Supporting Assessment of Practices in Software Engineering Courses

G. Rodríguez, Á. Soria and M. Campo

***Abstract*— The main goal of the software engineering courses is to prepare students to face typical problems that occur in professional contexts. Thus, students need to be provided with tools and resources to be trained correctly, and therefore, courses need to be reoriented towards software industry demands without neglecting the academic quality. In this context, Scrum, which is widely used for manufacturing software, arises as a suitable teaching strategy to introduce students the main software engineering practices usually performed by the industry. This work proposes a teaching model based on Scrum and reinforced with agile coaching. This novel model has been compared to RUP (Rational Unified Process) and validated using the framework CMMI (Capability Maturity Model Integration) to measure the coverage of practices achieved by our model. The results show that our agile teaching model allows students to achieve higher coverage of software engineering practices, and maximize their learning experience.**

***Keywords*— Software engineering education, CMMI, Scrum, teaching strategies.**

I. INTRODUCCIÓN

LA INDUSTRIA del software ha crecido rápidamente y está demandando ingenieros de software con las suficientes habilidades para entrar en un contexto desafiante de desarrollo de proyectos de software cada vez más competitivo, con cambios constantes en los requerimientos de los sistemas, y con movilidad frecuente de los desarrolladores. En efecto, los estudiantes, que están por insertarse en al ámbito laboral, deben ser capaces de defenderse en el campo del desarrollo de software, y para esto es crucial que aprendan buenas prácticas de ingeniería de software.

Como consecuencia, los profesores tienen que diseñar cursos de ingeniería de software en los cuales incluyan varios aspectos presentes en grandes proyectos de software que enseñen a los estudiantes cómo enfrentar actuales amenazas al éxito de un producto [1]. En este sentido, estructuramos un curso de ingeniería de software siguiendo el modelo CMMI para el desarrollo versión 1.3 (CMMI-DEV 1.3)(CMMI for Development version 1.3. Technical Report. Software Engineering Institute, Carnegie Mellon. November, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm). CMMI es un marco de referencia que cubre un conjunto de prácticas para implementar procesos maduros y de alta calidad para el desarrollo de software [12]. Nuestra implementación inicial de CMMI consistió en seguir las guías de RUP (Rational Unified Process) [11] para dar soporte a los procesos de proyectos de software.

Sin embargo, este enfoque basado en RUP sufrió varias complicaciones. Al tratarse de un marco de trabajo focalizado en planes de ante mano, RUP requiere que los estudiantes especifiquen fechas de entrega estrictas al inicio del proyecto. Esto lleva a los estudiantes a focalizarse en alcanzar dicha fecha con el producto acordado, descuidando la rigurosidad del seguimiento de las actividades propuestas en RUP. A este problema se le suma que la planificación del proyecto no es realizada por todos los estudiantes con lo que se torna compleja la obtención del compromiso del resto de los estudiantes. Por otro lado, aunque RUP promueve el solapamiento de fases, es inevitable que los estudiantes caigan un proceso de tipo cascada [19]. Esto sucede porque la falta de experiencia de los estudiantes hace que no sean capaces de detectar errores en las etapas tempranas del desarrollo y sean detectados recién al final del proceso, donde ya corregir el curso de acción es mucho más costoso.

En este contexto, los métodos ágiles (MA) aparecen como una alternativa mucho más viable que RUP para enfrentar estos problemas en un entorno académico. Los MA promueven un modelo de ciclo de vida iterativo e incremental con el objetivo de producir software de alta calidad, permitiendo una rápida adaptación a la volatilidad de requerimientos [8]. En el contexto educativo, los MA han demostrado ofrecer un escenario agradable para introducir a los estudiantes en el desarrollo de software [10, 15]. Sin embargo, para enseñar las prácticas y metodologías de trabajo con los estándares de calidad requeridos profesionalmente hay que tomar como referencia un modelo que guíe y oriente a los estudiantes y profesores en el proceso educativo.

En este trabajo presentamos un modelo de enseñanza basado en prácticas ágiles y usando como referencia el modelo CMMI. El objetivo de este modelo es maximizar las fortalezas de la disciplina y la agilidad para mejorar la enseñanza de buenas prácticas de ingeniería de software. Para implementar MA en un contexto académico, consideramos dos aspectos principales: el proceso ágil y el *coach* ágil. Para el soporte del proceso ágil se eligió *Scrum,* mientras que el rol del *coach* ágil se asignó al profesor, el cual es responsable de guiar y monitorear las actividades de los estudiantes. Con el objetivo de evaluar nuestro modelo, lo implementamos en un curso de IS en el período académico 2009/10. Para medir su efectividad, analizamos el impacto del desempeño de los estudiantes en la cobertura de las prácticas de CMMI utilizando nuestro modelo ágil y contrastamos los resultados con nuestra experiencia anterior con RUP en el año 2008. Los resultados obtenidos han mostrado que un balance entre

G. Rodríguez, ISISTAN Research Institute (CONICET - UNICEN), Tandil, Buenos Aires, Argentina, guillermo.rodriguez@isistan.unicen.edu.ar.
Á. Soria, ISISTAN Research Institute (CONICET - UNICEN), Tandil, Buenos Aires, Argentina, alvaro.soria@isistan.unicen.edu.ar.
M. Campo, ISISTAN Research Institute (CONICET - UNICEN), Tandil, Buenos Aires, Argentina, marcelo.campo@isistan.unicen.edu.ar.

Scrum, el rol del *coach* ágil y CMMI es más efectivo para los estudiantes, que alcanzan a cubrir más prácticas de CMMI que con la implementación de RUP.

El resto del trabajo se organiza de la siguiente manera. La sección 2 presenta nuestro enfoque. La sección 3 reporta los casos de estudios y resultados obtenidos. La sección 4 describe los trabajos relacionados en el área. Finalmente, la sección 5 concluye nuestro trabajo e identifica futuras líneas de trabajo.

II. ENFOQUE ÁGIL PARA LA ENSEÑANZA DE IS

El desarrollo ágil de software está pisando cada vez más en la industria del software acaparando una gran variedad de proyectos, ya que se observó que con los métodos ágiles es posible obtener niveles de madurez con menos costos y esfuerzo [2, 8]. En particular, *Scrum* es el método ágil más utilizado en el mercado ya que está especialmente enfocado en el manejo de proyectos. *Scrum* es un método ágil que organiza proyectos en equipos pequeños, auto-organizados y multi-funcionales, llamados *Scrum Teams* [24]. En el contexto educativo, *Scrum* permite a los estudiantes tener un mejor ambiente de trabajo y una mejor comunicación que resulta en productos de alta calidad [6]. Sin embargo, los profesores deben contar una guía de referencia para organizar y desarrollar el curso. Esa guía viene de la mano de CMMI, que provee el marco de referencia adecuado para los profesores que permite implementar diferentes estrategias en el aprendizaje de los estándares de madurez y calidad. CMMI es un modelo estándar que consta de buenas prácticas que permiten llevar adelante el desarrollo y mantenimiento de productos y servicios [12]. CMMI hace referencia a "qué hacer" más que a "cómo hacerlo". Junto con *Scrum* y CMMI, el rol del *coach* ágil es un importante aspecto que ha sido incorporado en cursos de ingeniería de software, de modo que el profesor puede entrenar a los estudiantes para ayudarlos a encarar las diferentes etapas del desarrollo de un producto.

Siguiendo esta línea, en la carrera de Ingeniería de Sistemas de la UNICEN, *Scrum* ha sido incluido en el curso de Ingeniería de Software. Antes de realizar este curso, los estudiantes cursan Introducción a la Ingeniería de Software, en el cual los estudiantes son entrenados en el entendimiento de las áreas de CMMI, en los artefactos de software de dichas áreas en relación a *Scrum*, y en el uso de un ambiente de desarrollo ágil. Asimismo, el profesor complementa el curso con anécdotas en empresas, con el objetivo de enfatizar las técnicas pedagógicas.

La Fig. 1 ilustra el modelo usado por los profesores para llevar a cabo el curso. El modelo consiste en un ciclo de vida iterativo e incremental basado en Scrum y el rol de un *coach* ágil. Por medio de este modelo se simula una organización de software real basada en los principios ágiles y la disciplina de CMMI, en la cual los estudiantes y profesores juegan los roles de *Scrum*. Los estudiantes son organizados en *Scrum Teams* y uno de ellos es seleccionado para ser el *Scrum Master*. El *Scrum Team* es responsable de desarrollar la funcionalidad requerida por el cliente y el *Scrum Master* es el facilitador del equipo y lo ayuda a tomar decisiones o adquirir los recursos necesarios. El rol del *Product Owner* es responsabilidad del profesor y se encarga de priorizar y negociar junto al equipo la funcionalidad a realizar en cada iteración. También, es el encargado de despejar dudas relacionadas con los requerimientos pedidos por el cliente. Además, el profesor juega el rol del *coach* ágil que alienta a los equipos a seguir el proceso de *Scrum* [13], ayuda a remover obstáculos organizacionales y enfatiza el uso de las herramientas para mantener la trazabilidad de los requerimientos. Asimismo, trata de internalizar los valores propuestos por el Manifiesto Ágil (http://agilemanifesto.org/).

El curso inicia con la actividad denominada Fase Inicial. La primera parte de esta fase consiste en realizar toda la configuración del ambiente de desarrollo y revisar que todas las estaciones de trabajo de los estudiantes estén funcionando correctamente con todas las características requeridas. La segunda parte consiste de la definición de los requerimientos a ser desarrollados durante el curso. Los requerimientos en *Scrum* se organizan en *user stories*, las cuales son tarjetas que describen cada una de las funcionalidades del sistema, y son desarrolladas en iteraciones llamadas *Sprints*. La definición de las *user stories* es realizada por el *Product Owner* y se almacenan en un artefacto denominado *Product Backlog*. Este artefacto es supervisado por el *Product Owner*, quien es encargado de priorizar las *user stories*. En cada *Sprint*, el *Product Owner* y el *Scrum Team* acuerdan el conjunto de *user stories* del *Product Backlog* que serán desarrolladas durante la iteración. Las *user stories* seleccionadas se cargan en el artefacto denominado *Sprint Backlog*. Los diferentes *Backlogs* son supervisados por el *Product Owner*.

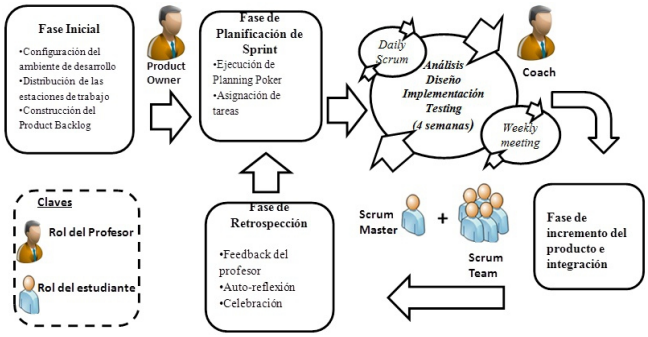

Figura 1. Modelo de enseñanza ágil.

En la siguiente fase, Fase de Planificación del Sprint, se lleva a cabo la estimación y planificación del trabajo a realizar durante el *sprint*. Antes de definir el *Sprint Backlog*, cada equipo estima la complejidad de las *user stories* usando la técnica de *Planning Poker* [3]. Como consecuencia, los miembros del equipo descomponen cada *user story* en tareas de menor alcance y se organizan para llevarlas a cabo. En esta fase, el *Product Owner* trabaja cerca del equipo para clarificar y aprobar las *user stories*. Así, se cubren prácticas de planificación definidas por CMMI.

Habiendo planificado las *user stories*, el *Scrum Team* está listo para empezar a desarrollar las mismas durante un período de 4 semanas. Durante este período, cada *user story* sigue un proceso miniatura que consiste en su análisis, diseño,

implementación y verificación. Los artefactos generados en esta fase deben cubrir las prácticas de CMMI relacionadas con el manejo de requerimientos, solución técnica, verificación de software y aseguramiento de calidad, tanto del producto como del proceso. En este contexto, surge el concepto de "completa". En nuestro modelo, una *user story* está "completa" si se ha seguido correctamente el proceso miniatura.

Una vez al día, los miembros del *Scrum Team* se reúnen cumplimentando la sub-fase conocida como *Daily Scrum*. Durante esta reunión, que no debe extenderse más allá de 15 minutos, cada miembro del *Scrum Team* expone su estado de avance. Esta exposición es conducida por las siguientes preguntas: ¿Qué has hecho desde la última reunión?; ¿Qué planeas hacer entre esta reunión y la próxima? y ¿Qué obstáculos te impiden realizar tu trabajo? Como consecuencia, estas reuniones permiten a los estudiantes cubrir prácticas relacionadas con el monitoreo y control del proyecto, manejo de riesgos y revisión por pares.

Una vez a la semana, cada *Scrum Team* se reúne con el *Coach* ágil. El objetivo de la reunión es monitorear el desempeño de los estudiantes en cada etapa del proceso miniatura. El *Coach* ágil promueve que los estudiantes muestren el diseño arquitectónico, las especificaciones de requerimientos, y cualquier documentación relevante. Sobre esta información, el *Coach* ágil debe realizar las devoluciones a los equipos que los guíen por el camino correcto. Es importante notar que estas devoluciones no deben interferir en la auto-organización del *Scrum Team* para mantener la esencia de este tipo de metodología ágil.

Al finalizar la semana 4, en la Fase de Incremento del producto e Integración, los *Scrum Teams* tienen que integrar sus desarrollos parciales en un único producto. Este producto es evaluado y validado por el *Product Owner*, quien realiza la devolución final del *Sprint* sobre los objetivos alcanzados. Finalmente, el equipo reflexiona sobre lo acontecido a lo largo de las 4 semanas en la Fase de Retrospección. Aquí, se revisan las *user stories* completas, incompletas y no comenzadas ajustando sus estimaciones de acuerdo a la experiencia de cada *Scrum Team.* Adicionalmente, el *Coach* realiza las devoluciones sobre la calidad de los productos, alienta a los miembros de los equipos a que reflexionen sobre sus desempeños y analicen las fuentes de sus errores. En base a las sugerencias del *Coach* y sus acciones correctivas, los equipos discuten sus errores e implementan modificaciones a su rutina.

Al finalizar el curso, los equipos muestran el producto final al *Product Owner*. Este producto es el resultado de la integración de los productos de cada equipo. Por su parte, en el *coach* ágil lleva a cabo la evaluación de los artefactos de software que complementan al producto entregado.

III. EL ROL DEL COACH ÁGIL

Un problema típico en contextos de enseñanza de Ingeniería de Software es la falta de entrenamiento de los equipos de Scrum. Aumentar los roles de Scrum con el rol de Coach Ágil permite a los profesores entrenar a los estudiantes en la diversidad de aspectos del desarrollo de software, para profundizar en la comprensión de un proceso basado en Scrum, y adquirir habilidades no técnicas relacionadas con el desarrollo de software [25]. En un entorno educativo, el Coach Ágil tiene la intención de apoyar y desarrollo de los estudiantes actuando como consultor y entrenador en prácticas de ingeniería de software, maximizando la experiencia de aprendizaje de los estudiantes [28].

Sería beneficioso integrar el entrenamiento en un curso de Scrum construido en una simulación de un escenario profesional, con el fin de proporcionar a los estudiantes la oportunidad de experimentar el uso de Scrum en un contexto seguro y controlado. A pesar de los beneficios reconocidos de entrenamiento, se ha demostrado difícil de implementar estrategias efectivas de *coaching* ágil incluso dentro de la academia [26].

Un Coach Ágil tiene en cuenta los problemas críticos dentro y fuera de los equipos de los alumnos [27], a resolver los impedimentos y alentar a los miembros del equipo y Scrum Masters de aplicar los valores ágiles. Aunque todos los trabajos académicos mencionados anteriormente han reconocido que el *coaching* ágil es un factor clave, ha habido insuficiente análisis sobre el impacto del *coaching* en el rendimiento de los equipos ágiles.

IV. CASO DE ESTUDIO

Con el objetivo de evaluar la efectividad de nuestro modelo ágil de enseñanza, llevamos a cabo 3 casos de estudio entre 2008 y 2010, en el contexto del curso de Ingeniería de Software de la carrera de Ingeniería de Sistemas (Fac. Cs. Exactas-UNICEN, Argentina). El primer caso de estudio implementamos CMMI con RUP y participaron 63 estudiantes en el año 2008. El caso de estudio del año siguiente consistió en evaluar el cambio de RUP por *Scrum*, donde participaron 56 estudiantes. Este curso siguió *Scrum* sin la inclusión del rol de *coach* ágil dentro del modelo. Por último, el caso de estudio del año 2010 consistió en complementar el modelo basado en *Scrum* con el rol de *Coach* ágil. En este tercer curso participaron 61 estudiantes. Los estudiantes que participaron de cada caso de estudio fueron divididos en equipos de 7±2 miembros. A cada equipo se le asignó un conjunto de requerimientos con similar complejidad que debían ser desarrollados a lo largo del curso. En cuanto al ambiente de desarrollo, los estudiantes usaron un conjunto de herramientas *open source* y con licencia para universidades. Dichas herramientas fueron provistas por la cátedra y ésta se encargó de entrenar a los estudiantes en el uso de todas ellas.

Para evaluar cada experimento, se consideraron las prácticas de CMMI y se definió la métrica cobertura de práctica de la siguiente manera: una práctica de CMMI fue considerada cubierta si había al menos un artefacto de software que evidencia el cumplimiento de la práctica para cada requerimiento (user story). Con el objetivo de uniformar la medición de la cobertura de prácticas, hemos establecido un mapeo, primero entre CMMI y RUP, y luego, entre CMMI y Scrum. El primer mapeo está basado en el trabajo presentado en [16, 29]. El segundo mapeo se basa en los trabajos de [6, 9, 17, 22]. Las columnas de la Tabla 1 "Prácticas de RUP" y "Prácticas de Scrum" resumen el mapeo con la columna "Prácticas de CMMI".

A. Resultados obtenidos

En esta sección se analizan los resultados obtenidos de la evolución del curso. Las columnas 2008, 2009 y 2010 de la Tabla 1 muestran el porcentaje de cobertura de las prácticas en cada caso de estudio. Los resultados muestran que los estudiantes alcanzaron los más altos niveles de cobertura de prácticas de CMMI al incluir al *Coach* ágil en el modelo. En base a los resultados, podemos establecer que este rol ayuda a los estudiantes a llegar a sus hitos con procesos de alta calidad y a internalizar el concepto de equipo ágil. Como se dijo anteriormente, el modelo de RUP consiste en un fuerte establecimiento de un plan y definición de entregables. Esta característica es representada por la alta cobertura de la práctica P1 en la experiencia con RUP. No obstante, encontramos una baja cobertura de prácticas P2 y P3. Sin embargo, *Scrum* por sí mismo fue insuficiente para tratar con este problema, ya que los estudiantes malinterpretaron varios principios ágiles asumiendo que no era necesario planificar en un contexto ágil. Varios estudiantes tenían conceptos erróneos de *Scrum*, como por ejemplo, "planificar es una pérdida de tiempo", "no es necesario documentar" y "el diseño es muy difícil de lograrlo". Esto llevó a una débil cobertura en la práctica P1. Cuando se incorpora el rol del *coach* ágil en el 2010, la cobertura de la práctica P1 aumentó casi llegando a los niveles del 2008.

En 2008, RUP llevo a los estudiantes a seguir un proceso tipo cascada que produjo un inevitable retraso en las etapas de capturas de requerimientos. Para recuperar este retraso, los estudiantes saltearon actividades de RUP repercutiendo en una débil cobertura de prácticas relacionas con el diseño e implementación (P4), verificación (P7), integración (P10) y despliegue (P11), tal como se muestra en la Tabla I.

TABLA I. MAPEO ENTRE CMMI, RUP Y SCRUM JUNTO CON LOS PORCENTAJES DE COBERTURAS DE PRÁCTICAS.

Prácticas de CMMI	Prácticas de RUP	%cobertura	Prácticas de Scrum	%cobertura	
		2008		2009	2010
P1. Establecer y mantener las estimaciones del Plan del Proyecto	Establecer el *business case* para el sistema y delimitar el alcance del proyecto	66.5	Establecer la fase de pre-juego de Scrum y realizar Planning Poker	36.6	66.4
P2. Establecer y mantener el Plan de Proyecto como los fundamentos para manejar el proyecto	Establecer el costo y presupuesto de las estimaciones, prioridades, riesgos y proceso de desarrollo	41.4	Establecer la Vision Definir y mantener el Product Backlog	52.2	57.9
P3. Establecer y mantener compromiso con el Plan del Proyecto	Establecer la concurrencia de las partes interesadas en la definición del alcance y costo/presupuesto de las estimaciones.	34	Realizar reuniones de planificación cara a cara	31.6	37.4
P4. Seleccionar soluciones del product o components del product a partir de soluciones alternativas	Establecer el plan de desarrollo para todo el proyecto, mostrando las iteraciones con sus criterios de evaluación	19.3	Desarrollar software en base a un ciclo de vida iterativo e incremental	49.5	50.4
P5. Desarrollar los diseños del producto o componentes del producto	Construir un prototipo ejecutable de la arquitectura	24.8		39.8	55.4
P6. Preparar el ambiente de verificación	Desarrollar una lista de riesgos revisados y revisar el *business case*	32.4	Establecer *done criteria* Realizar reunions de Sprint Review	49.2	55.5
P7. Verificar productos seleccionados contra sus especificaciones de requerimientos	Realizar *beta testing* para validar nuevas funcionalidad según las expectativas del cliente	9.1	Establecer *done criteria* Realizar reunions de Sprint Review	36.7	41.3
P8. Preparar el ambiente de validación	Liberar un *release* estable y maduro	44.1	Involucrar a las partes interesadas	38	54.7
P9. Validar el product o los componentes de product para asegurar su factible uso en el ambiente operativo	Asegurar que el producto sea integrado en las plataformas adecuadas para ser testeado por partes involucradas	29.5	Asignar los roles de Product Owner y Scrum Master	24.8	49.7
P10. Lograr que las interfaces de los componentes del producto sean interna y externamente compatibles	Definir la organización del código, en términos de capas de implementación	21.4	Organizar Scrum de Scrums en caso de equipos grandes Realizar reuniones de Sprint Retrospective	70.1	72
P11. Integrar y ensamblar los componentes del producto, y entregar el producto verificado y validado	Liberar *releases* de productos, y entregar el software a sus usuarios finales	22	Realizar entregar incremental del producto de software	44.3	55.4
P12. Preparar el ambiente para la administración de riesgos	Preparar un marco de trabajo para manejar riesgos	41.4	Definir y mantener el Product Backlog Identificar épicas	47	49.8
P13. Identificar y analizar los riesgos para determinar su importancia relativa	Realizar una evaluación inicial de riesgos	9.2	Realizar reuniones diarias	44.3	66.5
P14. Mitigar riesgos	Administrar la evaluación y remoción de riesgos	22.3	Realizar reuniones diarias Identificar impedimentos	49.9	59.2
P15. Administrar requerimientos e identificar inconsistencias con los planes de proyectos y los productos de trabajo	Entender los requisitos como se evidencia por la fidelidad de los casos de uso primarios	20.6	Establecer la fase de pre-juego de Scrum y realizar Planning Poker. Realizar reuniones de planificación cara a cara. Realizar reuniones de Sprint Retrospective. Administrar *user stories* en el Sprint Backlog	44.3	72
P16. Monitorear el desempeño real y el progreso del proyecto contra el plan del proyecto	Establecer un marco de trabajo para planificar, entrenar recursos, ejecutar y monitorear proyectos	42.5	Realizar reuniones diarias Realizar reuniones de Sprint Retrospective	47	64.7
P17. Administrar acciones correctivas cuando el rendimiento o los resultados del proyecto se apartan significativamente del plan	Lograr la concurrencia de las partes interesadas de que las líneas de base de despliegue sean completas y coherentes con los criterios de evaluación de la visión	33.2	Realizar reuniones de Sprint Review Realizar reuniones diarias	47	63.8
P18. Evaluar objetivamente la adherencia del proceso realizado, de	Evaluar la calidad dentro del proceso, involucrando a las partes interesadas y	16.5		43.9	59.5

los productos asociados y de los servicios	usando métricas y criterios objetivos		Realizar reuniones de Sprint Retrospective		
P19. Registrar y comunicar "no conformidades" objetivamente y asegurar su resolución	Controlar, registrar y monitorear cambios para asegurar que el desarollo iterativo sea exitoso	18.5		58.1	69

La mayoría de estas prácticas mostraron mejores coberturas cuando se incorpora *Scrum*, ya que los estudiantes ejercitaban todos los aspectos de desarrollo de software durante un *sprint*. En el 2010, con la incorporación del *coach* ágil, se encontraron mejores niveles de cobertura de las mismas prácticas. Esta mejora fue producto de la asistencia provista por el *coach* ágil a los *Scrum Masters* en cumplir su rol y en alentar a los estudiantes en seguir el proceso propuesto por nuestro modelo. Por otro lado, la insuficiente comunicación con el *Product Owner* y escaso hábito para la documentación resultaron en una débil cobertura de las prácticas relacionadas con la validación del producto (P8 y P9) en 2008 y 2009. Similar a P8, la cobertura de la práctica P9 alcanzó su más alto nivel en el tercer caso de estudio. En este sentido, el *coach* ágil enfatizó la interacción con el *Product Owner* como aspecto clave en la calidad final del producto lo que produjo también un incremento en la cobertura de las prácticas P12-P17.

Durante la implementación de RUP, encontramos que la falta de definición de criterios para considerar cuándo un requerimiento está completo y el retraso en etapas tempranas del proceso resultaron en una débil cobertura de prácticas P18 y P19. En 2009, Scrum ayudó a mejorar la cobertura de estas prácticas debido a que sólo se especificaron los requerimientos correspondientes a cada iteración. Sin embargo, algunos estudiantes seguían teniendo problemas en internalizar el criterio de "completo" para un requerimiento, dando por finalizada una user story contando únicamente con su implementación, faltando no sólo la documentación mínima sino que también los casos de prueba de unidad. Este inconveniente fue ajustado en el año 2010 por el Coach ágil quién recordaba a los estudiantes las actividades necesarias para dar por finalizada una user story.

Otros indicadores sobre el desempeño de los estudiantes en los tres cursos fueron considerados para reforzar los resultados observados en la Tabla 1. La Tabla 2 resume los valores de los siguientes indicadores: cantidad de miles de líneas de código Java desarrolladas por los estudiantes, cantidad de documentos completados por los estudiantes, cantidad de días que finalmente duró el desarrollo del proyecto, cantidad de requerimientos estimados por los equipos de estudiantes, cantidad de requerimientos finalmente completados por dichos equipos y cantidad de estudiantes participantes del experimento en cada año. Como puede observarse en la Tabla II, el modelo experimentado en el año académico 2010, con la intervención del Coach Ágil, resultó en una optimización de tiempo de desarrollo considerando las user stories desarrolladas, la documentación producida y el volumen de código generado

TABLA II. INDICADORES COMPLEMENTARIOS SOBRE EL DESEMPEÑO DE LOS CURSOS

Indicadores	2008	2009	2010
Líneas de código (en miles)	43.8	41.5	60.65
#Documentos completados	1855	528	840
Duración del proyecto (días)	180	135	120
#Requerimientos estimados	26	65	58
#Requerimientos realizados	12	30	33
#Estudiantes participantes	63	56	61

V. TRABAJOS RELACIONADOS

El desarrollo ágil de software ha recibido últimamente una atención significativa debido a su amplia aplicación en la industria del software [18, 20].

En respuesta a esta tendencia, hay una creciente necesidad de enseñar métodos ágiles en las universidades para preparar a los estudiantes para su desarrollo profesional. En [4] y [7], los autores reportan la experiencia de introducir XP (*extreme programming*) en grandes grupos de estudiantes. Se observó que el uso de esta metodología es altamente adecuada para introducirlos en IS. Nuestro trabajo difiere en que nuestro modelo es evaluado tomando como referencia la cobertura de prácticas de software propuestas por CMMI. En [5], los autores describen sus ocho años de experiencia en enseñar métodos ágiles a varios grupos de estudiantes de diversas universidades introduciendo prácticas tales como *refactoring* y *pair-programming.* También, los autores encontraron apropiados los roles de *Scrum*, las reuniones diarias (*daily meetings*) y las reuniones de retrospección (*sprint retrospective*) para el desarrollo del proceso. En [10], el autor introdujo *pair-programming* en un curso de IS con el objetivo de hacer del curso un escenario agradable para aprender.

Rico reportó un curso al estilo "capstone" basado en métodos ágiles [30], en el cual los estudiantes podían elegir un método ágil de un conjunto conformado por Scrum, XP, Feature-driven development (FDD), Dynamic Systems Development Method (DSDM) y Crystal Clear, entre otros. Un enfoque pedagógico se reportó en [31], en el cual los autores proponen un curso híbrido de Sistemas de Información basado en XP, Scrum y FFD. Mahnic [14] ha introducido Scrum en un curso de ingeniería de software como un marco para la planificación y gestión de los proyectos de los estudiantes y se ha centrado en el aprendizaje de los estudiantes sobre Scrum. En este contexto, un estudio anterior reveló que los estudiantes disfrutaron de trabajar con Scrum y realizaron una exitosa experiencia práctica [13]. Mediante la simulación de un entorno de ingeniería de software lo más cercano posible al mundo real, Scharf y otros presentaron un curso universitario sobre Scrum complementado con herramientas útiles para poner en práctica este método ágil [32]. Siguiendo la misma línea, Paasivaara y otros introdujeron un juego de simulación para enseñar los roles de Scrum, eventos y conceptos relacionados logrando resultados prometedores a través de la retroalimentación de los estudiantes [33]. Mahnic también ha proporcionado un resumen de la literatura abordar la enseñanza de Scrum en cursos de ingeniería de software. Este esquema pretende arrojar algo de luz sobre lo que se ha informado sobre el uso de Scrum en los cursos, y cuáles son los enfoques generalizados en la enseñanza de Scrum [34].

Sin embargo, la inclusión de un *coaching* ágil no es

discutida en estos trabajos. Más aún, nuestro enfoque considera la enseñanza de prácticas de CMMI mediante la realización de prácticas de *Scrum*.

En nuestro trabajo, nosotros planteamos la inclusión de un coach ágil para guiar, monitorear y asistir a los equipos formados por estudiantes. En este contexto, nuestro enfoque se alinea con el trabajo presentado en [1], en el cual los autores sostienen que los profesores debe jugar el rol de un gerente de proyecto que apunta a planificar, monitorear y controlar el proceso de aprendizaje eficientemente. Los autores proponen un modelo ágil e iterativo en un curso de IS. Este modelo sirvió tanto como una técnica educativa para los profesores, como así también una herramienta de aprendizaje para los estudiantes. Sin embargo, el impacto del gerente en el modelo de enseñanza no se describe en términos de la calidad de las prácticas de software y procesos con el objetivo de conocer los beneficios de incluir al gerente en el modelo.

Finalmente, la combinación de CMMI y los métodos ágiles en el desarrollo de software ha sido tratada por varios autores [2, 21]. Si bien estos autores demuestran el impacto en las empresas, no tratan sus efectos en la enseñanza de IS.

VI. CONCLUSIONES

En este trabajo presentamos un modelo de enseñanza basado en una combinación entre *Scrum*, CMMI y el rol del *Coach* ágil para introducir el desarrollo ágil de software en un curso de ingeniería de software, focalizándonos no sólo en mejorar el aprendizaje de las principales prácticas de software, sino también en mantener la calidad de los procesos de software. Nuestra estrategia de enseñanza puede ayudar a los estudiantes a integrarse con la industria del software de una mejor manera, nutriéndolos con las habilidades requeridas en los contextos profesionales actuales de desarrollo.

Asimismo, en este trabajo hemos mostrado las debilidades de los procesos rígidos de software basados en RUP. *Scrum* por sí solo aparece como una herramienta útil para disminuir estas limitaciones pero que sin un debido *coaching* surgen varias malas interpretaciones sobre cómo trabajar en un contexto ágil. Por lo tanto, la incorporación del rol del *coach* ágil para guiar y orientar a los estudiantes es un aspecto clave en la mejora su desempeño en la cobertura de buenas prácticas de ingeniería de software.

Esta evidencia es consistente con nuestra hipótesis que un modelo focalizado en la planificación estricta de entregables hace que los estudiantes se centren en llegar a las fechas críticas en lugar de seguir las actividades del proceso de desarrollo adecuadamente.

El principal problema en la implementación de nuestro modelo en un ambiente académico radica en la necesidad de armar un cuarto físico controlado donde llevar a cabo las actividades de *Scrum*, producto no sólo del gran número de alumnos sino que también de la falta de infraestructura edilicia. En este sentido, estamos trabajando en el desarrollo de un mundo virtual que permita a los estudiantes armar y configurar un ambiente de desarrollo ágil [23]. Sobre este mundo virtual, se incluirán herramientas inteligentes que permitan al profesor, por medio de sugerencias y acciones correctivas, proveer *feedback* permanente a los estudiantes sobre su desempeño en el *Scrum Team*. Como trabajo futuro, planeamos evaluar nuestro modelo en entornos profesionales de desarrollo de software, los cuales difieren de un entorno educativo ya que los desarrolladores son estudiantes con escaza experiencia y los roles y requerimientos son simulados. Asimismo, en nuestro entorno educativo no se han simulado las presiones de negocio y de otros stakeholders, recursos presupuestarios y dinero, entre otros. Efectivamente, estos factores podrían influir significativamente en los resultados obtenidos en un entorno educativo.

REFERENCIAS

[1] Alfonso, M. I., Botia, A.: An iterative and agile process model for teaching software engineering. Proceeding of the 18th Conf. on Soft. Eng. Educ. and Training, 9--16 (2005)

[2] Boehm, B., Turner, R.: Balancing Agility and Discipline: A Guide for the Perplexed. Addison Wesley (2008)

[3] Cohn, M.: Agile Estimating and Planning. Prentice Hall (2006)

[4] Coupal, C., Boechler, K.: Introducing agile into a software development capstone project. Proceedings of the Agile Conference, 289--297 (2005)

[5] Devedzic, V., Milenkovic, S. R.: Teaching agile software development: A case study. IEEE Transactions on Education, 54 (2011)

[6] Diaz, J., Garbajosa, J., Calvo-Manzano, J. A.: Mapping cmmi level 2 to scrum practices: An experience report. Software Process Improvement. Communications in Computer and Information Science, 42, 93--104, Springer Berlin Heidelberg (2009)

[7] Hedin, G., Bendix, L., Magnusson, B.: Teaching extreme programming to large groups of students. Journal of Systems and Software, 74(2), 133--146 (2005)

[8] Hurtado Alegría, J. A., Bastarrica, M. C.: Implementing CMMI using a combination of Agile Methods. CLEI Electronic Journal, 1:1 (2006)

[9] Keil, P., Fritzsche, M.: Agile methods and CMMI: Compatibility or conflict? e-Informatica Software Engineering Journal, 1 (2007)

[10] Koster, B.: Agile methods fix software engineering course. J. Comput. Small Coll. 22, 131--137 (2006)

[11] Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley (2003)

[12] Kulpa, M., Johnson, K.: Interpreting the CMMI. CRC Press (2008)

[13] Mahnic, V.: Teaching Scrum through Team-Project Work: Student's Perceptions and Teacher's Observations. The International Journal of Engineering Education (2010)

[14] Mahnic, V.: A capstone course on agile software development using scrum. IEEE Transactions on Education. 55, 99--106 (2011)

[15] Mahnic, V.: A case study on agile estimating and planning using scrum. Electronics and Electrical Engineering. 111, 123--128 (2011)

[16] Manzoni, L. V., Identifying extensions required by RUP to comply with CMM levels 2 and 3, IEEE Trans. Soft. Eng. 181--192 (2003)

[17] Marcal, A., Freitas, B., Soares, F., Furtado, M., Maciel, T. and Belchior, A. Blending Scrum practices and CMMI project management process areas. Inn. in Sys and Soft Eng. 4, 17--29 (2008)

[18] Mathiassen, L. and Pries-Heje, J. Business agility and diffusion of information technology. Eur. J. Inf. Syst. 15, 116--119 (2006)

[19] Osorio, J. A. et. al.: Moving from Waterfall to Iterative Development: An Empirical Evaluation of Advantages, Disadvantages and Risks of RUP. Proceedings of 37th EUROMICRO Conference on Soft. Eng. and Adv. Apps, 453--460, Finland (2011)

[20] Nerur, S., Balijepally, V.: Theoretical reflections on agile development methodologies. Communications ACM. 50, 79--83 (2007)

[21] Paulk, M. C.: Extreme programming from a CMM perspective. IEEE Soft. 18, 19--26 (2001)

[22] Pikkarainen, M., Mantyniemi, A.: An approach for using cmmi in agile software development assessments: Experiences from three case studies. Proceedings of SPICE 2006 conference, Luxemburg (2006)

[23] Rodríguez, G., Soria, Á., Campo, M.: Supporting Virtual Meetings in Distributed Scrum Teams. IEEE Latin America Transactions, 10 (6), 2316 – 2323, 2012.

[24] Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice Hall (2002)

[25] Scott, E., Rodríguez, G., Soria, Á., Campo, M.: Experiences in Software Engineering Education: Using Scrum, Agile Coaching. Overcoming Challenges. Software Engineering Education: Delivering Non-Technical

Knowledge and Skills: Delivering Non-Technical Knowledge and Skills, 250 (2013).
[26] Hedin, G., Bendix, L., Magnusson, B.: Teaching extreme programming to large groups of students. Journal of Systems and Software, 74 (2), 133-146 (2005).
[27] Hazzan, O., Dubinsky, Y.: Teaching framework for software development methods. In Proceedings of the 28th International Conference on Software Engineering, 703-706. New York, USA (2006).
[28] Stettina, C. J., Zhou, Z., Back, T., Katzy, B.: Academic education of software engineering practices: towards planning and improving capstone courses based upon intensive coaching and team routines. Proceeding of the 26th IEEE Conference on Software Engineering Education and Training, 169-178 (2013).
[29] Staff, I. B. M.: Rational unified process: Best practices for software development teams (2003).
[30] Rico, D. F., Sayani, H. H.: Use of agile methods in software engineering education. In IEEE Agile Conference, 174-179 (2009).
[31] Tan, C.-H., Tan, W.-K., Teo, H.-H.: Training students to be agile information systems developers: A pedagogical approach. In Proceedings of the 2008 ACM Conference on Computer personnel doctoral consortium and research, 88—96 (2008).
[32] Scharf, A., Koch, A.: Scrum in a software engineering course: An in-depth praxis report. In IEEE 26th Conference on Software Engineering Education and Training, 159-168 (2013).
[33] Paasivaara, M., Heikkilä, V., Lassenius, C., Toivola, T.: Teaching Students Scrum Using LEGO Blocks. In Companion Proceedings of the 36th ACM International Conference on Software Engineering, 382—391 (2014).
[34] Mahnic, V.: Scrum in software engineering courses: an outline of the literature. Global Journal of Engineering Education, 17 (2) (2015).

Guillermo Rodríguez es doctor e investigador de la Fac. de Ciencias Exactas de Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA-Argentina) desde 2001. Obtuvo el título de Ingeniero de Sistemas en 2010 y el título de Doctor en Ciencias de la Computación en 2014. Sus principales áreas de interés son Arquitecturas de Software, Educación en Ingeniería de Software y Metodologías Ágiles.

Álvaro Soria es doctor e investigador de la Fac. de Ciencias Exactas de Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA-Argentina) desde 2001. Obtuvo el título de Ingeniero de Software (UNCPBA) en el año 2001 y el título de Doctor en Ciencias de la Computación (UNCPBA) en el año 2009. Sus principales áreas de investigación son Arquitecturas de Software, Diseño dirigido por la Calidad, Frameworks Orientados a Objetos y Localización de Fallas.

Marcelo Campo es profesor, doctor e investigador de la Fac. de Ciencias Exactas de Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA-Argentina) desde 1988. Obtuvo el título de Ingeniero de Software (UNCPBA) en el año 1988 y el título de Doctor en Ciencias de la Computación (UFRGS-Brasil) en el año 1997. Sus principales áreas de investigación son Arquitecturas de Software, Agentes Inteligentes en Ingeniería de Software y Visualización de Software.