RUP versus Scrum: una comparación empírica en un ámbito académico

Santiago D'Andre[1], Miguel Martinez Soler[1], Gabriela Robiolo[1]

[1] Universidad Austral, Av. Juan de Garay 125,
1063 Buenos Aires, Argentina
santiagodandre@yahoo.com.ar, miguelmsoler@gmail.com, grobiolo@austral.edu.ar

Abstract. Rup y Scrum son dos métodos de desarrollo adoptados por la industria del software. Es posible comprender sus diferencias, pero no siempre es factible cuantificarlas. Con esta finalidad se realizó un experimentó formal comparando el desarrollo de un mismo producto –juego de estrategias por turno- aplicando en forma paralela estos dos métodos en un contexto académico. Los resultados obtenidos no permiten afirmar taxativamente que un método es mejor que otro, sino que resaltan las características particulares de cada uno. El producto desarrollado con el método Rup implementa menos funcionalidad, es más pequeño y su diseño es de mejor calidad. Scrum implementa una mayor cantidad de funcionalidad. Su diseño es de menor calidad y el producto es más grande y complejo.

Keywords: métodos de desarrollo, RUP, Scrum, Agile.

1 Introducción

En la década de 1990 surgieron nuevos procesos y metodologías para encarar proyectos de desarrollo de software. Este conjunto de métodos de desarrollo a medida que evolucionaron se agruparon en dos tendencias: tradicionales y ágiles. Ambas tendencias tienen características particulares que son puntualizadas en los párrafos siguientes analizando los sistemas que desarrolla, el estilo de conducción, el tipo de organización, la forma de comunicación y el ciclo de vida.

Los métodos tradicionales tienen la particularidad de que los sistemas pueden ser especificados en forma completa, son predecibles y son construidos por medio de una planificación meticulosa y extensa. La conducción de los proyectos es realizado por una cabeza claramente definida que controla las actividades basándose en un conocimiento explícito. La organización se espera que sea grande, burocrática con un grado alto de formalización. La comunicación es formal. El ciclo de vida suele ser de cascada o en espiral, donde la prueba se realiza al final [1].

Los métodos ágiles están basados en la premisa de que el software -de alta calidad adaptable a diferentes condiciones- es desarrollado por pequeños grupos usando un

diseño con mejoras continuas. La prueba brinda un retorno de la información rápido posibilitando la temprana incorporación de las mejoras. El estilo de conducción de los proyectos es de liderazgo y colaboración. El conocimiento para la administración de los proyectos es tácito. La organización se espera que sea pequeña o mediana, flexible y participativa. La comunicación es informal. El ciclo de vida suele ser evolutivo, con un control de requerimientos, diseño y testing continuo [1].

Si bien las características mencionadas delinean a los métodos tradicionales y ágiles algunos autores piensan que a futuro los aspectos claves que van a definir su selección son: la cantidad de personas trabajando en un proyecto, el dominio de la aplicación, y el tipo de proyecto (misión crítica o innovadores) [2].

Entre los artículos que comparan ambos métodos, llama la atención la poca cantidad de artículos que desarrollan estudios empíricos. Esto ha motivado el planteo de un *experimento formal que compara el desarrollo de un producto de software aplicando un método tradicional con otro producto desarrollado con un método ágil*, en un contexto académico. Para este fin se seleccionaron al Rational Unified Process (RUP) [3, 4, 5] y Scrum [6, 7 ,8 ,9], debido a que ambos nacen en el ámbito de la industria del software.

El RUP es el resultado de la unificación de diferentes enfoques de desarrollo de software usando el UML y de la unificación de diferentes metodologías de desarrollo. Es un proceso iterativo, incremental, centrado en la arquitectura y basado en casos de uso. Es un proceso organizado en cuatro fases: inicio, elaboración, construcción, transición, y cinco procesos: captura de requerimientos, análisis, diseño, implementación y prueba. Provee un disciplinado enfoque para definir roles, actividades y entregables. Puede ser utilizado en organizaciones grandes o pequeñas, formales o informales, puede soportar diferentes estilos de conducción debido a que su planteo es flexible [4].

Scrum es un método ágil enfocado a la administración de proyectos. El desarrollo de software es realizado por un grupo en incrementos llamados “sprints”, comenzando con una planificación del “sprint” y finalizando con una evaluación. Se registran las características a ser implementadas y un dueño del proyecto decide cuáles son las características a implementar en cada “sprint”. Los miembros del grupo coordinan su trabajo en una reunión diaria de pie [10].

Si es necesario clasificar el RUP se lo identifica como un método tradicional, dado que su planteo soporta desarrollos grandes en organizaciones burocráticas y formales. Pero al mismo tiempo Hirsch plantea como hacer de Rational Unified Process (RUP) un método ágil [11]. Demuestra con dos ejemplos cómo es posible, aprovechando la flexibilidad de RUP, aplicarlo a proyectos pequeños. Lo logra realizando una cuidadosa selección de entregables apropiados, que se desarrollan en forma concisa y despojando el proceso de todo formalismo innecesario. De la misma forma se lo ha flexibilizado en la experiencia desarrollada, por lo que se concluye que más que la comparación entre un método tradicional y uno ágil se logra la comparación entre procesos de desarrollo diferentes.

Es posible comprender las diferencias que existen entre estos dos procesos de desarrollo, pero resulta difícil cuantificar estas diferencias. Con este motivo se planteó un experimento formal para medir empíricamente estas diferencias. El foco de la comparación se puso en la funcionalidad desarrollada, la calidad del diseño obtenido,

el grado de comprensión de la aplicación y algunas características de la implementación de la arquitectura.

Con este fin, se desarrollaron las siguientes preguntas de investigación:

a. ¿Es el tamaño funcional de la aplicación desarrollada con Scrum mayor que el desarrollado aplicando RUP?

b. ¿Es el diseño del producto aplicando RUP mejor que el obtenido aplicando Scrum?

c. ¿Es el grado de comprensión del diseño aplicando RUP mayor que el logrado aplicando Scrum?

d. ¿Existen diferencias significativas entre la implementación de la arquitectura desarrollada según el método RUP y Scrum?

Para responder estas preguntas se organiza el artículo de la siguiente forma: el punto 2 describe el experimento formal, analiza los resultados y las amenazas de validez. El punto 3 comenta los trabajos relacionados, el 4 detalla las conclusiones y el 5 los trabajos futuros.

2 Experimento formal

Se definió un experimento formal que se desarrolló en el contexto de un taller de diseño que forma parte de la currícula de la carrera de Ingeniería Informática de la Facultad de Ingeniería de la Universidad Austral. En dicho taller, de tres horas semanales de duración, los alumnos de cuarto año desarrollan un producto de software, aplicando los conocimientos adquiridos en materias anteriores, partiendo de una idea inicial y finalizando con un prototipo.

A diferencia de los años anteriores, en los cuales los distintos grupos desarrollaban sus productos aplicando RUP, se dividió a los alumnos en dos grupos: RUP y Scrum. Ambos grupos trabajaron en forma independiente en el desarrollo de un juego de estrategia por turnos partiendo de una misma definición de requerimientos.

Para la conformación de los grupos se plantearon dos alternativas: la selección al azar de los integrantes o la conformación de dos grupos con capacidades similares. Dada las características del grupo de alumnos que se muestran en la Tabla 1, donde se evidencia un número impar con niveles muy distintos, se decidió la formación de dos grupos equilibrados. Para la distribución de las personas en uno u otro grupo se consideraron los siguientes parámetros:

Rendimiento académico: considerando la nota de cursada de la materia correlativa anterior.

Experiencia laboral: medida en cantidad de meses que han trabajado en la industria del software o en laboratorios de desarrollo.

Carga académica: evidenciada por la cantidad de materias en curso y la cantidad de finales adeudados por cada alumno.

De esta forma (ver Tabla 1) se logró definir dos grupos (RUP y Scrum) con capacidades similares, con la intención de evitar la distorsión de los resultados.

Tabla 1. Descripción de las capacidades de los integrantes de RUP y Scrum

Miembro del grupo	Nota	Meses trabajados	Finales adeudados	Materias en curso
RUP_1 (*)	9.5	3	1	7
RUP_2	4	12	5	6
RUP_3	9	4	1	7
$Scrum_1$	7	5	6	8
$Scrum_2$	6.5	0	5	7
$Scrum_3$ (*)	7.5	5	7	5
$Scrum_4$	8.5	12	4	6

(*) Alumno con experiencia previa en el desarrollo de videojuegos

También fue necesario controlar las siguientes variables independientes:

Ambiente de desarrollo: ambos grupos trabajaron en el mismo ambiente de desarrollo y utilizaron computadoras con iguales características técnicas.

Tiempo: ambos grupos solo trabajaron en clase, el tiempo establecido. Se controló que los dos grupos trabajaran la misma cantidad de horas hombre por grupo.

Nivel de capacitación: todos los alumnos tienen un avance similar en la carrera, por lo que se concluye que tienen un nivel similar de capacitación. Además, todos los alumnos recibieron una misma capacitación inicial en el tema específico de videojuegos y los dos alumnos que tenían experiencia previa en este tema fueron distribuidos en grupos diferentes.

Complejidad del producto: se definió un conjunto de reglas específico para el juego con el fin de evitar desarrollos dispares.

Herramientas: ambos grupos utilizaron el mismo lenguaje (java) y entorno de programación (Eclipse).

Además, la Figura 1 muestra los diferentes roles involucrados en cada grupo. En el grupo Scrum, dos profesores cumplieron respectivamente, el rol de Scrum Master y Product Owner. En el grupo RUP, un profesor cumplió el rol de Team Leader y otro de Product Owner. Los alumnos cumplieron simultaneamente los roles de usuario y equipo de desarrollo.

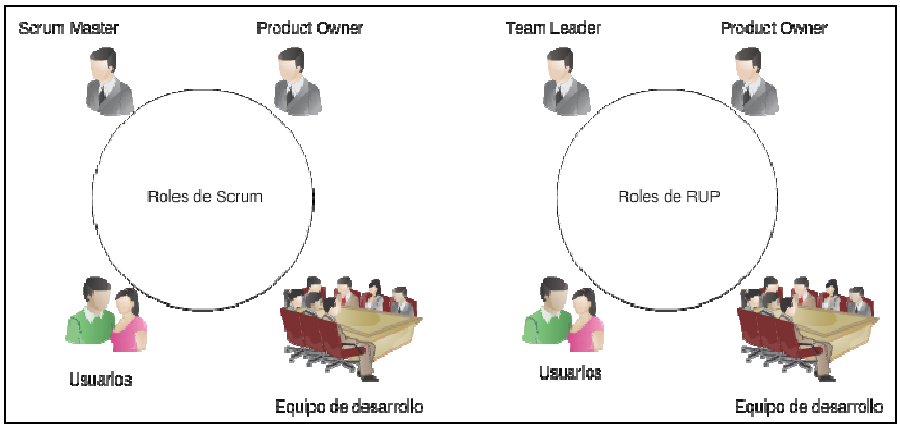

Fig. 1. Roles en Scrum y RUP

La Figura 2 muestra como trabajaron los dos grupos en forma paralela. El grupo RUP terminó el proyecto una semana antes que el grupo Scrum. El grupo Scrum, dividió el tiempo de desarrollo en cinco sprints, de cuatro a cinco clases de duración cada uno, sin embargo, al final del proceso de desarrollo solo llegó a realizar cuatro iteraciones. El grupo RUP realizó una única iteración, con diferentes hitos marcados por los entregables[1].

[1] Traducción al castellano del término artifacts.

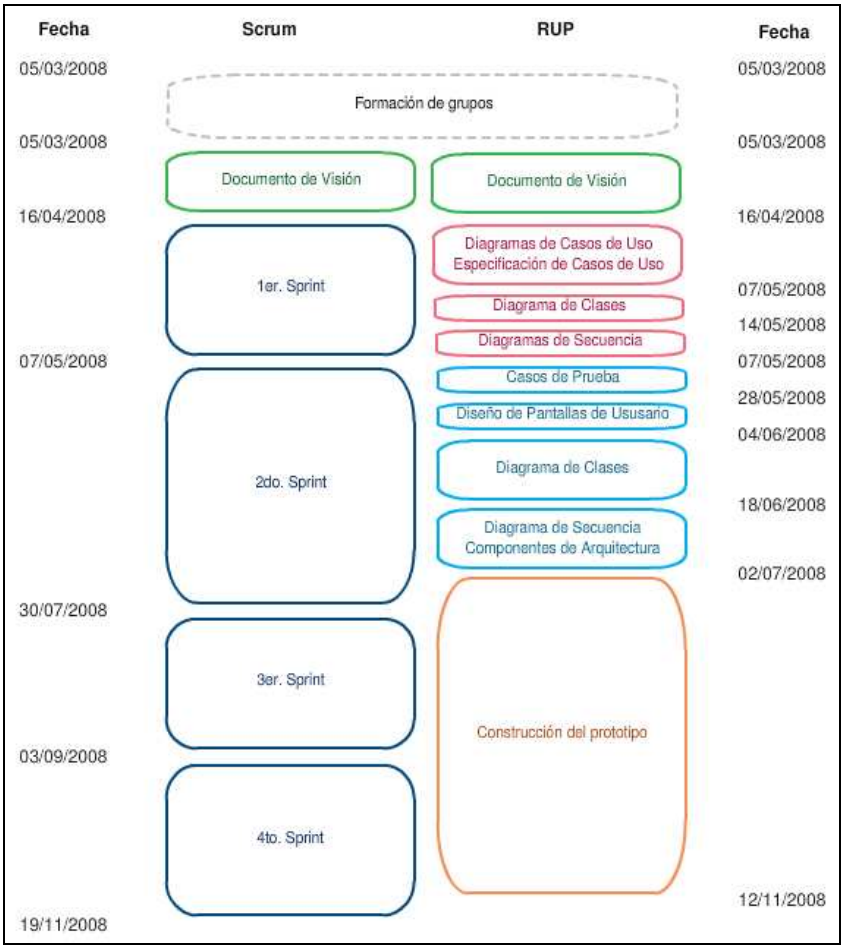

Fig. 2. Comparación de las actividades de Scrum y RUP

El Scrum Master -antes del inicio de cada clase- lideró una reunión en la que se analizó el progreso alcanzado en la iteración y asignó las tareas del día. En esta reunión cada integrante del grupo respondía a cada una de las siguientes preguntas:

a. ¿Qué se hizo desde la clase anterior?
b. ¿Qué se planea hacer hoy?
c. ¿Hubo algún problema en la clase anterior que haya impedido alcanzar el objetivo?

Al comienzo de cada sprint, se realizó una reunión en la que estuvieron presentes todas las personas involucradas en el proyecto y se definieron las tareas que serían realizadas durante esa iteración.

Al finalizar el tiempo de desarrollo se realizó una evaluación final. Con el objeto de medir las diferencias encontradas entre los productos elaborados por ambos

grupos, se seleccionaron los siguientes atributos: calidad de diseño, grado de compresión del diseño, tamaño funcional y características de la implementación de la arquitectura. La Tabla 2 muestra las métricas usadas para la medición de cada atributo.

Tamaño funcional: para medir la cantidad de funciones implementadas se utilizó el concepto de transacción. Cada transacción fue identifica a partir del estímulo disparado por el actor hacia el sistema [14]. El tamaño funcional se calculó como la cantidad de estímulos.

Tabla 2. Detalle de los atributos con sus respectivas métricas

Atributo	Métricas	Comentario
Tamaño funcional	Transacciones (T)	Cantidad de estímulos disparados por el actor hacia el sistema.
Calidad de diseño	Cantidad de clases afectadas por cambios.	No se tuvieron en cuenta clases anidadas y anónimas implementadas en Java.
Grado de compresión del diseño	Cantidad de respuestas claras, confusas, erróneas	Ver preguntas Tabla 4.
Diferencias significativas en la implementación de la arquitectura	Métodos ponderados por clase Número de hijos por clase Acoplamiento entre clases Número de métodos públicos por clase	Métricas de Chidamber-Keremer

Calidad de diseño*:* Una vez finalizado el tiempo de desarrollo a cada grupo se les propusieron los siguientes cambios:

a. Ubicar más de una tropa en un casillero.
b. Asignar múltiples mejoras a las tropas.
c. Crear una nueva unidad especial que tenga la capacidad de construir asentamientos.
d. Incorporar múltiples condiciones de fin de juego.
e. Insertar una etapa de compra y venta de recursos entre turnos.

Los alumnos analizaron los cambios y contaron las clases que serían afectadas por estos cambios.

Grado de comprensión del diseño del producto: para evaluar el grado de comprensión alcanzado por los alumnos sobre el diseño implementado, se realizo una evaluación sobre los aspectos más relevantes del mismo. Los resultados de la misma se volcaron en planillas de resultados, una por cada alumno, indicando la metodología

utilizada y clasificando las respuestas a cada pregunta en claras, confusas y erróneas. La Tabla 4 muestra el detalle de las preguntas realizadas.

Características de Implementación de la Arquitectura: para estudiar los efectos de la metodología en el código implementado se midieron un subconjunto de métricas Chidamber-Kemerer [12] utilizando la herramienta ckjm [13].

Tabla 4. Preguntas usadas para determinar el grado de compresión de los diseños

Pregunta
¿Cómo se implementa la toma de recursos?
¿Cómo se implementa la toma de fuentes de recursos?
¿Cómo se implementa el movimiento de tropas?
¿Cómo se implementa la finalización del turno?
¿Cómo se implementa la combinación de tropas?
¿Cómo se implementa el ataque?
¿Cómo se implementa la construcción de una tropa?
¿Cómo se implementa la mejora de una tropa?
¿Cómo se implementa la exploración de lugares misteriosos?
¿Cómo se implementa la mejora de un asentamiento?
¿Cómo se implementa la creación de un nuevo juego?
¿Cómo se implementa la finalización de un juego?
¿Cómo se implementa la actualización del juego en red?
¿Cómo se implementa la visualización y el renderizado del mapa?
¿Cómo se implementa el control general del juego?

2.1 Resultados de la evaluación final

Se describen a continuación los valores de las mediciones realizadas para cada aplicación de juego de estrategia por turnos.

Tamaño Funcional. La Tabla 5 muestra el tamaño funcional de cada producto medido en T. Se observa que la funcionalidad implementada por ambos grupos es similar. Sobre las 15 transacciones planteadas para el experimento, se observa que el grupo Scrum implementó dos transacciones más que el grupo RUP.

Tabla 5. Tamaño funcional de cada producto

Métrica	RUP	Scrum
T	8	10

Calidad del Diseño. La Tabla 6 muestra la cantidad clases afectadas por los cambios detallados en el punto anterior. Se observa que los cambios propuestos afectan en mayor medida al diseño producido por el grupo de Scrum, el cual para introducir los mismos cambios que el grupo RUP modifica más del doble de clases.

Tabla 6. Cantidad de clases afectadas por los cambios

Métrica	RUP	Scrum
Cantidad de clases afectadas por los cambios	8	17

Grado de compresión del diseño del producto. En la Tabla 7 se muestra la tabulación de las respuestas de las preguntas detalladas en la Tabla 4, ordenados de menor a mayor. Las respuestas se las clasificó en clara, confusa y errónea, y se le asignó el siguiente peso: 3, 1 y 0.

Tabla 7. Respuestas usadas para determinar el grado de compresión de los diseños

Alumno	Respuestas			Puntaje Total
	Claras	Confusas	Erroneas	
$SCRUM_2$	3	5	7	**14**
RUP_2	6	4	5	**22**
$SCRUM_4$	7	2	6	**23**
$SCRUM_3$	6	9	0	**27**
$SCRUM_1$	12	3	0	**39**
RUP_3	14	0	1	**42**
RUP_1	15	0	0	**45**

Con la finalidad de comprobar desde un punto de vista estadístico las diferencias en el grado de comprensión del diseño entre los dos grupos, se realizó una prueba de rangos de Mann-Whitney. La hipótesis nula planteada fue: no hay diferencias en el grado de compresión del diseño del producto. No fue posible rechazar con criterio estadístico esta hipótesis puesto que el nivel de significación obtenido fue 0,289. El valor del estadístico U de Mann-Whitney fue de 3.

Diferencias significativas en la implementación de la arquitectura. La Tabla 8 muestra los valores obtenidos al medir cada uno de los productos usando la herramienta ckjm [13]. Llama la atención la diferencia en la cantidad de clases implementadas en cada producto: Scrum implementa un número mucho mayor de clases. También los valores de los Métodos Ponderados por Clase y Número de Métodos Públicos por Clase evidencian diferencias significativas.

Tabla 8. Métricas que miden las diferencias significativas de la arquitectura

Métrica	RUP	Scrum
Cantidad de Clases	47	86
Métodos ponderados por clase	8.09	4.39
Número de hijos por clase	0.26	0.31
Acoplamiento entre clases	3.66	3.77
Número de métodos públicos por clase	7.32	3.59

2.2 Discusión de los resultados

Si bien el trabajo no marca una tendencia clara sobre cuál es el mejor método de desarrollo, los resultados obtenidos sugieren que ambos tienen sus fortalezas y su debilidades. El grupo de desarrollo de RUP produjo un diseño superior pero que a la vez implementa algo menos de funcionalidad en un intervalo igual de tiempo.

El grupo de Scrum produjo código antes que el grupo de RUP, pero la demora redundó en un diseño de menor calidad. Esta inferioridad del diseño puede deberse a que no se hizo un re-factoring del código, que también insume su tiempo.

Sin embargo la diferencia en funcionalidad (20%) se diluye si se considera que por las características del método RUP el tiempo de programación fue un 40% menor que el de Scrum, puesto que RUP dedicó bastante tiempo al diseño de la arquitectura. Se infiere que si los grupos hubieran tenido más tiempo el grupo de RUP posiblemente hubiera igualado o superado la cantidad de funcionalidad desarrollada por Scrum. Esto evidencia que el grupo Scrum re-programó alguna tarea.

En cuanto la comprensión del diseño por parte de los dos grupos no se probó con rigor estadístico que una fuera mayor que la otra. Se entiende que este resultado se debe al tamaño de la muestra, que la hace muy sensible a pequeñas variaciones.

Con respecto a las diferencias significativas en la implementación de la arquitectura, llama la atención la diferente cantidad de clases y métodos por clase. Esta puede deberse a diferencias de estilo de diseño (centralizado vs. descentralizado) o a un diseño menos trabajado como lo que aparenta en el caso de Scrum, donde una nueva funcionalidad debe haberse concretado en nuevas clases, tornando el diseño cada vez más complejo.

Al mismo tiempo, al realizar la medición se observó, que los integrantes del grupo RUP, conocían con bastante profundidad la implementación de las clases en las que no habían tenido ninguna participación, mientras que los integrantes del grupo Scrum evidenciaron un menor conocimiento de las clases que no habían desarrollado

2.3 Amenazas de validez

Una de las mayores preocupaciones al hacer el diseño del experimento fue que los productos obtenidos fueran comparables. Fue necesario escribir un conjunto de reglas específico para el juego con el fin de evitar desarrollos dispares. También, se limitó el tiempo dedicado a la interfaz gráfica a un nivel mínimo, necesario para la comprensión del producto. Se obtuvieron dos productos con un desarrollo de la interfaz gráfica similar.

En cuanto a la conformación de los grupos si bien se definieron los grupos de la forma más equilibrada posible se detectó que existen factores personales que son muy difíciles de controlar: la individualidad de cada persona en determinadas situaciones, lleva a los demás integrantes a potenciar alguna característica y podría provocar que el grupo entero cambie su comportamiento. A pesar de esta limitación propia de trabajar con personas, se entiende que los grupos tuvieron comportamientos distintos pero comparables.

Para guardar el equilibrio entre los grupos también se distribuyó las personas que tuvieran una experiencia previa en desarrollo de videojuegos. Pero en el transcurso del experimento se observó que afecto más al rendimiento de cada estudiante la experiencia previa en el uso de video juegos. Aspecto que no fue considerado en la preparación de la experiencia puesto que se supuso que todos tenían la misma experiencia.

En el diseño original del experimento, habían sido incluidas otros atributos a medir: Diseño Externo y Nivel de Calidad de la programación, que fueron descartados. El primero se descartó porque las limitaciones impuestas para mantener la complejidad de los productos a un nivel comparable incluían la indicación de concentrarse en la funcionalidad de la aplicación y no en la interfaz gráfica. Su comparación no habría arrojado ninguna conclusión. El segundo se midió contando la cantidad de fallas. Al final de la experiencia se comprobó que los productos estaban desarrollados a un nivel que no reportaban fallas, puesto que previa a la entrega, se les habían realizado pruebas y corregido los errores.

Es posible plantearse si los atributos seleccionados son los adecuados para el objetivo del experimento. Estos fueron seleccionados priorizando la objetividad y factibilidad de medición, centrándose en las características internas y externas de los productos obtenidos.

Si bien Scrum realizó una definición y priorización de las tareas, se observó que tenía una mayor dificultad para comprender la tarea que debía realizar ocasionando demoras o re-programación. Por el contrario el grupo de RUP, apoyándose en los entregables que elaboraron –casos de uso, diagramas de secuencia y diagramas de clase- realizó el trabajo con menor dificultad y pérdida de tiempo en la integración de las partes.

3 Trabajos Relacionados

En una búsqueda bibliográfica realizada con el buscador “google académico”, con las palabras Rational Unified Software y RUP, se han encontrado pocos estudios [15, 16, 17] empíricos en los sitios de IEEE, ACM, Springer, Science Direct o Elsevier, en comparación con la adopción de este método en la industria del software.

Hesse [15] individualiza algunos problemas particulares del RUP como ser: las fases siguen dominando el proceso y la estructura de iteración; el concepto de arquitectura no esta claramente definido y es todavía desestimado; las disciplinas son conceptos redundantes que complican el proceso más que soportarlo y no se les presta la atención debida a principios poderosos y transparentes como ser recursión y ortogonalidad. Como alternativa se contrasta un modelo desarrollado por el autor E*v*olutionary, Object-oriented *S*oftware development (EOS) con RUP.

Pilemalm et al. [16] presentan el MOPT-Systems Development Process, que integra una modificación del RUP con una vista del sistema desde una perspectiva social-técnica y una extensión del diseño participativo. La salida esperada es un producto sólido que se adapta a las necesidades de la organización. Los tres primeros “workflows” del MOPT-Systems Development Process han sido aplicados en un

proyecto que desarrolla un sistema de control y comando de un batallón de helicópteros.

De Nicola et al. [17] presentan una metodología llamada UPON para la construcción de ontologías derivadas del Unified Software Development Process. Realiza una comparación con otras metodologías y muestra los resultados de su adopción en un contexto particular.

Un estudio pormenorizado [2] que llega hasta el año 2005 reporta que existen pocos artículos que realizan un análisis empírico de los métodos ágiles. En este estudio, se identifican 13 artículos que comparan métodos de desarrollo. Cinco de ellos son referidos a eXtreme Programming (XP). De los 36 estudios empíricos identificados sólo uno era sobre Scrum [18], con una experiencia inicial en métodos ágiles. Este, es un estudio sobre el impacto de Scrum en el tiempo y el influjo sobre la satisfacción del cliente. El cliente afirma que las reuniones diarias lo ayudaron a involucrarse con el proyecto y a reducir la confusión sobre lo que se debe desarrollar. Además resalta que el cliente debe ser entrenado en el proceso de Scrum logrando que pueda comprender las nuevas expectativas que los desarrolladores tienen con respecto a su rol.

Los autores concluyen que es necesario realizar una mayor cantidad y calidad de estudios empíricos sobre desarrollos ágiles y que a los métodos ágiles orientados a la administración de proyectos se les han prestado muy poca atención si se tiene en cuenta que tienen una amplia adopción en la industria del software [2].

Posterior al año 2005 no se han encontrado artículos que realicen un análisis empírico de los métodos ágiles. Sólo se han encontrado uno que publica la opinión de expertos y otra lecciones aprendidas con los métodos ágiles. Chow y Cao realizan un estudio sobre los factores críticos de éxito de los proyectos desarrollados con métodos ágiles [19]. Concluyen que estos factores son: la estrategia de entrega, técnicas de ingeniería y capacidad del grupo. Nord y Tomayko resaltan la importancia de incluir diseño y análisis centrado en la arquitectura en XP puesto que logra alcanzar atributos de calidad en una forma explícita, metódica e ingenieril [20].

4 Conclusiones finales

Se realizó un experimentó formal con la finalidad de comparar el desarrollo de un mismo producto –juego de estrategias por turno- aplicando dos métodos de desarrollo Rup y Scrum, en un contexto académico.

Los resultados obtenidos no permiten afirmar taxativamente que un método es mejor que otro, sino que resaltan las características particulares de cada uno. Rup implementa menos funcionalidad, es un producto más pequeño, su diseño es de mejor calidad. Scrum implementa una mayor cantidad de funcionalidad, pero esta cantidad no es proporcional al tiempo dedicado a la programación. Su diseño es de menor calidad y el producto es más grande y complejo.

Como es frecuente en la ingeniería de software empírica no fue posible seleccionar al azar los productos a desarrollar. Además, la poca cantidad de alumnos que participaron en el experimento y la cantidad de las aplicaciones desarrolladas no

permiten generalizar los resultados. Estas limitaciones no invalidan los resultados obtenidos sino que muestran la necesidad de duplicar el experimento.

5 Trabajo futuro

El experimento está documentado por lo que es posible duplicarlo [21]. Sería conveniente duplicarlo incorporando otros métodos ágiles y dominios de desarrollo.

Agradecimientos. Se agradece a la Facultad de Ingeniería de la Universidad que hizo posible el desarrollo de este artículo.

6 Referencias

1. S. Nerur, R. Mahapatra, G. Mangalaraj: Challenges of Migrating to Agile Methodologies. Communications of the ACM May, 72—78 (2005)
2. T. Dyba˚, T. Dingsøyr: Empirical Studies of Agile Software Development: A Systematic Review, Inform. Softw. Technol (2008)
3. IBM Rational Unified Process, http://www-01.ibm.com/software/awdtools/rup/.
4. Jacobson, I., Grady, B., Rumbaugh, J: The Unified Software Development Process, Addison Wesley (1999)
5. Kruchten, P.: The Rational Unified Process: An Introduction, 3rd edition. Addison Wesley (2003)
6. Larman, C.: Agile and Iterative Development: A Manager's guide. Addison Wesley (2003)
7. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press (2004)
8. Schwaber, K. and Beedle, M.: Agile Software Development with Scrum. Prentice Hall (2002)
9. ScrumAlliance, http://www.scrumalliance.org
10. Vitulo, S.: Adecuación de Scrum a proyectos llevados a cabo en Laboratorio III y Dirección de Proyectos. Trabajo de fin de carrera, Facultad de Ingeniería, Universidad Austral (2008)
11. Hirsch**,** M**.** Making RUP Agile. OOPSLA 2002 Practitioners Report. ACM Press. (2002)
12. Henderson-Sellers, B.: Object-Oriented Metrics. Measures of Complexity. Prentice Hall (1996)
13. Spinellis, D.: Tool writing: A forgotten art? IEEE Software*,* 22(4):9 11, July/August (2005)
14. Robiolo, and Orosco, R.: Employing use cases to early estimate effort with simpler metrics. Innovations in Systems and Software Engineering, Volume 4, Number 1, April 2008 , pp. 31-43(13), Springer (2008)
15. Hesse, W.: Dinosaur Meets Archaeopteryx? or: Is there an Alternative for Rational's Unified Process?. Software and Systems Modeling (SoSyM), Expert's Voice, Vol. 2. No. 4, pp. 240-247 (2003)
16. Pilemalm, S., Lindell, P., Hallberg, N. and Eriksson, H.: Integrating The Rational Unified Process And Participatory Design for Development of Socio-Technical Systems: a User Participative Approach. Elsevier, Design Studies 28 (2007) 263e288 (2007)
17. De Nicola, A., Missikoff**,** M., Navigli, R.: **A** proposal for a Unified Process for Ontology building: UPON. Lectures Notes in Computer Science, Vol. 3588, 655—664(2005)
18. Mann, C., Maurer, F.: A case Study on the Impact of Scrum on Overtime end Customer Satisfaction. In: Agile Development Conference (2005)

19. Chow, T., Cao D.: A survey study of critical success factors in agile software projects. The Journal of Systems and Software 81, 961—971 (2008)
20. Nord, R., Tomayko J.E.: Software Architecture-Centric Methods and Agile Development. IEEE Software**,** Volume 23 , Issue 2, 47—53 (2006)
21. d' André, S.: Diseño, seguimiento y análisis de un experimento controlado para comparar dos procesos de desarrollo de software: Rational Unified Process y Scrum. Trabajo de fin de carrera, Facultad de Ingeniería, Universidad Austral (2008)