UNIVERSIDAD PRIVADA ANTENOR ORREGO

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA DE COMPUTACIÓN Y SISTEMAS

TESIS PARA OBTENER EL TÍTULO PROFESIONAL DE

INGENIERO DE COMPUTACIÓN Y SISTEMAS

CONSTRUCCIÓN DE UNA SOLUCIÓN CLOUD COMPUTING PARA FACILITAR LA ADOPCIÓN DEL PROCESO PERSONAL DE SOFTWARE EN EL DESARROLLO DE SOFTWARE

AUTORA:

Br. Flor María de Fátima Flores Jáuregui

ASESOR:

Ing. Wilder Adán Namay Zevallos

Trujillo – Perú, 2015

Nro. REGISTRO _____

TESIS: "CONSTRUCCIÓN DE UNA SOLUCIÓN CLOUD COMPUTING PARA FACILITAR LA ADOPCIÓN DEL PROCESO PERSONAL DE SOFTWARE EN EL DESARROLLO DE SOFTWARE"

Elaborado por:

Bach. Flor María de Fátima Flores Jáuregui

_____

Jaime Eduardo Díaz Sánchez
Presidente
CIP: 73304

_____

Walter Aurelio Lazo Aguirre
Secretario
CIP: 36034

_____

Carlos Alberto Gaytán Toledo
Vocal
CIP: 84519

:

_____

Wilder Adán Namay Zevallos
Asesor
CIP: 130945

PRESENTACION

Señores Miembros del Jurado

De conformidad con los requisitos estipulados en el Reglamento de Grados y Títulos de la Universidad Privada Antenor Orrego y Reglamento de Grados y Títulos de la Facultad de Ingeniería, para obtener el Título Profesional de Ingeniero de Computación y Sistemas, sometemos a vuestra consideración la Tesis titulada:

"CONSTRUCCIÓN DE UNA SOLUCIÓN CLOUD COMPUTING PARA FACILITAR LA ADOPCIÓN DEL PROCESO PERSONAL DE SOFTWARE EN EL DESARROLLO DE SOFTWARE"

Este trabajo de investigación, es el resultado del esfuerzo, donde se ha plasmado todos los conocimientos y experiencias adquiridas a lo largo de mi formación profesional, complementado además con la orientación y apoyo de mi asesor y las revisiones del jurado calificador.

Atentamente,

Bach. Flor María de Fátima Flores Jáuregui

DEDICATORIA

A Dios, por iluminarme en este largo camino que aún falta recorrer, por darme fuerza y perseverancia para salir adelante y continuar con mis objetivos trazados.

A mis padres Jorge y Martha por su gran amor y apoyo incondicional, sus sabios consejos y porque nunca dejaron de luchar para ver a mis hermanos y a mí salir adelante.

A mis hermanos por estar siempre en los buenos y malos momentos, y por confiar en mí, guiándome con sus buenas experiencias.

A mis sobrinos Jorge y Xiomara porque ellos son el motivo por cual yo quiero seguir adelante para ser un gran ejemplo hacia ellos.

A mi familia y amigos por el apoyo e interés que me dieron para el bienestar personal y familiar.

La autora.

AGRADECIMIENTO

Agradezco infinitamente a mis padres quienes depositaron sus esfuerzos y esperanzas en mí desde el inicio hasta la actualidad, gracias a ellos soy lo que soy hoy por hoy. Los amo infinitamente y estaré eternamente agradecida.

A mi asesor Wilder Adán Namay Zevallos por su asesoramiento profesional a lo largo del desarrollo de esta tesis, quien me proporciono ideas, guías y me inculcó a no darme por vencida antes de tiempo.

A los alumnos del Décimo Ciclo del año académico 2014-II por su tiempo y apoyo para poder llevar a cabo mi trabajo.

La autora.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

CONSTRUCCIÓN DE UNA SOLUCIÓN CLOUD COMPUTING PARA FACILITAR LA ADOPCIÓN DEL PROCESO PERSONAL DE SOFTWARE EN EL DESARROLLO DE SOFTWARE

Por el Br.

Br. Flor María de Fátima Flores Jáuregui

En el presente trabajo de investigación pre experimental se analiza la adopción del Proceso Personal de Software (PSP) por parte de los estudiantes del Décimo Ciclo de la Carrera Profesional de Ingeniería de Computación y Sistemas de la Universidad Privada Antenor Orrego, para lo cual construimos una solución Cloud Computing utilizando Google App Engine para automatizar las tareas del nivel 1.1 del PSP y evaluar el impacto de esta herramienta en la adopción de las practicas que el PSP sugiere tales como el registro de tiempos, defectos, sucesos ocurridos durante el desarrollo de software, estimación de tiempo y tamaño estimado de programas, entre otros.

En la construcción se utilizó la metodología ágil ICONIX, el Lenguaje Unificado de Modelado, el entorno de desarrollo NetBeans 6.9 configurado para soportar JDO y Google App Engine, para lograr lo mencionado anteriormente se tuvo como base las fases del nivel 0 del Proceso Personal de Software.

Para contrastar la hipótesis se creó un caso de trabajo y se aplicó un test cuyos resultados fueron procesados con el diseño de Pretest y Postest el cual permito evaluar la contribución de las variables definidas en el diseño de hipótesis en la adopción del PSP.

Los resultados finales obtenidos indican que la herramienta, estadísticamente, facilita significativamente la adopción del PSP por parte de los estudiantes en proceso de formación.

ABSTRACT

BUILDING A CLOUD COMPUTING SOLUTION TO FACILITATE DECISION OF PERSONAL SOFTWARE PROCESS IN SOFTWARE DEVELOPMENT

By Br.

Br. Flor María de Fátima Flores Jáuregui

In this paper pre research experimental adoption of Personal Software Process (PSP) is analyzed by the students of the Tenth Cycle of the Career of Computing and Systems Universidad Privada Antenor Orrego, for which we built a Cloud Computing solution using Google App Engine to automate tasks PSP 1.1 level and assess the impact of this tool in adopting practices that suggests PSP such as time recording, defects, events during software development, estimated time and estimated size of programs, among others.

Agile ICONIX, the Unified Modeling Language, the development environment NetBeans 6.9 configured to support JDO and Google App Engine was used in construction, to achieve the above is was based on the phases of the level 0 of the Personal Software Process .

To test the hypothesis a case of work was created and a test whose results were processed using the pretest and posttest design which allow to evaluate the contribution of the variables defined in the design assumptions in the adoption of PSP was applied.

The final results indicate that the tool statistically significantly facilitates the adoption of PSP by students in training.

I. INTRODUCCIÓN

El presente proyecto está enfocado en el área de Sistemas de Información, orientándose más a la aplicación, despliegue y configuración de tecnologías basados en temas sobre sistemas organizacionales. Por lo que se desarrollará una herramienta para ver la adopción del Proceso de Desarrollo Personal (PSP) durante el proceso de desarrollo de software, usando la tecnología cloud Google APP Engine.

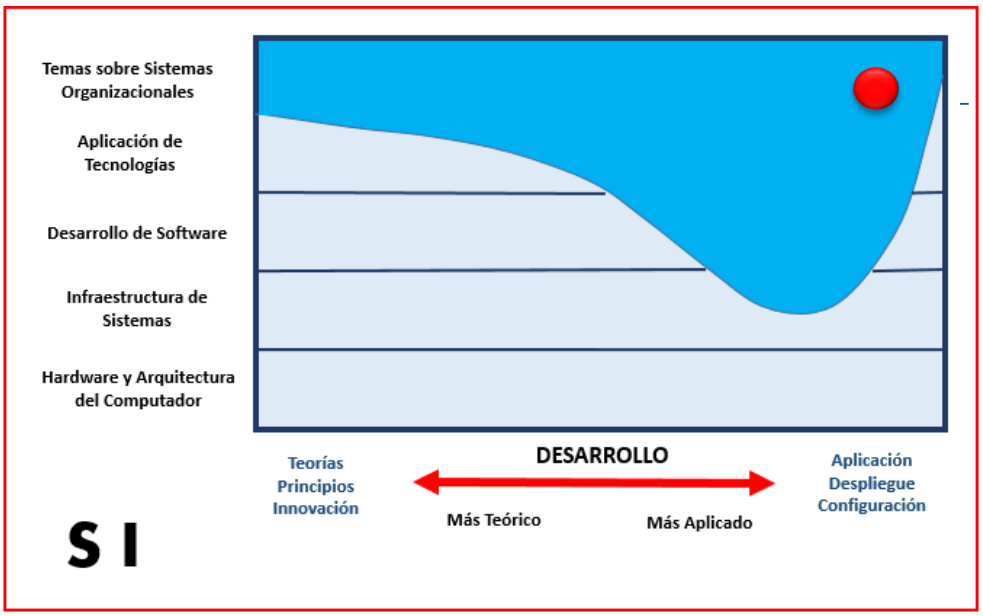


Figura 1: Focalización de la Investigación

Fuente: (Association For Computing Machinery, 2008)

Según Benetó la crisis del software hoy en día es un problema no resuelto, desde sus inicios en los años 40 (Benetó Micó, 2013). Tradicionalmente se considera que los problemas de software se relacionan exclusivamente con su funcionamiento, pero estos, no se limitan únicamente a la funcionalidad del producto software sino también se vincula a la propia praxis de su construcción, su mantenimiento y la creciente satisfacción e insatisfacción que se derivan de su uso. La demanda creciente por un software de calidad, este último se ha acentuado muchos más en esta última década originada, especialmente, por la necesidad de productos software que posibiliten la movilidad de las personas (Sommerville, 2011).

La literatura sobre la crisis de software describe lo que se ve a diario en las empresas de desarrollo de software esto es la inestabilidad del proceso de construcción de software, las debilidades en el manejo de riesgos, la complejidad del software y la formación del recurso humano competente para tales tareas (Pressman, 2010). Una de las causas de este problema está vinculada con el tema humano, esto es, los trabajadores del software o ingenieros de software quienes han tenido muy poco entrenamiento formal en las nuevas técnicas de desarrollo de software. Por ejemplo, en las empresas del estado, el recurso humano que trabaja con el desarrollo de software utiliza con frecuencia técnicas, herramientas y metodologías ad-hoc u obsoletas. En otras organizaciones, especialmente las privadas, se ve una suave forma de anarquía. Si nos enfocamos a nivel individual, es decir en el trabajador de software, encontramos que su enfoque es programar – codificar o escribir programas- con la experiencia obtenida en trabajo anterior o tan solo con la obtenida durante su formación técnica o universitaria. Es cierto que algunas personas desarrollan un método ordenado y eficiente de desarrollo del software mediante prueba y error, pero el común denominador es que las prácticas de desarrollo de software están llenas de malos hábitos que dan como resultado una pobre calidad y mantenibilidad del software (Iznaga Lamour, Delgado Fernández, & Febles Estrada, 2009).

De acuerdo al informe de PROMPERU, el Perú oferta software especializado para la banca y finanzas, comercialización, industria farmacéutica, naviera, turismo, industria minera y medio ambiente (PROMPERU, 2012). En el citado informe también se indica que la industria de software, tiene 18 años de experiencia y en los últimos siete se ha experimentado un crecimiento promedio anual de 15%, además, se cuenta con un poco más de 300 empresas formales y gran parte de estas empresas cuentan con certificaciones de calidad como el ISO 9001 y CMMI (Alianza Parlamentaria;, 2012).

La calidad y los procesos son temas que cada vez más las empresas del Perú, en general, toman en cuenta como un factor determinante para crecer. En particular, en el campo de la industria de software, se han aplicado modelos de procesos como es el caso del Modelo Integrado de Madurez y Capacidad (CMMI) u otros como el marco de referencia de procesos del ciclo de vida de software (ISO/IEC 12207, 2008). Sin embargo, éstos presentan dificultades para su implementación por ser grandes, poco accesibles y ser costosos para las pequeñas organizaciones (Sánchez Lorenzo, 2008). Para mitigar estos problemas el Perú junto con el Banco Interamericano de Desarrollo viene desarrollando

el Programa de Apoyo a la Competitividad de la Industria de Software (PACIS) en cual se ha definido al CMMI como modelo base para mejorar a las empresas involucradas con el desarrollo de software (APESOFT, 2011).

Si se realiza un análisis minucioso se puede determinar que el enfoque principal de CMMI está basado en un sistema administrativo de ayuda a los ingenieros de desarrollo, esto ha tenido un efecto positivo en el funcionamiento de las organizaciones del software. Pero existe otra medida significativa en pos de la mejora de calidad del software que se fundamenta en el proceso personal del software que se orienta a la mejora de las habilidades y conocimiento de la gente que realiza el trabajo de desarrollo de software (Iznaga Lamour, Delgado Fernández, & Febles Estrada, 2009).

Como bien conocemos, en el proceso de desarrollo de software se dan inconvenientes que hacen que su tiempo de implementación aumente considerablemente y muchas veces se alejan más allá de lo previsto. Este tipo de ocurrencias no solamente se presenta durante la formación de los futuros ingenieros sino también, y con mayor incidencia, en la industria de software y vinculada con el desarrollo del mismo.

Los inconvenientes no solo se relacionan con el tiempo de desarrollo o entrega del producto software, sino también con la calidad del mismo llegando a un punto en el cual el producto desarrollado o queda desfasado o no es utilizado.

Estos problemas, si bien es cierto, se intentan mitigar a través de los modelos de calidad mencionados en el apartado anterior. Uno de los problemas que aparecen es la exigencia de documentar todas las ocurrencias que se suceden durante el trabajo de desarrollo de software. Dada la cantidad de datos que se solicitan y el tiempo para procesar los mismos hacen que la propuesta sea difícil de poner en marcha por parte de las personas.
A fin de mitigar este problema, nos cuestionamos si es posible mejorar la adopción del Proceso Personal de Software mediante la inclusión de una herramienta que facilite el registro y procesamiento de los eventos que se ocurren durante el desarrollo de software y por tanto mejora la toma de decisiones a nivel personal y del líder del equipo de desarrollo.

Conocida la realidad planteamos el siguiente problema*:* ¿En qué medida la construcción de una solución Cloud Computing que automatiza las tareas del Proceso Personal de Software facilita su adopción?

Formulado el problema, se plantea la hipótesis nula como: "La construcción de una solución Cloud Computing que automatiza las tareas del Proceso Personal de Software no facilita su adopción y la alterna como: "La construcción de una solución Cloud Computing que automatiza las tareas del Proceso Personal de Software facilita su adopción".

Finalmente se formula el objetivo general: "Desarrollar una solución Cloud Computing que automatice las tareas del Proceso Personal de Software para facilitar su adopción" y los objetivos específicos: (1) analizar las prácticas de desarrollo de software en la industria de software, (2) analizar los procesos y tareas que requieren el Procesos Personal de Software, (3) construir una solución Cloud Computing bajo el enfoque de Google App Engine (GAE) usando el proceso ágil de Desarrollo ICONIX, que automatice las tareas y procesos del PSP - Proceso Personal de Software, (4) evaluar la herramienta como un facilitador para la adopción del Proceso Personal de Software en los procesos de desarrollo de los estudiantes del X ciclo de la carrera profesional de Ingeniería de Computación y Sistemas de la UPAO.

En relación a los aportes nuestra investigación pretende proporcionar una herramienta que automatiza las tareas y procesos que requieren el Proceso Personal de Software para facilitar su adopción por parte de los trabajadores de software. Así mismo, proporcionar una herramienta que ayude a los estudiantes a desarrollar conocimientos y habilidades convirtiéndolos en trabajadores de software de clase mundial. También, y a nivel de investigación, sentar las bases para trabajos futuros en el área de calidad de software con especial énfasis en las empresas de desarrollo de software y proporcionar un estudio sobre la adopción del PSP por parte de los estudiantes del X ciclo de la carrera profesional de Ingeniería de Computación y Sistemas.

Para una mejor lectura este documento se ha dividido en cinco capítulos. En el primero se detallan las características del problema surgido en la investigación, la hipótesis planteada y los objetivos que se pretende alcanzar. En el segundo se muestra el marco

teórico, el cual presenta la información necesaria para comprender como inicio la propuesta del presente proyecto de investigación. En el tercero se lista las actividades que se realizaron en la metodología propuesta en esta investigación. En el cuarto capítulo se muestra la construcción de la solución cloud, aquí se detalla desde la planificación hasta la etapa de post ejecución, las fases para el desarrollo están basadas en el Proceso de Software Personal. Finalmente presentamos los resultados, discusiones y sugerimos otras investigaciones que tomen como base este trabajo.

II. MARCO TEÓRICO

2.1. CLOUD COMPUTING

La National Institute of Standards and Technology (NIST) define a Cloud Computing como: "Un modelo que permite el acceso difundido, adaptado y bajo demanda en red a un conjunto compartido de recursos informáticos configurables y compartidos (redes, servidores, almacenamiento, aplicaciones, y servicios) que pueden ser rápidamente aprovisionados y liberados con un esfuerzo reducido de gestión o interacción mínima con el proveedor" (Takai, 2012)

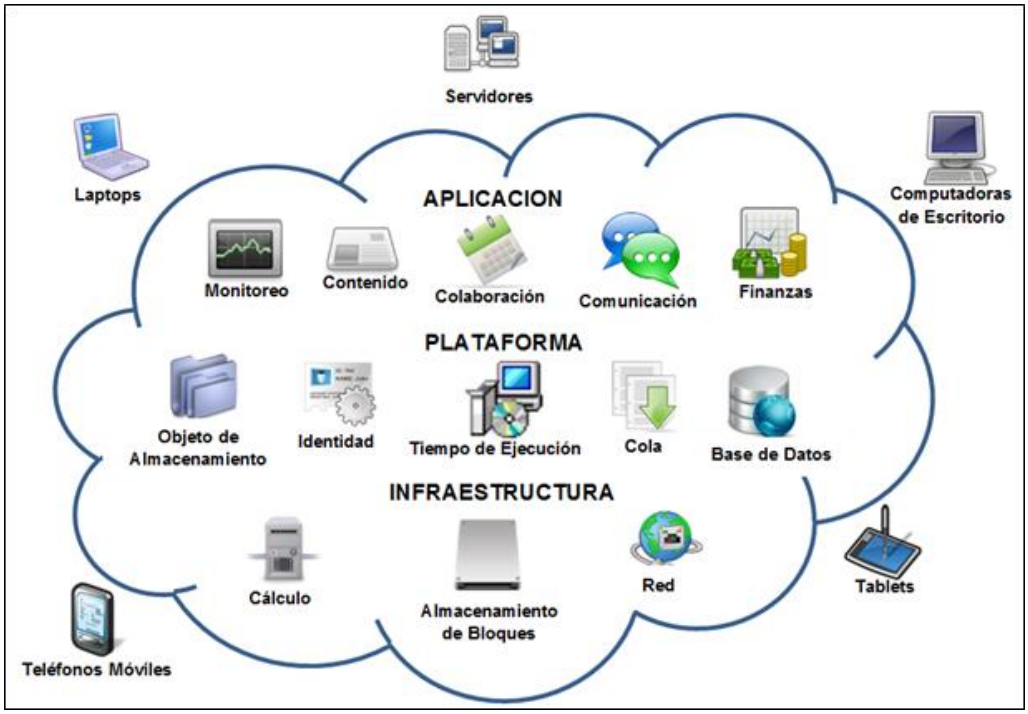


Figura 2: Esquema de Cloud Computing

Fuente: Instituto Nacional de Estándares y Tecnología - 2009

De acuerdo a lo que nos dice (Urubeña, Ferrari, Blanco, & Valdecasa, 2012), Cloud Computing presenta una serie de características señaladas a continuación:

- Su modelo de facturación se basa en el consumo del servicio.
- Aísla los recursos informáticos contratados al proveedor de servicios cloud de los equipos informáticos del cliente.

- Aumenta o disminuye las funcionalidades ofrecidas al cliente, en función de sus necesidades puntuales sin necesidad de nuevos contratos ni penalizaciones.
- Permite a varios usuarios compartir los medios y recursos informáticos, permitiendo la optimización de su uso.
- Permite al usuario acceder de manera flexible a las capacidades de computación en la nube de forma automática a medida que las vaya requiriendo, sin necesidad de una interacción humana con su proveedor o proveedores de servicios cloud.
- Posibilita a los usuarios de acceder a los servicios en cualquier lugar, en cualquier momento y con cualquier dispositivo que disponga de conexión a redes de servicio.

2.1.1. Modelos:

En atención al informe del "Programa Nacional de Infraestructura Criticas de Información y Ciberseguridad" (ICIC, 2008) la prestación de servicios de cloud computing pueden asociarse a tres modelos específicos: Software como Servicio (SaaS), Plataforma como Servicio(PaaS) e Infraestructura como Servicio (IaaS); como se muestra en la Figura 2, estos modelos de entrega pueden ser vistos en un contexto jerárquico. Para el usuario final sólo SaaS es visible, mientras que los desarrolladores utilizan PaaS y IaaS para desplegar sus aplicaciones. Posteriormente, las tres apariciones de Cloud Computing se introducen individualmente:

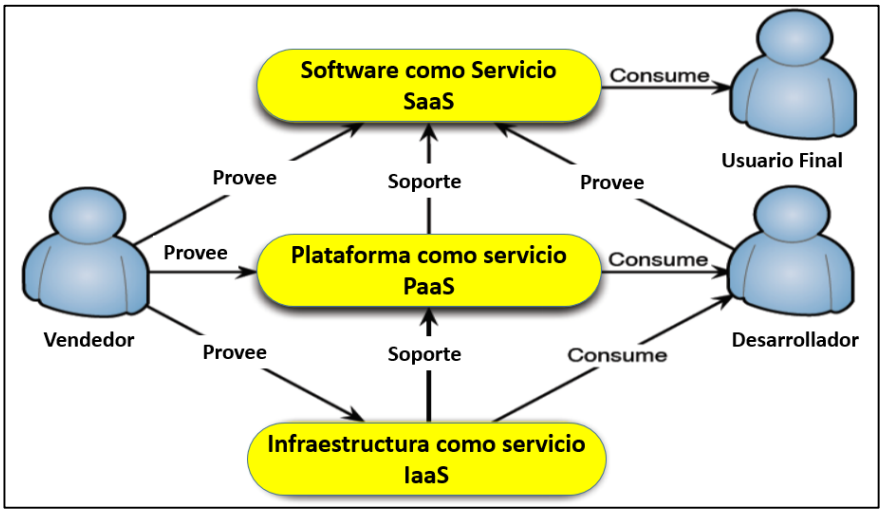


Figura 3: Esquema de Cloud Computing

Fuente: (Marinos & Briscoe, 2009)

2.1.1.1. Infraestructura como Servicio (IaaS)

Este modelo consiste en poner a disposición del cliente el uso de la infraestructura informática (capacidad de computación, espacio de disco y bases de datos entre otros) como un servicio tal como lo señala Ubereña y sus colegas, las facturas asociadas a este tipo de servicios se calculan en base a la cantidad de recursos consumidos por el cliente, basándose así en el modelo de pago por uso. Ejemplo: Amazon Web Services EC2.

2.1.1.2. Software como Servicio (SaaS)

Este servicio consiste en la entrega de aplicaciones como servicio, siendo un modelo de despliegue de software mediante el cual el proveedor ofrece licencias de su aplicación a los clientes para su uso, según el autor citado previamente, se debe pensar como un servicio bajo demanda tal como aquel que ha construido, y posibilita ser probado de manera gratuita, por la empresa. Ejemplo: SalesForce.

La solución de cloud computing de SaaS puede estar orientada a distintos tipos de clientes según su condición:

- Usuarios particulares:
  - Servicios de ofimática en cloud.
  - Redes sociales.
  - Red 2.0.
- Usuarios profesionales:
  - CRM.
  - ERP.

2.1.1.3. Plataforma como Servicio (PaaS)

Este servicio consistente en la entrega, como un servicio, de un conjunto de plataformas informáticas orientadas al desarrollo, testeo, despliegue, hosting y mantenimiento de los sistemas

operativos y aplicaciones propias del cliente (Urubeña, Ferrari, Blanco, & Valdecasa, 2012).

Características Principales:

- Facilita el despliegue de las aplicaciones del cliente, sin el coste y la complejidad derivados de la compra y gestión del hardware y de las capas de software asociadas.
- Ofrece a través de redes de servicio IP todos los requisitos necesarios para crear y entregar servicios y aplicaciones web.

2.1.2. Cloud Computing en Perú.

Según un reciente estudio a 150 empresas del país, realizado por la consultora Dominio Consultores en Marketing, sólo el 14.7% de empresas peruanas usa las aplicaciones del Cloud Computing; sin embargo se revela que el 48% de las firmas encuestadas proyecta implementar la Nube en un período de dos años (José Fuertes, 2012).

2.2. ARQUITECTURA DE GOOGLE APP ENGINE – GAE

Los autores (Chornyi, Riediger, & Wolfenstetter, 2010) definen a GAE como un servicio web de alojamiento de aplicaciones, lo cual permite el desarrollo y despliegue de aplicaciones basadas en la Web dentro de un entorno de ejecución pre-definido. En el citado informe también nos dice que a diferencia de otras ofertas de alojamiento en la nube, GAE proporciona una infraestructura de aplicaciones en el nivel PaaS, esto significa que GAE abstrae hardware subyacente y del sistema operativo capas subyacentes, proporcionando la aplicación alojada con un conjunto de servicios orientados a la aplicación.

GAE se puede dividir en tres partes: Entorno de tiempo de Ejecución, Almacén de Datos y Servicios (Chornyi, Riediger, & Wolfenstetter, 2010), los cuales se detallan a continuación:

2.2.1 Entorno de tiempo de ejecución - Runtime Environment

El entorno de tiempo de ejecución se refiere al lugar donde se ejecuta la aplicación real. (Srivastava, Trehan, & Yadav, 2012). De manera general este entorno funciona de la siguiente manera: La aplicación se invoca sólo una vez a través de una petición o solicitud HTTP la cual es procesada por el GAE a través de un navegador web o alguna otra interfaz, esto significa que la aplicación no se está ejecutando constantemente sino hasta que está la petición o invocación se haya realizado. En caso de una solicitud HTTP, el controlador de solicitudes reenvía la solicitud y el GAE selecciona una de los muchos posibles servidores de Google donde se encuentra la aplicación solo entonces ésta es instantáneamente desplegada y ejecutada por una cierta cantidad de tiempo. La aplicación puede realizar los caculos y devolver el resultado hacia al controlador de solicitudes GAE quien reenvía la respuesta HTTP al cliente. Es importante entender que la aplicación se ejecuta completamente embebida en el denominado sandbox pero sólo si las solicitudes siguen llegando o algún tipo de procesamiento se realiza dentro de la aplicación. La razón de este último se debe a que las solicitudes sólo se deben ejecutar cuando en realidad están realizando cálculos, de lo contrario se estaría asignando poder de cálculo y memoria sin necesidad. La figura muestra lo descrito.

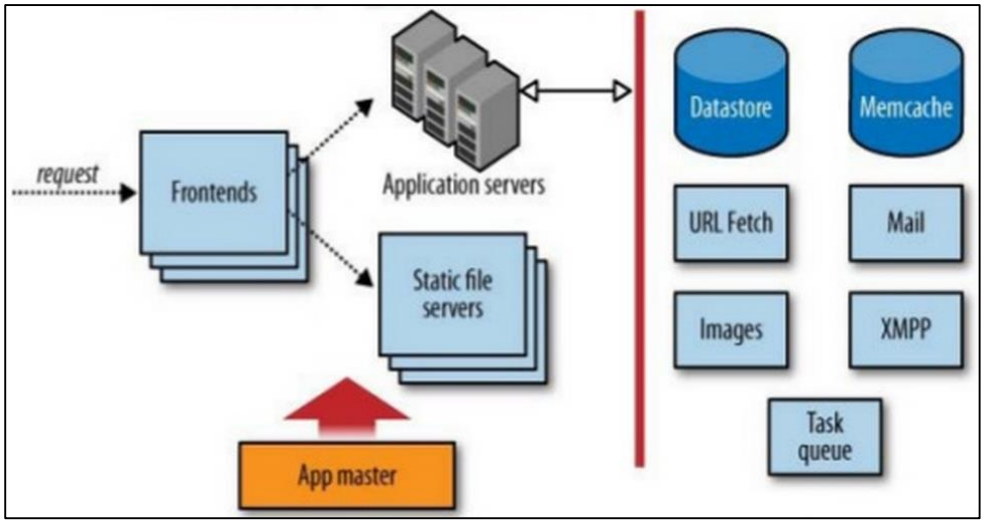


Figura 4: Arquitectura GAE

(Roche & Douglas, 2009)

El tipo de entorno de ejecución en los servidores de Google depende del lenguaje de programación utilizado. Para Java u otros lenguajes que tiene soporte para compiladores basados en Java (como JRuby, Rhino y Groovy) se proporciona un Java basado en Java Virtual Machine (JVM). GAE también soporta Google Web Toolkit (GWT) un marco de aplicaciones web enriquecidas. Para Python y marcos contextos se utiliza un entorno basado en Python.

2.2.2 Almacén de datos - DataStore

El almacén de datos de Google se puede escalar más allá de un millón de usuarios (Srivastava, Trehan, & Yadav, 2012). GAE utiliza un enfoque diferente para la persistencia de datos, llamado ***Bigtable*** (servicio de almacenamiento de datos distribuido, NoSQL, GQL). En lugar de usar filas (que se encuentran en una base de datos relacional), los datos Bigtable de Google se almacenan en las entidades (las entidades siempre están asociados con un cierto tipo). Estas entidades tienen propiedades que se asemeja a las columnas en los esquemas de bases de datos relacionales. Pero a diferencia de éstas, las entidades son en realidad esquemas, como dos entidades del mismo tipo no necesariamente tiene que tener las mismas propiedades o incluso el mismo tipo de valor para una determinada propiedad. La diferencia más importante para bases de datos relacionales es sin embargo, la consulta de entidades dentro de un almacén de datos Bigtable. En las bases de datos relacionales las consultas se procesan y ejecutan contra una base de datos en tiempo de ejecución de la aplicación. GAE utiliza un enfoque diferente. En lugar de procesar una consulta en tiempo de ejecución de aplicaciones, las consultas son pre-procesadas durante el tiempo de compilación cuando se crea un índice correspondiente. Este índice se utiliza posteriormente en tiempo de ejecución de la aplicación cuando se ejecuta la consulta real. Gracias al índice, cada consulta es sólo un recorrido en una tabla simple en donde se busca sólo el valor exacto del filtro. Este método hace que las consultas sean muy rápidas en comparación con las bases de datos relacionales, mientras que la actualización de las entidades es mucho más costosa. Las transacciones son similares a las de las bases de datos

relacionales. Cada transacción es atómica, lo que significa que tiene éxito, ya sea totalmente o fracasa.

2.2.3 Servicios:

App Engine ofrece una serie de servicios que permiten realizar varias operaciones comunes en el manejo de su solicitud (Srivastava, Trehan, & Yadav, 2012) disponibles a través de interfaces de programación de aplicaciones APIs con los que se puede accesar a los servicios de correo, envío de mensajes electrónicos (Mail), velocidad de recuperación en la misma solicitud e instancia de tiempo (MemCache), Manipulación de imágenes como cambiar el tamaño, recortar, rotar y voltear imágenes en cualquier formatos JPEG y PNG (images).

2.3. JAVA DATA OBJECTS

JDO especifica un servicio de persistencia de objeto que tiene un ciclo de vida sin estado (stateless) a través de una invocación simple a la máquina virtual java, tal como lo afirma (Ezzio, 2003). Este servicio conecta los objetos java (instancias de una clase que implementa las anotaciones JDO) desde la memoria con el almacenamiento interno de datos. Esta forma de persistencia permite que las aplicaciones se construyan de manera modular brindando el beneficio de mejorar sustancialmente la mantenibilidad de la aplicación. En Google App Engine JDO se ha definido como una interface que permite la conexión con los servicios de GAE en especial con el de manejo de datos a través el GAE SDK. Este último permite que la plataforma de datos DataNucleos disponible a través de App Engine Datastore tomando como vía de acceso JDO.

Los objetos JDO son definidos por el usuario incluyendo tanto la definición de los tipos de entidad, las clases de utilidad, y las capacidades de persistencia. Para posibilitar este último se debe incluir las anotaciones respectivas. De este modo JDO especifica el contrato de nivel de aplicación entre los componentes de la aplicación y el JDO PersistenceManager. La arquitectura de conectores J2EE especifican los

contratos estándar entre los servidores de aplicaciones y un conector EIS utilizado por una implementación JDO. Estos contratos son necesarios para una aplicación JDO para que pueda ser utilizado en un entorno de servidor de aplicaciones. La arquitectura de conectores define aspecto importante de la integración: administración de conexión, administración de transacciones, y seguridad. Los contratos de gestión de conexión son ejecutados por el adaptador de recursos EIS (que podría incluye un adaptador de recurso nativo de JDO).El contrato de gestión de la transacciones entre el administrador de transacciones (lógicamente distinto el servidor de aplicaciones) y el administrador de conexiones (Roche & Douglas, 2009) . Es compatible con transacciones distribuidas a través de múltiples servidores de aplicaciones y programas de gestión de datos heterogéneos. El contrato de garantía se requiere acceso seguro por la conexión de JDO a subyacente almacén de datos.

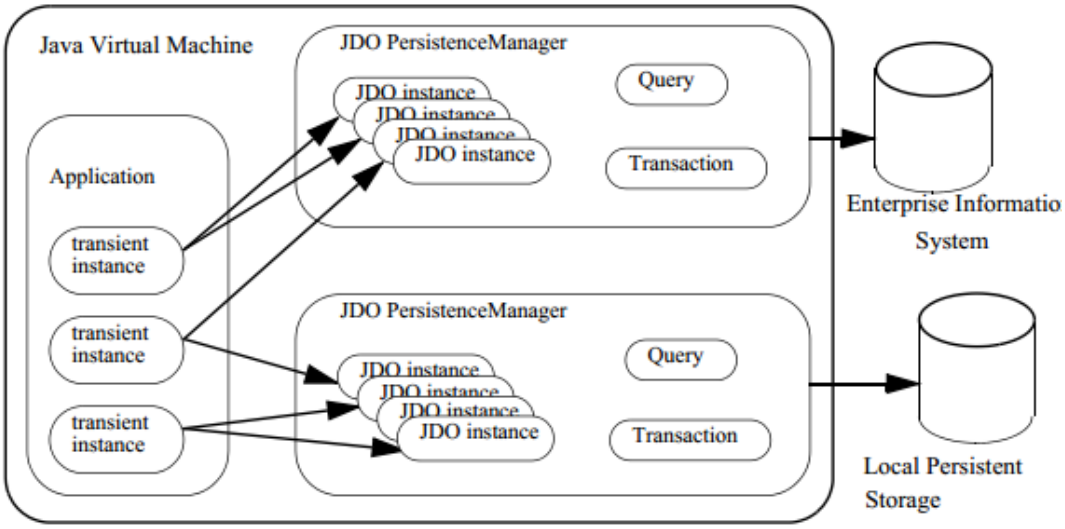


Figura 5: Arquitectura GAE

Fuente: (Ezzio, 2003)

2.4. INGENIERIA, CRISIS Y CALIDAD DE SOFTWARE

Según (Sommerville, 2011) la ingeniería de software es una disciplina de la ingeniería cuya meta es el desarrollo costeable de sistemas de software. Éste es abstracto e intangible. No está restringido por materiales, o gobernado por leyes físicas o por procesos de manufactura. Por su parte (Pressman, 2010) sostiene que

esta disciplina ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo. Es decir, es una disciplina que intenta racionalizar el proceso de desarrollo de software y establecer unas pautas a seguir para el desarrollo que minimicen tiempo, esfuerzo, y coste de desarrollo y maximicen la calidad del software tal como lo señala (Benetó Micó, 2013).

Existe, además, cierta confusión entre los conceptos entre ingeniería de software e ingeniería de sistemas, (Sommerville, 2011) nos da la diferencia entre estos dos conceptos: La ingeniería de sistemas se refiere a todos los aspectos del desarrollo y de la evolución de sistemas complejos donde el software desempeña un papel principal. Por lo tanto, la ingeniería de sistemas comprende el desarrollo de hardware, políticas y procesos de diseño y distribución de sistemas, así como la ingeniería del software.

Según Pressman (Pressman, 2010) sostiene que la crisis del software se refiere a un conjunto de problemas encontrados en el desarrollo del software de computadoras. Los orígenes de este problema se remontan a los primeros años de la década de los 70’s, cuando la ingeniería de software era prácticamente inexistente. Tal como apunta (Benetó Micó, 2013) esto se originó debido a que se presentaban dificultades en el desarrollo de software en base a que la demanda de software crecía rápidamente, la complejidad de resolver problemas y la falta de técnicas establecidas para el desarrollo de sistemas.

Por su parte (Campderrich Falgueras, 2008) sostiene que los grandes problemas de la Ingeniería del software son la calidad y la productividad. Para el primero la causa principal de las dificultades que se presentan se asocian con la gran complejidad del software comparado con otros tipos de productos, el cual origina, por ejemplo que no sea posible, ni mucho menos, probar el funcionamiento de un software en todas las combinaciones de condiciones que se puedan dar, y esto ocurre en una época en la que se da cada vez más la importancia a la calidad en todos los ámbitos, al considerarla un factor de competitividad dentro de unos mercados cada vez más saturados y por tanto más exigentes. Por lo que respecta a la productividad, cabe decir para empezar que cualquier fabricación en serie tiene necesariamente una

productividad mucho más elevada que la fabricación de un producto singular; pero incluso, si la comparamos con otras ingenierías de producto singular, la productividad es claramente inferior, un factor que tiene realmente relevancia en la baja de la productividad es el hecho de que, a diferencia de otras tecnologías, en un proyecto de software el desarrollo empieza tradicionalmente de cero (en algunos casos se utilizan fragmentos de software prefabricados).

Pese a estos inconvenientes listados anteriormente, para (Laguna, 2008) los ejecutivos y desarrolladores se siguen preguntado:

- ¿Por qué lleva tanto tiempo terminar los programas?
- ¿Por qué es tan elevado el costo?
- ¿Por qué no se puede detectar los errores antes de entregar el software a los clientes?
- ¿Por qué resulta tan difícil constatar el progreso del desarrollo del software?

Las respuestas a estas preguntas, en un inicio, permitió caracterizar la forma y el carácter de cómo se desarrolla el software. Estos cuestionamientos orientaron a su vez soluciones traducidas en metodologías y técnicas para la ingeniería de software tal como lo sugiere Benetó (Benetó Micó, 2013) en su obra citada.

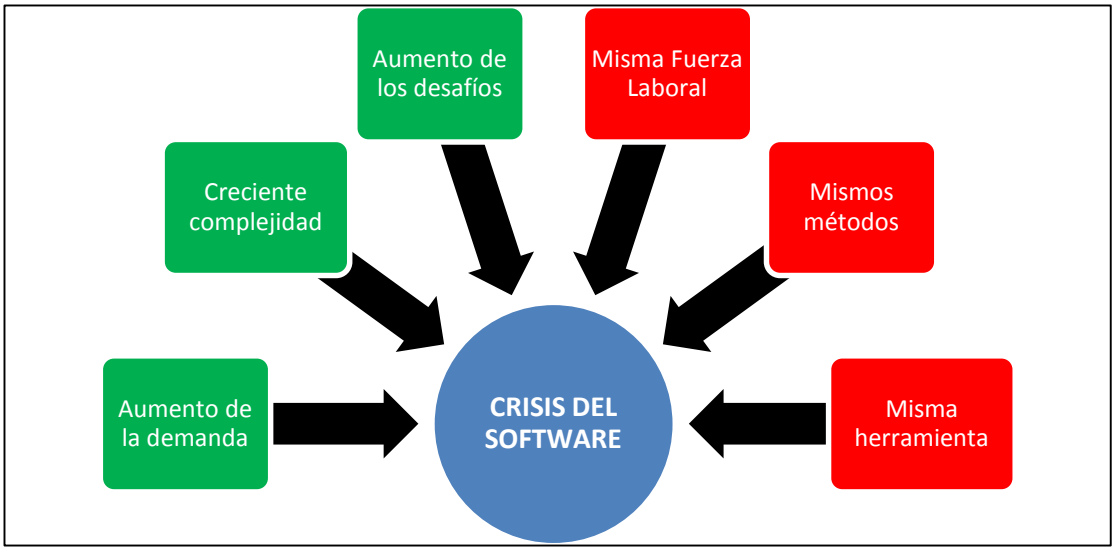


Figura 6: Causas de la Crisis en el Software

Fuente: (Benetó Micó, 2013)

Coincidimos con la definición de Calidad del Software, que plantea Pressman, descrita como la concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo documentados y con las características implícitas que se esperan de todo software desarrollado de manera profesional, esto implica que el concepto de calidad se encuentra en todo el ciclo de vida del software. (Pressman, 2010) Sugiere distintas actividades para la implantación del control de calidad en el desarrollo de software entre las que se encuentran:

- Aplicación de metodología y técnicas de desarrollo.
- Reutilización de procesos de revisión formales.
- Prueba del software.
- Ajustes a los estándares de desarrollo.
- Control de cambios, mediciones y recopilación de información.
- Gestión de informes sobre el control de calidad.

Para poder obtener calidad de software se necesita la implantación de un modelo o estándar, pero esto requiere de una Gestión de la Calidad del Software. La Calidad se logra a través de la Gestión de la Calidad, la cual, según (ISO 9001, 2008), consiste en la realización de actividades coordinadas que permiten dirigir y controlar una organización en todo aquello relativo al cumplimiento de los requisitos.

2.5. MODELOS DE CALIDAD DEL SOFTWARE

2.4.1. CMM (Capability Maturity Model)

Es un modelo que proporciona a las organizaciones de software una guía de cómo realizar un control de los procesos de desarrollo y mantenimiento de software (Forradellas, Pantaleo, & Rogers, 2011).

Lo que también nos indica dicho informe que es cada nivel de madurez establece un componente distinto en el proceso de software. Existen 5 niveles de madurez, como se muestra a continuación:

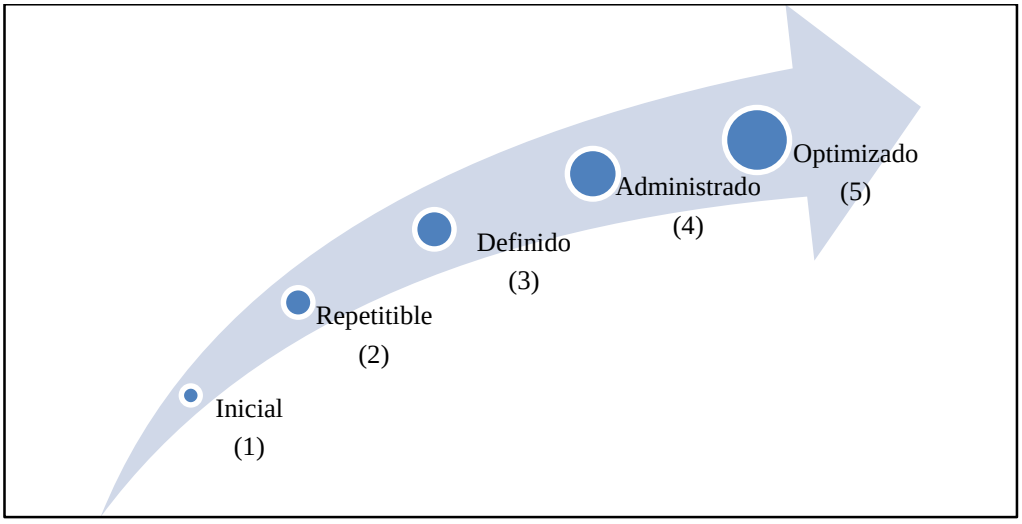


Figura 7: Niveles de Madurez del CMM

Fuente: (Forradellas, Pantaleo, & Rogers, 2011)

Por más de dos décadas el modelo CMM fue adoptado por la industria convirtiéndose en el modelo más utilizado, Buena parte de su expansión fue la adopción del mismo por parte de las "software factories" de la India, con la aplicación del modelo CMM y la experiencia acumulada se detectó la necesidad de contar con un modelo más abarcativo que incluyera el concepto más amplio de "capacitación y guía para el desarrollo de software" como sistema. Así surgió el modelo CMMI (Forradellas, Pantaleo, & Rogers, 2011).

2.4.2. CMMI (Capability Maturity Model Integration)

Es un método eficaz que las organizaciones pueden emplear para la gestión del rendimiento y así conllevar a la mejora de los procesos de desarrollo, adquisición y mantenimiento de productos y servicios (Institute CMMI, 2014).

El informe IT – Mentor (Forradellas, Pantaleo, & Rogers, 2011) menciona que el modelo CMMI entiende por organización inmadura a aquella que lleva sus proyectos por delante sin una pre definición de sus procesos a seguir, las consecuencias seria que estos proyectos frecuentemente sobrepasan sus

presupuestos y tiempos de terminación debido a que son iniciados con estimaciones poco realistas, sin una planificación adecuada, y son llevados adelante sin ningún tipo de gestión.

Para lograr cumplir con lo establecido en CMMI se debe concretizar la puesta en marcha de prácticas que están agrupadas en las 22 áreas de procesos (SEI, 2010), las cuales a su vez están agrupadas en cuatro categorías (Gestión de Proyectos, Ingeniería, Soporte, Gestión de Procesos). Existen dos formas para representar los niveles de madurez (SEI, 2010):

- ***Representación Continua***: Permite alcanzar "niveles de capacidad" y se ocupa de seleccionar tanto un área de proceso particular a mejorar como el nivel de capacidad deseado para esa área de proceso.
- ***Representación Por etapas***: Permite alcanzar "niveles de madurez" y se ocupa de seleccionar múltiples áreas de proceso a mejorar dentro de un nivel de madurez.

Para mayor entendimiento, a continuación en la siguiente tabla se muestran los niveles de madurez que CMMI identifica (García Coria, 2009):

NIVEL DE MADUREZ	ENFOCADO EN
Nivel de madurez 1: Inicial	Procesos no definidos
Nivel de madurez 2: Repetible	Gestión ineficiente del proyecto
Nivel de madurez 3 : Definido	Procesos normalizados por la Organización
Nivel de madurez 4: Administrado	Seguimiento y Control del proceso
Nivel de madurez 5: optimizado	Mejora continua del proceso

Tabla 1: Niveles de Madurez del CMMI

Así mismo el modelo CMMI incorpora al modelo por niveles de madurez una vista de niveles de capacidad por área de procesos (Forradellas, Pantaleo, & Rogers, 2011), los cuales se mencionan en la tabla 03.

NIVEL DE CAPACIDAD	CARACTERIZADO POR
Nivel 0: Incompleto	No existe implementación efectiva de los procesos.
Nivel 1: Ejecutada	Los procesos son implementados y alcanzan sus objetivos.
Nivel 2: Administrada	Los procesos se planifican y ejecutan de acuerdo a las políticas establecidas.
Nivel 3: Definida	Los procesos están definidos basados en estándares y buenas prácticas.
Nivel 4: Cuantitativamente Administrada	Los procesos están definidos y se controlan utilizando técnicas estadísticas y otras técnicas cuantitativas.
Nivel 5: Optimizada	Los procesos se mejoran continuamente.

Tabla 2: Niveles de Madurez basado en la capacidad del proceso del CMMI

2.4.3. TSP (Team Software Process)

Según el Instituto de Ingeniería de Software (SIE, 2010) lo define como proceso que esta designado específicamente para equipos de software, es una manera de guiar a los Ingenieros y a sus Gerentes en la utilización de métodos de trabajo en equipos efectivos.

Los autores (Humphrey, Chick, Nichols, & Pomeroy-Huff, 2010) describen que los objetivos de TSP son:

- Ayudar a los equipos de Ingeniería de Software a elaborar productos de calidad dentro de los costos y tiempos establecidos.
- Tener equipos rápidos y confiables.
- Optimizar el performance del equipo durante todo el proyecto.

Se debe tener en cuenta que para poder tener un uso eficiente de TSP, los desarrolladores de software deben ser entrenados primero en Personal Software Process. Los principios (Humphrey, Chick, Nichols, & Pomeroy-Huff, 2010) para la construcción de un equipo utilizando TSP son:

- Los miembros del equipo establecen objetivos en común y roles definidos.
- El equipo desarrolla una estrategia.
- Los miembros del equipo definen un proceso en común para su trabajo.
- Todos los miembros del equipo participan en la producción del planeamiento, y cada miembro conoce su rol en ese planeamiento.

El proceso de lanzamiento se grafica de la siguiente forma:

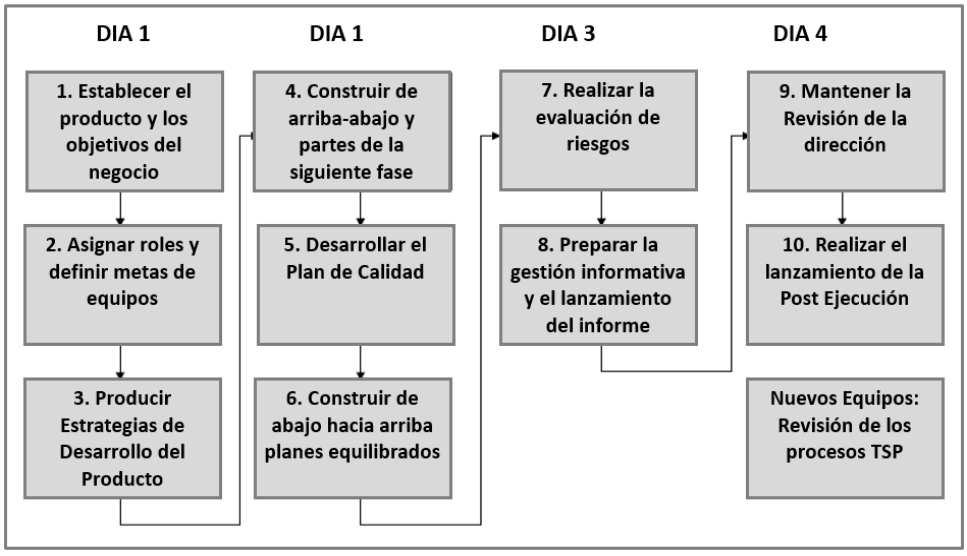


Figura 8: Proceso de Lanzamiento TSP

Fuente: (SIE, 2010)

Cada uno de los 9 lanzamientos mencionados anteriormente tiene un script que describe una actividad en detalle (Humphrey, 2000) (ver Tabla 09). Completando la figura anterior, todos los miembros del equipo participarán en producir el plan, estarán de acuerdo y serán confirmados en el plan.

PASO	ACTIVIDAD	DESCRIPCION
1	Establecer producto y objetivos del negocio	• Revisar el proceso de lanzamiento e incorporar los miembros del equipo. • Tratar los objetivos del proyecto con la dirección y responder preguntas.
2	Asignar roles y definir objetivos del equipo	• Seleccionar los roles del equipo • Definir y documentar los objetivos del equipo

3	Determinar una estrategia de desarrollo	• Producir un diseño conceptual del sistema • Determinar la estrategia de desarrollo y los productos a realizar • Definir el proceso de desarrollo a utilizar • Producir el proceso y soportar los planes
4	Desarrollar el plan general	• Desarrollar las estimaciones del tamaño y el plan general
5	Desarrollar el plan de calidad	• Desarrollar el plan de calidad
6	Construir un plan balanceado	• Asignación de trabajo a los miembros del equipo • Planear las próximas etapas para cada miembro del equipo • Armar un plan balanceado para el equipo y para cada miembro del equipo
7	Análisis del riesgo del proyecto	• Identificar y evaluar los riesgos del proyecto • Definir las responsabilidades y puntos de control de la evaluación del riesgo
8	Preparación del informe de lanzamiento	• Preparar un informe de lanzamiento para la dirección
9	Revisión de la dirección	• Revisar las actividades de lanzamiento y los planeamientos del proyecto con la dirección • Discutir los riesgos del proyecto, responsabilidades y acciones planeadas

Tabla 3: Pasos y Actividades del TCP

2.4.4. PSP (Personal Software Process)

Es un método que ayuda a la mejora del proceso de software, está formado por un conjunto estructurado de descripciones de procesos, de mediciones y de métodos; está orientado a la mejora individual de cada ingeniero de software, considerando aspectos como la planeación, calidad, estimación de costos y productividad. (Tuya, Ramos Román, & Dolado Cosín, 2007).

Fue definido por Watts Humphrey del Instituto de Ingeniería del Software (SEI) en la Universidad de Carnegie Mellon, en un inicio estuvo dirigido a estudiantes, luego con la publicación de sus primeros ejemplares ahora también está dirigido a ingenieros de software (Moreno, Tasistro, & Vallespir, 2011)

El proceso de PSP consiste de un conjunto de métodos, formularios y scripts que muestran a los Ingenieros de Software cómo planificar, medir y administrar su trabajo. Según (Humphrey W. S., 2001) , el diseño de PSP está basado en los siguientes principios de planificación y calidad:

- Los Ingenieros deben planificar su trabajo y deben basar sus planes en sus datos personales – Efectividad.
- Los Ingenieros deben usar los procesos bien definidos y medidos – Performance.
- Los Ingenieros deben ser responsables de la calidad de sus productos – Mejor Calidad.
- Menor costo en encontrar y arreglar los defectos de un proceso - Menor Costo.
- Es más eficiente prevenir defectos que encontrar y arreglarlos - Eficiencia.

Estructura del PSP

Lo que nos dice (Pomeroy-Huff, Cannon, Chick, Mullaney, & Nichols, 2009) es que la estructura del proceso PSP comienza con los requerimientos, y con la primera fase denominada ***"Planificación"***, esta fase según (Humphrey W. S., 2001) consta de un script de planificación que sirve de guía del trabajo y un resumen de la planificación para registrar los datos de la planificación. Luego se registra el tiempo y los datos de los defectos durante la fase de desarrollo, al final del trabajo, durante la última etapa (post mortem) se suman tanto los datos de los defectos como los tiempos, se mide el tamaño del programa y se ingresan estos datos en el resumen de plan; luego, se entrega el producto terminado con el resumen de la planificación, esto se ilustra en la figura siguiente:

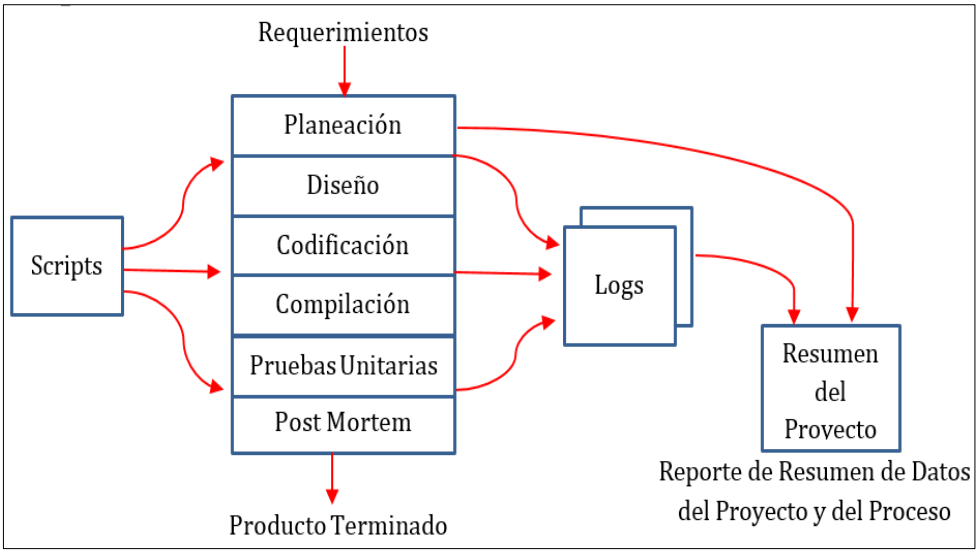


Figura 9: Estructura del Proceso PSP

Fuente: (Humphrey W. , 2000)

La importancia de los datos

Lo que se resalta de PSP es el uso de datos históricos para analizar y mejorar el rendimiento del proceso. La recopilación de datos PSP es apoyada por cuatro elementos principales:

- ✓ ***Guías (scripts)***: Son los elementos que documentan el proceso e indican que hacer y cuando hacerlo. Siendo apegados a la definición formal, su propósito es proveer una guía de alto nivel de cómo usar el proceso.
- ✓ **Formularios**: Son formularios para recopilar de manera sencilla y consistente la información. Entre los más básicos: Log de Tiempo (Donde se almacena cuando se invierte en cada fase o tarea del proyecto), Log de Defectos (En el cual se recopila la información de los defectos encontrados).
- ✓ **Medidas**: Miden el proceso y el producto, muestran si las cosas están funcionando bien. Algunas de las medidas que PSP recoge se enfoca en 4 aspectos, Tamaño, Esfuerzo, Calidad y Programación (Agenda ó Cronograma).
- ✓ **Estándares**: entregan una precisa y consistente definición que guía el trabajo, junto con la recopilación y uso de datos. Permiten aplicar

mediciones uniformes a través de múltiples proyectos y comparaciones entre unos y otros.

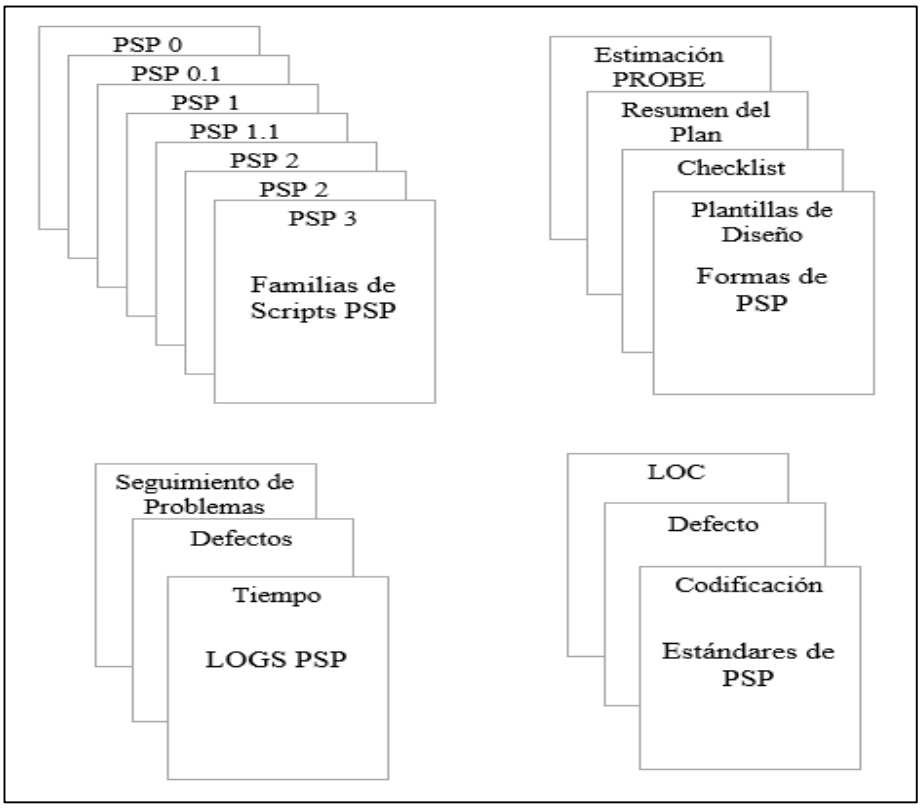


Figura 10: Principales elementos del PSP

Fuente (Humphrey W, 2000)

El ***Script de Planeación*** tiene como entrada los requerimientos del programa así como la estimación de los recursos que se emplea en este, el ***Script de Desarrollo*** exige los requerimientos de cada fase de desarrollo que toma en cuenta PSP para que al final se cuente con un programa bien probado y a prueba de errores, el ***Script de Postmortem*** pide todos los defectos que se encontraron durante la realización del proyecto y también el tiempo final que se utilizó durante la realización del programa, esto es para tener un historial y evitar caer en los mismos errores a futuro.

Cabe recalcar que se puede "personalizar" el proceso agregando o removiendo tareas conforme a las exigencias de cada persona o empresa. ***Esto quiere decir que por lo mismo de que PSP es un proceso y no un modelo***, se puede amoldar a las necesidades del programador.

Niveles del PSP:

El trabajo de (Pomeroy-Huff, Cannon, Chick, Mullaney, & Nichols, 2009) señala que el PSP tiene un número de métodos que generalmente no son practicados por los Ingenieros. Los métodos de PSP son introducidos en una serie de 7 versiones o niveles de proceso, tal como se muestra en la siguiente figura:

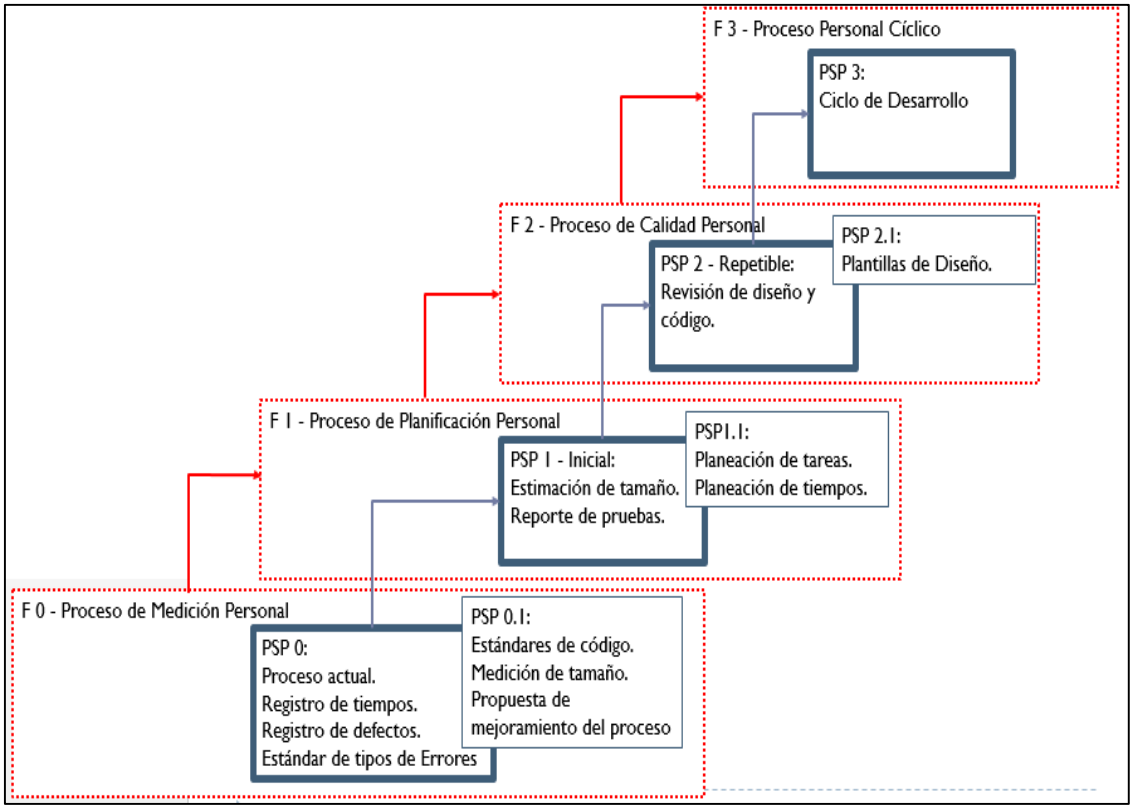


Figura 11: Niveles del Proceso PSP

Fuente: (Elaboración Propia)

Durante ***la fase 0*** el objetivo principal del ingeniero de software es aprender a seguir un proceso definido y a recopilar los datos básicos sobre tamaño, tiempo y defectos.

En ***la fase 1***, como ya se dispone de datos históricos para el proceso, el objetivo principal se centra en la estimación y en la planificación. Para ello es necesario conocer métodos de estimación de esfuerzo y de tamaño, y utilizar estos resultados en la planificación y en el seguimiento.

Ya en ***la fase2***, debido a que ya se realiza el control de la planificación, el objetivo principal está en la gestión de la calidad. Se debe conocer los métodos de detección temprana y de eliminación de defectos.

Una vez que se ha finalizado con las tres fases, el ingeniero debe realizar un informe sobre su rendimiento y debe ser capaz de analizar los datos que ha ido recopilando para efectuar los cambios que considere oportunos y que le conduzcan a la mejora de su trabajo.

La conceptualización y las características de la estructura del proceso PSP se describe detalladamente a continuación:

PSP 0

Es el punto de partida del PSP y provee un proceso definido para el desarrollo de pequeños programas. En este proceso, primero se desarrolla un plan para realizar el proyecto, luego se desarrolla el software necesario y después pasa a la fase de postmortem que se realizará después que se culminó el proyecto, en donde se compara el desempeño final con el planeado. (Hernández Hernández, 2013)

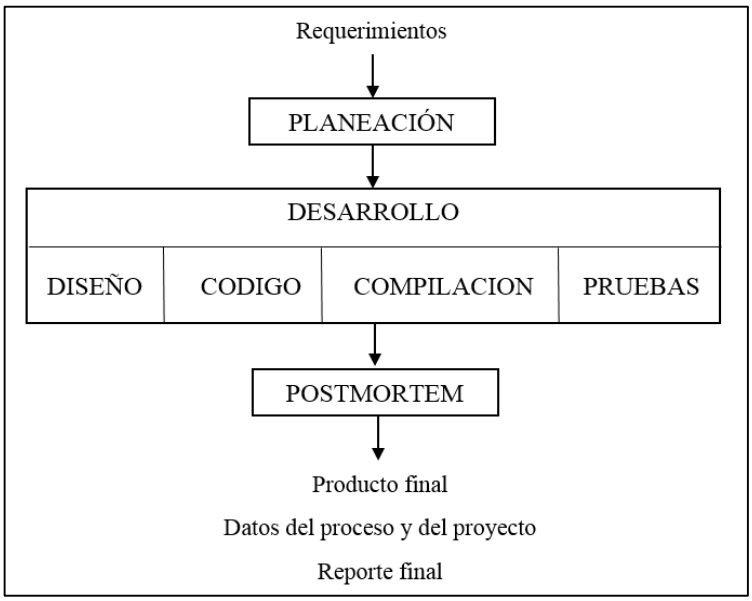


Figura 12: Estructura del Proceso PSP 0

Fuente: (Humphrey W. S., 1995)

Como se ha observado en la figura anterior PSP0 provee una estructura muy conveniente para hacer tareas a pequeña escala, un marco de trabajo para medir las tareas y un fundamento de mejor del proceso. Las tareas incluyen lo siguiente (Instituto Tecnológico de Morelia, 2008):

i. **Definición del proceso actual: Se** debe identificar el proceso actual en el proceso de desarrollo de software. En caso de que no se tenga un proceso definido, se puede utilizar el siguiente:

   Diseño → Codificación → Compilación → Prueba

   El proceso incluye las siguientes tareas:

   - Planeación: Produce un Plan de trabajo.
   - Desarrollo: Desarrollo del software actual.
   - Post ejecución: Comparación del funcionamiento actual con el plan de trabajo.

ii. **Registro de Tiempo: Es** un registro de las horas de trabajo en cada fase del proceso, para ver en qué fase se gastó más tiempo.

iii. **Registro de defectos:** Defectos son aquellos que pueden ser corregidos antes de que se complete el producto de software. Este puede ser un problema de requerimientos, de diseño o de implementación.

PSP 0.1

En este nivel se determinan las métricas del proceso, es decir la estimación de tamaño basado en la unidad de medida de tamaño de PSP que son las líneas de código (LOC). El PSP 0.1, incluye tareas como:

i. Categorías de Tamaño: Denominadas categorías LOC (Líneas de Código): base, agregadas, modificadas, eliminadas, nuevas y modificadas, reusadas, para reuso, total; más adelante se detalla cada una de estas categorías.

ii. **Estándares de código:** Se debe tener en cuenta la siguiente estructura para la programación: Cabecera, Uso/reuso, Identificadores, Comentarios, Sección más importante, Espacios en blanco, Identidad y Capitalización

iii. **Propuesta de Mejora del proceso (PIP):** Consiste en un formato donde se registrar los problemas que se encontraron durante el proceso de desarrollo, también se agregan las soluciones dadas de manera personal.

PSP 1

Ayuda a medir el tiempo empleado en cada fase y la detección de defectos. Para realizar la mayoría de los cálculos que exige cada nivel de PSP, es necesario partir de la base de los distintos tipos de LOC (Collazos, 2011). A las tareas incluidas dentro del PSP 0 y PSP 0.1, se les agrega lo siguiente:

i. **Estimación de tamaño:** Dicha estimación está basado en líneas de código, también es usado para mantener un registro de todos los datos estimados.

ii. **Reporte de Pruebas: La** cual consiste en mantener un registro constante de cada una de las pruebas que son ejecutadas y de los resultados obtenidos de cada una de estas pruebas.

iii. También agrega la **productividad** representada en LOC/hora, que se calcula dividiendo el tamaño total del programa entre el tiempo usado durante el tiempo de desarrollo, la cual se presenta con la fórmula siguiente (Pomeroy-Huff, Cannon, Chick, Mullaney, & Nichols, 2009): LOC/hora = tamaño A&M / Total tiempo de desarrollo * 60.

PSP 1.1

En este proceso se establece una base para realizar un seguimiento del trabajo de cada individuo, los ingenieros o programadores son enseñados a:

- Entender la relación entre el tamaño de los programas que escriben y el tiempo que les toma desarrollarlos.
- Aprender a realizar compromisos que puedan cumplir.
- Preparar un plan ordenado para realizar su trabajo.

Este nivel, agrega dos tareas más, descritas a continuación:

i. **Planeación de Tareas:** Está basado en la actividad a desarrollar, como escribir un programa o un reporte, implica estimar el tiempo de desarrollo y de los datos de la terminación para cada tarea del proyecto. El plan debe contener lo siguiente:
    - Nombre y número de la tarea.
    - Planeación de hora de acuerdo a la tarea por semana, y para el proyecto.
    - Tiempo actual por tarea por semana, y ara el proyecto.

ii. **Planeación de Horarios o Calendario:** Está basado en un periodo determinado de tiempo, como ejemplo se puede tomar cualquier segmento de un calendario (días, semanas, meses o años). El calendario de trabajo puede contener lo siguiente:
    - Numero de cada semana, típicamente empezando en 1.
    - Fechad de calendario para cada semana.
    - Planeación prevista para el trabajo en el proyecto por semana.
    - Horas previstas acumuladas.
    - Horas reales que se invierten en el proyecto por semana.
    - Horas reales acumuladas.

En este proceso se conoce el método del valor ganado (EV) que permite revisar el avance del proyecto, este método establece un valor para cada tarea, permite dar seguimiento al plan y facilitar dichos seguimientos incluso si existen cambios en el plan, cabe aclarar que esta planeación se usa siempre y cuando un proyecto o programa dura más de una semana (Instituto Tecnológico de Morelia, 2008).

PSP 2

En este nivel se revisa conceptos de calidad y se exponen los diferentes indicadores que PSP maneja para dicho tema, como son el ***Yield*** (rendimiento) de cada etapa de desarrollo, el ***análisis de defectos inyectados y removidos y el costo de calidad*** (COQ). Se incluyen tareas de PSP 1 y de PSP 1.1 más las siguientes (Humphrey W. S., 2001):

i. **Revisión de código:** Las revisiones técnicas o inspecciones de programa son similares excepto porque su objetivo principal es identificar las fallas o defectos tales como anomalías en el código, errores lógicos, incumplimiento de estándares, se debe cumplir con lo siguiente:
   - La revisión de código consiste en la comprobación de la inicialización de la variable y sus parámetros.
   - Llamada de función y formato de la llamada
   - Comprobación y deletreo de nombres
   - Comprobación de secuencias
   - Comprobación de archivos
   - Comprobación de punteros
   - Comprobación del formato de salida
   - Verificación de operadores lógicos
   - Asegurar que las llaves estén alineadas.
   - Verificar cada línea de código por instrucciones, sintaxis y puntuación apropiada.
   - Asegurar el código conforme al estándar de codificación.

ii. **Revisión de diseño:** Asegura que los requisitos, especificaciones y altos niveles de diseño sean cubiertos totalmente, se debe considerar lo siguiente (Instituto Tecnológico de Morelia, 2008):
   - Verificación de la lógica del programa
   - Verificación de casos especiales
   - Verificación de uso de funciones
   - Verificación de nombres

- Revisión del diseño de acuerdo al estándar aplicable al diseño.

Los formatos que se utilizan en este tipo de programas ayudan a listar los procedimientos. Lo nuevo que se introduce en los formatos de este nivel de PSP es que se utiliza el concepto de checklists que en realidad son listas que sirven para asegurar que cada elemento de programación esté en su lugar.

PSP 2.1

Complementa a PSP 2 y se encuentra bajo el mismo concepto de administración de la calidad, este proceso ayuda a orientar el criterio para la finalización del diseño, es decir, cuando han terminado que es lo que deben haber obtenido, también establece un criterio de completitud de diseño y examina varias técnicas de verificación y consistencia de diseño. Los formatos utilizados en este nivel son los mismos que se utilizan en PSP 2, sin embargo, existe una nueva plantilla que se encuentra en el checklist de revisión de diseño (Instituto Tecnológico de Morelia, 2008):

Diseño de Plantillas de PSP 2.1:

Hay cuatro plantillas de diseño que proveen de forma ordenada un marco de trabajo y el formato de registro para los diseños. Las plantillas incluyen lo siguiente:

i. **Escenario de operación de la plantilla:** Esta plantilla tiene las descripciones de los panoramas probables que se seguirán al usar el programa.

ii. **Especificación funcional de la plantilla:** Esta plantilla se puede usar para describir funciones y procedimientos de diseño de funciones o de objetos de diseño orientado a objetos.

iii. **Especificación de estado de la plantilla:** Esta plantilla se usa para registrar el comportamiento del programa, subprograma o clase en un sistema orientado a objetos.

iv. **Especificación lógica de la plantilla:** Esta plantilla mantiene la lógica del pseudocódigo para cada función o unidad del programa.

PSP 3

Es el proceso personal cíclico, en el cual presenta métodos para ser usados en la realización de programas medianamente grandes, por lo que combina múltiples proceso de PSP 2.1 para soportar el desarrollo de software a gran escala; este nivel ayuda al ingeniero a desarrollar programas más largos en poco tiempo y con menos errores. El orden con el que ahora debe de proceder debe de ser el siguiente (García Alonso, 2008): Planeación, Diseño de alto nivel, Ciclos de desarrollo. Las tareas incluyen todo lo de PSP 2 y PSP 2.1 y lo siguiente:

i. **Desarrollo Cíclico:** Cuando se usa PSP3, se debe de tener un plan para implementar grandes programas en módulos incrementales más o menos de 100 líneas de código (u otro tamaño apropiado) (García Alonso, 2008).

ESTRATEGIA GLOBAL DEL PSP

A. Recolección de Data

La recopilación de datos nos permite monitorear el trabajo y mejorar futuros planes (con un modelo estadístico que permita explicar el tiempo en función de otras medidas). A partir de los datos recopilados se obtienen las siguientes medidas:

- **Medidas de tiempo**: se registran tiempos de inicio y finalización de cada tarea. También se registran las interrupciones (ej.: atender el teléfono, recreo breve, alguien nos interrumpe para hacer alguna pregunta, etc.). Esto permite eliminar "ruido" y hacer mejores estimaciones.

❖ **Medidas de tamaño:** LOC (líneas de código) por categorías (Humphrey W. , The Personal Software Process, 2000):

- ✓ **LOC Base:** Es el tamaño de la versión original del programa antes de que se haga cualquier modificación.
- ✓ **LOC Agregado**: Es el nuevo código agregado a un programa existente.
- ✓ **LOC Modificado**: Es el código de un programa base que ha sido cambiado.
- ✓ **LOC Eliminado:** Es el código de un programa base que se suprime y no se usa para la próxima versión.
- ✓ **LOC Reusado:** Es el código que se toma de una librería de reutilización, sin hacer alguna modificación al nuevo programa o a la versión previa del programa.
- ✓ **LOC Agregado y Modificado**: Cuando el esfuerzo de añadir y de modificar son similares, conviene combinarlos, para ello se usa esta categoría, es decir A & M es igual a la suma de agregados más modificados.
- ✓ **LOC Nuevo Reusado:** Es el código nuevo que se ha desarrollado para incluirlo en una librería de elementos a reutilizar.
- ✓ **LOC Total:** El LOC total es el tamaño total de un programa, sin importar de dónde salió el código empleado.

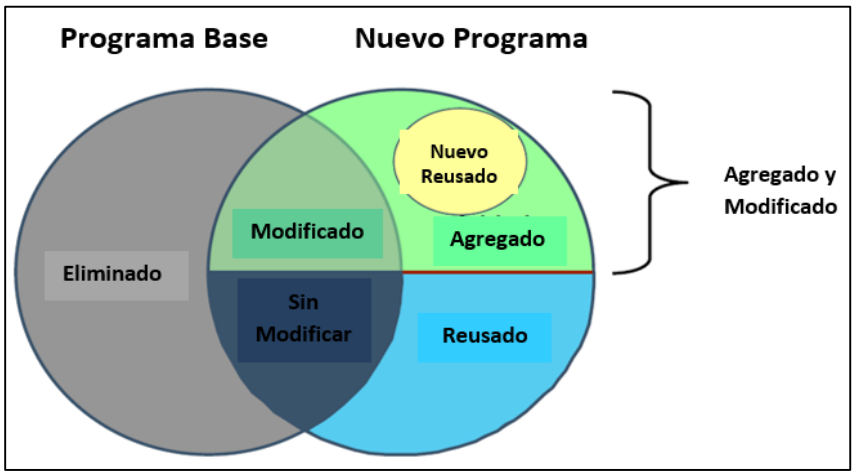


Figura 13: Categorías LOC

Fuente: (Hernández Hernández, 2013)

Por ejemplo, si un producto de 100 LOC fuera utilizado para desarrollar una nueva versión, y había 12 LOC de código suprimido, 23 LOC de código agregado, 5 LOC de código modificado y por último 3 LOC de código reutilizado, el LOC nuevo y cambiante sería:

LOC = Agregado + Modificado → 28 = 23 + 5.

Cuando se mide el tamaño total de un producto, los cálculos son como siguen:

Total LOC = Base - Suprimido + Agregado + Reutilizado → LOC Total = 100 - 12 + 23 + 3 = 114 LOC

- **Medidas de calidad:** basadas en la relación de cantidad de defectos vs. tamaño o bien defectos vs. Tiempo, se listan a continuación las más principales: (Humphrey W. , The Personal Software Process, 2000).
  - ✓ *Densidad de defectos*: cantidad de defectos / LOC
  - ✓ *Ratios de Tiempo de Desarrollo.*
  - ✓ *Ratios de Defectos* (cociente de defectos encontrados).
  - ✓ *Yield (Rendimiento)*: porcentaje de defectos encontrados y arreglados en la misma fase de revisión (es decir que no se propagan a fases posteriores).
  - ✓ *Defectos por Hora*: se pueden estimar a partir de la información del formulario de resumen del plan.

B. Estimación y Planificación

La estimación de tamaño y de los recursos del producto debe ser moderada, por lo que PSP se basa en el tamaño y en los datos de la productividad de cada ingeniero para estimar el tiempo requerido para hacer el trabajo. Seguidamente en la figura N° 13 se muestra el flujo de trabajo en la estimación y planificación:

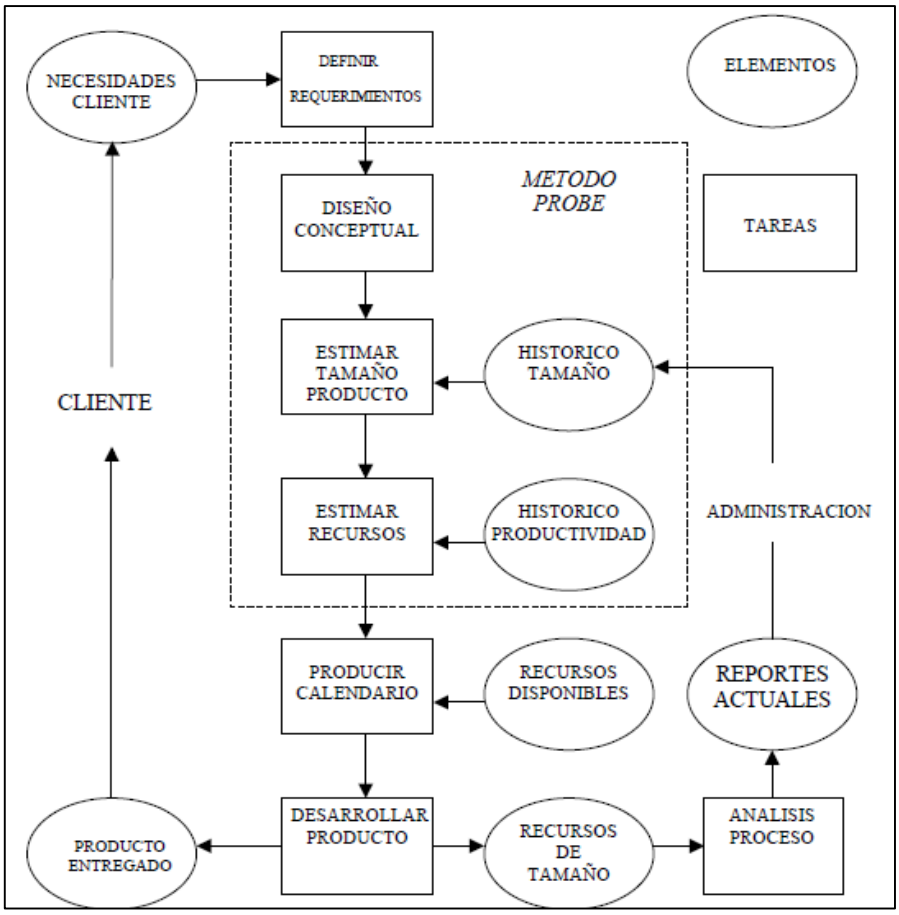


Figura 14: Marco de Planeación de Proyectos

Fuente: (Humphrey W. S., 1995)

Para mayor entendimiento a continuación se da un pequeño alcance que lo que significa cada tarea:

- ***Requerimientos***: el proceso comienza con el plan definiéndose el trabajo que se necesita realizar con el mayor detalle posible.
- ***Diseño conceptual***: se realiza un diseño a grandes rasgos para poder hacer una estimación inicial.
- ***Estimación del tamaño del producto y recursos***: el tiempo y el tamaño están muy relacionados cuando se trabaja individualmente. Con lo cual se estima el tamaño del proyecto y, en base a la productividad personal, se estima el tiempo.
- La estimación utiliza el método PROBE ***(Estimación Basada en Proxies)***.

C. Métodos de conteo

- **Conteo físico:** Consiste en contar las líneas de código físicas o una variante cercana de ellas, puede hacerse automáticamente con programas bastante sencillos; pero es fuertemente dependiente del formato.

- **Conteo lógico:** Consiste en contar instrucciones efectivas y estructuras de control, puede ser un método complejo de definir con toda precisión, pero es más difícil de implantar en forma automática.

D. Estimación basada en proxies

Un proxy es una característica del programa que es fácilmente visualizarle en etapas tempranas del desarrollo, pueden ser los pantallazos, objetos, archivos. Sus principales características son (Correal, 2009):

- La cuenta o medida del proxy debe tener una alta correlación con el esfuerzo necesario para construir el programa.
- El proxy debe poder contarse o medirse en forma automática sobre el producto terminado.
- Debe ser fácil de visualizar al comienzo del proyecto.
- Debe ser adaptable a necesidades específicas.
- Debe adaptarse a variaciones de implantación

También se puede definir a los Objetos como proxies (Correal, 2009), ya que dichos objetos se asimilan a los distintos entes que son relevantes en el ambiente propio de una aplicación, entendemos que es relativamente fácil identificar los objetos que tomarán parte en un programa con base en una mínima especificación. Los objetos son fácilmente contables sintácticamente dentro de un programa, por lo que es posible programarlos usando un lenguaje y una metodología orientados a objetos que sean uniformes.

El método PROBE

Este método usa proxies para estimar el tamaño del programa y el tiempo de desarrollo.

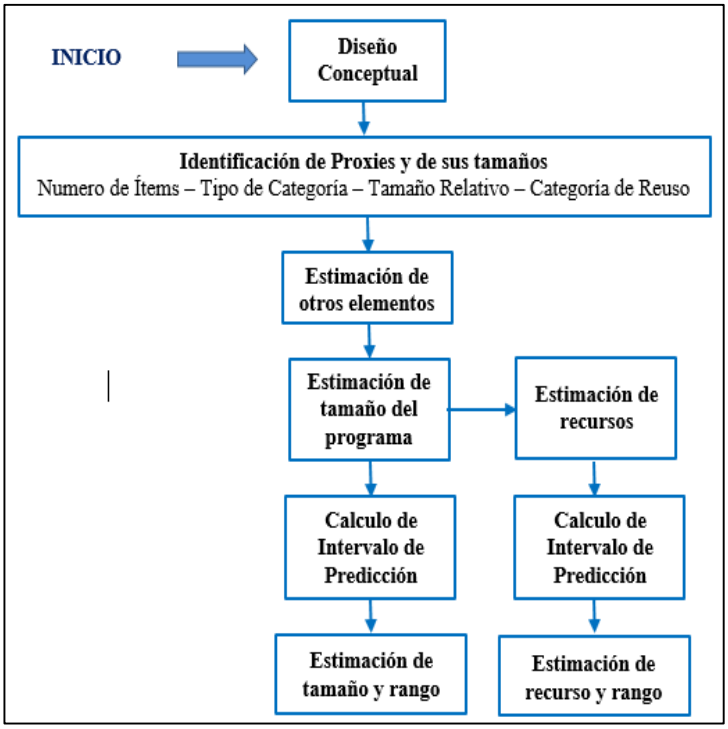


Figura 15: Pasos del Método PROBE

Fuente: (Software Engineering Institute, 2008)

El método PROBE consiste en:

- Primero determinar los objetos requeridos descritos en el diseño conceptual, luego determinar el tipo probable de métodos que se emplean en el programa y el número de métodos que cada objeto necesita. También se puede hacer uso de referencias de datos históricos sobre los tamaños de objetos similares que se han desarrollado previamente y que al mismo tiempo utilizan el cálculo de la regresión lineal para determinar el tamaño total del producto acabado.
- Cada ingeniero o programador debe tener una tabla de los proxies que define para su desarrollo y el tamaño de cada uno de ellos medidos en LOC's, estas tablas se calculan de manera aproximada o

haciendo uso de datos históricos sobre tamaños de objetos similares, PSP nos da un ejemplo del Tamaño estimado de objetos en C++ que se muestra en la tabla N° 00, la cual fue construida por datos históricos de programadores expertos en C++.

Tipo	Muy Pequeño	Pequeño	Mediano	Grande	Muy Grande
Cálculo	2.34	5.13	|11.25	24.66	54.04
Datos	2.60	4.79	8.84	16.31	30.09
E/S	9.01	12.06	16.15	21.62	28.93
Lógica	7.55	10.98	15.98	23.25	33.83
Texto	3.75	8.00	17.07	36.41	77.66

Tabla 4: Rangos de Tamaño de Objetos en C++

Fuente (Humphrey, 2000)

- Una vez que se estiman los tamaños de los objetos, se utiliza la regresión lineal para estimar la cantidad total de código o LOC (líneas de código fuente), que planean desarrollar. Para utilizar la regresión lineal, los ingenieros deben realizar una comparación de los datos históricos contra el resultado estimado del tamaño del programa actual, esta comparación se debe hacer por lo menos con tres programas anteriores.
- El método PROBE también utiliza la regresión lineal para estimar los recursos que se emplea en el desarrollo completo, esta estimación se basa en el tamaño estimado del programa contra los datos reales del esfuerzo con por lo menos tres proyectos anteriores. Los datos deben demostrar una correlación razonable entre el tamaño del programa y el tiempo de desarrollo, PSP requiere que el resultado de esta correlación sea de por lo menos 0.5.
- Estimado ya el tiempo total que se empleará para el trabajo, los ingenieros deben apoyarse en sus datos históricos para estimar el tiempo necesario que se tomará en cada fase. Por medio de los porcentajes que se obtienen en el campo del formato de registro de tiempo, los ingenieros tienen que asignar su tiempo de desarrollo

total estimado a las fases de planeamiento, diseño, revisión de diseño, código, revisión de código, compilación, pruebas y finalmente post-mortem. Cuando estos porcentajes han sido calculados, los ingenieros ahora cuentan con una estimación más real para el tamaño del programa, el tiempo de desarrollo total y el tiempo requerido para cada fase del desarrollo.

E. Gestión de Defectos

Durante el proceso de desarrollo se deberá registrar cada defecto encontrado, registrando la información suficiente sobre cada defecto para ser entendido posteriormente. Luego del registro, se analizan los datos para ver qué tipos de defectos causan los mayores problemas (Humphrey W. S., 2001).

PSP brinda un estándar de tipo de defectos que pueden ser de mucha ayuda, el cual se muestra a continuación.

No	Nombre o descripción
10	Comentario de documentación, mensajes
20	Deletreo de sintaxis, puntuación, formato
30	Construcción, Manejo de cambio de paquete, librería, control de versión
40	Declaración de asignación, duplicado de nombres, alcance, limitaciones
50	Procedimiento de la Interfaz, llamadas, referencias, I/O, formato de usuario
60	Chequeo de mensajes de error
70	Estructura de datos, contenido
80	Función de conexión, enlace, recursividad,
90	Configuración de sistema, sincronización, memoria
100	Diseño del entorno, compilación, prueba, otro sistema de soporte de problemas

Tabla 5: Tipo de Defectos

Fuente: (Humphrey W. S., 2001)

2.6. ESTÁNDARES DE CALIDAD

2.5.1. ISO 9001: 2008

Es la base del sistema de gestión de la calidad ya que es una norma internacional y que se centra en todos los elementos de administración de calidad con los que una empresa debe contar para tener un sistema efectivo que le permita administrar y mejorar la calidad de sus productos o servicios (ISO, 2011).

Según esta norma (ISO 9001, 2008), los principios que están involucrados con la Gestión de la Calidad son:

1. ***Enfoque en el cliente***: las organizaciones deben comprender las necesidades del cliente y satisfacer sus requisitos excediendo sus expectativas.
2. ***Liderazgo***: Los lideres deben crear y mantener un ambiente laboral donde las personas estén comprometidas con el logro de sus objetivos.
3. ***Participación del Personal***: Todos los colaboradores de la organización deben tener compromiso total para que sus habilidades colaboren en el éxito de la organización.
4. ***Enfoque en Procesos***: Se debe considerar las actividades y recursos como parte de un proceso.
5. ***Gerencia Sistemática***: Ver los procesos como un sistema para alcanzar los objetivos organizacionales de manera efectiva y eficiente.
6. ***Mejoramiento Continuo***: Dicho mejoramiento debe ser un objetivo permanente dentro de la organización.
7. ***Relación de beneficio mutuo con proveedores***: Esta relación permitirá aumentar el potencial de ambos para crear valor.
8. ***Decisiones basadas en hechos objetivos (cuantificables):*** Las decisiones deben estar basados en el análisis de datos e información.

Esta norma puede aplicarse a todos los procesos la metodología conocida como "Planificar-Hacer-Verificar-Actuar" (PHVA), el cual puede describirse corno (ISO 9001, 2008):

- **Planificar**: establecer los objetivos y procesos necesarios para conseguir resultados de acuerdo con los requisitos del cliente y las políticas de la organización.
- **Hacer**: implementar los procesos.
- **Verificar**: realizar el seguimiento y la medición de los productos respecto a las políticas, los objetivos y los requisitos para el producto, e informar sobra los resultados.
- **Actuar**: tornar acciones para mejorar continuamente el desempeño de los procesos.

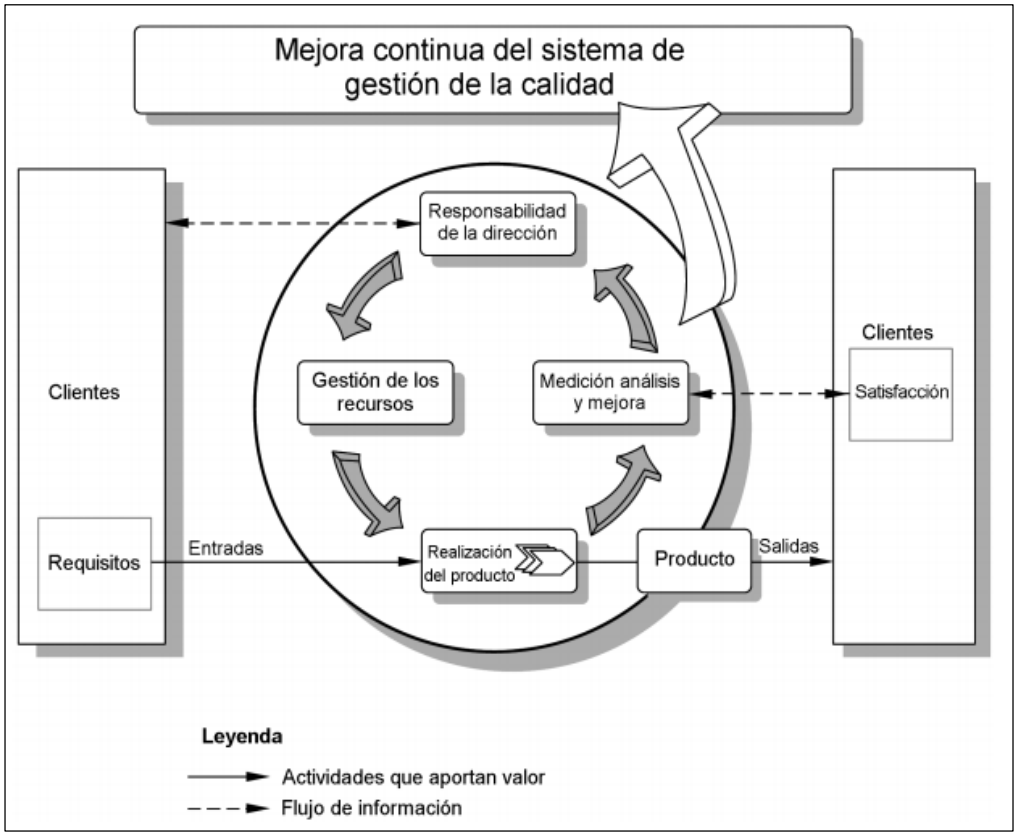


Figura 16: Modelo de un sistema de gestión de la calidad basado en procesos

Fuente: ISO 9001:2008

2.5.2. ISO/IEC 12207:2008

Norma internacional, la cual está orientada a los procesos del ciclo de vida del software de la organización ISO (ISO/IEC 12207, 2008), establece un proceso de ciclo de vida para el software que incluye procesos y actividades que se aplican desde la definición de requisitos, pasando por la adquisición y configuración de los servicios del sistemas hasta la finalización de su uso.

En la siguiente Tabla se puede observar que esta norma agrupa las actividades que pueden llevarse a cabo durante el ciclo de vida del software en cinco procesos principales, ocho procesos de apoyo y cuatro procesos organizativos (Ceballos Cardona & Velez Tascon, 2013):

Procesos principales del ciclo de vida	**Procesos de apoyo del ciclo de vida**	**Procesos organizativos del ciclo de vida**
Adquisición	Documentación	Administración
Suministro	Administración de la Configuración	Infraestructura
Desarrollo	Aseguramiento de la Calidad	Mejoras
Mantenimiento	Verificación	Entretenimiento
Operación	Validación	
	Revisiones Conjuntas	
	Auditorías	
	Resolución de Problemas	

Tabla 6: Procesos y subprocesos de ISO/IEC 12207:2008

III. MATERIAL Y METODOS

3.1. MATERIAL Y PROCEDIMIENTO

3.1.1 Procedimiento

Diseño De Técnicas De Recolección De Información:

Diseño de Investigación	Modelo	Técnica
Cuasi Experimental (Pre Test y Post Test)	Analítico	Examen Observación

Tabla 7: Diseño De Técnicas De Recolección De Información

3.1.2 Población y Muestra

POBLACION	MUESTRA
Estudiantes de la Carrera de Ingeniería de Computación y Sistemas	Estudiantes del X ciclo de la carrera de Ingeniería de Computación y Sistemas

Tabla 8: Población y Muestra

3.1.3 Diseño de prueba y tipo de muestreo

Diseño de la Prueba	Tipo de Muestreo
• Diseño de Pre Test y Post test. • Evaluación a través de una escala de valoración.	• Empírico Intencional.

Tabla 9: Prueba y Tipo de Muestreo

3.1.4 Variables e Indicadores

VARIABLE	INDICADOR
INDEPENDIENTE Construcción de una solución Cloud Computing que automatiza las tareas del Proceso Personal de Software	(X1) Registro de datos en la bitácora de planificación. (X2) Registrar el uso estándares de codificación y de definición de lenguajes de programación. (X3) Registro de la métrica de líneas de código (X4) Registrar tipologías de defectos.
DEPENDIENTE Adopción del Proceso Personal de Software	(Y1)Facilidad para adoptar la planificación en base a estadísticas registradas en una bitácora. (Y2)Facilidad para adoptar el registro del uso de estándares de codificación y de lenguajes de programación. (Y3)Facilidad para adoptar el uso de la métrica líneas de código. (Y4)Facilidad para adoptar el registro de tipologías de defectos.

Tabla 10: Variables Vs. Indicadores

3.2. METODOLOGÍA

Para el desarrollo de este proyecto de investigación se utilizará la metodología conformada de las siguientes fases:

a. Analizar las prácticas de desarrollo de software en la industria de software.
   - Técnica de recolección de datos: Análisis Bibliográfico, Evaluación documental, comparación.
   - Instrumento: Fichas Bibliográficas.

b. Analizar los procesos que con lleva el Personal Software Process:
   - Técnica de recolección de datos: Análisis Bibliográfico, Evaluación documental.
   - Instrumento: Fichas Bibliográficas, Videos Educativos.

c. Construir la herramienta bajo el esquema Cloud Computing de Google (GAE: Google App Engine) y el Proceso de Desarrollo Ágil ICONIX.

d. Determinar un caso estudio para evaluar el desempeño y adopción de la herramienta.

e. Aplicar la herramienta y analizar los resultados mediante la técnica estadística de pre y pos test.

f. Brindar conclusiones y/o recomendaciones en base a los resultados obtenidos mediante la aplicación de prueba estadística (T-Student para dos muestras relacionadas).

IV. CONSTRUCCIÓN DE UNA SOLUCIÓN CLOUD COMPUTING PARA FACILITAR LA ADOPCIÓN DEL PROCESO PERSONAL DE SOFTWARE

Esquema de Trabajo: En la siguiente tabla se presenta el resumen del uso de la metodología ICONIX con el PSP y los entregables de acuerdo a UML, esto constituye la guía para el desarrollo de la herramienta cloud en esta investigación.

FASE PSP	FASE ICONIX	Actividad	Entregable
Planificación	Análisis de Requerimientos	Identificar Requerimientos del Sistema	Lista de Requerimientos
		Identificar objetos del dominio y relaciones de agregación y generalización.	Modelo de Dominio
		Realizar un prototipo rápido	Prototipos
		Identificar Casos de Uso	Diagrama de Casos de Uso de Negocio (CUN)
		Organizar Casos de Uso en grupos (paquetes)	Diagrama de Casos de Uso Específico (requerimiento)
Diseño	Análisis y Diseño preliminar	Escribir Descripciones de Casos de Uso	Descripción de los Casos de Uso
		Análisis de Robustez	Diagrama de Robustez
		Finalizar Diagrama de Clases	Diagrama de Clases
	Diseño	Asignar comportamiento para cada Caso de Uso	Diagrama de Secuencia
Implementación	Implementación	Identificar los componentes	Diagrama de Componentes
		Escribir el código	Proyecto de Software en Netbeans (psp-tool.appspot.com)
		Muestra	Informe de aplicación de la muestra (Examen)
PostMortem		Resultados obtenidos en la implementación	Resumen del Plan de PSP (página 58)

Tabla 11: Matriz de Uso PSP-ICONIX-UML

4.1. PLANIFICACION

A. Análisis de requerimientos

- Requerimientos Funcionales basados en el Proceso de Software Personal.

  - ✓ Registrar y modificar los datos de los usuarios (docente y alumno).
  - ✓ Agregar proyectos en cada curso específico.
  - ✓ Registrar los tiempos en cada fase del proyecto.
  - ✓ Registrar defectos encontrados durante el proceso de desarrollo.
  - ✓ Registrar la estimación de tamaño y tiempo, basándose en el método PROBE.
  - ✓ Generar un resumen del Plan del Proyecto
  - ✓ Generar reportes de productividad para el docente y alumno.
  - ✓ Visualización de los Scripts por cada nivel del PSP.

- **Modelo de Dominio.**

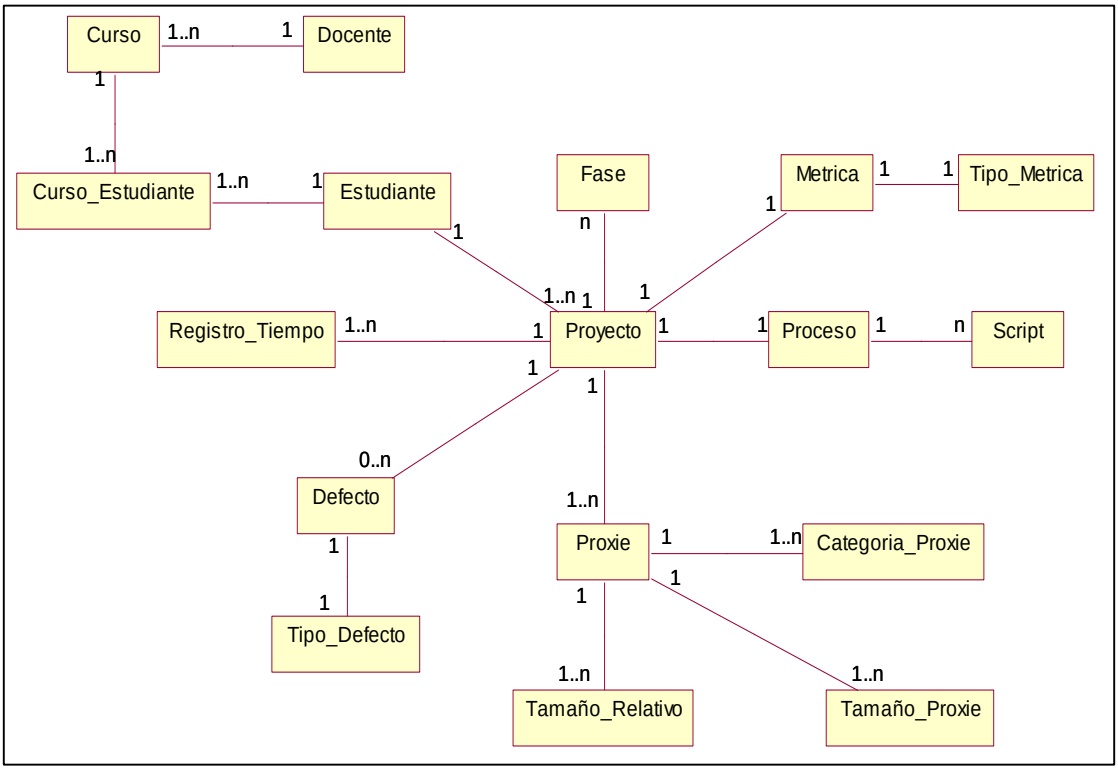


Figura 17: Modelo de Dominio

Fuente: Elaboración Propia

- **Prototipos Iniciales.**

  Para la elaboración de dichos prototipos se utilizó la herramienta Balsamiq Mockups.

  1) Página de Inicio

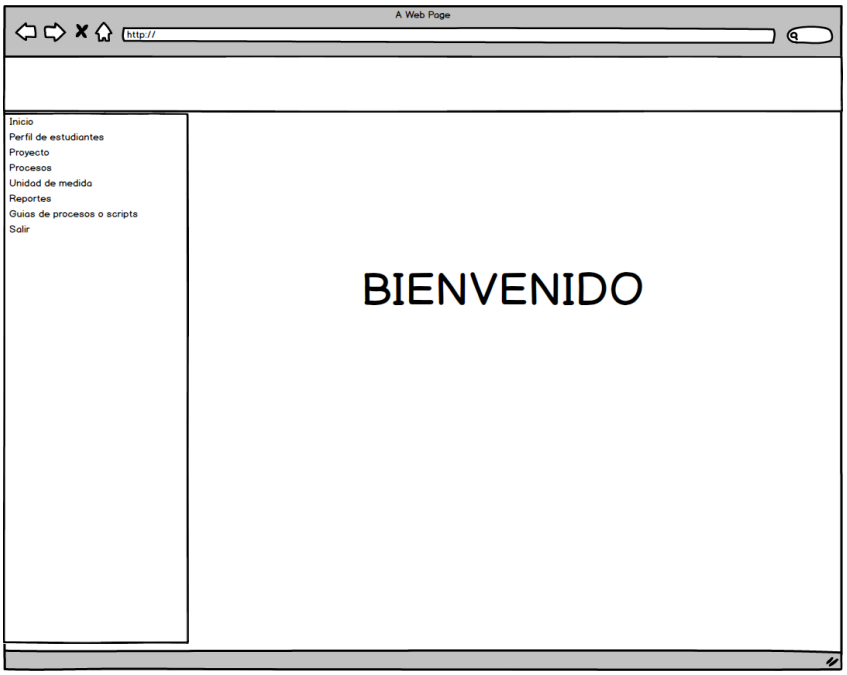


Figura 18: Prototipo - Inicio

Fuente: Elaboración Propia

  2) Creación de Proyecto

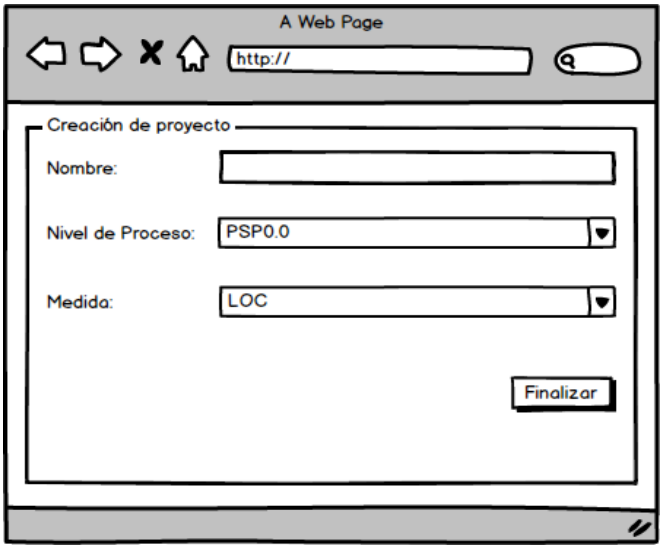


Figura 19: Prototipo – Creación de Proyecto

Fuente: Elaboración Propia

3) Creación del Proceso

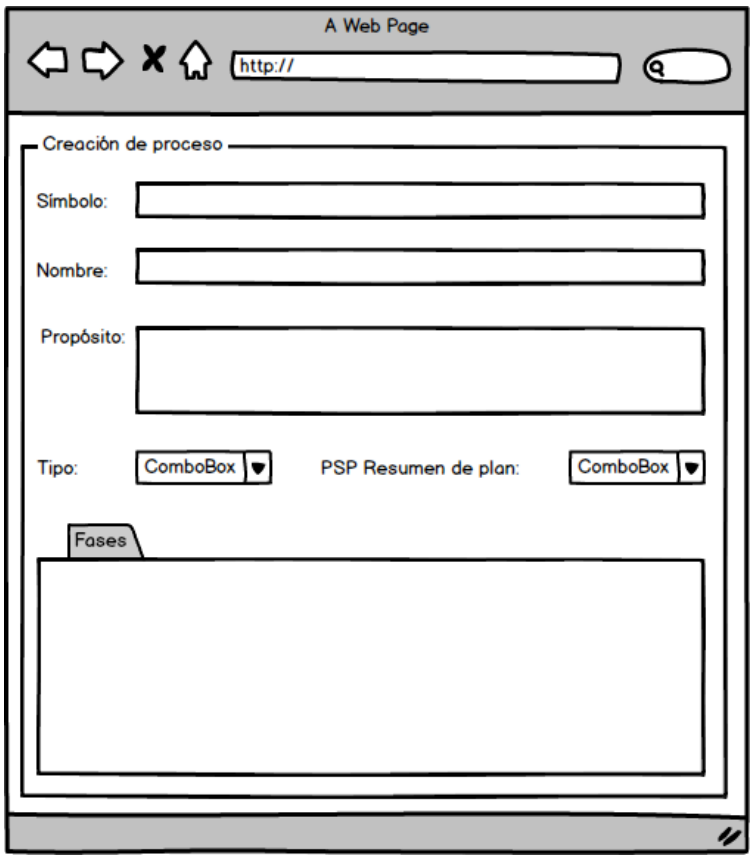


Figura 20: Prototipo – Creación de Proceso

Fuente: Elaboración Propia

4) Registro de Defectos

Figura 21: Prototipo – Registro de Defectos

Fuente: Elaboración Propia

5) Registro de Tiempo

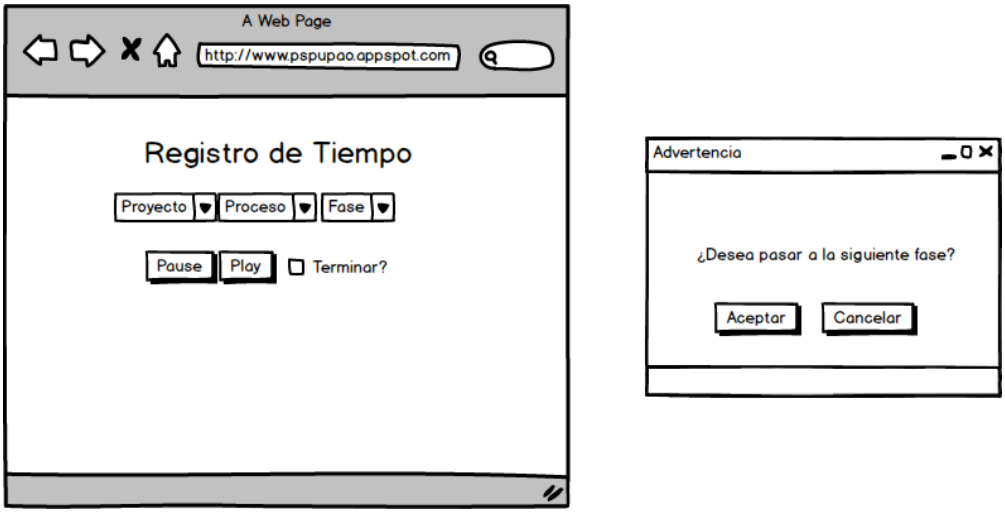


Figura 22: Prototipo – Registro de Tiempo

Fuente: Elaboración Propia

6) Resumen del Plan de Proyecto

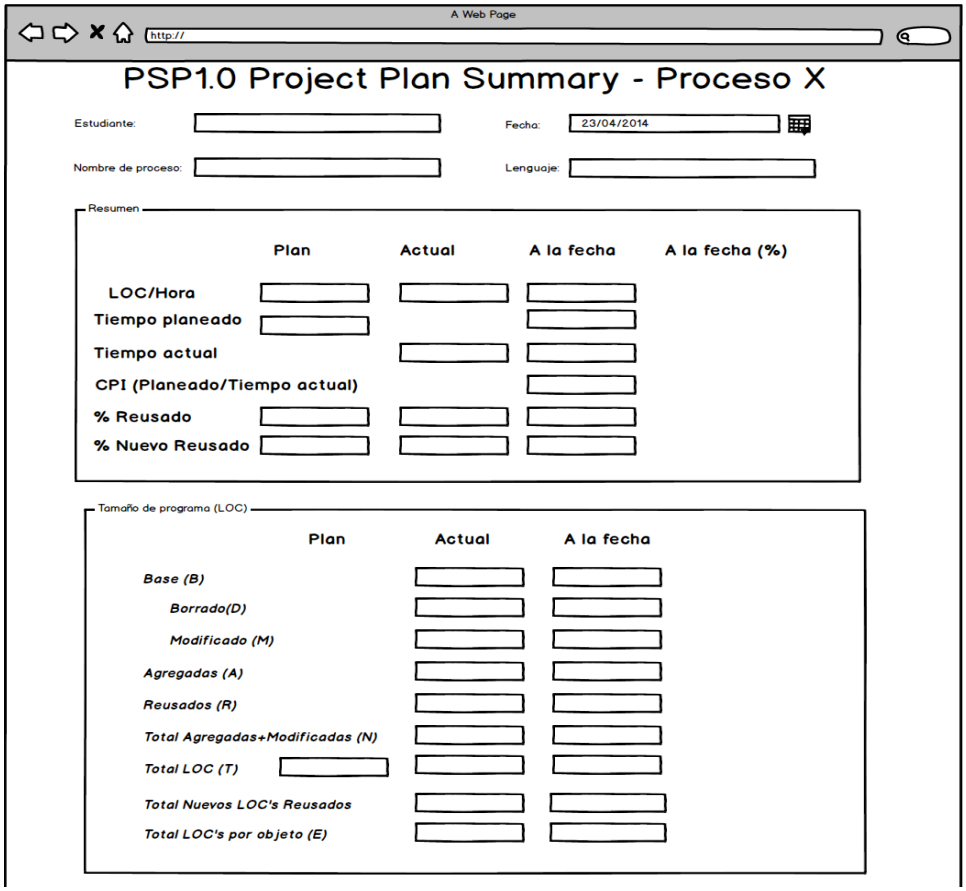


Figura 23: Prototipo – Resumen del Plan del Proyecto Parte 1

Fuente: Elaboración Propia

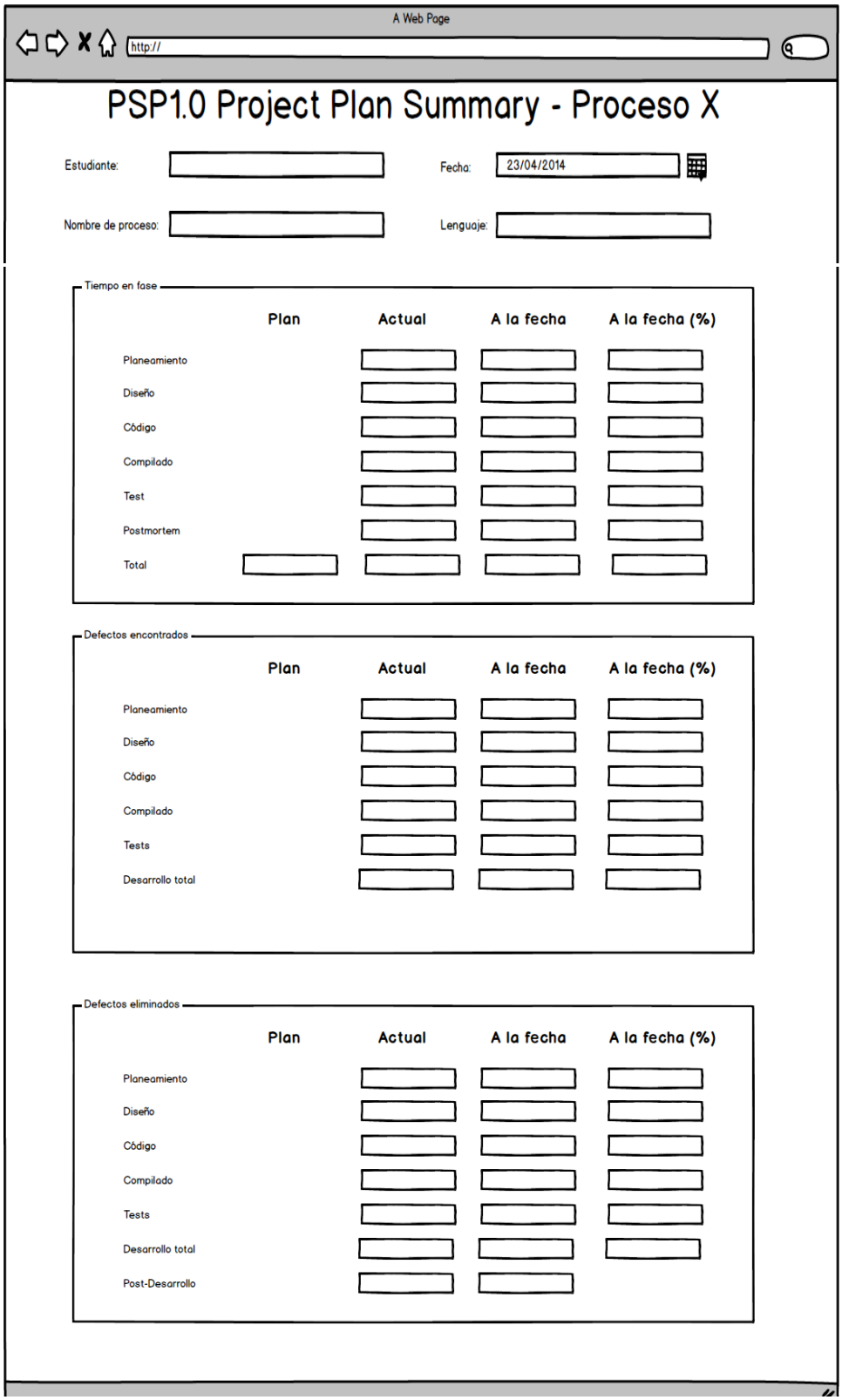


Figura 24: Prototipo – Resumen del Plan del Proyecto Parte 2

Fuente: Elaboración Propia

- **Diagrama de Caso de Uso de Negocio**

En base a lo estudiado sobre el Personal Software Process, se especificaron tres actores que engloban todo el proceso que conlleva PSP 1.1, los cuales se pueden visualizar en la siguiente figura:

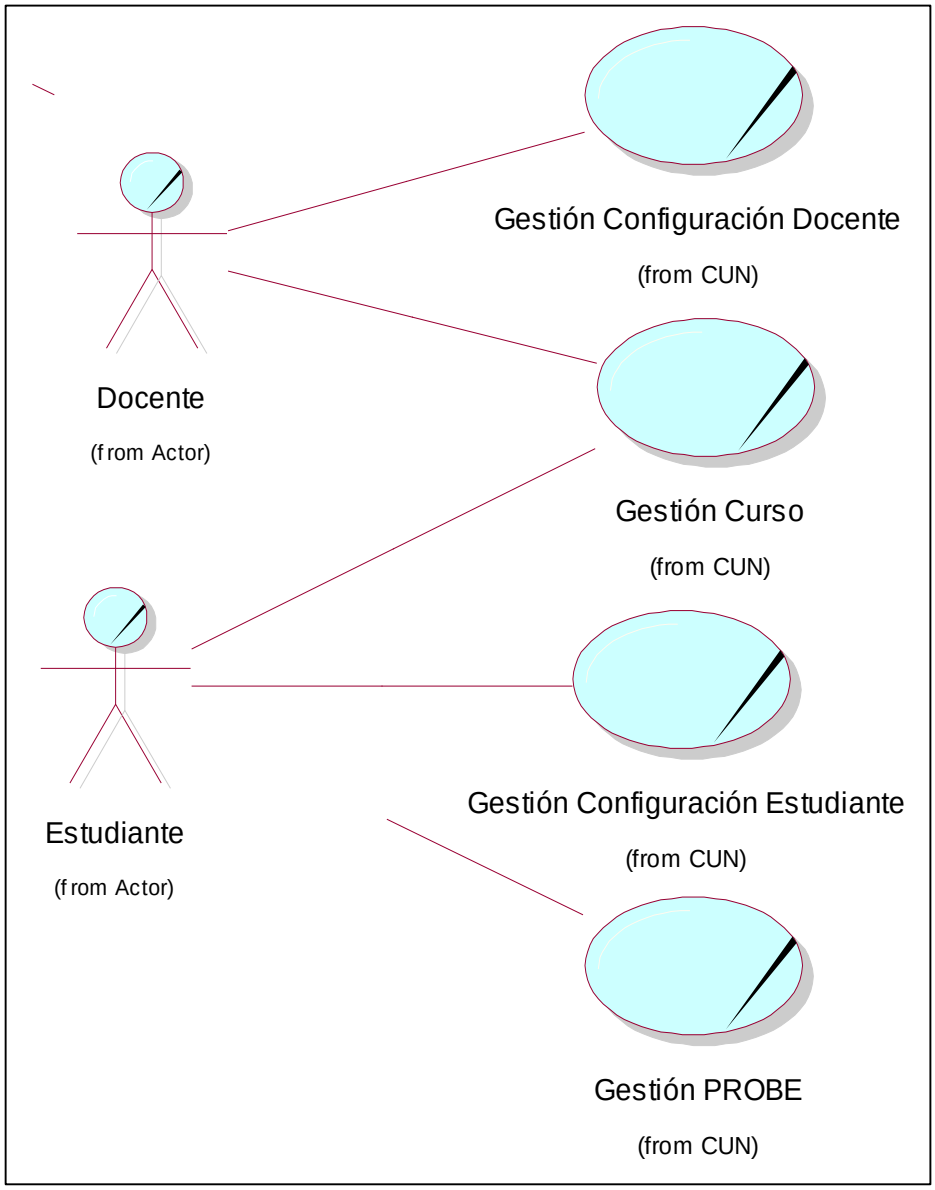


Figura 25: Caso de Uso de Negocio

Fuente: Elaboración Propia

- **Diagrama Casos de Uso Específico**

  ✓ Caso de Uso – Gestión Curso

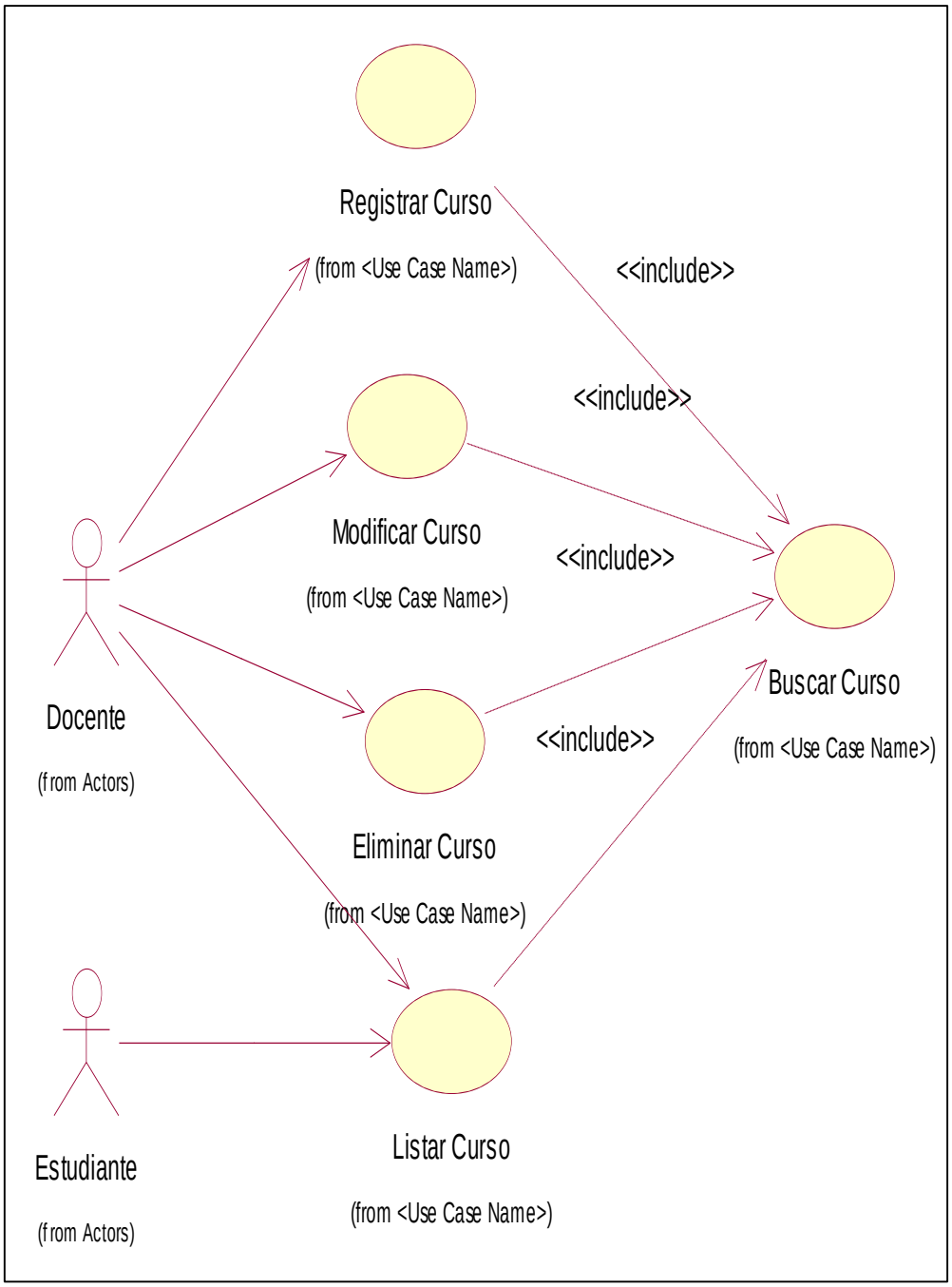


Figura 26: CU Gestión Curso

Fuente: Elaboración Propia

✓ Caso de Uso – Gestión Docente

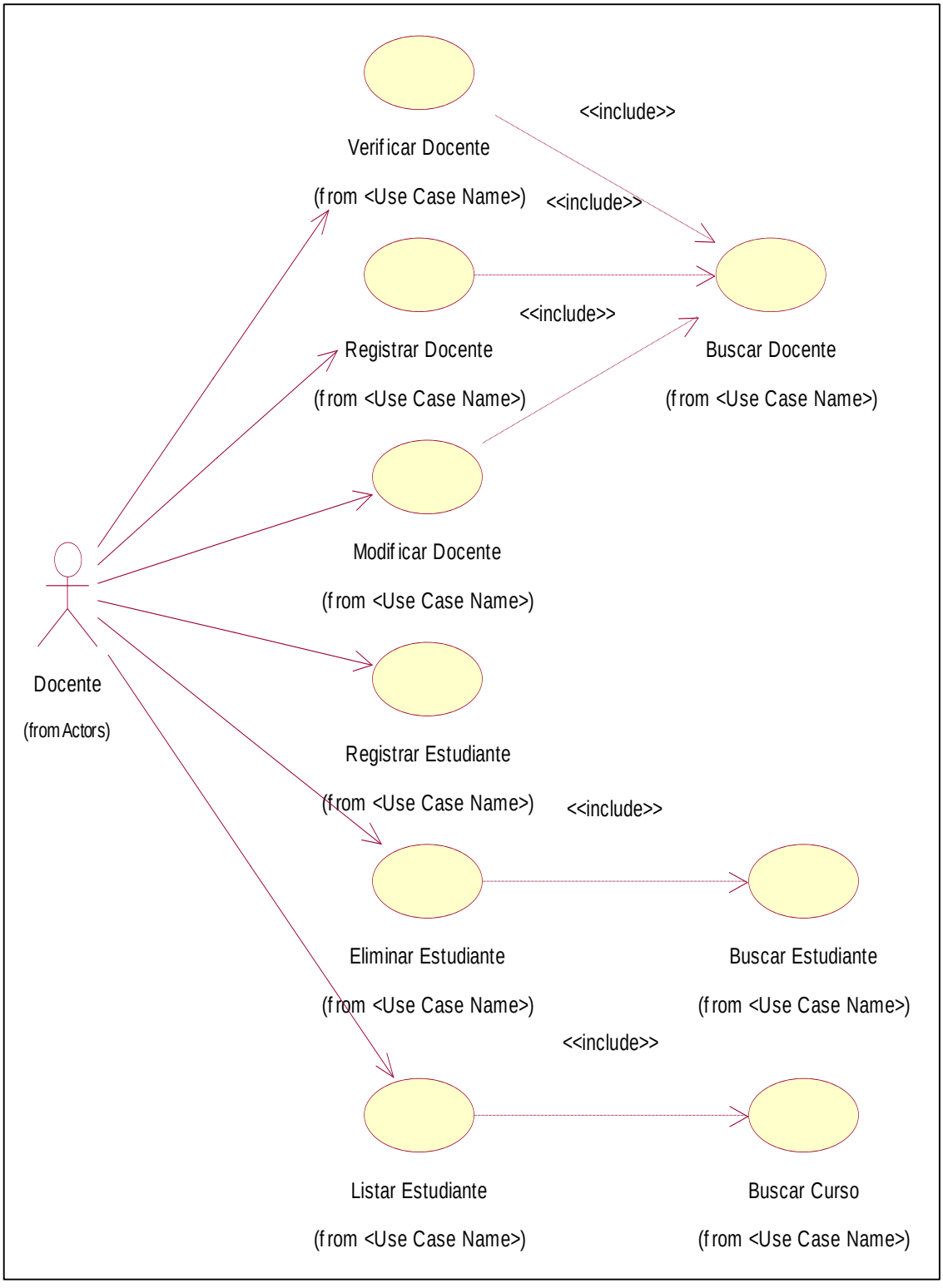


Figura 27: CU Gestión Docente

Fuente: Elaboración Propia

✓ Caso de Uso – Gestión Estudiante

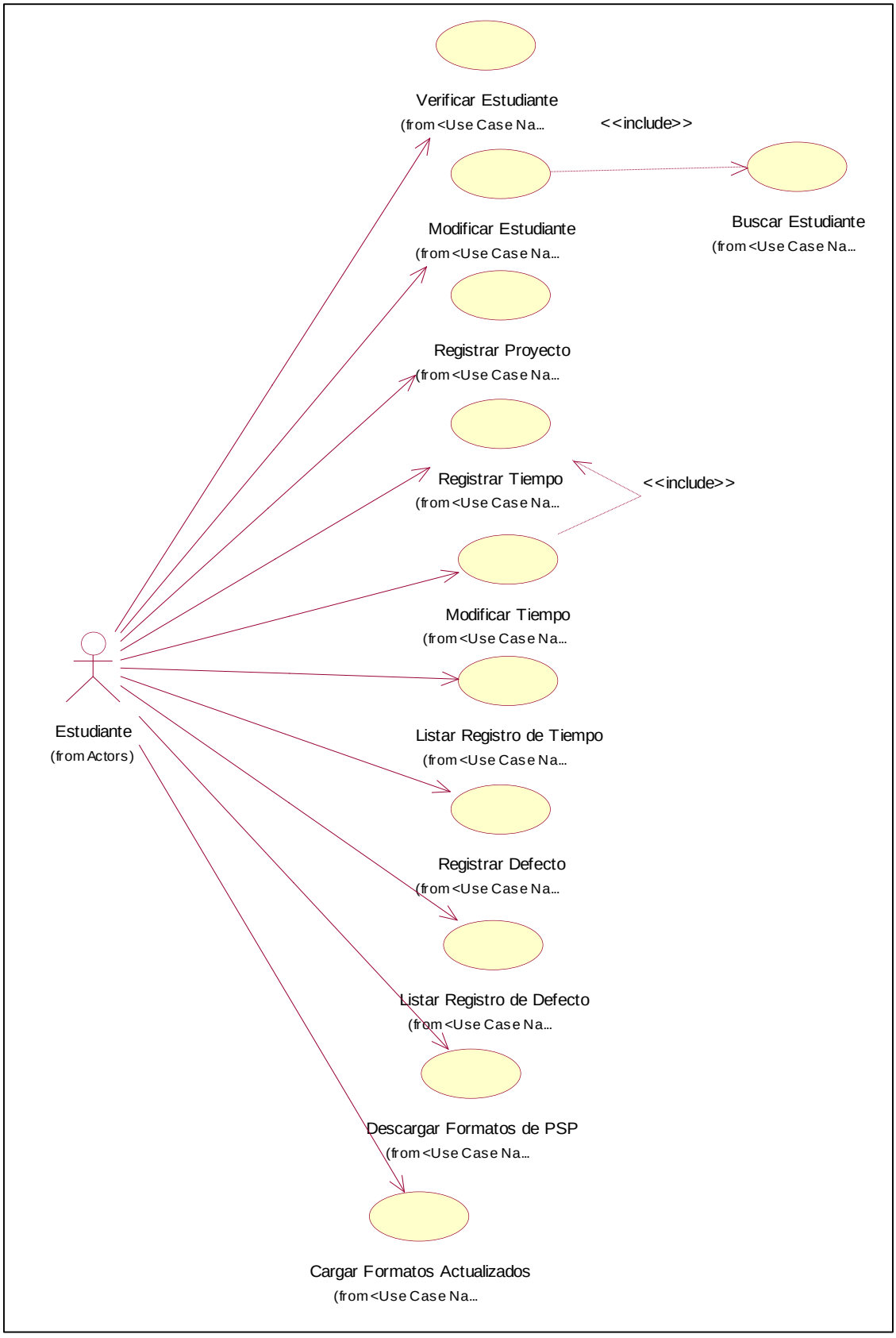


Figura 28: CU Gestión Estudiante

Fuente: Elaboración Propia

✓ Caso de Uso – Gestión PROBE

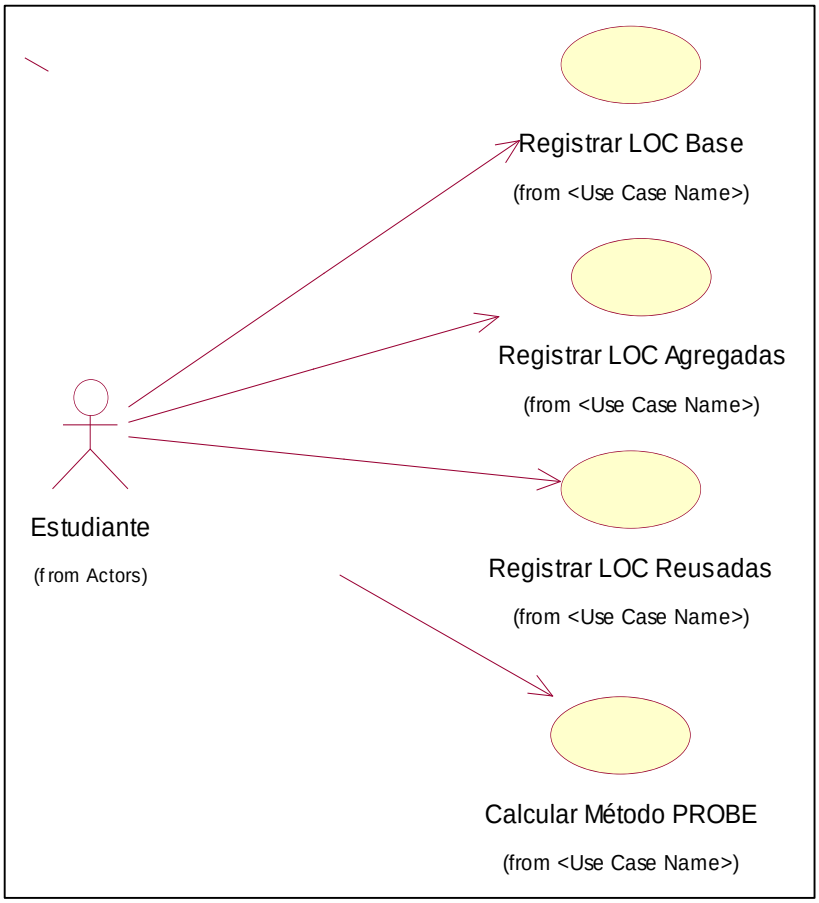


Figura 29: CU Gestión PROBE

Fuente: Elaboración Propia

4.2. DISEÑO

Dada que nuestra propuesta es una solución cloud computing. Es un tanto distinta de una aplicación web tradicional. Puesto que, al utilizar la arquitectura JDO junto con el Sandbox de GAE, no es necesario implementar mapeos de datos para un almacenamiento relacional, dado que, por definición la interface JDO junto GAE llevan a cabo esta "*on-the-fly mapping*". Es decir, es responsabilidad del propio GAE-SDK. La preocupación a nivel del desarrollo de la APP esta en implementar las clases en forma de entity-class e implementación de un controller-class para que las operaciones sobre los objetos sigan las reglas de negocio implementadas en esta última.

A. Análisis y Diseño Preliminar

- Descripción de los Casos de Uso

Caso de Uso	REGISTRAR CURSO	
Actor Principal	Docente	
Objetivo en Contexto	Permitir Gestionar el Registro de Cursos por Docentes registrados.	
Precondiciones	El docente tiene que estar registrado. Estar dentro del panel “Cursos”	
Secuencia	Paso	Acción
	1.	El docente da clic en la opción agregar Curso
	2.	El docente ingresa los datos del curso a registrar.
	3.	La aplicación confirma los datos guardados
	4.	Seleccionar la opción “Recargar Lista” para agregar a la lista el nuevo curso registrado
Post Condición	El docente podrá registrar más de un curso. El usuario podrá modificar los datos del curso.	

Tabla 12: Descripción CU - Registrar Curso

Caso de Uso	MODIFICAR CURSO	
Actor Principal	Docente.	
Objetivo en Contexto	Permitir al Docente, modificar los datos del curso registrado.	
Precondiciones	Haber registrado el curso. Estar dentro del panel “Cursos” Tener privilegios de docente	
Secuencia	Paso	Acción
	1.	El docente selecciona la opción “Modificar Curso”
	2.	El docente modifica los datos necesarios del curso seleccionado.
	3.	El docente guarda modificaciones.
	4.	La aplicación confirma datos guardados.
Post Condición	-	

Tabla 13: Descripción CU - Modificar Curso

Caso de Uso	LISTAR CURSO	
Actor Principal	Docente.	
Objetivo en Contexto	Permitir al Docente listar sus cursos registrados a la fecha.	
Precondiciones	Tener privilegios de docente	
Secuencia	Paso	Acción
	1.	El docente selecciona la opción “Cursos” dentro del panel Docente
	2.	La aplicación muestra todos los cursos registrados
Post Condición	-	

Tabla 14: Descripción CU - Listar Curso

Caso de Uso	REGISTRAR DOCENTE	
Actor Principal	Docente.	
Objetivo en Contexto	Admitir al docente registrar sus datos en la aplicación.	
Precondiciones	Tener cuenta en google.	
Secuencia	Paso	Acción
	1.	En la página principal seleccionar la opción “Ingresar”, o “Ingresar con una cuenta de google”
	2.	Google valida si el docente tiene una cuenta asociada al mismo, en caso de no ser así, se procederá a crear la cuenta en caso el docente lo quiera.
	3.	Si se tiene una cuenta en google, la aplicación le pedirá registrar los datos para Docente.
Post Condición	El docente podrá registrar sus cursos. El docente podrá agregar alumnos a sus cursos.	

Tabla 15: Descripción CU - Registrar Docente

Caso de Uso	MODIFICAR DOCENTE	
Actor Principal	Docente.	
Objetivo en Contexto	Permitir al Docente modificar sus datos.	
Precondiciones	Tener privilegios de docente	
Secuencia	Paso	Acción
	1.	El docente selecciona la opción “Perfil”
	2.	El docente actualiza sus datos.
	3.	El docente guarda las actualizaciones correspondientes.
	4.	La aplicación muestra el mensaje de confirmación.
Post Condición	-	

Tabla 16: Descripción CU - Modificar Docente

Caso de Uso	REGISTRAR ESTUDIANTE	
Actor Principal	Docente.	
Objetivo en Contexto	Permitir al Docente registrar a los estudiantes (mediante su cuenta de google) en un respectivo curso.	
Precondiciones	Tener privilegios de docente Haber registrado el curso El estudiante debe tener una cuenta de google	
Secuencia	Paso	Acción
	1.	El docente selecciona la opción “Cursos” dentro del panel Docente
	2.	En la lista de curso(s) , seleccionar la acción del curso “Listar Estudiantes”
	3.	Seleccionar la opción “Agregar Estudiante en Curso”
	4.	Ingresar la cuenta de google del estudiante y lo registra
Post Condición	El docente visualiza mediante una lista a los estudiantes agregados. El docente puede ver que estudiantes se han registrado completamente. El docente puede quitar a un estudiante del curso	

Tabla 17: Descripción CU - Registrar Estudiante

Caso de Uso	VERIFICAR ESTUDIANTE	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante validar sus datos principales dentro de la aplicación.	
Precondiciones	El estudiante debe tener una cuenta de google. El estudiante debe haber sido registrado por un docente.	
Secuencia	Paso	Acción
	1.	El estudiante ingresa al link, que se le envió a su cuenta de google cuando el docente lo registró.
	2.	Dentro de la aplicación el estudiante registra el resto de sus datos.
	3.	Se Guarda los cambios efectuados
	4.	La aplicación muestra un mensaje de confirmación de registro.
Post Condición	El estudiante puede agregar más de un proyecto en el curso registrado. El estudiante puede editar su perfil. El estudiante puede listar los cursos a los cuales se les ha registrado.	

Tabla 18: Descripción CU - Verificar Estudiante

Caso de Uso	VERIFICAR DOCENTE	
Actor Principal	Docente.	
Objetivo en Contexto	Permitir al docente validar su cuenta de Google dentro de la aplicación.	
Precondiciones	El docente debe tener una cuenta de google.	
Secuencia	Paso	Acción
	1.	El docente debe dar clic en la opción Ingresar de la página principal de la aplicación.
	2.	El docente ingresa su cuenta de Gmail para ser registrado.
	3.	Se Guarda los cambios efectuados
	4.	La aplicación muestra un mensaje de confirmación de registro.
Post Condición	El docente puede editar su perfil. El docente puede registrar cursos.	

Tabla 19: Descripción CU - Verificar Docente

Caso de Uso	MODIFICAR ESTUDIANTE	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante modificar sus datos.	
Precondiciones	Tener privilegios de estudiante	
Secuencia	Paso	Acción
	1.	El estudiante selecciona la opción “Perfil de Estudiante”
	2.	El estudiante actualiza sus datos.
	3.	El estudiante guarda las actualizaciones correspondientes.
	4.	La aplicación muestra el mensaje de confirmación.
Post Condición	-	

Tabla 20: Descripción CU - Modificar Estudiante

Caso de Uso	REGISTRAR PROYECTO	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante agregar un proyecto a un curso específico.	
Precondiciones	Tener privilegios de estudiante Tener un curso registrado	
Secuencia	Paso	Acción
	1.	El estudiante selecciona la opción “Cursos”
	2.	En el curso seleccionado, dar clic en la acción “Listar Proyectos”.
	3.	En la nueva ventana, seleccionar la opción “Agregar Proyecto”
	4.	Registrar todos los datos necesarios para el proyecto.
	5.	La aplicación muestra una lista con los proyectos agregados recientemente.
Post Condición	El estudiante puede dar de baja a un proyecto. El estudiante puede abrir un proyecto. El estudiante puede terminar un proyecto.	

Tabla 21: Descripción CU - Registrar Proyecto

Caso de Uso	REGISTRAR TIEMPO	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante registrar el tiempo usado en cada fase del proyecto.	
Precondiciones	Estar dentro del panel del proyecto.	
Secuencia	Paso	Acción
	1.	En el panel “Registro de Tiempo”, seleccionar la fase en la cual se encuentra el proyecto.
	2.	Dar clic en el botón play
	3.	Si se ha terminado con una fase, dar clic en el botón pause
	4.	La aplicación automáticamente registra el tiempo de cada fase, de acuerdo al tiempo del sistema operativo
Post Condición	El estudiante puede registrar más de una vez una sola fase. El estudiante puede listar el registro de tiempo El estudiante puede actualizar las interrupciones en la lista de registro de tiempo.	

Tabla 22: Descripción CU - Registrar Tiempo

Caso de Uso	MODIFICAR TIEMPO	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante actualizar las interrupciones y comentarios de la lista de los registros de tiempos guardados.	
Precondiciones	Tener un proyecto registrado Estar registrando el tiempo de una fase	
Secuencia	Paso	Acción
	1.	En el panel “Formatos”, seleccionar la opción “Bitácora de Tiempo”
	2.	En caso de haber interrupciones y/o comentarios, el estudiante da clic en el botón modificar interrupción y/o botón modificar comentario.
	3.	Ingresa los datos y automáticamente se actualiza las modificaciones correspondientes.
Post Condición	El total del tiempo registrado, se usa en el resumen del plan de proyecto.	

Tabla 23: Descripción CU - Modificar Tiempo

Caso de Uso	REGISTRAR DEFECTO	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante registrar los defectos encontrados dentro de un proyecto.	
Precondiciones	Tener privilegios de estudiante Tener un proyecto registrado	
Secuencia	Paso	Acción
	1.	En el panel “Registro de Defectos”, seleccionar la opción “Agregar Defecto”
	2.	Llenar todos los datos referentes al defecto
	3.	En la misma página se muestra el mensaje de confirmación de registro.
Post Condición	El estudiante puede listar todos los defectos registrados en la opción “Bitácora de Defectos” El estudiante puede agregar más de un defecto por fase	

Tabla 24: Descripción CU - Registrar Defecto

Caso de Uso	DESCARGAR FORMATOS PSP	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante descargar los formatos necesarios para ser llenados.	
Precondiciones	Tener privilegios de estudiante Tener un proyecto registrado	
Secuencia	Paso	Acción
	1.	Dentro del Panel "Formatos para llenar", seleccionar que formato se va a utilizar
	2.	En la misma página la aplicación muestra dos acciones a tomar.
	3.	Dar clic en la opción (o link) "Nombre del Formato"
	4.	Llenar la plantilla de acuerdo a lo que se pide.
Post Condición	El estudiante puede descargar más de un formato.	

Tabla 25: Descripción CU - Descargar Formato PSP

Caso de Uso	CARGAR FORMATOS ACTUALIZADOS	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante subir los formatos del PSP actualizados.	
Precondiciones	Tener privilegios de estudiante Tener un proyecto registrado Tener actualizado el formato a subir.	
Secuencia	Paso	Acción
	1.	Dentro del Panel "Formatos para llenar", el estudiante selecciona que formato va a subir
	2.	En la misma página la aplicación muestra dos acciones a tomar.
	3.	Dar clic en el botón "Seleccionar Archivo"
	4.	Dar clic en el botón "Subir"
Post Condición	El estudiante puede subir más de un formato actualizado.	

Tabla 26: Descripción CU - Cargar Formato PSP

Caso de Uso	REGISTRAR LOC BASE	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante registrar las líneas de código del programa base, ya sea en la fase de Planificación (LOC Base Planificadas) o en la fase de PostMortem (LOC Base Real)	
Precondiciones	Tener privilegios de estudiante El tiempo en la fase tiene que estar corriendo (registrándose). Estar dentro del panel de un Proyecto Registrado	
Secuencia	Paso	Acción
	1.	En el panel "Formatos", seleccionar la opción "Estimación de Tamaño"
	2.	Llenar los datos del entorno "Base"
	3.	La aplicación mostrará al estudiante el total de LOC Base, de acuerdo a lo llenado.
	4.	El estudiante debe hacer clic en el botón "Modificar", ya sea para partes bases planeadas o reales.
Post Condición	El estudiante puede estimar el tamaño de su programa en base a LOC. El estudiante puede registrar también los LOC Base Reales	

Tabla 27: Descripción CU - Registrar LOC Base

Caso de Uso	REGISTRAR LOC AGREGADAS	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante registrar las líneas de código agregadas al programa base, ya sea en la fase de Planificación (LOC Base Planificadas) o en la fase de PostMortem (LOC Base Real)	
Precondiciones	Tener privilegios de estudiante El tiempo en la fase tiene que estar corriendo (registrándose). Estar dentro del panel de un Proyecto Registrado	
Secuencia	Paso	Acción
	1.	En el panel "Formatos", seleccionar la opción "Estimación de Tamaño"
	2.	Llenar los datos del entorno "Agregadas"
	3.	La aplicación mostrará al estudiante el total de LOC Agregadas, de acuerdo a lo llenado.
	4.	El estudiante debe hacer clic en el botón "Modificar", ya sea para partes agregadas planeadas o reales.
Post Condición	El estudiante puede estimar el tamaño de su programa en base a LOC. El estudiante puede registrar también los LOC Agregadas Reales	

Tabla 28: Descripción CU - Registrar LOC Agregadas

Caso de Uso	REGISTRAR LOC REUSADAS	
Actor Principal	Estudiante.	
Objetivo en Contexto	Permitir al estudiante registrar las líneas de código reusadas en el programa, ya sea en la fase de Planificación (LOC Base Planificadas) o en la fase de PostMortem (LOC Base Real)	
Precondiciones	Tener privilegios de estudiante El tiempo en la fase tiene que estar corriendo (registrándose). Estar dentro del panel de un Proyecto Registrado	
Secuencia	Paso	Secuencia
	1.	En el panel "Formatos", seleccionar la opción "Estimación de Tamaño"
	2.	Llenar los datos del entorno "Reusado"
	3.	La aplicación mostrará al estudiante el total de LOC Agregadas, de acuerdo a lo llenado.
	4.	El estudiante debe hacer clic en el botón "Modificar", ya sea para partes reusadas planeadas o reales.
Post Condición	El estudiante puede estimar el tamaño de su programa en base a LOC. El estudiante puede registrar también los LOC Reusadas Reales	

Tabla 29: Descripción CU - Registrar LOC Reusadas

Caso de Uso	CALCULAR METODO PROBE	
Actor Principal	Estudiante.	
Objetivo en Contexto	Mostrar al estudiante los resultados obtenidos mediante el método PROBE.	
Precondiciones	Tener privilegios de estudiante El tiempo en la fase tiene que estar corriendo (registrándose). Estar dentro del proceso "Estimación de Tamaño" Haber registrado los LOC agregadas, base y reusadas.	
Secuencia	Paso	Secuencia
	1.	Después de registrar los LOC planeadas, dar clic en el botón "PROBE"
	2.	La aplicación muestra los cálculos realizados, señalando el tamaño estimado del programa.
Post Condición	El estudiante puede registrar la estimación de LOC Agregadas y Modificadas. El estudiante puede registrar el tiempo total estimado para el desarrollo del programa.	

Tabla 30: Descripción CU - Calcular método PROBE

a. Verificar Usuario (Docente-Estudiante)

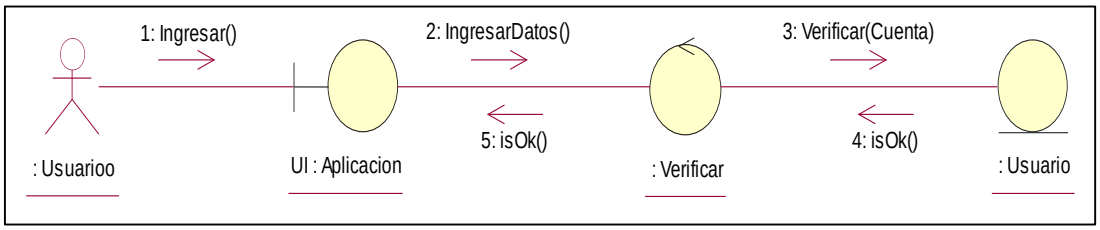


Figura 30: Diagrama de Robustez – Verificar Usuario

Fuente: Elaboración Propia

b. Gestión Curso por parte del Docente

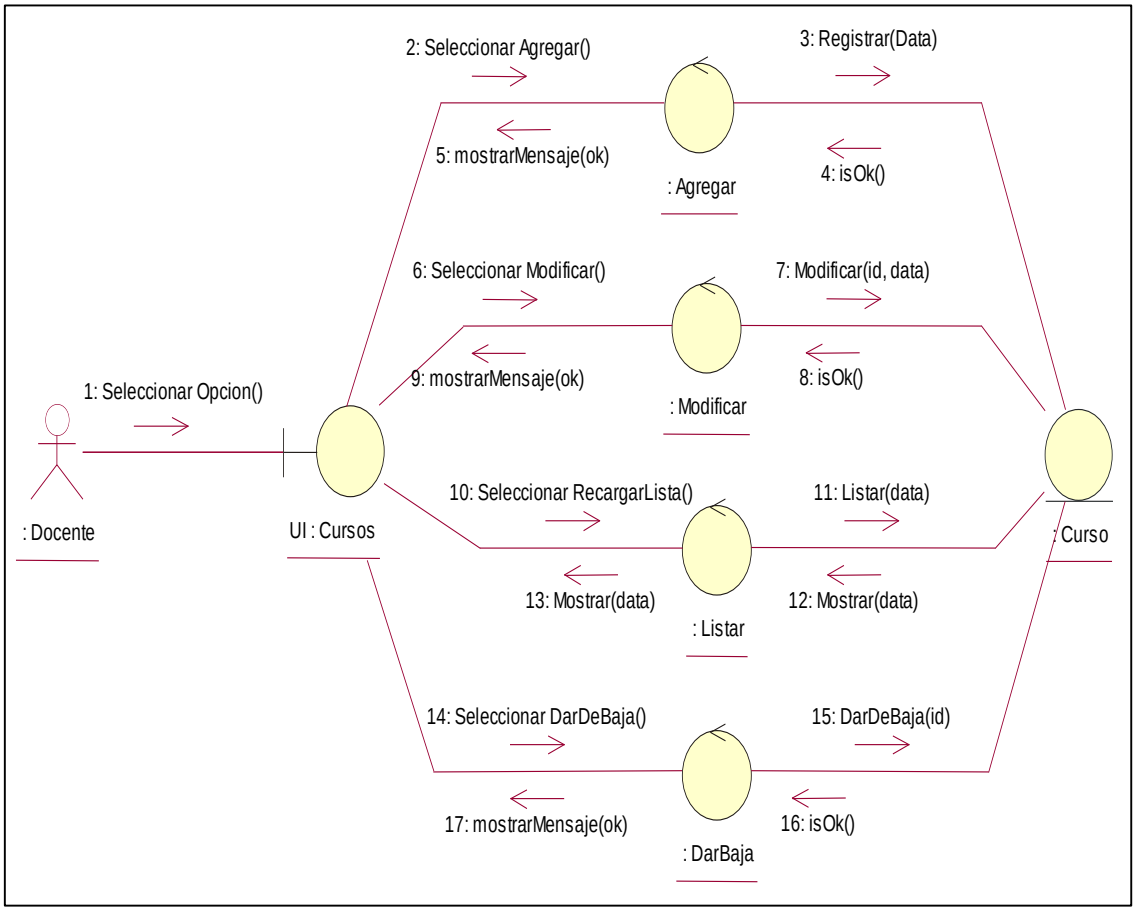


Figura 31: Diagrama de Robustez – Gestión Curso Docente

Fuente: Elaboración Propia

c. Gestión Docente (Registrar-Modificar)

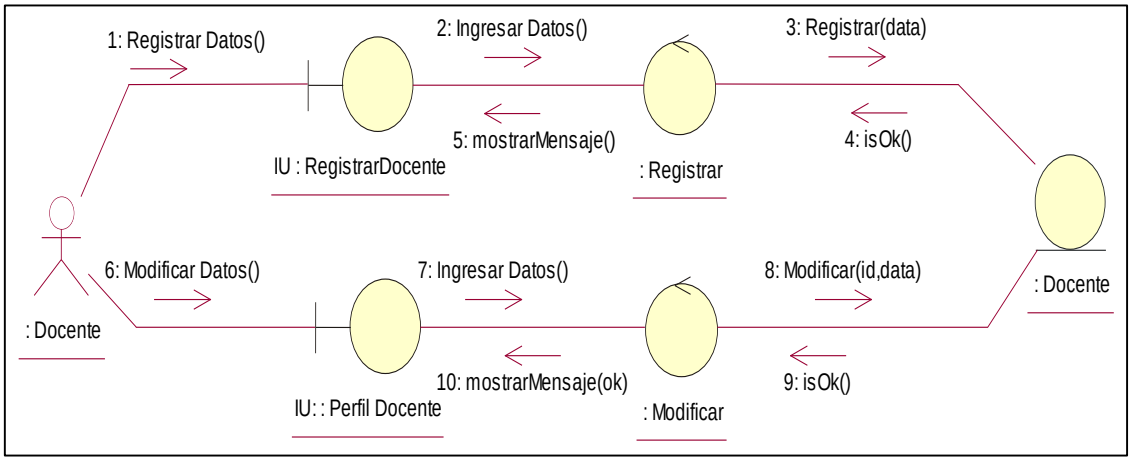


Figura 32: Diagrama de Robustez – Gestión Docente

Fuente: Elaboración Propia

d. Gestión Estudiante por parte del Docente

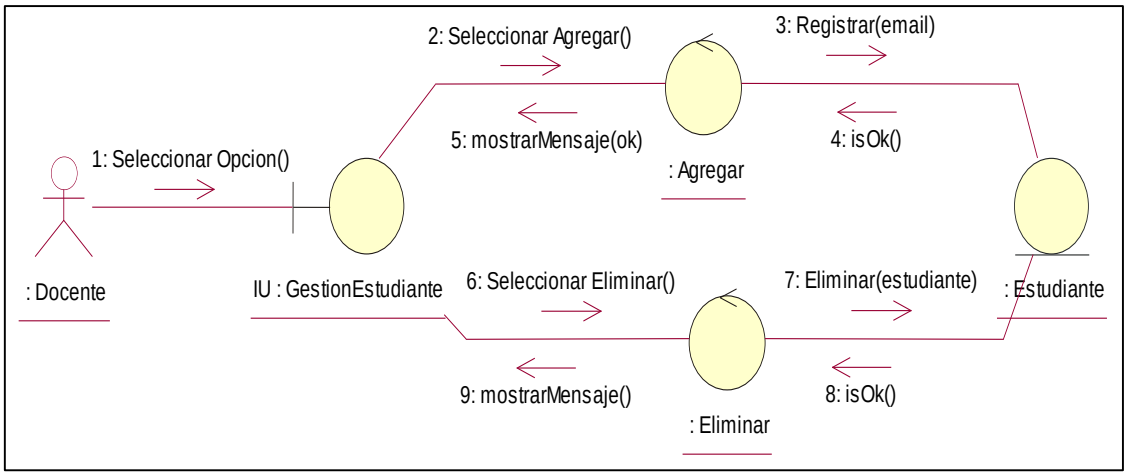


Figura 33: Diagrama de Robustez – Gestión Estudiante Docente

Fuente: Elaboración Propia

e. Listar Estudiantes para el Docente

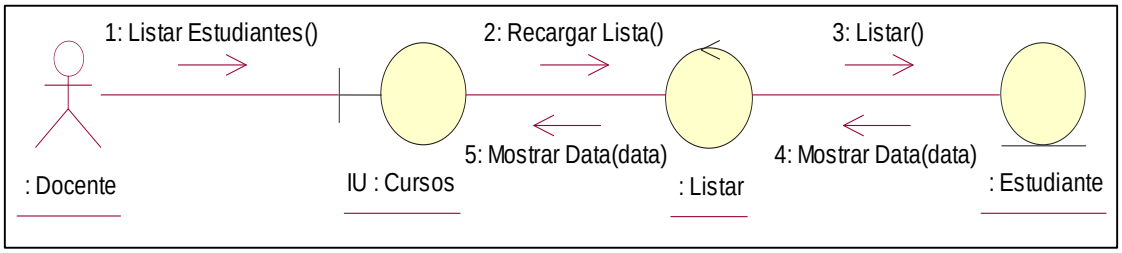


Figura 34: Diagrama de Robustez – Listar Estudiantes

Fuente: Elaboración Propia

f. Listar Cursos para el Estudiante

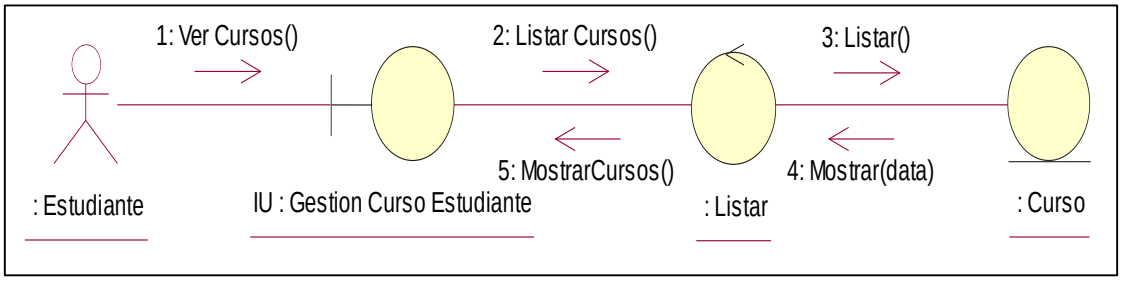


Figura 35: Diagrama de Robustez – Listar Cursos Estudiante

Fuente: Elaboración Propia

g. Gestión Proyecto (Agregar-Listar)

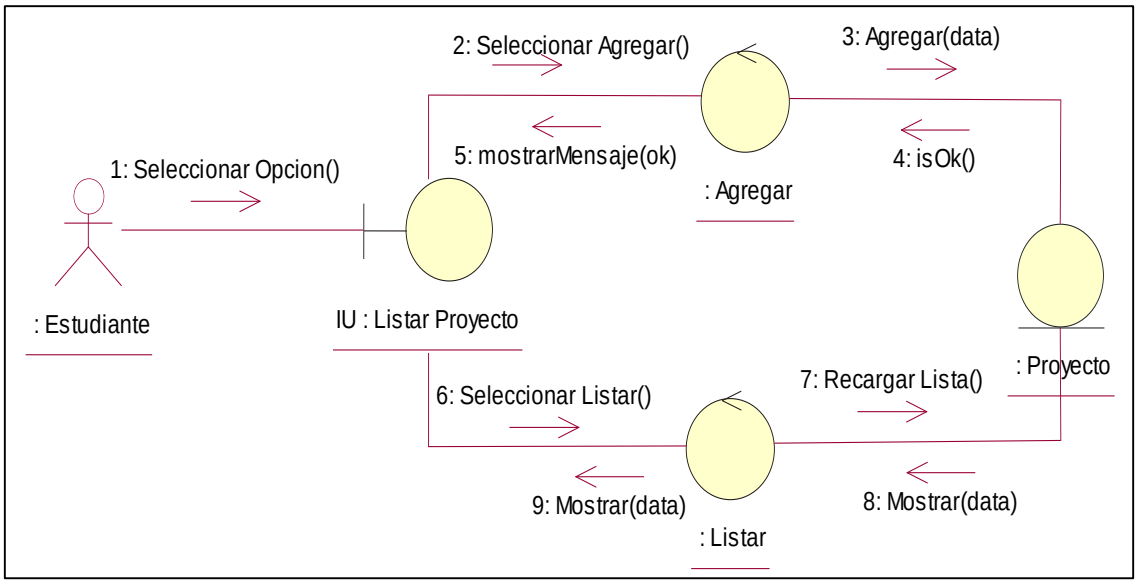


Figura 36: Diagrama de Robustez – Gestión Proyecto

Fuente: Elaboración Propia

h. Registrar Tiempo en cada Fase

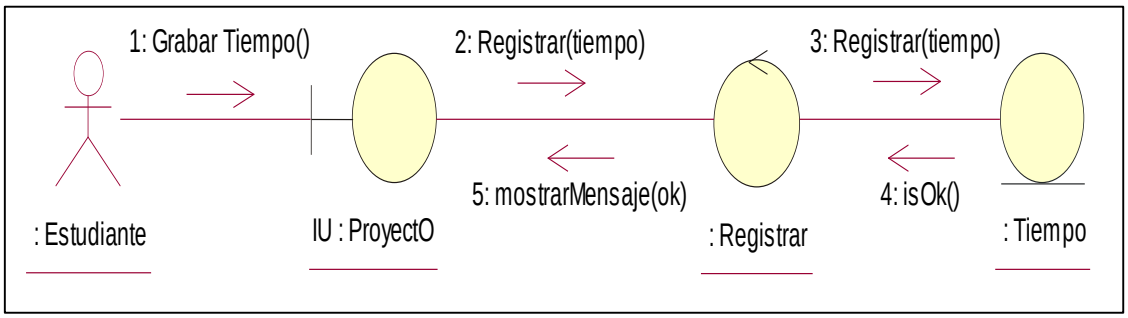


Figura 37: Diagrama de Robustez – Registrar Tiempo

Fuente: Elaboración Propia

i. Gestión Bitácora de Tiempo (Listar-Modificar)

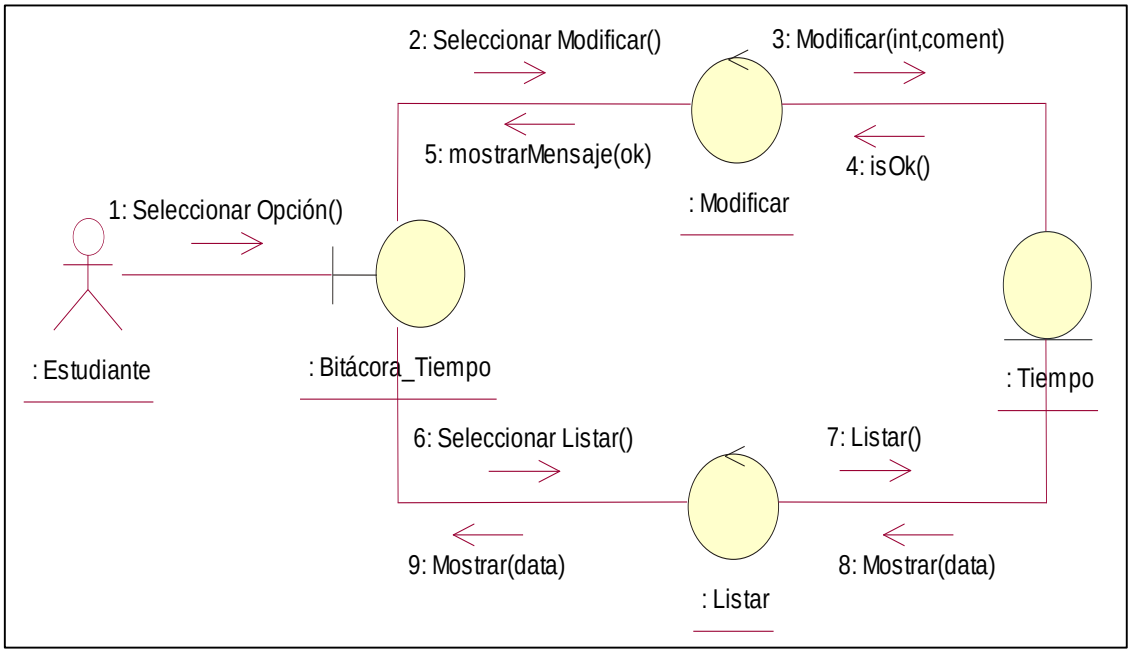


Figura 38: Diagrama de Robustez – Gestión Bitácora de Tiempo

Fuente: Elaboración Propia

j. Gestión Defecto (Agregar - Listar)

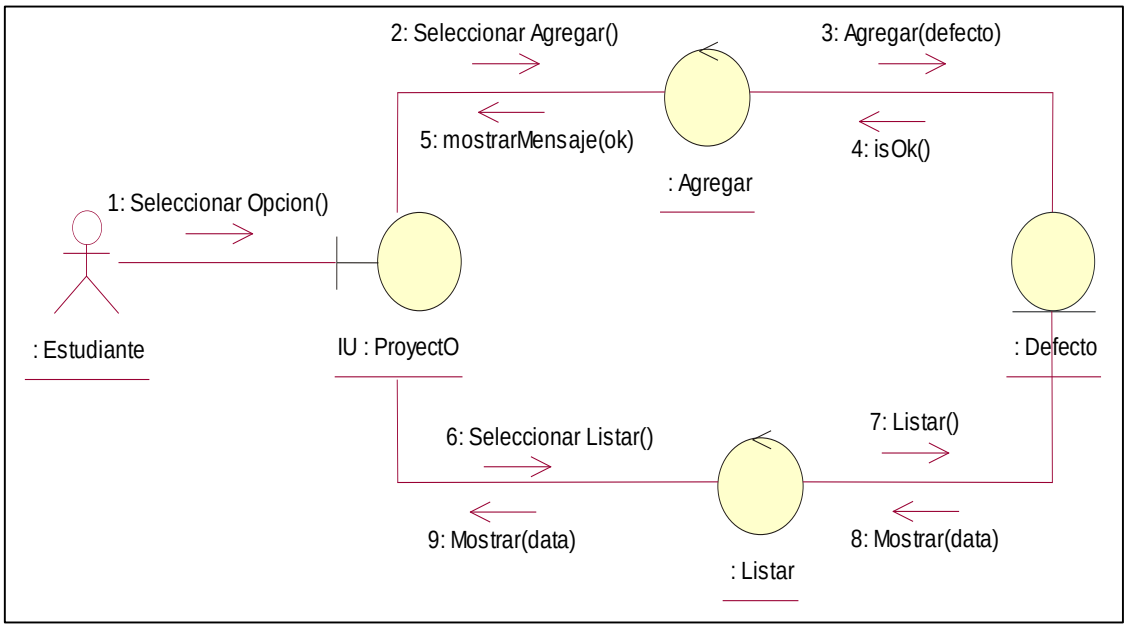


Figura 39: Diagrama de Robustez – Gestión Defecto

Fuente: Elaboración Propia

k. Gestión Archivo (Descargar-Cargar Formato)

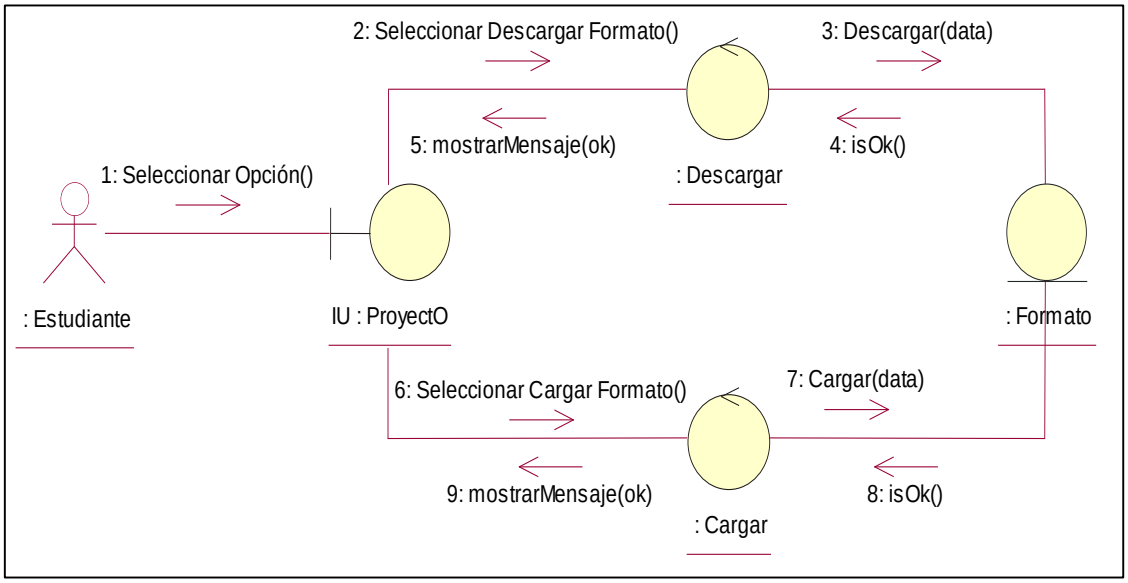


Figura 40: Diagrama de Robustez – Gestión Archivo

Fuente: Elaboración Propia

l. Registrar LOC (Base –Agregada-Reusada)

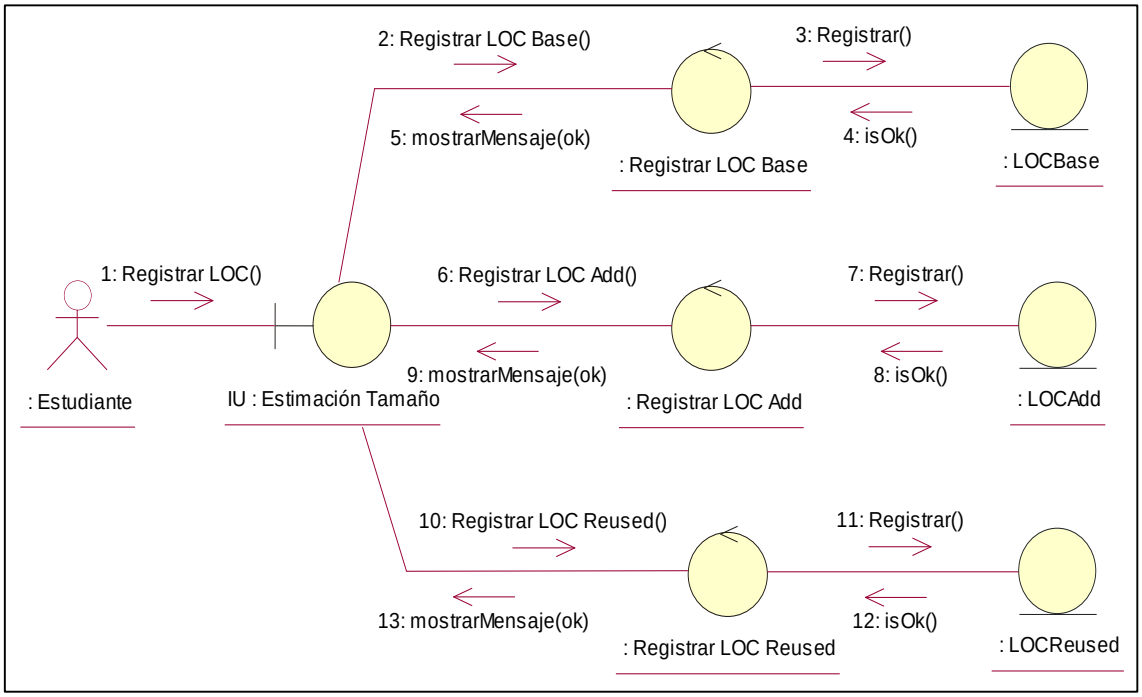


Figura 41: Diagrama de Robustez – Registrar LOC

Fuente: Elaboración Propia

- Diagrama de clases:

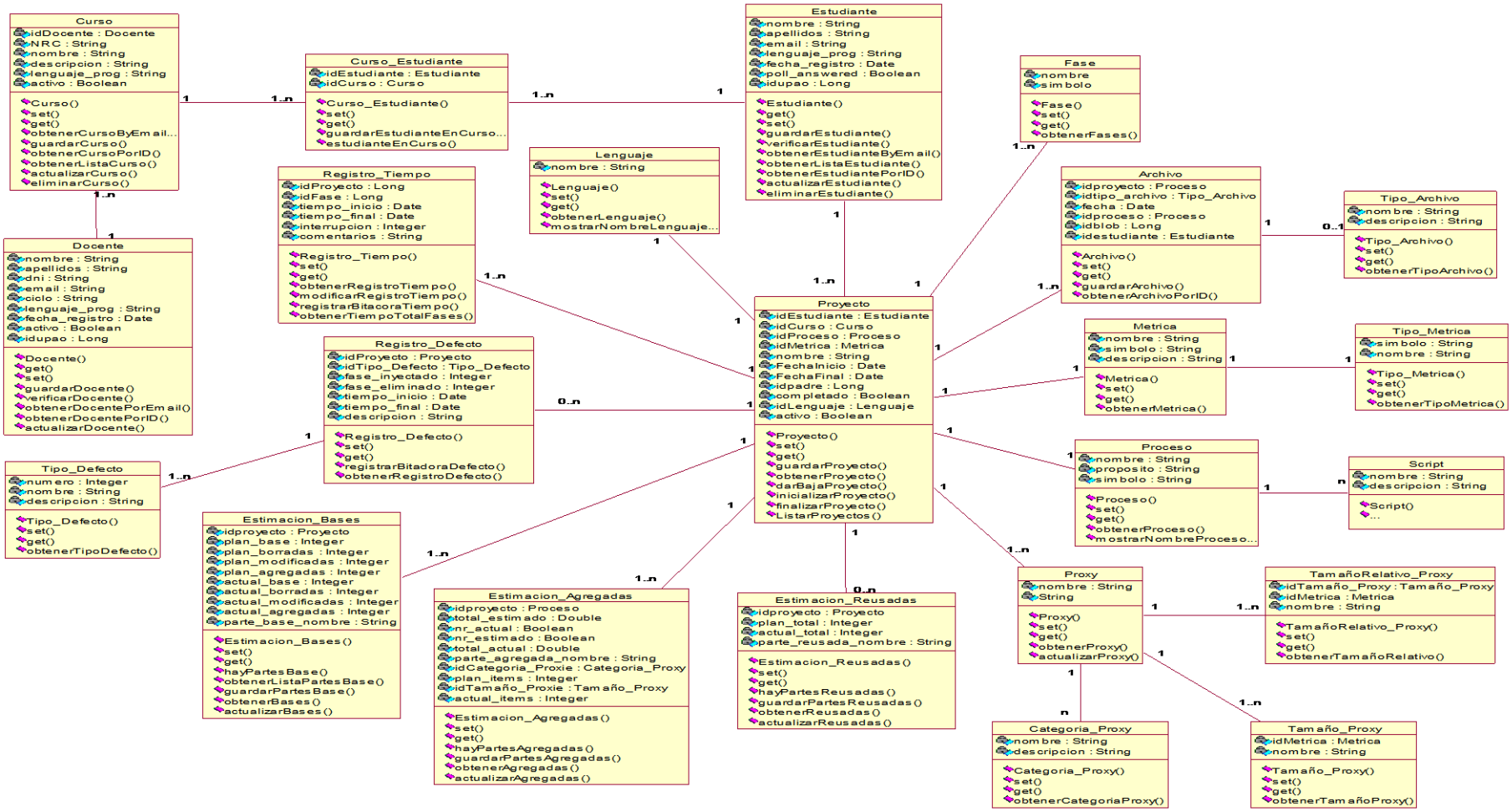


Figura 42: Diagrama Clase

Fuente: Elaboración Propia

B. Diseño

- Diagrama de Secuencia – acomodar, ampliar

a. Verificar Usuario

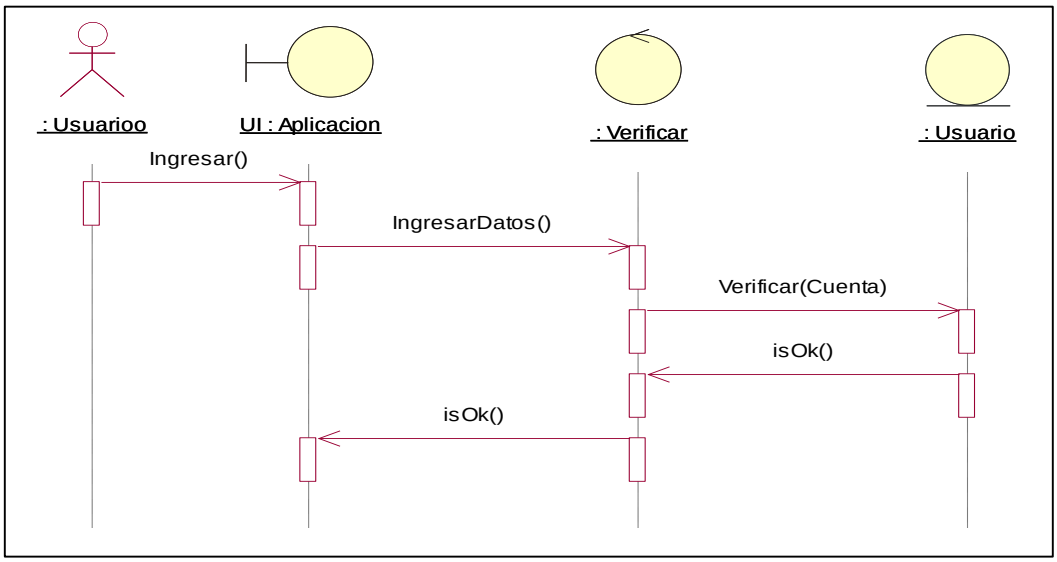


Figura 43: Diagrama de Secuencia – Verificar Usuario

Fuente: Elaboración Propia

b. Gestión Docente (Registrar-Modificar)

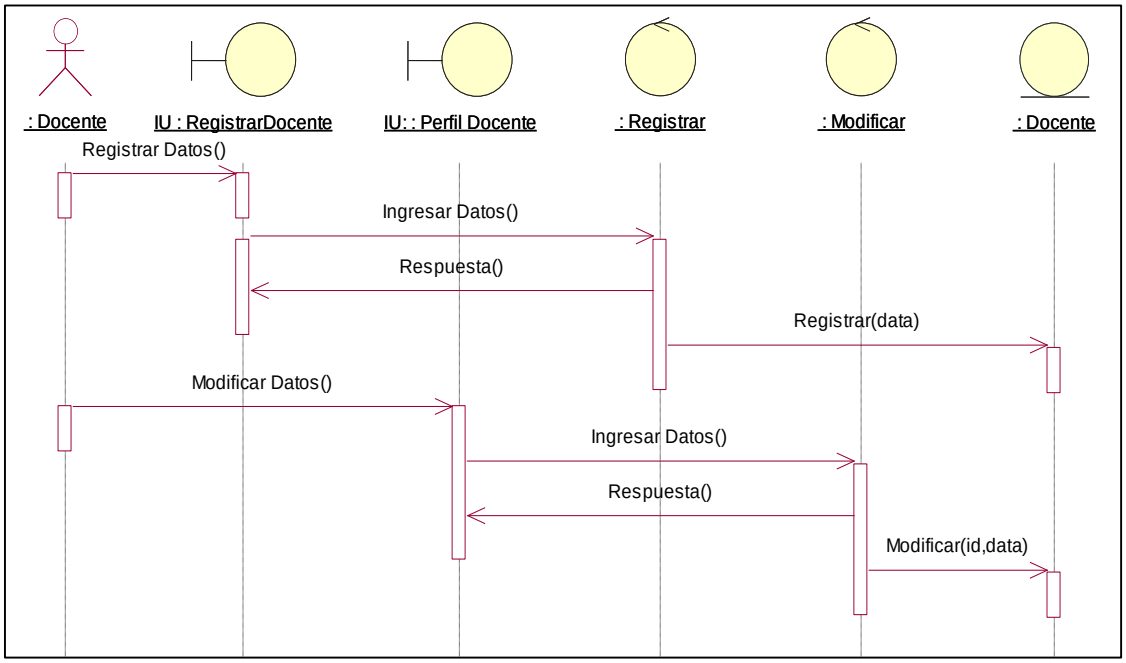


Figura 44: Diagrama de Secuencia – Gestión Docente

Fuente: Elaboración Propia

c. Gestión Curso por parte del Docente

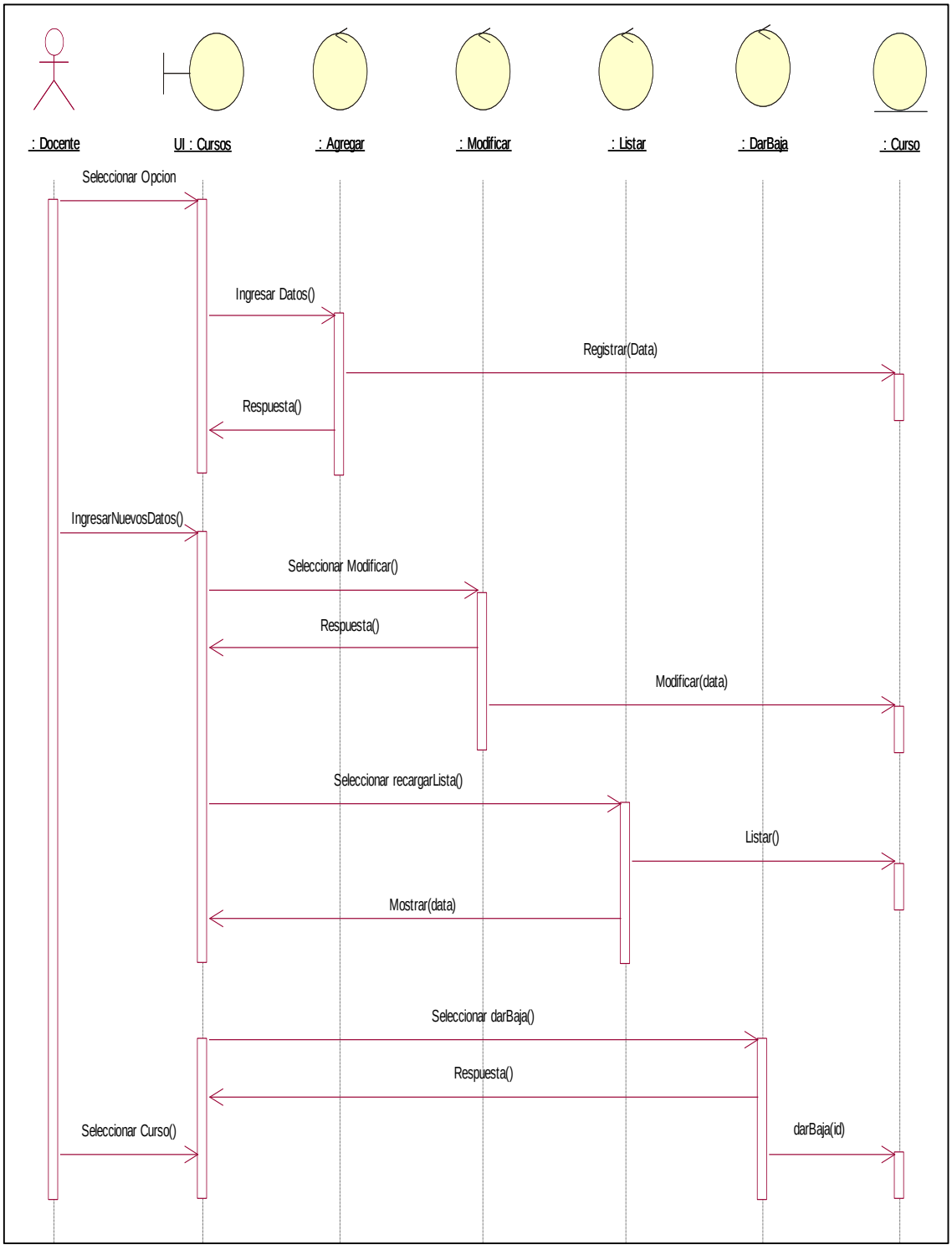


Figura 45: Diagrama de Secuencia – Gestión Curso Docente

Fuente: Elaboración Propia

d. Gestión Estudiante por parte del Docente

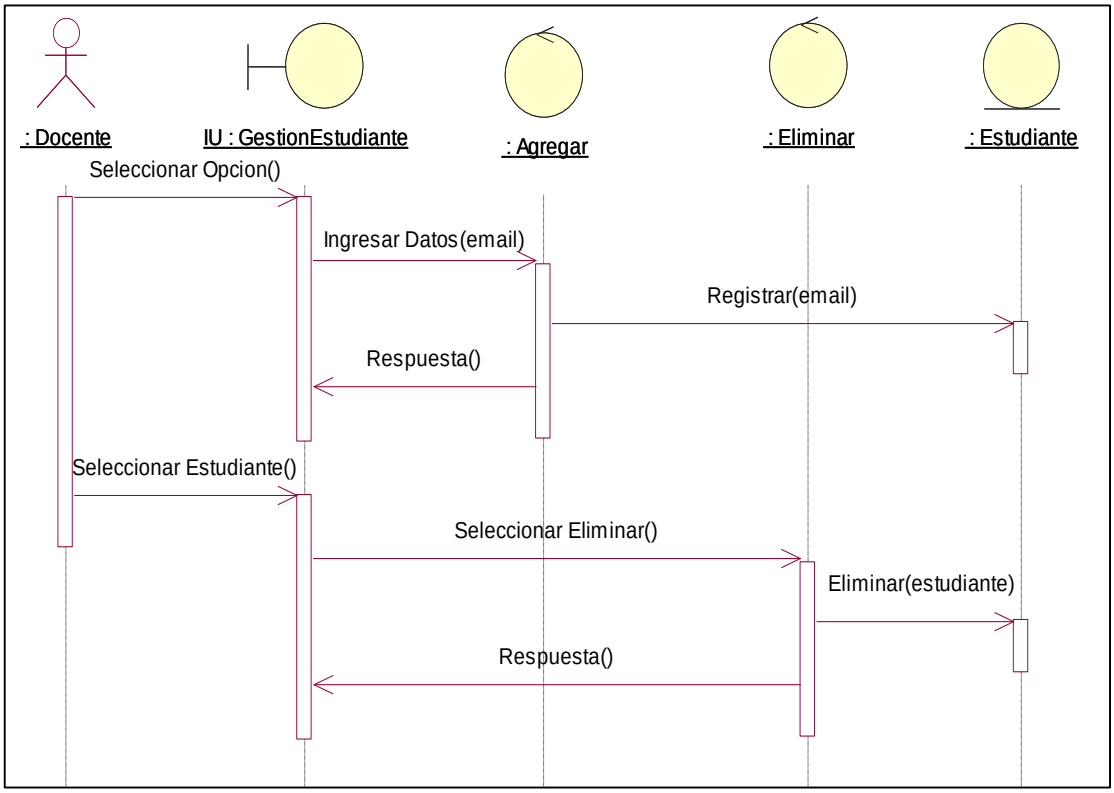


Figura 46: Diagrama de Secuencia – Gestión Estudiante Docente

Fuente: Elaboración Propia

e. Listar Estudiantes para el Docente

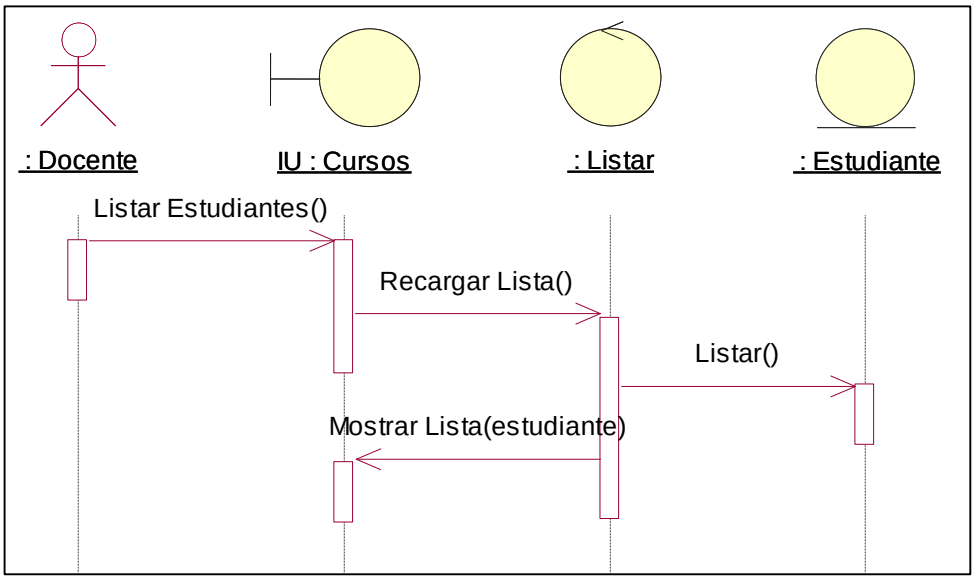


Figura 47: Diagrama de Secuencia – Listar Estudiantes

Fuente: Elaboración Propia

f. Listar Cursos para el Estudiante

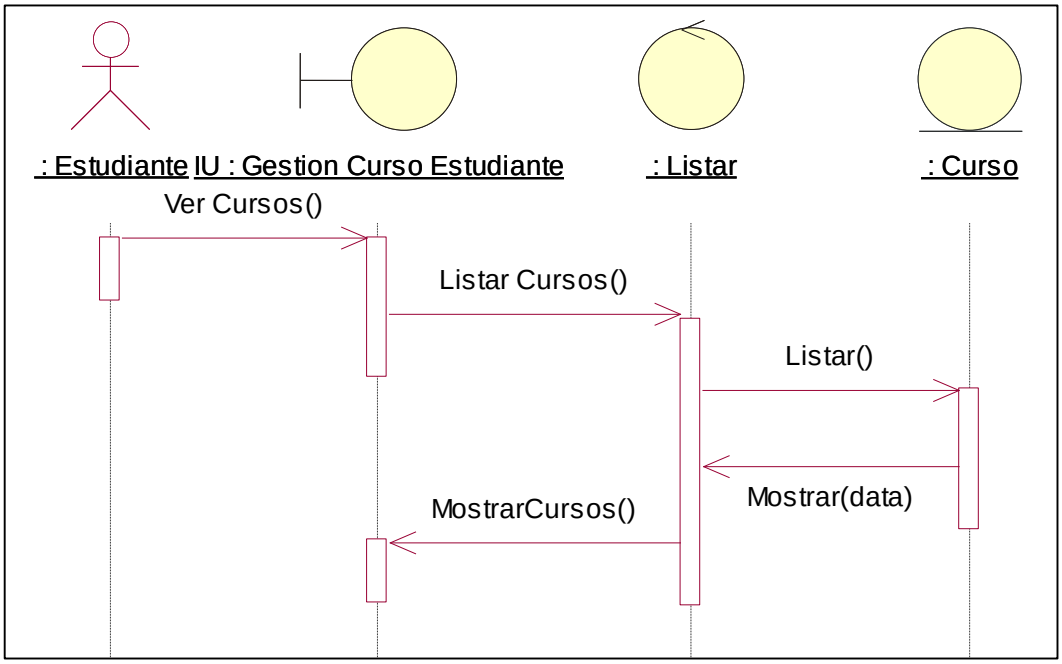


Figura 48: Diagrama de Secuencia – Listar Cursos Estudiante

Fuente: Elaboración Propia

g. Gestión Proyecto (Agregar-Listar)

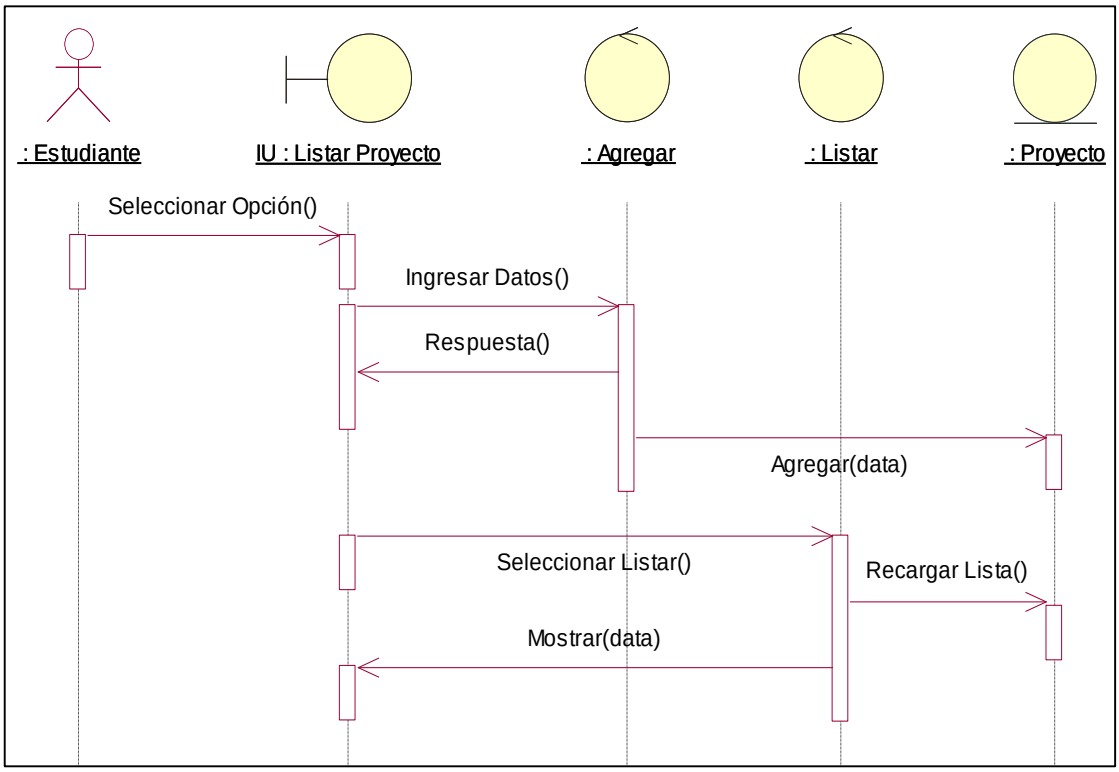


Figura 49: Diagrama de Secuencia – Gestión Proyecto

Fuente: Elaboración Propia

h. Registrar Tiempo en cada Fase

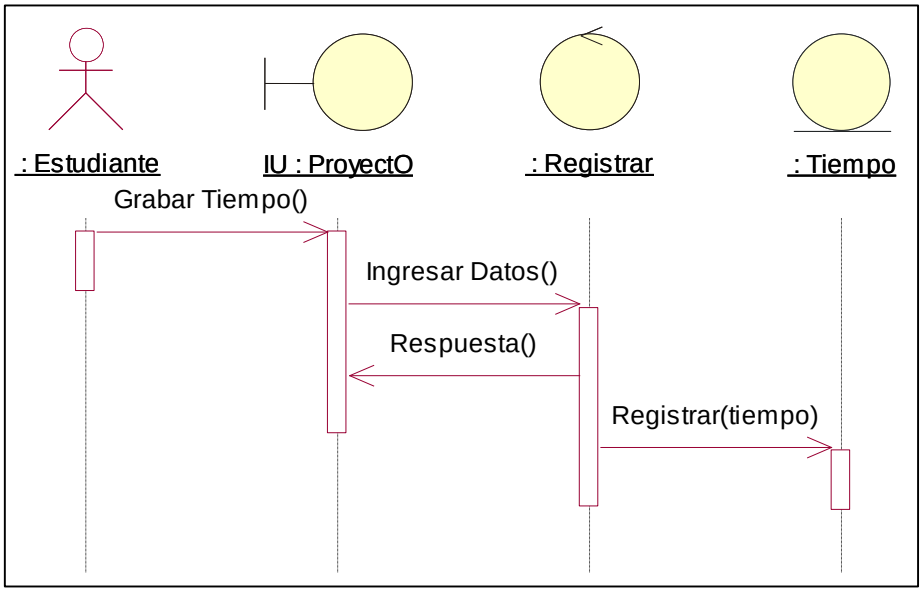


Figura 50: Diagrama de Secuencia – Registrar Tiempo

Fuente: Elaboración Propia

i. Gestión Bitácora de Tiempo (Listar-Modificar)

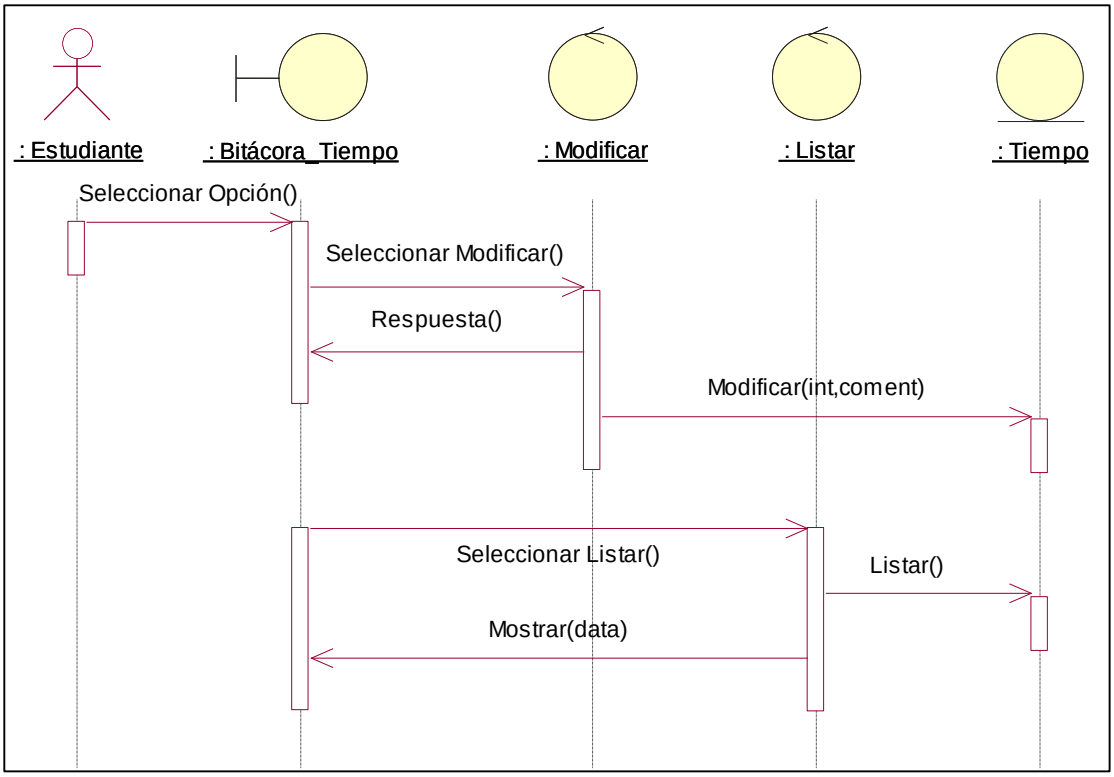


Figura 51: Diagrama de Secuencia – Gestión Bitácora de Tiempo

Fuente: Elaboración Propia

j. Gestión Defecto (Agregar - Listar)

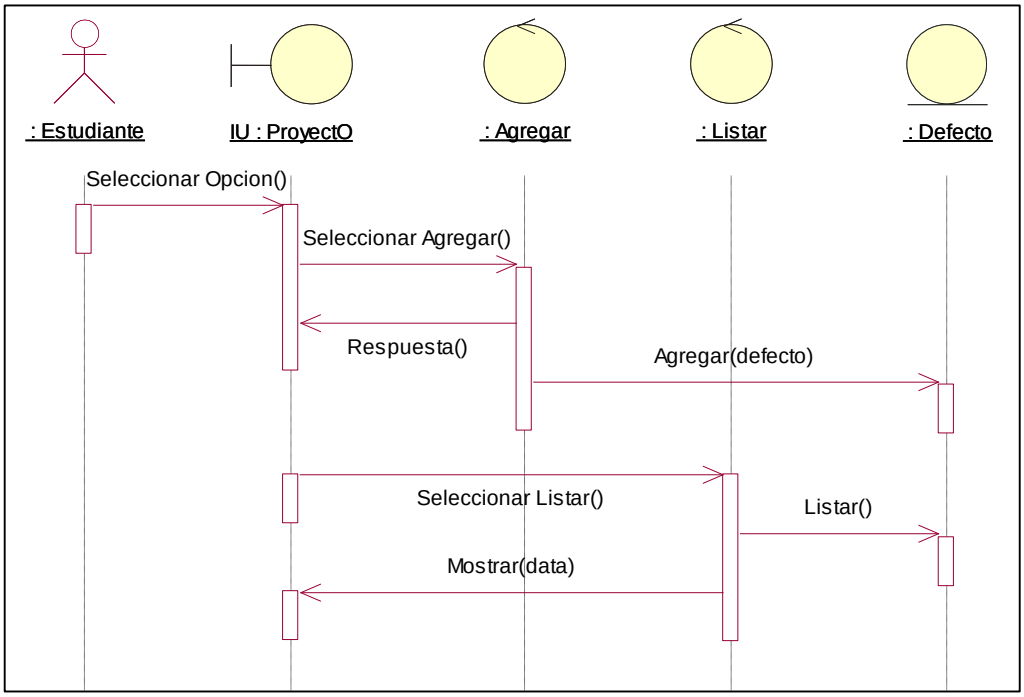


Figura 52: Diagrama de Secuencia – Gestión Defecto

Fuente: Elaboración Propia

k. Gestión Archivo (Descargar-Cargar Formato)

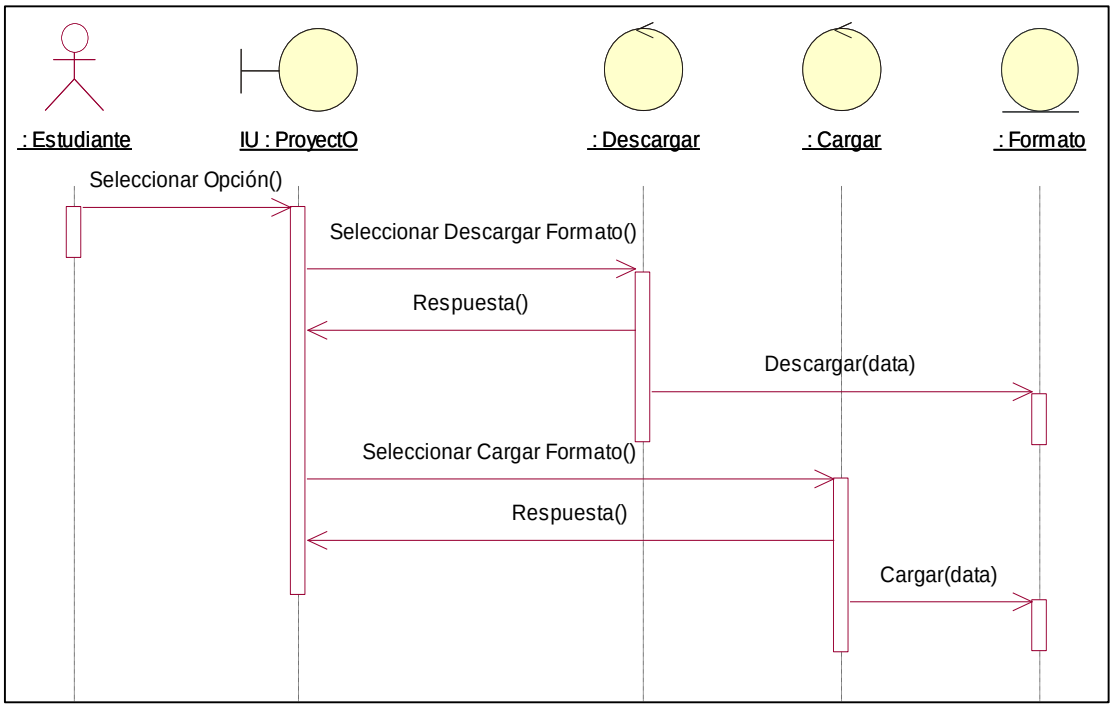


Figura 53: Diagrama de Secuencia – Gestión Archivo

Fuente: Elaboración Propia

l. Registrar LOC (Base –Agregada-Reusada)

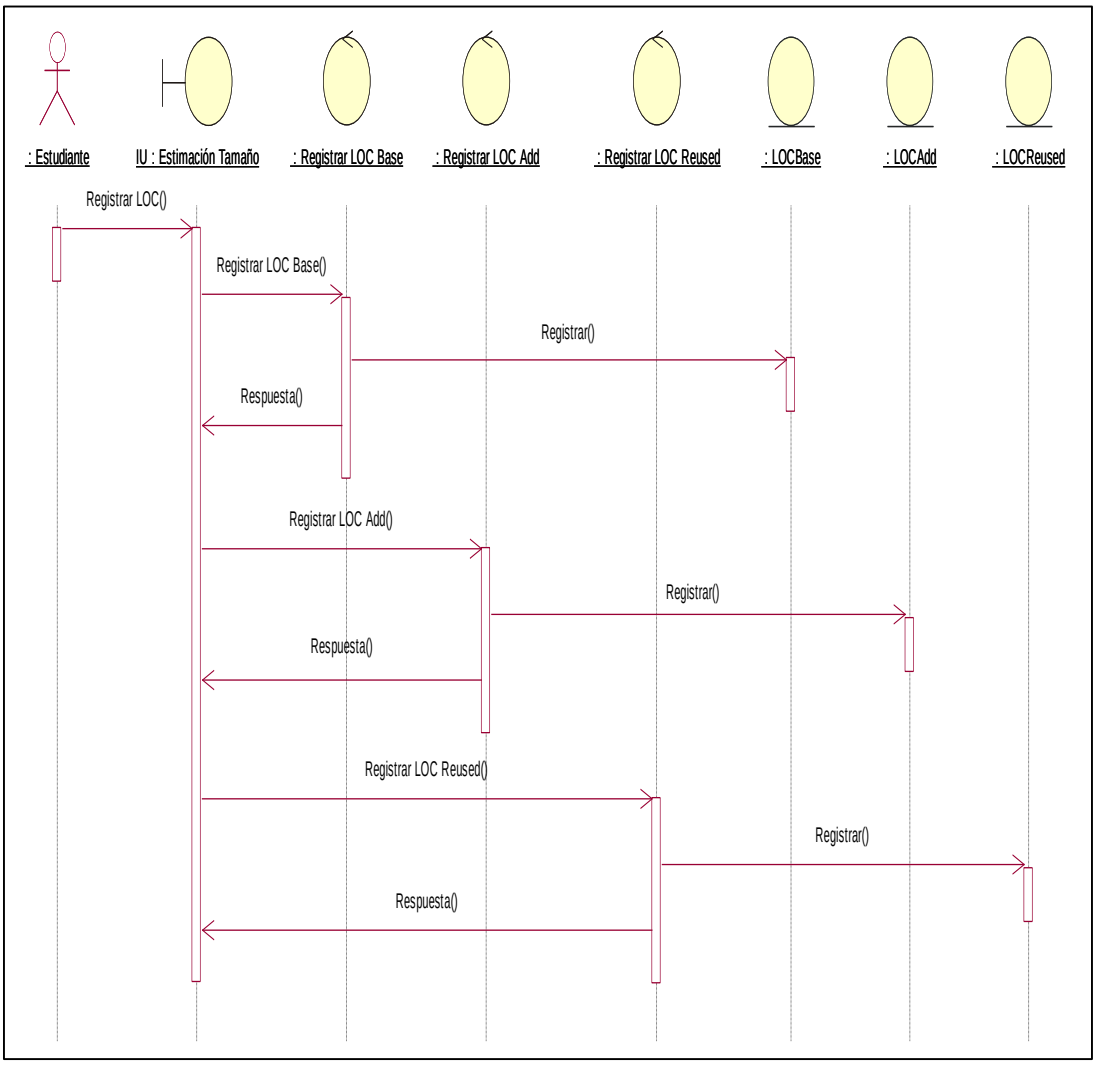


Figura 54: Diagrama de Secuencia – Registrar LOC

Fuente: Elaboración Propia

C. Modelo Lógico de Datos

Dado que se trabaja con un modelo de clases mapeadas a objetos java que sigue la especificación Java Data Objects (JDO), no se requiere implementar un modelo de datos dado que el contenedor no es tipo relacional sino de objetos que facilitan la persistencia de objetos Java en el Datastore (contenedor de objetos java), por lo que sólo se muestra las entidades creadas por el GAE durante la compilación de la aplicación.

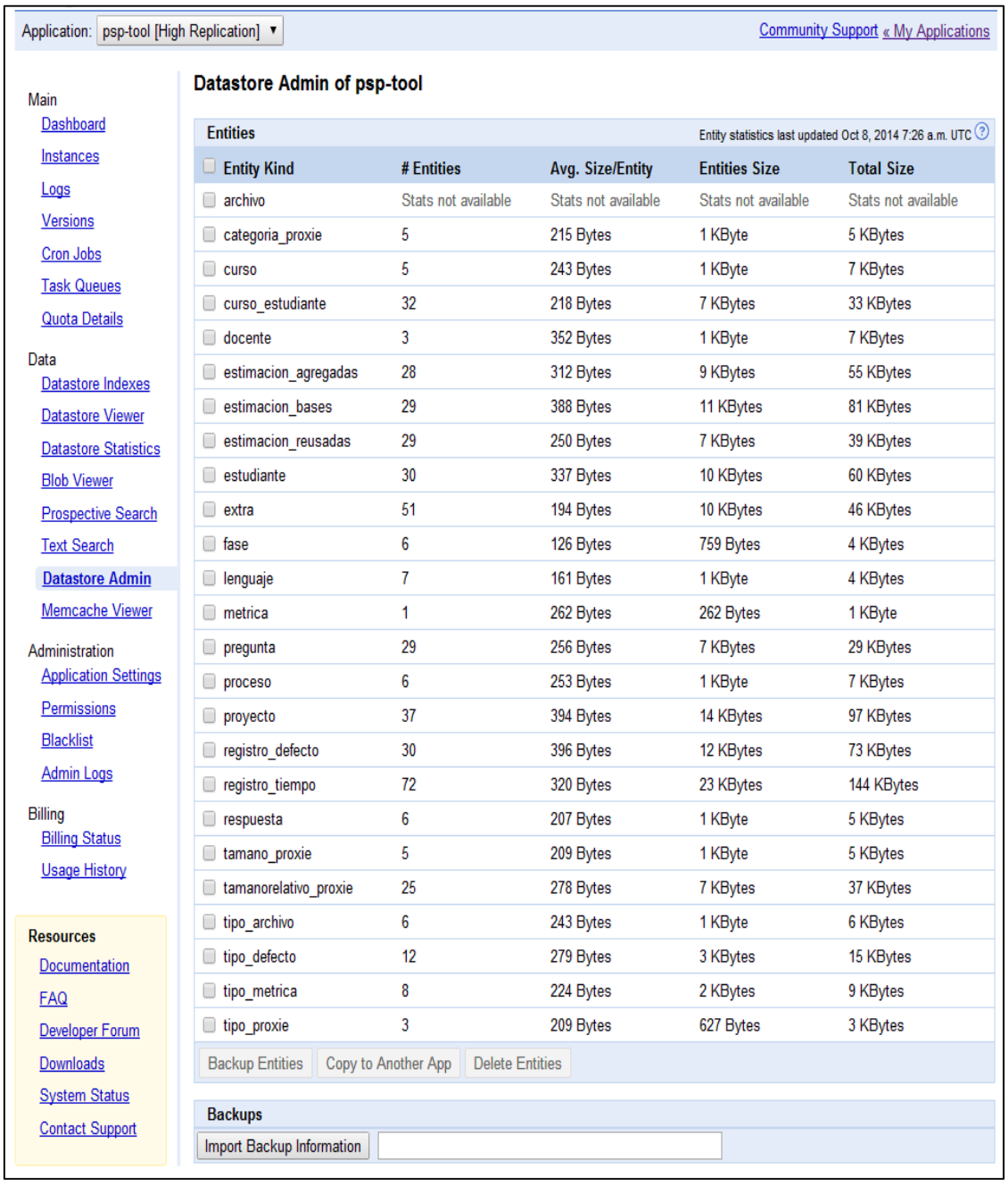


Figura 55: Entidades GAE

Fuente: Elaboración Propia

4.3. CODIFICACION

A. Diagrama de Componentes.

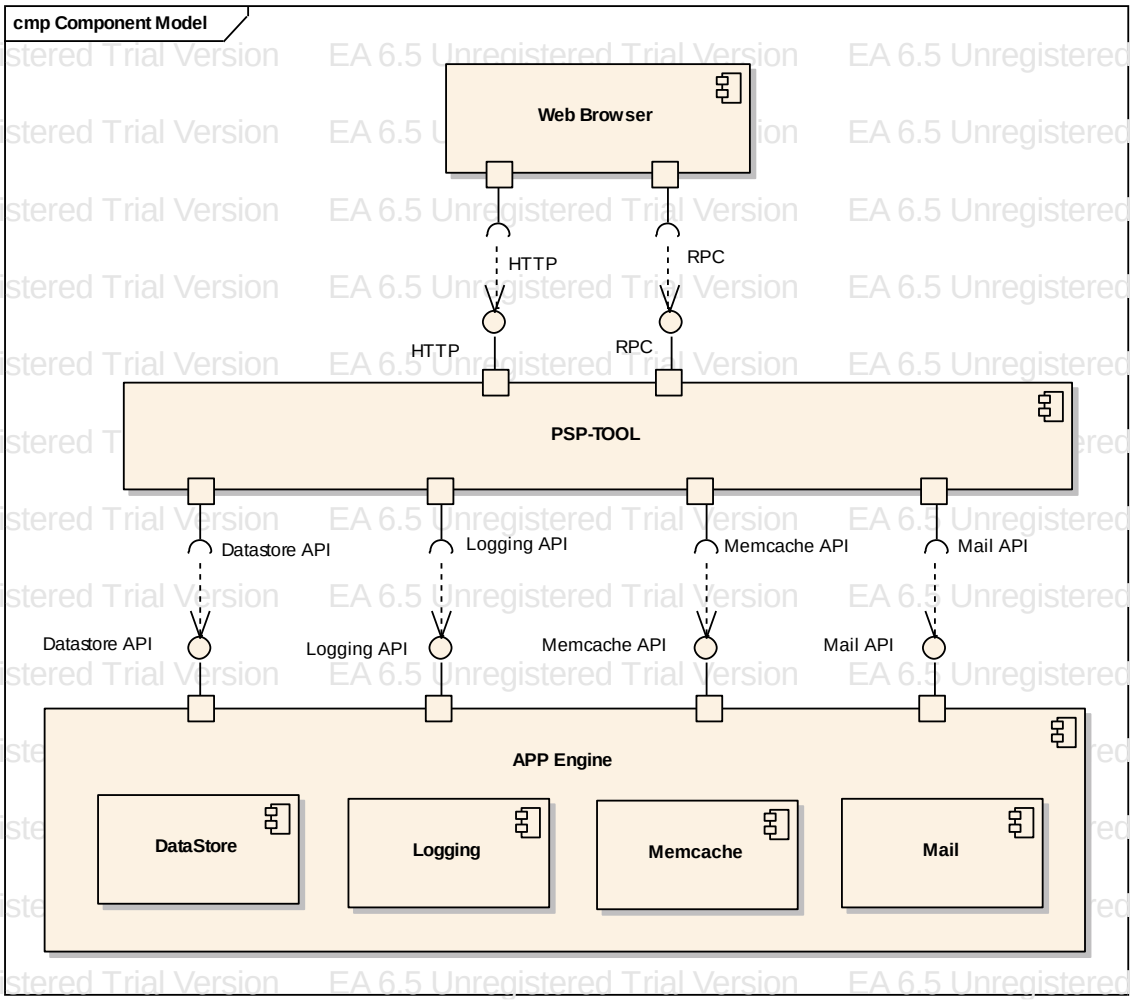


Figura 56: Diagrama de Componentes

Fuente: Elaboración Propia

B. Implementación

Para la implementación de esta herramienta cloud, se utilizó el IDE Netbeans 6.9, Java y el entorno cloud Google App Engine (GAE). Básicamente funciona como se detalla a continuación de acuerdo a que esta herramienta un Software como Servicio (SaaS):

En esta aplicación pueden acceder múltiples usuarios (estudiantes y/o docentes) de forma simultánea; dicha aplicación se encuentran instalada, configurada y

gestionada en Google App Engine de manera que es ejecutada en una sola instancia y ofrecida en multiacceso bajo demanda para todos los usuarios (estudiantes y/o docentes) que soliciten su uso. El software en la nube genera una arquitectura de software eliminando la necesidad de instalar y ejecutar la aplicación en el equipo del usuario final, eliminando la carga del mantenimiento del software y el soporte técnico. De manera más técnica, la Arquitectura se divide en tres capas: las interfaces de usuario, los controladores (Servlets), y las clases (que vienen a ser el datastore básicamente); pero además hay un "controlador maestro", que contiene los métodos (registrar, actualizar, etc.) de las clase, y dichas clases se quedan sólo con setter's y getter's.

A continuación se muestra el código de los Requerimientos principales.

- Clases implementadas según los requerimientos

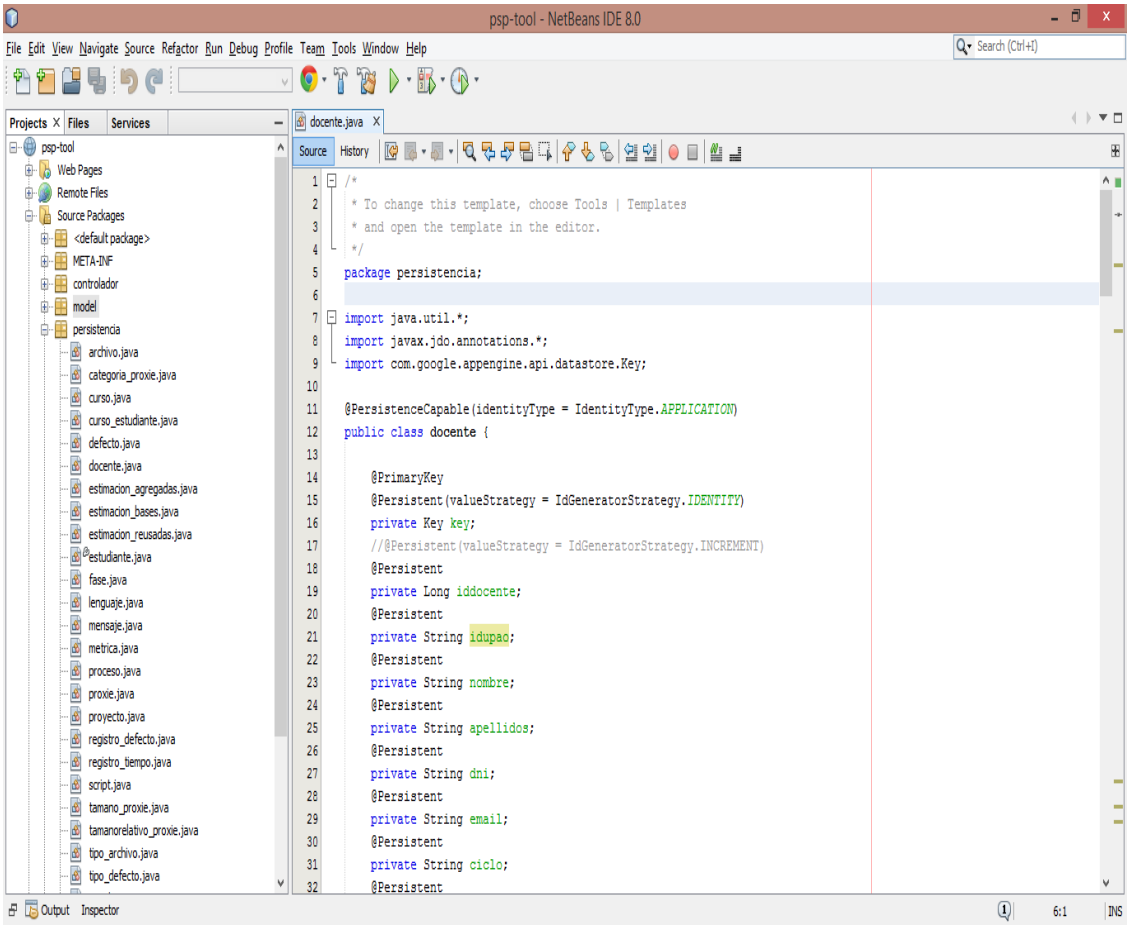


Figura 57: Implementación – Clases

Fuente: Elaboración Propia

- Controlador principal, dónde se maneja todas las acciones según el PSP

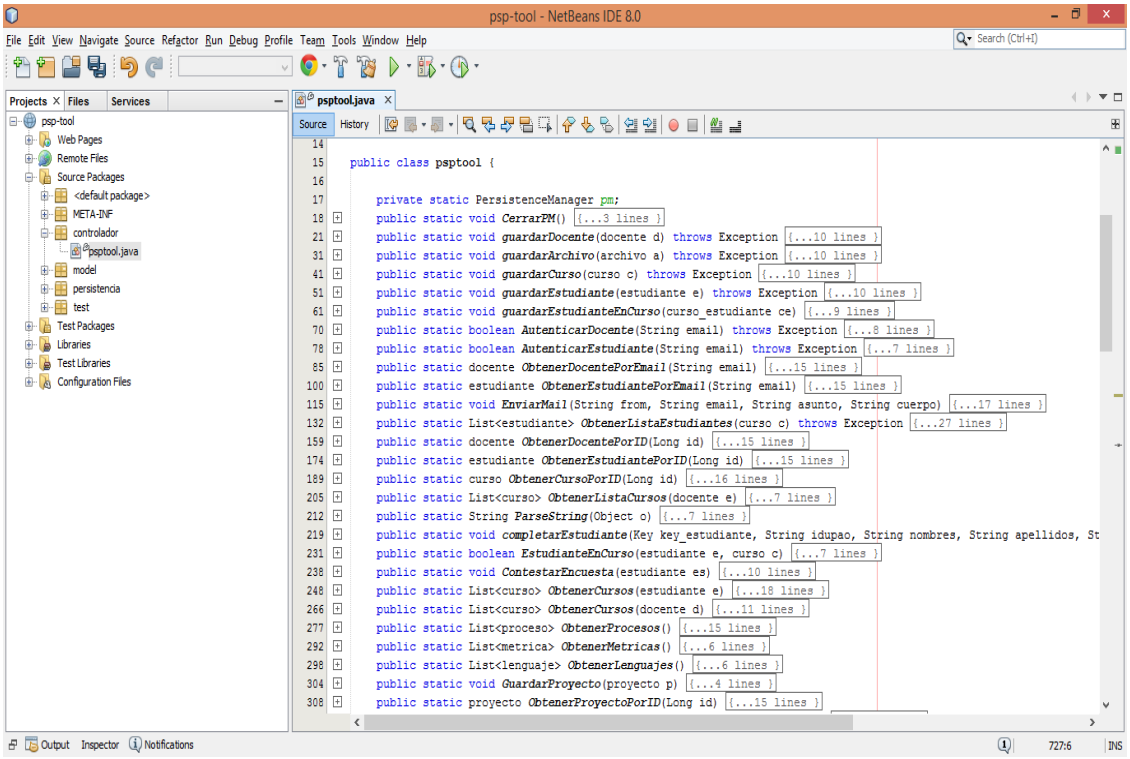


Figura 58: Implementación – Controlador Principal Parte 1

Fuente: Elaboración Propia

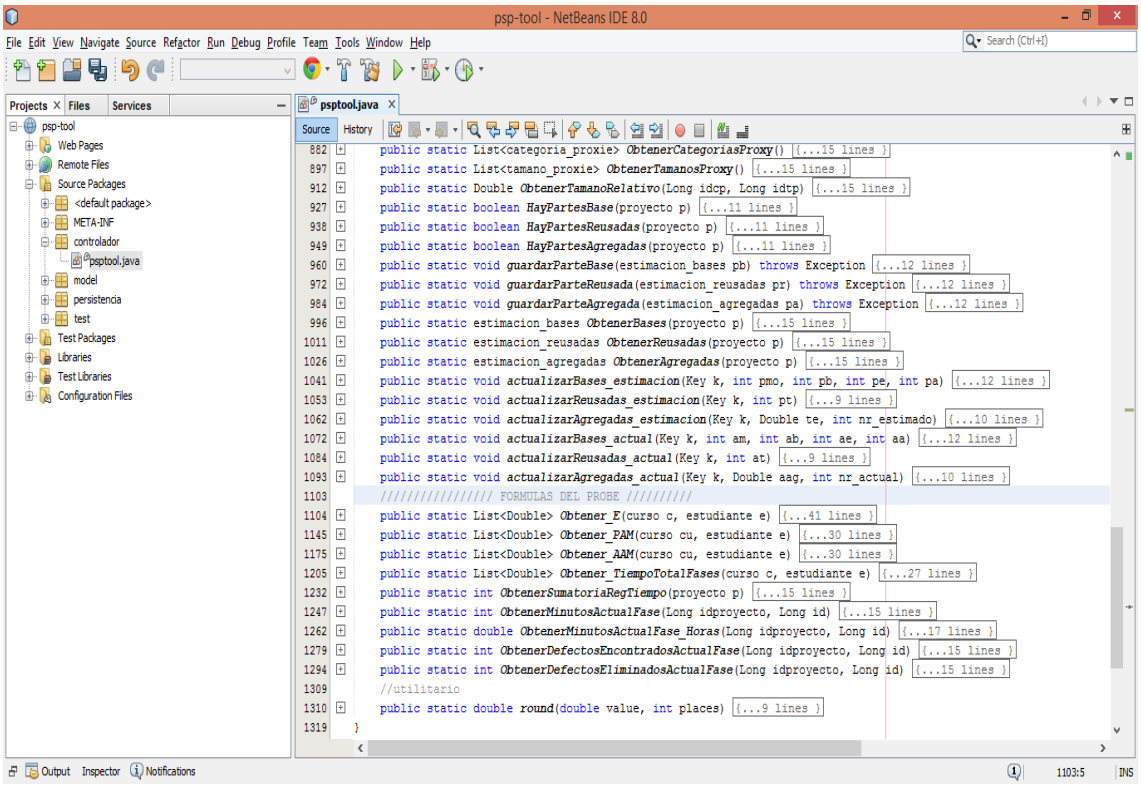


Figura 59: Implementación – Controlador Principal Parte 2

Fuente: Elaboración Propia

- Lista de las pantallas en JSP

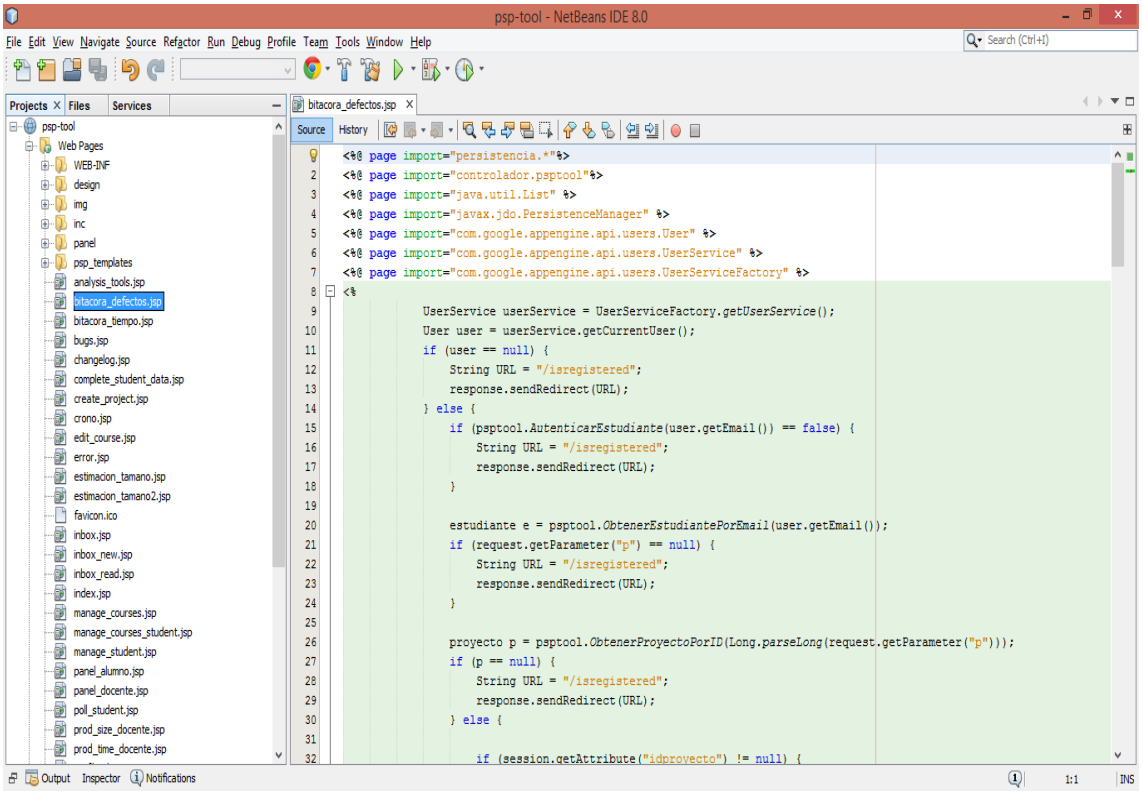


Figura 60: Implementación – Lista de pantallas en JSP

Fuente: Elaboración Propia

4.4. PRUEBAS

Gestión Docente:

i. Ingresar a la aplicación: psp-tool.appspot.com

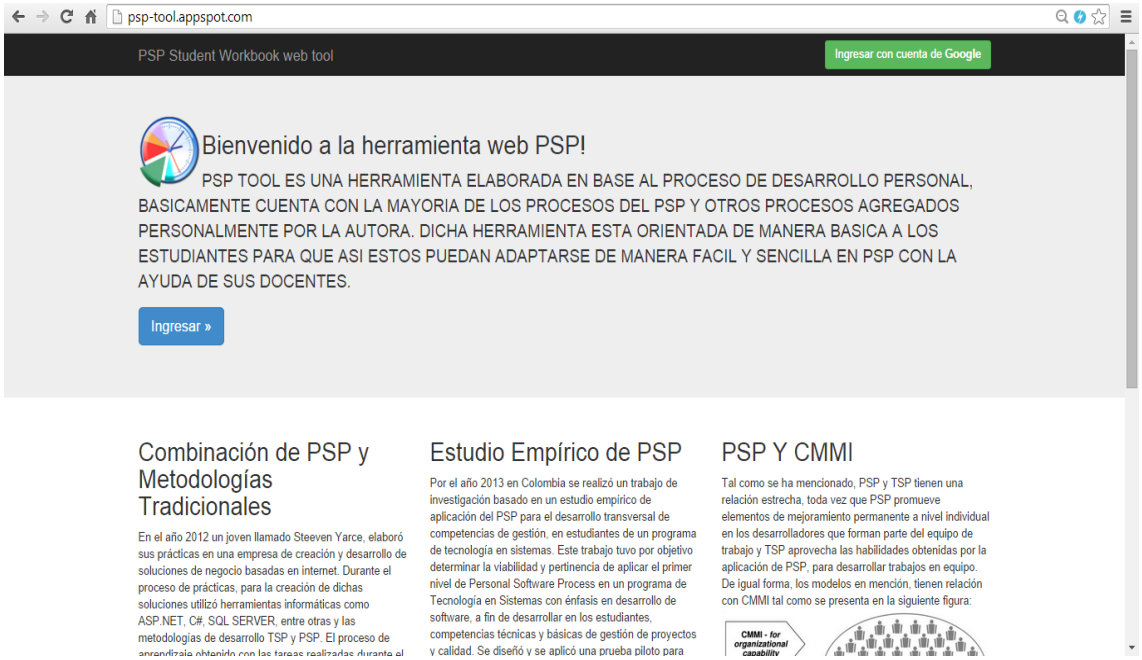


Figura 61: Pruebas – Index

Fuente: Elaboración Propia

ii. Si ha ingresado como Docente, se agrega el curso a registrar

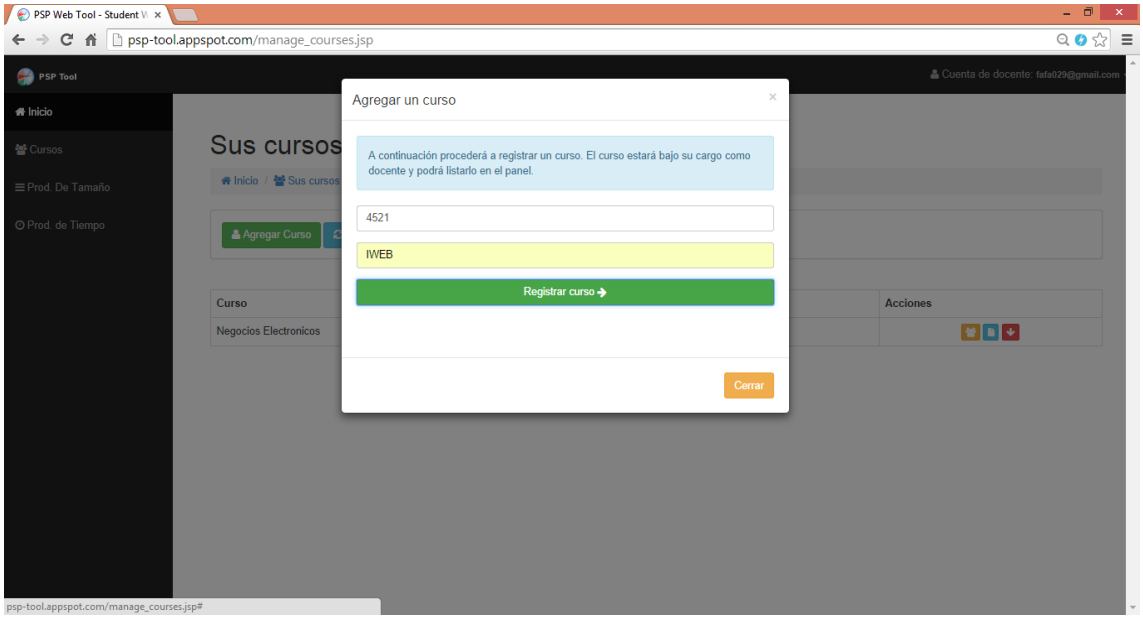


Figura 62: Pruebas – Agregar Curso

Fuente: Elaboración Propia

iii. Se lista los alumnos registrados de ese curso

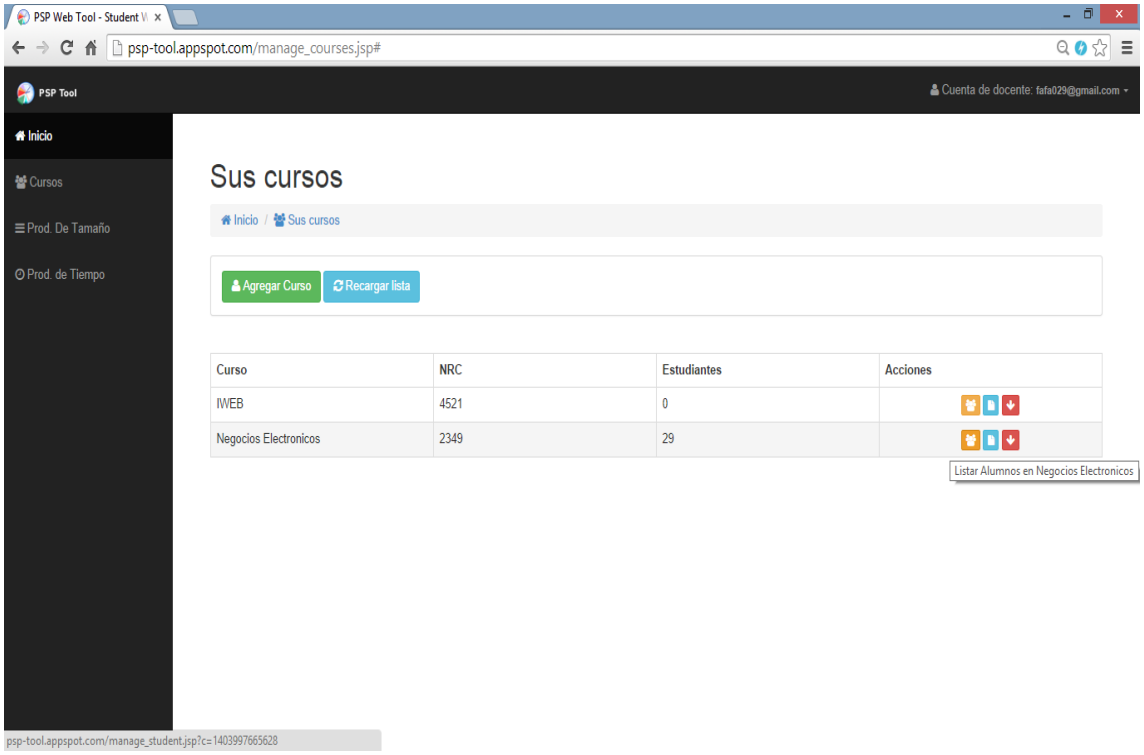


Figura 63: Pruebas – Listar Alumnos

Fuente: Elaboración Propia

iv. De no tener ningún estudiante, se procede a agregar un estudiante mediante la cuenta de google.

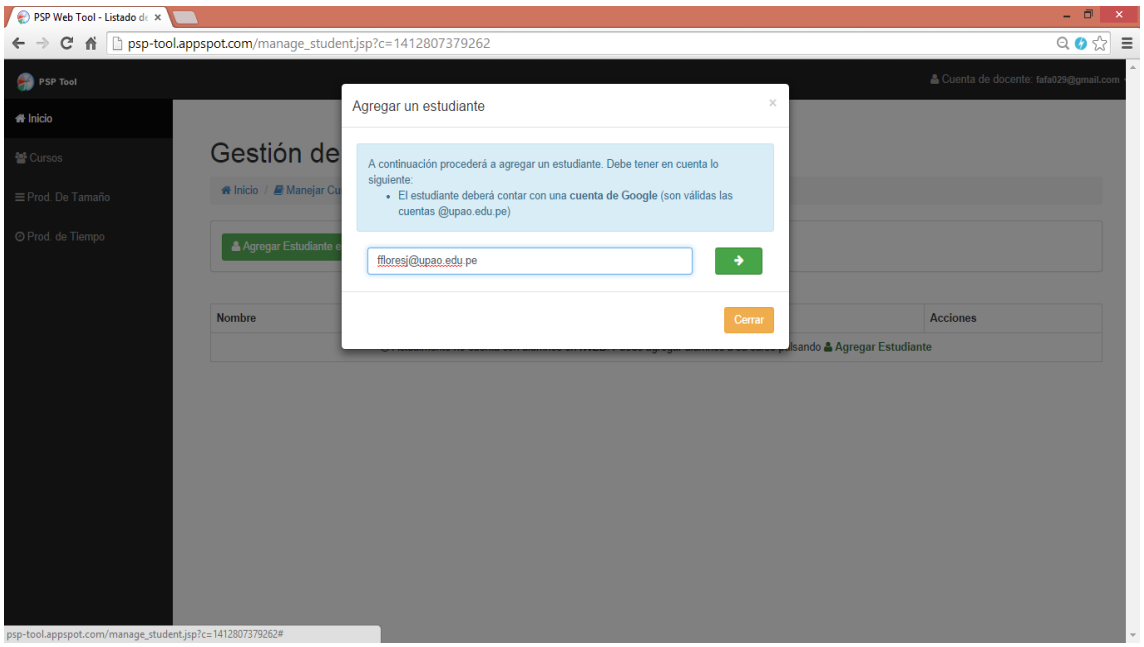


Figura 64: Pruebas – Agregar Estudiante

Fuente: Elaboración Propia

v. Seleccionar Recargar Lista para ver los alumnos agregados recientemente

Figura 65: Pruebas – Recargar Lista

Fuente: Elaboración Propia

Gestión Estudiante

vi. Al estudiante le llega un mensaje de registro, en caso acepte debe acceder mediante el Link el cual lo conducirá a la aplicación

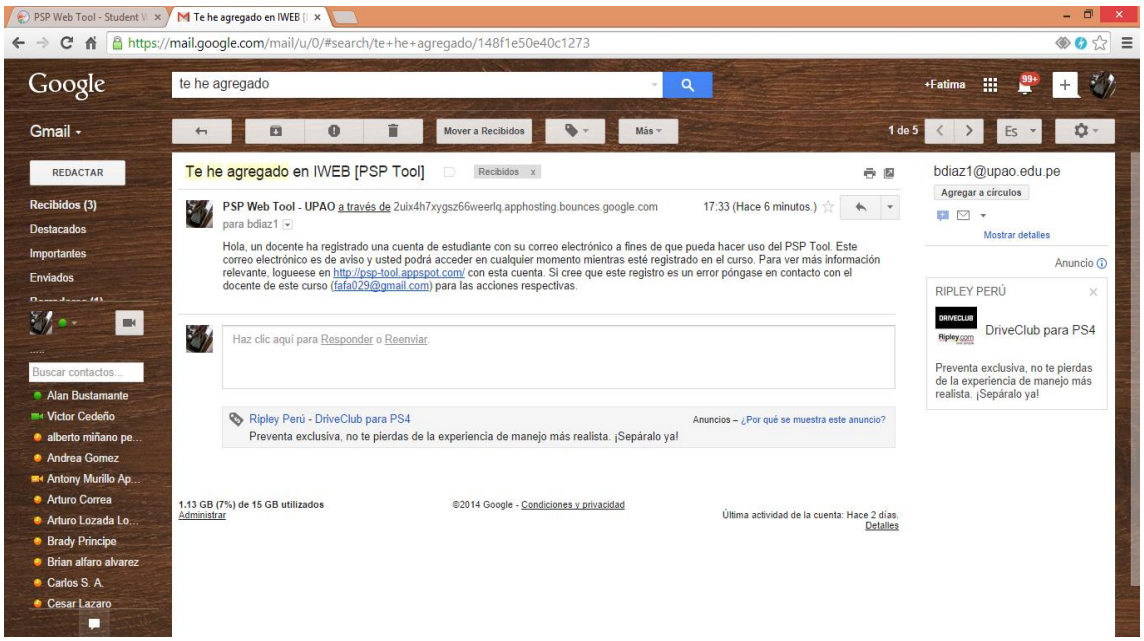


Figura 66: Pruebas – Mensaje de Confirmación

Fuente: Elaboración Propia

vii. Y automáticamente ingresa al panel de Estudiante

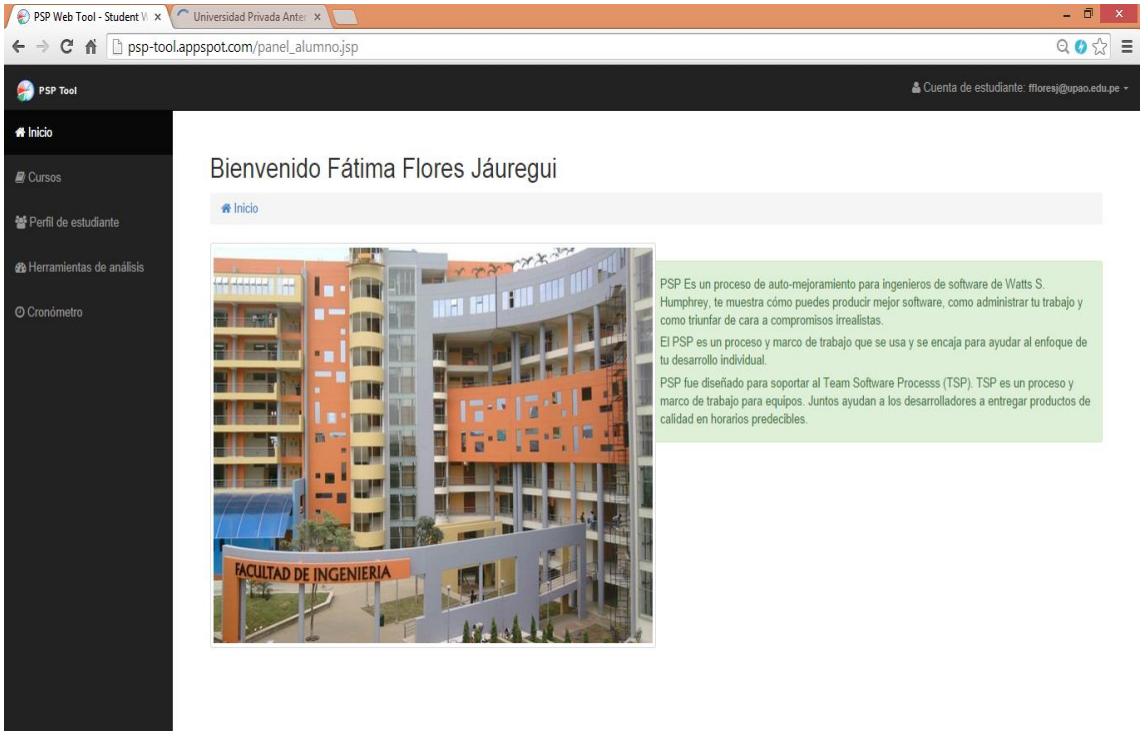


Figura 67: Pruebas – Panel Estudiante

Fuente: Elaboración Propia

viii. El estudiante puede listar los cursos en los cuales esta registrado

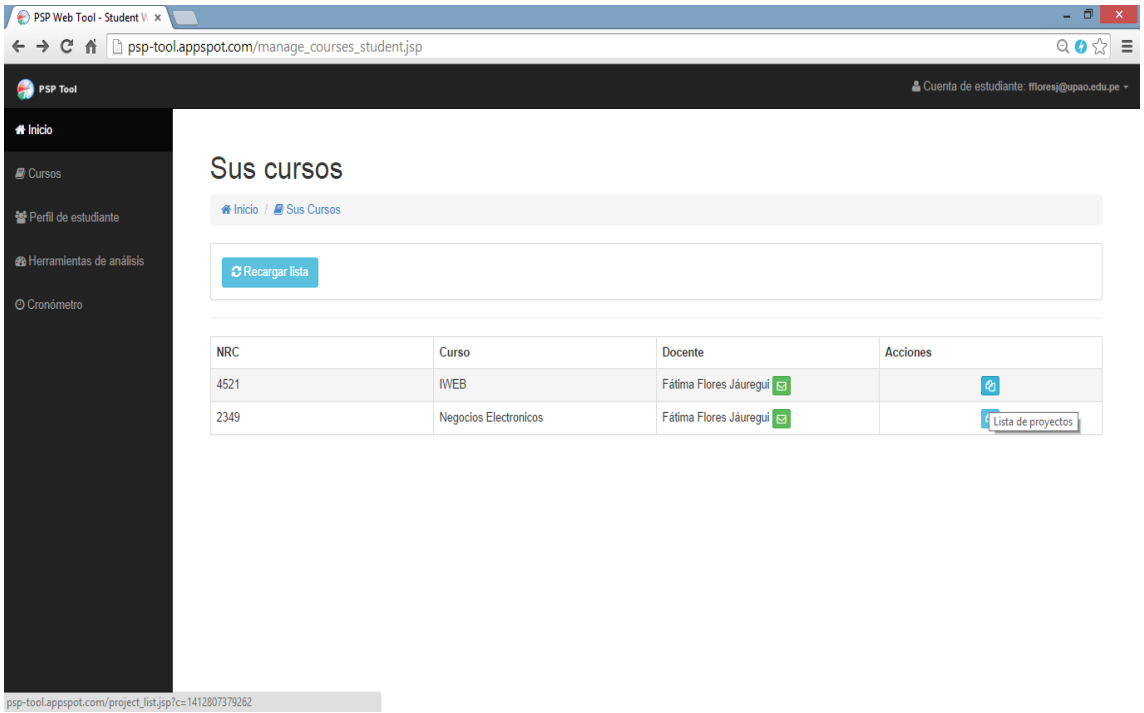


Figura 68: Pruebas – Cursos del Estudiante

Fuente: Elaboración Propia

ix. Se procede a crear un proyecto dentro del curso agregado

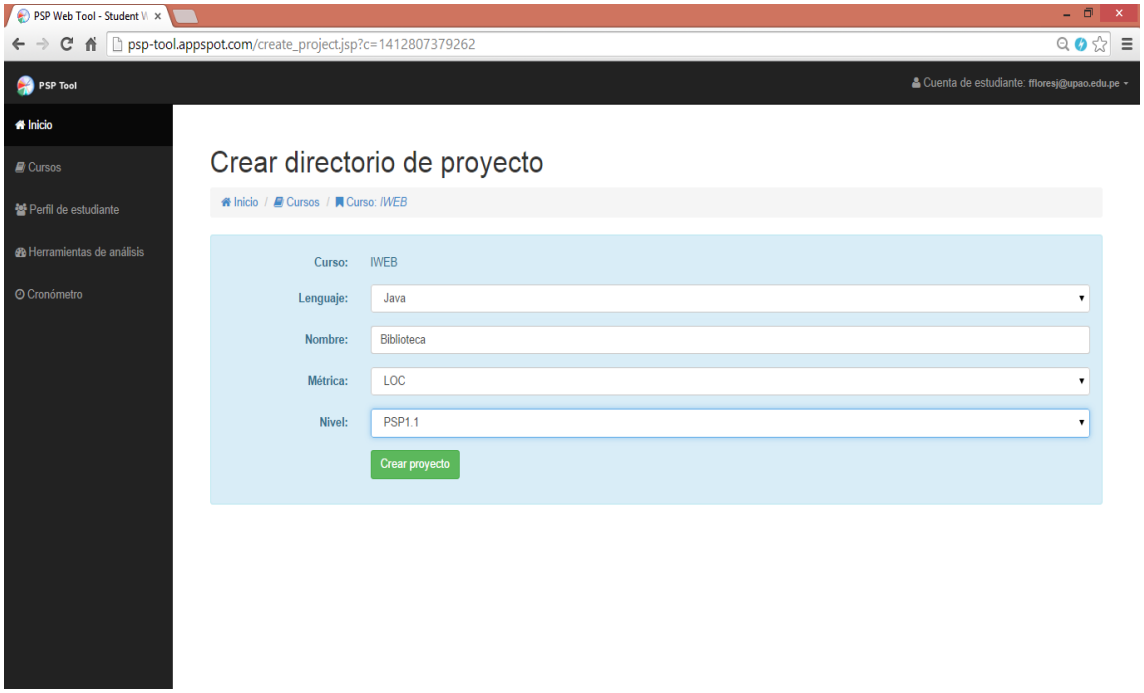


Figura 69: Pruebas – Crear Proyecto

Fuente: Elaboración Propia

x. Inicia el proyecto, al momento de elegir dicha opción automáticamente la Fecha de inicio cambia.

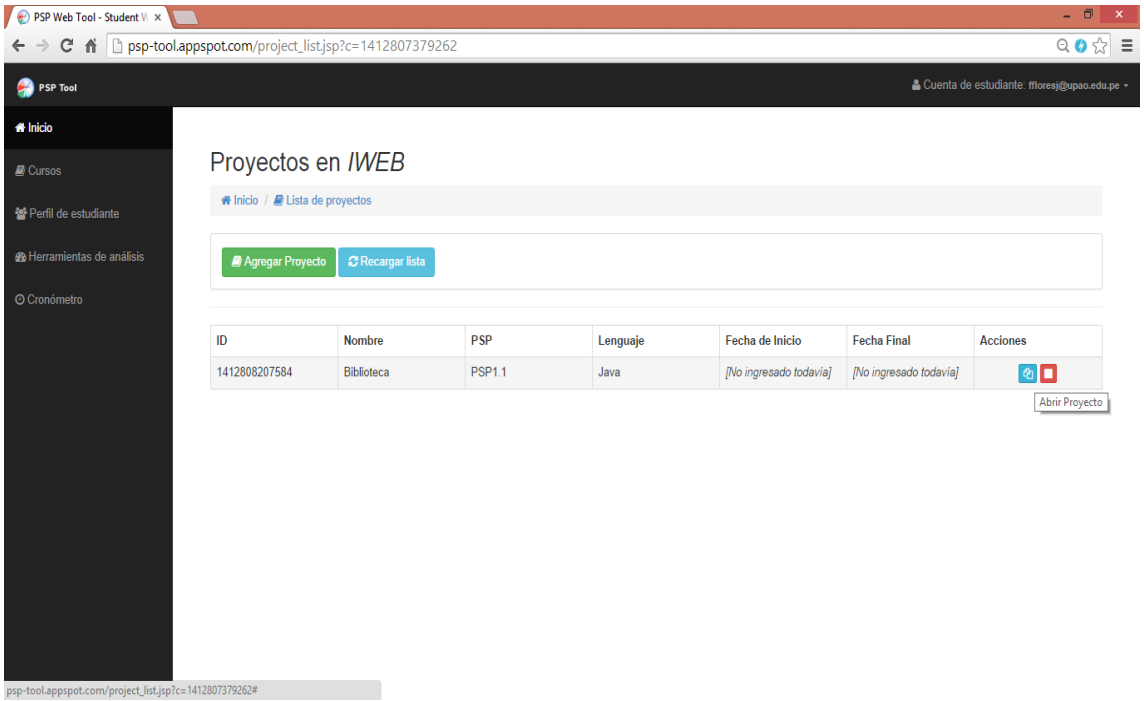


Figura 70: Pruebas – Gestión Proyecto

Fuente: Elaboración Propia

xi. Elegir la fase en que se encuentra durante el proyecto y luego registrar el tiempo en el botón play

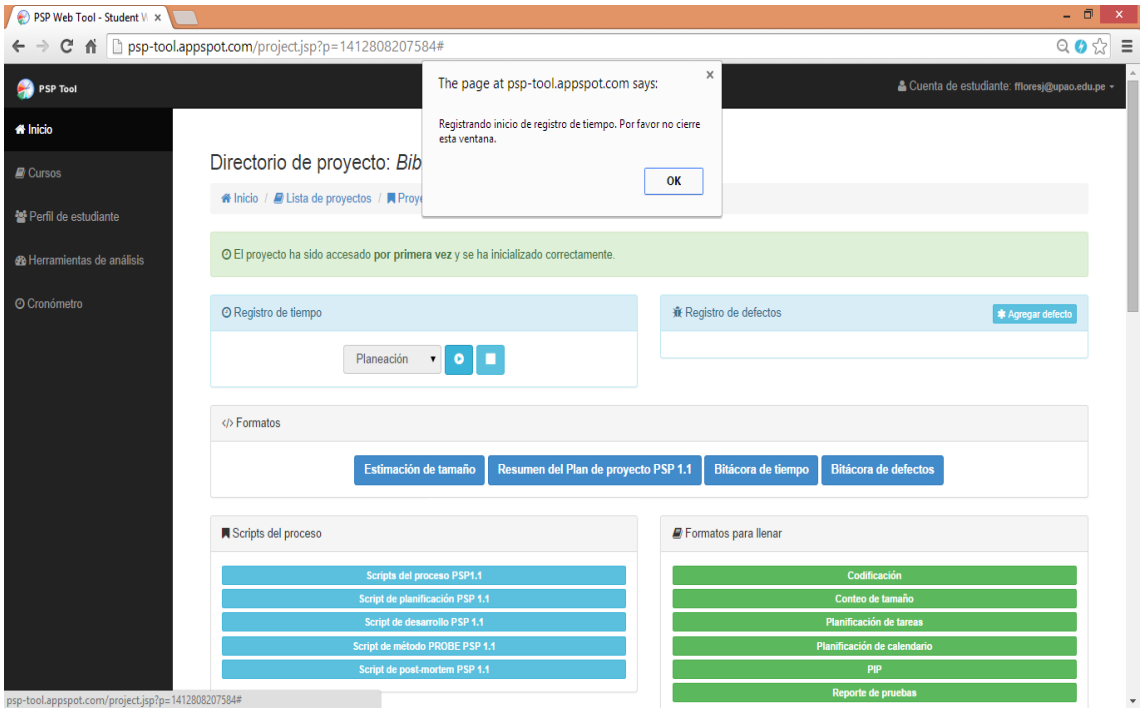


Figura 71: Pruebas – Registrar Tiempo

Fuente: Elaboración Propia

xii. Si se ha encontrado un defecto, registrarlo en el Panel de Registro de Defectos

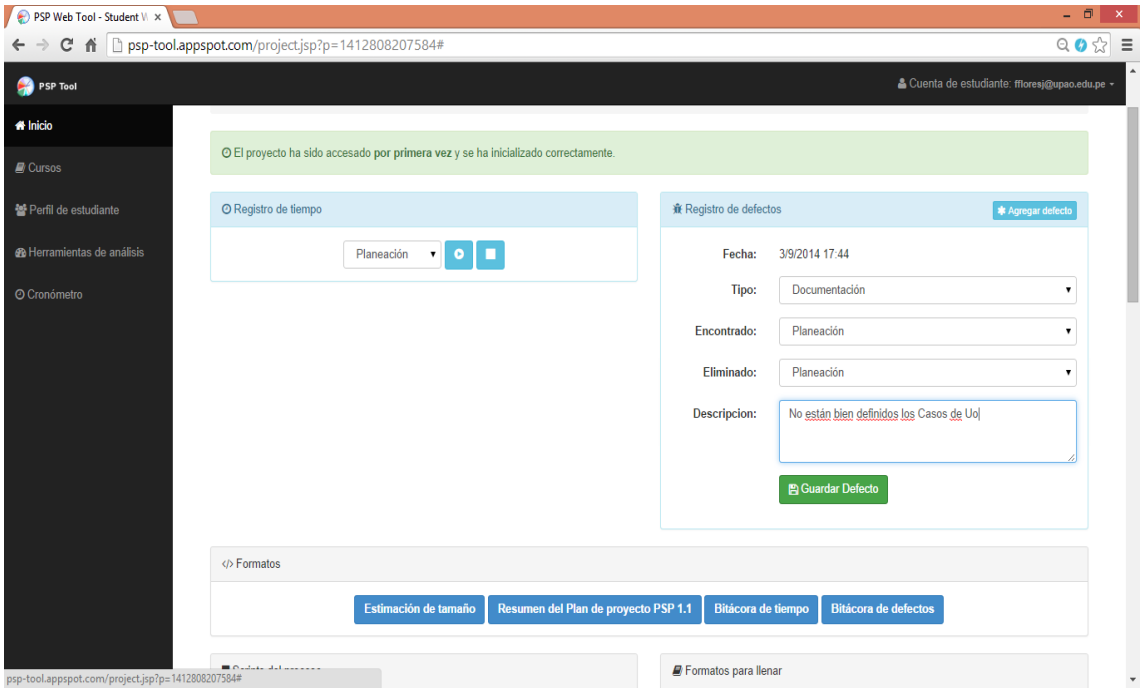


Figura 72: Pruebas – Registrar Defecto

Fuente: Elaboración Propia

xiii. En la fase de Planificación, elegir la opción Estimación de Tamaño, para luego registrar las partes base planeadas para calcular el Método PROBE

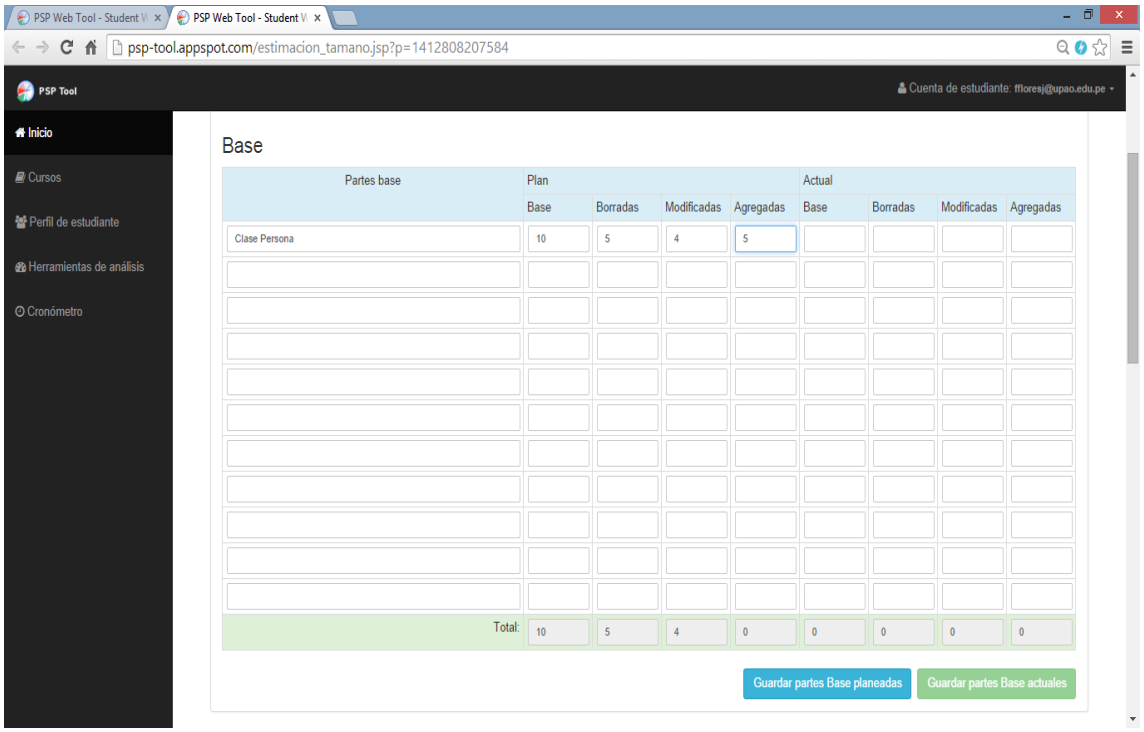


Figura 73: Pruebas – Registrar Partes Base

Fuente: Elaboración Propia

xiv. Registrar las partes agregadas planeadas para calcular el Método PROBE

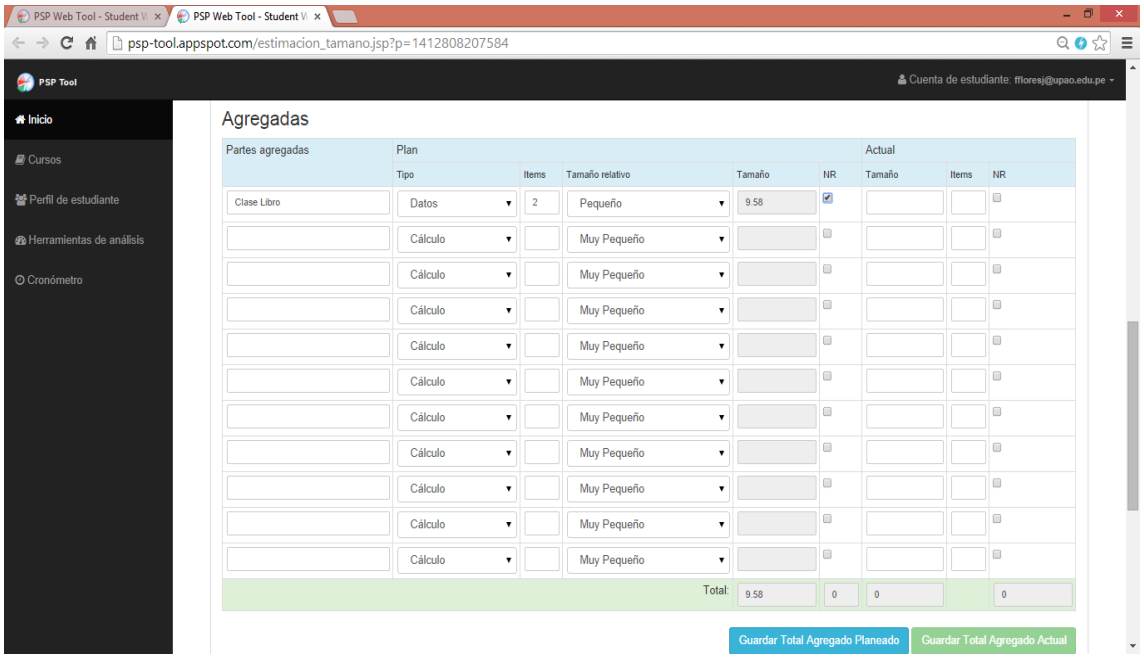


Figura 74: Pruebas – Registrar Partes Agregadas

Fuente: Elaboración Propia

xv. Automáticamente se calcular los tiempos estimados, previo a ello se debe registrar el tiempo estimado y el tamaño proyectado

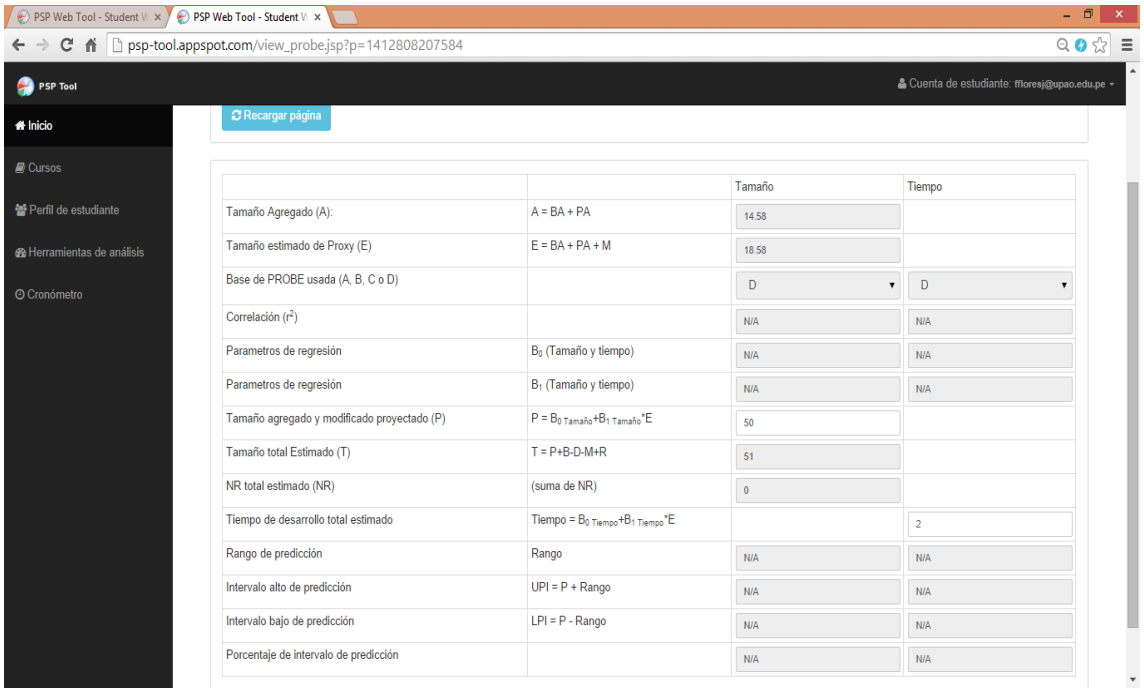


Figura 75: Pruebas – Método PROBE

Fuente: Elaboración Propia

xvi. Ver la bitácora de tiempo

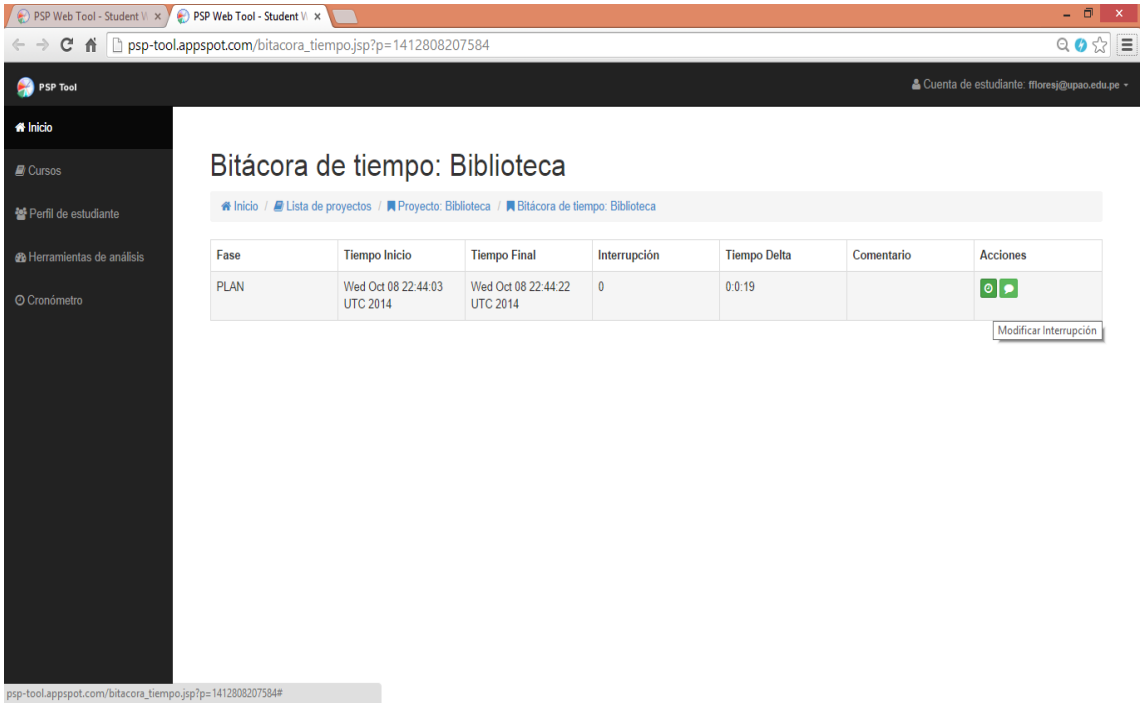


Figura 76: Pruebas – Bitácora de Tiempo

Fuente: Elaboración Propia

xvii. Ver la bitácora de defectos

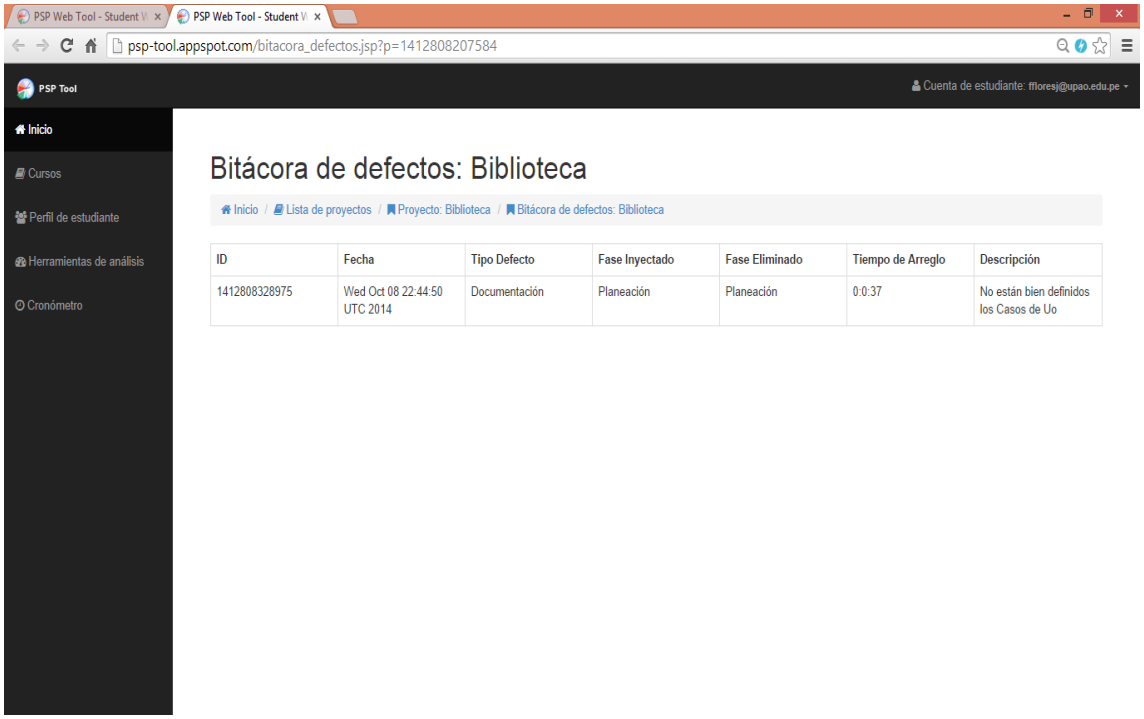


Figura 77: Pruebas – Bitácora de Defectos

Fuente: Elaboración Propia

xviii. En caso se tenga que usar un formato, descargarlo del panel "Formatos para LLenar"

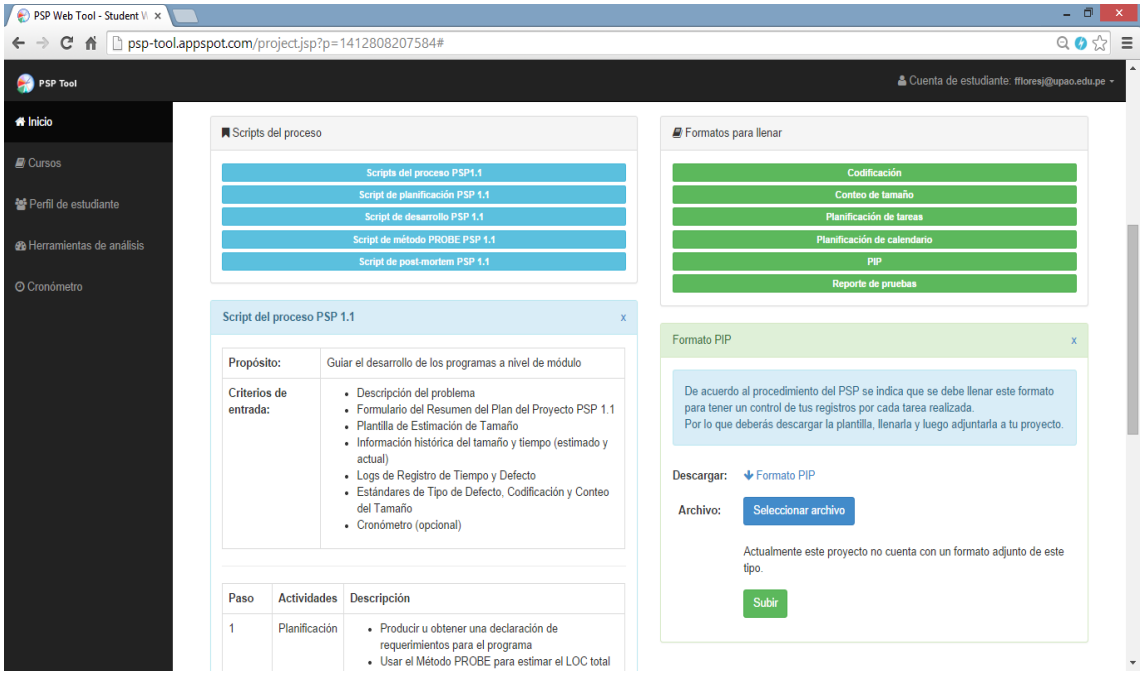


Figura 78: Pruebas – Descargar Formato

Fuente: Elaboración Propia

xix. Ya actualizado el formato, subirlo en la aplicación

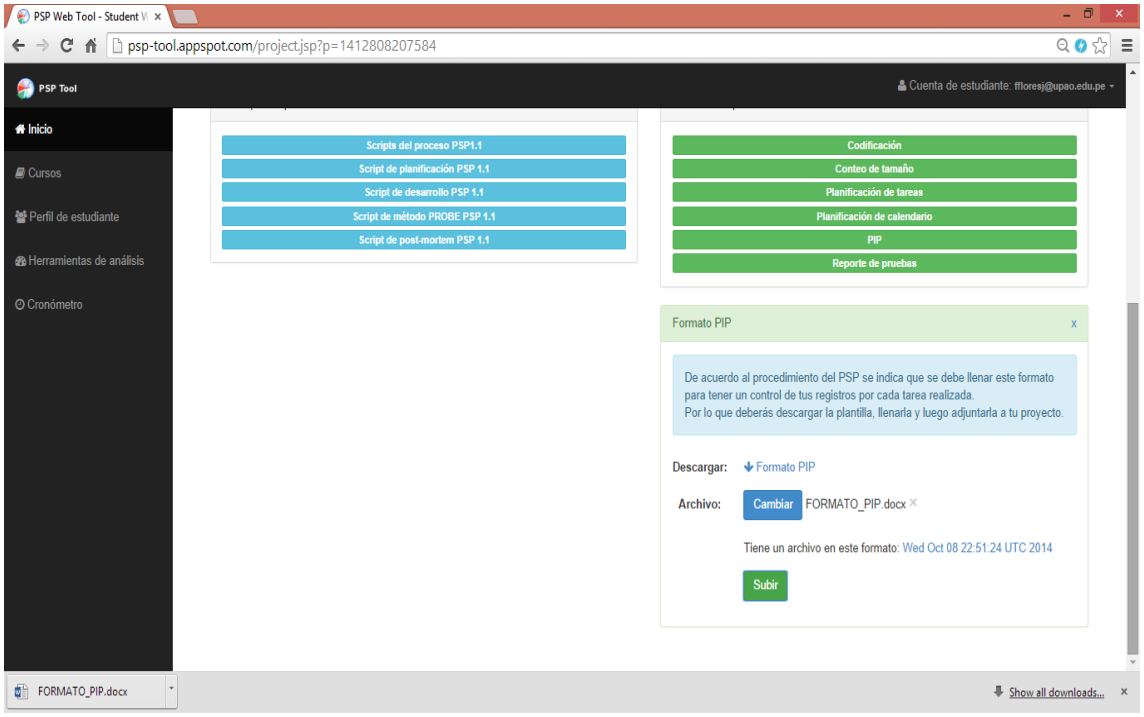


Figura 79: Pruebas – Cargar Formato

Fuente: Elaboración Propia

xx. Al finalizar el proceso de desarrollo, la aplicación muestra un resumen del proyecto

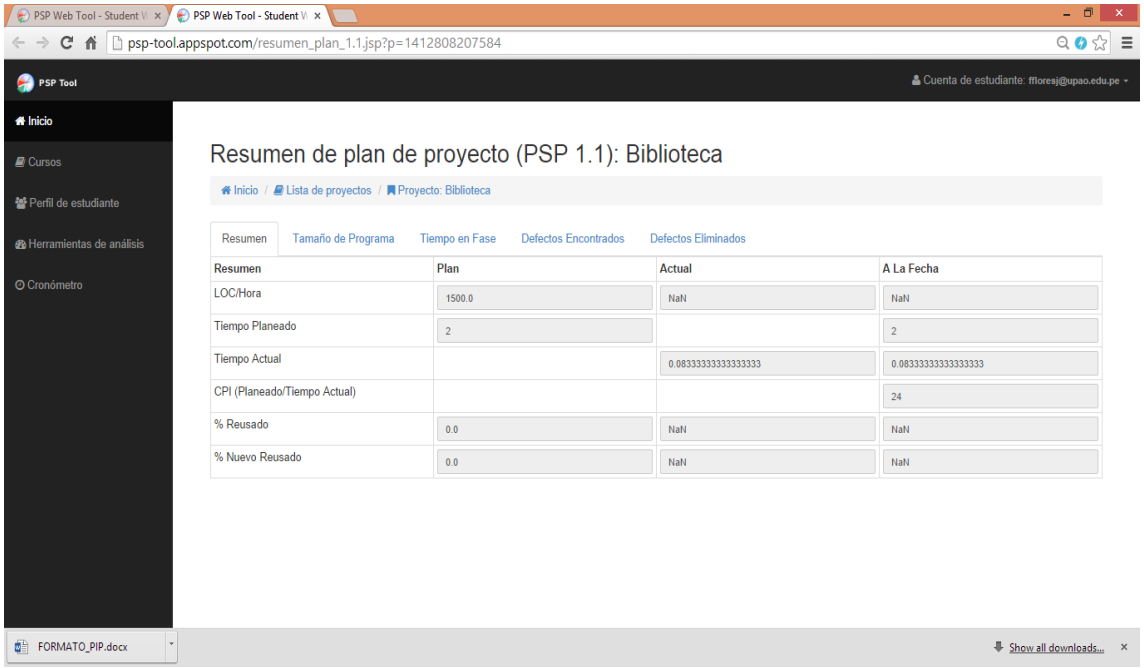


Figura 80: Pruebas – Resumen del Plan de Proyecto

Fuente: Elaboración Propia

xxi. A más detalle de las estimaciones, ir a la pestaña de Herramientas de Análisis

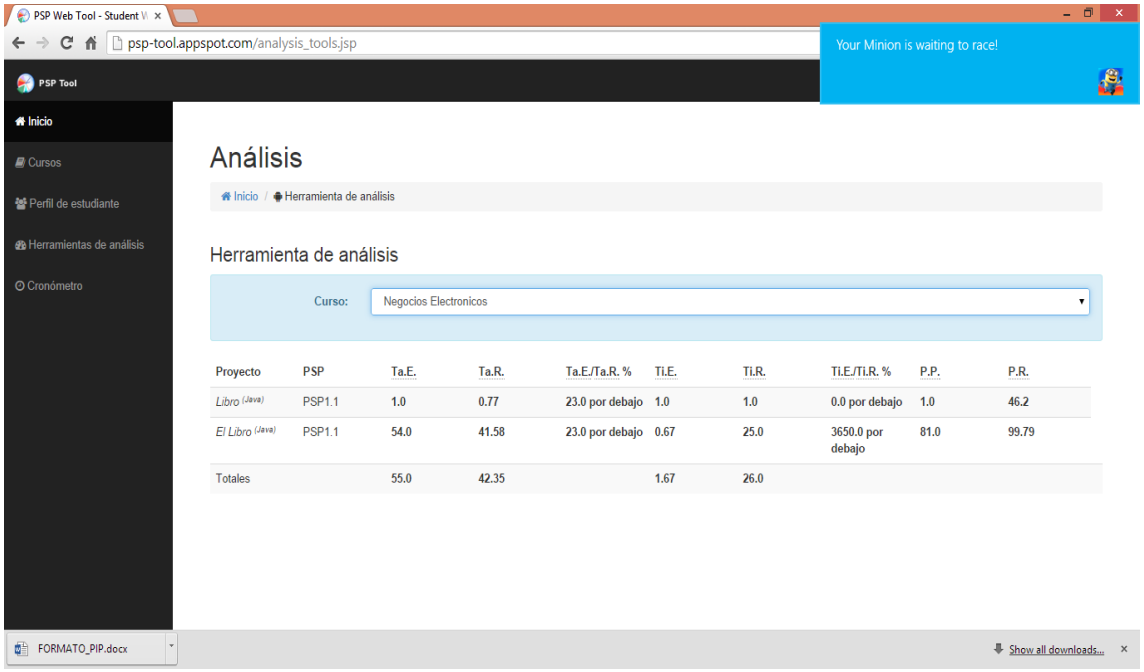


Figura 81: Pruebas – Herramientas de Análisis

Fuente: Elaboración Propia

Resultados de Aplicación de la Herramienta a los alumnos del Curso de Negocios Electrónicos (Muestra)

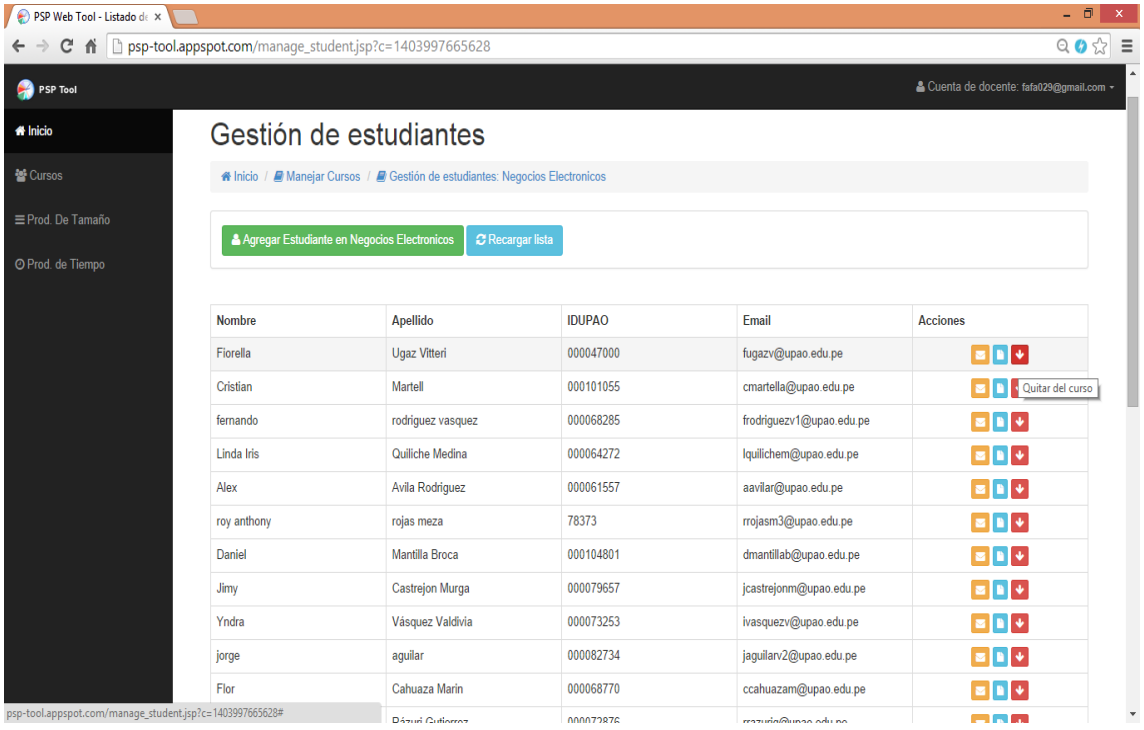


Figura 82: Pruebas – Relación de la Muestra

Fuente: Elaboración Propia

- Se obtuvieron los resultados de Productividad de cada alumno en cuanto a la estimación de tamaño de un proyecto.

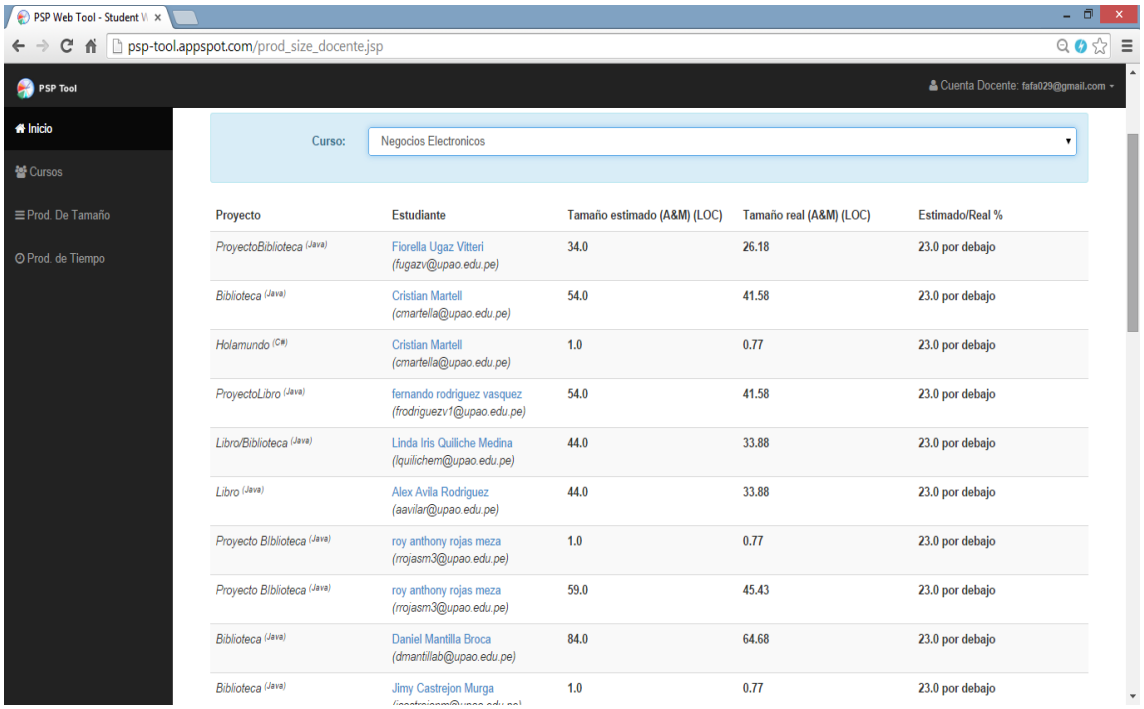


Figura 83: Pruebas – Productividad de Tamaño

Fuente: Elaboración Propia

- Se obtuvieron los resultados de Productividad de cada alumno en cuanto a la estimación de tiempo de un proyecto.

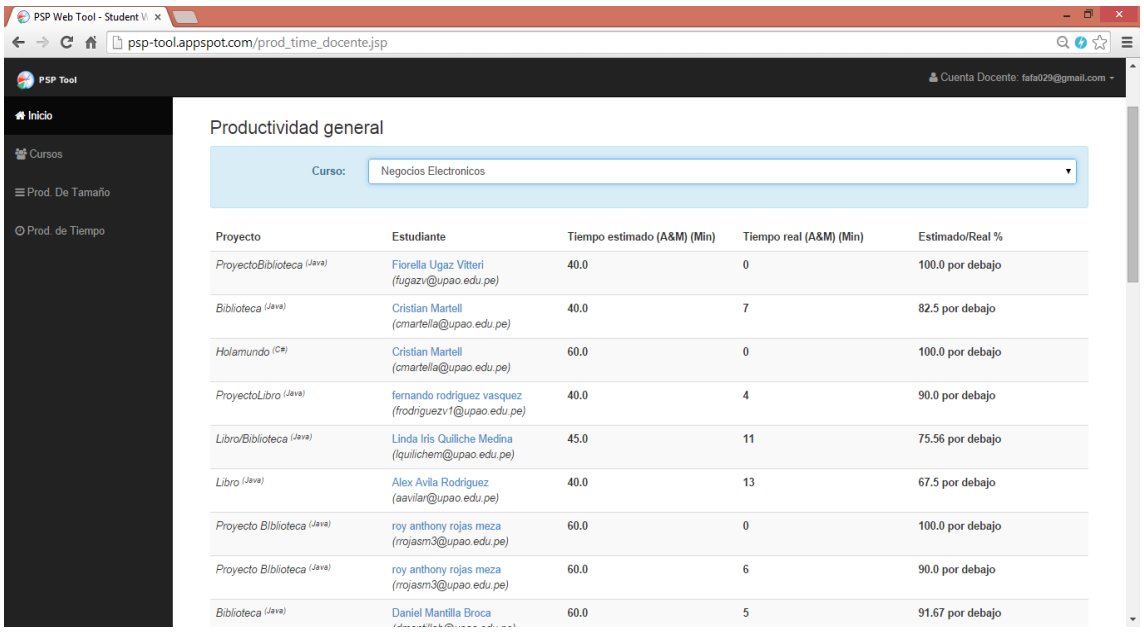


Figura 84: Pruebas – Productividad de Tiempo

Fuente: Elaboración Propia

4.5. POSTMORTEM

En la siguiente tabla se muestran los resultados obtenidos durante el proceso de desarrollo de la herramienta cloud “PSP Web Tool”

PSP0 Resumen del Plan de Proyecto					
Propietario del Proyecto:	Fátima Flores				
Fecha de Inicio:	mar 03, 2014 12:00:00 AM				
Fecha de Culminación:	jul 08, 2014 12:00:00 AM				Completado: *
Palabras claves:	Tesis - PSP -Cloud				
Lenguaje:	Java				
Tamaño del Programa	**Actual**				
Base (B)	0				
Eliminado (D)	0				
Modificado (M)	0				
Agregado (A)	1546				
Reusado (R)	0				
Agregado y Modificado (A+M)	1546				
Total Size (T)	1546				
Total New Reusable	0				
Tiempo en cada Fase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**	
Planificación		44:35	44:35	6,93%	
Diseño		57:57	57:57	9,01%	
Codificación		497:01	497:01	77,20%	
Compilación		35:51	35:51	5,57%	
Test		3:23	3:23	0,53%	
Postmortem		4:38	4:38	0,72%	
Total	650:00	643:25	643:25		
Defectos Encontrados		**Actual**	**To Date**	**To Date %**	
Antes del Desarrollo		0	0	0%	
Planificación		3	3	5,35%	
Diseño		1	1	1,78%	
Codificación		50	50	89,28%	
Compilación		0	0	0%	
Test		2	2	3,57%	
Total		56	56		
Defectos Eliminados		**Actual**	**To Date**	**To Date %**	
Planificación		1	1	1,78%	
Diseño		2	2	3,57%	
Codificación		48	48	85,71%	
Compilación		2	2	3,57%	
Test		2	2	3,57%	
Total		56	56		
Después del Desarrollo		5	5		

Tabla 31: Resumen del Plan de Proyecto PSP Web Tool

V. DISCUCIÓN

El propósito de esta investigación fue determinar en qué media la construcción de una solución Cloud Computing que automatiza las tareas del PSP facilita su adopción teniendo como muestra a los estudiantes del décimo ciclo de la escuela de Ingeniería de Computación y Sistemas de la Universidad Privada Antenor Orrego en el curso de Negocios Electrónicos.

El método que se ha empleado para esta investigación (T-Student) ha sido adecuado ya que lo que se pretendía lograr es evaluar la adopción de PSP. Siguiendo el diseño propuesto en la etapa pre experimental se determinó el nivel de conocimientos en relación a los conceptos de planificación, estimación, uso de métricas, clasificación de defectos y los alcances con los que cuenta el PSP. Posteriormente se procedió a capacitar sobre los temas antes señalados haciendo uso de la herramienta propuesta en este trabajo. Finalmente se evaluó los indicadores definidos para calificar los proyectos que los estudiantes han realizado en el curso de negocios electrónicos.

Durante el desarrollo del trabajo se tuvo limitaciones ya que la idea fue enseñarles todos los niveles del PSP, pero por motivo de que es un tema muy amplio y no se disponía tiempo por parte de los estudiantes, sólo se limitó hasta el nivel 1.1 el cual incluye tareas de planificación como: registrar tipos de defectos, registro de tiempos en cada fase, registro de defectos, realización del diseño conceptual para estimación de proxies, registro de información historial de desarrollo y comparación postmorten. Pero esto no implicó a que el instrumento utilizado (examen) fracasara, ya que este tipo de examen fue validado por Estadista y dos expertos en el tema de investigación.

Al principio se aplicó una prueba piloto en base a una encuesta, pero los datos encontrados fueron vagos, por lo que se optó mejor a un examen teniendo una calificación por cada pregunta del 1 al 4.

ANÁLISIS DE RESULTADOS

De acuerdo con los resultados encontrados en la investigación se puede decir que existe una diferencia significativa en el hecho de utilizar una Solución Cloud contra no utilizar ninguna para adoptar las practicas o tareas que sugiere el nivel 1.1 del PSP.

A la luz de los resultados obtenidos en experimento suponemos que pueden servir para realizar generalizaciones respecto a otros estudiantes que se encuentran en otros ciclos y que estén vinculados con los procesos de desarrollo de Software, también, supongo que es posible generalizar en relación a la práctica ingenieril de aquellos profesionales que recién estén empezando en el ámbito laboral y que cuenta con poca o ninguna experiencia con las ideas de calidad de software a nivel personal, como la planificación, estimación, métricas, entre otros, y que son demandados por el PSP 1.1.

DISCUSIÓN DE LOS RESULTADOS

Con la prueba T de dos colas rechazamos la hipótesis nula la cual indica que no existe diferencia entre la adopción del PSP vía una Solución Cloud o sin ella. Al rechazar la nula estamos aceptando la alterna y con esta afirmamos que existe diferencia significativa pues a la luz de los resultados hay un rango elevado en la adopción por parte de los estudiantes hacia el PSP facilitada por la herramienta Cloud.

Esta diferencia significativa puede explicarse, por un lado porque la solución facilita el proceso de registro pues se muestra indica y ejemplifica como realizarla, por otro lado al almacenarse en la nube no es preciso invertir tiempo y esfuerzo en transportar la información.

RELACIONES ENTRE LAS RESULTADOS ENCONTRADOS VS OTROS TRABAJOS REALIZADOS

Los resultados obtenidos nos demuestran que estamos casi por el mismo nivel de adopción llevados a cabo en México, España, Colombia y cuyos niveles fueron de 73%, 63.67% y 50% respectivamente, los resultados obtenidos por las investigaciones de los países mencionados anteriormente se obtuvieron con otro tipo de método (T-Student para

una media simple) y usaron como herramienta de adopción una macro en Excel que brinda el SEI (Instituto de Ingeniería de Software), pero lo que se asemeja con el resultado de esta investigación es que ellos también tuvieron como muestra a estudiantes de sistemas, software y tuvieron resultados favorables que se mencionaron anteriormente.

Es decir usaron diferente método de investigación, diferente tipo de herramienta pero la finalidad es la misma, se concluyó que un estudiante de sistemas o software puede llegar a adoptar el PSP desde su formación.

CONTRASTACIÓN DE LA HIPÓTESIS

La hipótesis se contrastó con el diseño experimental Pre-Test y Post-Test, trabajando con un grupo de estudiantes del décimo ciclo pertenecientes al curso de negocios electrónicos. Primero se realizaron las observaciones correspondientes antes de aplicar la variable independiente (X) luego se realizó otra observación después de la aplicación de la variable independiente.

Lo que se busco es facilitar la adopción del Proceso Personal de Desarrollo mediante una herramienta cloud hacia los programadores desde un inicio (desde sus Estudios en Instituciones Superiores) para poder lograr una buena planificación para un programa o proyecto de desarrollo de software abarcando la utilización de los estándares de codificación, determinación de métricas fáciles de entender y poder identificar que defectos frecuentemente se encuentran durante el proceso de desarrollo de software para poder ser gestionados de una manera más rápida. Las variables implicadas dentro de la hipótesis fueron X = Variable Independiente (Construcción de una solución Cloud Computing que automatiza las tareas del Proceso Personal de Software), Y = Variable Dependiente (Adopción del Proceso Personal de Software).

Los indicadores (I) de las variables fueron definidas como:

- Facilidad para planificar en base a estadísticas registradas en una bitácora.
- Facilidad para registrar el uso estándares de codificación y de lenguajes de programación.
- Facilidad para usar la métrica de líneas de código.
- Facilidad para registrar tipologías de defectos.

Indicadores	Forma de Trabajo	
	Sin la Herramienta Cloud	Con la Herramienta Cloud
I_1	X_1	Y_1
I_2	X_2	Y_2
I_3	X_3	Y_3
I_4	X_4	Y_4

Tabla 32: Matriz de Indicadores con datos de la muestra

Las mediciones para cada uno de estos indicadores antes y después se conocen como Xi y Yi respectivamente. Donde i indica la medición para un estudiante en particular, el valor máximo de i es 26.

El diseño estadístico de nuestro experimento es:

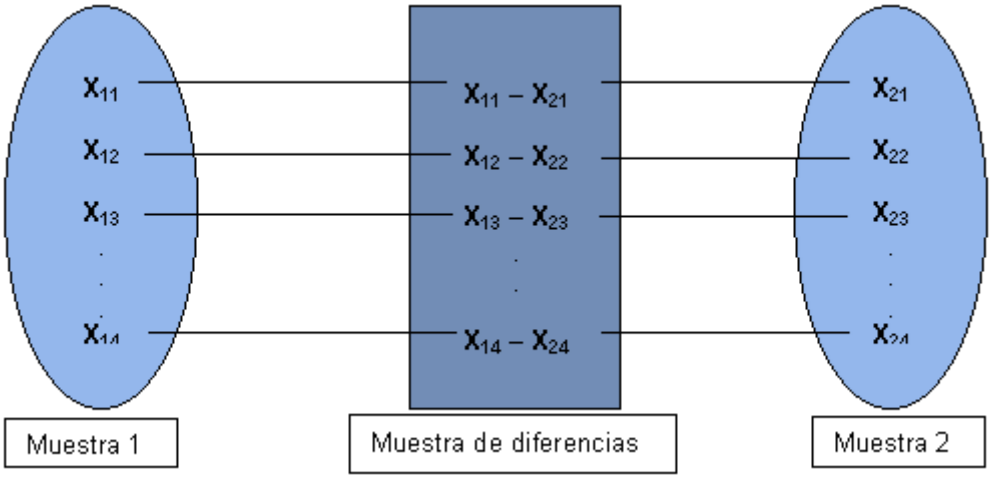


Figura 85: Diseño Estadístico

Para ponderar los indicadores, se ha elaborado un examen con el fin de evaluar si los resultados dados cumplen con la hipótesis trazada. Cada alternativa de respuesta del examen se ha ponderado según en una escala del 1 al 4, siendo 1 el nivel de puntaje más bajo y 4 el puntaje más alto.

El examen se ha aplicado a una población de 40 estudiantes con una muestra de 26 estudiantes del X ciclo de la Carrera de Ingeniería de Computación y Sistemas de la Universidad Privada Antenor Orrego. Teniendo en cuenta el diseño experimental de Pre-Test/Post-Test, este experimento se realizó en dos etapas:

- En la primera etapa se aplicó el examen antes de mostrada, explicada y utilizada la Herramienta Cloud (ver Anexo 01).
- La segunda etapa se realizó después de mostrada, explicada y utilizada la Herramienta Cloud, empleando las mismas preguntas, orientadas a los resultados mostrados por la Herramienta Cloud.

Los resultados que se obtuvieron fueron los siguientes:

- Puntaje total (antes de explicar y usar la herramienta cloud) por variable de acuerdo a la equivalencia de la tabla N° 33:

	Variables			
Estudiante	**X_1**	**X_2**	**X_3**	**X_4**
1	13	15	15	15
2	14	15	15	15
3	13	14	17	13
4	16	13	15	15
5	15	15	13	15
6	16	15	16	16
7	17	16	14	16
8	16	14	15	14
9	16	14	16	15
10	15	13	13	14
11	13	13	13	12
12	15	15	14	14
13	14	15	14	14
14	17	16	14	16
15	14	15	11	13
16	16	15	14	15
17	15	14	13	13
18	15	15	14	13
19	15	17	16	15
20	15	16	16	15
21	16	15	15	15
22	15	14	14	13
23	18	13	13	14
24	17	14	13	15
25	16	14	13	13
26	15	15	14	15

Tabla 33: Puntaje antes de aplicar la herramienta cloud

- Puntaje total (después de explicar y usar la herramienta cloud) por variable de acuerdo a la equivalencia de la tabla N° 33:

	Variables			
Estudiante	**Y_1**	**Y_2**	**Y_3**	**Y_4**
1	15	18	19	18
2	16	17	16	17
3	15	16	16	15
4	17	17	17	16
5	17	17	18	18
6	15	17	17	18
7	18	16	19	17
8	17	16	19	16
9	16	15	19	17
10	16	15	18	15
11	15	15	16	16
12	17	16	17	14
13	15	17	17	17
14	18	17	18	16
15	18	17	18	16
16	18	17	16	15
17	16	17	16	16
18	17	17	16	16
19	17	18	18	17
20	18	17	19	17
21	19	16	18	16
22	19	16	17	15
23	18	16	18	15
24	16	16	17	15
25	17	16	18	15
26	15	16	18	16

Tabla 34: Puntaje después de aplicar la herramienta cloud

RESULTADO DE PRUEBA T

Muestras pareadas		**Media**	**N**	**Desviación Estándar**	**Media Error Estándar**
Par 1	X1	15,2692	26	1,28243	0,25150
	Y1	16,7308	26	1,28243	0,25150
Par 2	X2	14,6154	26	1,02282	0,20059
	Y2	16,4615	26	0,81146	0,15914
Par 3	X3	14,2308	26	1,33589	0,26199
	Y3	17,5000	26	1,06771	0,20939
Par 4	X4	14,3462	26	1,09334	0,21442
	Y4	16,1154	26	1,07059	0,20996

Tabla 35: Muestras Estadísticas Pareadas

Muestras pareadas		N	Correlación	Sig.
Par 1	X1 & Y1	26	0,508	0,008
Par 2	X2 & Y2	26	0,608	0,001
Par 3	X3 & Y3	26	0,084	0,683
Par 4	X4 & Y4	26	0,477	0,014

Tabla 36: Correlación de Muestras Pareadas

		Diferencias Emparejadas					t	Df	Sig. (2-colas)
					99% Diferencia de Intervalo de Confianza				
Muestras pareadas		Media	Desviación Estándar	Media Error Estándar	Mínimo	Máximo			
Par 1	X1 - Y1	-1,46154	1,27219	0,24950	-2,15699	-0,76608	-5,858	25	0,000
Par 2	X2 - Y2	-1,84615	,83390	0,16354	-2,30201	-1,39029	-11,289	25	0,000
Par 3	X3 - Y3	-3,26923	1,63848	0,32133	-4,16492	-2,37354	-10,174	25	0,000
Par 4	X4 - Y4	-1,76923	1,10662	0,21703	-2,37418	-1,16428	-8,152	25	0,000

Tabla 37: Test de Muestras Pareadas

Región de Aceptación y Rechazo

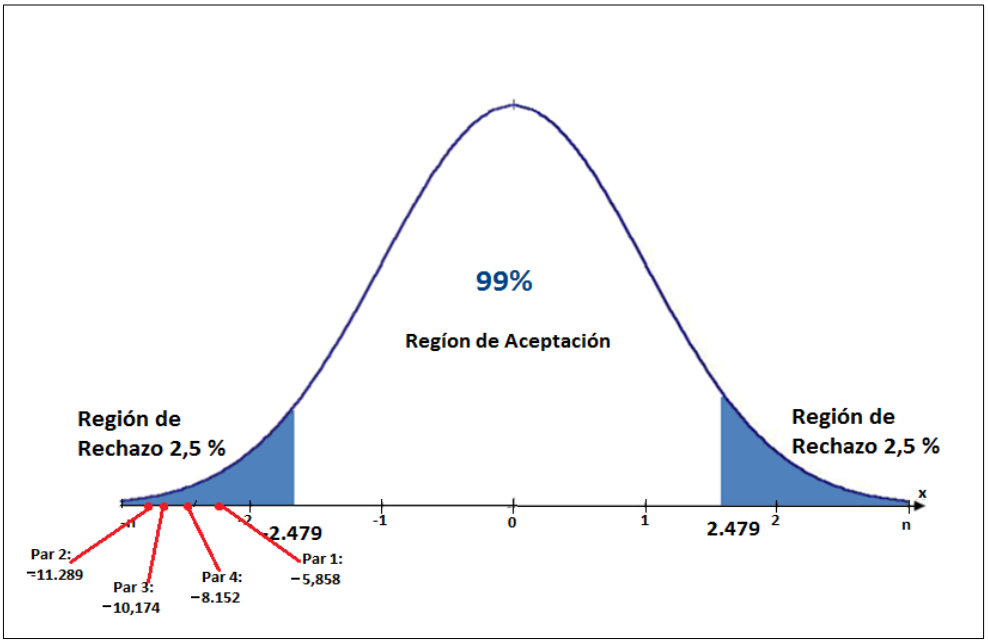


Figura 86: Región de Aceptación y Rechazo

Según la tabla de resultado de T-Student, nos indica que los límites de nuestra región de Rechazo son: <-∞, - 2.479> U< 2.479, ∞+>. Por lo que se a acepta la hipótesis principal, debido a que los valores estadísticos de t son: -5.858, -11.289, -8.152 y -10.174; estos resultados se encuentran dentro de la región de rechazo de la hipótesis nula. Por lo que se concluye que la herramienta, estadísticamente, facilita significativamente la adopción del PSP por parte de los estudiantes en proceso de formación.

TRABAJOS FUTUROS

A partir del trabajo realizado otros que pueden desprenderse de él son: determinar la adopción de todos los niveles del PSP por parte de los estudiantes, realizar experimentos con desarrolladores junior o senior para poder llegar a concluir que los programadores son tipo World Class.

Otros trabajos relacionados son investigar los diferentes tipos de herramientas que engloban al PSP para su uso o enseñanza y determinar cuales se acoplan o adaptar mejor a nuestra realidad. Completar la herramienta Cloud para incluya los demás nivel del PSP. Evaluar la factibilidad del TSP en el desarrollo de habilidades de trabajo en equipo.

CONCLUSIONES

- De acuerdo a la investigación realizada, se ha encontrado que en las empresas de desarrollo de software peruanas, el tema de la calidad y los procesos son de crucial interés (PROMPERU, 2012), es así que a través el Banco Interamericano de Desarrollo viene desarrollando el Programa de Apoyo a la Competitividad de la Industria de Software y en el cual cobra relevancia el modelo de CMMI. Sin embargo este es un modelo a nivel corporativo, mas creemos que son los potenciales humanos los que deben alterar y mejorar sus prácticas y habilidades en la construcción de software; primero a nivel personal, luego a nivel de equipo y finalmente a nivel corporativo. A partir de este último razonamiento es de vital interés que los futuros profesionales en formación desarrollen estas habilidades durante su proceso formativo.

- De manera general el PSP plantea tres fases (Planificación, Desarrollo y Postmortem). La fase de Desarrollo se subdivide en Diseño, Codificación, Compilación, y Testeo), siete procesos a los que denomina niveles (PSP0, PSP0.1, PSP 1, PSP 1.1, PSP 2, PSP2.1 y PSP 3) cada uno de ellos engloba al anterior. Para efectos de nuestro trabajo delimitamos nuestro investigación al PSP 1.1 (que incluye al PSP0, PSP0.1, PSP 1 y PSP 1) que conllevo a abarcar un 100% de las actividades de dicho proceso (PSP 1.1), el mismo que fue suficiente para los fines que este estudio persiguió durante la duración periodo académico en el cual se realizó la investigación. (esas son las restricciones que se tienen).

- En el objetivo 3 se propone una solución cloud que automatice las tareas del PSP, para lograrlo, se tuvo que pasar una curva de aprendizaje bastante elevada, pues fue necesario, por un lado entender el funcionamiento de las herramientas del PSP, luego diseñar e implementar la solución bajo tecnología de GAE de Google y finalmente elaborar un modelo de adiestramiento y entrenamiento.

- La solución creada fue puesta en producción a fin de evaluarla estadísticamente obteniendo como resultado la certeza que esta facilita significativamente el entrenamiento y la puesta en práctica del PSP, pues al ser una solución Web. La

transportabilidad de la información deja de ser un problema porque no es necesario llevarla dado que se encuentra disponible en la nube. Por otro lado y como consecuencia de nuestro trabajo tenemos el beneficio en términos de proporcionar información a los estudiantes porque pueden estimar en base a información histórica; y por parte del docente o los docentes mejorar su práctica educativa en el desarrollo de las competencias y capacidades del futuro profesional que se encuentra en formación. Finalmente podemos avizorar el potencial de negocios que la herramienta tiene.

- Estadísticamente la herramienta construida muestra de manera significativa su aporte en un proceso de adopción de las prácticas de desarrollo de software que el PSP 1.1 sugiere. Según los resultados los 26 estudiantes que participaron completamente en el proceso tuvieron una mejora sustancial en el proceso de adopción. Estos 26 representan el 65% de los estudiantes que completaron las actividades del experimento. Los restantes fueron excluidos por que no completaron o no empezaron el experimento y cayeron en los criterios de exclusión. Con base a estos resultados no garantizamos la confiabilidad de los resultados obtenidos.

- La experiencia que los estudiantes han obtenido durante el experimento nos indica que inicialmente trabajar con nuestra solución PSP pudiera parecer tedioso, ya que es preciso registrar cada fase, cada defecto, pero luego de este registro inicial las tareas de estimaciones se hacen mucho más rápidas y precisas pues ya se cuenta con un historial sobre los que se realiza la estimación, por lo tanto, es necesario que mientras más precisa sea la información inicial registrada mejor serán las estimaciones.

RECOMENDACIONES

- Desarrollar competencias relacionadas con procesos de calidad de software a nivel personal.

- Que la adopción del PSP sea gradual a medida que se vayan asimilando y desarrollando las prácticas y habilidades que el PSP sugiere.

- Implementar, el PSP, en el plan de estudios a partir del segundo año de estudios para favorecer el desarrollo de las competencias personales en la construcción de software.

- Construir y completar la solución cloud, siguiendo las directrices y técnicas de este trabajo.

REFERENCIAS BIBLIOGRAFIAS

- ✓ Alianza Parlamentaria;. (28 de Mayo de 2012). COMISION DE CIENCIA, INNOVACION Y TECNOLOGIA. *Creación del Instituto Nacional de Software Peruano*, 16. Lima, Perú: Congreso de la República. Obtenido de Congreso de La República.
- ✓ APESOFT. (2011). *PERU, Software PORTAFOLIO.* Lima: PROMPERU.
- ✓ Association For Computing Machinery. (5 de Mayo de 2008). *ACM*. Obtenido de http://www.acm.org/
- ✓ Benetó Micó, A. (2013). La Crisis del Software . Valencia, España.
- ✓ Campderrich Falgueras, B. (2008). *Ingeniería del Software.* Barcelona: UOC.
- ✓ Candermo, J. (2014). *Sistema de Gestión de la Calidad - Norma ISO 9001:2008.*
- ✓ Ceballos Cardona, Y., & Velez Tascon, L. (2013). *Norma ISO 12207.* MANIZALES.
- ✓ Chornyi, D., Riediger, J., & Wolfenstetter, T. (2010). *INTO THE CLOUD, An evaluation of the Google App Engine.* Alemania: SEBIS.
- ✓ Collazos, A. (2011). *Diseño Personal del Software.* Colombia.
- ✓ Correal, D. (2009). *Modelos, Estándares y Procesos en Ingenieria de Software.* Quito: ECOS.
- ✓ De la Villa, M., Ruiz, M., & Ramos, I. (2004). *EHU.* Recuperado el 20 de Enero de 2014, de http://www.sc.ehu.es/jiwdocoj/remis/docs/DelaVillaadis2004.doc
- ✓ Díaz Morales, R. (2013). *Amazon Web Services TM.* TSC.
- ✓ Forradellas, P., Pantaleo, G., & Rogers, J. (2011). *El modelo CMM/CMMI - Cómo garantizar el éxito del proceso de mejoras en las organizaciones, superando los conflictos y tensiones generados por su implementación.* ITM-MENTOR.

- ✓ García Alonso, M. (2008). *Métricas y Procesos PSP Personal Software Process.* Medellin.
- ✓ García Coria, J. A. (2009). *Modelo de Calidad CMMI.* Salamanca: CENIT.
- ✓ GLOBALES. (2007). *CMMI - Anexo 1.* Segovia.
- ✓ Hernández Hernández, M. J. (2013). *Experiencia en Personal Process.* México: SG VIRTUAL CONFERENCE.
- ✓ Humphrey , W. S. (1995). *A Discipline for Software Engineering.* Míchigan: Addison-Wesley.
- ✓ Humphrey, W. (2000). *The Personal Software Process.* Carnegie Mellon University.
- ✓ Humphrey, W. (2000). *The Team Software Process (TSP).* EE.UU.: SIE.
- ✓ Humphrey, W. S. (2001). *Introduccion al Proceso Software Personal.* (A. WESLEY, Trad.) Madrid: PEARSON EDUCACION, S.A.
- ✓ Humphrey, W., Chick, T., Nichols, W., & Pomeroy-Huff, M. (2010). *Team Software Process, Body of Knowledge.* Carnegie Mellon University.
- ✓ ICIC. (2008). *¿QUÉ ES EL CLOUD COMPUTING? Recomendaciones para Empresas.* Argentina.
- ✓ Institute CMMI. (2014). *What is CMMI?* Recuperado el 02 de Febrero de 2014, de http://cmmiinstitute.com/
- ✓ Instituto Tecnológico de Morelia. (2008). *Modelos de Proceso del Software.* Michoacan: ITM.
- ✓ ISO. (2011). *¿Qué es ISO 9001:2008?* Recuperado el 25 de Febrero de 2014, de http://www.normas9000.com/que-es-iso-9000.html
- ✓ ISO 9001. (2008). *ISO 9001:2008 Sistemas de Gestión de Calidad - Requisitos.* ISO 2008.
- ✓ ISO/IEC 12207. (2008). *ISO/IEC 12207: Systems and software engineering-Software life cycle processes.* ISO.

- ✓ Iznaga Lamour, Y., Delgado Fernández, M., & Febles Estrada, A. (2009). *Macro-Proceso de planificación de la calidad para los proyectos productivos en la Universidad de las Ciencias Informáticas.* La Habana: Universidad de las Ciencias informáticas.
- ✓ Jinesh, V., & Sajee, M. (2014). *Amazon Web Services, Overview of Amazon Web Services.* Seattle: AWS.
- ✓ José Fuertes, J. (18 de Mayo de 2012). Cloud Computing, herramienta de inclusión digital para empresarios. *RPP Noticias*, pág. 1.
- ✓ Laguna, I. T. (06 de 2008). *ITLALAGUNA.* Recuperado el 10 de Enero de 2014, de http://www.itlalaguna.edu.mx
- ✓ Landa Martín, I., & Zorrilla Castro, U. (2011). *Súbete a la nube de Micrososft - Parte1: Introducción a Windows Azure.* KRASIS PRESS.
- ✓ Microsoft. (03 de Febrero de 2014). *¿Qué es Windows Azure?* Recuperado el 25 de Febrero de 2014, de WindowsAzure: http://www.windowsazure.com/es-es/overview/what-is-windows-azure/
- ✓ Moreno, S., Tasistro, A., & Vallespir, D. (2011). *PSP - VDC: Una Propuesta que Incorpora el Diseño por Contrato Verificado al Personal Software Process.* Uruguay: FING.
- ✓ Mutafelija, B., & Stromberg, H. (2003). *Systematic Process Improvement using ISO 9001:2000 and CMMI.* Artech House Computing Library.
- ✓ Piattini, M., & García, F. (2003). *Calidad en el desarrollo y mantenimiento del.* Madrid: RA-MA.
- ✓ Pino, F., García, F., Ruiz, F., & Pattini, M. (2006). *Adaptación de las normas ISO/IEC 12207:2002 y ISO/IEC 15504:2003 para la evaluación de la madurez de procesos software en países en desarrollo.* IEEE América Latina.
- ✓ Pomeroy-Huff, M., Cannon, R., Chick, T., Mullaney, J., & Nichols, W. (2009). *The Personal Software Process (PSP) Body of Knowledge.* EE.UU.: Carnegie Mellon University.

- ✓ Pressman, R. (2010). *Software Engineering, A Practitioner's Approach* (7ma Edicion ed.). New York: McGraw-Hill Higher Education.
- ✓ PROMPERU. (2012). *Perú Service: Summit 2012.* Perú: SIICEX.
- ✓ Sánchez Lorenzo, G. A. (2008). *MEJORA DEl proceso software de una pequeña empresa desarrolladora de software: caso Competisoft-Perú.* Lima: UPC.
- ✓ Sanderson, D. (2009). *Programming Google App Engine.* oreilly.
- ✓ SEI. (2010). *CMMI® para Desarrollo Version 1.3.* Carnegie Mellon University.
- ✓ SIE. (2010). *Introduction to the Team Software Process.* Pittsburgh: Carnegie Mellon University.
- ✓ Software Engineering Institute. (2008). *Personal Software Process, Estimación con PROBE I.* Pensilvania: Carnegie Mellon University.
- ✓ Sommerville, I. (2011). *Ingeniería del Software* (9na Edición ed.). México, España: Pearson Educación de México.
- ✓ Srivastava, S., Trehan, V., & Yadav, P. (2012). *Google App Engine.* IJEIT.
- ✓ Takai, T. (2012). *Cloud Computing Strategy.* United States of America: DoD.
- ✓ Tuya, J., Ramos Román, I., & Dolado Cosín, J. J. (2007). *Técnicas cuantitativas para la gestión en la ingeniería del software.* España: Netbiblo.
- ✓ Urubeña, A., Ferrari, A., Blanco, D., & Valdecasa, E. (2012). *Cloud Computing Retos y Oportunidades.* Madrid: ONTSI.
- ✓ Watts, S., Chick, T., Nichols, W., & Pomeroy-Hulf, M. (2010). *Team Software Process (TSP) Body of Knowledge (BOK)* . EE.UU.: Carnegie Mellon University.
- ✓ WIKIPEDIA. (09 de Enero de 2014). *Google App Engine*. Recuperado el 20 de Febrero de 2014, de http://es.wikipedia.org/wiki/Google_App_Engine

ANEXOS

ANEXO 01: EXAMEN

NEGOCIOS ELECTRONICOS - PRACTICA CALIFICADA

Apellidos y Nombres: _____ ID:_____

INSTRUCCIONES:

a) *Descargue los proyectos contenidos en los archivos Tutorial.rar y Libros.rar, la presentación de la herramienta psp-training.rar en la carpeta PRACTICA II en la ruta F:\NE\LAB.*
b) *Vaya al sitio web http://psp-tool.appspot.com y entre con una cuenta de gmail (si no la tiene, deberá crearlo)*
c) *Atienda la instrucciones del instructor y luego resuelva las siguientes cuestiones*

X1: Facilidad para planificar en base a ESTADÍSTICAS REGISTRADAS en una bitácora.

1. En la siguiente porción de código, indique cuantas líneas de código hay.

```
package ejemplos;

public class matrices {
   public static void main( String args[] ) {

   // Declaramos un array de dos dimensiones con un tamaño de 3 en la
   // Decladramos el array con un tamano de 3 en su primera dimensión para
   // posteriormente declarar la segunda dimensión.

   int matriz[][] = new int[3][];
   matriz[0] = new int[2];
   matriz[1] = new int[3];
   matriz[2] = new int[4];

   // Ponemos datos en el array

   for ( int i=0; i < 3; i++ ) {
      for ( int j=0; j < matriz[i].length; j++ )
         matriz[i][j] = i * j;
   }

   // y vemos su contenido, utilizando un bucle for
   for ( int i=0; i < 3; i++ ) {
      for ( int j=0; j < matriz[i].length; j++ )
         System.out.print( matriz[i][j] );
      System.out.println();
   }

   // Intetamos acceder a un elemento que esta fuera de los limites del array
   System.out.println( "Elemento fuera de limites del array" );
   matriz[4][0] = 7;

   // El compilador lanzara una excepción de tipo ArrayIndexOutOfBounds
   }
}
```

_____

_____

_____

2. Cuál es el tiempo estimado para crear la clase Libro.java.

_____

_____

3. En el siguiente programa indique cuantos tipos de defecto hay.

```
public class Alumno
{
   private String nombre;
   private String apellidos;
   private int añoNacimiento;
   private String grupo;

   //Si el nombre o los apellidos son cadenas null
   //lanzamos una excepción y el objeto no se crea
   public Alumno (String nombre, String apellidos, int añoNacimiento) throws Exception {
      if (nombre == null || apellidos == null)
         throw new Exception("Argumentos no válidos");
      if (añoNacimiento < 0)
         throw new Exception("Año incorrecto");
      this.nombre = nombre;
      this.apellidos = apellidos;
      this.añoNacimiento = añoNacimiento;
   }

   public String dameGrupo() {
      return grupo;
   }

   public void imprime() {
      System.out.println("nombre:" + nombre);
      System.out.println("apellidos:" + nombre);
      System.out.println("año de nacimiento:" + añoNacimiento);
      System.out.println("grupo:" + grupo);
   }
}
```

_____

_____

_____

4. ¿Qué tipo de interrupciones aparecen durante el desarrollo de software?

_____

_____

_____

5. En el proyecto Tutorial que clases son candidatas para reutilizar, identifique e indique los proxys que encuentra.

_____

_____

_____

X2: Facilidad para registrar el uso estándares de codificación y de lenguajes de programación.

6. En el siguiente bloque de código Java indique que estándares de codificación aparecen.

```
public class matrices {
   public static void main( String args[] ) {

   // Declaramos un array de dos dimensiones con un tamaño de 3 en la
   // Decladramos el array con un tamano de 3 en su primera dimensión para
   // posteriormente declarar la segunda dimensión.

   int matriz[][] = new int[3][];
   matriz[0] = new int[2];
   matriz[1] = new int[3];
   matriz[2] = new int[4];

   // Ponemos datos en el array

   for ( int i=0; i < 3; i++ ) {
      for ( int j=0; j < matriz[i].length; j++ )
         matriz[i][j] = i * j;
   }

   // y vemos su contenido, utilizando un bucle for
   for ( int i=0; i < 3; i++ ) {
      for ( int j=0; j < matriz[i].length; j++ )
         System.out.print( matriz[i][j] );
      System.out.println();
   }

   // Intetamos acceder a un elemento que esta fuera de los limites del array
   System.out.println( "Elemento fuera de limites del array" );
   matriz[4][0] = 7;

   // El compilador lanzara una excepción de tipo ArrayIndexOutOfBounds
   }
}
```

_____
_____
_____

7. Según los requerimientos dados para la elaboración del proyecto tutorial, indique como serían los nombres de las clases y sus métodos respectivamente.

_____
_____
_____

8. En el diseño conceptual para el proyecto tutorial, cuales son los diagramas precisos que se deberían utilizar de acuerdo al tamaño y objetivo del proyecto

_____
_____
_____

9. Indicar en el siguiente cuadro de tipo de sentencias si se cuenta LOC o No, justificar la respuesta.

Tipo de Sentencia	¿Se puede Contar?
Ejecutable	
No-ejecutable	
Declaraciones	
Directivas de Compilación	
Comentarios	
Líneas propias	
Con código fuente	
Líneas en blanco	

10. Escriba los tipos de Notaciones para declaración de variables, métodos, clases, etc que usaría para el proyecto tutorial.

_____
_____
_____
_____

X3: Facilidad para usar la métrica de líneas de código.

11. Estime el tamaño en LOC (líneas de código) de una unidad de programa (proxy) en el proyecto tutorial.

_____
_____
_____

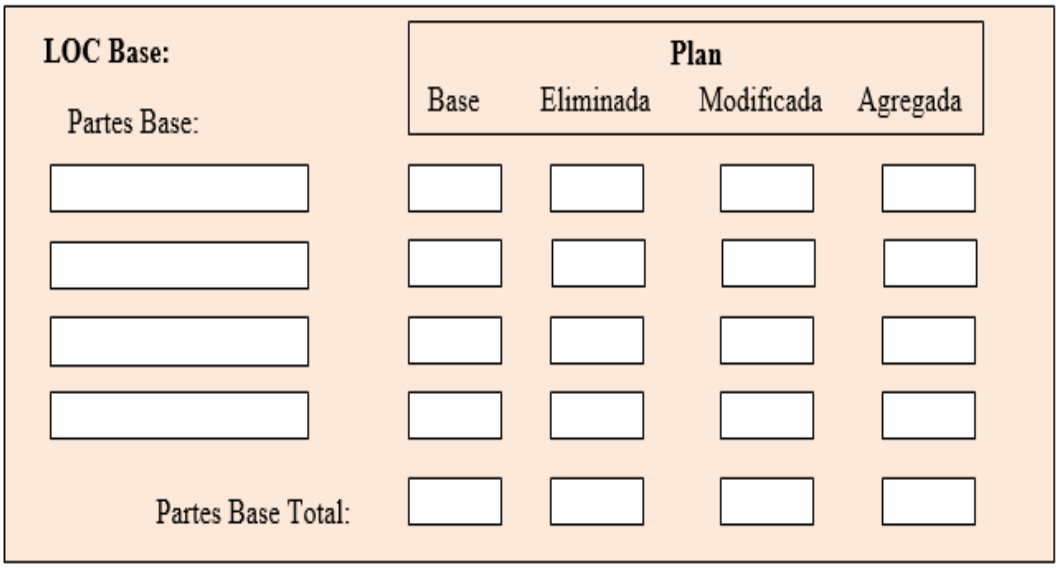
12. Suponga que el dueño del proyecto Tutorial el cliente ha solicitado una modificación a la clase Tutorial para incluir un atributo de estado cuyos valores son: cancelado, postergado, realizado. ¿Qué ajustes son necesarios?

_____
_____
_____

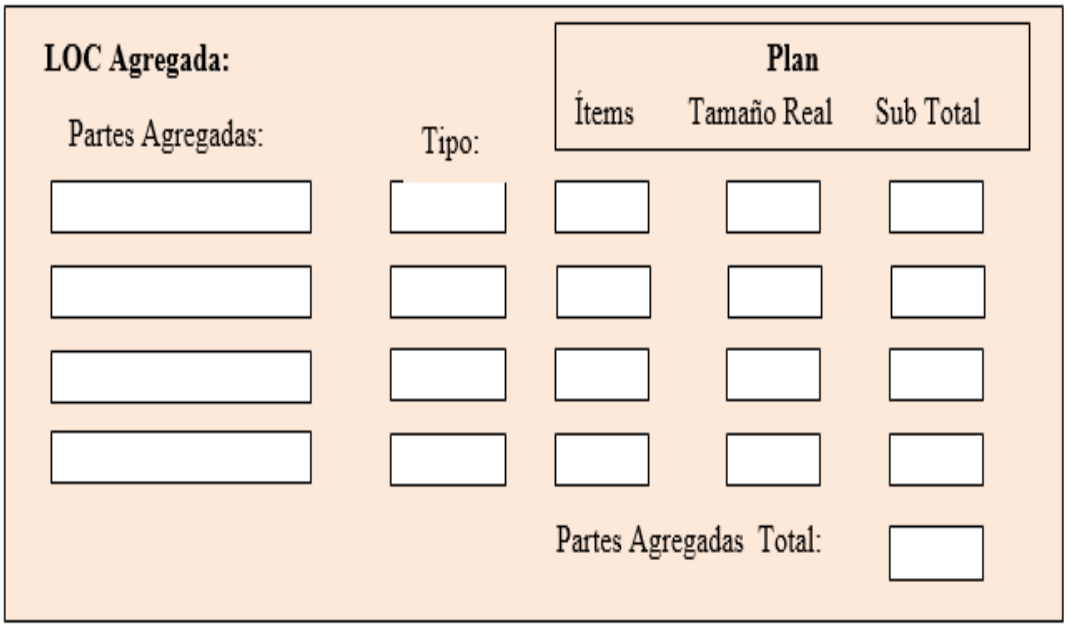
13. **¿**Qué categorías de código (LOC) cuenta el proyecto tutorial**?**

_____
_____
_____

14. Dado los requerimientos y/o explicación del proyecto tutorial, llenar los espacios en blanco, referentes a LOC Base y LOC Agregadas:



15. Dado los requerimientos y/o explicación del proyecto tutorial, llenar los espacios en blanco, referentes a LOC Base y LOC Agregadas sdsd



X4: Facilidad para registrar tipologías de defectos.

16. Dado el siguiente cuadro, relacione los tipo de defecto con su respectiva descripción:

	Descripción	Tipo de Defecto
	Falta de la eficiencia de ejecución prevista.	**1. Datos**
	Defectos gramaticales.	**2. Ejecución**
	Defectos en la comunicación con dispositivos.	**3. Lógico**
	Desviación sobre los estándares que debe seguir el producto.	**4. Entorno de pruebas**
	Defectos en la especificación de los datos (por ejemplo, declaraciones, inicializaciones o descripciones de datos incorrectas).	**5. Documentación**
	Especificaciones incompletas de una condición de prueba o una desviación del plan de pruebas.	**6. Casos de prueba**
	Defectos en la definición o especificación del alcance de las pruebas.	**7. Sintaxis**
	Defectos en la lógica de control de un módulo (por ejemplo, límites de los bucles incorrectos).	**8. Funcionalidad**
	Defectos en la definición o especificación software o hardware de pruebas, nivel de seguridad, etc.	**9. Entrada/Salida**
	Defectos en la comunicación entre componentes del software (por ejemplo, llamadas incorrectas de los módulos, paso de datos incorrectos entre módulos).	**10. Factores humanos**
	Descripciones inadecuadas de algún componente (por ejemplo, comentarios incorrectos).	**11. Plan de pruebas**
	Procedimientos operativos incorrectos.	**12. Interfaz**
	Defectos en la especificación de las funciones de un componente.	**13. Cumplimiento de 14. estándares**

17. Suponga que en el proyecto tutorial, durante la fase de codificación se han encontrado 18 defectos, y se ha gastado 45 minutos en eliminar dichos defectos; calcular los defectos eliminados por unidad de tiempo (hora).

_____
_____
_____

18. Suponga que gasto 38 minutos para eliminar los defectos encontrados en la etapa de compilación y en dicha etapa (compilación) demoró 50 minutos, calcular la eficiencia para la eliminación de defectos.

_____
_____
_____

19. En PSP, un programa con cinco o menos defectos/KLOC es considerado como un programa de buena calidad, suponga que en el proyecto tutorial se han encontrado 2 defectos en la fase planificación, 2 en la fase de diseño, 13 en la fase de desarrollo, y el programa tiene un total de 1500 LOC, calcular la densidad de defectos (teniendo en cuenta la fórmula: 1000*# defectos/tamaño del programa) e indicar si es considerado un programa de calidad.

    _____
    _____
    _____
    _____

20. Dada la siguiente tabla, indique el porcentaje de rendimiento de cada fase para encontrar y eliminar defectos, teniendo en cuenta la fórmula: 100*#defectos eliminados en fase/(#total de defectos removidos - #total de defectos eliminados en las fases anteriores.

Fase	**Defectos Encontrados**	**Defectos Eliminados**	**Rendimiento por Fase**
Diseño detallado	26	0	
Revisión de diseño	0	11	
Código	39	0	
Revisión de código	0	28	
Compilación	0	12	
Pruebas unitarias	0	7	
Final pruebas unitarias	0	7	
Total	65	65	

ANEXO 02: RESULTADOS PRIMER EXAMEN

Preguntas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Estudiante	**Variable X_1**					**Variable X_2**					**Variable X_3**				**Variable X_4**					
1	3	3	2	1	2	3	3	3	3	3	3	3	3	3	3	3	2	3	3	4
2	3	3	2	1	3	3	3	3	3	3	3	3	3	3	3	3	2	3	3	4
3	1	2	1	3	3	2	2	2	3	3	5	3	3	4	4	2	1	2	2	2
4	3	4	3	2	4	1	3	1	2	4	3	3	3	3	3	4	3	1	3	3
5	3	3	2	3	3	3	4	3	2	3	2	2	2	2	1	3	2	3	4	4
6	4	3	3	3	3	2	4	2	3	4	4	3	3	4	4	3	3	2	4	4
7	5	4	4	3	2	4	3	3	4	3	5	2	2	2	3	4	4	4	3	3
8	3	3	5	4	2	2	2	3	3	3	3	3	3	3	2	3	5	2	2	2
9	3	3	4	4	2	1	4	2	2	4	3	2	4	4	2	3	4	1	4	4
10	3	3	3	5	1	2	2	3	2	2	3	2	2	2	2	3	3	2	2	3
11	1	1	2	5	1	1	1	3	5	1	5	1	1	1	1	1	2	1	1	4
12	4	3	3	3	1	2	2	2	5	4	4	2	2	2	2	3	3	2	2	4
13	1	1	3	5	3	3	3	3	3	3	5	1	3	1	2	1	3	3	3	3
14	3	3	4	4	4	5	1	3	5	2	4	1	1	5	5	3	4	5	1	2
15	1	1	3	4	4	3	2	3	4	3	3	1	1	1	2	1	3	3	2	2
16	3	2	3	4	5	3	2	4	3	2	4	2	3	1	4	2	3	3	2	3
17	4	1	2	3	4	2	3	3	3	2	3	2	2	2	2	1	2	2	3	2
18	3	3	2	3	3	3	3	3	3	2	4	2	2	3	1	3	2	3	3	1
19	3	2	3	4	3	4	2	3	5	4	5	3	3	3	2	2	3	4	2	5
20	3	3	2	4	3	4	3	2	4	3	3	3	4	3	3	3	2	4	3	3
21	3	3	3	4	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2
22	3	3	4	2	3	1	2	4	3	3	3	2	3	2	2	3	4	1	2	2
23	4	4	4	4	4	2	2	3	1	2	2	2	1	3	3	4	4	2	2	1
24	3	4	5	4	2	2	3	3	2	2	2	1	3	3	1	4	5	2	3	3
25	4	2	4	4	2	1	2	3	4	2	3	2	2	2	2	2	4	1	2	3
26	3	4	3	3	1	3	2	3	3	3	4	2	3	2	3	4	3	3	2	3

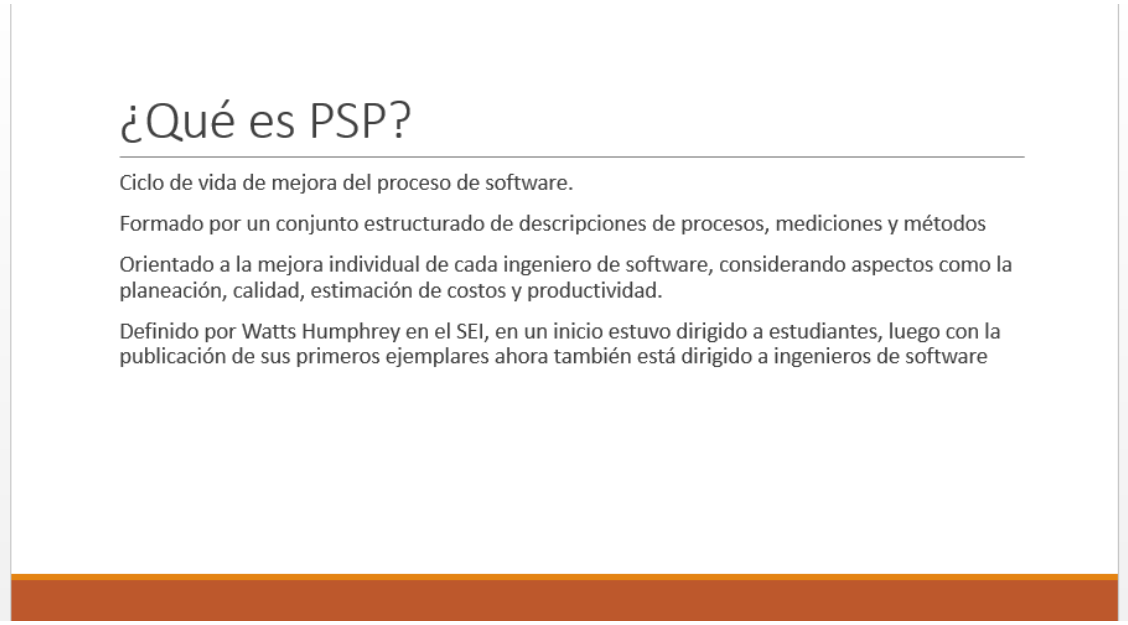
ANEXO 03: RESULTADOS SEGUNDO EXAMEN

Preguntas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Estudiante	**Variable Y_1**					**Variable Y_2**					**Variable Y_3**				**Variable Y_4**					
1	4	4	2	2	3	4	3	4	4	5	5	4	4	5	4	3	5	4	4	5
2	4	4	2	2	4	4	3	4	4	5	4	3	3	3	4	3	4	3	4	5
3	2	3	2	4	4	3	2	4	3	5	3	3	4	4	3	2	3	3	4	3
4	4	5	2	2	5	2	3	3	4	4	4	3	4	4	2	3	4	3	5	4
5	5	3	2	5	4	3	4	4	4	5	4	3	5	4	3	4	4	3	5	5
6	3	3	2	3	4	3	4	4	3	4	5	3	3	4	3	4	5	3	5	5
7	5	4	4	5	3	3	3	3	4	5	5	3	5	5	3	3	5	3	4	4
8	5	3	4	3	3	3	2	3	3	5	5	3	5	5	3	2	5	3	4	3
9	4	4	2	3	3	1	4	4	4	4	5	3	5	5	1	4	5	3	4	5
10	5	3	2	5	2	2	2	4	4	2	5	3	5	3	2	2	5	3	2	3
11	3	2	2	5	2	1	1	4	5	3	5	3	3	3	2	4	5	3	3	3
12	5	4	2	5	2	2	2	4	5	4	5	2	5	3	2	2	5	2	2	3
13	2	2	2	5	4	3	3	4	4	4	5	4	3	3	3	3	5	4	3	5
14	3	3	4	5	5	5	1	4	5	4	5	4	5	3	5	1	5	4	1	5
15	4	4	4	4	5	3	2	5	4	4	4	4	5	3	3	2	4	4	2	5
16	4	3	4	4	5	3	2	5	4	4	4	2	4	4	3	2	4	2	2	5
17	4	2	3	2	5	2	3	5	4	4	4	3	3	4	2	3	4	3	3	5
18	5	4	3	3	4	3	3	5	4	4	4	3	3	4	3	3	4	3	3	5
19	6	2	4	3	4	4	2	4	5	5	5	5	3	4	4	2	5	5	2	5
20	5	5	3	4	4	4	3	4	4	5	4	5	5	4	4	3	4	5	3	3
21	5	5	4	4	4	3	3	4	4	5	4	5	3	4	3	3	4	5	3	3
22	3	5	5	5	4	1	2	4	4	5	4	4	3	4	1	2	4	4	2	4
23	4	3	5	5	4	2	2	5	3	5	4	4	5	3	2	2	4	4	2	3
24	3	3	4	5	2	2	3	5	4	3	4	3	5	3	2	3	4	3	3	3
25	3	4	4	5	3	1	2	5	4	4	4	4	5	3	1	2	4	4	2	4
26	3	4	2	3	3	3	2	4	3	4	5	4	5	3	3	2	5	4	2	4

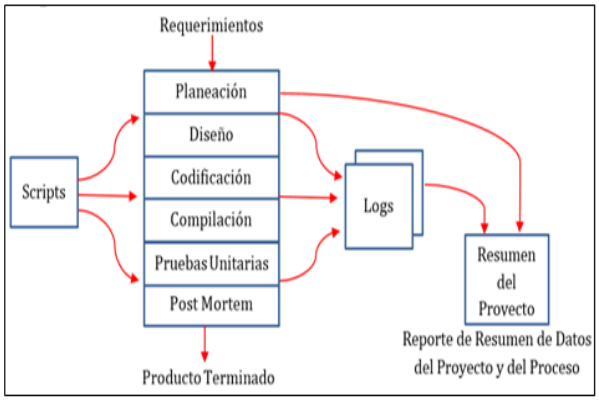
ANEXO 04: CAPACITACIONES

Presentación de Diapositivas para la capacitación hacia los estudiantes

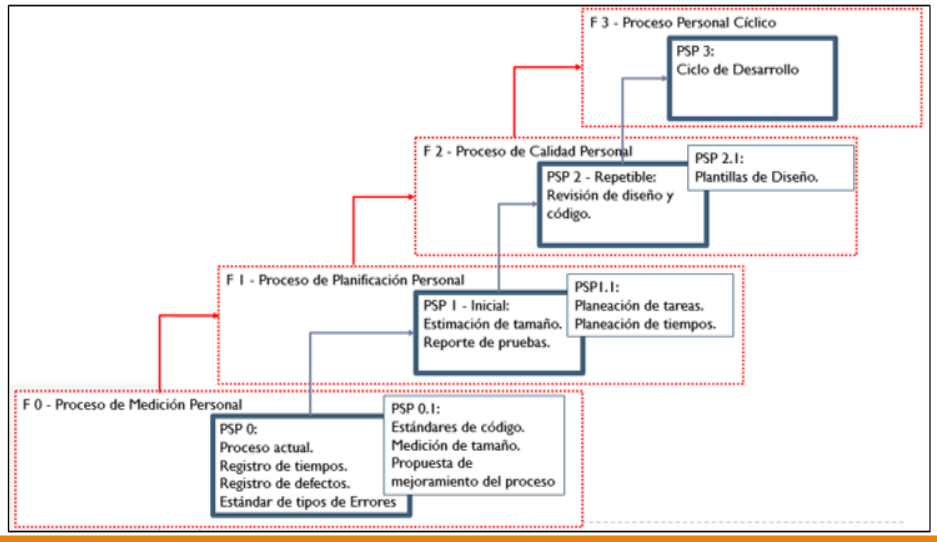
Anexo 4.A: Capacitación sobre PSP



Estructura



Niveles del PSP

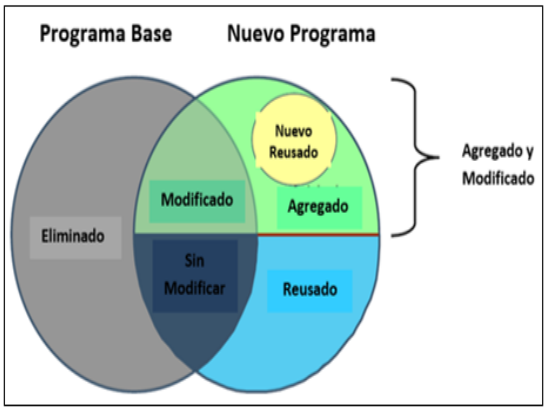


La importancia de los Datos

Lo que se resalta de PSP es el uso de datos históricos para analizar y mejorar el rendimiento del proceso. La recopilación de datos PSP es apoyada por cuatro elementos principales:

- ***Guías (scripts)***: son una descripción de nivel-experto para guiar el proceso.
- **Formularios**: proveen un conveniente y consistente marco de trabajo para recolectar y retener datos.
- **Medidas**: son las medidas de cuantificación del proceso y el producto.
- **Estándares**: entregan una precisa y consistente definición que guía el trabajo, junto con la recopilación y uso de datos.

Líneas de Código - LOC



Ejemplo: si un producto de 100 LOC es usado para una nueva versión, tenía 12 LOC suprimido, 23 LOC agregado, 5 LOC modificado y 3 LOC reutilizado, el LOC nuevo y cambiante sería:

LOC = Agregado + Modificado

→ 28 = 23 + 5.

El tamaño total sería:

Total LOC = Base - Suprimido + Agregado + Reutilizado

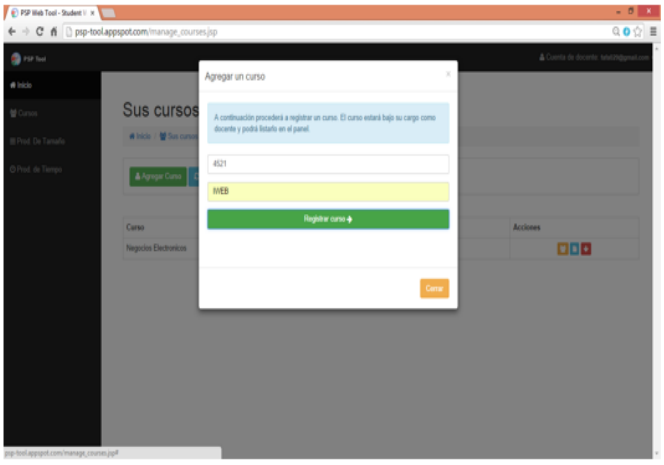
LOC Total = 100 - 12 + 23 + 3 = 114 LOC

Anexo 4.B: Capacitación de la Herramienta Cloud

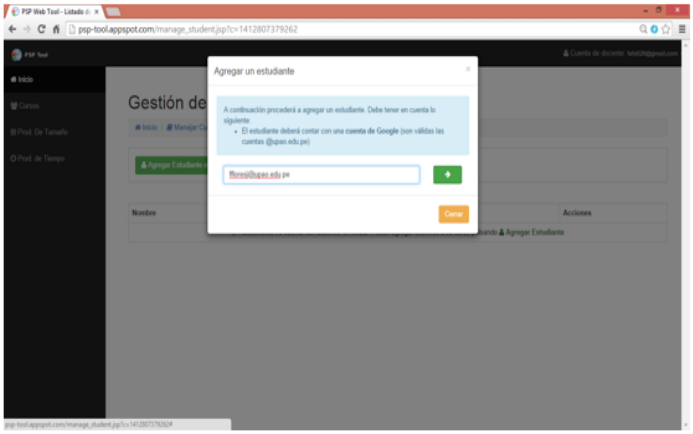
USANDO PSP A TRAVES DE PSP-TOOL

PARA EL DOCENTE

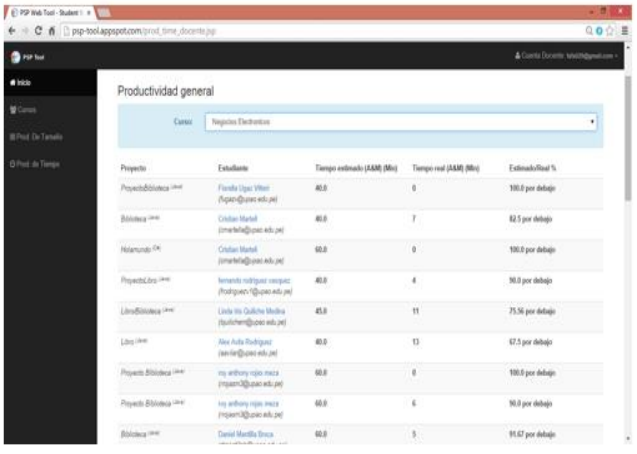
Agregar un Curso



Agregar Alumnos



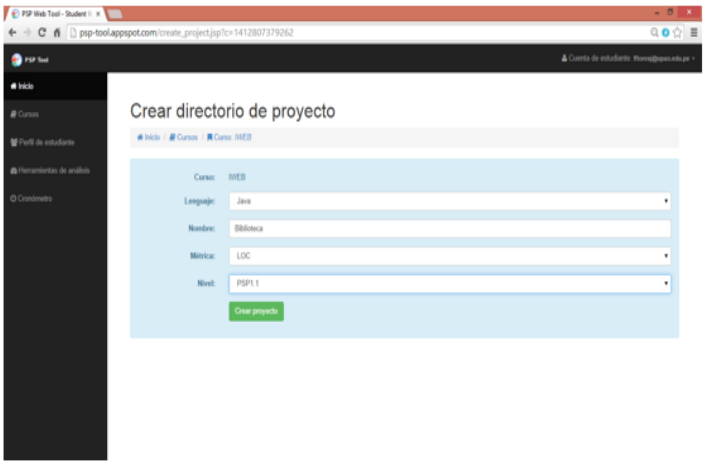
Ver Productividad



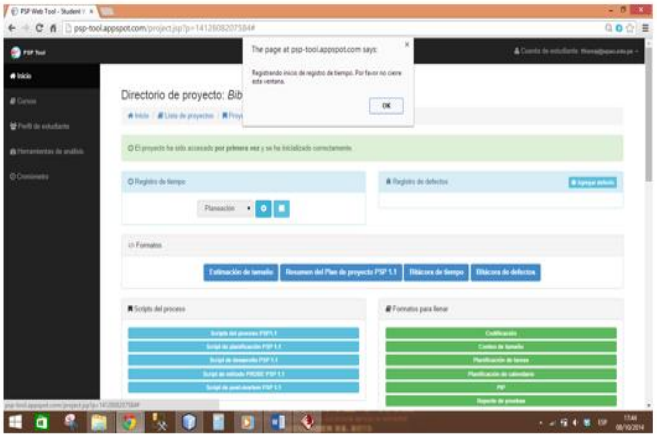
USANDO PSP A TRAVES DE PSP-TOOL

PARA EL ESTUDIANTE

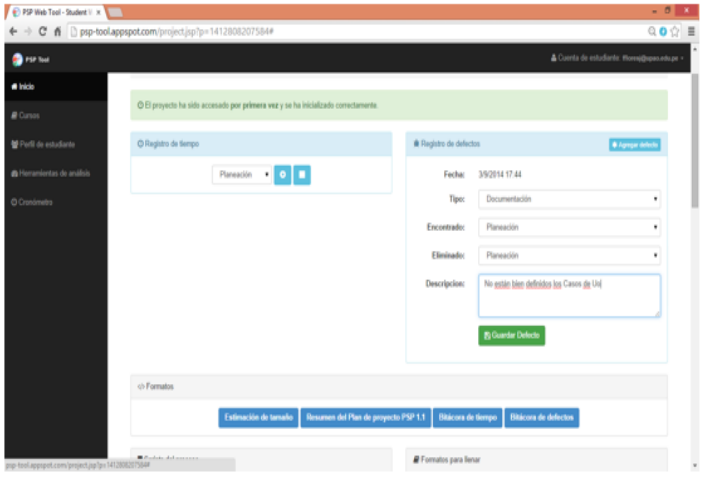
Crear Proyecto por Curso



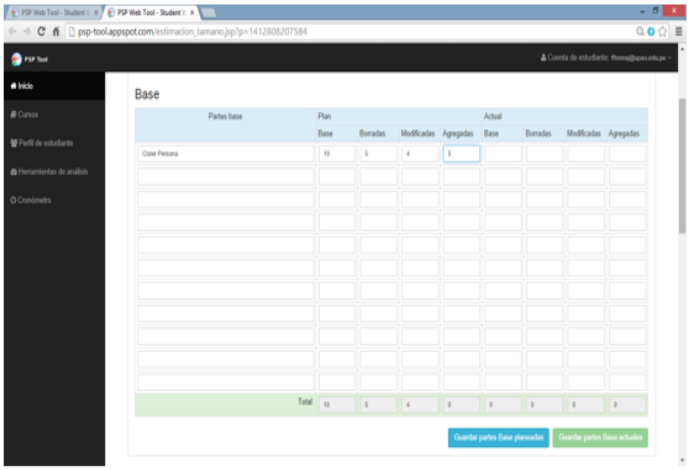
Registrar Tiempo



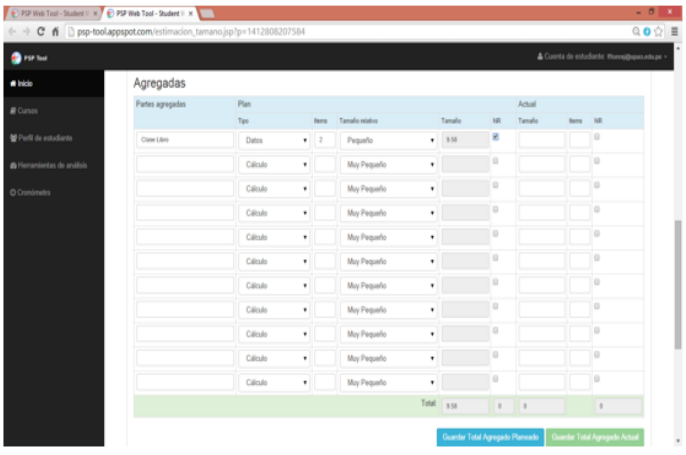
Registrar Defecto



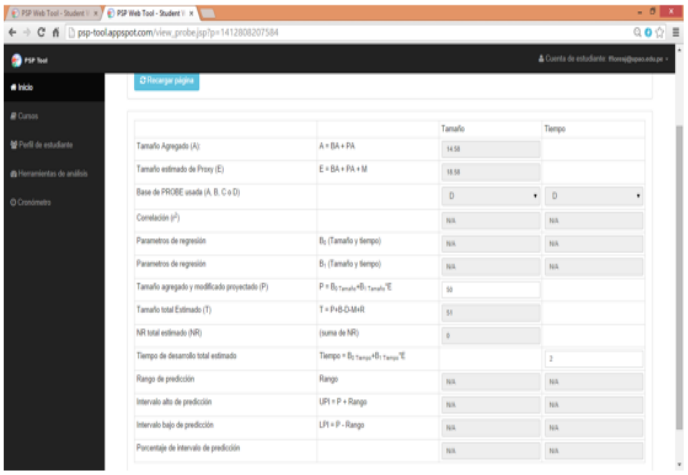
Registrar Partes Base



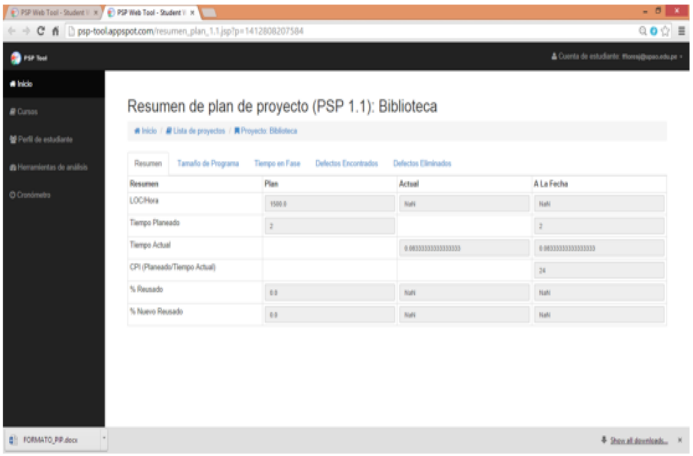
Registrar Partes Agregadas



Calcular Método PROBE



Ver Resumen del Proyecto



ANEXO 05: FORMATOS PSP

Anexo 5.A: PSP 0 - Resumen del Plan del Proyecto

Student		Date	
Program		Program #	
Instructor		Language	

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning				
Design				
Code				
Compile				
Test				
Postmortem				
Total				

Defects Injected	**Actual**	**To Date**	**To Date %**
Planning			
Design			
Code			
Compile			
Test			
Total Development			

Defects Removed	**Actual**	**To Date**	**To Date %**
Planning			
Design			
Code			
Compile			
Test			
Total Development			
After Development			

Anexo 5.B: PSP 0 - Tabla Log de Registro de Tiempos

Student _____ Date _____
Program _____ Program # _____
Instructor _____ Language _____

Project	Phase	Start Date and Time	Int. Time	Stop Date and Time	Delta Time	Comments

Anexo 5.C: PSP 0 - Tabla Log de Registro de Defectos

Student _____ Date _____
Program _____ Program # _____
Instructor _____ Language _____

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description: _____

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description: _____

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description: _____

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description: _____

Anexo 5.D: PSP 0.1 - Resumen del Plan del Proyecto

Student		Date	
Program		Program #	
Instructor		Language	

Program Size	***Plan***	***Actual***	***To Date***
Base (B)		***(Measured)***	
Deleted (D)		***(Counted)***	
Modified (M)		***(Counted)***	
Added (A)		***(T – B + D – R)***	
Reused (R)		***(Counted)***	
Added and Modified (A+M)		***(A + M)***	
Total Size (T)		***(Measured)***	
Total New Reusable			

Time in Phase (min.)	***Plan***	**Actual**	**To Date**	**To Date %**
Planning				
Design				
Code				
Compile				
Test				
Postmortem				
Total				

Defects Injected	**Actual**	**To Date**	**To Date %**
Planning			
Design			
Code			
Compile			
Test			
Total Development			

Defects Removed	**Actual**	**To Date**	**To Date %**
Planning			
Design			
Code			
Compile			
Test			
Total Development			
After Development			

Anexo 5.E: PSP 0.1 - Tabla Propuesta de Mejora de Proceso (PIP)

Student		Date	
Program		Program #	
Instructor		Language	

Problem Description

Briefly describe the problems that you encountered.

Proposal Description

Briefly describe the process improvements that you propose.

Other Notes and Comments

Note any other comments or observations that describe your experiences or improvement ideas.

Anexo 5.F: PSP 1 - Resumen del Plan del Proyecto

Student		Date	
Program		Program #	
Instructor		Language	

Summary	***Plan***	***Actual***	***To Date***
Size/Hour			

Program Size	***Plan***	**Actual**	**To Date**
Base (B)	***(Measured)***	(Measured)	
Deleted (D)	***(Estimated)***	(Counted)	
Modified (M)	***(Estimated)***	(Counted)	
Added (A)	***(A+M – M)***	(T – B + D – R)	
Reused (R)	***(Estimated)***	(Counted)	
Added and Modified (A+M)	***(Projected)***	(A + M)	
Total Size (T)	***(A+M + B – M – D + R)***	(Measured)	
Total New Reusable			
Estimated Proxy Size (E)			

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning				
Design				
Code				
Compile				
Test				
Postmortem				
Total				

Defects Injected	**Actual**	**To Date**	**To Date %**
Planning			
Design			
Code			
Compile			
Test			
Total Development			

Defects Removed	**Actual**	**To Date**	**To Date %**
Planning			
Design			
Code			
Compile			
Test			
Total Development			
After Development			

Anexo 5.G: PSP 1 - Plantilla de Reporte de Pruebas

Student		Date	
Program		Program #	
Instructor		Language	

Test Name/Number	
Test Objective	
Test Description	
Test Conditions	
Expected Results	
Actual Results	
Test Name/Number	
Test Objective	
Test Description	
Test Conditions	
Expected Results	
Actual Results	

Anexo 5.H: PSP 1 - Plantilla de Estimación de Tamaño

Student _____ Date _____
Program _____ Program # _____
Instructor _____ Language _____
Size Measure _____

Base Parts	Estimated Base	Deleted	Modified	Added
Total	B	D	M	BA

Base Parts	Actual Base	Deleted	Modified	Added
Total				

Parts Additions	Estimated Type	Items	Rel. Size	Size*	Actual Size*	Items
Total			PA			

Reused Parts	Estimated Size	Actual Size
Total	R	

PROBE Calculation Worksheet (Added and Modified)

		Size	Time
Added size (A):	A = BA+PA		
Estimated Proxy Size (E):	E = BA+PA+M		
PROBE estimating basis used: (A, B, C, or D)			
Correlation: (R^2)			

Regression Parameters:	β_0 Size and Time		
Regression Parameters:	β_1 Size and Time		
Projected Added and Modified Size (P):	$P = \beta 0_{size} + \beta 1_{size} * E$		
Estimated Total Size (T):	T = P + B - D - M + R		
Estimated Total New Reusable (NR):	sum of * items		
Estimated Total Development Time:	$Time = \beta 0_{time} + \beta 1_{time} * E$		
Prediction Range:	Range		
Upper Prediction Interval:	UPI = P + Range		
Lower Prediction Interval:	LPI = P - Range		
Prediction Interval Percent:			

Anexo 5.I: PSP 1.1 - Resumen del Plan del Proyecto

Student _____ Date _____
Program _____ Program # _____
Instructor _____ Language _____

Summary	**Plan**	**Actual**	**To Date**
Size/Hour			
Planned Time			.
Actual Time			
CPI (Cost-Performance Index)			(Planned/Actual)
% Reused			
% New Reusable			

Program Size	**Plan**	**Actual**	**To Date**
Base (B)	(Measured)	(Measured)	
Deleted (D)	(Estimated)	(Counted)	
Modified (M)	(Estimated)	(Counted)	
Added (A)	(A+M – M)	(T – B + D – R)	
Reused (R)	(Estimated)	(Counted)	
Added and Modified (A+M)	(Projected)	(A + M)	
Total Size (T)	(A+M + B – M – D + R)	(Measured)	
Total New Reusable			
Estimated Proxy Size (E)			

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning				
Design				
Code				
Compile				
Test				
Postmortem				
Total				

Defects Injected	Actual	To Date	To Date %
Planning			
Design			
Code			
Compile			
Test			
Total Development			

Defects Removed	Actual	To Date	To Date %
Planning			
Design			
Code			
Compile			
Test			
Total Development			
After Development			

Anexo 5.J: PSP 1.1 - Plantilla de Planificación de Tareas

Student _____ Date _____
Program _____ Program # _____
Instructor _____ Language _____

Task			Plan							Actual		
Program/Part	Phase	Task Name	Task Hours	Cumulative Task Hours	Week Due	Week	Week Predicted	Planned Value (PV)	Cumulative PV	Task Hours	Cumulative EV	Week
Totals												

Anexo 5.K: PSP 1.1 - Plantilla de Planificación de Calendario

Student _____ Date _____
Program _____ Program # _____
Instructor _____ Language _____

		Plan			Actual				Predicted
Week No.	Date	Schedule Hours	Cumulative Schedule Hours	Cumulative Planned Value	Schedule Hours	Cumulative Schedule Hours	Week Earned Value	Cumulative Earned Value	Cumulative Predicted Earned Value