Modelo de mejora en la determinación de RRHH en proyectos basados en la Metodología SCRUM

Patricia M. Navarro[1], **Diego P. Pinto-Roa**[2]

[1]Universidad Autónoma de Asunción (UAA)
Código Postal 1.209 – 595.495-873 – Asunción – Paraguay

[2]Universidad Nacional de Asunción (UNA)
San Lorenzo – Paraguay

pnavarro@uaa.edu.py, dpinto@pol.una.py

Abstract. *In Software Engineering, Scrum is an Agile Development Methodology that is characterized by the continuous delivery of work, rapid adaptation to changes and customer satisfaction. In Scrum, the software development project is divided into short time blocks called Sprint. Each Sprint has associated a set of specific tasks that must be solved in the compromised time and assigned to developers with different capacities for its execution. The objective of this paper is to propose to optimize the cost and time of a Systems Project, based on the SCRUM Methodology, improving the assignment of tasks to human resources in the development stage, for the progress of each task or sprint. To achieve this a mathematical model is proposed, through a computational optimization tool, which facilitates the assignment of resources in such a way to optimize in time and cost, considering precedence constraints between tasks.*

Resumo. *En Ingeniería de Software, Scrum es una Metodología Desarrollo Ágil que se caracteriza por la entrega continua del trabajo, la adaptación rápida a los cambios y la satisfacción del cliente. En el Scrum el proyecto se divide en bloques temporales cortos llamados Sprint. Cada Sprint tiene asociado un conjunto de tareas específicas que deben ser resueltos en el tiempo comprometido y asignados a desarrolladores para su ejecución. El objetivo de este trabajo es proponer optimizar el costo y tiempo de un Proyecto de Sistemas, basado en la Metodología SCRUM, mejorando la asignación de tareas a los recursos humanos en la etapa de desarrollo, para el avance de cada tarea o sprint. Para lograr esto se propone un modelo matemático, a través de una herramienta de optimización computacional, que facilita la asignación de recursos de tal forma a optimizar en tiempo y en costo, considerando restricciones de precedencias entre tareas.*

1. Introducción

Las empresas actualmente están expuestas a constantes cambios a nivel organizacional, legal o social, por lo que necesitan utilizar una metodología ágil y dinámica para el desarrollo e integración continua, de tal forma que las mismas puedan adaptarse de manera efectiva a las nuevas necesidades y nuevos desafíos. Entre estas metodologías se encuentran XP [Perez 2011], SCRUM [Trigas 2012], Kanban [Kniberg and Skarin 2010], Lean Startup [Ries 2011].

Hemos elegido la metodología ágil SCRUM debido a que es la más utilizada y conocida [Manaure 2018] [Rodriguez and Dorado 2015] [Deemer et al. 2015]. Entre sus beneficios se observa que mejora los tiempos de desarrollo, ya que cada tarea o sprint debe ser desarrollada en un periodo de tiempo no mayor a un mes. Otro beneficio de esta metodología es que contribuye a la satisfacción del cliente o el usuario final, debido a la interacción profunda entre los involucrados con el equipo de desarrollo.

Para poder llegar a la entrega del producto en el tiempo comprometido, es muy importante la correcta asignación de tareas. Cada miembro del equipo tiene una capacidad diferente, que debe ser bien aprovechada, con el fin de lograr el producto en un tiempo óptimo y con la calidad esperada. En la práctica cotidiana la asignación de tareas del equipo se realiza de forma semialeatoria basada en la experiencia empírica del equipo de desarrollo lo cual no necesariamente genera una asignación óptima, de ahí surge la problemática de cómo realizar dicha asignación de tal forma que no se produzcan retrasos en cuanto a tiempo por una mala administración del recurso, elevando así el costo del proyecto.

Este trabajo propone desarrollar un modelo de optimización multiobjetivo basado en la suma ponderada, que busca proponer la asignación de tareas adecuadas a los desarrolladores, minimizando el tiempo y el costo a la vez.

2. Manifiesto Ágil

En marzo de 2001, se reunieron en Salt Lake City 17 críticos de los modelos de producción basados en proceso a raíz de la convocatoria de Kend Beck, para discutir sobre el desarrollo de software [Trigas 2012]. En la reunión se empleó el término "Métodos ágiles", para los procesos que buscan la continua evolución y mejora, y permiten una mayor adaptación continua.

Además, se definieron cuatro postulados a los que se les denominó "Manifiesto Ágil", que son los valores en los cuales se basan estos métodos, los cuales se citan a continuación [Herrera and Valencia 2007]:

- Valorar más a los individuos y su interacción que a los procesos y las herramientas. Los procesos son una guía y las herramientas ayudan a mejorar la eficiencia, pero debe existir una interacción con las personas capaces que aporten sus conocimientos, obteniendo mayor creatividad e innovación.
- Valorar más al software más que a la documentación exhaustiva. En este punto se aclara que el método ágil no da por inútil la documentación, sólo a la documentación innecesaria.
- Valorar más la colaboración con el cliente, por encima de la negociación contractual. Se busca proporcionar el valor posible al producto, brindando una mayor adaptación ante los cambios en el negocio del cliente.
- Valorar más la respuesta al cambio que el seguimiento de un plan. Los valores principales del Método Ágil es su capacidad de anticipación y adaptación a los cambios, no busca seguir un plan estricto, donde se controle que no existan desviaciones en el mismo.

Los 12 principios del Manifiesto Ágil, basado en estos postulantes son [Trigas 2012] [Orjuela and Rojas 2008]:

1. La principal prioridad es satisfacer al cliente a través de una entrega temprana y continua del software.
2. Los procesos ágiles se adaptan mejor a los cambios, por lo que son una ventaja competitiva para el cliente.
3. Entrega continua del software en un periodo de un par de semanas hasta un par de meses, con preferencia de periodos breves.
4. Los usuarios y los desarrolladores deben trabajar juntos y en forma cotidiana durante todo el proyecto.
5. Equipo de trabajo motivado, gracias a las oportunidades y al respaldo que necesitan.
6. Comunicación fluida dentro del equipo de desarrollo.
7. El software funcionando es la principal medida del proceso.
8. Los procesos ágiles promueven el desarrollo sostenido. Los clientes y el equipo de trabajo mantienen un ritmo constante de forma indefinida.
9. La continua mejora favorece la agilidad.
10. La simplicidad con arte de aumentar la cantidad de trabajo es esencial.
11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos que se autoorganizan.
12. Regularmente el equipo se reúne, buscando la mejora y la efectividad del proceso cambiando así su gestión.

3. Metodologías Ágiles

Las metodologías ágiles aparecen como una nueva alternativa a las metodologías tradicionales de desarrollo, a raíz de la necesidad de mejorar los tiempos de desarrollo, adaptación a los cambios y la reducción de los costos.

Son modelos de desarrollo basados en la iteración, en la comunicación y en reducir los elementos medios. Entre las mismas podemos citar: XP o Programación Extrema, Lean startup, Kamban o Tarjeta Señalizadora, Scrum, entre otras.

El desarrollo con iteraciones se realiza en periodos de tiempos muy cortos y las tareas son realizadas por un equipo de desarrollo multidisciplinario y autoorganizado, que deciden cómo gestionar dichas tareas [Trigas 2012].

3.1. Diferencia entre las Metodologías Tradicionales y Ágiles

Las metodologías tradicionales son rigurosas, están basadas en la planificación y en las fases de vida de un proyecto: concepción, elaboración, construcción y transición [Figueroa et al. 2008]. Se concibe un sólo proyecto de grandes dimensiones. El proceso es secuencial, no cambia. El requerimiento es acordado de una vez y para todo el proyecto, existe poca comunicación con el cliente [Navarro et al. 2013]

Las metodologías ágiles utilizan técnicas para agilizar el desarrollo del software, da importancia a las personas, mayor participación y satisfacción del cliente, mayor flexibilidad para adaptarse a los cambios.

La Tabla 1 muestra las principales diferencias de las metodología ágiles con respecto a las tradicionales. Estas diferencias hacen referencia a aspectos organizacionales y

de desarrollo. Se puede observar que las metodologías de desarrollo ágil son más orientadas a procesos de desarrollo de software de pocas semanas de desarrollo y con bajos niveles de formalización en la documentación requerida [Canós et al.].

TABLA 1 - Diferencias entre metodologías ágiles y no ágiles

Fuente: José H. Canós, Patricio Letelier y Mª Carmen Penadés (2012)

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de códigos.	Basados en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo).	Impuestas externamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el tiempo de desarrollo mediante reuniones.
Menos artefactos.	Más artefactos.
Menos roles.	Más roles.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos

3.2. XP o Programación Extrema.

Es una metodología de desarrollo ágil, donde el desarrollo es integral, existe una participación activa con el cliente y una respuesta rápida al cambio [Perez 2011].

Perez [Perez 2011] cita los principios de desarrollo en XP:

1. Planificación Integral: Se registran las funcionalidades en historias de usuarios, las cuales son negociadas progresivamente con el cliente.
2. Entregas pequeñas: Se desarrolla la mínima parte que proporcione funcionalidad al sistema, y de a poco se van incrementando las funcionalidades a la primera entrega, el ciclo termina con la entrega del sistema.
3. Diseño sencillo: Sólo se realiza el diseño para el requerimiento actual.
4. Desarrollo previamente aprobado: Primero se realiza un prototipo y luego el desarrollo, esto brinda la satisfacción del usuario final.
5. Limpieza del código o refactorización: Simplificar el desarrollo sin perder funcionalidad.
6. Programación en parejas: La metodología propone que los programadores trabajen en pareja en una misma máquina, verificando mutuamente sus trabajos y ayudándose en buscar las mejores soluciones. Con esto se busca que el trabajo sea más eficiente y de mayor calidad.

7. Propiedad colectiva: El conocimiento y la información es de todos, todos los desarrolladores deben conocer el código y poder sugerir y realizar mejoras.
8. Integración continua: Al terminar una tarea, la misma se integra al sistema entero y se realizan pruebas de unidad a todo el sistema, esto hace que la aplicación sea funcional y se integre a los demás módulos.
9. Ritmo sostenible: No es aceptable trabajar tantas horas porque afecta la calidad y la productividad, se proponen 40 hs. semanales.
10. Cliente presente: Debe haber un cliente o usuario final tiempo completo, el mismo se hace parte del equipo.

3.3. Lean Startup

"Lean Startup es un conjunto de prácticas de ayuda a los emprendedores a incrementar las probabilidades de crear startup con éxito." [Ries 2011]. Un startup no es una empresa, es una institución humana diseñada para crear un nuevo producto o servicio bajo condiciones de incertidumbres extremas, por lo que cualquier persona que trabaje dentro de un startup es un emprendedor. Los startups no sólo sirven para crear un producto, son útiles también para aprender a cómo crear negocios sostenibles, debido a que se pueden llevar a cabo experimentos frecuentemente que permitan a los emprendedores probar todos los elementos de sus ideas. A medida que los usuarios utilizan el producto generan feedback y datos. El feedback puede ser cualitativo (por ejemplo, si le atrae o acepta la gente) o cuantitativo (por ejemplo, la cantidad de gente que le gusta o acepta) [Ries 2011].

En la tabla 2 se visualiza el circuito del feedback "Crear-Medir-Aprender", donde la clave es construir un prototipo del Producto Mínimo Viable (PMV), que sirva para medir su viabilidad y aprender en base a los resultados obtenidos para seguir mejorando.

TABLA 2 - Circuito del feedback "Crear-Medir-Aprender".

Fuente: Ries, E. (2011)

Etapas	Descripción
Crear	Convertir las ideas en productos. Se Elabora un prototipo que contenga el Producto Mínimo Viable (PMV) que se desea realizar.
Medir	El cliente prueba el prototipo y se mide su reacción, podría aceptar o rechazar la idea que llevó a la creación del PMV.
Aprender	Concluir en base a los resultados anteriores, para proponer una nueva iteración del ciclo.

Una vez finalizado el circuito el emprendedor debe decidir si pivotar de la idea inicial o perseverar [Ries 2011].

3.4. Kanban o Tarjeta Señalizadora

En la metodología ágil Kanban el trabajo se divide en bloques. No se puede iniciar un bloque, hasta que el otro haya sido entregado o haya pasado a una fase posterior de la cadena. Se genera una señal que indica que hay nuevos bloques a ser iniciados porque el trabajo en curso no alcanza el esperado. El bloque de trabajo se limita por el WIP (Work in Process, trabajo en curso), donde se establecen cuántos elementos pueden estar en proceso en cada estado del flujo de trabajo [Kniberg and Skarin 2010].

Dado que el WIP está limitado en el sistema Kanban cualquier elemento que se bloquea tiende a bloquear el proyecto, esto implica que todo el equipo de trabajo se concentre en solucionar el problema, eliminando el bloqueo y permitiendo la continuidad del trabajo. El tiempo fijo en las iteraciones es opcional, la duración puede variar en función a la planificación del producto o la mejora del proceso. El lead time o tiempo de entrega, o tiempo medio, se debe optimizar para completar un elemento. El mismo debe ser tan pequeño y predecible como sea posible [Kniberg and Skarin 2010].

Kanban es un sistema de planificación pull, esto implica que el equipo de trabajo indique en cuánto tiempo y cuánto trabajo se compromete realizar. El equipo "tira" del trabajo cuando está listo, en contraposición a que el exterior "empuje" al equipo a hacerlo. El equipo de trabajo puede ser multidisciplinario o especializado y no existen roles prescritos. El compromiso del equipo de trabajo, las estimaciones y la priorización son opcionales [Kniberg and Skarin 2010].

4. Scrum

4.1. Origen del Scrum

Esta metodología fue definida por Ikujiro Nonaka e Hiroka Takeuchi a principio de los 80, al estudiar el desarrollo de los nuevos productos de las principales empresas de manufactura tecnológicas: Fuji-Xerox, Canon, Honda, Nec, Epson, Brother, 3 M y Hewlett-Packard [Takeuchi and Nonaka 1986].

En su estudio compararon estas tecnologías, la forma de trabajo en equipo, con el avance de la formación de scrum de los jugadores de Rugby, de allí su denominación "Scrum".

En 1995 Ken Schwaber presentó "Scrum Development Process" en OOPSLA 95 (Object-Oriented Programming Systems & Applications conference), un marco de reglas para el desarrollo de software, basados en los principios de Scrum y que él empleó en el desarrollo de software de Delphi y Jeff Sutherland en su empresa Easel Corporation.

5. Definición del Scrum

"El Scrum es una metodología de Desarrollo de Software Ágil, iterativa, dispuesta al cambio, que fortalece la satisfacción del cliente, y se basa en principios de inspección y adaptación" [Rodriguez 2015].

Según Trigas [Trigas 2012], SCRUM es adecuado para empresas que tienen:

- Incertidumbre: Sobre esta variable se plantea el objetivo del proyecto que se desea alcanzar.
- Autoorganización: Los equipos deben ser autogestionados.
- Autonomía: Encontrar una solución utilizando la estrategia adecuada.
- Autosuperación: Las soluciones iniciales sufrirán mejoras.
- Autoenriquecimiento: El equipo debe ser autodisciplinario.
- Control moderado: Se realiza un control entre iguales para permitir la creatividad de los miembros del equipo.
- Transmisión del conocimiento: Todos aprenden de todo, las personas pasan de proyecto a proyecto, aprendiendo conocimiento y transmitiendo.

5.1. Componentes

Rodriguez [Rodriguez 2015] explica los componentes que se necesitan para llevar a cabo la Metodología Scrum:

- **Eventos llamados Sprint:** Es el ciclo de desarrollo, que se repite hasta finalizar el producto. Se divide en seis actividades que son: Sprint Planning, Sprint Execution, Daily Meeting, Sprint Review, Sprint Retrospective, Refinement.
- **Artefactos:** Se citan las herramientas utilizadas en Scrum:
  - **Product Backlog:** Lista de funcionalidades a desarrollar [Murrada and Pons 2010].
  - **Sprint Backlog:** Lista más pequeña de tareas a desarrollar, en un tiempo de máximo 4 semanas, se basa en las funcionalidades definidas en el Product Backlog [Murrada and Pons 2010].
  - **Product Increment:** El producto es ingresado a producción en forma incremental.
- **Roles:** En Scrum existen 3 roles:
  - **Product Owner:** Es una persona designada que cumple la función de Director de Proyecto en la institución, tiene un profundo conocimiento del negocio, es considerado el propietario de la aplicación que se va a construir. Es el responsable del Product Backlog.
  - **Scrum Master:** Persona con conocimientos de la Metodología Scrum, es un guía para el equipo en el proceso de desarrollo y puede formar parte del Scrum Team.
  - **Development Team:** Equipo de personas que realizarán el desarrollo [Deemer et al. 2015].

  Existe un rol secundario que son los Involucrados, personas a quienes interesa el producto final, pero que no forman parte del equipo de trabajo, en cualquier momento pueden aportar algo en el flujo de desarrollo [Deemer et al. 2015].

5.2. Proceso

El proceso se inicia con la reunión de las personas que cuentan con experiencia en el producto que se desea desarrollar. En la misma se elige al Product Owner y se elabora el "Product Backlog", que es una lista donde se encuentran todas las funcionalidades por orden de prioridad, y está compuesta por las historias de usuarios y casos de uso de cada funcionalidad del producto. Con esta primera reunión se inicia el ciclo de desarrollo, que será repetido varias veces a lo largo del proyecto [Rodriguez 2015].

El proceso continúa con la etapa **Sprint Planning**, se realiza una reunión larga con los desarrolladores, para crear el "Sprint Backlog", que es una lista que contiene una descripción del producto, el listado de las tareas que serán realizadas por el equipo y el tiempo estimado de cada una, que no debe ser mayor a cuatro semanas. Esta lista debe ser clara, porque deberá ser entendida por cada miembro del Equipo de Desarrollo. En esta etapa nace el "Product Increment", el resultado de lo que se venía haciendo más lo nuevo del producto [Rodriguez 2015].

Durante las semanas de desarrollo el Cliente no interviene y deja trabajar al Equipo de Desarrollo. Todos los días se realiza una reunión entre el Scrum Master y el equipo

de trabajo denominada **Daily Meeting** de 15 minutos en donde se verifica lo siguiente: qué se hizo, qué problemas hubo, qué se realizará antes de la siguiente reunión, qué hace falta o es necesario [Rodriguez 2015] [Gonzalez 2012] [Deemer et al. 2015].

Una vez que se cumple el tiempo estimado para la tarea se presenta una demostración al cliente de lo realizado, esta reunión se denomina **Sprint Review**, dura aproximadamente 2 horas, en la misma se verifica si se debe mejorar algo, o si se añadirán nuevas historias al Product Backlog, tales como bugs, nuevas funcionalidades, mejoras, etc. Se evalúa el avance del proyecto, qué falta por cumplir, qué se realizará a continuación y se define si el producto pasa o no a producción [Rodriguez 2015] [Gonzalez 2012] [Deemer et al. 2015].

El evento final es el Sprint **Retrospective**, que es guiado por el Scrum Master, y en él se verifica cómo fue el proceso, qué se hizo bien, qué se debe continuar haciendo y qué se debe mejorar [Rodriguez 2015] [Deemer et al. 2015].

En cualquier momento del ciclo se puede realizar el **Refinament** sobre el Product Backlog, que se encuentra administrado por el Product Owner. En este sprint se aclara las inquietudes de las funcionalidades esperadas, aumentan las funcionalidades, se incluyen temas técnicos y se pueden sacar funcionalidades que ya no apliquen o no sean necesarias [Rodriguez 2015] [Deemer et al. 2015].

5.3. Planning Póker

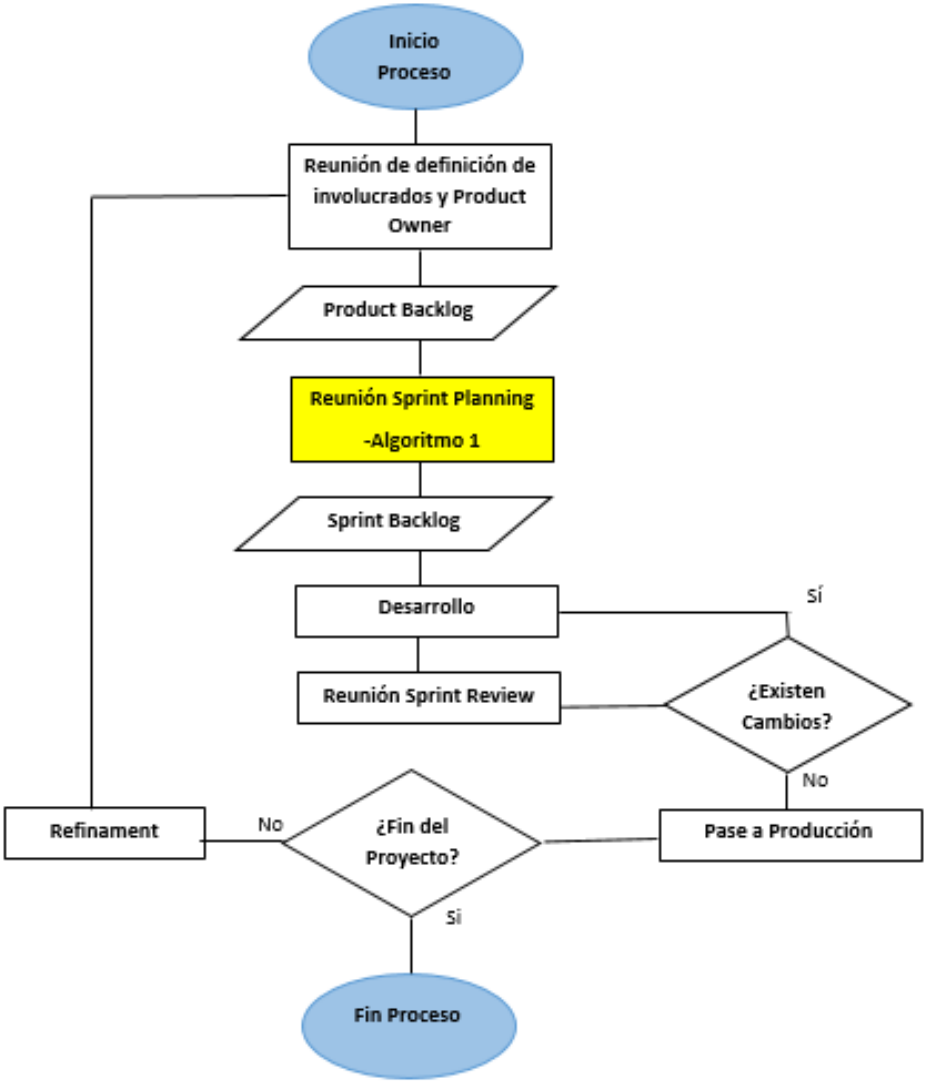
Según Trigas [Trigas 2012] y Javdani [Javdani et al. 2014] el Planning Póker es una técnica que se utiliza en Scrum para la estimación, basada en un consenso, de las historias de usuarios referida a la descripción de la funcionalidad durante el Sprint Planning. Este método inicialmente fue propuesto por J. Grenning en 2002 [Grenning 2002] y luego fue popularizado por M. Cohn en su libro en 2005 [Cohn 2005]. Él, quien es un famoso autor de Scrum, sugirió la aplicación de Planning Póker en la estimación de costos en el desarrollo de software Ágil. Se realizan los siguientes pasos:

1. A cada miembro del equipo se le entregará 13 barajas.
2. Se plantea una funcionalidad del Product Backlog, y cada miembro selecciona una carta y la coloca boca abajo. Esta carta representa la estimación para la funcionalidad propuesta.
3. Cuando todos los miembros realizaron su estimación, se dan vueltas las cartas, todas al mismo tiempo.
4. Se verifican las estimaciones y en caso de que existan muchas diferencias, se discute sobre las mismas y se trata de llegar a un acuerdo en común. El proceso se repite hasta que se llegue a un consenso, otra opción sería descartar las estimaciones mínimas y máximas y quedarse con la media.
5. Las cartas se encuentran enumeradas, cuanto mayor es el número existe mayor incertidumbre.
6. Se repiten los pasos por cada funcionalidad del Backlog Product.

5.4. Flujograma del Proceso de Scrum

Se presenta el flujograma del proceso de Scrum, en el mismo se resalta en color amarillo la reunión de Sprint Planning (véase Flujograma 1), donde se realiza la estimación de tiempo y asignación de desarrolladores a las diferentes tareas del Sprint en base a

la experiencia. No existe un método preciso para dicho trabajo, por lo que se pueden realizar las asignaciones en forma incorrecta, afectando el tiempo o el costo del Sprint. Este trabajo propone un algoritmo para la correcta asignación de desarrolladores que considera el tiempo y el costo del sprint.



Flujograma 1: Flujograma del proceso Scrum.

6. Scrum pragmático

El Scrum se puede utilizar en forma técnica, aplicando reglas definidas, o pragmática, adoptando los valores originales del Scrum y agregándole particularidades. En la forma técnica se utilizan las reglas definidas basándose en los roles, eventos y artefactos. El Scrum pragmático es personalizado y más adaptado a las necesidades del equipo y proyecto. Es importante conocer el Scrum técnico para poder adoptar el Scrum pragmático [Palacio 2014].

Según Palacio [Palacio 2014] en el Scrum pragmático las responsabilidades se asignan a los roles de la estructura de la empresa o se crean nuevos puestos, dependiendo

de la necesidad, lo importante es que se cuente con el conocimiento necesario. Un ejemplo de asignación flexible del proyecto podría ser:

- Garantía de Funcionamiento del SCRUM: Calidad o Procesos
- Garantía de Gestión de Proyectos: Product Manager
- Autoorganización y tecnología ágil: Equipo.

6.1. Proceso del Scrum Pragmático

Existen dos tipos de procesos procedimientos y prácticas. La diferencia de ambos es que en los procedimientos el protagonista es el proceso, es el que define cómo se tiene que hacer y quién debe hacer; y en las prácticas el protagonista es la persona, a través de su conocimiento y su experiencia, y utiliza el proceso como una herramienta de apoyo. En caso de los procedimientos el conocimiento es explícito, contenido en el proceso y la tecnología. En caso de las prácticas el conocimiento es tácito, se encuentra en la persona [Palacio 2014].

Palacio [Palacio 2014] indica que el Scrum Master, debe decidir qué tipo de proceso es necesario aplicar en base al requerimiento:

- Procesos, si su ejecución aporta, en conjunto con la tecnología, permite obtener un mejor resultado.
- Prácticas, que aportan el conocimiento a través de su experiencia y capacidad, obteniendo el mejor resultado.

"Los modelos de ingeniería de procesos, consideran al binomio "proceso-tecnología" como principal responsable de la calidad del resultado. Su antítesis, la agilidad, da el protagonismo del resultado a las personas", como lo muestra la figura 1 [Palacio 2014]. Para el Scrum Master ambas opciones son válidas, siempre y cuando logren el resultado esperado de la mejor manera.

FIGURA 1 - Personas, procedimientos y tecnología.
Juan Palacio, 2014.

7. Algoritmo

Según Vancells [Vancells 2002] el algoritmo es un "Procedimiento de cálculo que consiste en cumplir una serie o conjunto ordenado y finito de instrucciones que conducen, una vez especificados los datos, a la solución que el problema genérico en cuestión tiene para los datos considerados".

Martinez [Martinez 2003] indica "el concepto de algoritmo como un conjunto finito de procesos a su vez finitos y bien definidos que conducen a un resultado".

Diccionario de la Real Academia Española - DRAE (2015) define que es un "Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema".

Por lo que podemos concluir que un algoritmo es un conjunto de pasos que nos permiten solucionar un problema determinado.

8. Problema de la programación de tareas

Se tienen M recursos y N tareas con T_{ij} de duración de tiempo de cada tarea realizada por un recurso. El problema de la programación de tareas presenta una serie de variantes, una de ellas y la más compleja es que existen tareas que son dependientes y los recursos diferentes. Se busca realizar las N tareas con M recursos cumpliendo determinadas condiciones, en el orden apropiado y buscando optimizar la solución del problema [Tupia and Mauricio 2014].

9. Métodos existentes para resolver el problema de programación de tareas y sus derivados

9.1. Métodos Exactos

Buscan hallar un plan jerárquico de ejecución considerando todas las condiciones posibles, las tareas y los recursos involucrados. Este tipo de algoritmo realiza una exploración exhaustiva por lo que la solución es costosa y solo funciona en algunos tipos de instancia [Tupia and Mauricio 2014].

Entre los métodos exactos podemos citar: Programación Exacta, Programación lineal, Programación lineal binaria, Método Thangavelu, Valero y Programación Dinámica [Póveda and Florez 2009].

9.2. Métodos Aproximados

Buscan resolver las variantes más complejas que se podrían presentar en la combinación recursos y tareas. Estos métodos no analizan exhaustivamente todas las posibles combinaciones, sino que se basan en ciertos criterios y obtienen soluciones buenas que ayudan a resolver los problemas [Tupia and Mauricio 2014].

Entre los métodos exactos podemos citar: Algoritmo de Recocido Simulado, la búsqueda Tabú, algoritmos genéricos, redes neuronales artificiales, optimización por colonias de hormigas, búsqueda local iterativas, computación evolutiva, entre otros [Velez and Montoya 2007].

10. Algoritmos voraces

Algoritmo Voraz también denominado ávido, devorador o goloso, es aquel utilizado para resolver un determinado problema. Es una estrategia de búsqueda por la cual se sigue una heurística que consiste en elegir la opción óptima en cada paso local con la esperanza de llegar a una solución óptima general. Normalmente se aplica a resolver los problemas de optimización [Brassard 1997].

11. Algoritmo Voraz para resolver los problemas de la Programación de Tareas

Dado un cierto número de recursos y tareas, un algoritmo voraz busca minimizar el tiempo de procesamiento de lote, realizando una asignación correcta de las tareas a los recursos disponibles [Tupia and Mauricio 2014].

Clasificación basada en la naturaleza de los recursos y las tareas:

- Recursos con capacidades similares y tareas independientes
- Recursos con capacidades similares y tareas dependientes
- Recursos con capacidades diferentes y tareas independientes
- Recursos con capacidades diferentes y tareas dependientes

Los algoritmos voraces han demostrado ser una estrategia poderosa y exitosa para resolver problemas, demostrando el poder de los principios evolutivos. Se han utilizado algoritmos voraces en una amplia variedad de campos para desarrollar soluciones a problemas tan difíciles o más difíciles que los abordados por los diseñadores humanos. Las soluciones son a menudo más eficientes, más elegantes o más complejas que nada que un ingeniero humano produciría [Marczyk 2004].

Franco [Franco 2010] explica que ”los algoritmos Voraces Iterados iteran sobre algoritmos de construcción de la siguiente manera: Dada alguna solución inicial s, primero se quitan algunos componentes de solución, resultando en una solución candidata parcial s_p. A partir de s_p se reconstruye una solución candidata completa s' mediante una heurística de construcción voraz. Luego un criterio de aceptación decide a partir de cual de las dos soluciones (s o s') continúa la próxima iteración”.

12. Sprint Backlog como problema de Programación de tareas

En el Sprint Backlog se definen los tiempos de cada tarea y se realiza la asignación del desarrollador responsable de la misma. Esta tarea es realizada por el equipo de Desarrollo, en forma empírica, basada en la experiencia o el conocimiento de cada desarrollador.

Algunos equipos de desarrollo utilizan la técnica Planning Póker, donde se realiza la estimación de cada funcionalidad del Sprint Backlog, en base a la experiencia del desarrollador, el que brinda menor puntaje a la estimación es el que en teoría tiene mayor conocimiento sobre dicha funcionalidad.

La asignación de tareas del equipo a los desarrolladores realizada en forma semi-aleatoria basada en la experiencia empírica no necesariamente es la óptima o más adecuada, de ahí surge la duda de cómo realizar dicha asignación de tal forma que no se produzcan retrasos en cuanto a tiempo por una mala administración del recurso, elevando el costo del proyecto.

13. Objetivo

El objetivo de este estudio es "Proponer un modelo de optimización multiobjetivo basado en la suma ponderada, que permita optimizar el costo y tiempo de un Proyecto de Sistemas, basado en la Metodología SCRUM, mejorando la asignación de tareas a los recursos humanos en la etapa de desarrollo, para el avance de cada tarea o sprint".

14. Trabajos Relacionados

Al-Zewairi, et. al [Al-Zewairi et al. 2017] han realizado un estudio de la literatura de las diferentes metodologías ágiles, de su artículo se menciona:

- Santana, et. al [Santana et al. 2015] realizan una investigación para detectar la Mejora del Proceso de Sofware (Software Process Improvement, SPI) en un entorno ágil. Los autores identificaron y sugirieron utilizar alguno de los siguientes enfoques SPI ágiles: el enfoque top-down, el enfoque basado en la mejora del comportamiento y el SPI basado en mejores prácticas. El objetivo del SPI tradicional es elaborar un proceso repetible mejorado con el aprendizaje de la experiencia pasada. Por otro lado el SPI ágil es capaz de adaptarse rápidamente y reaccionar a los cambios que pudieran surgir a lo largo del proyecto. El aprendizaje debe basarse en las experiencias individuales de los miembros del equipo.
- Sriram y Mathew [Sriram and Mathew 2012] estudiaron la aplicación de metodologías ágiles en el desarrollo de software global (GSD). Tres ideas principales surgieron sobre GSD: rendimiento del desarrollo de software global, problemas relacionados a la gobernabilidad y al proceso de ingeniería de software. La aplicación de diversos tipos de métodos ágiles produjeron un rendimiento óptimo en el contexto de GSD. El método ágil más común utilizado en GSD es SCRUM.
- Radhakrishnan, et al. [Radhakrishnan et al. 2015] propusieron un modelo de software genérico para fines educativos que mejora el método común de integración de medición de software: punto de función completa (COSMIC FFP). El método sirve para medir el tamaño funcional del software, incluido el software en tiempo real para estimar mejor los recursos y programas del proyecto. El modelo propuesto "eXtreme Software Teaching" (XSOFT) integra el método XP con el método COSMIC FFP para ayudar a reducir el esfuerzo y la diferencia de tiempo entre el desarrollo del software de aprendizaje y el trabajo en el desarrollo de software.
- Raj, et al. [Raj et al. 2015] propuso un proceso de Scrum modificado que se enfoca más en las pruebas, utilizando la implementación de Test-as-a-Service (TAAS) para obtener los resultados más rápido sin aumentar el costo del proyecto.

Mendez [Mendez 2018] propuso un algoritmo de optimización monoobjetivo, que sugiere la asignación de tareas a desarrolladores, minimizando el tiempo de desarrollo, como alternativa al Planning Póker. En base a la búsqueda realizada en las diferentes literaturas sobre metodologías ágiles y Scrum, se considera que la problemática de la asignación de desarrolladores en la Metodología Scrum no fue abordada antes de la propuesta de Mendez.

Para continuar con este trabajo, se toman como muestra de pruebas los mismos proyectos de desarrollo, con sus desarrolladores, la matriz de tiempo y la matriz de precedencia. Se incorporará la matriz de costos y se formulará una función multiobjetivo con el fin de reducir el tiempo y el costo simultáneamente. Cabe destacar que el enfoque de Mendez es un caso particular de la propuesta.

15. Modelo Matemático Propuesto

Dados los siguientes datos de entrada:

M: Cantidad de Desarrolladores disponibles para realizar cada Sprint.
N: Cantidad de Tareas asociadas a cada Sprint.
i: Representa a la tarea unitaria, i = 1, 2,... N.
j: Representa a cada Desarrollador, j = 1, 2,... M.
S_{1j}: Matriz de Costo por hora de cada desarrollador, relacionado a su nivel de conocimiento.
T_{ij}: Matriz de tiempo que a un desarrollador le llevará realizar una determinada tarea.
$P_{ii'}$: Matriz binaria, que indica la precedencia entre la tareas. Si $P_{ii'}$= 1 indica que la tarea i debe ser culminada antes de iniciar la tarea i'. Si $P_{ii'}$ = 0 indica que la tarea i no es precedente a la tarea i', por lo que puede ser ejecutada en forma paralela.
TO: Tiempo libre Total.

Las notaciones y la formulación se presentan a continuación:

Constantes:
U: Tiempo máximo que puede considerarse. U= $\sum_j^i T_{ij}$.
L: Costo mayor de la matriz S.
CMAX: Costo Máximo que puede considerarse C= U · L.

Variables:
X_{ij}: Si X_{ij} = 1 la tarea i es asignada al desarrollador j. Si X_{ij} = 0 la tarea i no se encuentra asignada al desarrollador j.
$Y_{ii'}$: Si $Y_{ii'}$ = 1 indica que la tarea i es ejecutada antes de la tarea i'. Si $Y_{ii'}$ = 0 indica que la tarea i no es ejecutada antes que la tarea i'.
a_i: Tiempo de inicio de la tarea i.
Z : Tiempo total desde el inicio a la finalización de todas las tareas (makespan).
α : Coeficiente utilizado para medir la correlación entre el tiempo y el costo. El rango de α es de 0.01 a 0.99.
w_j: Tiempo total en horas estimado en base a la tarea asignada a cada desarrollador w_j = $\sum_j^i (T_{ij} \cdot X_{ij})$.
C: Costo total desde el inicio a la finalización de todas la tareas.
o_j: Tiempo no empleado por desarrollador.

Función Objetivo a Minimizar: (α · C/CMAX) + ((1 - α) · Z/U) (1)

Sujeto a:

Z >= (a_i + T_{ij}) · X_{ij}, ∀ i, j (2)
$\sum_j^i X_{ij}$ = 1, ∀ i (3)
$a_{i'}$ >= (a_i + T_{ij}) · X_{ij} · $P_{ii'}$, ∀ i ≠ i', ∀ j (4)
$Y_{i'i}$ + $Y_{ii'}$= 1, ∀ i ≠ i' (5)
$a_{i'}$ >= (a_i + T_{ij}) · X_{ij} – U · (3 – $Y_{ii'}$ – X_{ij} – $X_{i'j}$), ∀ i ≠ i', ∀j (6)
w_j = (T_{ij} · X_{ij}) (7)
C >= (S_j · w_j) (8)
o_j = Z – w_j (9)
TO = $\sum o_j$ (10)

La función objetivo (1), busca minimizar el tiempo y costo. El coeficiente α, establece la relación entre tiempo y costo, en caso de que α se acerque a 1, el peso del coeficiente priorizará la optimización del costo, caso contrario priorizará la optimización del tiempo.

La restricción (2) define el tiempo máximo entre todos los desarrolladores. La restricción (3) indica que cada tarea solo puede ser asignada a un desarrollador. La desigualdad (4) garantiza que no hay tiempo superpuesto entre dos tareas con precedencia. Las restricciones (5) y (6) garantiza el tiempo sin superposición entre las tareas asignadas al mismo desarrollador.

La restricción (7) define el tiempo total de cada desarrollador, por la tarea que se le ha sido asignada. La restricción (8) calcula el costo total de cada Sprint. Con la formulación del problema y el modelo matemático se ha formalizado el proceso de asignación de tareas, optimización de tiempo y costo, para los procesos de desarrollo de software orientados a metodologías ágiles (Scrum, para este caso).

Las restricciones (9) y (10) definen el tiempo no empleado total y por cada desarrollador, nos permite analizar si la cantidad de desarrolladores a ser asignado al sprint es la adecuada.

16. Pseudocódigo del Algoritmo propuesto

En el Flujograma 1, presentado en la sección 6.2, se visualiza en color amarillo, la reunión Sprint Planning, donde se realizan las asignaciones de tareas a cada desarrollador y la estimación de tiempo de cada tarea. Para facilitar dicho trabajo se propone el siguiente algoritmo del modelo matemático como herramienta, que se expresa a través del Pseudocódigo 1.

```
Proceso opt_sprint_scrum
    Leer
        numerico M          /*Cantidad de Desarrolladores*/
        numerico N          /*Cantidad de Tareas*/
        numerico alpha      /*Coeficioente de relacion Tiempo*/
        numerico L          /*Costo Máximo*/
        Dimension S[1,M]    /*Matriz de SalarioEntrada*/
        Dimension T[N,M]    /*Matriz de TiempoEntrada*/
        Dimension P[N,N]    /*Matriz de Precedencia*/

    Definir
        Dimension X[N,M]    /*Matriz de Asignación*/
        Dimension a[N]      /*Tiempo inicial de la tarea i*/
        Dimension Y[N,N]    /*Matriz de ejecución de Tarea*/
        numerico C          /*Costo*/
        Dimension w[N]      /*Tiempo de trabajo por desarrollador*/
        Dimension o[M]      /*Tiempo libre de desarrollador*/
        numerico  U         /*Tiempo Máximo*/
        numerico  CMAX      /*Costo Máximo*/
        numerico  x
        numerico  i
        numerico  i2
        numerico  j
        numerico  k
        Modelo_Bonmin m     /*Modelo de optimización Bonmin*/
```

```
    /*Cálculo del Tiempo Máximo*/
    Para i = 1 Hasta N Hacer
        Para j = 1 Hasta M Hacer
            U = U + T[i,j]
        Fin Para
    Fin Para

    /*Costo maximo*/
    CMAX = U * L

    /*Restricción de Asignación de Tareas*/
    Para i = 1 Hasta N Hacer
        Para j = 1 Hasta M Hacer
            x = x + X[i,j]
        Fin Para
        restriccion(m, x = 1)
    Fin Para

    /*Restricción de Ejecución de Tareas*/
    Para i= 1 Hasta N Hacer
        Para i2 = 1 Hasta N Hacer
            Si i <> i2 Entonces
                restriccion(m, Y[i,i2] + Y[i2,i] = 1)
            Fin Si
        Fin Para
    Fin Para

    /*Restricción de Precedencia*/
    Para j = 1 Hasta M Hacer
        Para i = 1 Hasta J Hacer
            Para i2 = 1 Hasta N Hacer
                Si i <> i2 Entonces
                    restriccion(m, a[i2]>= (a[i] + T[i,j]) * X[i,j] * P[i,i2])
                Fin Si
        Fin Para
    Fin Para

    /*Restricción de tiempo*/
    Para i = 1 Hasta N Hacer
        Para j = 1 Hasta M Hacer
            restriccion(m, Z>= (a[i] + T[i,j]) * X[i,j])
        Fin Para
    Fin Para

    /*estriccion de tiempo inicial*/
    Para j = 1 Hasta M Hacer
        Para i = 1 Hasta J Hacer
            Para i2 = 1 Hasta N Hacer
                Si i <> i2 Entonces
                    restriccion(m, a[i2]>= (a[i] + T[i,j]) * X[i,j] -(U * (3-Y[i,i2] - X[i,j] - X[i2,j])))
                Fin Si
            Fin Para
        Fin Para
    Fin Para

    /*estriccion de Costo*/
    Para i = 1 Hasta N Hacer
        Para j = 1 Hasta M Hacer
            k = k + (T[i,j] * X[i,j])
        Fin Para
        restriccion(m, w[i] = k)
    Fin Para

    Para j = 1 Hasta M Hacer
        restriccion(m, C>= C + (S[1,j] * w[j]))
    Fin Para

    /*Función Objetivo de Optimización Tiempo y Costo*/
    NLObjetivo(m, Min, (alpha * C/CMAX) + ((1 - alpha)*Z/U))
    Resolver(m)
FinProceso
```

Pseudocódigo 1: Pseudocódigo del Modelo Matemático propuesto.

17. Metodología

17.1. Ambiente Computacional

Para las pruebas del algoritmo propuesto se tomaron dos proyectos reales, se identificaron los sprints de cada proyecto, y los desarrolladores disponibles para cada proyecto, con niveles de conocimiento y experiencia.

Las pruebas fueron realizadas en una computadora, con sistema operativo Linux Mint 64 bits, 2 GB de Memoria RAM. El modelo propuesto fue desarrollado en la plataforma Julia 6, entorno web Jupyter para Python. Para la optimización del tiempo y el costo se utiliza el modelo de procesamiento Bonmin (Basic Open-source Nonlinear Mixed INteger) a través de AmplNLWriter.jl.

Para todas las ejecuciones realizadas fueron definidos los siguientes parámetros:

- nlp log level: Especifica el nivel de detalle de log para el motor de soluciones Nonlineal Programming (NLP). Para todos los casos se ha definido el nivel 0, que corresponde al nivel más general de impresión de mensajes.
- time limit: Define el tiempo global máximo de ejecución (en segundos) del algoritmo. Para los casos se ha definido el tiempo máximo de ejecución de 3000 segundos.

Este trabajo continúa la investigación realiza por Mendez [Mendez 2018], donde se propone un modelo matemático de asignación de tareas con el fin de optimizar el tiempo de desarrollo. Se incorpora la optimización de tiempo y costo, en base a los recursos disponibles y se realiza un análisis de los mismos, con el fin de determinar la cantidad de desarrolladores con diferentes capacidades necesaria para lograr el costo y tiempo óptimo de desarrollo.

El enfoque es cuantitativo, debido a que se plantea un estudio delimitado en el problema de la asignación de Recursos en la Metodología SCRUM, básicamente se propone una hipótesis para resolver el problema, que es la efectividad y eficiencia del modelo matemático propuesto, se verifica mediante simulaciones numéricas.

El diseño de un enfoque cuantitativo es estructurado y predeterminado.

El tipo de Investigación es Experimental, debido a que se realizan pruebas del algoritmo sugerido con distintas variables y se valida la eficacia del mismo en base a los resultados obtenidos. [Hernández et al. 2006]

Hipótesis: El empleo de un algoritmo matemático posibilitará la asignación adecuada y efectiva de Desarrolladores considerando las capacidades y experiencias de los miembros del equipo de trabajo, para cada funcionalidad del Product Backlog, mejorando así los tiempos de entrega de desarrollo y reduciendo los costos del proyecto.

18. Pruebas y Resultados

Para continuar con las pruebas realizadas por Mendez[Mendez 2018], se toman los mismos casos reales de los dos proyectos de desarrollo de software donde se utilizó la metodología Scrum. Para este trabajo se considerará la optimización Tiempo/Costo, y la asignación óptima de desarrolladores a cada proyecto.

En la tabla 1, se visualizan las características de los dos proyectos. El equipo del Proyecto 1, está compuesto por 5 a 6 desarrolladores, con diferentes niveles de conocimientos (Junior, Semisenior y Senior) y por 5 Sprint. El equipo del Proyecto 2, está compuesto por 2 o más desarrolladores y por 4 Sprint para su finalización.

Tabla 1. Características de los Proyectos [Mendez 2018]

Sprint	Proyecto 1			Proyecto 2		
	Desarrolladores	Tareas	Dependencias	Desarrolladores	Tareas	Dependencias
1	5	11	8	2	7	5
2	5	8	3	2	4	3
3	5	6	2	2	4	3
4	5	6	2	3	3	3
5	6	7	3			

18.1. Resultados Computacionales

Para las pruebas realizadas se asignan valores diferentes al coeficiente α, se puede observar que existe una correlación de Tiempo, Asignación, Dependencia y Costo.

Los valores de la matriz de tiempo y costo por cada desarrollador para las pruebas son estimaciones, por lo que no necesariamente se ajustan a la realidad. El tiempo se basa en un promedio de datos históricos del equipo y el costo por hora es un promedio calculado de acuerdo a la capacidad del desarrollador.

En el Proyecto 1, se tenía mayor cantidad de tareas, con mayor tiempo de desarrollo y mayor cantidad de desarrolladores de diferentes niveles de conocimiento, por lo que en cada corrida se observa una variante de Costo/Tiempo que puede ser tenido en cuenta a la hora de elegir la asignación de desarrolladores que más nos conviene de acuerdo a la situación.

En las pruebas al considerar el valor de α a 1 (sin tener en cuenta el tiempo), el optimizador devolvía un valor mucho mayor en cuanto a costo, y en caso en que tenga valor igual a 0 (no se considera costo), el optimizador devolvía un valor mucho más grande en cuanto al tiempo. Por lo tanto, para la representación gráfica sólo se consideran coeficientes decimales, de esta forma se obliga al optimizador a considerar una pequeña porción de tiempo o costo.

Se pude observar una correlación negativa entre el Tiempo y Costo, lo que determina que a valores altos de una de ellas le suelen corresponder valores bajos de la otra y viceversa.

En la Fig. 2 del Sprint 3, Proyecto 1, se visualiza el tiempo y costo real que invirtió el equipo de desarrollo antes de considerar el modelo propuesto. En la grilla se visualizan los diferentes coeficientes utilizados y los resultados de Tiempo y Costo, en base a las asignaciones realizadas por el optimizador.

SPRINT 3

Tiempo Real: 50 Hr.

Costo Real: 4.686.000 Gs.

Etiqueta	Coeficiente α	Costo en miles Gs.	Tiempo Hr.	Tiempo Hr. sin utilizar
e	0,01	4.698	36	42
e	0,20	4.698	36	42
e	0,40	4.698	36	42
e	0,50	4.698	36	42
b	0,60	4.590	54	108
d	0,80	4.698	48	102
a	0,99	4.236	180	102

Correlación Costo/Tiempo: -0,99

Figura 2. Resultados Corrida Proyecto 1, Sprint 3.

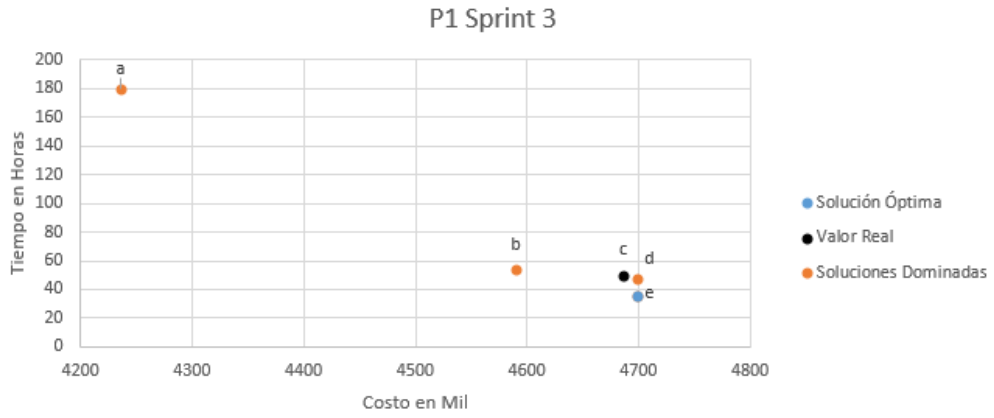


Figura 3. Corrida Proyecto 1, Sprint 3.

En la Fig. 3 se visualiza que las soluciones no dominadas o no comparables corresponden a:

- coeficiente 0,99 donde se mejora el costo, pero aumenta considerablemente el tiempo de desarrollo. Solución Factible en caso de que sólo se desee disminuir el costo sin importar el tiempo que conlleve el sprint. Véase etiqueta a.
- coeficiente 0,4 donde se reduce tanto el costo como el tiempo de desarrollo. Véase etiqueta e.
- coeficiente 0,5 donde se reduce tanto el costo como el tiempo de desarrollo. Véase etiqueta e.

El resto de las soluciones son dominadas (véase etiqueta b y d). Analizando la figura se puede concluir que la solución más óptima se da con las asignaciones, cuyo coeficiente fue 0,4. Debido a que se logra una reducción de tiempo y costo, en base a la función objetivo.

El punto negro (véase etiqueta c) representa el valor de tiempo real invertido en el desarrollo, se puede visualizar una disminución del costo y un aumento de tiempo de

desarrollo, en comparación con la solución óptima de la formulación propuesta.

18.2. Comparación de Resultados

En la Fig. 4 referente al Sprint 3 del Proyecto 1, se observa que en el resultado del estudio realizado en este trabajo de Optimización del tiempo y costo se produce una disminución de costo en relación a los trabajos de optimización de Tiempo realizados por Mendez [Mendez 2018], donde sólo se considera la optimización de tiempo. El tiempo óptimo coincide en ambos trabajos.

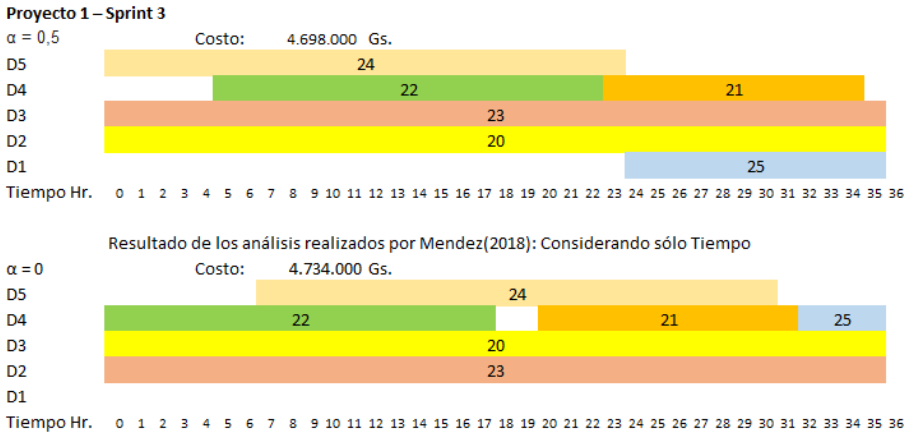
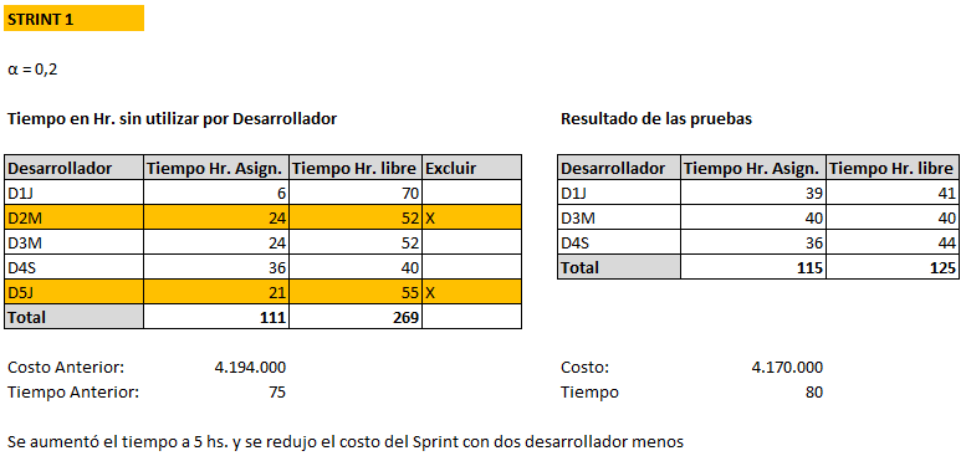


Figura 4. Corrida Proyecto 1, Sprint 3.

18.3. Mejora en la asignación de desarrolladores

Para mejorar la optimización del tiempo y costo, se realizaron nuevamente pruebas en base a los resultados obtenidos en las corridas anteriores. Se disminuyen n desarrolladores, con el fin de mejorar la asignación de desarrolladores disponibles.

Se selecciona para las pruebas del Sprint 1 del Proyecto 1, el resultado obtenido en caso de que el coeficiente sea 0,2. En la Fig. 5, se observa que excluyendo 2 desarrolladores, se logra minimizar el costo y un leve aumento del tiempo de desarrollo a 5 hs.



STRINT 1

α = 0,2

Tiempo en Hr. sin utilizar por Desarrollador

Desarrollador	Tiempo Hr. Asign.	Tiempo Hr. libre	Excluir
D1J	6	70	
D2M	24	52	X
D3M	24	52	
D4S	36	40	
D5J	21	55	X
Total	**111**	**269**	

Costo Anterior: 4.194.000
Tiempo Anterior: 75

Resultado de las pruebas

Desarrollador	Tiempo Hr. Asign.	Tiempo Hr. libre
D1J	39	41
D3M	40	40
D4S	36	44
Total	**115**	**125**

Costo: 4.170.000
Tiempo 80

Se aumentó el tiempo a 5 hs. y se redujo el costo del Sprint con dos desarrollador menos

Figura 5. Gestión de Asignación de Recursos Proyecto 1, Sprint 1.

En la Fig. 6 se observa la variación de tiempo y costo con la reducción de la cantidad de desarrollares para el Sprint 1. El punto negro (véase etiqueta a) representa el resultado obtenido después de la prueba, se visualiza un aumento de tiempo y una disminución de costo con relación a la asignación de desarrolladores anterior (véase etiqueta b).

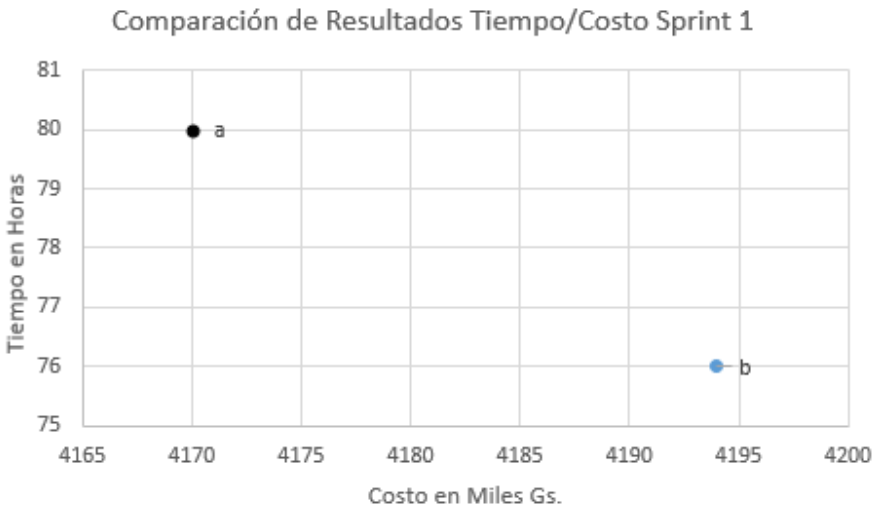


Figura 6. Comparación de Resultados de Tiempo/Costo Proyecto 1, Sprint 1.

19. Conclusión

Este trabajo propone una formulación matemática para optimizar el costo y el tiempo de cada Sprint de un Proyecto de desarrollo. Se realizaron pruebas de casos reales con diferentes coeficientes de relación de Tiempo y Costo.

Se demuestra que en proyectos basados en Scrum puede ser utilizado este modelo matemático para la asignación de Tareas, contribuyendo así el problema de asignación de tareas considerando un enfoque que genera soluciones óptimas.

Se realizó un análisis de la cantidad de desarrolladores disponible para cada Sprint considerando el tiempo libre de cada uno. En base a los resultados obtenidos de las corridas se disminuyó la cantidad de desarrolladores para mejorar la asignación de tareas, de esta manera se logró optimizar el tiempo y costo, y se liberaron recursos que podrían ser asignados a otros Sprints o Proyectos.

En función a los resultados obtenidos, y con el objetivo de futuras mejoras, se presentan las siguientes propuestas para dar continuidad al presente trabajo:

1. Realizar pruebas sobre otros conjuntos de datos con características diferentes.
2. Alternativa de contar con un recurso comodín o backup, en caso ausencia de alguno de los recursos para no afectar el tiempo comprometido al inicio del proyecto.
3. Realizar un estudio sobre la matriz de tiempo entre el desarrollador y la tarea correspondiente.
4. Agregar restricciones al modelo matemático propuesto, considerando por ejemplo la complejidad y la criticidad de la tarea.

Referencias

Al-Zewairi, M., Biltawi, M., Etaiwi, W., and Shaout, A. (2017). Agile software development methodologies: Survey of surveys. *Journal of Computer and Communications*.

Brassard, G. (1997). *Algoritmos voraces. Fundamentos de Algoritmia*. Madrid.

Canós, J., Penadés, M., and Letelier, P. Metodologías Ágiles en el desarrollo de software.

Cohn, M. (2005). Agile estimating and planning. *Prentice Hall, New Jersey, USA*.

Deemer, P., Benefield, G., Larman, C., and Voode, B. (2015). Información básica de scrum. *Scrum Training Institute*.

Figueroa, R., Solis, C., and Cabrera, A. (2008). Metodologías tradicionales vs metodologías Ágiles. *Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación*.

Franco, L. (2010). *Algoritmos de Optimización basados en Colonias de Hormigas aplicados al Problema de Asignación Cuadrática y otros problemas relacionados*. San Luis, Argentina.

Gonzalez, R. (2012). *Sistema de Información para el departamento de Acción Social (SISSA)*. Universidad Veracruzana, Veracruz.

Grenning, J. (2002). Planning poker or how to avoid analysis paralysis while release planning, hawthorn woods: Renaissance software consulting.

Hernández, M., Fernandez, R., and Batista, P. (2006). *Metodología de la Investigación*. México.

Herrera, E. and Valencia, L. (2007). Del manifiesto Ágil sus valores y principios. *Scientia et Technica Año XIII, No 34*.

Javdani, T., Tieng, W., and Binhamid, A. (2014). A case study research on software cost estimation using experts'. estimates, wideband delphi, and planning poker technique. *International Journal of Software Engineering and its Applications*.

Kniberg, H. and Skarin, M. (2010). *Bankan y Scrum - obteniendo lo mejor de ambos*. C4 MediaInc, EEUU.

Manaure, A. (2018). 71 % de las organizaciones usa metodologías Ágiles. `http://www.cioal.com/2018/01/09/71-las-organizaciones-usa-metodologias-agiles`.

Marczyk, A. (2004). Algoritmos genéticos y computación evolutiva. *Departamento de Informática, universidad de Colorado*.

Martinez, F. (2003). *Introducción a la programación estructurada en C. España*. Universidad de Valencia.

Mendez, E. (2018). *Modelo de optimización para la asignación de tareas en desarrollo ágil (SCRUM)*. Universidad Nacional de Asunción - Facultad Politécnica, San Lorenzo, Paraguay.

Murrada, H. and Pons, J. (2010). *Automatización de sistemas de desarrollo ágil-Scrum*.

Navarro, A., Fernandez, J., and Morales, J. (2013). Revisión de metodologías ágiles para el desarrollo de software. *Universidad Autónoma del Caribe, Colombia*.

Orjuela, A. and Rojas, C. (2008). Las metodologías de desarrollo Ágil como una oportunidad de ingenería del software educativo. *Universidad Nacional de Colombia*.

Palacio, J. (2014). *Gestión de proyectos Scrum Manager. (Scrum Manager I y II). (Versión 2.5)*. Scrum Manager®.

Perez, O. (2011). *Cuatro enfoques metodológicos para el desarrollo de SoftwareRUP – MSF – XP – SCRUM, Inventum No. 10*. Facultad de Ingeniería UNIMINUTO. Junio de 2011 64 , ISSN 1909 – 2520.

Póveda, J. and Florez, C. (2009). *Aplicación de algunos métodos exactos y heurísticos para resolver el problema de balanceo de linea simple*. Universidad Tecnológica de Pereira.

Radhakrishnan, P., Kanmani, S., and Nandhini, M. (2015). Xsoft: A generic software teaching and learning model. computer applications in engineering education.

Raj, G., Yavad, K., and Jaiswal, A. (2015). Emphasis on testing assimilation using cloud computing for improvised agile scrum framework. *International Conference on Futuristic Trends on Computational Analysis and Knowledge Management, New Delhi*.

Ries, E. (2011). *El método de Lean Startup*. Deusto S.A. Ediciones.

Rodriguez, C. and Dorado, R. (2015). ¿por qué implementar scrum? *ONTARE, Revista de Investigación de la facultad de Ingeniería*.

Rodriguez, E. (2015). Scrum aplicado. `https://www.youtube.com/watch?v=qRx8BkjY8lY`.

Santana, C., Queiroz, F., Vasconcelos, A., and Gusmao, C. (2015). *Agile, Web Engineering and Capability Maturity Model Integration: A Systematic Literature Review*.

Sriram, R. and Mathew, S. (2012). Global software development using agile methodologies: A review of literature. *IEEE International Conference on Management of Innovation and Technology, Bali*.

Takeuchi, H. and Nonaka, I. (1986). The new new product development game. *Harvard Business Review*.

Trigas, M. (2012). *Metodología Scrum (Bachelor's thesis, Universitat Oberta de Catalunya)*.

Tupia, M. and Mauricio, D. (2014). Un algoritmo voraz para resolver el problema de la programación de tareas dependientes de máquinas diferentes. *Universidad Nacional Mayor de San Carlos*.

Vancells, J. (2002). *Algoritmos y programas*. Editorial UOC.

Velez, M. and Montoya, J. (2007). Metaheurísticos: Una alternativa para la solución de problemas combinatorios en administración de operaciones. *Rev.EIA.Esc.Ing.Antioq no.8*.