

**UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS**

**Facultad 3**



# **Categorización de profesionales basado en información curricular.**

**Trabajo final presentado en opción al título de Ingeniero en Ciencias  
Informáticas**

**Autores:**

Laura Beatriz Martínez Rodríguez

Pablo Antonio Moreno García

**Tutor:**

Ing. Zénel Reyes Pérez

**La Habana, junio de 2015**

**“Año 57 de la Revolución”**

## **DECLARACIÓN DE AUTORÍA**

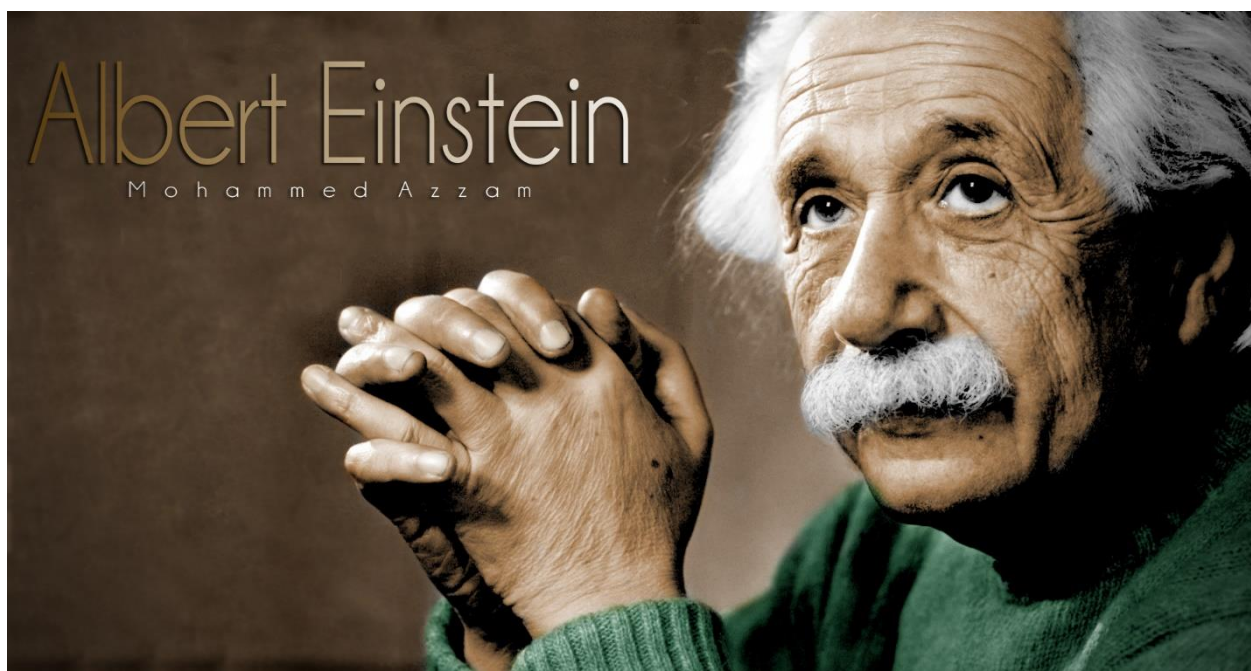
Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los \_\_\_\_\_ días del mes de \_\_\_\_\_ del año \_\_\_\_\_.

\_\_\_\_\_  
Laura Beatriz Martínez Rodríguez

\_\_\_\_\_  
Pablo Antonio Moreno García

\_\_\_\_\_  
Ing. Zénel Reyes Pérez



*Una nueva forma de pensar es esencial si la  
humanidad quiere sobrevivir y ascender a niveles  
más altos*

*Pablo:*

*Hoy después de 5 maravillosos y difíciles años, prácticamente llegamos al final de un camino bien bonito que fue la Universidad. Conocimos y vivimos experiencia incluso hasta el último día de estudio que para mí en lo particular fue el día 9 de junio 2015, cuando se terminó de ajustar el documento que recoge los esfuerzos de todo un año para lograr el objetivo planteado y por el cual quiero agradecerle a:*

*Mi familia por estar siempre ahí para mí, en los momentos difíciles y en los menos difíciles por apoyarme y por permitirme estudiar en esta universidad donde conocí a una persona muy especial que la quiero hoy con la vida y le agradezco todos los detalles que ha tenido conmigo, por escucharme, por entenderme y por lograr sacar lo mejor de mí incluso en los momentos más inesperados. A mi compañera de tesis por confiar en mí y saber salir adelante incluso cuando los obstáculos fueron bastante complejos. A mi grupo porque son únicos, al laboratorio de producción que se montó en el edificio 8 en el 3er paso, a mis equipos de desarrollo PAJ y Husky, porque lo que no aprendí en uno lo cogí en el otro, a mi tribunal y tutores, porque en realidad nada en la vida es fácil y al hacer un trabajo con rigor científico como el obtenido mereció la atención de todos. Sin más porque no me alcanzarían las hojas para dar gracias les recuerdo que siempre los voy a querer porque me da la gana.*

*Laura:*

*Agradezco a mi mamá, por ser mi apoyo y guía, a mi papá por todo su sacrificio. A mis abuelos, por ser mis segundos padres. A toda mi familia en general. A mis amigos, por todos estos años compartidos, por las experiencias vividas, que nunca se van a borrar de mi corazón. A mi compañero, por haber puesto su confianza en mí, a pesar de los obstáculos. A mi consultor Alternán, por toda su preocupación, y a mi tutor por toda su ayuda.*

*Pablo:*

*El presente trabajo va dedicado a todas las personas que me ayudaron, que se mantuvieron en pie cuando la tormenta era brava, a todos por estar siempre conmigo muchas gracias y en especial para toda mi familia porque sin ustedes hoy no estuviera aquí. Chiki, espero que entiendas el mensaje y que ya tengas noción para cuando te toque la tuya. Un Beso para todos.*

*Laura:*

*A mis padres, por todo su sacrificio*

*A mis abuelos, por su cariño*

*A mi hermano, porque a partir de ahora comienzan sus estudios*

## RESUMEN

El presente trabajo surge a partir de la necesidad de usar la información curricular como apoyo al proceso de categorización de los profesionales de una entidad. La solución tiene como objetivo extender el Sistema de Gestión Curricular mediante un mantenimiento perfectivo, que incorpore funcionalidades que correspondan con evaluar la producción y permita categorizar a los profesionales a partir de la información registrada. La productividad científica se calcula utilizando el “Indicador de desempeño en la actividad científico-técnica de los investigadores”, desarrollado en la Universidad de las Ciencias Informáticas. Se utilizan las líneas de investigación previamente definidas por la Universidad con el fin de determinar quiénes están a la vanguardia en el desarrollo de las mismas y de esta manera poder categorizarlos atendiendo a su producción por cada una de ellas. El resultado final es un “Sistema de Gestión Curricular” que incluye entre sus funcionalidades la evaluación de la producción de los profesionales por cada línea de investigación desarrollada. Además permitirá obtener un ranking de investigación para los trabajadores a partir de su información curricular.

## PALABRAS CLAVE

Categorización, Mantenimiento, Indicador de desempeño, Currículum vitae.

## ÍNDICE

<b>INTRODUCCIÓN</b> .....	1
<b>CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA</b> .....	5
<b>Introducción</b> .....	5
<b>1.1 Generalidades del mantenimiento de software</b> .....	5
<b>1.1.1 Mantenimiento de software</b> .....	6
<b>1.1.2 Tipos de mantenimiento</b> .....	6
<b>1.1.3 Leyes del Mantenimiento del Software</b> .....	8
<b>1.1.4 Problemas de mantenimiento</b> .....	9
<b>1.1.5 Actividades del mantenimiento</b> .....	9
<b>1.1.6 Fases de desarrollo en el mantenimiento</b> .....	10
<b>1.2 Indicador de evaluación de la producción científica</b> .....	12
<b>1.3 Metodología de desarrollo de software</b> .....	14
<b>1.4 Herramientas y tecnologías de desarrollo</b> .....	16
<b>1.5.1 Lenguaje de modelado</b> .....	16
<b>1.5.2 Herramientas CASE</b> .....	17
<b>1.5.3 Lenguaje de programación</b> .....	17
<b>1.5.4 Marco de trabajo</b> .....	18
<b>1.5.5 Entorno de desarrollo integrado</b> .....	19
<b>1.5.6 Sistema gestor de bases de datos</b> .....	19
<b>1.5.7 Servidor de aplicaciones</b> .....	20
<b>1.5.8 Control de versiones</b> .....	20
<b>Conclusiones parciales</b> .....	21
<b>Capítulo 2: DISEÑO DE LA PROPUESTA DE SOLUCIÓN</b> .....	22
<b>Introducción</b> .....	22
<b>2.1 Descripción del sistema.</b> .....	22
<b>2.2 Modelo de domino</b> .....	22
<b>2.3 Ingeniería de Requisitos</b> .....	23
<b>2.3.1 Obtención de Requisitos</b> .....	24
<b>2.3.2 Especificación de requisitos funcionales</b> .....	25
<b>2.3.3 Requisitos no funcionales</b> .....	26
<b>2.3.4 Validación de requisitos funcionales</b> .....	27
<b>2.4 Fase de planificación</b> .....	28
<b>2.4.1 Plan de entregas</b> .....	29
<b>2.4.2 Plan de iteraciones</b> .....	29

---

<b>2.5</b>	<b>Fase de diseño</b> .....	30
<b>2.5.1</b>	<b>Arquitectura del sistema</b> .....	30
<b>2.5.2</b>	<b>Patrones arquitectónicos</b> .....	31
<b>2.5.3</b>	<b>Patrones de diseño</b> .....	35
<b>2.5.4</b>	<b>Tarjetas CRC</b> .....	38
<b>2.5.5</b>	<b>Modelo de datos relacional</b> .....	39
<b>2.6</b>	<b>Fase de implementación</b> .....	41
<b>2.7</b>	<b>Estándares de codificación</b> .....	41
<b>2.8</b>	<b>Diagrama de despliegue</b> .....	42
<b>2.9</b>	<b>Diagrama de componentes</b> .....	42
	<b>Conclusiones parciales</b> .....	43
<b>CAPÍTULO 3. VALIDACIÓN DE LA PROPUESTA</b> .....		44
	<b>Introducción</b> .....	44
<b>3.1</b>	<b>Validación del diseño</b> .....	44
<b>3.1.1</b>	<b>Tamaño operacional de las clases (TOC)</b> .....	44
<b>3.1.2</b>	<b>Relaciones entre clases (RC)</b> .....	46
<b>3.2</b>	<b>Pruebas</b> .....	49
<b>3.2.1</b>	<b>Pruebas Unitarias</b> .....	49
<b>3.3</b>	<b>Categorización de profesionales</b> .....	52
	<b>Conclusiones parciales</b> .....	54
<b>CONCLUSIONES</b> .....		56
<b>RECOMENDACIONES</b> .....		57
<b>REFERENCIAS BIBLIOGRÁFICAS</b> .....		58
<b>ANEXOS</b> .....		62



**ÍNDICE DE FIGURAS**

Figura 1. Fases del proceso de mantenimiento.....	10
Figura 2. Modelo de dominio .....	23
Figura 3. Prototipo de interfaz. Ponderar perfil de usuario por líneas de investigación .....	28
Figura 4. Prototipo de interfaz. Calcular indicador de desempeño para cada evidencia .....	28
Figura 5. Prototipo de interfaz. Mostrar categorizaciones automáticas .....	28
Figura 6. Organización propuesta por Symfony.....	32
Figura 7. Modelo .....	33
Figura 8. Vista.....	34
Figura 9. Controlador .....	34
Figura 10. Patrón de diseño experto.....	35
Figura 11. Patrón de diseño controlador.....	36
Figura 12. Patrón de diseño creador .....	36
Figura 13. Patrón de diseño alta cohesión.....	37
Figura 14. Tarjeta CRC de la clase Recomendación .....	38
Figura 15. Tarjeta CRC de la clase Notificación .....	39
Figura 16. Modelo Entidad-Relación general.....	40
Figura 17. Entidades con las que interactúa el sistema.....	41
Figura 18. Diagrama de despliegue.....	42
Figura 19. Diagrama de componentes.....	43
Figura 20. Comportamiento del atributo responsabilidad de la métrica TOC .....	45
Figura 21. Complejidad .....	45
Figura 22. Comportamiento del atributo complejidad de la métrica TOC .....	45
Figura 23. Comportamiento del atributo reutilización de la métrica TOC .....	46
Figura 24. Comportamiento de las dependencias entre clases.....	47
Figura 25. Comportamiento del atributo acoplamiento en RC.....	47
Figura 26. Comportamiento del atributo complejidad en CR.....	48
Figura 27. Comportamiento del atributo cantidad de pruebas en CR.....	48
Figura 28. Comportamiento del atributo reutilización en RC .....	49
Figura 29. Resultados de las pruebas unitarias utilizando PHPUnit.....	50
Figura 30. Resultado de las pruebas de aceptación al término de cada iteración.....	52
Figura 31. Listado de evidencias evaluadas .....	53
Figura 32. Desempeño de un profesional por líneas desarrolladas .....	53
Figura 33. Categorización de profesionales por líneas de investigación.....	54
Figura 34. Análisis de la producción por años .....	54

**ÍNDICE DE TABLAS**

Tabla 1. Consideraciones para calcular el IDEC.....	13
Tabla 2. Comparación entre las metodologías ágiles .....	15
Tabla 3. HU Ponderar perfil por líneas de investigación .....	25
Tabla 4. HU Mostrar categorizaciones automáticas.....	25
Tabla 5. HU Calcular indicador de desempeño para cada evidencia .....	26
Tabla 6. Plan de entregas .....	29
Tabla 7. Distribución de iteraciones por historia de usuario.....	30
Tabla 8. Tarjeta CRC de la clase Evidencia .....	38
Tabla 9. Responsabilidad .....	45
Tabla 10. Reutilización .....	46
Tabla 11. Dependencia entre clases .....	47
Tabla 12. Acoplamiento.....	47
Tabla 13. Complejidad de mantenimiento .....	48
Tabla 14. Cantidad de pruebas .....	48
Tabla 15. Reutilización .....	49
Tabla 16. Ponderación de la variable artículo para autor principal.....	62
Tabla 17. Ponderación de la variable artículo para otro tipo de autor .....	62
Tabla 18. Ponderación para la variable proyecto para el rol de jefe.....	63
Tabla 19. Ponderación para la variable proyecto para el rol de participante principal.....	63
Tabla 20. Ponderación para la variable proyecto para el rol de colaborador.....	64
Tabla 21. Ponderación para la variable doctorado.....	64
Tabla 22. Ponderación para la variable maestría.....	64
Tabla 23. Ponderación de la variable premio para el autor principal.....	65
Tabla 24. Ponderación de la variable premio para otro tipo de autor .....	65
Tabla 25. Ponderación de la variable introducción de resultados para el autor principal .....	65
Tabla 26. Ponderación de la variable introducción de resultados para otro tipo de autor.....	66
Tabla 27. Ponderación de la variable registros para el autor principal .....	66
Tabla 28. Ponderación de la variable registros para otro tipo de autor .....	66
Tabla 29. Ponderación de la variable evento para el autor principal .....	67
Tabla 30. Ponderación de la variable evento para otro tipo de autor .....	67
Tabla 31. Ponderación de la variable moneda nacional para el autor principal.....	67
Tabla 32. Ponderación de la variable moneda nacional para otro tipo de autor .....	68
Tabla 33. Ponderación de la variable moneda convertible para el autor principal.....	68
Tabla 34. Ponderación de la variable moneda convertible para otro tipo de autor .....	68
Tabla 35. HU Ponderar perfil de usuario por año.....	69

---

Tabla 36. HU Ponderar perfil histórico de usuario .....	69
Tabla 37. HU Generar categorización a partir del IDEC .....	70
Tabla 38. HU Generar categorización a partir de asignaturas impartidas .....	70
Tabla 39. HU Mostrar usuarios que se siguen .....	71
Tabla 40. HU Notificar a una usuario sobre cambios en un perfil .....	71

## INTRODUCCIÓN

*El problema de fondo es que el software contiene errores, y cuanto más software hay, más errores contienen.*

*Andrew Tanenbaum*

La productividad y la calidad de la producción científica constituyen una ocupación y una preocupación tanto de los propios científicos como de los decisores y gestores de la ciencia (1).

La creación de nuevo conocimiento, como base de la actividad de investigación, se mide a través de los productos y efectos que generen. El producto de la investigación adquiere diferentes formas y varía según cada disciplina, desde los artículos en revistas científicas, libros, presentaciones en eventos y otros productos relacionados con la posibilidad de aplicación y transferencia del conocimiento. Por su parte el efecto, también conocido como resultado, se puede manifestar como producción de graduados de alta calidad, innovaciones tecnológicas, relaciones internacionales de intercambio, entre otros (2). El producto de la actividad de investigación es generalmente cuantificable, en cambio los resultados son más difíciles de cuantificar. Tradicionalmente se utiliza la bibliometría <sup>1</sup>para medir la producción tanto de las instituciones como de los profesionales, pero esta por sí sola no basta para medir la totalidad de los productos y efectos.

La evaluación de la producción científica permite cuantificar el rendimiento de un investigador o de un grupo de investigadores, de una colección de artículos seleccionados, de una revista científica, o de una institución dedicada a la investigación. Este proceso implica la utilización de una serie de plataformas, herramientas e indicadores que permiten analizar y clasificar la producción científica conforme a criterios de evaluación nacionales e internacionales (índice de impacto, citas e indicadores específicos) (3).

Con la evaluación de la producción se obtiene una medida del desempeño productivo de los investigadores. Esta información es utilizada en muchas instituciones para categorizar a sus profesionales en dependencia de su realización científica y a partir de ello determinar las compensaciones a otorgar.

El término categorizar, según la Real Academia de la Lengua Española (RAE) se refiere a organizar o clasificar por categorías y la categoría es definida como la condición social de unas personas

---

<sup>1</sup> Bibliometría: ciencia que aplica métodos matemáticos y estadísticos a la literatura científica y a los autores que la producen, con el objetivo de estudiar y analizar la actividad científica. Los instrumentos utilizados para medir los aspectos de este fenómeno social son los indicadores bibliométricos, medidas que proporcionan información sobre los resultados de la actividad científica en cualquiera de sus manifestaciones.

respecto de las demás (4). Otras fuentes definen categoría como cada una de las jerarquías establecidas en una profesión o carrera (5).

Para la presente investigación se entiende como categoría al grado de jerarquía dentro de un orden establecido por el puesto que ocupa un determinado profesional en una institución relacionado con su evaluación en la producción científica.

El modo más utilizado para medir la producción es la literatura (6), evaluando su presencia en bases de datos bibliográficas internacionales, pero existen otros modos de evaluación como: la participación en eventos, los resultados y premios obtenidos, tesis tutoradas, entre otras, que no son tomados en cuenta y se pueden encontrar en el currículum vitae de los profesionales. Para medir estos elementos se debe utilizar un indicador que los tome en cuenta a la hora de evaluar, además de contar con un sistema que gestione la información curricular de los profesionales. Realizando un mantenimiento de software se puede incluir la evaluación de la producción al sistema que gestione la información curricular.

Una simple definición del mantenimiento es: el proceso de registro y seguimiento de problemas y la petición y gestión de respuestas. Un problema no necesariamente implica un defecto del software. Los problemas aparecen simplemente porque el comportamiento esperado del sistema no coincide con el comportamiento actual (7).

En la Facultad 3 de la Universidad de las Ciencias Informáticas (UCI) existe un sistema informático para la gestión del currículum vitae. Aunque esta herramienta permite al usuario gestionar libremente su información curricular, no facilita la búsqueda de otros usuarios que tengan conocimientos en una línea de investigación específica ni desarrolla ránkines que categorizan a los profesionales de acuerdo a su producción científica en una línea de investigación.

En muchas oportunidades el investigador inexperto que recurre a este sistema, se pierde entre un conjunto evidencias que generalmente no son esenciales ni están acorde a la investigación que este desea realizar, lo que ocasiona que pierda su tiempo en una búsqueda sin resultados satisfactorios. Conocer a las figuras más destacadas desde el punto de vista del resultado científico ayudaría a los investigadores nuevos a establecer estrategias para adquirir nuevos conocimientos y contrastar sus ideas de investigación. Los directivos administrativos de igual forma obtendrían ventajas del sistema de este tipo ya que podrían contar con información fidedigna de la producción científica de los investigadores de la Universidad.

Partiendo de la problemática antes descrita, se presenta como **problema a resolver**: ¿Cómo categorizar a los profesionales de la Facultad 3 de acuerdo a su información curricular?

Se define como **objeto de estudio** la gestión de software, concibiendo como **campo de acción** el mantenimiento perfectivo de software.

En respuesta al problema, se toma como **objetivo general** extender el Sistema de Gestión Curricular de la Facultad 3 de manera tal que incorpore la categorización de los profesionales de acuerdo a su información curricular.

Los **objetivos específicos** que se definen son:

- Fundamentar los conceptos y características del mantenimiento perfectivo de software mediante un estudio bibliográfico, haciendo hincapié en la metodología y las herramientas a utilizar en su desarrollo.
- Incorporar las funcionalidades asociadas a la evaluación y categorización de investigadores teniendo en cuenta su producción científica.
- Validar la solución mediante la aplicación de un caso de estudio con un conjunto reducido de investigadores en una línea de investigación.

Se prevé como **posible resultado**: que el Sistema de Gestión Curricular incorpore la categorización de los profesionales de acuerdo a su información curricular. **Defendiendo la idea** de que si se realiza un mantenimiento perfectivo al Sistema de Gestión Curricular de la Facultad 3, teniendo en cuenta la información curricular de los profesionales entonces se podrá categorizar a los mismos en el sistema.

Para el desarrollo satisfactorio de la investigación, se emplean los **métodos teóricos**: histórico-lógico, analítico-sintético e inductivo-deductivo, así como, de carácter **empírico**, la entrevista.

**Histórico-lógico**: permitió realizar un análisis de los fundamentos teóricos del mantenimiento de software, las diferentes fases por las que está compuesto y cómo se relaciona cada una.

**Analítico-sintético**: se utilizó en el análisis de la documentación existente relacionada con el mantenimiento de software, donde se analizaron las leyes planteadas, los tipos de mantenimiento y documentos relacionados para luego sintetizar toda la información y adaptarla a la solución.

**Inductivo-deductivo**: permitió ir de las características generales a las particulares y viceversa aplicando los conceptos fundamentales relacionados con el mantenimiento de software, logrando un entendimiento propio sobre los mismos.

**Entrevista**: se utilizó con el propósito de conocer cuáles son las características del mantenimiento a desarrollar atendiendo a las necesidades de la Facultad 3.

Para un mejor entendimiento del documento, a continuación se describe brevemente cada uno de los capítulos en que se encuentra estructurado:

- **Capítulo I. Fundamentación teórica**: se definen los principales conceptos asociados al mantenimiento de software, sus características, los tipos en que se puede presentar, las

fases en que está compuesto. Además se describen las herramientas y metodología empleadas para el mantenimiento de la aplicación.

- **Capítulo II. Diseño de la propuesta de solución:** se describen las características esenciales que posee el sistema en su conjunto. Se identifican las funcionalidades a partir de las historias de usuarios y los requerimientos no funcionales. De forma general, se generan los artefactos y diagramas correspondientes a las fases de la metodología seleccionada.
- **Capítulo III. Validación de la propuesta:** en este capítulo se presentan las métricas utilizadas para validar el diseño, así como los métodos de prueba aplicados a las funcionalidades que se adicionaron.

## CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

### Introducción

*“La práctica debe siempre ser edificada sobre la buena teoría.”*

*Leonardo Da Vinci*

En el siguiente capítulo se hace referencia a los principales conceptos asociados al dominio del problema que permiten un mayor entendimiento sobre el tema a tratar. Se brinda el resultado de un análisis investigativo acerca de los tipos de mantenimiento existentes y sus características. Además se realiza una exposición de la metodología que regirá el proceso de desarrollo del software, el lenguaje de modelado que esta utiliza, el lenguaje de programación y marcos de trabajo, así como el sistema gestor de bases de datos empleado.

### 1.1 Generalidades del mantenimiento de software

El desarrollo de un software no termina cuando este es entregado, sino que continúa mientras el sistema esté en uso (8). Luego de desplegar un software, surgen nuevos requerimientos y en ocasiones los requerimientos que existen cambian. Esto provoca que algunas partes del software tengan que modificarse para corregir errores encontrados en su funcionamiento, adaptarlo a nuevas plataformas o adicionarle nuevas funcionalidades. El mantenimiento es la etapa del ciclo de vida de software que se encarga de resolver estas cuestiones.

Mantener y mejorar el software para enfrentar errores descubiertos y nuevos requisitos, puede llevar más tiempo incluso que el desarrollo inicial del software. La mayoría de las actividades desarrolladas en el ciclo de vida del software tiene que ver con dar mantenimiento (7). Una pequeña parte del mantenimiento se encarga de arreglar errores, mientras que la otra parte es dedicada a incluir nuevas funcionalidades.

Las actividades relacionadas con el mantenimiento se pueden clasificar en:

- **Mejora:** Cuando el sistema está en explotación, a veces es necesario introducir mejoras.
- **Ampliación:** En ocasiones es necesario introducir nuevas funcionalidades (requisitos) en el producto de software cuando ya está en marcha.
- **Corrección de errores:** Los errores suelen aparecer con frecuencia en el software cuando está en explotación (7).

El mantenimiento de software es un proceso organizado que permite que los sistemas evolucionen y puedan continuar funcionando de acuerdo a las necesidades de los usuarios.



### 1.1.1 Mantenimiento de software

El mantenimiento de software según la IEEE es la modificación de un producto software después de la entrega para corregir fallos, para mejorar el rendimiento u otros atributos, o para adaptar el producto a un entorno modificado (9). Esta definición asume que las actividades de mantenimiento de un producto de software comienzan luego de este ha sido entregado.

Por su parte Pressman dice que la fase de mantenimiento se centra en el cambio que va a asociado a la corrección de errores, a las adaptaciones requeridas a medida que evoluciona el entorno del software, y a cambios debidos a las mejoras producidas por los requisitos cambiantes del cliente (10).

Sommerville plantea que el mantenimiento implica corregir errores no descubiertos en las etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema y resaltar los servicios del sistema una vez que se descubren nuevos requerimientos (8).

Partiendo de los conceptos asociados al mantenimiento de software se decide utilizar la definición establecida por AESIS<sup>2</sup>. El soporte a una aplicación no consiste únicamente en “mantener las luces encendidas” (es decir, que siga funcionando), sino que además, es necesario ir incorporando todas las necesidades que van surgiendo con la propia dinámica del negocio, buscando que la aplicación garantice las operaciones (11).

Atendiendo a los objetivos que se desean modificar en el software, se puede encontrar varias clasificaciones del mantenimiento.

### 1.1.2 Tipos de mantenimiento

Según el estándar IEEE 1219 existen tres categorías de mantenimiento de software en dependencia de las demandas de los usuarios (9). Estos son:

- Mantenimiento Adaptativo
- Mantenimiento Correctivo
  - ✓ Mantenimiento de emergencia
- Mantenimiento Perfectivo
  - ✓ Mantenimiento Preventivo

Pressman coincide con estas clasificaciones, distinguiendo al mantenimiento preventivo como una clasificación independiente del perfectivo.

---

<sup>2</sup> AESIS es una empresa de Software y computación en la nube, especializada en Google Apps, movilidad, ERP, CRM, desarrollos empresariales y su integración con sistemas tradicionales.

### **Mantenimiento Correctivo**

Este tipo de mantenimiento ofrece soluciones a las incidencias surgidas en los distintos sistemas. Estas incidencias se analizan y solventan atendiendo principalmente a los aspectos que implica: **Análisis de la aplicación:** problemas identificados como errores del código de la aplicación.

**Análisis de la funcionalidad:** problemas detectados a raíz de funcionalidades incorrectas o mal implementadas, así como las distintas implicaciones que ese fallo puede tener en otras aplicaciones incluidas en el servicio.

**Análisis de base de datos:** problemas derivados de errores en la base de datos, coherencia de los datos, etc.

El principal objetivo del Mantenimiento Correctivo es subsanar los fallos detectados en el sistema y asegurar que éstos no han producido incoherencias en la integridad en los datos.

### **Mantenimiento Preventivo**

Consiste en anticipar fallos del sistema en cualquiera de los niveles analizados para ser pro activo en la resolución de los mismos y, por tanto, mitigarlos y que éstos fallos nunca lleguen a suceder. Se basa en la modificación del producto Software sin alterar las especificaciones del mismo, para mejorar las propiedades de mantenimiento del producto y facilitar así las futuras tareas de mantenimiento.

### **Mantenimiento Perfectivo**

No estará únicamente enfocado a mejorar técnicamente una solución, sino que también incluye un proceso continuo de optimización a nivel funcional y de procesos. Este mantenimiento se enfoca en:

- La optimización constante del rendimiento de las aplicaciones mediante análisis técnicos.
- La adaptación de las aplicaciones a las nuevas necesidades del cliente en función de los análisis funcionales.
- La detección de posibles puntos a mejorar en el diseño y uso de las bases de datos mediante el análisis de la base de datos.

Por lo tanto, el principal objetivo del mantenimiento perfectivo es llevar a cabo las tareas y procesos necesarios para identificar aquellos puntos susceptibles de mejora, aportando las soluciones óptimas y haciendo efectivos esos cambios en las aplicaciones.

Entre otros, los procesos necesarios que se ponen en marcha para la consecución de estos objetivos son:

- Aseguramiento del rendimiento óptimo de los servicios del cliente.
- Análisis de posibles cambios de las necesidades del cliente, para aportar soluciones funcionales a sistemas existentes o a nuevos servicios.
- Análisis pro activo de puntos a mejorar o perfeccionar

Está dividido en dos tipos diferentes:

- **Mantenimiento de Ampliación:** incorporación de nuevas funcionalidades.
- **Mantenimiento de Eficiencia:** mejora de la eficiencia de ejecución.

### **Mantenimiento Adaptativo**

Es la modificación de un producto software, después de su puesta en producción, para mantener operativo un programa mientras se realiza un cambio en el entorno de producción. Tiene por objetivo la modificación de un programa debido a cambios en el entorno, cambios en el hardware o en el software, en el que se ejecuta.

Tiene por objetivo la modificación de un programa debido a cambios en el entorno, bien cambios en el hardware o en el software, en el que se ejecuta.

### **Mantenimiento de Emergencia**

Es un mantenimiento correctivo realizado sin planificación previa, utilizado para mantener operativo el sistema (7).

Para realizar el mantenimiento es necesario ejecutar tareas o actividades que garanticen la efectividad del proceso.

Atendiendo a las características de los diferentes tipos de mantenimiento y teniendo en cuenta las necesidades del cliente, se decide realizar un mantenimiento perfectivo, ya que se requiere ampliar el Sistema de Gestión Curricular incluyendo nuevas funcionalidades.

Además de las actividades analizadas anteriormente el mantenimiento de software debe regirse por leyes: las leyes de Lehman. Estas son invariantes y aplicables.

#### **1.1.3 Leyes del Mantenimiento del Software**

- **Continuidad del cambio:** un programa utilizado en un entorno del mundo real debe cambiar si no quiere dejar de ser usado. Esto se debe a que surgen nuevas funcionalidades, nuevo hardware puede permitir mejoras en el software, se corrigen defectos, se instala en otro sistema operativo/máquina o el software necesita ser más eficiente.
- **Incremento de la Complejidad:** cuando se realizan cambios en un programa la estructura se hace más compleja si no se utilizan técnicas de ingeniería del software.

- **Evolución del programa:** es un proceso autorregulado. Se mantienen las tendencias e invariantes de las propiedades del programa.
- **Conservación de la Estabilidad Organizacional:** la carga que supone el desarrollo de un programa es aproximadamente constante e independiente de los recursos dedicados a lo largo del tiempo de vida del mismo.
- **Conservación de la Familiaridad:** el incremento en el número de cambios introducidos con cada versión es aproximadamente constante (12).

Realizar mantenimiento al software puede traer consigo problemas para el sistema.

#### 1.1.4 Problemas de mantenimiento

- a) Es habitual realizar el mantenimiento de forma ad hoc en un estilo libre del programador. Esto es debido a que no existen o son poco conocidos los métodos, técnicas y herramientas que proporcionan soluciones globales al problema del mantenimiento.
- b) Después de cada cambio los programas tienden a ser menos estructurados. Como consecuencia se produce una documentación desfasada, código que no cumple los estándares, incremento en el tiempo de comprensión de los programas o incremento de los efectos secundarios de los cambios.
- c) Los sistemas que son mantenidos son cada vez más difíciles de cambiar.
- d) Los usuarios participan poco en el desarrollo del software, con el riesgo de que no satisfaga sus necesidades y aumenten los esfuerzos en el mantenimiento.
- e) Problemas de gestión. Existe una visión de que el trabajo de mantenimiento es de una escala inferior al trabajo de desarrollo de software. Se realiza mantenimiento precipitado, no documentado adecuadamente y poco integrado en el código existente (12).

#### 1.1.5 Actividades del mantenimiento

Las actividades de mantenimiento se agrupan en tres categorías funcionales (12):

- **Comprensión del software y de los cambios a realizar (Comprender):** es necesario el conocimiento a fondo de la funcionalidad, objetivos, estructura interna y requisitos del software. Alrededor del 50% de tiempo de mantenimiento se dedica a esta actividad, a consecuencia de lo cual, las herramientas CASE incorporan utilidades que automatizan este tipo de tareas aumentando de manera notable la productividad.
- **Modificación del software (Corregir):** crear y modificar las estructuras de datos, la lógica de procesos, las interfaces y la documentación. Los programadores deben evitar los efectos laterales provocados por sus cambios. Esta actividad representa  $\frac{1}{4}$  del tiempo total de mantenimiento.

- **Realización de pruebas (Comprobar):** realizar pruebas selectivas que nos aseguren la corrección del software.

### 1.1.6 Fases de desarrollo en el mantenimiento

Los procesos de mantenimiento proveen las actividades requeridas así como las entradas/salidas detalladas para esas actividades, que están descritos en los estándares del mantenimiento del software (9).

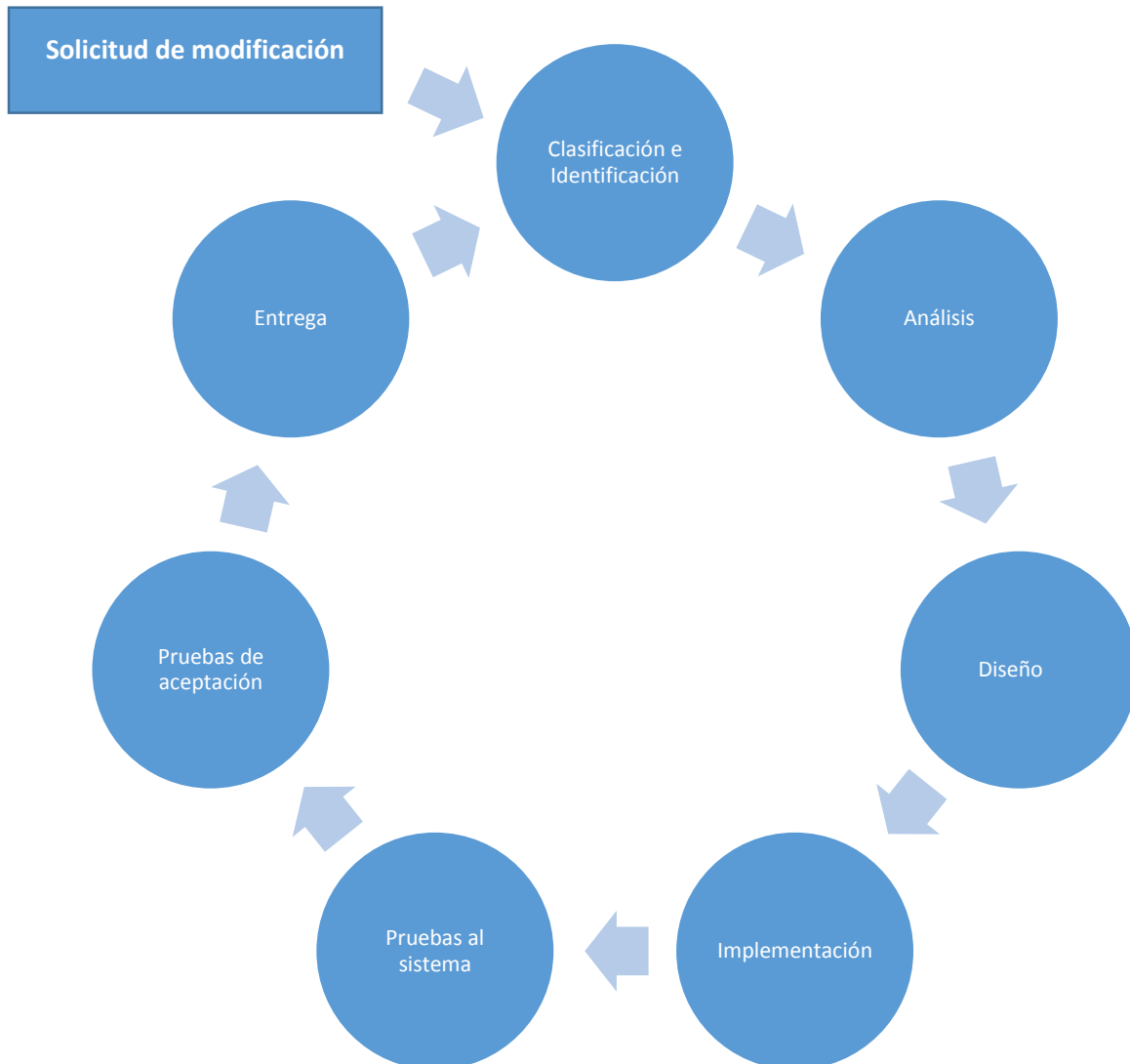


Figura 1. Fases del proceso de mantenimiento.

#### Identificación, clasificación y priorización del problema

En esta fase se identifican, clasifican y asignan una prioridad inicial a las modificaciones del software. Cada Solicitud de Modificación es evaluada para determinar su clasificación y prioridad. La clasificación será identificada según los tipos de mantenimiento: correctivo, adaptativo, perfectivo y de emergencia.

## **Análisis**

En esta fase se estudia la viabilidad y el alcance de las modificaciones, que ya se tienen clasificadas y priorizadas, así como la generación de un plan preliminar de diseño, implementación, pruebas y liberación del software. La información que se utiliza en esta fase proviene del repositorio y de la Solicitud de Modificación validada en la fase anterior, además de la documentación del proyecto y del sistema existente.

## **Diseño**

A partir de toda la documentación existente del proyecto y del sistema que esté en producción, así como todo el código fuente y las bases de datos de la última versión, se realiza el diseño de las modificaciones del sistema en base a la documentación generada en la fase de análisis.

El primer paso en la fase de diseño es identificar los módulos software que van a ser objeto de modificación, con el fin de hacer constar la planificación de tareas y ver la previsión de las mejoras a introducir. Según se avanza en los módulos se realizan las modificaciones oportunas a la documentación de los mismos.

Para las modificaciones a realizar, se generará unos casos de pruebas que incluyan los elementos de seguridad y robustez.

## **Implementación**

Para dirigir apropiadamente el esfuerzo en la fase de implementación es necesario disponer, como en las otras fases, de la documentación actualizada del proyecto y del sistema, del código fuente actual, y los resultados de la fase de diseño. Otros elementos necesarios para el éxito de esta fase:

- Documentación de diseño y requisitos aprobados y controlados.
- Estándar de codificación acordado para ser utilizado por el grupo de mantenimiento.
- Métricas de diseño que podrían ser aplicables en la fase de implementación (podrían clarificar la complejidad del código a desarrollar o mantener).
- Una planificación de desarrollo detallada, incluyendo todas las revisiones de código que se harán y a qué nivel.
- Un conjunto de respuestas a los riesgos identificados en las fases anteriores y que serán aplicables en la fase de pruebas.

## **Pruebas del sistema**

Las pruebas de sistema se realizan sobre un sistema modificado. Deberían ser realizadas por una organización independiente y siempre estar presente el cliente y el usuario final. La organización

responsable de las pruebas del sistema deberá ser independiente de los desarrolladores y diseñadores del software, pero podrían utilizarse como recursos para el personal de pruebas.

### Pruebas de aceptación

Al igual que las pruebas de sistema, las pruebas de aceptación son realizadas sobre un sistema completamente integrado. Estas pruebas se realizan por el cliente, por el usuario o por un tercero designado por el cliente. Se llevan a cabo para asegurar que el resultado de las modificaciones es satisfactorio para el cliente, tanto del software como de la documentación generada.

## 1.2 Indicador de evaluación de la producción científica

Con el propósito de medir los resultados de la producción científica de los profesionales de la Facultad 3, se utiliza el "Indicador de desempeño de la actividad científico-técnica de los investigadores" (IDEC), el cual permite medir el desempeño científico-tecnológico de los investigadores dentro y fuera de la universidad. Favorece el incremento de la pertinencia, la calidad, la relevancia, el impacto económico y social, además de la visibilidad de los resultados científicos de la Universidad. A partir del IDEC se puede obtener una medida de la productividad alcanzada por el usuario en las líneas de investigación donde se haya desempeñado. Este método se centra en las actividades realizadas por el usuario, valorando su participación en proyectos productivos, las publicaciones realizadas, las remuneraciones obtenidas, el grado científico alcanzado y la participación en eventos. El cálculo del IDEC (13) viene dado por la fórmula:

$$\text{IDEC} = 5*(\text{PUB}+\text{PROY}) + 4*(\text{MN}+\text{MC}+\text{MAES}+\text{DOC}) + 3*(\text{PREM}+\text{IRES}+\text{REG}) + 2*\text{EVEN}$$

En la tabla 1 se presentan las consideraciones que se deben tener en cuenta para calcular el IDEC y en el [Anexo 1](#) se describe el criterio de ponderación para cada variable del algoritmo.

Variable	Descripción
<b>Publicaciones (PUB)</b>	Se otorgan puntos por cada artículo publicado en que el investigador aparezca como autor principal u otro tipo de autor.
<b>Proyecto (PROY)</b>	Se otorgan puntos por cada proyecto en que participe el investigador (de acuerdo al listado oficial emitido por la Dirección de Ciencia y Técnica de la universidad).
<b>Maestrías y Doctorados (MAES , DOC)</b>	Se otorgan puntos a los investigadores que hayan defendido exitosamente el doctorado o maestría, además de si han sido tutores de defensas exitosas de doctorado o maestría.
<b>Moneda Nacional (MN)</b>	Describe la bonificación que otorga el sistema al investigador que haya participado en el año en servicios, proyectos u otras acciones que hayan aportado al centro dinero en moneda nacional.

<b>Moneda convertible (MC)</b>	Describe la bonificación que otorga el sistema cuando el investigador haya participado durante el año en servicios, proyectos u otras acciones que hayan aportado al centro en metálico, equipos o insumos una cantidad de dinero en moneda convertible.
<b>Premios (PREM)</b>	Se otorgan puntos por cada premio recibido.
<b>Registros (REG)</b>	Puntos concedidos por cada uno de los registros que tenga el autor.
<b>Introducción de resultados (IRES)</b>	Puntos otorgados por cada aval obtenido que certifique la participación del investigador.
<b>Evento (EVEN)</b>	Se otorgan puntos por cada evento donde haya participado el investigador.

Tabla 1. Consideraciones para calcular el IDEC

En la presente investigación se propone el cálculo del IDEC como indicador de la producción científica. Una vez conocido el desempeño de los investigadores en una línea, se ordenan de forma descendente, quedando categorizados por el puesto que ocupen los profesionales en el sistema.

El cálculo del IDEC no contempla la categorización que pueda alcanzar un profesional cuando imparte una asignatura, por lo que se necesita un índice de evaluación para estos casos.

El psicólogo K. A Ericsson sugiere que toma aproximadamente 10 000 horas de práctica deliberada dominar algo, lo que se conoce como la Teoría de las 10 000 horas (14). A partir de esta idea, es posible medir los conocimientos adquiridos sobre la base de una escala de magnitud de 10 horas, de una forma aproximada:

- Con 1 hora: Se puede saber lo básico.
- Con 10 horas: Se tiene una noción más amplia de los conceptos básicos.
- Con 100 horas: Se adquiere un nivel medio.
- Con 1.000 horas: Se avanza a ser un especialista.
- Con 10.000 horas: Uno puede considerarse maestro en esa habilidad.

Estas cifras, expresadas en años sugieren que se necesitan 10 horas por semana en 20 años para considerarse experto en un tema, y para ser especialista se necesitarían entonces 4 años, teniendo en cuenta que se dediquen 5 horas semanales a la preparación para una clase.

Por tanto, el sistema categoriza a los profesores atendiendo al número de años que lleven impartiendo una asignatura, tomando como partida los profesores que hayan alcanzado 4 años o más impartiendo clases de una misma asignatura.



### 1.3 Metodología de desarrollo de software

La metodología es una parte del proceso de investigación o método científico. Está compuesta por un conjunto de procedimientos racionales utilizados para alcanzar una gama de objetivos que rigen en una investigación científica. El proceso de desarrollo de software define el conjunto de actividades precisas para convertir los requisitos de los usuarios en el conjunto de artefactos que componen un producto de software. Es como un libro de recetas de cocina, en el que se van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado. Existen varias metodologías, las cuales se pueden clasificar en dos grandes grupos: las tradicionales o pesadas y las ágiles (15).

- Las metodologías tradicionales se centran en la definición detallada de los procesos y tareas a realizar, las herramientas a utilizar y requiere una extensa documentación, ya que pretende prever todo de antemano (16).
- Las metodologías ágiles se encargan de valorar al individuo y las iteraciones del equipo más que a las herramientas o los procesos utilizados, es más importante crear un producto que funcione que escribir mucha documentación. Es más importante la capacidad de respuesta ante un cambio realizado que el seguimiento estricto de un plan (16).

Atendiendo las características de estos dos grupos de metodologías, teniendo en cuenta que el proyecto es pequeño, contando con solo dos integrantes y el tiempo dedicado a la solución es corto; además de observar el éxito que ha traído el uso de las metodologías ágiles en el desarrollo de proyectos en la Universidad, se decide emplear una metodología ágil.

En la tabla 2 se presenta una comparación entre las metodologías ágiles que más se utilizan en la actualidad.

Metodología	Objetivo	Características
<b>Programación extrema (XP)</b>	Centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software.	1. Tratamiento de las historias de usuario de forma dinámica y flexible. 2. El período de desarrollo es corto, en dependencia del proyecto que se elabora. 3. El cliente forma parte del equipo de desarrollo. 4. Las dimensiones del proyecto son pequeñas.
<b>Scrum</b>	Maximizar la retroalimentación sobre el desarrollo pudiendo corregir problemas y mitigar riesgos de forma temprana.	Define un proceso empírico, iterativo e incremental de desarrollo que intenta obtener ventajas respecto a los procesos definidos. Suministra un marco de administración basado en patrones organizacionales.

		<p>Especialmente indicada para proyectos con un rápido cambio de funcionalidades.</p> <p>Se realizan reuniones a lo largo del proyecto.</p>
<b>Crystal Clear</b>	Centrada en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos.	<p>Maneja iteraciones cortas con retroalimentación frecuente por parte de los usuarios/clientes, minimizando de esta forma la necesidad de productos intermedios.</p> <p>Es más fácil de aprender e implementar.</p>

Tabla 2. Comparación entre las metodologías ágiles

Después de analizar las características de las metodologías ágiles y del sistema que se requiere, se decide hacer uso de la metodología XP.

### Programación Extrema

La programación extrema es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo. El cliente forma parte del equipo de desarrollo, por lo que existe una retroalimentación continua entre ambos. Durante el desarrollo del proyecto genera poca documentación, lo cual permite un aprovechamiento al máximo del tiempo disponible. Es considerada ligera, flexible, predecible, de bajo riesgo, y no por ello menos científica. Especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, donde existe un alto riesgo técnico (17).

Es una metodología ligera basada en una serie de valores y prácticas que favorecen un aumento en la productividad del software. Los valores representan aspectos fundamentales para garantizar el éxito de un proyecto de software. Estos son:

- **Simplicidad:** XP propone el principio de hacer la cosa más simple que pueda funcionar, en relación al proceso y la codificación.
- **Comunicación:** algunos problemas en los proyectos tienen origen en que alguien no dijo algo importante en algún momento. XP hace casi imposible la falta de comunicación.
- **Retroalimentación** concreta y frecuente del cliente, del equipo y de los usuarios finales da una mayor oportunidad de dirigir el esfuerzo eficientemente (18).

La metodología XP define como artefacto primordial las historias de usuario que son la base del software a desarrollar. Estas historias son escritas por el cliente y describen las interacciones entre los usuarios y el sistema y generalmente son complementadas con otro tipo de descripción. A partir de ellas y de la arquitectura que se utilizará, se planifican las entregas, así como los objetivos de cada una y las iteraciones con que contará (16).

**Ventajas:**

- Programación organizada.
- Menor tasa de errores.
- Satisfacción del programador.

La Programación Extrema asume que la planificación nunca será perfecta y que los requisitos pueden cambiar a lo largo del ciclo de vida del software, a medida que varíen las necesidades del negocio. El período de desarrollo es corto y las dimensiones del proyecto son pequeñas.

**1.4 Herramientas y tecnologías de desarrollo**

Para poder desarrollar la aplicación se debe tener en cuenta las herramientas adecuadas. Para ello, se dividen en diferentes grupos como son: herramientas de modelado, de desarrollo, de base de datos, entre otras.

**1.5.1 Lenguaje de modelado**

Un lenguaje de modelado es un lenguaje artificial diseñado para expresar modelos. Ya que los modelos habitualmente se muestran en forma de diagramas por comodidad, los lenguajes de modelado suelen incorporar notaciones gráficas. Igual que los lenguajes naturales, los lenguajes de modelado poseen un conjunto de palabras que existen en el lenguaje denominadas léxico y un conjunto de reglas que dicen cómo se pueden combinar dichas palabras para componer "frases" que tengan sentido, lo que se conoce como sintaxis. Las "palabras" de los lenguajes de modelado no se transmiten mediante texto o sonido como en el caso de los lenguajes naturales, sino que, habitualmente, lo hacen en forma de íconos o dibujos para facilitar la visualización formal de los conceptos, que son abstractos. Las reglas sintácticas de los lenguajes de modelado nos dicen cómo se pueden conectar estos íconos para expresar modelos, y qué significa cada tipo de conexión (19).

**UML 2.1**

El Lenguaje de Modelado Unificado, por sus siglas en inglés (UML), es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. Ofrece un estándar para describir un modelo del sistema, incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como: expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados. Se utiliza para definir un sistema, para detallar los artefactos en el sistema, para documentar y construir. Permite modelar sistemas utilizando tecnologías orientada a objetos (20).

En el desarrollo de la solución se utiliza UML en su versión 2.0 para modelar y diseñar los principales artefactos generados en el proceso de desarrollo de las funcionalidades requeridas, ya que está

consolidado como un lenguaje estándar en el análisis y diseño de sistemas de cómputo y permite modelar todo el ciclo de vida de un proyecto informático.

### 1.5.2 Herramientas CASE

Ingeniería de Software Asistida por Computadoras, por sus siglas en inglés (CASE), son diversas herramientas informáticas individuales destinadas a aumentar la productividad en el desarrollo de software. Proponen una nueva filosofía del concepto de ciclo de vida basado en la automatización, para lo cual proporcionan un conjunto de herramientas bien integradas que, enmarcadas en una metodología, permiten automatizar las fases del ciclo de vida de un sistema de software (21).

#### Visual Paradigm 8.0

Visual Paradigm es una herramienta CASE para UML visual que soporta el ciclo de vida completo del desarrollo de software: análisis, diseño orientados a objetos, construcción, pruebas y despliegue. Fue diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque orientado a objetos. Está diseñada para ayudar al desarrollo ágil, lo que reduce en gran medida los costes de desarrollo y mejorar la calidad del producto final. Es una herramienta colaborativa, pues soporta múltiples usuarios trabajando sobre el mismo proyecto (22).

Se decide emplear Visual Paradigm en su versión 8.0 para el modelado de los artefactos generados para realizar el mantenimiento perfectivo, teniendo en cuenta la importancia del uso del software libre en Cuba y la existencia de una licencia gratuita. Además porque soporta todo el ciclo de vida del desarrollo de software, permite que varios usuarios trabajen sobre el mismo diagrama simultáneamente, es multiplataforma y posee la capacidad de generar código a partir de diagramas y viceversa.

### 1.5.3 Lenguaje de programación

El lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión (23).

#### PHP 5.6

Hipertexto Pre-processor (inicialmente PHP Tools, o, Personal Home Page Tools). Es un lenguaje de programación, interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en interpretación del lado del servidor, pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica (24).

Considerando que se va a realizar un mantenimiento perfectivo al Sistema de Gestión Curricular y este último está desarrollado utilizando el lenguaje PHP, entonces se hace necesario utilizarlo para la implementación de las nuevas funcionalidades.

#### 1.5.4 Marco de trabajo

Marco de trabajo del inglés framework, se define como un conjunto de componentes físicos y lógicos estructurados de tal forma que permiten ser reutilizados en el diseño y desarrollo de nuevos sistemas de información. Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado, a partir de una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. Los marcos de trabajo contienen patrones y buenas prácticas que apoyan el desarrollo de un producto y un proceso con calidad. Lo anterior permite vislumbrar la importancia de adoptar un Marco de Trabajo y cómo su selección debe ser una de las actividades relevantes al inicio de todo proceso de desarrollo software (25).

#### Symfony 2.6.5

Symfony es un marco de trabajo basado en lenguaje PHP, diseñado para optimizar el desarrollo de las aplicaciones web usando en el patrón Modelo Vista Controlador (MVC). Se describe como el marco de trabajo principal de PHP para crear sitios y aplicaciones web. Tiene una gran comunidad de desarrolladores y una amplia documentación. Es un conjunto de herramientas y servicios heterogéneos que permiten desarrollar páginas y aplicaciones web complejas de forma eficiente, reduciendo el tiempo de desarrollo y automatizando las tareas comunes; entre ellas un gestor de plantillas (Twig)<sup>3</sup> y un gestor ORM (Doctrine)<sup>4</sup>. Además es sumamente sencillo incorporar herramientas y aplicaciones de terceros (26).

#### Bootstrap 3.1

Bootstrap es el marco de trabajo de Twitter que permite crear interfaces web con CSS y JavaScript que adaptan la interfaz dependiendo del tamaño del dispositivo en el que se visualice de forma nativa, es decir, automáticamente se adapta al tamaño de un ordenador o de una Tablet sin que el usuario tenga que hacer nada, esto se denomina diseño adaptativo o Responsive Design (27).

Es un gran marco de trabajo a la hora de crear interfaces web, los diseños creados con él son simples, limpios e intuitivos, esto les da agilidad a la hora de cargar y al adaptarse a otros dispositivos (28).

---

<sup>3</sup> **Twig** es un motor de plantillas para el lenguaje de programación PHP. Su sintaxis se origina en plantillas de Jinja y Django. Es un producto de código abierto desarrollado por Fabien Potencier.

<sup>4</sup> **Mapeo de Objeto Relacional** (ORM) por sus siglas en inglés, es un componente de software que permite trabajar con datos persistentes como si fueran parte de una base de datos orientada a objetos.

El marco de trabajo trae varios elementos con estilos predefinidos fáciles de configurar: botones, menús desplegables, formularios.

### **1.5.5 Entorno de desarrollo integrado**

Un Entorno de Desarrollo Integrado (IDE), es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o utilizarse para varios. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación (29).

#### **NetBeans 8.0**

El IDE NetBeans es una aplicación de código abierto diseñada para el desarrollo de aplicaciones fácilmente portables entre las distintas plataformas, haciendo uso de la tecnología Java. Dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones web, control de versiones, colaboración entre varias personas, creación de aplicaciones compatibles con teléfonos móviles, resaltado de sintaxis (30).

### **1.5.6 Sistema gestor de bases de datos**

Un Sistema Gestor de Bases de Datos (SGBD) es una colección de datos relacionados entre sí, estructurados y organizados, y un conjunto de programas que acceden y gestionan esos datos. La colección de esos datos se denomina Base de Datos (BD) (31).

#### **PostgreSQL 9.3**

Es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado. Estabilidad, potencia, robustez, facilidad de administración e implementación de estándares han sido las características que más se han tenido en cuenta durante su desarrollo. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema (32).

Tomando como punto de partida las características antes mencionadas, teniendo en cuenta que es una herramienta multiplataforma, libre y que para el desarrollo del Sistema de Gestión Curricular se utilizó PostgreSQL, se decide utilizar este gestor de bases de datos para realizar el mantenimiento.

#### **PgAdmin 1.16.1**

PgAdmin III es una aplicación gráfica para facilitar el trabajo con el gestor de bases de datos PostgreSQL, siendo la más completa y popular con licencia de código abierto. Es capaz de gestionar

versiones a partir de PostgreSQL 7.3 .Incluye un editor SQL con resaltado de sintaxis, un editor de código de la parte del servidor, la conexión al servidor puede hacerse mediante conexión TCP/IP o Unix Domain Sockets (en plataformas \*nix), y puede cifrarse mediante SSL para mayor seguridad (33).

### 1.5.7 Servidor de aplicaciones

Un servidor de aplicaciones es un elemento de software que es capaz de traducir las instrucciones y además comunicar con otros servidores, como los servidores de bases de datos, para extraer información de la empresa que se necesita para resolver la petición. Trabajan en conjunto con los servidores web para que el proceso se haga de forma transparente al usuario.

#### Apache 2.2.22-13

Apache es el servidor web hecho por excelencia. Su facilidad de configuración, robustez y estabilidad lo han convertido en el servidor web más popular de la actualidad (34). Es un software de código abierto que utiliza una licencia de tipo Apache License, variante de la licencia GPL de Linux (35). Eso significa que se puede distribuir libremente e incluso modificar el código siempre y cuando el resultado mantenga la licencia original. Está caracterizado porque:

- Corre en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal.
- Es una tecnología gratuita.
- Es un servidor altamente configurable de diseño modular.
- Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Es posible configurar Apache para que ejecute un determinado script cuando ocurra un error en concreto.
- Permite la creación de ficheros de log a medida del administrador, de este modo se puede tener un mayor control sobre lo que suceder en el servidor.

### 1.5.8 Control de versiones

El control de versiones es el proceso de almacenar y recuperar cambios de un proyecto de desarrollo. Los sistemas de control de versiones SCV, permiten disponer de una versión anterior para corregir errores o actualizar funciones. Dentro de sus funcionalidades está el conservar las versiones que se hayan generado a través del tiempo, así como los diferentes archivos que integran el proyecto en cuestión, uniendo en forma automática las aportaciones de los integrantes de un equipo de trabajo (36).

Los principales beneficios de utilizar una herramienta

- Cualquier revisión almacenada de un archivo puede ser recuperada para visualizarse o modificarse.

- Puede desplegarse las diferencias entre distintas versiones.
- Las correcciones pueden ser creadas automáticamente.
- Múltiples desarrolladores pueden trabajar simultáneamente en el mismo proyecto o archivo sin pérdida de datos.
- Los proyectos pueden ser divididos para permitir el desarrollo simultáneo en varias versiones, estas divisiones pueden ser fusionadas para alcanzar el objetivo principal del desarrollo.
- El desarrollo distribuido, es soportado a través de las redes de datos con diferentes mecanismos de autenticación.

## **Git**

Git es un sistema de control de versiones distribuido y de código abierto. Fue impulsado por Linus Torvalds y el equipo de desarrollo del Kernel de Linux. Es multiplataforma, lo que permite que se pueda utilizar en los sistemas operativos más comunes.

El control de versiones se realizó a través de la herramienta GitLab, que es una aplicación de código abierto que permite administrar repositorios en Git mediante una interfaz web.

## **Conclusiones parciales**

Luego de fundamentar los conceptos y características del mantenimiento de software mediante el estudio de la información asociada a la investigación se concluye:

- Atendiendo a las necesidades de incorporar nuevas funcionalidades al Sistema de Gestión Curricular y luego de analizar los tipos de mantenimiento de software que existen, se decide realizar un mantenimiento perfectivo.
- El análisis de las diferentes metodologías, herramientas y lenguajes sugiere que para el desarrollo del presente trabajo se debe utilizar la metodología XP, que facilita la creación de una documentación con los artefactos necesarios.
- Se realizó el análisis del lenguaje de modelado y la herramienta CASE seleccionada en apoyo al mantenimiento del software, así como las herramientas definidas para la implementación de las nuevas funcionalidades.
- Se define como propuesta final el lenguaje de programación PHP, además los marcos de trabajo Symfony y Bootstrap, teniendo como IDE al NetBeans.



## Capítulo 2: DISEÑO DE LA PROPUESTA DE SOLUCIÓN

### Introducción

*“Puede que tengas grandes ideas en la cabeza, pero lo que importa es la acción. Una idea, si no se lleva a cabo, no producirá ninguna manifestación, ni resultados ni recompensas.”*

*Miguel Ruiz*

En el presente capítulo se realiza el análisis y diseño de las funcionalidades que se incluyen en el Sistema de Gestión Curricular. Se describe la arquitectura y se realiza un levantamiento de las funcionalidades, mediante la descripción contenida en las historias de usuario, y de los requisitos no funcionales necesarios para un buen funcionamiento de la aplicación.

### 2.1 Descripción del sistema.

Tomando como punto de partida el problema a resolver, se propone realizar un mantenimiento perfectivo al Sistema de Gestión Curricular, con el fin de incorporar nuevas funcionalidades que respondan con la necesidad de evaluar la producción científica y categorizar a los profesionales, atendiendo a las líneas de investigación por las que cada usuario tenga interés. El usuario tiene creado un perfil en el sistema el cual está compuesto por las líneas de investigación de su interés, las evidencias de los eventos en los que ha participado que tributan al currículum vitae, entre otros aspectos. Una vez conocidas las líneas de investigación preferidas por un usuario, el programa debe mostrar las personas con mayor producción científica en cada tema. Además, cuando un usuario adicione una nueva evidencia a su currículum vitae, el sistema debe enviar notificaciones a los usuarios que sigan su desarrollo profesional.

### 2.2 Modelo de domino

Se decide realizar un modelo de dominio para expresar el funcionamiento de la solución propuesta, ya que esta carece de procesos de negocio estructurados, que permitan realizar un modelado del negocio. Un modelo de dominio permite a los desarrolladores y las empresas de software tener un lenguaje común, que a su vez hace mucho más sencilla la comunicación de los requisitos y el mantenimiento de la aplicación. La utilización de un modelo de dominio tiene como principal objetivo lograr identificar y describir todos los conceptos presentes en el dominio del problema (37).

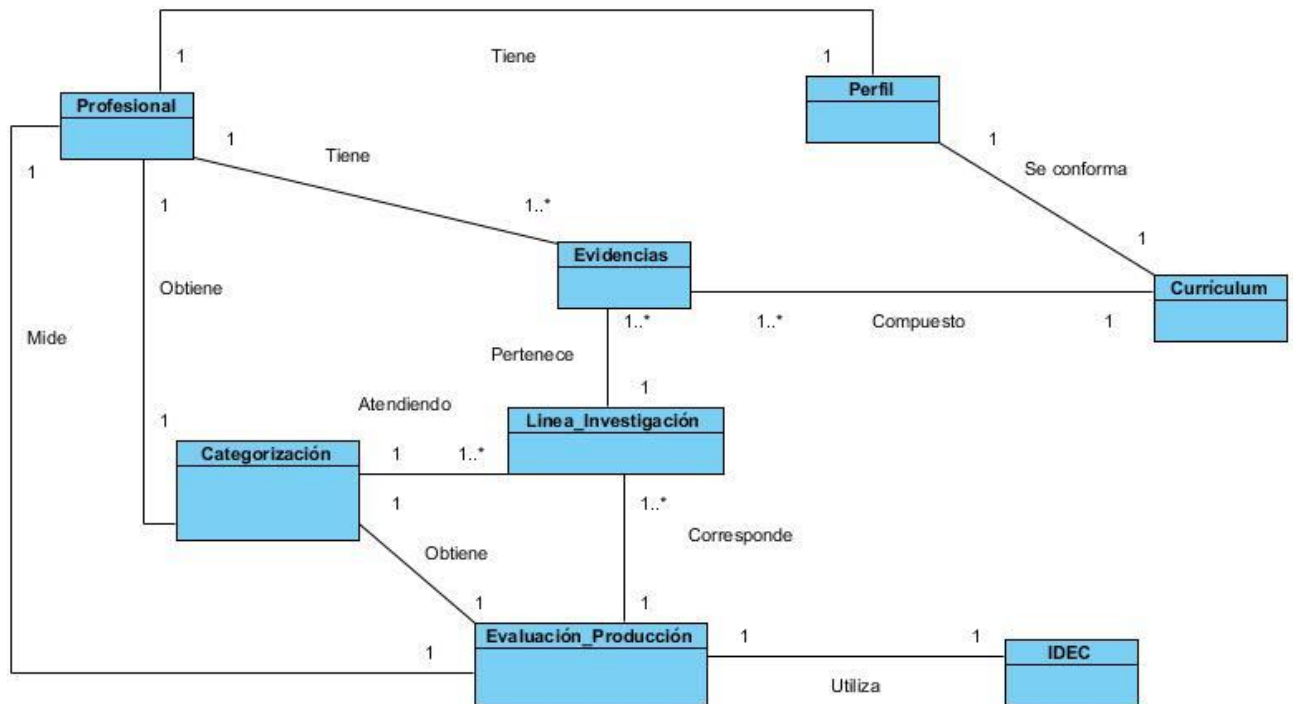


Figura 2. Modelo de dominio

### Definición de los conceptos del modelo de dominio:

**Profesional:** es la persona que interactúa con el sistema y sus funcionalidades.

**Perfil:** lugar donde se publican los datos, preferencias del usuario y las recomendaciones realizadas.

**Currículum:** contiene la información curricular del profesional registrado en el sistema.

**IDEC:** indicador utilizado para evaluar la producción científica.

**Evaluación\_Producción:** Mecanismo que utiliza el IDEC para evaluar el desempeño productivo de un profesional en cada línea de investigación, por cada evidencia.

**Categorización:** Puesto que ocupa un profesional atendiendo a la evaluación de la producción de una línea de investigación que desarrolla.

### 2.3 Ingeniería de requisitos

La Ingeniería de Requisitos cumple un papel primordial en el proceso de producción de software, debido a que va enfocada a un área fundamental en el desarrollo de todo sistema, que es la definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones que describan con claridad, sin ambigüedades, en forma consistente y compacta el comportamiento del sistema, de manera que le garantice minimizar los problemas relacionados

con el desarrollo (37). El primordial objetivo de los requisitos es guiar el desarrollo hacia una solución que se corresponda con las necesidades del cliente.

### **2.3.1 Obtención de requisitos**

El proceso de obtención de requisitos, cuya finalidad es llevar a la luz los requisitos, no solo es un proceso técnico, sino también un proceso social que envuelve tanto a desarrolladores como clientes y usuarios. Se enfoca en la descripción del propósito del sistema para obtener como resultado la especificación de los requisitos de software.

#### **Tormenta de ideas**

Consiste en reuniones entre diferentes personas (de 4 a 10) donde como primer paso sugieren toda clase de ideas sin juzgar su validez, y después de recopilar todas las ideas se realiza un análisis detallado de cada propuesta. Esta técnica se puede utilizar para identificar un primer conjunto de requisitos en aquellos casos donde no están muy claras las necesidades que hay que cubrir, o cuando se está creando un sistema que habilitará un servicio nuevo para la organización (38).

En cuatro encuentros se realizaron tormentas de ideas donde participaron los realizadores del mantenimiento y el cliente del sistema. Se obtuvo como resultado una amplia perspectiva sobre los requisitos que el software debía suplir.

#### **Entrevistas**

La entrevista es de gran utilidad para obtener información cualitativa como opiniones, o descripciones subjetivas de actividades. Es una técnica muy utilizada, y requiere una mayor preparación y experiencia por parte del analista. La entrevista se puede definir como un “intento sistemático de recoger información de otra persona” a través de una comunicación interpersonal que se lleva a cabo por medio de una conversación estructurada. Debe quedar claro que no basta con hacer preguntas para obtener toda la información necesaria. Es muy importante la forma en que se plantea la conversación y la relación que se establece en la entrevista.

Se realizaron entrevistas a especialistas de la dirección de investigación de la Universidad. Se utilizó una combinación de entrevista cerrada y abierta, permitiendo las primeras centrar la conversación en el mantenimiento a desarrollar, mientras que las segundas fueron surgiendo durante las conversaciones.

Con las entrevistas realizadas se obtuvo información precisa acerca de las necesidades reales y perspectivas de utilización del sistema.

### 2.3.2 Especificación de requisitos funcionales

Las Historias de Usuario son un enfoque de requerimientos ágil que se focaliza en establecer conversaciones acerca de las necesidades de los clientes. Son descripciones cortas y simples de las funcionalidades del sistema, narradas desde la perspectiva de la persona que desea dicha funcionalidad, usualmente un usuario (39). A continuación se presentan 3 de las historias de usuarios del sistema, el resto se puede encontrar en el [Anexo 2](#):

Historia de usuario	
<b>Número: 3</b>	<b>Usuario:</b>
<b>Nombre historia: Ponderar el perfil de usuario por líneas de investigación</b>	
<b>Prioridad en negocio:</b> Alta	<b>Riesgo en desarrollo:</b> Alto
<b>Puntos estimados: 4.5</b>	<b>Iteración asignada: 1</b>
<b>Programador responsable: Laura Beatriz – Pablo Moreno</b>	
<b>Descripción:</b> El sistema debe ser capaz de ponderar el perfil de usuario por cada línea de investigación desarrollada.	

Tabla 3. HU Ponderar perfil por líneas de investigación

Historia de usuario	
<b>Número: 6</b>	<b>Usuario:</b>
<b>Nombre historia: Mostrar categorizaciones automáticas</b>	
<b>Prioridad en negocio:</b> Alta	<b>Riesgo en desarrollo:</b> Alto
<b>Puntos estimados: 3</b>	<b>Iteración asignada: 2</b>
<b>Programador responsable: Laura Beatriz – Pablo Moreno</b>	
<b>Descripción:</b> El sistema debe ser capaz de mostrar automáticamente las recomendaciones realizadas a un usuario.	

Tabla 4. HU Mostrar categorizaciones automáticas

Historia de usuario	
<b>Número: 9</b>	<b>Usuario:</b>
<b>Nombre historia: Calcular indicador de desempeño para cada evidencia</b>	
<b>Prioridad en negocio:</b> <b>Baja</b>	<b>Riesgo en desarrollo:</b> <b>Medio</b>
<b>Puntos estimados: 3</b>	<b>Iteración asignada: 1</b>
<b>Programador responsable: Laura Beatriz – Pablo Moreno</b>	
<b>Descripción:</b> <b>Se calcula el indicador de desempeño de cada evidencia, certificada o no.</b>	

Tabla 5. HU Calcular indicador de desempeño para cada evidencia

### 2.3.3 Requisitos no funcionales

Los requerimientos no funcionales son restricciones de los servicios o funciones ofrecidas por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. A menudo se aplican al sistema en su totalidad. Normalmente apenas se aplican a características o servicios individuales del sistema (8).

- **Disponibilidad**

**RNF-1:** el sistema debe estar disponible para todos los usuarios al menos durante toda la jornada laboral.

- **Seguridad**

**RNF-2:** el sistema debe cumplir con los principios básicos de seguridad de cualquier sistema informático, manteniendo la integridad, confidencialidad y disponibilidad de la información.

**RNF-3:** el sistema brinda control de acceso, garantizando que solo los profesionales que radican en la universidad sean capaces de acceder al mismo, así como permitir las funcionalidades permitidas a cada rol.

**RNF-4:** el sistema deberá garantizar la protección ante acciones no autorizadas como las inyecciones SQL.

- **Portabilidad**

**RNF-5:** el sistema podrá ejecutarse desde Linux, Windows, MacOS.

- **Tipo de Aplicación Informática**

**RNF-6:** el producto constituye una aplicación web enfocada en las necesidades del gestor curricular de la facultad 3.

- **Finalidad:**

**RNF-7:** este sistema está enfocado a la gestión de información relacionada con el perfil del usuario que maneja el gestor de currículos de la facultad 3.

- **Ambiente:**

**RNF-8:** el sistema se desarrollará con tecnología PHP versión 5.6.

**RNF-9:** se empleará como Gestor de Base de Datos PostgreSQL en su versión 9.3

**RNF-10:** se empleará como Servidor de Aplicaciones Web Apache en su versión 2.0.

**RNF-11:** las computadoras de los clientes solo requerirán de un navegador Internet.

- **Usabilidad:**

**RNF-12:** en el sistema no existirán más de 3 interfaces para lograr una funcionalidad completa.

**RNF-13:** la tipografía y colores serán estándares en toda la aplicación.

- **Interfaz**

**RNF-14:** el sistema deberá tener un diseño de interfaz de usuario basada en las características descritas en el gestor curricular de la facultad 3.

- **Hardware**

**RNF-15:** se requiere para el servidor como mínimo una computadora con procesador Dual-Core a 2.7GHz, con una memoria RAM de 1GB o superior, y 250 GB de espacio en el disco duro.

**RNF-16:** se requiere para la máquina cliente como mínimo una computadora con procesador Pentium 3 a 800MHz, con una memoria RAM de 256 MB o superior, y 40 GB o más de espacio en el disco duro.

- **Software**

**RNF-17:** se requieren las librerías de php5 como son ldap, mcrypt y curl para poder ejecutar la aplicación en el servidor.

**RNF-18:** al ser una aplicación Web, se necesita un navegador, como Internet Explorer, Mozilla Firefox, Chrome, etc.

### 2.3.4 Validación de requisitos funcionales

La validación de requerimientos trata de mostrar que estos realmente definen el sistema que desea el cliente. Es importante debido a que los errores en el documento de requerimientos pueden conducir a importantes costes al repetir el trabajo cuando son descubiertos durante el desarrollo o después de que el sistema esté en uso (8).

La construcción de prototipos en un enfoque de validación donde se presenta a los usuarios finales y los clientes un modelo ejecutable del sistema. Estos pueden experimentar con los modelos para ver si cumple sus necesidades reales.

A continuación se muestran los prototipos correspondientes a la validación de requisitos especificados en las historias de usuario mostradas anteriormente.

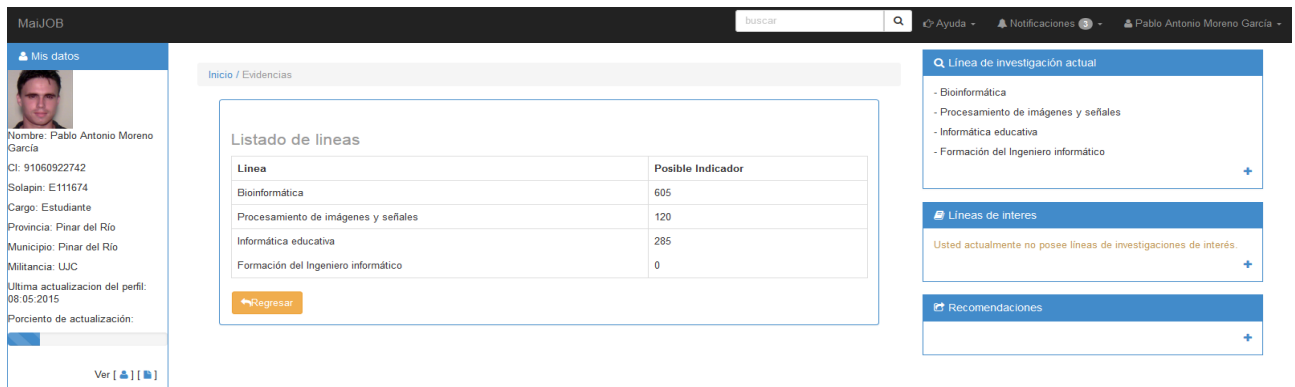


Figura 3. Prototipo de interfaz. Ponderar perfil de usuario por líneas de investigación

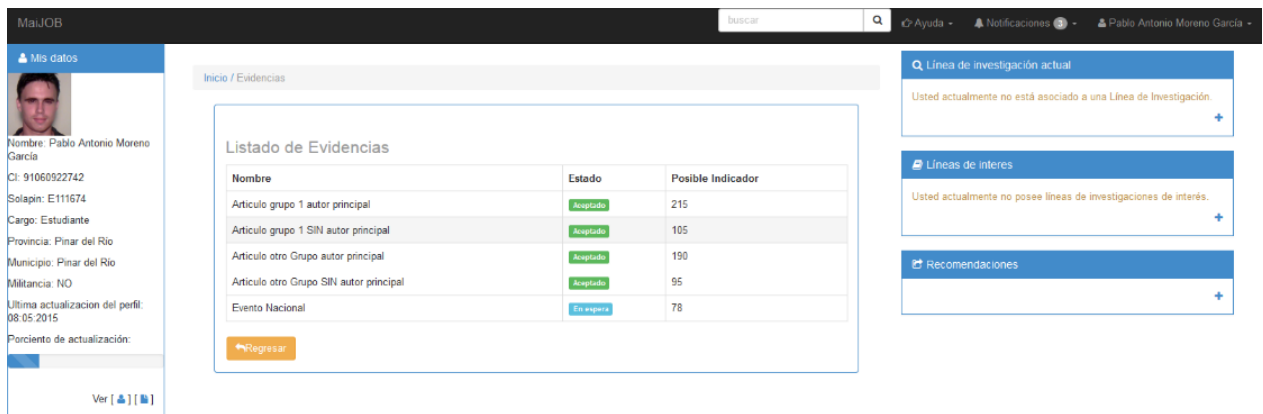


Figura 4. Prototipo de interfaz. Calcular indicador de desempeño para cada evidencia

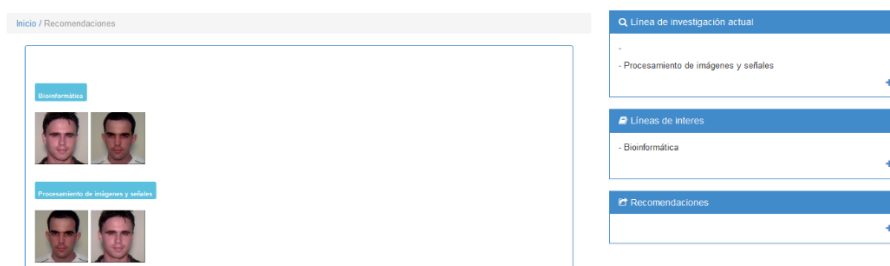


Figura 5. Prototipo de interfaz. Mostrar categorizaciones automáticas

## 2.4 Fase de planificación

En esta fase se definen las historias de usuario, en las que se especifican las características que debe tener cada una y el equipo de desarrollo se familiariza con las herramientas y tecnologías que debe utilizar para una correcta implementación de la solución. Al finalizar esta etapa, se obtiene como resultado una visión general del sistema.

### 2.4.1 Plan de entregas

El plan de entregas es un documento donde se manifiesta el compromiso final del equipo de desarrollo con el cliente. En él se especifica exactamente qué historias de usuario serán implementadas en cada entrega del sistema y sus prioridades, de modo que también permita conocer con exactitud qué historias de usuario serán implementadas en la próxima liberación. El cronograma de entregas se realiza en base a las estimaciones de tiempos de desarrollo realizadas por los desarrolladores. Luego de algunas iteraciones es recomendable realizar nuevamente una reunión con los actores del proyecto, para evaluar nuevamente el plan de entregas y ajustarlo si es necesario (40). La tabla 6 muestra un resumen del plan de entregas de los objetivos cumplidos por cada iteración:

Iteración	Historia de usuario	Prioridad	Tiempo inicio	Tiempo fin
1	HU 1	Alta	10-03-2015	13-03-2015
1	HU 2	Alta	13-03-2015	18-03-2015
1	HU 3	Alta	19-03-2015	25-03-2015
1	HU 9	Baja	26-03-2015	31-03-2015
2	HU 4	Alta	01-04-2015	10-04-2015
2	HU 5	Alta	13-04-2015	20-04-2015
2	HU 6	Media	21-04-2015	23-04-2015
3	HU 8	Baja	23-04-2015	30-04-2015
3	HU 7	Media	04-05-2015	07-05-2015

Tabla 6. Plan de entregas

### 2.4.2 Plan de iteraciones

El plan de iteración consiste en seleccionar las historias de usuario que corresponden a cada iteración. Se determinaron tres iteraciones, realizando al inicio de cada una las actividades de análisis puntualizando con el cliente los datos necesarios, ya que las historias de usuario no contienen suficientes detalles como para realizar esta actividad. En esta etapa es primordial la participación del cliente. En la primera iteración se realizan las HU correspondientes al cálculo del IDEC de los investigadores; en la segunda se garantiza la generación de sugerencias y en la tercera, las HU que responden a la notificación de eventos a usuarios que pertenecen a una misma red de investigación. A continuación se muestran las iteraciones necesarias para la producción del software.

Iteraciones	Historias de usuario a implementar	Duración del trabajo (en semanas)
Iteración 1	<ul style="list-style-type: none"> <li>Ponderar el perfil de usuario por año</li> <li>Ponderar el perfil histórico de usuario</li> </ul>	3



	<ul style="list-style-type: none"> <li>• Ponderar perfil de usuario por líneas de investigación</li> <li>• Calcular indicador de desempeño para cada evidencia</li> </ul>	
Iteración 2	<ul style="list-style-type: none"> <li>• Generar categorización a partir del IDEC</li> <li>• Generar categorización a partir de asignaturas impartidas</li> <li>• Mostrar recomendaciones automáticas</li> </ul>	3
Iteración 3	<ul style="list-style-type: none"> <li>• Mostrar personas que se siguen</li> <li>• Notificar cambios en un perfil</li> </ul>	2

Tabla 7. Distribución de iteraciones por historia de usuario

## 2.5 Fase de diseño

El diseño de XP sigue rigurosamente el principio de diseño sencillo. Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto. La complejidad innecesaria y el código extra debe ser removido inmediatamente. Kent Beck dice que en cualquier momento el diseño adecuado para el software es aquel que: supera con éxito todas las pruebas, no tiene lógica duplicada, refleja claramente la intención de implementación de los programadores y tiene el menor número posible de clases y métodos (41).

### 2.5.1 Arquitectura del sistema

El concepto de arquitectura de software se refiere a la estructuración del sistema que, idealmente, se crea en etapas tempranas del desarrollo. Esta estructuración representa un diseño de alto nivel del sistema que tiene dos propósitos primarios: satisfacer los atributos de calidad (desempeño, seguridad, modificación), y servir como guía en el desarrollo (42).

La arquitectura en tres niveles es la especialización de la arquitectura cliente-servidor donde la carga se divide en tres capas con un reparto claro de funciones: una capa para la presentación (interfaz de usuario), otra para el cálculo (donde se encuentra modelado el negocio) y otra para el almacenamiento (persistencia). Una capa solamente tiene relación con la siguiente y entre ellas. El desarrollo se puede llevar a cabo en varios niveles y, en caso de cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado (43).

Para el diseño arquitectónico se utiliza el patrón Modelo Vista Controlador (MVC), el cual sugiere dividir el software en tres componentes el modelo del negocio, las vistas y las clases controladoras.

**Modelo:** es la representación de la información que maneja la aplicación. El modelo en sí lo constituyen los datos puros y la lógica de los propios datos que puestos en el contexto del sistema proveen de información al usuario y en algunos casos a la propia aplicación. Define la lógica del negocio.

**Vista:** es la representación del modelo en forma gráfica disponible para la interacción con el usuario. En el caso de una aplicación web, la “Vista” sería una página HTML con contenido dinámico sobre la cual el usuario puede realizar sus operaciones.

**Controlador:** es la parte encargada de manejar y responder las solicitudes del usuario, procesando toda la información necesaria y modificando el modelo en caso de ser necesario.

### 2.5.2 Patrones arquitectónicos

Un patrón impone una regla sobre la arquitectura, pues describe la manera en que el software manejará algún aspecto de su funcionalidad al nivel de la infraestructura. Los patrones se usan junto con un estilo arquitectónico para determinar la estructura general de un sistema (10).

El patrón arquitectónico Modelo Vista Controlador (MVC) separa los datos y la lógica del negocio de la interfaz de usuario y del módulo que se encarga del control, en tres componentes distintos. Symfony utiliza MVC, lo que obliga a dividir y organizar el código de acuerdo a las convenciones establecidas por el marco de trabajo. A continuación se muestra la figura 6 donde se evidencia la organización propuesta por Symfony:

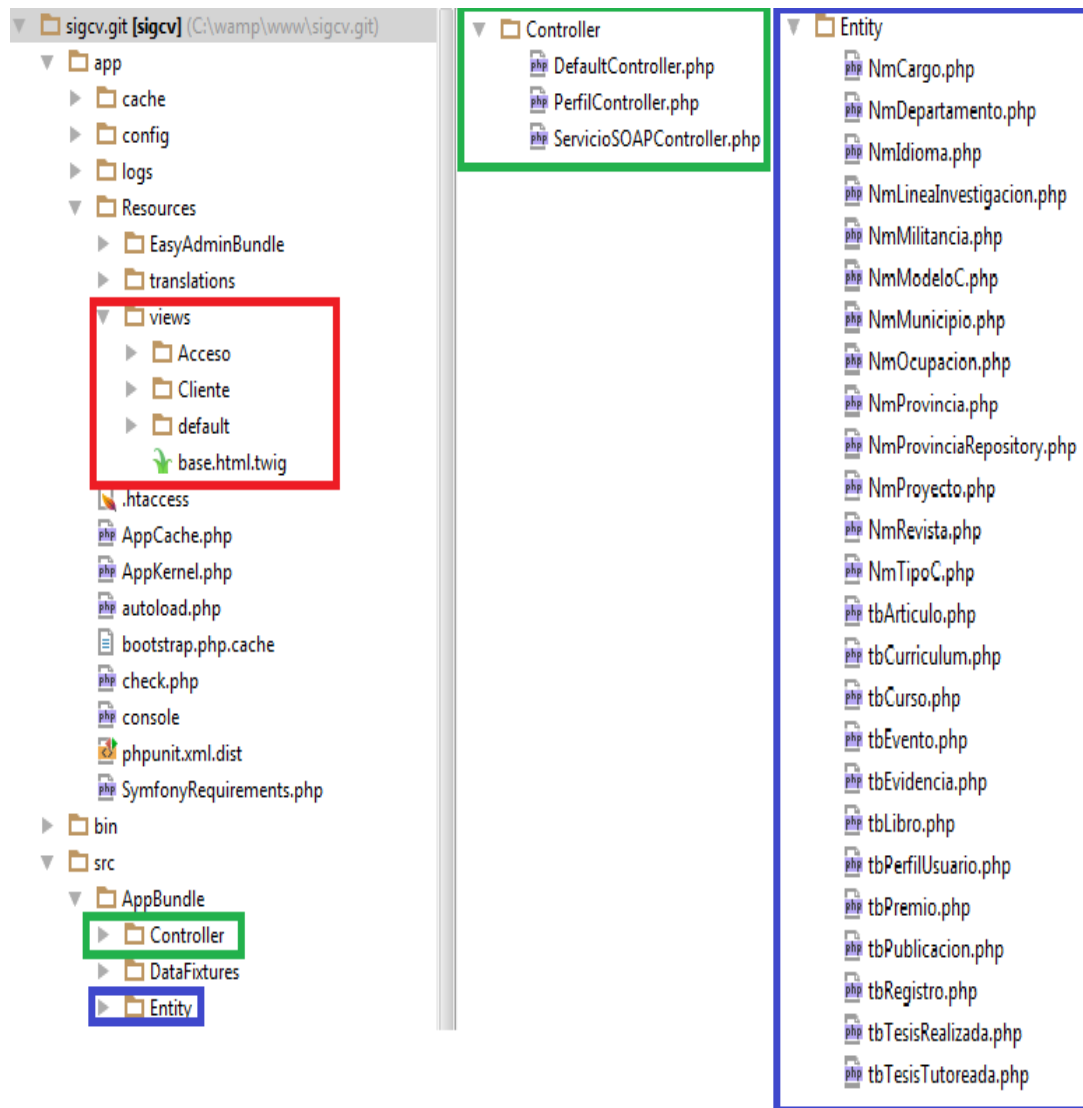


Figura 6. Organización propuesta por Symfony

## Modelo

El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes del tipo de gestor de bases de datos utilizado por la aplicación. En el modelo se encuentran las clases, que son generadas de forma automática según la estructura de la BD (44). En la figura 7 se presenta la estructura del modelo, donde se encuentran las entidades y las clases que generan consultas a la base de datos.

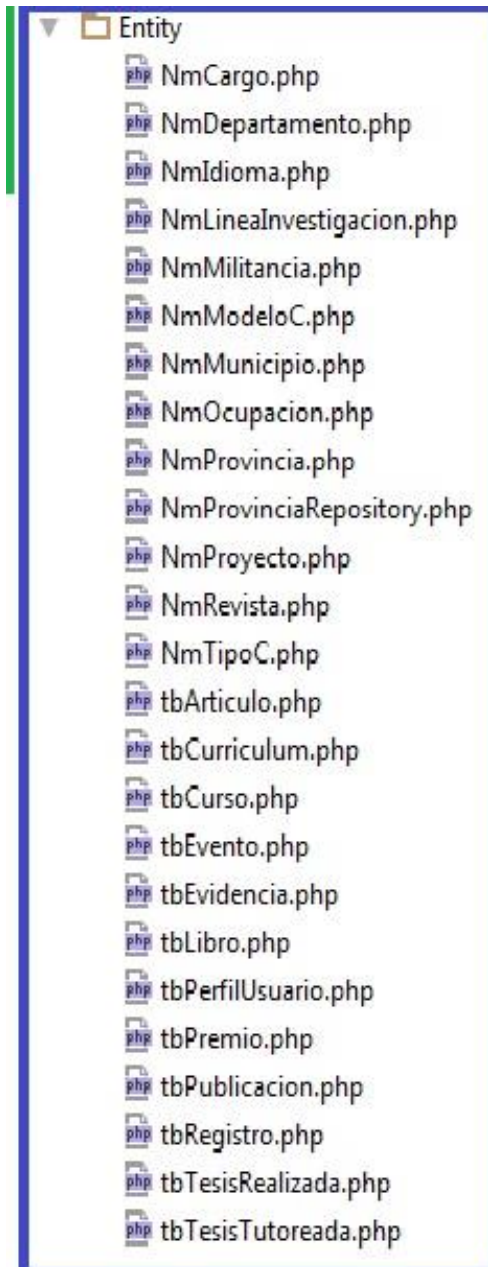


Figura 7. Modelo

## Vista

La vista es la encargada de originar las páginas que son mostradas como resultado de las acciones. La vista en Symfony está conformada por varias partes, preparadas cada una de ellas para ser fácilmente transformada por la persona que normalmente trabaja con cada aspecto del diseño de las aplicaciones (44). En la figura 8 se muestran las vistas generadas para el sistema.

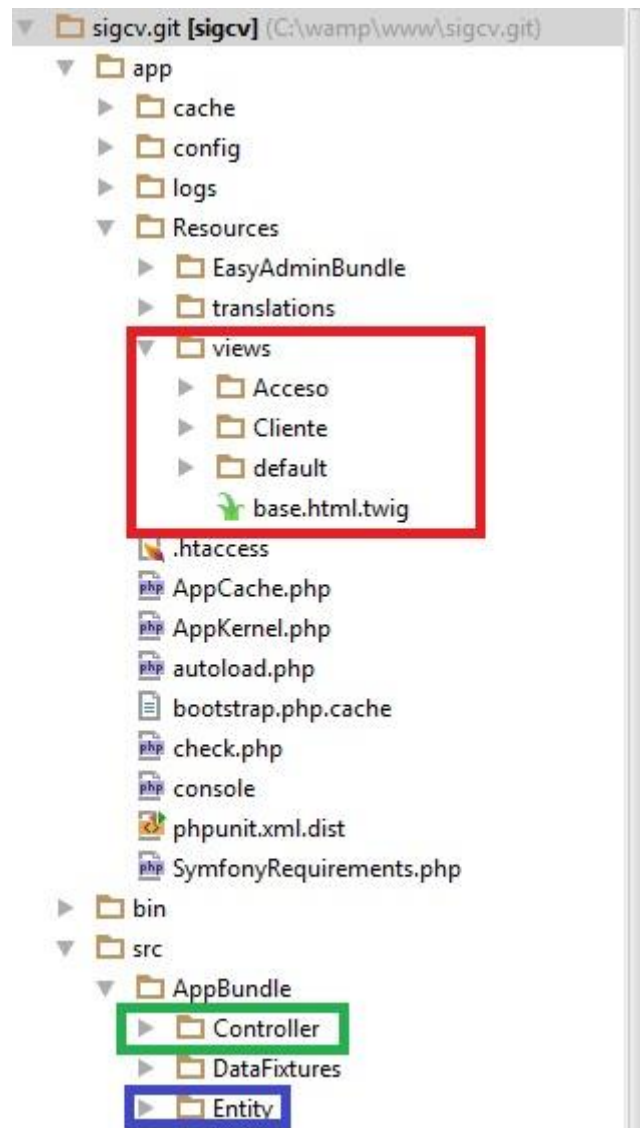


Figura 8. Vista

## Controlador

El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones. En él se encuentran las acciones, las cuales son el núcleo de la aplicación, pues contienen toda la lógica del negocio. Estas acciones utilizan el modelo y precisan las variables para la vista (44).

La figura 9 muestra la estructura de controladores generados para desarrollar el sistema.

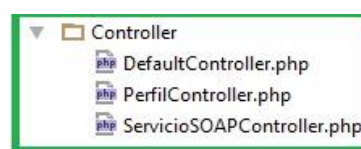


Figura 9. Controlador

### 2.5.3 Patrones de diseño

En la tecnología de objetos, un patrón es una descripción de un problema y la solución, a la que se da un nombre, y que se puede aplicar a nuevos contextos; idealmente, proporciona consejos sobre el modo de aplicarlo en varias circunstancias, y considera los puntos fuertes y compromisos. Muchos patrones proporcionan guías sobre el modo en el que deberían asignarse las responsabilidades a los objetos, dada una categoría específica del problema.

En el presente trabajo se emplean los patrones **GRASP**<sup>5</sup>:

**Experto:** se encarga de asignar una responsabilidad al experto en información, o sea, a la clase que cuenta con la información necesaria para cumplir con la responsabilidad. Este patrón es ampliamente usado en las clases del dominio para acceder a la información específica de cada una de las entidades (45).

Este patrón se evidencia en la clase *tbEvidenciaController* donde la entidad evidencia tiene toda la información necesaria para cumplir con sus responsabilidades.

```
return $this->render(':Cliente:tbEvidencia/index', array(
    'curso' => $curso, 'evento' => $evento, 'premio' => $premio, 'registro' => $registro, 'tutorada' => $tutorada,
    'realizada' => $realizada, 'libro' => $libro, 'articulo' => $articulo, 'asignatura' => $asignatura,
    'diplomado' => $diplomado, 'graduacion' => $graduacion, 'reconocimiento' => $reconocimiento
));
```

Figura 10. Patrón de diseño experto

**Controlador:** Mantiene el control actuando como intermediario entre una determinada interfaz y el algoritmo que la implementa, posibilita la asignación de responsabilidades para controlar el flujo de eventos del sistema a clases específicas (45). Este se evidencia en el proceso de notificación o en el cálculo del algoritmo IDEC. Está reflejado en la clase *tbEvidenciaController*, ya que es donde se controla toda la información referente a una evidencia.

<sup>5</sup> GRASP (General Responsibility Assignment Software Patterns): Patrones generales para asignar responsabilidades. Describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones.

```

/**
 * @Route("evidencia/listado", name="evidencia")
 */
public function indexAction() {...}

/**
 * @Route("evidencia/{tipo}/crear", name="evidencia_create")
 */
public function createAction($tipo) {...}

/** Creates a form to create a tbEvidencia entity. ...*/
private function createCreateForm(tbEvidencia $entity) {...}

/** @Route("evidencia/{tipo}/adicionar", name="evidencia_new") ...*/
public function newAction($tipo) {...}

/** @Route("evidencia/{id}/mostrar", name="evidencia_show") ...*/
public function showAction($id) {...}

/** @Route("evidencia/{id}/modificar", name="evidencia_edit") ...*/
public function editAction($id) {...}

/** Creates a form to edit a tbEvidencia entity. ...*/

```

Figura 11. Patrón de diseño controlador

**Creador:** este patrón es el responsable de asignarle a la clase B la responsabilidad de crear una instancia de clase A, B es un creador de los objetos A. El objetivo de este patrón es encontrar al creador que se debe relacionar con el objeto que se produce (45).

Este patrón se evidencia en la clase *tbEvidenciaController* donde se permite que un perfil pueda crear una nueva evidencia.

```

if ($form->isValid()) {

    $perfil->addEvidencium($entity);
    $em->persist($perfil);
    $em->persist($entity);
    $em->flush();
}

```

Figura 12. Patrón de diseño creador

**Alta cohesión:** La cohesión es la medida que determina cuán relacionadas y adecuadas están las responsabilidades de una clase. La alta cohesión consiste en garantizar que una clase colabore con otras para compartir el trabajo si la tarea es grande (45). Hace de los componentes del sistema un conjunto bien acoplado. Se pone de manifiesto al calcular el indicador de desempeño de una

evidencia, ya que en esta se relacionan cinco clases (*tbUsuario*, *tbPerfilUsuario*, *tbPerfilLinea*, *tbEvidencia* y *NmLineaInvestigacion*).

```
public function calcularIdecEvidencia()
{
    $em = $this->getDoctrine()->getManager();
    $parametro = 0;
    //obtener usuario en session
    $context = $this->get('security.context');
    $usuario = $context->getToken()->getUsername();
    $tbusuario = $em->getRepository('AppBundle:tbUsuario')->findOneBy(array('usuario' => $usuario));
    $evidencias = $tbusuario->getPerfil()->getEvidencia();
    $profile = $tbusuario->getPerfil();
    $lineas = $tbusuario->getPerfil()->getLinvestigacionactual();
    $pl = new PerfilLinea();

    foreach ($evidencias as $evidencia) {
        $ponderar = $this->idecP($evidencia);
        $evidencia->setIdecE($ponderar);
        $em->persist($evidencia);
    }
}
```

Figura 13. Patrón de diseño alta cohesión

**Bajo Acoplamiento:** Es un patrón que tiene como principal objetivo asignar una responsabilidad para mantener el bajo acoplamiento, o sea, mantener las clases lo menos ligadas posibles. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios y también más reutilizables, acrecentando la oportunidad de una mayor productividad (45).

En el sistema propuesto el bajo acoplamiento se puede encontrar en las clases que implementan la lógica del negocio y de acceso a datos en el modelo.

Además se utiliza el patrón **GOF**:

### Decorador

Se utiliza en la plantilla *base.html.twig* que se encuentra en */app/Resources/*. Esta contiene el código HTML, las referencias a los CSS y JavaScript, elementos que son comunes en todas las vistas del sitio.



### 2.5.4 Tarjetas CRC

Las tarjetas CRC (Clases, Responsabilidad y Colaboración) permiten trabajar con una metodología basada en objetos, logrando que el equipo de desarrollo en su totalidad contribuya en la tarea del diseño. A continuación se representan las tarjetas CRC de las clases *Evidencia*, *Recomendación* y *Notificación*.

Clase: Evidencia	
Responsabilidad	Colaboración
<ul style="list-style-type: none"> <li>- Gestionar evidencia</li> <li>- Calcular IDEC por evidencia</li> <li>- Calcular IDEC por línea de investigación</li> <li>- Calcular IDEC histórico</li> <li>- Calcular IDEC por año</li> </ul>	<ul style="list-style-type: none"> <li>- Publicación</li> <li>- Registro</li> <li>- TesisTutoradas</li> <li>- Evento</li> <li>- Premio</li> <li>- Curso</li> </ul>

Tabla 8. Tarjeta CRC de la clase Evidencia

Clase: Categorización	
Responsabilidad	Colaboración
<ul style="list-style-type: none"> <li>- Categorizar usuario</li> <li>- Categorizar usuario por línea de investigación</li> </ul>	<ul style="list-style-type: none"> <li>- PerfilUsuario</li> <li>- Evidencia</li> <li>- LineaInvestigación</li> <li>- PerfilLinea</li> </ul>

Figura 14. Tarjeta CRC de la clase Recomendación

Clase: Notificación	
Responsabilidad	Colaboración

<ul style="list-style-type: none"> <li>- Notificar categorizaciones</li> <li>- Notificar cambios en el perfil de usuario</li> </ul>	<ul style="list-style-type: none"> <li>- PerfilUsuario</li> <li>- Evidencia</li> <li>- Categorización</li> </ul>
---	--

Figura 15. Tarjeta CRC de la clase Notificación

### 2.5.5 Modelo de datos relacional

El modelo de datos es un conjunto de conceptos, reglas y convenciones para describir distintos aspectos del negocio del sistema. Su función es la descripción estructurada de la organización del negocio del cliente para la construcción del software. El modelo general del sistema cuenta con un total de 37 entidades para el almacenamiento de toda la información. La categorización de los profesionales interactúa directamente con las entidades encargadas de almacenar las evidencias del usuario y con las relacionadas con las líneas de investigación.

Para la realización del Modelo Entidad-Relación asociado a la propuesta de solución se utilizaron varios patrones de diseño de bases de datos, los cuales contribuyeron a la confección de una base de datos robusta. Estos fueron:

#### Llaves subrogadas

Una clave subrogada es un identificador único que se asigna a cada registro de una tabla de dimensión. Esta clave, generalmente, no tiene ningún sentido específico de negocio. Son siempre de tipo numérico. Preferiblemente, un entero que se autoincrementa (46).

Este patrón se pone de manifiesto en la case tbEvidencia.php, como se puede observar en siguiente fragmento de código:

```
/**
 * @var integer
 *
 * @ORM\Column(name="id", type="integer")
 * @ORM\Id
 * @ORM\GeneratedValue(strategy="AUTO")
 */
private $id;
```

El patrón **Representación de Objetos como Tablas** propone definir una tabla para cada clase de objeto persistente. Los atributos de objetos que contienen tipos primitivos de datos (número, cadena, booleano y otros) se mapean en las columnas (45).

El patrón **Identificador de Objetos (IDO)** se propone asignar un ID0 a cada registro y objeto (45).

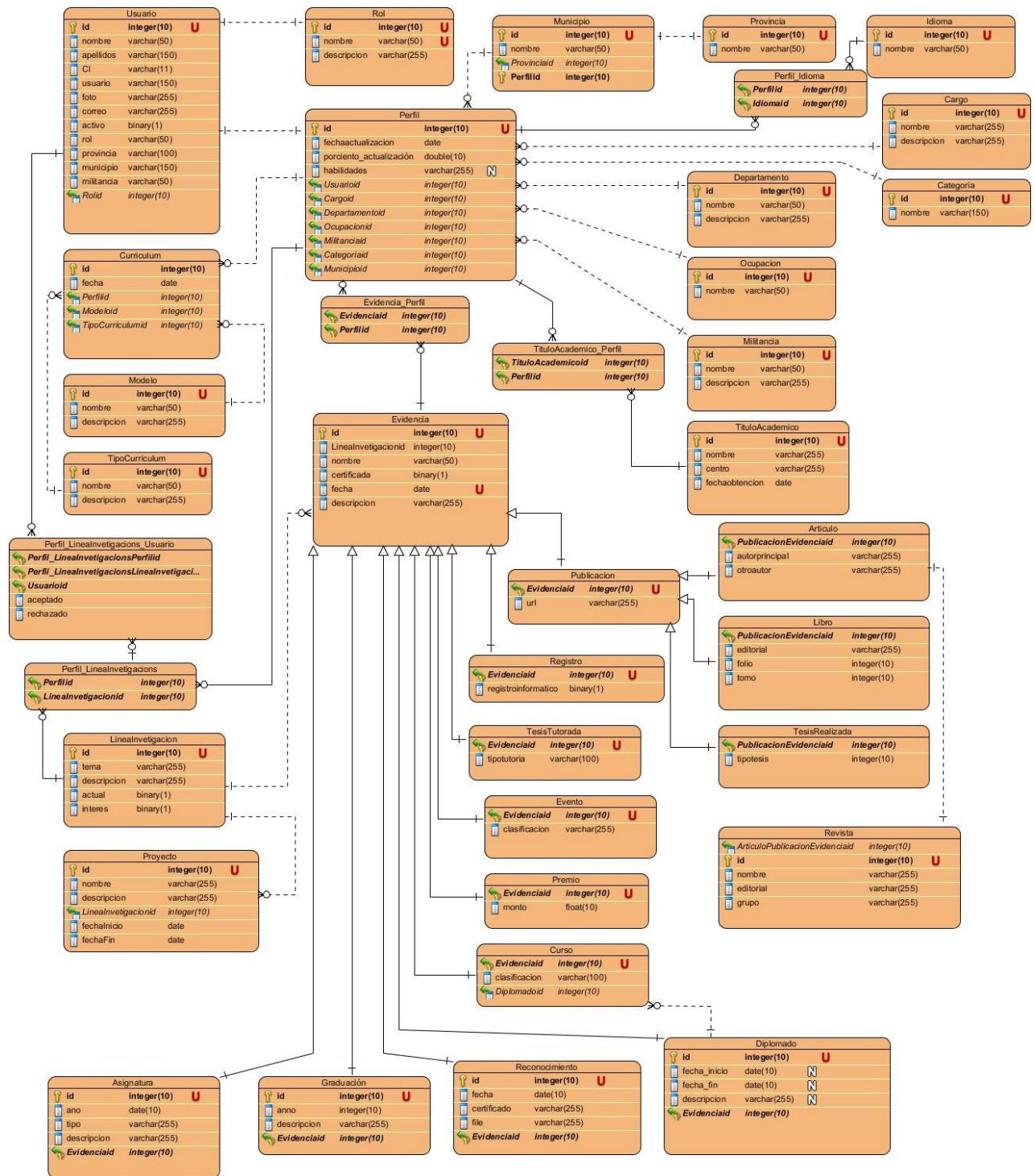


Figura 16. Modelo Entidad-Relación general

A continuación se muestran las entidades con las que interactúa directamente el sistema.

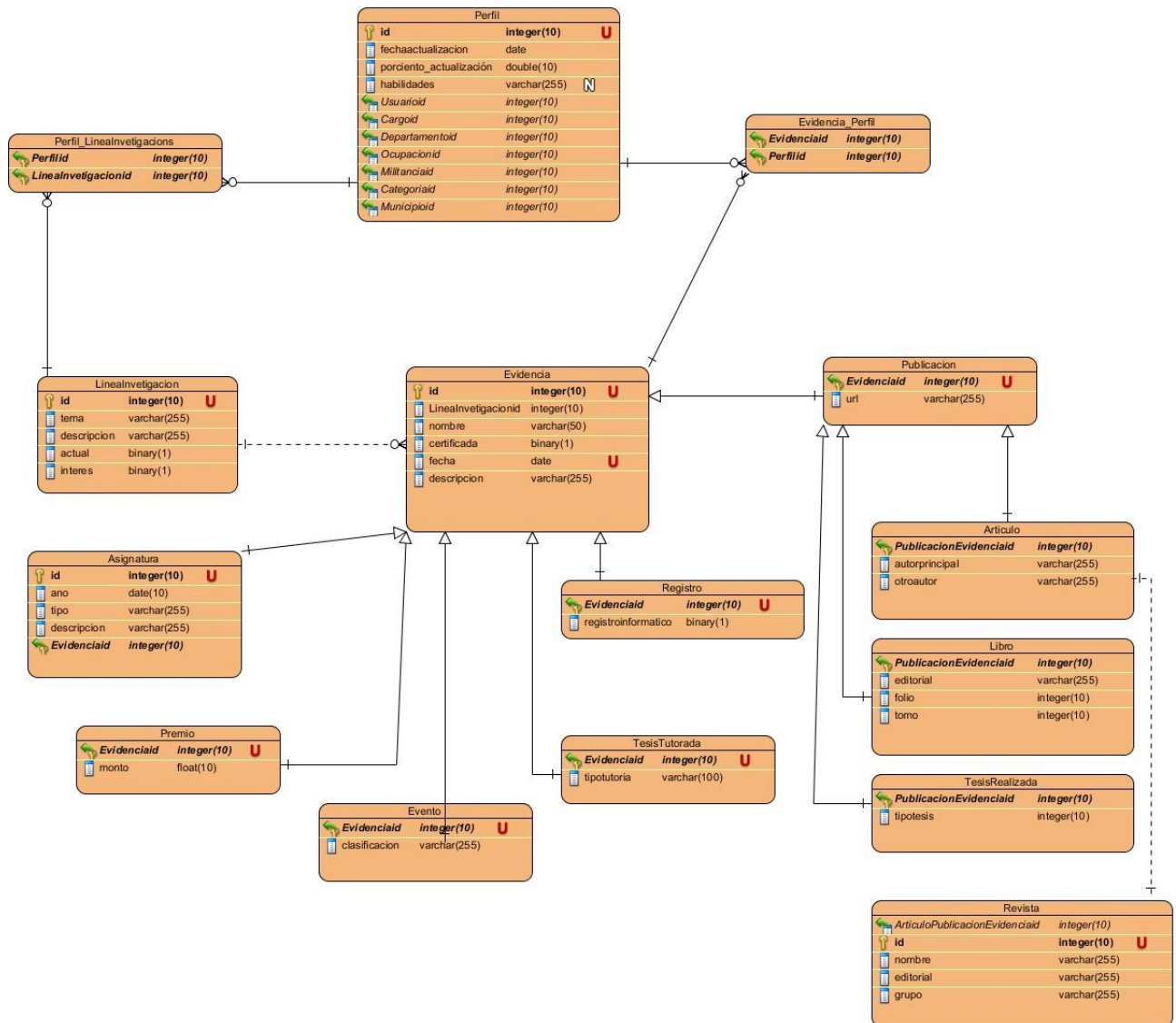


Figura 17. Entidades con las que interactúa el sistema

## 2.6 Fase de implementación

Una vez conocidas las funcionalidades que debe desempeñar el sistema y decidido cómo se van a distribuir sus componentes, se procede a implementar.

La metodología XP propone realizar una implementación iterativa, donde se obtiene al final de cada iteración un producto funcional que debe ser mostrado y probado por el cliente. Durante el transcurso de las iteraciones se realiza la implementación de las historias de usuario.

## 2.7 Estándares de codificación

Los estándares de codificación son reglas que se utilizan para estructurar el código fuente. Permite que los desarrolladores puedan interpretar de forma eficiente la estructura del código. Además aseguran la generación de código legible.

Para dar mantenimiento al Sistema de Gestión Curricular se tienen en cuenta los siguientes estándares de codificación definidos en dicho sistema:

- Los namespaces y las clases deben tener la siguiente estructura <Vendor name>(<Namespace>)\*<Class Name>.
- Todos los archivos deben tener la extensión .php.
- Los nombres de los namespaces o clases deben ser ordenadas alfabéticamente.
- Los archivos sólo deben utilizar una codificación UTF-8.
- El número de caracteres por línea deben ser de 80 columnas aunque también esta aceptado que sean hasta 120.
- Las constantes de PHP true, false y null deben estar en minúsculas.

## 2.8 Diagrama de despliegue

El diagrama de despliegue es un tipo de diagrama del Lenguaje Unificado de Modelado, que muestra las relaciones físicas de las distintas partes que componen un sistema. Describen la topología del sistema, la estructura de los elementos de hardware y sus relaciones.



Figura 18. Diagrama de despliegue

## 2.9 Diagrama de componentes

Un diagrama de componentes muestra un conjunto de componentes y sus relaciones de manera gráfica a través del uso de nodos y arcos entre estos. Puede ser visto como un tipo especial de diagrama de clases que centra su atención en los componentes físicos del sistema. Normalmente los diagramas de componentes contienen:

- Componentes
- Interfaces
- Relaciones de dependencia, generalización, asociaciones y realización
- Paquetes o subsistemas
- Instancias de algunas clases

A continuación se presenta el diagrama de componentes correspondiente a la solución propuesta:

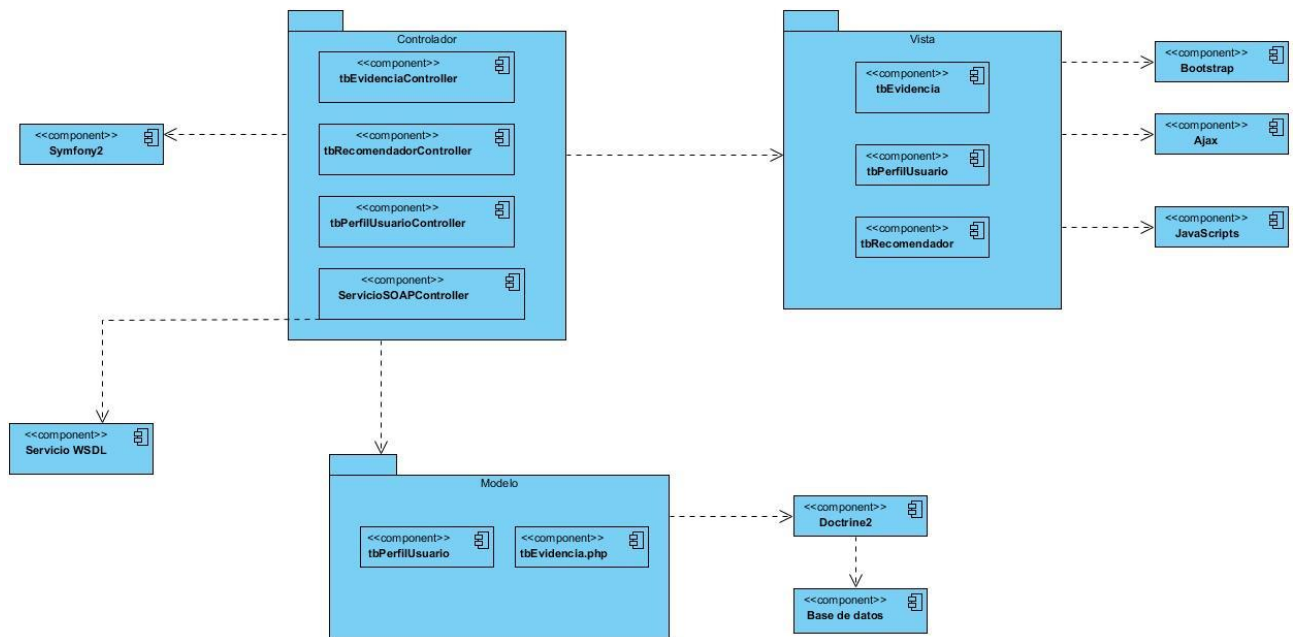


Figura 19. Diagrama de componentes

### Conclusiones parciales

Luego realizar el diseño de las funcionalidades asociadas a la evaluación y categorización de los investigadores se concluye:

- La descripción de los principales conceptos con los que interactúa el sistema y la construcción del modelo de dominio, permitió realizar el levantamiento de las funcionalidades a partir de la descripción encontrada en las historias de usuario.
- Con la identificación de los requisitos no funcionales y las principales clases del sistema se logró realizar el mantenimiento a una aplicación de software que responde a las restricciones establecidas por los usuarios.
- La definición de la arquitectura y los patrones (arquitectónicos, del diseño y de bases de datos) permitió garantizar la flexibilidad del sistema para adecuarse a entornos similares.

## CAPÍTULO 3. VALIDACIÓN DE LA PROPUESTA

### Introducción

*“Medir el progreso de la programación por líneas de código es como medir el progreso en la construcción de aviones por el peso”.*

*Bill Gates*

En el desarrollo de software las posibilidades de cometer errores son inevitables. Pueden surgir errores que se manifiesten en una mala especificación de requisitos funcionales o el uso indebido de las estructuras de datos. Estos contratiempos se pueden evitar si se lleva a cabo una actividad que garantice la calidad. Las pruebas son elementos que permiten evaluar de forma crítica el software para obtener un resultado que cumpla con las expectativas del cliente. En el presente capítulo se muestra el resultado de la aplicación de pruebas que aseguran la calidad del sistema. Se realizan pruebas para validar el diseño, utilizando las métricas tamaño operacional de las clases y relaciones entre ellas. Además se mide la calidad del software, estableciendo una estrategia de prueba y mostrando al final los resultados obtenidos.

### 3.1 Validación del diseño

Las métricas del diseño orientadas a clases permiten evaluar cuantitativamente la calidad de los atributos del software durante el desarrollo, midiendo la complejidad, funcionalidad y eficiencia de este proceso (16).

Para realizar la validación del diseño de las nuevas funcionalidades incluidas en el Sistema de Gestión Curricular se utilizaron las métricas tamaño operacional de clases y relaciones entre clases. A continuación se presentan los resultados.

#### 3.1.1 Tamaño operacional de las clases (TOC)

En el libro de Lorenz y Kidd Object-Oriented Systems, dividen las métricas basadas en clases en cuatro grupos: herencia, valores internos, valores externos y tamaño. Las métricas orientadas a tamaños para una clase se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema orientado a objetos en su totalidad (47). El promedio de los valores individuales es conocido como umbral.

Con esta métrica se puede medir la responsabilidad, complejidad de implementación y reutilización de las clases del diseño. Para esta métrica es importante que la reutilización sea inversamente proporcional a la complejidad de implementación y la responsabilidad. A mayor reutilización habrá menor responsabilidad y complejidad. Para la realización de esta métrica se determina la cantidad de atributos y de operaciones por cada clase. El resultado obtenido se denomina umbral.

Se determinaron un total de 40 clases, con un promedio de procedimientos de 11,2. A continuación se presentan los resultados obtenidos:

**Responsabilidad**

Responsabilidad	Cantidad de clases	Promedio
Baja	25	62.5
Media	10	25
Alta	5	12.5

Tabla 9. Responsabilidad

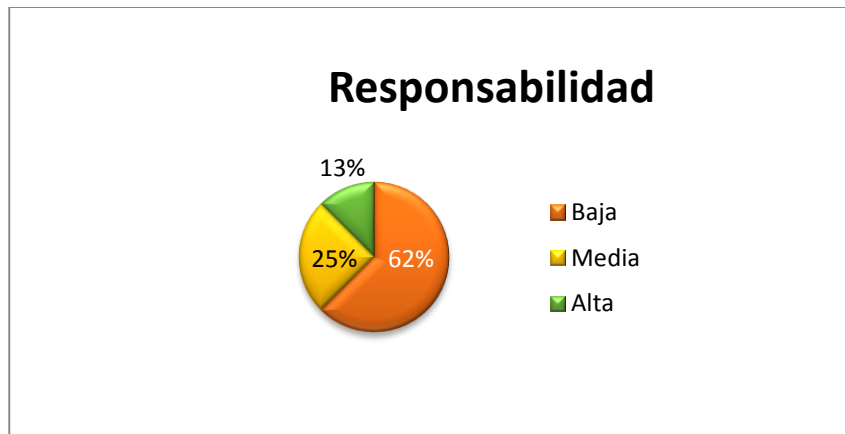


Figura 20. Comportamiento del atributo responsabilidad de la métrica TOC

**Complejidad de implementación**

Complejidad	Cantidad de clases	Promedio
Baja	25	62.5
Media	10	25
Alta	5	12.5

Figura 21. Complejidad

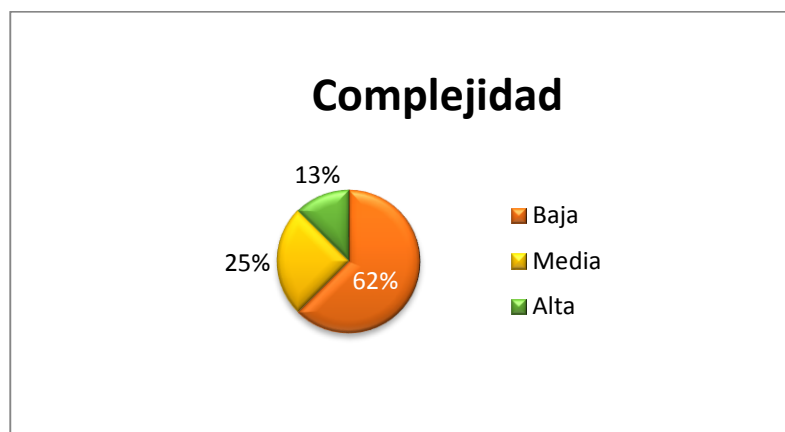


Figura 22. Comportamiento del atributo complejidad de la métrica TOC



## Reutilización

Reutilización	Cantidad de clases	Promedio
Alta	25	62.5
Media	10	25
Baja	5	12.5

Tabla 10. Reutilización

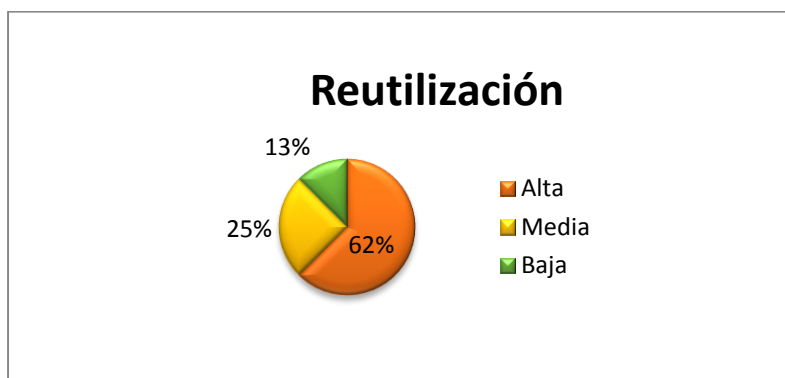


Figura 23. Comportamiento del atributo reutilización de la métrica TOC

Luego de aplicar la métrica TOC se puede concluir que la propuesta de diseño cumple con los requisitos de calidad propuestos. Como se puede observar, se obtiene como resultado que las clases contienen un porcentaje bajo de responsabilidad y complejidad, mientras que la reutilización es alta.

### 3.1.2 Relaciones entre clases (RC)

También conocida como Coupling between object classes, relaciona la noción de que un objeto está acoplado a otro si instancias de la primera clase, utilizan métodos o variables de instancia de los objetos de la otra clase. Esta situación no es deseable en el paradigma orientado a objetos por las siguientes causas (48):

- Mientras más independiente sea una clase, más fácil será para otra aplicación de reutilizar dicha clase.
- Cuanto más alto sea el acoplamiento más riguroso ha de ser las pruebas requeridas sobre las clases involucradas.
- Si hay mucha dependencia entre las clases que conforman el sistema, es fácil ver que realizar cambios en una parte tendrá amplias consecuencias. La sensibilidad ante cambios hará más difícil las tareas de mantenimiento realizadas.

La métrica RC permite evaluar el acoplamiento, la complejidad de mantenimiento, reutilización y la cantidad de pruebas para medir la calidad de una clase, teniendo en cuenta las relaciones entre ellas. El procedimiento comienza al calcular el promedio de asociaciones de uso, siendo en este caso 1,55

y con esto se determinan las categorías a las que corresponde cada atributo de calidad. Un aumento de RC provoca el aumento del acoplamiento, la complejidad de mantenimiento y la cantidad de pruebas, mientras que disminuye la reutilización.

A continuación se muestra el resultado de aplicar las métricas RC al diseño del sistema.

**Dependencia entre clases**

Criterio	Categoría	Cantidad de clases	Promedio
0 dependencias	Muy Bueno	0	0
1 dependencias	Bueno	29	72,5
2 dependencias	Regular	8	20
3 dependencias	Malo	2	5
>3 dependencias	Muy Malo	1	2.5

Tabla 11. Dependencia entre clases

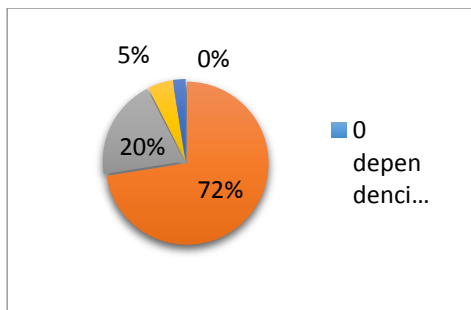


Figura 24. Comportamiento de las dependencias entre clases

**Acoplamiento**

Acoplamiento	Cantidad de clases	Promedio
Ninguno	2	0
Bajo	29	72,5
Medio	8	20
Alto	3	7,5

Tabla 12. Acoplamiento

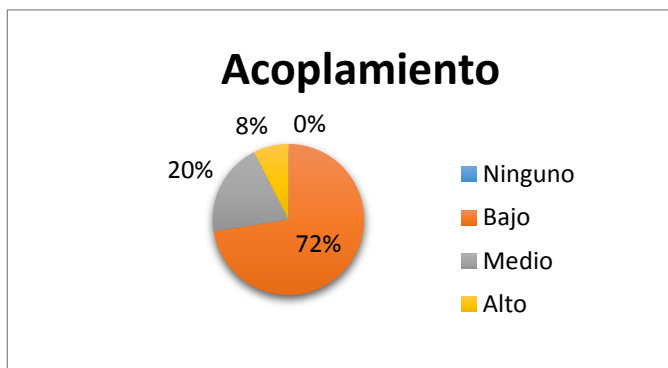


Figura 25. Comportamiento del atributo acoplamiento en RC

**Complejidad de mantenimiento**

Complejidad mant	Cantidad de clases	Promedio
Baja	28	70
Media	11	27,5
Alta	1	2,5

Tabla 13. Complejidad de mantenimiento

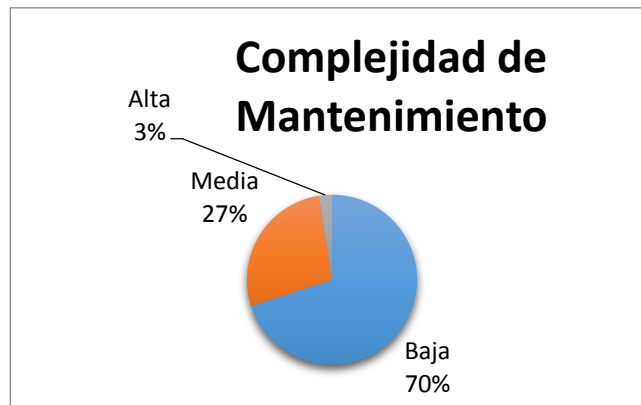


Figura 26. Comportamiento del atributo complejidad en CR

**Cantidad de pruebas**

Cant pruebas	Cantidad de clases	Promedio
Baja	28	70
Media	10	25
Alta	1	2,5

Tabla 14. Cantidad de pruebas

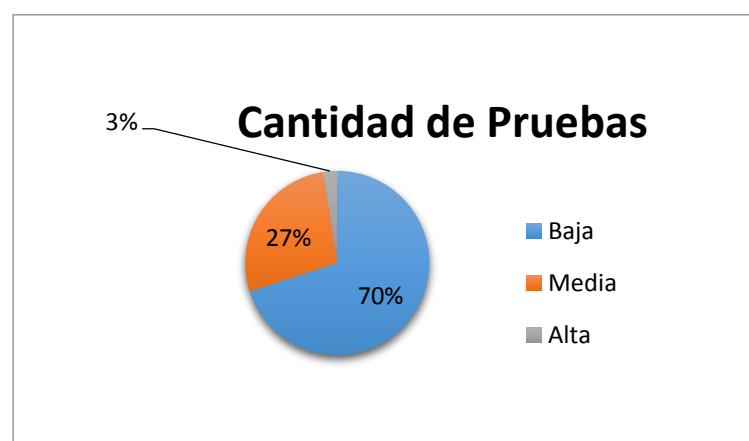


Figura 27. Comportamiento del atributo cantidad de pruebas en CR

**Reutilización**

Reutilización	Cantidad de clases	Promedio
Baja	1	2,5
Media	9	22.5
Alta	30	75

Tabla 15. Reutilización

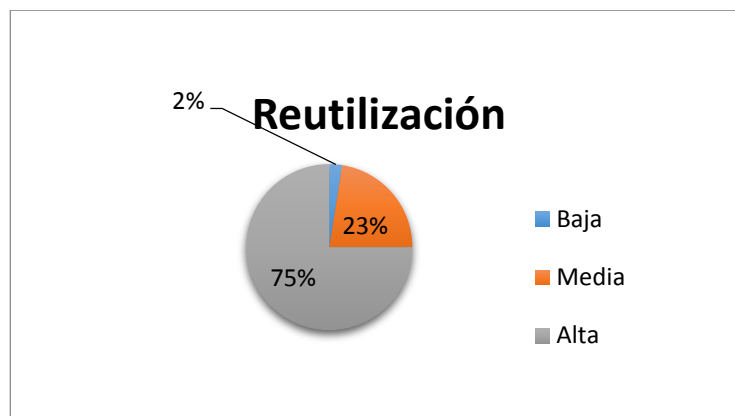


Figura 28. Comportamiento del atributo reutilización en RC

Los resultados después de aplicar la métrica RC demuestran que el 72% de las clases presentan un bajo acoplamiento. El grado de complejidad de mantenimiento no es muy alto y existe un bajo grado de esfuerzo para realizar correcciones y cambios en el mejorado Sistema de Gestión Curricular. El resultado final favorece a la reutilización.

**3.2 Pruebas**


La prueba es un conjunto de actividades que se planean con anticipación y se realizan de manera sistemática. Son el último bastión para la evaluación de la calidad y, de manera más pragmática, el descubrimiento de errores (10).

**3.2.1 Pruebas Unitarias**

Las pruebas unitarias centran su esfuerzo en la validación de la unidad más pequeña del software. Tomando como guía la descripción del diseño al nivel de componentes, se comprueban importantes caminos de control para descubrir errores dentro de los límites del módulo. Se concentran en la lógica del procesamiento interno y en la estructuras de datos (10). En el contexto Orientado a Objetos se validan las operaciones dentro de las clases. Estas pruebas garantizan que si se prueba una operación produce una salida correcta para una determinada entrada.

En la figura 29 se muestra la estructura de los directorios donde se pueden encontrar las pruebas en el proyecto. Se realizaron un total de 65 pruebas, de las cuales no se encontró no conformidades, obteniendo un 100% de validez.

- ▼ Tests
  - ▶ Controller
  - ▼ Entity
    - NmCargoTest.php
    - NmDepartamentoTest.php
    - NmIdiomaTest.php
    - NmLineaInvestigacionTest.php
    - NmMilitanciaTest.php
    - NmModeloCTest.php
    - NmMunicipioTest.php
    - NmOcupacionTest.php
    - NmProvinciaTest.php
    - NmProyectoTest.php
    - NmRevistaTest.php
    - NmTematicaTest.php
    - NmTipoCTest.php
    - tbArticuloTest.php
    - tbAsignaturaTest.php
    - tbCursoTest.php
    - tbDiplomadoTest.php
    - tbEventoTest.php
    - tbEvidenciaTest.php
    - tbGraduacionTest.php
    - tbLibroTest.php
    - tbPremioTest.php
    - tbReconocimientoTest.php
    - tbRegistroTest.php
    - tbRoleTest.php
    - tbTesisRealizadaTest.php
    - tbTesisTutoreadaTest.php
    - tbTituloAcademicoTest.php
    - tbUsuarioTest.php




```

class tbEvidenciaTest extends \PHPUnit_Framework_TestCase
{
    private $validator;

    public function setUp()
    {
        $this->validator = Validation::createValidatorBuilder()
            ->enableAnnotationMapping()
            ->getValidator();
    }

    public function testNombre()
    {
        $evidencia = new tbEvidencia();

        $listaErrores = $this->validator->validate($evidencia);
        $this->assertEquals(3, $listaErrores->count(),
            'El nombre no puede dejarse en blanco'
        );
        $error = $listaErrores[0];
        $this->assertEquals('This value should not be blank.',
            $error->getMessage());
    }
}
        
```



```

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\wamp\www\git\sigcv>$ phpunit -c app
PHPUnit 4.4.6 by Sebastian Bergmann.

Configuration read from /www/sigcv/app/phpunit.xml.dist

..... 65 / 65 (100%)

Time: 2.21 seconds, Memory 25.25Mb

OK (65 tests, 129 assertions)
        
```

Figura 29. Resultados de las pruebas unitarias utilizando PHPUnit

### Pruebas de aceptación

La metodología XP propone realizar pruebas de aceptación, donde el cliente interactúa con el software. Estas pruebas son especificadas por el cliente y se centran en las características y funcionalidades del sistema que son visibles y revisables por el cliente. Estas pruebas se definen para cada historia de usuario.

Para las pruebas de aceptación se realizaron pruebas alfa luego de cada iteración de desarrollo y una prueba beta al finalizar la implementación de todas las funcionalidades.

Las pruebas alfa se realizan en un entorno controlado. Los usuarios finales son los que aplican estas pruebas bajo la custodia de los desarrolladores. El software se utiliza de forma natural donde el desarrollador observa y registra los errores y problemas de uso (10).

La gráfica de la Figura 30 presenta los resultados de las pruebas de aceptación obtenidos al finalizar cada iteración en que estuvo dividido el desarrollo del software.

Como se puede observar, los casos de prueba fueron en ascenso en cada iteración, así como la cantidad de pruebas satisfechas.

En la primera iteración se implementaron y probaron 4 HU, de las cuales 3 fueron aceptadas satisfactoriamente. En la segunda iteración se implementaron las funcionalidades de otras 3 HU y se corrigió la que no había sido aceptada en la iteración anterior. Se realizaron 7 pruebas en total, de las cuales fueron aceptadas 6. Por último en la tercera iteración se resolvieron las no conformidades encontradas en las iteraciones anteriores y se implementaron 2 HU. Al finalizar la iteración se repitieron las pruebas anteriores, resultando 2 no satisfactorias. Finalmente se realizó una última iteración donde se resolvieron todas las no conformidades.

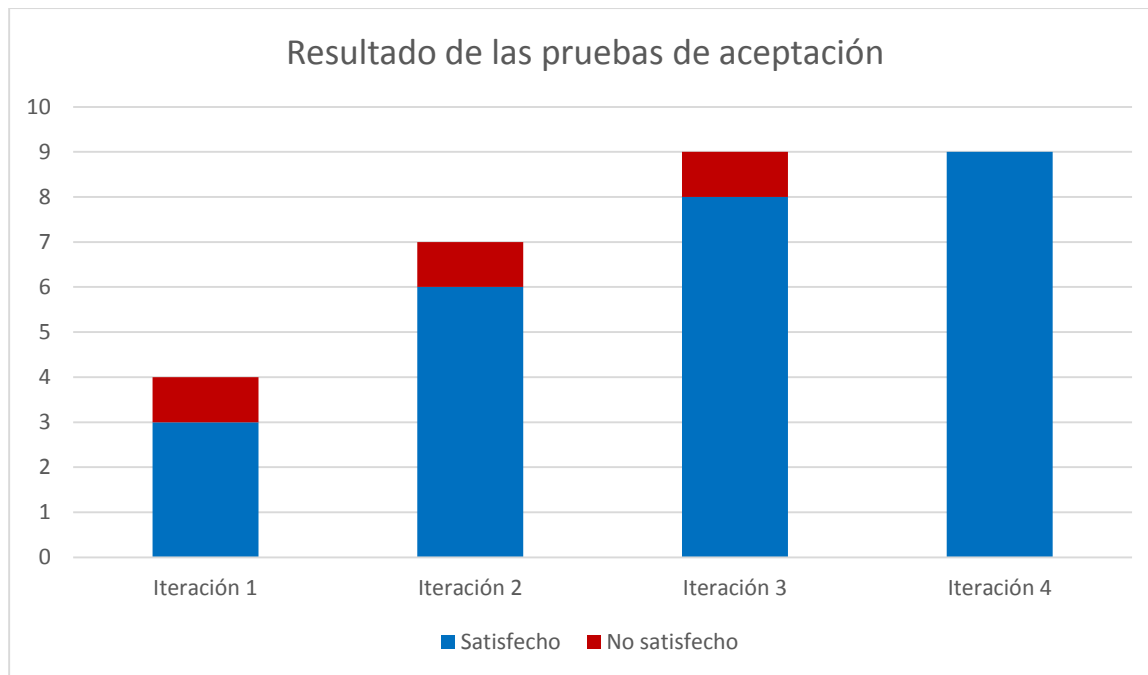


Figura 30. Resultado de las pruebas de aceptación al término de cada iteración

### 3.3 Categorización de profesionales

La categorización de profesionales permite conocer a las figuras más destacadas desde el punto de vista del resultado científico, permitiendo que los investigadores que incursan en una línea puedan tener conciencia de cuáles son las figuras más destacadas en el ámbito. Esto puede ayudar a los investigadores a establecer estrategias para adquirir nuevos conocimientos y contrastar sus ideas de investigación. De igual forma los directivos administrativos pueden contar con información irrefutable acerca de la producción científica de los investigadores de la Universidad.

Para validar la solución propuesta se desarrolló un caso de estudio, donde se arrojaron los siguientes resultados:

La evidencia es el elemento fundamental del Sistema de Gestión Curricular. A cada evidencia le corresponde un valor dependiendo de su tipo y del nivel de participación del autor. La figura 31 muestra el listado de evidencias que recoge un profesional en su perfil y sus valores correspondientes. El valor de las evidencias que no se han certificado no tributa a la evaluación del desempeño, pero de igual manera el usuario puede verificar su valor.

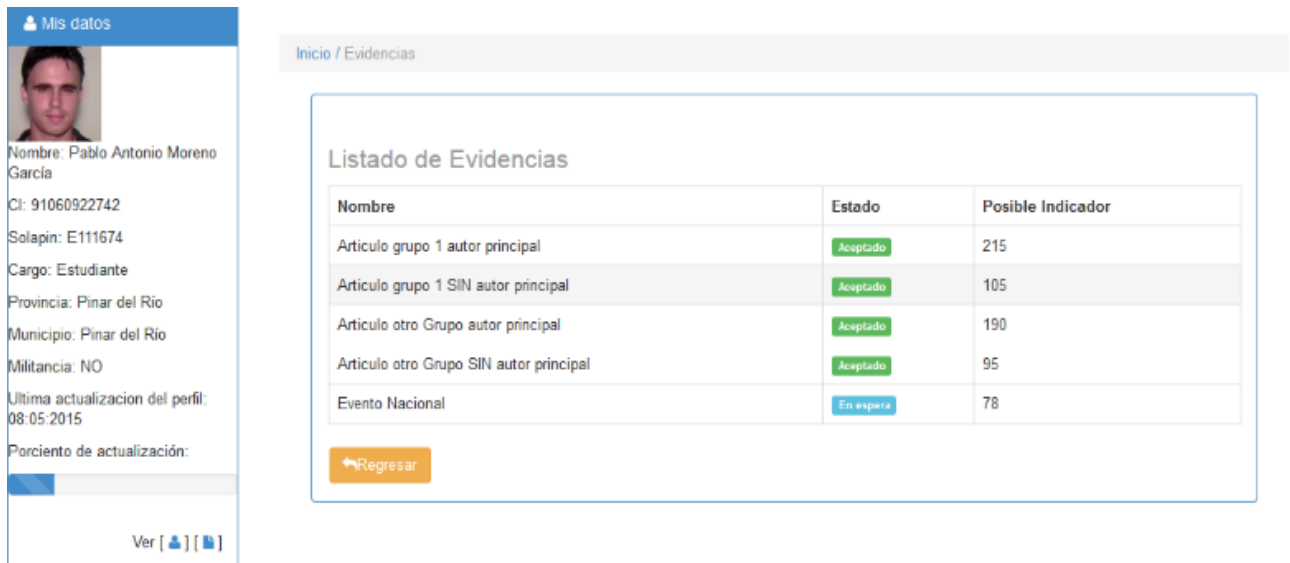


Figura 31. Listado de evidencias evaluadas

Por otro lado, la evaluación de la producción de los profesionales atendiendo a las líneas de investigación en las que se han desempeñado, permite realizar un análisis del desarrollo en cada una de las líneas, así como establecer una categorización del desarrollo profesional por cada una. En la figura 32 se muestra el análisis de la producción de un profesional por las líneas desarrolladas y en la figura 33 la categorización de los profesionales por líneas de investigación.

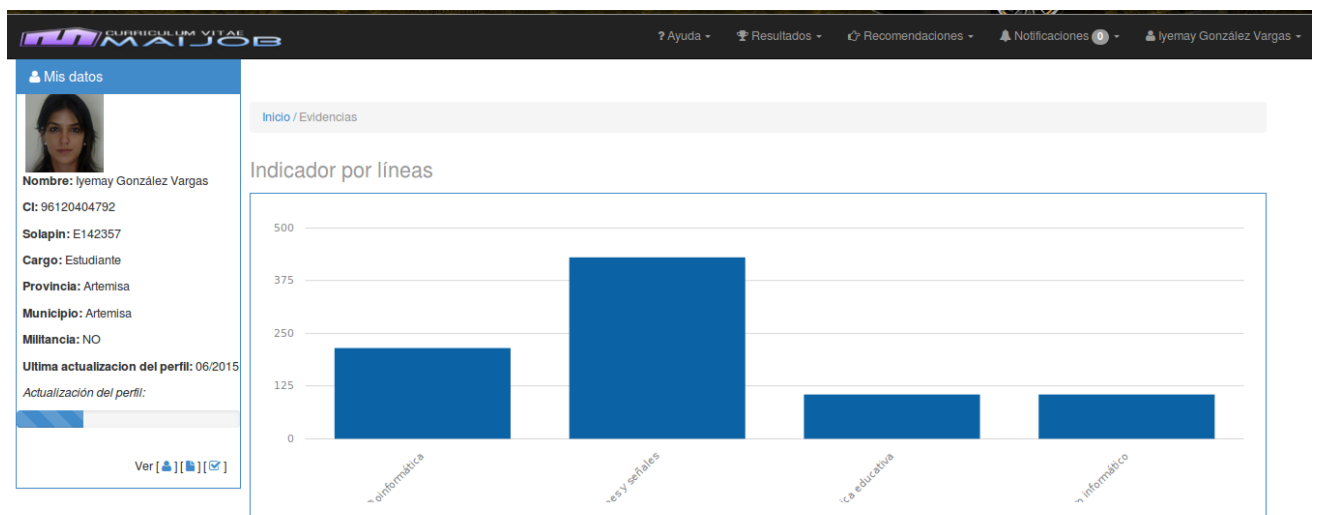


Figura 32. Desempeño de un profesional por líneas desarrolladas



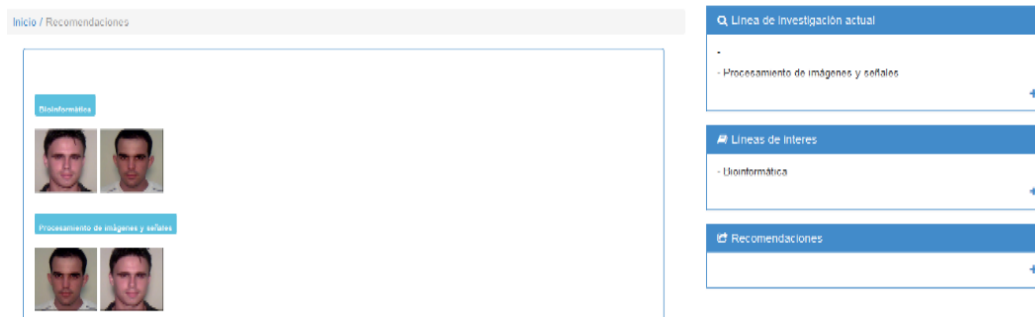


Figura 33. Categorización de profesionales por líneas de investigación

La evaluación de la producción anual se utiliza para analizar el desempeño productivo de los profesionales a lo largo de un año. En la figura 34 se muestra el resultado del análisis de la productividad.

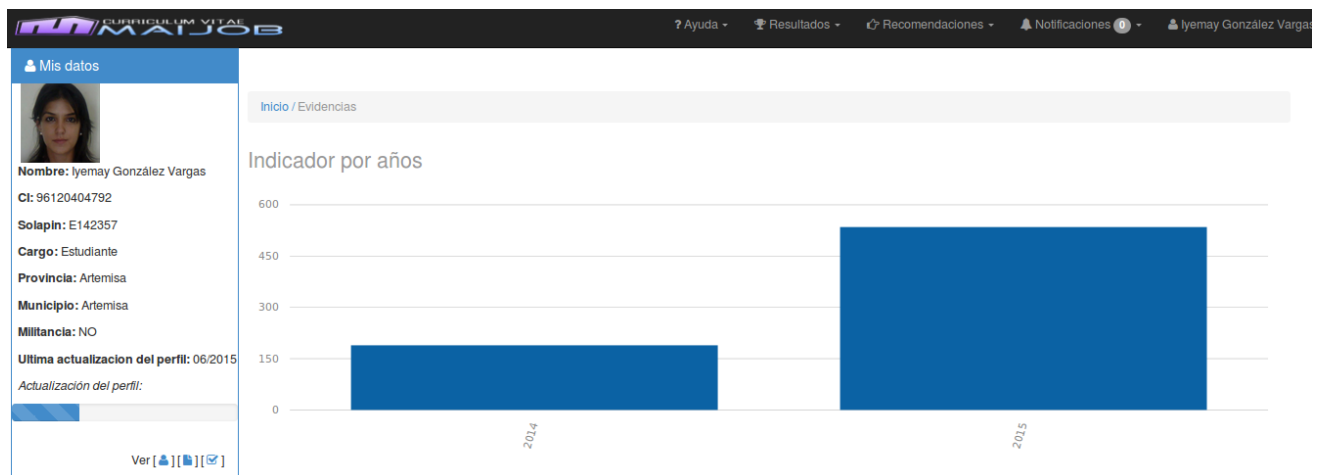


Figura 34. Análisis de la producción por años

La evaluación de la producción también permite categorizar de forma general a los profesionales, contribuyendo a la toma de decisiones por parte de los directivos a la hora de identificar a los profesionales que mayor producción hayan alcanzado.

### Conclusiones parciales

Con la validación de la solución en la Facultad 3 de la Universidad de las Ciencias Informáticas se puede concluir:

- Mediante las métricas TOC y RC se validó el diseño, obteniéndose un diseño de clases sencillo. Con la métrica TOC se obtuvo como resultado que el 62% de las clases poseen una alta reutilización, mientras que las RC arrojaron que el 72% de las clases poseen un bajo acoplamiento, con un 70% de bajo mantenimiento y un 75% de reutilización alta.
- Con la implantación de pruebas de aceptación se encontraron un total de 4 HU no aceptadas,

las cuales fueron resueltas en la próxima iteración.

- La categorización de los profesionales a partir de la evaluación de la producción permitió validar la propuesta de solución.

## **CONCLUSIONES**

- Con el estudio de los tipos de mantenimiento y el análisis de las necesidades del usuario se decidió que con realizar un mantenimiento perfectivo sobre el Sistema de Gestión Curricular que incorpore los nuevos requerimientos del usuario.
- El análisis de las diferentes metodologías y tecnologías permitió elegir las más propicias para el desarrollo del mantenimiento perfectivo. Utilizando la metodología XP se mantuvo un proceso de desarrollo de software satisfactorio, obteniendo como resultado un sistema que se ajusta a las necesidades del cliente.
- Con la implementación se logró incluir nuevas funcionalidades al Sistema de Gestión Curricular de la Facultad 3 que categoriza los perfiles de usuario que más se hayan destacado en la producción por cada una de las líneas de investigación preferidas por el usuario.
- Las validaciones y pruebas realizadas sobre el diseño y la implementación del sistema demostraron que la solución obtenida cumple con la calidad requerida por el cliente.

## RECOMENDACIONES

- La incorporación de nuevas variables al algoritmo: “Indicador de desempeño de la actividad científico-técnica”, que permita ampliar el espectro de actividades que tributen a medir la experiencia de los profesionales.

## REFERENCIAS BIBLIOGRÁFICAS

1. *Producción científica de Cuba: una perspectiva desde la obra de dos mujeres académicas*. Contreras, Dr. C. Alberto J. Dorta y Díaz, Dr. C. Liliam Álvarez. 5, La Habana : ECIMED, 2007, Vol. 16.
2. Prat, Anna María. La importancia de medir la producción científica. *Red de indicadores de ciencia y tecnología*. [En línea] 2003. [Citado el: 6 de 06 de 2015.] [http://www.ricyt.org/manuales/doc\\_view/137-la-importancia-de-medir-la-produccion-cientifica](http://www.ricyt.org/manuales/doc_view/137-la-importancia-de-medir-la-produccion-cientifica).
3. Universidad Europea. Biblioteca CRAI Dulce Chacón. *Evaluación de la producción científica*. [En línea] Universidad Europea. [Citado el: 6 de 06 de 2015.] <http://uecrai.universidadeuropea.es/ayuda/index.php/la-evaluacion-de-la-produccion-cientifica>.
4. Real Academia Española. Real Academia Española. [En línea] [Citado el: 7 de 06 de 2015.] <http://buscon.rae.es/drae/srv/search?id=TnvapsX68DXX26WkzIO7>.
5. Definición ABC. Definición ABC. *Definición ABC*. [En línea] [Citado el: 7 de 06 de 2015.] <http://www.definicionabc.com/general/categoria.php>.
6. *Evaluación de la producción científica como instrumento para el desarrollo de la ciencia y la tecnología*. Prat, Anna María. 4, La Habana : ACIMED, 2001, Vol. 9. ISSN 1024-9435.
7. González, Danay Serrano. *Procedimiento para la gestión del mantenimiento de software del sub-sistema Planificación del Sistema Integral de Gestión, Cedrux*. Habana : s.n., 2010.
8. Sommerville, Ian. *Ingeniería del Software. Séptima Edición*. Madrid : Pearson Educación, 2005. ISBN: 84-7829-074-5.
9. Párraga, Juan Angel Martínez. *ESTÁNDAR IEEE 1219 DE MANTENIMIENTO DEL SOFTWARE*. 1999.
10. Pressman, Roger S. *Software Engineering a practitioner's approach. 6th Edition*.
11. Software para soluciones empresariales. *Software para soluciones empresariales*. [En línea] [Citado el: 7 de 06 de 2015.] <http://www.aesist.com/soporte>.
12. Universidad de Valencia Departamento de Informática. Universidad de Valencia Departamento de Informática. *Universidad de Valencia Departamento de Informática*. [En línea] [Citado el: 7 de 06 de 2015.] [informatica.uv.es/iiguia/2000/IPI/material/tema7.pdf](http://informatica.uv.es/iiguia/2000/IPI/material/tema7.pdf).
13. Dirección de Investigación de la Universidad de las Ciencias Informáticas. Reykernel Izquierdo Herrera, Raynel Batista. *Indicador del Desempeño Científico Tecnológico*. Habana, Cuba : s.n., 2014.

14. *The Role of Deliberate Practice in the Acquisition of Expert Performance*. K.AndersEricsson, RalfTh.Krampe y ClemensTesch-Römer. 3, s.l. : American Psychological Association, 1993, Vol. 100.
15. Pérez, María José Pérez. *Guía Comparativa de Metodologías Ágiles*. s.l. : Universidad de Valladolid.
16. Alitimón, Arletis Francis y Serrano, Nicole Almenares. *Portal Web del Ministerio de . La Habana* : s.n., 2014.
17. Joskowicz, José. *Reglas y Prácticas en eXtreme Programming*.
18. Q, JOSE M. BAUTISTA. *PROGRAMACIÓN EXTREMA*. s.l. : UNIVERSIDAD UNION BOLIVARIANA.
19. ConML. *ConML*. [En línea] [Citado el: 8 de 02 de 2015.] <http://www.conml.org/FAQ.aspx>.
20. Unified Modeling Language™ (UML®) Resource Page. *Unified Modeling Language™ (UML®) Resource Page*. [En línea] [Citado el: 8 de 02 de 2015.] <http://www.uml.org/>.
21. López, Pascual González, López, Ana Amelia González y Lázaro, José Antonio Gallud. *Herramientas CASE. ¿Cómo incorporarlas con éxito en nuestra organización?* Castilla : s.n.
22. Visual Paradigm. [En línea] [Citado el: 8 de 02 de 2015.] <http://www.visual-paradigm.com/aboutus/>.
23. Muñoz, Yeni Ferrer y Lara, Daniel Rafael Acevedo. *Agente Inteligente en un Entorno Virtual Inmersivo 3D en función del trabajo con los estilos de aprendizajes de los usuarios en los Centros de Autoaprendizaje y Servicios de Idiomas Extranjeros*. La Habana : s.n., 2013.
24. PHP: Hipertext Preprocesor. [En línea] febrero de 2011. <http://php.net/manual/es/intro-what-is.php..>
25. *ESTUDIO COMPARATIVO DE MARCOS DE TRABAJO PARA EL DESARROLLO SOFTWARE ORIENTADO A ASPECTOS*. Guerrero, Carlos A., y otros. 2, La Habana : s.n., 2014, Vol. 25. ISSN 0718-0764.
26. Symfony 1.4, la guía definitiva. [En línea] [Citado el: 2015 de 02 de 08.] [http://librosweb.es/libro/symfony\\_1\\_4/capitulo\\_1/symfony\\_en\\_pocas\\_palabras.html](http://librosweb.es/libro/symfony_1_4/capitulo_1/symfony_en_pocas_palabras.html).
27. OpenWebCMS. [En línea] <http://openwebcms.es/2013/que-es-bootstrap/>.
28. getBootstrap. [En línea] <http://getbootstrap.com/>.

29. 25trabajode Scridb Visual Basic 2. [En línea] [Citado el: 9 de 2 de 2015.] <http://es.scribd.com/doc/61498499/25trabajode-Scridb-Visual-Basic-2>.
30. NetBeans IDE. [En línea] [Citado el: 9 de 2 de 2015.] <http://netbeans-ide.softonic.com/>.
31. Hill, McGraw. *Sistemas Gestores de Bases de Datos*.
32. PostgreSQL-es. [En línea] 02 de 10 de 2010. [Citado el: 09 de 02 de 2015.] [http://www.postgresql.org.es/sobre\\_postgresql](http://www.postgresql.org.es/sobre_postgresql).
33. Ubuntu, Guía. Guía Ubuntu. [En línea] marzo de 2008. [http://www.guia-ubuntu.org/index.php?title=PgAdmin\\_III](http://www.guia-ubuntu.org/index.php?title=PgAdmin_III).
34. Mundo de Linux. *Mundo de Linux*. [En línea] [Citado el: 2015 de 02 de 09.] <http://www.elmundolinux.com/apachelinux.php>.
35. Asenjo, Jorge Sánchez. *Servidores de Aplicaciones Web*. 2012.
36. Empezando - Acerca del control de versiones. [En línea] [Citado el: 08 de 02 de 2015.] <https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones>.
37. Cedeño, Elvis Ferrera. *Sistema de recomendación basado en las preferencias de los* . 2011.
38. *Obtención de Requerimientos. Técnicas y Estrategia*. Guerra, César Arturo. 17, s.l. : Software Guru, 2007.
39. Cohn, M. Mountain Goat Software. [En línea] <http://www.mountaingoatsoftware.com/topics/user-stories>.
40. Joskowicz, José. El Instituto de Ingeniería Eléctrica (IIE). *Reglas y Prácticas en Extreme Programming*. [En línea] 10 de 2 de 2008. [Citado el: 8 de 04 de 2015.] <http://ie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf>.
41. Letelier, Patricio y Penadés, M<sup>a</sup> Carmen. *Metodologías ágiles para el desarrollo de software: eXtreme Programming*. Valencia : Universidad Politécnica de Valencia.
42. *Arquitectura de Software*. Cervantes, Humberto. 27, s.l. : SG Buzz, 2010.
43. Kruchten, Philippe. *Architectural Blueprints. The "4+1" View Model of Software Architecture* .
44. Fabien Potencier, François Zaninotto. *Symfony 1.1, la guía definitiva*. [En línea] 2009. [Citado el: 09 de 04 de 2015.] [http://www.librosweb.es/symfony\\_1\\_1](http://www.librosweb.es/symfony_1_1).
45. Larman, Craig. *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado*. 2da Edición.

46. Claves subrogadas. [En línea] [Citado el: 05 de 05 de 2015.]  
<http://www.businessintelligence.info/serie-dwh/claves-subrogadas.html>.
47. Sánchez, Ana María García. *Evaluación de métricas de calidad del software sobre un programa Java*. Madrid : Universidad Complutense de Madrid, 2010.
48. Soldado, Rosana Montes. *Métricas aplicables al Diseño Orientado a Objetos*. Granada : s.n., 2000.



## ANEXOS

### Anexo 1 Variables del algoritmo IDEC

#### a) *Publicaciones (PUB)*

Se otorgarán puntos por cada artículo publicado en que el investigador aparezca como autor principal u otro tipo de autor, dependiendo además del grupo de la revista donde fue publicada, quedando la ponderación:

Si es un autor principal:

Grupo de revista	Ponderación
Publicaciones indizadas no incluidas en el grupo del MES	5
Revistas del grupo 4	10
Revistas del grupo 3	15
Revistas del grupo 2	20
Revistas del grupo 1	25

Tabla 16. Ponderación de la variable artículo para autor principal

Si es otro tipo de autor

Grupo de revista	Ponderación
Publicaciones indizadas no incluidas en el grupo del MES	3
Revistas del grupo 4	5
Revistas del grupo 3	7
Revistas del grupo 2	10
Revistas del grupo 1	12

Tabla 17. Ponderación de la variable artículo para otro tipo de autor

#### b) *Proyecto (PROY)*

Se otorgarán puntos por cada proyecto en que participe el investigador (de acuerdo al listado oficial emitido por la Dirección de Ciencia y Técnica de la UCI). Se tiene en cuenta el rol que tuvo en el proyecto:

Jefe

	Ponderación
<b>Institucionales</b>	6
<b>Conjuntos con otros CES o centros de investigación</b>	8
<b>Conjuntos con empresas o entidades de la producción o los servicios o demandados por estas (Empresariales)</b>	20
<b>De Programas Ramales o Territoriales o PNAP de ese nivel</b>	24
<b>De Programas Nacionales o PNAP de ese nivel</b>	30
<b>Internacionales</b>	40

Tabla 18. Ponderación para la variable proyecto para el rol de jefe

Participante principal

	Ponderación
<b>Institucionales</b>	4
<b>Conjuntos con otros CES o centros de investigación</b>	6
<b>Conjuntos con empresas o entidades de la producción o los servicios o demandados por estas (Empresariales)</b>	16
<b>De Programas Ramales o Territoriales o PNAP de ese nivel</b>	20
<b>De Programas Nacionales o PNAP de ese nivel</b>	24
<b>Internacionales</b>	30

Tabla 19. Ponderación para la variable proyecto para el rol de participante principal

Colaborador

	Ponderación
<b>Institucionales</b>	2
<b>Conjuntos con otros CES o centros de investigación</b>	4
<b>Conjuntos con empresas o entidades de la producción o los servicios o demandados por estas (Empresariales)</b>	12
<b>De Programas Ramales o Territoriales o PNAP de ese nivel</b>	16

<b>De Programas Nacionales o PNAP de ese nivel</b>	20
<b>Internacionales</b>	24

Tabla 20. Ponderación para la variable proyecto para el rol de colaborador

c) *Maestrías y Doctorados (MAES , DOC)*

Se otorgarán puntos a los investigadores que hayan defendido exitosamente el doctorado o maestría, además de si han sido tutores de defensas exitosas de doctorado o maestría.

Doctorado

	Ponderación
<b>Defensa</b>	50
<b>Tutoría</b>	30

Tabla 21. Ponderación para la variable doctorado

Maestría

	Ponderación
<b>Defensa</b>	30
<b>Tutoría</b>	20

Tabla 22. Ponderación para la variable maestría

1. *Premios*

Se otorgarán puntos por cada premio recibido y en consideración de si fue un autor principal u otro tipo de autor.

Autor principal

	Ponderación
<b>Nivel UCI</b>	5
<b>MES u otro OACE ( se incluyen los Sellos forjadores del futuro)</b>	10
<b>ACC</b>	15
<b>Premio de la innovación (CITMA)</b>	20
<b>Internacionales</b>	30

Tabla 23. Ponderación de la variable premio para el autor principal

Otro tipo de autor

	Ponderación
<b>Nivel UCI</b>	3
<b>MES u otro OACE ( se incluyen los Sellos forjadores del futuro)</b>	5
<b>ACC</b>	7
<b>Premio de la innovación (CITMA)</b>	10
<b>Internacionales</b>	15

Tabla 24. Ponderación de la variable premio para otro tipo de autor

d) *Introducción de resultados (IRES)*

Se otorgarán puntos por cada aval obtenido que certifique la participación del investigador según el tipo de autor y en:

Autor principal

	Ponderación
<b>Un resultado con impacto a nivel de empresa , entidad o grupo de empresas</b>	10
<b>Un resultado con impacto a nivel ramal o territorial</b>	15
<b>Un resultado con impacto a nivel nacional</b>	20
<b>Un resultado con impacto internacional</b>	30

Tabla 25. Ponderación de la variable introducción de resultados para el autor principal

Otro tipo de autor

	Ponderación
--	-------------

<b>Un resultado con impacto a nivel de empresa , entidad o grupo de empresas</b>	5
<b>Un resultado con impacto a nivel ramal o territorial</b>	7
<b>Un resultado con impacto a nivel nacional</b>	10
<b>Un resultado con impacto internacional</b>	15

Tabla 26. Ponderación de la variable introducción de resultados para otro tipo de autor

e) *Registros*

Se otorgarán puntos por cada uno de los registros (patentes, software, normas, marcas y otros) que según el tipo de autor y la forma en cuanto a informatización que tenga el mismo

Autor principal

	<b>Ponderación</b>
<b>No informáticos</b>	15
<b>Informáticos</b>	25

Tabla 27. Ponderación de la variable registros para el autor principal

Otro tipo de autor

	<b>Ponderación</b>
<b>No informáticos</b>	7
<b>Informáticos</b>	12

Tabla 28. Ponderación de la variable registros para otro tipo de autor

f) *Evento*

Se otorgaran los puntos por cada participación en el que el investigador aparezca como autor u alguna otra variante.

Autor principal

<b>Alcance</b>	<b>Ponderación</b>
<b>Municipal</b>	3

<b>Provincial</b>	6
<b>Nacionales</b>	9
<b>Internacional en Cuba</b>	12
<b>Internacional en el extranjero</b>	15
<b>1er Nivel en el extranjero</b>	18

Tabla 29. Ponderación de la variable evento para el autor principal

Otro tipo de autor

<b>Alcance</b>	<b>Ponderación</b>
<b>Municipal</b>	1
<b>Provincial</b>	3
<b>Nacionales</b>	5
<b>Internacional en Cuba</b>	6
<b>Internacional en el extranjero</b>	7
<b>1er Nivel en el extranjero</b>	9

Tabla 30. Ponderación de la variable evento para otro tipo de autor

*g) Moneda Nacional (MN)*

Se otorga puntos cuando el investigador haya participado durante el año en servicios, proyectos u otras acciones que hayan aportado al centro en metálico, equipos o insumos un total de:

Autor principal

<b>Cantidad abonada</b>	<b>Ponderación</b>
<b>Hasta 2000 cup</b>	10
<b>Entre 2000 y 5000 cup</b>	15
<b>Superior a 5000 cup</b>	20

Tabla 31. Ponderación de la variable moneda nacional para el autor principal

Otro tipo de autor

<b>Cantidad abonada</b>	<b>Ponderación</b>
-------------------------	--------------------

<b>Hasta 2000 cup</b>	5
<b>Entre 2000 y 5000 cup</b>	7
<b>Superior a 5000 cup</b>	10

Tabla 32. Ponderación de la variable moneda nacional para otro tipo de autor

*h) Moneda convertible (MC)*

Se otorgan puntos cuando el investigador haya participado en el año en servicios, proyectos u otras acciones que hayan aportado al centro un total en el año de:

Autor principal

<b>Cantidad abonada</b>	<b>Ponderación</b>
<b>Hasta 500 cuc</b>	10
<b>Entre 500 y 1000 cuc</b>	15
<b>Superior a 1000 cuc</b>	20

Tabla 33. Ponderación de la variable moneda convertible para el autor principal

Otro autor

<b>Cantidad abonada</b>	<b>Ponderación</b>
<b>Hasta 500 cuc</b>	5
<b>Entre 500 y 1000 cuc</b>	7
<b>Superior a 1000 cuc</b>	10

Tabla 34. Ponderación de la variable moneda convertible para otro tipo de autor

## Anexo 2 Historias de usuario

Historia de usuario	
Número: 1	Usuario:
Nombre historia: Ponderar el perfil de usuario por año	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alta
Puntos estimados: 3	Iteración asignada: 1
Programador responsable: Laura Beatriz – Pablo Moreno	
<p><b>Descripción:</b></p> <p>El sistema debe ser capaz de ponderar los currículos basados en evidencia confirmada en el año actual.</p>	

Tabla 35. HU Ponderar perfil de usuario por año

Historia de usuario	
Número: 2	Usuario:
Nombre historia: Ponderar perfil histórico de usuario	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alta
Puntos estimados: 4.5	Iteración asignada: 1
Programador responsable: Laura Beatriz – Pablo Moreno	
<p><b>Descripción:</b></p> <p>El sistema debe ser capaz de ponderar los perfiles de usuario basados en evidencia confirmada en todo el currículo.</p>	

Tabla 36. HU Ponderar perfil histórico de usuario



<b>Número: 4</b>	<b>Usuario:</b>		
<b>Nombre historia: Generar categorización a partir del IDEC</b>			
<b>Prioridad en negocio:</b> Alta		<b>Riesgo en desarrollo:</b> Alta	
<b>Puntos estimados: 8</b>		<b>Iteración asignada: 2</b>	
<b>Programador responsable: Laura Beatriz – Pablo Moreno</b>			
<b>Descripción:</b> El sistema realiza la categorización de perfiles que tengan el mayor IDEC de la red			

Tabla 37. HU Generar categorización a partir del IDEC

Historia de usuario			
<b>Número: 5</b>	<b>Usuario:</b>		
<b>Nombre historia: Generar categorización a partir de asignaturas impartidas</b>			
<b>Prioridad en negocio:</b> Alta		<b>Riesgo en desarrollo:</b> Alta	
<b>Puntos estimados: 6</b>		<b>Iteración asignada: 2</b>	
<b>Programador responsable: Laura Beatriz – Pablo Moreno</b>			
<b>Descripción:</b> El sistema categoriza a los profesores que mayor experiencia hayan alcanzado impartiendo clases.			

Tabla 38. HU Generar categorización a partir de asignaturas impartidas

Historia de usuario	
<b>Número: 7</b>	<b>Usuario:</b>
<b>Nombre historia: Mostrar usuario que sigue</b>	
<b>Prioridad en negocio:</b> <b>Media</b>	<b>Riesgo en desarrollo:</b> <b>Media</b>
<b>Puntos estimados: 3</b>	<b>Iteración asignada: 3</b>
<b>Programador responsable: Laura Beatriz – Pablo Moreno</b>	
<b>Descripción:</b> <b>Debe ser capaz de mostrar a las personas que sigue un usuario.</b>	

Tabla 39. HU Mostrar usuarios que se siguen

Historia de usuario	
<b>Número: 8</b>	<b>Usuario:</b>
<b>Nombre historia: Notificar a un usuario sobre cambios en el perfil que sigue</b>	
<b>Prioridad en negocio:</b> <b>Baja</b>	<b>Riesgo en desarrollo:</b> <b>Bajo</b>
<b>Puntos estimados: 5</b>	<b>Iteración asignada: 3</b>
<b>Programador responsable: Laura Beatriz – Pablo Moreno</b>	
<b>Descripción:</b> <b>Se debe notificar a un usuario cuando el perfil que este sigue introduzca una nueva evidencia, o realice cambios en las líneas de investigación de su preferencia.</b>	

Tabla 40. HU Notificar a una usuario sobre cambios en un perfil