ESCUELA NAVAL DE CADETES "ALMIRANTE PADILLA"
Honor y Tradición

OnBoard Knowledge Journal

Article

Intelligent Management of Alternative Energy Sources through a System based on Fuzzy Logic

Mejoramiento del proceso de planificación en el desarrollo de software mediante el método de puntos de caso de uso

Jose David Polo-Vanegas [1], Yeinis Paola Espitia-Priolo [1] and Daniel Jose Salas-Álvarez [1]*

[1] Department of Systems Engineering, Faculty of Engineering, University of Córdoba, Monteria, 230002, Colombia; jpolovanegas90@correo.unicordoba.edu.co; yespitiapriolo@correo.unicordoba.edu.co; danielsalas@correo.unicordoba.edu.co
* Correspondence: danielsalas@correo.unicordoba.edu.co

Abstract: Planning is crucial for the success of software projects; however, it often faces problems such as lack of time, experience, and understanding of requirements. These challenges can lead to inaccurate estimates, delays, and products that do not meet customer expectations. This study aims to analyze the implementation of the Use Case Points (UCP) method to support planning in software development and make it more efficient. The research was carried out in several phases: first, common problems in software planning were analyzed; then, based on these, the UCP method was chosen and implemented in a solution supported by tools like Microsoft Excel. Finally, three case studies were conducted to evaluate the effectiveness of the UCP method and compare it with agile methodology, which allowed improvements in planning processes to be observed, in terms of time, in software project development.

Keywords: Agile Methodology; Estimation; Software development planning; Software projects; Use Case Point Method (UCPM)

Resumen: La planificación es crucial para el éxito de proyectos software; sin embargo, suele enfrentar problemas como la falta de tiempo, experiencia y comprensión de los requisitos. Estos desafíos pueden provocar estimaciones inexactas, retrasos y productos que no cumplen con las expectativas del cliente. Este estudio tiene como propósito analizar la implementación del método de puntos de casos de uso (MPCU) para apoyar la planificación en el desarrollo de software y hacerla más eficiente. La investigación se desarrolló en varias fases: primero, se analizaron los problemas comunes en la planificación de software; luego, con base en estos, se eligió el MPCU implementándolo en una solución apoyada en herramientas como Microsoft Excel, finalmente, se implementaron tres casos de estudio para evaluar la efectividad del MPCU, y compararlo con la metodología ágil, lo que permitió observar mejoras en los procesos de planificación en términos de tiempo, en el desarrollo de proyectos software.

Palabras clave: Estimación; Método de puntos de casos de uso (MPCU); Metodología Ágil; Planificación del desarrollo de software; Proyectos de software

check for updates

Citation: Polo, J.; Espitia, Y.; Salas, D. Intelligent Management of Alternative Energy Sources through a System based on Fuzzy Logic. *OnBoard Knowledge Journal* **2025**, *1*, 0. https://doi.org/

Received: 17/02/2025
Accepted: 29/04/2025
Published: 21/05/2025

1. Introduction

The software development process is essential for technological advancement and is closely linked to software engineering, which encompasses the structures, tools, and methods employed in the creation of computer programs. This process includes situation analysis, project drafting, software development, and the necessary testing to ensure proper functionality before implementation. In this context, planning plays a crucial role, as the success of the developed software depends on it. According to Alboka Soft, the benefits of good planning include: realistic monitoring of each phase, reduction of unnecessary work and costs, evaluation of the impact of changes, obtaining software development and testing, stabilization of requirements, and absolute project traceability. Additionally, proper planning grants independence to developers and ensures that the final objective adds maximum value to the project by organizing and documenting each step [1].

Being competitive is a characteristic that companies require today, and therefore they must exercise greater control, making this goal increasingly difficult to achieve. Clients demand precise estimates and budgets related to software development projects in very short timeframes, indicating that incorrect responses and estimates lead to project failure and financial losses for companies. Therefore, it is important to clarify that preliminary project planning and clarification of planned objectives is a necessary task for commercial success and the reasonable achievement of established goals. Conversely, neglecting this increases the potential for error and raises the level of failure and conflict [12].

For a project to achieve its goal, it must consider all possible scenarios by consulting client needs. Hence, failing to cover all these possibilities can lead to project failure. Thus, by improving the project planning capacity, an optimal result would be obtained, reflected not only in the company's profits but also in customer satisfaction. For this reason, it is necessary for companies to have a good pre-production plan for projects.

A question included in the survey conducted by Coding Sans: "What is the number one cause of delays in your teams?", was asked to over 500 software developers worldwide as part of their annual report "The State of Software Development 2021." According to the report, the main cause of delivery problems in software development teams was the lack of effective planning, identified by most respondents as the primary issue. This indicates that planning is a crucial element for the successful delivery of software projects, and teams must ensure proper planning from the beginning of the project.

In particular, the report highlighted the main factors contributing to poor planning as: lack of time, lack of experience, and lack of understanding of project requirements. The lack of time for planning is a common problem in many software projects, especially when teams face tight deadlines. Teams often feel pressured to start development quickly, which may lead them to skip the planning phase or perform rushed and superficial planning. This can cause a lack of understanding of project requirements and increase errors and delivery delays.

Lack of experience in planning is also a challenge for some teams. Those without experience may not know how to properly approach planning and struggle to identify key requirements and tasks necessary to achieve project goals. Finally, a lack of understanding of project requirements is another key factor contributing to poor planning. If teams do not fully comprehend the project requirements or lack a clear vision of the scope, they may plan inadequately and underestimate the time and resources needed [2].

Therefore, it is crucial for software development teams to dedicate the necessary time and effort to pre-production to ensure project success. Proper planning helps identify and address potential problems and risks before they become serious, which can save time and resources in the future.

This study's main objective is to analyze the use of use case point methods in development and the agile approach to verify their efficiency and understand the strengths and weaknesses of both approaches.

The structure of this article is organized as follows: Section 1 introduces 1 the importance of effective planning in software development and identifies common challenges.

Section 2 details the main contributions of this study, highlighting the implementation of the Use Case Point method, its integration with agile methodologies, and the validation through case studies. Section 3 reviews related work and existing methodologies for project estimation and management. Section 4 describes the methodology applied. Section 5 presents the results and comparative analysis from the conducted case studies. Finally, Section 6 summarizes the conclusions and discusses future directions.

2. Contributions

This section summarizes the key contributions of the study aimed at enhancing the software development planning process. By addressing common challenges in estimation and project management, the research proposes an integrated methodology combining the Use Case Point Method with agile practices. The following points highlight the main advancements and practical outcomes achieved through this work.

1. This study identifies and explores the primary challenges affecting software development planning, such as lack of time, insufficient experience, and limited understanding of requirements—factors that often lead to inaccurate estimates and project delays.
2. A solution based on the UCP methodology supported by tools like Microsoft Excel was developed to facilitate effort estimation in man-hours for software projects. This implementation incorporates technical and environmental factors to enhance planning accuracy.
3. A comparative analysis between UCP and the Agile Scrum methodology was conducted, highlighting the strengths and weaknesses of each approach. Furthermore, a hybrid approach is proposed to leverage UCP's initial estimation accuracy alongside Agile's flexibility during project execution.
4. The proposed solution was evaluated using three case studies (one fictitious and two real), demonstrating improvements in effort estimation and time management in software development projects, thereby evidencing the effectiveness and applicability of the hybrid approach in different contexts.

3. Related Works

The planning process in software development is a relevant activity because good project planning is crucial for its success, contributing to effective team management and significantly improving product quality [5]. To improve the software planning process, it is first necessary to understand what is being done and how it is being done, since many software projects fail due to poor requirements management and unrealistic deadlines [4]. Another cause is the lack of rigorous and detailed estimation, which allows clarity regarding the activities to be performed both in pre- and post-production, as well as their possible changes, leaving a sufficiently clear margin for potential errors. This often results in uncertainty during development about what to do in certain situations or how to handle changing requirements [11].

Based on the above, a question arises: what methods do developers use to mitigate these problems? A common technique used by developers for software estimation is the use of work breakdown structures (WBS), which involve dividing the project into smaller, manageable tasks, facilitating the estimation of the time and resources needed to complete each task. Another widely used technique is function points, which measure the functional size of the software that is, how many functions the software performs and how complex they are. From this measurement, the effort required to develop the software can be estimated. Following this path, there is no single method that guarantees success in all cases, but many researchers agree that the agile model is the best for software development projects due to its flexibility in responding to changes and new requirements [10].

The Agile method is so popular that Ortiz Álvarez presented a tool for managing activities in software projects using an agile methodology based on Scrum, which involves weekly follow-up meetings and delivery cycles of activities. The tool allows the creation of a client space where tasks can be recorded and edited to ensure direct communication with the development team [16].

However, the agile approach has several disadvantages, including the lack of a detailed plan, which can make it difficult to estimate the time and resources needed to complete the project. This is a risk when considering the function point technique, where if not applied correctly, resulting estimates may be inaccurate or incomplete. For example, if all the software functions are not identified or their complexity is underestimated, the estimate may be too low, leading to problems such as delivery delays or cost overruns. Therefore, it is important to correctly apply this technique and ensure that all relevant functions are properly identified and evaluated [11].

In a review of software project case studies, Ibraigheeth Mohammad and Fadzli Syed Abdullah identified common factors contributing to software project success: successful software projects have realistic and stable objectives, a team with adequate knowledge and experience, efficient technology, user involvement, and efficient management. Additionally, they note that project failures can be useful for identifying key factors for project success. They assert that no single factor guarantees project success; rather, it is a combination of several factors that contribute to success. Understanding these key success factors can help project managers make informed decisions about resource allocation and project management [6].

Project success is measured by the ability to complete it according to desired specifications, within the specified budget and promised schedule, while keeping the client and stakeholders satisfied. For correct project completion, both planning and execution must be properly implemented.

Furthermore, software defect prediction is one of the most active research areas in software engineering and plays an important role in software quality assurance. The increasing complexity and reliance on software have raised the difficulty of delivering high-quality, low-cost, and maintainable software, as well as the likelihood of creating software defects. Software defects often cause incorrect or unexpected results and behaviors in undesirable ways. Defect prediction is a crucial and essential activity. Using defect predictors can reduce costs and improve software quality by identifying modules (instances) prone to defects before testing, enabling software engineers to effectively optimize the allocation of limited resources for testing and maintenance [7].

In addition to recent advances in software defect prediction, ensemble learning approaches have been explored. These approaches combine multiple classification techniques to improve prediction performance. This research provides a systematic review of the use of ensemble learning for software defect prediction, identifying the most employed methods and their performance metrics. Results show that ensemble approaches, such as random forests and boosting, offer better classification accuracy compared to individual classifiers. Furthermore, the importance of feature selection and data sampling as preprocessing steps to improve ensemble classifiers is highlighted [8].

4. Methodology

This study focused on conducting a comprehensive and detailed analysis of case studies and reports collected from software development companies that have employed the Use Case Point (UCP) method. The UCP method, which serves as a systematic approach to estimating the effort and resources required in software projects, was critically examined to gain a deeper understanding of its practical application. Particular attention was given to identifying common challenges, potential sources of error, and limitations that may arise during its implementation, such as estimator subjectivity and variability in assessing use case complexity and environmental factors.

In parallel, the study performed a comparative evaluation between the UCP method and the agile approach, a widely adopted and flexible methodology known for its iterative planning and adaptability to change throughout the software development lifecycle. This comparative analysis explored the strengths and weaknesses of both methods in the context of project planning accuracy, flexibility, effort estimation, and risk management. By analyzing these aspects, the research sought to provide a more nuanced and comprehensive perspective on how software development planning processes could be optimized by leveraging the complementary benefits of both approaches.

Based on the insights gained from this analysis, a novel solution was proposed and developed that integrates the Use Case Point method within an agile framework, aiming to address the identified problems in traditional software planning practices. This hybrid approach was designed to offer a solid initial estimation grounded in UCP's structured assessment while maintaining the iterative flexibility and responsiveness characteristic of agile methodologies.

To evaluate the practicality and effectiveness of this integrated solution, three case studies were conducted: one fictitious project designed to test the approach in a controlled scenario, and two real-world projects undertaken by students in a systems engineering program. These case studies provided empirical data to assess the solution's applicability across different contexts, allowing for comparison of estimated effort, time management, and overall planning improvements. The outcomes from these studies served as a foundation for validating the proposed methodology and identifying areas for further refinement.

5. Results

Comparative analysis of MPCU and Agile method

The Use Case Points estimation method proposed by Karner is used as a basis to calculate the effort required for software implementation. This method allows an early estimation based on a certain knowledge of the requirements to be developed, which is of interest to companies engaged in software construction. The effort estimation, measured in man-hours (MH), required for the development of a specific software product is performed based on the number and complexity of use cases identified in the project. However, the effort estimation for the implementation process of information systems using this method shows a significant deviation from the actual estimated effort. One influencing factor in this deviation may be the lack of experience and subjectivity of the estimator when assigning values to technical and environmental factors, as well as when classifying the complexity levels of use cases and actors [15].

The agile methodology Scrum, one of the most widely used, is characterized by its flexibility and adaptability to changes and new requirements that may arise during the development process. Instead of following a rigid plan, the agile model focuses on the continuous delivery of small parts of the project, called iterations or sprints, which improve over time. However, agile has several disadvantages, such as the difficulty of estimating the time and resources necessary to complete the project due to the lack of a detailed plan (Table 1).

Table 1. Comparative table of the MPCU and the Agile Method (Scrum).

Aspect	Use Case Points (UCP)	Agile Methodology
Estimation Basis	Based on the number and complexity of identified use cases.	Estimated based on iterations (sprints) and smaller scope tasks.
Initial Accuracy	High accuracy in initial estimation if requirements are well-defined.	Lower initial accuracy due to the iterative and flexible nature of the process.
Effort Calculation	Effort is calculated in man-hours, considering technical and functional complexity.	Effort is continuously adjusted each sprint, without a detailed global estimate at the start.
Adaptation to Changes	Requires major revisions and adjustments if requirements change.	Changes are easily incorporated in the next sprint without greatly affecting the overall estimate.
Estimation Deviation	Higher risk of deviation if use cases are not correctly identified or complexity is underestimated.	Lower risk of deviation, as estimation is continuously adjusted based on actual project progress.
Prerequisites	Requires clear and detailed knowledge of requirements from the beginning.	Does not require complete knowledge of requirements, as they may change during the project.
Flexibility in Estimation	Low flexibility, since estimation is done at the start and depends on requirement stability.	High flexibility, as estimation adapts in each sprint according to new needs or changes.
Impact of Complexity	Use case complexity directly affects the estimate, potentially causing deviations if not measured properly.	Complexity is managed iteratively, adjusting the estimate each sprint based on the team's capabilities.

Implementation of the MPCU using the tool

Based on the research, a solution was developed to apply the Use Case Point Method (UCP) within the Scrum-based methodology: we took the UCP implementation from an Excel spreadsheet presented by Scott Sehlhorst in a Tyner Blain article [14], translated it into Spanish, and applied recommendation tables to facilitate its use along with a simple spreadsheet to organize scrums, enabling its combination with the agile methodology (Figure 1).

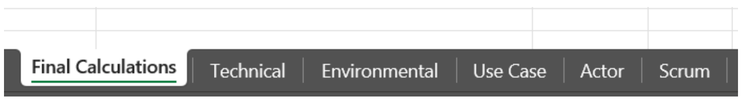


Figure 1. Spreadsheet tabs. Source: "Final Calculations" Excel spreadsheet.

The spreadsheet originally contained five tabs for processing and collecting the data necessary to estimate effort using the Use Case Point Method (UCP). Later, we added a tab called "Scrum" to facilitate comparison with the agile methodology.

To calculate Use Case Points (UCP), the first step is to determine a numerical representation of the technical factors of the software, known as the Technical Complexity Factor (TCF), which covers non-functional aspects of the system such as performance, security, and the use of reusable components. The second step is to create a number representing the environmental factors that influence the team's ability to perform the work, called the Environmental Factor (EF), which includes characteristics of the implementation team and the process, such as team experience and requirements stability.

The third step is to measure the use cases and create a representation of the quantity and complexity of the use cases that the software must support, called Unadjusted Use Case Points (UUCP), classifying use cases as simple, average, or complex.

In the fourth step, the software users—both people and other systems—are analyzed, and actors are classified as simple, average, or complex (Actor Weight, AW).

Finally, the formula to calculate Use Case Points is described in terms of variables such as TCF, EF, UUCP, and AW [13].

In the UCP cell (see Figure 2), equation 1 is used to calculate the Use Case Points. These points are then multiplied by the "Ratio," which represents the effort hours per use case point, to obtain the effort hours for the project's use cases. This translates into the estimated programming hours for those use cases.

For the total project calculation, we recommend that programming time be 40% of the total development time, based on the work of Suresh Nageswaran [9]. For the remaining development activities, the percentage of total time allocated to each can be chosen freely. As shown in the spreadsheet (see Figure 2), these recommendations are based on the work of Lianny O'Farrill Fernández [3].

$$\mathrm{UCP} = (\mathrm{UUCP} + \mathrm{AW}) \times \mathrm{TCF} \times \mathrm{EF} \quad (1)$$

Figures 2 to 7 show the spreadsheets corresponding to each factor.

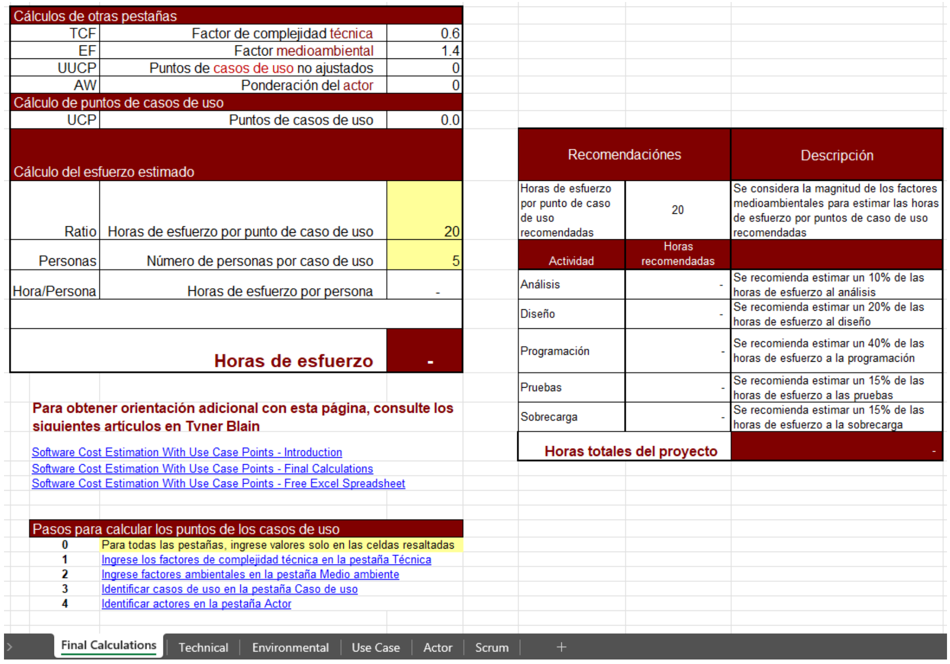
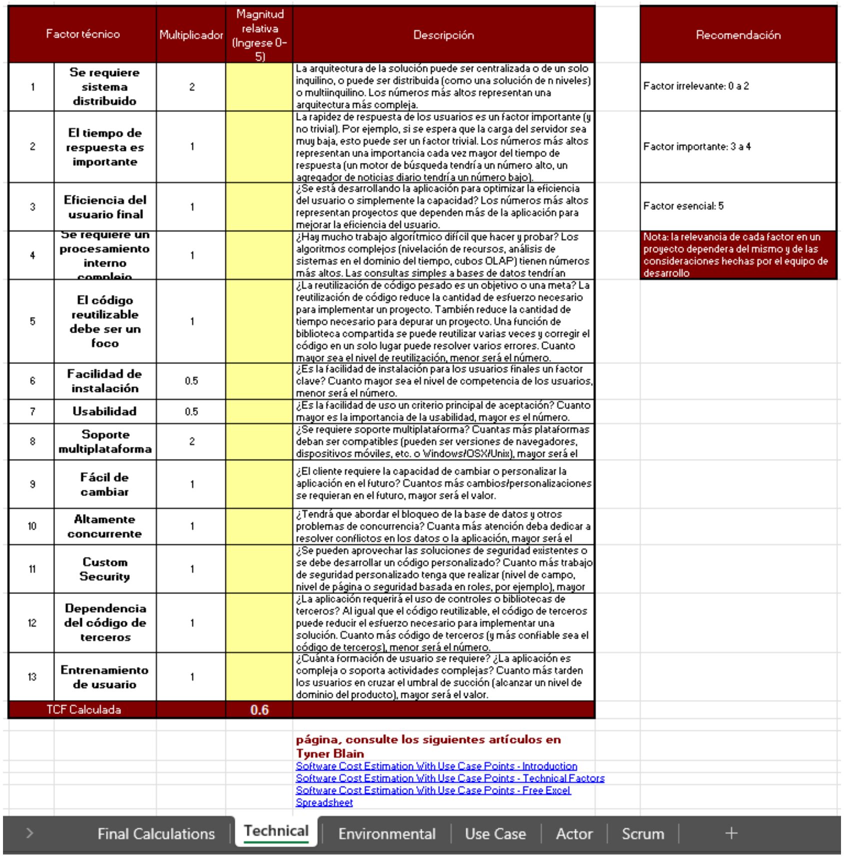


Figure 2. Final calculations spreadsheet. Source: "Final Calculations" Excel spreadsheet.

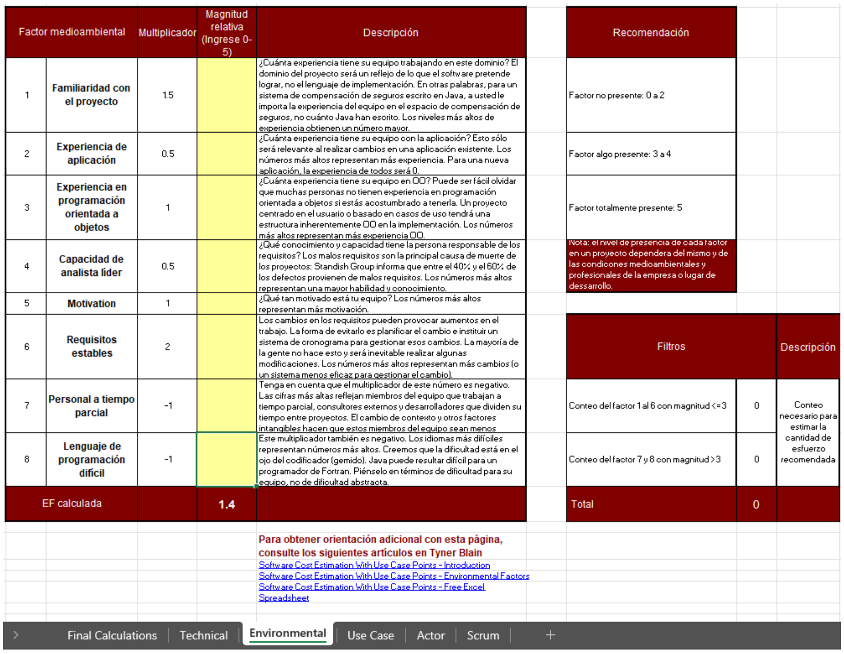


Factor técnico		Multiplicador	Magnitud relativa (Ingrese 0-5)	Descripción	Recomendación
1	Se requiere sistema distribuido	2		La arquitectura de la solución puede ser centralizada o de un solo inquilino, o puede ser distribuida (como una solución de n niveles) o multiinquilino. Los números más altos representan una arquitectura más compleja.	Factor irrelevante: 0 a 2
2	El tiempo de respuesta es importante	1		La rapidez de respuesta de los usuarios es un factor importante (y no trivial). Por ejemplo, si se espera que la carga del servidor sea muy baja, esto puede ser un factor trivial. Los números más altos representan una importancia cada vez mayor del tiempo de respuesta (un motor de búsqueda tendría un número alto, un agregador de noticias diario tendría un número bajo).	Factor importante: 3 a 4
3	Eficiencia del usuario final	1		¿Se está desarrollando la aplicación para optimizar la eficiencia del usuario o simplemente la capacidad? Los números más altos representan proyectos que dependen más de la aplicación para mejorar la eficiencia del usuario.	Factor esencial: 5
4	Se requiere un procesamiento interno complejo	1		¿Hay mucho trabajo algorítmico difícil que hacer y probar? Los algoritmos complejos (nivelación de recursos, análisis de sistemas en el dominio del tiempo, cubos OLAP) tienen números más altos. Las consultas simples a bases de datos tendrían	Nota: la relevancia de cada factor en un proyecto dependera del mismo y de las consideraciones hechas por el equipo de desarrollo
5	El código reutilizable debe ser un foco	1		¿La reutilización de código pesado es un objetivo o una meta? La reutilización de código reduce la cantidad de esfuerzo necesario para implementar un proyecto. También reduce la cantidad de tiempo necesario para depurar un proyecto. Una función de biblioteca compartida se puede reutilizar varias veces y corregir el código en un solo lugar puede resolver varios errores. Cuanto mayor sea el nivel de reutilización, menor será el número.	
6	Facilidad de instalación	0.5		¿Es la facilidad de instalación para los usuarios finales un factor clave? Cuanto mayor sea el nivel de competencia de los usuarios, menor será el número.	
7	Usabilidad	0.5		¿Es la facilidad de uso un criterio principal de aceptación? Cuanto mayor es la importancia de la usabilidad, mayor es el número.	
8	Soporte multiplataforma	2		¿Se requiere soporte multiplataforma? Cuantas más plataformas deban ser compatibles (pueden ser versiones de navegadores, dispositivos móviles, etc. o Windows/OSX/Unix), mayor será el	
9	Fácil de cambiar	1		¿El cliente requiere la capacidad de cambiar o personalizar la aplicación en el futuro? Cuantos más cambios/personalizaciones se requieran en el futuro, mayor será el valor.	
10	Altamente concurrente	1		¿Tendrá que abordar el bloqueo de la base de datos y otros problemas de concurrencia? Cuanta más atención deba dedicar a resolver conflictos en los datos o la aplicación, mayor será el	
11	Custom Security	1		¿Se pueden aprovechar las soluciones de seguridad existentes o se debe desarrollar un código personalizado? Cuanto más trabajo de seguridad personalizado tenga que realizar (nivel de campo, nivel de página o seguridad basada en roles, por ejemplo), mayor	
12	Dependencia del código de terceros	1		¿La aplicación requerirá el uso de controles o bibliotecas de terceros? Al igual que el código reutilizable, el código de terceros puede reducir el esfuerzo necesario para implementar una solución. Cuanto más código de terceros (y más confiable sea el código de terceros), menor será el número.	
13	Entrenamiento de usuario	1		¿Cuánta formación de usuario se requiere? ¿La aplicación es compleja o soporta actividades complejas? Cuanto más tarden los usuarios en cruzar el umbral de succión (alcanzar un nivel de dominio del producto), mayor será el valor.	
TCF Calculada			0.6		

página, consulte los siguientes artículos en Tyner Blain
Software Cost Estimation With Use Case Points - Introduction
Software Cost Estimation With Use Case Points - Technical Factors
Software Cost Estimation With Use Case Points - Free Excel Spreadsheet

Final Calculations | Technical | Environmental | Use Case | Actor | Scrum | +

Figure 3. Technical factors spreadsheet. Source: "Technical" Excel spreadsheet.



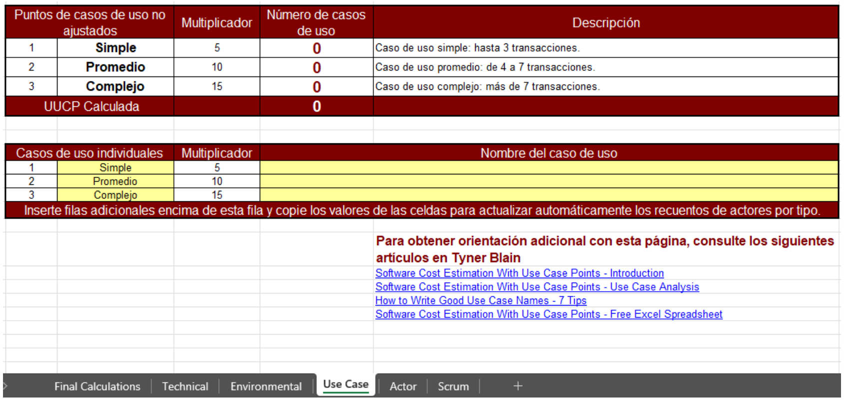
Factor medioambiental		Multiplicador	Magnitud relativa (Ingrese 0-5)	Descripción	Recomendación
1	Familiaridad con el proyecto	1.5		¿Cuánta experiencia tiene su equipo trabajando en este dominio? El dominio del proyecto será un reflejo de lo que el software pretende lograr, no el lenguaje de implementación. En otras palabras, para un sistema de compensación de seguros escrito en Java, a usted le importa la experiencia del equipo en el espacio de compensación de seguros, no cuánto Java han escrito. Los niveles más altos de experiencia obtienen un número mayor.	Factor no presente: 0 a 2
2	Experiencia de aplicación	0.5		¿Cuánta experiencia tiene su equipo con la aplicación? Esto sólo será relevante al realizar cambios en una aplicación existente. Los números más altos representan más experiencia. Para una nueva aplicación, la experiencia de todos será 0.	Factor algo presente: 3 a 4
3	Experiencia en programación orientada a objetos	1		¿Cuánta experiencia tiene su equipo en OO? Puede ser fácil olvidar que muchas personas no tienen experiencia en programación orientada a objetos si estás acostumbrado a tenerla. Un proyecto centrado en el usuario o basado en casos de uso tendrá una estructura inherentemente OO en la implementación. Los números más altos representan más experiencia OO.	Factor totalmente presente: 5
4	Capacidad de analista líder	0.5		¿Qué conocimiento y capacidad tiene la persona responsable de los requisitos? Los malos requisitos son la principal causa de muerte de los proyectos: Standish Group informa que entre el 40% y el 60% de los defectos provienen de malos requisitos. Los números más altos representan una mayor habilidad y conocimiento.	Nota: el nivel de presencia de cada factor en un proyecto dependera del mismo y de las condicones medioambientales y profesionales de la empresa o lugar de dessarrollo.
5	Motivation	1		¿Qué tan motivado está tu equipo? Los números más altos representan más motivación.	
6	Requisitos estables	2		Los cambios en los requisitos pueden provocar aumentos en el trabajo. La forma de evitarlo es planificar el cambio e instituir un sistema de cronograma para gestionar esos cambios. La mayoría de la gente no hace esto y será inevitable realizar algunas modificaciones. Los números más altos representan más cambios (o un sistema menos eficaz para gestionar el cambio).	
7	Personal a tiempo parcial	-1		Tenga en cuenta que el multiplicador de este número es negativo. Las cifras más altas reflejan miembros del equipo que trabajan a tiempo parcial, consultores externos y desarrolladores que dividen su tiempo entre proyectos. El cambio de contexto y otros factores intangibles hacen que estos miembros del equipo sean menos	
8	Lenguaje de programación difícil	-1		Este multiplicador también es negativo. Los idiomas más difíciles representan números más altos. Creemos que la dificultad está en el ojo del codificador (gemido). Java puede resultar difícil para un programador de Fortran. Piénselo en términos de dificultad para su equipo, no de dificultad abstracta.	
EF calculada			1.4		

Filtros		Descripción
Conteo del factor 1 al 6 con magnitud <=3	0	Conteo necesario para estimar la cantidad de esfuerzo recomendada
Conteo del factor 7 y 8 con magnitud >3	0	
Total	0	

Para obtener orientación adicional con esta página, consulte los siguientes artículos en Tyner Blain
Software Cost Estimation With Use Case Points - Introduction
Software Cost Estimation With Use Case Points - Environmental Factors
Software Cost Estimation With Use Case Points - Free Excel Spreadsheet

Final Calculations | Technical | Environmental | Use Case | Actor | Scrum | +

Figure 4. Environmental factors spreadsheet. Source: "Environmental" Excel spreadsheet.



Puntos de casos de uso no ajustados		Multiplicador	Número de casos de uso	Descripción
1	**Simple**	5	0	Caso de uso simple: hasta 3 transacciones.
2	**Promedio**	10	0	Caso de uso promedio: de 4 a 7 transacciones.
3	**Complejo**	15	0	Caso de uso complejo: más de 7 transacciones.
UUCP Calculada			0	

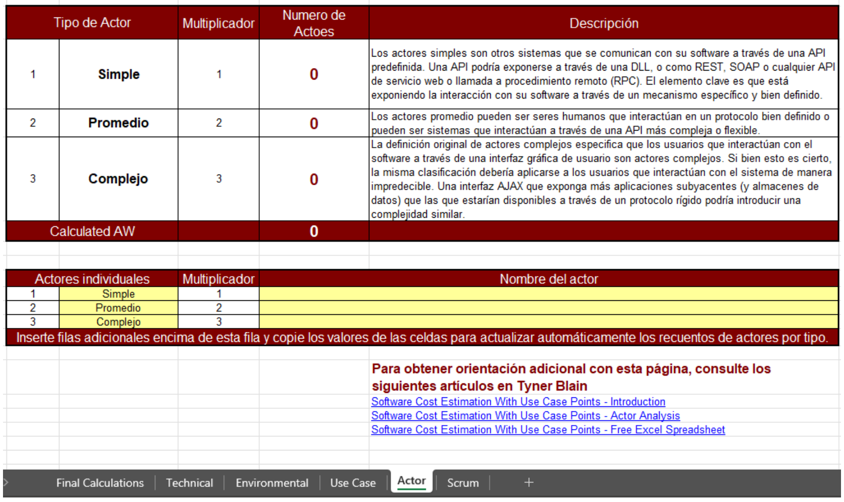
Casos de uso individuales		Multiplicador	Nombre del caso de uso
1	Simple	5	
2	Promedio	10	
3	Complejo	15	

Inserte filas adicionales encima de esta fila y copie los valores de las celdas para actualizar automáticamente los recuentos de actores por tipo.

Para obtener orientación adicional con esta página, consulte los siguientes artículos en Tyner Blain
Software Cost Estimation With Use Case Points - Introduction
Software Cost Estimation With Use Case Points - Use Case Analysis
How to Write Good Use Case Names - 7 Tips
Software Cost Estimation With Use Case Points - Free Excel Spreadsheet

Final Calculations | Technical | Environmental | Use Case | Actor | Scrum

Figure 5. Use case spreadsheet. Source: "Use Case" Excel spreadsheet.



Tipo de Actor		Multiplicador	Numero de Actoes	Descripción
1	**Simple**	1	0	Los actores simples son otros sistemas que se comunican con su software a través de una API predefinida. Una API podría exponerse a través de una DLL, o como REST, SOAP o cualquier API de servicio web o llamada a procedimiento remoto (RPC). El elemento clave es que está exponiendo la interacción con su software a través de un mecanismo específico y bien definido.
2	**Promedio**	2	0	Los actores promedio pueden ser seres humanos que interactúan en un protocolo bien definido o pueden ser sistemas que interactúan a través de una API más compleja o flexible.
3	**Complejo**	3	0	La definición original de actores complejos especifica que los usuarios que interactúan con el software a través de una interfaz gráfica de usuario son actores complejos. Si bien esto es cierto, la misma clasificación debería aplicarse a los usuarios que interactúan con el sistema de manera impredecible. Una interfaz AJAX que exponga más aplicaciones subyacentes (y almacenes de datos) que las que estarían disponibles a través de un protocolo rígido podría introducir una complejidad similar.
Calculated AW			0	

Actores individuales		Multiplicador	Nombre del actor
1	Simple	1	
2	Promedio	2	
3	Complejo	3	

Inserte filas adicionales encima de esta fila y copie los valores de las celdas para actualizar automáticamente los recuentos de actores por tipo.

Para obtener orientación adicional con esta página, consulte los siguientes artículos en Tyner Blain
Software Cost Estimation With Use Case Points - Introduction
Software Cost Estimation With Use Case Points - Actor Analysis
Software Cost Estimation With Use Case Points - Free Excel Spreadsheet

Final Calculations | Technical | Environmental | Use Case | Actor | Scrum

Figure 6. Actor spreadsheet. Source: "Actor" Excel spreadsheet.

Sprint	Tarea principal	Descripción de tarea	Tipo de desarrollo	Tipo de actividad	Prioridad	Fase	Desarrollador	Horas por tarea	Estado
Sprint 1			No Aplica	Análisis	Media	Fase 1			En proceso
Sprint 1			No Aplica	Diseño	Media	Fase 1			En proceso
Sprint 2			No Aplica	Programación	Alta	Fase 1			Terminado
Sprint 2			No Aplica	Pruebas	Alta	Fase 1			Terminado
Sprint 3			No Aplica	Sobrecarga	Alta	Fase 3			Pendiente
							Horas Totales:	0	

Horas Estimadas del proyecto 0

Actividad	Horas recomendadas	Horas Establecidas
Análisis	-	-
Diseño	-	-
Programación	-	-
Pruebas	-	-
Sobrecarga	-	-

Final Calculations | Technical | Environmental | Use Case | Actor | Scrum

Figure 7. Scrum organization spreadsheet. Source: "Scrum" Excel spreadsheet.

With the changes and additions made in this implementation of the Use Case Point Method (UCP), it was possible to compare the initial estimations obtained by both methodologies, with the UCP providing estimates that were more distant than those made by developers using Scrum.

Implemented case studies

Three tests of the solution were conducted as follows:

Case 1: Geek Web – Communication and socialization platform for communities (not executed). Geek Web is a communication platform designed to connect people with shared interests, allowing them to share knowledge, experiences, and hobbies. The idea arises from the need to create a space where enthusiasts of specific topics can find and interact with others who share their interests. Its objectives are to create a secure and accessible communication platform for interest-based communities and to facilitate connection and knowledge exchange among like-minded individuals (Table 2).

For the case studies, students of Systems Engineering at the University of Córdoba, who were working on their respective projects, were asked to complete the technical

Table 2. MPCU results for Geek Web.

Factor	Weight
TCF	1.195
EF	0.815
UUCP	105
AW	8
UCP	110.1
Ratio	28
Effort Hours	3081

complexity, environmental factors, actors, and use cases for the estimation of Use Case Points (UCP) and consequently fill in the "Scrum" tab, emphasizing the estimates they considered.

Case 2: Medical Appointment Scheduling at Camus (Testing Phase). At Camus, the medical appointment scheduling system combines in-person service by quota and phone calls, resulting in long queues of users waiting to be attended. The manual scheduling process can take up to 25 minutes. Since many of the people who come to schedule appointments are from remote villages or distant places, quotas are limited, and therefore, not everyone is attended to on the same day. This causes people to have to queue again to reserve a spot. The objectives are to implement an efficient and accessible medical appointment scheduling system, reduce waiting times and queues, increase the user service capacity, and improve user experience (Table 3).

Table 3. MPCU results for Scheduling Medical Appointments at the Campuses.

Factor	Weight
TCF	1.015
EF	0.74
UUCP	25
AW	2
UCP	20.3
Ratio	28
Effort Hours	568

Case 3: University Wheels (Testing Phase). In the city of Montería, located in the department of Córdoba, mobility is a fundamental aspect for the development of its inhabitants, particularly for university students. The university transportation system plays a crucial role in seeking an efficient and reliable solution to facilitate mobility within the city and ensure that students can access their educational institutions in a timely and safe manner. Its objectives are to improve the safety of students and the community, offer an efficient and accessible transportation service, and facilitate the movement of students to their educational institutions and the community to their destinations of interest (Table 4).

Table 5 presents a comparison of effort estimations, measured in hours, obtained through the Use Case Point Method (UCP) and the Scrum methodology for three different projects. This comparison highlights differences in estimation approaches and provides insight into how each method evaluates the time required for software development.

Table 4. MPCU results for University Rounds.

Factor	Weight
TCF	1.09
EF	0.77
UUCP	50
AW	6
UCP	47
Ratio	20
Effort Hours	940

Table 5. MPCU vs SCRUM results.

Project	MPCU (Hours)	Scrum (Hours)	Remarks
Geek Web	3081.4987	2266	The estimated time is lower in Scrum than in UCP. Here, only programming time is estimated, not the entire project (analysis, design, programming, testing, etc.).
Medical Appointment Scheduling	567.8316	140	
University Wheels	1504.0256	1230	Total project time is estimated. UCP estimation remains less "optimistic" than Scrum.

6. Conclusions

The primary focus of this study was to conduct an appreciative observation of the current state of software planning by using the agile methodology as a benchmark and thoroughly exploring the strengths and limitations of the Use Case Point (UCP) Method. To facilitate the implementation and evaluation of the proposed hybrid solution, the widely accessible accounting tool Microsoft Excel was employed, enabling a practical and replicable estimation process. The case studies revealed a notable divergence between the effort estimates generated by the UCP method and the initially more "optimistic" projections derived from the Scrum methodology, highlighting differences in estimation philosophies and potential biases.

These findings underscore the value of integrating the structured and systematic nature of UCP with the adaptive and iterative characteristics of agile methods. It is anticipated that this combination can provide a robust and reliable initial estimate while simultaneously maintaining the flexibility necessary to accommodate evolving requirements and unforeseen changes during the software development lifecycle. Following the establishment of a solid baseline estimate, project teams are positioned to leverage agile principles for ongoing project execution and refinement.

Nevertheless, the study emphasizes the importance of managing requirement changes carefully, as excessive modifications during development can undermine the accuracy of the preliminary estimate and necessitate continuous re-planning and adjustment for both methodologies. Future research is recommended to evaluate the scalability and adaptability of this integrated approach in more complex and larger-scale software development environments, thereby further validating its applicability and identifying opportunities for enhancement.

Author Contributions: **Jose Polo:** Conceptualization, Methodology, Software, Visualization, Validation, Formal analysis.
Yeinis Espitia: Investigation, Resources, Data curation, Writing – original draft.
Daniel Salas: Writing – review & editing, Supervision, Project administration, Funding acquisition. All authors have read and agreed to the published version of the manuscript. Please refer to the CRediT taxonomy for the definitions of the terms. Authorship is limited to those who have made substantial contributions to the reported work.

Funding: This study did not receive external funding.

Institutional Review Board Statement: Not applicable, since the present study does not involve human personnel or animals.

Informed Consent Statement: This study is limited to the use of technological resources, so no human personnel or animals are involved.

Conflicts of Interest: Under the authorship of this research, it is declared that there is no conflict of interest with the present research.

References

1. Alboka Soft (2023). Planificar en un proyecto de software. https://www.albokasoft.com/index.php/blog/80-proyecto-de-software-a-medida. Accessed: Apr. 24, 2023.
2. Coding Sans (2021). The state of software development 2021.
3. Fernández, L. O. (2010). Estimación de tiempo y esfuerzo en proyectos de software. Technical report, Universidad Central Marta Abreu de las Villas.
4. Garzón, E. (2019). Elaboración de un modelo para la implementación de controles eficaces para la gestión de riesgos en proyectos de software bajo el marco de estándares internacionales en empresas fábricas de software en bogotá. Technical report, Universidad Militar Nueva Granada.
5. Ibeto, O. E., Gbadegesin, M., Fakunle, I., and Wunmi, A. S. (2022). Software project planning, people management, and effects on product quality. *American Journal of Computer Science and Information Technology*, pages 2–9.
6. Ibraigheeth, M. and Fadzli, S. A. (2019). Core factors for software projects success.
7. Li, Z., Jing, X. Y., and Zhu, X. (2018). Progress on approaches to software defect prediction.
8. Matloob, F. et al. (2021). Software defect prediction using ensemble learning: A systematic literature review.
9. Nageswaran, S. (2001). Test effort estimation using use case points.
10. Nakigudde, S. (2019). Project management models and software development project success. ResearchGate.
11. O'Regan, G. (2017). *Software Project Management*. SpringerLink.
12. Pérez, A. (2021). Errores a evitar en la planificación de recursos de un proyecto. https://www.obsbusiness.school/blog/errores-evitar-en-la-planificacion-de-recursos-de-un-proyecto. Accessed: Apr. 24, 2023.
13. Sehlhorst, S. (2007a). Software cost estimation with use case points – final calculations. Tyner Blain. Feb. 19, 2007.
14. Sehlhorst, S. (2007b). Software cost estimation with use case points – free excel spreadsheet. Tyner Blain. Feb. 20, 2007.
15. Vazquez, P., Panizzi, M., and Bertone, R. (2018). Estimación del esfuerzo del proceso de implantación de software basada en el método de puntos de caso de uso. Technical report, Universidad Tecnológica Nacional.
16. Álvarez, B. O. (2022). Herramienta para la gestión de actividades en los proyectos de software. Technical report, Universidad de Antioquia.

Authors' Biography

Jose David Polo-Vanegas Systems engineering student at the University of Córdoba.

Yeinis Paola Espitia-Priolo Systems engineering student at the University of Córdoba.

Daniel Jose Salas-Álvarez Master's Degree in Computer Science from the Industrial University of Santander.