Universidad de Matanzas sede "Camilo Cienfuegos"

Facultad de Ciencias Técnicas

Ingeniería Informática

"Algoritmo para la estimación de prioridad y riesgo en los requisitos funcionales de metodologías ágiles"

Trabajo de Diploma en opción al título de Ingeniero Informático

Autor: Robert Hernández Zamora

Tutor: MSc. Liz Pérez Martínez

Ing. Yeiniel Alfonso

Matanzas, Junio de 2018

Dedicatoria:

A mi madre por todo el apoyo brindado durante mi vida y en especial estos cinco años.

A mi novia que me ha apoyado en las largas horas de estudio.

Agradecimientos

A todos los profesores del Departamento de Informática que me han hecho lo que soy.

A mi familia y amigos por apoyarme.

Y a todo el que de una manera u otra me ayudo en el desarrollo de esta investigación.

Declaración de autoría

Yo, Robert Hernández Zamora, declaro que soy el único autor de este trabajo y autorizo a la Universidad de Matanzas, especialmente a la Facultad de Ciencias Técnicas, a que hagan el uso que estimen pertinente de él.

Y para que así conste, firmo la presente a los 14 días del mes de junio del 2018.

_____ _____

Firma del Autor Firma del Tutor

Opinión del Tutor

DATOS PERSONALES DEL TUTOR

Nombre y apellidos: Liz Pérez Martínez.
Centro de trabajo: Universidad de Matanzas.
Organismo a que pertenece: Ministerio de Educación Superior – MES.
Cargo que ocupa: Vicedecana Facultad de Ciencias Técnicas.
Especialidad de la que es graduado: Ingeniería Informática. Universidad de Matanzas, 2012.
Categoría docente o investigativa: Asistente.
Grado científico: Master en Ciencias, Universidad de Matanzas, 2015.

DATOS DE LA TESIS Y EL DIPLOMANTE

Nombre y apellidos: Robert Hernández Zamora.
Centro de estudio: Universidad de Matanzas sede “Camilo Cienfuegos”.
Título de la Tesis: Algoritmo para la estimación de prioridad y riesgo en los requisitos funcionales de metodologías ágiles.

OPINION SOBRE EL TRABAJO

La tesis presentada posee gran novedad investigativa, pues intenta resolver un problema real presente en las metodologías ágiles de desarrollo de software, métodos muy difundidos actualmente en los diseños de proyectos de mediana y pequeña envergadura.

El tutor de este trabajo de diploma considera que, durante su ejecución, el estudiante mostró las cualidades que a continuación se detallan:

Una absoluta independencia en el desarrollo de su trabajo. Tenacidad y espíritu de investigación. Incursionando en una temática de gran complejidad que exige del conocimiento no sólo del área informática, sino de varias especialidades, que el estudiante no recibió en su formación como ingeniero. A pesar de esto logró captar con rapidez y profesionalidad el conocimiento necesario para enfrentar el problema planteado. Logró ofrecer soluciones importantes que permite apreciar la profesionalidad que alcanza como investigador.

Los aspectos tanto metodológicos como de la investigación científica se evidencian con facilidad en todo el desarrollo de su trabajo. Desempeño su investigación con rigor y

siguiendo las etapas consecuentemente. La documentación y las pruebas realizadas, tienen gran valor y están correctamente estructurados. Demostró dominio sobre la temática enfrentada.

En el trabajo se aprecia profesionalidad, manifestado desde el tratamiento de los conceptos estudiados y referenciados en la bibliografía, hasta las conclusiones a las que arribó, lo que ha contribuido en gran medida a la solución de los problemas enfrentados. Ha dejado planteados importantes elementos a tener en consideración en futuras investigaciones. Las conclusiones están correctamente estructuradas y en concordancia con los objetivos a lograr. El trabajo cumple con los objetivos propuestos, aborda una temática de gran novedad y rigor científico

Por todo lo anteriormente señalado, considero que el estudiante Robert Hernández Zamora reúne los requisitos para el título de Ingeniero Informático y espero le sea otorgada la mejor calificación que pueda emitir este tribunal. Espero, que su futuro como profesional colme todas sus expectativas y le depare metas que amplíen su capacidad como Ingeniero Informático.

MSc. Liz Pérez Martínez
Dpto. Informática
Universidad de Matanzas
Junio/2018

Resumen

Los proyectos que utilizan metodologías ágiles de desarrollo requieren de un proceso de planificación para estimar la prioridad y el riesgo de sus requisitos funcionales y medir diferentes aspectos de los mismos para garantizar la calidad del mismo. Los procesos de estimación tradicionales y las métricas usuales de la Ingeniería de Software y la Ingeniería del Conocimiento no son adecuados para estos proyectos, ya que las etapas de desarrollo y los parámetros utilizados son de naturaleza y características diferentes. En ese contexto, se ha desarrollado un algoritmo para estimar prioridad y riesgo en dichas metodologías. No obstante, existe la necesidad de abordar métricas específicas aplicables a este proceso. Utilizando teoría de grafos y algoritmos de búsqueda es posible optimizar el proceso de estimación para dar una respuesta más correcta y mejorar los índices de tiempo, esfuerzo y costo de nuestro proyecto. Después el riesgo se calcula con una formula y una matriz de probabilidad – impacto para dar un valor más certero y saber a qué consecuencias nos enfrentamos en caso de un error o infortunio. Posteriormente a este algoritmo se le aplicaron una serie de pruebas resultando todas satisfactorias demostrando que es más viable que utilizar la estimación a ciegas o basada en el criterio del cliente.

Abstract

Projects that use agile development methodologies require a planning process to estimate the priority and risk of their functional requirements and measure different aspects of them to guarantee their quality. The traditional estimation processes and the usual metrics of Software Engineering and Knowledge Engineering are not suitable for these projects, since the stages of development and the parameters used are of different nature and characteristics. In this context, an algorithm has been developed to estimate priority and risk in said methodologies. However, there is a need to address specific metrics applicable to this process. Using graph theory and search algorithms it is possible to optimize the estimation process to give a more correct answer and improve the time, effort and cost indices of our project. Then the risk is calculated with a formula and a probability - impact matrix to give a more accurate value and know what consequences we face in case of an error or misfortune. After this algorithm, a series of tests were applied, all of them satisfactory, demonstrating that it is more viable than using blind estimation or based on the client's criteria.

Índice de Contenido

Introducción

Actualmente la ejecución de proyectos es muy importante, dada la necesidad de integrar un diverso conjunto de recursos, personas y tecnologías para cumplir uno o más objetivos, permitiendo la evolución de los recursos de las tecnologías de información, de tal forma que faciliten las actividades laborales o personales, disminuyendo su complejidad y tiempo. Existen diferentes tipos de proyectos; entre ellos se encuentran los informáticos, que son “un sistema de cursos de acción simultáneos y/o secuenciales que incluye personas, equipamientos de hardware, software y comunicaciones, enfocados en obtener uno o más resultados deseables sobre un sistema de información” (Moguel, 2005), los cuales son importantes dado que permiten la evolución de los recursos de la tecnología de la información facilitando las actividades laborales o personales, disminuyendo su complejidad y tiempo.

Generalmente el proceso de desarrollo de un sistema informático lleva asociado un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. (Fuentes, 2016) Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad.

En este sentido, un requisito funcional define una función del sistema de software o sus componentes. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Los requisitos funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que se supone, un sistema debe cumplir. Los requisitos de comportamiento para cada requisito funcional se muestran en los casos de uso. Son complementados por los requisitos no funcionales, que se enfocan en cambio en el diseño o la implementación.

Como se define en la ingeniería de requisitos, los requisitos funcionales establecen los comportamientos del software. Típicamente, un analista de requisitos genera requisitos funcionales después de realizar los casos de uso. Sin embargo, esto puede tener excepciones, ya que el desarrollo de software es un proceso iterativo y algunos requisitos son previos al diseño de los casos de uso. (Cooke, 2012) Ambos elementos (casos de uso y requisitos) se complementan en un proceso bidireccional.

Un requisito funcional típico contiene un nombre, un número de serie único y un resumen. Esta información se utiliza para ayudar al lector a entender por qué el requisito es necesario, y para seguir al mismo durante el desarrollo del producto. El núcleo del requisito es la descripción del comportamiento requerido, que debe ser clara y concisa. Este comportamiento puede provenir de reglas organizacionales o del negocio, o ser descubiertas por interacción con usuarios, inversores y otros expertos en la organización. (Prieto, 2012) La importancia de la priorización y riesgos en las descripciones de comportamiento se hace necesaria si se quiere tener un mayor grado de exactitud y eficiencia en un proyecto dado.

Sin embargo, estos métodos, aunque efectivos son costosos. Para realizarlos es necesario aplicar una serie de fórmulas y algoritmos, y así controlar mejor los imprevistos del proyecto. En este trabajo se busca presentar una propuesta que mejore la estimación de prioridad y riesgo en los requisitos funcionales de las metodologías agiles usando la teoría de grafos como vía de solución.

A partir del planteamiento anterior se logró identificar el siguiente **problema científico**: ¿cómo estandarizar el proceso de obtención de los indicadores de prioridad y riesgo de los requisitos funcionales en las metodologías ágiles a través del empleo de un algoritmo de estimación?

La práctica demuestra que existen grandes dificultades sobre este tema, por lo que se plantea la creación de un algoritmo usando la teoría de grafos para representar los riesgos y prioridades de los requisitos funcionales de las metodologías agiles, realizar distintas corridas y hallar la forma más óptima. Lo anterior justifica la necesidad de un procedimiento aplicando las técnicas computacionales, en específico las estructuras de datos. Por lo que se define

como **objeto de estudio**: el análisis de los resultados utilizando un algoritmo basado en teoría de grafos.

Además, como **campo de acción** de la investigación: la clasificación de los resultados obtenidos en las diferentes corridas del algoritmo.

Lo anterior reafirma la **hipótesis** de que si se crea un algoritmo de estimación entonces se estandarizará el proceso de obtención de los indicadores de prioridad y riesgo de los requisitos funcionales en las metodologías ágiles.

Para lo cual se propuso como **objetivo general**, crear un algoritmo de estimación que estandarice el proceso de obtención de los indicadores de prioridad y riesgo de los requisitos funcionales en las metodologías ágiles.

Atendiendo al objetivo general, se definen como **objetivos específicos** de la investigación los siguientes:

1. Elaborar un marco teórico referencial sobre la correlación en las tareas de estimación de riesgo y prioridad en los requisitos funcionales de las metodologías ágiles.
2. Desarrollar un algoritmo de estimación para la obtención de los indicadores de prioridad y riesgo de los requisitos funcionales en las metodologías ágiles.
3. Validar el algoritmo de estimación mediante la realización de varias corridas para evaluar resultados.

Para el desarrollo de la investigación se emplearon diferentes métodos científicos tales como:

- ✓ El método inductivo - deductivo: su uso fue necesario tanto en la revisión bibliográfica, como en el análisis de los resultados, permitiendo arribar a conclusiones que se infirieron a partir de propiedades y relaciones existentes entre los elementos que conforman el fenómeno objeto de estudio.
- ✓ El método analítico - sintético al descomponer el problema de investigación en elementos por separado y profundizar en el estudio de cada uno de ellos de forma independiente, para luego sintetizarlos en la solución de la propuesta.

Entre los aportes de la investigación se destacan:

- ✓ El teórico-investigativo, al integrar los procedimientos tradicionales más utilizados por autores relacionados con el tema, a través de diferentes fases, etapas y pasos que permiten orientar metodológicamente la secuencia de acciones lógicas a desarrollar; y los elementos a tener en cuenta para la continuidad de la investigación.
- ✓ El práctico, al desarrollar una herramienta informática que permita mejorar los índices de cálculo de riesgo y prioridad de los requisitos funcionales en las metodologías ágiles.

El **resultado esperado** de este trabajo es la creación de un algoritmo basado en teoría de grafos para la estimación del riesgo y prioridad en los requisitos funcionales de las metodologías ágiles. Con ello se lograría mejorar el desarrollo de un proyecto, así como lograr un mejor tiempo de ejecución y resultado obtenido.

Atendiendo a lo planteado anteriormente, la tesis queda estructurada en introducción, tres capítulos, conclusiones, recomendaciones y referencias bibliográficas, según sigue:

- ✓ Una introducción donde se caracteriza la situación problemática y se fundamenta el problema científico a resolver.
- ✓ Un primer capítulo dedicado al marco teórico-referencial donde se ponen ejemplos de diferentes metodologías ágiles de desarrollo de software y su forma de estimar riesgo y prioridad en las historias de usuario. También se abordan los aspectos teóricos que ayudan a comprender la problemática planteada y que se tuvieron en cuenta para proponer la solución. Por otra parte, se da un breve resumen de cada uno de los recursos que se tomaron como punto de partida de la solución propuesta.
- ✓ Un capítulo dos donde se describe detalladamente como es el funcionamiento del procedimiento propuesto para desarrollar un algoritmo utilizando la teoría de grafos. Mediante un ejemplo, se va mostrando el resultado que se obtiene al aplicar cada paso del procedimiento a la misma.

- ✓ Un tercer capítulo dedicado a la experimentación, discusión y análisis de resultados. Se realizaron diferentes pruebas, con el objetivo mejorar los resultados. Todas ellas se explicarán en el presente capítulo, así como un pequeño análisis de las pruebas, comparaciones entre las mismas y se señalarán los aspectos más notables en cada una de ellas.
- ✓ Un apartado de conclusiones donde se verifica el cumplimiento de los objetivos trazados al inicio de la investigación.
- ✓ Las recomendaciones en la cual se plasman una serie de propuestas encaminadas a la continuidad de esta investigación.
- ✓ Y las referencias de la bibliografía citada.

Capítulo 1: Marco Teórico – Referencial

1.1 Introducción

Para la realización de la investigación fue necesario el estudio de diferentes conceptos y recursos que resultaron de gran utilidad para realizar nuestra propuesta. En este capítulo se hace una valoración del estado del arte a través del estudio de la bibliografía utilizada, consistente en diferentes métodos para la estimación de prioridad y riesgo en historias de usuario. Se abordan los aspectos teóricos que ayudarán a entender la problemática planteada y que se tuvieron en cuenta para proponer la solución. Se presentan diferentes tecnologías, algoritmos y recursos que han sido combinados para realizar tareas de estimación de prioridad y riesgo en historias de usuario. Se dará una breve explicación de todos los recursos utilizados, sus características y funcionamiento.

Hablar de metodologías agiles implica hacer referencia a las metodologías de desarrollo de software tradicionales ya que las primeras surgieron como una reacción a las segundas; sus características principales son antagónicas y su uso ideal aplica en contextos diferentes. (Mcconnell, 2012) Las metodologías ágiles son flexibles, pueden ser modificadas para que se ajusten a la realidad de cada equipo y proyecto. Los proyectos ágiles se subdividen en proyectos más pequeños mediante una lista ordenada de características. Cada proyecto es tratado de manera independiente y desarrolla un subconjunto de características durante un periodo de tiempo corto, de entre dos y seis semanas. La comunicación con el cliente es constante al punto de requerir un representante de él durante el desarrollo. Los proyectos son altamente colaborativos y se adaptan mejor a los cambios; de hecho, el cambio en los requerimientos es una característica esperada y deseada, al igual que las entregas constantes al cliente y la retroalimentación por parte de él. Tanto el producto como el proceso son mejorados frecuentemente.

1.2 Antecedentes del trabajo

Como se ha visto con anterioridad las tareas de estimación de prioridad y riesgo en los requisitos funcionales de las metodologías ágiles son complejas, muchos investigadores plantean resolver esto mediante diferentes métodos con el fin de

mejorar los resultados alcanzados hasta el momento en este campo, pero ninguno de estos contiene el uso de un algoritmo.

A continuación, se describen las metodologías ágiles estudiadas para el desarrollo de esta investigación y cómo afrontan dicha problemática.

Programación extrema (XP)

La programación extrema o *eXtreme Programming* (XP) es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck. Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de la XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos. Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software. (Rodriguez, 2015)

El primer paso de cualquier proyecto que siga la metodología XP es definir las historias de usuario con el cliente. Las historias de usuario tienen la misma finalidad que los casos de uso, pero con algunas diferencias: Constan de 3 ó 4 líneas escritas por el cliente en un lenguaje no técnico sin hacer mucho hincapié en los detalles; no se debe hablar ni de posibles algoritmos para su implementación ni de diseños de base de datos adecuados, etc. Son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cuando llega la hora de implementar una historia de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia. El tiempo de desarrollo ideal para una historia de usuario es entre 1 y 3 semanas. En cuanto a riesgos si surgen

problemas potenciales durante el diseño, X.P sugiere utilizar una pareja de desarrolladores para que investiguen y reduzcan al máximo el riesgo que supone ese problema. Después de tener ya definidas las historias de usuario es necesario crear un plan de publicaciones, un plan de publicaciones es una planificación donde los desarrolladores y clientes establecen los tiempos de implementación ideales de las historias de usuario, la prioridad con la que serán implementadas y las historias que serán implementadas en cada versión del programa. Estos métodos de estimación dependen mucho del programador y no logran ser muy eficientes a la hora de estimar la prioridad o riesgo de los requisitos funcionales. (Barros, 2014)

Scrum

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos. En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales. Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto. (Mcconnell, 2012)

Las historias de usuario deben describir qué se espera como salida de la implementación, y cómo se ve beneficiado el usuario final. Se expresa en lenguaje natural y sencillo, para poder conversar directamente con ellos sobre el tema. Hay que considerar que los usuarios finales tienden a desconocen el

lenguaje técnico, por lo que deben evitarse las palabras difíciles. Al usuario no le importa qué tecnología se usa, lo que busca es que le ayudemos a resolver su problema, y cómo usará esta solución; desea que sea simple, intuitiva y fácil de usar. Los elementos de que componen el producto a desarrollar deben estar estimados y, lo que, es más, deberían revisarse y re-estimarse cada iteración conforme vamos obteniendo nueva información. El Equipo proporciona al Dueño de Producto estimaciones de esfuerzo para cada elemento del Producto, y quizás también proporcione estimaciones de riesgo técnico. El Dueño de Producto y otros participantes proporcionan información sobre el valor de cada petición, lo cual puede incluir ingresos producidos, reducción de costes, riesgo de negocio, importancia para distintos participantes y otros datos. (Almunia, 2016)

Por lo tanto, hay una diferencia de opinión sobre si todos los riesgos pueden gestionarse con Scrum o si los riesgos deben ser administrados explícitamente. La mayoría de los miembros de la comunidad Ágil están de acuerdo en que los riesgos relacionados con el proyecto se pueden manejar bien con Scrum. Sin embargo, hay una desconexión cuando los riesgos son externos al proyecto. Del mismo modo, no hay consenso en quien es el responsable de la gestión del riesgo. Para muchos todo parece indicar que la responsabilidad de la gestión del riesgo la tiene que llevar el dueño del producto, sin embargo, algunos defensores de las metodologías ágiles creen que el riesgo debe ser administrado por el equipo en su conjunto.

Crystal

Las metodologías Crystal son una familia de metodologías ágiles, donde cada una de ellas está adecuada para un tipo de proyecto. Su creador es el popular Cockburn uno de los firmantes del manifiesto ágil. En las metodologías Crystal, proyectos grandes, que necesitan más coordinación y comunicación, se asocian con colores más oscuros. Proyectos en los que un fallo pueda causar mayores problemas, también se asocian con colores más oscuros. (Pressman, Ingeniería del software un enfoque práctico, 2002) Así, aparece una familia de metodologías:

- ✓ **Clear**, para equipos de hasta 8 personas o menos.

- ✓ **Amarillo**, de entre 10 y 20 personas.
- ✓ **Naranja**, para equipos entre 20 y 50 personas.
- ✓ **Roja**, entre 50 y 100 personas.

A más personas en el proyecto, más coordinación. A más criticidad en el software, más rigurosidad en el proceso. El factor más determinante, en cualquier caso, la comunicación entre los participantes en el proyecto. Las metodologías Crystal cumplen todas ellas con 7 propiedades esenciales, las siguientes:

- ✓ Entregas frecuentes, en base a un ciclo de vida iterativo e incremental. En función del proyecto puede haber desde entregas semanales hasta trimestrales.
- ✓ Mejora reflexiva. Que viene a ser mejora continua. Las iteraciones ayudan a ir ajustando el proyecto, a ir mejorándolo.
- ✓ Comunicación osmótica. Traducido al castellano, que el equipo esté en una misma ubicación física, para lograr la comunicación cara a cara.
- ✓ Seguridad personal. Todo el mundo puede expresar su opinión sin miedos, teniéndosele en cuenta, considerándose su opinión, etc.
- ✓ Enfoque. Períodos de no interrupción al equipo (2h horas), objetivos y prioridades claros, definiendo así tareas concretas.
- ✓ Fácil acceso a usuarios expertos. Las Crystal (a diferencia de otras como XP) no exigen que los usuarios estén continuamente junto al equipo de proyecto (no todas las organizaciones pueden hacerlo), sí que, como mínimo, semanalmente debe haber reuniones y los usuarios deben estar accesibles.
- ✓ Entorno técnico con pruebas automatizadas, gestión de la configuración e integración continua.

La estimación de riesgo y prioridad en crystal está dada por la cantidad de personas que integren el proyecto, se ajusta a esto y puede ser muy cambiante, pero es menos eficiente que XP ya que considera una metodología menos disciplinada.

Método de desarrollo de sistemas dinámicos (MDSD)

MDSD es un marco de trabajo creado para entregar la solución correcta en el momento correcto. Utiliza un ciclo de vida iterativo, fragmenta el proyecto en periodos cortos de tiempo y define entregables para cada uno de estos periodos. Tiene roles claramente definidos y especifica su trabajo dentro de periodos de tiempo. El Consorcio MDSD es el responsable del mantenimiento de esta metodología. Los principios de DSDM son: la necesidad del negocio como eje central; las entregas a tiempo; la colaboración; nunca comprometer la calidad; construir de modo incremental sobre una base sólida; el desarrollo iterativo; la comunicación clara y continua; y la demostración de control. (SCHWALBE, 2013)

Esta tecnología no posee una estimación de prioridad y riesgo concreta, se basa en la cantidad de tareas asignadas y el criterio del desarrollador, por lo tanto, no es muy eficiente.

Desarrollo adaptativo de software (DAS)

ASD tiene como fundamento la teoría de sistemas adaptativos complejos. Por ello, interpreta los proyectos de software como sistemas adaptativos complejos compuestos por agentes los interesados, entornos organizacionales, tecnológico y salidas el producto desarrollado. El ciclo de vida de ASD está orientado al cambio y se compone de las fases: especulación, colaboración y aprendizaje. Estas fases se caracterizan por estar enfocadas en la misión, estar basadas en características, ser iterativas, tener marcos de tiempo especificados, ser orientadas por los riesgos y ser tolerantes a los cambios. (Coram, 1996)

La estimación de prioridad y riesgo en esta tecnología es muy variante, se utiliza para grandes proyectos y da como resultado valores muy altos y costos.

Desarrollo orientado a funcionalidades (DOF)

DOF tiene como rasgo característico la planeación y el diseño por adelantado. En consecuencia, el modelo de objetos, la lista de características y la planeación se hacen al inicio del proyecto. Las iteraciones son incrementos con características identificadas.

Las prácticas que DOF pregona son: el modelado de objetos de dominio, el desarrollo por características, autoría de clases, los equipos de características,

las inspecciones, la construcción regular de planificación, la gestión de configuración y los reportes y visibilidad de los resultados.

Su ciclo de vida está compuesto por cinco etapas: el desarrollo de un modelo general, la construcción de la lista de características, la planeación por característica, el diseño por característica y la construcción por característica. FDD se enfoca en las fases de diseño, diseño por característica y desarrollo construcción por característica siendo estas las etapas del proceso que se iteran. (Booch, 1999)

Al ser una metodología muy ramificada en etapas no le presta la atención suficiente a la estimación del riesgo y la prioridad, priorizando otras fases para lograr más rápido el resultado final sin importar el costo.

Después de analizadas una serie de metodologías ágil de desarrollo populares hoy en día, llegamos a la conclusión que todas estiman la prioridad y el riesgo basados en criterios y opiniones ya sea del Cliente, Equipo de Desarrollo o Programador, dando gran margen al error y poca importancia a este proceso. A continuación, una tabla de los responsables de estimación en las distintas metodologías estudiadas.

Metodología Ágil de Desarrollo	Estimación de Prioridad	Estimación de Riesgo
XP	Programador y Cliente	Programador y Cliente
SCRUM	Equipo de Desarrollo	Dueño del Producto
CRYSTAL	Equipo de Desarrollo	Equipo de Desarrollo
MDSD	Criterio del Desarrollador	Criterio del Desarrollador
DAS	Programador	Programador
DOF	Programador	Programador

1.3 Técnicas utilizadas en la investigación

En el presente epígrafe se hace una breve explicación de las principales técnicas utilizadas en la investigación; así como una descripción de las definiciones más importantes asociadas a la misma. Esto garantiza una mejor compresión de los términos utilizados en la descripción del procedimiento propuesto.

1.3.1 Requisitos funcionales

Un requisito funcional define una función del sistema de software o sus componentes. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Los requisitos funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que se supone, un sistema debe cumplir. Los requisitos de comportamiento para cada requisito funcional se muestran en los casos de uso. Son complementados por los requisitos no funcionales, que se enfocan en cambio en el diseño o la implementación. Como se define en la ingeniería de requisitos, los requisitos funcionales establecen los comportamientos del software. Típicamente, un analista de requisitos genera requisitos funcionales después de realizar los casos de uso. Sin embargo, esto puede tener excepciones, ya que el desarrollo de software es un proceso iterativo y algunos requisitos son previos al diseño de los casos de uso. Ambos elementos (casos de uso y requisitos) se complementan en un proceso bidireccional.

Un requisito funcional típico contiene un nombre, un número de serie único y un resumen. Esta información se utiliza para ayudar al lector a entender por qué el requisito es necesario, y para seguir al mismo durante el desarrollo del producto. El núcleo del requisito es la descripción del comportamiento requerido, que debe ser clara y concisa. Este comportamiento puede provenir de reglas organizacionales o del negocio, o ser descubiertas por interacción con usuarios, inversores y otros expertos en la organización. (Sommerville, 2010)

1.3.2 Lenguaje de Programación C#

C# (pronunciado si sharp en inglés) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común. Su sintaxis básica deriva de C/C++ y

utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes. El nombre C Sharp fue inspirado por el signo '#' que se compone de cuatro signos '+' pegados. Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows Microsoft, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux. (Wikipedia, 2018)

Cabe destacar que los controles comunes que ofrece la plataforma .NET se pueden personalizar y/o editar para satisfacer las diferentes necesidades de los desarrolladores. El tomar la decisión de crear un control personalizado llega cuando se desea hacer una componente en donde se tiene el control total sobre su aspecto funcional y visual; con la posibilidad de no cargar las funcionalidades innecesarias para el desarrollo. (Center, 2016)

Los casos comunes en dónde se suelen usar estas características son:

- ✓ Controles simples con funcionalidad limitada (como un botón al que se le agrega movimiento).
- ✓ Controles en dónde se tengan características innecesarias para el desarrollo (un botón sin los márgenes).
- ✓ Controles en dónde se necesite la misma funcionalidad en un diseño diferente (botones en forma de línea para representar una arista).

1.3.3 Grafos

En matemáticas y ciencias de la computación, un grafo (del griego grafos: dibujo, imagen) es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto. Son objeto de estudio de la teoría de grafos. Típicamente, un grafo se representa gráficamente como un conjunto de puntos (vértices o nodos) unidos por líneas (aristas). Desde un punto de vista práctico, los grafos permiten estudiar las interrelaciones entre unidades que interactúan unas con otras. Por ejemplo, una red de computadoras puede representarse y

estudiarse mediante un grafo, en el cual los vértices representan terminales y las aristas representan conexiones (las cuales, a su vez, pueden ser cables o conexiones inalámbricas). Prácticamente cualquier problema puede representarse mediante un grafo, y su estudio trasciende a las diversas áreas de las ciencias exactas y las ciencias sociales. (Pressman, Introduction to Algorithms, 2002)

1.3.4 Grafos de dependencias

Si un atributo b en un nodo de un árbol de análisis sintáctico depende de un atributo c, entonces se debe evaluar la regla semántica para b en ese nodo después de la regla semantica que define a c. Las interdependencias entre los atributos heredados y sintetizados en los nodos de un árbol de análisis sintáctico se pueden representar mediante un grafo dirigido a cíclico. (Pressman, Introduction to Algorithms, 2002)

1.3.5 Matriz de riesgos

Una matriz de riesgos es una sencilla pero eficaz herramienta para identificar los riesgos más significativos inherentes a las actividades de una empresa, tanto de procesos como de fabricación de productos o puesta en marcha de servicios. Por lo tanto, es un instrumento válido para mejorar el control de riesgos y la seguridad de una organización. (Gamboa, 2011)

A través de este instrumento se puede realizar un diagnóstico objetivo y global de empresas de diferentes tamaños y sectores de actividad. Asimismo, mediante la matriz de riesgo es posible evaluar la efectividad de la gestión de los riesgos, tanto financieros como operativos y estratégicos, que están impactando en la misión de una determinada organización. (Beck, 2005)

Características de la matriz de riesgo:

Con el fin de garantizar su eficacia y utilidad, una matriz de riesgo debe tener las siguientes características:

- ✓ Debe ser flexible.
- ✓ Sencilla de elaborar y consultar.
- ✓ Que permita realizar un diagnóstico objetivo de la totalidad de los factores de riesgo.

- ✓ Ser capaz de comparar proyectos, áreas y actividades.

1.3.7 Búsqueda en Anchura (BFS)

Es un algoritmo para recorrer o buscar elementos en un grafo (usado frecuentemente sobre árboles). Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran todos los vecinos de este nodo. A continuación, para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.

Formalmente, BFS (en inglés BFS - Breadth First Search) es un algoritmo de búsqueda sin información, que expande y examina todos los nodos de un árbol sistemáticamente para buscar una solución. El algoritmo no usa ninguna estrategia heurística. (Helm, 2016)

Procedimiento

- ✓ Dado un vértice fuente S, BFS sistemáticamente explora los vértices de G para “descubrir” todos los vértices alcanzables desde S.
- ✓ Calcula la distancia (menor número de vértices) desde S a todos los vértices alcanzables.
- ✓ Después produce un árbol BF con raíz en S y que contiene a todos los vértices alcanzables.
- ✓ El camino desde S a cada vértice en este recorrido contiene el mínimo número de vértices. Es el camino más corto medido en número de vértices.

Su nombre se debe a que expande uniformemente la frontera entre lo descubierto y lo no descubierto. Llega a los nodos de distancia k, sólo tras haber llegado a todos los nodos a distancia k-1.

Código

La nomenclatura adicional utilizada es: Q = Estructura de datos cola como se muestra en la Ilustración 1.1

```
BFS(grafo G, nodo_fuente s)
  {
     // recorremos todos los vértices del grafo inicializándolos a NO_VISITADO,
     // distancia INFINITA y padre de cada nodo NULL
     for u ∈ V[G] do
     {
        estado[u] = NO_VISITADO;
        distancia[u] = INFINITO; /* distancia infinita si el nodo no es alcanzable */
        padre[u] = NULL;
     }
     estado[s] = VISITADO;
     distancia[s] = 0;
     padre[s] = NULL;
     CrearCola(Q); /* nos aseguramos que la cola está vacía */
     Encolar(Q, s);
     while !vacia(Q) do
     {
        // extraemos el nodo u de la cola Q y exploramos todos sus nodos adyacentes
        u = extraer(Q);
        for  v □ adyacencia[u]  do
        {
           if estado[v] == NO_VISITADO then
           {
               estado[v] = VISITADO;
               distancia[v] = distancia[u] + 1;
               padre[v] = u;
               Encolar(Q, v);
           }
        }
     }

```

Ilustración *1.1 Código en C# para el algoritmo de búsqueda a lo ancho (BFS)*

1.4 Conclusiones del capítulo

Luego de analizar el estado actual de la temática y sus antecedentes, así como las principales técnicas y recursos utilizados, se concluye que:

1. Los procedimientos existentes presentan no siempre son los más adecuados en cuanto a estimación de refiere.
2. El análisis del estado del arte y de los antecedentes de la temática consolida las bases para el desarrollo de la solución del problema planteado.
3. Las técnicas y recursos utilizados son las adecuadas para la solución del problema.
4. La creación de un algoritmo basado en la teoría de grafos permitirá mejorar los índices de estimación en las tareas de prioridad y riesgo de las historias de usuario.

En sentido general se ha contribuido a una mejor comprensión del objeto de estudio y se han establecido las bases para las siguientes fases de la investigación.

Capítulo 2: Descripción del procedimiento propuesto

2.1. Introducción

Las técnicas utilizadas para la estimación de prioridad y riesgo en requisitos funcionales de las metodologías ágiles no presentan homogeneidad en para su obtención, lo que se traduce en ineficiencia en la adquisición de estos parámetros a la hora de realizar el diseño de un proyecto de software. Por lo que en este capítulo se propone un procedimiento para la estimación de estos parámetros, basado en un algoritmo sustentado en la teoría de grafos.

2.2 Principales errores en la estimación en Metodologías Ágiles

La mayoría de los errores que se producen con más frecuencia en los proyectos software están relacionados con aspectos de estimación. (Jimenez, 2014) Entre estos errores se pueden destacar:

- ✓ **Calendario optimista**: la tendencia al estimar es hacerlo de manera optimista. Esta tendencia es general y se va disminuyendo con la experiencia. Aun así, en un histórico de estimaciones estimar por encima es mucho menos frecuente que estimar por debajo.
- ✓ **Expectativas no realistas:** muchas veces las tareas sobre las que se ha estimado son ambiguas. Posteriormente el equipo seguramente se llega a un consenso común pero el cliente puede mantener otra expectativa. Y presionará para obtener el resultado que él cree es el correcto.
- ✓ **Confundir estimaciones con objetivos:** al estimar tenemos que tener en cuenta nuestra capacidad actual real y el histórico de productividad de la empresa. Es decir, si queremos terminar en 3 meses (un objetivo) puede que lo consigamos o no sabiendo que contamos con 2 programadores y que el 20% del tiempo estamos respondiendo incidencias de otros proyectos.
- ✓ **Omisión de estimar tareas necesarias:** las pruebas, la documentación, las tareas relacionadas con la gestión de configuración o la calidad puede que no se planifiquen. Pero muchas veces consumen tiempo.

2.3 Análisis cualitativo de prioridades

La tarea de asignar prioridades requiere de la participación de clientes y usuarios con cierto nivel de decisión y puede realizarse de diversas maneras, tales como

reuniones, cuestionarios y otras. Se debe determinar la importancia relativa que tiene un requisito para los clientes y usuarios, y organizar aquellos requisitos que deben implementarse inicialmente frente a aquellos que pueden postergarse. Al asignar prioridades, se deben tener en cuenta la dependencia entre requisitos, la multiplicidad de intereses de los clientes y usuarios, las limitaciones de recursos, las necesidades del negocio, las imposiciones del mercado y los costos de implementación, entre otros factores. Por ende, los ingenieros de software deberán orientar a los clientes y usuarios respecto a estas contingencias.

2.4 Análisis cualitativo de riesgos

El análisis cualitativo de riesgos permite establecer prioridades para posteriormente crear el plan de respuesta a los riesgos; es también una base para el análisis cuantitativo, y es importante que este análisis sea revisado y actualizado durante todo el ciclo de vida del proyecto. El análisis cualitativo involucra la evaluación de probabilidad e impacto de los riesgos que han sido identificados en el proyecto; en este caso se le da un valor cualitativo a la probabilidad (por ejemplo: baja, media, alta) y al impacto (por ejemplo: bajo, medio, alto). Los riesgos negativos que tienen probabilidad e impacto más altos son los que pueden afectar más gravemente al proyecto, mientras que los riesgos positivos aumentan las oportunidades de que se presente un evento que beneficie el desarrollo del mismo, razón por la cual se les debe diseñar un adecuado plan de respuesta. Existen diferentes técnicas tales como: la evaluación de probabilidad e impacto de los riesgos, matriz de probabilidad e impacto, la evaluación de la calidad de los datos sobre los riesgos, y la evaluación de la urgencia de los mismos. Es importante aclarar que, para realizar el análisis cualitativo, se debe tener un registro de los riesgos del proyecto, el cual se obtiene en el proceso de identificarlos. (Morales, 2010)

2.5 Procedimiento General

El procedimiento general se divide en un conjunto de pasos necesarios para su funcionamiento. Primeramente, se debe recoger los requisitos funcionales correspondientes al desarrollo de la metodología, con estos se elabora un resumen de los requisitos funcionales y sus dependencias, se establece

una primera prioridad para cada una basada en el criterio del cliente según su importancia en el negocio. Este criterio es importante ya que es el que normalmente utilizan todas las metodologías para calcular la prioridad, posteriormente comparamos este criterio con el resultado obtenido por el algoritmo.

Paso 1:

La tabla muestra un ejemplo de determinación de las prioridades por requisitos definidas por un cliente, y las dependencias entre ellos.

No	Requisitos	Prioridad Cliente	Requisitos que dependen de el
1	Gestionar Procesos	Baja	Ninguno
2	Gestionar Auditoría	Alta	1, 5, 6, 8
3	Gestionar Importancia	Media	7
4	Gestionar Probabilidades	Media	1
5	Gestionar Tipo de Auditoría	Baja	Ninguno
6	Gestionar Tipo de Riesgo	Baja	Ninguno
7	Gestionar Actividad de Empresa	Media	1
8	Gestionar Fases de Auditoria	Baja	Ninguno
9	Autenticarse en el Sistema	Alta	1, 2, 3, 4, 5, 6, 7, 8

Tabla 2.1 Listado de requisitos funcionales

Paso 2:

Posteriormente se elabora un grafo donde cada nodo seria el requisito funcional y sus conexiones serían las dependencias entre estos. Este es el grafo que se utiliza para la corrida del algoritmo.

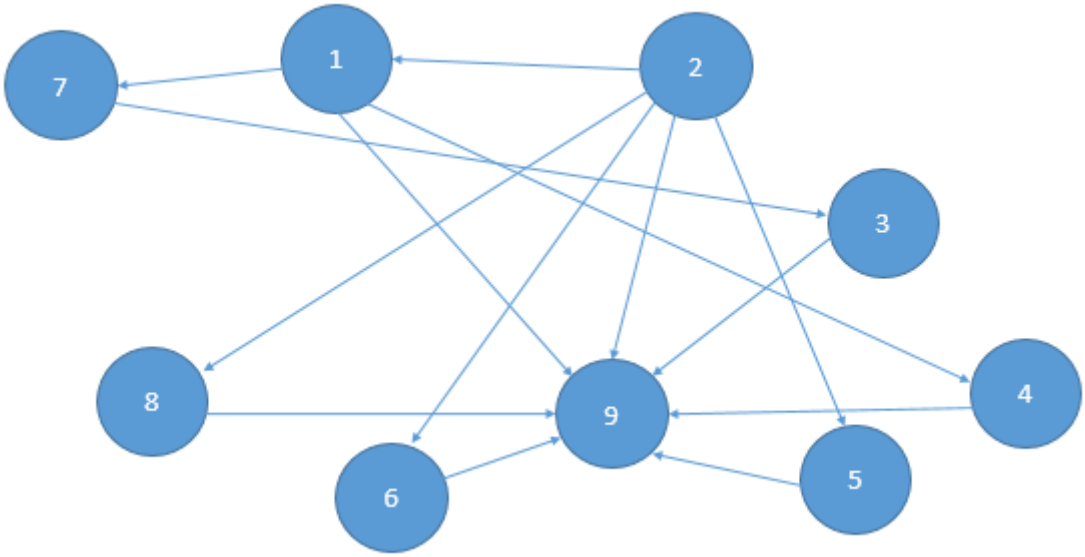

Ilustración 2.2 Grafo Elaborado a partir de los requisitos

Paso 3:

Después de elaborado el grafo se recorre a lo ancho con el algoritmo de búsqueda BFS, **se utiliza dicho algoritmo** porque es muy importante realizar la búsqueda por niveles y no a profundidad puesto que existen tareas que dependen unas de otras creando un grafo casi conexo, este algoritmo es un poco más costoso en cuanto a tiempo porque es de orden O(N*N), pero eficiente puesto que no queda ningún nodo sin visitar al final del proceso. Es importante visitar todos los nodos, aunque estos no tengan dependencias, puesto esto garantiza la certeza del algoritmo. A medida que visitamos un nodo le asignamos un valor temporal de 1 y cada vez que en cierto punto del algoritmo volvemos a visitar ese nodo se incrementa su valor en el **valor de su nodo dependiente más su valor actual**. Puede que en el proceso existan nodos que no se visiten nunca, puesto que hay requisitos funcionales que tienen dependencias, pero nunca dependen de ellos otros requisitos funcionales. Al final del algoritmo quedaran visitados todos los nodos y los más dependientes se

le asignara mayor prioridad en el proyecto dejando para el final los nodos cuyo valor es mínimo. De esta manera logramos abarcar todos los nodos y quedarían organizados por prioridad.

La tabla muestra una corrida BFS para el grafo obtenido.

NODOS / ITERACIONES	1	2	3	4	5	6	7	8	9
1ra	1	1	1	1	1	1	1	1	1
2da	1	5	1	1	1	1	1	1	1
3ra	1	5	2	1	1	1	1	1	1
4ta	1	5	2	2	1	1	1	1	1
5ta	1	5	2	2	1	1	1	1	1
6ta	1	5	2	2	1	1	1	1	1
7ma	1	5	2	2	1	1	2	1	1
8va	1	5	2	2	1	1	2	1	1
9na	1	5	2	2	1	1	2	1	16

Tabla 2.3 Tabla de Nodos vs Iteraciones

Paso 4:

Al final del algoritmo señalamos las tareas con una prioridad mayor que la media en rojo, las cercanas o igual a la media en verde y las inferiores en verde. De esta manera quedan definidas cuales son los requisitos que mayor prioridad necesitan por parte del equipo de trabajo.

Requisitos funcionales	Prioridad según cliente	Prioridad según el algoritmo
1	Baja	Baja

2	Alta	Alta
3	Media	Media
4	Media	Media
5	Baja	Baja
6	Baja	Baja
7	Media	Media
8	Baja	Baja
9	Alta	Alta

Tabla 2.4 Prioridad Cliente vs Prioridad del Algoritmo

Cuando las prioridades de los requisitos son iguales en algunos casos, debemos tomar para comparar la prioridad inicial del cliente, para ver que tarea realizar primero, si aún existe igualdad de prioridades se programa una reunión con el cliente para redefinir cuales requisitos desea realizar primero.

Paso 5:

Posteriormente procedemos a calcular el riesgo de cada tarea, para esto necesitaremos la probabilidad e impacto de las mismas. Un problema típico en el momento de realizar la evaluación de probabilidad e impacto es que no se cuenta con una base para realizar la asignación de estos valores a cada riesgo; muchas veces se asignan según el criterio de la persona que está evaluando los riesgos, y puede que se estén pasando por alto algunos puntos importantes; por ello se ha utiliza la matriz Probabilidad – Impacto.

La matriz de Probabilidad – Impacto es una herramienta de análisis cualitativo de riesgos que nos permite establecer prioridades en cuanto a los posibles riesgos de un proyecto en función tanto de la probabilidad de que ocurran como de las repercusiones que podrían tener sobre nuestro proyecto en caso de que ocurrieran. (Gómez, 2017)

Como se aprecia en la siguiente ilustración, la matriz se compone de dos ejes: un eje vertical en donde se establecen los valores de probabilidad (entre 0 – imposible y 1 – siempre) y un eje horizontal en donde se establecen los valores del impacto del riesgo sobre los objetivos de nuestro proyecto (en donde 0 implica que ese riesgo no repercutiría en los objetivos y 1 que dificultaría en gran medida el cumplimiento de los mismos). Los valores obtenidos en las diferentes celdas de la matriz son el resultado de multiplicar la probabilidad de ocurrencia por el impacto del riesgo, indicando los valores más altos (máximo 1) los riesgos más críticos del proyecto y los más bajos los menos relevantes.

RIESGO					
Impacto / Probabilidad	Muy Bajo .05	Bajo .1	Moderado .2	Alto .4	Muy Alto .8
0.9	0.05	0.09	0.18	0.36	0.72
0.7	0.04	0.07	0.14	0.28	0.56
0.5	0.03	0.05	0.10	0.20	0.40
0.3	0.02	0.03	0.06	0.12	0.24
0.1	0.01	0.01	0.02	0.04	0.08

Verde – Riesgo Bajo Amarillo – Riesgo Moderado Rojo – Riesgo Alto

Tabla 2.5 Matriz de Probabilidad – Impacto

Los índices cualitativos o rangos que serían los siguientes:

Probabilidad

- ✓ **Cierto**: probabilidad muy alta (**0.9**)
- ✓ **Probable**: probabilidad alta (**0.7**)
- ✓ **Posible**: probabilidad media (**0.5**)
- ✓ **Improbable**: probabilidad baja (**0.3**)
- ✓ **Excepcional**: sería especialmente raro que ocurriera (**0.1**)

Impacto

- ✓ **Catastrófico**: pérdida de negocio (**0.8**)

- ✓ **Crítico**: afección grave al negocio (**0.4**)
- ✓ **Moderado**: causarán problemas no significativos en el negocio (**0.2**)
- ✓ **Marginal**: muy poca influencia sobre el negocio, impacto leve (**0.1**)
- ✓ **Despreciable**: prácticamente ninguna influencia negativa sobre el negocio, pueden dejarse sin mediar (**0.05**)

Paso 6:

Basado en estos factores y en la prioridad asignamos una probabilidad e impacto correspondiente a la prioridad de nuestro proyecto, a más alta prioridad existe una mayor probabilidad de que un evento ocurra y tendría más impacto sobre el sistema, lo que daría como resultado la siguiente tabla:

Requisito	Prioridad	Probabilidad	Impacto	Riesgo
1	Baja	0.3	0.1	0.03
2	Alta	0.9	0.8	0.72
3	Media	0.5	0.2	0.10
4	Media	0.5	0.2	0.10
5	Baja	0.3	0.1	0.03
6	Baja	0.3	0.1	0.03
7	Media	0.5	0.2	0.10
8	Baja	0.3	0.1	0.03
9	Alta	0.9	0.8	0.72

Tabla 2.6 Tabla de Riesgos Resultantes

Los pesos y valores asignados se obtienen de la revisión de casos desarrollados, de la literatura y de los análisis y acuerdos obtenidos por el equipo de trabajo del

proyecto. Cabe aclarar que estos valores pueden ser sujetos de modificación, según se considere por el equipo de trabajo del proyecto. (Saez, 2015)

2.6 Prototipo para la determinación de prioridad y riesgo

Después de obtenido el algoritmo se desarrolla un prototipo de aplicación, empleando el lenguaje de programación C#, que estima el cálculo de la prioridad y riesgo implementando dicho algoritmo. Para calcular la prioridad de un proyecto primero necesitamos la cantidad de requisitos que funcionaran como nodos del grafo a construir, posteriormente se insertan las dependencias entre los requisitos que serán las conexiones de los nodos en dicho grafo, al hacer clic en la opción estimar la aplicación construye un grafo tomando como vértices los requisitos y aristas sus dependencias. Al quedar construido el grafo la aplicación utiliza un método de búsqueda a lo ancho optimizado para estos casos en especial. Al finalizar la corrida a lo ancho la aplicación muestra el requisito, la cantidad de dependencias encontradas en la corrida del algoritmo y una estimación de la prioridad basada en la media de los resultados obtenidos.

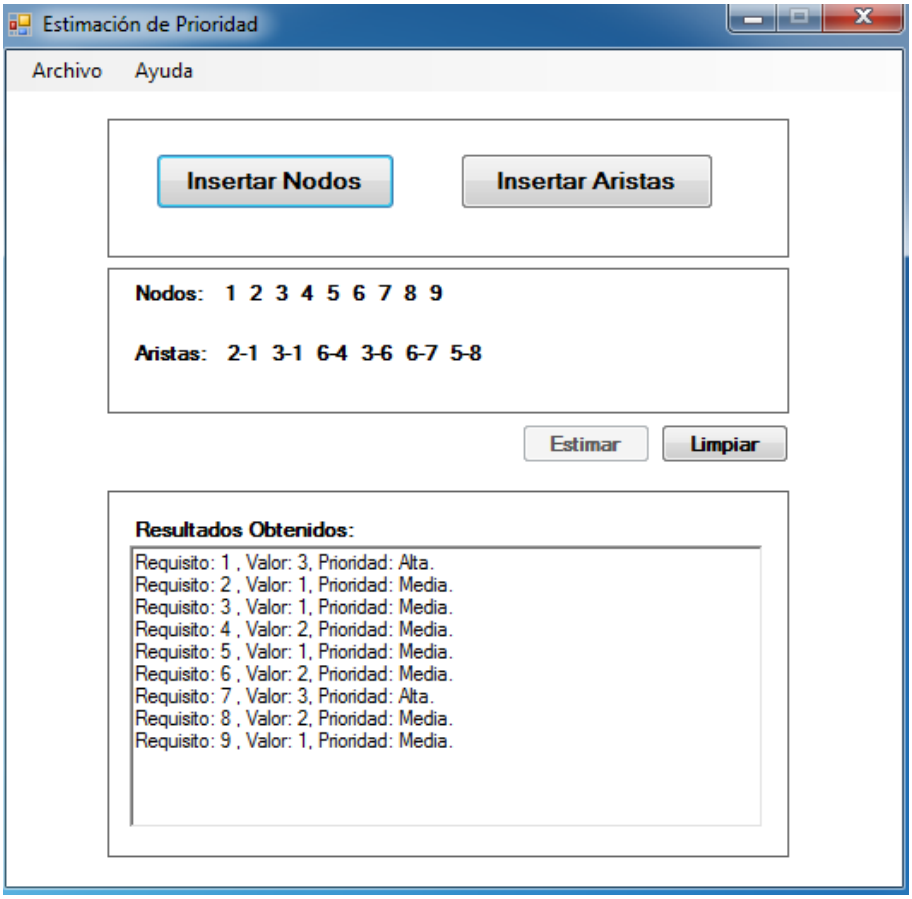

Ilustración 2.7 Pantalla de la aplicación correspondiente a la entrada de información

Para realizar el cálculo de riesgos la aplicación requiere la entrada de los valores de impacto y probabilidad, al ser introducidos dichos valores se realiza el cálculo basado en la fórmula de **riesgo = impacto * probabilidad** con la que se elaboró la tabla 4 y se muestran los resultados pertinentes al requisito funcional dado.

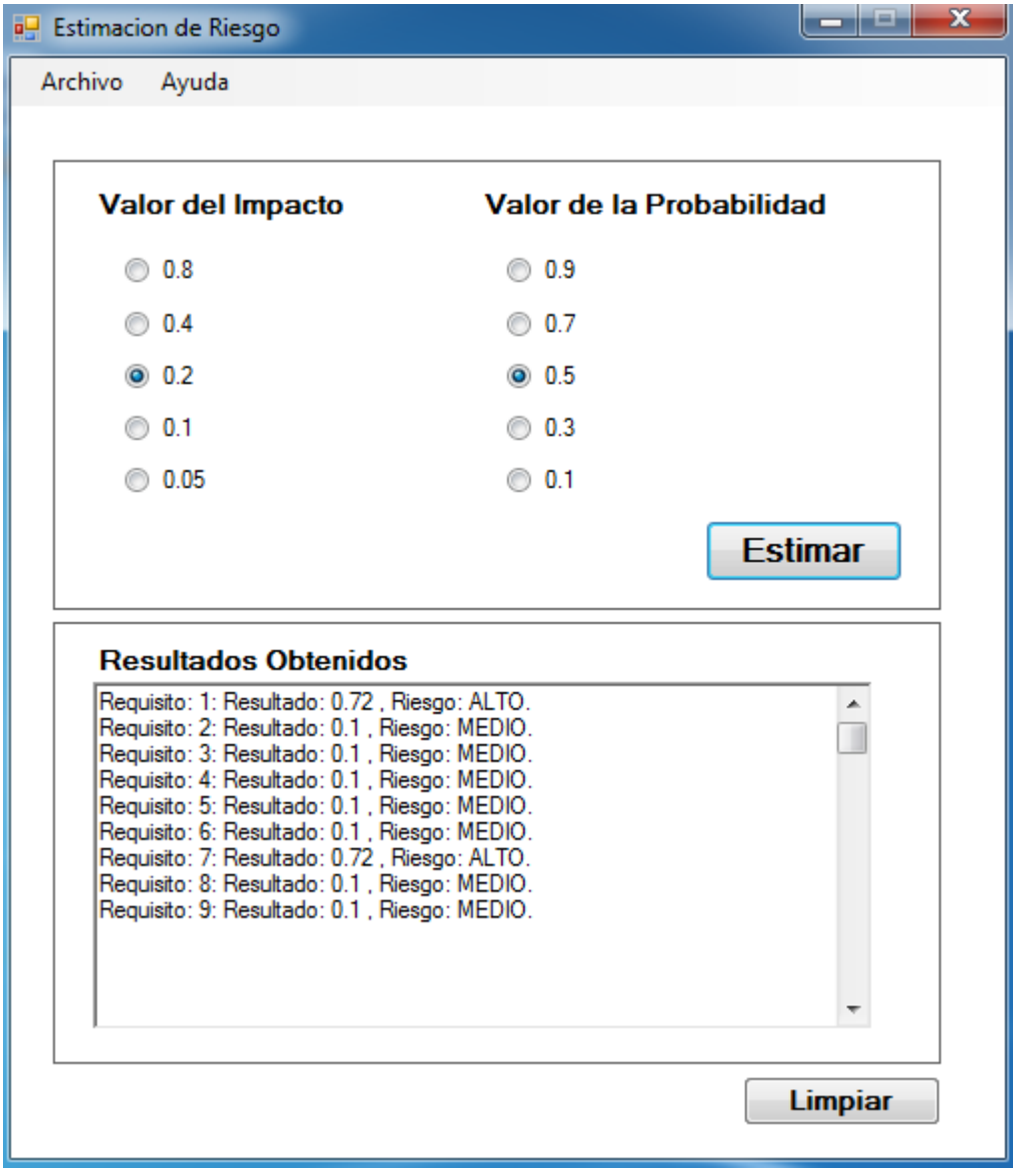

Ilustración 2.8 Pantalla de la aplicación correspondiente al cálculo de riesgos

2.7 Conclusiones del Capítulo

Con el procedimiento antes descrito, se demuestra que:

1. Se determinaron los errores más comunes en la determinación de la prioridad y riesgo de los requisitos funcionales de las metodologías ágiles.
2. Se elaboró un procedimiento para estimar indicadores de prioridad y riego basado en teoría de grafos, logrando valores significativos para estos índices que se derivan de la simulación y la estadística.
3. Se empleó la estrategia de búsqueda a ciega BFS para recorrer el grafo de dependencia y se demostró su pertinencia.
4. Se empleó la matriz de riesgo para analizar el indicador riesgo de los requisitos funcionales.

Capítulo 3: Experimentación discusión y análisis de resultados.

3.1. Introducción

Una vez obtenida la propuesta de procedimiento solo resta corroborar los resultados que se obtienen de su aplicación. Las diferentes pruebas fueron realizadas sobre documentaciones de sistemas realizados como trabajos de diploma de la carrera Ingeniería Informática de la Universidad de Matanzas. Se seleccionaros los trabajos que fueron diseñados empleando metodologías agiles, para comprobar que los resultados ofrecidos por el sistema son los acertados. Se realizaron diferentes pruebas, con el objetivo de mejorar los resultados con cada una de ellas, apoyándose en el prototipo desarrollado.

En el presente capítulo se explican todos los ensayos realizados, así como un pequeño análisis y comparación de los mismos. Además, se señalarán los aspectos más notables de cada uno de ellos.

3.2. Pruebas realizadas

La corrida del algoritmo consiste en insertar todos los nodos en la aplicación, posteriormente establecemos sus dependencias insertando las aristas y nos aseguramos que tenemos toda la información correcta insertada en la aplicación.

Al hacer click en el botón estimar comienza la iteración del BFS recorriendo todos los nodos y asignando un valor de 1 y el valor que contienen sus dependencias, de esta manera al final del algoritmo en la cola quedan los nodos con mayor peso y estos obtienen una prioridad mayor comparándolos contra la media.

La corrida del algoritmo también incluye la estimación de costo, se selecciona el valor de los nodos en la aplicación y posteriormente se hace click en botón estimar, la aplicación comparara el resultado contra la matriz de probabilidad e impacto y muestra los resultados obtenidos.

Con el fin de evaluar la exactitud del sistema se llevaron a cabo pruebas 20 tesis de diferentes temáticas. Primeramente, se elaboró una tabla con los requisitos funcionales para cada tesis, sus dependencias, la prioridad y el riesgo según su autor, posteriormente se realiza la corrida del algoritmo y se comparan los resultados obtenidos.

A continuación, se muestran detalles de cinco de las 20 pruebas realizadas.

3.2.1 Prueba 1 a Tesis Titulada "Aplicación web para la gestión de la información de la reserva en matanzas" (Matanzas, Aplicación web para la gestión de la información de la reserva en Matanzas , 2013)

Elaboramos la tabla de requisitos funcionales a partir de la documentación del sistema, se toma la prioridad y el riesgo del autor y se hallan las dependencias a partir de los mismos como se muestra en la tabla 3.1

Requisitos Funcionales	Nombre	Prioridad	Riesgo	Dependencias
1	Mostrar información del cuadro de mando.	Alta	Alto	2, 3
2	Gestionar usuario.	Alta	Alto	Ninguna
3	Gestionar depositario.	Alta	Alto	Ninguna
4	Gestionar almacén.	Alta	Alto	6
5	Gestionar unidad de medidas.	Media	Alto	Ninguna
6	Gestionar certificado de calidad.	Media	Alto	3
7	Gestionar multas.	Media	Alto	6
8	Gestionar Medidas.	Media	Alto	5

9	Registro de salva.	Alta	Alto	Ninguna

Tabla 3.1 Requisitos funcionales según el autor del documento

Posteriormente aplicamos el algoritmo de estimación de prioridad manualmente según sus dependencias, todas las corridas se realizaron sin ningún error, arrojando los resultados finales como se muestra en la tabla 3.2

NODOS / ITERACIONES	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1ra	3	1	1	1	1	1	1	1	1
2da	3	1	1	1	1	1	1	1	1
3ra	3	1	1	1	1	1	1	1	1
4ta	3	1	1	2	1	1	1	1	1
5ta	3	1	1	2	1	1	1	1	1
6ta	3	1	1	2	1	2	1	1	1
7ma	3	1	1	2	1	2	3	1	1
8va	3	1	1	2	1	2	3	2	1
9na	3	1	1	2	1	2	3	2	1

Tabla 3.2 Corrida manual del algoritmo de estimación de prioridad

Después se realizó la estimación de prioridad con la aplicación desarrollada, coincidiendo los resultados con la estimación manual, al estar seguros de los resultados finales realizamos una tabla comparativa de los valores de prioridad según el autor y los valores según el algoritmo como se muestra en la Ilustración 3.3

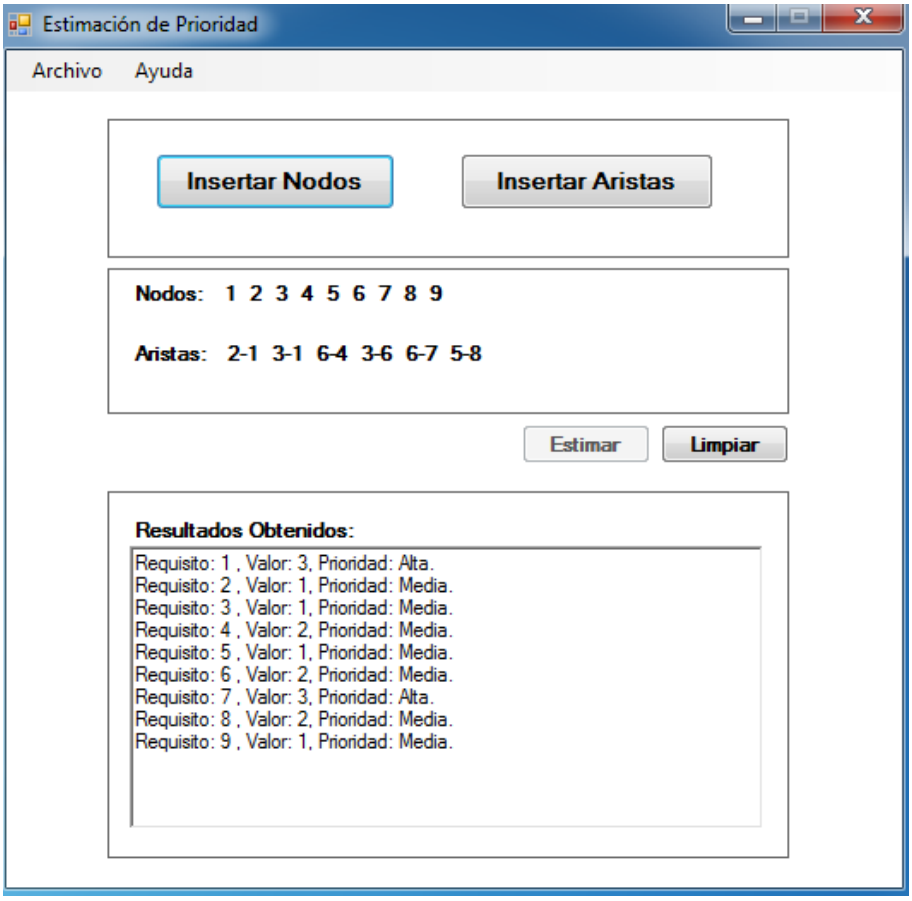

Ilustración 3.3 Corrida del algoritmo de estimación de prioridad utilizando la aplicación desarrollada

Requisitos funcionales	Prioridad según autor	Prioridad según el algoritmo
1	Alta	Alta
2	Alta	Media
3	Alta	Media
4	Alta	Media
5	Media	Media
6	Media	Media
7	Media	Alta

8	Media	Media
9	Alta	Media

Tabla 3.4 Comparación de los Resultados obtenidos

Analizando la tabla 3.4 podemos concluir que se logró una estimación más certera en los requisitos 2, 3, 4, 9 que tenían una estimación del autor alta y el algoritmo determino media. Lo que conlleva a que se puedan mejorar los tiempos de desarrollo y costos del proyecto ya que podemos determinar cuáles son realmente los requisitos más importantes y enfocarnos en ello.

Teniendo la prioridad segura procedemos a calcular el riesgo, tomamos como punto de partida la prioridad resultante del algoritmo y estimamos los valores más correctos de probabilidad e impacto, después aplicamos manualmente la fórmula de **Riesgo = Impacto * Probabilidad**, como se muestra en la tabla 3.5

Requisito	Prioridad	Probabilidad	Impacto	Riesgo
1	Alta	0.9	0.8	0.72
2	Media	0.5	0.2	0.10
3	Media	0.5	0.2	0.10
4	Media	0.5	0.2	0.10
5	Media	0.5	0.2	0.10
6	Media	0.5	0.2	0.10
7	Alta	0.9	0.8	0.72
8	Media	0.5	0.2	0.10
9	Media	0.5	0.2	0.10

Tabla 3.5 Estimación manual del algoritmo utilizando los valores obtenidos por la prioridad

Después de hallados los valores manualmente con el algoritmo procedimos a realizar la corrida de dichos valores en la aplicación desarrollada, coincidiendo los resultados del riesgo, como se muestra en la Ilustración 3.6

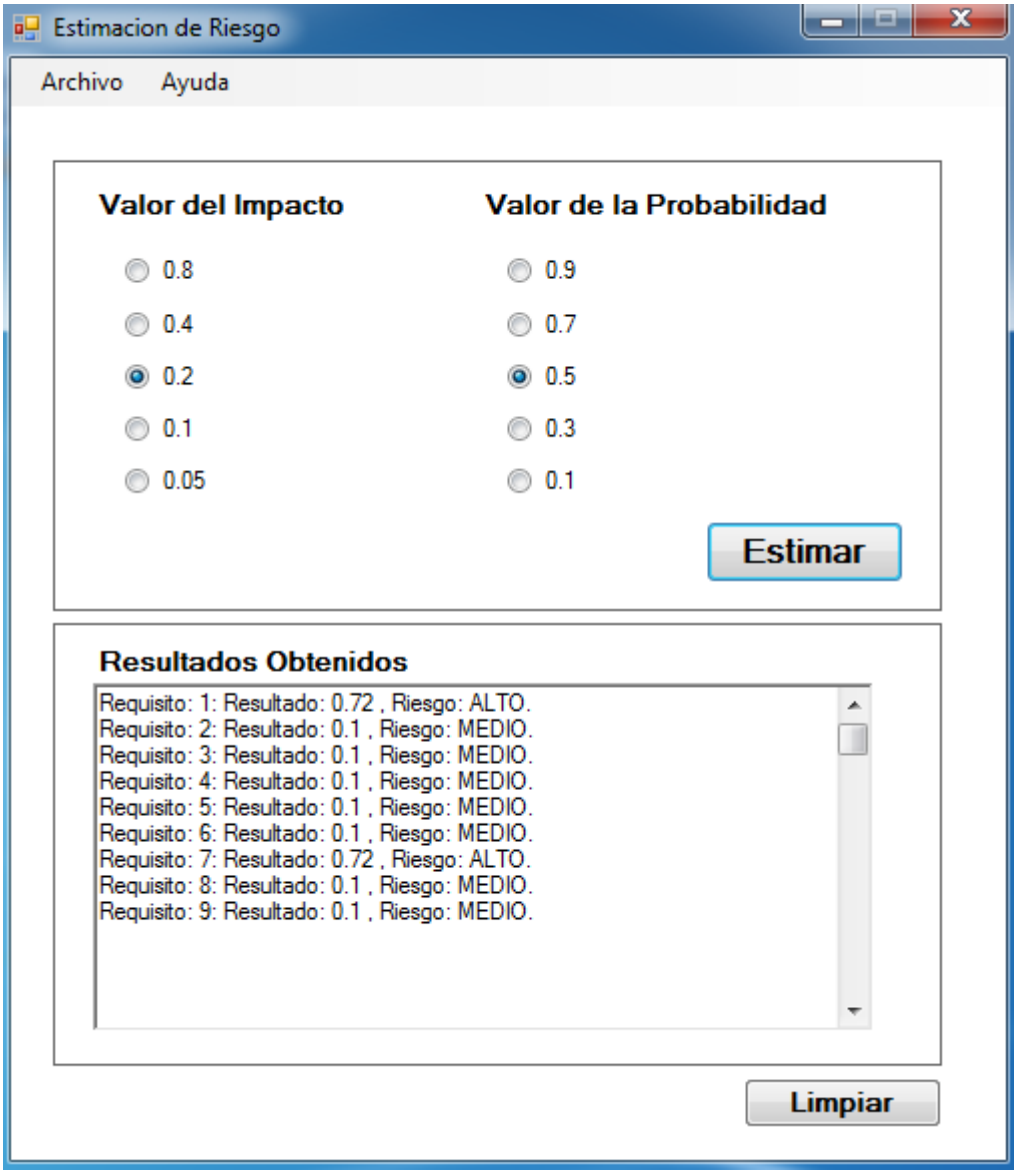

Ilustración 3.6 Resultados de la estimación de riesgo utilizando la aplicación desarrollada

Requisitos funcionales	Riesgo según autor	Riesgo según el algoritmo
1	Alto	Alto
2	Alto	Medio
3	Alto	Medio

4	Alto	Medio
5	Alto	Medio
6	Alto	Medio
7	Alto	Alto
8	Alto	Medio
9	Alto	Medio

Tabla 3.7 Comparación de los Resultados obtenidos

Analizando la tabla 3.7 podemos concluir que mejoraron los indicadores de riesgo en los requisitos 2, 3, 4, 9 que tenían una estimación del autor alta y el algoritmo determinó media, esto puede mejorar en el desarrollo del proyecto puesto que conocemos con seguridad que tareas tienen un impacto mayor sobre nuestra aplicación y debemos trabajar sobre estas para que no exista fallos graves en un futuro.

3.2.2 Prueba 2 a Tesis Titulada " (Matanzas, Aplicación Web para la gestión de la información en el procedimiento P 02-4 Comunicación con el Cliente en la UEB Divep Matanzas, 2014)"

Elaboramos la tabla de requisitos funcionales a partir de la documentación del sistema, se toma la prioridad y el riesgo del autor y se hallan las dependencias a partir de los mismos como se muestra en la tabla 3.8

Requisitos Funcionales	Nombre	Prioridad	Riesgo	Dependencias
1	Diseño y Creación de La Base de Datos	Alta	Alto	Ninguna
2	Diseño interfaz de Usuario	Alta	Alto	Ninguna
3	Autentificar	Media	Bajo	Ninguna

4	Gestionar Usuarios	Media	Medio	2, 5
5	Gestionar Roles y Permisos	Media	Alto	2
6	Gestionar Clientes	Alta	Alto	2, 4
7	Gestionar Tiendas	Alta	Alto	2
8	Gestionar Encuesta	Alta	Alto	2, 9
9	Gestionar Quejas y Reclamaciones	Alta	Alto	2
10	Generar Reporte	Media	Medio	2, 6, 7, 8, 9

Tabla 3.8 Requisitos funcionales según el autor del documento

Posteriormente aplicamos el algoritmo de estimación de prioridad manualmente según sus dependencias, todas las corridas se realizaron sin ningún error, arrojando los resultados finales como se muestra en la tabla 3.9

NODOS / ITERACIONES	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1ra	1	1	1	1	1	1	1	1	1
2da	1	7	1	1	1	1	1	1	1
3ra	1	7	1	1	1	1	1	1	1
4ta	1	7	1	2	1	1	1	1	1
5ta	1	7	1	2	3	1	1	1	1
6ta	1	7	1	2	3	1	1	1	1
7ma	1	7	1	2	3	1	1	1	1

8va	1	7	1	2	3	1	1	1	1
9na	1	7	1	2	3	1	1	2	1
10ma	1	7	1	2	3	1	1	2	1

Tabla 3.9 Corrida manual del algoritmo de estimación de prioridad

Después se realizó la estimación de prioridad con la aplicación desarrollada, coincidiendo los resultados con la estimación manual, al estar seguros de los resultados finales realizamos una tabla comparativa de los valores de prioridad según el autor y los valores según el algoritmo como se muestra en la Ilustración 3.10

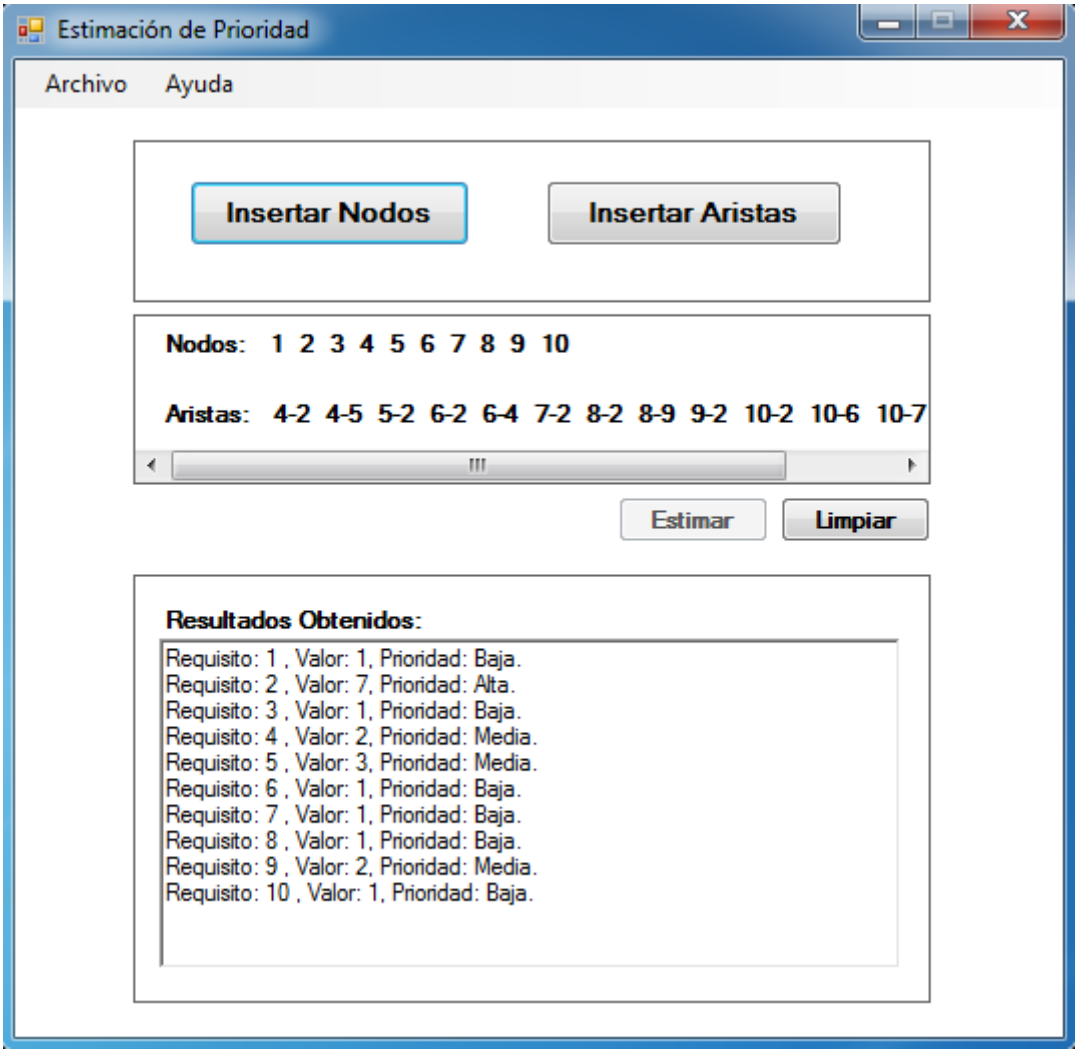

Ilustración 3.10 Corrida del algoritmo de estimación de prioridad utilizando la aplicación desarrollada

Requisitos funcionales	Prioridad según autor	Prioridad según el algoritmo
1	Alta	Baja
2	Alta	Alta
3	Media	Baja
4	Media	Media
5	Media	Media
6	Alta	Baja
7	Alta	Baja
8	Alta	Media
9	Alta	Baja
10	Media	Baja

Tabla 3.11 Comparación de los Resultados obtenidos

Analizando la tabla 3.11 podemos concluir que se logró una estimación más certera en los requisitos 3, 6, 7, 8, 9, 10 esto puede mejorar los tiempos de desarrollo y costos del proyecto ya que podemos determinar cuáles son realmente los requisitos más importantes y enfocarnos en ello.

Teniendo la prioridad segura procedemos a calcular el riesgo, tomamos como punto de partida la prioridad resultante del algoritmo y estimamos los valores más correctos de probabilidad e impacto, después aplicamos manualmente la fórmula de **Riesgo = Impacto * Probabilidad**, como se muestra en la tabla 3.12

Requisito	Prioridad	Probabilidad	Impacto	Riesgo
1	Baja	0.3	0.1	0.03
2	Alta	0.9	0.8	0.72

3	Baja	0.3	0.1	0.03
4	Media	0.5	0.2	0.10
5	Media	0.5	0.2	0.10
6	Baja	0.3	0.1	0.03
7	Baja	0.3	0.1	0.03
8	Media	0.5	0.2	0.10
9	Baja	0.3	0.1	0.03
10	Baja	0.3	0.1	0.03

Tabla 3.12 Estimación manual del algoritmo utilizando los valores obtenidos por la prioridad

Después de hallados los valores manualmente con el algoritmo procedimos a realizar la corrida de dichos valores en la aplicación desarrollada, coincidiendo los resultados del riesgo, como se muestra en la Ilustración 3.13

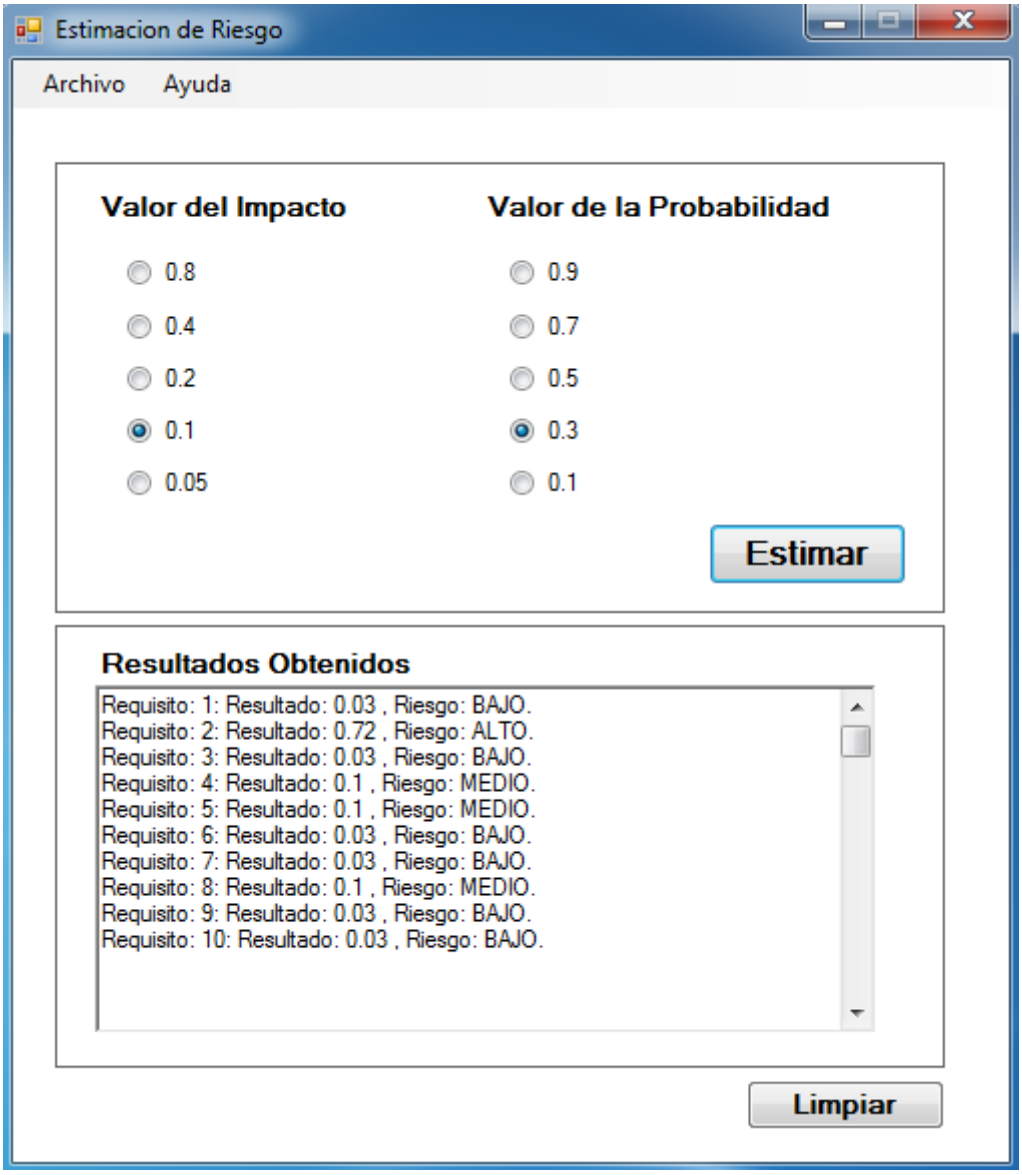

Ilustración 3.13 Resultados de la estimación de riesgo utilizando la aplicación desarrollada

Requisitos funcionales	Riesgo según autor	Riesgo según el algoritmo
1	Alto	Bajo
2	Alto	Alto
3	Bajo	Bajo

4	Medio	Medio
5	Alto	Medio
6	Alto	Bajo
7	Alto	Bajo
8	Alto	Medio
9	Alto	Bajo
10	Medio	Bajo

Tabla 3.14 Comparación de los Resultados obtenidos

Analizando la tabla 3.14 podemos concluir que mejoraron los indicadores de riesgo en los requisitos 1, 5, 6, 7, 8, 9, 10 que tenían una estimación del autor alta y el algoritmo determinó media, esto puede mejorar en el desarrollo del proyecto puesto que conocemos con seguridad que tareas tienen un impacto mayor sobre nuestra aplicación y debemos trabajar sobre estas para que no exista fallos graves en un futuro.

3.2.3 Prueba 3 a Tesis Titulada "Software de gestión y costeo por actividades (ABC/ABM)" (Matanzas, Software de gestión y costeo por actividades (ABC/ABM), 2015)

Elaboramos la tabla de requisitos funcionales a partir de la documentación del sistema, se toma la prioridad y el riesgo del autor y se hallan las dependencias a partir de los mismos como se muestra en la tabla 3.15

Requisitos Funcionales	Nombre	Prioridad	Riesgo	Dependencias
1	Gestionar usuarios	Media	Media	Ninguna
2	Gestionar actividades	Media	Media	Ninguna
3	Costear actividades	Alto	Alto	2

4	Gestionar recursos	Media	Media	Ninguna
5	Gestionar costos	Media	Media	Ninguna
6	Mostrar reportes por actividades costeadas	Media	Media	3, 4
7	Mostrar reportes de costos máximos mínimos	Media	Media	5

Tabla 3.15 Requisitos funcionales según el autor del documento

Posteriormente aplicamos el algoritmo de estimación de prioridad manualmente según sus dependencias, todas las corridas se realizaron sin ningún error, arrojando los resultados finales como se muestra en la tabla 3.16

NODOS / ITERACIONES	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1ra	1	1	1	1	1	1	1
2da	1	2	1	1	1	1	1
3ra	1	2	2	1	1	1	1
4ta	1	2	2	2	1	1	1
5ta	1	2	2	2	2	1	1
6ta	1	2	2	2	2	1	1
7ma	1	2	2	2	2	1	1

Tabla 3.16 Corrida manual del algoritmo de estimación de prioridad

Después se realizó la estimación de prioridad con la aplicación desarrollada, coincidiendo los resultados con la estimación manual, al estar seguros de los resultados finales realizamos una tabla comparativa de los valores de prioridad según el autor y los valores según el algoritmo como se muestra en la Ilustración 3.17

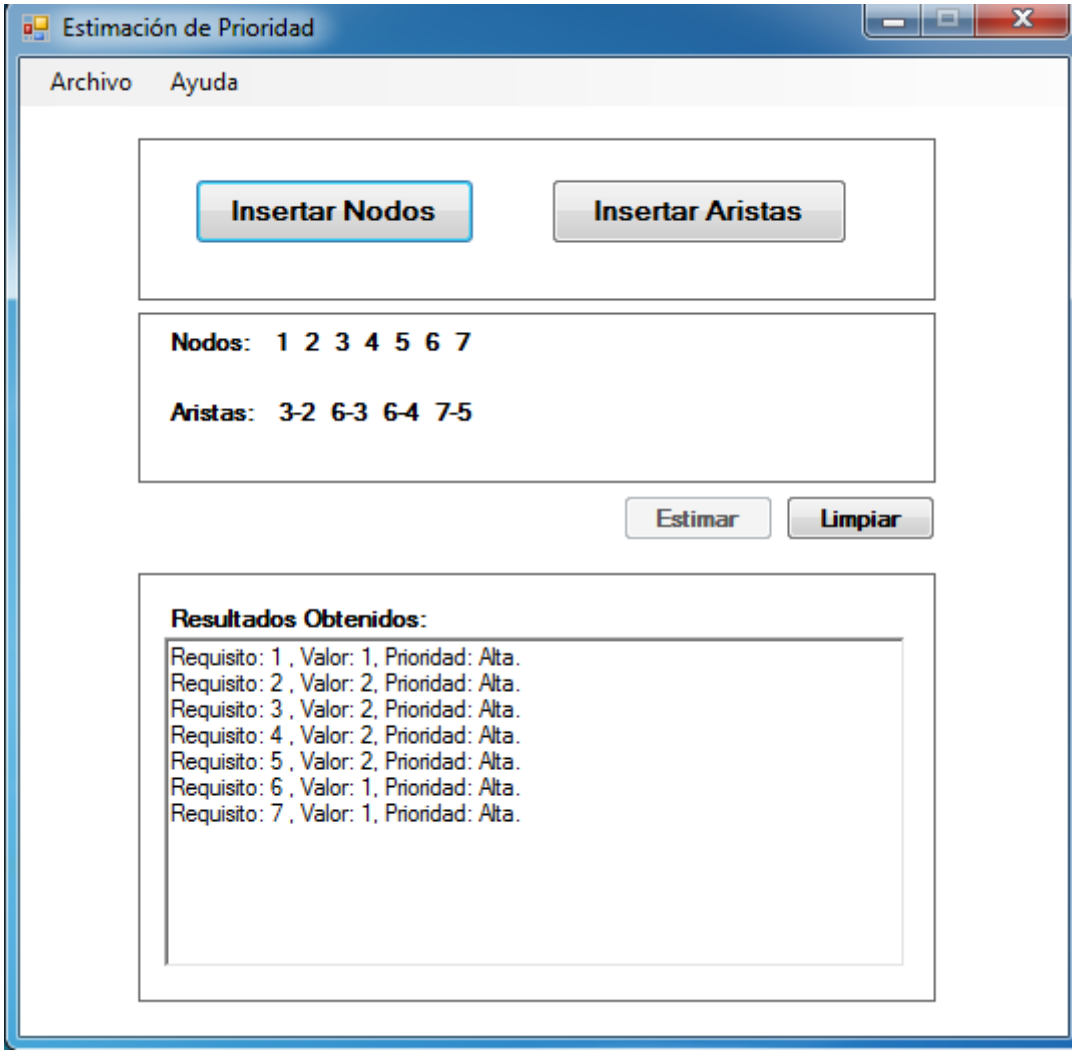

Ilustración 3.17 Corrida del algoritmo de estimación de prioridad utilizando la aplicación desarrollada

Requisitos funcionales	Prioridad según autor	Prioridad según el algoritmo
1	Media	Alta
2	Media	Alta

3	Alta	Alta
4	Media	Alta
5	Media	Alta
6	Media	Alta
7	Media	Alta

Tabla 3.18 Comparación de los Resultados obtenidos

Analizando la tabla 3.18 podemos concluir que se logró una estimación más certera en los requisitos 1, 2, 4, 5, 6, 7 esto puede mejorar los tiempos de desarrollo y costos del proyecto ya que podemos determinar cuáles son realmente los requisitos más importantes y enfocarnos en ello.

Teniendo la prioridad segura procedemos a calcular el riesgo, tomamos como punto de partida la prioridad resultante del algoritmo y estimamos los valores más correctos de probabilidad e impacto, después aplicamos manualmente la fórmula de **Riesgo = Impacto * Probabilidad**, como se muestra en la tabla 3.19

Requisito	**Prioridad**	**Probabilidad**	**Impacto**	**Riesgo**
1	Alta	0.9	0.8	0.72
2	Alta	0.9	0.8	0.72
3	Alta	0.9	0.8	0.72
4	Alta	0.9	0.8	0.72
5	Alta	0.9	0.8	0.72
6	Alta	0.9	0.8	0.72
7	Alta	0.9	0.8	0.72

Tabla 3.19 Estimación manual del algoritmo utilizando los valores obtenidos por la prioridad

Después de hallados los valores manualmente con el algoritmo procedimos a realizar la corrida de dichos valores en la aplicación desarrollada, coincidiendo los resultados del riesgo, como se muestra en la Ilustración 3.20

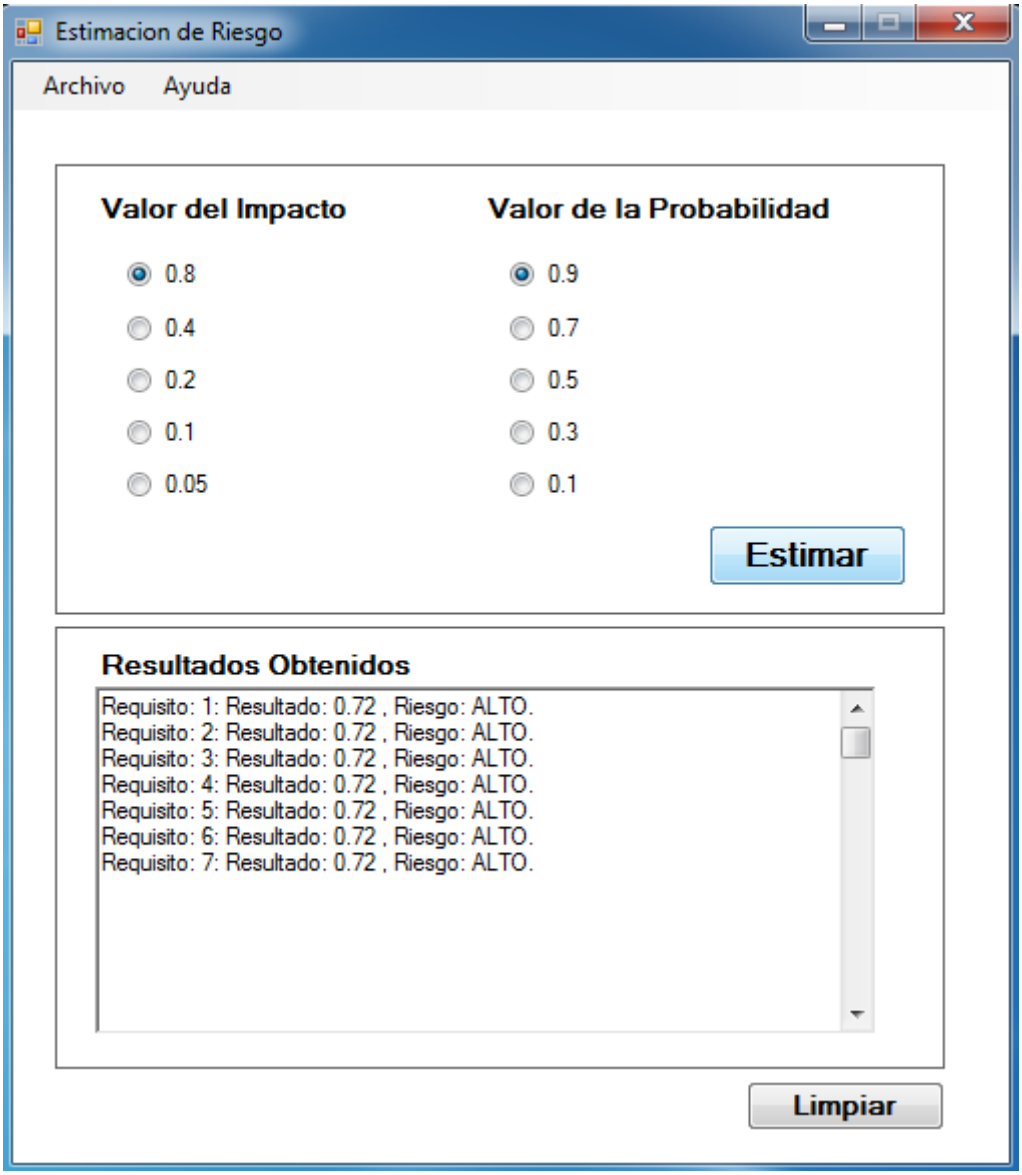

Ilustración 3.20 Resultados de la estimación de riesgo utilizando la aplicación desarrollada

Requisitos funcionales	Riesgo según autor	Riesgo según el algoritmo
1	Medio	Alto

2	Medio	Alto
3	Alta	Alto
4	Medio	Alto
5	Medio	Alto
6	Medio	Alto
7	Medio	Alto

Tabla 3.21 Comparación de los Resultados obtenidos

Analizando la tabla 3.7 podemos concluir que aumentaron los indicadores de riesgo en los requisitos 1, 2, 4, 5, 6, 7 que tenían una estimación del autor Media y el algoritmo determinó Alta, esto puede mejorar en el desarrollo del proyecto puesto que conocemos con seguridad que tareas tienen un impacto mayor sobre nuestra aplicación y debemos trabajar sobre estas para que no exista fallos graves en un futuro.

3.2.4 Prueba 4 a Tesis Titulada "Herramienta Informática de soporte tecnológico al Modelo de Gestión del Conocimiento del Centro de Información y Gestión Tecnológica de Matanzas" (Matanzas, Herramienta Informática de soporte tecnológico al Modelo de Gestión del Conocimiento del Centro de Información y Gestión Tecnológica de Matanzas, 2015)

Elaboramos la tabla de requisitos funcionales a partir de la documentación del sistema, se toma la prioridad y el riesgo del autor y se hallan las dependencias a partir de los mismos como se muestra en la tabla 3.22

Requisitos Funcionales	Nombre	Prioridad	Riesgo	Dependencias
1	Gestión de la herramienta	Alta	Medio	Ninguna
2	Gestión del Repositorio	Alta	Medio	1, 6
3	Gestión del	Alta	Medio	1

	FAQ			
4	Gestión del How-To	Alta	Medio	1
5	Gestión del Glosario	Alta	Medio	1, 6
6	Gestión de Búsquedas	Baja	Bajo	1
7	Visualización de reportes	Baja	Bajo	1, 2, 3, 4, 5, 6
8	Gestión de los términos	Baja	Bajo	2, 3, 4

Tabla 3.22 Requisitos funcionales según el autor del documento

Posteriormente aplicamos el algoritmo de estimación de prioridad manualmente según sus dependencias, todas las corridas se realizaron sin ningún error, arrojando los resultados finales como se muestra en la tabla 3.23

NODOS / ITERACIONES	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1ra	7	1	1	1	1	1	1	1
2da	7	3	1	1	1	1	1	1
3ra	7	3	3	1	1	1	1	1
4ta	7	3	3	3	1	1	1	1
5ta	7	3	3	3	2	1	1	1
6ta	7	3	3	3	2	7	1	1
7ma	7	3	3	3	2	7	1	1
8va	7	3	3	3	2	7	1	1

Tabla 3.23 Corrida manual del algoritmo de estimación de prioridad

Después se realizó la estimación de prioridad con la aplicación desarrollada, coincidiendo los resultados con la estimación manual, al estar seguros de los resultados finales realizamos una tabla comparativa de los valores de prioridad según el autor y los valores según el algoritmo como se muestra en la Ilustración 3.24

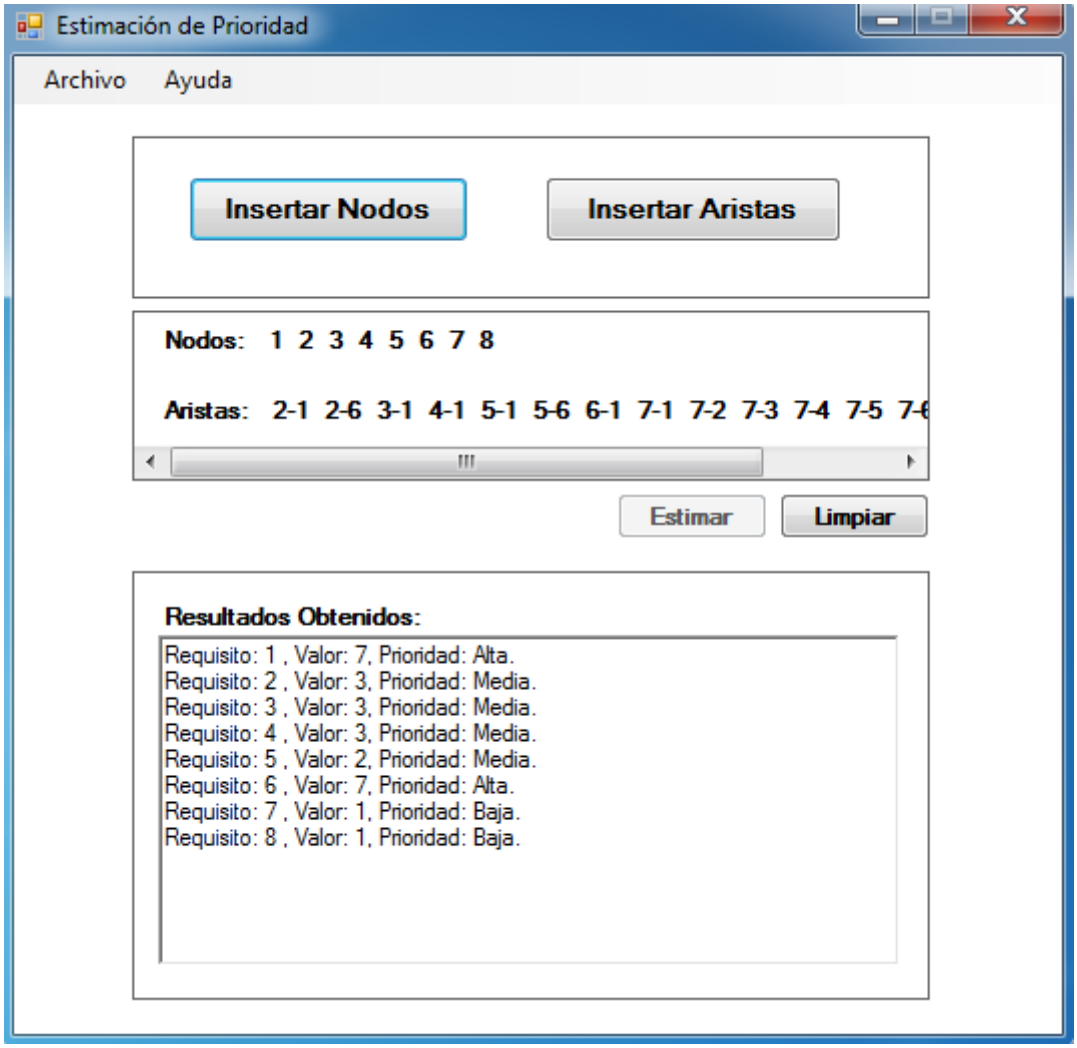

Ilustración 3.24 Corrida del algoritmo de estimación de prioridad utilizando la aplicación desarrollada

Requisitos funcionales	Prioridad según autor	Prioridad según el algoritmo
1	Alta	Alta

2	Alta	Media
3	Alta	Media
4	Alta	Media
5	Alta	Media
6	Baja	Alta
7	Baja	Baja
8	Baja	Baja

Tabla 3.25 Comparación de los Resultados obtenidos

Analizando la tabla 3.25 podemos concluir que se logró una estimación más certera en los requisitos 2, 3, 4, 5 esto puede mejorar los tiempos de desarrollo y costos del proyecto ya que podemos determinar cuáles son realmente los requisitos más importantes y enfocarnos en ello.

Teniendo la prioridad segura procedemos a calcular el riesgo, tomamos como punto de partida la prioridad resultante del algoritmo y estimamos los valores más correctos de probabilidad e impacto, después aplicamos manualmente la fórmula de **Riesgo = Impacto * Probabilidad**, como se muestra en la tabla 3.26

Requisito	**Prioridad**	**Probabilidad**	**Impacto**	**Riesgo**
1	Alta	0.9	0.8	0.72
2	Media	0.5	0.2	0.10
3	Media	0.5	0.2	0.10
4	Media	0.5	0.2	0.10
5	Media	0.5	0.2	0.10
6	Alta	0.9	0.8	0.72
7	Baja	0.3	0.1	0.03

8	Baja	0.3	0.1	0.03

Tabla 3.26 Estimación manual del algoritmo utilizando los valores obtenidos por la prioridad

Después de hallados los valores manualmente con el algoritmo procedimos a realizar la corrida de dichos valores en la aplicación desarrollada, coincidiendo los resultados del riesgo, como se muestra en la Ilustración 3.27

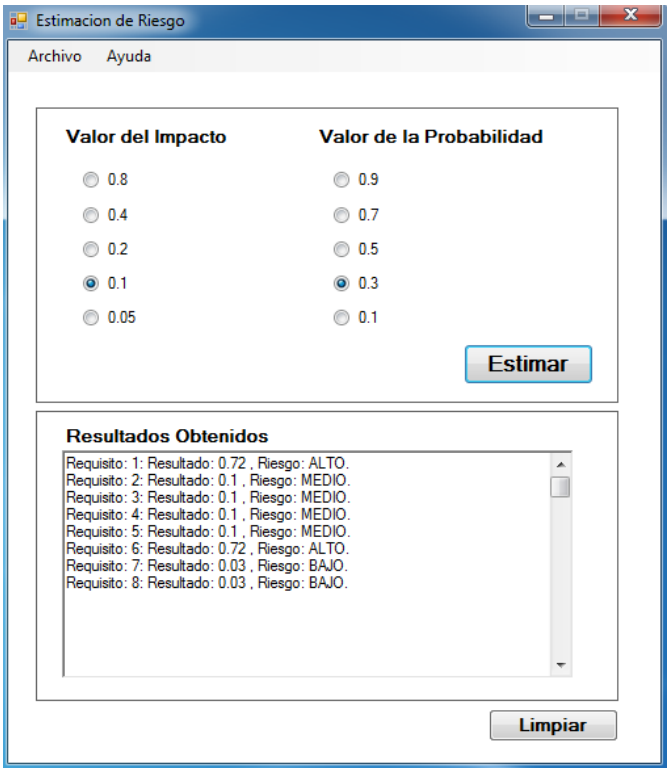

Ilustración 3.27 Resultados de la estimación de riesgo utilizando la aplicación desarrollada

Requisitos funcionales	Riesgo según autor	Riesgo según el algoritmo
1	Medio	Alto
2	Medio	Medio
3	Medio	Medio
4	Medio	Medio

5	Medio	Medio
6	Bajo	Alto
7	Bajo	Bajo
8	Bajo	Bajo

Tabla 3.28 Comparación de los Resultados obtenidos

Analizando la tabla 3.7 podemos concluir que se mantuvieron los indicadores de riesgo en los requisitos, esto quiere decir que existió una buena estimación por parte del equipo de trabajo.

3.2.5 Prueba 5 a Tesis Titulada "Sistema informático para la automatización del análisis de indicadores económicos por medio de redes neuronales". (Matanzas, Sistema informático para la automatización del análisis de indicadores económicos por medio de redes neuronales, 2015)

Elaboramos la tabla de requisitos funcionales a partir de la documentación del sistema, se toma la prioridad y el riesgo del autor y se hallan las dependencias a partir de los mismos como se muestra en la tabla 3.29

Requisitos Funcionales	Nombre	Prioridad	Riesgo	Dependencias
1	Diseño de la interfaz de usuario	Alta	Medio	Ninguna
2	Diseño y creación de la base de datos	Alta	Medio	Ninguna
3	Autentificar	Alta	Alto	Ninguna
4	Calcular indicadores mensuales	Alta	Medio	1
5	Calcular indicadores anuales	Alta	Medio	1, 4
6	Predecir indicadores	Alta	Alto	1, 4, 5

7	Generar reportes	Alta	Medio	1, 4, 5, 6
8	Diseño de la interfaz para la web	Alta	Medio	1, 2

Tabla 3.29 Requisitos funcionales según el autor del documento

Posteriormente aplicamos el algoritmo de estimación de prioridad manualmente según sus dependencias, todas las corridas se realizaron sin ningún error, arrojando los resultados finales como se muestra en la tabla 3.30

NODOS / ITERACIONES	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1ra	6	1	1	1	1	1	1	1
2da	6	2	1	1	1	1	1	1
3ra	6	2	1	1	1	1	1	1
4ta	6	2	1	4	1	1	1	1
5ta	6	2	1	4	3	1	1	1
6ta	6	2	1	4	3	2	1	1
7ma	6	2	1	4	3	2	1	1
8va	6	2	1	4	3	2	1	1

Tabla 3.30 Corrida manual del algoritmo de estimación de prioridad

Después se realizó la estimación de prioridad con la aplicación desarrollada, coincidiendo los resultados con la estimación manual, al estar seguros de los resultados finales realizamos una tabla comparativa de los valores de prioridad según el autor y los valores según el algoritmo como se muestra en la Ilustración 3.31

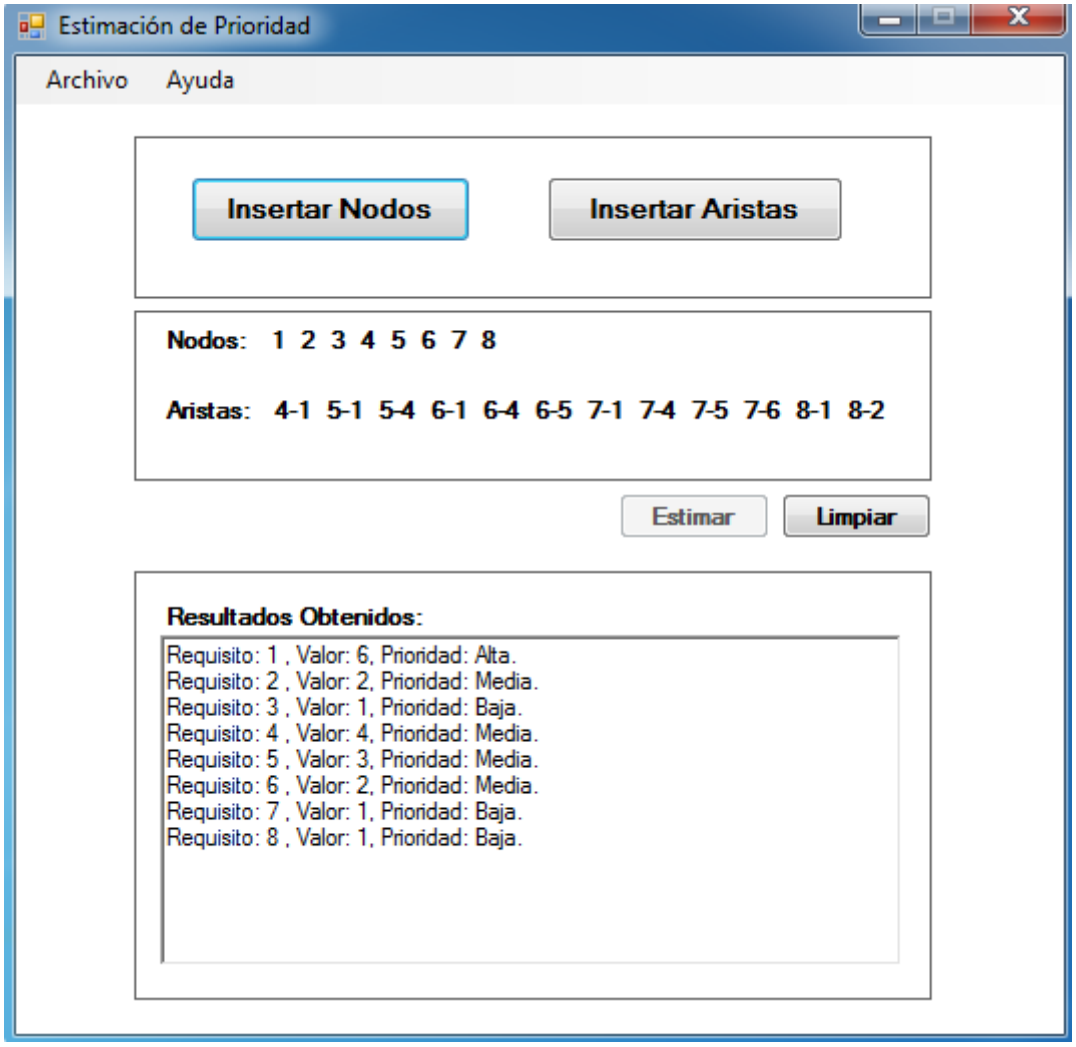

Ilustración 3.31 Corrida del algoritmo de estimación de prioridad utilizando la aplicación desarrollada

Requisitos funcionales	Prioridad según autor	Prioridad según el algoritmo
1	Alta	Alta
2	Alta	Media
3	Alta	Baja
4	Alta	Media
5	Alta	Media

6	Alta	Media
7	Alta	Baja
8	Alta	Baja

Tabla 3.32 Comparación de los Resultados obtenidos

Analizando la tabla 3.32 podemos concluir que se logró una estimación más certera en los requisitos 2, 3, 4, 5, 6, 7, 8 esto puede mejorar los tiempos de desarrollo y costos del proyecto ya que podemos determinar cuáles son realmente los requisitos más importantes y enfocarnos en ello.

Teniendo la prioridad segura procedemos a calcular el riesgo, tomamos como punto de partida la prioridad resultante del algoritmo y estimamos los valores más correctos de probabilidad e impacto, después aplicamos manualmente la fórmula de **Riesgo = Impacto * Probabilidad**, como se muestra en la tabla 3.33

Requisito	**Prioridad**	**Probabilidad**	**Impacto**	**Riesgo**
1	Alta	0.9	0.8	0.72
2	Media	0.5	0.2	0.10
3	Baja	0.3	0.1	0.03
4	Media	0.5	0.2	0.10
5	Media	0.5	0.2	0.10
6	Media	0.5	0.2	0.10
7	Baja	0.3	0.1	0.03
8	Baja	0.3	0.1	0.03

Tabla 3.33 Estimación manual del algoritmo utilizando los valores obtenidos por la prioridad

Después de hallados los valores manualmente con el algoritmo procedimos a realizar la corrida de dichos valores en la aplicación desarrollada, coincidiendo los resultados del riesgo, como se muestra en la Ilustración 3.34

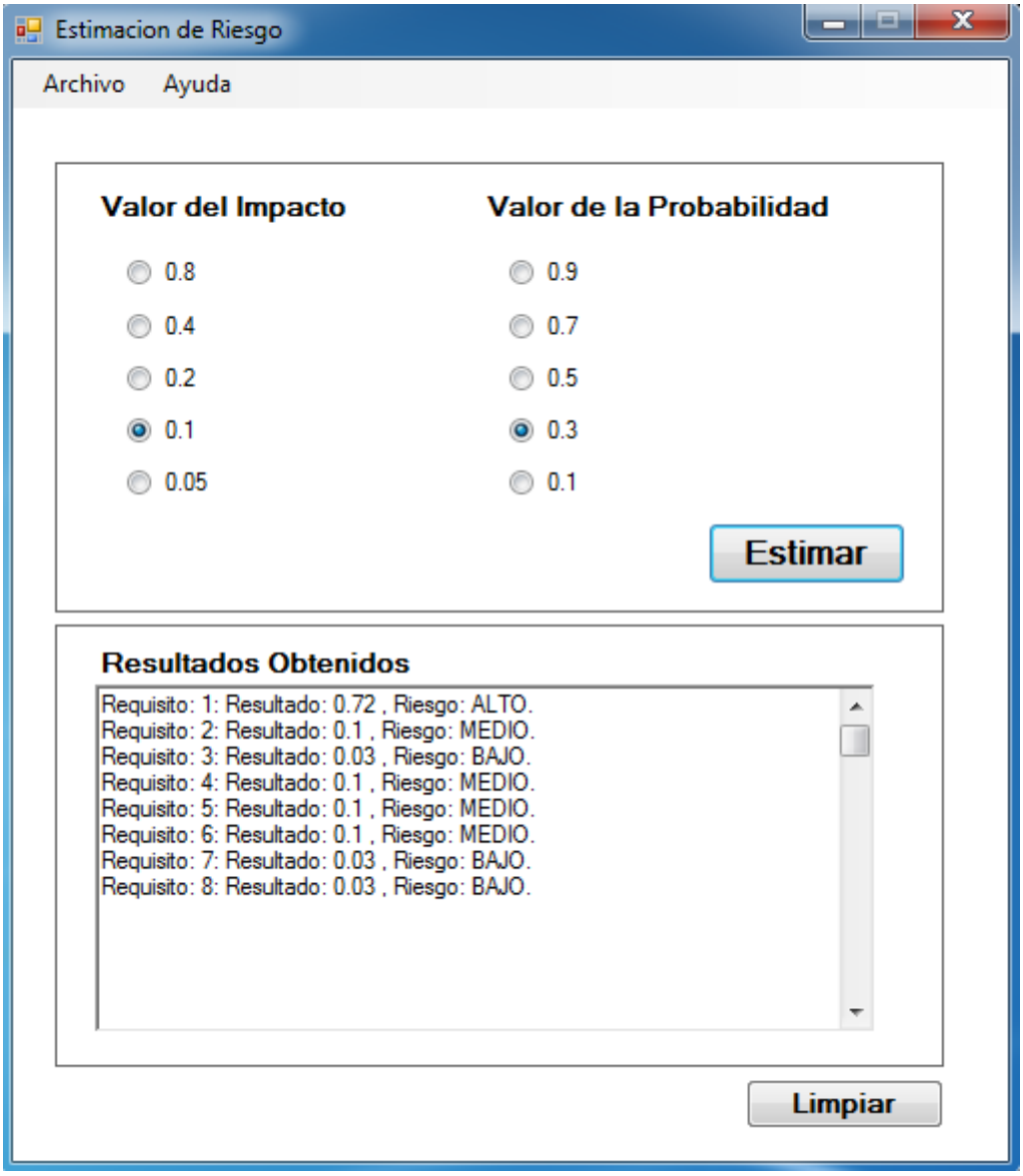

Ilustración 3.34 Resultados de la estimación de riesgo utilizando la aplicación desarrollada

Requisitos funcionales	Riesgo según autor	Riesgo según el algoritmo
1	Medio	Alto
2	Medio	Medio
3	Alto	Bajo

4	Medio	Medio
5	Medio	Medio
6	Alto	Medio
7	Medio	Bajo
8	Medio	Bajo

Tabla 3.35 Comparación de los Resultados obtenidos

Analizando la tabla 3.7 podemos concluir que mejoraron los indicadores de riesgo en los requisitos 3, 6, 7, 8 que tenían una estimación del autor media y el algoritmo determinó baja, esto puede mejorar en el desarrollo del proyecto puesto que conocemos con seguridad que tareas tienen un impacto mayor sobre nuestra aplicación y debemos trabajar sobre estas para que no exista fallos graves en un futuro.

3.3 Análisis de los resultados obtenidos

Después de corridas las pruebas sobre distintas tesis que utilizaron metodologías agiles de desarrollo se obtuvo como resultados que en más del 90% de las tesis analizadas se logró mejorar la estimación de prioridades y riesgo arrojando valores más precisos y acordes a los requisitos planteados. De las cinco tesis más importantes que se le aplicaron pruebas y plasmamos en este documento se elaboró la siguiente tabla basados en el promedio de las estimaciones del autor y las estimaciones resultantes del algoritmo:

Tesis	**Estimación de Prioridad según Autor**	**Estimación de Riesgo según Autor**	**Estimación de Prioridad según Algoritmo**	**Estimación de Riesgo según Algoritmo**
1	Alta / Media	Alto	Media	Medio
2	Alta / Media	Alto	Media / Baja	Medio / Bajo

3	Media	Medio	Alta	Alto
4	Alta	Medio	Media	Medio
5	Alta	Alto / Medio	Media	Medio

3.4. Conclusiones del capítulo

Luego de realizados los experimentos pertinentes para validar la propuesta, podemos arribar a las siguientes conclusiones:

1. Los resultados obtenidos demostraron que el algoritmo para la determinación de indicadores de prioridad y riesgo reporta estimados más confiables y homogéneos que las estimaciones a ciegas.
2. Se logró una mejora en la estimación de prioridad y riesgo en los requisitos funcionales utilizando un algoritmo basado en la teoría de grafos.
3. Se desarrolló una aplicación de escritorio basado en el algoritmo que estima las prioridades y los riesgos dados los requisitos.
4. Los resultados obtenidos con la aplicación del algoritmo evitan errores de subjetividad, toda vez que se derivan de la simulación y la estadística.
5. Se realizaron una serie de puebas para comprobar la validez del algoritmo desarrollado, arrojando resultados satisfactorios.

Conclusiones Generales

Como resultado de esta investigación quedaron satisfechos los objetivos trazados arribando a las siguientes conclusiones:

1. El estudio realizado sobre los antecedentes, el estado actual de la temática, la bibliografía y documentos relacionados con el objeto de estudio, permitió contar con los elementos necesarios para dar solución a la problemática planteada.
2. Los trabajos encontrados, vinculados al tema, no brindan la mejor solución a los problemas planteados que constituyen la base de esta investigación.
3. Se elaboró un procedimiento para estimar indicadores de prioridad y riego basado en teoría de grafos, logrando valores significativos para estos índices que se derivan de la simulación y la estadística.
4. Se logró una mejora en la estimación de prioridad y riesgo en los requisitos funcionales utilizando un algoritmo basado en la teoría de grafos.
5. Se desarrolló una aplicación de escritorio basado en el algoritmo que estima las prioridades y los riesgos dados los requisitos.
6. Los resultados obtenidos demostraron que el algoritmo para la determinación de indicadores de prioridad y riesgo reporta estimados más confiables y homogéneos que las estimaciones a ciegas.
7. Se realizaron una serie de pruebas para comprobar la validez del algoritmo desarrollado, arrojando resultados satisfactorios.

Recomendaciones

En el transcurso de esta investigación han surgido cuestiones que pueden resultar de interés, a continuación, exponemos algunas ideas y recomendaciones para trabajos futuros:

1. Continuar profundizando en el tema de investigación en aras de optimizar los resultados.
2. Se recomienda utilizar este algoritmo para ayudar a futuras investigaciones en nuestra universidad.
3. Se recomienda socializar el empleo de este algoritmo en nuestra universidad.

Referencias Bibliográficas

Almunia, P. (2016). *Dirección de Proyectos Informaticos.*

Barros, A. (2014). *Comportamiento de proyectos.*

Beck, K. (2005). *Extreme Programming Explained: Embrace Change.*

Booch, R. (1999). *El Lenguaje Unificado de Modelado.*

Center, V. C. (2016). *Frequently Asked Questions About Visual C# .NET 2016.*

Cooke, L. (2012). *Everything you want to know about Agile.*

Coram, L. (1996). *A Pattern Language for User Interface Design.*

Doe, J. (2017). *Xtreme Programming Step by Step.*

Fernandez, J. (2008). *Análisis y Diseño detallado de Aplicaciones Informáticas de Gestión.*

Fuentes, J. L. (2016). *Desarrollo de Softwre Ágil. Extremme Programming y Scrum.*

Gamboa, L. (2011). *Diseñando un modelo computacional para apoyar el uso de un proceso de gestión de riesgos en proyectos.*

Gómez, M. O. (2017). *Gestión de los Riesgos del Proyecto.*

Helm, R. (2016). *Design Patterns. Elements of Reusable Object-Oriented Software.*

Jimenez, J. (2014). *Riesgo e incertidumbre en la gestión de proyectos informáticos.*

Matanzas, U. d. (2013). *Aplicación web para la gestión de la información de la reserva en Matanzas .*

Matanzas, U. d. (2014). *Aplicación Web para la gestión de la información en el procedimiento P 02-4 Comunicación con el Cliente en la UEB Divep Matanzas.*

Matanzas, U. d. (2015). *Herramienta Informática de soporte tecnológico al Modelo de Gestión del Conocimiento del Centro de Información y Gestión Tecnológica de Matanzas.*

Matanzas, U. d. (2015). *Sistema informático para la automatización del análisis de indicadores económicos por medio de redes neuronales.*

Matanzas, U. d. (2015). *Software de gestión y costeo por actividades (ABC/ABM).*

Mcconnell, S. (2012). *Desarrollo y gestión de proyectos informáticos.*

Moguel. (2005). *Metodología de la Investigación.*

Morales, E. M. (2010). *Sistema para el análisis cuantitativo de los riesgos para los proyectos de producción de software.*

Pressman, R. (2002). *Ingeniería del software un enfoque práctico.*

Pressman, R. (2002). *Introduction to Algorithms.*

Prieto. (2012). *Estratagias de enseñanza-aprendizaje.*

Riley, J. (2012). *Sistemas Expertos. Principios y Programación.*

Rodriguez, R. R. (2015). *Metodologías Ágiles para el desarrollo de Software.*

Saez, J. (2015). *El método científico, fundamentos metodológicos.*

SCHWALBE, K. (2013). *Information Technology Project Management.*

Sommerville, I. (2010). *Ingeniería de Software.*

TANGIENT. (2018). *http://programacion-extrema.wikispaces.com. 2013*.

Wikipedia. (2018). *https://es.wikipedia.org/wiki/C_Sharp*.

Anexos:

Anexo 1: Algoritmo y Código para la aplicación de estimación de prioridad.

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Collections.Generic;


namespace Estimacion
{
   public partial class FormPrioridad : Form
   {
      // defino mis globales
      List <int> nodos;
      List <string> aristas;
      public FormPrioridad()
      {
         InitializeComponent();
         nodos = new List<int>();
         aristas = new List<string>();
      }
      // propiedad para acceder a los nodos desde otro formulario.
      public List <int> getNodos(){
         return this.nodos;
      }
      // mi clase de nodos
      public class Nodos
      {
         public string nombre;
         public int valor;
         public string dependencias;
         public Nodos(string nombre)
         {
            this.nombre = nombre;
```

```
        dependencias = ",";
        valor = 1;
    }
}
// clase que construye el grafo.
public class Grafo
{
    //creo un arreglo de nodos que sera mi grafo.
    public Nodos[] grafo;
    public void Construir(List <int> nod, List <string> aristas)
    {
        grafo = new Nodos[nod.Count];
        for (int x = 0; x < nod.Count; x++) {
            grafo[x] = new Nodos(nod[x].ToString());
            // ciclo para incluir los nodos dependientes.
            for (int j = 0; j < aristas.Count; j++) {
                if (aristas[j][0].ToString() == grafo[x].nombre) {
                    grafo[x].dependencias += aristas[j][2] + ",";
                }
            }
        }
    }
    // metodo bfs para moverme por los nodos dependientes.
    public void BFS()
    {
        // ciclo pro todos los nodos de mi clase grafo.
        for (int x = 0; x < grafo.Length; x++) {
            string nodoactual = grafo[x].nombre;
            for (int j = 0; j < grafo.Length; j++) {
                // si me encuentro un nodo dependiente acualizo su valor con este.
                if (grafo[j].dependencias.Contains("," + nodoactual + ",")) {
                    grafo[x].valor += grafo[j].valor;
                }
            }
        }
```

```
    }
}
public void Button1Click(object sender, EventArgs e)
{
    // creo el grafo para llenarlo.
    Grafo g = new Grafo();
    // lo contruyo con mis nodos y aristas que inserte desde otros formularios.
    g.Construir(nodos, aristas);
    // lo recorro.
    g.BFS();
    // tamaño del arreglo de grafos o cantidad e nodos existentes.
    int longitud = g.grafo.Length;
    // arreglo para gaurdar los resultados.
    string[] result = new string[longitud];
    //
    int valorminimo = 0;
    int valormedio = 0;
    int mayor=g.grafo[0].valor;
    // ciclo para determinar el nodo de mas peso y usar su valor para dividir en 3.
    for (int x = 1; x < longitud; x++) {
        if(g.grafo[x].valor>mayor){
            mayor=g.grafo[x].valor;
        }
    }
    // divido en 3 mi mayor numero
    valorminimo = mayor / 3;
    // determino la mitad de 3
    valormedio= valorminimo*2;
    // recorrido por mis nodos y ver a que intervalo pertecenen.
    for (int j = 0; j < longitud; j++) {
        string prioridad = "";
        if (g.grafo[j].valor > valormedio) {
            prioridad = "Alta.";
        } else if (g.grafo[j].valor >= valorminimo && g.grafo[j].valor <= valormedio) {
            prioridad = "Media.";
```

```
            } else {
                prioridad = "Baja.";
            }
            result[j] = "Requisito: " + g.grafo[j].nombre + " , Valor: " + g.grafo[j].valor + ",
Prioridad: " + prioridad;
        }
        // guardo los valores en el cuadro de texto del formulario.
        richTextBox1.Lines = result;
        button1.Enabled = false;
    }
    // estos son los metodos para acceder desde otros formularios y actualizar los valores de
mis nodos y aristas.
    public void ActualizarNodos(List<int> nodos)
    {
        //actualizo mi valor por el que me pasan
        this.nodos = nodos;
        label1.Text = "Nodos: ";
        for (int i = 0; i < nodos.Count; i++) {
            label1.Text += "  " + nodos[i].ToString();
        }
        //actualizo mi formulario apra que mustre lso cambios.
        this.Refresh();
        button3.Enabled = true;
    }
    public void ActualizarAristas(List<string> aristas)
    {
        //actualizo mi valor por el que me pasan
        this.aristas = aristas;
        label2.Text = "Aristas: ";
        for (int i = 0; i < aristas.Count; i++) {
            label2.Text += "  " + aristas[i].ToString();
        }
        //actualizo mi formulario apra que muestre lso cambios.
        this.Refresh();
        button1.Enabled = true;
```

```
        }
        // metodo para limpiar todo
        void Button4Click(object sender, EventArgs e)
        {
            nodos = new List<int>();
            aristas = new List<string>();
            label1.Text = "Nodos:";
            label2.Text = "Aristas:";
            button3.Enabled = false;
            richTextBox1.Lines = new string[10];
        }
        // metodo para limpiar todo
        void NuevoToolStripMenuItemClick(object sender, EventArgs e)
        {
            nodos = new List<int>();
            aristas = new List<string>();
            label1.Text = "Nodos:";
            label2.Text = "Aristas:";
            button3.Enabled = false;
            richTextBox1.Lines = new string[10];
        }
        void AyudaToolStripMenuItem1Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start("Ayuda.pdf");
        }
        void AcercaDeToolStripMenuItemClick(object sender, EventArgs e)
        {
            MessageBox.Show("(C) 2018, Universidad de Matanzas, Todos los derechos
reservados.");
        }
        void Button3Click(object sender, EventArgs e)
        {
            FormAristas aristasform = new FormAristas(this);
            aristasform.Show();
        }
```

```
        // llamar al formulario de insertar nodos.
        void Button2Click(object sender, EventArgs e)
        {
            FormNodos nodosform = new FormNodos(this);
            nodosform.Show();

        }
    }
}
```

Anexo 2: Algoritmo y Código para la aplicación de estimación de riesgo.

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using System.IO;

namespace Estimacion
{
   public partial class FormRiesgo : Form
   {
      //defino mis variables globales
      public string[] matriz;
      public string[] resultados;
      public int j;
      public FormRiesgo()
      {
         InitializeComponent();
         // leo la matriz del archivo.
         matriz = File.ReadAllLines("config/matrizriesgo.txt");
         resultados = new string[100];
         j = 0;
      }
      void Button1Click(object sender, EventArgs e)
      {
         double impacto = 0;
         double prob = 0;
         // Valores de los radiobutton para el impacto.
         if (radioButton1.Checked) {
            impacto = double.Parse(radioButton1.Text);
         }
         if (radioButton2.Checked) {
            impacto = double.Parse(radioButton2.Text);
         }
         if (radioButton3.Checked) {
```

```
            impacto = double.Parse(radioButton3.Text);
        }
        if (radioButton4.Checked) {
            impacto = double.Parse(radioButton4.Text);
        }
        if (radioButton5.Checked) {
            impacto = double.Parse(radioButton5.Text);
        }
        // Valores de los radiobutton para la probabilidad.
        if (radioButton6.Checked) {
            prob = double.Parse(radioButton6.Text);
        }
        if (radioButton7.Checked) {
            prob = double.Parse(radioButton7.Text);
        }
        if (radioButton8.Checked) {
            prob = double.Parse(radioButton8.Text);
        }
        if (radioButton9.Checked) {
            prob = double.Parse(radioButton9.Text);
        }
        if (radioButton10.Checked) {
            prob = double.Parse(radioButton10.Text);
        }
        //formula del riesgo
        double riesgo = impacto * prob;
        // redondear a dos lugares despues de la coma
        riesgo = Math.Round(riesgo,2);
        string resultadofinal="";
        // recorro la matriz para encontrar mi resultado y ver que riesgo tiene.
        for (int i = 0; i < matriz.Length; i++) {
            string[] temp = matriz[i].Split(' ');
            if (temp[0] == riesgo.ToString()) {
                resultadofinal = "Requisito: " + (j + 1).ToString() + ": Resultado: " + riesgo.ToString()
+ " , Riesgo: " + temp[1] + ".";
```

```
                resultados[j] = resultadofinal;
                j++;
                richTextBox1.Lines = resultados;
                break;
            }
        }
    }
    // salir del formulario
    void SalirToolStripMenuItemClick(object sender, EventArgs e)
    {
        this.Close();
    }
    // llamar al pdf de la ayuda
    void AyudaToolStripMenuItem1Click(object sender, EventArgs e)
    {
        System.Diagnostics.Process.Start("Ayuda.pdf");
    }
    // acerca de
    void AcercaDeToolStripMenuItemClick(object sender, EventArgs e)
    {
        MessageBox.Show("(C) 2018, Universidad de Matanzas, Todos los derechos
reservados.");
    }
    // Metodos para Limpiar
    void NuevoToolStripMenuItemClick(object sender, EventArgs e)
    {
        resultados = new string[100];
        richTextBox1.Lines = new string[10];
        j = 0;
    }
    void Button2Click(object sender, EventArgs e)
    {
        resultados = new string[100];
        richTextBox1.Lines = new string[10];
        j = 0;
```

```
        }
    }
}
```