PROPUESTA DE UN MODELO DE ENSEÑANZA/APRENDIZAJE DE INGENIERÍA DEL SOFTWARE APOYADO EN TIC

MARTHA SUSANA CONTRERAS ORTIZ

Ingeniera de Sistemas

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS

ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

BUCARAMANGA

2015

PROPUESTA DE UN MODELO DE ENSEÑANZA/APRENDIZAJE DE INGENIERÍA DEL SOFTWARE APOYADO EN TIC

ING. MARTHA SUSANA CONTRERAS ORTIZ

Trabajo de Investigación presentado para optar al título de Magíster en Ingeniería de Sistemas e Informática

Director:
Fernando Antonio Rojas Morales
Magíster en Ciencias Computacionales

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

DEDICATORIA

En primer lugar dedico este logro a Dios, por ser mi guía y motor para no desfallecer a pesar de los inconvenientes.

A mis padres, Eduardo y Martha, por su amor y voces de aliento. Por creer siempre en mí.

A mis hermanas, Sonia y Silvia, por su amor, apoyo y ayuda constante. Por ser mi ejemplo a seguir.

AGRADECIMIENTOS

A la UNIVERSIDAD INDUSTRIAL DE SANTANDER por apoyar mi proceso de formación profesional y personal.

A FERNANDO ANTONIO ROJAS MORALES, director de este proyecto, por su apoyo constante, colaboración, dedicación y comprensión.

A MIS PADRES Y HERMANAS, por su amor, voces de aliento, ayuda incondicional y comprensión en todo momento.

A MARÍA CRISTINA MENDOZA, MÓNICA MANTILLA Y GLADYS NORIEGA, por su colaboración, disposición, cariño y estar pendientes de mi proceso en cada etapa.

A FABIO PARDO, por su apoyo y amistad.

A todos ellos toda mi gratitud, porque este logro no hubiera sido posible sin ellos. Bendiciones

CONTENIDO

pág.

LISTA DE TABLAS

pág.

LISTA DE FIGURAS

pág.

RESUMEN

TÍTULO: PROPUESTA DE UN MODELO DE ENSEÑANZA/APRENDIZAJE DE INGENIERÍA DEL SOFTWARE APOYADO EN TIC[*]

AUTOR: MARTHA SUSANA CONTRERAS ORTIZ[**]

PALABRAS CLAVE: Metodologías de Enseñanza / Aprendizaje, Ingeniería del Software, TIC, Aprendizaje Basado en Proyectos, Requerimientos de Software.

DESCRIPCIÓN:

El proceso de desarrollo de software puede convertirse en una labor altamente compleja, debido a diversos factores como: la cantidad de actores que intervienen y las fallas de comunicación entre ellos, el amplio o poco conocimiento que poseen los desarrolladores y en especial, el nivel de dificultad de la situación problema que se desea resolver. Como consecuencia, el proceso de enseñanza y aprendizaje de la Ingeniería del Software (IS) se ve afectado y requiere la aplicación de estrategias que lo faciliten. El presente documento propone un modelo de enseñanza / aprendizaje de la asignatura de Ingeniería del Software I impartida en la Universidad Industrial de Santander, basándose en una revisión de diversos modelos de enseñanza que se han desarrollado en universidades de prestigio del mundo, enfocado en la aplicación de estrategias educativas fundamentales, apoyándose en el uso de las TIC y partiendo del perfil profesional en el área que se desea alcanzar. Así mismo, se delimita el cuerpo del conocimiento según las guías del SWEBOK y el SEEK, se hace un análisis de diferentes herramientas computacionales de apoyo y se finaliza con el análisis de encuestas realizadas a estudiantes que aplicaron la metodología y a otros que no, para validar su apreciación y satisfacción en las metas cumplidas durante el curso.

[*] Proyecto de grado de Maestría.

[**] Facultad de Ingenierías Físico Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Mcc. Fernando Antonio Rojas Morales.

ABSTRACT

TITLE: PROPOSAL TO A TEACHING/LEARNING MODEL OF SOFTWARE ENGINEERING SUPPORTED IN ICT*

AUTHOR: MARTHA SUSANA CONTRERAS ORTIZ**

KEYWORDS: Teaching/Learning Methodologies, Software Engineering, ICT, Proyect-based Learning, Software Requirements.

DESCRIPTION:

The software development process can become in a highly complex task, due to several factors such as: the number of actors involved in the process and the communications failures between them, the large or little developers' knowledge about the problem domain and specially, the level of difficulty of the problem that they want to solve. As a result, the process of software engineering (SE) teaching and learning is affected and therefore, it requires strategies that do it easier. This research proposes a teaching/learning model of a Software Engineering course I of Universidad Industrial de Santander, based on a review of various SE teaching models that have been developed at important universities around the world, focusing in the implementation of fundamental instructional strategies, relying in the use of ITC and based of the professional profile in the area to be achieved. Likewise, the body of knowdledge is delimited according to the SWEBOK and SEEK guides in order to structuring the curriculum, an analysis of different computational support tools is made and finally, is shown the analysis of a survey of students who applied the methodology and others that not do it, to validate their appreciation and satisfaction about the goals achieved during the course.

* Master's degree project.

** Faculty of Physical – Mechanical Engineering. System Engineering School. Advisor: Mcc. Fernando Antonio Rojas Morales.

INTRODUCCIÓN

El ejercicio de las buenas prácticas en Ingeniería del Software debe llevar a cumplir con su objetivo primordial: desarrollar software de calidad, es decir, software que satisfaga los requerimientos de los usuarios y clientes, en términos de desempeño, seguridad y mantenimiento, dentro del tiempo y con el presupuesto estimado. Esta labor comprende todas las fases del ciclo de vida de un sistema software. No obstante, cabe resaltar que a través de la experiencia, se ha demostrado que errores en las fases tempranas del proceso de desarrollo de software (obtención y análisis de requerimientos) elevan sustancialmente los costos y tiempos de dedicación, razón por la cual debe prestarse especial atención en estos periodos. [Winbladh, 2004], [Mirian-Hosseinabadi, 2010].

El Ingeniero de Software debe estar en la capacidad de usar normas para el control de calidad, conocer metodologías de planificación y gestión de proyectos, usar herramientas de análisis, diseñar e implementar procesos de prueba y verificación de los requerimientos. También debe tener experiencia y conocimiento de métodos y prácticas en administración de equipos de proyectos, poseer habilidades comunicativas, de abstracción, síntesis y ser proactivo en la toma de decisiones [Michael, 2000], [Seffah, 2002]. Por lo tanto, es muy importante que durante su preparación profesional reciba las herramientas y apoyo necesarios para alcanzar estas habilidades y conocimientos, de una forma precisa y contundente, para lo cual el presente estudio centra el interés en:

(1) Los estándares establecidos para el contenido que se debe abarcar en la cátedra de Ingeniería de Software.
(2) El análisis descriptivo de diversas metodologías que se han aplicado a la enseñanza de la IS a nivel de pregrado, que reúnen lo anterior y se enfocan en lograr el aprendizaje significativo, visto como el resultado de apropiar exitosamente el conocimiento. [Mirian-Hosseinabadi, 2010], [Hilburn, 1999]. Se tendrá un acercamiento a las herramientas TIC que se aplican en la

enseñanza de IS, para poder identificarlas y definir cuáles serían de gran apoyo en este campo.

(3) Los parámetros que definen la comprensión y aplicación de procesos de definición de requerimientos de software.

(4) Finalmente y como producto principal, en el planteamiento de un modelo de enseñanza/aprendizaje de la IS, que vincule los aspectos valorados en los ítems anteriores y que brinde un apoyo notorio en el proceso de formación de nuevos ingenieros de software.

1. DESCRIPCIÓN DEL PROYECTO

1.1 PLANTEAMIENTO DEL PROBLEMA DE INVESTIGACIÓN

El proceso de desarrollo de software puede convertirse en una labor altamente compleja, debido a diversos factores, tales como: la cantidad de actores que intervienen, el conocimiento que poseen los desarrolladores, tanto del dominio de aplicación como de la tecnología de solución, y en especial el nivel de dificultad de la situación problema que se desea resolver con el producto de software. Como consecuencia, el proceso de enseñanza y aprendizaje de la Ingeniería del Software **(IS)**, vista la IS como "la aplicación de un enfoque sistemático, disciplinado y cuantificable, al desarrollo, operación y mantenimiento de software"[1] se ve afectado por esta complejidad y requiere la aplicación de estrategias que lo faciliten.

Una alternativa para manejar estos aspectos es analizar el contexto educativo de la asignatura, su contenido temático, herramientas de apoyo existentes y los problemas inherentes de la enseñanza y aprendizaje del área, con el fin de buscar técnicas que maximicen el desarrollo de competencias. En este proceso toma fuerza el uso de TIC[2], ya que por tratarse de un área de conocimiento que se basa en la tecnología, se espera la aplicación de técnicas y dinámicas que hagan uso de instrumentos de TIC.

Son precisamente las TIC las que ofrecen: (1) Acceso a la información (vital en el desarrollo de cualquier proyecto), (2) Proceso de datos, (3) Canales de información inmediata y ubicua (4) La posibilidad de trabajo colaborativo (con entidades o instituciones interdisciplinares alrededor del mundo); las que permiten adquirir, automatizar, almacenar y compartir información y, los resultados y avances de las investigaciones realizadas en este contexto.

---

[1] IEEE Standard Glossary of Software Engineering Terminology. IEEE std 610.12-1990. 1990.
[2] Tecnologías de Información y Comunicación

Teniendo en cuenta lo anterior, la Universidad Industrial de Santander **(UIS)**, y específicamente la Escuela de Ingeniería de Sistemas e Informática, ha visto la necesidad de fortalecer las iniciativas en este campo y brindarle a la comunidad estudiantil y docente mecanismos para apoyar sus procesos de enseñanza y aprendizaje de IS, para promover la formación de profesionales con mejor preparación en este campo (entre más herramientas tenga el mercado, más exigente se vuelve, por tanto, se esperan profesionales más competentes).

Actualmente, la UIS tiene dos cursos de Ingeniería del Software (I y II), impartidos en el cuarto año de carrera; sin embargo, nuestra investigación se centra en el estudio del primero de ellos y en la presentación de un modelo de enseñanza estructurada que brinde mejores oportunidades en el desempeño y habilidades de los estudiantes en el área. Aunque existe una metodología establecida, se desea vincular con mayor énfasis a la tecnología, analizar la viabilidad de trabajo en equipo y manejo de roles, parametrizar la documentación de entrega de proyectos, vincular a los estudiantes en todo el proceso y aplicar una metodología consistente con la necesidad del mundo productivo actual, dentro de un entorno educativo.

1.2 JUSTIFICACIÓN

El objetivo principal de la educación superior es lograr la formación de profesionales competentes; por tanto, en asignaturas con alto nivel de dificultad, como la Ingeniería del Software, es necesaria la aplicación de una metodología que permita un mayor aprovechamiento y ofrezca respaldo al proceso educativo.

La Ingeniería del Software (IS) utiliza métodos de ingeniería, procesos, técnicas y mediciones, que los estudiantes, en su mayoría, no pueden relacionar y poner en práctica [Winbladh, 2004], [Cockburn, 2006]. Por lo anterior, en el presente estudio no solo se busca proporcionar un modelo de apoyo para la enseñanza/aprendizaje de la asignatura, sino que se experimentará su viabilidad y su adaptabilidad en un curso presencial.

Durante el proceso será de vital importancia el uso de las TIC como herramienta de soporte, debido a las ventajas que aportan al proceso, tales como la facilidad y diversidad en el acceso a la información, respaldo para su administración y la posibilidad de trabajo colaborativo. Así mismo, la implementación de estas tecnologías permite diversidad en los métodos de enseñanza/aprendizaje, teniendo en cuenta que cada individuo aprende a un ritmo y motivación diferente.

Precisamente, buscando disminuir la distancia que existe entre lo ofrecido en el proceso educativo de la asignatura de Ingeniería del Software I y lo solicitado por el mercado industrial en el desarrollo de Software, en la presente investigación se propone el desarrollo de un modelo que se acerque a la realidad en este campo, desarrollando habilidades de comunicación y captura de requisitos, toma de decisiones, trabajo en equipos interdisciplinarios, uso de TIC y control de riesgos.

1.3 OBJETIVOS

1.3.1 Objetivo General.

Desarrollar un modelo de enseñanza/aprendizaje basado en TIC y en diferentes propuestas en el área, para la asignatura de Ingeniería del Software I limitado a la etapa de Requerimientos de Software, como un mecanismo educativo de soporte para estudiantes y docentes del programa académico de Ingeniería de Sistemas de la UIS.

1.3.2 Objetivos Específicos

- Realizar un análisis descriptivo del cuerpo de conocimiento de la Ingeniería del Software (SWEBOK y SEEK).
- Analizar enfoques metodológicos de aprendizaje y metodologías existentes para la enseñanza de la Ingeniería del Software.
- Identificar herramientas TIC utilizadas en la enseñanza/aprendizaje de Ingeniería del Software I que ofrezca soporte al análisis de requerimientos.

- Definir técnicas de medición de la comprensión y aplicación de los procesos de definición de requerimientos de software en la asignatura de Ingeniería del Software I del programa de Ingeniería de Sistemas de la UIS.
- Diseñar un modelo de enseñanza/aprendizaje de la asignatura Ingeniería del Software I, con base en los resultados del cumplimiento de los objetivos anteriores.
- Experimentar el modelo en un curso real de la asignatura Ingeniería del Software I de la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander para identificar debilidades y fortalezas de su aplicación.

2 MARCO TEÓRICO

En este capítulo se abordará el marco conceptual correspondiente a la fundamentación pedagógica, base indispensable para contextualizar el presente trabajo de investigación.

2.1 FUNDAMENTACIÓN PEDAGÓGICA

Cuando se habla de una metodología, se hace referencia al estudio analítico del objeto del conocimiento, a las técnicas, tareas o métodos empleados para desarrollar un proceso, ya sea teórico o práctico. Por tanto, las metodologías para la enseñanza y el aprendizaje se fundamentan en las actividades, técnicas o herramientas que permiten acompañar este proceso para llevarlo a cabo satisfactoriamente.

Los objetivos que se buscan al aplicar una metodología en el proceso educativo son:

- Encontrar el objeto de aprendizaje y reconocer las metas que se desean lograr.
- Planificar las actividades que se seguirán para alcanzarlas.
- Identificar en el profesor a un agente facilitador, supervisor y generador de actividades e información.
- Proponer el uso de herramientas de apoyo que enriquecerán el proceso de E/A.
- Evaluar el proceso de aprendizaje paso a paso (antes, durante y después).

2.1.1 La educación basada en competencias. Al término competencia se le puede establecer como un conjunto de conocimientos, capacidades, habilidades y comportamientos que le permiten a una persona realizar de forma exitosa una actividad; se relaciona básicamente al saber hacer en contexto. Esto conlleva a que la funcionalidad del aprendizaje esté íntimamente ligada con el hecho que la enseñanza se contextualice con situaciones cercanas a la vida de los estudiantes, de modo que ellos descubran que lo que están aprendiendo les es útil para comprender mejor el mundo que los rodea.

El Instituto Colombiano para el Fomento de la Educación Superior (ICFES) es el ente encargado en Colombia de enfocar la evaluación por competencias, definiendo el término como "las acciones que el sujeto realiza cuando interactúa significativamente en un contexto determinado", lo cual se resume en "saber hacer en contexto" [Salas, 2005]. A nivel general comprenden la capacidad de **Interpretar** (dar sentido a los problemas que surgen de una situación, es decir, identificar cuál es la situación que se requiere resolver), la capacidad de **Argumentar** (sustentar o explicar una situación que se presenta, reconocer y entender el entorno del problema) y la capacidad de **Proponer** (dar alternativas de solución a situaciones problemáticas, construir modelos).

Dentro las diferentes competencias específicas que puede desarrollar un ser humano, se encuentran las competencias matemáticas y lógicas (que incluyen aspectos tan importantes como la abstracción, análisis, diseño y demás procesos del pensamiento), las comunicativas y lingüísticas (tratamiento de la información y expresividad), sociales (interacción con el entorno), artísticas y culturales (HCI: Human Computer Interaction), motrices (incluyen la habilidad en el uso de instrumentos, herramientas de apoyo en el campo educativo), cognoscitivas (construcción de modelos mentales y de conocimiento), entre otras [Escamilla, 2008]. Ahora bien, dentro del ámbito de interés de la presente investigación, se destaca el desarrollo de aptitudes matemáticas, en el contexto de manejo y

procesamiento lógico, en el ámbito de abstracción, análisis y diseño, en la construcción de modelos (combinando la competencia cognoscitiva) y uso de la ciencia y tecnología. Así mismo, y no menos importante, el desarrollo de competencias comunicativas, por ser un pilar en el engranaje del proceso de desarrollo de software, incluyendo los focos básicos del lenguaje: la oralidad (comunicación verbal), la escritura y lectura.

2.1.1.1 Implicaciones de la formación por competencias. Desde el punto de vista curricular, cuando se piensa en vincular las competencias en el proceso educativo, es necesario construir el plan de estudios basado en procesos y no en contenidos. [Salas, 2005]. De esta forma, en el caso de la Ingeniería del Software, se recomienda un aprendizaje basado en proyectos y casos, donde el estudiante a través del desarrollo de ejercicios prácticos se va apropiando del conocimiento.

Desde el punto de vista didáctico, la educación basada en competencias debe apuntar a una educación centrada en el estudiante y en su proceso de aprendizaje, dejando atrás la metodología transmisionista. Como ejemplo se pueden mencionar las metodología activas tales como el Aprendizaje Basado en Problemas [Salas, 2005].

Desde el punto de vista evaluativo, la formación por competencias se vuelve una labor compleja, pues sugiere un cambio de una evaluación por logros a una evaluación por procesos. [Salas, 2005]. En este caso, para dar un concepto del aprendizaje del estudiante, no bastaría con una respuesta acertada, sino con el seguimiento al proceso seguido para obtener dicha respuesta.

2.1.2 Teorías del aprendizaje. Las teorías del aprendizaje han surgido por la necesidad de describir y comprender los procesos mediante los seres humanos y los animales aprenden, con el fin de generar estrategias para fortalecer la adquisición de destrezas y habilidades y en especial, la apropiación del conocimiento. Diversos psicólogos y pedagogos han investigado en el tema. A continuación se describen las teorías más representativas e importantes [Huber, 2008], [Universitat Jaume, 2003]:

- **Conductismo**: analiza los cambios en la conducta de un individuo, es decir, la repetición automática de patrones de conducta ante diferentes situaciones. El objetivo de estudio debe ser cuantificable. El estudiante se aprecia como un reactivo al ambiente y sus condiciones, es decir, no se visualiza como un agente activo en el descubrimiento del conocimiento. En el caso de estudio de la presente investigación no es de mucha utilidad, ya que como se mencionó anteriormente, se busca que el aprendizaje se base en el estudiante más que en aspectos externos.
- **Cognoscitivismo**: analiza los procesos que dieron origen a los cambios de conducta, de modo que se identifiquen los indicadores de cambio para entender qué sucede en la mente del que aprende. Define el aprendizaje como un proceso donde el individuo identifica la información que recibe, la reorganiza y la asimila, es decir, basa la comprensión de las cosas en la percepción de objetos y las relaciones entre ellos. En este modelo el estudiante ya asume un papel más activo en su proceso de aprendizaje, sin embargo, el ambiente sigue siendo un elemento principal por lo cual no es relevante para este estudio.
- **Constructivismo**: sugiere que el estudiante construya su conocimiento y modelos mentales a partir de sus experiencias. El aprendizaje es más abierto y es más difícil medir los resultados. Este enfoque vino a reformar la psicología del aprendizaje durante los años de 1970 y 1990. Su propulsor en las teorías de orientación cognitiva o psicológica, Jean Peaget, indagaba la relación entre

el individuo y su entorno, llegando a la idea del uso de procesos de aprendizaje basados en experiencias para que el individuo se adapte al entorno. Von Glasersfeld, otro importante investigador del tema, indica el primer principio del constructivismo: “No se percibe conocimiento pasivamente sea por los sentidos sea por la comunicación, pero el sujeto en proceso de conocimiento lo construye activamente”. Este modelo de aprendizaje si se ubica dentro de lo deseable en esta investigación: lograr en el estudiante una participación activa, donde no solo se adquiera el conocimiento, sino donde él descubra cómo y de dónde lo obtuvo y para qué le será útil. Esto se obtiene solo si el “vive” su proceso y es consciente del mismo ya que lo construye, es decir, se implementa un aprendizaje activo.

- **Construccionismo:** afirma que el aprendizaje es superior si los estudiantes se comprometen en la construcción de un producto significativo, involucrando el desarrollo de elementos en el mundo externo y conocimiento al interior de su pensamiento. Así mismo, este modelo será una valiosa herramienta para preparar nuestra investigación, ya que combina el constructivismo, con la relación que existe con el mundo exterior, evitando que el conocimiento sea subjetivo a solo la experiencia del estudiante y entendiendo que todos hacemos parte de un sistema externo que debe involucrarse en el proceso.

2.1.3 Estrategias de enseñanza y aprendizaje. Debido a las múltiples investigaciones en el campo educativo, han surgido una serie de estrategias, que combinadas, han dado excelentes resultados en términos de adquisición y aprehensión del conocimiento. A continuación se describen las principales estrategias de enseñanza y aprendizaje que operan hoy en día:

- **Aprendizaje basado en problemas o casos:** Surge en Canadá, concentrando la atención del aprendizaje en objetivos determinados por la realidad o por situaciones muy realistas. Se desarrolla siguiendo estos pasos [Huber, 2008]]:
  a) Presentación del problema a los estudiantes.

b) Análisis del problema.
c) Generación de hipótesis que puedan aclarar el problema.
d) Identificar faltas de conocimiento: reconocer qué no conocen aún sobre la situación.
e) Decidir metas de aprendizaje.
f) Aprendizaje individual (a través de autoestudio usando libros, TIC, etc).
g) Intercambio de resultados y formulación de conclusiones.

Se observa que el objetivo de este enfoque no es resolver el problema, sino descubrir cómo identificar y conocer acerca del problema [Rodríguez, 2012]. Se le presenta al estudiante una situación problemática con ciertas características que deberá analizar, con el fin de darle solución. El propósito es que el estudiante desarrolle destrezas en el análisis generalizado de problemas para deducir la solución sistemáticamente y la toma de decisiones.

En la presente investigación se puede usar en los talleres o actividades que se propongan, pues es una herramienta favorable en el sentido de análisis que imprime en los estudiantes, pero no será el eje central del modelo. Cabe resaltar que unos de los factores que resaltan esta estrategia de aprendizaje es el efecto que causa en los estudiantes al permitirles descubrir, con una buena orientación, sus faltas de conocimiento, es decir, sus falencias en la resolución del problema. Además, con el uso de esta estrategia, se da un enfoque por competencias al aprendizaje.

- **Aprendizaje cooperativo y proceso de enseñanza:** Permite la interrelación del estudiante con sus compañeros y tutores. Contribuye al avance en los procesos cognitivos, motivacionales y afectivo-relacionales. Ejemplos donde se aplica esta técnica: consultas, resolución de ejercicios, taller de ejercicios.

- **Aprendizaje individual:** A través de consultas, tareas individuales, resúmenes, análisis y resolución de problemas, entre otros, el estudiante desarrolla la habilidad para crear su conocimiento. Cada uno debe dar cuenta de su labor y aprendizaje, aunque en ocasiones se trabaje de forma grupal.

- **Aprendizaje interactivo:** Mediante la comunicación con otras personas, a través de exposiciones, entrevistas, debates, seminarios, etc., el estudiante comparte sus experiencias y confronta el conocimiento.

- **Aprendizaje significativo:** Es el resultado de aplicar todo lo anterior, siempre y cuando, se establezca una relación sustantiva entre la información que está adquiriendo (nueva), con el conocimiento previo que posee.

- **Aprendizaje basado en proyectos:** Al igual que el aprendizaje basado en problemas o casos, presupone la solución de una situación o caso problemático a partir de un análisis sistemático de las características; además, incluye la ejecución de tareas, para lo cual los estudiantes requieren instrucciones claras y precisas antes de iniciar el proceso. Los estudiantes trabajan autónomamente y en pequeños grupos [Rodríguez, 2012]. El objetivo es “aprender haciendo”. Sus primeras propuestas surgen entre 1900 y 1933. En esta época, en Estados Unidos y Alemania, la mayoría de las instituciones educativas eran privadas, pero utilizaban este método en la educación profesional. Ya hacia los años sesenta, este método se aplica en todos los ámbitos educativos.

  Este método unifica el aprendizaje teórico y práctico, con elementos de la vida cotidiana, tal como se ve a continuación [Galeana, 2007]:

  a) El aprendizaje se basa en un interés auténtico y/o en una iniciativa.

b) Los estudiantes intercambian información sobre sus intereses y perspectivas del campo de estudio.
c) Desarrollan su propio ámbito de actividad.
d) Cuando suspenden sus actividades, reflexionan sobre las mismas.
e) El proyecto culmina cuando se ha cumplido con la tarea.

Por la experiencia del director del proyecto y analizando otras investigaciones en el área, esta estrategia de aprendizaje es la elegida en la metodología a desarrollar, por ser una guía y ejemplificar muy de cerca el proceso de desarrollo de software. Además en esta estrategia de aprendizaje se mezclan factores que promueven las demás, ya que busca el aprendizaje significativo (permitiendo que los estudiantes escojan un proyecto de su interés, con el cual se encuentren motivados y reconozcan la importancia o relevancia que tendrá en su formación profesional), requiere trabajo cooperativo, interactivo e individual, y dentro del proyecto se incursiona en la resolución de pequeñas situaciones problemas, cuando el docente plantea una serie de patrones que los estudiantes deben analizar, dar seguimiento y resolver.

La utilización de esta estrategia de aprendizaje conlleva a la idea del trabajo en equipos, para acercarse más a los escenarios reales, con personal de diferente perfil. Sin embargo, en el ámbito educativo, se presume que los estudiantes de un mismo curso se encuentran en un nivel de conocimiento similar. Por tanto, lo que se aprovecha en este ejercicio son las habilidades que cada uno posee, para la formación de los grupos.

3 CUERPO DEL CONOCIMIENTO DE LA INGENIERÍA DEL SOFTWARE (SWEBOK Y SEEK)

El término de Ingeniería del Software empezó a usarse en los años 60's, a raíz de la llamada "crisis del software" (1965 – 1970), donde los proyectos de desarrollo de software sufrían de retrasos considerables en su planeación y ejecución por lo cual existía poca productividad, el mantenimiento se convertía en una labor muy costosa e impredecible y la calidad de los productos era poco fiable. En la búsqueda de soluciones para esta situación, se establecieron técnicas y procedimientos que formalizaron el proceso de desarrollo de software, brindándole a la fase de diseño la importancia que se había omitido anteriormente, a lo cual se le denominó "Ingeniería del Software". Como parte de las tareas para lograr que fuera reconocida como disciplina y profesión, la Sociedad de Computadores IEEE[3] diseñó una guía que recopila el cuerpo del conocimiento de esta área, llamada SWEBOK. Más adelante, y a partir de este documento, el IEEE en asociación con la ACM (Association for Computing Machinery) desarrollaron otra guía que provee una orientación a las instituciones educativas y a las agencias de acreditación sobre lo que debería constituir la educación de un ingeniero del software en pregrado; a este otro documento se le llamó SEEK. En la presente investigación se analizarán ambos documentos por integrar el cuerpo del conocimiento del área en general y por estar respaldados por entidades de prestigio en el campo de la informática a nivel mundial.

3.1 SWEBOK

En el presente año 2014 acaba de publicarse la tercera versión de esta guía; la primera versión fue publicada en 1996, seguida por la segunda versión de 2004. El objetivo primordial de esta actualización (SWEBOK Guide v3) es mejorar el valor, legibilidad, consistencia y usabilidad de la Guía. Además, para reflejar los cambios que ha experimentado la Ingeniería del Software desde la anterior publicación.

[3] Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos)

La Guía del Cuerpo del Conocimiento de Ingeniería del Software (SWEBOK) se estableció con los siguientes objetivos [IEEE, 2014]:

1. Promover una visión consistente de la Ingeniería del Software a nivel mundial.
2. Especificar el alcance de, y aclarar el lugar que ocupa la Ingeniería del Software con respecto a otras disciplinas como las ciencias de la computación, la administración de proyectos, la ingeniería informática y las matemáticas.
3. Caracterizar los contenidos de la disciplina Ingeniería del Software.
4. Proporcionar un acceso actual al cuerpo del conocimiento de la Ingeniería del Software.
5. Proveer un fundamento para el desarrollo del plan de estudios y para certificación y licenciamiento de material.

Estos 5 objetivos mencionados anteriormente le proporcionan a esta investigación los pilares para fundamentar y justificar su estudio como fuente para establecer los contenidos que son esenciales abarcar en el área. El SWEBOK es un proyecto investigativo de gran envergadura de la IEEE Computer Society que contó con la participación de aproximadamente 150 críticos de 33 países, lo cual motiva y soporta su aplicación.

Antes de continuar, se va a formalizar la definición de Ingeniería del Software, según lo planteado por la Sociedad de Computadores IEEE, que hace referencia a ella como: (1). La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento de software; esto es, la aplicación de ingeniería al software. [IEEE, 2014].

3.1.1 Áreas de conocimiento del SWEBOK. A continuación se presentan las áreas del conocimiento del SWEBOK [IEEE, 2014], sin embargo cabe resaltar que el presente estudio profundiza en la primera de ellas y se complementa con la número 11:

1. Requerimientos de Software.
2. Diseño de Software.
3. Construcción de Software.
4. Pruebas de Software.
5. Mantenimiento de Software.
6. Gestión de configuración de Software.
7. Gestión de la Ingeniería del Software.
8. Procesos de la Ingeniería del Software.
9. Modelos y Métodos de la Ingeniería del Software.
10. Calidad del Software.
11. Práctica profesional de la Ingeniería del Software
12. Economía de la Ingeniería del Software
13. Fundamentos de computación
14. Fundamentos de Matemáticas
15. Fundamentos de ingeniería.

El SWEBOK reconoce que existen otras disciplinas que limitan y tienen aspectos en común con la Ingeniería del Software, por lo cual ha definido también otras 7 disciplinas relacionadas; sin embargo, la guía no tiene por objeto caracterizarlas. Estas disciplinas se listan a continuación:

1. Ingeniería de la Computación.
2. Ciencias de la Computación.
3. Administración general.
4. Matemáticas.

5. Administración de proyectos.
6. Administración de calidad.
7. Ingeniería de Sistemas

3.1.1.1 Requerimientos de software. El IEEE (IEEE83) define Requerimientos como [IEEE, 1990, p. 62]:

(1) Una condición o capacidad necesaria por un usuario para resolver un problema o alcanzar un objetivo.
(2) Una condición o capacidad que debe ser encontrada o poseída por un sistema o componentes del sistema para satisfacer un convenio, estándar, especificación u otro documento formalmente impuesto.

El conjunto de todos los requerimientos establecen las bases para desarrollos subsecuentes del sistema o componentes del sistema.

Los actores o personas que intervienen en el proceso de requerimientos se denominan **agentes de proceso** [IEEE, 2014]:

- Usuarios: quienes operarán el software. Generalmente involucra personas de grupos heterogéneos con diferentes roles y requerimientos.
- Clientes: aquellos que hicieron la solicitud del software o quienes representan la compañía que hizo la solicitud.
- Analistas de mercado: son quienes descubren la necesidad del software. En ocasiones son los mismos clientes.
- Reguladores: quienes supervisan el buen funcionamiento de las aplicaciones que requieren ser reguladas, tales como los sistemas bancarios y los sistemas del transporte público.
- Ingenieros de software: dirigen el desarrollo del software.

El proceso de definición de requerimientos comprende los siguientes pasos, aunque no son necesariamente una secuencia estricta [Brackett, 1990]:

- Identificación de requerimientos: los requerimientos de software son obtenidos desde personas o derivados desde requerimientos del sistema.
- Identificación de las limitaciones de desarrollo de software.
- Análisis de requerimientos.
- Representación de requerimientos.
- Comunicación de requerimientos.
- Preparación para la validación de requerimientos de software.

Sin embargo, según el SWEBOK es posible resumirlos y definirlos concretamente como sigue: captura, análisis, especificación y validación.

- **Captura de requerimientos:** Permite comprender el problema que el software pretende solucionar. Para que esta fase sea exitosa es fundamental que exista buena comunicación entre los usuarios del software y los ingenieros, para lo cual se debe mediar entre el dominio de los usuarios y el lenguaje técnico del ingeniero de software. [IEEE, 2014]

Dentro de las fuentes de donde surgen los requerimientos, encontramos [IEEE, 2014]:

- Metas: son los objetivos que quiere alcanzar el software. Los ingenieros de software deben establecer la prioridad y el costo de las metas, que puede realizarse a través de un estudio de viabilidad.
- Conocimiento del dominio: le permite al ingeniero del software deducir o interpretar más eficazmente lo que el usuario quiere expresar.

- Participantes: los ingenieros de software deben estar atentos y analizar los puntos de vista de los participantes del proyecto para evitar polémicas y tardanzas en las definiciones de los requerimientos.
- Reglas de negocio: son estatutos que definen o limitan algunos aspectos de la estructura o comportamiento del negocio por sí mismo.
- Entorno operacional: los requerimientos dependen del contexto en el cual se desarrollará el software.
- Entorno de la organización: hace referencia a la estructura, cultura y política de la empresa. Es importante que los ingenieros de software identifiquen este entorno para que no generen cambios imprevistos en el proceso de negocio.

Para la captura de requerimientos, existen diversas técnicas, tales como [IEEE, 2014]:

- Entrevistas: es el proceso más tradicional. Permite tener una visión inicial del contexto y de las solicitudes primarias del consumidor. Es importante entender las ventajas y limitaciones de las entrevistas y cómo ellas deberían ser conducidas.
- Escenarios: permiten dar significado a las solicitudes del usuario, por ejemplo a través de casos de uso.
- Prototipos: facilitan la comprensión de requerimientos imprecisos.
- Reuniones: permite la interacción de varios agentes, descubrir conflictos y confusiones en la definición de los requerimientos. Es fundamental que sean dirigidas para que no se generen discusiones interminables o sin sentido.
- Observación: los ingenieros de software adquieren especial habilidad para identificar características o procesos mediante la observación detallada a los usuarios.

- **Análisis de requerimientos:** En esta etapa se busca explorar y entender los requerimientos, para lo cual es importante conocer su naturaleza y diferenciarlos entre sí. Una forma para entender más fácilmente el problema a solucionar es el desarrollo de modelos, los cuales deben ser una representación de la situación a resolver. A través de este modelado se definen instancias y relaciones.

El SWEBOK sugiere la siguiente clasificación de requerimientos de software [IEEE, 2014]

- Requerimientos funcionales y no funcionales: los requerimientos funcionales hacen referencia a las funciones que el software va a desarrollar; mientras que los no funcionales son aquellos que se ejecutan para forzar la solución. También son conocidos como requerimientos de calidad. Se clasifican en requerimientos de funcionamiento, de capacidad de mantenimiento, de seguridad, de confiabilidad, entre otros.
- Requerimientos derivados o directos (si se deducen de otros requerimientos o se imponen de forma directa).
- Requerimientos de producto o de proceso.
- Según prioridades de los requerimientos: cuando un requerimiento tiene alta prioridad tiende a ser más importante para alcanzar las metas finales del software. (obligatorio, altamente deseable, deseable u opcional).
- Según alcances del requerimiento: grado con el cual el requerimiento afecta al software y sus componentes.
- Según la volatilidad o estabilidad de los requerimientos: hace referencia a los cambios o tendencia al cambio que pueden tener los requerimientos durante el ciclo de desarrollo del software.

El desarrollo de modelos de un problema es fundamental para el análisis de requerimientos de software, ya que permiten entender el problema antes de iniciar el diseño de la solución. [IEEE, 2014]

Existen infinidad de modelos que pueden ser desarrollados, tales como diagramas de casos de uso, modelos de flujo de datos, modelos de estado, modelos basados en metas, interacciones de los usuarios, modelos de objetos, modelos de datos, entre otros. Los factores que determinan la escogencia de un modelo son:

- La naturaleza del problema: para algunas situaciones son más factibles o recomendables ciertos modelos que otros.
- La experiencia del ingeniero de software: es más práctico trabajar con modelos que sean conocidos por el ingeniero de software.
- Los requerimientos de proceso del cliente: los clientes pueden imponer su notación o método favoritos, o prohibir cualquiera con el cual no se sientan familiarizados. Este factor obstaculiza el factor anterior.

- **Especificación de requerimientos:** Hace referencia al establecimiento de los alcances del producto, es decir sus límites, mediante la elaboración de un documento, que cuando se habla de sistemas complejos incluye la definición de sistema, requerimientos de sistema y los requerimientos de software; mientras que cuando se tiene un sistema simple se requiere solo el último de éstos [IEEE, 2014].

  - **Documento de la definición de sistema:** describe el sistema de requerimientos (objetivos del sistema, visión, requerimientos no funcionales) y va enfocado a los usuarios del sistema y/o clientes. Es posible que contenga modelos conceptuales para documentar el entorno del sistema, su uso, dominio y flujos de trabajo.

- **Documento de requerimientos de sistema:** describe la visión y el contexto del sistema en general.
- **Documento de requerimientos de software:** especifica las funciones, características y solicitudes que hicieron los clientes para dar una solución sistematizada a una situación dada. Además indica los alcances del producto, es decir lo que podrá (alcance) y no podrá (límites) hacer el software. Debe incluir un estudio de costos, riesgos y un cronograma de trabajo.

- **Validación de requerimientos:** Es importante que los requerimientos se puedan validar para asegurar su comprensión, manejo de estándares y exactitud en su interpretación y definición por parte del ingeniero de software [IEEE, 2014]. Este proceso se hace mediante el análisis o inspección del documento de requerimientos. También es importante realizar pruebas de aceptación de los requerimientos para verificar que el producto final lo satisface.

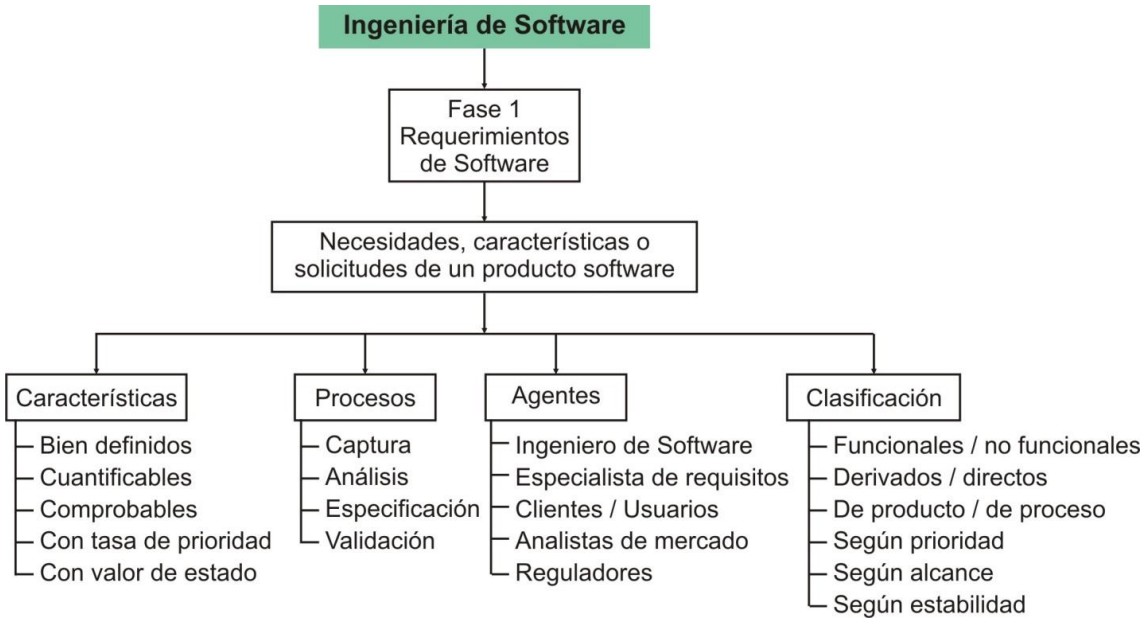
  Dentro de las actividades que incluye el proceso de validación de requerimientos encontramos:

  - **Revisión de requerimientos**: es quizás el mecanismo más común de validación de requerimientos. Se asigna un grupo de críticos que buscan errores, supuestos equivocados, falta de claridad y desviación de los estándares.
  - **Creación de prototipos:** facilita la interpretación de las presunciones del ingeniero de software, pero desvía la atención por asuntos de estética de la interfaz o problemas de calidad con el prototipo. Por tal motivo, se recomienda evitar prototipos de software en esta etapa.
  - **Validación de modelos:** se valida la calidad de los modelos desarrollados durante el análisis.

- **Pruebas de aceptación:** una labor importante es planear cómo verificar cada requerimiento, generalmente mediante pruebas de aceptación.

En la Figura 1 se puede apreciar un cuadro resumen que indica los procesos, agentes que intervienen y clasificación de los requisitos, como parte de la primera fase de desarrollo de software, propuestos por el SWEBOK.

Figura 1. Cuadro resumen de Requerimientos de Software, según el SWEBOK.



Fuente: Autor del proyecto, a partir de IEEE (2014, p. 32 – 45)

3.1.1.2 Diseño de software. Esta fase del desarrollo de software es de vital importancia, porque permite el engranaje en todo el proceso de construcción y orienta las actividades subsiguientes tanto de desarrollo como de pruebas, y de su buen manejo depende en gran medida la calidad y éxito del proyecto en curso.

El diseño es definido por el IEEE como: “el proceso de definir la arquitectura, los componentes, las interfaces y otras características de un sistema o un componente y el resultado de este proceso”. En general, se puede determinar que el diseño de software como proceso es el análisis de los requerimientos para

definir la estructura en la cual se basará su construcción; mientras que como resultado del proceso, describe en detalle la organización de los componentes (diseño arquitectónico) y la relación entre ellos y su comportamiento (diseño detallado). [IEEE, 2014]

El diseño de software permite la producción de modelos, los cuales guiarán cada etapa siguiente de la construcción del software.

Para garantizar que el diseño de software esté bien ejecutado, es necesario usar principios o normas que guíen el proceso; dentro de los más destacados se encuentran:

- **Abstracción**: el IEEE la define como (1) una vista de un objeto que se enfoca en la información relevante para un propósito particular e ignora la información restante. (2) El proceso de formular una vista como en (1). [IEEE, 1990]
- **Acoplamiento y cohesión**: el acoplamiento se define como el grado de interdependencia entre módulos de software, mientras que la cohesión hace referencia a la forma en la cual las tareas desarrolladas por un módulo particular de software se relacionan con las demás. [IEEE, 1990]
- **Descomposición y modularidad**: fraccionamiento del software en partes independientes más pequeñas para distribuir las funcionalidades. [IEEE, 2014].
- **Encapsulación/ocultar información**: se agrupan y empaquetan los elementos y detalles internos de una abstracción para hacerlos inasequibles a entidades externas. [IEEE, 2014].
- **Separación de la interface e implementación**: define un componente especificando una interface pública, pero separando los detalles de cómo se desarrollaron los componentes. [IEEE, 2014].
- **Suficiencia, completitud y primitivismo:** las dos primeras aseguran que un componente software capture todas las características importantes de una

abstracción y nada más. El primitivismo significa que el diseño debería estar basado en patrones que sean fáciles de implementar.

Los principales conceptos referentes al diseño de software son [IEEE, 2014]

- **Concurrencia**: indica cómo se descompone el software dentro de procesos, tareas e hilos y se ocupa de temas relacionados a la eficiencia, atomicidad, sincronización y cronograma.
- **Control y manipulación de eventos**: hace referencia a la organización de datos y flujos de control, a cómo manipular eventos.
- **Persistencia de datos**: cómo manipular los datos.
- **Distribución de componentes**: cómo distribuir el software a través de hardware y cómo comunicar los componentes.
- **Manipulación de errores y excepciones y tolerancia al error**: cómo prevenir y tolerar las fallas.
- **Interacción y presentación**: cómo estructurar y organizar las interacciones con usuarios y la presentación de información.
- **Seguridad**: cómo prevenir ataques de seguridad, revelaciones no autorizadas, creación, cambio, supresión o negación de acceso a la información y a otros recursos.

La arquitectura software se define como un conjunto de estructuras necesarias para razonar sobre el sistema, el cual abarca los elementos de software, las relaciones entre ellos y sus propiedades. [IEEE, 2014]. Dentro de ella, deberían ser descritas y documentadas las comúnmente denominadas vistas de software (facetas de alto nivel del diseño de software que especifica aspectos parciales).

Los principales conceptos sobre calidad de diseño de software son:

- **Atributos de calidad:** sostenibilidad, portabilidad, trazabilidad, que se pueda probar, robustez, seguridad, funcionalidad, usabilidad, que sea modificable, integridad, entre otros.
- **Calidad de análisis y técnicas de evaluación:** revisiones del diseño de software que permitan verificar y asegurar la calidad de los artefactos de diseño. Análisis estático y simulación y prototipado.
- **Medidas:** permiten determinar cuantitativamente aspectos del tamaño, de la estructura o de la calidad de un diseño de software.

Las principales notaciones de diseño de software son:

- **Descripciones estructurales (vista estática):** descripción de los componentes principales del software y cómo se interconectan entre sí y con el sistema. (Lenguajes descriptivos de la arquitectura, diagramas de clases y objetos, diagramas de componentes, diagramas de despliegue, diagramas de Entidad/Relación, lenguajes de descripción de interfaz, entre otras)
- **Descripciones del comportamiento (visión dinámica):** se incluyen diagramas de actividad, de colaboración, diagramas de flujo de datos, tablas y diagramas de decisión, organigramas estructurados, diagramas de secuencia, diagramas de estado y transición, lenguajes formales, lenguajes de diseño de programas y pseudocódigo.

Las estrategias y métodos de diseño de software más destacados son:

- **Estrategias generales:** algunas estrategias útiles en el proceso de diseño son las denominadas divide y vencerás, refinamiento, estrategias de barrido de arriba hacia abajo vs. de abajo hacia arriba, abstracción de datos y ocultamiento de información, uso de heurística, uso de patrones.

- **Diseño (estructurado) orientado a función:** se usa generalmente después del análisis estructurado, dando como resultado diagramas de flujo de datos y descripciones del proceso asociado.
- **Diseño orientado a objetos**
- **Diseño centrado en estructuras de datos:** parte de las estructuras que un programa manipula, más que de las funciones que desarrolla.
- **Diseño basado en componentes:** se centra en el desarrollo e integración de los componentes del software.

3.1.1.3 Construcción de software. La construcción de software hace referencia a la creación detallada de trabajo, a través de una combinación de código, verificación, pruebas, integración de pruebas y depuración. Esta etapa está profundamente ligada al diseño y pruebas de software, debido a que parte de los productos del diseño y provee las entradas de las pruebas.

Los fundamentos de la construcción de software son:

- **Minimizar la complejidad:** es una de las principales razones de ser del software y se logra mediante la aplicación de estándares y el uso de técnicas de construcción enfocadas en la calidad.
- **Anticiparse el cambio:** el software está en constante cambio y afecta y se ve afectado por el entorno de diversas formas, por lo tanto es importante anticiparse a dicho cambio para lograr los resultados esperados.
- **Construir para la verificación:** consiste en la elaboración de mecanismos para detectar fallas fácilmente durante la etapa de construcción del software.
- **Reutilización:** usar bienes existentes para solucionar diferentes problemas, tales como librerías, módulos, componentes y código fuente.
- **Estándares de construcción:** tales como lenguajes de programación, métodos de comunicación, plataformas y herramientas. Estos estándares pueden provenir de diversas fuentes externas (de especificación de interfaces

de software y hardware como el OMG y de organizaciones internacionales como el IEEE o ISO) y de fuentes internas (creados por una organización para proyectos específicos).

Dentro de las consideraciones prácticas de la construcción de software están:

- **Diseño de la construcción:** aunque en menor medida que la etapa anterior, en esta fase del desarrollo también se trabaja en diseño, donde los ingenieros de software realizan pequeñas modificaciones para ajustarse a los problemas que están siendo afrontados por el proyecto.
- **Lenguajes de construcción:** consisten en mecanismos de comunicación para la solución ejecutable de un problema a través de la tecnología. Existen diversos tipos de lenguajes, dentro de los que se encuentran los lenguajes de configuración (para crear nuevas o típicas instalaciones del software), lenguajes de herramientas (lenguajes de programación de aplicaciones) y lenguajes de programación.
- **Codificación:** dentro de la construcción de software, es importante tener en cuenta las siguientes consideraciones:
  - Usar técnicas para la construcción de código fuente comprensible.
  - Utilizar clases, variables, constantes predefinidas, estructuras de control y otras entidades.
  - Tratar las condiciones de error.
  - Prevenir brechas en la seguridad.
  - Organizar y documentar el código fuente.
- **Pruebas de construcción:** existen dos tipos de pruebas: las unitarias y las de integración, cuya principal meta es la detección temprana de fallas. Es posible planear las pruebas incluso antes de la elaboración del código.
- **Reutilización:** consiste en la detección de unidades, bases de datos, procedimientos o datos que pueden ser usados varias veces en el código, con el fin de minimizar complejidad y tiempo de desarrollo.

- **Calidad de construcción:** dentro de las diversas técnicas existentes para el propósito de construir software de calidad, podemos destacar: las pruebas unitarias y de integración, el código paso a paso, depuración, revisiones técnicas, auditorías, entre otras.
- **Integración:** es una actividad crítica que consiste en la unificación o integración de componentes, rutinas, procesos o incluso subsistemas o sistemas que han sido construidos por separado. En esta fase es de vital importancia la planeación de la secuencia de integración y determinar las pruebas que se realizarán para determinar su integridad y evitar fallos.

3.1.1.4 Pruebas de software. El objetivo de las pruebas de software es evaluar, identificar, prevenir defectos y problemas y mejorar la calidad del producto. Para esto se desarrollan una serie de casos de prueba seleccionados dentro de la amplia gama de posibilidades de ejecución, analizando el comportamiento esperado del software ante las entradas de datos dada [IEEE, 2014].

Antiguamente se iniciaba esta etapa al finalizar la construcción, sin embargo, hoy en día se ha logrado que esté vigente durante todo el proceso de desarrollo y mantenimiento, incluso su planeación debería empezar en las primeras fases de los requerimientos.

Dentro de los fundamentos de las pruebas de software tenemos:

- **Criterios de selección de pruebas:** consiste en decidir cuáles casos de prueba serán utilizados dentro del proceso y en comprobar si los elegidos son apropiados.
- **Efectividad del grupo de pruebas:** se puede determinar en función del objetivo que las originó.
- **Identificación de defectos:** cuando se busca encontrar algún defecto, la prueba resulta satisfactoria si se produce un error en el sistema.

- **Limitaciones de las pruebas:** no es posible realizar un grupo completo de pruebas en un software real, debido a los innumerables casos que pueden existir; por tanto, las pruebas pueden comprobar la presencia de errores, pero no puede asegurar que no habrán más después. Es así, como las pruebas se deben dirigir en función de los riesgos.

Debido a que las pruebas se ejecutan durante diferentes procesos de desarrollo y mantenimiento del software, el objeto de las pruebas también es diferente (un módulo, un grupo de módulos o un sistema completo).

- **Pruebas de unidad:** su objetivo es verificar el correcto funcionamiento de partes aisladas del software que pueden probarse independientemente. Estas partes pueden ser componentes de software con unidades relacionadas fuertemente o subprogramas individuales. Normalmente los hacen los mismos que escribieron el código.
- **Pruebas de integración:** se encarga de verificar la correcta integración e interacción entre componentes de software. Se ejecutan en forma jerárquica y paulatina a medida que va avanzando el proceso de desarrollo de software.
- **Pruebas del sistema:** abarcan la verificación de un sistema completo. Se espera que la mayoría de los fallos hayan sido detectados y corregidos durante las pruebas anteriores, por tanto este tipo de pruebas se enfoca a verificar los requisitos no funcionales (seguridad, velocidad, exactitud y confiabilidad), así como las interconexiones con otras aplicaciones.

Es muy importante definir claramente el objetivo de las pruebas, con el fin de controlar efectivamente el proceso. [IEEE, 2014]

- **Pruebas de aceptación-calificación:** validan si el sistema se comporta conforme a los requerimientos establecidos por el cliente.

- **Pruebas de instalación:** se realizan una vez han finalizado las pruebas de aceptación; su objetivo es verificar el sistema en el entorno final.
- **Pruebas alfa y beta:** en ocasiones se distribuye el software a un grupo representativo de usuarios potenciales para que prueben el software y validen su comportamiento en las instalaciones del desarrollador (pruebas alfa) o externamente (pruebas beta).
- **Pruebas de conformidad-funcionales-de corrección:** verifican si el software se comporta según las especificaciones iniciales.
- **Materialización de la confiabilidad y evaluación:** la confiabilidad se puede lograr en la medida en que las pruebas identifiquen los errores.
- **Pruebas de regresión:** son pruebas selectivas que se repiten en un componente para verificar que los cambios no han producido efectos indeseados [IEEE, 1990]. Estas pruebas se realizan para demostrar que el software que ya fue probado y aprobado, pasa aún dichas pruebas. Se pueden desarrollar en cada uno de los niveles descritos en el punto 2.1.4.2.
- **Pruebas de rendimiento:** permiten verificar que el software alcanza el rendimiento especificado, específicamente el de capacidad y tiempo de respuesta.
- **Pruebas de desgaste:** hacen trabajar al sistema en su máxima capacidad y más, con el fin de determinar su límite.
- **Pruebas de continuidad:** se realizan pruebas en dos versiones diferentes del software para comparar sus resultados.
- **Pruebas de recuperación:** permite verificar si el software se puede reiniciar después de alguna dificultad mayúscula.
- **Pruebas de configuración:** son aquellas que verifican el desempeño del software para diferentes tipos de usuario.
- **Pruebas de usabilidad:** evalúa qué tan fácil resulta de usar el software para el usuario, incluyendo la documentación, la efectividad de las funciones y la habilidad para recuperarse después de errores provocados por el usuario.

3.2 SEEK (SOFTWARE ENGINEERING EDUCATION KNOWLEDGE)

Como parte de otro esfuerzo por establecer y documentar las recomendaciones para el currículo de la Ingeniería del Software para estudiantes de pregrado, el IEEE y la ACM (Association for Computing Machinery) crearon en conjunto el SEEK (Software Engineering Education Knowledge). El punto de partida para determinar las áreas de conocimiento del SEEK fue provisto por el SWEBOK.

3.2.1 Áreas de conocimiento, unidades y tópicos del SEEK. El conocimiento es un término usado para describir todo el espectro del contenido para una disciplina: información, terminología, artefactos, datos, roles, métodos, modelos, procedimientos, técnicas, prácticas, procesos y literatura. El SEEK está organizado jerárquicamente dentro de tres niveles: [IEEE, 2004]

- El más alto nivel de jerarquía es la educación: **área de conocimiento**, representando una particular subdisciplina de ingeniería del software que un profesional debería saber. Las áreas de conocimiento son elementos de alto nivel estructural usados para organizar, clasificar y describir el conocimiento de la ingeniería del software. Cada área es identificada por una abreviatura, tal como PRF para Prácticas profesionales.
- Cada área está dividida dentro de pequeñas fracciones llamadas **unidades**, las cuales representan módulos temáticos individuales dentro de un área. Adicionando dos o tres letras de sufijo para el área se identifica cada unidad; por ejemplo, PRF.com es una unidad de habilidades de comunicación.
- Cada unidad es a su vez subdividida dentro de un conjunto de **tópicos**, los cuales son el más bajo nivel de la jerarquía.

3.2.2 Áreas de conocimiento en educación de la ingeniería del software según el SEEK. Después de varias actualizaciones, las áreas de conocimiento definidas por el SEEK se unificaron de la siguiente forma:

- **Esencia de Computación (CMP)**: incluye la fundamentación en ciencia de la computación que soporta el diseño y la construcción de productos de software. Esta área incluye el conocimiento sobre la transformación de un diseño en implementación, las herramientas usadas durante este proceso y los métodos de construcción del software formal.
- **Fundamentos en Matemáticas e Ingeniería (FND):** proporcionan el soporte teórico y científico para el modelado y la construcción de productos de software con los atributos deseados.
- **Práctica profesional (PRF):** concierne al conocimiento, las habilidades y actitudes que los ingenieros de software deben poseer para practicar la ingeniería del software en una forma profesional, responsable y ética. El estudio de prácticas profesionales incluyen áreas de comunicación técnica, dinámicas de grupo y psicología y responsabilidades profesionales y sociales.
- **Análisis y modelado de software (MAA):** procedimientos que son aplicados inicialmente al análisis, especificación y validación de requerimientos.
- **Diseño de Software (DES):** incluye temas, técnicas, estrategias, representaciones y patrones usados para determinar cómo implementar un componente o un sistema. Se fundamenta en los requerimientos y limitaciones definidas con anterioridad.
- **Validación y Verificación de software (VAV):** usan técnicas dinámicas y estáticas de chequeo del sistema para asegurar que el programa resultante satisfaga su especificación y que alcance las expectativas de los participantes. Las técnicas estáticas corresponden con el análisis y chequeo de representaciones del sistema a través de todos los estados del ciclo de vida del software, mientras que las técnicas dinámicas envuelven solo el sistema implementado.
- **Evolución del software (EVL):** hace referencia al resultado del seguimiento necesario para apoyar la misión de los participantes en pro de cambiar las suposiciones, los problemas, requerimientos, arquitecturas y tecnologías.

- **Procesamiento de Software (PRO):** corresponde al conocimiento sobre la descripción comúnmente usada en los modelos del ciclo de vida del software y en los contenidos de los estándares de proceso institucional; la definición, implementación, medición, administración, cambio y mejoramiento de los procesos de software; y en el uso de un proceso definido para desarrollar las técnicas y actividades necesarias para la implementación y administración del software.
- **Calidad de software (QUA):** es un concepto que afecta y es afectado por todos los aspectos del desarrollo, soporte, revisión y administración del software. Los atributos de calidad de productos de trabajo incluyen funcionalidad, usabilidad, fiabilidad, seguridad, mantenimiento, portabilidad, eficiencia, rendimiento y disponibilidad.
- **Administración de software (MGT):** corresponde al conocimiento sobre la planeación, la organización y el monitoreo de todas las fases del ciclo de vida del software. La administración es crítica para asegurar que los proyectos de desarrollo de software sean apropiados a una organización, el trabajo en diferentes unidades organizacionales sea coordinado, las versiones del software y las configuraciones sean mantenidas, los recursos estén disponibles cuando sea necesario, el trabajo del proyecto esté dividido apropiadamente, la comunicación sea fácil, y el progreso esté avanzando correctamente.

En la tabla 1 se aprecia el resumen de las áreas de conocimiento del SEEK

Tabla 1. Áreas de Conocimiento y Unidades de conocimiento SEEK

Abrev	Título	hrs	Abrev	Título	Horas
CMP	Esencia de computación	172	VAV	V & V de Software	42
CMP.cf	Fundamentos en Ciencia de la computación	140	VAV.fnd	Fundamentos y terminología V & V	5
CMP.ct	Tecnologías de construcción	20	VAV.rev	Revisiones	6

Fuente: IEEE (2004, p. 27)

Tabla 1. Áreas de Conocimiento y Unidades de conocimiento SEEK (Continuación)

CMP.tl	Herramientas de construcción	4	VAV.tst	Pruebas	21
CMP.fm	Métodos de construcción formal	8	VAV.hct	Unidades de pruebas y evaluación	6
			VAV.par	Análisis y reportes de problemas	4
FND	Fundamentos de Matemática e Ingeniería	89	EVL	Evolución del software	10
FND.mf	Fundamentos en Matemática	56	EVL.pro	Procesos de evolución	6
FND.ef	Fundamentos de ingeniería para software	23	EVL.ac	Actividades de evolución	4
FND.ec	Economía de ingeniería para software	10			
PRF	Práctica Profesional	35	PRO	Procesamiento de software	13
PRF.psy	Grupo de dinámica / psicología	5	PRO.com	Concepto de procesamiento	3
PRF.com	Habilidades de comunicación (específicas para IS)	10	PRO.imp	Implementación de procesamiento	10
PRF.pr	Profesionalismo	20			
MAA	Análisis y Modelado de Software	53	QUA	Calidad de software	16
MAA.md	Fundamentos en modelado	19	QUA.cc	Conceptos y cultura de calidad de SW	2
MAA.tm	Tipos de modelos	12	QUA.std	Estándares de calidad de software	2
MAA.af	Fundamentación en Análisis	6	QUA.pro	Procesamiento de calidad de software	4
MAA.rfd	Fundamentación en Requerimientos	3	QUA.pca	Procesos de aseguramiento	4
MAA.er	Fundamentación en especificación	4	QUA.pda	Productos de aseguramiento	4
MAA.rsd	Documentación y especificación de requerimientos	6			
MAA.rv	Validación de requerimientos	3			
DES	Diseño de Software	45	MGT	Administración de Software	19
DES.con	Conceptos de diseño	3	MGT.con	Conceptos de administración	2
DES.str	Estrategias de diseño	6	MGT.pp	Planeación de proyectos	6
DES.ar	Diseño de arquitectura	9	MGT.per	Organización y personal de proyecto	2
DES.hci	Diseño de interface humano-computador	12	MGT.ctl	Control de proyecto	4
DES.dd	Diseño detallado	12	MGT.cm	Administración de configuración de software	5
DES.ste	Herramientas y evaluación de soporte de diseño	3			

Fuente: IEEE (2004, p. 27)

3.3 COMPARATIVO ENTRE LAS DOS GUÍAS (SWEBOK VS. SEEK)

En la Tabla 2 se muestra un comparativo de las áreas de conocimiento de las dos guías SWEBOK y SEEK. Actualizada de [Bourque, 2002]

Tabla 2. Comparativo de las áreas de conocimiento del SWEBOK y el SEEK.

Áreas de conocimiento del SWEBOK	Áreas de conocimiento del SEEK
Fundamentos de Computación	Esencia de la computación
Fundamentos de Matemáticas	Matemáticas y fundamentos de ingeniería
Fundamentos de Ingeniería	
Práctica profesional en Ingeniería del Software	Práctica profesional
Requerimientos de Software	Modelado y análisis de software
Diseño de Software	Diseño de software
Construcción de Software	(No se especifica como tal, pero incluye ítems referentes a este campo dentro de la esencia de la computación y procesamiento de software)
Pruebas de Software	Validación y verificación de software
Mantenimiento de Software	Evolución de software
Proceso de la Ingeniería del Software	Procesamiento de software
Métodos y herramientas de la Ingeniería del Software	No se especifica como área, pero abarca una parte en el área de modelado y análisis de software
Calidad del software	Calidad de software
Gestión de configuración del software	Administración de software
Gestión de la Ingeniería del software	
Economía de la Ingeniería del Software	No se especifica como área, pero se abarca una parte en el área de matemáticas y fundamentos de ingeniería.

Fuente: Autor del proyecto.

De la Tabla 2 se puede apreciar que las diferencias no son sustanciales y que ambas se encargan de abarcar todo el ciclo de vida del software y de identificar a la Ingeniería del Software como una disciplina diferenciada de las demás. En las primeras versiones de las guías, la cobertura del SEEK era un poco más amplia; sin embargo, con la más reciente actualización del SWEBOK, se ampliaron las áreas de conocimiento de 10 a 15, tomando las temáticas que omitía respecto al

SEEK e incluyendo la Economía como área principal. Además, el SWEBOK tiene en cuenta las disciplinas relacionadas, las cuales sugiere se involucren como complemento al currículo.

Ambas guías manejan un área correspondiente a los fundamentos o esencia de la computación. Mientras el SWEBOK separa las áreas de Matemáticas y de Ingeniería, el SEEK las presenta en una misma área, incluyendo además una parte de economía.

En el área de Práctica Profesional, ambas guías proponen temas básicos sobre profesionalismo, psicología y habilidades comunicativas. Las modificaciones implantadas en el proyecto SWEBOK incluyen en su nueva versión esta área, lo cual certifica que se busca reforzar en este tema debido a su importancia e impacto en la formación de nuevos profesionales en este campo.

El área de Requerimientos de Software en el SEEK es llamado Modelado y Análisis de Software, sin embargo, abarcan temas muy similares, incluyendo los pasos de la definición de requerimientos.

El SEEK involucra el concepto de diseño de interface humano – computador dentro del área de diseño de software, mientras que el SWEBOK lo tiene en cuenta en el área de pruebas.

El SWEBOK tiene un área de conocimiento destinada a la construcción de software que involucra los fundamentos, métodos, planeación, lenguajes, codificación entre otros aspectos. Por su parte el SEEK no destinó un área específica para este campo; sin embargo, incluye las tecnologías, herramientas y métodos formales de construcción dentro del área de Esencia de la Computación.

El área de conocimiento de pruebas de Software no se denomina igual en ambas guías, ya que el SEEK la llama Verificación y Validación de Software. No obstante, la temática tratada en ambas es muy similar: fundamentos, técnicas y herramientas. El SEEK profundiza en el análisis y registro de problemas.

El SWEBOK utiliza el término de mantenimiento de software, mientras que el SEEK utiliza el término Evolución del Software, para referirse a la fase de mantenimiento y soporte. En ambas guías se incluyen los fundamentos, los temas claves (impacto, estimación de costos, actividades de mantenimiento) y técnicas de mantenimiento como reingeniería e ingeniería inversa.

Una diferencia marcada en las dos guías, corresponde a la inserción del área de Economía por parte del SWEBOK. Se abarcan temas como: finanzas, flujo de dinero, eficiencia, productividad, economía del ciclo de vida, riesgos e incertidumbre, métodos de análisis de economía, entre otros, que son de vital importancia para evitar riesgos de tipo financiero en el desarrollo de los proyectos. El SEEK incluye parte de esta temática en el área de Matemáticas y Fundamentos de ingeniería.

Mientras el SEEK clasifica los ítems de cada área de conocimiento en Esencial, Deseable y Opcional, el SWEBOK los clasificaba hasta la versión anterior según la taxonomía del conocimiento a tratar: conocimiento, comprensión, aplicación, análisis, evaluación y síntesis; sin embargo, en la actualización más reciente omite esta clasificación y solo hace mención a los tópicos propuestos.

En conclusión, apreciamos que el proyecto SWEBOK ha hecho un esfuerzo por incluir áreas de conocimiento y establecerlas como básicas que son de gran importancia en el ejercicio de la Ingeniería del Software, y por tanto, en la formación de profesionales. Estas áreas, si bien hacen el currículo más extenso, aportan los fundamentos para determinar un plan de estudios más completo.

4 APORTES DE OTRAS UNIVERSIDADES AL PROGRAMA DE INGENIERÍA DEL SOFTWARE

En el presente capítulo se revisan enfoques metodológicos planteados por diferentes instituciones de educación superior que imparten la asignatura de Ingeniería del Software. Se hace un análisis de estas metodologías para verificar cómo apoyan o contradicen el modelo metodológico que se ha venido utilizando hasta la fecha en la Universidad Industrial de Santander. Además, se explorar nuevas propuestas e ideas fundamentadas que aporten al enfoque UIS con miras al logro de la excelencia en la enseñanza de la ingeniería del software.

4.1 OBJETIVOS Y COMPETENCIAS DEL CURSO

Según la experiencia del director, un curso de enseñanza de Ingeniería del Software debe buscar proveer un espacio de formación para:

- La ejercitación de valores: promulgar el liderazgo, el cooperativismo, la puntualidad y pulcritud en la entrega de informes y trabajos, la honestidad, entre otros.
- La apropiación de fundamentos conceptuales: enfocarse en el estudio del cuerpo del conocimiento de la Ingeniería del Software. Para el logro de este objetivo, es fundamental contar con el apoyo de guías o estándares que aporten el marco conceptual del área, que tal como se abordó en el capítulo 3 de este documento, se utilizan el SWEBOK y el SEEK.
- El desarrollo de habilidades para la realización de software de calidad: ejercitar, mediante proyectos de software, las buenas prácticas de la Ingeniería del Software.

Universidades como California State University [Winbladh, 2004] y el Instituto de Tecnología Terre Haute [Surendran, 2000] promueven que la enseñanza de la Ingeniería del Software debe imitar los procesos industriales de software [Mirian-

Hosseinabadi, 2010] [Hilburn, 1999] [Návrat, 1999], de modo que se prepare a los estudiantes para los retos del mundo real en el desarrollo de sistemas profesionales, pues autores como Cockburn [Cockburn, 2006] y Anaya [Anaya, 2006] comentan que es precisamente esta una falencia de la enseñanza en este campo. Incluso, existen estudios donde se muestra que las empresas de software latinoamericanas se encuentran en desventaja frente a los niveles de competitividad que exigidos a nivel internacional [Anaya, 2006], factor atribuible a la debilidad en la formación en este campo.

4.2 CARACTERÍSTICAS GENERALES DE LOS MODELOS DE E/A DE INGENIERÍA DEL SOFTWARE

Para poder determinar las características que darían forma al modelo de enseñanza / aprendizaje que se va a proponer, es necesario tener presentes las metas que se desean obtener:

- Estudiantes con formación suficiente para enfrentar el mundo laboral, que dediquen el tiempo necesario a su preparación. Para esto se requiere analizar el tiempo semanal dedicado al estudio de la asignatura (Jornada de trabajo). Además, como se ha mencionado anteriormente, en el proceso de formación se debe enfocar a los requerimiento de la industria, por tanto es fundamental analizar cómo se siguen los proceso de desarrollo de software a nivel de equipos de trabajo (Formas de trabajo).
- Estudiantes con formación basada en competencias, acorde con el enfoque pedagógico actual, donde la educación se base en el estudiante y en su proceso de enseñanza; por tanto se estudiarán las propuestas en términos de las Estrategias de Aprendizaje.
- Uso de estándares y guías que apoyen el estudio de la asignatura. En este aspecto, como se analizó en el capítulo 3, el cuerpo del conocimiento de la asignatura será determinado por las guías SWEBOK y SEEK, las cuales proponen el uso de Herramientas computacionales de apoyo.

- Como complemento a la formación por competencias, centrado en el proceso de aprendizaje de los estudiantes, se hace necesario el análisis de los Criterios de Evaluación que se usarán en la metodología de E/A.

A continuación se presentan los aportes de diversas universidades en cada una de las características que formarán parte de la metodología a proponer

4.2.1 Jornada de trabajo. En la asignatura de IS impartida en la Universidad Industrial de Santander se proponen 4 horas teóricas y 8 horas de trabajo independiente: es común que los estudiantes tengan que dedicar tiempo extra para su preparación [Galeana, 2007], como parte del proceso de aprendizaje activo, que inculca a los estudiantes el interés y la disciplina para construir su conocimiento y experimentar activamente utilizando las herramientas que le proporcionan; por ejemplo en instituciones como la Universidad de los Andes, se espera que cada estudiante trabaje 3 horas adicionales a las 3 horas de clase [Casallas, 2002], lo cual indica que se espera que por lo menos trabajen en casa el mismo tiempo dedicado en clase. Este tiempo extra tiene su fundamento en estudios que se han desarrollado basados en el desempeño esperado en el ámbito industrial, tal como lo sugiere Kristina Windbladh [Winbladh, 2004], quien analiza que la tasa de desempleo en el área de ingeniería, y especialmente en ingenieros en el área de ciencias de la computación, ha aumentado (debido a la recesión económica a nivel mundial), motivo por el cual se hace fundamental que los estudiantes, quienes se encuentran en su proceso de formación, reconozcan la importancia de mejorar su preparación y descubrir qué habilidades está buscando la industria. Por tanto, es imperativa la dedicación extra de cada estudiante para aprovechar mejor las enseñanzas obtenidas en el curso.

4.2.2 Formas de trabajo. En instituciones de prestigio como California State University [Winbladh, 2004], el Institute of Technology Terre Haute [Surendran, 2000], Humans and Technology Technical Report [Cockburn, 2006] y la Universidad de Colima [Galeana, 2007] han estado aplicando el enfoque de trabajo en grupo, integrando herramientas, metodologías, clases teóricas y talleres, logrando buenos resultados [Casallas, 2002]: buen desempeño de los grupos, satisfacción de los requerimientos, calidad en los productos desarrollados, verdadera comprensión de los conceptos, desarrollo de habilidades y afianzamiento de una disciplina para desarrollar software con compromiso de calidad.

Por su parte, en la Universidad EAFIT [Anaya, 2006] y la Ecole Polytechnique de Montreal [Robillard, 2004] sugieren una mezcla de trabajo personal y trabajo en equipo, como "instrumento de formación integral del aprendiz". En la Universidad de Colima [Galeana, 2007] y en la Universidad de los Andes [Casallas, 2002], incluso sugieren la definición de roles en el trabajo por equipos.

4.2.3 Estrategias pedagógicas. El cambio de paradigma que ha venido enfrentando en los últimos años la educación, donde el rol del estudiante debe ser mucho más activo, ha inspirado el uso de metodologías de enseñanza y aprendizaje que lo promuevan; tal es el caso del llamado Aprendizaje Basado en Proyectos (ABP), donde las actividades que se realizan son fundamentalmente prácticas, motivando el desarrollo de habilidades, competencias en el área de estudio y participación activa y constante de los estudiantes; en una sola frase, revolucionando la dinámica diaria de la clase hacia un enfoque práctico.

En diversas universidades [Winbladh, 2004], [Surendran, 2000], [Cockburn, 2006], [Anaya, 2006], [Galeana, 2007], [Casallas, 2002], [Robillard, 2004], [Port, 2001], se desarrollan cursos de Ingeniería del Software organizados alrededor de un proyecto (que en el nuestro es un caso de aplicación – proyecto de software

pequeño), acompañado de clases teóricas y talleres (guiados por el docente) para fortalecer las bases conceptuales y orientar el proceso de desarrollo de los estudiantes.

Mediante la aplicación del ABP los estudiantes aprenden a aprender a través del trabajo en grupo [Robillard, 2004], [Santa, 2008], [Huber, 2008], [Zapata, 2008], además su incursión ha sido parte de la evolución de la enseñanza de la Ingeniería del Software [Aguilar Sierra, 2003]. Así lo exponen estos autores de la Universidad de Murcia, quienes aseveran que es en la propia realización del proyecto donde se encuentran nuevas necesidades de aprendizaje, que los estudiantes deben subsanar por ellos mismos, de forma cooperativa a través de sus grupos de trabajo. Lo anterior refuerza el interés de la IS de lograr un producto de calidad y un proceso óptimo de desarrollo de software, mediante la utilización de dinámicas de clase que favorezcan este ambiente de aprendizaje significativo.

En la Universidad de Colima también están de acuerdo con la aplicación de esta metodología [Galeana, 2007], pues indica que los estudiantes aprenden a descubrir sus propios errores, están motivados para el uso de tecnología y desarrollan habilidades tales como la colaboración, comunicación, toma de decisiones, responsabilidad por el propio aprendizaje y manejo del tiempo.

Otro punto a favor de aplicar el Aprendizaje Basado en Proyectos lo describe [Winbladh, 2004], quien señala que el uso de esta metodología mejora las habilidades de resolver problemas para el mundo real y las habilidades de comunicación de los estudiantes.

4.2.4 Herramientas de apoyo. El uso de herramientas software como apoyo para el proceso de Enseñanza / Aprendizaje de la Ingeniería del Software se hace fundamental, debido a que es un área de estudio donde se trabaja en torno a la tecnología software. Además, es innegable la capacidad que tienen las TIC para el aprovechamiento de las diversas habilidades comunicativas de los estudiantes. [Aguilar Sierra, 2003], [Granda, 2010].

Dentro de la asignatura de IS es común establecer el uso de herramientas para Modelos UML (básicas dentro del marco del aprendizaje de análisis y diseño de requerimientos). Si dentro del curso se plantea el desarrollo de un prototipo funcional, se hace necesaria la utilización de una aplicación software que cumpla con los fundamentos planteados y que sea de fácil aprehensión por parte de los estudiantes. No debe ser objetivo del curso enseñar o ampliar los conocimientos en estas herramientas, sino que deben servir como apoyo para realizar un mejor proyecto aplicado. Los estudiantes deberán escoger aquella con la cual tengan mayor familiaridad o aquella que consideren de mejor aprovechamiento según su caso de aplicación.

En la Universidad de los Andes proponen el uso de herramientas para diagramas UML como ArgoUML, Dia o Rose [Casallas, 2002], pero enfatizan en que los estudiantes tengan conocimiento previo de ellas.

4.2.5 Criterios de evaluación. Es importante evaluar el progreso de cada grupo (cuando se trabajan en equipos), y por ende de los estudiantes en particular, mediante entregas incrementales de las tareas y artefactos. Así, como la enseñanza de los fundamentos teóricos que se van proporcionando a medida que avanza el semestre y se van requiriendo los conceptos para ser llevados a la práctica en el caso de aplicación.

Esta metodología, de entrega incremental en el marco del aprendizaje basado en proyectos, es apoyada por Alistair Cockburn en su investigación sobre el tema [Cockburn, 2006]. Por su parte, quienes más promueven el desarrollo incremental son las metodologías ágiles y sus promotores, debido a que se orientan principalmente hacia la gente y son de carácter adaptativo [Aguilar, 2003]. Estas metodologías se recomiendan para proyectos y equipos pequeños y requerimientos cambiantes, condiciones que se cumplen en la academia, debido a que, por estar en una fase de aprendizaje, es lógico que no se logren definir los requerimientos desde un comienzo y se desencadenen una serie de cambios a medida que se va comprendiendo mejor el sistema y se va madurando en el conocimiento.

Dentro de las metodologías ágiles más conocidas y aplicadas se encuentran XP, SCRUM, RUP y FDD. En la Universidad de Costa Rica confirman que la aplicación de estas metodologías "logró aligerar el proceso de desarrollo al flexibilizar las exigencias de los entregables que los estudiantes desarrollan" asegurando la calidad del producto. Además, "la aplicación de XP mejora la construcción de los componentes, garantizando el cumplimiento de los requerimientos y la satisfacción del usuario". [Salazar, 2012],

Por otra parte, en la evaluación planteada se debe tener en cuenta un aspecto muy importante: el desarrollo de las habilidades comunicativas, como lo plantea claramente Mark Michael [Michael, 2000], quien en su proceso de evaluación utiliza dos formatos (reportes escritos y presentación en público del proyecto), donde tiene en cuenta el contenido, la organización, los mecanismos y el estilo demostrado por los estudiantes. Otro aspecto que destaca Mark Michael y que se usa en el plan de trabajo de la asignatura, es el hecho de exponer claramente a los estudiantes la forma de evaluación que será usada. El asegura que si el estudiante conoce con exactitud cómo será evaluado, podrá enfocar su atención a las habilidades deseadas y en la dirección esperada.

Por otra parte se debe considerar en la metodología de aprendizaje, es el proceso guiado del estudiante, evidenciado a través de las tareas, del entendimiento paulatino del problema a resolver. Esto lo resalta Pierre Robillard [Robillard, 2004], quien destaca el objetivo de un curso orientado a proyectos: la síntesis e integración del conocimiento en diseño de los estudiantes. Por eso, es muy importante la realización de modelos, organización de subsistemas, descripciones, listados de entradas y salidas, etc., para poder entender adecuadamente el caso de estudio y poder llegar al diseño y a la codificación con una idea suficientemente clara para el desarrollo del sistema.

El docente juega un papel fundamental, pues no solo imparte los conceptos teóricos sino que también es el guía en el proceso de aseguramiento de calidad del producto entregado [Casallas, 2002]. En la Universidad de Colima, se sugiere que la evaluación debe ser real e integral, algunas veces denominada "valoración de desempeño" [Galeana, 2007], que en el caso que nos ocupa se tienen en cuenta los siguientes criterios: puntualidad, aplicación disciplinada de la metodología, completitud, claridad, consistencia y creatividad en el desarrollo de los artefactos de software.

Ken Surendran y Frank Young [Surendran, 2000] indican la conveniencia de usar plantillas y estándares en el momento de la evaluación. Por su parte, en la Universidad de los Andes, los grupos deben entregar un documento en el cual especifican los participantes de los equipos, definen las metas individuales y grupales, identifican los riesgos del producto y hacen una estimación preliminar del tamaño del mismo [Casallas, 2002].

A su vez, [Robillard, 2004] motiva a la utilización de hitos o puntos de corte de la evaluación, donde se tiene en cuenta la completitud y el progreso de las entregas. El autor [Robillard, 2004] señala que todos los documentos son construidos desde

plantillas existentes. Además, avala que los productos software desarrollados en el transcurso del curso sean evaluados basados en pruebas de aceptación.

4.2.6 Metodología actual aplicada en la Universidad Industrial de Santander

Como parte preliminar de una investigación que busca la formalización de una metodología de enseñanza / aprendizaje de la Ingeniería del Software I, apoyada en TIC y basada en otras propuestas en el área y en la experiencia propia de la institución, se presenta una descripción del estado actual de la enseñanza en este campo en la Universidad Industrial de Santander, quien tiene dos cursos de Ingeniería del Software dentro del programa de Ingeniería de Sistemas, los cuales se desarrollan en el cuarto año de carrera. Esta descripción data del segundo semestre de 2011. El plan de trabajo de ese momento se incluye en el Anexo 1.

Dentro de la fundamentación teórica se estudian: (1) Introducción a la Ingeniería del Software, incluyendo origen y situación actual, ciclos de vida clásicos y modernos de desarrollo de software, comunicación en proyectos y métodos de obtención de requerimientos (intrusivos y no intrusivos). (2) Modelado del Negocio: definición de actores y eventos del negocio, Casos de Uso Esenciales y Diagramas de Secuencia del Sistema. (3) Requerimientos de Software: definición del Modelo de Contexto, Escenarios y Casos de Uso, Clases del dominio de aplicación. (4) Análisis de Requerimientos: Arquitectura MVC (Modelo Vista Controlador), Modelos de comportamiento dinámico: Diagramas de Secuencia y Diagramas de Estado, prototipo. (5) Diseño: Diseño del Sistema y Diseño de Objetos. Conceptos de diseño de GUIs (interfaces gráficas de usuario) y diseño de interfaces externas. El desarrollo de estos contenidos temáticos se apoya en el cuerpo del conocimiento presentado en el SWEBOK.

Los estudiantes deben trabajar durante todo el semestre en un caso de aplicación de su elección, que en algunos casos se apoyan de clientes reales y en otros no. Sin embargo, la definición de los requisitos del producto es realizada por los

estudiantes, ya sea mediante captura de información a través del cliente o por especificación personal basada en ejemplos y experiencias del docente o propias y documentación.

Las tareas principales solicitadas a lo largo del curso son:

- Identificación de Casos de Uso.
- Especificación del modelo de Negocios.
- Definición del modelo de Requerimientos.
- Desarrollo del modelo de Análisis.
- Realización de un prototipo de Software parcialmente funcional (se hace entrega de un diagrama de navegación y un diseño detallado de la interfaz).

Cada equipo debe presentar a todos los grupos los artefactos y documentos obtenidos en cada tarea, para socializar los resultados y analizar su proceso, habilidades en la comunicación, consistencia y creatividad. El docente juega un papel de guía y facilitador de los procesos de aplicación y aprendizaje. Es además, un administrador de la calidad de los productos.

En el desarrollo del curso se brinda autonomía a los estudiantes para el uso de herramientas para Modelos UML, como mecanismo de incentivar la búsqueda e investigación fuera de las horas de clase.

Se evalúan las tareas y presentaciones de los artefactos, teniendo en cuenta los siguientes aspectos:

- Aplicación disciplinada de la metodología.
- Completitud de los artefactos solicitados.
- Puntualidad en la entrega.
- Claridad en la comunicación de ideas.
- Consistencia.

- Creatividad y uso de herramientas de apoyo.

A continuación se describen los formatos de las tareas o talleres propuestos en la Universidad Industrial de Santander, diseñados por el docente Fernando Rojas, para el desarrollo del curso de Ingeniería del Software I.

4.2.6.1 Tarea 1. Se realiza una breve, pero precisa descripción del sistema deseado (Requerimientos –C: Requerimientos de Software enfocados en el Cliente) en el caso de aplicación del grupo, se especifican los interesados y se detalla una primera tabla de eventos del negocio (evento, entrada al sistema, actor que provee la entrada, salida del sistema y actor que recibe la salida). Finalmente se desarrolla una lista con los problemas, oportunidades y objetivos asociados al caso de aplicación.

Esta tarea posee una plantilla que, al ser diligenciada, se convierte en el primer entregable del proyecto de cada equipo.

4.2.6.2 Tarea 2. Consiste en describir con detalle los requerimientos –D (Requerimientos de Software enfocados en el Desarrollador o, dicho de otra forma, perspectiva del desarrollador); incluye una descripción del sistema deseado y el bosquejo del modelo de casos de uso (diagrama y especificaciones), un prototipo desechable inicial desarrollado con tecnología básica (puede ser con papel y lápiz) y el modelo del dominio de aplicación (un diagrama de clases UML), junto con una descripción de las clases. Esta idea del uso de modelos es apoyada por Daniel Port y Barry Boehm [Port, 2001], quienes lo promueven basados en la idea que esta práctica provee una exposición tangible a importantes conceptos de la Ingeniería del Software, tales como la gerencia de proyectos, arquitectura del producto, adquisición de requerimientos, entre otros, sugiriendo que se haga de forma consciente y sistemática.

Otra justificación para el desarrollo de esta tarea, se fundamenta en el punto de vista de la definición, donde la captura de requisitos se basa en describir el propósito del sistema [Bruegge, 2004]. Para esto, se deben desarrollar ciertas tareas que formalicen esta labor. Dentro de estas actividades se encuentran: identificar a actores, escenarios, casos de uso (y refinarlos), las relaciones entre los casos de uso y los requerimientos no funcionales. Más adelante, en un proceso incluido en la tarea Nº2, se pasa a la etapa de análisis de los requerimientos, que es más formal y conlleva al modelo del dominio (identificación, diagramación y descripción de clases).

En la especificación de los casos de uso, es fundamental hacer una descripción textual detallada de cada uno de los actores y casos de uso identificados, como lo propone Weitzenfeld [Weitzenfeld, 2005], quien sugiere un formato que se implanta en esta tarea (nombre del caso de uso, propósito, precondiciones, postcondiciones, flujo principal y flujos alternos).

4.2.6.3 Tarea 3. Se profundiza aún más la etapa de análisis de los requerimientos, dando cabida a la arquitectura de clases MVC: Modelo – Vista - Controlador (objetos de borde, de entidad, de control y diagramas de clases refinados, de secuencia y de estados). Finalmente, se solicita el diagrama de navegación de ventanas para dejar un diseño inicial del prototipo y al final del curso entregar el prototipo funcional. Esta labor es un reflejo de lo propuesto por muchos autores, entre los que se encuentran Bernd Bruegge y Allen Dutoit en (Brueggel, 2004). Ellos plantean que modelar el sistema con objetos de entidad, borde y control tiene muchas ventajas, tales como: diferenciación de objetos y funcionalidades.

En la tabla 3 se presenta un cuadro resumen y comparativo de los aportes de diversas universidades basados en las características básicas de los métodos de enseñanza de la asignatura.

Tabla 3. Resumen comparativo aportes de diversas universidades en el método de enseñanza de IS.

UNIVERSIDAD /Autor	JORNADA DE TRABAJO	FORMA DE TRABAJO	ESTRATEGIAS DE APRENDIZAJE	HERRAMIENTAS DE APOYO	CRITERIOS DE EVALUACIÓN
Universidad Industrial de Santander	4 horas teóricas y 8 de trabajo independiente	Trabajo en grupo, sin definición de roles	Aprendizaje basado en proyectos	Herramientas de modelado UML, a elección de los estudiantes.	Presentación de tareas que muestran avances en el desarrollo del proyecto. Evaluaciones intermedia y final. Uso de plantillas para entregas.
Universidad de Colima – México / Galeana, 2007	Es común que los estudiantes tengan que dedicar tiempo extra para su preparación	Trabajo en grupo y definición de roles	Aprendizaje basado en proyectos	Procesadores de texto, programas de diseño gráfico, bases de datos, internet, cámaras fotográficas	Evaluaciones y autoevaluaciones del proceso
Universidad de los Andes – Colombia / Casallas, 2002	3 horas de clase y 3 horas de trabajo independiente	Trabajo en equipos de 5 - 6, implementación de roles	Aprendizaje basado en proyectos	Herramientas de diagramación UML como ArgoUML, Dia o Rose	Entregas de documento previo al inicio del proyecto, definiendo metas individuales y grupales, riesgos del producto y estimación preliminar del mismo.
California State University – Estados Unidos / Winbladh, 2004	Dedicar tiempo extra para mejorar formación de estudiantes en miras al desempeño esperado en el ámbito industrial	Trabajo en grupo	Aprendizaje alrededor de un proyecto		

Fuente: Autor del proyecto

Tabla 3. Resumen comparativo aportes de diversas universidades en el método de enseñanza de IS. (Continuación)

UNIVERSIDAD /Autor	JORNADA DE TRABAJO	FORMA DE TRABAJO	ESTRATEGIAS DE APRENDIZAJE	HERRAMIENTAS DE APOYO	CRITERIOS DE EVALUACIÓN
Institute of Technology Terre Haute / Surendran, 2000		Trabajo en grupo, con 2 o 3 integrantes. Juego de roles por equipos	Aprendizaje alrededor de un proyecto		Uso de plantillas y estándares.
Humans and Technology Report / Cockburn, 2006		Trabajo en grupo	Aprendizaje alrededor de un proyecto		Entregas incrementales de avances del proyecto
Universidad EAFIT - Colombia / Anaya, 2006		Trabajo personal y trabajo en equipo. Definición de roles	Aprendizaje basado en proyectos colaborativos		
Ecole Polytechnique de Montreal / Robillard, 2004		Trabajo personal y trabajo en equipo	Aprendizaje alrededor de un proyecto	Herramientas CASE, SCHEMACODE, estación de trabajo SUN y correo electrónico para comunicarse entre equipos	Realización de tareas a través de modelos, organización de subsistemas, descripciones, listados de entradas y salidas, etc. Apoya uso de plantillas y el uso de hitos o puntos de corte de avance del proyecto
University of Sourthern California / Port, 2001			Aprendizaje alrededor de un proyecto	Herramientas de modelado	
Universidad de Murcia – España / Santa, 2008			Aprendizaje basado en proyectos		

Fuente: Autor del proyecto

Tabla 3. Resumen comparativo aportes de diversas universidades en el método de enseñanza de IS. (Continuación)

UNIVERSIDAD /Autor	JORNADA DE TRABAJO	FORMA DE TRABAJO	ESTRATEGIAS DE APRENDIZAJE	HERRAMIENTAS DE APOYO	CRITERIOS DE EVALUACIÓN
Universität Tübingen – Alemania / Huber, 2008			Aprendizaje basado en proyectos y en problemas		
Universidad Nacional de Colombia / Zapata, 2008			Aprendizaje basado en proyectos		
Universidad Nacional Autónoma de México / Aguilar Sierra, 2003		Grupal. Programación en pares	Aprendizaje basado en proyectos	Aplicación de TIC para el aprovechamiento de habilidades comunicativas de los estudiantes	Entregas incrementales de avances del proyecto. Desarrollo dirigido a pruebas.
Universidad de las Ciencias Informáticas – Cuba / Granda, 2010				Aplicación de TIC para el aprovechamiento de habilidades comunicativas de los estudiantes	Evaluación formativa y sumativa.
Universidad de Costa Rica / Salazar, 2012			Aprendizaje basado en proyectos		Aplicación de metodologías ágiles para dar seguimiento a proyectos
King's College – Estados Unidos / Michael, 2000			Aprendizaje basado en proyectos		Evaluación de habilidades comunicativas: reportes escritos y presentación en público del proyecto

Fuente: Autor del proyecto

5 HERRAMIENTAS TIC COMO SOPORTE AL ANÁLISIS DE REQUERIMIENTOS

En el capítulo 3, dentro del cuerpo del conocimiento de la Ingeniería del Software, se presenta como tema esencial de la asignatura la aplicación de herramientas de apoyo para la etapa de Requerimientos de Software, las cuales se enmarcan en dos categorías: herramientas para el modelado y herramientas para la administración de requerimientos. En el presente capítulo, se abordarán herramientas TIC de apoyo, tales como el uso de Aulas virtuales, pero se enfocará especialmente en el uso de herramientas de modelado, por ser la etapa de Requerimientos de Software el eje principal de la asignatura Ingeniería del Software I.

5.1 TIC EN LA EDUCACIÓN

La velocidad con que se mueve el mundo actual y el continuo y rápido avance en los medios masivos de comunicación ha creado la necesidad en la población de vincularse activamente con la información, con el fin de renovar y ampliar su conocimiento y estar a la vanguardia del modernismo. Sin lugar a dudas, esta necesidad ha venido siendo favorablemente satisfecha por la aparición y evolución de la tecnología, como es el caso de campos tan complejos y competitivos como el de la educación, que ha visto la importancia de utilizar los adelantos tecnológicos para el apoyo del proceso de enseñanza - aprendizaje, teniendo en cuenta que cada estudiante tiene su propio ritmo de apropiación del conocimiento, sus propias necesidades y diferentes habilidades y aspectos por mejorar.

Dentro de la diversa gama de avances que se han desarrollado, las TIC (Tecnologías de Información y Comunicación) aportan acceso a todo tipo de información, permiten realizar procesamiento de datos y poseen canales de comunicación inmediata, sincrónica y asincrónica, para compartir información con personas e instituciones de cualquier parte del mundo, lo cual promueve la

globalización del proceso de aprendizaje en cualquier ámbito. Precisamente, en 2008 aparece un nuevo término sobre TIC denominado MOOCs (Massive Open Online Course – Curso en línea abierto y masivo) que hace referencia a un curso online que se dirige a una amplia participación y que tiene libre acceso a través de Internet. Una ventaja clara es la masificación de la información, aunque no se recibe titulación, por lo que no es considerado dentro de la educación formal. Dentro de las múltiples ventajas del uso de TIC se encuentran la automatización de tareas e interactividad, la posibilidad de almacenar gran cantidad de información en múltiples soportes portables y la digitalización estándar de la información. [Marqués, 2008]

La proliferación de las TIC se magnificó gracias al desarrollo y uso extendido de Internet, lo que ha permitido la creación de entornos de aprendizaje novedosos y de gran impacto para la población, tales como los llamados campus virtuales y aulas virtuales, que sirven para apoyar el proceso educativo, llegando a todo tipo de población y ofreciendo una infinidad de herramientas y material didáctico que, sin duda, agiliza y mejora la labor docente en términos de seguimiento del proceso de aprendizaje y variedad en la oferta de instrumentos formativos de apoyo.

Aunque la expresión Campus Virtual se viene manejando aproximadamente desde 1997, aún no se ha definido de forma precisa y puntual. Sin embargo, la Comisión Europea [European Commission] ha planteado un acercamiento general que sirve para acotar el término: “Cooperación entre instituciones de Educación Superior en el campo de aprendizaje a través de Internet, con respecto a: diseño de desarrollo de planes de estudio para diversas universidades, incluyendo acuerdos para la evaluación, validación y reconocimiento de competencias adquiridas, sujetos a procedimientos nacionales, experimentos a gran escala de movilidad virtual en adición a la movilidad física y desarrollo de planes de estudio innovadores, basados tanto en métodos de aprendizaje tradicionales como on-line. “. Esto es,

toda una infraestructura tecnológica que permite ofrecer todos los servicios educativos de una Institución de Educación Superior, a través de Internet.

Por su parte, Aula Virtual se considera al entorno de enseñanza – aprendizaje que se basa en la comunicación mediada por el computador [Gisbert], es decir, se desarrollan aplicaciones telemáticas para implementar comunicación en la educación a distancia, buscando obtener resultados de calidad, tal como la educación presencial.

Según el Dr. Pere Marqués [Marqués, 2008], las principales funcionalidades que ofrecen los Campus Virtuales son:

- Plan docente como apoyo en el sistema de evaluación.
- Material de apoyo y consulta.
- Servicio de correo electrónico para mantener activa la comunicación entre docentes y estudiantes.
- Tablón de anuncios y calendario para informar sobre próximas actividades.
- Servicio de foros, para debatir sobre puntos de vistas alrededor de los diversos temas tratados, incluso permitiendo la participación de otros especialistas invitados.
- Espacios e instrumentos para el trabajo colaborativo.
- Plantillas creadas por los docentes de pruebas de evaluación objetivas.
- Facilidades administrativas para los docentes: creación y modificación de listas de estudiantes, entrada parcial y total de calificaciones, publicación de material, etc.
- Facilidades administrativas para los estudiantes: consulta de calificaciones, solicitudes de información, comunicación sincrónica y asincrónica con el docente y compañeros de estudio, publicación de trabajos y tareas, incluso la presentación de exámenes en tiempo real.

- Acceso a enlaces de interés, posiblemente de otras instituciones educativas a nivel mundial.
- Servicios de teleconferencias.

Aunque un encuentro presencial docente – estudiante siempre será mejor visto que uno virtual, la posibilidad de apoyar la labor educativa en un momento en el cual se dificulte el desplazamiento, se desee reforzar el proceso de adquisición del conocimiento con la publicación en Internet de instrumentos educativos extras o simplemente sea innecesario por la brevedad de la consulta, es una ventaja que ofrecen las TIC, junto con otros cambios importantes que implica su uso [Marqués, 2008]:

- Mayor universalización de la información: el material de apoyo se puede "publicar" en la web para que la comunidad educativa tenga libre acceso a él, en el momento que lo considere necesario.
- Enfoques crítico-aplicativos para el autoaprendizaje: el estudiante ya dispone de la información, ahora su preocupación primordial es apropiarse de ella de forma crítica, analítica y aplicativa, mediante el asesoramiento del docente.
- Actualización de los programas: debido a la diversidad de información que está en la web, se hace necesario que el docente esté en constante renovación.
- Trabajo colaborativo: se comparte información y soluciones ante situaciones problemáticas para apoyar el proceso de aprendizaje propio y de los compañeros de curso.
- Construcción personalizada de aprendizajes significativos: cada individuo puede utilizar los instrumentos que requiera, una y otra vez, según su avance en el proceso de aprendizaje, esto es, puede ir a su propio ritmo y según su necesidad (sin salirse de los parámetros y la normatividad establecidos por el docente administrador de la plataforma).

5.1.1 TIC en la enseñanza de la Ingeniería del Software. En la Ingeniería del Software se busca principalmente agilidad y calidad del proceso de desarrollo y del producto final, por lo cual es muy importante evaluar y dar seguimiento a cada etapa que se lleva a cabo en la industria del software [Granda, 2011]. Por tanto, la enseñanza de esta área del conocimiento se ha convertido en un reto para las Instituciones de Educación Superior que se mueven en este campo, la cual no ha sido ajena al desarrollo alcanzado por las TIC y su inmersión en la educación.

Las ventajas que pueden significar la incorporación de la tecnología en la enseñanza de esta disciplina son variadas, pues promueve el desarrollo de competencias que harán de los Ingenieros de Software profesionales más competitivos en su labor. Estas competencias se basan en el saber hacer: teniendo la información requerida a su disposición, un estudiante deberá estar en la capacidad de poder procesarla y asimilarla de manera correcta, gracias a los diversos mecanismos que se entrelazan al tiempo en el entorno educativo y a la orientación oportuna del docente.

En los últimos años se ha empezado a desarrollar una infraestructura llamada plataforma virtual o EVA (Entornos Virtuales de Aprendizaje), la cual condensa en un solo lugar una infinidad de herramientas tecnológicas educativas que permiten la interacción entre docentes, estudiantes y contenidos de los cursos, de una forma ágil, organizada y de fácil planeación.

Dentro de las diversas ventajas para los docentes que ofrecen los EVA, están [WizIQ]:

- El docente puede llevar el control de los cursos (contenidos publicados, tareas, calificaciones, cronogramas de trabajo, materiales de apoyo, entre otros).
- Permite el manejo de diferentes formatos de archivos electrónicos.
- Facilita el proceso de seguimiento a los estudiantes.

- Ofrece comunicación sincrónica y asincrónica.
- Registra los movimientos realizados: actividades y fechas de entregas y publicaciones.

Por su parte, los estudiantes también tienen ventajas al utilizar los EVA [WizIQ]:

- Posibilidad de entregar trabajos y tareas de forma electrónica, ahorrando tiempo, esfuerzo y dinero, al no tener que imprimirlos ni llevarlos físicamente.
- Acceso a los contenidos publicados por el docente, en cualquier momento y lugar, incluso en repetidas ocasiones si es necesario.
- Facilidad para revisar calificaciones.
- Comunicación constante con el docente y demás compañeros de estudio.

Todas estas funciones y ventajas pueden y deben ser aprovechadas por la enseñanza de la Ingeniería del Software, de modo que contribuyan en el proceso de aprendizaje de una forma notoria y satisfactoria.

5.1.2 La Universidad Industrial de Santander como parte activa del uso de TIC. La Universidad Industrial de Santander, en su interés por ofrecer mecanismos educativos de calidad y modernos, no ha sido ajena a estos avances tecnológicos y a los beneficios de su aplicación. Por esto, ha venido desarrollando herramientas para apoyar a sus estudiantes en los diversos campos de estudio que ofrece. Específicamente dentro de la Escuela de Ingeniería de Sistemas se han desarrollado dos plataformas: MeiWeb y una Aula Virtual de la Escuela; las dos han sido desarrollada por estudiantes bajo la guía de profesores de planta.

5.1.2.1 MeiWeb – Descripción

a) **Ingreso**: es una plataforma web, donde pueden ingresar a revisar contenidos los usuarios registrados previamente por el docente, quien juega el papel adicional de administrador del entorno en sus clases. El ingreso al

sitio se hace a través de la dirección electrónica: sistemas.uis.edu.co/Meiweb.

b) **Pantalla de inicio:** la Figura 2 presenta la pantalla de inicio de MeiWeb, tal como se aprecia una vez se ingresa la URL.

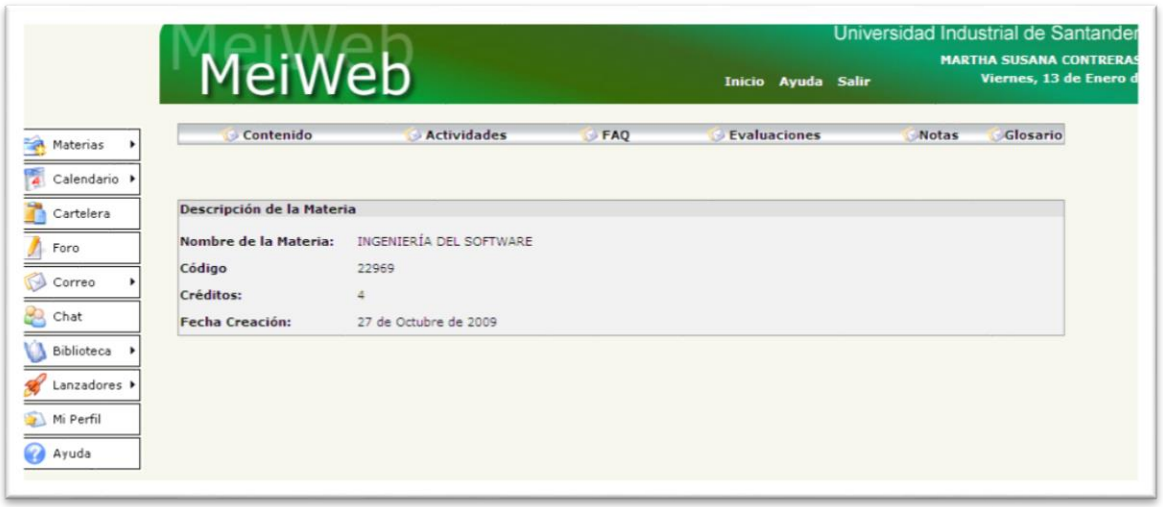
Figura 2. Pantalla de inicio - MeiWeb.

Fuente: Captura de pantalla de sistemas.uis.edu.co/Meiweb.

Se realiza el inicio de sesión para tener acceso a los contenidos de las asignaturas, a través del perfil que corresponda: virtual o presencial.

c) **Pantalla de usuario**: una vez se ha registrado el usuario (a través de contraseña), se puede ver el contenido y demás aplicaciones habilitadas para el estudiante. Ver Figura 3.

Figura 3. Pantalla de usuario de MeiWeb.



Fuente: Captura de pantalla de sistemas.uis.edu.co/Meiweb.

d) Secciones por asignatura:

- Contenido del curso: el docente publica documentos de interés para los alumnos en el desarrollo de la asignatura. Los archivos pueden ser imágenes, videos, documentos en formato de lectura pdf, Word o incluso aplicaciones, las cuales se organizan por capítulos para mayor especificación. Cada documento puede poseer aproximadamente hasta 10 MB.
- Actividades: descripción de tareas o sesiones en general.
- FAQ: preguntas frecuentes.
- Evaluaciones: el docente puede publicar material para evaluación, que debe ser completado por el estudiante en un tiempo límite.
- Notas: el docente puede publicar las calificaciones de sus estudiantes, proporcionando el porcentaje a cada actividad, según lo considere. De esta forma, cada estudiante puede revisar constantemente su proceso evaluativo, teniendo conocimiento desde el principio del curso sobre el peso o valor dentro del puntaje final que tiene cada ítem.

- Glosario: palabras clave y definiciones de utilidad para los temas que abarca la asignatura.

e) Aplicaciones:

- Materias: asignaturas en las cuales está registrado el estudiante y dentro de las cuales puede alternar para ver sus contenidos.
- Calendario: cronograma de actividades. El docente puede dar aviso por escrito a sus estudiantes de las diversas actividades a realizar dentro de la asignatura que dirige.
- Cartelera: publicación de mensajes de interés.
- Foro: discusiones sobre temas de interés en el cual pueden participar los estudiantes y el docente.
- Correo: facilita la comunicación con el docente y con los compañeros de clase a través de la recepción y envío de correos electrónicos, teniendo acceso al correo de los usuarios registrados por el docente. Permite adjuntar archivos.
- Chat: conversaciones establecidas por el docente sobre algún tema en particular o en un momento abierto para aclarar dudas.
- Biblioteca: se pueden apreciar todos los documentos subidos a la plataforma por el docente, organizados por carpetas para un fácil acceso y reconocimiento.
- Lanzadores: espacio definido para publicar aplicaciones de uso libre y gratuito.
- Mi Perfil: aplicación que permite la especificación de información de cada estudiante, como los datos de inicio de sesión y la foto.
- Ayuda: guía de uso de la plataforma.

f) Ventajas de su aplicación en la enseñanza de la Ingeniería del Software:

- La plataforma está diseñada para que el estudiante tenga acceso al seguimiento de su proceso, gracias a que dispone de sus calificaciones y mensajes personalizados o grupales por parte del docente.
- El usuario solo requiere tener en su computador un navegador tan común como el Internet Explorer y acceso a Internet.
- Permite al docente administrar contenidos, planear, monitorear y diseñar el curso y, en general, el sitio de su asignatura: puede publicar y/o sugerir documentos de interés a sus alumnos de una forma rápida y segura, ya que solamente los usuarios registrados por él tendrán acceso a la información suministrada. Además, permite la integración de productos de audio y video, los cuales alimentan de forma positiva el proceso de enseñanza-aprendizaje. A su vez, los estudiantes tendrán acceso al material en cualquier momento, mientras estén activos, facilitando su proceso de estudio.
- Los estudiantes podrán comunicarse continuamente con el docente y sus demás compañeros de clase, dentro y fuera de la universidad, a través de los correos electrónicos, foros o chats que se habiliten.
- El docente recibirá las tareas asignadas en el tiempo determinado por él, a través de los correos con archivos adjuntos. Es fácil de usar y permite el monitoreo constante de las actividades de los estudiantes.
- Es un sitio gratuito, avalado y desarrollado por la Universidad Industrial de Santander, lo cual permite su uso sin restricciones.

5.1.2.2 AULA VIRTUAL – Escuela de Ingeniería de Sistemas e Informática UIS – Descripción

a) Ingreso: registro de usuarios en plataforma web. Administrado por el docente de la asignatura. Su ingreso se realiza a través de la dirección electrónica: http://cormoran.uis.edu.co/eisi/

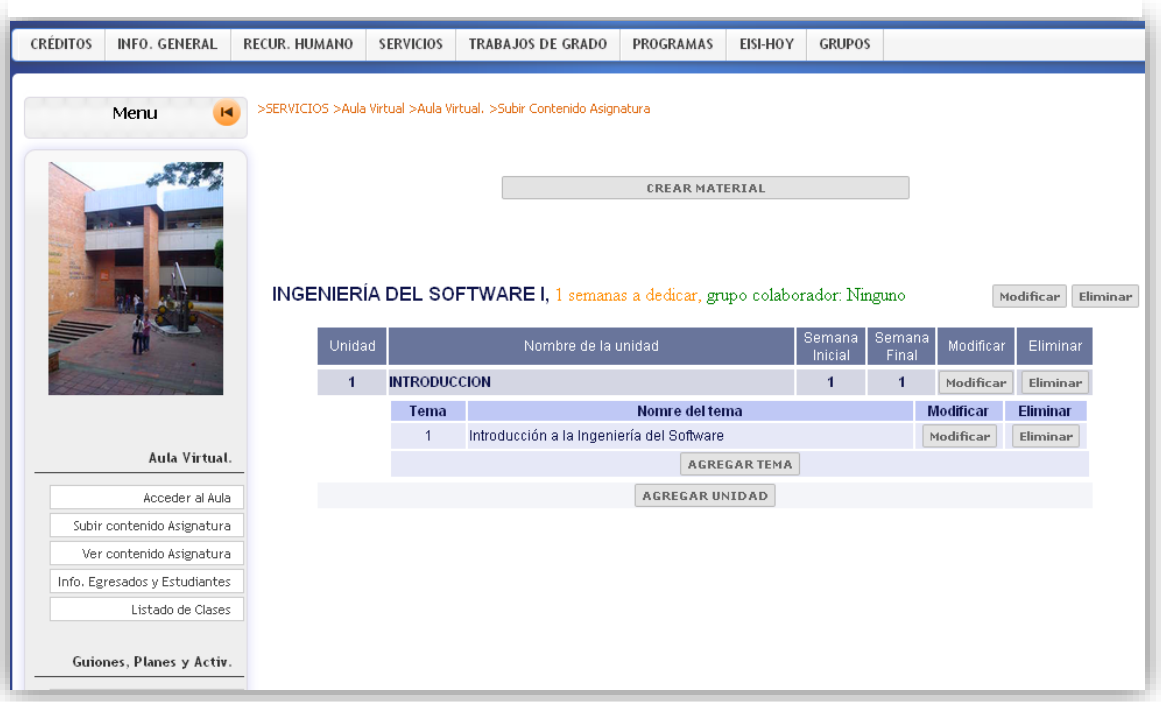
Pantalla de inicio: como lo indica la Figura 4, esta aplicación web también requiere un acceso a través de usuario y contraseña.

Figura 4. Pantalla de inicio de Aula Virtual EISI.

Fuente: Captura de pantalla de http://cormoran.uis.edu.co/eisi/

b) Pantalla de usuario: una vez se ha registrado el usuario, es posible acceder a las utilidades según el perfil del usuario. Ver Figura 5.

Figura 5. Pantalla de usuario Aula Virtual EISI.



Fuente: Captura de pantalla de http://cormoran.uis.edu.co/eisi/

c) Secciones para el docente:

- Acceder al aula.
- Subir contenido Asignatura: permite agregar, modificar y eliminar contenidos, al igual que adjuntar material de apoyo en diferentes formatos (pdf de máximo 2 MB, imágenes, videos, animaciones, simuladores, presentaciones y enlaces de interés).
- Ver contenido Asignatura: le permite al docente ver una vista previa del contenido que ha subido a la plataforma.
- Información de Egresados y Estudiantes: habilita información de contacto de los estudiantes activos y egresados del programa de Ingeniería de Sistemas.
- Listado de clases: presenta el listado de estudiantes matriculados en cada asignatura del programa de Ingeniería de Sistemas.

- Guiones de clase: le facilita al docente la ponderación de la calificación, pues permite asignar el peso a cada actividad sujeta a evaluación que planteará en su asignatura.
- Planes de actividades: diseñada para que el docente cree y modifique las actividades de evaluación que desea incluir en el desarrollo de la asignatura que lidera (exposiciones, foros, trabajos, laboratorios, talleres, exámenes). Puede dar enfoque según filtro que desee (por semana, por actividad o por pendientes), de modo que los estudiantes inscritos sabrán las actividades programadas en el tiempo estipulado. El docente tendrá la posibilidad de clasificar las actividades dentro de la gama preestablecida para este fin (exámenes, talleres, exposiciones, trabajos, quices, laboratorios, foros) y determinar el peso que considera a cada una para su calificación.
- Banco de preguntas: permite crear foros con preguntas para ser solucionadas por los visitantes.

d) Ventajas de su uso:

- Facilita la comunicación entre los miembros de la comunidad educativa de la Escuela, a través de la generación de chats, foros, correos electrónicos.
- El docente puede apoyarse para la publicación y ponderación de notas.
- El docente puede tener el listado de sus alumnos y tener incluso contacto con egresados.
- Permite la creación de un banco de preguntas que podrá ser alimentado por el docente y podrá ser activado o desactivado en cualquier momento.
- Es posible dar seguimiento a las actividades, pues el docente puede revisar quién se comunicó y publicó información en un foro, chat o correo. Además, puede poner límite de entrega de trabajos y tareas y revisarlos al final del tiempo determinado para esto.

5.2 HERRAMIENTAS DE APOYO PARA LA ENSEÑANZA DE LA INGENIERÍA DEL SOFTWARE

Aunque existen diversas herramientas para el apoyo de la enseñanza de la Ingeniería del Software, la presente investigación se centra en las que se refieren al modelado, porque en los procesos estudiados solo se abarca la fase de Requerimientos de Software con profundidad. No obstante, se abordará el tema de herramientas de apoyo para el desarrollo de aplicaciones.

5.2.1 Herramientas de software para la diagramación de modelos. Durante los inicios de la informática, cuando no existían estándares en términos de diseño de modelos, la comunicación entre los actores principales del proceso de desarrollo de software era muy complicada y esta dificultad se reflejaba en demoras en los tiempos de entrega por la tardanza en la unificación de la comprensión global del problema. Con el objetivo de resolver esta situación, surge el Lenguaje Unificado de Modelado (UML), el cual aporta diversas funcionalidades enfocadas a la etapa de diseño y análisis de Software, dentro de las cuales se encuentran principalmente [Hernández, 2002]:

- representar en forma gráfica un sistema que sea entendible fácilmente por otra persona.
- especificar las características de un sistema antes de su construcción.
- construir los sistemas diseñados a partir de los modelos especificados.
- documentar el sistema desarrollado a partir de los elementos gráficos representados en los modelos, los cuales servirán para posteriores revisiones.

Debido al surgimiento, difusión y estandarización del Lenguaje Unificado de Modelado UML, en los últimos años han venido desarrollándose infinidad de herramientas que permiten representar modelos de diseño de software sin mayor dificultad. Dentro del marco de la presente investigación, este tipo de herramientas son de gran utilidad, si tenemos en cuenta que el campo de aplicación hace

referencia precisamente a la enseñanza y, especialmente, al aprendizaje de la Ingeniería del Software en su fase de Requerimientos.

A continuación se analizará el uso, las utilidades que ofrece y la factibilidad de aplicación en la asignatura de Ingeniería del Software I que imparte la Universidad Industrial de Santander de tres reconocidas y potentes herramientas de modelado que existen en el mercado actualmente: ArgoUML, StarUML y Visio. El uso de este tipo de herramientas se debe a que es fundamental el desarrollo de modelos en la fase de definición de Requerimientos, y debido a que la enseñanza del área debe estar enfocada a la formación por competencias, centrada en el estudiante y basada en el proceso, se hace esencial su aplicación.

5.2.1.1 ArgoUML. Herramienta software para diagramación UML, escrita en Java y con Licencia BSD (Berkeley Software Distribution – Licencia de Software libre permisiva). [ArgoUML]

Fue pensado como herramienta para apoyar el análisis y diseño de sistemas de software orientados a objetos. ArgoUML provee interfaces gráficas de usuario automáticas que reducen el trabajo manual al ingresar un diseño y transformarlo a código. Aunque existen diversas aplicaciones para este fin, se destaca, entre otras características, por [ArgoUML, 2011]:

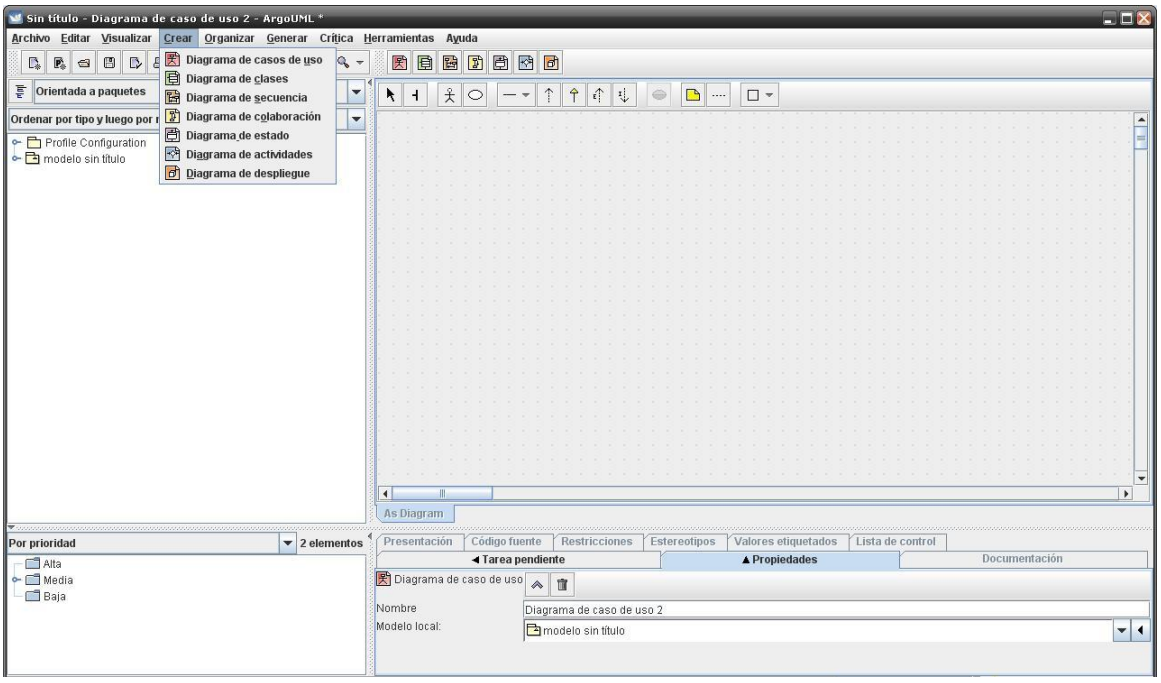
- Enfocarse en proporcionar características para incrementar la productividad, a partir de las necesidades cognitivas de los diseñadores y arquitectos de software.
- Soportar estándares abiertos (UML, XML, SVG, OCL, entre otros).
- Ser una aplicación Java pura, permitiendo funcionar en todas las plataformas donde Java esté disponible.
- Ser un proyecto de código abierto.
- Ser de licencia gratuita.

ArgoUML es una herramienta de gran valor, ya que combina dos aplicaciones de gran importancia: UML y Java. UML es el lenguaje de modelado orientado a objetos más prevalente y Java es una de las plataformas de desarrollo orientado a objetos más productivas. [ArgoUML, 2011]

A continuación se explicará brevemente la funcionalidad de ArgoUML.

En la Figura 6 se aprecia una vista general de la interfaz de la aplicación.

Figura 6. Vista general de la interfaz de ArgoUML.



Fuente: Captura de pantalla ArgoUML

Dentro de sus principales características y funciones se encuentran:

- Diseñar modelos de casos de uso (actores, casos de uso, relaciones). Estos modelos están incluidos como esenciales dentro del SWEBOK.

- Diseñar diagrama de clases, de secuencia, de colaboración, de estado, de actividades y de despliegue.
- Generar código fuente de las clases elaboradas en un lenguaje de programación seleccionado por el usuario (Java, PHP, SQL, entre otros).

Los diseños mencionados anteriormente tienen una interfaz gráfica de fácil reconocimiento, con todos los links necesarios para agregar cada componente. Al final se obtienen diseños con apariencia profesional, con los estándares ya predefinidos y con la posibilidad de editar cada atributo del modelo.

Para ilustrar y analizar el proceso que siguen los estudiantes en la utilización de esta y otras herramientas de modelado durante su proceso de aprendizaje de la asignatura de Ingeniería del Software I, en el desarrollo del presente documento se dará seguimiento a un ejercicio práctico, donde se dibujarán los diagramas de modelado principales para el caso de una aplicación online que permite consultas, préstamo y administración de libros de una biblioteca en una institución educativa.

El Sistema de Biblioteca online es un sistema que permite a los usuarios comunes (estudiantes y docentes) hacer consultas y reservas de libros. Para ingresar al sistema deberán autenticarse si desean consultar información específica de un libro, tal como su tabla de contenido o su estado (libre/reservado/prestado) o si desean ejecutar una reserva; sin embargo, tendrán ingreso libre para realizar consultas generales (ver catálogo de libros existentes). Por otra parte, el administrador del sistema ingresará en su puesto de trabajo mediante una intranet, con previa autenticación, para alimentar el inventario del material bibliográfico (ingresar y eliminar libros). Así mismo, el bibliotecario, después de su autenticación desde la intranet para ingresar al sistema, podrá realizar préstamos correspondientes a una reserva previa y registrar devoluciones; si es el caso, también podrá efectuar una sanción por demora, pérdida o daño de algún ejemplar de la biblioteca.

- **Representación del modelo de casos de uso:**

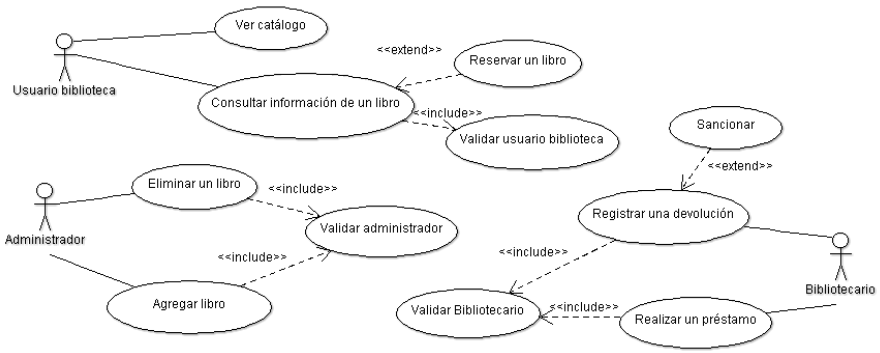
ArgoUML permite la creación de los principales diagramas de modelado que están planteados por UML. Para la diagramación del modelo de casos de uso, esta herramienta activa los elementos necesarios para esta actividad (actores, casos de uso y relaciones), como se muestra en la Figura 7:

Figura 7. Barra de herramientas para la creación de un modelo de Casos de Uso en ArgoUML.

Fuente: Captura de pantalla ArgoUML

Para el caso de aplicación que estamos implementando no es importante definir en detalle el diseño del modelo, sino discutir las características que tiene la herramienta para facilitar su desarrollo; por tanto en la Figura 8 se presenta el modelo ya definido:

Figura 8. Diagrama de Casos de Uso Sistema Biblioteca - ArgoUML.

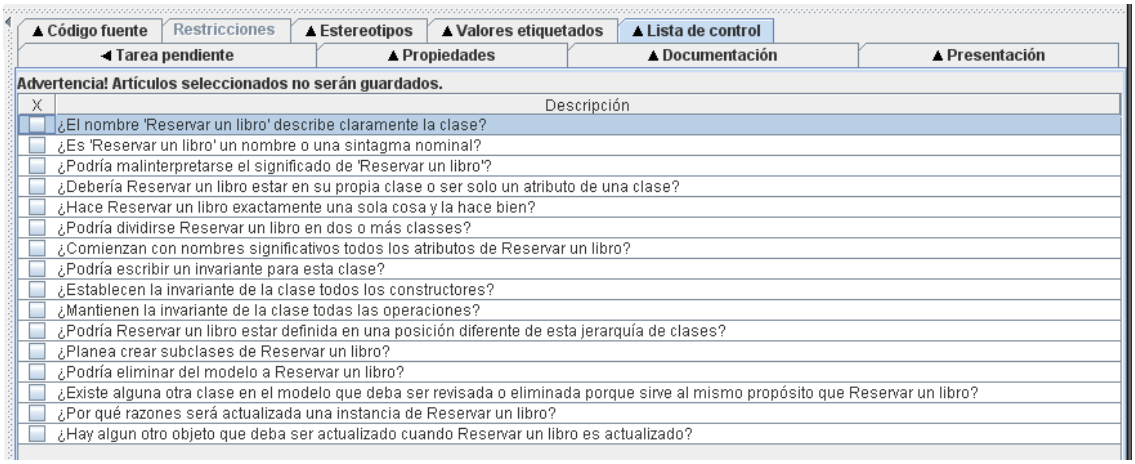


Fuente: Autor del proyecto.

El usuario de ArgoUML tiene la capacidad de realizar el diagrama intuitivamente, gracias a la simbología estándar que posee la herramienta, facilitando su utilización. Los elementos se pueden agregar, editar o remover sin dificultad. El

resultado final, se puede exportar en un formato de imagen para poder documentarlo (Admite formato de imágenes png, gif, svg, eps). Finalmente, se destaca una funcionalidad que tiene ArgoUML y que se denomina lista de control, que tiene como fin guiar al usuario en la toma de decisiones acerca de la efectividad de los elementos que define. En la Figura 9 se muestran las recomendaciones de control sobre el caso de uso “Reservar un libro”:

Figura 9. Recomendaciones en Lista de Control para Diagrama de Casos de Uso - ArgoUML.



Fuente: Captura de pantalla ArgoUML

- **Diagrama de Clases:**

Al igual que en el caso anterior, esta funcionalidad agrega una barra de herramientas específica para este caso (paquetes, clases, asociaciones), como se aprecia en la Figura 10:

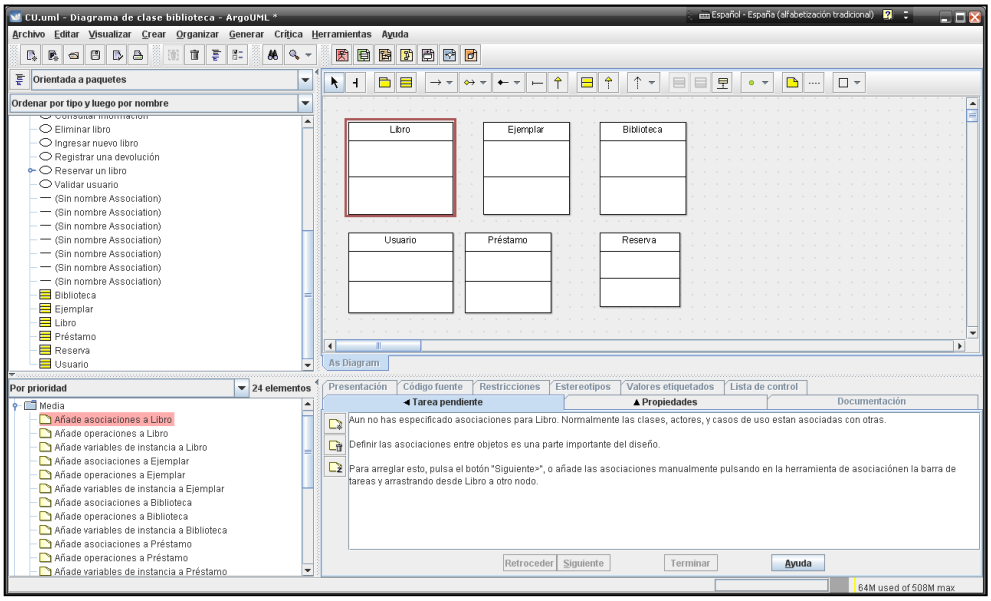
Figura 10. Barra de herramientas Diagrama de Clases - ArgoUML

Fuente: Captura de pantalla ArgoUML

En el ejercicio de aplicación se destacan 6 clases principales: Libro, Usuario, Ejemplar, Biblioteca, Reservación y Préstamo.

A medida que se construyen las clases, el sistema va mostrando comentarios sobre buenas prácticas o estándares que deben ser seguidos para que no existan inconsistencias en el diagrama, tal como se muestra en la Figura 11:

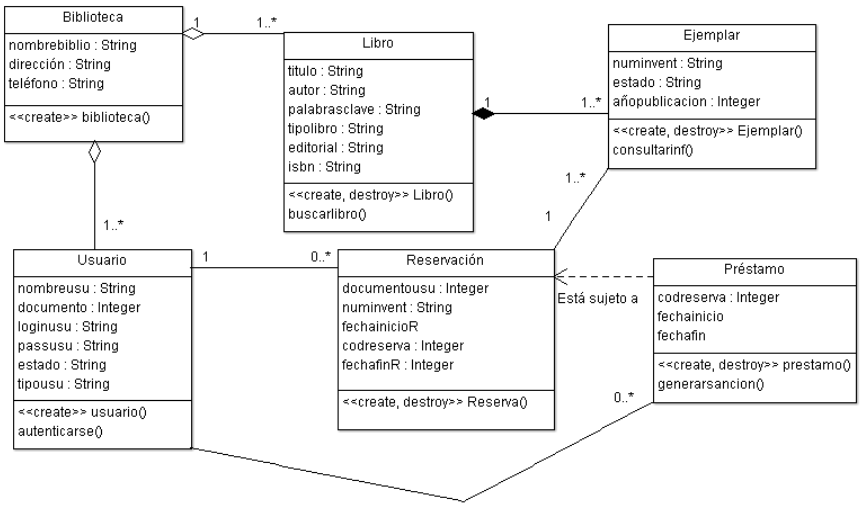
Figura 11. Comentarios producidos por ArgoUML sobre inconsistencias en diagrama de clases.



Fuente: Captura de pantalla ArgoUML

En la Figura 12 se presenta el diagrama de clases terminado, donde se incluyen los atributos y las operaciones de cada clase, así como sus asociaciones:

Figura 12. Diagrama de clases Sistema de Biblioteca - ArgoUML.

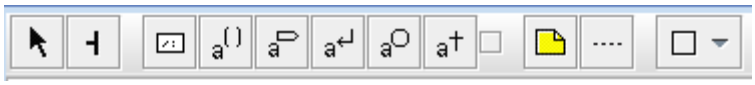


Fuente: Captura de pantalla ArgoUML

- **Diagrama de Secuencia:**

En este tipo de diagrama se especifica el procedimiento seguido por las acciones de cada caso de uso definido. ArgoUML presenta una barra de herramientas donde se encuentran los elementos necesarios para la elaboración de este diagrama (Figura 13).

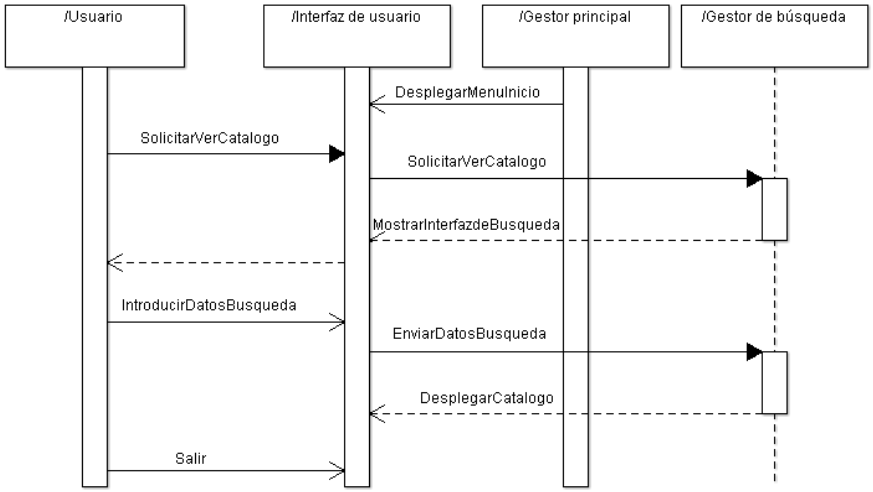
Figura 13. Barra de herramientas para elaborar diagrama de secuencia - ArgoUML.



Fuente: Captura de pantalla ArgoUML

- Diagrama de secuencia del caso Ver catálogo (Figura 14):

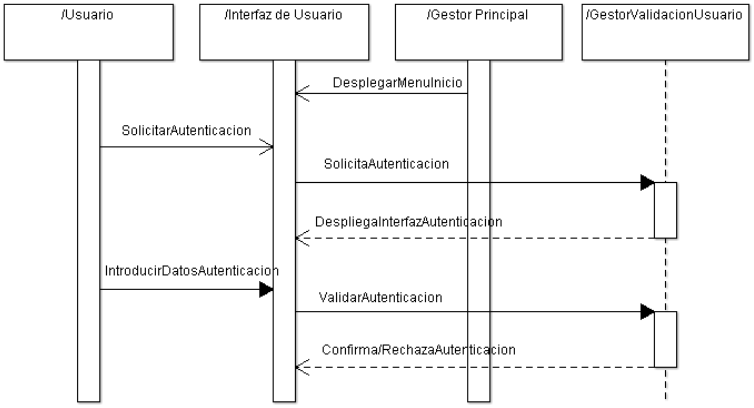
Figura 14. Diagrama de secuencia del caso Ver Catálogo - ArgoUML.



Fuente: Autor del proyecto.

- Diagrama de secuencia del caso Validar Usuario (Figura 15):

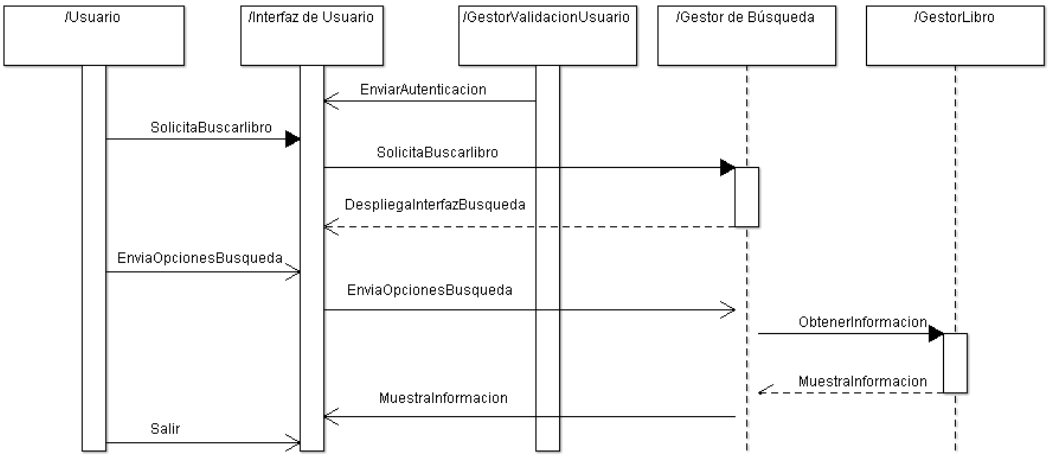
Figura 15. Diagrama de secuencia del caso de uso Validar Usuario - ArgoUML.



Fuente: Autor del proyecto.

- Diagrama de secuencia del caso Consultar Información (Figura 16)

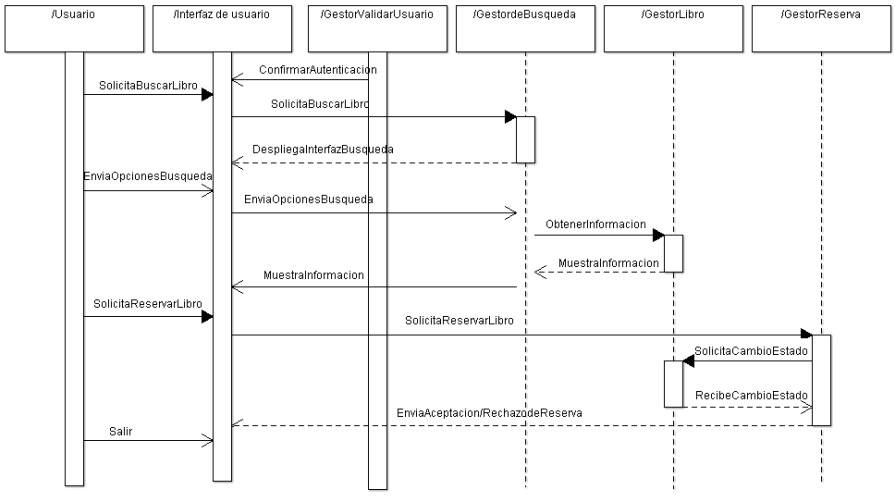
Figura 16. Diagrama de secuencia del caso de uso Consultar Información - ArgoUML.



Fuente: Autor del proyecto.

- Diagrama de secuencia del caso Reservar un libro (Figura 17)

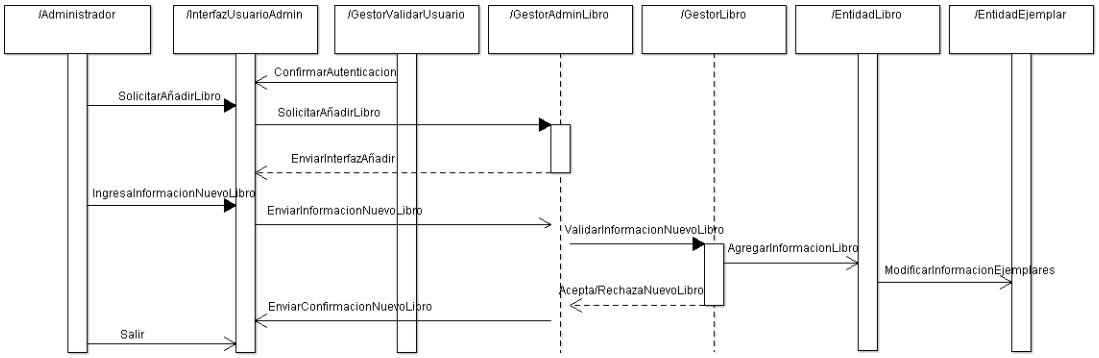
Figura 17. Diagrama de secuencia caso de uso Reservar un libro - ArgoUML.



Fuente: Autor del proyecto.

- Diagrama de secuencia del caso Agregar un Libro (Figura 18)

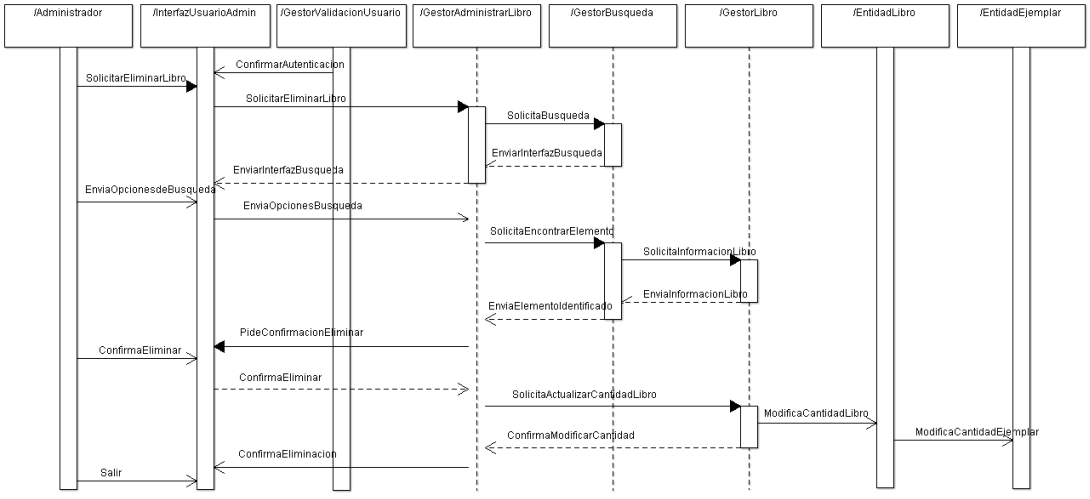
Figura 18. Diagrama de secuencia del caso de uso Agregar un libro - ArgoUML.



Fuente: Autor del proyecto.

- Diagrama de secuencia del caso Eliminar un Libro (Figura 19)

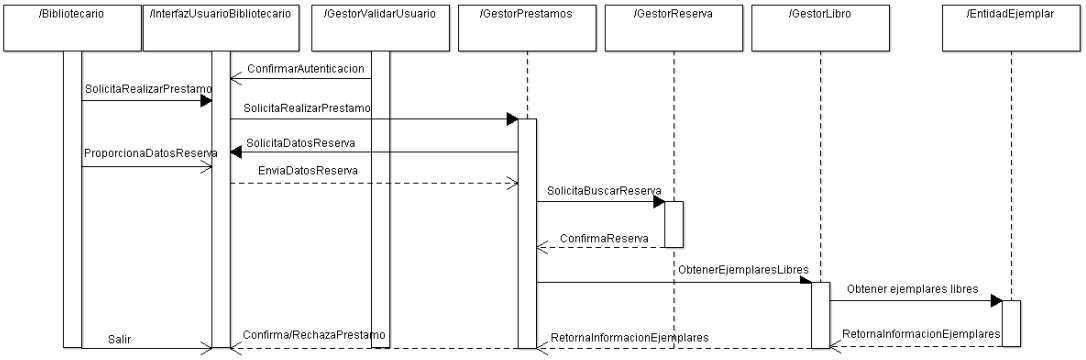
Figura 19. Diagrama de secuencia del caso de uso Eliminar un libro - ArgoUML.



Fuente: Autor del proyecto.

- Diagrama de secuencia del caso Realizar un Préstamo de un libro (Figura 20)

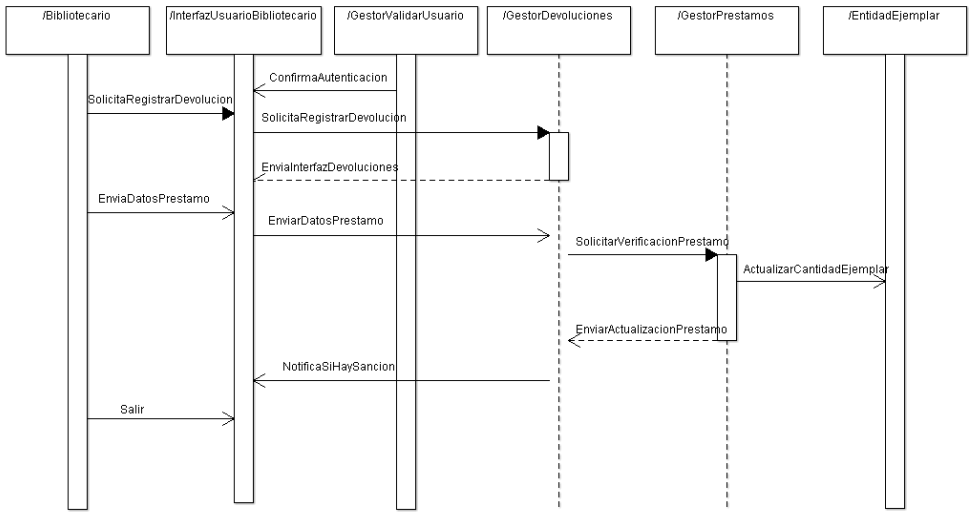
Figura 20. Diagrama de secuencia del caso de uso Realizar un Préstamo - ArgoUML.



Fuente: Autor del proyecto.

- Diagrama de secuencia del caso Registrar una Devolución de un libro (Figura 21)

Figura 21. Diagrama de secuencia del caso de uso Registrar una Devolución - ArgoUML.



Fuente: Autor del proyecto.

- **Diagrama de actividades:**

La barra de herramientas para esta fase del diseño desarrollada por ArgoUML se muestra en la Figura 22:

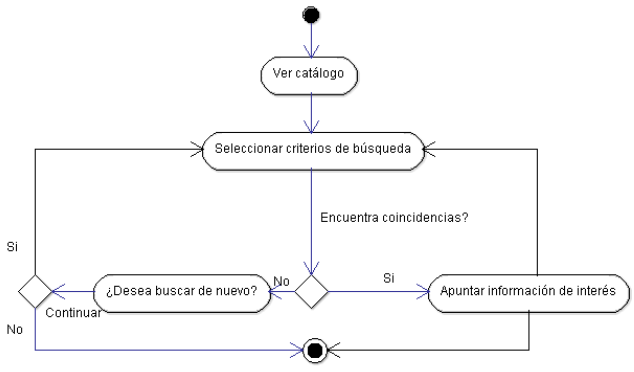
Figura 22. Barra de herramientas Diagrama de Actividades - ArgoUML.

Fuente: Captura de pantalla ArgoUML

Para continuar con el ejercicio de aplicar un modelo de diseño práctico, se mostrarán los diagramas de Actividades para un Sistema de Biblioteca en línea, realizados mediante la herramienta ArgoUML. En esta etapa, se aprecia cierta dificultad para entender los mensajes de apoyo o recomendaciones, tales como el uso de leyendas de transición, debido a la falta de ayuda de la aplicación, sin embargo, los elementos para la construcción de los modelos son intuitivos y fáciles de manejar.

- ✓ Diagrama de Actividades del caso Ver Catálogo (Figura 23)

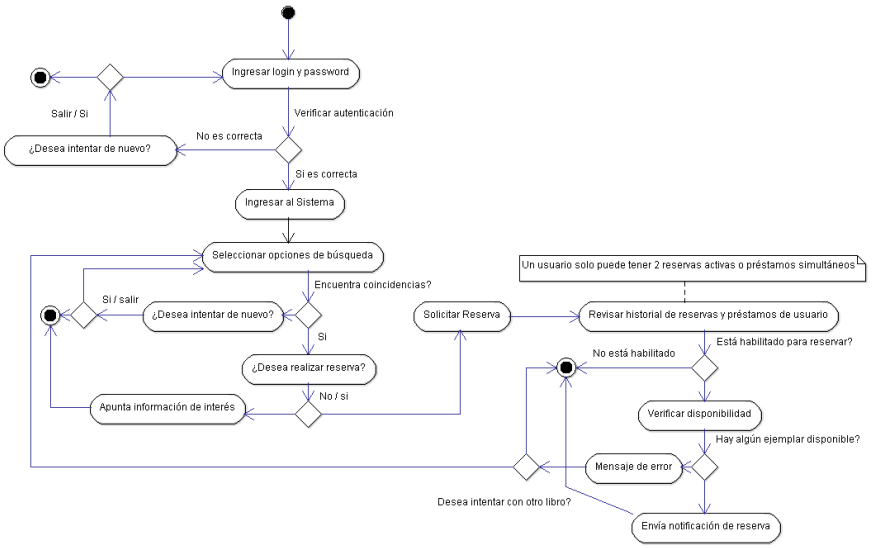
Figura 23. Diagrama de Actividades del caso Ver Catálogo - ArgoUML.



Fuente: Autor del proyecto.

✓ Diagrama de Actividades del caso Reservar un libro (Figura 24)

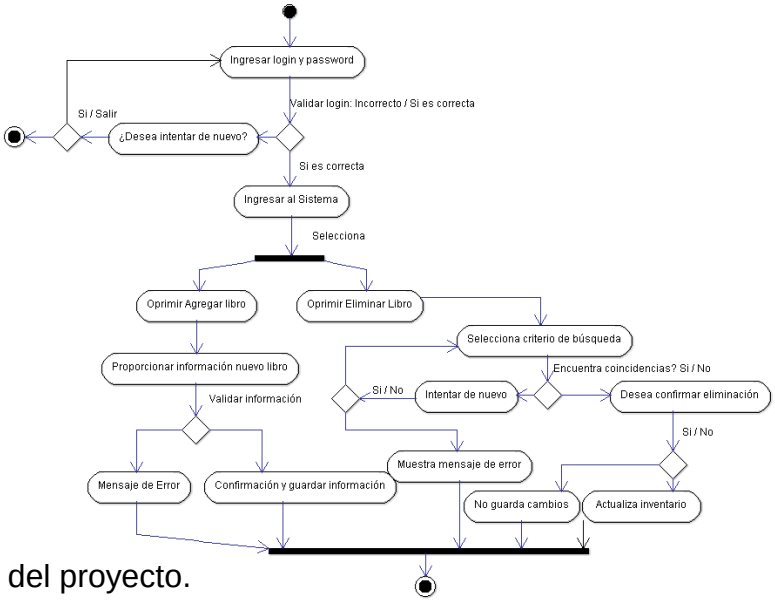
Figura 24. Diagrama de actividades del caso Reservar un libro - ArgoUML.



Fuente: Autor del proyecto.

✓ Diagrama de Actividades de los casos correspondientes a la Administración del Inventario de la Biblioteca (Figura 25)

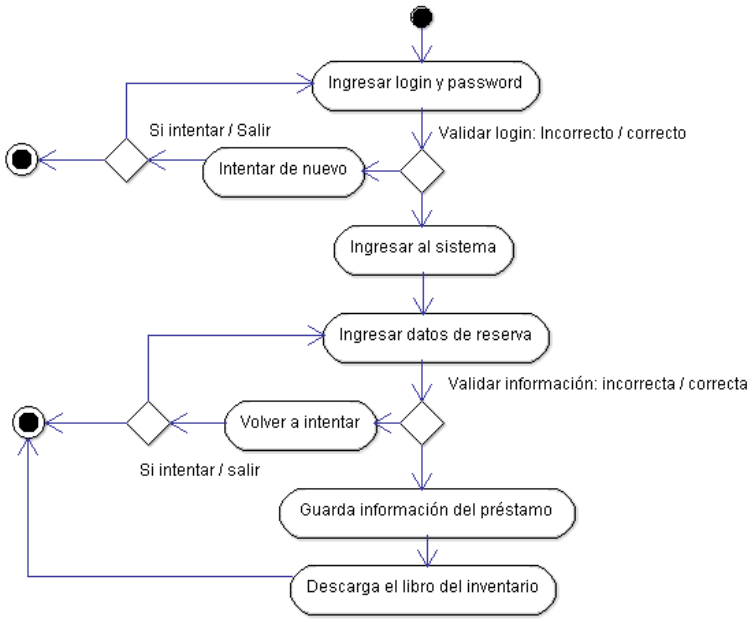
Figura 25. Diagrama de actividades de Administrar el inventario - ArgoUML.



Fuente: Autor del proyecto.

✓ Diagrama de Actividades del caso Realizar un Préstamo (Figura 26)

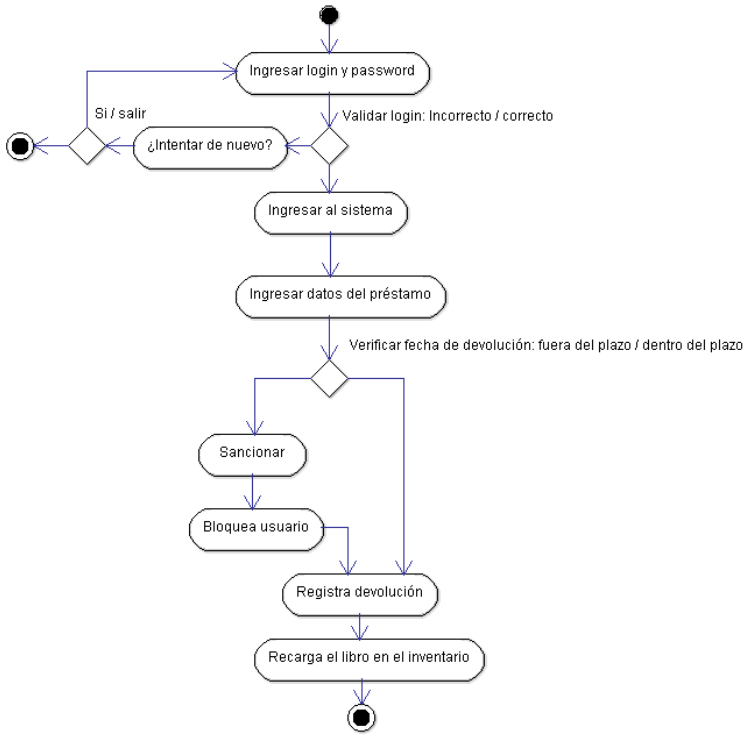
Figura 26. Diagrama de actividades del caso Realizar un Préstamo - ArgoUML.



Fuente: Autor del proyecto.

✓ Diagrama de Actividades del caso Registrar una Devolución (Figura 27)

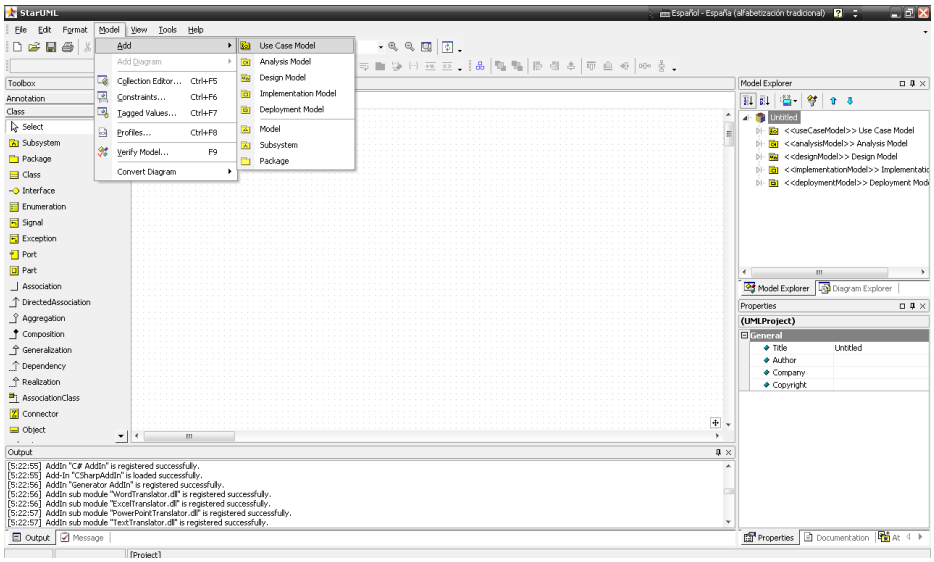
Figura 27. Diagrama de Actividades del caso Registrar una Devolución - ArgoUML.



Fuente: Autor del proyecto.

5.2.1.2 StarUML. Es una aplicación de código abierto para el modelado de software que soporta UML (Unified Modeling Language). Está licenciado bajo una versión modificada de GNU GLP (General Public Licence). En la figura 28 se aprecia una vista general del entorno.

Figura 28. Vista del entorno general de StarUML.



Fuente: Captura de pantalla de StarUML.

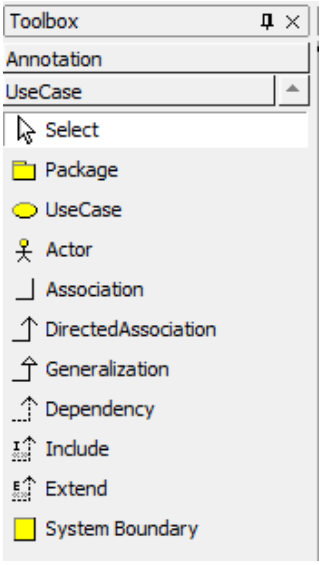
- Soporta 11 diferentes diagramas estándares de UML (Diagrama de Casos de Uso, de Clases, de Secuencia, de Colaboración, de Estados, de Actividad, de Componentes, de Despliegue y de Estructura compuesta).
- Soporta lenguajes de programación (Java, C++, Visual Basic, Delphi, VB.NET, etc)
- Compatible con documentos de Microsoft Office (Word, Excel, PowerPoint).
- Se puede integrar con herramientas específicas (Visual SourceSafe, CVS, Eclipse, Visual Studio.Net).
- Provee fácil adaptación al entorno, facilitando al usuario su uso y aplicación en proyectos de software.
- Permite la verificación automática de los modelos desarrollados, facilitando descubrir de forma temprana los errores y mejorando la completitud del producto final.

Estas características hacen de esta aplicación una herramienta muy completa y de uso intuitivo, que puede ser de gran utilidad en el campo de acción de la

asignatura Ingeniería del Software I, donde el enfoque primordial está inclinado hacia la búsqueda de requerimientos y su diseño. Los estudiantes pueden familiarizarse con los componentes y los diagramas a medida que los vayan diseñando y elaborando.

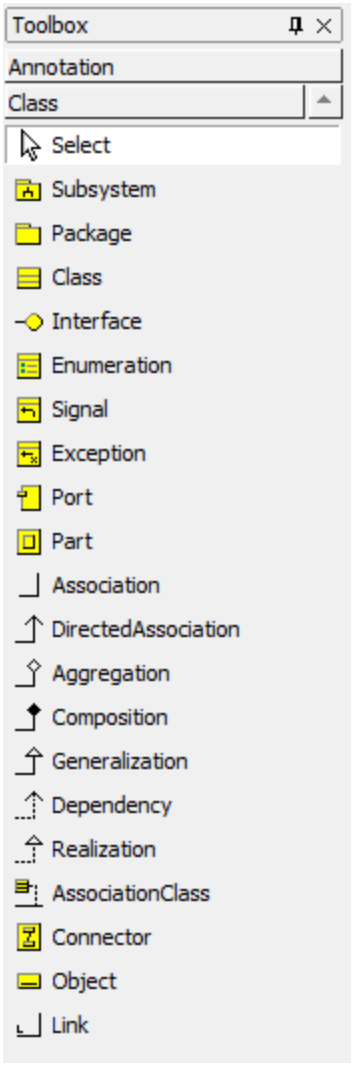
Al igual que ArgoUML, esta aplicación presenta barras de herramientas intuitivas y con la presentación estándar de los elementos de modelado. Para ilustrar estos componentes, se encuentran las Figuras 29, 30, 31 y 32. Presentan una ventaja respecto a ArgoUML y es la presencia de etiquetas permanentes que referencian cada elemento y no con mensajes emergentes.

Figura 29. Barra de herramientas Diagrama de Casos de Uso - StarUML.



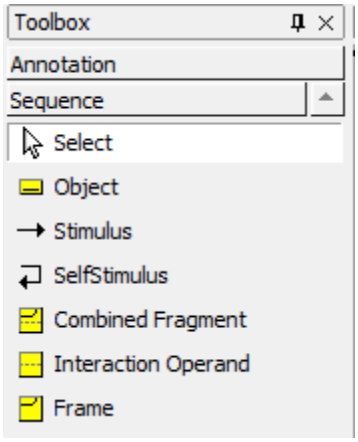
Fuente: Captura de pantalla de StarUML.

Figura 30. Barra de herramientas Diagrama de Clases - StarUML.



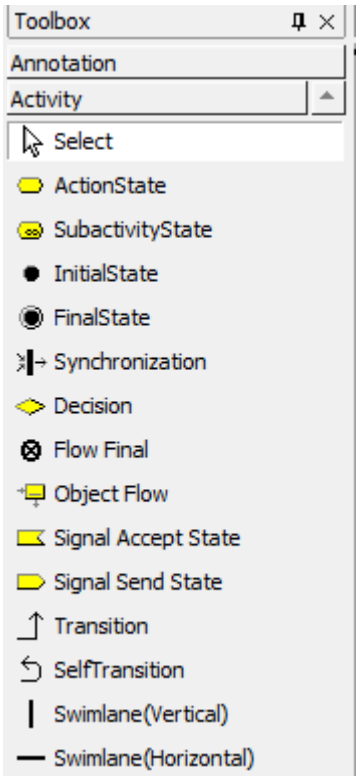
Fuente: Captura de pantalla de StarUML.

Figura 31. Barra de herramientas Diagrama de Secuencia - StarUML.



Fuente: Captura de pantalla de StarUML.

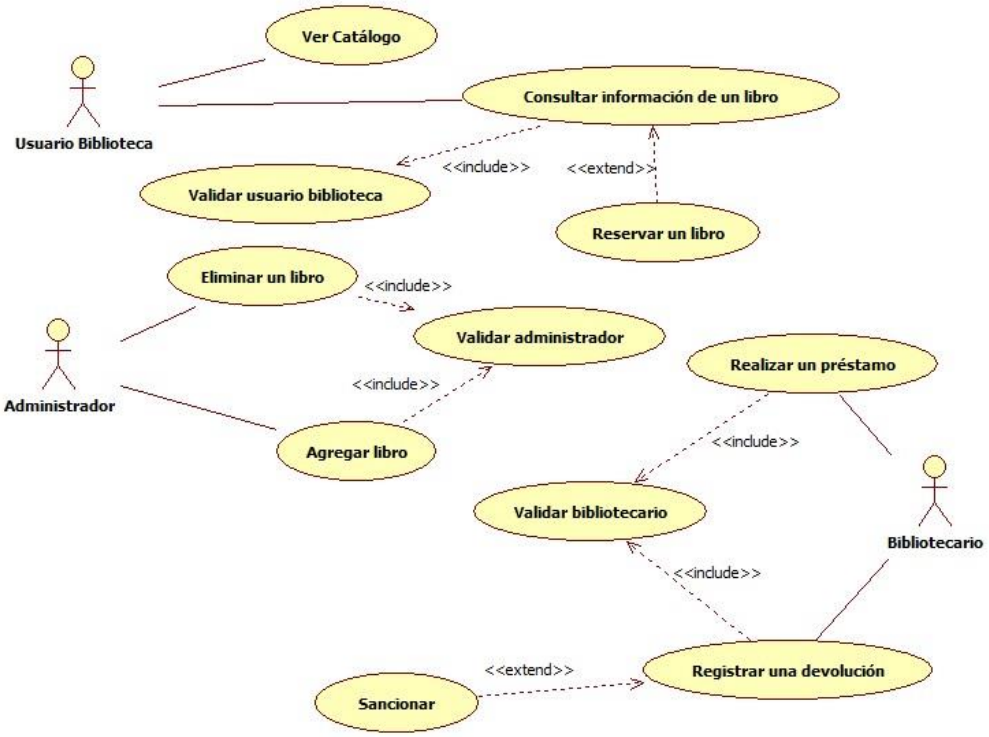
Figura 32. Barra de herramientas Diagrama de Actividades - StarUML.



Fuente: Captura de pantalla de StarUML.

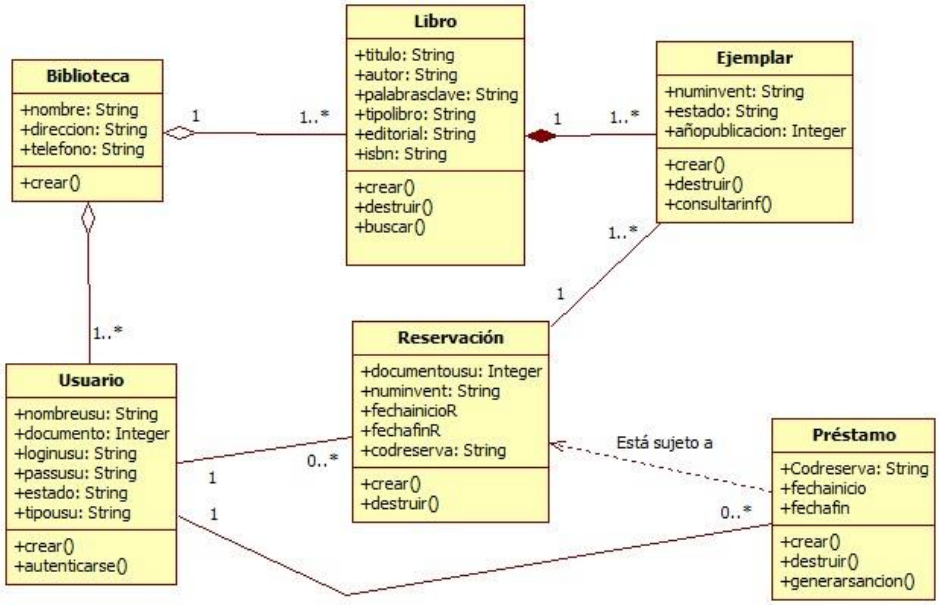
Con el propósito de apreciar el resultado final que se obtiene en los diagramas de diseño utilizando la herramienta de StarUML, en las Figuras 33, 34, 35 y 36 se presentan algunos de los principales modelos del ejercicio práctico.

Figura 33. Diagrama de casos de uso Sistema Biblioteca en Línea - StarUML.



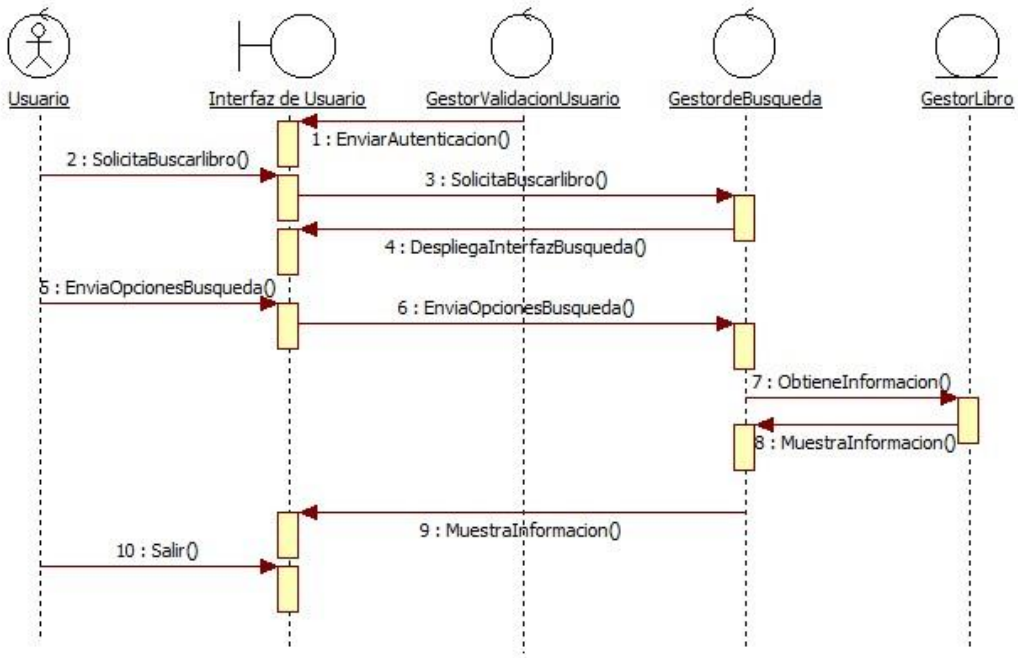
Fuente: Autor del proyecto.

Figura 34. Diagrama de Clases Sistema Biblioteca en Línea - StarUML.



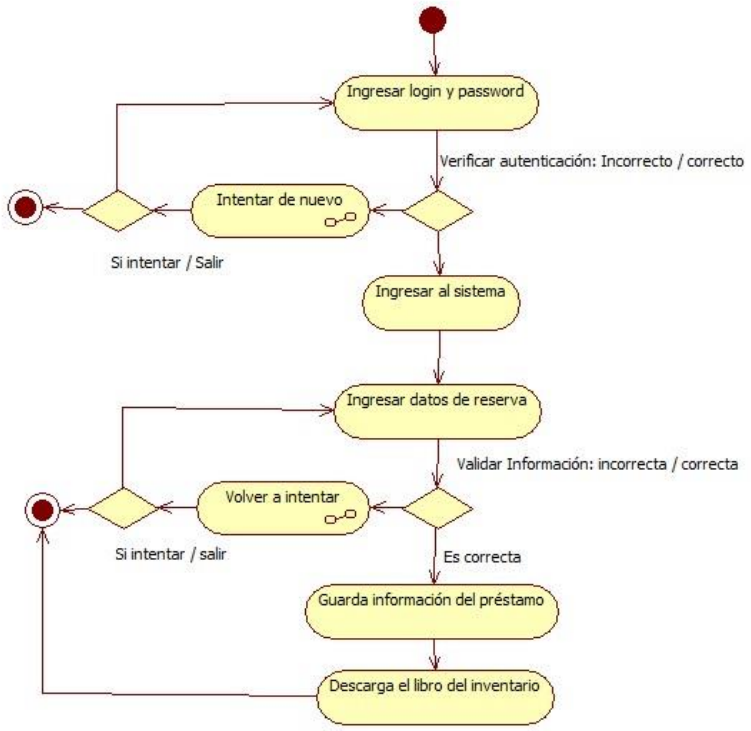
Fuente: Autor del proyecto.

Figura 35. Diagrama de Secuencia Caso Consultar Información - StarUML.



Fuente: Autor del proyecto.

Figura 36. Diagrama de Actividades caso de uso Realizar un Préstamo - StarUML.

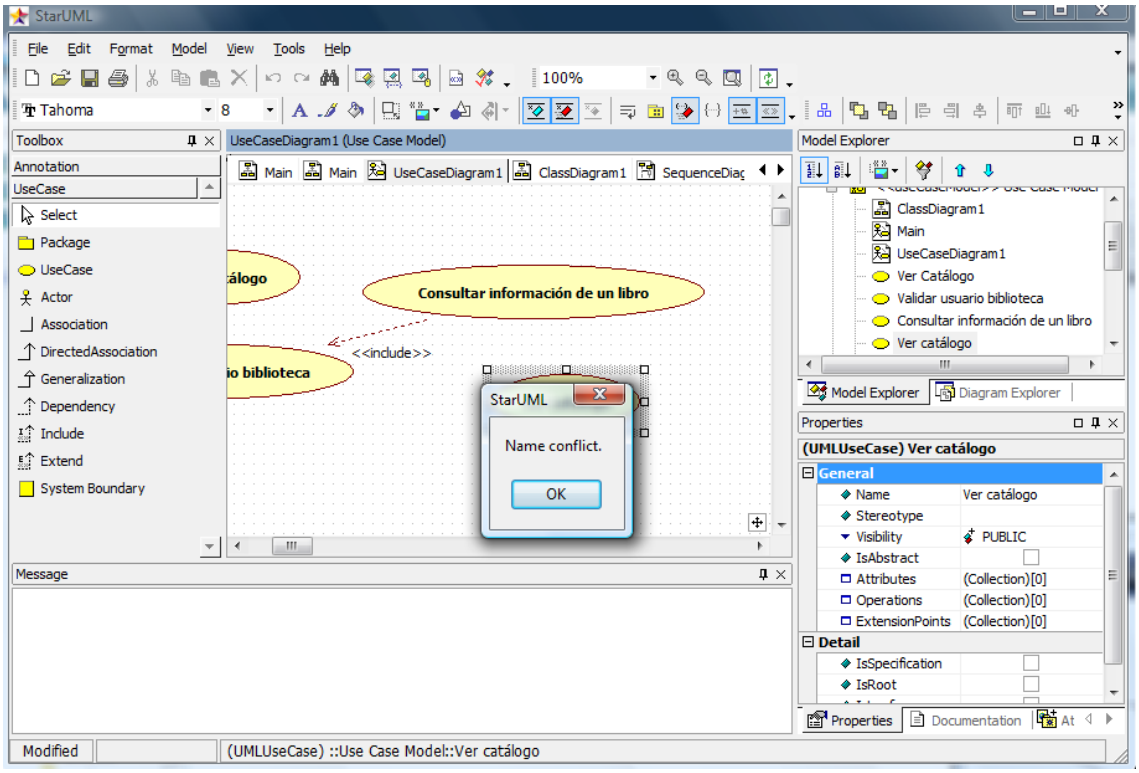


Fuente: Autor del proyecto.

De las figuras anteriores se detalla una buena presentación en términos de claridad, distribución y estandarización. El manejo de la herramienta es práctico, intuitivo, completo, agradable al usuario y organizado.

StarUML también presenta un manejo de errores, que se muestran en forma de mensaje emergente, y ubica al usuario en el lugar del error, lo cual facilita su corrección de forma temprana y precisa, como es apreciado en la Figura 37.

Figura 37. Mensajes de error - StarUML.



Fuente: Captura de pantalla StarUML.

5.2.1.3 VISIO – Microsoft. Producto software que incluye herramientas de diagramación avanzada para simplificar la complejidad, a través de plantillas que lucen profesionales y modernas con figuras predibujadas de uso intuitivo. Permite utilizar fuentes de datos tan populares como Microsoft Excel, Microsoft Access, Microsoft SQL Server, entre otros, para refrescar los diagramas con efectos visuales como íconos, símbolos, colores y gráficas de barras. Además, hace posible exportar los diagramas en formato de imagen para poder compartirlo en la web, sin necesidad que los demás tengan Visio. [Visio].

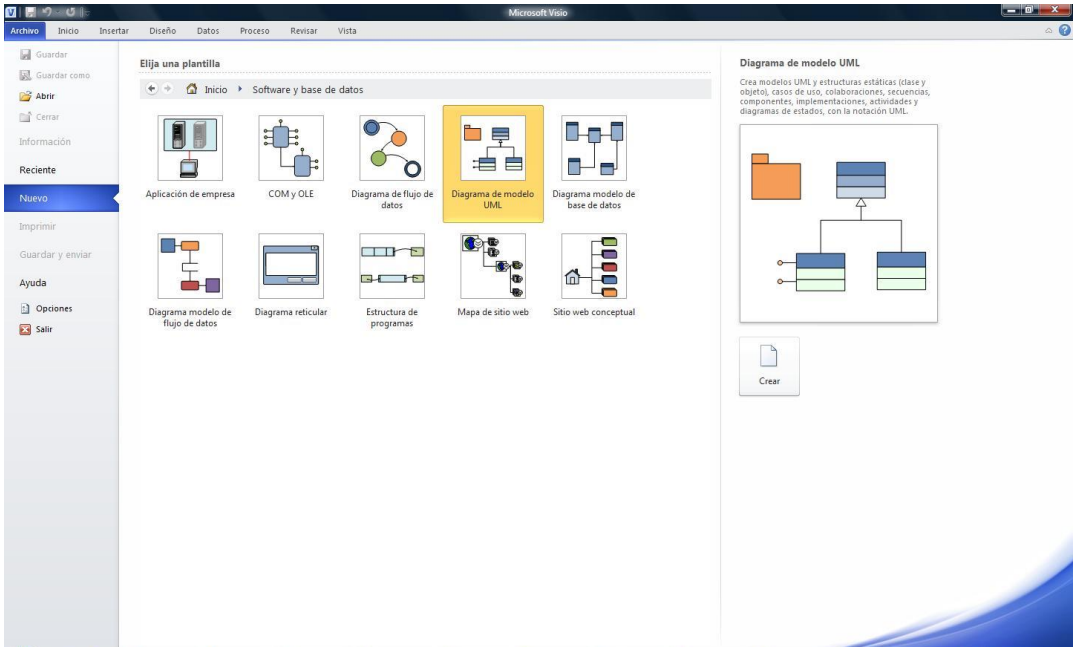
Un punto desfavorable para el interés de la investigación es que requiere licencia, al ser un producto incluido en el paquete de Microsoft Office Profesional, por tanto se ilustra su funcionamiento a manera informativa, para apreciar otras herramientas de apoyo, que en determinado momento podrían ser de utilidad.

Las principales características de Visio 2010 son:

- Visio 2010 ofrece gran variedad de opciones para desarrollar sus diagramas necesarios para negocios, administración de procesos y más, con figuras predibujadas, plantillas inteligentes y modernas, y dibujos de ejemplo.
- Cada paso en la creación de diagramas es más intuitivo, con agrupaciones lógicas de funciones en pestañas, por lo cual facilita su aplicación y libera a los estudiantes del estrés que produce el aprendizaje “a la carrera” de una herramienta software extra durante su actividad escolar.
- Permite dibujar diagramas de forma más rápida con funciones automáticas mejoradas. El estudiante no tendría que dedicar tiempo extra a la parte estética de sus diagramas, sino al diseño que es lo básicamente importante.
- Simplifica la elaboración de grandes y complejos diagramas, ayudando a mantener la información más organizada y comprensible.
- Ayuda a diseñar diagramas más atractivos con un amplio rango de herramientas de formato.
- Facilita la conexión con fuentes de datos tales como Excel o SQL Server, permitiendo desplegar los datos en tiempo real con diagramas definidos por el usuario.
- Los usuarios en línea pueden ver los diagramas con la información en tiempo real, en una alta definición, incluso si no tienen Visio.
- Asegura la consistencia y precisión con validación de diagramas, chequeando errores comunes, para que el usuario esté seguro que esté construido lógica y apropiadamente.
- Modela y monitorea los flujos de trabajo.

En la Figura 38 se presenta una vista general del entorno de Visio 2010.

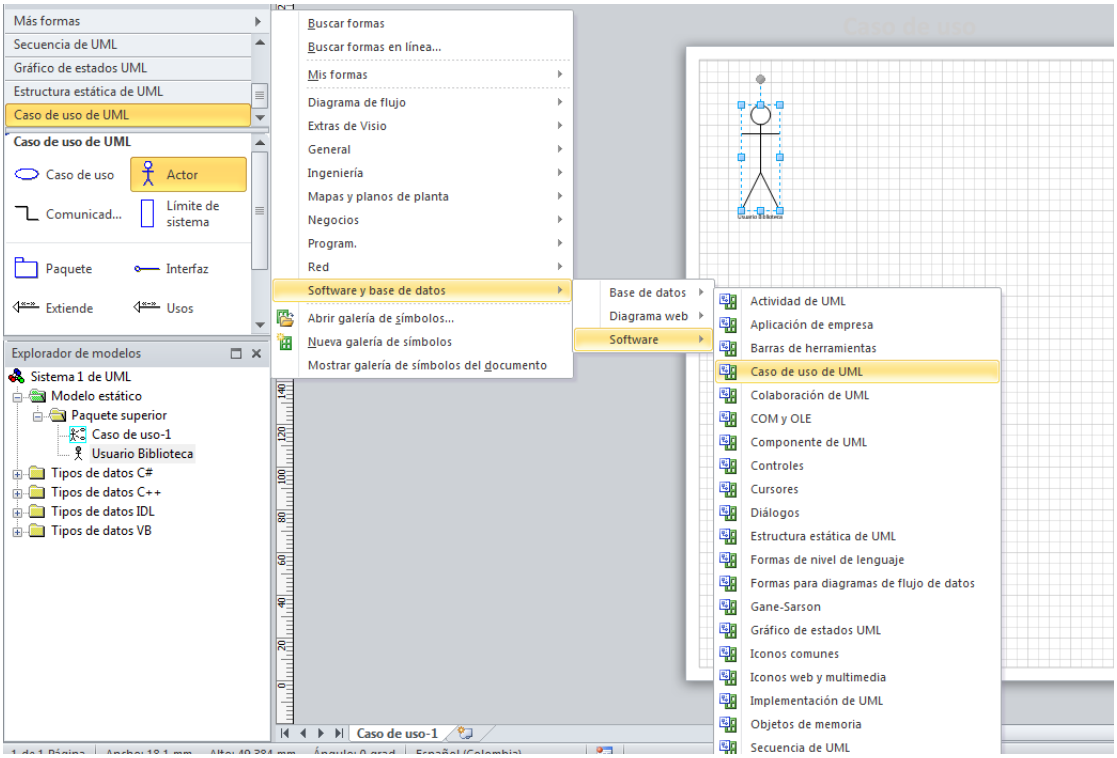
Figura 38. Vista de funciones de Visio.



Fuente: Captura de pantalla Microsoft Visio.

Una vez se define que se desea realizar un diagrama de modelado UML, Visio 2010 permite seleccionar el tipo de diagrama que se desea desarrollar (Figura 39).

Figura 39. Menú de diagramas y herramientas Visio 2010.

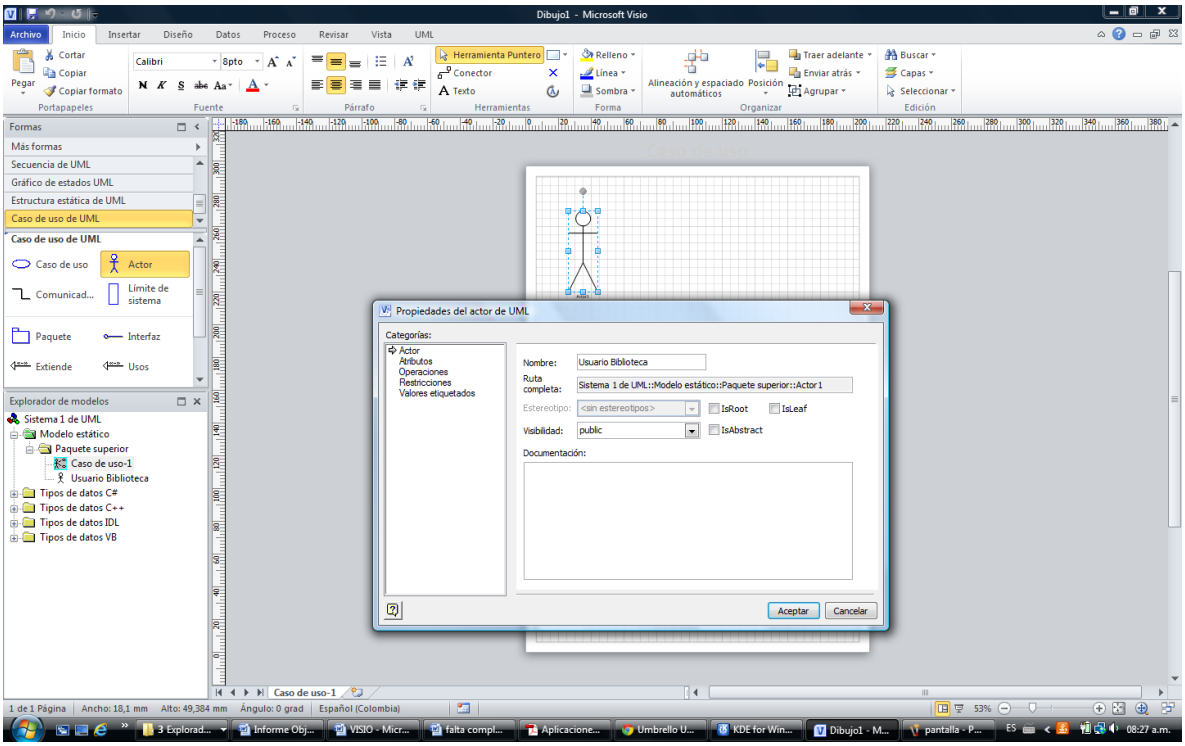


Fuente: Captura de pantalla Microsoft Visio.

Una desventaja clara de Visio es que al no ser especializado en diagramas UML, no presenta compilación de errores del estándar, permitiendo equivocaciones tan sencillas como la duplicación de elementos. Otra desventaja clara es que los elementos se encuentran agrupados de forma muy general y en ocasiones puede resultar difícil ubicarlos.

En la Figura 40 encontramos un ejemplo de cómo se muestran los elementos distribuidos en Visio 2010 y también cómo se determinan las propiedades de uno de ellos, en este ejemplo de un Actor.

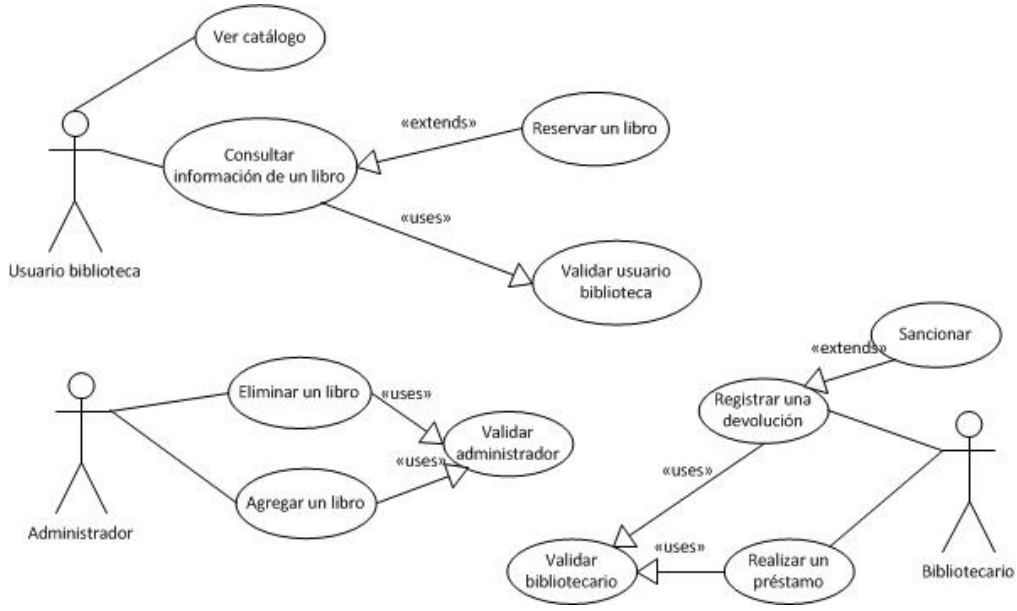
Figura 40. Organización de elementos UML en Visio 2010.



Fuente: Captura de pantalla Microsoft Visio.

En la figura 41 se aprecia el diagrama de casos de uso final que resulta utilizando esta herramienta; se aprecia que la parte visual es muy básica y puede servir para su propósito de documentar.

Figura 41. Diagrama de Casos de Uso del Sistema Biblioteca en Línea - Visio 2010.



Fuente: Autor del proyecto.

Como conclusión, Visio es una herramienta de utilidad para realizar diagramas UML en la documentación de un proyecto, pero no para validar consistencias ni para elaborar modelos integrales de una aplicación de Software. Además, como ya se mencionó, al ser una aplicación que no posee licencia libre, se dificulta tenerla en la lista de elegibles como apoyo en el proceso de aprendizaje de la asignatura de Ingeniería del Software I que se imparte en la Universidad Industrial de Santander.

5.2.2 Comparativo de herramientas de modelado UML. Habiendo ya descrito y presentado las principales características de algunas de las principales y más comunes herramientas de modelado UML, a continuación se evidencia un cuadro comparativo para poder determinar cuál de ellas sería la opción más acertada para apoyar el aprendizaje de la asignatura Ingeniería del Software I.

Los parámetros principales del comparativo se basaron en un estudio realizado por la Universidad del País Vasco, donde se analizaron diferentes herramientas educativas [González, 2009].

En todos los parámetros se usará un valor que oscila entre 0 y 10, donde 10 es la mejor puntuación, que corresponde al mejor funcionamiento para el ítem correspondiente. [González, 2009], [López, 2012]

- **Funcionalidad:** cantidad de características que posee la aplicación, comparada con otras de su misma categoría.
- **Estabilidad:** hace referencia a la ausencia de fallos de programación en la aplicación, es decir, que el software puede ejecutarse por mucho tiempo sin presentar errores o problemas y funciona como se espera, enfocándose en obtener resultados de calidad.
- **Licenciamiento:** se valora la licencia con la cual se distribuye la aplicación. Se valora con la mayor puntuación a aquellas que siguen los principios de Software Libre o que permiten mayor libertad de uso.
- **Popularidad:** cuando una aplicación tiene gran cantidad de usuarios finales, es posible encontrar soluciones a posibles inquietudes o inconvenientes que aparezcan durante su utilización, gracias a foros, comunidades o investigaciones que hagan uso de ella.
- **Usabilidad:** facilidad en su uso y operabilidad, especialmente teniendo en cuenta que no es objeto del curso de Ingeniería del Software enseñar o especializarse en el uso de herramientas de modelado, por tanto la amigabilidad de la aplicación con el usuario es muy importante para adaptarse rápidamente a su uso y obtener los resultados esperados con una dedicación de tiempo adecuada.
- **Portabilidad:** tiene en cuenta la posibilidad de la aplicación de ser transferido a otro ambiente, sin importar el sistema operativo o versión. Además, evalúa la facilidad de instalación.

5.2.2.1 Funcionalidad

Tabla 4. Comparativo funcionalidad ArgoUML, StarUML y Visio.

HERRAMIENTA	CALIFICACIÓN	OBSERVACIONES
ArgoUML	**7**	• Soporta la mayoría de los diagramas especificados en UML (Clases, Estados, Casos de Uso, Actividad, Colaboración, Desarrollo, Secuencia). • Permite exportar imágenes de diagramas a formatos ampliamente conocidos (PNG, GIF, JPG, SVG, EPS). • Permite generar código fuente (Java, PHP, C++, Python, CSharp, C#). • Soporta ingeniería inversa. • No permite exportar a formatos de Microsoft Office. • Falta seguridad en el aspecto de autoguardados periódicos. • Muestra mensajes de alertas, cuando existen inconsistencias o errores en la generación de modelos. • No tiene posibilidad de deshacer.
StarUML	**9**	• Soporta la mayoría de los diagramas especificados en UML (Clases, Estados, Casos de Uso, Actividad, Colaboración, Desarrollo, Secuencia, Componentes, Estructura compuesta). • Permite exportar imágenes de diagramas a formatos ampliamente difundidos (JPG, BMP, EMF, WMF). • Tiene habilidad para generar código fuente desde los diagramas UML (Java, C++, C#). • Soporta ingeniería inversa. • Permite crear documentación en aplicaciones de Microsoft Office, tales como Word, Excel y PowerPoint. • Presenta verificación de modelos. • Falta seguridad en el aspecto de autoguardados periódicos.
	5	• Soporta la mayoría de los diagramas especificados en UML (Estados, Casos de Uso, Actividad, Colaboración, Desarrollo, Secuencia, Componentes, Estructura estática). Falta el diagrama de clases. • Permite exportar imágenes en múltiples formatos: Tiff, JPG, GIF, PNG, Mapa de bits, metaarchivo. • No genera código fuente. • No presenta verificación de modelos. • Aunque no genera documentación en aplicaciones de Microsoft Office, si es compatible con esta aplicación, por pertenecer a esta casa matriz. • Permite deshacer y hacer autoguardados.

Fuente: Autor del proyecto.

5.2.2.2 Estabilidad

Tabla 5. Comparativo Estabilidad ArgoUML, StarUML y Visio

HERRAMIENTA	CALIFICACIÓN	OBSERVACIONES
ArgoUML	**8**	• Aunque no presentó fallos durante su valoración, el hecho de no permitir autoguardados o la función de deshacer, puede conllevar a pérdidas de información en caso de existir fallas en el hardware o bloqueos del sistema.
StarUML	**9**	• Tampoco presentó fallos durante su valoración, sin embargo, no posee la función de autoguardado, lo cual también puede ser una falencia en el momento de errores o fallos del equipo donde se esté trabajando.
Microsoft Visio	**10**	• Facilita la recuperación de información al tener la función de autoguardado y deshacer. No presentó fallos durante la valoración.

Fuente: Autor del proyecto.

5.2.2.3 Licenciamiento

Tabla 6. Comparativo Licenciamiento ArgoUML, StarUML y Visio.

HERRAMIENTA	CALIFICACIÓN	OBSERVACIONES
ArgoUML	**9**	• Distribuido bajo licencia EPL (Eclipse Public License)
StarUML	**10**	• Distribuido bajo licencia GNU GLP (General Public License)
Microsoft Visio	**0**	• Requiere licencia.

Fuente: Autor del proyecto.

5.2.2.4 Popularidad

Tabla 7. Comparativo Popularidad ArgoUML, StarUML y Visio.

HERRAMIENTA	CALIFICACIÓN	OBSERVACIONES
ArgoUML	**8**	• Fue desarrollado en 1998 y a mediados de 2001 ya más de 100.000 descargas habían sido efectuadas. A finales de 2005 ya tenía más de 1.200.000 descargas. De las últimas 3 versiones, desarrolladas en el 2001, ya se han realizado más de 180.000 descargas, lo cual muestra el gran impacto de esta aplicación. Datos tomados de http://argouml-users.net/index.php?title=Download_statistics y de [ArgoUML, 2011]. • Existen más de 1.200 autores inscritos en foros o discusiones sobre la aplicación.

Fuente: Autor del proyecto.

Tabla 7. Comparativo Popularidad ArgoUML, StarUML y Visio. (Continuación)

HERRAMIENTA	CALIFICACIÓN	OBSERVACIONES
StarUML	**9**	• Aunque fue desarrollado en 1996, solo hasta 2005 recibe el nombre de StarUML y se torna como un proyecto de código abierto. En el 2006 ya tenía 20.000 usuarios registrados. • Desde 2005 hasta noviembre de 2012 se han realizado más de 2'500.000 descargas de la aplicación en su página oficial. Datos tomados de http://sourceforge.net/projects/staruml/files/stats/timeline?dates=2005-11-09+to+2012-12-01.
Microsoft Visio	**5**	• Por ser una aplicación que requiere licencia, no es muy popular. Viene con el paquete de Microsoft Office, por tanto depende de quienes usen estas herramientas. • No tiene la especialidad de desarrollo de diagramas UML, sino que maneja infinidad de diagramas (de flujo, de negocios, bases de datos, de ingeniería, etc). Es más apropiado para documentación que para análisis de modelos.

Fuente: Autor del proyecto.

5.2.2.5 Usabilidad

Tabla 8. Comparativo Usabilidad ArgoUML, StarUML y Visio.

HERRAMIENTA	CALIFICACIÓN	OBSERVACIONES
ArgoUML	**8**	• Presenta facilidad en la selección de diagramas y elementos, aunque los nombres de los componentes se muestran en mensajes emergentes y no permanentes. • No presenta la función de deshacer. • Muestra los diagramas elaborados de forma ordenada. Tiene interfaz amigable, aunque es difícil entender algunas propiedades. • Es intuitivo. • No permite representar estereotipos en diagramas de secuencia. • No muestra las acciones numeradas en el diagrama de secuencia. • El resultado de los diagramas tienen formato estándar y son fácilmente exportables. • Muestra mensajes de advertencia a los usuarios para facilitar su detección y temprana corrección.

Fuente: Autor del proyecto.

Tabla 8. Comparativo Usabilidad ArgoUML, StarUML y Visio. (Continuación)

HERRAMIENTA	CALIFICACIÓN	OBSERVACIONES
StarUML	**10**	• Facilita la selección de diagramas y elementos y los representa con sus nombres permanentemente. • Tiene función de deshacer. • Es intuitivo y organizado. Tiene interfaz amigable. • Permite la representación de estereotipos en los diagramas de secuencia. • Muestra las acciones numeradas en los diagramas de secuencia. • El resultado de los diagramas tiene formato estándar y son fácilmente exportables. • Muestra mensajes de advertencia emergentes para facilitar detección y corrección.
Microsoft Visio	**5**	• Es complicado la selección de diagramas y elementos. Al ser una aplicación que no es solo para diagramas UML, no agrupa estos diagramas en una sola categoría sino que los mezcla con otros correspondientes a ingeniería. • Tiene función de deshacer y de de autoguardado. • Es amigable al usuario, pero no es intuitivo. Los componentes se deben buscar, pues no aparecen en la interfaz principal. • No numera las acciones en los diagramas de secuencia. • No maneja representación de estereotipos en diagramas de secuencia. • Los diagramas se pueden exportar fácilmente en formato de imagen. • No muestra mensajes de advertencia, debido a que no hace verificación.

Fuente: Autor del proyecto.

5.2.2.6 Portabilidad

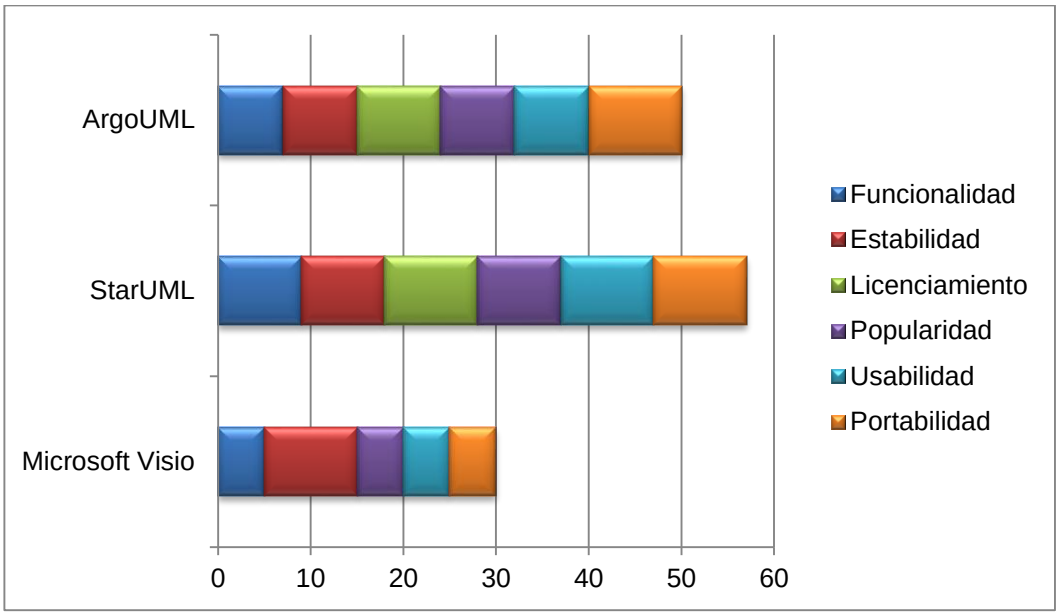
Tabla 9. Comparativo Portabilidad ArgoUML, StarUML y Visio.

HERRAMIENTA	CALIFICACIÓN	OBSERVACIONES
ArgoUML	**10**	• Es compatible con Linux, pero para nuestro caso de estudio, no es relevante, pues se trabaja en ambiente Windows. • La instalación es fácil e intuitiva. El archivo ejecutable se descarga del sitio oficial de la aplicación: http://argouml.tigris.org/
StarUML	**10**	• Está diseñado para trabajar bajo Windows. • La instalación es fácil e intuitiva. El archivo ejecutable se descarga del sitio oficial de la aplicación: http://staruml.sourceforge.net/en/download.php
Microsoft Visio	**6**	• Está diseñado para trabajar bajo Windows. • Instalación fácil e intuitiva. Se requiere licencia

Fuente: Autor del proyecto.

En la Figura 42 se aprecia un resumen gráfico de la valoración dada a cada herramienta, según las categorías descritas.

Figura 42. Cuadro comparativo de herramientas de modelado UML.



Fuente: Autor del proyecto.

En conclusión, se puede notar una ventaja de la aplicación StarUML frente a las otras dos aplicaciones. Presenta un profesional acabado y maneja muchos detalles en la elaboración de los diagramas. Es amigable, funcional y de licencia libre. Por tanto, la herramienta escogida para esta categoría es STARUML.

Para complementar esta decisión, se realizó una tabla comparativa de las principales características de las tres herramientas (Tabla 10)

Tabla 10. Comparativo principales características de ArgoUML, StarUML y Visio

CARACTERÍSTICA	ARGOUML	STARUML	VISIO
Licencia libre	Si	Si	No
Código abierto	Si	Si	No
Funcionalidad para diseñar modelos de casos de uso	Si	Si	Si
Funcionalidad para diagramas de clase, secuencia, colaboración, estado, actividades y despliegue	Si	Si	No (falta diagrama de clases)
Genera código fuente de clases elaboradas en un lenguaje de programación (Java, PHP, SQL, entre otros)	Si	Si	No
Fácil manejo, en términos de simbología estándar y organización del entorno	Si	Si	No
Exportar diagramas a formato de imagen	Si	Si	Si
Detección y sugerencias ante errores	Si	Si	No
Permite exportar a formatos de Microsoft Office	No	Si	Si
Etiquetas permanentes para referenciar cada elemento	No (maneja mensajes emergentes)	Si	Si
Soporta ingeniería inversa	Si	Si	No
Autoguardados periódicos	No	No	Si
Utilidad de deshacer	No	Si	Si
Muestra acciones numeradas en el diagrama de secuencia	No	Si	No
Instalación fácil e intuitiva	Si	Si	Si

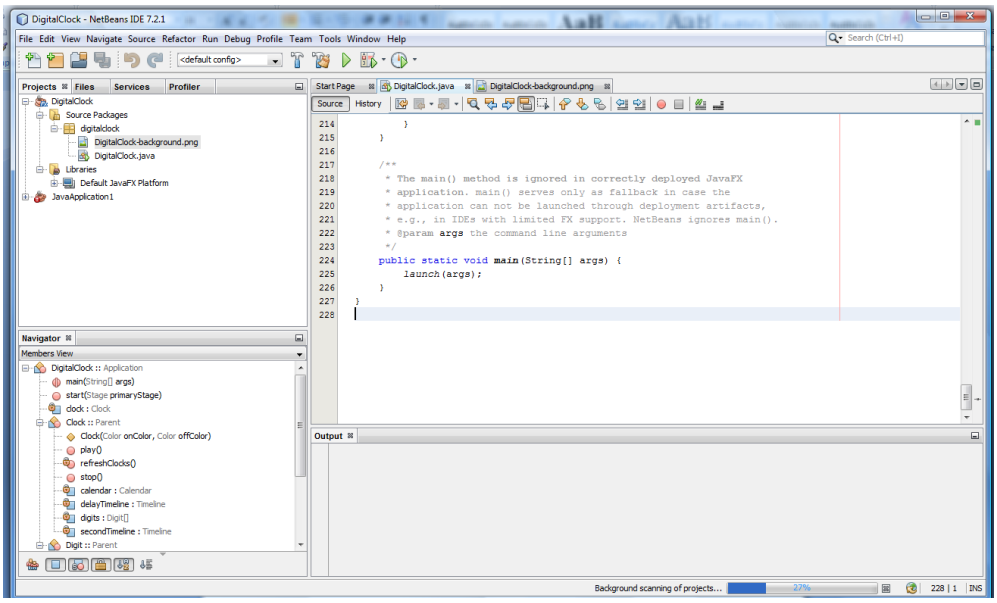
Fuente: Autor del proyecto.

5.2.3 Herramientas de software para la programación de aplicaciones. Al final de la asignatura Ingeniería del Software I, impartida en la Universidad Industrial de Santander como parte del programa de Ingeniería de Sistemas, el estudiante debe hacer entrega de una aplicación funcional del sistema diseñado y trabajado durante el semestre. Por tanto, es de interés de la presente investigación, analizar cuál sería la herramienta de programación que podría encajar mejor según las necesidades de la asignatura.

Debido a que las aplicaciones de diagramación UML están basadas en JAVA y teniendo en cuenta que este lenguaje es ampliamente conocido, funcional, estable y de licenciamiento GNU (Licencia Pública General), se ha determinado como la

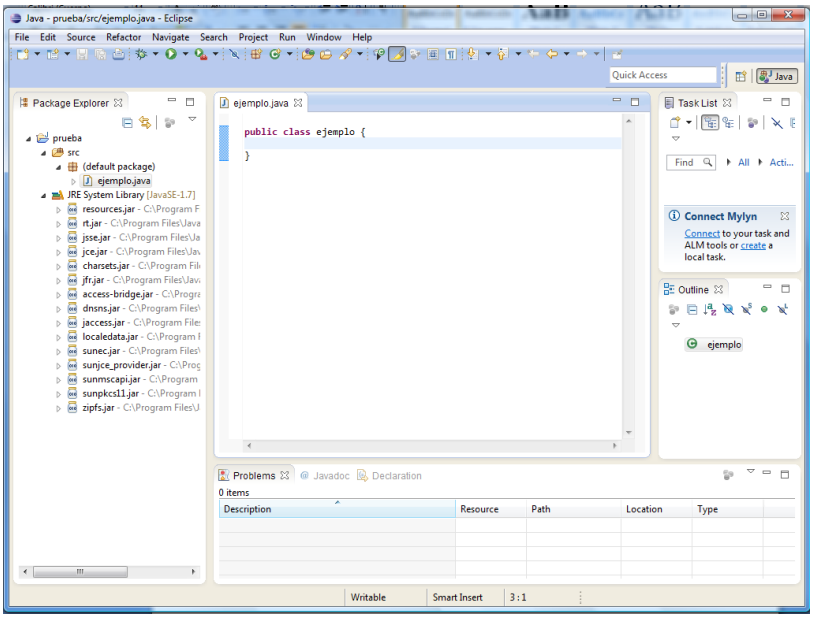
herramienta de desarrollo software. Dentro de la presente investigación se analizarán dos de las más conocidas que representan esta aplicación: Eclipse y Netbeans [Netbeans vs. Eclipse]. Ambas son entornos de desarrollo integrados de código abierto. En las Figuras 43 y 44 se aprecia una vista general de la interfaz de estas herramientas.

Figura 43. Vista general de la interfaz de Netbeans 7.2.1.



Fuente: Captura de pantalla Netbeans.

Figura 44. Vista general de la interfaz de Eclipse 4.2.



Fuente: Captura de pantalla Eclipse.

5.2.4 Comparativo de herramientas de software para la programación de aplicaciones

5.2.4.1 Usabilidad

Tabla 11. Comparativo Usabilidad Eclipse y Netbeans.

HERRAMIENTA	CALIFICACIÓN	OBSERVACIONES
Eclipse	**7**	• Es complicado encontrar funcionalidades. • Falta documentación en el sitio oficial (http://www.eclipse.org/). • Permite autocompletar funciones ahorrando tiempo.
Netbeans	**9**	• Es intuitivo y es más rápido para encontrar las funcionalidades en la barra de tareas que en Eclipse. • Presenta mejor y más documentación y soporte en su sitio oficial (http://netbeans.org/). • Permite autocompletar funciones, al igual que Eclipse, pero tiene más opciones y acierta con más frecuencia que ésta. • Es fácil de entender un proyecto, debido a la pulida organización de la estructura del código.

Fuente: Autor del proyecto.

5.2.4.2 Funcionalidad

Tabla 12. Comparativo Funcionalidad Eclipse y Netbeans.

HERRAMIENTA	CALIFICACIÓN	OBSERVACIONES
Eclipse	**9**	• Es más rápido y toma menos memoria. • Soporta diversos lenguajes tales como C/C++, PHP, JavaScript. • Tiene control de versiones con CVS.
Netbeans	**8**	• Es rápido pero toma más memoria. • Soporta muchos lenguajes para Java (C/C++, XML y HTML, PHP, Groovy, Javadoc, JavaScript y JSP. • Es organizado en la estructura de archivos. • Presenta validación de código.

Fuente: Autor del proyecto.

Debido a que ambas herramientas son de uso libre, adjuntamos la ubicación de descarga del sitio oficial:

- Descarga de Netbeans: http://www.oracle.com/technetwork/java/javase/downloads/jdk-7-netbeans-download-432126.html
- Descarga de Eclipse: http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/juno/SR1/eclipse-jee-juno-SR1-win32.zip

Ambas herramientas cumplen con la funcionalidad requerida, sin embargo, en términos de amigabilidad con el usuario, NetBeans es una mejor alternativa, es más intuitiva, permite autocompletar, es más organizada, aunque es un poco más lenta al abrir. En conclusión, aunque la diferencia con Eclipse es mínima, la herramienta de desarrollo recomendada es NetBeans.

En la tabla 13 se muestra un comparativo de las dos herramientas incluyendo todas las características

Tabla 13. Comparativo de herramientas para el desarrollo de aplicaciones

CARACTERÍSTICA	ECLIPSE	NETBEANS
Licencia libre	Si	Si
Código abierto	Si	Si
Aplicación basada en JAVA	Si	Si
Facilidad para encontrar funcionalidades	No	Si
Facilidad para encontrar documentación	No	Si
Permite autocompletar código	Si	Si
Es rápido	Si	Si
Consume poca memoria	Si	No
Soporta diversos lenguajes como C/C++, PHP, JavaScript	Si	Si

Fuente: Autor del proyecto.

6 ENCUESTA PARA ANALIZAR PROCESOS DE APRENDIZAJE EN EL CURSO DE INGENIERÍA DEL SOFTWARE

La encuesta es una técnica estandarizada de obtención de información, basada en un conjunto objetivo y articulado de preguntas que, a través de su análisis, permite entender mejor un fenómeno o situación como apoyo para la toma de decisiones.

En el presente estudio, se realizó una encuesta a los estudiantes del curso de Ingeniería de Software II (**quienes ya cursaron el programa de ingeniería del software I y tienen criterio para definir cuáles estrategias fueron relevantes para su proceso de aprendizaje**), con el propósito de valorar su proceso académico en el área, a partir de la metodología usada por sus docentes. Se tomó la decisión de realizar una encuesta, porque, entre otras ventajas, esta técnica de obtención de información se destaca por [Grande, 2005]

- Estandarización: por basarse en un cuestionario permite que las preguntas sean las mismas para todos los individuos evaluados. Esto conlleva a la homogeneidad de la información.
- Facilidad de administración: es de fácil entendimiento y es una técnica ágil e inmediata para obtener información.
- Simplificación del tratamiento de datos: debido a que las respuestas se asocian a códigos, es posible analizar la información de forma cuantitativa, incluso si los datos son de índole cualitativa.
- Posibilidad de hacer estudios parciales: las encuestas poseen identificadores de las características de los individuos evaluados, de modo que permiten estudiar el comportamiento generalizado de la población, según el prototipo de los encuestados.

6.1 GENERACIÓN DEL CUESTIONARIO

Para determinar el cuestionario que se plantearía a los estudiantes, se especificaron los parámetros principales que involucra una metodología de enseñanza del curso de Ingeniería de Software y que son relevantes para obtener un buen desempeño. Estos parámetros se basaron en lo analizado en el capítulo 4, donde se definieron las características que se abordarían para estructurar la metodología de enseñanza/aprendizaje:

- Tiempo de dedicación independiente.
- Investigación y lectura complementaria en temas de interés de la asignatura.
- Tipo de trabajo en clase (grupal o individual).
- Desarrollo o no de un proyecto de software. Origen del proyecto, forma de trabajo, resultados en términos de metas cumplidas, forma de evaluación del proyecto y tiempo de destinación.
- Uso de herramientas de apoyo.
- Forma de evaluación de la asignatura.
- Aplicación de foros como forma de socialización resultados.
- Habilidades desarrolladas en el curso.

A partir de los parámetros mencionados, se estableció el cuestionario definitivo, teniendo en cuenta que se aplicaría en línea.

La herramienta escogida para este fin se llama encuesta fácil y es de uso gratuito por un término de noventa días (www.encuestafacil.com). Permite la creación de plantillas a la medida, con diversas opciones de respuesta y algunas propiedades de diseño, lo cual la hace personalizada a las necesidades de la investigación. Además, guarda la información recopilada de forma segura. El usuario puede responder de forma rápida y anónima, lo cual brinda autonomía y confianza para

contestar con total honestidad. En la figura 45 se aprecia el encabezado de la encuesta.

Figura 45. Encabezado del formulario de la encuesta.

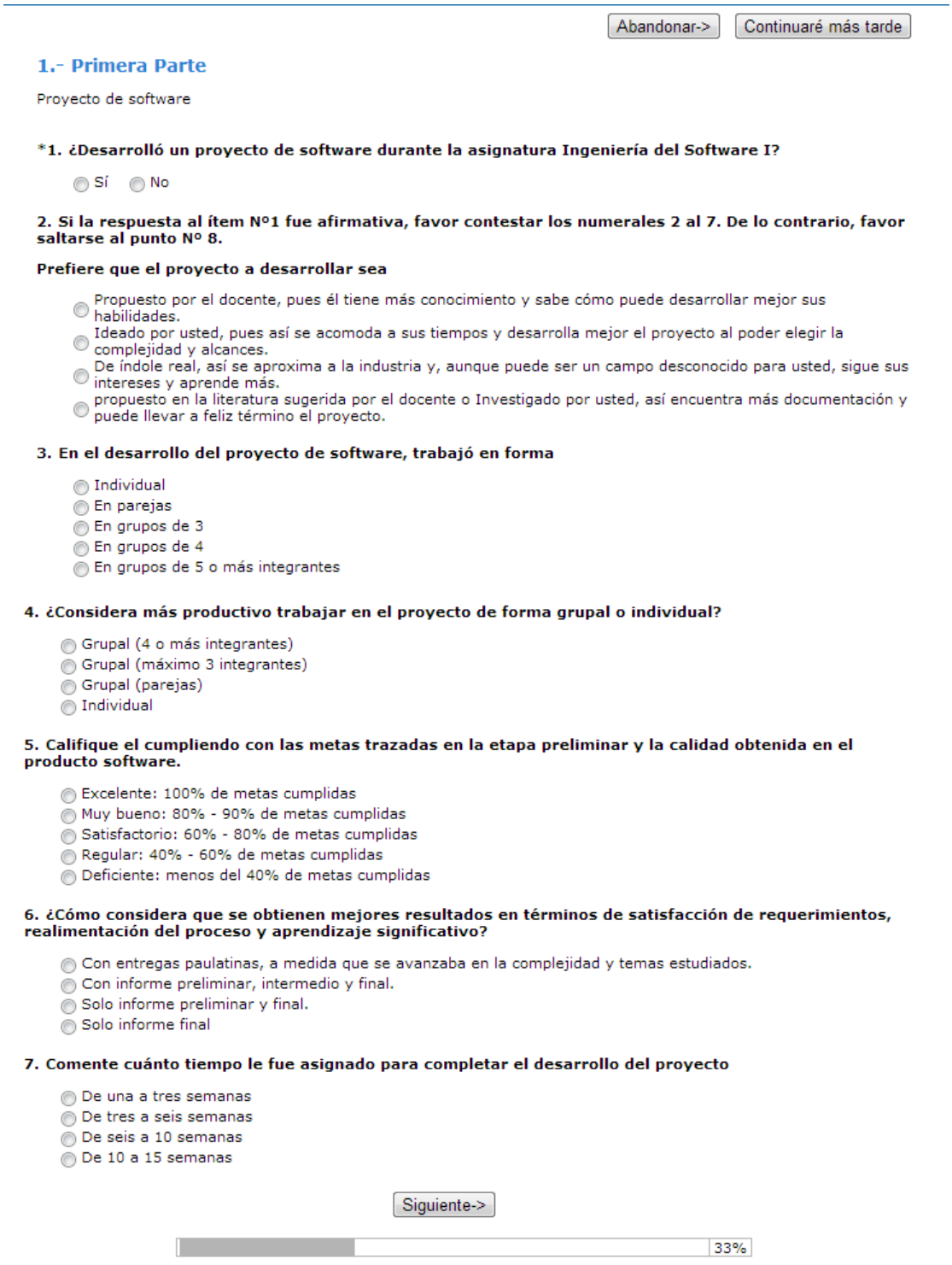
Fuente: Captura de pantalla formulario de la encuesta.

La encuesta se dividió en tres partes principales, agrupadas de la siguiente forma:

1. Proyecto de software: Si se desarrolló un proyecto de software durante la asignatura y las principales características de dicho proyecto. (Ver Figura 46).
2. Lecturas complementarias y herramientas de apoyo. (Ver Figura 47).
3. Aplicación de foros y evaluación. (Ver Figura 48).

Al crear el formulario con la encuesta, se generó un link que fue activado en cada computador donde se iba a aplicar. Las preguntas seleccionadas con un asterisco (*) son de respuesta obligatoria, es decir, son vitales para poder continuar y finalizar el cuestionario.

Figura 46. Primera parte de la encuesta - Proyecto de Software.



Abandonar-> Continuaré más tarde

1.- Primera Parte

Proyecto de software

***1. ¿Desarrolló un proyecto de software durante la asignatura Ingeniería del Software I?**

Sí No

2. Si la respuesta al ítem Nº1 fue afirmativa, favor contestar los numerales 2 al 7. De lo contrario, favor saltarse al punto Nº 8.

Prefiere que el proyecto a desarrollar sea

Propuesto por el docente, pues él tiene más conocimiento y sabe cómo puede desarrollar mejor sus habilidades.
Ideado por usted, pues así se acomoda a sus tiempos y desarrolla mejor el proyecto al poder elegir la complejidad y alcances.
De índole real, así se aproxima a la industria y, aunque puede ser un campo desconocido para usted, sigue sus intereses y aprende más.
propuesto en la literatura sugerida por el docente o Investigado por usted, así encuentra más documentación y puede llevar a feliz término el proyecto.

3. En el desarrollo del proyecto de software, trabajó en forma

Individual
En parejas
En grupos de 3
En grupos de 4
En grupos de 5 o más integrantes

4. ¿Considera más productivo trabajar en el proyecto de forma grupal o individual?

Grupal (4 o más integrantes)
Grupal (máximo 3 integrantes)
Grupal (parejas)
Individual

5. Califique el cumpliendo con las metas trazadas en la etapa preliminar y la calidad obtenida en el producto software.

Excelente: 100% de metas cumplidas
Muy bueno: 80% - 90% de metas cumplidas
Satisfactorio: 60% - 80% de metas cumplidas
Regular: 40% - 60% de metas cumplidas
Deficiente: menos del 40% de metas cumplidas

6. ¿Cómo considera que se obtienen mejores resultados en términos de satisfacción de requerimientos, realimentación del proceso y aprendizaje significativo?

Con entregas paulatinas, a medida que se avanzaba en la complejidad y temas estudiados.
Con informe preliminar, intermedio y final.
Solo informe preliminar y final.
Solo informe final

7. Comente cuánto tiempo le fue asignado para completar el desarrollo del proyecto

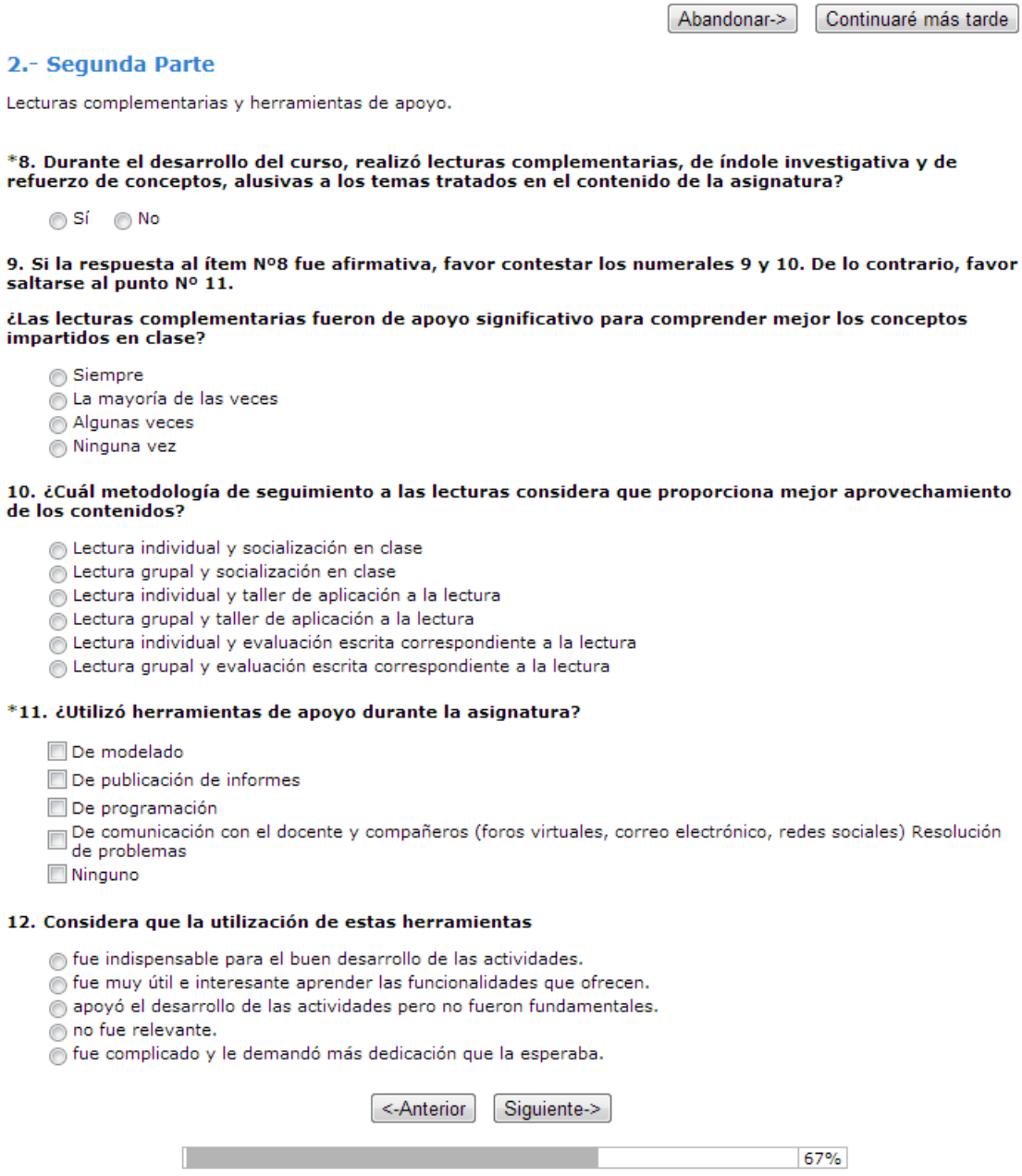
De una a tres semanas
De tres a seis semanas
De seis a 10 semanas
De 10 a 15 semanas

Siguiente->

33%

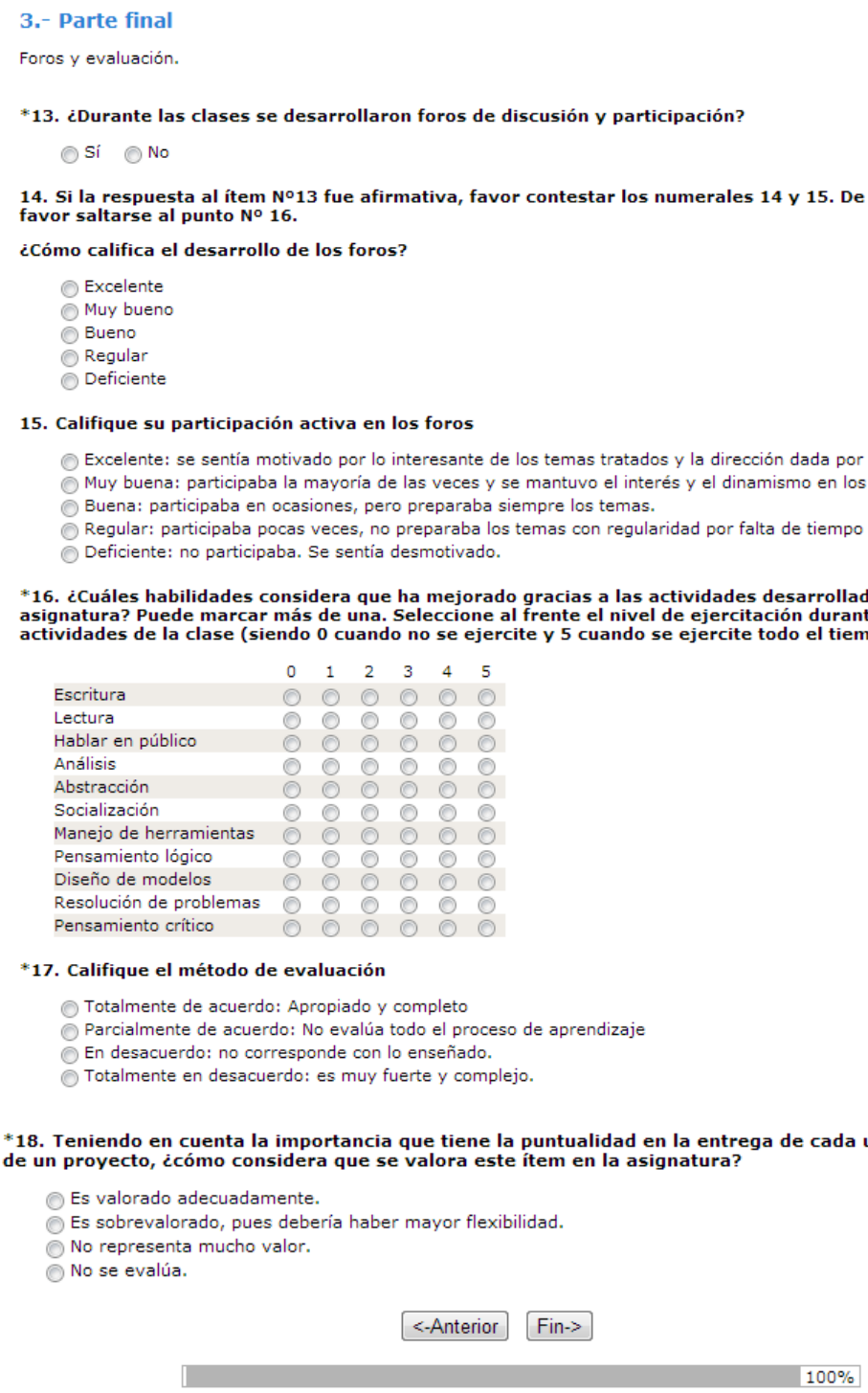
Fuente: Captura de pantalla formulario de la encuesta.

Figura 47. Segunda parte de la encuesta - Lecturas complementarias y herramientas de apoyo.



Fuente: Captura de pantalla formulario de la encuesta.

Figura 48. Parte final de la encuesta. Foros y evaluación.



3.- Parte final

Foros y evaluación.

***13. ¿Durante las clases se desarrollaron foros de discusión y participación?**

Sí No

14. Si la respuesta al ítem Nº13 fue afirmativa, favor contestar los numerales 14 y 15. De lo contrario, favor saltarse al punto Nº 16.

¿Cómo califica el desarrollo de los foros?

Excelente
Muy bueno
Bueno
Regular
Deficiente

15. Califique su participación activa en los foros

Excelente: se sentía motivado por lo interesante de los temas tratados y la dirección dada por el docente.
Muy buena: participaba la mayoría de las veces y se mantuvo el interés y el dinamismo en los foros.
Buena: participaba en ocasiones, pero preparaba siempre los temas.
Regular: participaba pocas veces, no preparaba los temas con regularidad por falta de tiempo o motivación.
Deficiente: no participaba. Se sentía desmotivado.

***16. ¿Cuáles habilidades considera que ha mejorado gracias a las actividades desarrolladas durante la asignatura? Puede marcar más de una. Seleccione al frente el nivel de ejercitación durante las actividades de la clase (siendo 0 cuando no se ejercite y 5 cuando se ejercite todo el tiempo)**

	0	1	2	3	4	5
Escritura						
Lectura						
Hablar en público						
Análisis						
Abstracción						
Socialización						
Manejo de herramientas						
Pensamiento lógico						
Diseño de modelos						
Resolución de problemas						
Pensamiento crítico						

***17. Califique el método de evaluación**

Totalmente de acuerdo: Apropiado y completo
Parcialmente de acuerdo: No evalúa todo el proceso de aprendizaje
En desacuerdo: no corresponde con lo enseñado.
Totalmente en desacuerdo: es muy fuerte y complejo.

***18. Teniendo en cuenta la importancia que tiene la puntualidad en la entrega de cada una de las fases de un proyecto, ¿cómo considera que se valora este ítem en la asignatura?**

Es valorado adecuadamente.
Es sobrevalorado, pues debería haber mayor flexibilidad.
No representa mucho valor.
No se evalúa.

<-Anterior Fin->

100%

Fuente: Captura de pantalla formulario de la encuesta.

6.2 FICHA TÉCNICA DE LA ENCUESTA

Tabla 14. Ficha técnica de la encuesta.

POBLACIÓN OBJETIVO	Estudiantes activos de Ingeniería de Sistemas de la Universidad Industrial de Santander (Semestre 1 de 2013). • Ingeniería del Software II (vieron IS I el semestre anterior).
MÉTODO DE MUESTREO	No probabilístico por conglomerados. Estudiantes que se encontraban en clase el día seleccionado para la aplicación de la encuesta.
FECHA DE APLICACIÓN DE LA ENCUESTA	• Grupo 1: 15 de Agosto de 2013 • Grupo 2: 25 de Septiembre de 2013
TAMAÑO DE LA MUESTRA	42 estudiantes (17 en el grupo 1 y 25 en el grupo 2)
TIEMPO PROMEDIO POR ENCUESTA	7 minutos y 8 segundos
FORMA DE EVALUACIÓN	En línea (www.encuestafacil.com)
NIVEL DE CONFIANZA	95 %

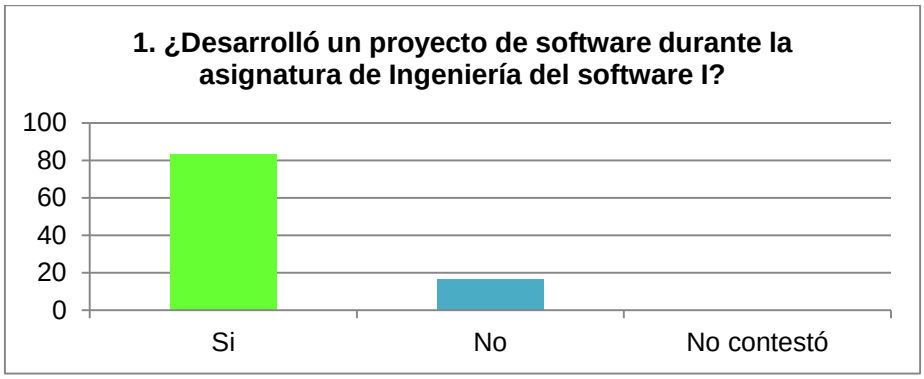
Fuente: Autor del proyecto.

6.3 ANÁLISIS POR PREGUNTA

6.3.1 Primera parte: Proyecto de Software. Dentro de los aspectos más importantes que debe llevar un curso de Ingeniería del Software es un el desarrollo de un proyecto de software, donde los estudiantes tendrán la oportunidad de practicar y ejecutar sus conceptos sobre las fases iniciales del campo de estudio (Aprendizaje Basado en Proyectos). Por este motivo, se inicia esta recopilación de experiencias y opiniones con este ítem. A través de la experiencia del director del proyecto, y con lo estudiado por otras universidades en el mundo, se aprecia la relevancia que implica que los estudiantes se enfrenten con un proyecto software durante su preparación, para poder enfrentarse con mayor seguridad y experiencia en su ámbito laboral.

A continuación se analizan una a una las preguntas referentes al desarrollo de un proyecto de software en la asignatura Ingeniería del Software I.

Figura 49. Diagrama de barras y tabla de resultados pregunta 1.



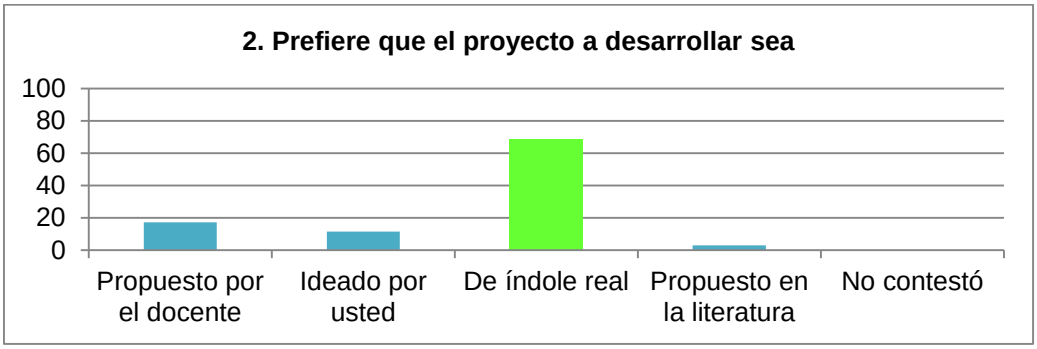
	SI	NO	NO CONTESTÓ	TOTAL
Respuestas	**35**	**7**	**0**	**42**
Porcentaje	**83,33 %**	**16,67 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

La mayoría de los estudiantes encuestados (83,33 %) afirmó haber desarrollado un proyecto de software durante la asignatura Ingeniería del Software I. Es una tendencia que se aprecia en el curso. (Figura 49)

Si contestó afirmativamente la pregunta 1, favor contestar los numerales 2 al 7. De lo contrario favor saltarse al ítem N° 8.

Figura 50. Diagrama de barras y tabla de resultados pregunta 2.



	Opción 1	Opción 2	Opción 3	Opción 4	No contestó	TOTAL
Respuestas	**6**	**4**	**24**	**1**	**0**	**35**
Porcentaje	**17,14 %**	**11,43 %**	**68,57 %**	**2,86 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

Opción 1: Propuesto por el docente, pues él tiene más conocimiento y sabe cómo puede desarrollar mejor sus habilidades.

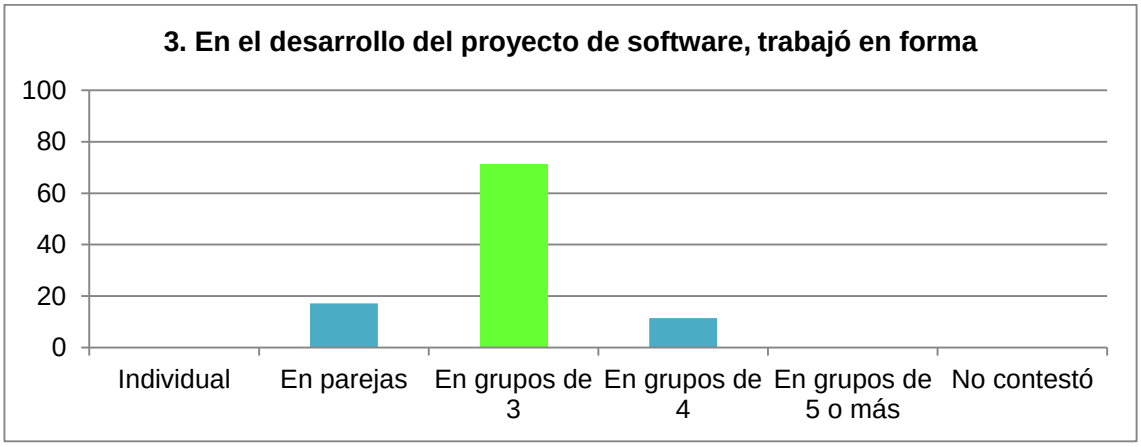
Opción 2: Ideado por usted, pues así se acomoda a sus tiempos y desarrolla mejor el proyecto al poder elegir la complejidad y alcances.

Opción 3: De índole real, así se aproxima a la industria y, aunque puede ser un campo desconocido para usted, sigue sus intereses y aprende más.

Opción 4: Propuesto en la literatura sugerida por el docente o investigada por usted, así encuentra más documentación y puede llevar a feliz término el proyecto.

La opción más escogida fue la N° 3 (de índole real) y la menos escogida la opción 4 (propuesto en la literatura). (Ver Figura 50). Esto nos demuestra que, los estudiantes entienden la importancia de enfocar los esfuerzos del aprendizaje y la práctica en la industria y sus requerimientos. Aunque el proyecto no sea aplicado a una industria en específico, puede plantearse de modo genérico, cumpliendo con los requerimientos de alguna necesidad en particular de un negocio determinado.

Figura 51. Diagrama de barras y tabla de resultados pregunta 3.

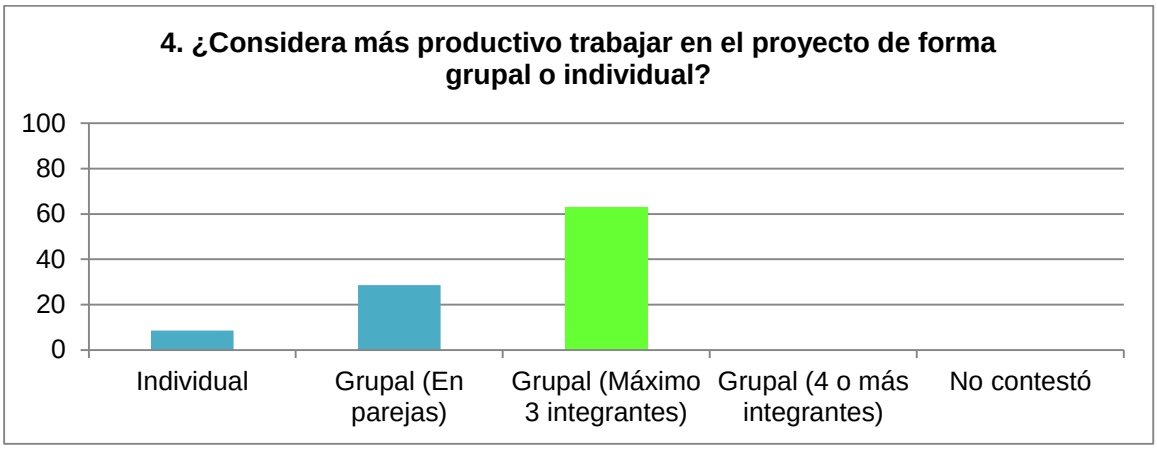


	Individual	En parejas	En grupos de 3	En grupos de 4	En grupos de 5 o más	No contestó	TOTAL
Respuestas	0	6	25	4	0	0	35
Porcentaje	0 %	17,14 %	71,43 %	11,43 %	0	0 %	100 %

Fuente: Autor del proyecto.

La mayoría de los estudiantes que realizaron un proyecto de software durante el curso de Ingeniería del Software I (**71.43%**) trabajó en equipos conformado por 3 estudiantes. Ninguno trabajó de forma individual ni en grupos con 5 integrantes o más (Figura 51); esto nos hace inferir que la tendencia se centra en grupos pequeños, lo cual indudablemente permite un mejor entendimiento entre sus integrantes y un mejor rendimiento en proyectos pequeños de índole académica. Esta característica se evidenció en el capítulo 4, cuando se analizaron diversas investigaciones sobre el tema a nivel mundial, donde prevalecía el trabajo en grupo, aplicando el aprendizaje entorno a un proyecto software.

Figura 52. Diagrama de barras y tabla de resultados pregunta 4.

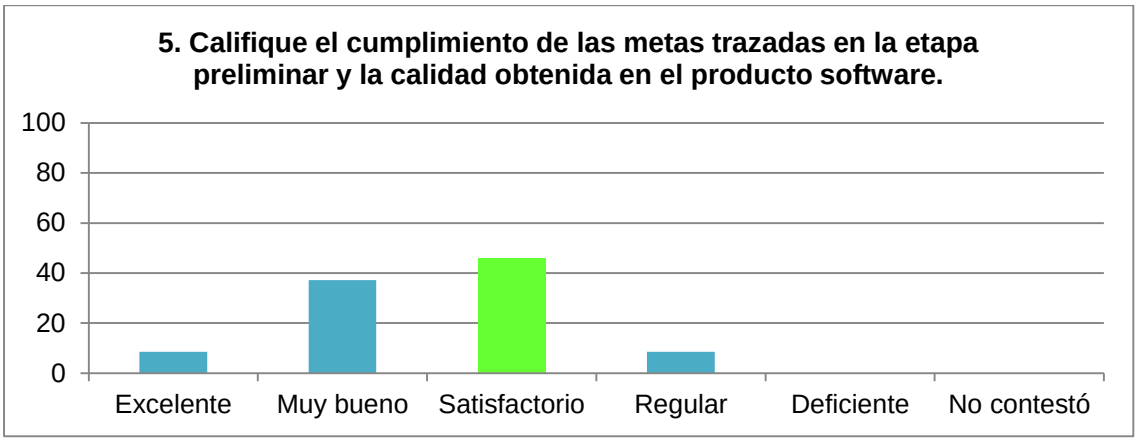


	Individual	Grupal (En parejas)	Grupal (Máximo 3 integrantes)	Grupal (4 o más integrantes)	No contestó	TOTAL
Respuestas	**3**	**10**	**22**	**0**	**0**	**35**
Porcentaje	**8,57 %**	**28,57 %**	**62,86 %**	**0 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

La tendencia se sigue manteniendo, donde la mayoría (más del 60%) prefiere trabajar en grupos pero pequeños, máximo de 3 integrantes. Nadie optó por grupos de 4 o más integrantes. (Figura 52). Esto nos confirma que los estudiantes entienden la importancia del trabajo en grupo, en miras del desarrollo a nivel industrial.

Figura 53. Diagrama de barras y tabla de resultados pregunta 5.



	Excelente	Muy bueno	Satisfactorio	Regular	Deficiente	No contestó	TOTAL
Respuestas	**3**	**13**	**16**	**3**	**0**	**0**	**35**
Porcentaje	**8,57 %**	**37,14 %**	**45,71 %**	**8,57 %**	**0 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

Los criterios dados se refieren directamente al porcentaje de metas cumplidas, es decir, cuando se determina que el cumplimiento fue excelente, es porque se completaron el 100% de metas trazadas y esto influye en la calidad del producto respecto a la planeación establecida.

Excelente: 100 % de las metas cumplidas.

Muy bueno: 80% - 90% de las metas cumplidas.

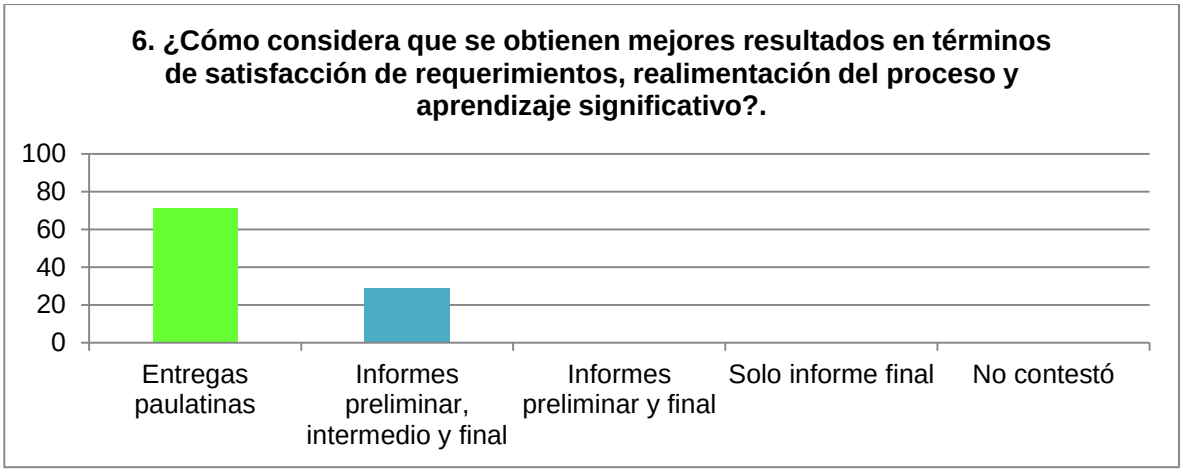
Satisfactorio: 60% - 80% de las metas cumplidas.

Regular: 40% - 60% de las metas cumplidas.

Deficiente: menos del 40% de las metas cumplidas.

Un poco más del 90% de los estudiantes que desarrollaron un proyecto software alcanzaron niveles de más del 60% del cumplimiento de metas trazadas, lo cual infiere un buen desempeño en la asignatura con los cambios que se han venido dando. (Figura 53).

Figura 54. Diagrama de barras y tabla de resultados pregunta 6.



	Opción 1	Opción 2	Opción 3	Opción 4	No contestó	TOTAL
Respuestas	**25**	**10**	**0**	**0**	**0**	**35**
Porcentaje	**71,43 %**	**28,57 %**	**0 %**	**0 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

Opción 1: Entregas paulatinas, a medida que se avanzaba en la complejidad y temas estudiados.

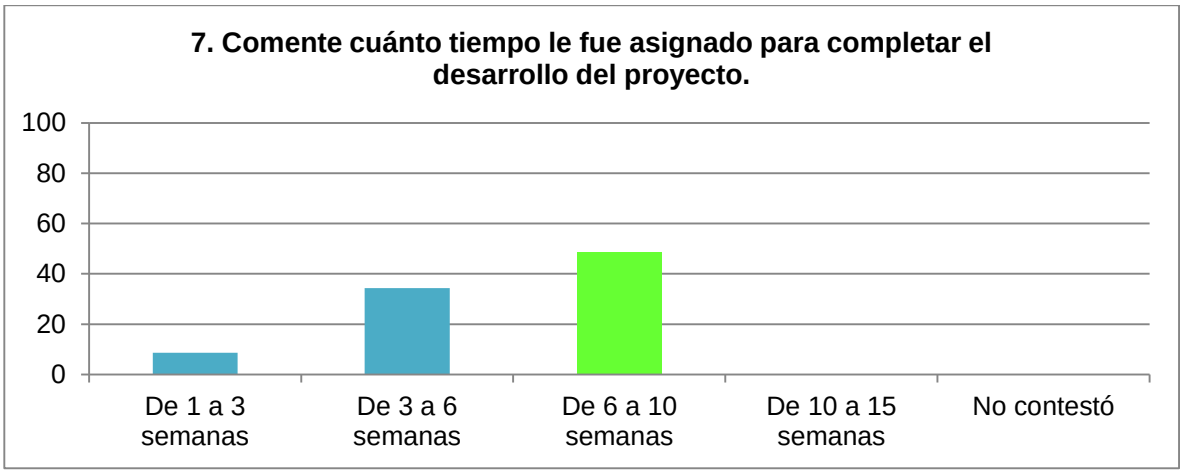
Opción 2: con informe preliminar, intermedio y final.

Opción 3: solo informe preliminar y final.

Opción 4: solo informe final.

Solo se escogieron las dos primeras opciones, lo cual indica que es aconsejable tener varios informes del avance del producto, ya que se puede ir mejorando el análisis y corrigiendo los modelos. La opción más escogida es la de entregas paulatinas. (Figura 54).

Figura 55. Diagrama de barras y tabla de resultados pregunta 7.



	De 1 a 3 semanas	De 3 a 6 semanas	De 6 a 10 semanas	De 10 a 15 semanas	No contestó	TOTAL
Respuestas	**3**	**12**	**17**	**3**	**0**	**35**
Porcentaje	**8,57 %**	**34,29 %**	**48,57 %**	**8,57 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

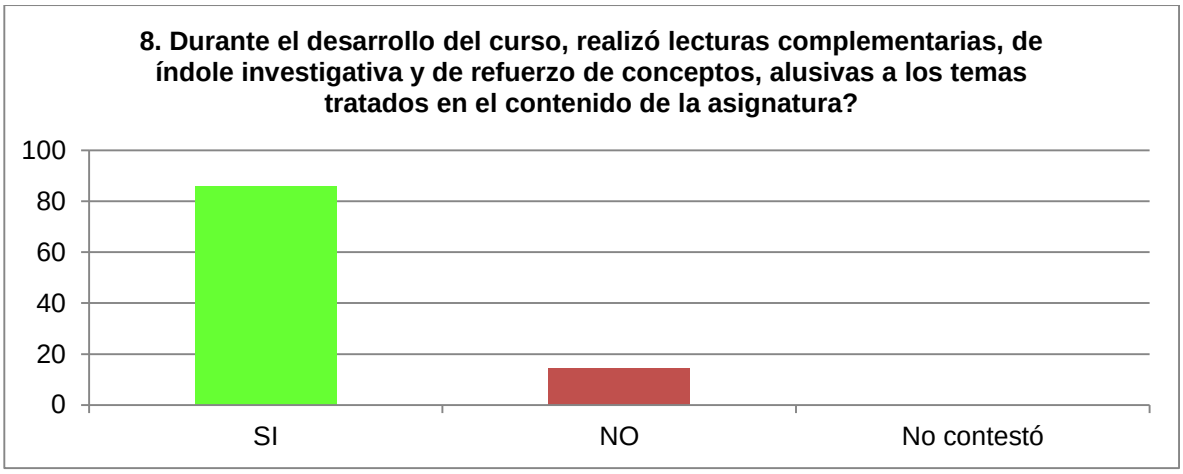
Esta pregunta infiere que el proyecto se debe extender la mayoría del semestre (10 semanas de 16), ver Figura 55. Al comienzo se deben establecer las pautas y dar los fundamentos teóricos que se requieren para escoger adecuadamente el negocio del proyecto. Proyectos muy cortos no permiten generar los modelos de requerimientos necesarios y si toman todo el semestre tampoco es aconsejable, pues no se puede hacer la socialización y no se podrían llevar a cabo otras actividades de soporte tales como los foros, minilecturas, talleres conceptuales y evaluaciones.

6.3.2 Segunda parte: Lecturas complementarias y herramientas de apoyo.

Dentro de los aspectos fundamentales que sirven como apoyo en el proceso de aprendizaje de la asignatura, se encontró que así como la fase experimental en el desarrollo de software en la academia es vital, es también indispensable desarrollar bases conceptuales sólidas; esto se logra a través de diversas estrategias, sin embargo, una forma muy dinámica de enfocar este proceso es a través de lecturas pequeñas y actividades de socialización ágiles, que mantengan el interés de los estudiantes y fortalezcan sus diversas competencias. Además, escoger herramientas software de apoyo es una tarea que permite la implementación del proyecto software y por tanto, vale la pena analizar esta labor.

En esta segunda parte, se abordan estos dos aspectos fundamentales para la enseñanza y sobre todo en el aprendizaje significativo de la asignatura. Se presentan a continuación los resultados detallados por ítem:

Figura 56. Diagrama de barras y tabla de resultados pregunta 8.



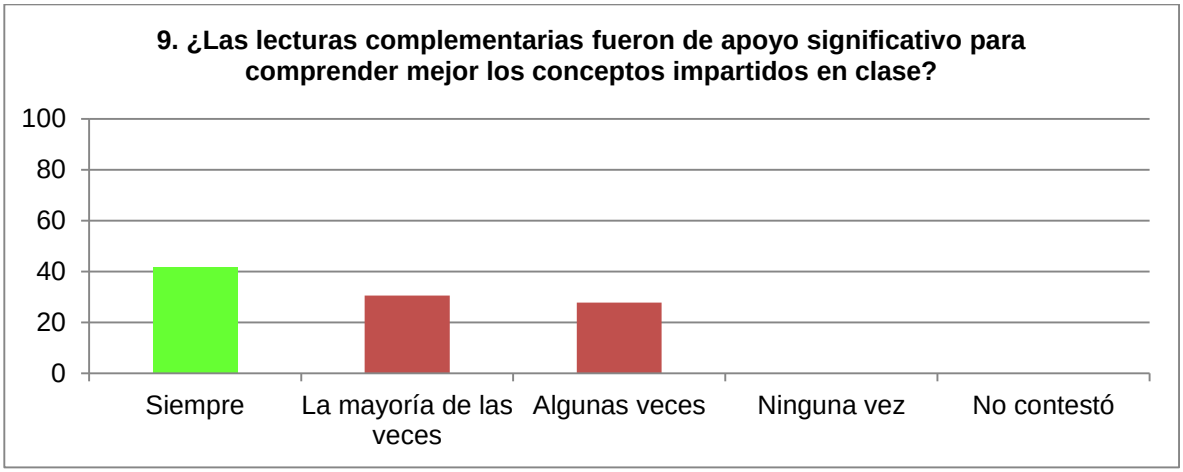
	SI	NO	No contestó	TOTAL
Respuestas	**36**	**6**	**0**	**42**
Porcentaje	**85,71 %**	**14,29 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

Más del 80% de los estudiantes encuestados confirman que se realizaron lecturas de apoyo. (Figura 56). Aquí resaltamos la importancia de ejercitar en los estudiantes el interés investigativo que promueve la lectura de temas fundamentales en el área.

Si la respuesta al ítem 8 fue afirmativa, favor contestar los numerales 9 y 10. De lo contrario, favor saltarse al punto N° 11.

Figura 57. Diagrama de barras y tabla de resultados pregunta 9.

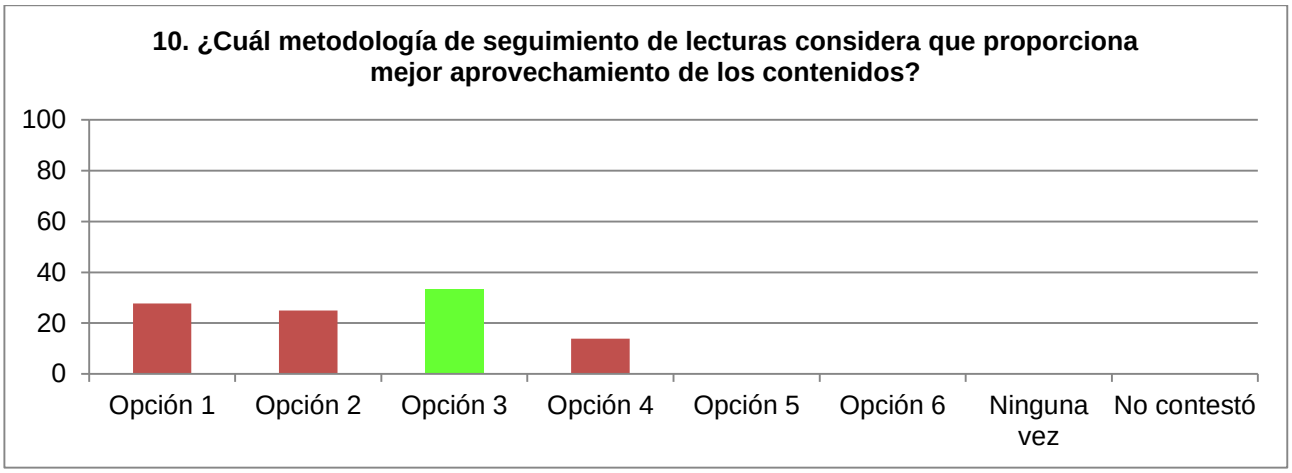


	Siempre	La mayoría de las veces	Algunas veces	Ninguna vez	No contestó	TOTAL
Respuestas	**15**	**11**	**10**	**0**	**0**	**36**
Porcentaje	**41,67 %**	**30,55 %**	**27,78 %**	**0 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

En esta pregunta se resalta que ningún estudiante de los que realizaron lecturas complementarias encontró esta labor como ineficiente (Figura 57). Además, más del 40% coincide que siempre fueron de apoyo significativo para mejorar su aprendizaje. El éxito de esta actividad depende básicamente del tipo de lectura que se escoja y del seguimiento y socialización que se realice. Es clave que se mantenga la motivación en los estudiantes y se guie adecuadamente su desarrollo.

Figura 58. Diagrama de barras y tabla de resultados pregunta 10.



	Opción 1	Opción 2	Opción 3	Opción 4	Opción 5	Opción 6	No contestó	TOTAL
Respuestas	**10**	**9**	**12**	**5**	**0**	**0**	**0**	**36**
Porcentaje	**27,78 %**	**25,00 %**	**33,33 %**	**13,89 %**	**0 %**	**0 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

Opción 1: lectura individual y socialización en clase.

Opción 2: lectura grupal y socialización en clase.

Opción 3: lectura individual y taller de aplicación a la lectura.

Opción 4: lectura grupal y taller de aplicación a la lectura.

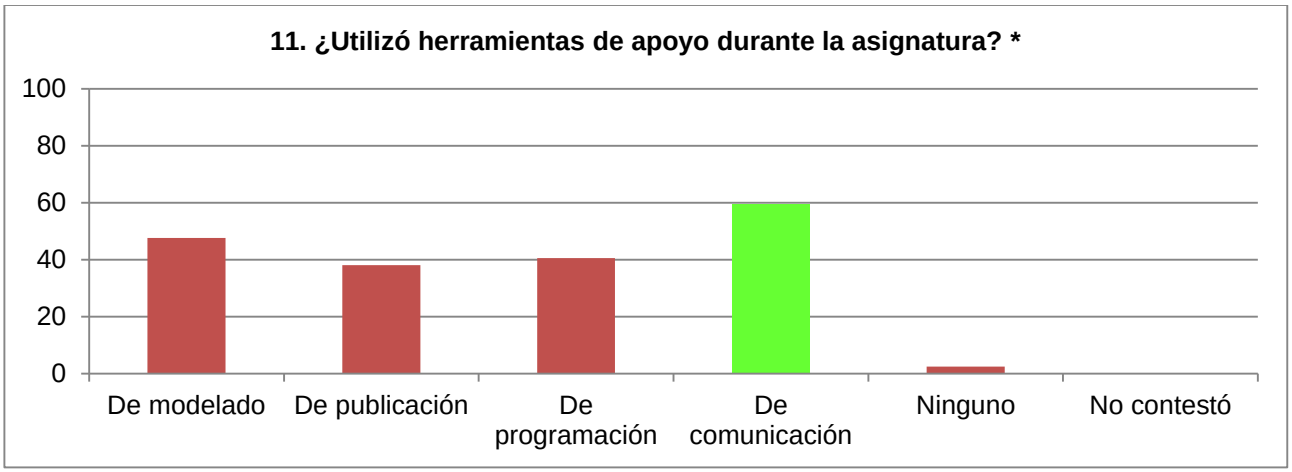
Opción 5: lectura individual y evaluación escrita correspondiente a la lectura.

Opción 6: lectura grupal y evaluación escrita correspondiente a la lectura.

Se aprecia que la tendencia es preferir la lectura individual (61,11%) sobre la grupal.(Ver Figura 58). Esto ocurre por el hecho de que cada individuo tiene diferentes estrategias de lectura y, al tener diferentes niveles conceptuales, esta tarea conlleva diferentes ritmos de avance. Además cabe resaltar que ningún estudiante optó por definir que la evaluación escrita es la mejor forma de calificar la apropiación de los conceptos de una lectura. No obstante, no es clara la tendencia, pues solo la tercera parte de la población encuestada que realizó lecturas complementarias, prefiere la aplicación de talleres después de una lectura individual. La forma de evaluación más escogida es la socialización en clase

(52,78 %), por ser una forma globalizada y cooperativa de lograr la apropiación de los conceptos.

Figura 59. Diagrama de barras y tabla de resultados pregunta 11.



	De modelado	De publicación	De programación	De comunicación	Ninguno	No contestó	TOTAL
Respuestas	**20**	**16**	**17**	**25**	**1**	**0**	**42**
Porcentaje	**47,62 %**	**38,10 %**	**40,48 %**	**59,52 %**	**2,38 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

* Se podían escoger varias opciones al tiempo.

Opción 1: de modelado.

Opción 2: de publicación de informes.

Opción 3: de programación.

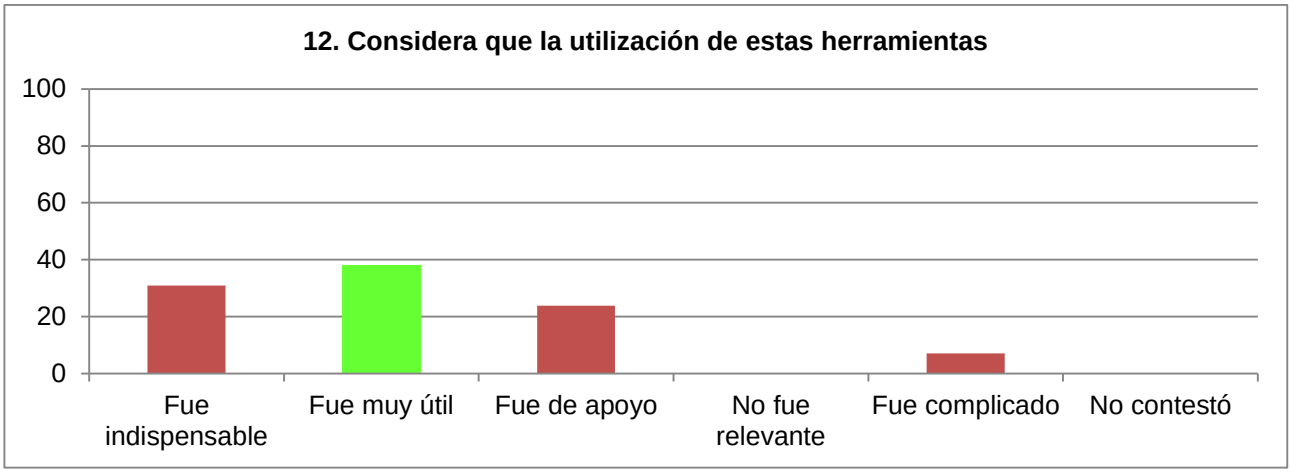
Opción 4: de comunicación con el docente y compañeros (foros virtuales, correo electrónico, redes sociales).

Opción 5: ninguno.

Es notoria la tendencia a usar diferentes herramientas de apoyo que son muy importantes como soporte al aprendizaje de la asignatura y que permiten, sin duda, lograr un aprendizaje significativo a partir de la ejercitación continuada de las mismas. (Figura 59). Es vital que estas herramientas sean de fácil uso y no

requieran mucho tiempo para su aprehensión, pues no son el objeto de estudio de la asignatura.

Figura 60. Diagrama de barras y tabla de resultados pregunta 12.



	Fue indispensable	Fue muy útil	Fue de apoyo	No fue relevante	Fue complicado	No contestó	TOTAL
Respuestas	**13**	**16**	**10**	**0**	**3**	**0**	**42**
Porcentaje	**30,95 %**	**38,10 %**	**23,81 %**	**0 %**	**7,14 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

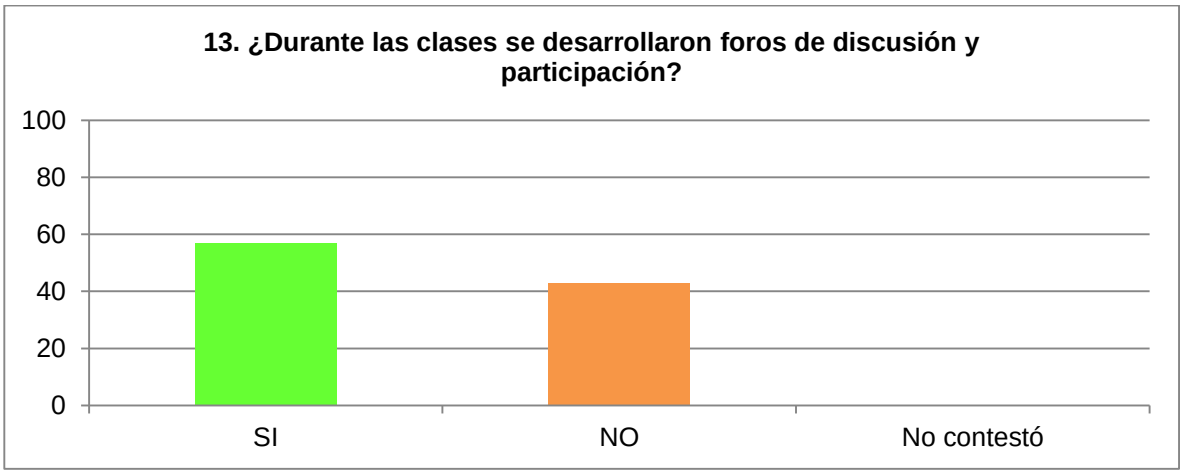
Opción 1: fue indispensable para el buen desarrollo de las actividades.
Opción 2: fue muy útil e interesante aprender las funcionalidades que ofrecen.
Opción 3: apoyó el desarrollo de las actividades pero no fueron fundamentales.
Opción 4: no fue relevante.
Opción 5: fue complicado y demandó más dedicación que la esperada.

Se aprecia una clara aceptación del uso de herramientas, donde un pequeño porcentaje solo objeta que fue complicado su uso (Ver Figura 60). Es fundamental que el docente sepa guiar la escogencia de las herramientas, basados en funcionalidad, eficiencia, amigabilidad, portabilidad, licencia libre, facilidad de instalación y buen soporte a los requerimientos de la asignatura.

6.3.3 Tercera parte (final): Foros y evaluación. Finalmente, se abordaron dos temas que nos permiten medir las habilidades desarrolladas por los estudiantes en el proceso de aprendizaje de la asignatura: los foros como herramienta de socialización de conocimiento y la evaluación que mide la evolución obtenida.

A continuación se muestra cada pregunta referente a estos temas y su análisis:

Figura 61. Diagrama de barras y tabla de resultados pregunta 13.



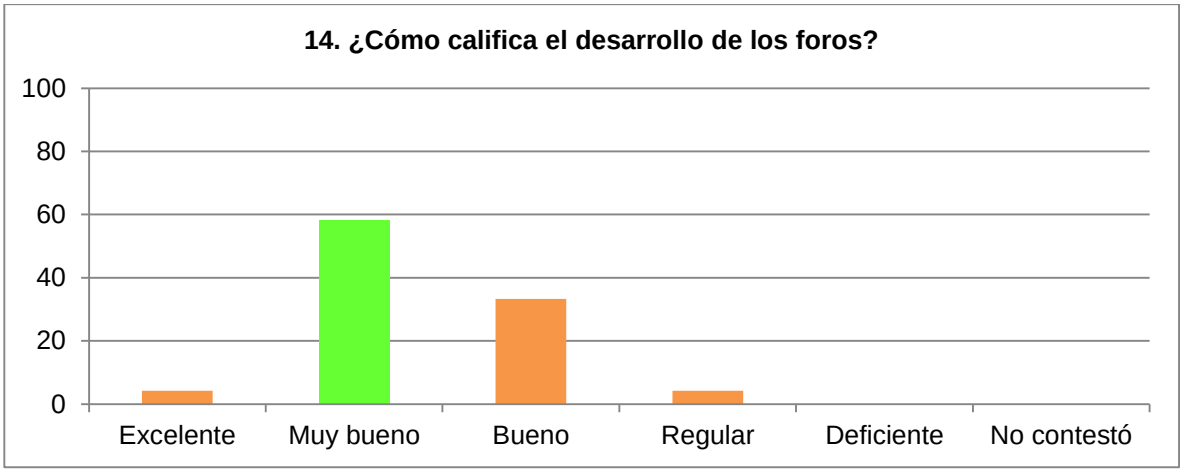
	SI	NO	No contestó	TOTAL
Respuestas	**24**	**18**	**0**	**42**
Porcentaje	**57,14 %**	**42,86 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

Más del 50% de los estudiantes encuestados confirman que se realizaron foros de discusión y participación sobre temas de interés de la asignatura, sin embargo, un alto porcentaje no los realizó. (Ver Figura 61). En esta actividad es fundamental que el docente tenga muy clara la forma de guiar a sus estudiantes, para evitar que se vuelva monótono, desordenado, ineficaz y en general, que no se pueda avanzar de la forma deseada. El perfil docente debe ser proactivo y dirigente para evitar desviaciones del tema. No obstante, bien orientado, un foro de discusión puede llegar a ser una actividad de mucho valor para el proceso de aprendizaje en el área.

Si la respuesta al ítem 13 fue afirmativa, favor contestar los numerales 14 y 15. De lo contrario, favor saltarse al punto N° 16.

Figura 62. Diagrama de barras y tabla de resultados pregunta 14.

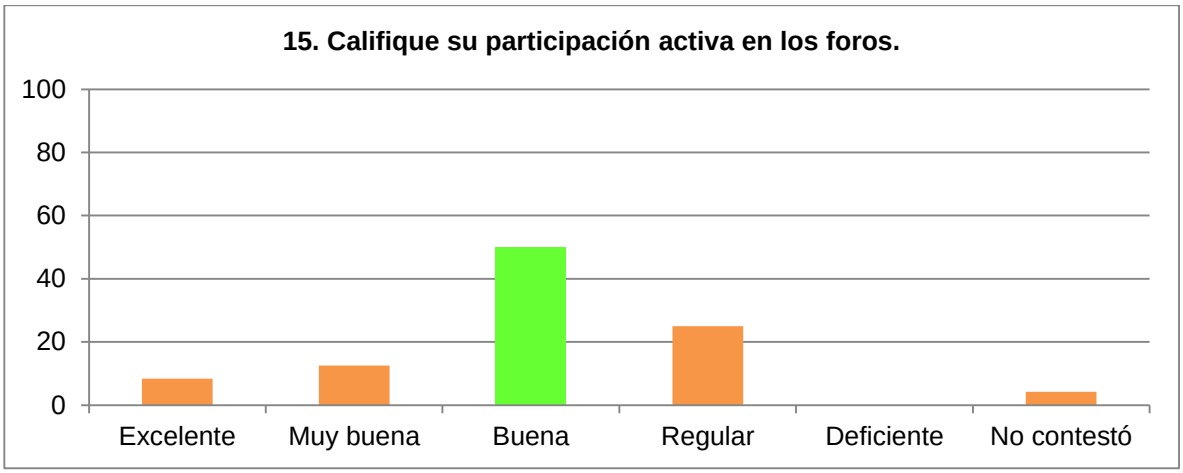


	Excelente	Muy bueno	Bueno	Regular	Deficiente	No contestó	TOTAL
Respuestas	**1**	**14**	**8**	**1**	**0**	**0**	**24**
Porcentaje	**4,17 %**	**58,33 %**	**33,33 %**	**4,17 %**	**42,86 %**	**0 %**	**100 %**

Fuente: Autor del proyecto.

Se aprecia una buena aceptación del desarrollo de los foros en clase (62,5 % de los estudiantes califica la aplicación de los foros de forma excelente o muy buena), ver Figura 62.

Figura 63. Diagrama de barras y tabla de resultados pregunta 15.



	Excelente	Muy buena	Buena	Regular	Deficiente	No contestó	TOTAL
Respuestas	**2**	**3**	**12**	**6**	**0**	**1**	**24**
Porcentaje	**8,33 %**	**12,5 %**	**50,00 %**	**25,00 %**	**0 %**	**4,17 %**	**100 %**

Fuente: Autor del proyecto.

Excelente: se sentía motivado por lo interesante de los temas tratados y la dirección dada por el docente.

Muy buena: participaba la mayoría de las veces y se mantuvo el interés y el dinamismo en los foros.

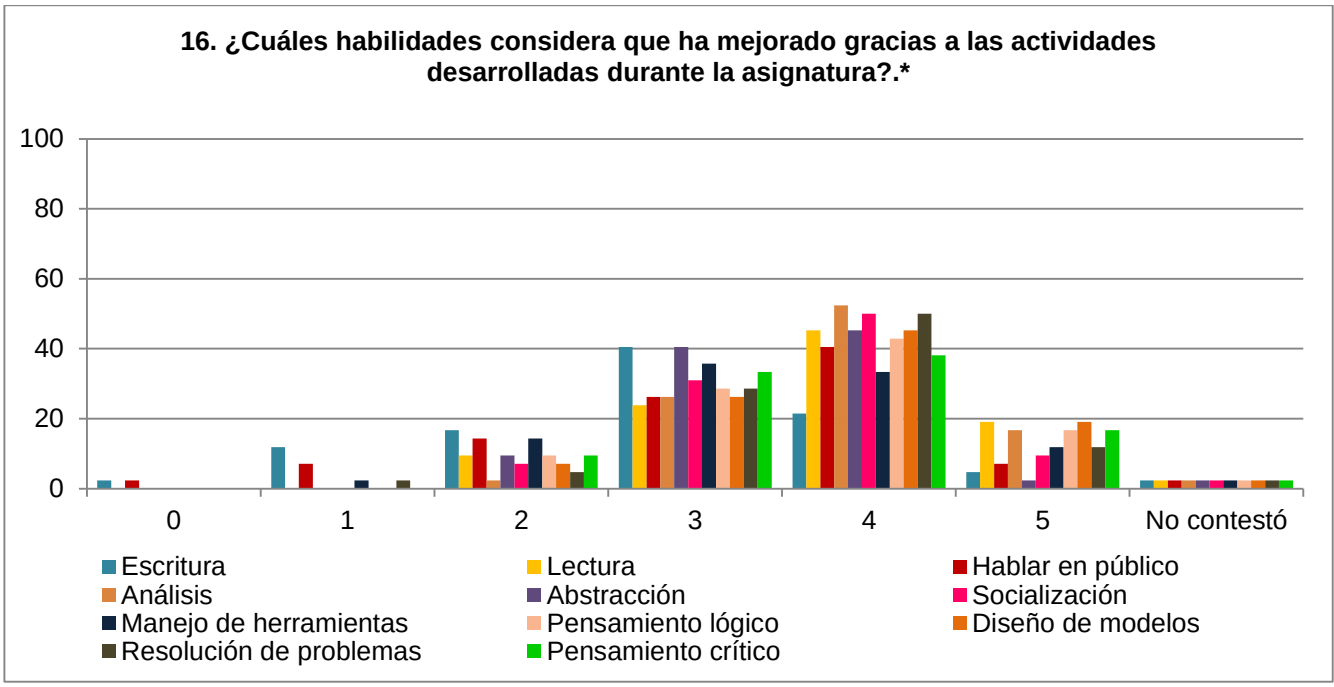
Buena: participaba en ocasiones, pero preparaba siempre los temas.

Regular: participaba pocas veces, no preparaba los temas con regularidad por falta de tiempo o motivación.

Deficiente: no participaba, se sentía desmotivado.

Se aprecia una buena participación del desarrollo de los foros en clase, aunque preocupa el 25% de estudiantes que indican un regular desempeño. (Figura 63). Esto se debe a la falta de motivación y tiempo, lo cual puede mejorarse notoriamente con un enfoque más estructurado del docente.

Figura 64. Diagrama de barras y tabla de resultados pregunta 16.



Puntuación	0 (Nulo)	1	2	3	4	5 (Excelente)	No contestó	TOTAL
Escritura	1 (2,38%)	5 (11,9%)	7 (16,67%)	17 (40,48%)	9 (21,43%)	2 (4,76%)	1 (2,38%)	
Lectura	0 (0%)	0 (0%)	4 (9,52%)	10 (23,81%)	19 (45,24%)	8 (19,05%)	1 (2,38%)	
Hablar en público	1 (2,38%)	3 (7,14%)	6 (14,29%)	11 (26,19%)	17 (40,48%)	3 (7,14%)	1 (2,38%)	
Análisis	0 (0%)	0 (0%)	1 (2,38%)	11 (26,19%)	22 (52,38%)	7 (16,67%)	1 (2,38%)	
Abstracción	0 (0%)	0 (0%)	4 (9,52%)	17 (40,48%)	19 (45,24%)	1 (2,38%)	1 (2,38%)	
Socialización	0 (0%)	0 (0%)	3 (7,14%)	13 (30,95%)	21 (50%)	4 (9,52%)	1 (2,38%)	**42 100 %**
Manejo de herramientas	0 (0%)	1 (2,38%)	6 (14,29%)	15 (35,71%)	14 (33,33%)	5 (11,9%)	1 (2,38%)	
Pensamiento lógico	0 (0%)	0 (0%)	4 (9,52%)	12 (28,57%)	18 (42,86%)	7 (16,67%)	1 (2,38%)	
Diseño de modelos	0 (0%)	0 (0%)	3 (7,14%)	11 (26,19%)	19 (45,24%)	8 (19,05%)	1 (2,38%)	
Resolución de problemas	0 (0%)	1 (2,38%)	2 (4,76%)	12 (28,57%)	21 (50%)	5 (11,9%)	1 (2,38%)	
Pensamiento crítico	0 (0%)	0 (0%)	4 (9,52%)	14 (33,33%)	16 (38,1%)	7 (16,67%)	1 (2,38%)	

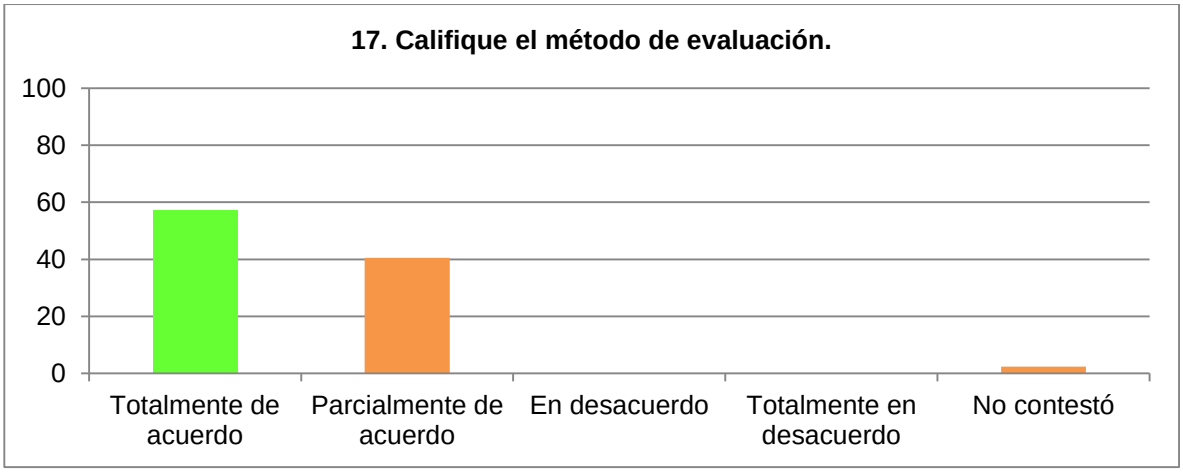
Fuente: Autor del proyecto.

***Puede seleccionar más de una opción. Seleccione al frente el nivel de ejercitación durante las actividades de la clase (siendo 0 cuando no se ejercita y 5 cuando se ejercite todo el tiempo).**

En la tabla anexa a la Figura 64 se resalta la opción más escogida en cada ítem de la pregunta 16. Cabe notar, con gran satisfacción, que la mayoría de las opciones tienen un nivel de 4 en ejercitación, siendo 5 el más alto. Esto nos conduce a pensar que se está haciendo un buen trabajo en pro del desarrollo de estas habilidades, pero falta especialmente un esfuerzo más en el ejercicio de la escritura y del pensamiento crítico. También se desea enfatizar que en el curso dirigido por el docente director de este proyecto se está dando un valor agregado a las lecturas y es el hecho que la mayoría del material escrito que se proporciona y propone se encuentra en inglés, promoviendo la ejercitación y la comprensión de su importancia.

Dentro de la guía SWEBOK se resalta el desarrollo y ejercitación de las habilidades dentro del ejercicio de la práctica profesional en Ingeniería del Software, por tanto, este ítem es de gran valor.

Figura 65. Diagrama de barras y tabla de resultados pregunta 17.



	Totalmente de acuerdo	Parcialmente de acuerdo	En desacuerdo	Totalmente en desacuerdo	No contestó	TOTAL
Respuestas	**24**	**17**	**0**	**0**	**1**	**42**
Porcentaje	**57,14 %**	**40,48 %**	**0 %**	**0 %**	**2,38 %**	**100 %**

Fuente: Autor del proyecto.

Totalmente de acuerdo: apropiado y completo.

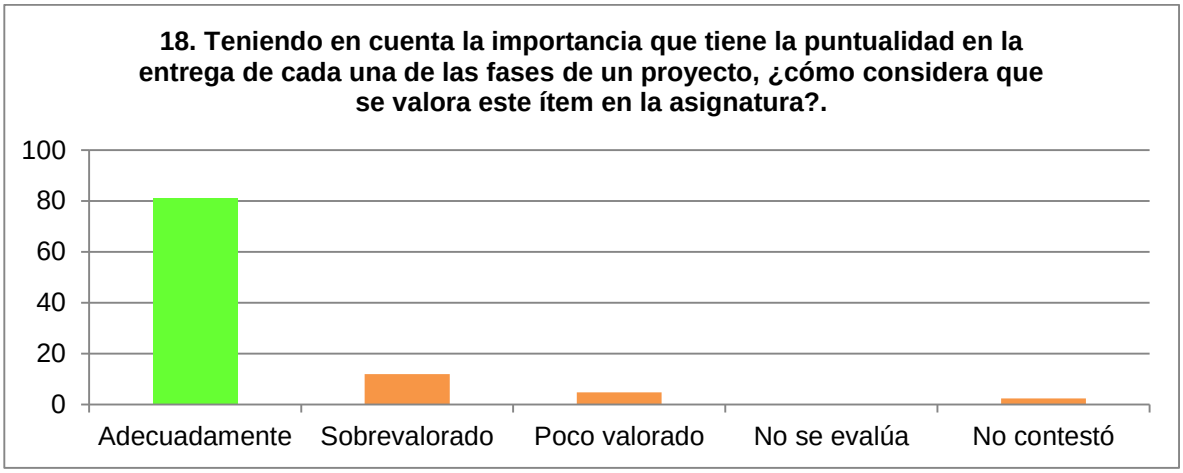
Parcialmente de acuerdo: no evalúa todo el proceso de aprendizaje.

En desacuerdo: no corresponde con lo enseñado.

Totalmente en desacuerdo: es muy fuerte y complejo.

Se destaca que ningún estudiante está en desacuerdo con la forma de evaluación, estando más del 50% totalmente de acuerdo. (Ver Figura 65). Esta pregunta se incluyó en la encuesta porque se deseaba comprobar el nivel de satisfacción en cuanto la exigencia, especialmente, en el compromiso en las entregas de forma puntual. En los últimos semestres se ha visto la necesidad de promover la puntualidad en las entregas penalizando con calificación cero (0) en el ítem de cada tarea, aunque se le permite socializar su trabajo. Se aprecia con satisfacción que los estudiantes han entendido la importancia de este valor y no lo ven de forma negativa.

Figura 66. Diagrama de barras y tabla de resultados pregunta 18.



	Adecuadamente	Sobrevalorado	Poco valorado	No se evalúa	No contestó	TOTAL
Respuestas	**34**	**5**	**2**	**0**	**1**	**42**
Porcentaje	**80,95 %**	**11,9 %**	**4,76 %**	**0 %**	**2,38 %**	**100 %**

Fuente: Autor del proyecto.

7 MODELO DE ENSEÑANZA / APRENDIZAJE DE LA INGENIERÍA DEL SOFTWARE I BASADO EN TIC

Apoyándonos en la idea de que el fruto y motor principal de la educación es lograr el aprendizaje significativo en cada área estudiada y, que la educación superior busca la formación de profesionales competentes, en el presente estudio se promueve la presentación formal de una metodología en la enseñanza/aprendizaje de Ingeniería del Software I, que permita un mayor aprovechamiento de las herramientas de apoyo brindadas y que ofrezca respaldo al proceso educativo en este campo, en la Universidad Industrial de Santander (UIS), como parte del programa académico de Ingeniería de Sistemas.

La principal motivación que se desea incentivar es que los estudiantes reconozcan realmente que los contenidos estudiados en la asignatura y las prácticas realizadas en el curso son un hito fundamental requerido y que les serán efectivamente útiles en el ejercicio de su profesión.

La asignatura de Ingeniería del Software I se imparte en la UIS en el cuarto año (séptimo semestre), cuando los estudiantes ya han recibido formación básica en lenguajes de programación, en programación orientada a objetos, en bases y estructura de datos, en sistemas de información, entre otros, proporcionándoles un fundamento teórico – práctico que falta formalizar y orientar a las buenas prácticas de desarrollo de software.

Por todo lo anterior, esta metodología estará centrada en el estudiante (aprendizaje activo), basado en competencias, quien jugará el principal rol en el proceso y quien deberá administrar las técnicas, actividades y material de apoyo que se proporcionarán durante el desarrollo del curso, en pro de lograr su propio aprendizaje.

7.1.1 Contenido del curso de Ingeniería del Software I. Como se ha mencionado previamente, se desean formalizar los contenidos y en general, las actividades desarrolladas en la asignatura de Ingeniería del Software I. Como punto de apoyo en el primer punto nombrado, se referencia el SWEBOK (Cuerpo del Conocimiento de Ingeniería del Software), porque sus objetivos son claramente compatibles con lo anterior: [Lizano, 2010]

- Promover una visión consistente de la Ingeniería del Software a escala mundial.
- Clarificar el lugar y límites de la Ingeniería del Software respecto a otras disciplinas relacionadas.
- Caracterización de los contenidos de la Ingeniería de Software.
- Proveer acceso a los tópicos de la Ingeniería de Software.
- Proveer los fundamentos para desarrollo de currículums y material de certificación individual.

Además, este proyecto documentado del SWEBOK es liderado por la IEEE Computer Society, quien le da un amplio respaldo de estandarización y calidad, brindando el soporte sólido requerido para brindar confianza en el contenido.

0. Introducción a la Ingeniería de Software (IdS)
0.1. Origen y situación actual de la IdS; código de ética; Definición
0.2. Ciclos de vida del proceso de software
1. Requerimientos de Software: definición y clasificación.
1.1. Obtención de requerimientos: fuentes y técnicas.
1.2. Modelo de Casos de Uso: análisis de requerimientos.
1.3. Modelo de Clases del Dominio.
1.4. Uso de Prototipos para especificación de requerimientos software
2. Análisis

2.1. Arquitectura MVC
2.2. Refactorización
2.3. Modelos de comportamiento dinámico
3. Diseño de Software
3.1. Fundamentos del Diseño
3.2. Conceptos clave en el Diseño de Software.
3.3. Diseño de interfaces externas
4. Construcción de Software: generalidades
5. Pruebas de Software: generalidades
6. Mantenimiento de Software: generalidades.

7.1.2 Modelo de enseñanza en el curso de Ingeniería del Software I. Según las estrategias de enseñanza y aprendizaje descritas, se aprecia la necesidad de combinarlas y enriquecer el trabajo del curso con diferentes técnicas, tal y como se describe a continuación:

- En la primera fase del curso, donde se abordan temas generales y más teóricos como la introducción a la Ingeniería del Software y los ciclos de vida de desarrollo de software se promueve hacer uso de estrategias de organización, elaboración y exposición, mediante lecturas orientadas (propuestas también por Pollock [Pollock, 2001]), que generen discusión en un foro, donde por grupos de estudiantes se asignan previamente para discutir en clase. Los estudiantes a cargo, deberán formular preguntas que se harán en el día del foro, mientras que el rol del docente en las presentaciones será aclarar o refutar temas cuando fuera necesario. Además, el grupo a cargo deberá entregar un resumen técnico de la lectura (terminología desconocida, resumen, palabras críticas) y calificar las respuestas de sus compañeros. Cabe aclarar que todo el grupo hará la lectura, pero el grupo a cargo dirigirá la discusión.

Esta técnica favorece la atención de todo el grupo, se ejercitan las habilidades comunicativas, se va abordando la temática de forma dinámica y además los estudiantes van participando activamente en el proceso.

La duración de esta fase se propone que sea de **2 semanas**, teniendo en cuenta que todos los estudiantes del curso hagan parte, en al menos una ocasión, del grupo de los estudiantes a cargo, cubriendo toda la fase inicial (Unidad cero).

- En la segunda fase del curso, ya cuando se incursiona en el tema de los Requerimientos de Software, es necesario realizar un estudio analítico de los estándares en este campo, como lo son el SWEBOK (Guía del Cuerpo del Conocimiento de la Ingeniería del Software) y el SEEK (Conocimiento en Educación de la Ingeniería del Software), desarrollados por el IEEE en un esfuerzo por profesionalizar la Ingeniería del Software y para apoyar la formulación e implementación de currículos, programas de actualización y certificación, códigos de ética, entre otros, formalizando la enseñanza de esta asignatura. [Lizano, 2010].

  A partir de esta etapa del curso, la cual puede durar **12 semanas**, se da inicio al desarrollo de los modelos de especificación de Requerimientos, es decir, los estudiantes, en equipos de trabajo de máximo 3 personas, escogerán un proyecto de su interés, SOFTWARE A LA MEDIDA de las necesidades de una empresa particular o SOFTWARE GENÉRICO, aplicable a un nicho de negocio o cluster de empresas (de preferencia con aplicación en el mundo laboral), para desarrollar los diferentes artefactos propuestos en clase.

  Los equipos tendrán un máximo de tres personas, porque es importante incentivar el trabajo grupal y cooperativo, como se analizó en el capítulo 4, porque en pequeños grupos se establece mejor comunicación, porque por el

detalle de los artefactos, es importante poder avanzar a un ritmo apropiado, lo cual con uno o dos integrantes puede llegar a ser menos ágil este proceso y la dimensión alcanzada del proyecto sería muy básica.

La definición de roles en los equipos es una tarea complicada, debido a la falta de experiencia de los estudiantes en este campo; además, partimos en que aunque es factible que los grupos se conformen con individuos que se conocen en el ámbito educativo, por haber compartido otros cursos previamente, también es evidente que el desarrollo de las temáticas del curso son nuevas y por tanto, los perfiles no han sido compartidos ni ejecutados con anterioridad, de la misma forma, es decir, aplicando un proceso estructurado y haciendo uso de las buenas prácticas de la IS. Además, aunque algunos autores [Casallas, 2002], [Galeana, 2007] proponen el uso de juegos de roles, en el caso planteado en esta investigación no es conveniente porque no se dan las condiciones expuestas por ellos: ser un curso avanzado, trabajar en equipos grandes (4 – 6 estudiantes) y trabajar proyectos iguales o equivalentes.

Los artefactos generados en el curso deben cumplir con las características establecidas en los estándares: ser realistas, completos, consistentes, no ambiguos, correctos, legibles y bien definidos. Para esto, el docente será el coordinador general, desarrollando artefactos y proponiendo patrones de solución de problemas, para que los estudiantes, a través de talleres prácticos se ejerciten en este ámbito.

En este caso se aprecia que se evalúa el progreso de cada grupo, y por ende de los estudiantes en particular, en las entregas incrementales de las tareas y artefactos. Así, como la enseñanza de los fundamentos teóricos que se van proporcionando a medida que avanza el semestre y se van requiriendo los conceptos para ser llevados a la práctica en el caso de aplicación. Esta metodología, de entrega incremental en el marco del aprendizaje basado en

proyectos, es apoyada por Alistair Cockburn en su investigación sobre el tema [Cockburn, 2006].

Un mecanismo para controlar y orientar al estudiante en su disciplina es el hecho de evaluar su puntualidad en la entrega de tareas. Ellos deben ser diligentes y proactivos en la toma de decisiones y en el desarrollo general de las actividades. Por tanto, si entrega su trabajo después de la fecha destinada para ello, se penalizará con no recibir valoración en dicha tarea, sin embargo, el grupo tendrá la oportunidad de socializar el trabajo realizado. Como existe realimentación en clase y por parte del docente, cada grupo tendrá la oportunidad de mejorar su proceso en la próxima entrega.

En la tabla 15 se muestra la plantilla correspondiente a la primera tarea, que involucra la segunda fase y deberá estar lista en la cuarta semana. Esta plantilla deberá llevarse a cada clase, para ir desarrollándola paulatinamente, con apoyo de las explicaciones y los talleres que se realizan durante cada jornada. Esta plantilla ha sido desarrolla y mejorada por el docente director del presente proyecto y apoyado en la literatura, como se detalló en el capítulo 4.

Aunque el docente es quien guía el desarrollo de la temática, se evaluará el progreso de los estudiantes, es decir, se tendrá en cuenta la evolución de la plantilla de cada equipo, de modo que quienes recaigan en errores aclarados en clase, serán penalizados. Es importante que al final, luego de entregar formalmente esta tarea se realice una socialización evaluable, también, de los ítems desarrollados.

A continuación se definen los parámetros de evaluación de las tareas (valoración de la competencia escrita, es decir en el llenado de cada plantilla):

a) 20%: *completitud* – realización de todos los modelos y artefactos solicitados.

b) 20%: *corrección* – consistencia entre modelos y artefactos con el 'mundo observado'.
c) 20%: *consistencia* – que todos los modelos y artefactos describan el mismo sistema.
d) 20%: *legibilidad* – claridad de los artefactos; que sean comprensibles por personas ajenas al equipo de desarrollo.
e) 20%: *factibilidad* – que el sistema que se pretende sea viable con los recursos disponibles.

Tabla 15. Tarea 1: Descripción del sistema y modelo preliminar del negocio.

UNIVERSIDAD INDUSTRIAL DE SANTANDER Escuela de Ingeniería de Sistemas e Informática	Fecha límite de recepción de tareas: **DD-MM-AA (HORA)** a: <frojas@uis.edu.co>
INGENIERÍA DE SOFTWARE 1 – 22969	Tarea 1 - **Requerimientos-C**[4]
CASO DE APLICACIÓN	
RESPONSABLES	
REQUIREMENTS – C [BUSSINESS PERSPECTIVE]	
DESCRIPCIÓN DEL SISTEMA DESEADO	
INTERESADOS	
MODELO DE EVENTOS DE NEGOCIO	

Fuente: Director del proyecto.

[4] Requerimientos del cliente (-C).

Tabla 15. Tarea 1: Descripción del sistema y modelo preliminar del negocio. (Continuación)

TABLA DE EVENTOS DE NEGOCIO					
No	**Evento**	**Entrada al Sistema**	**Actor que provee la Entrada**	**Salida del Sistema**	**Actor que recibe la Salida**
LISTA DE PROBLEMAS ASOCIADOS A LOS EVENTOS					
LISTA DE OPORTUNIDADES ASOCIADAS A LOS EVENTOS					
LISTA DE OBJETIVOS (Con Factor-X y ordenados por Prioridad)					
LISTA DE PRECONDICIONES Y LIMITACIONES DEL SISTEMA					

Fuente: Director del proyecto.

La segunda tarea (Tabla 16), consiste en el desarrollo preliminar de los modelos del desarrollador (casos de uso y diagrama de clases). Esta tarea tendrá las mismas valoraciones de la tarea anterior, incluyendo la penalidad por entrega después de la fecha pactada. Estos modelos son referenciados en las guías del SWEBOK y del SEEK.

Tabla 16. Plantilla tarea 2: Modelo de Requerimientos -D.

UNIVERSIDAD INDUSTRIAL DE SANTANDER Escuela de Ingeniería de Sistemas e Informática	Fecha límite de recepción de tareas: **dd/mm/aa (Hora)** a: <frojas@uis.edu.co>
INGENIERÍA DE SOFTWARE 1 – 22969	Tarea 2 **Modelo de Requerimientos-D**[5]

CASO DE APLICACIÓN	
RESPONSABLES	

REQUIREMENTS – D [DEVELOPERS PERSPECTIVE]

DESCRIPCION DEL SISTEMA DESEADO

MODELO DE CASOS DE USO

DIAGRAMA DE CASOS DE USO

ESPECIFICACIONES DE CASOS DE USO

ACTOR(ES)		
Actor		
Casos de Uso		
Tipo		
Descripción		
CASOS DE USO		
Caso de Uso		
Actor(es)		
Propósito		
Descripción		
Precondición(es)		
Flujo Principal	**Acciones de Actor(es)**	**Respuestas del Sistema**
Sub-flujos		
Poscondición(es)		

Fuente: Director del proyecto.

[5] Requerimientos del desarrollador (-D)

Tabla 16. Plantilla tarea 2: Modelo de Requerimientos -D. (Continuación)

PROTOTIPO INICIAL - *SKETCHES*
MODELO DEL DOMINIO DE APLICACIÓN [o del PROBLEMA]
DESCRIPCION DEL SISTEMA SUBRAYADO
LISTA DE CONCEPTOS INVOLUCRADOS EN EL SISTEMA
DIAGRAMA DE CLASES (Con roles y cardinalidad)
DICCIONARIO DE CLASES

Clase	**Descripción**

Fuente: Director del proyecto.

El desarrollo de la plantilla N° 2 llevará 4 semanas, que incluyen la presentación y socialización.

Al finalizar la socialización de la plantilla N°2 se ejecutará el primer examen o examen intermedio, que valora la fundamentación teórica abarcada en la primera parte del curso (definición, historia y características de la IS, ciclos de vida del software, el proceso de formalización de los requerimientos de software, modelado de casos de uso, diagrama de clases, de actividades, de secuencia, de estado). Esta evaluación tendrá una valoración del 20% de la nota final y se llevará a cabo alrededor de la semana 9.

La tercera y última tarea, consiste en el desarrollo del modelo de análisis. Se profundiza aún más la etapa de análisis de los requerimientos, dando cabida a la arquitectura de clases MVC: Modelo – Vista - Controlador (objetos de borde, de entidad, de control y diagramas de clases, de secuencia y de estados). Finalmente, se solicita el diagrama de navegación de ventanas para dejar un diseño inicial del prototipo y al final del curso entregar el prototipo funcional (Tabla 17). Esta labor es un reflejo de lo propuesto por muchos autores, entre los que se encuentran Bernd Bruegge y Allen Dutoit en [Bruegge, 2004]]. Ellos plantean que modelar el sistema con objetos de entidad, borde y control tiene muchas ventajas, tales como: diferenciación de objetos y funcionalidades.

Tabla 17. Plantilla tarea 3: Modelo de Análisis.

UNIVERSIDAD INDUSTRIAL DE SANTANDER Escuela de Ingeniería de Sistemas e Informática	Enviar antes de: **dd-mm-aaaa; hora.** a: <frojas@uis.edu.co>
INGENIERÍA DE SOFTWARE 1 – 22969	Tarea 3 **Modelo de Análisis**
CASO DE APLICACIÓN	
RESPONSABLES	

Fuente: Director del proyecto.

Tabla 17. Plantilla tarea 3: Modelo de Análisis. (Continuación)

0. Introducción

MODELO DE ANÁLISIS
0. DIAGRAMA DE CASOS DE USO (Refinado)

1. Arquitectura de clases

1.1 Objetos de borde
Caso de uso 1
Caso de uso 2
Caso de uso 3
Caso de uso 4

1.2 Objetos de entidad
Caso de uso 1
Caso de uso 2
Caso de uso 3
Caso de uso 4

1.3 Objetos de Control
Caso de uso 1
Caso de uso 2
Caso de uso 3
Caso de uso 4

Fuente: Director del proyecto.

Tabla 17. Plantilla tarea 3: Modelo de Análisis. (Continuación)

1.4 Clases (Entidad+Borde+Control) para cada caso de uso

Caso de uso 1
Caso de uso 2
Caso de uso 3
Caso de uso 4

2. Diagramas de secuencia para cada caso de uso (deben incluir Objetos de tipo Entidad y hacer uso de notación UML con íconos)

DIAGRAMA DE SECUENCIA: CASO DE USO 1
DIAGRAMA DE SECUENCIA: CASO DE USO 2
DIAGRAMA DE SECUENCIA: CASO DE USO 3
DIAGRAMA DE SECUENCIA: CASO DE USO 4

3. ESPECIFICACIONES DE CASOS DE USO DE ANÁLISIS	
Caso de Uso:	
Condición inicial:	
Flujo Principal:	
Condición de salida:	

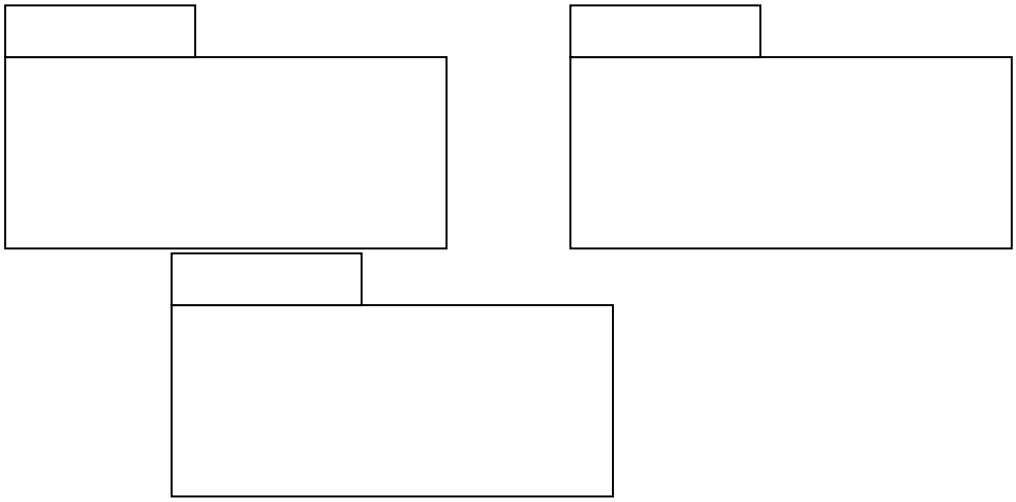
4. DIAGRAMA DE CLASES REFINADO

Fuente: Director del proyecto.

Tabla 17. Plantilla tarea 3: Modelo de Análisis. (Continuación)

5. Diccionario de clases agrupadas por módulos

Módulos:



Módulo	Descripción

6. DIAGRAMA DE ESTADOS

7. DISEÑO DE PROTOTIPO
DIAGRAMA DE NAVEGACIÓN DE VENTANAS
DISPOSICIÓN DE VENTANAS

Fuente: Director del proyecto.

Cabe anotar que el llenado de las plantillas se va dando a medida que se va estudiando el Cuerpo del conocimiento de la IS, y a medida que los estudiantes van completando sus modelos según el proyecto escogido. Sin embargo, la entrega se establece en una fecha dada, según las semanas como se ha venido describiendo.

7.1.3 Socialización del prototipo funcional. Al finalizar la entrega de la tarea 3, se hará la evaluación final. Acto seguido se dará inicio a la socialización y entrega del prototipo funcional correspondiente al proyecto desarrollado durante todo el semestre por los grupos de trabajo. Los principales criterios que se valorarán serán:

- ✓ Organización de ventanas y elementos.
- ✓ Claridad en la presentación y solicitud de información (Legibilidad).
- ✓ Validación de datos (Eficacia).
- ✓ Conformidad con los modelos desarrollados.
- ✓ Completitud.

7.1.4 Herramientas de modelado. Según lo analizado en el capítulo anterior se llegó a la conclusión de motivar el uso de StarUML como aplicación de soporte para el modelado del sistema en la fase de Requerimientos, principalmente por las siguientes características:

- Soporta la mayoría de los diagramas especificados en UML (Clases, Estados, Casos de Uso, Actividad, Colaboración, Desarrollo, Secuencia, Componentes, Estructura compuesta).
- Permite exportar imágenes de diagramas a formatos ampliamente difundidos (JPG, BMP, EMF, WMF).
- Tiene habilidad para generar código fuente desde los diagramas UML (Java, C++, C#).
- Soporta ingeniería inversa.
- Permite crear documentación en aplicaciones de Microsoft Office, tales como Word, Excel y PowerPoint.
- Presenta verificación de modelos.
- Es distribuido bajo licencia GNU GLP (General Public License).

7.1.5 Herramientas de desarrollo. A través de lo analizado durante el capítulo anterior se concluyó que una excelente alternativa para el desarrollo de un sistema software funcional que de solución al problema planteado por el proyecto definido por los estudiantes es la herramienta NETBEANS, que se destaca por las siguientes características:

- Es intuitivo y es rápido para encontrar las funcionalidades en la barra de tareas.
- Presenta gran variedad de documentación y soporte en su sitio oficial (http://netbeans.org/).
- Permite autocompletar funciones.
- Soporta muchos lenguajes para Java (C/C++, XML y HTML, PHP, Groovy, Javadoc, JavaScript y JSP.
- Es organizado en la estructura de archivos.
- Presenta validación de código.
- Es de uso libre.
- Es fácil de entender un proyecto, debido a la pulida organización de la estructura del código.

7.2 RESUMEN DEL MODELO DEL CURSO DE INGENIERÍA DEL SOFTWARE I PROPUESTO

- **Enfoque:** centrado en el estudiante (Aprendizaje Activo). Aprendizaje basado en Competencias.

- **Modelo pedagógico - Teoría de aprendizaje:** Construccionismo.

- **Estrategia de enseñanza - aprendizaje:** Aprendizaje Basado en Proyectos, por ser el más completo, involucrando otros tipos de aprendizaje (cooperativo e

individual), con miras de enriquecer el proceso educativo con diversas dinámicas y técnicas para obtener el Aprendizaje Significativo.

- **Objetivos del curso:** Profundizar en:
  - Requerimientos de software: obtención, análisis, especificación y validación.
  - Desarrollo de modelos de análisis y diseño, usando estándares.
  - Trabajo en equipo en el desarrollo de un proyecto software, focalizando en el uso de valores como disciplina, colaboración, buena comunicación, toma de decisiones, responsabilidad y liderazgo.
  - Ejercicio de habilidades comunicativas.

  Todo lo anterior, en pro de lograr el objetivo primordial que busca la Ingeniería del Software: realización de software de calidad.

- **Organización de trabajo:** se promueve el trabajo en equipos de máximo 3 integrantes, sin formulación de roles específicos. Se motiva a que los estudiantes se esfuercen por igual en lograr las metas y en el desarrollo de los artefactos.

- **Modelo de enseñanza:** se motiva a realizar la ejecución de minilecturas con los temas introductorios del contenido temático de la asignatura, con la participación activa de los estudiantes y orientando la discusión para evitar incongruencias y desvíos. Además, clases magistrales, con apoyo de talleres en la especificación teórica de la fase de Requerimientos de Software, con apoyo del SWEBOK y el SEEK.

- **Forma de evaluación:** se dará énfasis en el uso de diferentes tipos de competencias comunicativas, matemáticas, y cognoscitivas.

  a) Minilecturas, talleres y participación en clase: 10%.

b) **Tareas:** la primera tendrá una valoración del 10%, la segunda del 10% y la tercera del 10%.
c) **Prototipo funcional:** tendrá una valoración del 20%.
d) **Evaluaciones:** cada una tendrá una valoración del 20% (total 40%).

- **Herramientas de apoyo:**
  - ✓ **Modelado:** StarUML.
  - ✓ **Desarrollo:** NetBeans.
  - ✓ **Comunicación docente y compañeros, publicación de documentos de interés:** MeiWeb.

- **Seguimiento por semanas:** en la tabla 18 se presenta un resumen de las actividades a realizar por semanas.

Tabla 18. Seguimiento por semanas de las actividades del curso.

Semana	Actividad desarrollada
1-2	Minilecturas guiadas sobre introducción a la IS y ciclos de vida de desarrollo de software. Socialización mediante foros.
3-4	SWEBOK y SEEK. Desarrollo de la tarea 1 (Requerimientos –C).
4-8	SWEBOK y SEEK. Desarrollo de la tarea 2 (Requerimientos –D).
9	Evaluación intermedia.
10 - 14	SWEBOK y SEEK. Desarrollo de la tarea 3 (Modelo de análisis).
15	Evaluación final.
16	Socialización prototipo functional.

Fuente: Autor del proyecto

- **Bibliografía sugerida:**
  - ✓ Bruegge, B., Dutoit, A. Object –Oriented Software Engineering – 2E. Prentice-Hall, 2004.
  - ✓ Stumpf, R., Teague, L. Object –Oriented System Analysis. Prentice-Hall, 2005.
  - ✓ Weitzenfeld, A. Ingeniería de software con UML y Java. Thompson, 2005.

- ✓ Braude, E. Ingeniería de software. Una perspectiva orientada a objetos. AlfaOmega, 2000.
- ✓ Pressman, R. Ingeniería del Software – 6E. McGraw-Hill, 2005.
- ✓ Sommerville, I. Ingeniería del Software – 7e. Prentice-Hall, 2005.
- ✓ Pfleeger, S. Ingeniería de software. Teoría y Práctica. Prentice-Hall. 2002.
- ✓ Ruble, D. Análisis y Diseño Práctico de Sistemas Cliente/Servidor con GUI. Prentice-Hall, 1998.
- ✓ Kendall & Kendall. Análisis y Diseño de Sistemas. Prentice-Hall, 2005.

Además de las lecturas sugeridas que se presentarán especialmente al inicio del curso.

8 CONCLUSIONES, CONTRIBUCIONES Y TRABAJOS FUTUROS

8.1 CONCLUSIONES

A partir del desarrollo de la investigación, se puede concluir que

1. Dentro del proceso de elaboración de software, el proceso que representa el mayor riesgo en la búsqueda de la calidad es la etapa de Requerimientos. Por tanto, es fundamental que el ingeniero tenga las herramientas conceptuales y prácticas para enfrentarse a la industria con resultados exitosos, lo cual depende, sin duda, de su preparación académica. Debido a esto se requiere que la asignatura de Ingeniería del Software incluya un componente bien establecido sobre este tópico específico.
2. Un curso de Ingeniería del Software debe promulgar la ejercitación de valores, especialmente los establecidos en el código de ética[6] de la profesión (Ver Anexo 2): "(a) Sociedad – actuar de forma congruente con el interés social, (b) Cliente – actuarán de forma que concilien con los mejores intereses del cliente, (c) Producto – asegurarán que sus productos cumplen con los estándares profesionales más altos, (d) Juicio – mantendrán la integridad e independencia en su juicio profesional, (e) Administración – promoverán y se suscribirán a un enfoque ético en la administración del desarrollo y mantenimiento de software, (f) Profesión – incrementarán la integridad y reputación de la profesión congruentemente con el interés social, (g) Colegas – apoyarán y serán justos con sus colegas y (h) Personal – participarán toda su vida en el aprendizaje relacionado con la práctica de su profesión y promoverán un enfoque ético en la práctica de la profesión"; además debe tener una fundamentación teórica estandarizada (SWEBOK, SEEK) y promover el desarrollo de competencias (abstracción, razonamiento lógico, comunicativas, entre otras) y buenas

[6] Tomado de http://www.acm.org/about/se-code-s

prácticas de ingeniería del software, con el propósito de educar profesionales integrales.

3. Se logró establecer, a través del análisis de diversas investigaciones alrededor del mundo sobre la enseñanza de Ingeniería del Software, que el trabajo que se ha venido consolidando en la UIS converge con los enfoques revisados. No obstante, algunos de ellos fueron fundamentales para aportar nuevas ideas y estrategias que enriquecen el proceso educativo, tales como las minilecturas y el enfoque de aprendizaje basado en competencias (saber hacer en contexto).
4. Las TIC son fundamentales para apoyar cualquier proceso educativo, y aún más si se trata de un curso de ingeniería, pues se enfatiza en el uso de la tecnología. Como resultado de la investigación se lograron analizar varias herramientas computacionales para acompañar y fortalecer el aprendizaje de los estudiantes en el área. Estas son:
    - Tecnologías de Información y Comunicación: MeiWeb.
    - Herramientas de modelado: StarUML.
    - Herramientas de desarrollo: NetBeans.

    Los parámetros usados para definir las herramientas computacionales fueron [González, 2009], [López, 2012]:
    - Funcionalidad
    - Estabilidad
    - Licenciamiento
    - Popularidad
    - Usabilidad
    - Portabilidad

5. Se realizó una encuesta a estudiantes que ya cursaron Ingeniería del Software I y se encontró que el proceso de aprendizaje del área requiere

diversas estrategias para consolidarse con éxito. Dentro de los principales análisis obtenidos a partir de la encuesta se encuentran:

- Es fundamental que se desarrolle un proyecto de software en el curso para afianzar los conceptos adquiridos.
- Los estudiantes prefieren (68,57 %) que los proyectos a desarrollar en el curso sean de índole real, es decir sean el resultado de una necesidad específica de la industria y no producto de la literatura o ajeno a sus intereses.
- Se apoya la idea de trabajar en equipos pequeños, de 2 ó 3 integrantes, pues se logra mayor comunicación y entendimiento entre sus integrantes, logrando alcanzar en gran medida las metas establecidas.
- Es importante inculcar las entregas paulatinas, para ir descubriendo debilidades y fortalezas en el avance de los alumnos y en el desarrollo del proyecto.
- Los estudiantes, aunque puede significar mayor esfuerzo, apoyan las lecturas complementarias y la socialización en clase, pues les permite ejercitar sus competencias.
- El uso de herramientas computacionales de modelado y desarrollo son necesarias para el desarrollo del proyecto de clase; por tanto se sugieren aquellas que sean de fácil entendimiento, con excelente funcionalidad, portables, libres y estables.
- Es necesario fortalecer el ejercicio de las competencias de escritura y manejo de herramientas (esta última se mejora con lo apreciado en el ítem anterior).
- Los estudiantes aprueban que se inculque la puntualidad como valor fundamental del curso.
- Los demás valores evaluados en los modelos y proyecto en general del curso son: completitud, corrección, consistencia, legibilidad y factibilidad.

8.2 CONTRIBUCIONES

Modelo de Enseñanza – Aprendizaje de Ingeniería del software

Aunque la asignatura ha sido impartida en la UIS por varios años, no se había establecido un modelo educativo parametrizado, fundamentado en investigaciones externas y apoyado con TIC y herramientas computacionales. Esta investigación aporta un modelo diseñado y estructurado, con aportes de otras universidades locales y foráneas y el planteamiento de diversas estrategias que enriquecen el proceso de aprendizaje de los estudiantes.

Plantillas de evaluación

El docente director aporta plantillas de evaluación desarrolladas y mejoradas a través de su experiencia y corroboradas con otras investigaciones.

Ponencia nacional

Se presentó la ponencia nacional titulada: Investigación descriptiva de Metodologías de Enseñanza/Aprendizaje de la Ingeniería del Software, en el XII Foro Investigaciones en Informática Educativa, versión internacional, 2013, desarrollado en la ciudad de Medellín.

8.3 TRABAJO FUTURO

A partir de la experiencia adquirida en el desarrollo del proyecto, se propone como trabajo futuro los siguientes proyectos:

1. El diseño de objetos de aprendizaje en el área de ingeniería del software para la aprehensión de los conceptos del cuerpo del conocimiento. Este trabajo fortalecerá y apoyará en gran medida la etapa de conceptualización.
2. Investigar sobre proyectos de software que representen necesidades determinadas de la industria, creando diversos escenarios que puedan ser ejercitados en clase y durante los proyectos de producción de software.

BIBLIOGRAFÍA

Aguilar Sierra, A. (2003). Las *Metodologías Ágiles en la Enseñanza de la Ingeniería del Software*. Universidad Nacional Autónoma de México. ENC 2003.

Anaya, R. (2006). *Una visión de la enseñanza de la Ingeniería de Software como apoyo al mejoramiento de las empresas de software.* Revista Universidad EAFIT. Vol 42. N° 141. p. 60-76.

ArgoUML. Internet: http://es.wikipedia.org/wiki/ArgoUML

ArgoUML (2011). *User Manual. A tutorial and reference description*. Recuperado el 29 de Abril de 2012 de http://argouml.tigris.org/files/documents/4/0/argouml-0.16/argomanual-0.16.pdf

Brackett, John W. (1990). *Software Requirements*. SEI Curriculum Module SEI-CM-19-1.2. Software Engineering Institute - Carnegie Mellon University. p. 6.

Bourque, P; Robert, F; Lavoie, J; Lee, A; Trudel, S and Lethbridge, T. (2002). *Guide to the Software Engineering Body of Knowledge (SWEBOK) and the Software Engineering Education Knowledge (SEEK) –A preliminary mapping.* 10th International Workshop on Software Technology and Engineering Practice.

Bruegge, B. and Dutoit, A. (2004). *Object-Oriented Software Engineering*. Prentice-Hall. 507 p.

Casallas, R; Dávila, J. y Quiroga, J. (2002). *Enseñanza de la Ingeniería del Software por procesos instrumentados*. Universidad de los Andes, Depto. De Ingeniería de Sistemas y Computación. Bogotá, Colombia. X Congreso

Iberoamericano de Educación Superior en Computación en el marco de CLEI 2002.

Cockburn, A. (2006). *Designing an incremental-iterative one-semester, undergraduate course in Software Engineering*. Humans and Technology Technical Report HaT TR. Recuperado el 19 de Agosto de 2012 de http://alistair.cockburn.us/Designing+an+incremental-iterative+one-semester%2c+undergraduate+course+in+software+engineering

Escamilla González, A. (2008). *Las competencias básicas: Claves y propuestas para su desarrollo en los centros*. Editorial GRAÓ. España. 219 p.

European Commission. *Education and training - Glossary.* Recuperado el 30 de Junio de 2012 de http://ec.europa.eu/education/programmes/llp/guide/glossary_en.html

Galeana de la O, L. (2007). *Aprendizaje basado en proyectos*. Universidad de Colima, México. Revista CEUPROMED. Recuperado el 9 de Septiembre de 2011 de http://ceupromed.ucol.mx/revista/PdfArt/1/27.pdf

Gisbert Cervera, M.; Adell Segura, J.; Rallo Moya, R. y Bellver Torla, A. *Entornos virtuales de enseñanza – aprendizaje: el proyecto GET*. Recuperado el 20 de Julio de 2012 de http://www.ucm.es/info/multidoc/multidoc/revista/cuad6-7/evea.htm

González, B.; Jiménez, M. y Nieto, M. A. (2009). *Estudio comparativo Aplicaciones educativas GNU/Linux de la UPV-EHU*. Universidad del País Vasco.

Granda Dihigo, A. (2010). *Diseño de curso virtual para apoyar el proceso de enseñanza aprendizaje de la disciplina de Ingeniería y Gestión de Software en la Universidad de las Ciencias Informáticas.* Universidad de las Ciencias

Informáticas. Cuba. Revista Electrónica de Tecnología Educativa Edutec-e. Núm. 34

Granda Dihigo, A. y Santos Ramírez, Y. (2011). *Las TIC en la Enseñanza de la Ingeniería del Software en la Universidad de las Ciencias Informáticas. Pasado, presente y futuro*. Universidad de las Ciencias Informáticas.

Grande, I. y Abascal, E. (2005). *Análisis de encuestas*. ESIC Editorial. Madrid.

Hernández Orallo, E. (2002). *El Lenguaje Unificado de Modelado (UML)*. Recuperado el 25 de Agosto de 2012 de http://www.acta.es/medios/articulos/informatica_y_computacion/026067.pdf

Hilburn, T. B. and Bagert, D. J. (1999). *A Software Engineering Curriculum Model*. Fronteries in Education Conference. IEEE Conference Publications. Vol 1. P 12A4/6 – 12A4/11.

Huber, G. (2008). *Active learning and methods of teaching*. Universität Tübingen, Institut für Erziehungswissenschaft. Tübingen, Alemania. Revista de Educación. P. 59 – 81.

IEEE. (1990). *IEEE Standards Board, "IEEE standard glossary of software engineering terminology – IEEE std 610.12-1990"*. New York, NY, USA.

IEEE and ACM. (2004). *Curriculum guidelines for undergraduate degree programs in software engineering*. IEEE Computer Society and Association for Computing Machinery. Software Engineering 2004.

IEEE (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK).* Versión 3.0. Editores Pierre Bourque y Richard Fairley. Tomado de http://www.computer.org/portal/web/swebok

Lizano Madriz, F.; Sandoval Carvajal, M. y García Vargas, M. (2010). *SWEBOK aplicado: "Experiencias en la cátedra de Ingeniería de Sistemas"*. Universidad Nacional de Heredia, Costa Rica.

López Ortega, D. y Santa Villa, J. A. (2012). *Estudio comparativo de las herramientas case: Staruml, Poseidon for UML y Enterprise Architect, para el modelamiento de diagramas UML*. Universidad Tecnológica de Pereira.

Marques Graells, Pere. (2008). *Impacto de las TIC en la enseñanza Universitaria*. Departamento de Pedagogía Aplicada, Facultad de Educación. Universidad Autónoma de Barcelona.

Michael, M. (2000). *Fostering and assessing communications skills in the computer science context*. ACM SIGCSE Bulletin, Vol 32, Issue 1. Pp. 119 – 123. NY, Estados Unidos.

Mirian-Hosseinabadi, S.; Aghakasiri, Z.; Sadeghi, A.; Delfani, P. and Ghandehari, M. (2010). *Emphasizing Experiences in Teaching Software Engineering*. 2nd International Conference on Education Technology and Computer (ICETC). Computer Engineering Department Sharif Univeristy of Technology. Tehran, Irán. p. 149- 153.

Návrat, P. and Biellková, M. (1999). *Software Engineering Education: Different Contexts, Similar Contents*. ACM SIGCSE Bulletin. Volume 31, Issue 2. P. 55 – 59.

Netbeans vs. Eclipse: an IDE comparison. Recuperado el 10 de diciembre de 2012 de http://software-talk.org/blog/2012/01/netbeans-vs-eclipse-an-ide-comparison/

Pollock, L. (2001). *Integrating an intensive experience with communication skills development into a Computer Science course*. University of Delaware, Newark, Estados Unidos. p. 287 - 291

Port, D. and Boehm, B. (2001). *Using a model framework in developing and delivering a family of Software Engineering Project courses*. University of Sourthern California for Software Engineering. Estados Unidos. Publicado en Software Engineering Education and Training, 14th Conference. P. 44 – 55.

Robillard, P. (2004). *Teaching Software Engineering through a Project-Oriented Course*. CSEE'96 Proceedings of the 9th Conference on Software Engineering Education, p. 85 – 95. Department de génie électrique et informatique. Ecole Polytechnique de Montréal. Canada.

Rodríguez S., K..; Maya R., M. y Jaen P., J. (2012). *Educación en Ingenierías: de las clases magistrales a la pedagogía del aprendizaje activo*. Revista Ingeniería y Desarrollo. Volumen 30, Nº 1. Enero – Junio.

Roy, G and Veraart, V. (1996). *Software Engineering Education: from an Engineering Perspective*. Computer Science, PSET. Murdoch University, Australia. p. 256 – 262.

Salas Zapata, W. (2005). *Formación por competencias en Educación Superior. Una aproximación conceptual al propósito del caso colombiano.* Universidad de Antioquia, Colombia. Revista Iberoamericana de Educación.

Salazar Bermúdez, G. (2012). *Desafíos del curso de ingeniería del software.* Universidad de Costa Rica. Revista Educación en Ingeniería Vol 7, N° 13. p. 32 – 43.

Santa Lozano, J.; Zamora Izquierdo, M. A. y Ubeda Miñarro, B. (2008). *El Aprendizaje Basado en Proyectos en materias de Ingeniería Informática y sus implicaciones*. Universidad de Murcia. España. Comunicación presentada en la I Jornada sobre nuevas tendencias en la enseñanza de las ciencias y las ingenierías.

Seffah, A. and Grogono, P. (2002). *Learner-centered Software Engineering Education: from resources to skills and pedagogical patterns*. Software Engineering Education and Training (CSEE&T 2002) 15th Conference. p. 14 – 21. Concordia University. Montreal, Canada.

Surendran, K. Young, F. H. (2000). *Teaching Software Engineering in a practical way*. 13th Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ 2000), Wellington, New Zeland. p. 345 - 350.

Universitat Jaume. (2003). *Espacio interuniversitario de recursos para la EEES: Metodologías de E-A /Estrategia didáctica*. Recuperado el día 12 de Mayo de 2013, de http://www.recursoseees.uji.es/fichas/fichas.php,.

Visio. *Visio 2010 features and benefits*. Recuperado el 4 de Marzo de 2012 de http://office.microsoft.com/en-us/visio/visio-2010-features-and-benefits-HA101631752.aspx

Weitzenfeld, A. (2005). *Ingeniería del Software con UML y Java*. Thompson. 438 p.

Winbladh, K. (2004). *Requirements Engineering: closing the gap between academic supply and industry demand*. ACM Crossroads, Vol 10, Issue 4. p. 4-14. New York, Estados Unidos.

WizIQ. *¿Qué son los sistemas de administración del aprendizaje (LMS, Learning Management Systems)*? Recuperado el 4 de Marzo de 2012 de http://www.wiziq.com/tutorial/52614-Plataformas-LMS

Zapata J., C. M. y Duarte H., M. I. (2008). *El juego de la consistencia: una estrategia didáctica para la ingeniería de software*. Universidad Nacional de Colombia. Revista técnica de la Facultad de Ingeniería de la Universidad de Zulia. Vol. 31. Fascículo 1. P. 3 – 12.

ANEXOS

Anexo 1: Plan de trabajo asignatura Ingeniería del Software I

UNIVERSIDAD INDUSTRIAL DE SANTANDER Escuela de Ingeniería de Sistemas e Informática	MSc. FERNANDO ROJAS MORALES <frojas@uis.edu.co>
INGENIERÍA DE SOFTWARE 1 - 22969 4 horas teóricas – ***8 Trabajo Independiente Estudiante***	2o Semestre / 2011

¿Cómo realizar software de calidad, en el tiempo y con el presupuesto, estimados?

OBJETIVOS Y COMPETENCIAS DEL CURSO

Proveer un espacio de formación para: la ejercitación de valores, la apropiación de fundamentos conceptuales, y el desarrollo de habilidades para la realización de software de calidad utilizando un enfoque de ingeniería en asuntos como: {obtención de requerimientos, desarrollo de modelos de análisis y diseño, uso de estándares, documentación y comunicación de artefactos} de software.

CONTENIDO

0. Introducción a la Ingeniería de Software (IdS)
0.1. Origen y situación actual de la IdS; código de ética
0.2. Ciclos de vida del proceso de software
1. Requerimientos de Software
1.1. Obtención de requerimientos
1.2. Modelo de Casos de Uso
1.3. Modelo de Clases del Dominio
1.4. Uso de Prototipos para especificación de requisitos software
2. Análisis
2.1. Arquitectura MVC
2.2. Refactorización
2.3. Modelos de comportamiento dinámico
3. Diseño de Software
3.1. Fundamentos del Diseño
3.2. Diseño de interfaces externas
4. Construcción de Software
5. Pruebas de Software
6. Mantenimiento de Software

ESTRATEGIAS PEDAGÓGICAS

- Presentación de temas e ilustración con casos de aplicación (*profesor*)
- Realización y presentación de artefactos de casos de aplicación (*estudiantes*)

Fuente: Director del proyecto.

CASO DE APLICACIÓN (CA)

En las horas de TI, por grupos (de 3 estudiantes), se desarrollan artefactos de software para un caso de aplicación (CA), y por turno cada uno de los miembros del grupo los presenta en clase:	***Formato***
1. Modelo de Requerimientos-C/CA	*tarea_1.do*
2. Modelo de Requerimientos-D/CA	*tarea_2.do*
3. Modelo de Análisis /CA	*tarea_3.do*
4. Prototipo Funcional de Software /CA	

Herramientas para Modelos UML y para el desarrollo del prototipo, a discreción de los estudiantes. Se propone el uso de: NetBeans, Eclipse, Aptana, StarUML, Argo-UML, etc.

SISTEMA DE EVALUACION

Caso de Aplicación

- Tarea 1 (*documento*) 10%
- Tarea 2 (*documento*) 10%
- Tarea 3 (*documento*) 10%
- Prototipo de Software funcional (*estudiantes*)..... 20%
- Presentación Artefactos (*estudiante*)..... 10%

Fundamentación Teórica

- Examen de medio término (***Semana 9***)..... 20%
- Examen Final (***Semana 15***) 20%

CRITERIOS DE EVALUACION

Caso de Aplicación

- Sólo se calificarán las tareas recibidas antes de la fecha y hora establecida. Las tareas recibidas tarde tendrán una calificación de 0.0, aunque el equipo tendrá derecho a realizar la presentación en clase
- Aplicación disciplinada de la metodología
- Completitud de los artefactos requeridos

Presentaciones

- Puntualidad: *llegada tarde se califica sobre 4.0, inasistencia se califica sobre 3.0 en la siguiente sesión*
- Claridad en la comunicación de las ideas
- Consistencia
- Creatividad y materiales de apoyo utilizados

Fuente: Director del proyecto.

BIBLIOGRAFÍA

- Bruegge, B., Dutoit, A. Object –Oriented Software Engineering – 2E. Prentice-Hall, 2004.
- Stumpf, R., Teague, L. Object –Oriented System Analysis. Prentice-Hall, 2005.
- Weitzenfeld, A. Ingeniería de software con UML y Java. Thompson, 2005.
- Braude, E. Ingeniería de software. *Una perspectiva orientada a objetos*. AlfaOmega, 2000.
- Pressman, R. Ingeniería del Software – 6E. McGraw-Hill, 2005.
- Sommerville, I. Ingeniería del Software – 7e. Prentice-Hall, 2005.
- Pfleeger, S. Ingeniería de software. *Teoría y Práctica*. Prentice-Hall. 2002.
- Ruble, D. Análisis y Diseño Práctico de Sistemas Cliente/Servidor con GUI. Prentice-Hall, 1998.
- Kendall & Kendall. Análisis y Diseño de Sistemas. Prentice-Hall, 2005.

CRONOGRAMA (*Fechas importantes a tener en cuenta*)

SEMANA	SESION	TEMA
4	9	Modelo de Requerimientos-C/CA
	10	Modelo de Requerimientos-C/CA
8	15	Modelo de Requerimientos-D/CA
	16	Modelo de Requerimientos-D/CA
9	17	Examen de Medio Término
13	25	*Modelo de Análisis*
	26	*Modelo de Análisis*
15	30	Examen Final
16	31	*Prototipo*
	32	*Prototipo*

HORAS DE CONSULTA: *Martes, Viernes: 2-4 pm. LP-223.*

Fuente: Director del proyecto.

Anexo 2: Código de ética y Práctica profesional

PREÁMBULO

Las computadoras tienen un papel central cada vez mayor en el comercio, industria, gobierno, medicina, educación, entretenimiento, y sociedad. Los ingenieros de software son aquellos que contribuyen, mediante la participación directa o enseñanza, al análisis, especificación, diseño, desarrollo, certificación, mantenimiento y pruebas de sistemas de software. Debido a sus funciones en el desarrollo de sistemas de software, los ingenieros de software tienen suficientes oportunidades para causar beneficio o generar daño y para habilitar o influenciar a otros a causar daño o beneficio. Para asegurar, en la medida de lo posible, que sus esfuerzos se utilizarán para hacer el bien, los ingenieros de software deben comprometerse a hacer de la ingeniería del software una profesión benéfica y respetada. De acuerdo con tal compromiso, los ingenieros de software deberán adherirse al siguiente Código De Ética Y Práctica Profesional.

El Código contiene ocho Principios relacionados con la conducta y las decisiones tomadas por los ingenieros de software profesionales, bien sean profesionales en ejercicio, educadores, administradores, supervisores y directivos, así como educandos y estudiantes de la profesión. Los Principios identifican las relaciones éticamente responsables en las que los individuos, grupos y organizaciones participan, y las principales obligaciones de tales relaciones. Las Cláusulas de cada Principio son ejemplos de algunas de las obligaciones incluidas en estas relaciones. Estas obligaciones se fundamentan en las características humanas del ingeniero de software en los deberes hacia las personas que se ven 10 afectadas por el trabajo del ingeniero de software, y en los elementos peculiares de la práctica de la ingeniería del software. El Código prescribe éstas como obligaciones de cualquiera que se identifique como ingeniero de software o que aspire a serlo.

No se pretende que se utilicen partes individuales del Código aisladamente, para justificar errores por omisión o comisión. La lista de Principios y Cláusulas no es exhaustiva. Las Cláusulas no deben leerse como la frontera entre lo aceptable y lo inaceptable en todas las situaciones prácticas de la conducta profesional. El Código no es un simple algoritmo ético que genera decisiones éticas. En algunas situaciones los estándares pueden entrar en conflicto entre sí o con estándares de otras fuentes. Estas situaciones requieren que el ingeniero de software utilice su juicio ético para actuar en la manera más congruente con el espíritu del Código de Ética y Práctica Profesional, teniendo en cuenta las circunstancias.

Los conflictos éticos pueden manejarse mediante una consideración cuidadosa de los principios fundamentales, más bien que apoyándose ciegamente en reglamentos detallados. Estos Principios deberían influenciar a los ingenieros de software a considerar ampliamente a quién se ve afectado por su trabajo; a examinar si ellos o sus colegas tratan al resto de las personas con el debido respeto; a reflexionar en cómo la sociedad vería sus decisiones si estuviera bien informada; a analizar cómo el menos favorecido será afectado por su decisión; y a considerar si sus actos lo juzgarían como un valioso profesional ideal que trabaja como ingeniero de software. En todas estas valoraciones la preocupación por la salud, seguridad y bienestar público es primordial; esto es, el "Interés Social" es central en este Código.

El contexto dinámico y exigente de la ingeniería de software requiere un código que sea adaptable y relevante a las nuevas situaciones a medida que ocurran. Sin embargo, incluso en esta generalidad, el Código proporciona apoyo a los ingenieros del software y administradores que necesitan actuar positivamente en un caso específico documentando la postura ética de la profesión. El Código proporciona un fundamento ético al cual los individuos de un equipo o el propio equipo pueden acudir. El Código también ayuda a definir aquellas cuestiones que

son éticamente impropias de solicitar a un ingeniero de software o equipo de ingenieros de software.

El Código no está simplemente orientado a identificar la naturaleza de los actos cuestionables, sino que también tiene una función educativa importante. Puesto que este código representa el consenso de la profesión en cuestiones éticas, es un medio para educar tanto a la sociedad como a los futuros profesionales acerca de las obligaciones éticas de todos los ingenieros de software.

PRINCIPIOS

Principio 1. Sociedad.

Los ingenieros de software actuarán de forma congruente con el interés social. Particularmente, los ingenieros de software deberán, cuando sea apropiado:

1. Aceptar la responsabilidad total de su trabajo.
2. Moderar los intereses del ingeniero de software, el empresario, el cliente y los usuarios con el bienestar social.
3. Aprobar software sólo si se tiene una creencia bien fundamentada de que es seguro, cumple las especificaciones, pasa las pruebas apropiadas y no reduce la calidad de vida, la privacidad o daña el medio ambiente. El efecto último del trabajo deberá ser el bien social.
4. Exponer a las personas o autoridades apropiadas cualquier daño real o potencial al usuario, a la sociedad o el medio ambiente, que razonablemente se cree que está asociado con el software o documentos relacionados.
5. Cooperar en los esfuerzos para solucionar asuntos importantes de interés social causados por el software, su instalación, mantenimiento, soporte o documentación.
6. Ser justo y veraz en todas las afirmaciones, particularmente las públicas, relativas al software o documentos asociados, métodos y herramientas.

7. Considerar incapacidad física, distribución de recursos, desventajas económicas y otros factores que pueden reducir el acceso a los beneficios del software.
8. Estar motivado a ofrecer voluntariamente asistencia técnica a buenas causas y contribuir a la educación pública relacionada con esta profesión.

Principio 2. Cliente y empresario.

Los ingenieros de software actuarán de manera que se concilien lo mejores intereses de sus clientes y empresarios, congruentemente con el interés social. Particularmente, los ingenieros de software deberán, cuando sea apropiado:

1. Prestar servicios en sus áreas de competencia, siendo honestos y francos acerca de sus limitaciones en su experiencia y educación.
2. No usar conscientemente software que se obtiene o retiene ya sea ilegalmente o sin ética.
3. Usar la propiedad de un cliente o empresario sólo en forma propiamente autorizada y con el conocimiento y consentimiento del cliente o empresario.
4. Cuando se requiera, asegurar que cualquier documento en el que se confía ha sido aprobado por alguien autorizado para aprobarlo.
5. Mantener secreta cualquier información confidencial obtenida en su labor profesional, donde tal confidencialidad es congruente con el interés social y congruente con la ley.
6. Identificar, documentar, reunir evidencia y reportar oportunamente al cliente o al empresario si, en su opinión, un proyecto tiene probabilidades de fracasar, de ser muy costoso, de violar la ley de propiedad intelectual o ser problemático de cualquier otro modo.
7. Identificar, documentar y reportar al cliente o empresario asuntos significativos de interés social, de los cuales se tiene conocimiento, acerca del software o documentos relacionados.
8. Rechazar trabajos externos que vayan en detrimento del trabajo que se realiza para su patrón primario.

9. No promover intereses adversos a su empresario o cliente, a menos que se comprometa un interés ético más alto; en ese caso, informar al empresario u otra autoridad apropiada del interés ético en cuestión.

Principio 3. Producto.

Los ingenieros de software asegurarán que sus productos y sus modificaciones correspondientes cumplen los estándares profesionales más altos posibles. Particularmente, los ingenieros de software deberán, cuando sea apropiado:

1. Procurar la alta calidad, costos aceptables y una agenda razonable asegurando que los costos y beneficios significativos son claros y aceptados por el empresario y el cliente, y están disponibles para consideración del usuario y de la sociedad.
2. Asegurar que las metas y objetivos para cualquier proyecto que se propone o en el que se trabaja sean adecuados y alcanzables.
3. Identificar, definir y atender asuntos éticos, económicos, culturales, legales y ambientales relacionados a los proyectos de trabajo.
4. Asegurar que se está calificado, con una combinación apropiada de educación, adiestramiento y experiencia para cualquier proyecto en que se trabaje o que se proponga trabajar.
5. Asegurar que se usan los métodos apropiados en cualquier proyecto en el que se trabaja o se propone trabajar.
6. Tratar de seguir los estándares profesionales más adecuados, siempre que estén disponibles, para el proyecto en que se trabaja. Sólo en caso de que hubiera una justificación ética o técnica mayor, se permitirá alterar dichos estándares.
7. Esforzarse por entender completamente las especificaciones del software en el que se trabaja.

8. Asegurar que las especificaciones del software en el que se trabaja están bien documentadas, satisfacen los requerimientos del usuario y cuentan con las aprobaciones adecuadas.
9. Asegurar estimaciones cuantitativas realistas de costos, agenda, personal, calidad y resultados de cualquier proyecto en el que se trabaja o se propone trabajar, proporcionando una evaluación de la incertidumbre de esas estimaciones.
10. Asegurar que las pruebas, depuración, revisión del software y documentos relacionados con los que se trabaja sean adecuados**.**
11. Asegurar que la documentación sea adecuada, incluyendo problemas significativos encontrados y soluciones adoptadas, para cualquier proyecto en el que se trabaja.
12. Trabajar para desarrollar software y documentos relacionados que respeten la privacidad de aquellos a quienes está dirigido este software.
13. Ser cuidadoso para usar sólo datos precisos derivados por medios éticos y legales, y usarlos sólo de las maneras propiamente autorizadas.
14. Mantener la integridad de los datos siendo sensible a aquellos inexactos u obsoletos.
15. Tratar todas las formas de mantenimiento de software con el mismo profesionalismo que los desarrollos nuevos.

Principio 4. Juicio.

Los ingenieros de software mantendrán integridad e independencia en su juicio profesional. Particularmente, los ingenieros de software deberán, cuando sea apropiado:

1. Moderar todos los juicios técnicos por la necesidad de apoyar y mantener llos valores humanos.
2. Endosar documentos únicamente cuando han sido preparados bajo su supervisión o dentro de sus áreas de competencia y con los cuales se está de acuerdo.

3. Mantener objetividad profesional con respecto a cualquier software o documento relacionado del cual se le pidió una evaluación.
4. No involucrarse en prácticas financieras fraudulentas tal como corrupción, facturación doble u otras prácticas financieras impropias.
5. Exponer a todas las partes involucradas aquellos conflictos de interés que no puedan evitarse o evadirse razonablemente.
6. Negarse a participar como miembro o asesor en organismos profesionales, privados o gubernamentales vinculados en asuntos relacionados con software donde sus empresarios o clientes pudieran tener conflictos de intereses no declarados todavía.

Principio 5. Administración.

Los ingenieros de software gerentes y líderes promoverán y se suscribirán a un enfoque ético en la administración del desarrollo y mantenimiento de software. Particularmente, los ingenieros de software administrando o dirigiendo deberán, cuando sea apropiado:

1. Asegurar una buena administración para cualquier proyecto en el cual trabaje, incluyendo procedimientos efectivos para promover la calidad y reducir riesgos.
2. Asegurar que los ingenieros de software estén informados de los estándares antes de sujetarse a ellos.
3. Asegurar que los ingenieros de software conozcan las políticas y procedimientos del empresario para protegar las contraseñas, archivos e información que es confidencial al empresario o confidencial a otros.
4. Asignar trabajo sólo después de tomar en cuenta contribuciones adecuadas de educación y experiencia moderadas con un deseo de continuar esa educación y experiencia.
5. Asegurar estimaciones de costos, agendas, personal, calidad y resultados cuantitativamente realistas en cualquier proyecto que trabaje o se propone trabajar, proporcionando una evaluación de la incertidumbre de esas estimaciones.

6. Atraer ingenieros de software potenciales sólo bajo una descripción completa y precisa de las condiciones del empleo.
7. Ofrecer una remuneración justa y equitativa.
8. No impedir injustamente que alguna persona ocupe una posición para la cual está perfectamente calificada.
9. Asegurar que exista un acuerdo equitativo en lo referente a la propiedad de cualquier software, proceso, investigación, documentación u otra propiedad intelectual a la cual el ingeniero de software ha contribuido.
10. Tomar medidas prudentes en procesos legales vinculados a la violación de la política de un empresario o de este código.
11. No pedir a un ingeniero de software hacer algo incongruente con este código.
12. No castigar a nadie por expresar temas éticos relativos a cualquier proyecto.

Principio 6. Profesión.

Los ingenieros de software incrementarén la integridad y reputación de la profesión congruentemente con el interés social**.** Particularmente, los ingenieros de software deberán, cuando sea apropiado:

1. Ayudar a desarrollar un ambiente organizacional favorable para actuar éticamente.
2. Promover el conocimiento público de la ingeniería de software.
3. Extender el conocimiento de la ingeniería de software participando apropiadamente en organizaciones, reuniones y publicaciones profesionales.
4. Apoyar, como miembros de una profesión, a otros ingenieros de software que se esfuercen por seguir este código.
5. No promover el interés propio a costa de la profesión, cliente o empresario.
6. Obedecer todas las leyes que gobiernan su trabajo, salvo en circunstancias excepcionales, donde tal obediencia es incongruente con el interés social.
7. Ser preciso en la descripción de las características del software en el que trabaja, evitando no sólo declaraciones falsas, sino también declaraciones que

podrían ser razonablemente asumidas como especulativas, vacías, fraudulentas, engañosas o dudosas.

8. Tomar la responsabilidad de detectar, corregir y reportar errores en el software y documentos asociados en los que se trabaja.
9. Asegurar que los clientes, empresarios y supervisores conozcan el compromiso de los ingenieros de software con este código de ética, y las subsecuentes ramificaciones de tal compromiso.
10. Evitar asociaciones con negocios y organizaciones que estén en conflicto con este código.
11. Reconocer que las violaciones de este código son incongruentes con ser un ingeniero de software profesional.
12. Hablar seriamente con la gente involucrada cuando se detecten violaciones significativas de este código, a menos que sea imposible, contra productivo o peligroso.
13. Reportar las violaciones significativas de este código a las autoridades correspondientes cuando está claro que consultar con la gente involucrada en estas violaciones es imposible, contra productivo o peligroso**.**

Principio 7. Colegas.

Los ingenieros de software apoyarán y serán justos con sus colegas. Particularmente, los ingenieros de software deberán, cuando sea apropiado:

1. Motivar a sus colegas a sujetarse a este código.
2. Ayudar a sus colegas en el desarrollo profesional.
3. Reconocer completamente el trabajo de otros y abstenerse de atribuirse méritos indebidos.
4. Revisar el trabajo de otros en forma objetiva, sincera y propiamente documentada.
5. Escuchar equitativamente las opiniones, preocupaciones y quejas de un colega.

6. Ayudar a sus colegas a que estén totalmente alertas a los actuales estándares incluyendo políticas y procedimientos de protección de contraseñas, archivos, información confidencial y las medidas de seguridad en general.
7. No intervenir injustamente en la carrera de algún colega; sin embargo, el interés del empresario, del cliente o el interés social puede conducir a ingenieros de software, de buena fe, a cuestionar la competencia de un colega.
8. En situaciones fuera de sus propias áreas de competencia, solicitar las opiniones de otros profesionales que tengan competencia en esa área.

Principio 8. Personal.

Los ingenieros de software participarán toda su vida en el aprendizaje relacionado con la práctica de su profesión y promoverán un enfoque ético en la práctica de la profesión. Particularmente, los ingenieros de software deberán, cuando sea apropiado:

1. Mejorar su conocimiento de los avances en el análisis, especificación, diseño, desarrollo, mantenimiento, pruebas del software y documentos relacionados, junto con la administración del proceso de desarrollo.
2. Mejorar su habilidad para crear software seguro, confiable, útil y de calidad a costos razonables y en un tiempo razonable.
3. Mejorar su habilidad para producir documentación precisa, informativa y bien redactada.
4. Mejorar su comprensión del software de los documentos con que se trabaja y del medio ambiente donde serán usados.
5. Mejorar su conocimiento de los estándares relevantes y de las leyes que gobiernan el software y los documentos con que se trabaja.
6. Mejorar su conocimiento de este código, su interpretación y su aplicación al trabajo.
7. No tratar injustamente a nadie debido a prejuicios irrelevantes.
8. No influenciar a otros a emprender alguna acción que involucre una violación de este código.

9. Reconocer que las violaciones personales de este código son incongruentes con ser un ingeniero de software profesional.

Este Código ha sido redactado por la IEEE-CS/ACM fuerza unida operante para la Ética y Práctica Professional de la Ingeniería de Software (SEEPP)