Actas de las Jenui, vol. 6. 2021.
Páginas: 99-106

Aprendizaje basado en metodologías ágiles centradas en diseño evolutivo dirigido por pruebas de aceptación

Alberto González Pérez, Ramón A. Mollineda Cárdenas, David Llorens Piñana
Departament de Llenguatges i Sistemes Informàtics
Universitat Jaume I
Castelló de la Plana
{alberto.gonzalez,ramon.mollineda,david.llorens}@uji.es

Resumen

Este artículo presenta una experiencia de aprendizaje basado en proyecto a partir de la coordinación docente entre dos asignaturas del Grado en Ingeniería Informática de la Universitat Jaume I, con el objetivo principal de mejorar competencias prácticas en el uso de metodologías ágiles de desarrollo de software muy difíciles de adquirir en asignaturas aisladas. La propuesta consiste en un proyecto de prácticas compartido entre las asignaturas Diseño de software y Paradigmas de software, las cuales se imparten en el primer cuatrimestre del cuarto curso en la intensificación en Ingeniería de Software. La primera asignatura introduce fundamentos de diseño de software, mientras que la segunda estudia la metodología ágil Desarrollo Dirigido por Pruebas de Aceptación (ATDD, de *Acceptance Test Driven Development*). El proyecto fue concebido para promover estrategias de diseño evolutivo de arriba a abajo centradas en la gestión eficiente de dependencias, según necesidades de usuarios formuladas en términos de pruebas de aceptación ejecutables escritas antes de diseñar el código objetivo. La especificación incluyó el uso de tecnologías de desarrollo web, aplicaciones móviles y servicios en la nube, contexto en el que se generaron escenarios ricos en gestión de dependencias desde la doble perspectiva del diseño y de la validación del software. Además de fomentar valores de la cultura ágil, la propuesta pretendía eliminar tareas redundantes (presentes en proyectos diferentes) y ofrecer una experiencia más cercana al desarrollo de soluciones profesionales. Los resultados de una encuesta revelaron un alumnado motivado con un proyecto realista, así como la percepción mayoritaria de haber experimentado principios claves del diseño y desarrollo ágil bajo condiciones de incertidumbres.

Abstract

This paper presents a project-based learning experience resulting from the coordination between two subjects of the Degree in Computer Engineering at the Universitat Jaume I, with the main objective of improving practical skills in the use of agile software development methodologies, which are very difficult to acquire in isolated subjects. The proposal consists in a shared project between the subjects Software Design and Software Paradigms, which are taught in the same semester of the fourth year within the Software Engineering intensification. The first subject introduces software design fundamentals, while the second one studies the agile methodology Acceptance Test Driven Development (ATDD). The project was conceived to promote top-down evolutionary design strategies focused on the efficient management of dependencies, driven by user needs formulated in terms of executable acceptance tests written before designing the target code. The specification included the use of web development technologies, mobile applications and cloud services, a context in which rich dependency management scenarios were generated from the dual perspective of software design and validation. In addition to promoting agile culture values, the proposal was aimed to eliminate redundant tasks (present in different projects) and to offer an experience closer to the development of professional solutions. The results of a survey revealed students motivated with a realistic project, as well as a widespread perception of having experienced key principles of agile design and development under conditions of uncertainty.

Palabras clave

Aprendizaje basado en proyectos, diseño y desarrollo ágil, desarrollo dirigido por pruebas de aceptación.

1. Introducción

El desarrollo ágil de software puede definirse como un proceso iterativo dirigido a producir versiones incrementales del software objetivo con valor creciente para el usuario final. Las metodologías ágiles han inspirado numerosas estrategias formativas, y algunos de

sus principios han sido transferidos a contextos educativos en forma de competencias [9]. Entre los valores que promueve la cultura ágil, resumidos a través de doce principios del Manifiesto Ágil [1], destacan el trabajo en equipo, la responsabilidad individual y colectiva en la toma de decisiones, la interacción entre personas, la mejora continua, la gestión de incertidumbres y el uso eficiente de recursos. Una adaptación de estos valores al entorno del aula es explícitamente propuesta en [12], siendo cada principio reinterpretado en forma de actividades educativas específicas. Estos principios han sido principalmente usados en experiencias de aprendizaje basado en proyectos de Ingeniería de software [5,8], aunque también existen esfuerzos en Ingeniería eléctrica [11] y Matemática Discreta [3].

En el caso particular de la metodología ágil *Desarrollo Dirigido por Pruebas de Aceptación* (ATDD) [7], cada iteración es guiada por una secuencia de pruebas de aceptación que se definen e implementan antes de diseñar el código que se pretende validar, lo cual plantea a los alumnos un paradigma de programación poco convencional. Esta forma de desarrollo, a partir de pruebas tempranas definidas de forma conjunta por el cliente y el personal técnico, contribuye a clarificar requisitos, proporciona una medida objetiva del progreso del proyecto y fomenta diseños simples centrados en las necesidades del usuario.

Cada iteración ágil involucra todas las fases de desarrollo, en particular, el diseño de software que, en este contexto, se conoce como diseño evolutivo. Es decir, en cada iteración se toman decisiones de diseño desconociendo requisitos aún no tratados (que se analizarán en iteraciones futuras), por lo que deberá mantenerse una estructura de código flexible que admita nuevos servicios con bajo impacto en el diseño existente. Esta práctica no implica necesariamente mayores esfuerzos, siempre que el diseño se base en buenos principios como los agrupados bajo el acrónimo SOLID: 1) entidades con responsabilidad única, 2) entidades abiertas a extensión, pero cerradas a modificación, 3) diseño por contrato (los tipos bases deberían poder sustituirse por subtipos derivados), 4) principio de inversión de dependencia (las entidades de alto nivel no dependen de entidades de bajo nivel, sino de interfaces), y 5) principio de segregación de interfaces (las clases clientes no dependen de interfaces que no usan). Además de beneficios para el diseño, estos principios permiten una validación eficiente del software mediante dobles de prueba (*mocks*), y la creación de conjuntos de pruebas simples y fáciles de mantener.

Este paradigma puede reinterpretarse como diseño dirigido por pruebas, es decir, diseñar a partir de necesidades de usuarios promoviendo diseños flexibles basados en entidades simples cohesionadas y mínimamente acopladas. Esta perspectiva de análisis justifica la definición de proyectos integradores centrados en pruebas y en diseño del software, los cuales podrían contribuir a adquirir competencias muy difíciles de trabajar en asignaturas aisladas. Desde el punto de vista del diseño [6], este tipo de proyectos ofrece oportunidades para trabajar la interacción del usuario con el sistema, el prototipado (simulación de alternativas de diseño, validación de ideas, etc.), el diseño visual (estética, consistencia, propósito, etc.), la redacción de documentación e instrucciones concisas y útiles (especificaciones, manual de usuario, etc.), la estructuración de contenidos (modelos, taxonomías, navegación, espacios de nombre, etc.), el diseño basado en las necesidades de usuarios, entre otras capacidades.

El aprendizaje basado en proyectos que aprovechan sinergias naturales entre asignaturas es una práctica que se ha mostrado efectiva en la formación de los alumnos de grados en Ingeniería Informática [10,11]. Un proyecto vertebrado en torno a la metodología ágil *Scrum* [13] que involucra tres asignaturas fue propuesto en [10], en el marco de la intensificación en Ingeniería del Software del Grado en Ingeniería Informática de la Universidad de Cantabria. Los autores plantearon una integración parcial de las asignaturas, a través de un proyecto común durante una parte del cuatrimestre. El proyecto fue desarrollado en equipos de 6 alumnos que trabajaron indistintamente en los horarios de las tres asignaturas con ajuste al modelo Scrum. Aunque se destaca la importancia de las pruebas en el desarrollo del proyecto, a diferencia de nuestra propuesta, no parece estar dirigido por pruebas de aceptación tempranas, y omite detalles sobre el tipo y alcance de las pruebas definidas, las metodologías involucradas, las tecnologías de automatización, y los beneficios de la definición temprana de pruebas más allá de la validación de la calidad funcional del software. En [4], la coordinación también se articula a través de un proyecto compartido por tres asignaturas. Su principal novedad es que estas asignaturas pertenecen a cursos distintos, por lo que los alumnos desempeñan un rol acorde con su nivel de formación. A diferencia de nuestra propuesta, el desarrollo no sigue una metodología ágil, y las pruebas se definen en las últimas etapas del ciclo de vida. Tampoco hay información sobre el tipo y alcance de las pruebas.

Desde la perspectiva de la formación de ingenieros informáticos, la novedad de esta iniciativa de coordinación docente consiste en la definición de un proyecto poco convencional de desarrollo de software en el marco de las metodologías ágiles, centrado en la creación y gestión de un diseño evolutivo que debe ir satisfaciendo progresivamente las necesidades de los usuarios representadas a través de pruebas de aceptación tempranas y automatizadas. Estas pruebas se entenderán como una especificación formal de necesidades de usuarios. Como se ha comentado, se espera que esta dinámica favorezca competencias difíciles de adquirir en asignaturas aisladas, así como diseños simples, efectivos y bien cohesionados.

2. Contexto

Las asignaturas *Diseño de software* y *Paradigmas de software* forman parte de la intensificación de Ingeniería de Software del Grado en Ingeniería Informática de la Universitat Jaume I, y se imparten en el primer cuatrimestre del cuarto curso del grado.

Ambas asignaturas siguen una planificación docente muy similar: las primeras semanas del curso se dedican a impartir clases magistrales y seminarios en los que los alumnos aprenden fundamentos teóricos. En el caso de la asignatura Diseño de software, estas primeras clases se acompañan de sesiones prácticas en las que realizan ejercicios (de alcance reducido) relacionados con los conceptos revisados en teoría.

Antes del curso 2020-2021, tras las primeras semanas, ambas asignaturas proponían un proyecto realista de desarrollo de software de tamaño medio. En el caso de la asignatura Diseño de software, el proyecto constaba habitualmente de varias aplicaciones. El desarrollo comenzaba con una etapa inicial de análisis de software, que incluía obtención de requisitos, descripción de casos de uso y definición de historias de usuario. A continuación, los alumnos priorizaban las historias de usuario, antes de proceder al diseño e implementación del proyecto considerando dos requisitos técnicos fundamentales: incluir al menos dos patrones de diseño, y definir interfaces gráficas con usabilidad aceptable.

En el caso de la asignatura de Paradigmas de software, se proponía a los alumnos la elaboración de un proyecto compuesto por una única aplicación, aunque esta debía consumir servicios externos que proporcionaran información real para acercarla a una solución real, por ejemplo, la previsión meteorológica. El proyecto debía desarrollarse siguiendo la metodología ágil ATDD, comenzando por un proceso de análisis de software consistente en la obtención de requisitos y definición de historias de usuario. Las historias se descomponían en escenarios de uso/prueba, a partir de los cuales se identificaban ejemplos concretos de uso del software que se convertían en pruebas de aceptación. Estas pruebas se automatizaban y se usaban como guía para el desarrollo iterativo (cada iteración requería superar una nueva prueba), y como criterio de validación objetiva de la historia y el escenario vinculado. Además, se pedía realizar pruebas de integración sobre aquellos componentes software que hicieran uso de las dependencias externas (e.g. bases de datos, servicios web, etc.), mediante la sustitución de estas últimas por dobles de prueba (*mocks*). Este objetivo necesita del patrón de inyección de dependencias, lo cual "forzaba" a los alumnos a mantener una estructura de código simple y flexible, resultado de aplicar buenos principios de diseño. Además, se promovía la creación de pruebas de unidad siguiendo el método de Desarrollo Dirigido por Pruebas (TDD, *Test Driven Development*), así como el diseño de interfaces gráficas simples y con buena usabilidad (a criterio del profesor).

En resumen, los alumnos debían atender dos proyectos distintos de desarrollo de software en un mismo período docente, con exigencias técnicas complementarias o redundantes. Esta situación provocaba una serie de inconvenientes, algunos identificados en [10], los cuales se resumen a continuación:

- *Actividades redundantes*. Ambos proyectos incluían sendas fases de análisis de software que coincidían en la obtención de requisitos y en la definición de historias de usuario. Por experiencias anteriores, este análisis puede llegar a consumir hasta un tercio del tiempo previsto para cada proyecto, reduciendo así considerablemente el tiempo disponible para completar las tareas con competencias propias de las asignaturas, por ejemplo, diseño de software, escritura de pruebas, etc. Ambos proyectos también coincidían en exigir un diseño de calidad de la lógica del software, así como interfaces gráficas de usuario con arreglo a buenos criterios de usabilidad. Estas duplicidades tenían un efecto desmotivador y generaban una gestión ineficiente de recursos.
- *Falta de visión integradora*. Al finalizar el cuatrimestre, los proyectos de la asignatura Paradigmas de software presentaban una calidad notable, validada por pruebas de aceptación, integración y unitarias. Sin embargo, la cantidad y complejidad de las funcionalidades implementadas eran escasas, reduciendo el valor práctico del resultado. En cambio, los proyectos de Diseño de software solían tener una gran complejidad funcional, así como un diseño de interfaces mucho más elaborado. No obstante, estas funcionalidades no venían (habitualmente) acompañadas de pruebas sistemáticas y automatizadas. La realización de ambos proyectos por separado desaprovechaba oportunidades de ofrecer a los alumnos una visión más realista del desarrollo de software.
- *Sinergias no aprovechadas*. Un ciclo de vida ágil suele promover el diseño de la solución y la definición de pruebas conjuntamente. Hasta ahora, cada asignatura se centraba en una de las dos áreas, mientras que la otra jugaba un rol de mero acompañamiento. Además, la relación entre la calidad del diseño y la necesidad de escribir pruebas no invasivas era poco considerada. De la misma forma, la capacidad de las pruebas de aceptación tempranas para fomentar diseños centrados en las necesidades del usuario, era desaprovechada.
- *Mala gestión del tiempo*. Dos proyectos distintos y complejos, que debían desarrollarse en paralelo, dificultaban la organización del tiempo. Una práctica común era priorizar aquella asignatura que tuviese la fecha de entrega más temprana, lo cual perjudicaba el trabajo metodológico en una o en ambas asignaturas. En general, la coincidencia

de dos proyectos en un mismo cuatrimestre generaba en los alumnos sensaciones de malestar, sobrecarga de trabajo y de escasa coordinación docente. La experiencia en su conjunto terminaba siendo más estresante que enriquecedora.

La unificación de ambos proyectos pretende eliminar redundancias y corregir deficiencias.

3. Proyecto docente

3.1. Fundamentos

ATDD es una metodología de desarrollo ágil que promueve la especificación de las necesidades de clientes y usuarios a través de pruebas de aceptación ejecutables, antes del diseño e implementación del código destinado a cubrir tales necesidades. ATDD comparte filosofía con TDD, con la diferencia de que la primera se centra en pruebas de aceptación y el desarrollo del sistema, mientras que la segunda suele limitarse a pruebas de unidad y a métodos de clase.

Ambas estrategias son procesos iterativos que comienzan escribiendo una nueva prueba que falla, pues evalúa una funcionalidad aún no implementada, seguida de una etapa de desarrollo mínimo cuyo objetivo es pasar la nueva prueba (junto con aquellas definidas previamente). La Figura 1 muestra un esquema resumido que integra ambos procesos.

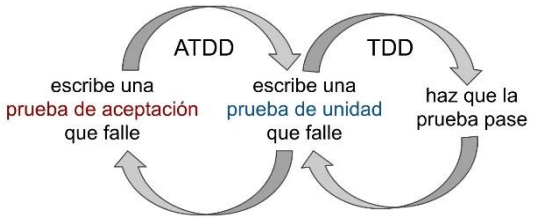

Figura 1: Integración entre ATDD y TDD.

En el ámbito del proyecto, las pruebas de aceptación son el resultado de un proceso de refinamiento de las necesidades de los usuarios, que descompone requisitos (abstractos) en historias de usuario, estas en escenarios de prueba, y estos en pruebas de aceptación (ejemplos concretos). Los escenarios recorren patrones válidos e inválidos de uso, hasta cubrir el alcance funcional de cada historia. Finalmente, cada escenario es representado mediante ejemplos concretos (pruebas de aceptación), los cuales son formalmente especificados a través del modelo *Given-When-Then* e implementados a través de entornos de la familia *xUnit* [2].

Las historias, escenarios y pruebas derivadas se usan como base para planificar un desarrollo en ciclos o iteraciones. Las historias de usuario son distribuidas en iteraciones de duración similar, siguiendo criterios de prioridad para el propietario del producto (el profesor) y de complejidad de las historias. Se propone usar la descomposición de una historia en escenarios y pruebas como medida de complejidad: el número de escenarios y de casos extremos se entenderá como un indicador objetivo de la complejidad de la historia y se usará para estimar su coste de desarrollo.

Cada ciclo comienza con la implementación (automatización) de las pruebas de aceptación de las historias de usuario incluidas en dicho ciclo, lo cual obliga a imaginar o concebir las entidades de más alto nivel del futuro diseño involucradas en las pruebas. De esta forma, las pruebas promueven directamente el diseño. Tras una implementación sin lógica de estas entidades, que admita una ejecución fallida de las pruebas, comienza otro proceso iterativo interior (al ciclo actual) dirigido por las pruebas, que realiza en cada paso un desarrollo mínimo que permita pasar una nueva prueba. Nótese que se trata de una implementación de extremo a extremo, que fuerza un diseño en cascada de entidades de mayor a menor nivel que intervienen en la prestación del servicio que se valida en la prueba, proceso en el que se fomenta la modelación de dependencias a través de abstracciones (interfaces).

Además, se decidió proponer una iteración 0 con una única historia de desarrollador (*spike*), consistente en implementar un prototipo con funcionalidad mínima que demostrase la integración de las tecnologías involucradas. El objetivo era aprender a usar tales tecnologías, mitigar riesgos críticos, y aprovechar este conocimiento para realizar estimaciones más fiables.

Como parte de la asignatura Diseño de software, se exige al estudiante el diseño de los protocolos de comunicación y de las interfaces de usuario según patrones MVC, MVP o MVVM, el uso de al menos otros dos patrones de diseño (además de los usados en la interfaz de usuario), y la escritura de un manual de usuario y de un manual de procedimiento.

Se propone una dinámica de trabajo en equipos para analizar historias de usuarios, definir escenarios y pruebas, y alternar programación en pareja (*pair programming*) e individual de código con integración y control de cambios centralizado en repositorio en la nube. La programación por pareja se recomienda en la automatización de pruebas de aceptación, cuando se toman las primeras decisiones de diseño, y la individual en el desarrollo de entidades concretas. No se promueve la división en roles, pues se trata de incentivar que los alumnos colaboren en todas las tareas.

En Paradigmas de software también se exige la implementación de pruebas de integración en las que dependencias externas, por ejemplo, un servicio en la nube, sean sustituidas por dobles de prueba (*mocks*).

3.2. Enunciado del proyecto

El proyecto proponía la implementación de un sistema domótico de alarmas que constaba de un servidor y dos clientes. El servidor no tenía interfaz de usuario,

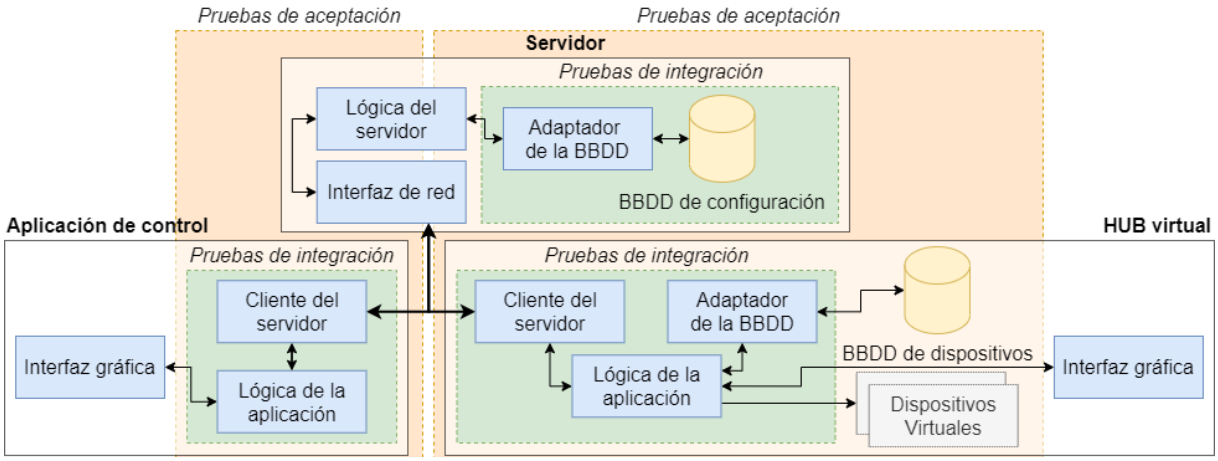

Figura 2: Alcance del proyecto.

incluía la lógica de la aplicación, y se promovía su despliegue como servicio en la nube. Los clientes consistían en un *hub* virtual, para simular a través de una interfaz de usuario un *hub* real conectado a dispositivos sensores o actuadores, y una aplicación de control para monitorizar los dispositivos conectados al *hub* y definir políticas de actuación. La Figura 2 ilustra la relación entre los tres sistemas y el alcance de las pruebas.

El servidor recibía, por un lado, el estado de los sensores desde el *hub* y, por otro, las políticas de actuación definidas por los usuarios a través de la aplicación de control. Estas políticas permitían al servidor, ante un cambio de estado en los sensores, activar los actuadores correspondientes. El *hub* virtual era un simulador (componente de prueba) que permitía añadir, eliminar y modificar dispositivos conectados a través de una interfaz de usuario, así como modificar el estado de los sensores virtuales para provocar escenarios de uso deseados. La aplicación de control debía ofrecer funciones para agrupar dispositivos en habitaciones, activar y desactivar dispositivos, monitorizar estado actual del sistema y garantizar su persistencia.

La lógica del sistema era intencionalmente simple para facilitar que los estudiantes pudieran centrarse en la metodología de desarrollo y en la integración de tecnologías potencialmente novedosas. Al tratarse de dos clientes asíncronos con ámbitos funcionales independientes, estos podían desarrollarse por separado siguiendo el paradigma ATDD.

Se propusieron dos modalidades con diferentes alcances funcionales, requisitos técnicos y nota máxima. La primera, denominada básica, debía constar de dos clientes Java de escritorio y una aplicación servidor también en Java que se ejecutaría en el mismo equipo que los clientes (dirección *localhost*). La segunda modalidad, denominada avanzada, debía implementar la aplicación de control para su uso en una plataforma móvil o web, el *hub* virtual con una interfaz gráfica más semántica y evolucionada, y la aplicación servidor en una plataforma accesible desde Internet.

La mayor parte de las tecnologías requeridas en la modalidad avanzada eran desconocidas para la mayoría de los alumnos. El objetivo principal fue recrear un escenario exigente y realista, en el que los alumnos tuviesen que aprender, de forma autodidacta, a integrar herramientas nuevas para la solución de una tarea. Para facilitar su aprendizaje, el profesor de prácticas desarrolló ejemplos y demostraciones en el aula.

3.3. Sistemas de evaluación

En esta primera experiencia se han mantenido separados los sistemas de evaluación, aunque la defensa final se realizó frente a un tribunal formado por los profesores de ambas asignaturas, lo cual permitió una valoración más integral del proyecto.

En la asignatura Diseño de Software, la evaluación del proyecto se realiza de manera global mediante una única entrega final. Dicha entrega representa el 40% de la nota final de la asignatura. Se establecen dos requisitos mínimos para poder obtener una calificación de aprobado: se debe aplicar más de un patrón de diseño al proyecto, uno de ellos debe ser un patrón de interfaz (MVC, MVP o MVVM), y la interfaz de usuario debe estar razonada y correctamente elaborada.

En la asignatura Paradigmas de Software, el proyecto se evalúa de forma continua a través de tres entregas parciales. La primera (30% de la nota) tiene como objetivo describir jerárquicamente el espacio funcional a través de la identificación de requisitos, historias de usuario por requisito, análisis de escenarios por historia, y pruebas de aceptación por escenario. La segunda entrega (30%) incluye la planificación de iteraciones a partir de estimaciones basadas en la cantidad y complejidad de los escenarios, y del conocimiento adquirido en la iteración 0 (prototipo exploratorio *spike*). Esta segunda entrega también incluye el código que automatiza las pruebas de aceptación de las historias asignadas a la iteración 1. La tercera entrega (40%) incluye el software desarrollado y un informe que describe el proceso de desarrollo, con especial énfasis en la documentación de las pruebas implementadas, su tipología (unitarias, integración, aceptación) y

un breve razonamiento sobre su necesidad. La nota del proyecto representa un 70% de la nota final.

Aunque ambas asignaturas pertenecen a una misma intensificación y se imparten en el mismo semestre del grado, hubo un estudiante en cada asignatura que no se matriculó en la otra. En principio, se logró que ambos formasen un mismo equipo con un tercer alumno que sí estaba matriculado en las dos asignaturas, con la expectativa de que los dos primeros se complementasen. Este equipo fue considerado de dos miembros con relación al alcance del proyecto, mientras que cada alumno sería evaluado en función de los criterios de la asignatura matriculada. No obstante, la experiencia no resultó satisfactoria, pues su éxito descansaba en gran medida en un buen funcionamiento del equipo. En futuras realizaciones de esta propuesta deberán definirse itinerarios alternativos precisos que garanticen una gestión eficaz de estas singularidades.

3.4. Oportunidades

Desde una perspectiva formativa, el proyecto propuesto ofrece amplias oportunidades a los alumnos:

- Experiencia en valores ágiles, por ejemplo, versiones frecuentes de software funcional, colaboración con el cliente (profesor), diseño determinado por necesidades de usuario, etc.
- Valor de las pruebas de aceptación como criterio objetivo de progreso, especificación de las necesidades del usuario, guía para el diseño y el desarrollo, base para estimar esfuerzo, etc.
- Valor de diseños simples y flexibles, con gestión eficiente de dependencias.
- Experiencia con un proyecto realista, pues se propone un dominio real de aplicación.
- Aprendizaje autónomo e integración de diversas tecnologías avanzadas: web, móvil, nube, etc.
- Prototipos exploratorios tempranos para entender nuevas tecnologías, mitigar riesgos estructurales, ganar conocimiento que permita estimaciones más fiables de complejidad, esfuerzo, etc.
- Gestión de la incertidumbre en el diseño y desarrollo de un proyecto de software.

4. Resultados

La evaluación de la experiencia involucró dos perspectivas de análisis. La primera se basó en una encuesta realizada a los alumnos que presentaron el proyecto en primera convocatoria. La segunda consistió en la comparación de las distribuciones de las notas de proyectos de los cursos 2019-2020 y 2020-2021.

4.1. Análisis de las encuestas

La encuesta, mostrada en el Cuadro 1, consta de tres secciones: *Perfil del proyecto*, *Experiencias en tareas de diseño* y *Experiencias en ATDD*. La primera sección está inspirada en el cuestionario propuesto en [10]. Las otras dos son específicas del proyecto propuesto, y su objetivo ha sido doble: promover la autorreflexión en los alumnos y detectar áreas de mejora.

La Figura 3 resume los resultados de la encuesta. Las barras muestran las puntuaciones medias de las cuestiones individuales de las secciones Perfil del proyecto (verde), Experiencias en tareas de diseño (azul) y Experiencias en ATDD (naranja). Medias por sección de 4,70, 4,18 y 4,20 respectivamente, sugieren una muy buena aceptación del proyecto en su conjunto (4,70), aunque también algunas dudas en el logro de algunos objetivos de diseño y desarrollo.

Perfil del proyecto (1-Poco de acuerdo; 5-Muy de acuerdo)
1. Un único proyecto me ha permitido ser más productivo.
2. El proyecto desarrollado me ha parecido realista.
3. Mi motivación ha sido mayor que en proyectos anteriores.
4. He integrado contenidos de ambas asignaturas.
5. Me ha resultado estimulante que el proyecto incluyese tecnologías web, de aplicaciones móviles y servicios en la nube.
Experiencias en tareas de diseño (1-5)
6. He experimentado el valor de diseñar a partir de ejemplos concretos de escenarios de uso/prueba.
7. He experimentado el valor de diseñar un sistema progresivamente de arriba a abajo.
8. He tenido la oportunidad de usar patrones de diseño.
9. He rediseñado menos de lo habitual.
10. He experimentado el valor de implementar un prototipo exploratorio para aprender a integrar las tecnologías del proyecto.
11. He experimentado el valor de diseñar clases con responsabilidad única.
12. He experimentado el valor de extender clases en lugar de modificarlas.
13. He experimentado el valor de diseñar clases dependientes de abstracciones.
14. He experimentado el valor de la inyección de dependencias.
15. Considero que el diseño final es relativamente simple.
Experiencias en ATDD (1-5)
16. Comenzar escribiendo pruebas me ha facilitado la comprensión de los requisitos funcionales.
17. Comenzar escribiendo pruebas me ha facilitado la planificación de cada iteración.
18. Comenzar escribiendo pruebas me ha facilitado la adopción de decisiones de diseño.
19. Comenzar escribiendo pruebas me ha facilitado la colaboración dentro del equipo.
20. Las pruebas de aceptación automatizadas me han permitido una evaluación objetiva del final de la iteración.
21. He experimentado el valor de recibir realimentación frecuente del propietario del sistema (el profesor).
22. He experimentado el valor de entregas periódicas del software con funcionalidad creciente.
23. He visto a mi equipo motivado.

Cuadro 1: Encuesta realizada a los estudiantes.

Un análisis de las cuestiones menos valoradas nos ha permitido detectar limitaciones de la propuesta. En la primera sección, la cuestión menos valorada (4,17) resultó ser "El proyecto desarrollado me ha parecido realista". Aunque es una media favorable, el hecho de desarrollar un *hub* virtual, por la imposibilidad de disponer de dispositivos reales, podría haber influido negativamente. Quizás debió haberse hecho mayor énfasis en el valor del *hub* virtual como herramienta útil para generar escenarios de prueba, incluso en un contexto real que contase con todos los recursos. En la segunda sección, como cuestión menos valorada aparece "He rediseñado menos de lo habitual", con una puntuación media de 3,67. Este resultado es consistente con la complejidad y singularidad de la estrategia de diseño de arriba a abajo, guiada por pruebas de aceptación, y realizando una gestión disciplinada de dependencias e incertidumbres. En realizaciones futuras del proyecto, deberá hacerse un seguimiento exhaustivo de las decisiones de diseño que adopten los equipos. Finalmente, el aspecto peor puntuado de la tercera sección fue "He experimentado el valor de entregas periódicas del software...", con un valor medio de 3,83. Este resultado es coherente con comentarios de texto libre recogidos en la encuesta, que valoran como escaso el tiempo planificado para el desarrollo del proyecto, situación que limitó objetivamente el número de ciclos de desarrollo y las opciones de experimentar mayor progresividad en los incrementos funcionales.

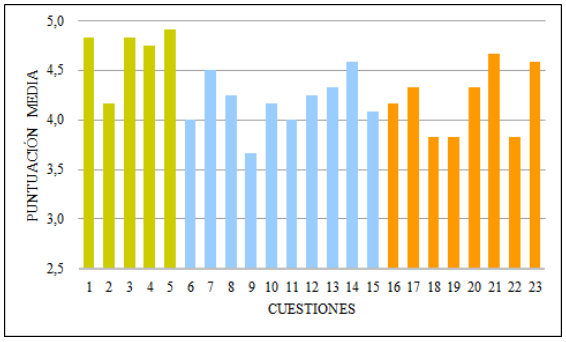

Figura 3: Puntuación media por cuestiones. Los colores ilustran la separación entre secciones.

Merece la pena destacar las altas valoraciones de las cuestiones 7, 13 y 14, relacionadas con las experiencias en diseño de arriba abajo y en gestión de dependencias, y de las cuestiones 16, 17 y 20, relacionadas con el valor de las pruebas de aceptación tempranas para mejorar la comprensión de la tarea, y como criterio objetivo de planificación y validación de ciclos. Además, se incluyeron dos apartados para texto libre para que los alumnos expresaran opiniones positivas y negativas acerca de la experiencia.

Entre las opiniones positivas hubo referencias a la libertad de elección y a la oportunidad para aprender nuevas tecnologías, a poder desarrollar un único proyecto para dos asignaturas, al nivel de motivación colectivo alcanzado, y a la integración de conocimientos de muchas áreas de la carrera. Como aspectos negativos se destacó el poco tiempo previsto para el desarrollo de un proyecto bajo una nueva metodología.

4.2. Análisis de distribuciones de notas

La experiencia también fue evaluada comparando las distribuciones de las notas de proyectos en los cursos 2019-2020 y 2020-2021 (ver Figura 4). El gráfico superior corresponde a la asignatura Diseño de software, y el inferior, a Paradigmas de software. Ambas asignaturas tuvieron proyectos distintos durante 2019-2020, mientras compartieron proyecto en 2020-2021. Los criterios de evaluación se mantuvieron invariables durante los dos cursos y fueron aplicados por el mismo profesorado (diferente entre asignaturas).

En ambas asignaturas se observa un patrón común: la distribución de notas de 2020-2021 (columnas verdes) tiende a situarse a la derecha de la distribución de 2019-2020 (columnas azules). Es decir, las notas del curso 2020-2021 tienden a ser más altas que las de 2019-2020. Este resultado puede contrastarse en el Cuadro 2, donde se muestran las medias y medianas de estas distribuciones de notas. Ambos indicadores crecen en las dos asignaturas con el cambio de curso.

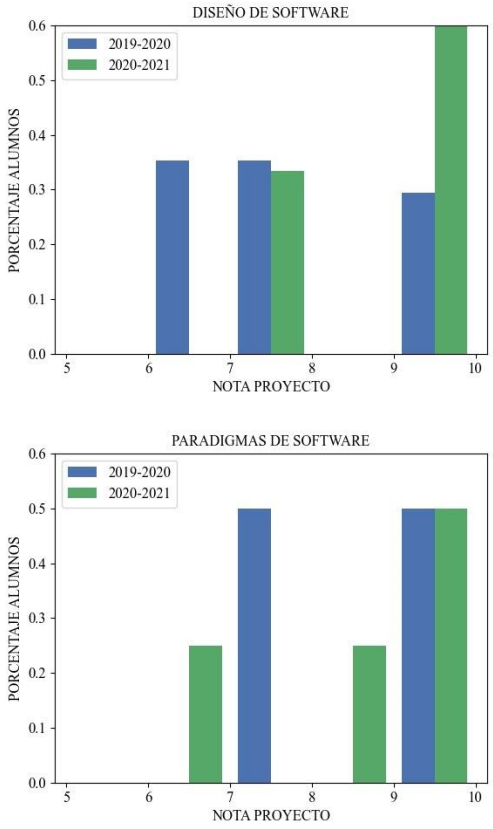

Figura 4: Distribución de notas de proyecto en ambas asignaturas en los cursos 2019-2020 y 2020-2021.

Al margen del patrón descrito, las distribuciones muestran una tendencia bimodal, es decir, dos grupos claramente definidos por notas altas y medias, respectivamente. En la práctica, esto se traduce en equipos que logran una motivación colectiva muy alta, y cuyos proyectos finales son comparables a soluciones profesionales considerando criterios de funcionalidad, usabilidad (según valoración de profesores), uso e integración de tecnologías avanzadas, rendimiento, robustez del diseño y calidad del código. Por el contrario, hay equipos que optan por tecnologías conocidas, desarrollos de bajo riesgo y alcance funcional reducido, dando lugar a soluciones claramente mejorables.

	Diseño de Soft.		Paradigmas de Soft.	
	19-20	20-21	19-20	20-21
media	7,5	9,3	8,2	8,8
mediana	7,0	10,0	7,7	9,4

Cuadro 2: Media y mediana de las distribuciones de notas de proyecto por asignatura y curso.

5. Conclusiones

Este artículo describe una experiencia de coordinación docente que promueve el aprendizaje basado en estrategias de diseño evolutivo de software, dirigidas por necesidades de usuarios formuladas en términos de pruebas de aceptación ejecutables. Los resultados de una encuesta realizada entre los estudiantes reflejaron una buena aceptación de la propuesta, aunque también revelaron la necesidad de mejorar la planificación docente para permitir un desarrollo más sostenible.

La comparación entre distribuciones de notas de los proyectos en los cursos 2019-2020 y 2020-2021 mostró que el proyecto compartido promueve altos grados de motivación en un mayor número de alumnos, los cuales fueron capaces de desarrollar software con un nivel comparable al profesional, mientras se observa un segundo grupo, menor que el primero, que opta por tecnologías conocidas y soluciones más simples.

Futuras implementaciones de este proyecto deberán avanzar hacia una mayor integración de los sistemas de seguimiento y evaluación de ambas asignaturas, desde la sincronización de las entregas, hasta una nota única del proyecto que contemple una visión más integradora del resultado del ejercicio.

Referencias

[1] Kent Beck y otros 16 coautores. Manifesto for agile software development, 2001.

[2] David Chelimsky, Dave Astels, Bryan Helmkamp, Dan North, Zach Dennis, y Aslak Hellesoy, The RSpec book: Behaviour driven development with Rspec. Cucumber, and Friends, Pragmatic Bookshelf, 2010.

[3] Shannon Duvall, Duke Hutchings y Michele Kleckner. Changing perceptions of discrete mathematics through scrum-based course management practices. *Journal of Computing Sciences in Colleges*, vol. 33, pp. 182-189, 2017.

[4] Maria Ferré, Carlos García-Barroso, Montse García-Famoso, David Sánchez y Aida Valls. Mejora de la formación en el diseño y desarrollo de software a partir de la coordinación de distintas asignaturas. En *Actas de las XXV Jornadas de Enseñanza Universitaria de Informática, Jenui 2019*, pp. 23-30, Murcia, julio 2019.

[5] Nicolás Martín Paez. A flipped classroom experience teaching software engineering. En *IEEE/ACM 1st Int. Workshop on Software Engineering Curricula for Millennials (SECM)*. IEEE, pp. 16-20, Argentina, mayo 2017.

[6] Peter Merholz y Kristin Skinner. Org design for design orgs: Building and managing in-house design teams. O'Reilly Media, Inc., 2016.

[7] Ken Pugh. Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration. Addison-Wesley, 2011.

[8] David F. Rico y Hasan H. Sayani. Use of Agile Methods in Software Engineering Education. En *Proceedings of the Agile Conference*, pp. 174-179, Chicago, IL, 2009.

[9] Pasquale Salza, Paolo Musmarra y Filomena Ferrucci. Agile Methodologies in Education: A Review. En *Agile and Lean Concepts for Teaching and Learning*, pp. 25-45, Springer, 2019.

[10] Pablo Sánchez, Carlos Blanco, Alejandro Pérez, Julio Medina, Patricia López, Alfonso de la Vega, Diego García y Miguel Sierra. Experiencia y Lecciones Aprendidas durante el Desarrollo de un Proyecto Software Común a Diversas Asignaturas. En *Actas de las XXIII Jornadas de Enseñanza Universitaria de Informática*, *Jenui 2017*, pp. 127-134, Cáceres, julio 2017.

[11] Laio O. Seman, Romeu Hausmann y Eduardo A. Bezerra. On the students' perceptions of the knowledge formation when submitted to a project-based learning environment using web applications. *Computers & Education*, vol. 117, pp. 16-30, 2018.

[12] John C. Stewart, Carolyn S. DeCusatis, Kevin Kidder, Joseph R. Massi y Kirk M. Anne. Evaluating agile principles in active and cooperative learning. En *Proc. of Student-Faculty Research Day, CSIS,* Pace University, B3, pp. 1-8, 2009.

[13] Jeff Sutherland. Scrum: The Art of Doing Twice the Work in Half the Time. Random House Business, 2015.