Actas de las Jenui, vol. 8. 2023.
Páginas: 89-96

Aprendiendo arquitectura software a partir de proyectos de código abierto en GitHub*

Ana B. Sánchez, José Antonio Parejo, Bedilia Estrada-Torres,
Alfonso E. Márquez-Chamorro, Adela del-Río-Ortega, Sergio Segura
SCORE Lab, Instituto de Investigación I3US, Universidad de Sevilla.
{anabsanchez,japarejo,iestrada,amarquez6,adeladelrio,sergiosegura}@us.es

Resumen

La enseñanza de arquitectura del software supone todo un reto. Los conceptos teóricos son a menudo muy abstractos y los problemas arquitectónicos sólo son claramente visibles en aplicaciones de cierta envergadura. El reto es aún mayor cuando estos conceptos se enseñan en las primeras etapas del grado cuando los conocimientos de diseño y programación del alumnado aún son limitados. Para abordar este reto, inspirados por una propuesta llevada a cabo en la Delft University, decidimos adoptar un enfoque novedoso: enseñar arquitectura del software a través del análisis, evaluación y documentación de la arquitectura de proyectos existentes alojados en la plataforma GitHub. Para ello, fue necesario adaptar el método original, empleado a nivel de máster, a la asignatura objeto del estudio impartida durante el segundo curso de grado. Para evaluar este enfoque realizamos un total de 258 encuestas a estudiantes de dos cursos consecutivos. Los resultados del estudio, respaldados por un sólido análisis estadístico de los datos, demuestran la idoneidad de este método para la enseñanza de arquitectura del software en los primeros cursos de grado.

Abstract

Teaching software architecture is a challenge. Theoretical concepts are often very abstract and architectural problems are only clearly visible in applications of a certain magnitude. The challenge is even greater when these concepts are taught in the early stages of the bachelor's degree when the students' knowledge of design and programming is still limited. To address this challenge, inspired by a proposal carried out at Delft University, we decided to adopt a novel approach: teaching software architecture by analysing, evaluating and documenting the architecture of existing projects hosted on the GitHub platform. To do so, it was necessary to adapt the original method, used in a master course, to the course under study, taught during the second year of the bachelor's degree. To evaluate this approach we conducted a total of 258 student surveys in two consecutive years. The results of the study, supported by a robust statistical analysis of the data, demonstrate the suitability of this method for teaching software architecture in the first years of the bachelor's degree.

Palabras clave

Arquitectura software, ingeniería inversa, GitHub.

1. Introducción

La arquitectura del software comprende la estructura y la relación entre las diferentes partes de un sistema software y sus propiedades [11]. Se trata de una materia clave en la educación de todo ingeniero del software. El diseño de la arquitectura del software suele incluir vistas arquitectónicas, decisiones de diseño, puntos de variabilidad y extensión, patrones y tácticas de diseño, entre otros [1, 2]. Entre sus ventajas se incluyen la de proporcionar una representación del sistema sobre la que poder discutir, ayudar a gestionar la complejidad del sistema, permitir analizar el impacto de futuros cambios, ayudar a detectar errores de diseño en fases tempranas y ser una buena referencia para formar a nuevos participantes [1, 2].

Es habitual que la docencia de arquitectura del software se lleve a cabo en los últimos cursos de gradoo en máster, cuando los alumnos ya cuentan con una buena base de conocimientos de programación. Esto facilita el aprendizaje de los conceptos de diseño necesarios y su puesta en práctica, generalmente a través del diseño e implementación de aplicaciones software usando diferentes estilos y patrones arquitectónicos.

*Trabajo parcialmente financiado por el Dpto de Lenguajes y Sistemas de la Universidad de Sevilla y los proyectos TED2021-131023B-C21, TED2021-131023B-C22, PID2021-126227NB-C21, PID2021-126227NB-C22 financiados por MCIN/AEI/10.13039/501100011033/FEDER y por la Unión Europea NextGenerationEU/PRTR.

La Universidad de Sevilla, sin embargo, sigue un enfoque diferente y concentra las materias de arquitectura del software e integración software en la primera etapa del grado, concretamente en la asignatura de segundo curso *Arquitectura e Integración de Sistemas Software (AISS)*. Esto permite aprender la importancia de las decisiones de diseño y la reutilización pronto, pero también dificulta la docencia tradicional de las mismas—basada en el desarrollo de aplicaciones de cierta envergadura—debido a los limitados conocimientos tecnológicos del alumnado de segundo curso.

En años anteriores, el foco de la asignatura se ha dirigido a la explicación de conceptos teóricos de diseño y arquitectura (ej. estilos y patrones arquitectónicos) y ponerlos en práctica construyendo una aplicación software, en concreto, un mashup haciendo uso del patrón Modelo-Vista-Presentador. Sin embargo, este enfoque suponía un enorme reto ya que los alumnos tenían conocimientos muy básicos de programación y se invertía una parte significativa del tiempo explicando conceptos puramente tecnológicos en lugar de los aspectos propios de arquitectura del software. Este es el problema que motiva nuestro trabajo.

Este artículo presenta nuestra experiencia aplicando un enfoque novedoso inspirado en la propuesta de Van Deursen et al. [15] para afrontar el aprendizaje de la arquitectura del software desde un punto de vista más práctico y colaborativo. En este método, puesto en práctica en un máster universitario de la Delf University, los alumnos tenían que analizar, evaluar y documentar proyectos de código abierto en GitHub y presentar el resultado al resto de sus compañeros en clase [15]. Entre los beneficios del método destacan el aprendizaje de conceptos de arquitectura software a través del análisis de proyectos no triviales, aumentar la motivación de los estudiantes mediante el estudio de proyectos reales, permitirles que se familiaricen con el ecosistema de desarrollo de proyectos de software libre y favorecer el trabajo en equipo y colaborativo.

Nosotros nos enfrentamos al reto de aplicar este método con alumnos de segundo curso de grado frente a los alumnos de máster de la propuesta original, lo que dificultaba su aplicación y planteaba dudas sobre su efectividad. Para evaluar el nuevo enfoque realizamos encuestas a estudiantes de dos cursos consecutivos 2020/21 y 2021/22 donde se aplicó este método. Obtuvimos un total de 258 respuestas que fueron analizadas rigurosamente. Este análisis demostró la idoneidad de la propuesta para la enseñanza de arquitectura del software en los primeros cursos de grado.

El resto del artículo se estructura como sigue. El trabajo relacionado aparece en el apartado 2. El apartado 3 describe el contexto de la asignatura. El apartado 4 presenta el diseño del proyecto de arquitectura del software llevado a cabo. El resultado del enfoque propuesto se presenta en el apartado 5. Finalmente, las lecciones aprendidas y conclusiones obtenidas en el trabajo se recogen en los apartados 6y 7, respectivamente.

2. Trabajo relacionado

La literatura sobre la enseñanza de arquitectura del software se puede dividir a grandes rasgos en dos enfoques metodológicos: el *tradicional*, en el que se imparten conocimientos teóricos y luego se aplican para construir un proyecto software, y el *inverso*, en el que se parte de proyectos existentes, típicamente de código abierto, para estudiar la arquitectura de los mismos.

Dentro del enfoque tradicional, Wedemann [17] propuso el uso de la metodología Scrum en el aprendizaje de arquitectura del software. También se han propuesto herramientas de soporte a la enseñanza de arquitectura del software. Algunas siguen el diseño de arquitectura dirigido por atributos y dan soporte para definir incrementalmente [7] y/o colaborativamente [14] la arquitectura de un determinado producto software. Otras se centran en la visualización 3D de una arquitectura software cuya documentación ha debido ser previamente creada con un editor de arquitectura software [10]. Wang [16] describió una propuesta basada en el diseño de un videojuego para el aprendizaje de arquitectura del software. Otras propuestas describen experiencias en la asignatura. En [8], otorgan a la asignatura un enfoque más industrial, centrándose en la resolución de problemas de diseño de arquitectura de software. En [6] detallan dos modalidades de asignatura, una con enfoque inverso y otra tradicional.

Dentro del enfoque inverso, podemos mencionar varios trabajos [3, 4] que proponen métodos de ingeniería inversa para aprender arquitectura de software con proyectos de código abierto de gran tamaño. Utilizando este enfoque *bottom-up*, el proceso de aprendizaje de estas propuestas se divide en varias fases: análisis, diseño y mantenimiento un subsistema. Por otro lado, el trabajo de Van Deursen et al. [15], referencia principal de nuestra propuesta, propone que los alumnos analicen proyectos de GitHub para aprender arquitectura del software en una asignatura de máster. La asignatura persigue adoptar el código abierto como fuente de aprendizaje, fomentar el trabajo en equipo y colaborativo, permitir a los alumnos interactuar con los arquitectos de sistemas reales y favorecer la realización de contribuciones en proyectos abiertos. Otros autores han propuesto el uso de proyectos de código abierto para la docencia de ingeniería del software, tratando aspectos de diseño y arquitectura, aunque no era la motivación principal del trabajo [5].

A diferencia de estos trabajos, donde el foco suele estar en la modificación de un proyecto existente, el objetivo de nuestra propuesta es la documentación

de la arquitectura. En concreto, la principal contribución de nuestro trabajo es la aplicación y adaptación del enfoque de docencia propuesto por Van Deursen et al. [15] a etapas tempranas del grado. Nuestras adaptaciones incluyen aspectos originales, como el análisis de la calidad de la arquitectura a partir de métricas de código. Además, proporcionamos los datos de la evaluación, apoyada en un sólido análisis estadístico.

3. Contexto

El estudio se llevó a cabo en el contexto de la asignatura de grado Arquitectura e Integración de Sistemas Software. Esta asignatura es obligatoria y se cursa durante el segundo cuatrimestre del segundo curso del grado en Ingeniería del Software de la Universidad de Sevilla. La parte teórica se compone de 18 clases presenciales, mientras que la parte práctica consiste en 12 sesiones de laboratorio en clases habilitadas con ordenadores. Cada sesión dura 1 hora y 50 minutos. La asignatura se divide en dos bloques: un primer bloque sobre arquitectura del software y un segundo bloque sobre integración de sistemas software. Este estudio está centrado en el primer bloque de la asignatura, y abarcó dos cursos lectivos, 2020/21 y 2021/22, implicando a 8 profesores y 258 alumnos (146 el primer año y 112 el segundo). A los estudiantes se les evaluó del mismo modo en los dos cursos que abarca el estudio.

El sistema de evaluación de la asignatura es continua. La parte teórica de la asignatura se evalúa a través de dos exámenes de tipo test, uno al final de cada bloque con un peso del 20 % cada uno con respecto a la nota total de la asignatura. La parte práctica se evalúa con el desempeño de dos proyectos, uno sobre arquitectura software, donde los alumnos tienen que analizar y documentar la arquitectura de proyectos software, y otro sobre integración de sistemas software, donde los alumnos tienen que desarrollar y documentar una API Web. Para llevar a cabo los proyectos los alumnos se organizaron en grupos de entre 3 y 5 personas, aunque la calificación final es individual, ya que normalmente no todos los estudiantes contribuyen y se implican por igual en el proyecto. La realización de ambos proyectos es obligatoria y supone un 30 % cada uno sobre la nota final de la asignatura. Este artículo se centra en el proyecto de arquitectura del software.

4. Diseño del proyecto

El primer paso para llevar a cabo esta experiencia piloto fue diseñar un proyecto de arquitectura del software donde los alumnos tuvieran que enfrentarse al análisis, comprensión y documentación de software existente como método de aprendizaje.

4.1. Competencias

Tomando como referencia el trabajo de Van Deursen et al. [15], la propuesta tiene como objetivo desarrollar tres competencias principales: A nivel de *conocimiento*, buscamos que los estudiantes se familiaricen con conceptos clave de la arquitectura del software, tales como vistas, perspectivas, estilos, principios de diseño, y atributos de calidad, entre otros. A nivel de *aplicación*, se pretende que los estudiantes puedan identificar y aplicar estos conceptos vistos en las clases teóricas a sistemas concretos y existentes que son mantenidos por un equipo de personas y utilizados en todo el mundo. Por último, a nivel de *evaluación*, el objetivo es dar a los estudiantes la oportunidad de evaluar y discutir el efecto de las decisiones arquitectónicas tomadas en los proyectos software (de código abierto).

4.2. Método

Para lograr las tres competencias de aprendizaje, se proponen dos ideas principales. La primera es utilizar aplicaciones de código abierto que permitan a los estudiantes aplicar y evaluar los conceptos de arquitectura del software en proyectos reales, permitiendo además la interacción de los alumnos con los participantes de los proyectos. La segunda idea clave del curso es tener una comunicación abierta entre todos los grupos para que los alumnos puedan intercambiar ideas de los aspectos de diseño de los proyectos software estudiados.

El primer paso que dimos para la puesta en marcha de la propuesta fue seleccionar un conjunto de proyectos reales de código abierto alojados en GitHub. La selección fue realizada por dos profesores titulares con más de 15 años de experiencia en la asignatura. Para la selección de los proyectos de GitHub se consideraron los siguientes criterios de inclusión: 1) proyectos descritos en inglés; 2) dificultad aceptable para los alumnos; 3) proyectos actualizados y activos; 4) a ser posible, proyectos con issues; 5) a ser posible, proyectos conectados a plataformas de evaluación del código fuente, como SonarQube o Codecov.

Los proyectos seleccionados se resumen en el Cuadro 1. Por cada proyecto, se muestra el año del curso académico en el que el proyecto fue utilizado, su nombre en GitHub y el lenguaje de programación. Puede consultarse una versión extendida en el repositorio público habilitado[1]. Estos proyectos suelen estar vinculados a herramientas de monitorización y gestión de pruebas que proporcionan información relevante para el análisis de su calidad. Los profesores creamos una organización GitHub dedicada a la asignatura donde alojamos todos los repositorios que se utilizarían durante el curso. Aquellos proyectos que no estaban enlazados a una plataforma de análisis del código fuente,

[1] Material suplementario: https://doi.org/10.5281/zenodo.7860444

Año	Nombre del proyecto	Lenguaje
2020/2021	google-java-format	Java
2020/2021	gson	Java
2020/2021	h2database	Java
2020/2021	JSON-java	Java
2020/2021	rest-assured	Java
2020/2021	spotify-web-api-java	Java
2020/2021	swagger-parser	Java
2021/2022	cats	Java
2021/2022	client-encryption-java	Java
2021/2022	meme	JavaScript
2021/2022	monica	PHP
2021/2022	prime-simplereport	TypeScript/ Java
2021/2022	scrapy	Python
2021/2022	ta4j	Java
2021/2022	twilio-python	Phyton
2020/21-2021/22	extjwnl	Java
2020/21-2021/22	fastjson	Java
2020/21-2021/22	Flickr4Java	Java
2020/21-2021/22	jgrapht	Java
2020/21-2021/22	mathjs	JavaScript
2020/21-2021/22	openapi-generator	Java
2020/21-2021/22	youtube-dl	Phyton
2020/21-2021/22	zxcvbn4j	Java

Cuadro 1: Proyectos utilizados por los alumnos.

fueron enlazados por los profesores a SonarCloud para que todos tuviesen acceso a dicha información. Los estudiantes, por tanto, hacen uso de GitHub y SonarCloud desde el comienzo de la asignatura.

A cada grupo de alumnos les fue asignado un proyecto software del Cuadro 1. Junto con el proyecto, se les proporcionó un documento plantilla con todos los epígrafes que tenían que cubrir, y un documento de referencia a modo de ejemplo describiendo un proyecto real de GitHub (más detalles en la siguiente sección). Además, se les proporcionó a los alumnos acceso a un libro[2] con todos los proyectos de arquitectura de los alumnos de Van Deursen [15] de 2018 para que contaran con distintas referencias. Dados estos datos, se les pidió a los alumnos que analizaran la arquitectura software del proyecto de GitHub que se les había asignado mientras aprendían en clase los conceptos teóricos. Debemos mencionar también que se diseñaron dos prácticas específicas para dar soporte a este proyecto (una sobre GitHub y otra sobre modelado de software disponibles en el material suplementario). En las clases de prácticas utilizamos el enfoque de clase invertida [13]. Los vídeos correspondientes a ambas prácticas están disponibles online[3]. La parte final de las clases teóricas y prácticas se dedicaba a preguntas que surgían sobre el análisis del proyecto. Además, se dedicaron clases completas a resolver dudas sobre todos los proyectos para que pudieran aprender unos de otros. El resultado final es la documentación del proyecto software que explicaremos en detalle en la siguiente sección.

[2]https://delftswa.gitbooks.io/desosa2018/content/

[3]https://www.youtube.com/playlist?list=PLoz1KhfHj3yERibpfDZzVxvQUARd7DJI_

Con el objetivo de fomentar un aprendizaje abierto y colaborativo entre todos los alumnos, además de ayudarles a mejorar las destrezas comunicativas, los estudiantes deben exponer los progresos de su grupo durante una presentación en clase de unos 5 - 10 minutos de duración. Durante la exposición el profesor sugiere mejoras y cambios en caso de detectar algún error. Esta presentación en público permite a otros compañeros aprender sobre otras arquitecturas software y obtener otros puntos de vista del análisis. Además, las sugerencias del profesor permiten a los alumnos mejorar la documentación de cara a la entrega final del documento que se produce aproximadamente una semana y media después de la presentación. Una vez han entregado la documentación, se lleva a cabo una defensa de cada grupo con el profesor tutor, donde el profesor realiza preguntas grupales e individuales a cada estudiante con el fin de otorgar la nota individual que merece cada alumno según su trabajo y conocimientos. Las rúbricas y detalles de la evaluación se detallan en la sección 4.4.

4.3. Documentación de la arquitectura

La documentación de la arquitectura requerida se basa en gran medida en la estructura recomendada por Nick Rozanski and Eoin Woods [11], también seguida por Van Deursen et al. [15]. No obstante, realizamos algunas modificaciones a esta guía basadas en nuestra experiencia y el temario de la asignatura. El material suplementario incluye una comparativa detallada de ambas propuestas. Seguidamente, se indican los apartados principales destacando las adaptaciones más relevantes:

Participantes. Los alumnos tenían que analizar los tipos de participantes involucrados en el sistema: usuarios, desarrolladores, testeadores, mantenedores, etc. Para ello, examinaban cualquier información que pudieran encontrar en la web del proyecto o en el repositorio de GitHub: la documentación original del proyecto, listas de correos, comentarios en el código, issues, pull requests en GitHub, etc.

Vistas arquitectónicas. Los alumnos debían presentar la descripción visual y textual de, al menos, las vistas de contexto, escenarios, funcional, despliegue y desarrollo. El lenguaje de modelado recomendado era UML, excepto para la vista de contexto. A diferencia de la propuesta de Van Deursen et al. [15], se requerían vista funcional, de escenarios y de despliegue.

Puntos de variabilidad y extensión. A los alumnos se les requería identificar y documentar los puntos de variabilidad y extensión del sistema. Este punto no aparecía en la propuesta de Van Deursen et al. [15].

Estudio de los atributos de calidad del sistema. Los alumnos debían estudiar y documentar los atributos de calidad a partir del análisis de las métricas del código fuente del proyecto asignado, datos de issues de

GitHub o medidas de rendimiento observadas en el proyecto. El estudio de los atributos de calidad no aparecía en la propuesta de Van Deursen et al. [15].
Sugerencias de mejora. Los alumnos tenían que realizar propuestas de mejora donde podían abordar aspectos de diseño, código, documentación, o gestión de incidencias, entre otros.
Contribuciones al proyecto. De manera opcional, los alumnos podían realizar y documentar contribuciones concretas abordando cualquier dimensión del proyecto: mejora de la documentación, resolución de incidencias, mejoras en el código, etc.

El índice y documentación entregada por los alumnos en los cursos 20/21 y 21/22 aparece en el material.

4.4. Calificación

Para la evaluación de los proyectos de arquitectura se elaboró un documento con las rúbricas que se puede consultar en el material suplementario. La evaluación del proyecto de arquitectura se dividió fundamentalmente en dos partes: documentación y presentación del proyecto. Los criterios de evaluación de la *documentación* incluyen la completitud de cada uno de los apartados descritos en la sección anterior así como la calidad de la escritura y formato del documento. La evaluación de la *presentación* considera la claridad, alcance y originalidad de la exposición. La nota de la documentación y la presentación suponen el 85 % y 15 % de la nota total del proyecto, respectivamente.

5. Resultados

Para evaluar la propuesta llevamos a cabo una encuesta con los alumnos de los dos cursos involucrados en el estudio. Para ello se elaboró una encuesta, que se puede encontrar en el material suplementario, basada en un formulario web que los alumnos podían rellenar voluntariamente. En ambos cursos, la encuesta se realizó tras la evaluación del proyecto de arquitectura presentado en este artículo. Se obtuvieron 146 respuestas del curso 2020/21 y 112 del curso 2021/22.

La encuesta tenía como objetivo evaluar tres aspectos fundamentales: i) la satisfacción con la formación para realizar el proyecto, ii) la utilidad percibida por los alumnos para dichos elementos, es decir, hasta qué punto creen que cada aspecto del proyecto resulta útil en futuros proyectos de desarrollo software, y iii) la satisfacción general con la metodología aplicada. Todos estos aspectos han sido evaluados usando escalas de Likert a cinco niveles con simetría entorno a un valor neutral, como por ejemplo: Muy insatisfecho/a, Algo insatisfecho/a, Ni satisfecho/a ni insatisfecho/a, Algo satisfecho/a, Muy satisfecho/a. Además, en la encuesta se pedía a los alumnos que valorasen también sus conocimientos con el uso de GitHub antes y después de realizar el proyecto, y que indicaran si fueron capaces de ejecutar el proyecto a partir del repositorio y ver su funcionamiento en la práctica. Como último paso de la encuesta, se daba la oportunidad a los alumnos de aportar cualquier crítica, valoración, o sugerencia respecto del proyecto de arquitectura.

La figura 1 muestra un diagrama de barras apiladas divergentes con los resultados de satisfacción del alumnado con la formación recibida para elaborar cada uno de los apartados de la documentación de la arquitectura, su análisis, y la presentación. El grado de satisfacción con la formación es muy alto para todos los aspectos evaluados. Las respuestas positivas (formación adecuada o muy adecuada) tienen más del doble de porcentaje que las respuestas negativas (formación insuficiente o muy insuficiente), y las respuestas neutras en ningún caso superan un tercio de las respuestas emitidas. La única excepción es la formación respecto de la vista funcional (documentada mediante un diagrama UML de componentes), donde las respuestas negativas suponen un 43 % y las respuestas positivas un 30 %, indicando un aspecto a mejorar en futuras ediciones.

La figura 2 muestra los resultados sobre la utilidad percibida por los encuestados respecto de los distintos elementos del proyecto. Los resultados son extremadamente positivos, destacando especialmente los resultados para las vistas de contexto y escenarios de uso, las métricas y su análisis, y la propia herramienta SonarQube, para los que más del 75 % de las respuestas son positivas (valoradas como algo útil o muy útil). El aspecto del proyecto con menor utilidad percibida es la presentación, que aún así presenta un 59 % de respuestas positivas. Por todo ello consideramos que la metodología y la formación impartida han conseguido convencer a los futuros ingenieros de la utilidad e importancia que tiene el diseño arquitectónico y su documentación para el desarrollo de sistemas software.

La figura 3 muestra los resultados referidos a diversos aspectos generales de la metodología aplicada, como la claridad y nivel de detalle de los criterios de evaluación, los materiales proporcionados para la realización del proyecto, como la plantilla o el proyecto de referencia, y el nivel de complejidad del proyecto asignado. En todos los casos la satisfacción de los encuestados es muy alta, destacando especialmente el 50 % de alumnos que afirman estar muy satisfechos con la plantilla del proyecto y el 42 % que afirman estar muy satisfechos con el proyecto de referencia. Además en ambos casos se supera el 75 % de respuestas positivas.

El aspecto con el que los alumnos se sienten menos satisfechos es la complejidad de los proyectos asignados, con un 23 % de respuesta negativas, lo cual es razonable dado el reto que supone aplicar este método con alumnos de segundo curso. Sin embargo, esta per-

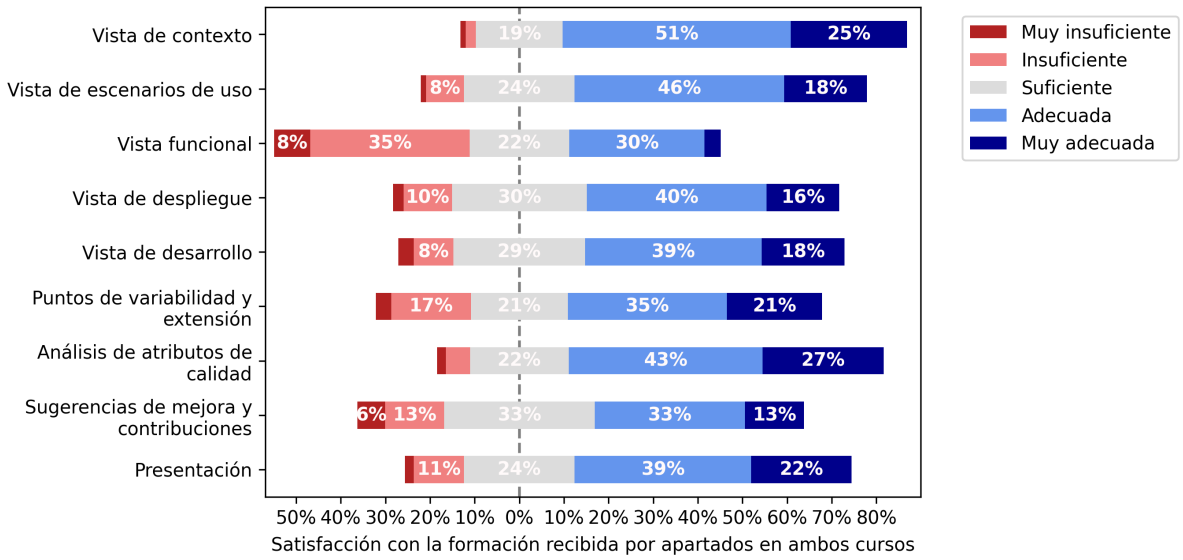

Figura 1: Satisfacción con la formación recibida para realizar cada apartado del proyecto

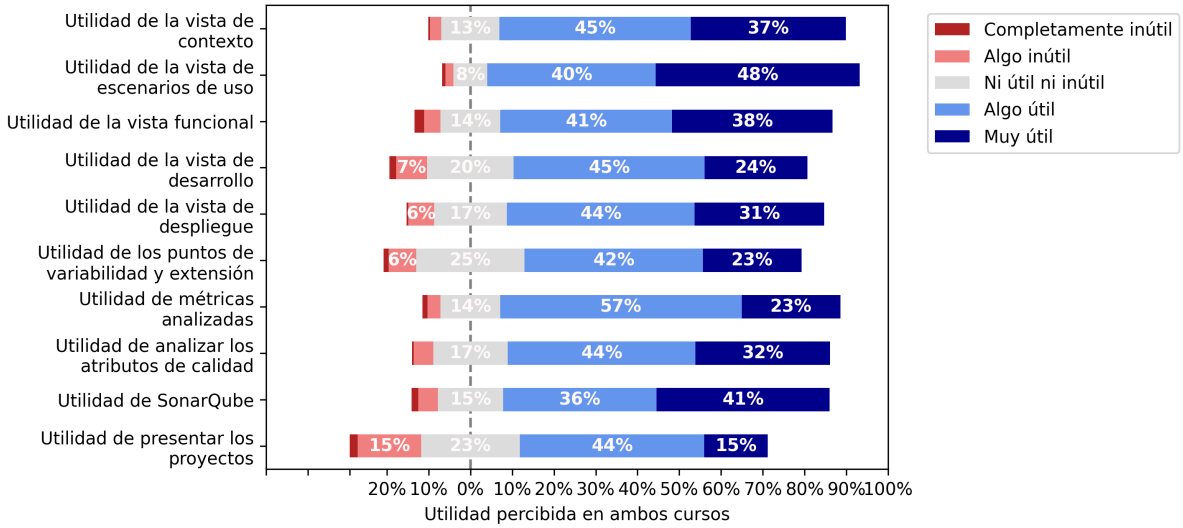

Figura 2: Utilidad de analizar los distintos aspectos de la metodología y el proyecto

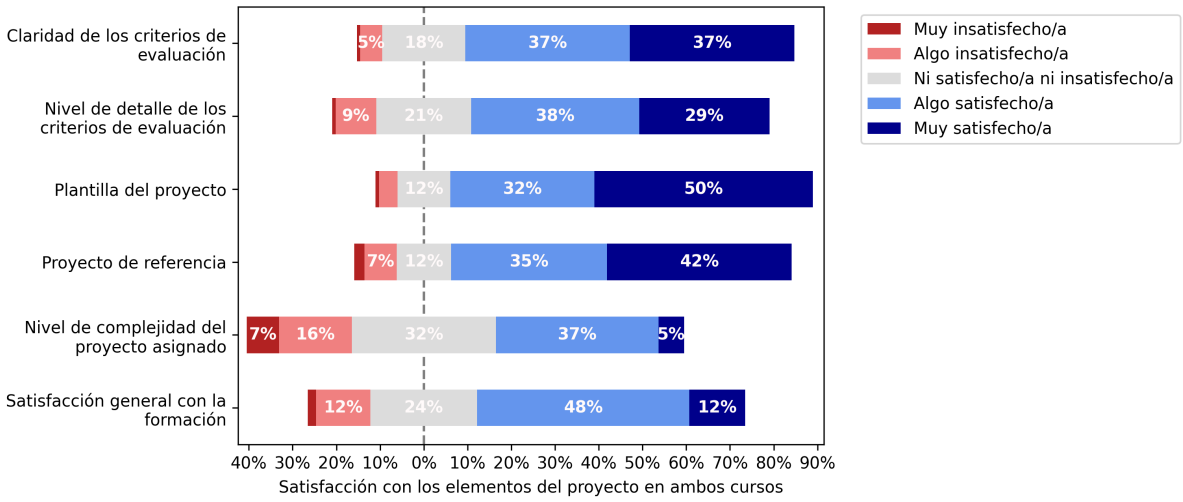

Figura 3: Satisfacción con los distintos aspectos de la metodología aplicada y el proyecto

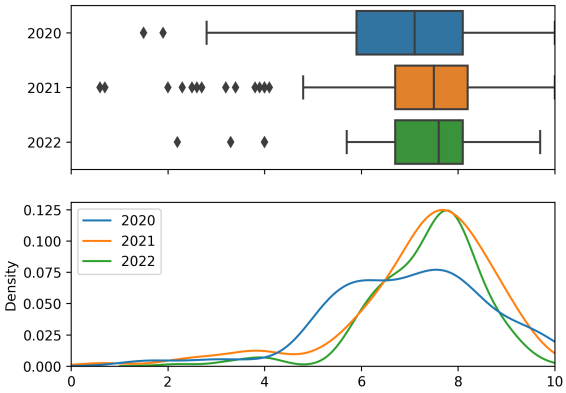

Figura 4: Diagrama de cajas y densidad de distribución de las notas de los cursos 2020-2022

cepción del alumnado no se refleja en los resultados académicos observados. La figura 4 muestra en su parte superior un diagrama de cajas con la distribución de las notas de los cursos 2020-22, donde puede apreciarse que la mediana de las notas es ligeramente superior en los cursos en los que se aplicó la metodología (2021 y 2022). Otro efecto interesante que se observa es que, aunque el límite superior de la caja es muy similar (cuartil superior de la distribución de notas, es decir, el valor tal que el 75 % de las notas son más bajas), el límite inferior de la caja y el bigote inferior sí presentan valores diferentes dependiendo de si se aplicó la nueva metodología o no. Lo que significa que, en general, los alumnos con notas más bajas tienden a tener mejores notas con la nueva metodología, mientras que los alumnos con notas más altas no se ven afectados por la misma. Este efecto se aprecia mejor en la parte inferior de la figura 4, que muestra la distribución de las notas por año, y además puede observarse una acusada disminución de notas obtenidas entre un 4 y un 5 en los cursos en los que se aplicó la metodología. Dado que las notas no siguen una distribución normal (lo comprobamos usando el test de Shapiro-Wilk), aplicamos el test de Mann-Whitney, obteniendo diferencias altamente significativas para las medias, concretamente con un p-value de $0,008$ para la comparación entre los cursos 2019/20 y 2020/21 y de $0{,}005$ para la comparación entre los cursos 2019/20 y 2021/22. Además, se obtuvo un valor d de Cohen para el tamaño del efecto de $0,168$ comparación entre los cursos 2019/20 y 2020/21 lo que está muy cerca del límite de los valores de referencia para un efecto pequeño. Para la comparación de las notas entre los cursos 2019/20 y 2021/22, el valor d de Cohen para el tamaño del efecto fue $0{,}266$, lo que se considera un tamaño del efecto pequeño. Por todo ello concluimos que la aplicación de la metodología ha tenido un efecto positivo sobre las notas de los alumnos aumentándolas ligeramente.

Respecto de los conocimientos de GitHub, un 79,5 % de los alumnos considera que han mejorado, por lo que consideramos que se han adquirido las competencias que nos planteamos como objetivo.

Como aspecto negativo, la encuesta reveló que solo un 54,3 % de los alumnos fue capaz de ejecutar el proyecto y probar su funcionamiento. Esto constituye uno de los aspectos a mejorar en la aplicación de la metodología, que abordaremos proporcionando material adicional de ayuda para la puesta en marcha de los proyectos. El análisis no arrojó diferencias significativas entre cursos académicos.

Finalmente, en cuanto a las sugerencias en texto libre, cabe destacar que la mayoría de las críticas iban destinadas a la diferencia de complejidad entre los ejemplos vistos en clase y los proyectos reales a analizar, la necesidad de más ejemplos de las distintas vistas, y un feedback más detallado para las presentaciones realizadas. Las diferencias de complejidad entre los proyectos es otro de los aspectos más criticados.

El material que contiene todos los datos usados para la evaluación de este trabajo así como los scripts de análisis usados para generar todas las figuras y resultados de esta sección se encuentra en el material online.

La evaluación realizada presenta algunas limitaciones como: i) la dificultad para medir objetivamente el impacto de la nueva metodología, puesto que el rendimiento académico y la satisfacción de los alumnos pueden no ser suficientes para evaluarlo completamente; y ii) la falta de control de algunas variables externas, como la formación previa de los alumnos, que pueden influir en los resultados de la evaluación y limitar la generalizabilidad de los resultados.

6. Lecciones aprendidas

Destacamos las siguientes lecciones aprendidas:
Método adecuado para etapas tempranas de aprendizaje. Nuestra experiencia confirma la eficacia del método, antes demostrada a nivel de máster, para la docencia de arquitectura del software en los primeros cursos de grado. En concreto, los alumnos tienen la oportunidad de ver aplicados en proyectos reales los conceptos vistos en teoría, ayudándoles a comprenderlos mejor.
Importancia de la selección de los proyectos. La cuidadosa selección de los proyectos de código abierto que serán objeto de estudio resulta primordial. Esto incluye una correcta evaluación de su nivel de dificultad, estado de madurez y nivel de soporte.
Aprendizaje sobre proyectos de código abierto. El método propuesto presenta beneficios adicionales de gran valor añadido para los estudiantes, en especial en los primeros cursos. Entre ellos, cabe destacar los conocimientos adquiridos sobre ecosistemas de desarrollo

(ej. GitHub, SonarCloud) y proyectos de código abierto (issues, pull requests, etc.).

Diferenciación entre teoría y práctica. Resulta difícil encontrar proyectos reales que implementen y documenten de manera prototípica los estilos y patrones arquitectónicos estudiados en clase. Esto supone a la vez un inconveniente y una ventaja: inconveniente porque dificulta ver la aplicación directa de los conceptos estudiados en clase; pero también es una ventaja porque muestra a los estudiantes la realidad del software y, además, la importancia de una buena documentación.

Dificultad para elevar el nivel de abstracción. Se identificó una notable dificultad por parte del alumnado para elevar el nivel de abstracción. Mientras que las vistas que suponían una descripción directa del proyecto resultaban relativamente sencillas (ej. vista de contexto), aquellas que suponían abstraerse del código y centrarse en las funcionalidades plantearon problemas a los alumnos. El caso más destacado fue la vista funcional, en la que los alumnos debían modelar un diagrama de componentes del sistema. Abordar este reto forma parte de nuestro trabajo futuro.

7. Conclusiones

Este artículo presenta nuestra experiencia aplicando un enfoque inverso para afrontar el aprendizaje de la arquitectura del software en etapas tempranas a través del análisis, evaluación y documentación de la arquitectura de proyectos de código abierto alojados en GitHub. Para evaluar este enfoque realizamos un total de 258 encuestas a estudiantes de dos cursos consecutivos. Los resultados del estudio muestran la idoneidad del método dando lugar a una docencia más práctica y efectiva, acompañada de un alto grado de satisfacción por parte del alumnado. Por último, se describen varias lecciones aprendidas y se proporciona un exhaustivo material suplementario que esperamos ayuden a otros docentes a replicarlo con éxito.

Referencias

[1] Len Bass, Paul Clements, y Rick Kazman. *Software Architecture In Practice*. Addison-Wesley Professional. 3rd edition, 01 2003.

[2] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, y Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley. 2ª edition, 2010.

[3] Cristobal Costa-Soria, Manuel Llavador, y Maria del Carmen Penades. An approach for teaching software engineering through reverse engineering. En *EAEEIE Conf.*, pp. 1–6, 06 2009.

[4] Cristóbal Costa-Soria y Jennifer Pérez. Teaching software architectures and aspect-oriented software development using open-source projects. En *SIGCSE-ITiCSE*, volumen 41, p. 385, 2009.

[5] Mohsen Dorodchi, Erfan Al-Hossami, Mohammad Nagahisarchoghaei, Rohit Diwadkar, y Aileen Benedict. Teaching an undergraduate software engineering course using active learning and open source projects. En , pp. 1–5, 10 2019.

[6] Patricia Lago y Hans Vliet. Teaching a course on software architecture. En *CSEET Conf.*, pp. 35 – 42, 05 2005.

[7] John Mcgregor, Felix Bachmann, Len Bass, Philip Bianco, y Mark Klein. Using an architecture reasoning tool to teach software architecture. En *CSEET Conf.*, pp. 275–282, 07 2007.

[8] Tomi Männistö, Juha Savolainen, y Varvana Myllärniemi. Teaching software architecture design. En *WICSA Conf.*, pp. 117 – 124, 03 2008.

[9] Naomi B. Robbins y Richard M. Heiberger. Plotting likert and other rating scales. En *2011 joint statistical meeting*, volumen 1. American Statistical Association, 2011.

[10] Claudia Rodrigues. Visar3d: an approach to software architecture teaching based on virtual and augmented reality. En *ICSE*, pp. 351–352, 2010.

[11] Nick Rozanski y Eóin Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional, 2005.

[12] Chandan R. Rupakheti y Stephen V. Chenoweth. Teaching software architecture to undergraduate students: An experience report. En *IEEE International Conference on Software Engineering*, volumen 2, pp. 445–454, 2015.

[13] Javier Troya, José A Parejo, Sergio Segura, Antonio Gámez-Díaz, Alfonso E Márquez-Chamorro, y Adela del Río-Ortega. Flipping laboratory sessions in a computer science course: An experience report. *IEEE Transactions on Education*, 64(2):139–146, 2020.

[14] Juan Urrego y Dario Correal. Archinotes: A tool for assisting software architecture courses. En *CSEE&T Conf.*, pp. 80–88, 05 2013.

[15] Arie Van Deursen, Maurício Aniche, Joop Aué, Rogier Slag, Michael De Jong, Alex Nederlof, y Eric Bouwers. A collaborative approach to teaching software architecture. En *ACM SIGCSE Technical Symposium*, p. 591–596, 2017.

[16] Alf Wang. Extensive evaluation of using a game project in a software architecture course. *TOCE*, 11:5, 02 2011.

[17] Gero Wedemann. Scrum as a method of teaching software architecture. En *ECSEE Conf.*, p. 108–112, 2018.