseño por adelantado en el cual, el modelo de objetos, la lista de características y la planificación se hacen al inicio del proyecto. Cada iteración produce un incremento que se enfoca en características o funcionalidades claramente identificadas. Su proceso está compuesto por cinco etapas: el desarrollo de un modelo general, la construcción de la lista de características, la planificación por característica, el diseño por característica y la construcción por característica, siendo estas dos ultimas fases las que se iteran hasta lograr el desarrollo de toda la lista de funcionalidades.
- **Crystal** no es una metodología puntual sino una familia de metodologías basadas en conceptos de Rational Unified Process (RUP), en ella cada metodología está identificada por un color dependiendo de dos variables, el número de personas

[1]Escribe pruebas unitarias y el código, comunicándose oportunamente con otros programadores y con el cliente si lo requiere.

[2]Un interlocutor que representa a los usuarios del software, escribe las historias de usuario y las pruebas funcionales para la aceptación de la implementación y asigna prioridad a las historias de usuario.

[3]Big boss, vínculo entre clientes y programadores, genera el ambiente adecuado para el trabajo de los programadores y su coordinación.

[4]Tester, quien ayuda al cliente a escribir las pruebas funcionales, ejecuta las pruebas frecuentemente sobre el código integrado y comunica los resultados al equipo.

[5]Tracker, proporciona reatroalimentación al equipo en el proceso XP, verificando el acierto de las estimaciones respecto al tiempo real dedicado en procura de mejorar futuras estimaciones, realiza el seguimiento al progreso en cada iteración y evalúa si los objetivos son alcanzables con las restricciones de tiempo y recursos disponibles en un momento dado.

[6]Coachs quien conoce el proceso XP para asistir a los miembros del equipo en la aplicación de las prácticas XP.

[7]Miembro externo del equipo con un conocimiento del dominio del producto software o algún tema específico en algún tema necesario para el proyecto.

involucradas en el desarrollo, y el grado de criticidad[1] del producto a desarrollar, siendo Crystal clear ideal para proyectos de baja criticidad y equipos de máximo 10 personas. Crystal están centrados en las personas y en la comunicación. Utiliza un proceso iterativo e incremental, en el cual ciclos donde se crean los incrementos no deben ser de mas de cuatro meses y se realizan reuniones de reflexión después de cada entrega para afinar la metodología.

- **Método de desarrollo de sistemas dinámicos DSDM**, (Dynamic Systems Development Method). Es iterativo e incremental, fragmentando el proyecto en periodos cortos de tiempo y comprometiendo explícitamente los entregables para cada uno de estos periodos.
- **Desarrollo adaptativo de software ASD** (Adaptative Software Development) Un proyecto software se considera un sistema adaptativo complejo, compuestos por agentes que son los interesados, entornos organizacional, tecnológico y como salidas el producto desarrollado. El ciclo de vida ASD está orientado al cambio y se compone de las fases: especulación, colaboración y aprendizaje.

2.2.5. Documentación de procesos

El concepto de proceso es ampliamente utilizado en diferentes ingenierías y no es un termino exclusivo para el desarrollo de software, por lo que existe un buen número de formas para documentar un proceso, es así como existen muchas alternativas para describir un proceso desde el texto en prosa pura hasta sofisticados diagramas propios de un contexto de proceso.

Entre notaciones, lenguajes y alternativas para la descripción de procesos se pueden encontrar las siguientes:

- **Diagramas de flujo.** Es la representación gráfica de un proceso mediante símbolos interconectados. Varias organizaciones han definido simbologías para la construcción de diagramas de flujo entre las que se encuentran: La American Society of Mechanical Engineers (ASME), La American National Standard Institute (ANSI), y quizás la más conocida de las notaciones definida por la International Organization for Standarization (ISO) para apoyar el aseguramiento de calidad a consumidores y clientes de acuerdo con las normas ISO-9000:2000
- **Business Process Modeling Notation (BPMN)** es una notación gráfica estandarizada que permite el modelado de procesos en una organización (negocio), en

[1] Esta medida hace referencia a las consecuencias de los errores dentro de un producto y va desde una escala de perdida de conformidad, hasta pérdida de vidas humanas.

un formato de flujo de trabajo (workflow), BPMN[1] proporciona una forma legible y entendible por parte de todos los miembros de una organización, como son los analistas de negocio que definen los procesos, los responsables de implementar los procesos y directivos gerentes y administradores de una organización, quienes hacen seguimiento a los procesos. BPMN no define simbología para modelar estructuras organizativas ni su descomposición funcional, tampoco para modelos de datos. Con BPMN se realizan diagramas simples que facilitan entender el flujo del proceso, mediante cuatro categorías básicas de elementos: Objetos de Flujo (Eventos, Actividades, rombos de control de flujo), Objetos de Conexión (Flujo de Secuencia, Flujo de Mensaje, Asociación); Carriles de nado[2](Piscina[3], Carril[4]), Artefactos (Objetos de Datos, Grupo, Anotación).

- **XML Process Definition Language (XPDL)** es un lenguaje para la definición de un Flujo de trabajo, creado por WfMC[5] para ser usado en el intercambio de modelos de procesos de negocio entre distintas herramientas. Es un formato de archivo XML que representa el "diagrama" en el cual sus nodos y las líneas de conexión tienen atributos que pueden especificar información como roles, descripción de actividades, temporizadores, llamadas a servicios web, entre otras.

- **JBPM Process Definition Language JPDL**JBPM (Java Business Process Model) BPM es una librería de java que actúa como motor Workflow que puede ejecutar los procesos de negocio descritos en BPMN 2.0 o en su propio lenguaje de definición jPDL.

- **ARchitecture of Integrated Information Systems (ARIS)** Arquitectura de Sistema de Información Integrada, es una herramienta de modelado empresarial que ofrece métodos para el diseño, gestión, flujo de trabajo, y análisis de sus procesos. Tiene un lenguaje de modelado, conocido como Event-driven Process Chains (EPC), que permite la ejecución de una secuencia de proceso disparados por eventos. En un modelo ARIS se dispone de cuatro perspectivas principales:Vista Organizacional (Organizational view), Vista de Datos (Data view), Vista de Control (Control view) y Vista Funcional (Functional view).

- **Integration DEFinition (IDEF)**es una familia de lenguajes de modelado en el campo de la Ingeniería de sistemas y la ingeniería de software que cubren una amplia gama de usos, desde el modelado funcional, simulación, análisis orientado a objetos hasta el diseño y adquisición de conocimientos.

[1] BPMN fue inicialmente desarrollada por la organización Business Process Management Initiative (BPMI) pero actualmente es soportada por el Object Management Group (OMG).

[2]Swimlanes, son un mecanismo para organizar y categorizar actividades dentro de un proceso.

[3]Representa los participantes de un subproceso y contiene uno o más carriles.

[4]Usado para organizar y categorizar las actividades dentro de una piscina de acuerdo a su función o rol y contiene objetos de flujo, objetos de conexión y artefactos.

[5]La Workflow Management Coalition (WfMC) es un consorcio industrial que define estándares para la interoperabilidad de sistemas de gestión de flujos de trabajo.

- **Unified Modeling Language (UML)** Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema software, incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes re-utilizables. Pese a que fue pensado para describir planos de software, algunos de sus diagramas como los modelos de negocio o los de actividades pueden ser utilizados en otros contextos como los procesos industriales o empresariales.

- **OPF Open Process Framework** es una propuesta de dominio público basada en la industria para la producción sistemática de metodologías de desarrollo. OPF se compone de un metamodelo de procesos[1], un conjunto de métodos re utilizables y uno de directrices para la creación de nuevas metodologías o adaptar las ya existentes.

- **Metamodel for Development Methodologies (SEMDM)** definido por el estándar ISO/IEC 24744[2] como una alternativa para el modelado de procesos de desarrollo software basado en una arquitectura de metamodelado de tres capas (Endeavour Domain, Methodology Domain y Metamodel Domain), en lugar de la más extendida arquitectura de cuatro capas propuesta por el consorcio Object Management Group (OMG). Incluye tres aspectos principales: el proceso a seguir, los productos utilizados y generados y las personas implicadas. El estándar ofrece un método de descripción centrado en los artefactos[3] (en lugar de centrarse en la práctica o en el proceso). El metamodelo también le permite describir por separado micro y macroprocesos utilizando etapas y unidades de trabajo.

- **Microsoft Solution Framework MSF** es el marco de trabajo que propone Microsoft para definir procesos software. El metamodelo de MSF se compone de una serie de principios fundamentales, un modelo de equipo y uno de fases e iteraciones soportando múltiples enfoques de procesos, los cuales pueden adaptarse a cualquier proyecto con independencia de su complejidad o tamaño.

- **El lenguaje SPEM Software Process Engineering Metamodel** SPEM [4] es un lenguaje diseñado para modelar procesos de Ingeniería del Software. Sin duda, se trata del lenguaje de definición de procesos software mas utilizado. Proporciona una serie de elementos para representar de forma estandarizada aspectos de los procesos desarrollo, incluyendo:Roles, Tareas, Artefactos, Lista de verificación,Productos de trabajo, Técnicas y herramientas, Estructuras de trabajo, Capacidad de rastreo y refinamiento, Ayuda sensible al contexto, guía o lineamientos y Descripción textual de elementos.

[1] Define los tipos fundamentales de componentes de métodos re-utilizables y cómo se relacionan entre sí.

[2] Software Engineering Metamodel for Development Methodologies.

[3] ISO 24744 ofrece describir la relación de productos y actividades con lo que es posible describir cómo los diferentes tipos de productos son utilizados y modificados por cada actividad.

[4] Especificación publicada en el 2002 por el consorcio OMG, organización ampliamente conocida por la especificación UML.

A pesar de la existencia de esta gran variedad de alternativas para la descripción de procesos de desarrollo software, es claro que SPEM se constituye en la primera opción para documentar los procesos en una comunidad FLOSS, pues esta notación nació precisamente para la ingeniería del software, además centra su atención en los roles y como estos realizan las actividades propias de su responsabilidad, generando artefactos mediante el uso de herramientas. Por otro lado es importante tener en cuenta que la vinculación a una comunidad FLOSS se realiza siempre por un rol, claramente definido en termino de habilidades en la ejecución de actividades propias del rol y en el manejo de herramientas apropiadas para las actividades asociadas al rol.

2.3. Despliegue de proceso

El despliegue de procesos software y el uso del proceso en despliegue, es un tema que enfatiza en las personas, es decir en como facilitar que los participantes en un proyecto, comprendan el proceso, lo apliquen adecuadamente y lo mejoren continuamente. Se trata de conseguir que los procesos correctos se desplieguen efectivamente en estructuras organizativas adecuadas para que las personas puedan responder mejor a las necesidades del negocio.(Forrester et al., 2006)

Es así como el punto de partida del despliegue de un proceso de desarrollo, es la definición documentada inicial del proceso, seguida por su adecuada socialización y comprensión por parte de los miembros de un proyecto, la puesta en marcha del proceso dentro de un proyecto y su mejoramiento continuo a través de cambios sugeridos por una adecuada evaluación.

Fernandez (2010) Afirma que a pesar de los esfuerzos realizados por las organizaciones, se presentan dificultades en el despliegue de sus procesos, debido a que en su gran mayoría, los esfuerzos realizados están más orientados hacia los aspectos técnicos, soslayando los aspectos relacionados con las personas y por lo tanto es importante identificar un conjunto de factores que condicionan el éxito del despliegue de los procesos nuevos o que han sido modificados.

2.4. Procesos de software en ambientes empresariales vs proceso en comunidades FLOSS

Es indispensable reconocer las diferencias entre los procesos de desarrollo en ambientes empresariales y los procesos de desarrollo en comunidades FLOSS, puesto que los primeros son formalmente definidos y controlados, mientras que los segundos son intuitivos y

auto-organizados, es así como para este trabajo se hace necesario reconocer como factor relevante esta diferencia, pues el despliegue de procesos se constituye en una actividad más centrada en la argumentación y convencimiento de los involucrados que en una capacitación sobre el uso del proceso.

En algunos estudios puntuales sobre el proceso de requerimientos en comunidades de opensource(Muruzábal and Castillo Acuña, 2014) se concluye que no existe un modelo a escala mundial que defina cómo debe ser el desarrollo del software, aun así se pueden observar características comunes en las comunidades. Habitualmente los participantes utilizan seudónimos para ser identificados, los desarrolladores utilizan su tiempo libre para desarrollar el código fuente y realizar aportaciones que implican el reconocimiento del resto de los participantes, y los usuarios utilizan foros de discusión, listas de correo electrónico, chats y wikis para observar, participar y contribuir en el desarrollo del proyecto (Escribano Luis and Martínez, 2011).

Los proyectos FLOSS se basan en una estructura jerárquica, donde normalmente hay un responsable, elegido por la comunidad, que toma las decisiones. Al contrario que en el software propietario los usuarios realizan una gran actividad en la fase de desarrollo, aunque pueden surgir problemas como cross-participations; es decir, un usuario que participa en discusiones del mismo tema en paralelo sobre diferentes listas de correo.(Barcellini et al., 2007)

En este comparativo de procesos es inevitable hacer referencia al ensayo ampliamente conocido en el contexto de la ingeniería del software, titulado: LA CATEDRAL Y EL BAZAR(Raymond, 1997) del autor Eric S. Raymond[1], en la cual se comparan metafóricamente dos formas[2] de obtener un producto software:

- **El modelo Catedral**[3]: en donde todo es estrictamente planeado como en la construcción de una catedral, los trabajadores son expertos artistas que tallan de manera individual cada una de las piedras, la cuales son instaladas solamente cuando pasan rigurosas pruebas de aceptación hasta que el producto final sale a la luz cuando esta totalmente terminado esperando que le sea útil y agradable a quienes necesitan de esta construcción.
- **El modelo Bazar**[4]: en donde se reúnen de manera auto organizada personas que tienen algunas cosas por ofrecer y otras que necesitan alguna cosa, con intereses,

[1]Se encuentran buenas traducciones al español como la realizada por José Soto Pérez.

[2]En este trabajo el modelo bazar representa la forma tradicional usada por las empresas para desarrollo de software, mientras que el modelo Bazar representa a las comunidades FLOSS. Pese a que hoy en día muchas organizaciones empresariales usan prácticas ágiles, que son propias del modelo Bazar.

[3]Se creía que un proyecto no se podía llevar a cabo sin un planeamiento central y un cuidadoso trabajo de concepción previa, en el que cada etapa estuviera meticulosamente planeada, se ajustara a un cronograma y se mantuviera unos grados de jerarquía.

[4]La ley principal es liberar lo más rápido posible versiones del software al alcance de toda la comunidad.

> actitudes y criterios tan variados que a simple vista puede parecer un caos, pero que al final gran parte de los allí congregados quedan satisfechos con la jornada ya sea porque pudieron vender o intercambiar algo, o porque encontraron lo que realmente necesitaba y sienten que el esfuerzo dedicado valió la pena.

En dicho ensayo se presenta la experiencia del autor en el liderazgo que tuvo en un proyecto de software y hace mucha referencia a linux como el proyecto de mayor envergadura y exitoso aplicando un modelo Bazar, sin embargo su aporte mas importante esta en un listado de 18 lecciones aprendidas que claramente representan las mejores prácticas o procesos aplicados en comunidades FLOSS:

1. Todo buen trabajo de software comienza a partir de las necesidades personales del programador[1]. En el contexto FLOSS una comunidad no comienza desde cero, siempre se cuenta con un producto software publicado por una sola persona que tiene un problema y ha escrito una solución en forma de código que funciona aunque esta incompleto, poco probado, con errores, probablemente mal diseñado, pero lo que si es cierto, es que dicha persona necesita del producto, usa lo que tiene, le interesa una mejor solución y conoce el dominio del problema. En el contexto empresarial el desarrollador solo atiende las necesidades de otras personas, por lo que no se garantiza un interés en el tema, mas allá de su remuneración salarial, probablemente tiene que empezar de cero, sin conocer exactamente el dominio del problema y en los casos que se disponga de un producto lo mas probable es que el desarrollador jamas lo haya usado.

2. Los buenos programadores saben qué escribir. Los mejores, qué reescribir (y reutilizar). En el contexto FLOSS el gran volumen de desarrolladores garantiza que alguno de ellos conozca algún código o haya desarrollado algo en otro contexto que pueda adaptar y mejorar para lo que necesita en su comunidad. En el contexto Empresarial los proyectos cerrados en algunas ocasiones prefieren escribir su código desde cero, principalmente por garantizar sus derechos de autor.

3. Considere desecharlo; de todos modos tendrá que hacerlo. En el contexto FLOSS el volumen de aportes hace que mucho código sea desechado ya sea porque una solución es mejor que otra o porque ha cambiado una necesidad y lo que se desarrollo anteriormente ya no es importante para momento en el que se encuentra un producto,

De esta forma, cada uno puede retomar un trabajo de su interés, hacer las mejoras que según su criterio sean necesarias, hacer las sugerencias que desee y agregar nuevas características si le parece; luego el resto de la comunidad juzgará y retomará, para seguir reformulando, su nueva versión. Así, los problemas pueden aislarse y solucionarse rápidamente. Ya que no hay instancias jerárquicas, cada individuo participa según sus propios intereses.

[1] todo buen trabajo empieza cuando uno tiene que rascarse su propia comezón, haciendo referencia a que un proyecto nace porque el desarrollador lo necesita realmente.

por lo tanto desechar código es habitual y no se ve como trabajo perdido. En el contexto empresarial algunas veces se ve como un detrimento patrimonial el desechar un código desarrollado dado que la empresa pagó por el tiempo que un desarrollador utilizó para escribir, probar y documentar dicho código.

4. Si tienes la actitud adecuada, encontrarás problemas interesantes. En el contexto FLOSS la colaboración es voluntaria, por lo que quienes se involucran siempre tendrán un interés en el producto que los mantiene activos y con ganas de aportar toda idea que se le ocurra sin importar que se acepte o no. En el contexto empresarial un desarrollador puede verse inhibido de expresar una idea por el miedo al rechazo.

5. Cuando pierdes el interés en un programa, debes darlo en herencia a un sucesor competente. En el contexto FLOSS si un líder de un proyecto pierde el interés por el tema que esta desarrollando probablemente no seguirá aportando y alguien mas lo seguirá alimentando. En el contexto empresariales los contratos con los desarrolladores hacen que sea poco probable abandonar el proyecto, por lo tanto preferirá continuar con una baja motivación.

6. Tratar a los usuarios como colaboradores es la forma más apropiada de mejorar el código, y la más efectiva de depurarlo. En el contexto FLOSS dada las frecuentes liberaciones de código, cada usuario se convierte en un ingeniero de prueba y si el volumen de uso es mayor, seguro los errores y nuevas necesidades van apareciendo con la misma velocidad que se usa. En el contexto empresarial las liberaciones son menos periódicas y solo se hacen después de procesos técnicos de pruebas realizados por ingenieros que no usaran el software y desconocen del dominio del problema, por lo tanto se desconoce la importancia de que sea, el mismo usuario quien identifique los errores y las nuevas necesidades a partir del uso del producto software.

7. Libere rápido y a menudo, y escuche a sus clientes. En el contexto FLOSS el cliente o usuario que por lo general también es desarrollador, esta pendiente de las liberaciones en búsqueda de nuevas funcionalidades que le sean útiles y por lo tanto encontrará, dependiendo de la dinámica de la comunidad, nuevo código para probar y retroalimentar el proyecto, ya sea con reporte de errores o pidiendo soluciones a sus necesidades. En el contexto empresarial se puede ver a los usuarios como delatores del equipo de desarrollo cuando encuentran errores y los reportan; por lo tanto toda liberación se hace con mucho cuidado para minimizar estos reportes, y se crea una barrera de comunicación entre usuarios y desarrolladores.

8. Dada una base suficiente de desarrolladores asistentes y beta-testers, casi cualquier problema puede ser caracterizado rápidamente, y su solución ser obvia al menos para alguien.[1] en el contexto FLOSS el volumen de participantes garantiza el éxito de un proyecto permitiendo que cada usuario, en un momento determinado, ademas de estar usando el producto, esta ejecutando cientos de casos de prueba y en caso de

[1]Dicho de otra manera menos formal, çon muchas miradas, todos los errores saltarán a la vista". A esto le conoce como la Ley de Linus.

encontrar una falla la reportará. En el contexto empresarial el número de casos de prueba que se le pueden aplicar a una funcionalidad, esta limitado por el número de ingenieros de pruebas y el tiempo que se disponga.

9. Las estructuras de datos inteligentes y el código burdo funcionan mucho mejor que en el caso inverso. En el contexto FLOSS tanto los diseños como el código van evolucionando pero no necesariamente creciendo, con el aporte de diferentes puntos de vista, esto hace que la misma dinámica del desarrollo llegue a los modelos de datos mas simples, mínimos pero suficientes para lograr lo deseado. En el contexto empresarial, debido al concepto cultural de que desechar es perder, las estructuras de datos crecen a partir de su concepción, y se evita simplificar por no contradecir las reglas de integridad de información o por ejemplo que al eliminar un campo de una tabla, hay que reescribir todo el código que utiliza dicha tabla, lo que se considera re-proceso, por lo tanto prefieren agregar otra entidad, aumentando la complejidad de los datos.

10. Si usted trata a sus analistas (beta-testers) como si fueran su recurso más valioso, ellos le responderán convirtiéndose en su recurso más valioso. En el contexto FLOSS cuando se es atendida una corrección de error reportada por un usuario o una solicitud de un nuevo requerimiento, éste usuario reconoce su importancia en el proyecto y se motiva a seguir buscando errores. En el contexto empresarial los desarrolladores e ingenieros de pruebas en algunas oportunidad son considerados rivales en un juego de perder la imagen de buen desarrollador por el reporte de un error, por lo tanto los ingenieros de prueba no han logrado ser reconocidos como uno de los roles mas importantes en el equipo.

11. Lo mejor después de tener buenas ideas es reconocer las buenas ideas de sus usuarios. Esto último es a veces lo mejor. En el contexto FLOSS no se descarta la posibilidad de encontrar mejores soluciones que las existentes hasta un determinado momento, pues la diversidad de puntos de vista que se presentan, hacen que comparar una solución con otra sea un tema de conveniencia para el proyecto mas allá del nombre de quien proviene la mejor solución. En el contexto empresarial se da poca oportunidad para poder disponer de más de una solución, por lo que normalmente la primera idea será la que se tenga como hilo conductor y por lo tanto no hay forma de apreciar otros puntos de vistas.

12. Con frecuencia, las soluciones más innovadoras y espectaculares provienen de comprender que la concepción del problema era errónea. En el contexto FLOSS la diversidad de puntos de vista hace que un problema tenga diversas interpretaciones y eventualmente una de estas se saldrá de contexto dejando en evidencia que el problema esta mal identificado. En el contexto empresarial la contratación de un proyecto define de manera inmodificable el problema y la especificación rigurosa solo contiene la mirada de quien realiza dicho proceso.

13. La perfección (en diseño) se alcanza no cuando ya no hay nada que agregar, sino cuando ya no hay nada que quitar. En el contexto FLOSS, debido a no tener

restricciones para desechar código cuando sea necesario, los diseños evolucionan de manera natural hacia una simplicidad, sin perder la suficiencia. En el contexto empresarial se prefiere agregar complejidad a los diseños frente a reescribir código por cambio en los diseños.

14. Toda herramienta es útil empleándose de la forma prevista, pero una *gran* herramienta es la que se presta a ser utilizada de la manera menos esperada. En el contexto FLOSS las herramientas puedan ser utilizadas incluso para contextos que a primera vista pueden parecer fuera de contexto, y puede ser allí donde se obtenga un gran beneficio para un proyecto particular. En el contexto empresarial se definen procesos estrictos en el sentido del uso de herramientas, por lo tanto no siempre se genera la posibilidad de pensar en usos diferentes para los cuales fue diseñada una herramienta.

15. Cuándo se escribe software para una puerta de enlace de cualquier tipo, hay que tomar la precaución de alterar el flujo de datos lo menos posible, y ¡*nunca* eliminar información a menos que los receptores obliguen a hacerlo!

16. Cuando su lenguaje está lejos de un Turing completo, entonces el azúcar sintáctico puede ser su amigo. En los contextos FLOSS los estándares de programación evolucionan a partir de la terminología propia de los participantes conocedores del dominio del problema y por lo tanto entendibles por todos, lo que hace un código mas legible. En el contexto empresarial por lo general el estándar de programación se limita a la forma de llamar las variables, métodos, librerías y otros aspectos dentro de un lenguaje de programación.

17. Un sistema de seguridad es tan seguro como secreto. Cuídese de los secretos a medias.En el contexto FLOSS dado que la totalidad de código está disponible, la seguridad no depende de lo oculto que este el algoritmo de cifrado. En el contexto empresarial se confía la seguridad de un software a la capacidad de ocultar los algoritmos de cifrado, pues el código solo es visible para los miembros del equipo y es probable que los aspectos de seguridad solo sean confiados a un solo desarrollador.

18. Para resolver un problema interesante, comience por encontrar un problema que le resulte interesante.

19. Si el coordinador de desarrollo tiene un medio al menos tan bueno como lo es Internet, y sabe dirigir sin coerción, muchas cabezas serán, inevitablemente, mejor que una. En el contexto FLOSS las habilidades para liderar un trabajo voluntario pero autorganizado se constituyen en el motivo mas importante para lograr la simpatía por un proyecto. En el contexto empresarial por lo general los directores de proyecto logran su estatus por su capacidad de cumplir metas en tiempos limitados por lo que debe planificar detalladamente e imponer a los desarrolladores metas puntuales para el logro de los objetivos del proyecto.

Esta comparación puede verse como una critica a los procesos de desarrollo software de los contextos empresariales, sin embargo su intención no es esta, tan solo es diferenciar

los contextos FLOSS como representantes más cercanos al modelo Bazar y los contextos empresariales como representantes más cercanos al Modelo Catedral.

Por otro lado es importante resaltar que pocos años después de la publicación de este ensayo, surgió el movimiento de los métodos ágiles de desarrollo software (Agile Manifesto, 2001), que han influenciado de manera significativa el contexto empresarial, por lo tanto hoy 20 años después, las empresas adoptan practicas de métodos ágiles, como la incorporación de los usuarios o clientes al proceso de desarrollo, los procesos de trabajo auto-organizados, propiedad colectiva del código entre los miembros del equipo de desarrollo, estándares de programación, re-factorización de código, entre muchas otras practicas propias del modelo Bazar.

Finalmente en esta sección se muestra (ver figura 2.23)en una linea de tiempo la evolución de los principales aportes en los procesos de software en la parte inferior de la figura y la evolución del movimiento FLOSS representado por los proyectos más exitosos que se han dado en este contexto.

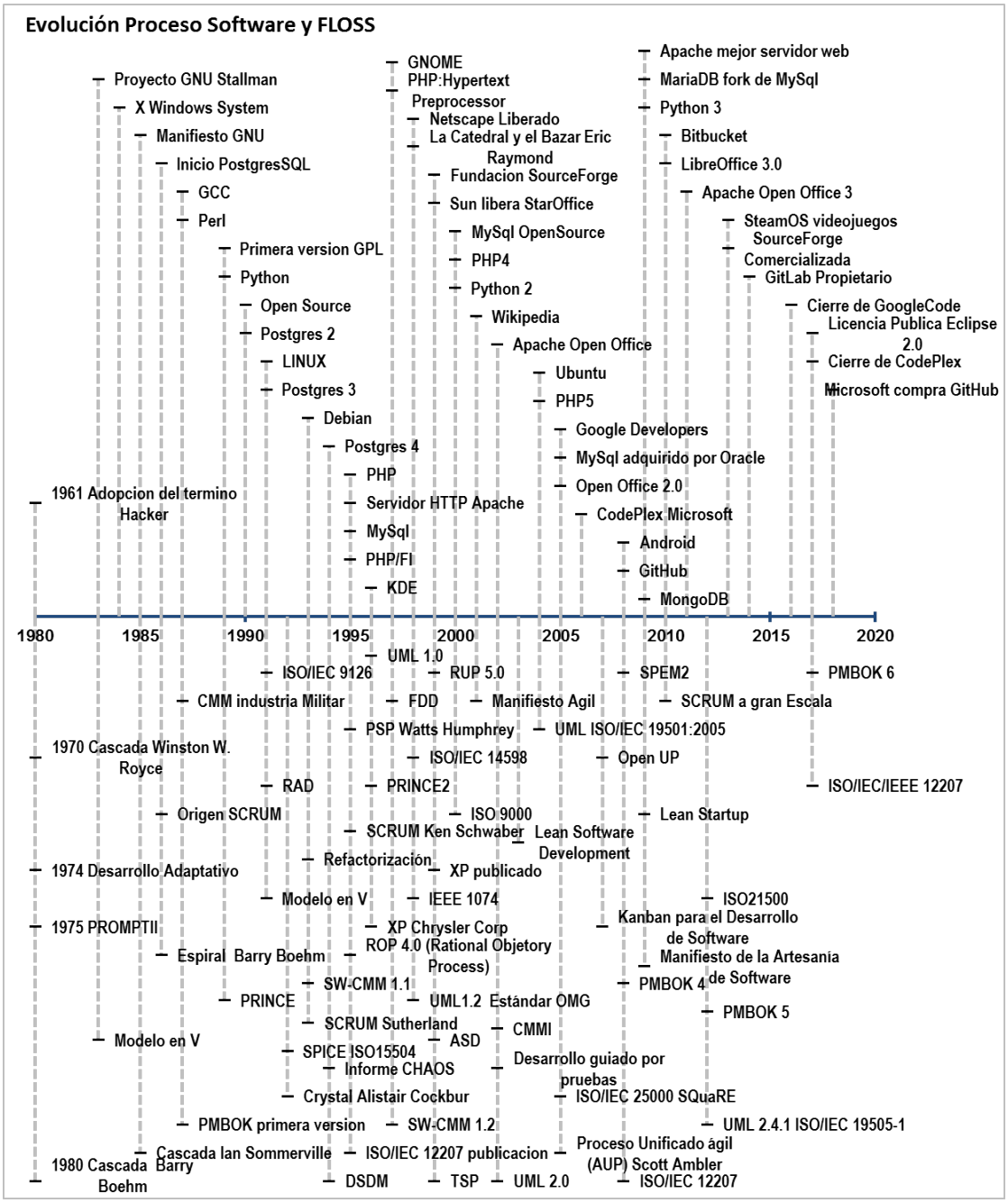

Figura 2.23: Evolución del proceso de software y el Movimiento FLOSS

2.5. Estado del Arte

Para la construcción de este estado del arte se realizó una búsqueda de artículos y otros documentos en bases de datos como sciencedirect, con el criterio de búsqueda «open source software process», contenido ya sea en el título, resumen o palabras claves del documento. aunque esta consulta generó cientos de resultados, se realizó un primer filtro por el titulo, seleccionando aproximadamente 120 artículos, de los cuales se analizó el abstrac y quedaron 96 descartando aquellos que no correspondían al objeto de estudio aquí definido[1]. Usando como herramienta mendeley se organizaron dichos artículos en las siguientes categorías como muestra la tabla 2.1.

Tema	Total	Doc Completo	Solo Abstrac
Proceso de Software en general	13	10	3
Análisis de datos FLOSS	30	15	15
Despliegue de Procesos	1	1	0
Generalidades FLOSS	25	16	9
Procesos FLOSS	25	16	9

Cuadro 2.1: Documentos consultados para el estado del arte

De manera gráfica (figura 2.24) se puede ver la distribución de los documentos en las cinco áreas de la clasificación.

[1]Se consiguieron 63 de estos documentos con su contenido completo y 33 de ellos solo se cuenta con el abstrac, por lo que se consideró más que suficiente para analizar el estado de investigación relacionado con el objeto de estudio. Por otro lado es importante mencionar que no se dio lectura detallada de todo el contenido de los documentos, fue suficiente con el abstrac y en algunos casos con las conclusiones.

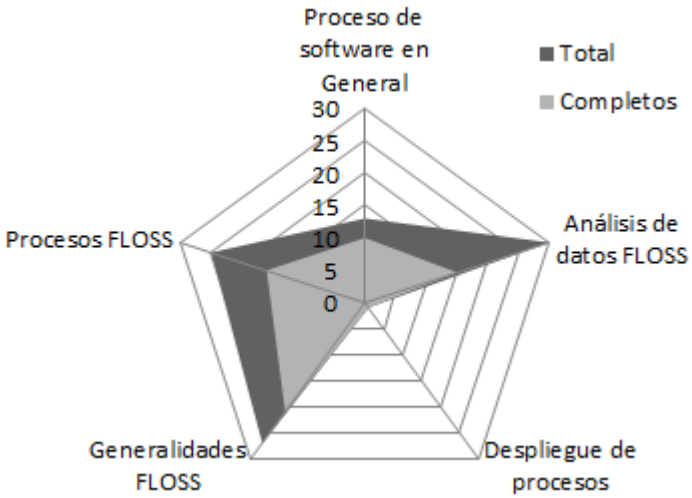

Figura 2.24: Distribución de los documentos analizados

2.5.1. Proceso de software en general

En esta temática se encontraron como principales tendencias de las investigaciones el modelado de los procesos y en la integración de herramientas. Los principales trabajos consultados al respecto son:

Como documento base para la formulación del presente trabajo, Forrester et al. (2006) estructura los principales temas de investigación relacionados con el proceso de software Fuggetta (2000) presenta una breve historia y los logros de la investigación de procesos software, alguna evaluación crítica de los resultados producidos hasta ese momento. Acuña et al. (2000) hace una revisión de los trabajos relacionados con la variedad de modelado de procesos, incluyendo la representación de los elementos de modelado. Canfora and Ruiz González (2004) realiza un estudio sobre la integración de procesos de producción y de gestión en los proyectos software, presentado algunas herramientas integradas, conocidas como Entornos de Ingeniería del Software (EIS), cuyo objetivo es dar soporte a los citados procesos, dejando como problema abierto la integración de las herramientas en solo entorno orientado a procesos. Matturro Mazoni and Silva Vázquez (2010) propone una forma de gestionar los conocimientos y experiencias adquiridas durante un proyecto de software, como el activo mas valioso para las organizaciones en la mejora de procesos. Bermón-Angarita (2010) propone el uso de una wiki para la gestión de las librerías de activos de proceso PAL[1]. Rolandsson et al. (2011) presenta un estudio de cómo los programadores hacen frente a la coexistencia de un modo de producción industrial

[1]Las PAL Process Asset Library son repositorios de documentos con información útil para el personal que está definiendo, implementando, gestionando y ejecutando procesos en las organizaciones.

comercial (propio de las empresas) y el comunitario cooperativo propio del contexto FLOSS. Ruiz Rube and Dodero Beardo (2013) propone un marco de trabajo basado en la aplicación de las técnicas de la Ingeniería del Software dirigida por modelos[1] y de la integración de información mediante datos abiertos enlazados[2]. Ruiz-Rube and Dodero Beardo (2014) afirma que pese a que SPEM nació para la documentación de procesos software, ha sido poco utilizado por sus falencias respecto a complejidad del lenguaje, a ciertas carencias existentes en aspectos como la gestión de la variabilidad de los procesos y su ejecutabilidad, y la falta de mecanismos para la automatización del despliegue sobre herramientas de soporte, para lo cual propone un marco de trabajo para el despliegue y evaluación de procesos software. Alvertis et al. (2016) propone una metodología para cerrar la brecha entre clientes y desarrolladores como aspecto clave del éxito de un proyecto de desarrollo, sugiriendo la plataforma CloudTeams como un sistema de groupware que apoya la noción de desarrollo de software colaborativo. Linåker et al. (2018) propone un modelo de Proceso de aceptación de contribución (CAP) a partir del cual las empresas pueden adoptar estrategias de contribución que se alineen con las estrategias y la planificación del producto. Alshakhouri et al. (2018) presenta forma de visualizar un proceso software situando artefactos de código, junto con sus procesos de desarrollo, con el fin de vincular y sincronizar estos componentes típicamente separados. Ewenike et al. (2018) revisa el proceso de desarrollo, con el fin de evaluar mejor los factores y las brechas que crean la necesidad de mejorar el proceso de desarrollo de software colaborativo en la nube.

2.5.2. Análisis de datos FLOSS

En estas temática se clasificaron los artículos que dentro de sus procesos metodológicos utilizaron datos de los repositorios donde se encuentran los proyectos FLOSS o de otras fuentes como foros[3], metarepositorios[4]. Entre las tendencias encontradas en estos documentos se encuentra la búsqueda de componentes FLOSS, corrección de errores y sus tiempos, re-utilización de código, comportamiento y caracterización de los miembros de una comunidad y sobre liberación de versiones. Entre los aportes mas relevantes a este tema se encontraron:

[1] Utilizando estas, se consigue la adaptación semi-automática de las herramientas de soporte mediante la transformación sucesiva de modelos, partiendo desde el modelo de procesos.

[2] Con los datos abiertos enlazados, se consigue que las herramientas expongan de manera controlada la información que gestionan, para así facilitar la construcción de soluciones de integración destinadas a la evaluación de los procesos.

[3] Stack Overflow.

[4] FLOSSmole y FLOSSMetrics.

Gerea et al. (2007) propone un método para la selección de componentes OST (Off-the-Shelf) [1] de código abierto, brindando indicaciones para responder a ¿Dónde y cómo encontrar componentes de OSS? ¿Cómo evaluarlos? ¿Cómo aprender los componentes? ¿Cómo guardar el conocimiento sobre el componente elegido? ¿Cómo hacer un mejor uso de OSS? ¿Qué versión del componente se debe elegir y qué se puede hacer con respecto al mantenimiento y las nuevas versiones del componente?. Wu et al. (2007) analiza la supervivencia de los proyectos FLOSS utilizando el análisis de redes sociales, con base a datos empíricos recopilados de repositorios en línea se estudia el impacto de los patrones de comunicación de los equipos de desarrollo, en los resultados de estos proyectos teniendo en cuenta las características específicas del proyecto. Rigby et al. (2008) estudia datos del proyecto servidor apache, respecto a sus procesos de revisión por pares, llegando a la conclusión de que son revisiones tempranas y frecuentes, de contribuciones pequeñas, independientes y completas, realizadas de forma asíncrona por un grupo potencialmente grande, pero en realidad pequeño, de expertos autoseleccionados y en general que este proceso es eficiente y efectivo. Gonzalez-Barahona et al. (2010) describe dos de las meta repositorios sobre el desarrollo de software: FLOSSmole y FLOSSMetrics, proporcionando una idea de cómo estos meta-repositorios (a veces llamados repositorios de repositorios", RoR) de datos sobre proyectos de código abierto deben usarse para ayudar a los investigadores. Grottke et al. (2010) investiga la asociación entre los factores del equipo y la eficiencia del procesamiento de fallas para las versiones de software de fuente cerrada de un gran proveedor de software comercial y para los proyectos de software de código abierto registrados en SourceForge.net, encontrando relación entre la experiencia del equipo y la eficiencia del procesamiento de fallas. Sojer and Henkel (2010) hace un análisis multivariado del comportamiento de la re-utilización de código encontrando que un paradigma de desarrollo que exige la liberación anticipada de una versión de funcionamiento inicial del software, como la "promesa creíbleçonduce a una mayor re-utilización. Ghapanchi et al. (2012) examina el impacto de la efectividad en la corrección de defectos por el interés del usuario en contribuir a los proyectos de FLOSS. Robles and González-Barahona (2012)Afirma que el contexto FLOSS las bifurcaciones de un proyecto son inherente a las cuatro libertades y con datos de algunos proyectos analizó las fecha en que ocurrió la bifurcación, la razón y el resultado de la bifurcación, es decir, si el proyecto original o el proyecto de bifurcación aún están desarrollados. Feitelson (2012) analizan datos de evolución del kernel de Linux para la formulación de un modelo denominado desarrollo perpetuo[2], integrando el progreso en el tiempo con el crecimiento de la base de código y diferenciando entre el desarrollo de nuevas funcionalidades y el mantenimiento de versiones de producción. Jongyindee et al. (2012) analiza los Committ

[1] Existen varios métodos técnicos para seleccionar y evaluar los componentes listos para usar OTS (Off-the-Shelf), pero la investigación muestra que estos métodos rara vez se usan. Los componentes OTS vienen en dos tipos: COTS (Comercial-Off-The-Shelf) y OSS (Open-Source-Software).

[2] El modelo de desarrollo perpetuo se utiliza como un marco en el que se pueden demostrar los beneficios comúnmente reconocidos del desarrollo incremental y evolutivo, y para criticar cuestiones como la arquitectura, la conservación de la familiaridad y los proyectos fallidos.

en el proyecto Eclipse-Platform para evaluar su calidad y encontrar las consecuencias de los errores dentro de los Commit en un sistema de control de versiones. Midha and Prashant (2012) aplicó dos medidas de éxito del proyecto: popularidad del proyecto y actividad del desarrollador, en 283 proyectos OSS en un lapso de 3 años, para observar los cambios a lo largo del tiempo. Farah and Correal (2013) presenta una herramienta para apoyar el análisis y la selección de componentes y aplicaciones de software en repositorios públicos, particularmente GitHub. Aksulu and Wade (2013) investiga el comportamiento de los desarrolladores de código abierto y la estructura del desarrollo de proyectos a través del análisis de un conjunto de datos muy grande: 10 proyectos de software conocidos y ampliamente utilizados Ma et al. (2013) muestran que el software de código abierto es desarrollado tanto por voluntarios, como por empresas comerciales[1], a menudo conjuntamente; concluyendo que con una mayor participación comercial, la mayoría de reportes de problemas correrían a cargo de desarrolladores comerciales, y se reduciría el tiempo de resolución de problemas, lo que implica una mejor experiencia de usuario. Kumar (2015) muestra cómo el desarrollo de código abierto se puede representar como un grafo de red de colaboración y cómo la red se puede caracterizar por varias métricas de estructura de red, incluyendo cuatro métricas: tamaño, centralización, densidad y agrupamiento. Agrawal et al. (2015) realiza un estudio de los Commit en proyectos de computacion de alto rendimiento HPC (High Performance Computing) con el uso de herramientas y repositorios con datos históricos de los proyectos de SVN, Mercurial y Git, concluyendo la existencia de una calidad de compromiso relativamente alta pero decreciente en proyectos de HPC. AlMarzouq et al. (2015) desarrolla un modelo que explica cómo la capacidad de una estructura para administrar la información afecta el desempeño de la comunidad FLOSS, encontrando[2] que el desempeño de las comunidades centralizadas y descentralizadas variaba con tres condiciones vinculadas a los flujos de información: rutina de tareas, incertidumbre e interdependencia entre tareas. Farias et al. (2015) analiza la información de dos repositorios del proyecto Apache Httpd con información sobre el comportamiento de sus desarrolladores, concluyendo que el uso de minería de datos y software de visualización son importantes de los procesos de desarrollo. Gamalielsson et al. (2015) analiza datos del proyecto Drupal de los cuales se concluye que los estándares ampliamente implementados pueden beneficiarse de las contribuciones proporcionadas por una variedad de individuos, organizaciones y tipos de organizaciones diferentes, ya sea directamente a un proyecto de estandarización o indirectamente a través de un proyecto de código abierto que implementa el estándar. Zaimi et al. (2015) investiga los procesos de re-utilización en cinco proyectos

[1]Las empresas se involucran en proyectos de código abierto por razones comerciales y traen consigo un proceso de desarrollo de software comercial.

[2]Con base en datos de archivo extraídos de 237 comunidades FLOSS activas, los resultados empíricos apoyan la idea de que las comunidades FLOSS que realizan tareas de desarrollo que son generalmente rutinarias, altamente interdependientes y generan poca incertidumbre del contribuyente funcionarán mejor bajo una estructura de compromiso centralizada. Por otro lado, las estructuras de compromiso descentralizadas prosperan bajo las condiciones de la tarea no rutinaria , la baja interdependencia de la tarea y la alta incertidumbre del contribuyente.

FLOSS, con respecto al grado en que la funcionalidad del software se construye desde cero o se reutiliza, la frecuencia con la que se modifican las decisiones de re-utilización, y el efecto en la calidad del producto de software. Jurado and Rodriguez (2015) hace un análisis de datos en los aportes de los desarrolladores a los proyecto FLOOS, concluyendo que estos dejan sentimientos subyacentes en el texto, y esa información podría ser monitoreada como cualquier otra característica en el proceso de desarrollo. Sahu et al. (2016) hace un análisis cruzado de repositorios para cinco proyectos FLOSS respecto al número de errores y tiempo promedio de corrección de errores mostrando que el tiempo de corrección de errores, puede reducirse publicando los errores en el repositorio de Stack Overflow. Filippova and Cho (2016) investiga diferentes tipos de conflictos en los equipos de desarrollo FLOSS, sus antecedentes e impacto en la participación sostenida de los desarrolladores. Encuesta a 222 desarrolladores de FOSS, encontrando que el conflicto tiene un efecto negativo general en la retención del desarrollador y que diferentes tipos de conflicto tienen un impacto variable en los resultados. Daniel and Stewart (2016) explora cómo los factores que facilitan la integración del conocimiento[1] impactan el éxito del proyecto y cómo la atención de los desarrolladores a proyectos externos puede frenar el éxito de un proyecto focal. Poo-Caamaño et al. (2017) a través de un estudio de caso, concluye que un calendario de publicación, la influencia (en lugar de control directo) y la diversidad son los principales factores[2] que impactan positivamente en el proceso de lanzamiento en el ecosistema GNOME. Wei et al. (2017) estudia cómo los roles núcleo-periferia están relacionados con el comportamiento social de cortesía, con los datos de dos proyectos de FLOSS, sugiere que tanto los miembros centrales como los periféricos usan más estrategias de cortesía positiva que estrategias negativas. Joia and dos Santos Vinhais (2017) analiza la transición de software de código cerrado a código abierto en una empresa privada, usando como caso de estudio la transición en el uso de Microsoft Office con Open Office. Shimagaki et al. (2017) realiza un análisis sobre dos proyectos propietarios y cuatro proyectos FLOSS para comprender el por qué se revierten commits identificando 13 causas comunes para ello. Daniel et al. (2018) analiza 329 proyectos SourceForge, respecto a las diferencias de los participantes (lenguaje, función y contribución) y las diferencias del proyecto (entorno de desarrollo y conexión), demostrando que la diferencia del entorno de desarrollo y la conexión reduce el impacto positivo de la diversidad de roles y contribuciones, tiene una relación con el éxito de un proyecto. Li and Zhou (2018) concluye que la conexión estructural entre las comunidades de OSS a través de la superposición de membresía, está asociada con la similitud entre sus procesos de desarrollo.

[1] Bajo acoplamiento de software y alta discusión interactiva.

[2] Otras conclusiones asegurar que el equipo de publicación siga los principales canales de comunicación utilizados por los desarrolladores, (2) proporcionar un lugar para la coordinación de un ecosistema, (3) considerar incluir buenas habilidades técnicas y sociales en un equipo de liberación, (4) apuntar a un equipo de liberación diverso, (5) basado en falta de poder, cabildeo y gestión basada en consenso deben seguirse , (6) ayudar al equipo de lanzamiento en el proceso de coordinación con un cronograma bien definido, y (7) el trabajo del equipo de liberación es diferente del trabajo de software regular.

2.5.3. Procesos FLOSS

En esta sección la tendencia de los aportes se encuentran en la comprensión de procesos como requerimientos, diseño, pruebas, organización y control de participación, técnicas de revisión por pares y liberación de versiones en comunidades FLOSS. Entre los principales aportes se puede citar:

Pese a que este documento se escribió en 1997, sigue siendo una de las metáforas que mejor describe los procesos FLOSS. Raymond (1997) afirma que los procesos en el desarrollo software de mayor envergadura[1] requería construirse como las catedrales, es decir, que debía ser cuidadosamente planeado y elaborado por genios o pequeñas bandas de magos trabajando encerrados a piedra y lodo, sin liberar versiones beta antes de tiempo. Al contrario, la comunidad Linux se asemejaba más a un bullicioso bazar de Babel, colmado de individuos con propósitos y enfoques dispares[2], de donde surgiría un sistema estable y coherente únicamente a partir de una serie de artilugios. Potdar and Chang (2004) muestra un estudio comparativo de los procesos de código abierto y enfoques de desarrollo de código cerrado. Barcellini et al. (2007) indaga sobre los roles emergentes y las formas de participación en el proceso de diseño FLOSS. La metodología propuesta, articula análisis estructurales de las discusiones (organización de las discusiones, participación), a las acciones del código y la documentación realizada por los participantes al proyecto. Spinellis et al. (2009) presenta herramientas y técnicas que se pueden utilizar para evaluar la calidad del producto FLOSS, a partir del código fuente y los datos asociados almacenados en el sistema de control de versiones, las bases de datos de seguimiento a los errores, las listas de correo y las wikis que permiten evaluar la calidad de manera transparente. Huysmans et al. (2010) propone una metodología para desarrollar modelos de proceso FLOSS, analizando los procesos a nivel ontológico y proporcionando descripciones de procesos de alto nivel y estudiando los patrones de comunicación entre los actores humanos, en lugar de las secuencias en las que se realizan las actividades. Chung et al. (2010) concluye que aunque los contribuyentes entienden las ventajas de usar diagramas para actividades relacionadas con el diseño, los diagramas se utilizan con poca frecuencia en OSS. y propone algunas ideas para apoyar actividades de diseño en proyectos de OSS. Escribano Luis and Martínez (2011) estudia cómo se llevan a cabo los procesos y actividades relacionadas con la Ingeniería de Requisitos en las comunidades FLOSS. Xi et al. (2011) propone un marco de trabajo que consta de dos modelos: 1) un modelo dinámico del sistema junto con una metaheurística para obtener un cronograma óptimo de proyectos de desarrollo de software considerando sus atributos (por ejemplo, prioridad, esfuerzo, duración) y 2) un modelo basado en agente para representar la comunidad de desarrollo como una red social, donde los gerentes de desarrollo forman un equipo óptimo para cada proyecto y equilibran la carga de trabajo entre múltiples proyectos programados según el cronograma óptimo obtenido del modelo

[1] De gran tamaño como sistemas operativos y herramientas realmente grandes, tales como Emacs.

[2] Fielmente representados por los repositorios de archivos de Linux, que pueden aceptar aportaciones de quien sea).

dinámico del sistema. Wang and Carroll (2011) estudia dos comunidades FLOSS (Mozilla y Python) encontrando que las prácticas de corrección de errores son inevitablemente creativas, caracterizando su proceso de corrección de errores como cuatro subprocesos comunes, identificación de problemas, preparación, generación de soluciones y evaluación de soluciones. Xu et al. (2011) investiga los modos de control en las comunidades de proyectos OSS y sus efectos en el rendimiento del proyecto. Con base en una encuesta web y datos de archivo de proyectos OSS, se revela que hay tres tipos de modos de control: basado en resultados, clanes [1] y el autocontrol. Oliveros and Aguilera (2012) investiga las prácticas que predominan en las comunidades FLOSS en cuanto a la gestión, desarrollo, operación y mantenimiento en los proyectos, en ambientes cooperativos y multiculturales[2]. Acuña et al. (2012) afirma que no existe un proceso[3] de desarrollo FLOSS aceptado globalmente para definir cómo se desarrolla el software en la práctica y por lo tanto la descripción de un proceso es importante para coordinar todas las actividades de desarrollo de software que involucran tanto personas como tecnología. Castro Llanos and Acuña Castillo (2012) presenta un estudio para determinar las diferencias y similitudes entre las actividades del proceso de desarrollo (requisitos, diseño e implementación) de las comunidad FLOSS y los procesos establecidos por el Estándar IEEE 1074: 2006, haciendo un mapeo sistemático para descubrir qué actividades de este, son parte del proceso de desarrollo de FLOSS. Chen and Pan (2013) utiliza la teoría de redes sociales para abstraer la topología de redes colaborativas en comunidades FLOSS, proponiendo un método para construir el modelo de red social, que considera tanto la relación de contacto como el nivel de colaboración entre los participantes. Siau and Tian (2013) propone un modelo de procesos FLOSS, para mejorar la capacidad de supervivencia de los proyectos, fundamentado en Fases, Roles, Habilidades y Responsabilidades, dentro de tres fases del proceso OSSD (Etapa de lanzamiento, Antes de la primera versión y Entre versiones). Crowston et al. (2013) presenta tres artículos relacionados con el proceso en comunidades FLOSS: La ingeniería de requisitos en software de código abierto: el papel del entorno externo; Prediciendo fallas en el software de código abierto: tendencias y desafíos; y La búsqueda de UML en proyectos de código abierto Hallazgos iniciales de GitHub. Castro Llanos (2014) identifica tres grupos de condiciones desfavorables que impiden la incorporación de la usabilidad en proyectos FLOSS: participación de un experto en usabilidad; participación de usuarios; y complejidad de aplicación (varios pasos para su ejecución o preparación previa o necesitan de cierta información inicial). Muruzábal and Castillo Acuña (2014) estudia[4] las actividades de requisitos y métodos de trabajo del proceso de desarrollo FLOSS y sus

[1] Control basado en la concepción de valores y creencias de una organización que opera más como una familia que como una empresa.

[2] Con ello contribuir al fortalecimiento y mejora de los procesos de aquellas organizaciones que desarrollan y gestionan FLOSS y detectar practicas que se puedan "migrar" a otros contextos de desarrollo.

[3] Los grupos de actividades en los que se centra son Exploración de conceptos, Requisitos de software, Diseño, Mantenimiento y Evaluación.

[4] Para esto se utilizaron como caso de estudio el proceso en cuatro comunidades: OpenOffice, Mozilla, NetBeans y Eclipse.

comunidades de usuarios y desarrolladores, diferenciandolas de los procesos tradicionales. Wang et al. (2015) estudia las prácticas de revisión por pares en diferentes comunidades FLOSS, especialmente las que involucran distintos tipos de usuarios. Magdaleno et al. (2015) aborda la composición de un proceso para un contexto de proyecto de desarrollo de software específico con el objetivo de maximizar la colaboración entre los miembros del equipo. Gopal et al. (2016) realiza un análisis comparativo del proceso de requerimientos en cuatro proyectos OSS, observando que el volumen de requisitos define en gran medida la naturaleza de su formación social, mientras que la volatilidad en los requisitos determina la superposición que el proyecto tiene con una comunidad externa más grande. Aversano et al. (2017) realiza una evaluación de calidad de la documentación de los sistemas de código abierto, con el objetivo de comprender el soporte que puede ofrecer para adoptarlos y ejecutar actividades de mantenimiento y propone un modelo de calidad para los procesos de documentación. da Silva et al. (2017) realiza un estudio sobre la adopción de practicas ágiles en la liberación de versiones de software en comunidades FLOSS. Alves and De Souza Matos (2017) presenta un resumen y análisis de métodos, técnicas, herramientas, estrategias y enfoques para el diseño de interacción en el contexto del desarrollo de FLOSS. Dong et al. (2018) estudia el proceso de liberación o entrega de software en ambientes FLOSS e identificar los factores que influyen en la duración del proceso de liberación

2.5.4. Generalidades FLOSS

En esta sección se incluyeron los trabajos que de una u otra forma describen conceptos, evolución del movimiento FLOSS y perspectivas futuras.

Stallman et al. (1999) presenta un resumen del movimiento FLOSS, incluyendo historia, los 20 años de unix Berkeley, el movimiento GNU, la perspectiva desde la ingeniería del software, el proyecto linux, el open sources como estrategia de negocio, la venganza de los hackers, entre otros temas escritos por personalidades ampliamente reconocidas. Fuggetta (2003) propone algunas reflexiones y observaciones cualitativas sobre la naturaleza del software de código abierto y sobre las afirmaciones más populares e importantes asociadas con el enfoque de código abierto, distinguiendo las características y aspectos que se pueden aplicar o encontrar por igual en el software propietario. Rossi (2004) hace una revisión de artículos relacionados con FLOSS desde otras disciplinas como economía, derecho, psicología, antropología e informática, concluyendo que lo interesante del movimiento FLOSS no esta en el hecho de que las personas contribuyen libremente, sino más bien en la complejidad de su estructura institucional y su capacidad de evolucionar auto-organizadamente. Feller et al. (2005) presenta un listado de artículos que brindan un panorama general del movimiento FLOSS entre los que se encuentran: Motivaciones para la participación en proyectos, perspectivas económicas y modelos de negocio, Evaluación, procesos y herramientas, casos de estudio de comunidades, legislación y sociedad, y perspectivas de futuro del FLOSS. Al-Marzouq et al. (2005) describe el movimiento FLOSS como un sistema de entrada-proceso-salida en el que se beneficia el

software, la comunidad y la licencia, para cada uno de los cuales identifica beneficios y desafíos. Nelson et al. (2006) realiza una revisión de investigaciones sobre FLOSS y propone un marco de clasificación de investigación que proporciona una estructura formal para clasificarlas y permitir identificar futuras oportunidades de investigación en el área. Hauge et al. (2007) estudia la industria FLOSS desde cuatro roles identificados: Proveedor, integrador, participante y participante en proyectos internos con practicas OSS. Méndez et al. (2008) aborda el intercambio de conocimiento entre los proyectos de software de código abierto basan su operación en una estructura colaborativa para proveer y recepcionar información, experiencia y comentarios a través de las redes sociales y su impacto en el éxito del proyecto, identificando los roles de membresía o de contribución que las personas juegan dentro de los proyectos. Scacchi (2010) concluye que según algunas investigaciones FLOSS nuevos tipos de prácticas, procesos y formas organizativas para descubrir, observar, analizar, modelar y simular. Del mismo modo, se comprende cómo las prácticas, los procesos y los proyectos de FLOSS son similares o diferentes a las contra partes tradicionales, lo que genera futuras investigaciones y estudios comparativos. Yamakami (2010) propone un modelo de tres etapas de la evolución del FLOSS. Yamakami (2011) analiza múltiples caminos evolutivos para FLOSS y presenta un modelo de clasificación bidimensional. Khanjani and Sulaiman (2011) presenta las razones por las cuales se prefiere el desarrollo FLOSS, más que el código cerrado, identificando los desafíos y beneficios que tiene sus usuarios. Batikas et al. (2011) estudia las motivaciones que tiene las pymes en participar en comunidades FLOSS, bajo una perspectiva de comportamiento mediante el uso de un modelo de investigación basado en TPB (Teoría del comportamiento planificado). Kon et al. (2011) discute cómo la investigación y la educación en Ingeniería de Software pueden beneficiarse de la gran cantidad de información disponible en el ecosistema de FLOSS, proponiendo formas concretas de explotar las sinergias entre estas y los proyectos FLOSS. Androutsellis-Theotokis et al. (2011) presenta un completo resumen del movimiento FLOSS desde su historia, proyectos exitosos, organización de las comunidades, los procesos de producción, licenciamiento, modelos de negocio, adopción y re-utilización, motivaciones para la participación e impacto en la sociedad. Bergquist et al. (2012) realiza un estudio mediante entrevista con desarrolladores de software profesionales empleados por empresas, para identificar cómo los valores asociados con FLOSS se convierten en arreglos justificativos que dan legitimidad a FLOSS y cómo estos arreglos cambian con el tiempo[1]. Mahmod and Dahalin (2012) afirma que menos del 2 por ciento de los contribuyentes en los proyectos FLOSS son mujeres y busca comprender las contribuciones femeninas en la innovación de FLOSS, no solo desde un punto de vista técnico singular, sino también desde una perspectiva social constructivista y feminista. Manikas and Hansen (2013) realiza un estudio sobre los ecosistemas de software [2], concluyendo que hay poco consenso sobre lo que constituye un ecosistema de software, existen pocos modelos analíticos de estos y se realiza poca investigación en el contexto del mundo real. Yan and Wang (2013)

[1]Desde el movimiento inicial de software libre hasta la adopción emergente de software libre en el desarrollo de software profesional contemporáneo.

[2]Definidos como el espacio de interacción de un conjunto de actores sobre una plataforma tecnológica común que da como resultado una serie de soluciones o servicios de software.

describe un modelo de desarrollo comunitario de software basado en Crowdsourcing [1], que consta de tres elementos: 1. comunidades en línea[2]; 2. Proporcionando incentivos para la participación de los desarrolladores y también motivando la competencia; y 3. mecanismo de gestión de procesos y control de calidad[3]. Dartsch (2013) presenta una breve historia del movimiento FLOSS, incluyendo la diferenciación del modelo catedral y bazar. Caulkins et al. (2013) concluye que una organización debe de mantener el software propietario durante un período de tiempo finito, determinado de manera óptima, antes de hacerlo de código abierto, haciendo estudios impacto de los costos de cambio y mostrando que en caso de altos costos internos de I + D, la empresa siempre hace que el software sea de código abierto en algún momento, a menos que el cambio en sí sea demasiado costoso. Okoli and Carillo (2013) busca transferir los conceptos propios del funcionamiento del proceso de código abierto, en nuevos medios de expresión como son obras utilitarias, fácticas, estéticas o de opinión, desarrollando un marco para clasificar las obras con derechos de autor. Margan and Candrlic (2015) hace una revisión de artículos relacionada con FLOSS de diferentes campos y disciplinas de las ciencias sociales y de la información permitiendo la comprensión de los motivos que han llevado OSS a su prosperidad y presenta una discusión de los factores clave que contribuyeron al surgimiento del OSS, principalmente desde la perspectiva de la ingeniería y la economía. Alshaikh et al. (2015) presenta un estudio sobre el proceso de localización de software en proyectos FLOSS, concluyendo que del 60 por ciento de las palabras claves usadas tiene éxito en la búsqueda de una aplicación software. Meirelles et al. (2017) presenta un plataforma integrada de desarrollo de software, utilizando varios proyectos de FLOSS, proporcionando una integración no trivial entre ellos. Shaikh and Henfridsson (2017) sostiene que la naturaleza de la gobernanza varía entre las comunidades FLOSS y, en su evolución, pueden coexistir múltiples rastros de autoridad con diferentes estructuras como la autocracia, oligarquía, federación y meritocracia. Franco-Bedoya et al. (2017) evalúa el estado de la investigación en ecosistemas FLOSS, específicamente: cuáles son las definiciones más relevantes, cuáles son las particularidades de este tipo de ecosistemas; y cómo se representa el conocimiento.

[1] Crowdsourcing del inglés crowd –multitud– y outsourcing –recursos externos–) que puede traducirse como colaboración abierta distribuida o externalización abierta de tareas, y consiste en externalizar tareas que, tradicionalmente, realizaban empleados o contratistas, dejándolas a cargo de un grupo numeroso de personas o de una comunidad, a través de una convocatoria abierta.

[2] Proporcionando abundantes desarrolladores de bajo costo con diversas capacidades técnicas y antecedentes

[3] Tomado de la práctica de desarrollo de software interno, que garantiza la calidad del producto y el cumplimiento del cronograma del proyecto.

2.5.5. Despliegue de procesos

Esta sección solo cuenta con los aportes de un trabajo titulado:Taxonomía de factores críticos para el despliegue de procesos software. Bayona et al. (2010) afirma que el despliegue de procesos se centra en la gestión de cambios y debe ser de tal manera que minimice la resistencia del equipo de trabajo.

La revisión de literatura en (Bayona et al., 2010) identifica 24 factores críticos en el despliegue de procesos entre los mas frecuentes participación del personal, la formación, el fortalecimiento de la comunicación, la colaboración y el mentoring.

La taxonomía propuesta define cinco categorías de factores críticos: organización, personas, procesos, producto y otros. En la categoría de factores relacionados con el proceso define las subcategorías:definición de los procesos, liberación de procesos, institucionalización y despliegue del proceso.

Los resultados de la revisión de los estudios de Bayona et al. (2010) muestran el interés que existe para identificar los factores críticos, que condicionan el éxito de una iniciativa de mejora, en el siguiente orden de relevancia, según la frecuencia de estudios encontrados:

- Compromiso de la alta dirección.
- Objetivos de mejora claros, relevantes y aplicables/ objetivos medibles.
- Formación.
- Participación del personal /Iniciativas Bottom-up.
- Fortalecimiento de la comunicación y la colaboración.
- Mentoring.
- Disponibilidad de Recursos.
- Personal con conocimientos y habilidades.
- Experiencia del personal.
- Creación de equipos.
- Agentes de cambio y líderes de opinión.
- Asignación de responsabilidades clara y compensada.
- Políticas de la organización.
- Diferencias de cultura organizacional.
- Gestión del proceso de mejora (seguimiento).

- Personas altamente respetadas.
- Tiempo del personal y recursos.
- Metodología formal.
- Proveer mayor entendimiento.
- Adoptar iniciativas de mejora.
- Sensibilización.
- Estabilidad en los cambios de los procesos.
- Incentivos tangibles.
- Descongelar la organización.

Capítulo 3

Modelo de despliegue de procesos

Este capitulo presenta el modelo propuesto y validado mediante juicio de expertos a quienes se les aplicó una encuesta en linea para evaluar si el modelo propuesto cumple con las características definidas en las tres hipótesis de este trabajo: «Nivel de rigor apropiado» [1], nivel de dependencia del proceso en las herramientas [2], y nivel de formalización apropiado[3].

En esa sección se presentará el modelo objeto de estudio de este trabajo, para lo cual se requiere una contextualización del proceso de software en comunidades FLOSS, posteriormente se describirá el modelo y finalmente se presenta los resultados de la validacion aplicados a ingenieros conocedores del movimiento FLOSS y de los procesos de desarrollo dentro de una comunidad.

[1] Los procesos de desarrollo software, carecen de rigor procedimental debido a su carácter colaborativo, pues los miembros del equipo participan voluntariamente sin remuneraciones económicas ni compromisos contractuales. Variable: nivel de rigor.

[2] Las herramientas de comunicación son indispensables en las comunidades FLOSS dado su carácter distribuido, pues los miembros no se encuentran en un mismo espacio físico. Variable: Nivel de dependencia del proceso en herramientas.

[3] Es posible formalizar el proceso de desarrollo en una comunidad FLOSS, de manera que motive la participación del voluntariado novato en este tipo de comunidades. Variable: nivel de formalización del proceso software.

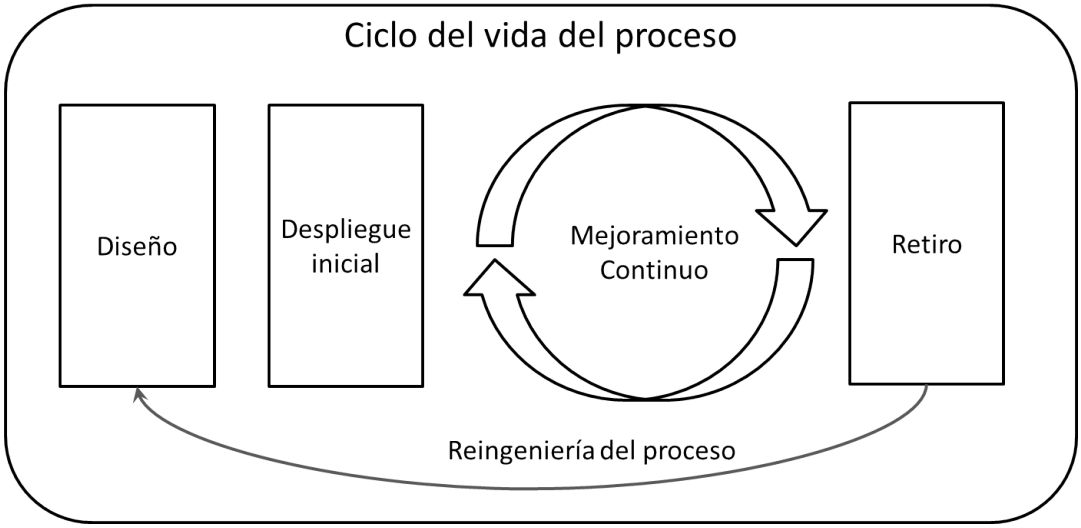

Figura 3.1: Ciclo de vida de un Proceso

3.1. Contextualización

Ante el auge del movimiento FLOSS a inicio de la primera década del 2000, se produjo una primera división entre lo que se considera software libre y open source. Mientras el primero fue un movimiento social alrededor de una filosofía de libertades, el segundo en torno a una metodología de desarrollo software y un modelo de negocio, sin embargo hoy en día estos dos enfoques convergen en un solo concepto, gracias al punto en común de los dos movimientos en lo que respecta al acceso del código fuente como elemento indispensable para lograr las libertades (del movimiento de software libre) y el adecuado proceso de desarrollo (del movimiento de Open Source) . Este punto en común unifico en lo que se reconoce como FLOSS "libre / Free Open Source Software", Software Libre/ Open Source.

Independiente de las intención de origen de los términos, es claro que se trata de desarrollar software, y por lo tanto es indispensable hablar del proceso que las diversas comunidades afines a estos movimientos usan para obtener productos software bajo criterios de cooperación voluntaria, y en la mayor parte de los casos de manera distribuida hablando de la ubicación de los miembros del equipo.

Por otro lado cualquier proceso no solo en el contexto FLOSS, sino generalizado a otros contextos incluso los industriales, pasan por una serie de fases durante su ciclo de vida y es por lo tanto necesario interpretar la finalidad y el producto de cada una de estas fases.

De acuerdo a la figura (3.1), la etapa del diseño, consiste en determinar principalmente los pasos del proceso, probablemente organizados en fases, definiendo relaciones de precedencia, entradas, salidas, herramientas y técnicas necesarias para llevar a cabo cada

uno de los pasos constitutivos de un proceso.

El despliegue inicial tiene como objetivo, poner en marcha por primera vez un proceso en un contexto real y no abstracto como en el caso del diseño. Esta fase incluye la adecuada documentación del proceso y la eficaz comunicación a las personas que lo ejecutan. La fase de mejoramiento continuo consiste principalmente en el control de cambios del proceso, realizado mediante el continuo monitoreo (medición) de cada uno de sus subprocesos, y pensado en función de la eficacia y eficiencia. Se espera que esta fase sea la más larga del ciclo de vida en la medida que el proceso se pueda seguir mejorando sin que se declare obsoleto a la luz de los avances tecnológicos que se den, principalmente en las herramientas y técnicas utilizadas por el proceso.

La fase de retiro, consiste en declarar la obsolescencia de las herramientas y técnicas, de tal manera que los procesos de mejoramiento continuo pierden su eficiencia y por lo tanto resulta más económico el cambio total del proceso. Esto se da principalmente en procesos de otros contextos en los que se requieren de maquinarias o herramientas físicas que por el desgaste y por la evolución tecnológica, resultan ineficientes. Es probable que esta etapa de retiro no se dé en algunos procesos como el de desarrollo software, pues no dependen de herramientas físicas, por lo cual se puede dar el caso que un proceso a través del mejoramiento continuo cambie de manera radical en el tiempo y por lo tanto el mejoramiento continuo puede ser visto como un proceso de reingeniería, al incluir los cambios naturales de herramientas y técnicas al ritmo del avance de las tecnologías utilizadas en el desarrollo de software.

Particularmente en los procesos de desarrollo software, se puede interpretar este mismo ciclo de vida del proceso en términos de los conocimientos requeridos para hacer efectivas cada una de estas etapas como se muestra en la figura (3.2)

La fase de diseño requiere un adecuado estudio de los modelos de ciclo de vida del software existentes de acuerdo a las necesidades de la organización y tipos de proyectos de desarrollo, ofrecidos por dicha organización. De igual manera se requiere el estudio de diversas metodologías que han mostrado ser útiles, o simplemente que se han convertido en moda gracias a procesos de comunicación y divulgación eficaces. No hay que descartar el estudio de estándares formalmente definidos como el 12207 y 1074 de la ISO, dado que todo proceso de software típicamente se subdividen en subprocesos y estas normas dan claridad de los alcances de cada uno de estos subprocesos, ya sea a nivel de un proyecto particular o a nivel de una organización que se dedica a desarrollar software.

La etapa de despliegue inicial, se fundamenta en la capacidad de describir sin ambigüedades cada una de las actividades del proceso y para ello SPEM2 se han convertido en estándar de facto, por lo que el proceso esta descrito en términos de roles, actividades, herramientas y técnicas utilizadas en cada actividad. Un segundo factor dentro de esta etapa es la capacitación a las personas que van a usar el proceso, y es así como se garantizará la utilidad de la documentación, pues tener documentado un proceso no implica que las personan ejecuten las actividades como se definen en un documento.

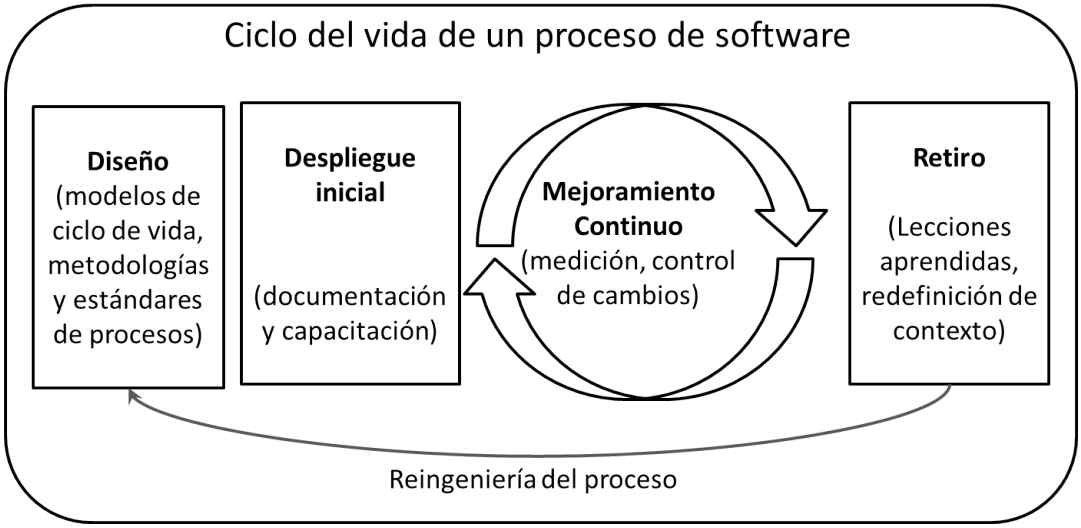

Figura 3.2: Ciclo de vida Proceso de desarrollo de software

La fase de mejoramiento continuo está relacionada con modelos de calidad en los procesos de software y para lo cual se reconocen en el ámbito académico y empresarial a CMMI como modelo ampliamente probado y efectivo en la mejora de procesos a nivel organizacional, con sus descendientes TSP a nivel de equipos de un proyecto, o PSP a nivel de miembros individuales de un desarrollo software.

Los cambios de proceso también surgen a partir de las mediciones de calidad del producto y es por lo tanto necesario tener en cuenta modelos y estándares relacionados como McCall[1], GQM o Goal Question Metric[2], FURPS[3], Dromey[4], GILB[5], ISO 9126[6],

[1]Uno de los modelos pioneros en la evaluación de la calidad de software, tiene una estructura basada en tres niveles: factores, criterios y métricas. Los once criterios base, son: exactitud, confiabilidad, eficiencia, integridad, usabilidad, mantenibilidad, testeabilidad, flexibilidad, portabilidad, reusabilidad e interoperabilidad.

[2]Proporciona una forma para definir métricas a partir de la aplicación de unas preguntas relacionadas con el proyecto, que permitan alcanzar unas metas previamente planteadas. Este modelo se basa en tres niveles: metas, preguntas y métricas.

[3]Modelo desarrollado por Hewlett-Packard, cuyo nombre proviene de los criterios que evalúa: Funcionalidad, Usabilidad, confiabilidad (Reliability), desempeño (Performance) y Soportabilidad.

[4]Propone 3 modelos para cada etapa del proceso de desarrollo: modelo de requerimientos, modelo de diseño y modelo de calidad de la implementación.

[5]Orienta la evaluación de software a partir de los atributos: capacidad de trabajo, adaptabilidad, disponibilidad y utilizabilidad, los cuales se dividen en subatributos.

[6]Estándar basado en el modelo de McCall. Está organizado en cuatro partes: modelo de calidad, métricas externas, métricas internas y calidad de métricas en uso. El modelo incluye la evaluaciond e

SQAE o Software Quality Assessment Exercise[1], WebQEM[2], ISO 25000[3]. Por otro lado es importante reconocer que gran parte de las investigaciones ligan el proceso de despliegue a la mejora de procesos y principalmente al modelo CMMI.

Como se dijo previamente, el proceso de retiro y su posterior fase de reingeniería pueden considerarse inexistente en los procesos de desarrollo software, en el sentido de que al no depender de herramientas físicas, puede sufrir cambios drásticos como parte del mejoramiento continuo, es decir que el proceso inicialmente desplegado en una organización pude ser muy diferente al usado en un punto del ciclo de vida, siempre y cuando el mejoramiento continuo garantice el control de cambios en la documentación y un proceso eficaz de comunicación de dichos cambios por medio de capacitaciones. Por lo tanto,las lecciones aprendidas y el cambio de tecnologías se incorporan de manera rápida en la evolución de un subproceso, lo que a largo plazo se puede ver como una reingeniería del proceso.

Finalmente es importante resaltar la diferencia que puede haber en los contextos del desarrollo FLOSS y de desarrollo propietario, en el cual no existe una forma de libre acceso al código fuente por fuera de los límites de una organización que necesita el software (y ella misma lo desarrolla) o de una organización que se dedica a producir software y que fue contratada por quien necesita el producto.

La principal diferencia en estos dos contextos, FLOSS y propietario, radica principalmente en dos factores la colaboración voluntaria y la distribución geográfica de los miembros del equipo. El primer factor y más importante diferenciador está relacionado con la vinculación de los miembros a un equipo de desarrollo, mientras en el contexto de FLOSS, en la mayor parte de los casos no hay una retribución económica para todos los miembros del equipo, en el contexto propietario si hay un contrato que de una u otra forma obliga a los miembros del equipo a cumplir con las reglas definidas por el contratante. También es cierto que el segundo factor hoy en día puede ser superado mediante el

caracteristicas como: funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad, para cada una de las cuales define subcaracterísticas.

[1]Modelo, basado en Boehm, McCall, Dromey e ISO 9126, está orientado principalmente a realizar evaluación por terceros que no están directamente involucrados con el desarrollo. Está organizado en tres capas: área, factor y atributo de calidad, que permiten orientar la evaluación jerárquicamente.

[2]Es una metodología de evaluación de calidad de sitios Web (Web-site Quality Evaluation method), siguiendo seis fases: planificación y programación de la evaluación de calidad¸ definición y especificación de requerimientos de calidad, definición e implementación de la evaluación elemental¸ definición e implementación de la evaluación global¸ análisis de resultados, conclusión y documentación¸ validación de métricas.

[3]También llamadas como SQuaRE, cuyo propósito es guiar el desarrollo con los requisitos y la evaluación de atributos de calidad, principalmente: la adecuación funcional, eficiencia de desempeño, compatibilidad , capacidad de uso, fiabilidad, seguridad, mantenibilidad y portabilida.

uso adecuado de herramientas tecnológicas para trabajo en equipo, sin embargo se ha demostrado que el trabajo colaborativo en un mismo espacio físico redunda en calidad tanto del proceso, como del producto, tal es el caso de metodologías como SCRUM.

3.2. Descripción general del modelo propuesto

En el contexto de comunidades FLOSS, el despliegue de un proceso de software debe traspasar las propias fronteras del concepto de despliegue es decir las fases de Despliegue inicial y mejoramiento continuo y por lo tanto el modelo propuesto involucra la fase de diseño, pues esta debe garantizar la usabilidad de los procesos definidos para una comunidad en particular.

Como muestra la figura 3.3 el modelo plantea tres componentes (restricciones, técnicas y herramientas) indispensables para garantizar un adecuado despliegue de procesos en una comunidad FLOSS probablemente, inexistente cuando un proyecto de este tipo se inicia. Este modelo además de incorporar la fase de diseño del proceso como parte del despliegue, también incluye las particularidades de un contexto de desarrollo, en el cual la colaboración es voluntaria y sus miembros se encuentran distribuidos geográfica.

Como todo modelo presenta generalidades que cada comunidad FLOSS que lo utilice, adaptará a sus propias particularidades, por lo que las siguientes secciones muestran ejemplos de restricciones, técnicas y herramientas que pueden ser parte del despliegue de procesos en una comunidad, mas que exigencia como de obligatoriedad dentro del modelo propuesto.

3.2.1. Diseño

Un proceso de software debe ser visto de manera general como todo el proceso de desarrollo que pude ser subdivido en subprocesos, por lo tanto, cuando se habla de despliegue inicial del proceso se considera por primera vez el proceso integral cuyo único producto es un software funcional, lo que incluye el código fuente, especificaciones, diseños, y documentación (tanto para los desarrolladores como para los usuarios del software).

En una comunidad FLOSS, antes que un proceso se debe contar con una versión inicial conocida como «Promesa Creíble», y es esta la que motiva la vinculación de miembros a un proyecto, por lo tanto esta versión inicial del producto no aplicó un proceso de desarrollo comunitario y por lo tanto el diseño del proceso será una tarea paralela a la vinculación de los primeros participantes.

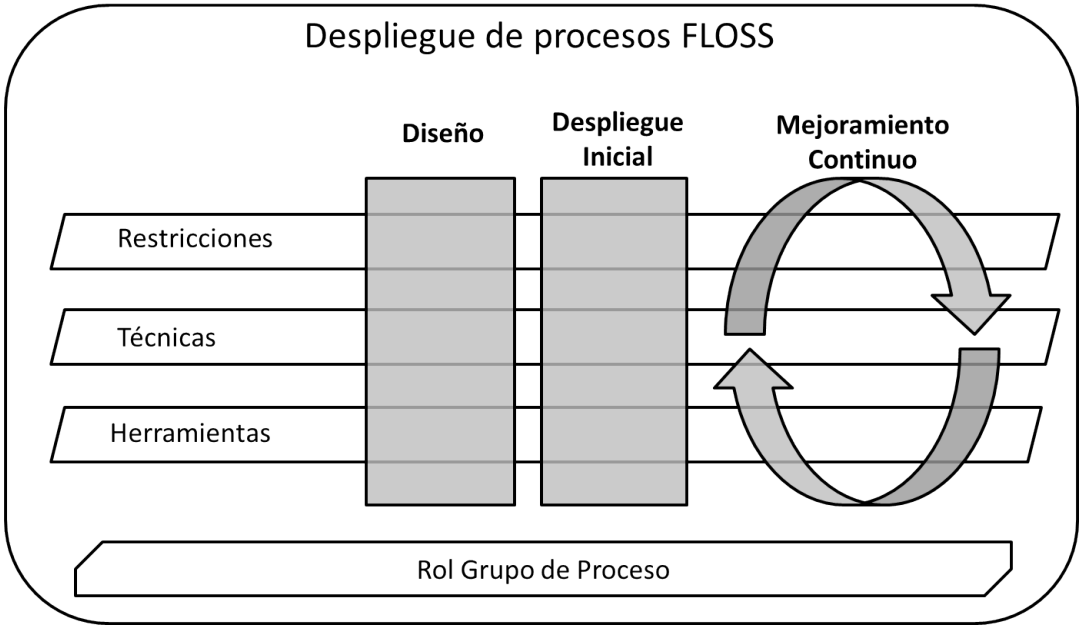

Figura 3.3: Modelo de despliegue Propuesto

Es ahí donde se requiere la conformación del grupo de proceso [1], como un rol en la cual sus integrantes diseñarán, documentarán y dan soporte permanente a todos los demás miembros en los diferentes roles que se determinen.

3.2.1.1. Restricciones

La principal restricción de la componente de diseño de proceso tiene que ver con el **contexto**: tener en cuenta las características propias de las personas que va a usar el proceso, por lo que se hace necesario caracterizar esta población, que si bien no esta definida claramente al inicio del proyecto, se puede predecir dependiendo del dominio del problema que aborda el proyecto. Parte importante del contexto es la distribución geográfica de sus miembros, para lo cual el proceso debe estar soportado en las herramientas de comunicación apropiadas.

Iterativo e incremental: Los contextos de desarrollo FLOSS son caracterizados por las frecuentes liberaciones, por lo tanto el proceso diseñado debe incluir las adecuadas estrategias de desarrollo iterativo e incremental.

Diseño basado en roles y no en subprocesos: Las Notaciones y herramientas de documentación de procesos, permiten la descripción global de un proceso de software, sin

[1] Que puede ser considerado un rol más dentro del proceso global de desarrollo.

embargo debido a la gran variedad de perfiles en los miembros de una comunidad, estos se autorganizan según sus habilidades por lo tanto su vinculación siempre hace en función del rol que desempeñan, es así como el diseño de los procesos FLOSS deben estar en función de cada uno de los roles necesarios, entre los cuales se encuentran: desarrolladores, diseñadores, tester, documentadores y un nuevo rol propuesto en este modelo de ingeniero de procesos.

Flexible:El rol de ingeniero de procesos incluye dentro de sus responsabilidades la medición y análisis de los procesos de cada uno de los roles definidos en una comunidad. por lo que el diseño inicial del proceso debe incluir aspectos que faciliten los cambios, sin sufrir traumas al interior de los participantes en el rol afectado por el cambio. Es así como esto podría denominarse mantenibilidad del proceso.

Simplicidad: Mínimo necesario y suficiente Basados en la idea que algo es simple cuando no hay nada que quitar, por lo tanto los procesos deben ser mínimos necesarios para lograr la participación del mayor numero de interesados.

3.2.1.2. Técnicas

La principal técnica en el diseño de procesos la constituyen las Librerías de Activos de Procesos -Process Asset Library PAL-, para lo cual se debe promover repositorios de procesos de libre acceso. Al igual que el código fuente de un producto software está disponible, los procesos de las comunidades FLOSS deben estar adecuadamente documentados y de libre acceso.

3.2.1.3. Herramientas

La principal herramienta para la documentación de procesos software, es Eclipse Process Framework Composer -EPF composer-, mediante la cual se puede especificar para cada uno de los roles las actividades propias de su función, los artefactos generados en dichas actividades y las herramientas utilizadas. De igual forma el proyecto EPF composer cuenta con un repositorio de librerías de procesos con la documentación de procesos como OpenUP, Scrum y XP de libre acceso y por lo tanto pueden ser usadas como punto de partida para documentar los procesos propios de una comunidad FLOSS. Otras herramientas adecuadas para mantener un repositorio de procesos son las WikiPAL propuestas por Bermón-Angarita (2010).

3.2.2. Despliegue Inicial

Esta componente del modelo incluye la publicación de la documentación del proceso, la capacitación de los miembros del equipo, y la instalación e integración de las herramientas que soportan las actividades para cada uno de los roles definidos en el proceso previamente diseñado.

3.2.2.1. Restricciones

La principal restricción en este aspecto corresponde a la **legibilidad del proceso**, es decir la facilidad con que cada miembro del equipo comprenda las actividades propias de su rol y el uso de las herramientas para la construcción de los artefactos bajo su responsabilidad. **Integración de herramientas**: Los repositorios y control de versiones se han constituido en la principal herramienta para el trabajo en comunidades FLOSS, sin embargo se requieren otras herramientas como las de comunicación, foros, reporte de errores, pruebas,herramientas de documentación, que deben ser adecuadamente integradas para buscar que los miembros de una comunidad, consideren un solo contexto, de tal manera que con un solo login puedan acceder a las herramientas involucradas en un rol determinado.

3.2.2.2. Técnicas

El despliegue de un proceso inicial no es tan crítico, como es el caso de despliegue de código, puesto que se debe garantizar la disponibilidad de la funcionalidad en tiempo real. Para el contexto de este trabajo se sugiere que el despliegue inicial se haga desviando el flujo de participaciones de los usuarios más expertos a una instancia del nuevo proceso, de tal manera que se pueda considerar una prueba beta del proceso, que con su retroalimentación se hagan los ajustes pertinentes para dar en servicio al cien por ciento de los participantes. La **capacitación** por diferentes medios se constituye en la principal técnica para la comprensión de las actividades, herramientas y técnicas usadas por cada miembro dentro de un rol determinado, es así como el grupo de procesos debe disponer de las estrategias correspondientes para asesorar principalmente en el uso de las herramientas.

3.2.2.3. Herramientas

Plataformas de desarrollo colaborativo: También conocidas como forjas son principalmente herramientas de control de versiones, en la cual se registran todo tipo de documentos digitales, por lo que en estas se puede llevar control no solo del código

fuente, sino de documentación de requerimientos, diseños, pruebas, manuales y en general todo tipo de artefacto en un proceso de desarrollo software. **Comunicación entre miembros del equipo**: Las listas de correos y foros se constituyen en la principal herramienta para comunicación entre los miembros de las comunidades FLOSS. Por medio de estas se comunican nuevos requerimientos, se discuten alternativas de solución y se emprenden procesos de codificación y pruebas. dentro de este tipo de herramientas también se pueden incluir las de **Reporte y seguimiento de bugs**, [1] son usados para el registro de fallos encontrados por los usuarios, y por lo tanto sirven como herramienta de definición de requerimientos, planificación y asignación de tareas, que al ser integradas con herramientas de control de versiones y de comunicaciones como las listas de correo, se constituyen en toda una plataforma de desarrollo colaborativo. Las herramientas de **Documentación**: Los manuales técnicos[2], de usuario y cualquier otra documentación que facilitan la modificación[3], el uso y operación de un producto software se constituyen en garantía de sobrevivencia a largo plazo de un proyecto FLOSS. **Wiki**:Probablemente este tipo de herramientas son las mas versátiles para el contexto FLOSS, pues con ellas se podría soportar procesos de documentación para usuario, documentación del mismo proceso, requerimientos, diseños e incluso en forma de portal de promoción de un proyecto FLOSS.

3.2.3. Mejoramiento Continuo

El componente se basa en el manejo de los cambios del proceso a partir de procesos de medición, que detectan partes de un proceso que puede mejorar.

3.2.3.1. Restricciones

En las comunidades FLOSS, los incrementos en el producto se dan a medida que se registran los commit[4] y sobre estos se hace necesario realizar los análisis pertinentes en

[1] El término usado en inglés es Bug Tracking System BTS. Puede considerarse como un tipo especial de sistema de seguimiento de incidentes, para lo cual existen una variedad de aplicaciones software como Bugzilla,Trac,Mantis,Redmine, Flyspray, entre otras.

[2] Descripción de diseños, requerimientos, paquetes, clases, código fuente, los códigos de error y sus significados, entre otra información de utilidad para comprender como esta construida una aplicación.

[3] El código documentado por lo general esta asociado a una herramienta dependiendo del lenguaje de programación, por lo tanto existen variedad de herramientas para este propósito como javadoc, Doxygen, Slate API Docs Generator, DocFX, NDoc, phpDocumentor, Sandcastle, DoxyS, Natural Docs, Asciidoclet, LuaDoc, TypeDoc, LDoc, entre otras.

[4] Consolidar, confirmar, registrar un cambio o hacer un commit se refiere a confirmar un conjunto de cambios provisionales de forma permanente, en uno o varios artefactos constitutivos de un producto

búsqueda de aspectos en el proceso que puedan ser susceptibles de mejoras, por lo tanto la principal restricción para la mejora de procesos esta dada por los artefactos sobre los cuales se pueda hacer análisis de información.

El acceso a la información de los commit al interior de una comunidad, debe ser ilimitada para el grupo de procesos, sin embargo puede ser más útil al estar disponible para cualquier persona mediante repositorios como SourceForge Research Data Archive, FLOSSmole, FLOSSMetrics.

3.2.3.2. Técnicas

La **Caracterización de defectos**, realizada para encontrar oportunidades de mejora, permite encontrar la causa raíz de los fallos detectados en los procesos de prueba, **el análisis de commit revertidos**, dado que un volumen de los aportes en una comunidad no son incorporados al producto por diferente motivos y el análisis de esta información permite la mejora de procesos.

3.2.3.3. Herramientas

Análisis de datos: Las herramientas estadísticas y de inteligencia artificial que permitan el tratamiento de grandes volúmenes de datos que permitan encontrar patrones de comportamiento, con los cuales se pueda identificar oportunidades de mejora de un proceso.

3.2.4. Rol de Ingeniero de procesos

Este rol dentro de una comunidad FLOSS tiene como misión permanente el identificar, analizar y proponer soluciones a ineficiencias del proceso de desarrollo, desempeñando como principales funciones: Hacer análisis permanente sobre los commit para identificar aspectos de mejora en el proceso. Ayudar a los miembros del equipo de trabajo con el manejo de las herramientas en cada uno de los roles definidos. Ayudar a entender y adoptar el proceso definido para la comunidad Gestionar los cambios del proceso, incluyendo la integración de herramientas y capacitación a los miembros de la comunidad.

Dentro de las capacidades de las personas que participen en este rol se deben encontrar: Ser perseverante y organizado, conocedor de procesos de desarrollo software,

software.

con capacidades de auto-aprendizaje [1], el respeto, el compromiso, la coherencia, trabajar en equipo, ser comunicativo [2] y asertivo [3] y finalmente debe tener desarrollar habilidades especiales para la resolución de conflictos [4].

3.3. Validación del modelo propuesto

El juicio de expertos se utiliza en este trabajo como estrategia para la validación de la hipótesis número 3: "Es posible formalizar el proceso de desarrollo en una comunidad FLOSS, de manera que motive la participación del voluntariado novato en este tipo de comunidades.(Variable nivel de formalización del proceso software)".

3.3.1. Instrumento

El siguiente instrumento se diseñó para medir la conveniencia o no de la utilización del modelo en una comunidad FLOSS, según el criterio de los expertos. La primera parte del instrumento permite recolectar información sobre el nivel de experiencia de cada uno de los invitados a diligenciar el instrumento. La segunda parte corresponde a la conveniencia de la utilización del modelo[5].

1. Datos del participante
    - *a*) Nombre completo
    - *b*) Nivel de formación
    - *c*) Profesión desempeñada
    - *d*) En que grado conoce el funcionamiento de una comunidad FLOSS? (básico, medio, alto, muy alto)
    - *e*) Ha participado en una comunidad FLOSS? en caso afirmativo Cual? con que rol?

[1] La capacidad de aprender por uno mismo incluye entre otras cosas, la experimentación y el aprender de los errores.

[2] Saber transmitir la esencia de los proceso y persuadir a los miembros para ver las ventajas en el uso del proceso.

[3] Debe ser capaz de ponerse en el rol de cada uno de los grupos con los que debe comunicarse, y construir un ambiente de interacción apropiado entre los miembros de una comunidad.

[4] Probablemente en grandes comunidades es el aspecto inevitablemente que siempre surge cuando hay personas con diferentes intereses.

[5] Para dar a conocer el modelo se elaboró una síntesis que se encuentra en los anexos.

2. Sobre el modelo propuesto

   a) La utilización del modelo propuesto motivará la vinculación de miembros novatos en una comunidad FLOSS.

   b) La utilización del modelo propuesto motivará la participación de miembros que tienen experiencias previas en comunidades FLOSS.

   c) La formalización de un proceso mediante SPEM, como propone el modelo, creará inconformidad en los miembros de un equipo de trabajo FLOSS.

   d) Un procesos crítico al momento de trabajar con equipos virtuales, es el de comunicaciones. El modelo propuesto facilita el proceso de comunicación en una comunidad FLOSS.

   e) Favor dejar aquí las observaciones y comentarios que considere pertinentes respecto al modelo evaluado. (opcional)

Esta segunda parte de la encuesta usa una escala de Likert[1] para las opciones de respuesta:

1. Totalmente en desacuerdo
2. En desacuerdo
3. Ni de acuerdo ni en desacuerdo
4. De acuerdo
5. Totalmente de acuerdo

3.3.2. Análisis de resultados

Una vez aplicado el instrumento, para lo cual los expertos usaron un resumen del modelo propuesto y el instrumento en linea, que también se presento en formato impreso para conocer de antemano las preguntas sin necesidad de visitar el instrumento en linea.

Se obtuvieron 15 participaciones de expertos con la siguiente caracterización:

- **Profesión actualmente desempeñada:** Los expertos reportaron[2] un perfil

[1] Se denomina así por su autor Rensis Likert y es una escala psicométrica comúnmente utilizada en encuestas, en la cual se especifica el nivel de acuerdo o desacuerdo con una declaración.

[2] Estudiante, Ingeniera de Sistemas, Secretaria académica, Líder de Desarrollo, Docente,ingeniero de sistemas, Ingeniero Informático, Consultor, Ingeniería Electrónica, Consultora Inteligencia de negocios, Jefe de Centro de Tecnologías de Información y Comunicación, Ingeniero de Sistemas, Gestor oficina tecnologías de información, Oficial de seguridad informática,Ingeniero de sistemas.

variado entre los cuales se encuentran estudiantes, ingenieros, docentes y consultores principalmente.

- **Máximo nivel de Formación Académica obtenido hasta el momento**. Ver figura 3.4. Los participantes en su mayoría tienen titulo profesional, lo que permite deducir que sus conceptos son basados en la teoría que adquirieron en el programa académico de pregrado, más que en su experiencia en comunidades FLOSS.

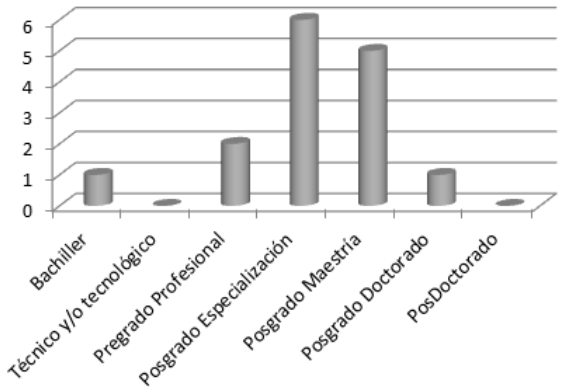

Figura 3.4: Nivel académico de los participantes

- **¿En qué grado conoce el funcionamiento de una comunidad FLOSS?** Ver figura 3.5. Los participantes en su mayoría tienen un nivel básico de conocimiento de las comunidades FLOSS, lo que no garantiza en su totalidad la viabilidad de la aplicación del modelo, por lo cual se requiere conseguir conceptos de miembros que hayan estado involucrados en procesos de desarrollo FLOSS.

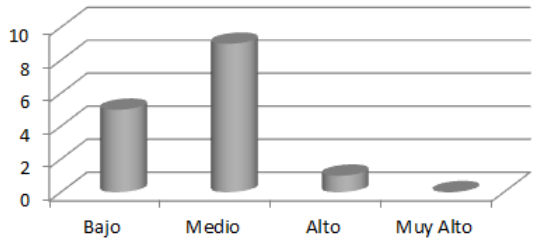

Figura 3.5: Conocimiento de los participantes respecto a las comunidades FLOSS

- **¿Ha participado en una comunidad FLOSS?** ver figura 3.6. La baja participación en comunidades FLOSS, coincide con el bajo conocimiento sobre los procesos internos de desarrollo en este tipo de comunidades.

Respecto a la evaluación del modelo propuesto por parte de los expertos se obtuvo como análisis de las respuestas a cada una de las siguientes preguntas:

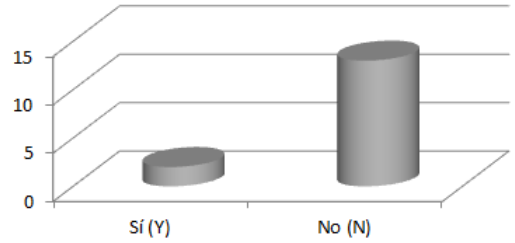

Figura 3.6: Sobre la participación de los expertos en Comunidades FLOSS

- **La utilización del modelo propuesto motivará la vinculación de miembros novatos en una comunidad FLOSS**: Ver figura 3.7. Es viable la implementación del modelo, pues el éxito de una comunidad FLOSS depende en gran medida del numero de participantes, y aunque han pasado varias décadas desde que el movimiento se inició, muchos interesados en participar en una comunidad por primera vez, encuentra un obstáculo en el entendimiento del proceso interno, más aun cuando no tienen como profesión la informática.

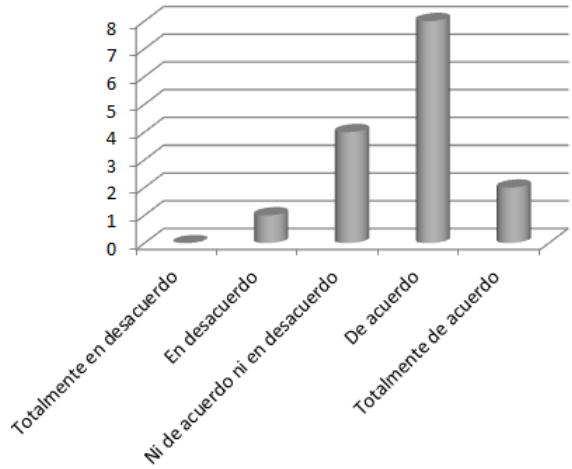

Figura 3.7: El modelo fomentara la participación de novatos en una Comunidades FLOSS

- **La utilización del modelo propuesto motivará la participación de miembros que tienen experiencias previas en comunidades FLOSS**: Ver figura 3.8. De acuerdo a los resultados es viable implementar el modelo, pues no impide que los usuarios con experiencias previas en comunidades FLOSS, vea la documentación de un proceso como perdida de libertad en su forma de desarrollar o de participar en la comunidad.
- **La formalización de un proceso mediante SPEM, como propone el modelo, creará inconformidad en los miembros con experiencia en el trabajo de equipos FLOSS**. Ver figura 3.9. La documentación de procesos mediante

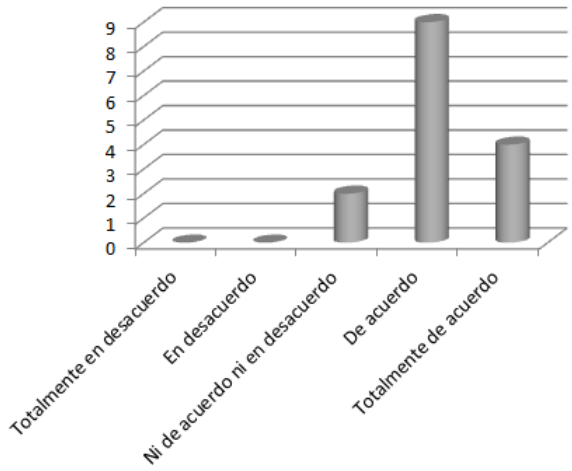

Figura 3.8: El modelo motivará la participación de miembros expertos en una Comunidades FLOSS

notaciones simbólicas facilita su entendimiento, independiente del idioma, es así como los miembros con experiencia tendrán oportunidad de asimilar fácilmente los cambios en un proceso, y no verán en esta notación una imposición que obstaculiza la libertad de participación en una comunidad.

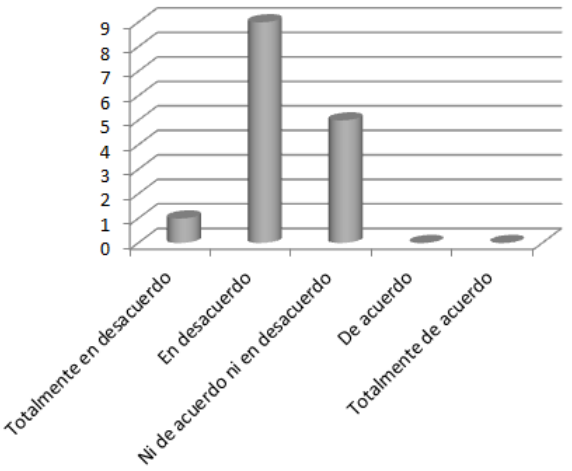

Figura 3.9: La formalización con SPEM, creará inconformidad en los miembros de una comunidad FLOSS

- **Un procesos crítico al momento de trabajar con equipos virtuales, es el de comunicaciones. El modelo propuesto facilita el proceso de comunicación en una comunidad FLOSS**.Ver figura 3.10. En su mayoría los participantes están de acuerdo que facilita la comunicación dentro de un proceso de

desarrollo, es altamente probable que si todos los miembros de un equipo tienen una misma interpretación, los canales de comunicación entre ellos estarán claramente determinados.

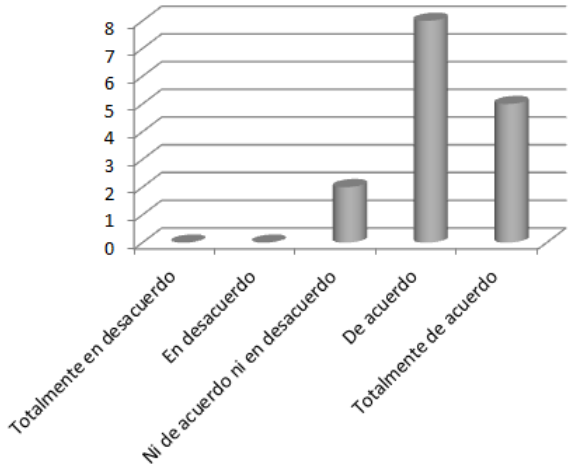

Figura 3.10: El modelo facilitará el proceso de comunicación

Al final del instrumento se permitió el registro de comentarios y observaciones que cada experto considerase pertinentes respecto al modelo evaluado. Estos son los comentarios registrados

- A partir del resumen del articulo y de la intencionalidad propuesta por el investigador, se evidencia su finalidad y la pertinencia del Modelo es coherente con los objetivos planteados, porque propende al mejoramiento de despliegue en comunidades FLOSS. No obstante la sintaxis de cada párrafo no refleja coherencia y cohesión respecto a la redacción.
- El modelo se puede adaptar a cambios.
- La caracterización deberá contar con un mínimo de personas a tener en cuenta en el modelo, a su vez no se tiene claridad en las herramientas a utilizar para la integración de herramientas y plataformas de trabajo colaborativo es muy abierta las opciones.
- En el modelo propuesto no me queda claro cómo las actividades son clasificadas en función de su prioridad, el tiempo estimado para ser completadas, que dependencias tiene de otras actividades, cómo es monitoreada en función del individuo asignado a la misma.
- Es adecuado y pertinente pues se definen de buena manera los roles dentro de la comunidad.

- El modelo propuesto define unos roles del equipo de trabajo. El producto de software inicial se conoce como Promesa Creíble. El diseño plantea la conformación de un grupo de procesos con integrantes con el rol de diseñar, documentar y brindar soporte permanente a todos los demás miembros de los diferentes roles.
- El modelo permite diseñar y documentar procesos de desarrollos de software de manera gráfica y con cumplimiento de estándares teniendo en cuenta los roles que se definen en el de tareas y actividades, productos de trabajo o artefactos, herramientas y procesos.
- Se puede apoyar con DevOps
- Considero sería útil describir brevemente el actual proceso para entender mejor las ventajas del modelo propuesto, realizando una comparativa entre estos dos.

En general, los comentarios reflejan una viabilidad de la aplicación del modelo, pues algunos de ellos están enfocados hacia evaluar el documento resumen, más no el modelo en si. Los comentarios negativos hace referencia al diseño del proceso más que al despliegue, por lo que el modelo en sí no describe las actividades puntuales del desarrollo, sino que plantea una fase de diseño del proceso en el cual se deben especificar no solo las actividades concretas, sino los roles, herramientas y artefactos en notación SPEM.

Capítulo 4

Conclusiones, recomendaciones y trabajos futuros

4.1. Conclusiones

- El estudio de los procesos de desarrollo software en comunidades FLOSS, requiere de un entendimiento de los modelos de ciclo de vida tradicionales, los estándares de desarrollo, las metodologías tanto tradicionales como ágiles, y los modelos de mejora de procesos usados en ambientes empresariales, en contraste con la simplicidad de los procesos requeridos en una comunidad FLOSS, pues si bien los métodos ágiles procuran simplicidad, puede no ser suficiente frente a procesos donde los miembros son voluntarios y la reglas impuestas por una metodología pueden desmotivar fácilmente la participación de una persona al considerarlas obstáculo en su forma de trabajo.
- Han pasado varias décadas desde la publicación del articulo "la Catedrál y el Bazar", pero sigue siendo probablemente el material más importante que resume el modelo de desarrollo FLOSS, en el cual frente a un casi caótico proceso, se obtienen productos de alta calidad y se logra la coordinación de grandes equipos de trabajo, con variedad de perfiles e interés. Este tal vez sea un objeto de estudió mas pertinente para los sociólogos.
- Gran parte de las investigaciones encontradas en el estado del arte relacionado con el despliegue de procesos software, se encuentran enfocadas en la mejora de procesos y por lo tanto en modelos como CMMI, TSP, y PSP, sin embargo la dinámica de funcionamiento de los ambientes colaborativos y distribuidos propios de las comunidades FLOSS, requiere estudios respecto a los procesos de comunicación

y socialización de los procesos, como herramienta para fomentar la participación masiva de voluntarios.

- El modelo de despliegue de procesos software, propuesto para comunidades FLOSS, traspasa los limites del despliegue, por lo que se hizo necesario la incorporación de una fase de diseño del proceso, en reconocimiento de las particularidades respecto a los procesos empresariales, donde se tiene un control total de los participantes en el equipo de trabajo. Es así como el carácter de voluntariado en el desarrollo FLOSS, requiere que el mismo proceso sea acordado entre los miembros iniciales de una comunidad, sin sobrecargar de actividades que desmotiven la participación de gran variedad de perfiles interesados en un proyecto FLOSS particular.
- El proceso de validación muestra viabilidad en la utilización del modelo, sin embargo el carácter académico y teórico de las evaluaciones, no garantizan el éxito de la aplicación del modelo en un contexto puntual. De todas formas es claro que el conocer anticipadamente un proceso motiva la participación voluntaria de los interesados en un proyecto FLOSS.

4.2. Recomendaciones

- Ampliar la búsqueda de expertos para la aplicación del instrumento de evaluación del modelo, intentando recopilar la mayor cantidad de opiniones de personas con experiencias puntuales de participación en comunidades FLOSS.
- Aplicar el modelo propuesto en una comunidad nueva, promocionando la participación en torno a una aplicación de pequeño tamaño.

4.3. Trabajos Futuros

- Crear una librería de procesos documentados con SPEM, a partir de la participación en comunidades FLOSS exitosas.
- Realizar estudios comparativos de procesos en diferentes comunidades FLOSS.
- Crear una comunidad de desarrollo FLOSS entorno a la gestión académica y administrativas adecuado para las Universidades Colombianas.
- Adaptar el modelo a otros contextos como por ejemplo al trabajo colaborativo en ambientes virtuales e learning.

Bibliografía

Acuña, S., Castro, J., Dieste, O., and Juristo, N. (2012). A systematic mapping study on the open source software development process. In *IET Seminar Digest*, volume 2012, pages 42–46. 53

Acuña, S. T., S, A. B., Estero, S., and Ferré, X. (2000). Software Process Modelling. *Lecture Notes in Computer Science*, pages 237–242. 47

Agile Manifesto (2001). Principios del Manifiesto Ágil. 27, 44

Agrawal, K., Amreen, S., and Mockus, A. (2015). Commit quality in five high performance computing projects. In *Proceedings - 2015 International Workshop on Software Engineering for High Performance Computing in Science, SE4HPCS 2015*, pages 24–29. 50

Aksulu, A. and Wade, M. (2013). A multivariate classification of open source developers. *Information Sciences*, 221:72–83. 50

Al-Marzouq, M., Zheng, L., Rong, G., and Grover, V. (2005). Open Source: Concepts, Benefits, and Challenges. *CAIS*, 16:37. 54

AlMarzouq, M., Grover, V., and Thatcher, J. B. (2015). Taxing the development structure of open source communities: An information processing view. *Decision Support Systems*, 80:27–41. 50

Alshaikh, Z., Mostafa, S., Wang, X., and He, S. (2015). A empirical study on the status of software localization in open source projects. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, volume 2015-Janua, pages 692–695. 56

Alshakhouri, M., Buchan, J., and MacDonell, S. G. (2018). Synchronised visualisation of software process and product artefacts: Concept, design and prototype implementation. *Information and Software Technology*, 98:131–145. 48

Alvertis, I., Koussouris, S., Papaspyros, D., Arvanitakis, E., Mouzakitis, S., Franken, S., Kolvenbach, S., and Prinz, W. (2016). User Involvement in Software Development Processes. *Procedia Computer Science*, 97:73–83. 48

Alves, D. and De Souza Matos, E. (2017). Interaction design in free/libre/open source software development: A systematic mapping. In *ACM International Conference Proceeding Series*. 54

Androutsellis-Theotokis, S., Spinellis, D., Kechagia, M., and Gousios, G. (2011). Open Source Software: A Survey from 10, 000 Feet. *Foundations and Trends in Technology, Information and Operations Management*, 4:187–347. 55

Aversano, L., Guardabascio, D., and Tortorella, M. (2017). Evaluating the quality of the documentation of open source software. In *ENASE 2017 - Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 308–313. 54

Barbosa, O. and Alves, C. (2011). A Systematic Mapping Study on Software Ecosystems. *Proceedings of the Workshop on Software Ecosystems 2011*, pages 15–26. 4

Barcellini, F., Détienne, F., and Burkhardt, J.-M. (2007). Cross-Participants : fostering design-use mediation in an Open Source Software community. *International Journal of Industrial Ergonomics*, 39(August):28–31. 39, 52

Batikas, M., Riera, M. O. i., and Mezquita, E. A. (2011). *SME's participation to Free Libre Open Source Software Communities*. PhD thesis, Universitat Pompeu Fabra. 55

Bayona, S., Calvo-Manzano, J., Cuevas, G., and San Feliu, T. (2010). Taxonomía de factores críticos para el despliegue de procesos software. *Innovación, Calidad e Ingeniería del Software*, 6(3):6. 57

Bergquist, M., Ljungberg, J., and Rolandsson, B. (2012). Justifying the value of open source. In *ECIS 2012 - Proceedings of the 20th European Conference on Information Systems*. 55

Bermón-Angarita, L. (2010). *Librería de activos para la gestión del conocimiento sobre procesos software: PAL-Wiki*. PhD thesis, Carlos III de Madrid. 47, 66

Canfora, G. and Ruiz González, F. (2004). Procesos Software: características, tecnología y entornos. *Novática: Revista de la Asociación de Técnicos de Informática*, (171):5–8. 47

Castro Llanos, J. W. (2014). Incorporación de la usabilidad en el proceso de desarrollo open source software. pages 1–422. 53

Castro Llanos, J. W. and Acuña Castillo, S. T. (2012). Differences between traditional and open source development activities. *13th International Conference on Product-Focused Software Process Improvement, PROFES 2012*, 7343 LNCS:131–144. 53

Caulkins, J. P., Feichtinger, G., Grass, D., Hartl, R. F., Kort, P. M., and Seidl, A. (2013). When to make proprietary software open source. *Journal of Economic Dynamics and Control*, 37(6):1182–1194. 56

Chen, X. and Pan, Y.-H. (2013). The study of open source software collaborative user model based on social network and tag similarity. *Journal of Electronic Commerce Research*, 15(1):77–86. 53

Chung, E., Jensen, C., Yatani, K., Kuechler, V., and Truong, K. (2010). Sketching and drawing in the design of open source software. In *Proceedings - 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2010*, pages 195–202. 52

Cossentino, M., Hilaire, V., and Seidita, V. (2014). The openup process. In *Handbook on Agent-Oriented Design Processes*, pages 491–566. 28

Crowston, K., Lundell, I. H., Lindman, J., Lundell, B., and Robles, G. (2013). *Proceedings of the Doctoral Consortium at the 9th International Conference on Open Source Systems, 2013*. Number June. 53

da Silva, A. C. e. B. a. G., de Paula, G. d. F. C. A. C. M., Monteiro, M. P., and Abreu, F. B. e. (2017). Agility and quality attributes in open source software projects release practices. In *Proceedings - 2016 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016*, pages 107–112. 54

Daniel, S., Midha, V., Bhattacherhjee, A., and Singh, S. (2018). Sourcing knowledge in open source software projects: The impacts of internal and external social capital on project success. *The Journal of Strategic Information Systems*. 51

Daniel, S. and Stewart, K. (2016). Open source project success: Resource access, flow, and integration. *The Journal of Strategic Information Systems*, 25(3):159–176. 51

Dartsch, G. M. (2013). El software libre desde el punto de vista de las filosofías de la multiplicidad. *Question*, 1. 56

Dávila, A. (1995). *Las perspectivas metodológicas cualitativa y cuantitativa en las ciencias sociales*. Madrid. 6

Dong, L., Liu, B., Li, Z., Xue, B., Chen, D., and Chen, T. (2018). Mining Handover Process in Open Source Development: An Exploratory Study. In *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, volume 2017-Decem, pages 378–387. 54

Escribano Luis, Y. and Martínez, C. P. A. (2011). *Estado de la práctica de la Ingeniería de Requisitos en proyectos de Software Open Source*. PhD thesis, Universidad Politécnica de Cataluña, Barcelona, España. 39, 52

Ewenike, S., Benkhelifa, E., and Chibelushi, C. (2018). Cloud based collaborative software development: A review, gap analysis and future directions. In *Proceedings of IEEE/ACS*

International Conference on Computer Systems and Applications, AICCSA, volume 2017-Octob, pages 901–909. 48

Farah, G. and Correal, D. (2013). Analysis of intercrossed open-source software repositories data in GitHub. In *2013 8th Computing Colombian Conference, 8CCC 2013*. 50

Farias, M., Novais, R., Ortins, P., Colaço, M., and Mendonça, M. (2015). Analyzing distributions of emails and commits from OSS contributors through mining software repositories: An exploratory study. In *ICEIS 2015 - 17th International Conference on Enterprise Information Systems, Proceedings*, volume 2, pages 303–310. 50

Feitelson, D. G. (2012). Perpetual development: A model of the Linux kernel life cycle. *Journal of Systems and Software*, 85(4):859–875. 49

Feller, J., Fitzgerald, B., Hissam, S. A., and Lakhani, K. R. (2005). Perspectives on Free and Open Source Software. Technical report, Massachusetts Institute of Technology, London, England. 54

Fernandez, L. (2010). Innovación, Calidad e Ingeniería del Software. *https://www.ati.es/IMG/pdf/MinoliVol6Num3.pdf*, page 94. 38

Filippova, A. and Cho, H. (2016). The effects and antecedents of conflict in free and open source software development. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, volume 27, pages 705–716. 51

Forrester, E., Allen, J., Basili, V., Boehm, B., Cuevas, G., D ', M., Khaled, A., Emam, E., Ferber, S., Fusani, M., Garcia, S., Goodenough, J., Graettinger, C., Jeffrey, R., Jung, H.-W., Kitchenham, B., Konrad, M., Kugler, H.-J., Lawson, A., Oktaba, H., Penn, L., Peterson, W., Raffo, D., Rombach, D., Rout, T., Srivastava, N., Wilkie, G., and Wohlin, C. (2006). The International Process Research Consortium: A Process Research Framework. Technical report, SEI. 3, 4, 38, 47

Franco-Bedoya, O., Ameller, D., Costal, D., and Franch, X. (2017). Open source software ecosystems: A Systematic mapping. *Information and Software Technology*, 91:160–185. 56

Fuggetta, A. (2000). Software Process: A Roadmap. *Proceedings of the Conference on The Future of Software Engineering, International Conference on Software Engineering*, pages 25–34. 47

Fuggetta, A. (2003). Open source software—an evaluation. *Journal of Systems and Software*, 66(1):77–90. 54

Gamalielsson, J., Lundell, B., Feist, J., Gustavsson, T., and Landqvist, F. (2015). On organisational influences in software standards and their open source implementations. *Information and Software Technology*, 67:30–43. 50

Gerea, M. M., Sørensen, C.-F., and Conradi, R. (2007). *Selection of Open Source Components – A Qualitative Survey in Norwegian IT Industry.* PhD thesis, NTNU Norwegian University of Science and Technology. 48

Ghapanchi, A., Aurum, A., and Daneshgar, F. (2012). The impact of process effectiveness on user interest in contributing to the open source software projects. *Journal of Software*, 7(1):212–219. 49

Gonzalez-Barahona, J., Izquierdo-Cortazar, D., and Squire, M. (2010). Repositories with public data about software development. *International Journal of Open Source Software and Processes*, 2(2):1–13. 49

González-Barahona, J. M. and Koch, S. (2005). El Software Libre al microscopio. *Novatica*, 175. 4

Gopal, D., Lindberg, A., and Lyytinen, K. (2016). Attributes of open source software requirements - The effect of the external environment and internal social structure. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, volume 2016-March, pages 4982–4991. 54

Grottke, M., Karg, L., and Beckhaus, A. (2010). Team factors and failure processing efficiency: An exploratory study of closed and open source software development. In *Proceedings - International Computer Software and Applications Conference*, pages 188–197. 49

Hauge, Ø., Sørensen, C. F., and Røsdal, A. (2007). Surveying industrial roles in open source software development. *IFIP International Federation for Information Processing*, 234:259–264. 55

Hernando Ramírez, A. F. and Anne Marie, Z.-V. (2012). Metodología de la investigación : más que una receta Research Methodology : More than a recipe. (20):91–111. 5

Humphrey, W. S. (2000). The team software process (TSP). *Software Engineering Institute, CMU/SEI-*, (November):CMU/SEI–2000–TR–023. 21, 23

Humphrey, W. S. (2002). Personal Software Process (PSP). In *Encyclopedia of Software Engineering*. 21, 25

Huysmans, P., Ven, K., and Verelst, J. (2010). Using the DEMO methodology for modeling open source software development processes. *Information and Software Technology*, 52(6):656–671. 52

IEEE (2006). *IEEE 1074-2006: IEEE Standard for Developing a Software Project Life Cycle Process* , volume 2006. 19

ISO/IEC (2008). TR 15504-6:2008 – Information technology – Process Assessment – Part 6: An exemplar system life cycle process assessment model. 21

Joia, L. A. and dos Santos Vinhais, J. C. (2017). From closed source to open source software: Analysis of the migration process to Open Office. *The Journal of High Technology Management Research*, 28(2):261–272. 51

Jongyindee, A., Ohira, M., Ihara, A., and Matsumoto, K.-I. (2012). Good or bad committers? - A case study of committer's activities on the eclipse's bug fixing process. *IEICE Transactions on Information and Systems*, E95-D(9):2202–2210. 49

Jurado, F. and Rodriguez, P. (2015). Sentiment Analysis in monitoring software development processes: An exploratory case study on GitHub's project issues. *Journal of Systems and Software*, 104:82–89. 51

Keith, R. and Lawrence, C. (2015). PRINCE2® Agile. *Axelos*, page 351. 15

Khanjani, A. and Sulaiman, R. (2011). The aspects of choosing open source versus closed source. In *ISCI 2011 - 2011 IEEE Symposium on Computers and Informatics*, pages 646–649. 55

Kon, F., Meirelles, P., and Lago, N. (2011). Free and Open Source Software Development and Research: Opportunities for Software Engineering. *... Engineering (SBES) ...*. 55

Kumar, S. (2015). Using social network analysis to inform management of open source software development. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, volume 2015-March, pages 5154–5163. 50

Li, X. and Zhou, F. (2018). The Relationship Between Process Variability and Structural Connectivity in Open Source Software Development. In *ICIS 2017: Transforming Society with Digital Innovation*. 51

Linåker, J., Munir, H., Wnuk, K., and Mols, C. E. (2018). Motivating the contributions: An Open Innovation perspective on what to share as Open Source Software. *Journal of Systems and Software*, 135:17–36. 48

Ma, X., Zhou, M., and Riehle, D. (2013). How commercial involvement affects open source projects: Three case studies on issue reporting. *Science China Information Sciences*, 56(8):1–13. 50

Magdaleno, A. M., de Oliveira Barros, M., Werner, C. M. L., de Araujo, R. M., and Batista, C. F. A. (2015). Collaboration optimization in software process composition. *Journal of Systems and Software*, 103:452–466. 54

Mahmod, M. and Dahalin, Z. (2012). Women in open source software innovation process: Where are they? *Journal of Information and Communication Technology*, 11(1):113–129. 55

Manikas, K. and Hansen, K. M. (2013). Software ecosystems – A systematic literature review. *Journal of Systems and Software*, 86(5):1294–1306. 55

Margan, D. and Candrlic, S. (2015). The success of open source software: A review. *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1463–1468. 56

Matturro Mazoni, G. and Silva Vázquez, A. (2010). *Modelo para la gestión de conocimiento y la experiencia integrada a las prácticas y procesos de desarrollo de software*. PhD thesis, Universidad Politécnica de Madrid, Madrid. 47

Meirelles, P., Wen, M., Terceiro, A., Siqueira, R., Kanashiro, L., and Neri, H. (2017). Brazilian public software portal: An integrated platform for collaborative development. In *Proceedings of the 13th International Symposium on Open Collaboration, OpenSym 2017*. 56

Méndez, R. d. C., García, C. E., and Giarratana, M. (2008). *Management issues in open source software networks*. PhD thesis, Universidad Carlos III de Madrid Repositorio. 55

Midha, V. and Prashant, P. (2012). Factors Affecting the Success of Open Source Software. *Journal of Systems and Software*, 85(4):895–905. 50

Muruzábal, I. O. and Castillo Acuña, S. T. (2014). El proceso de requisitos en el desarrollo de Open Source Software. Technical report, Escuela Politecnica Superior, Universidad Autonoma de Madird, Madrid. 4, 39, 53

Mutafelija, B. and Stromberg, H. (2008). Iso 12207. In *Process Improvement with CMMI® v1.2 and ISO Standards*, pages 321–357. 19

Nelson, M. L., Sen, R., and Subramaniam, C. (2006). Understanding open source software: a research classification framework. *Communications of AIS*, 17(12):2–37. 55

Object Managemen Group (2008). Software & Systems Process Engineering Meta-Model Specification Verson 2.0. Technical Report April, Object Management Group. 4

Okoli, C. and Carillo, K. (2013). Beyond open source software: Framework and implications for open content research. In *ECIS 2013 - Proceedings of the 21st European Conference on Information Systems*. 56

Oliveros, A. and Aguilera, S. (2012). Proyectos de Desarrollo de Software en ambientes cooperativos y colaborativos. 53

PMI (2014). *A Guide to the Project Management Body of Knowledge: PMBOK (®) Guide*. 5th edition. 15

PMI (2017). *PMBOK Guide - 6th Edition*, volume 40. 15

Poo-Caamaño, G., Knauss, E., Singer, L., and German, D. (2017). Herding cats in a FOSS ecosystem: a tale of communication and coordination for release management. *Journal of Internet Services and Applications*, 8(1). 51

Potdar, V. and Chang, E. (2004). Open source and closed source software development methodologies. *Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering - 26th International Conference on Software Engineering*, pages 105–109. 52

Raymond, E. S. (1997). The Cathedral and the Bazaar. (July 1997):1–35. 39, 52

Rigby, P. C., German, D. M., and Storey, M.-A. (2008). Open source software peer review practices: a case study of the apache server. *Proceedings of the 30th international conference on Software engineering*, pages 541–550. 49

Robles, G. and González-Barahona, J. M. (2012). A comprehensive study of software forks: Dates, reasons and outcomes. *IFIP Advances in Information and Communication Technology*, 378 AICT:1–14. 49

Rolandsson, B., Bergquist, M., and Ljungberg, J. (2011). Open source in the firm: Opening up professional practices of software development. *Research Policy*, 40(4):576–587. 47

Rossi, M. (2004). Decoding the " Free / Open Source (F / OSS) Software Puzzle " a survey of theoretical and empirical contributions. *The Economics of Open Source Software Development*, (May 2004):40. 54

Ruiz Rube, I. and Dodero Beardo, J. M. (2013). *Un framework para el despliegue y evaluación de procesos software*. PhD thesis, Universidad de Cádiz. 48

Ruiz-Rube, I. and Dodero Beardo, J. M. (2014). *Un framework para el despliegue y evaluación de procesos software*. PhD thesis. 48

Sahu, T., Nagwani, N., and Verma, S. (2016). An empirical analysis on reducing open source software development tasks using stack overflow. *Indian Journal of Science and Technology*, 9(21). 51

Scacchi, W. (2010). The future of research in Free/Open Source Software development. In *Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010*, pages 315–319. 55

Schwaber Ken, S. J. (2017). The Scrum Guide — Scrum Guides. 15, 29

Shaikh, M. and Henfridsson, O. (2017). Governing open source software through coordination processes. *Information and Organization*, 27(2):116–135. 56

Shimagaki, J., Kamei, Y., McIntosh, S., Pursehouse, D., and Ubayashi, N. (2017). Why are commits being reverted? A comparative study of industrial and open source projects. In *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016*, pages 301–311. 51

Siau, K. and Tian, Y. (2013). Open source software development process model–a grounded theory approach. In *19th Americas Conference on Information Systems, AMCIS 2013 - Hyperconnected World: Anything, Anywhere, Anytime*, volume 4, pages 2618–2628. 53

Sillitti, A. and Succi, G. (2005). Agilidad y desarrollo de Software Libre. *Novatica*, 175. 4, 28

Sojer, M. and Henkel, J. (2010). Code reuse in open source software development: Quantitative evidence, drivers, and impediments. *Journal of the Association of Information Systems*, 11(12):868–901. 49

Spinellis, D., Gousios, G., Karakoidas, V., Louridas, P., Adams, P. J., Samoladas, I., and Stamelos, I. (2009). Evaluating the Quality of Open Source Software. *Electronic Notes in Theoretical Computer Science*, 233(C):5–28. 52

Stallman, R. M., Behlendorf, B., Bradner, S., Hamerly, J., Mckusick, K., Tim, O., Paquin, T., Perens, B., Raymond, E., Stallman, R., Tiemann, M., and Others (1999). Open Sources: Voices from the Open Source Revolution. 4, 10, 54

Team, C. P. (2002a). CMMI for Software Engineering, Version 1.1, Continuous Representation (CMMI-SW, V1.1, Continuous). Technical Report CMU/SEI-2002-TR-028, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 3, 21, 23

Team, C. P. (2002b). CMMI for Software Engineering, Version 1.1, Staged Representation (CMMI-SW, V1.1, Staged). Technical Report CMU/SEI-2002-TR-029, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 3, 21, 23

Wang, J. and Carroll, J. (2011). Beyond fixing bugs: Case studies of creative collaboration in open source software bug fixing processes. In *C and C 2011 - Proceedings of the 8th ACM Conference on Creativity and Cognition*, pages 397–398. 53

Wang, J., Shih, P. C., Wu, Y., and Carroll, J. M. (2015). Comparative case studies of open source software peer review practices. *Information and Software Technology*, 67:1–12. 54

Wei, K., Crowston, K., Eseryel, U., and Heckman, R. (2017). Roles and politeness behavior in community-based free/libre open source software development. *Information & Management*, 54(5):573–582. 51

Wu, J., Goh, K.-Y., and Tang, Q. C. (2007). Investigating Success of Open Source Software Projects: A Social Network Perspective. In *ICIS*. 49

Xi, H., Xu, D., and Son, Y.-J. (2011). Dynamic scheduling and workforce assignment in open source software development. In *61st Annual IIE Conference and Expo Proceedings*. 52

Xu, B., Lin, Z., and Xu, Y. (2011). A study of open source software development from control Perspective. *Journal of Database Management*, 22(1):26–42. 53

Yamakami, T. (2010). A stage model of evolution of open source software: Implications for the next stage of open source software development. In *Proceeding - 6th International*

Conference on Digital Content, Multimedia Technology and Its Applications, IDC2010, pages 203–207. 55

Yamakami, T. (2011). A two-dimensional classification model of OSS: Towards successful management of the evolution of OSS. In *International Conference on Advanced Communication Technology, ICACT*, pages 1336–1341. 55

Yan, J. and Wang, X. (2013). From open source to commercial software development - the community based software development model. In *International Conference on Information Systems (ICIS 2013): Reshaping Society Through Information Systems Design*, volume 2, pages 1717–1736. 55

Zaimi, A., Ampatzoglou, A., Triantafyllidou, N., Chatzigeorgiou, A., Mavridis, A., Chaikalis, T., Deligiannis, I., Sfetsos, P., and Stamelos, I. (2015). An empirical study on the reuse of third-party libraries in open-source software development. In *ACM International Conference Proceeding Series*, volume 02-04-Sept. 50

Apéndice A

Documento síntesis presentado a los expertos para evaluar el modelo propuesto

Modelo de Despliegue de procesos para comunidades FLOSS

Trabajo de Maestría
Mastria en Gestión de proyectos Informáticos
Universidad de Pamplona
Colombia

Resumen

La sigla FLOSS (Free-Libre / Open Source Software) se ha usado para unificar los movimientos de desarrollo de software en comunidades de voluntarios distribuidos geográficamente, y para lo cual la disposición del código fuente es la esencia del proceso. Es así como esta expresión une tanto los movimientos de software libre como de open source.

Estas comunidades dependen en su mayor parte del voluntariado de cientos de personas interesadas en un dominio particular y por ello es importante que el proceso de desarrollo de software sea lo más simple y claro, pero suficiente para garantizar la vinculación de nuevos voluntarios y evitar la desmotivación de quienes participan activamente en la comunidad.

Este documento presenta una síntesis de un modelo de despliegue de procesos de desarrollo software en comunidades FLOSS, para lo cual se hace necesario poner en contexto las siguientes definiciones:

- **Proceso software:** Conjunto de actividades, métodos, prácticas, herramientas, guias y transformaciones que las personas organizadas en roles, usan para desarrollar una aplicación software y los productos asociados a este tales como planes de proyecto, diseños, especificaciones, código fuente, pruebas, manuales de usuario, etc.
- **Despliegue de procesos**: Actividades relacionadas con la puesta en funcionamiento de un proceso, incluyendo la socialización con quienes participan en el proceso, y el seguimiento a la forma de funcionamiento.
- **Modelo**:Cosa que sirve de patrón, guía o pauta para ser imitada, reproducida o copiada.
- **SPEM**: Software Systems Process Engineering Meta-Model Specification, es un "metamodelo" que se usa para definir y documentar procesos de desarrollo de software, mediante una notación gráfica como esquema estandarizado de describir procesos, basado en los siguientes conceptos básicos.
  1. **Roles**: Conjunto de habilidades, competencias y responsabilidades (define el "quién").
  2. **Tareas y Actividades**: Unidad de trabajo que se asigna a un rol para lograr un resultado significativo (definen el "cómo").
  3. **Productos de Trabajo o artefactos**: Resultado significativo de un proceso (define el "qué").
  4. **Herramientas**: Técnica o instrumento que se utiliza para elaborar un artefacto (define el "Con qué").
  5. **Procesos**: Estructuras de trabajo secuenciales que deben realizarse para desarrollar un software, normalmente compuestos por diversos flujos de trabajo y fases (define el "Cuándo").

1. Modelo propuesto

En el contexto de comunidades FLOSS, el despliegue de un proceso de software debe traspasar las propias fronteras del concepto de despliegue es decir las fases de Despliegue inicial y mejoramiento continuo y por lo tanto el modelo propuesto involucra la fase de diseño, pues esta debe garantizar la usabilidad de los procesos definidos para una comunidad en particular.

La figura 1 el modelo propuesto plantea tres componentes (restricciones, técnicas y herramientas) transversales al ciclo de vida de un proceso y dirigido por un grupo de personas representado mediante el rol "grupo de proceso"dentro de una comunidad FLOSS probablemente, inexistente cuando un proyecto se inicia con tan solo una versión incial conocida como "Promesa Creíble".

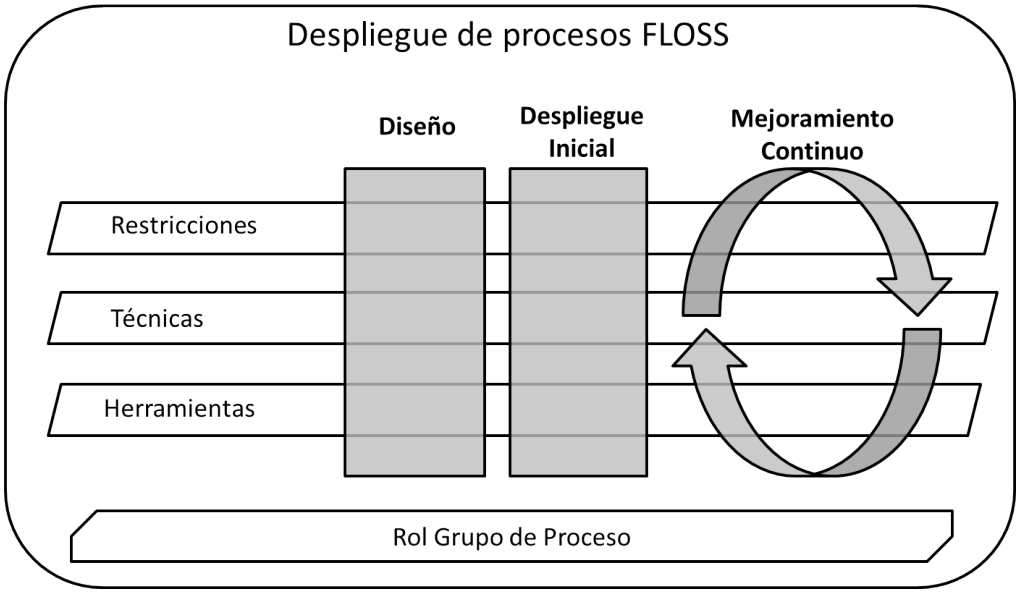

Figura 1: Modelo de despliegue Propuesto

Como todo modelo presenta generalidades que cada comunidad FLOSS que lo utilice, adaptará a sus propias particularidades, por lo que en este documento se muestran ejemplos de restricciones, técnicas y herramientas que pueden ser parte del despliegue de procesos en una comunidad, mas que exigencia como de obligatoriedad al usar el modelo propuesto.

2. Diseño

Un proceso de software debe ser visto de manera general como todo el proceso de desarrollo que pude ser subdivido en subprocesos, por lo tanto, cuando se habla de despliegue inicial del proceso se considera por primera vez el proceso integral cuyo único producto es un software funcional.

La versión inicial de un producto FLOSS, conocida como "Promesa Creíble", es la principal motivante de la vinculación de miembros a un proyecto, y por lo tanto esta versión inicial no aplicó un proceso de desarrollo comunitario, siendo así el diseño del proceso una tarea paralela a la vinculación de los primeros participantes.

Es ahí donde se requiere la conformación del grupo de proceso [1], como un rol, en la cual sus integrantes diseñarán, documentarán y dan soporte permanente a todos los demás miembros en los diferentes roles que se determinen.

- **Restricciones:** Caracterizar la población que va a usar el proceso[2], iterativo e incremental[3], diseño basado en roles y no en subprocesos[4], Flexiblilidad[5] y Simplicidad[6]
- **Técnicas:** Las Librerías de Activos de Procesos -Process Asset Library PAL-, para lo cual se debe promover repositorios de procesos de libre acceso. Al igual que el código fuente está disponible en un producto software, los procesos de las comunidades FLOSS deben estar adecuadamente documentados y de libre acceso.

[1] Que puede ser considerado un rol más dentro del proceso global de desarrollo.

[2] Que si bien no esta definida claramente al inicio del proyecto, se puede predecir dependiendo del dominio del problema que aborda el proyecto, esta caracterización debe incluir su distribución geográfica.

[3] Debido a las frecuentes liberaciones que caracterizan a una comunidad FLOSS.

[4] La gran variedad de perfiles e intereses de los miembros de una comunidad FLOSS, estos se autorganizan según sus habilidades por lo tanto su vinculación siempre se hace en función del rol que desempeña.

[5] Por lo que el diseño inicial del proceso debe incluir aspectos que faciliten los cambios, sin sufrir traumas al interior de los participantes en el rol afectado por el cambio. Es así como esto podría denominarse mantenibilidad del proceso.

[6] Basados en la idea que algo es simple cuando no hay nada que quitar, por lo tanto los procesos deben ser mínimos necesarios para lograr la participación del mayor numero de interesados.

- **Herramientas:** Eclipse Process Framework Composer -EPF composer-, mediante la cual se puede especificar y documentar cada uno de los roles las actividades propias de su función, los artefactos generados en dichas actividades y las herramientas utilizadas.

3. Despliegue Inicial

Esta componente incluye la publicación de la documentación del proceso, la capacitación de los miembros del equipo, y la instalación e integración de las herramientas que soportan las actividades para cada uno de los roles definidos en el proceso previamente diseñado.

- **Restricciones** Legibilidad del proceso[7] e Integración de herramientas[8].
- **Técnicas:**Desvío del flujo de participaciones de los usuarios más expertos a una instancia del nuevo proceso[9] y capacitación por diferentes medios[10].
- **Herramientas:**Plataformas de desarrollo colaborativo[11], comunicación entre miembros del equipo[12], los manuales técnicos[13], los manuales de usuario y cualquier otra documentación que facilitan la modificación[14], las Wikis[15].

4. Mejoramiento Continuo

El componente se basa en el manejo de los cambios del proceso a partir de procesos de medición, que detectan partes de un proceso que puede mejorar.

- **Restricciones:** Disponibilidad de metadatos de los commit[16], facilidad para la publicación de información de los commit[17].
- **Técnicas:** Caracterización de defectos[18] y análisis de commit revertidos[19].

[7] Es decir la facilidad con que cada miembro del equipo comprenda las actividades propias de su rol y el uso de las herramientas para la construcción de los artefactos bajo su responsabilidad.

[8] Los repositorios y el control de versiones se han constituido en la principal herramienta para el trabajo en comunidades FLOSS, sin embargo se requieren otras herramientas como las de comunicación, foros, reporte de errores, pruebas,herramientas de documentación, que deben ser adecuadamente integradas para buscar que los miembros de una comunidad, consideren un solo contexto, de tal manera que con un solo login puedan acceder a las herramientas involucradas en un rol determinado.

[9] De tal manera que se pueda considerar una prueba beta del proceso y con su retroalimentación se hacen los ajuste pertinentes para dar en servicio al 100 por ciento de los participantes.

[10] Esta se constituye en la principal técnica para la comprensión de las actividades, herramientas y técnicas usadas por cada miembro dentro de un rol determinado, es así como el grupo de procesos debe disponer de las estrategias correspondientes para asesorar principalmente en el uso de las herramientas.

[11] También conocidas como forjas son principalmente herramientas de control de versiones, en la cual se registran todo tipo de documentos digitales, por lo que en estas se puede llevar control no solo del código fuente, sino de documentación de requerimientos, diseños, pruebas, manuales y en general todo tipo de artefacto en un proceso de desarrollo software.

[12] Las listas de correos y foros se constituyen en la principal herramienta para comunicación entre los miembros de las comunidades FLOSS. Por medio de estas se comunican nuevos requerimientos, se discuten alternativas de solución y se emprenden procesos de codificación y pruebas. dentro de este tipo de herramientas también se pueden incluir las de reporte y seguimiento de bugs y de documentación.

[13] Como descripción de diseños, requerimientos, paquetes, clases, código fuente, los códigos de error y sus significados, entre otra información de utilidad para comprender como esta construida una aplicación.

[14] Como el código documentado que garantice el uso y operación de un producto software a largo plazo.

[15] Probablemente este tipo de herramientas son las mas versátiles para el contexto FLOSS, pues con ellas se podría soportar procesos de documentación para usuario, documentación del mismo proceso, requerimientos, diseños e incluso en forma de portal de promoción de un proyecto FLOSS.

[16] Consolidar, confirmar, registrar un cambio o hacer un commit se refiere a confirmar un conjunto de cambios provisionales de forma permanente, en uno o varios artefactos constitutivos de un producto software y sobre estos se hace necesario realizar los análisis pertinentes en búsqueda de aspectos en el proceso que puedan ser susceptibles de mejoras.

[17] Esta información debe estar disponible al interior de una comunidad de manera ilimitada para el grupo de procesos, sin embargo puede ser más muy útil al estar disponible para cualquier persona mediante repositorios como SourceForge Research Data Archive, FLOSSmole, FLOSSMetrics.

[18] Vista como una oportunidad para encontrar oportunidades de mejora, la caracterización de defectos permite encontrar la causa raíz de un los fallos detectados en los procesos de prueba.

[19] Un volumen de los aportes en una comunidad no son incorporados al producto por diferente motivos, y el análisis de esta información permite la mejora de procesos.

- **Herramientas:**Análisis de datos[20].

5. Rol de Ingeniero de procesos

Este rol dentro de una comunidad FLOSS tiene como misión permanente el identificar, analizar y proponer soluciones a ineficiencias del proceso de desarrollo, desempeñando como principales funciones:

- Hacer análisis permanente sobre los commit para identificar aspectos de mejora en el proceso.
- Ayudar a los miembros del equipo de trabajo con el manejo de las herramientas en cada uno de los roles definidos.
- Ayudar a entender y adoptar el proceso definido para la comunidad.
- Gestionar los cambios del proceso, incluyendo la integración de herramientas y capacitación a los miembros de la comunidad.

Dentro de las capacidades de las personas que participen en este rol se deben encontrar: Ser perseverante y organizado, conocedor de procesos de desarrollo software, con capacidades de auto-aprendizaje [21], el respeto, el compromiso, la coherencia, trabajar en equipo, ser comunicativo [22] y asertivo [23] y finalmente debe tener desarrollar habilidades especiales para la resolución de conflictos[24].

[20] Las herramientas estadísticas y de inteligencia artificial que permitan el tratamiento de grandes volúmenes de datos permiten encontrar patrones de comportamiento, con los cuales se pueda identificar oportunidades de mejora de un proceso.

[21] La capacidad de aprender por uno mismo incluye entre otras cosas, la experimentación y el aprender de los errores.

[22] Saber transmitir la esencia de los proceso y persuadir a los miembros para ver las ventajas en el uso del proceso.

[23] Debe ser capaz de ponerse en el rol de cada uno de los grupos con los que debe comunicarse, y construir un ambiente de interacción apropiado entre los miembros de una comunidad.

[24] Probablemente en grandes comunidades es el aspecto inevitablemente que siempre surge cuando hay personas con diferentes intereses.

Apéndice B

Instrumento Aplicado para la Validación

Evaluación del Modelo de Despliegue de Procesos Software en Comunidades FLOSS
Trabajo de Grado en la Maestría en Gestión de Proyectos Informáticos
Universidad de Pamplona, Diciembre de 2019

- **Objetivo:** Evaluar un modelo propuesto para el despliegue de procesos software en comunidades FLOSS.
- **Instrucciones Generales:** Leer el documento síntesis del modelo a evaluar antes de responder a las siguientes preguntas

---

Datos del participante

---

1. Nombre Completo: _____
2. Profesión actualmente desempeñada: _____
3. Máximo **nivel de Formación Académica** obtenido hasta el momento
   - ◯ Bachiller
   - ◯ Técnico y/o tecnológico
   - ◯ Pregrado Profesional
   - ◯ Posgrado Especialización
   - ◯ Posgrado Maestría
   - ◯ Doctorado
4. En que grado conoce el funcionamiento de una comunidad FLOSS?
   - ◯ Bajo
   - ◯ Medio
   - ◯ Alto
   - ◯ Muy Alto
5. Ha participado en una comunidad FLOSS?
   - ◯ Si
   - ◯ No

   En caso afirmativo responder:
   - En cuál o cuáles comunidades ha participado? _____
     _____
   - Qué rol o roles ha desempeñado?_____
   - Desde que año participa en este tipo de comunidades?_____

Sobre el Modelo a Evaluar.

---

1. La utilización del modelo propuesto Motivará la vinculación de miembros novatos en una comunidad FLOSS
   - ◯ Totalmente en desacuerdo
   - ◯ En desacuerdo
   - ◯ Ni de acuerdo ni en desacuerdo
   - ◯ De acuerdo
   - ◯ Totalmente de acuerdo

2. La utilización del modelo propuesto Motivará la participación de miembros que tienen experiencias previas en comunidades FLOSS.
   - ◯ Totalmente en desacuerdo
   - ◯ En desacuerdo
   - ◯ Ni de acuerdo ni en desacuerdo
   - ◯ De acuerdo
   - ◯ Totalmente de acuerdo

3. La formalización de un proceso mediante SPEM, como propone el modelo, creará inconformidad en los miembros con experiencia en el trabajo de equipos FLOSS.
   - ◯ Totalmente en desacuerdo
   - ◯ En desacuerdo
   - ◯ Ni de acuerdo ni en desacuerdo
   - ◯ De acuerdo
   - ◯ Totalmente de acuerdo

4. Un procesos crítico al momento de trabajar con un equipo virtual, es el de comunicaciones. El modelo propuesto facilita el proceso de comunicación en una comunidad FLOSS.
   - ◯ Totalmente en desacuerdo
   - ◯ En desacuerdo
   - ◯ Ni de acuerdo ni en desacuerdo
   - ◯ De acuerdo
   - ◯ Totalmente de acuerdo

5. Favor dejar aquí las observaciones y comentarios que considere pertinentes respecto al modelo evaluado