Procedimiento para la gestión de pruebas funcionales de software web en ambientes de desarrollo colaborativo y distribuido.

Diego Armando Aguilar Sanguino

Universidad de Pamplona

Facultad de Ingenierías y Arquitectura

Programa de Ingeniería de Sistemas

Pamplona, Norte de Santander

2016

Procedimiento para la gestión de pruebas funcionales de software web en ambientes de desarrollo colaborativo y distribuido.

Diego Armando Aguilar Sanguino

Trabajo de grado presentado como requisito para optar al título de

INGENIERO DE SISTEMAS

Director: Luis Alberto Esteban Villamizar

Licenciado en Matemáticas y Computación

Mg. en Informática

lesteban@unipamplona.edu.co

Universidad de Pamplona

Pamplona, Norte de Santander

2016

Tabla de contenido

Lista de tablas

Lista de figuras

1 Introducción

La necesidad de producir un software que sea de utilidad común para muchas organizaciones como es el caso de un software para la gestión académica y administrativa de instituciones educativas, se puede abordar de diversas maneras: tal como se viene haciendo, cada Universidad invierte recursos de su propios para el desarrollo de una aplicación a su medida (adecuada a sus procesos), algunas Universidades deciden comprar un software licenciado y adaptar sus procesos al software, una alternativa podría ser participar en la creación de una comunidad que desarrolle un software bajo conceptos Open Source, en donde todas las Universidades pudiesen participar, obteniendo un producto genérico de fácil adaptación a cada uno de los contextos establecidos por cada una de las Universidades participantes.

Dado que un proceso de software está constituido de muchos subprocesos, este trabajo se enfoca en el subproceso de pruebas de software, por lo que se desarrolló un procedimiento apoyados en el software de Bugzilla junto a su extensión de Testopia para gestionar las pruebas de software y llevar un seguimiento de las fallas que se presenten en éste.

El procedimiento está basado en la experiencia obtenida de diferentes autores en cuanto a la participación en comunidades de Open Source y de la colaboración en el desarrollo de Academusoft 4.0 el cual pertenece a uno de los productos de software de la Universidad de Pamplona, el fin de este procedimiento es otorgarle a la comunidad educativa libre de la Universidad un medio para mejorar y aliviar la fatal de ésta en el desarrollos de software.

1.1 Formulación del problema

La Universidad de Pamplona cuenta con una dependencia dedicada al desarrollo de sus propias herramientas para la gestión administrativa y académica de la misma Universidad, sin embargo dichas aplicaciones no han sido liberadas como software libre, por lo que sus procesos son definidos y aplicados al interior de la organización sin permitir la participación de personas externas. Esta dependencia ha desarrollado cinco productos software basados en web, siendo el

más representativo entre ellos Academusoft el cual es usado por alrededor de 46 Universidades que adquirieron su licencia.

La Universidad de Pamplona no cuenta con una comunidad Open Source formal dedica al desarrollo de productos software basados en web, esto se debe al desconocimiento que existe en la comunidad educativa sobre los procesos para la interacción en comunidades Open Source o de software libre, tal desconocimiento se ha convertido en el principal obstáculo en la promoción del uso y desarrollo de software web en ambientes colaborativos (la participación de los desarrolladores es voluntaria y no remunerada económicamente) y distribuidos (los desarrolladores no tiene un espacio físico de reunión) típicos del software libre.

La creación y dirección de por lo menos una comunidad Open Source, al interior de la Universidad de Pamplona, que sirva de laboratorio se ha convertido en una necesidad prioritaria para los procesos de desarrollo de software web, dado que la participación en proyectos Open Source son un reflejo de estándares de buenas prácticas, esto quiere decir que el desarrollo del producto software será más limpio y rápido, además hay que destacar que las comunidades Open Source son sinónimos de innovación y ésta es una característica fundamental para un ente como una Universidad.

Es por ello que este trabajo presenta una propuesta que mitigue la falta de involucramiento de la comunidad educativa de la Universidad de Pamplona en proyectos de software Open Source, por lo cual el proyecto está enfocado en unos de los pilares que implica un desarrollo de producto software, la gestión de pruebas, específicamente en las pruebas funcionales utilizando una herramienta de código libre para tal fin.

Para lograr este cometido se definieron las siguientes preguntas de investigación:

¿Es factible ejecutar un procedimiento para la gestión de pruebas en proyectos de software desarrollados en ambientes colaborativos y distribuidos?

¿Garantiza un proyecto de código abierto la calidad del producto software al no seguir las prácticas clásicas de desarrollo de software?

¿Permiten los entornos socioculturales y académicos de la Universidad de Pamplona participar

en proyectos de código abierto dadas las características filosóficas de estas comunidades?

1.2 Objetivos

1.2.1 Objetivo general

Diseñar un procedimiento para la gestión de pruebas funcionales de software web en ambientes de desarrollo colaborativos y distribuidos.

1.2.2 Objetivos específicos

- Realizar una consulta bibliográfica sobre el funcionamiento del rol de ingeniero de pruebas en proyectos Open Source.
- Proponer un procedimiento para la gestión de pruebas en proyectos Open Source para la comunidad libre de la Universidad de Pamplona.
- Validar el procedimiento propuesto mediante un ejemplo de aplicación en un ambiente simulado.

1.3 Justificación

Los proyectos de software Open Source son sin duda sinónimo de productos de calidad y versatilidad, esto se debe a que son los involucrados los que identifican las nuevas funcionalidades que debe llevar el software y al tener a disposición el código pueden modificarlo a sus necesidades, compartirlo o tener una solución comercial sin tener que empezar el proyecto desde cero, es decir, evitar reinventar la rueda, lo cual es muy significativo y deja una experiencia alcanzable por parte de un equipo que apenas comienza a desarrollar un producto de software. Además este tipo de proyectos implica a que el producto software que se desarrolló sea más transparente y que los fallos que se presentan puedan ser corregidos con mayor rapidez a comparación de un software tradicional.

Aun así, tanto los productos Open Source como el de código privado no se excusan de fallos, Facebook, Firefox, Gmail, Open Office, Windows e incluso uno de los productos de departamento de gestión y desarrollo tecnológico de la vice-rectoría de la Universidad de Pamplona puede no

estar exento de alguna anomalía en su desempeño, luego se necesitan actividades que prueben que el software haga lo que debe hacer o que no haga lo que no debe hacer, es allí donde la gestión de pruebas participa en el desarrollo, evitando que el producto software no se vea afectado en su desempeño y calidad.

Ahora, ¿es correcto hablar de una actividad como la de gestión de prueba en un desarrollo de software Open Source?, antes de resolver esta inquietud hay que plantearse si el desarrollo de software en comunidades Open Source difieren de los procesos de desarrollo tradicional, esta cuestión ha sido abordada por muchos autores, luego se encontró a (Acuña Castillo, Castro, & Dieste, 2012) quienes plantean que existen muchas diferencias en los procesos de desarrollo en los OSS con respecto al desarrollo tradicional, de hecho ha llegado a la conclusión de que "La comunidad OSS no sigue los modelos y estándares prescriptivos de la ingeniería del software tradicional", aun así, se encuentran a otros autores tales como (Fuggetta, 2003) el cual defiende la idea de que el tipo de desarrollo en comunidades OSS es solo una nueva perspectiva de las actividades de la ingeniería del software a la hora de desarrollar un producto software y no son una nueva descripción del proceso, de hecho (Fuggetta, 2003) plantea que el desarrollo de metodologías ágiles puede ser aplicado tanto a software de código privado como al de código abierto.

Al analizar el desarrollo de metodologías ágiles se pueden encontrar muchas similitudes con los proyectos Open Source, dentro de ellas se puede apreciar el desarrollo incremental, entregas tempranas y frecuentes, el producto software es constantemente testeado y la más significativa es la cooperación entre en el desarrollador y el cliente. Pero, es factible involucrar una actividad como la de gestión de pruebas en un proyecto OSS, se sabe que en este tipo de comunidades normalmente los desarrolladores realizan pruebas a su propio código antes de subirlo a los repositorios donde se encuentra el proyecto, pero, ¿existen pruebas formales en los desarrollos de estos ambientes?, pues el autor (Crowston, Howison, & Wiggins, 2010) describe de que si existe un proceso de pruebas formales a la hora de desarrollar en ambientes colaborativos y distribuidos, de hecho (Dinh-Trong & Bieman, 2004) realizaron una investigación al proyecto FreeBSD donde describe un proceso formal a la hora de hacer pruebas para que un fragmento de código haga parte del desarrollo de FreeBSD. Por lo tanto, implementar un procedimiento para el desarrollo de software web en comunidades OSS cuenta con las bases suficientes para poder implantarse en la comunidad Open Source de la Universidad de Pamplona.

1.4 Metodología

La metodología aplicada cuenta en su primera etapa con una investigación de modalidad documental, es por ello que se tomarán experiencias de diferentes autores en el involucramiento de desarrollos de software en comunidades de trabajo colaborativo y distribuido, ya sean Open Source o de software libre, esto es con el deseo de analizar los procedimientos que existen en dichas comunidades, específicamente en la fase de pruebas y entender el funcionamiento de esta fase en dichas comunidades a la hora de desarrollar software.

Para encontrar dichas experiencias se tomó proyectos de trabajo de grado realizando búsquedas en la base de datos de google académico, seleccionando los más relevantes y actuales con respecto a esta alternativa de desarrollo de software (OSS), además se participó en la nueva versión de unos de los productos software de la Universidad de Pamplona, este es Academusoft 4.0, la participación consistió en testear las funcionalidades de una nueva versión, la cual está a la fecha en desarrollo.

La segunda etapa consistió en realizar un análisis de los proyectos y realizar un procedimiento en el cual se especificó una serie de pasos para garantizar la gestión de pruebas del producto software en el que se quiera participar, el procedimiento está constituido por tres fases, la fase 1 radicó en la especificación de las herramientas que se utilizaron, su instalación y configuración. La fase 2 radicó en los procesos para garantizar la gestión de pruebas, es decir, aspectos tales como registrar un bug, cambiar el estado de éste, registrar planes de pruebas, casos de pruebas, etc. Por último, la fase 3 consiste en el mantenimiento del procedimiento, es decir, realizar backups.

Por último la etapa 3 cuenta con la validación del procedimiento planteado, esto se llevó a cabo en un ambiente que simuló un entorno de desarrollo colaborativo y distribuido, en el cual se instalaron herramientas, se configuraron y en general se realizó gestión de pruebas de un producto software llamado cursosLibres, este consiste en la oferta de cursos ofrecidos por Min TIC.

La comprobación del procedimiento consistió en que los procesos de pruebas reflejaron las descripciones de las experiencias en las participaciones de los diferentes autores en los ambientes de desarrollo colaborativo y distribuido, además, que cumplieron con el estándar IEEE 829 de 1998 los planes de pruebas y casos de prueba.

2 Marco teórico

En este capítulo se abordan las ideas acerca de las investigaciones realizadas en las comunidades Open Source y Software libre, se habla un poco del origen de estos movimientos, que aun compartiendo similitudes en los procesos a la hora de desarrollar software, sus conceptos filosóficos del porque comparten el código fuente difieren por completo, además se habla sobre las herramientas de software muy utilizadas al momento de llevar un control de las fallas producidas en el software, estas son, Bugzilla junto con su extensión para gestionar pruebas de software funcionales, Testopia.

Por último se define un marco conceptual para que el lector pueda comprender los términos utilizados en el marco teórico y un marco legal el cual da una claridad de como la legislación de Colombia protege los productos software con licenciamiento GPL y sus parecidos.

2.1 Antecedentes

Un primer trabajo de investigación revisado corresponde al de (Bartolomé, 2014) quien realizó el "Desarrollo de Software Open Source Analizado desde Dentro", En este trabajo se estudiaron los diferentes procesos en el desarrollo de software en comunidades colaborativas y distribuidas para realizar una comparación con los procesos en el desarrollo tradicional de software. La técnica de la investigación consistió en participar tanto en desarrollos de software de código abierto y de código privado, para ello hubo una vinculación en la empresa Accenture realizando tareas en diferentes áreas del desarrollo del producto software de la compañía, por otra parte, hubo una participación en cinco proyectos de software de código abierto, los cuales son Doublé Commander, FileZilla, VLC Media Player, GanttProject y Social Network Manager, al igual que en la empresa Accenture hubo participación en los diferentes procesos de desarrollo lo cual llevó a la investigación a concluir cuales son los puntos fuertes y débiles en cada uno de los procesos de desarrollo en los diferentes ambientes de desarrollo.

Esta investigación se relaciona con el presente trabajo ya que en su material expone cómo se refleja el proceso de pruebas tanto en proyectos de código cerrado como de código libre y señala en la investigación las diferencias de cómo es percibida esta vital fase para un producto software,

mostrando fortalezas a la hora de gestionar pruebas en proyectos de código abierto.

Un segundo trabajo de investigación que se revisó es el de (Ishigetani, 2014) quien realizo:" El Proceso De Requisitos en el Desarrollo de Open Source Software", en esta investigación se hace énfasis en proceso de requerimientos en las comunidades de desarrollo Open Source siendo su objetivo determinar cómo se llevan a cabo los procesos y actividades relacionadas con la Ingeniería de Requisitos en un conjunto de comunidades OSS. La metodología aplicada en dicha investigación consistió en participar en varios proyectos Open Source y documentar como es el proceso de requerimientos en este tipo de comunidades de desarrollo, además en la investigación se describen los roles que existen en la participación con estas comunidades. La investigación concluye que los objetivos de los procesos tradicionales y OSS son los mismos, pero difieren completamente en las técnicas, herramientas e infraestructuras para alcanzar estos objetivos. El trabajo se relaciona con la investigación planteada, ya que en él se muestran los diferentes tipos de roles que existen en las comunidades Open Source, allí se observa que existe un rol el cual está a cargo del control sobre los fallos que presenta el producto software, otro rol que describe en el proyecto de Mozilla es el rol de Propietarios de componentes de Bugzilla, los cuales tiene a cargo revisar los informes de errores regularmente, reasignar estos para ser corregidos y asegurarse hacer un seguimiento de su progreso.

Por último, un tercer proyecto revisado fue el de (Jhonatan Enrique Cuevas Obregon, 2005), el cual realizó un: "Estudio sobre el uso del software libre en la Universidad de Pamplona", en este proyecto se hace un análisis del estado del software libre en Colombia para solventar la falta de conocimiento por parte de las empresas e instituciones educativas para que puedan sacar provecho de las herramientas de software que otorga el software libre. La técnica empleada en la investigación se fundamentó en la indagación de las opiniones que tienen las distintas comunidades de software libre, organizaciones y la Universidad de Pamplona, para ello se utilizó técnicas y herramientas para garantizar una eficaz obtención de información y con ella se elaboraron estrategias para que el software libre sea promovido en todos los organismos posibles, concluyendo que las Universidades como tal no cuentan con una comunidad dedicada al involucramiento de los procesos de desarrollo de software libre. La relación del presente trabajo con la investigación mencionado radica en que en ella se expone la falta de involucramiento en la participación de desarrollo por parte de la Universidad de Pamplona en los productos software realizados en

ambientes de desarrollo colaborativo y distribuido, aun así, el proyecto refleja que la Universidad de Pamplona si usa las herramientas desarrolladas en estos ambientes.

2.2 Comunidades de desarrollo de software libre

2.2.1 Software Libre

La noción hacia el concepto de software libre viene de la traducción de free software pero ha sido mal interpretada ya que en el idioma el inglés la palabra free puede tomar dos significados, gratis o libre, por este ámbito muchas personas al escuchar el término software libre ya tiene un chip programado en la cabeza, que los lleva a concluir de que el software del que se está hablando es gratis.

La idea de software libre va más allá de utilizar una herramienta informática bajo una licencia de uso gratis, según (FSF, 2016)[1] 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 software libre (ver Tabla I Libertades del software 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.

Tabla I Libertades del software libre

Libertad 0	Libertad 1	Libertad 2	Libertad 3
La libertad de ejecutar el programa como se desea, con cualquier	La libertad de estudiar cómo funciona el programa y cambiarlo para que	La libertad de redistribuir copias para ayudar a su prójimo.	La libertad de distribuir copias de sus versiones modificadas a terceros. Esto le

[1] FSF: Free Software Fundation

propósito	haga lo que usted quiera. El acceso al código fuente es una condición necesaria para ello.		permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

Cuando se habla de software libre este no implica en sus cuatro libertades la idea de no involucrar un precio por el software, esto es planteado por (Fuggetta, 2003) el cual explica que la expresión de software libre se debe entender como "free speech" (libertad de expresión) y no como "free beer" (cerveza gratis). De hecho, si se analizan las libertades 2 y 3 se ve que la libertad de redistribuir copias ya sea modificadas o no, no hablan de que estas se hagan sin involucrar una tarifa por distribuirlas, luego se puede resolver la ambigüedad existente de que el software libre es necesariamente gratis.

Ahora, ¿cómo garantiza el software libre las libertades 1 y 3?, es aquí donde nace el concepto de Open Source, una de las características primordiales para reconocer un software como libre es que su código fuente esté disponible para todos.

2.2.2 Open Source

Se puede apreciar que el origen del Open Source está muy relacionada con el software libre, de hecho para poder poner en práctica las reglas 1 y 3 (realizar cambios y publicar versiones modificadas) se debe tener acceso al código fuente del producto software, es así que el proyecto GNU expone que para el año de 1998 un sector de la comunidad de software libre decidió tomar un nuevo rumbo para promover el Open Source.

Dentro de este nuevo 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, lo cual refleja que este nuevo movimiento deja a un lado la esencia de lo que es software libre.

Tabla II Premisas del Open Source

1. **Libre redistribución:** el software debe poder ser regalado o vendido libremente.
2. **Código fuente:** el código fuente debe estar incluido u obtenerse libremente.
3. **Trabajos derivados:** la redistribución de modificaciones debe estar permitida.
4. **Integridad del código fuente del autor:** las licencias pueden requerir que las modificaciones sean redistribuidas sólo como parches.
5. Sin discriminación de personas o grupos: nadie puede dejarse fuera.
6. **Sin discriminación de áreas de iniciativa:** los usuarios comerciales no pueden ser excluidos.
7. **Distribución de la licencia:** deben aplicarse los mismos derechos a todo el que reciba el programa.
8. **La licencia no debe ser específica de un producto:** el programa no puede licenciarse solo como parte de una distribución mayor.
9. **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.
10. **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.

2.2.3 Diferencias entre Open Source y software libre

Para entender las diferencias entre estos dos movimientos hay que 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.

Es por esta diferencia que Christine Peterson acuñó el término de Open Source para que no fuese confundido con el software libre y más bien se entendiese como una marca registrada para un producto software considerado libre, la Tabla III Software Libre vs Open Source muestra la relación entre estos tipos de comunidades.

Tabla III Software Libre vs Open Source

	Software libre	Open Source
Uso gratis	X	X
Código fuente disponible	X	X
Posibilidad de modificar	X	X
Posibilidad de compartir	X	X
Carácter Ideológico de libertad	X	

2.2.4 Forja (Software)

Una forja de software es una plataforma normalmente basada en web en donde se puede hacer un desarrollo colaborativo, esto se refiere a que dentro de estas plataformas normalmente se pueden

albergar el desarrollo de muchos proyectos de software, si se analiza el trabajo de (Bartolomé, 2014) se puede apreciar que algunos de los 5 proyectos en los que participó mejorando y reportando errores son desarrollados en la forja conocida como Sourceforge y en general los 5 proyectos se encuentran ubicados dentro de una comunidad de desarrollo colaborativo y distribuida.

Aun así, existen muchas forjas de software utilizadas por las comunidades Open Source o Software libre, de las más famosas están, Bitbucket, Google Code, la ya mencionada Sourceforge y GitHub, esta última forja alberga los proyectos utilizando el control de versiones Git, en ella se pueden encontrar desarrollos tales como Mozilla, Bootstrap, Rails, AngularJS y muchos más, al mencionar estos proyectos, todos de gran peso en cuanto a importancia dentro de programadores, se ve claramente como las forjas de software son un importante mecanismo para el desarrollo colaborativo, tanto así que han creado proyectos de software de tan gran reputación, esto se logra ya que estas forjas hacen posible la difusión de software y otorgan soporte a los usuario.

2.2.5 Roles dentro de las comunidades Open Source

Al analizar el trabajo de (Ishigetani, 2014) se observa que dentro de las comunidades Open Source en las que participó, existen diferentes formas de contribuir al proyecto, es decir, existen roles al momento de participar en la creación de un software, normalmente se piensa que al no tener conocimientos de programación no se puede participar en estos tipos de proyectos pero no es así, las comunidades de código abierto requieren no solo participantes en roles de desarrolladores que son muy importantes, sino que también y sin desmeritar los otros roles, necesitan de participantes para contribuir en aspectos tales como artísticos, este rol se puede apreciar en el desarrollo de Ubuntu el cual le otorga a este tipo de rol la responsabilidad de crear iconos, botones y logos y así otros diferentes tipos de roles (ver Tabla IV Roles comunes dentro comunidades Open Source).

Ahora uno de los roles que debe presentar un proyecto así sea desarrollado en la versión clásica de desarrollo de software o en el tipo de desarrollo de comunidades Open Source, es el rol de Testing, es decir, a todo desarrollo de software le es más que beneficioso tener un control sobre los bugs que presenta el software, llevar un seguimiento de estos y un control, de hecho, tanto desarrollos clásicos como de comunidades Open Source llevan en todos sus desarrollos un proceso de control de errores, esto es la gestión de pruebas, obviamente ejecutada de forma diferente

dependiendo del ambiente de desarrollo pero con el mismo fin, asegurar la calidad del producto software.

Tabla IV Roles comunes dentro comunidades Open Source

Administrador del OSS: Es el individuo o grupo que inicia el proyecto del para el desarrollo del producto software.
Desarrollador: Es el encargado de la codificación de las funcionalidades del desarrollo del producto software.
Escritores de documentación: Este rol tiene a cargo de escribir la documentación del producto software, es decir, este rol documenta cómo instalar, configurar y usar el producto software.
Traductor: Las comunidades Open Source son comunidades distribuidas esto quiere decir que existe la internalización en la participación del proyecto, luego existen diferentes lenguajes.
Testing: Son los participantes que verifican que el software haga lo que debe hacer en diferentes ambientes de hardware y software además son los que hacen reportes de bugs.
Soporte a usuarios: Esta tarea consiste en que usuarios responden inquietudes acerca del software de otros usuarios
Usuarios: Son los individuos que usan el producto software, esta definición incluye a empresas o cualquier otra organización.

2.3 Pruebas de software

Según (Bourque & Fairley, 2014) “las pruebas de software consisten en la verificación dinámica que un programa proporciona del comportamiento esperado sobre un conjunto finito de casos de prueba, adecuadamente seleccionados de un dominio de ejecución usualmente infinito.”

La anterior definición de las pruebas presenta cuatro palabras significativas, *dinámicamente, finito, seleccionados y esperados*, una prueba de software es dinámica ya que en éstas existen entrada de datos, esto es, si se ejecutara una prueba con los mismos datos a un sistema complejo no determinístico, no se tendrán los mismos resultados al momento de ejecutar la prueba.

La palabra clave *finito* se refiere a que inclusive un desarrollo de software relativamente pequeño presenta un conjunto de pruebas infinitas, luego se deben seleccionar un subconjunto posible dependiendo de criterios como la priorización, recursos y cronogramas.

Por otro lado la palabra significativa *seleccionados* se refiere a las diferentes técnicas empleadas por los ingenieros de software para hallar el subconjunto finito de pruebas del conjunto infinito de posibles pruebas de software, básicamente todos tienen en común la técnica de utilizar la experiencia del ingeniero de software en otros proyectos junto con el análisis de riesgos.

Para finalizar, cuando el autor utiliza la palabra *esperados* hace énfasis en la capacidad que tienen los ingenieros de software para decidir si la prueba de software arrojo los resultados deseados, para tomar esta decisión normalmente se basan en los requerimientos especificados, resultados esperados por parte del usuario, o finalmente, de expectativas razonables.

2.3.1 Clasificación las pruebas

Las pruebas pueden ser clasificadas según el destino de la prueba o según el objetivo de es ésta, la primera hace referencia a que “las pruebas del software se realizan normalmente a diferentes niveles durante los procesos de desarrollo y mantenimiento. Esto significa que el objeto de las pruebas puede cambiar: un módulo, un grupo de dichos módulos (relacionados por propósito, uso, comportamiento, o estructura), o un sistema completo.”, en resumen, la primera clasificación se enfoca en definir a que se le hace una prueba de software. A estos niveles se les conoce como, pruebas *de unidad, de integración y de sistemas*, donde cada nivel de pruebas no tienen prioridades

con respecto a los otros niveles, es decir, ninguno de ellos es más importante que los otros.

Las pruebas de unidad están enfocadas en verificar el correcto funcionamiento de módulos o componentes individuales aislados, normalmente estas pruebas de software se hacen cuando se tiene acceso al código fuente y se tienen las herramientas necesarias para la depuración de éste, luego este tipo de pruebas puede ser implementado en las comunidades de Open Source ya que el código está disponible a todo el mundo.

Siguiendo con los niveles de pruebas, el nivel de pruebas de integración está enfocado en la verificación del correcto ensamble entre módulos de software, el desarrollo tradicional plantea técnicas para asegurar el cumplimiento del enfoque de este nivel de prueba, entre estas técnicas se encuentran las de Arriba-Abajo o Abajo-Arriba.

Por último el nivel de pruebas de sistema se enfoca en verificar el correcto funcionamiento de todo el sistema, básicamente lo que se busca en este nivel es encontrar fallas en los requerimientos no funcionales ya que en los niveles anteriores se hayan las fallas de los funcionales, todo esto se refiere a la verificación que se hace para que el sistema cumpla con las especificaciones de seguridad, exactitud, velocidad y la confiabilidad.

La segunda forma de clasificar las pruebas de software consiste en definir para que se hace una prueba, es decir, "las pruebas se realizan en relación a conseguir un determinado objetivo, que se ha definido más o menos explícitamente y con diversos niveles de precisión."

En esta categoría de clasificación entran tipos de pruebas tales como pruebas de aceptación/calificación, pruebas de instalación, pruebas alfa y beta, pruebas funcionales, de rendimiento, de carga, entre otras (Bourque & Fairley, 2014).

2.3.2 Técnicas o estrategias para definir casos de pruebas

Uno de los propósitos u objetivos de implementar una prueba consiste en "revelar el máximo número posible de fallos potenciales", para lo cual se han implementado numerosas técnicas para lograr este cometido. Dentro de las relevantes se encuentran las que están basadas en la intuición y experiencia del ingeniero de software, en este grupo resalta la prueba de software basada en ad hoc, básicamente consiste en crear casos de pruebas basados en la habilidad, intuición y

experiencias de desarrollos similares.

Existen otras técnicas como la técnica basada en la especificación, las técnicas basadas en el código, técnicas basadas en errores, dentro de esta técnica uno de las que más destaca es la de conjetura de errores, en ella "los casos de pruebas se han diseñado específicamente por ingenieros de software intentando imaginar los errores más probables en un programa determinado. La historia de errores descubiertos en proyectos anteriores es una buena fuente de información, como lo es también la experiencia del ingeniero", para mayor información sobre estas técnicas se pueden realizar consultas a (Bourque & Fairley, 2014).

2.3.3 Pruebas Funcionales

Las pruebas funcionales tiene el objetivo de verificar si el comportamiento del software se corresponde con las especificaciones, esto se refiere a que las pruebas funcionales son las encargadas de validar que el producto software hace lo que debe y sobre todo lo que se ha especificado en los requerimientos. Las pruebas funcionales se pueden categorizar de la siguiente forma:

Según su ejecución: Pueden ser manuales o automáticas, las primeras consisten en hacerse pasar como un usuario normal pero siguiendo unas pautas establecidas en un test plan o plan de pruebas, el segundo radica en automatizar dichas pruebas manuales con el propósito de ahorrar tiempo.

Según el tipo de pruebas: De acuerdo a esta categoría se pueden encontrar pruebas de integración, aceptación, compatibilidad, de regresión y exploratorias, por lo cual se modificó la base de datos de Bugzilla, para que al momento de crear un plan de pruebas éstas puedan ser configuradas en el plan.

Según la metodología: En esta clase de clasificación pueden aplicarse tanto la metodología estándar así como la de desarrollos ágiles.

En el caso de Testopia el cual está a cargo de las pruebas funcionales puede manejar según la

ejecución de pruebas los dos tipo de pruebas, tanto manuales como automáticas, asimismo maneja tipos de pruebas según su tipo como las de aceptación las cuales no solo verifican que el sistema haga lo esperado sino que además involucra que el usuario final determine su aceptación, otro tipo de prueba que se puede apreciar al momento de trabajar con Testopia son las pruebas de integración estas están encargadas de que el ensamblaje de módulos no se solapen una vez hayan sido probadas individualmente, además este procedimiento agregara más tipos de pruebas funcionales.

2.3.4 Uso de Bugzilla

Bugzilla es una herramienta la cual es utilizada especialmente para el seguimiento de errores, aunque es también usada para incorporar nuevas funcionalidades al desarrollo, los conceptos más relevantes para entender a Bugzilla es comprender que es un producto, versión, componente y un milestone.

Clasificación: Las clasificaciones en Bugzilla son usualmente utilizadas para relacionar varios productos, ésta está deshabilitada por defecto al momento de la instalación, para este proyecto específico se redefine el concepto de clasificación por lo que estos representan los módulos de un desarrollo de software, estos módulos están fundamentados en la descomposición de Academusoft 4.0 (ver Ilustración 2 Módulos principales de Academusoft 4.0)

Producto: Un producto en Bugzilla usualmente representa a un objeto de la vida real, es decir si una compañía se dedica a desarrollar juegos puede tener como clasificación “juegos” y dentro de esta un producto para cada uno de ellos, esta connotación es observada en el Bugzilla del proyecto de Mozilla Firefox, allí se puede apreciar claramente como cada producto representa cada lanzamiento que ellos han hecho. Aun así, no todos los proyectos usan como regla asignar en el apartado de producto a cada producto de que han desarrollado sino a módulos que conforman un proyecto, tal es el caso de Yocto Project, ellos definen como producto no solo sus desarrollos sino módulos que componen ese proyecto. Para este específico procedimiento se tratan a los productos como submódulos de un desarrollo de software, esto basándose en la descomposición de Academusoft 4.0 (ver Ilustración 3 Submódulo de Academusoft 4.0).

Componente: Los componentes usualmente son subestaciones o módulos del software, siguiendo con el ejemplo de la empresa desarrolladora de juegos, un componente para juego sería

“API”, “Sonido del Sistema”, “Plugins”, los cuales pueden ser asignados a un programador o grupo de programadores, sucede igualmente que con el producto, estos no son necesariamente módulos sino que para este procedimiento representa una funcionalidad (basado en la estructura de Academusoft 4.0, ver Ilustración 4 Funcionalidades Academusoft 4.0) , es esta una de las características de Bugzilla, la capacidad que tiene para que el usuario se adapte de acuerdo a su necesidad de administración del desarrollo.

Versión: Una versión representa las revisiones del producto, tales como "Flinders 3.1", "Flinders 95", and "Flinders 2000", estas son importantes dado que son a ellas a las que se les asigna un bug, resulta que puede ser muy tedioso asignar versiones a tantos submódulos los cuales representan a los productos, por lo cual se planteó crear un proyecto en Java (ver Ilustración 2 Diagrama de clase, proyecto Java "toolToBugzilla") que haga este proceso automáticamente, es decir dada una versión, asignar está a todos los productos (submódulos) que exista.

Milestone: Un milestone o hito son objetivos para el cual está planeado que se solucione un bug o se haga una mejora al código, es decir, si se desea solucionar una falla en algún componente (Funcionalidad) para la versión 3 de cierto producto software, entonces se debe registrar un milestone 3.0. Al igual que los productos (submódulos), agregar un milestone a cada submódulo puede ser una tarea engorrosa dada la cantidad de productos que se pueden crear en un software como por ejemplo el de Academusoft 4.0, por lo cual el proyecto de java (ver Ilustración 2 Diagrama de clase, proyecto Java "toolToBugzilla") soluciona a la perfección esta dificultad.

2.3.5 Estados de un bug en Bugzilla

Al momento de reportar un bug en Bugzilla este tiene una serie de estados, de acuerdo a ciclo de vida que maneja **Open Office** un bug puede tener las siguientes variaciones

Cuando el bug está abierto

Dentro de estos estados se encuentran tres (3) estados que puede tener el bug:

- **UNCONFIRMED:** Es un nuevo reporte sobre una falla, nadie lo ha confirmado

que efectivamente este error es válido. Esta opción está deshabilitada para el procedimiento propuesto.

- **CONFIRMED:** Un usuario con los privilegios suficientes le cambia a este nuevo estado para referirse que lo que se ha reportado es efectivamente una falla.
- **IN PROGRESS:** El usuario responsable del componente se hace responsable de la falla y empieza a trabajar sobre esta.

Cuando el bug está cerrado

Un bug se encuentra cerrado cuando este cambio al estado **RESOLVED,** esta solución cuenta además con otros estados dependiendo del tipo de solución que se le ha dado.

- **FIXED:** Se solucionó el bug.
- **INVALID:** El reporte de la falla no es una falla.
- **WONTFIX:** Es un bug, pero no se va solucionar.
- **DUPLICATED:** El bug ya fue reportado.
- **WORKSFORME:** No se ha podido reproducir el error.

Si el usuario que reporta el bug está de acuerdo con la solución planteada por el encargado de la funcionalidad entonces cambiara el estado del bug a **VERIFIED.** En la sección 3.1.3 Definir WorkFlow se especifica el flujo de estos de estados.

2.3.6 Uso de Testopia

Testopia maneja como administrador de casos de prueba diferentes tipos de pruebas entre ellos está el de caja negra para la cual fue diseñado, pero esto no la exenta de poder realizar pruebas de caja blanca, para este proyecto se trabaja con las pruebas de caja negra dado que se le dará la importancia de que el software haga lo que tiene que hacer sin preocuparnos por cómo lo hace.

A continuación se muestra los apartados con los que trabaja Testopia:

Plan de pruebas: Es lo primero que se debe realizar dado que los planes de pruebas representan a un producto, aun así se pueden asociar múltiples planes de prueba a un solo producto, es importante realizarlo como primero dado que es en él en donde se albergarán los casos de prueba. En los planes de pruebas se especifican la versión del producto, el tipo de prueba, autor, etc.

Testopia maneja los siguientes tipos de pruebas:

- Aceptación: Las pruebas de aceptación hacen parte de las pruebas funcionales y reflejan la aprobación del usuario con respecto al sistema.
- Función: Estas son las pruebas que se manejan en este procedimiento, las pruebas funcionales prueban y validan que el software hace lo que debe y sobre todo lo que se ha especificado.
- Instalación: Las pruebas de instalación pertenecen al tipo de pruebas de soportabilidad, éstas validan que el elemento que se prueba se instala y puede ser configurado en diferentes S.O y hardware de acuerdo a lo especificado.
- Integración: Este tipo de pruebas tienen como objetivo verificar el correcto ensamblaje de sus diferentes componentes.
- Interoperabilidad: Están enfocadas hacia a validar que productos o sistemas puedan comunicarse entre sí, es tipo de prueba tampoco está presente el procedimiento porque se ha planteado que solo se manejara un solo producto software.
- Desempeño: Este tipo de pruebas están orientadas a verificar tiempos de respuestas, de capacidad, etc.
- Sistema: Estas pruebas buscan probar el sistema como un todo es decir probar todas las partes del sistema.
- Unidad: Las pruebas de unidad buscan verificar en general todo el código.

Para este procedimiento muchas de estos tipos de pruebas serán eliminadas y reemplazadas por

los diferentes de tipos de pruebas funcionales.

Casos de pruebas: En Testopia los casos de pruebas pueden estar asociados a múltiples planes de pruebas dado que son semiindependientes, en estos se especifican las entradas y salidas esperadas de la prueba, si no corresponden las salidas a las entradas establecidas, se estaría hablando de que existe una falla en el software. Dentro de este gestor de casos de prueba se pueden asignar a los casos de pruebas categorías, estas pueden ser tantas como sean necesarias, no hay que confundirlas con los componentes en de Bugzilla.

Categoría del caso de prueba: Las categorías de los casos de pruebas representan cada cuanto se debe ejecutar un caso de prueba, para este procedimiento se definirán 2 aparte de la que trae el Testopia al momento de crear el proyecto, al igual que las versiones y milestones del Bugzilla, por la redefinición del concepto producto, es complicado agregar a todas estas las categorías, por lo que el proyecto java (ver Ilustración 2 Diagrama de clase, proyecto Java "toolToBugzilla") soluciona este inconveniente a la perfección.

Entorno de ejecución de prueba: Aquí se especifica en donde se van realizar las pruebas, lo cual significa que toda prueba es ejecutada en cierta plataforma, algunos desarrollos de software están diseñados para ser ejecutados en ciertas plataformas, tal es el caso de Academusoft, el cual está disponible para ciertos navegadores web, al igual que las Categorías, por la redefinición del concepto de producto es necesario implementar el proyecto java (ver Ilustración 2 Diagrama de clase, proyecto Java "toolToBugzilla") para apaciguar la dificultad de agregar a cada submódulo (producto) un entorno de pruebas,

Build: Normalmente un desarrollo de software viene dado por procesos iterativos, es decir se construye código para solucionar ya sea fallas o hacer mejoras al producto software, en el mundo de Testopia, se le conoce como Build a cada iteración realizada.

Ejecución de pruebas: Una vez que se tenga un conjunto de casos de pruebas, las categorías, los entornos y el Build se estará listo para realizar una ejecución de prueba, cada ejecución se encuentra asociada a un plan de pruebas y pueden estar presente varios casos de pruebas, siempre y cuando éstos pertenezcan al plan de prueba.

Prueba Caso-Ejecución: Una prueba Caso-Ejecución es un registro de cómo un determinado caso de prueba se comportó en una ejecución determinada para un Build dado en un entorno o ambiente determinado.

2.3.7 Bugzilla y Testopia

Bugzilla es una herramienta de software que está basada en web, la cual está encargada del seguimiento de errores, éste cuenta dentro de sus múltiples extensiones con Testopia quien está a cargo de administrar los casos de prueba del software, la interacción de entre Bugzilla y Testopia puede reflejarse en Ilustración 1 Relación entre Bugzilla y Testopia**.**

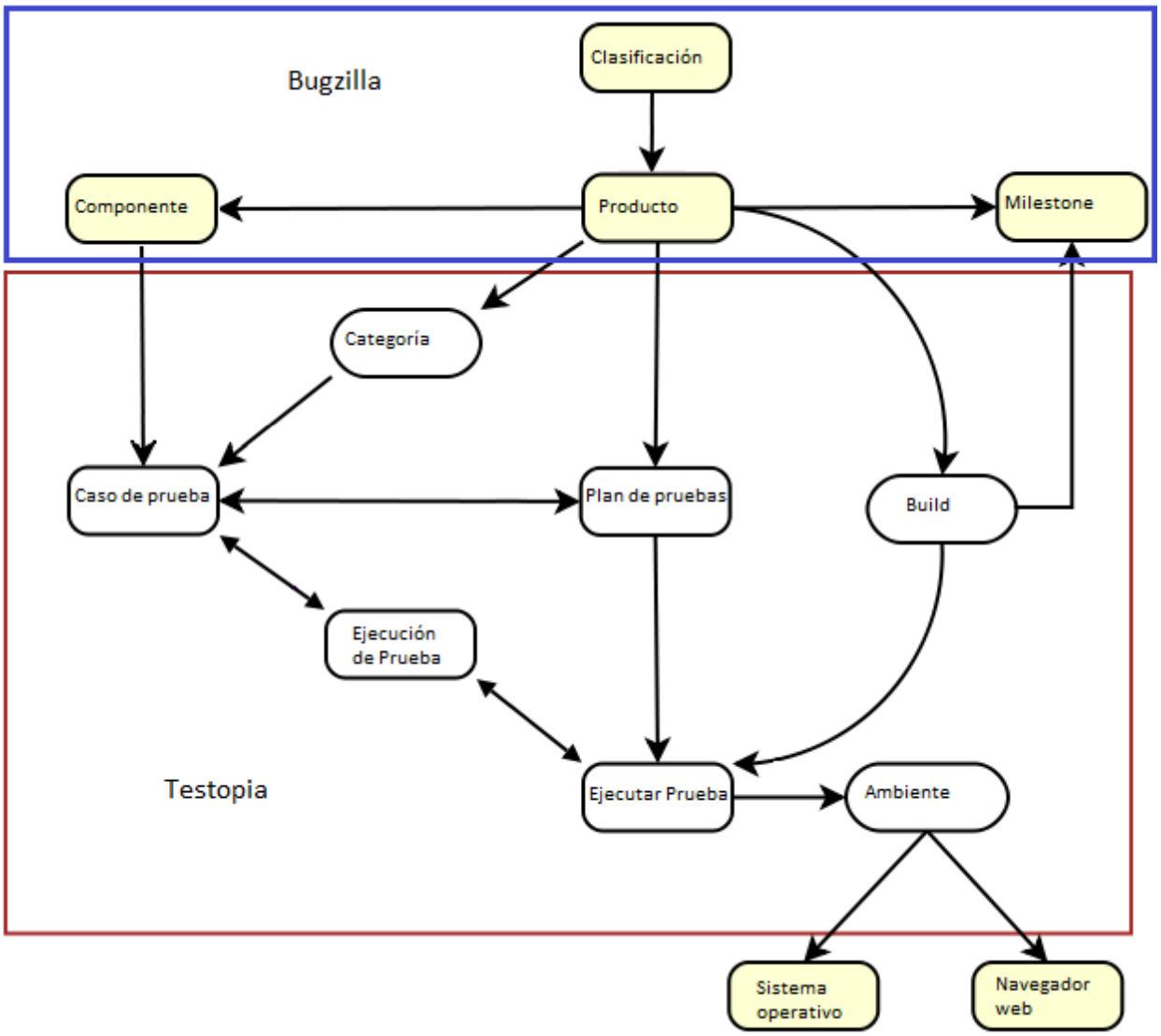


Ilustración 1 Relación entre Bugzilla y Testopia

La Ilustración 1 Relación entre Bugzilla y Testopia, muestra el normal funcionamiento del Bugzilla con Testopia, pero como para este procedimiento se ha planteado redefinir el concepto de producto lo cual lleva a que existan muchos productos (Submódulos), se decidió crear un proyecto en java conocido como “toolToBugzilla” que dada una versión, un milestone, una categoría de prueba y un ambiente registra a cada uno ellos en todos los productos. Esto se ideó dado que resulta demasiado tedioso agregar por ejemplo una nueva versión a todos los submódulos, El Diagrama de clase se puede observar en la Ilustración 2 Diagrama de clase, proyecto Java "toolToBugzilla",

este lo puede encontrar en https://github.com/daas93/toolToBugzilla.

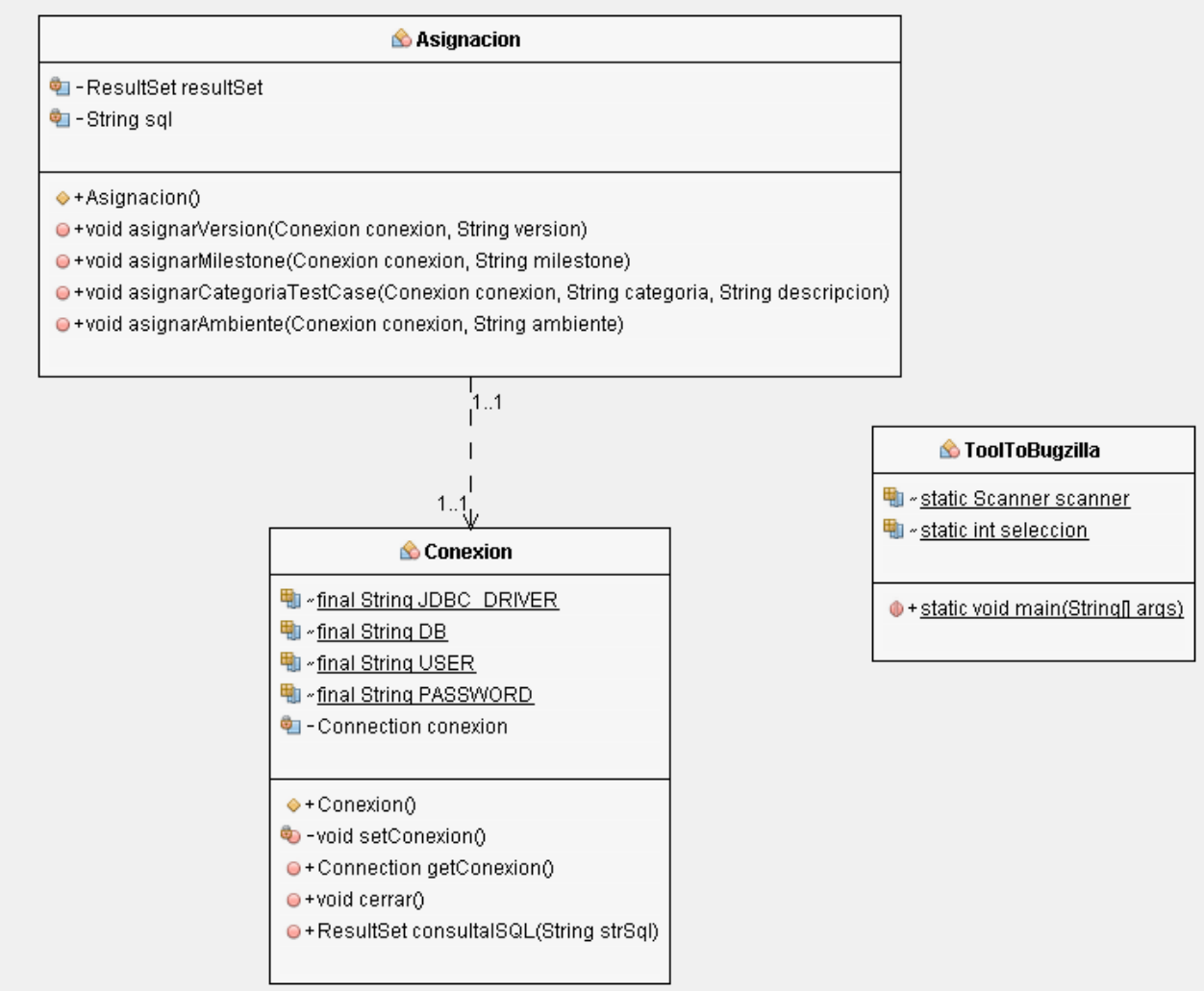


Ilustración 2 Diagrama de clase, proyecto Java "toolToBugzilla"

El código fuente de la clase Conexión se puede apreciar en la tabla Tabla V Clase Conexión la cual ésta encargada de crear la conexión a la base de datos del Bugzilla y define los métodos necesarios para cerrar la conexión y hacer una consulta, para el correcto funcionamiento de esta clase se debe existir la importación de la librería del JDBC de MySQL. Al momento de utilizar esta clase se debe cambiar el valor de las variables estáticas DB, USER y PASSWORD y volver a compilar la clase, la cual se encuentra como todas en el package tooltobugzilla.

Tabla V Clase Conexión

```
1.  /*
2.   * To change this license header, choose License Headers in Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package tooltobugzilla;
7.
8.  import java.sql.Connection;
9.  import java.sql.DriverManager;
10. import java.sql.ResultSet;
11. import java.sql.SQLException;
12.
13. /**
14.  *
15.  * @author Diego Aguilar
16.  */
17. public class Conexion {
18.
19.  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
20.  static final String DB = "jdbc:mysql://localhost:3306/bugzilla";
21.  static final String USER = "root";
22.  static final String PASSWORD = "";
23.
24.  private Connection conexion;
25.
26.  public Conexion() throws ClassNotFoundException, InstantiationException,
     IllegalAccessException, SQLException {
27.
28.  this.setConexion();
29.  }
30.
31.  private void setConexion() throws ClassNotFoundException, InstantiationException,
     IllegalAccessException, SQLException {
32.
33.  Class.forName("com.mysql.jdbc.Driver");
34.
35.  conexion = DriverManager.getConnection(DB, USER, PASSWORD);
36.  }
37.
38.  public Connection getConexion() {
39.  return this.conexion;
40.  }
41.
```

```
42.  public void cerrar() throws SQLException {
43.  this.conexion.close();
44.  }
45.
46.  public ResultSet consultalSQL(String strSql) throws SQLException {
47.  return conexion.prepareStatement(strSql).executeQuery();
48.  }
49.
50. }
51.
```

La clase Asignación contiene la lógica para establecer una nueva conexión a la base de datos, hacer consultas y modificar los registros de acuerdo al método invocado desde la clase principal, es decir, de acuerdo a la petición del usuario con respecto a si desea asignar una versión, milestone, categoría o ambiente a los submódulos (productos), su código fuente puede apreciarse en la tabla Tabla VI Clase Asignación

Tabla VI Clase Asignación

```
1.  /*
2.   * To change this license header, choose License Headers in Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package tooltobugzilla;
7.
8.  import java.sql.PreparedStatement;
9.  import java.sql.ResultSet;
10. import java.sql.SQLException;
11.
12. /**
13.  *
14.  * @author Diego Aguilar
15.  */
16. public class Asignacion {
17.
```

```
18.  private ResultSet resultSet;
19.  private String sql;
20.
21.  public Asignacion() {
22.  resultSet = null;
23.  sql = "";
24.  }
25.
26.  public void asignarVersion(Conexion conexion, String version) {
27.  try {
28.  resultSet = conexion.consultalSQL("SELECT id,name FROM products");
29.  sql = "INSERT INTO `versions` VALUES(null,?,?,?)";
30.  while (resultSet.next()) {
31.   PreparedStatement preparedStatement =
     conexion.getConexion().prepareStatement(sql);
32.  preparedStatement.setString(1, version);
33.  preparedStatement.setInt(2, resultSet.getInt("id"));
34.  preparedStatement.setInt(3, 1);
35.  int rows = preparedStatement.executeUpdate();
36.  if (rows > 0) {
37.   System.out.println("Versión " + version + " añadida al producto " +
     resultSet.getString("name"));
38.  }
39.  }
40.  } catch (SQLException e) {
41.   System.err.println("Error en base de datos: " + e);
42.  }
43.
44.  }
45.
46.  public void asignarMilestone(Conexion conexion, String milestone) {
47.  try {
48.  resultSet = conexion.consultalSQL("SELECT id,name FROM products");
49.  sql = "INSERT INTO `milestones` VALUES(null,?,?,?,?)";
50.  while (resultSet.next()) {
51.   PreparedStatement preparedStatement =
     conexion.getConexion().prepareStatement(sql);
52.  preparedStatement.setInt(1, resultSet.getInt("id"));
53.  preparedStatement.setString(2, milestone);
54.  preparedStatement.setInt(3, 0);
55.  preparedStatement.setInt(4, 1);
56.  int rows = preparedStatement.executeUpdate();
57.  if (rows > 0) {
58.   System.out.println("Milestone " + milestone + " añadida al producto " +
     resultSet.getString("name"));
59.  }
```

```
60.  }
61.  } catch (SQLException e) {
62.   System.err.println("Error en base de datos: " + e);
63.  }
64.
65.  }
66.
67.  public void asignarCategoriaTestCase(Conexion conexion, String categoria, String
     descripcion) {
68.  try {
69.  resultSet = conexion.consultalSQL("SELECT id,name FROM products");
70.  sql = "INSERT INTO `test_case_categories` VALUES(null,?,?,?)";
71.  while (resultSet.next()) {
72.   PreparedStatement preparedStatement =
     conexion.getConexion().prepareStatement(sql);
73.  preparedStatement.setInt(1, resultSet.getInt("id"));
74.  preparedStatement.setString(2, categoria);
75.  preparedStatement.setString(3, descripcion);
76.  int rows = preparedStatement.executeUpdate();
77.  if (rows > 0) {
78.   System.out.println("La categoria " + categoria + " fue añadida al producto " +
     resultSet.getString("name"));
79.  }
80.  }
81.  } catch (SQLException e) {
82.   System.err.println("Error en base de datos: " + e);
83.  }
84.
85.  }
86.
87.  public void asignarAmbiente(Conexion conexion, String ambiente) {
88.  try {
89.  resultSet = conexion.consultalSQL("SELECT id,name FROM products");
90.  sql = "INSERT INTO `test_environments` VALUES(null,?,?,?)";
91.  while (resultSet.next()) {
92.   PreparedStatement preparedStatement =
     conexion.getConexion().prepareStatement(sql);
93.  preparedStatement.setInt(1, resultSet.getInt("id"));
94.  preparedStatement.setString(2, ambiente);
95.  preparedStatement.setInt(3, 1);
96.  int rows = preparedStatement.executeUpdate();
97.  if (rows > 0) {
98.   System.out.println("Ambiente " + ambiente + " añadida al producto " +
     resultSet.getString("name"));
99.  }
100.          }
```

```
101.            } catch (SQLException e) {
102.             System.err.println("Error en base de datos: " + e);
103.            }
104.
105.            }
106.
107.           }
108.
```

Por último, la clase principal ToolToBugzilla es la encarga de comunicar un objeto Conexion con el objeto Asignarcion, leer los datos del usuario y enviarlos a los métodos de la clase Asignacion dependiendo de la petición del usuario o administrador del Bugzilla, el código fuente de la clase puede verse en la Tabla VII Clase Principal.

Tabla VII Clase Principal

```
1.  /*
2.   * To change this license header, choose License Headers in Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package tooltobugzilla;
7.
8.  import java.sql.SQLException;
9.  import java.util.Scanner;
10.
11. /**
12.  *
13.  * @author Diego Aguilar
14.  */
15. public class ToolToBugzilla {
16.
17.  /**
18.  * @param args the command line arguments
19.  */
20.  static Scanner scanner = new Scanner(System.in);
```

```
21.  static int seleccion = -1;
22.
23.  public static void main(String[] args) throws ClassNotFoundException,
     InstantiationException, IllegalAccessException, SQLException {
24.  // TODO code application logic here
25.  Conexion conexion = new Conexion();
26.  Asignacion asignacion=new Asignacion();
27.
28.
29.  while (seleccion != 0) {
30.   System.out.println("Elige opción:\n1.- Agregar una versión a todos los submódulos-
     productos"
31.  + "\n2.- Agregar un milestones a todos los submódulos-productos\n"
32.  + "3.- Agregar una categoria a todos los submódulos-productos\n"
33.  + "4.- Agregar un ambiente/entorno a todos los submódulos-productos\n"
34.  + "0.- Salir");
35.
36.  seleccion = scanner.nextInt();
37.
38.  switch (seleccion) {
39.  case 1:
40.  Scanner versionScanner = new Scanner(System.in);
41.   System.out.println("Digita la versión para agregar todos los productos");
42.   String version = versionScanner.nextLine();
43.  asignacion.asignarVersion(conexion, version);
44.  break;
45.
46.  case 2:
47.  Scanner milestonesScanner = new Scanner(System.in);
48.   System.out.println("Digita la mailestone para agregar todos los productos");
49.   String milestone = milestonesScanner.nextLine();
50.  asignacion.asignarMilestone(conexion, milestone);
51.  break;
52.
53.  case 3:
54.  Scanner categoriaTestScanner = new Scanner(System.in);
55.   System.out.println("Digita la categoria para agregar a todos los productos");
56.   String categoriaTest = categoriaTestScanner.nextLine();
57.   System.out.println("Digita la descripción para agregar a todos los productos");
58.   String descripcion = categoriaTestScanner.nextLine();
59.  asignacion.asignarCategoriaTestCase(conexion, categoriaTest, descripcion);
60.  break;
61.
62.  case 4:
63.  Scanner ambienteScanner = new Scanner(System.in);
64.   System.out.println("Digita el ambiente/entorno para agregar a todos los productos");
```

```
65.   String ambiente = ambienteScanner.nextLine();
66.  asignacion.asignarAmbiente(conexion, ambiente);
67.  break;
68.  case 0:
69.  conexion.cerrar();
70.   System.out.println("Bye");
71.
72.  }
73.
74.  }
75.
76.
77.  }
78.
79. }
```

2.4 Proceso de pruebas en comunidades Open Source

En las comunidades de Open Source se puede observar que normalmente los desarrolladores son los que hace las pruebas de sus entregables, aun así, es el administrador (o los mismos colaboradores dependiendo del proyecto) quien aprueba la incorporación del entregable, de hecho, al ver el trabajo de (Bartolomé, 2014) se refleja como es el proceso dentro de las comunidades Open Source con respecto a las pruebas de software, en su participación en el proyecto de FileZilla, realizando traducciones, encontró que antes de subir sus aportes debía testearlos y se le pedía que el entregable debía estar de cierta forma estructurado de lo contrario ésta podría ser rechazado.

En este tipo de desarrollos de software existe normalmente un sistema de seguimiento de defectos (errores), uno de los más conocidos es Bugzilla, esta herramienta lanzada en 1998 la cual fue inicialmente usada en el proyecto de Mozilla, es una de las más utilizadas en las comunidades Open Source con el fin de administrar los bugs del producto software, permitiendo llevar un seguimiento de errores de múltiples productos con sus diferentes versiones categorizando a los bugs de acuerdo a la prioridad de éstos y la severidad que resultan ser si no son solucionados. Además los bugs reportados pueden encontrarse en diferentes estados y ser asignados directamente

a un desarrollador aunque proyectos como Mozilla también utilizan a Bugzilla para el registro de nuevas funcionalidades.

2.5 Marco Conceptual

Forja: Una forja de software es una plataforma normalmente basada en web de desarrollo colaborativo la cual alberga múltiples proyectos de software, está enfocada para la difusión del producto software y soporte al usuario.

Bugzilla: Es un sistema de seguimiento de errores basado en web para la gestión de múltiples productos con múltiples versiones, su función principal es gestionar los defectos presentes en los productos software aunque también es usado como herramienta para incorporar nuevos requerimientos.

Testopia: Testopia es un administrador de casos de prueba su función principal radica en seguir los casos de pruebas de software para hacer un reporte de los defectos encontrados, además está integrado con Bugzilla.

Ambiente simulado: Para este proyecto cuando se hable de un ambiente simulado se hace referencia a emular el comportamiento de una comunidad Open Source para implementar un procedimiento de gestión de pruebas, evaluar su desempeño y validar su funcionamiento.

Bug: Un Bug es un error de software lo cual lleva a que el producto software se comporte de una forma no deseada, esto puede deberse a defectos en el diseño del software, un error de programación o algún defecto de instalación.

Rol: El rol representa el conjunto de funciones que tiene asociados un usuario en un lugar o situación, esto es desarrollador, diseñador, administrador, Testing etc.

Desarrollo colaborativo: Un desarrollo colaborativo es un trabajo en grupo el cual es diseñado

para el desarrollo de actividades o procesos efectivos y productivos, todo desarrollo colaborativo es trabajo en grupo, pero no todo trabajo en grupo es colaborativo.

Desarrollo distribuido: Un desarrollo distribuido es aquel en donde los participantes se encuentran en diferentes lugares al mismo tiempo y participan conjuntamente en un producto en este caso, un producto software.

2.6 Marco Legal y Jurídico

En Colombia se puede proteger el software jurídicamente desde los derechos de autor, patente o propiedad intelectual, esto es por medio de la **ley 23 de 1982**.

En el **artículo 1**°, dice "Los autores de obras literarias, científicas y artísticas gozarán de protección para sus obras en la forma prescrita por la presente Ley y, en cuanto fuere compatible con ella, por el derecho común…"

El **artículo 2**, dice "Los derechos de autor recaen sobre las obras científicas literarias y artísticas las cuales se comprenden todas las creaciones del espíritu en el campo científico, literario y artístico, cualquiera que sea el modo o forma de expresión y cualquiera que sea su destinación , tales como: los libros, folletos y otros escritos; las conferencias, alocuciones, sermones y otras obras de la misma naturaleza; las obras dramáticas o dramático-musicales; las obras coreográficas y las pantomimas; las composiciones musicales con letra o sin ella; las obras cinematográficas, a las cuales se asimilan las obras expresadas por procedimiento análogo a la cinematografía, inclusive los videogramas…"

La legislación de Colombia involucra el software en su legislación por medio de un acuerdo establecida con la Organización Mundial del Comercio reflejada en la **ley 170 de 1994.**

En el **artículo 10** dice "Los programas de ordenador, sean programas fuente o programas objeto, serán protegidos como obras literarias en virtud del Convenio de Berna (1971)."

Al igual que en la decisión andina **351 de 1993** la cual expone en su **artículo 23** que "Los programas de ordenador se protegen en los mismos términos que las obras literarias… Sin perjuicio de ello, los autores o titulares de los programas de ordenador podrán autorizar las modificaciones necesarias para la correcta utilización de los programas."

Además existe la ley de **Ley 565 de 2000** en el cual se aprobó el tratado de OMPI sobre los derechos de autor, el cual el **artículo 4** dice: "Los programas de ordenador están protegidos como obras literarias en el marco de lo dispuesto en el artículo 2 del Convenio de Berna. Dicha protección se aplica a los programas de ordenador, cualquiera que sea su modo o forma de expresión."

Ahora, el aspecto penal que controla la protección del software es la **Ley 599 de 2000** (Código Penal).

El **artículo 270** sanciona a quien viole derechos morales de la siguiente manera:

1. Publique, total o parcialmente, sin autorización previa y expresa del titular del derecho, una obra inédita de carácter literario, artístico, científico, cinematográfico, audiovisual o fonograma, programa de ordenador o soporte lógico.

2. Inscriba en el registro de autor con nombre de persona distinta del autor verdadero, o con título cambiado o suprimido, o con el texto alterado, deformado, modificado o mutilado, o mencionando falsamente el nombre del editor o productor de una obra de carácter literario, artístico, científico, audiovisual o fonograma, programa de ordenador o soporte lógico.

3. Por cualquier medio o procedimiento compendie, mutile o transforme, sin autorización previa o expresa de su titular, una obra de carácter literario, artístico, científico, audiovisual o fonograma, programa de ordenador o soporte lógico.

PARÁGRAFO. Si en el soporte material, carátula o presentación de una obra de carácter literario, artístico, científico, fonograma, videogramas, programa de ordenador o soporte lógico u obra cinematográfica se emplea el nombre, razón social, logotipo o distintivo del titular legítimo del derecho, en los casos de cambio, supresión, alteración, modificación o mutilación del título o del texto de la obra, las penas anteriores se aumentarán hasta en la mitad.

Otro artículo que defiende al software es el **271** el cual expone los siguientes hechos:

1. Por cualquier medio o procedimiento reproduzca una obra de carácter literario, científico, artístico o cinematográfico, fonograma, videograma, soporte lógico o programa de ordenador, o, quien transporte, almacene, conserve, distribuya, importe, venda, ofrezca, adquiera para la venta o distribución, o suministre a cualquier título dichas reproducciones.

2. Alquile o, de cualquier otro modo, comercialice fonogramas, videogramas, programas de ordenador o soportes lógicos u obras cinematográficas.

Para finalizar se tiene el **artículo 272** el cual sanciona:

1. Supere o eluda las medidas tecnológicas adoptadas para restringir los usos no autorizados.
2. Suprima o altere la información esencial para la gestión electrónica de derechos, o importe, distribuya o comunique ejemplares con la información suprimida o alterada.

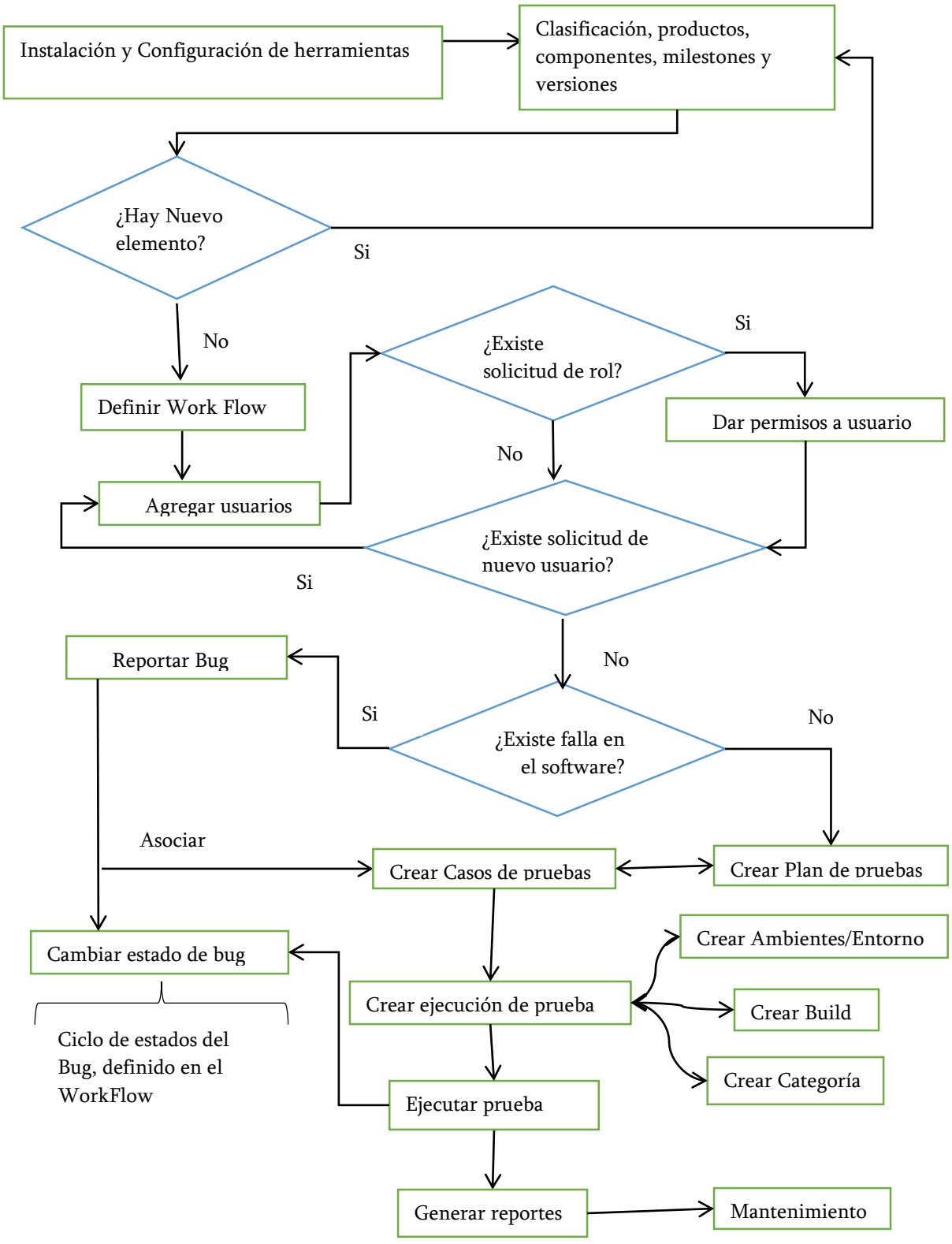
Hay que aclarar que el **software libre** está cobijado por las leyes de derechos de autor, esto es, cualquier persona que desarrolle software libre aún si es ambientes de desarrollo colaborativo queda cobijado por los derechos morales sobre este desarrollo o si simplemente la ha modificado o mejorado también recae sobre esta persona el derecho moral sobre el producto software.

Tanto los softwares privados como los libres cuentan con licencias de funcionamientos, en las cuales restringen u otorgan libertades sobre el producto software en cuestión. En algunos licenciamientos de software libre puede haber condicionamientos sobre los cobros máximos que se puedan hacer por distribución del producto desarrollado u alguna asesoría sobre éste.

3 Procedimiento propuesto

Este capítulo está compuesto por un diagrama del flujo de trabajo del procedimiento propuesto, el cual da una perspectiva clara de los pasos relevantes que plantea el procedimiento para que sea genérico y pueda adaptarse a los demás procesos de desarrollo de software. Además de la descripción detallada de cada uno de los pasos propuestos y la implementación de toolToBugzilla quien facilitara la implementación de estos, junto con su respectiva validación, esta validación fue basada en la construcción de un software en los servidores de la comunidad de software libre de la Universidad de Pamplona, el cual es conocido como *cursosLibres.*

3.1 Descripción del procedimiento



3.1.1 Instalación y Configuración de herramientas

- Instalar Perl (preferiblemente 5.8.1 o una versión mayor)
- Instalar motor de base de datos (Bugzilla soporta Mysql 5.0 o mayor, Postgres 8.03 o mayor y Oracle 10.02 o mayor)
- Instalar un servidor web (Se recomienda usar Apache web server entre 1.3.x o 2.x)
- Instalar Bugzilla (Tarballs Bugzilla se recomienda Bugzilla 4.2.2)
- Instalar Módulos de Perl (Lista de módulos)
- Instalar un Agente de transporte de correo (Preferiblemente Sendmail)
- Instalar Testopia (Tarballs Testopia se recomienda Testopia 2.5)
- Configurar todo lo anterior (Guía de instalación de Bugzilla con Testopia)
- Montar el proyecto java toolToBugzilla (toolToBugzilla)
- Configurar la clase Conexion
- Compilar las clases del proyecto toolToBugzilla.

3.1.2 Clasificaciones, productos, componentes, versiones y milestones

El siguiente procedimiento está enfocado en la gestión de pruebas para un único desarrollo de software, es decir, estos pasos se deben entender como si existiera un Bugzilla para un solo desarrollo con múltiples versiones y no un Bugzilla para múltiples proyectos de desarrollo. La siguiente imagen muestra el desglose de un software genérico y se utilizara para explicar este paso.

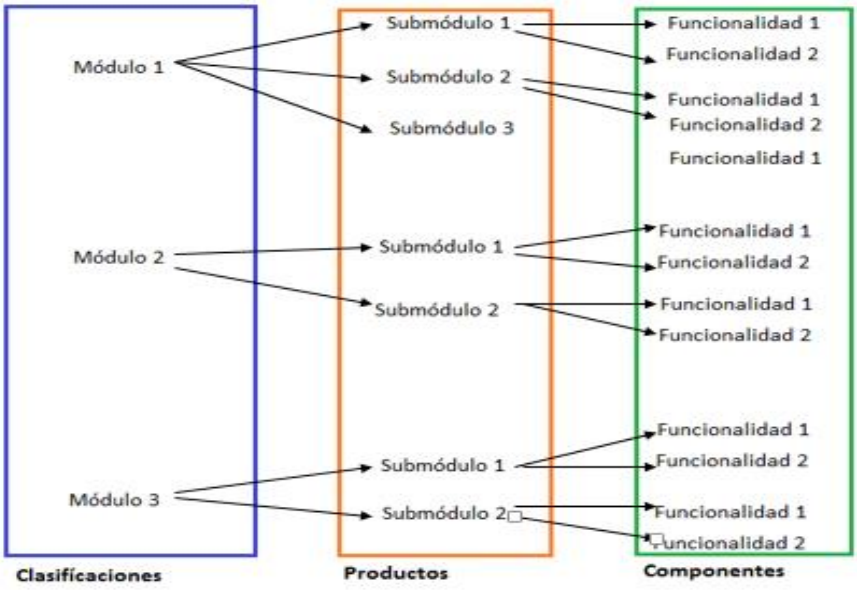


Ilustración 3 Ejemplo genérico del desglose de un software

- Registrar las clasificaciones

Las clasificaciones representarán módulos, que a su vez tienen sub módulos que a su vez tienen componentes, este comportamiento se reflejado en la Ilustración 3 Ejemplo genérico del desglose de un software.

Se debe registrar estas en Bugzilla y tener en cuenta al momento de la instalación que se debe habilitar el parámetro **useclassification** en la sección de Parámetros, luego editar a Test Product por el nombre que se desee dar a la primera clasificación. Para las siguientes, estas podrán ser añadidas tantas como se necesiten.

- Registrar los productos

Los productos representan submódulos del desarrollo del software. Se debe entender un submódulo como un conjunto de elementos agrupados los cuales lo forman, en este caso están formados por los componentes. Este comportamiento se ve reflejado en la Ilustración 3 Ejemplo genérico del desglose de un software.

Si existiera una funcionalidad que no perteneciera a un submódulo entonces se creará un submódulo con el nombre de “funcionalidades directas” el cual representará a las funcionalidades que no pertenecen a un submódulo. Se recomienda que al momento de registrar un producto no se le asignen versiones al producto y que se deshabilite la opción de UNSPECIIED, ya que se ha creado en java un método el cual se encarga de registrar una versión a todos los productos.

- Registrar los componentes

Las componentes representan las funcionalidades del desarrollo, tales como Login, Log Out, etc. Esto se puede ver reflejado en Ilustración 3 Ejemplo genérico del desglose de un software.

Cada componente debe pertenecer a un submódulo, los cuales simbolizan a los productos, tener en cuenta que al momento de registrar una componente debe tenerse definido quien estará a cargo de esta, y en lo más posible contar con una lista de usuarios que participan y responden por ésta. Si una componente representa un conjunto de funcionalidades, es decir, no representa un solo elemento, entonces se tomarán los elementos de dicho conjunto y se registran como componentes.

- Asignar Versiones del producto

Consiste en asignar un nombre único o darle un valor numérico el cual representa el nivel de desarrollo del producto software, por ejemplo la Universidad de Pamplona cuenta con varias versiones de Academusoft, entre ellas está Academusoft 3.2 y próximamente saldrá Academusoft 4.0.

Se trabajara con una versión conocida como **unspecified** la cual tiene como significado que no se conoce en qué versión se ha producido la falla, las versiones representan el nivel de desarrollo de todo el producto software y serán agregadas a los submódulos de acuerdo avance el nivel de desarrollo.

Utilizando el proyecto java toolToBugzilla se pueden agregar las versiones que se desean a los submódulos, la utilización de esta facilita la agregación de las versiones a todos los productos, aun así, estos pueden ser agregados manualmente a cada producto desde el Bugzilla, para el caso de los Administradores.

- Asignar Milestone (Hito) del producto

Los Milestones pueden verse como un objetivo a alcanzar, es decir, si se tiene un bug/falla o se planea agregar un nueva característica de software y se proyectó liberarlo en cierta versión, entonces se tiene un milestone, por ejemplo, la Universidad de Pamplona planea en la liberación de Academusoft 4.0 asignar características tales multi idioma, tener varias pestañas abiertas en la misma sesión, entre otras.

Como el caso de las versiones, resulta dificultoso agregar un milestone por lo que el proyecto java toolToBugzilla soluciona esta dificultad, igualmente el administrador o un usuario con los permisos suficientes puede agregar uno por uno en el Bugzilla.

3.1.3 Definir WorkFlow

La matriz de estados posibles que puede presentar un bug en este procedimiento esta especificada en la Ilustración 4 WorkFlow de un bug, allí se detalla el flujo que existe al momento de que se reporte una falla de software y la explicación del estado que se debe marcar dependiendo de la situación actual que presenta la falla.

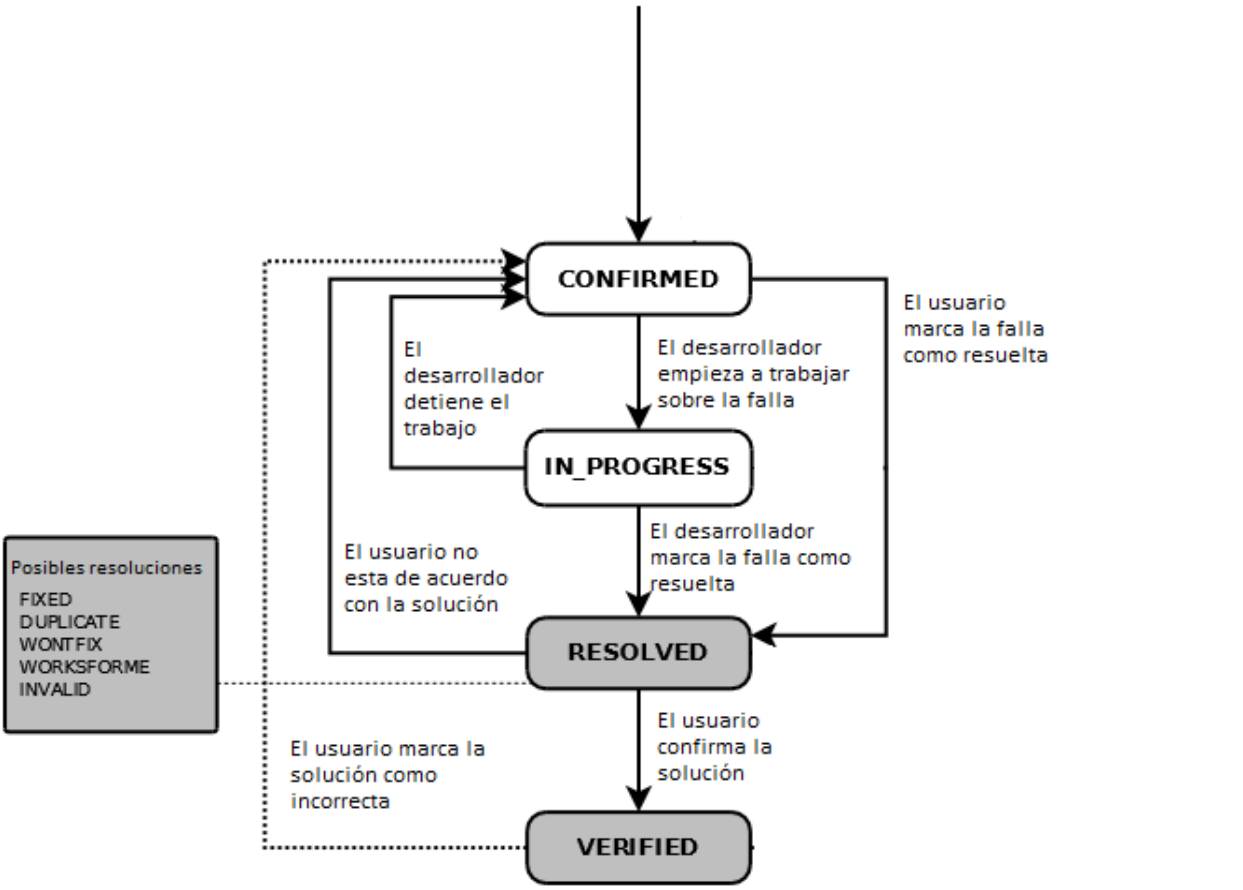


Ilustración 4 WorkFlow de un bug

Aunque aparezca el estado UNCONFIRMED al momento de utilizar Bugzilla, este no se verá al momento de asignar un nuevo estado al bug dado que esto se especifica al momento de registrar un producto.

3.1.4 Agregar usuarios

Los usuarios del proyecto pueden ser agregados tanto por el administrador como por todo aquel que quiera crear una cuenta para participar en el desarrollo.

3.1.5 Permisos de usuarios

Estos permisos representan los roles que tienen los participantes en el desarrollo del producto

software, los principales roles son:

Admin: Control total, tiene acceso a todo el funcionamiento de Bugzilla y Testopia.

Testing: Además de poder leer y escribir un bug, este puede leer y escribir planes de pruebas, casos de prueba y de ejecución.

Básico: Lee y escribe bugs, además podrá leer planes de prueba, casos de prueba y de ejecución.

Anónimo: Lee bugs.

Si el proyecto necesita que se definan otros tipos de roles, estos podrán ser agregados en el apartado de Grupos de Administración. Los usuarios básicos podrán enviar peticiones a los administradores para que se les otorguen más permisos.

3.1.6 Reporte de Bugs

Los reportes de bug o fallas están enfocados hacia los componentes, las cuales representan las funcionalidades, para ello se selecciona la clasificación, el producto y el componente.

Es esencial en el reporte que se haga que se especifique lo más minucioso posible la falla, es decir, ser específico con la versión, con el hardware, arquitectura, que las descripciones den una imagen completa del fallo y en lo más posible subir una imagen que muestre la falla.

3.1.7 Definir estado del Bug

Al momento de reportarse un bug se debe notificar al responsable del submódulo, para el caso de Bugzilla este envía un Email al responsable del submódulo para notificado del evento. El usuario a cargo del submódulo recibe el bug con el estado de CONFIRMED y de acuerdo a lo mencionado en la sección 2.3.5 (Estados de un bug en Bugzilla) este podrá ir cambiando su estado de acuerdo a la situación actual que presenta la falla.

3.1.8 Crear Plan de pruebas

El plan de pruebas determina el propósito, la estructura y métodos que deberán ser empleados en las pruebas, estos deben ser en lo más posible como lo define la norma IEEE 829-1998 para documentación de pruebas de software, es decir deben respetar como mínimo las siguientes pautas:

1) Los planes de prueba está asociado al producto es decir a submódulos para este caso.
2) Los planes de pruebas nacen de la necesidad de asociar un caso de prueba que se cree pertinente llevar a cabo o porque un usuario con los permisos necesarios lo cree pertinente.
3) Los planes de pruebas solo podrán ser creados por usuarios con roles de admin o Testers.
4) Un producto puede tener muchos planes de prueba.
5) Los planes de pruebas de tener un nombre corto.
6) Los planes de pruebas se les debe asignar una versión excepto “unspecified”.
7) Los planes de prueba deben llevar una introducción donde se especifica el propósito del plan.
8) El tipo de plan de pruebas será Funcional.
9) Los planes de pruebas deben llevar una lista de lo que se va a probar, ya sean especificación de requerimientos, guías de usuario, etc. Para este procedimiento solo se trabajarán especificación de requerimientos.
10) Los planes de pruebas deben especificar las características a ser probadas, es decir, lo que hace el sistema desde el punto de vista del usuario.
11) Los planes de pruebas deben especificar si existen lo que no se prueba, es decir, lo que hace el sistema desde el punto de vista del usuario.
12) Los planes de prueba deben especificar el entorno donde serán ejecutados.
13) Los planes de pruebas tendrán como mínimo permiso de lectura para todos los usuarios.
14) Un usuario puede enviar peticiones al creador del plan de pruebas para que se le den más permisos.
15) Si se desea detallar más el plan de pruebas puede usar como guía la plantilla formal de la IEEE 829 para planes de prueba

(http://www.ecs.csun.edu/~rlingard/comp480/TestPlanTemplate.pdf).

Para que solo se trabaje pruebas funcionales en el plan de pruebas se debe modificar la tabla test_plan_types para que solo tengan los registros de Integración, Funcional, Aceptación, Exploratorias, Regresión y Compatibilidad.

3.1.9 Crear casos de pruebas

Los casos de pruebas explican los pasos a seguir para ejecutar una prueba y los resultados esperados al momento de llevarla a cabo, los casos de pruebas deben respetar las siguientes pautas:

1. Los casos de prueba están enfocados hacia las componentes, es decir, funcionalidades.
2. Debe existir por lo menos un caso de prueba por componente por cada versión de lanzamiento que exista o falla solucionada sobre la componente.
3. Los casos de pruebas nacen de la necesidad de asociar un bug con caso de prueba o porque un usuario con los permisos necesarios lo cree pertinente.
4. Un caso de prueba sólo será diseñado para una componente.
5. Un caso de prueba debe tener como mínimo un plan de prueba.
6. Los casos de pruebas pueden ser compartidos entre planes de pruebas.
7. Los casos de prueba pueden estar relacionados a uno o más bugs.
8. Los casos de prueba nuevos por defecto tendrán como estado Confirmado.
9. Un caso de prueba tiene dos asignaciones especiales Prioridad y Categoría.
10. Un caso de prueba puede depender de otro caso de prueba.
11. Los casos de pruebas deben llevar una lista de lo que se va a probar, ya sean especificación de requerimientos, guías de usuario, etc.
12. Los casos de pruebas deben especificar una lista de las entradas requeridas para ejecutar el caso de prueba, estas entradas pueden ser datos, acciones humanas, precondiciones, etc.
13. Los casos de pruebas deben especificar las salidas para verificar el caso de prueba, estas salidas pueden ser datos, archivos, precondiciones, etc.
14. Los casos de pruebas deben especificar en qué ambientes serán ejecutados

15. Si se desea detallar más los casos de pruebas puede usar como guía la plantilla formal de la IEEE 829 para casos de pruebas (http://www.ufjf.br/eduardo_barrere/files/2011/06/SQETestCaseSpecificationTemplate.pdf).

3.1.10 Crear Build

Al comenzar el proyecto se debe asignar un Build inicial el cual representa el origen del proyecto, estos están estrechamente relacionados con los milestones, a medida que se hagan test run estos se irán agregando, los Build están muy relacionados con los repositorios que manejan el proyecto, por ejemplo, si se utiliza un repositorio Git se debe colocar como nombre la rama en la que se hace la prueba acompañada del identificador del commit y en la descripción la fecha.

3.1.11 Crear Categorías

Testopia cuenta con una categoría inicial conocida como --default--, para este procedimiento se definirán las siguientes categorías y podrán ser añadidas de acuerdo a las necesidades del desarrollo, siempre y cuando se piense en un Bugzilla para un Testopia.

1. **--default--**: categoría por defecto del Testopia, esta es añadida a un caso de prueba siempre y cuando no exista otra categoría que se ajuste a la necesidad del caso de prueba.
2. **Semanal:** Esta categoría es añadida a los casos de prueba que tienen como finalidad ser ejecutados cada semana.
3. **Inmediato:** Esta categoría fue diseñada para los casos de pruebas que deben ser ejecutados lo más pronto posible.

Con el proyecto de java toolToBugzilla se pueden agregar fácilmente las categorías definidas, aun asi el administrador puede ir agregando cada uno de estos a cada submódulo definido.

3.1.12 Crear ambientes/entornos

Estos representan en donde se van a ejecutar los casos de pruebas, como es un proyecto de software basado en web, se crearon ambientes en de ejecución de los navegadores más utilizados,

estos son: Firefox, google Chrome, Microsoft Edge y All (representa a todos los navegadores), de acuerdo a las necesidades del proyecto se podrán ir agregando los ambientes. El proyecto java toolToBugzilla simplifica este procedimiento, aun así, estos podrán ser agregados uno a uno desde la sección de Testopia.

3.1.13 Crear ejecución de prueba

Una ejecución de prueba reúne todos los componentes configurados anteriormente, es decir, en esta se asignan Builds, casos de pruebas que a su vez pertenecen a un plan de prueba, categorías las cuales están asociadas a los milestones y los ambientes o entornos. Es importante definir en esta ejecución de prueba si esta será manual o automática. Una ejecución de prueba puede ejecutar múltiples casos de pruebas siempre y cuando pertenezcan al mismo plan de pruebas.

3.1.14 Ejecutar pruebas

Se asignan nuevos estados a la ejecución del caso de prueba, puede tomar las siguientes asignaciones: Marcada como no iniciada, Marcada como pasada, Marcada como fallida, Marcada en ejecución, Marcada como pausada, Marcada como bloqueada, Marcada como error, dependiendo del estado en que se encuentra la ejecución.

3.1.15 Generar reportes

Los reportes pueden ir los diferentes estados que pueda tener todos los bugs, a planes de prueba, casos de pruebas, ejecución de pruebas, caso de ejecución, éstos pueden ser genéricos, específicos o trabajos en progreso, serán creados cada cuanto sean necesarios por los usuarios con los privilegios suficientes.

3.1.16 Mantenimiento

- Hacer backups de base de datos asignada a Bugzilla cada semana.
- Actualizar Bugzilla (Si es pertinente).
- Actualizar Testopia (Si es pertinente).

3.2 Validación del procedimiento

Para validar el procedimiento se trabajó sobre el desarrollo de un producto software basado en web, en este proyecto se diseñó un modelo entidad relación que gestiona un caso hipotético de inscripciones a cursos gratis dados por el ministerio de las TIC, se diseñaron requerimientos funcionales iniciales, y se implementó el procedimiento para que fuese Testopia quien hiciera la gestión de pruebas en el ambiente de desarrollo colaborativo y distribuido.

El modelo entidad relación es el siguiente:

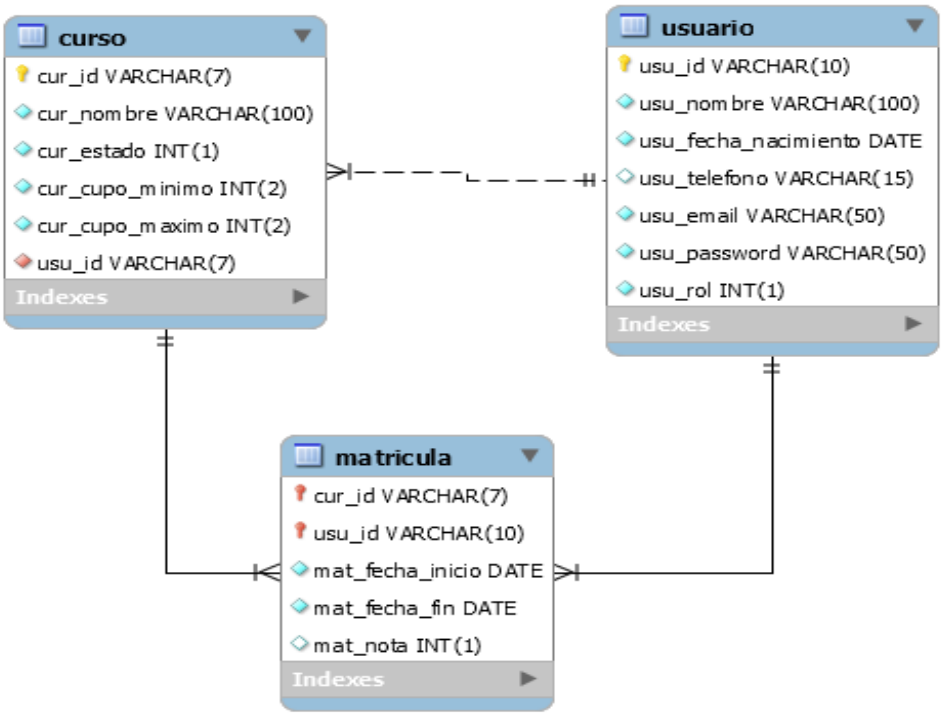


Ilustración 5 Modelo Entidad Relación cursoslibres

3.2.1 Instalación y configuración

Para realizar la validación del procedimiento planteado anteriormente se puso en marcha en los servidores de software libre de la Universidad de Pamplona la instalación y configuración de las

herramientas de software mencionadas en la sección 3.1.1 Instalación y Configuración de herramientas es decir Bugzilla junto con Testopia para gestionar las pruebas funcionales del proyecto de cursosLibres.

3.2.2 Configuración de Bugzilla

Luego se procedió a configurar el Bugzilla de acuerdo al procedimiento, es decir, lo primero fue asignar clasificaciones, productos, componentes, versiones y milestones, estas dos últimas fueron agregadas con toolToBugzilla. La siguiente imagen muestra cómo se organizó el proyecto de cursosLibres para poder gestionar las pruebas funcionales sobre éste:

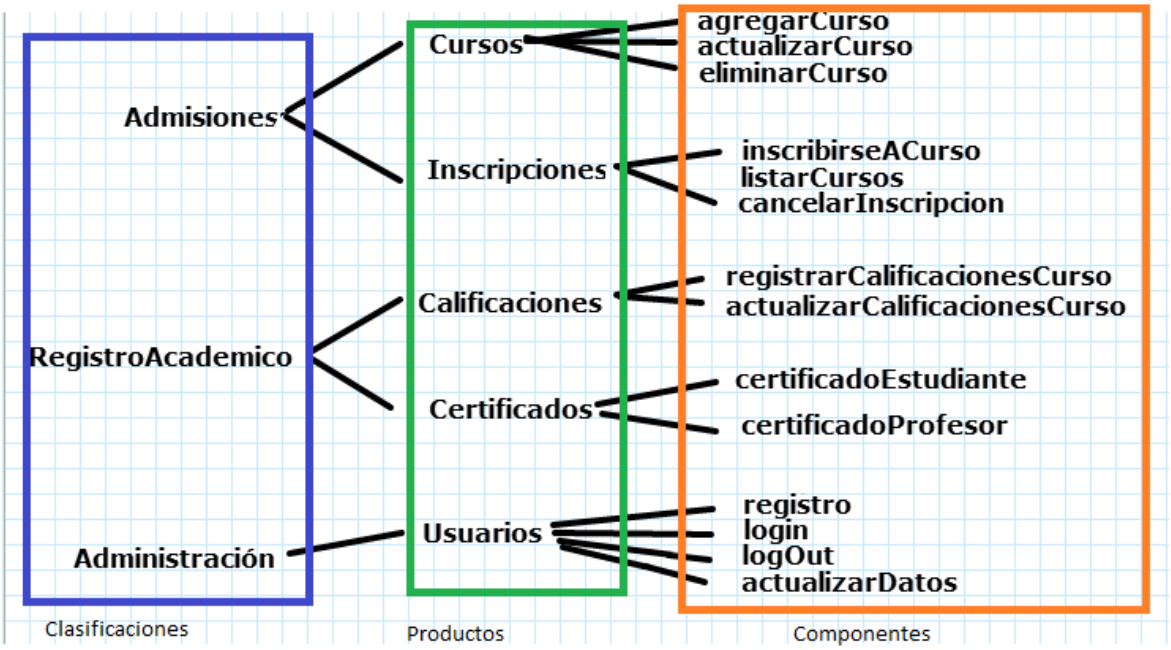


Ilustración 6 Configuración de Bugzilla para cursosLibres

La Ilustración 7 muestra cómo se ve en Bugzilla la configuración de las clasificaciones, la Ilustración 8 muestra al producto Usuarios configurado en el Bugzilla y por último la Ilustración 9 muestra las componentes del producto Usuario que de acuerdo a Ilustración 6 son: registro, login, logOut y actualizarDatos.

Edit Classification ...	Description	Sortkey	Products	Action
Administración	Esta clasificación esta acargo de todo lo relacionado al perfil de usuario.	0	reclassify (1)	delete
Admisiones	Esta clasificación esta acargo de registrar nuevos cursos, actualizarlos, etc. Además cuenta con los procesos de inscripciones por parte de los usuarios a los cursos, cancelar la inscripcion, etc.	0	2	
Registro Academico	Esta clasificación esta acargo de registrar calificaciones a los estudiantes, de otorgar certificaciones tanto a profesores como estudiantes.	0	reclassify (2)	delete
Add a new classification				Add

Ilustración 7 Clasificaciones cursoslibres

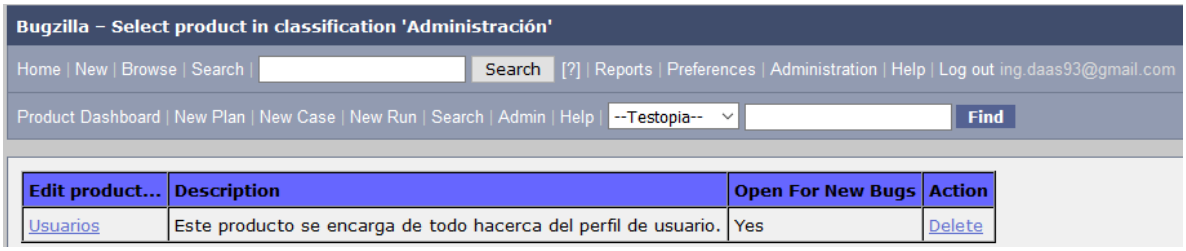


Ilustración 8 producto Usuario de la clasificación Administración

Edit component...	Description
actualizarDatos	Esta funcionalidad esta acargo de que el usuario pueda modificar sus datos en todo lo referente a su perfil.
login	Esta funcionalidad se encarga de generar una sesión de usuario para que este pueda acceder a sus privilegios.
logOut	Esta funcionalidad esta acargo de sacar de sesión a un usuario
registro	Esta funcionalidad se encarga de registran tantos docentes como estudiantes

Add a new component to product 'Usuarios'

Ilustración 9 Componentes del producto Usuarios

3.2.3 Definir WorkFlow

Lo siguiente fue definir el WorkFlow que se utiliza para llevar el seguimiento de las fallas del producto software, de acuerdo a la Ilustración 4 WorkFlow de un bug, se procedió a configurar el Bugzilla de acuerdo a la Ilustración 10.

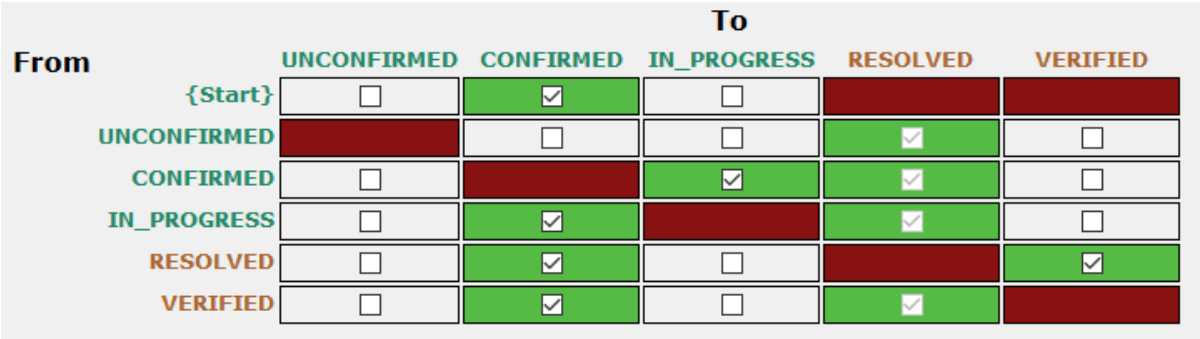


Ilustración 10 matriz WorkFlow de estado del Bug

3.2.4 Agregar usuarios

Lo siguiente fue contar con usuarios para que participaran en el desarrollo de cursosLibres y usuarios que lo utilizaran, esto se ve reflejado en la Ilustración 11.

Edit user...	Real name	Account History	Action
andmen0212@hotmail.com	Andres	View	Delete
daas-93@hotmail.com	Armando	View	Delete
freddy.sandoval@unipamplona.edu.co	Freddy	View	Delete
ing.daas93@gmail.com	Diego Aguilar	View	Delete

Ilustración 11 Usuarios del Bugzilla del proyecto cursosLibres

3.2.5 Permisos de usuarios

Dada la existencia de usuarios en el Bugzilla para el proyecto de cursosLibres se prosiguió a darle privilegios de usuario a los responsable de las componentes, es decir de las funcionalidades, por ejemplo, la Ilustración 12 Usuario con permiso de Testing muestra al usuario Andrés con los privilegios de Testing ya que este está a cargo de la componente inscribirseACurso.

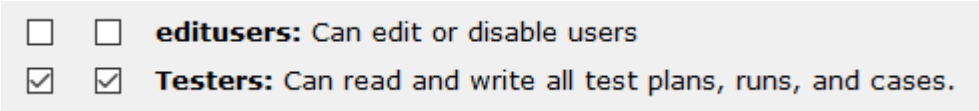


Ilustración 12 Usuario con permiso de Testing

3.2.6 Reporte de Bugs

Después de haber configurado el Bugzilla y darle privilegios a los usuarios llegó un reporte de un bug en el componente login, el cual se programó con la intención para que tuviera una falla con la sesión del usuario y ver si alguien de la comunidad lo notaba.

El fallo consistió en que no se removía de sesión el atributo "name" línea 37 de la tabla Tabla VIII LoginServlet y cuando se diera clic al botón atrás después de haberse logueado correctamente, el formulario dejaba entrar a un usuario con email y password inválidos ya que existe una sesión creada.

Tabla VIII LoginServlet

```
1.
2.
3.  import DB.Conexion;
4.  import java.io.IOException;
5.  import java.io.PrintWriter;
6.  import java.sql.ResultSet;
7.  import java.sql.SQLException;
8.  import java.util.logging.Level;
9.  import java.util.logging.Logger;
10.
11. import javax.servlet.ServletException;
12. import javax.servlet.http.HttpServlet;
13. import javax.servlet.http.HttpServletRequest;
14. import javax.servlet.http.HttpServletResponse;
15. import javax.servlet.http.HttpSession;
16. public class LoginServlet extends HttpServlet {
17.  Conexion conection;
```

```
18.
19.  public LoginServlet() throws ClassNotFoundException, InstantiationException,
     IllegalAccessException, SQLException {
20.  this.conection = new Conexion();
21.  }
22.  protected void doPost(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {
23.  response.setContentType("text/html");
24.  PrintWriter out=response.getWriter();
25.  //request.getRequestDispatcher("link.html").include(request, response);
26.
27.
28.  String email=request.getParameter("email");
29.  System.out.println(email);
30.  String password=request.getParameter("password");
31.  int rol= Integer.parseInt(request.getParameter("rol"));
32.
33.  String sql="SELECT * FROM usuario as u WHERE u.usu_email='"+email+"' and
     u.usu_password='"+password+"'";
34.
35.
36.  HttpSession session=request.getSession();
37.  // session.removeAttribute("name");
38.  session.removeAttribute("rol");
39.  try {
40.  ResultSet resultSet = conection.consultalSQL(sql);
41.  while (resultSet.next()) {
42.  session.setAttribute("name",resultSet.getString("usu_nombre"));
43.  session.setAttribute("rol", rol);
44.  }
45.  } catch (SQLException ex) {
```

```
46.  Logger.getLogger(LoginServlet.class.getName()).log(Level.SEVERE, null, ex);
47.  }
48.
49.
50.  if(session.getAttribute("name")!=null){
51.
52.  if(rol==0) response.sendRedirect("indexUsuario.jsp");
53.  if(rol==1) response.sendRedirect("indexProfesor.jsp");
54.
55.  }
56.  else{
57.  String respuesta="";
58.  if(rol==1) respuesta="estudiante";
59.  if(rol==0) respuesta="profesor";
60.  out.print("Disculpe pero usted no es un "+respuesta);
61.  // response.sendRedirect("LoginSer");
62.  request.getRequestDispatcher("login.html").include(request, response);
63.  }
64.  out.close();
65.  }
66.
67. }
68.
```

3.2.7 Definir estado del Bug

Después de esta notificación por Email, se decidió cambiar el estado de la falla a IN_PROGRESS ya que se decidió corregir este bug.

3.2.8 Crear Plan de pruebas

Después de cambiar el estado del bug a IN_PROGRESS, se decidió agregar un plan de pruebas de acuerdo a lo especificado en el procedimiento para poder agregar los casos de pruebas pertinentes, este plan de pruebas fue realizado al producto Usuarios el cual está a cargo de las componentes login y otras funcionalidades. De acuerdo al procedimiento se agregó al plan de pruebas los siguientes datos.

Plan name: Plan_Usuarios

Product: Usuarios

Plan type: Function

Product version: 1.0

Estos campos son los requeridos directamente por el Testopia al momento de generar un nuevo plan de pruebas, lo importante radica en el plan del documento, en este, según el procedimiento debe llevar como mínimo una introducción especificando el propósito de la prueba, características a probar, que según lo planteado siempre será especificación de requerimientos, las características para testear y el ambiente, eso son los elementos mínimos que debe llevar el plan de pruebas. Por lo siguiente éste se configura de la siguiente forma en el Testopia del Bugzilla.

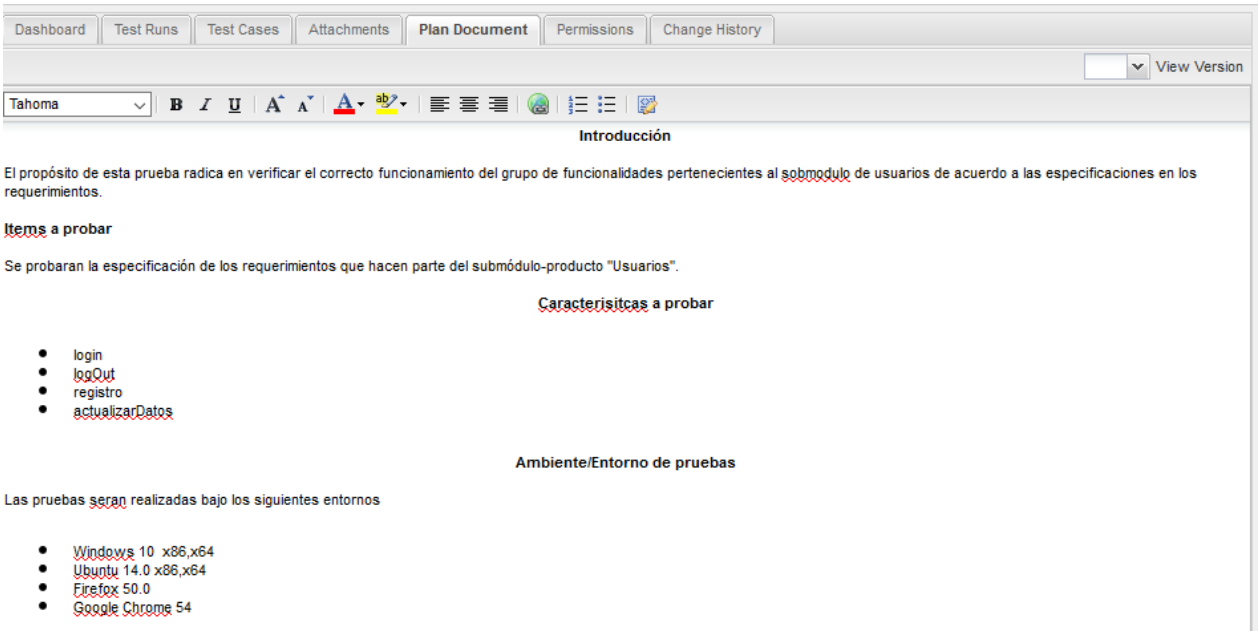


Ilustración 13 Plan de pruebas "Plan_Usuarios"

Inmediatamente después de haber creado el plan de pruebas conforme a lo descrito en el procedimiento, el usuario debe asignar con una expresión regular en la pestaña de permiso la autorización a todos los usuarios para que como mínimo puedan tener derechos de lecturas sobre éste.

Ilustración 14 Asignación de permisos sobre un Plan de Pruebas

3.2.9 Crear casos de pruebas

Luego se crearon dos casos de pruebas de acuerdo a la especificación del procedimiento, el primero estaba fundamentado en validar que un estudiante existente en la base de datos puede tener

acceso a una sesión en el sistema, y el otro se creó para ser asociado al reporte del bug.

Lo siguiente fue ir al proyecto de desarrollado de cursosLibres y en la clase ServletLogin (ver Tabla VIII LoginServlet) en la línea 37 quitar el comentario, para solucionar la falla.

3.2.10 Crear Build

Como se tenía la intención de crear una ejecución de prueba que implementara los casos de pruebas mencionados anteriormente, fue necesario crear primero los Builds, el primer Build se creó con el nombre de la rama "master" y el commit que inicio el desarrollo, lo mismo se hizo para cuando se solucionó el bug del login, se creó un Build con la misma rama "master" pero con un commit diferente.

3.2.11 Crear Categorías

Al igual que el Build, ésta debe ser agregada al iniciar el proyecto para poder crear una ejecución de prueba, por lo cual se agregaron las categorías mencionadas en 3.2.11 Crear Categorías pero con la ayuda del desarrollo implementado en java, para facilitar el registro de estos, este es toolToBugzilla, la Ilustración 15 Categorías en Testopia del desarrollo cursosLibres muestra estas categorías registradas en Testopia.

Name	Description ▲
Semanal	Casos de pruebas ejecutados cada semana
Inmediato	Casos de pruebas ejecutados lo mas pronto posible
--default--	Categoria por defecto

Ilustración 15 Categorías en Testopia del desarrollo cursosLibres

3.2.12 Crear ambientes/entornos

Este es el último de los elementos que se deben tener configurados en Testopia para poder lograr el cometido de crear una ejecución de pruebas, es por esto, que con la ayuda de tooToBugzilla se prosiguió a registrar los entornos donde se ejecutaron los casos de pruebas de acuerdo a lo definido en la sección 3.1.12 Crear ambientes/entornos (ver Ilustración 16 Entornos para casos de pruebas

del desarrollo cursosLibres)

ID	Environment Name	Run Count	Active
20	All	2	☑
5	Firefox	0	☑
10	Google Chrome	0	☑
15	Microsoft Edge	0	☑

Ilustración 16 Entornos para casos de pruebas del desarrollo cursosLibres

3.2.13 Crear ejecución de prueba

Después de haber definido los Builts, Categorías y Entornos, ya se estaba listo para poder construir una ejecución de prueba al caso de prueba relacionado al bug, para ello se necesitó de todos los pasos anteriores para poder definirlo, es decir, una ejecución de prueba congrega todos los elementos descritos en los pasos anteriores, además se especificó el tipo de prueba que se llevó a cabo, es decir, se planeó realizar la ejecución de la prueba manualmente.

3.2.14 Ejecutar pruebas

Como ya se estaba listo para ejecutar la prueba del caso de prueba que está relacionado con el bug que describe una falla al momento de tener una sesión activa, y regresar al Login, se decidió realizar manualmente la prueba de acuerdo a lo descrito en el paso anterior, efectivamente esta ejecución cumplió con lo establecido en el caso de prueba por lo que se cambió el estado del bug a RESOLVER, FIXED y de la ejecución de la prueba como pasada.

3.2.15 Generar reportes

Se decidió generar un reporte de los casos de pruebas que estaban planteados sobre el producto Usuarios y que se hicieran sobre la componente Login y que además tuvieran como tipo de prueba una ejecución manual, la Ilustración 17 Reporte de casos de pruebas del desarrollo cursosLibres muestra estos casos de pruebas.

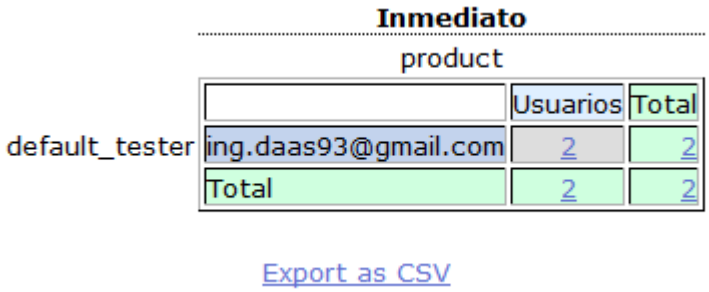


Ilustración 17 Reporte de casos de pruebas del desarrollo cursosLibres

3.2.16 Mantenimiento

Por último se descubrió que ya existe una nueva versión del Bugzilla, la 5.0 pero aunque el desarrollo de Testopia en su forja de GitHub diga que ya tienen un Testopia compatible con esta versión aún no ha habido un lanzamiento, por lo que no es conveniente actualizar Bugzilla hasta que lancen la nueva versión del Testopia. Además se ha venido salvaguardando la información con backups de la base de datos Bugzilla, la cual está asociada a la herramienta Bugzilla.

4 Conclusiones, recomendaciones y trabajos futuros

4.1 Conclusiones

La implementación del Bugzilla 4.2 con Testopia 2.5 facilitó el desarrollo del proyecto cursoslibres ya que al momento de definir los casos pruebas surgió uno el cual nunca se había pensado, es decir, dados los requerimientos no se pensó en desarrollar un caso de prueba en la cual se tuviera una sesión activa, regresar a la página de login y verificar que no dejara avanzar en la sesión. Estos errores son muy comunes pero difícilmente son añadidos en un caso de prueba, esto refleja que en las comunidades de software libre u Open Source si garantiza la calidad del desarrollo del producto software.

Al observar el procedimiento se puede apreciar que al definir un plan de pruebas y un caso de prueba estos están basados en el estándar IEEE 829 1998 Documentación de pruebas de software, lo cual garantiza la calidad de las pruebas al momento de definirlas y ejecutarlas, esto reafirma la inquietud de sí un desarrollo colaborativo y distribuido garantiza la calidad de los desarrollos.

Se ve claramente que desarrollar un procedimiento para la gestión de pruebas de software basado en web en ambientes de desarrollo colaborativo y distribuido es totalmente factible, ya que existe autores como (Bruegge & Dutoit, 2002) quienes plantean en el libro de ingeniera de software orientada a objetos que debe existir en un seguimiento al momento de producirse una falla como un mecanismo o técnica que lleve un seguimiento de esta , lo cual hace a la perfección Bugzilla, el software ampliamente usado por comunidades de software libre. Además Bugzilla puede ser muy personalizable para ajustarse a las necesidades de un proyecto de desarrollo, tal como este caso, en donde se redefinieron conceptos tales como clasificaciones, productos y componentes. Estos atributos no fueron necesariamente diseñados para otros propósitos, aun así, Bugzilla da la posibilidad de poder utilizar cada uno de ellos para que se ajusten al desarrollo del producto, asegurando la calidad del desarrollo.

Al momento de la verificación del procedimiento, es decir, la puesta en marcha del proyecto de cursoslibres, se observó que los entornos socioculturales de la universidad de Pamplona si están disponibles para que las áreas relacionadas al desarrollo de software puedan participar en estos,

prueba de ello, fue la facilitación de los servidores de software libre de la Ingeniería de sistemas para implementar el proyecto cursoslibres.

Tal como lo dice Fuggetta, la creación de software en comunidades open source, de software libre o ambientes de desarrollo colaborativo y distribuido no es una nueva definición de los procesos para crear software, sino que simplemente son una nueva perspectiva de estos procesos, esto se pudo percibir en la comunidad open source de Yocto Project y cursoslibres el cual inició su desarrollo en los laboratorios de software libre de la universidad de Pamplona, al igual que en Academusoft 4.0 un software propietario, cuentan con gestión de pruebas pero desde una perspectiva diferente del desarrollo tradicional, aun así los mencionados aseguran la calidad del desarrollo.

4.2 Recomendaciones

Al CIADTI de la Universidad de Pamplona, para que gestione Academusoft 4.0 con Bugzilla, para que la comunidad que lo utiliza, tanto administrativos, desarrolladores, estudiantes, profesores, etc. puedan reportar fallas al momento de utilizar cualquiera de sus componentes.

Al departamento de ingeniería de sistemas para que involucre más a los ingenieros con desarrollos de software libre u Open Source.

4.3 Trabajos Futuros

A partir de este procedimiento se puede implementar un proyecto de software basado en web que no solo gestione la pruebas funcionales del software por medio de Testopia, sino que se haga una adaptación del Bugzilla para que este pueda administrar todo el desarrollo del producto, es decir, que existan secciones para capturas de requerimientos, esto significa colocar a la disponibilidad de la comunidad que participa en el desarrollo de sugerir nuevas funcionalidades, tener una sección de diseños, de implementación y de pruebas, y que se les den a ciertos grupos de usuarios permisos para que se hagan responsables de ellas.

Este procedimiento también se puede integrar a un Bugzilla para que las pruebas de software vayan enfocadas a las ramas de un repositorio como Git y que en éstas no solo se hagan a pruebas funcionales sino que también pruebas no funcionales.

5 Referencias bibliográficas

5.1 Bibliografía

Acuña Castillo, S. T., Castro, J. W., & Dieste, O. (2012). Diferencias entre las Actividades de Mantenimiento en los Procesos de Desarrollo Tradicional y Open Source, 651–664. Retrieved from http://sistedes2012.ual.es/

Bartolomé, C. L. (2014). TRABAJO FIN DE GRADO El Desarrollo de Software Open Source Analizado desde Dentro.

Bourque, P., & Fairley, R. E. (2014). *Guide to the Software Engineering - Body of Knowledge. IEEE Computer Society*. http://doi.org/10.1234/12345678

Bruegge, B., & Dutoit, A. (2002). *Ingeniería de sofware orientada a objetos*. México: PEARSON EDUCACION,.

Crowston, K., Howison, J., & Wiggins, A. (2010). Free/Libre Open Source Software Development: What We Know and What We Do Not Know. *ACM Computing Surveys*, *40*(2), 1–37. http://doi.org/10.1145/2089125.2089127

Dinh-Trong, T. T., & Bieman, J. M. (2004). Open Source Software Development: A Case Study of FreeBSD. *Ieee Metrics*, (Metrics), 96–105.

FSF, F. S. F. (2016). www.fsf.org. Retrieved from http://www.fsf.org/about/what-is-free-software

Fuggetta, A. (2003). Open source software - An evaluation. *Journal of Systems and Software*, *66*(1), 77–90. http://doi.org/10.1016/S0164-1212(02)00065-1

Ishigetani, O. M. (2014). Proceso De Requisitos En El Desarrollo De Open Source Software, 55357.

Jhonatan Enrique Cuevas Obregon. (2005). Estudio sobre el uso de software libre de la universidad de Pamplona.

5.2 Infografía

(http://br1an6wp-br1an6.rhcloud.com/?p=53,2016).

(https://programacionwebisc.wordpress.com/2-1-arquitectura-de-las-aplicaciones-web/, 2016).

(https://es.wikipedia.org/wiki/Pruebas_funcionales, 2016).

(http://carolina.terna.net/ingsw3/datos/Pruebas_de_Soporte.pdf ,2016).

(http://wikitel.info/wiki/Interoperabilidad, 2016).

(http://carolina.terna.net/ingsw3/datos/Pruebas_de_Desempe%F1o.pdf, 2016).

(http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/moreno_a_jl/capitulo5.pdf , 2016).

(https://wiki.openoffice.org/wiki/OOoES/Calidad/Gestion_de_la_Calidad:_Campos_de_un_bug#Other_Fields, 2016).

(http://docs.devzing.com/testopia-training/, 2016).

6 Anexos

6.1 Anexo 1

Instalación de bugzilla y testopia en servidores de software libre de la Universidad de Pamplona en Debian 8.

1. Tener previamente instalado Mysql y Apache2
2. Ubicarse en la ruta del apache con la instrucción **cd /var/www/html**
3. Ingresar a Mysql con la instrucción **mysql -u root -p**
4. Crear base de datos con la instrucción **create table bugzilla;**
5. Dar permisos al usuario sobre la base de datos **GRANT ALL PRIVILEGES ON * . * TO 'root'@'localhost';**
6. Refrescar los privilegios con la instrucción **FLUSH PRIVILEGES**; y salir del Mysql.
7. Configurar el archivo **/etc/mysql/my.cnf** en la sección [mysqld] a **max_allowed_packet=4M** y **ft_min_word_len=2.**
8. Configurar Apache2 con la instrucción **/etc/apache2/sites-available/bugzilla** de la siguiente forma:

```
<Directory <b>/var/www/bugzilla-3.0</b>>
    AddHandler cgi-script .cgi
    Options +Indexes +ExecCGI
    DirectoryIndex index.cgi
    AllowOverride Limit
</Directory>
```

9. Descargar Bugzilla 4.2 de https://ftp.mozilla.org/pub/webtools/

10. Descargar Testopia 2.5 de https://ftp.mozilla.org/pub/webtools/testopia/
11. Copiar Bugzilla 4.2.2 y Testopia 2.5 al directorio actual.
12. Descomprimir el Bugzilla con la instrucción **tar zxvf bugzilla-4.2.2.tar.gz**
13. Renombrar la carpeta de bugzilla con la instrucción **mv bugzilla-4.2.2 bugzilla**
14. Ubicarse en el directorio de bugzilla con la instrucción **cp bugzilla**
15. Ejecutar el archivo oculto para chequear módulos faltantes con la instrucción **./checksetup.pl --check-modules**
16. Instalar los módulos faltantes con instrucción **./install-module.pl --all**
17. Volver a ejecutar el archivo **./checksetup.pl** pedira que que se configure en archivo localconfig, para ello hay que definir los siguientes valores:
    a. $webservergroup = 'www-data';
    b. $db_name = 'bugzilla';
    c. $db_user = 'root';
    d. $db_user = 'passwor_your_bd';
18. Descomprimir Testopia con la instrucción **tar zxvf testopia-2.5-BUGZILLA-4.2.tar.gz**
19. Ejecutar **perl -MCPAN -e shell**
20. Ejecutar **install Email::Send::Gmail**
21. Editar Miler.pm con la instrucción nano **Bugzilla/Mailer.pm** agregar los siguientes valores:
    a. Después de la última línea que empieza con **use** agregar las siguientes líneas

       use

       Email::Send::Gmail;
       use Email::Simple::Creator;

    b. Encontrar la línea **if ($method eq "SMTP")** y reemplazarla por **if ($method eq "SMTP" || $method eq "Gmail")**
22. Restablecer servicios Mysql y Apache2 con instrucción **sudo service restart apache2** y **sudo service restart mysql**
23. Volver a ejecutar Volver a ejecutar el archivo **./checksetup.pl**
24. Abrir en el navegador el apache con /bugzilla, **ingsistemas.uniPamplona.edu.co/Bugzilla.**