Actas de las Jenui, vol. 5. 2020.
Páginas: 21-28

# Fin de un viaje infinito: integración continua para prácticas de programación

Jesús Aransay y Jónathan Heras
Departamento de Matemáticas y Computación
Facultad de Ciencia y Tecnología, Universidad de La Rioja
{jesus-maria.aransay,jonathan.heras}@unirioja.es

## Resumen

En este trabajo presentamos un proyecto de tres años de duración para implantar herramientas y técnicas de Integración Continua en la gestión y corrección de prácticas de Programación Orientada a Objetos. El objetivo del proyecto era doble: por una parte, introducir a los estudiantes en el uso de herramientas que consideramos valiosas y relevantes para su formación; por otra parte, facilitar a los profesores la gestión y corrección de las prácticas de los estudiantes. A lo largo del proyecto hemos ido identificando y eligiendo diversas herramientas (desde IDEs de desarrollo hasta repositorios de código o servidores de integración continua) que hicieran más sencillo para nosotros y más útil para los estudiantes el trabajo desarrollado (es interesante decir que no se ha aumentado la carga de trabajo de los estudiantes). También hemos identificado carencias de las herramientas introducidas. Finalmente, hemos encontrado nuevas ideas y vías de trabajo para mejorar el proceso de enseñanza - aprendizaje de los estudiantes y aumentar su competencia tecnológica. A lo largo del trabajo presentaremos tanto el desarrollo de este proyecto, la valoración del mismo por parte de los estudiantes, nuestras reflexiones sobre sus ventajas y debilidades y también algunas líneas de trabajo futuro que nos planteamos partiendo desde la situación actual.

## Abstract

In this work, we introduce a long term project to implant Continuous Integration techniques and tools for managing and marking labs work of Object Oriented programming students. The goal of the project is twofold: first, we aim at introducing the students in the ordinary use of tools that we find relevant in their field (Computer Science); second, we aim at helping the teachers to manage and mark the students work. Along the project we have identified and chosen different tools (IDEs, code repositories, and continuous integration servers) that helped us in our work and also made more useful for the students their work (interestingly, we have introduced the tools without increasing their workload). We have also identified some shortages of the proposed tools. Finally, we have found some fresh ideas and roads to improve the learning process for the students as well as to improve their technological background. Along this work, we present our project, the students valuation of it, our particular assessment about its weaknesses and strengths, as well as some future work lines from the current situation.

## Palabras clave

Prácticas de programación, Gestores de versiones, Integración Continua, Testing.

## 1. Contexto

Este proyecto se ha desarrollado en la asignatura denominada “Programación Orientada a Objetos” (POO) (también conocida entre los estudiantes como “Programación III”) que pertenece a los planes de estudios del Grado en Ingeniería Informática y del Grado en Matemáticas de la Universidad de La Rioja, y es cursada de manera conjunta por estudiantes de ambas titulaciones. Antes de llegar a esta asignatura se han cursado dos asignaturas previas de programación cuyos contenidos incluyen entre otros tipos básicos de datos, variables, estructuras de control y Tipos Abstractos de Datos, y utilizan C++ como lenguaje de programación. La asignatura en que nos centramos (Programación III) presenta los conceptos propios de POO y tiene la particularidad de que toma como lenguaje de partida C++, pero también introduce Java, con el objetivo de establecer una comparación entre ambos lenguajes y porque Java va a ser el lenguaje usado en muchas de las asignaturas posteriores de, especialmente, el Grado en Ingeniería Informática.

Esta particularidad del uso de dos lenguajes de programación duplica el esfuerzo de gestión y corrección

de las prácticas de laboratorio de la asignatura, ya que muchos de los ejemplos y prácticas se implementan en ambos lenguajes. Como un dato adicional, la asignatura ha contado en cada uno los tres últimos cursos con aproximadamente 90 estudiantes matriculados, con un único "Grupo Grande" (3 créditos ECTS) y cuatro "Grupos Informáticos" (3 créditos ECTS repartidos en 15 sesiones de 2 horas). A lo largo de las 15 sesiones de Grupo Informático se completan 11 prácticas dentro de cada cual los estudiantes reciben un guion, que consiste en uno o varios diagramas UML que deben programar en C++ y en Java y el desarrollo de uno o varios clientes de esos diagramas de clases. Las prácticas y las entregas son individuales. Generalizando, podemos decir que en cada práctica cada estudiante genera entre 6 y 8 ficheros Java y otros tantos en C++. Hasta hace tres cursos, la entrega de esas prácticas se hacía por medio de empaquetados de ficheros que se entregaban a través del Aula Virtual, que usa el sistema Blackboard Collaborate.

## 2. Objetivos

Ante el contexto presentado, hace tres cursos los profesores de la asignatura nos planteamos la posibilidad de introducir herramientas que permitieran agilizar la gestión, tanto para estudiantes como para profesores, de las prácticas y su código asociado.

En la siguiente enumeración detallamos los objetivos buscados en este trabajo tanto para los estudiantes ("E") como para los profesores ("P"). Los etiquetamos para futura referencia:

- **O1E** Agilizar el proceso de gestión y entrega de las prácticas de los estudiantes.
- **O2E** Facilitar a los estudiantes herramientas que les permitan comprobar propiedades de su código.
- **O3E** Introducir a los estudiantes en el uso de herramientas de gestión de control de versiones y de Integración Continua.
- **O1P** Facilitar el proceso de gestión de las entregas de los estudiantes a los profesores.
- **O2P** Automatizar ciertas fases de la comprobación del código de los estudiantes.
- **O3P** Profundizar en los conocimientos de herramientas de gestión de código y de Integración Continua.

## 3. Análisis

Para intentar alcanzar los objetivos expuestos, partimos de plantear el uso de tres herramientas: uso de tests para comprobar propiedades del código generado, uso de herramientas de integración continua para automatizar comprobaciones y uso por parte de los estudiantes de sistemas de control de versiones de código. La cobertura de objetivos por parte de estas tres herramientas se proporciona en el Cuadro 1.

| | O1E | O2E | O3E | O1P | O2P | O3P |
|---|---|---|---|---|---|---|
| Tests | | ✓ | | | ✓ | |
| Integración continua | | | ✓ | | ✓ | ✓ |
| Control de versiones | ✓ | | ✓ | ✓ | | ✓ |

Cuadro 1: Cobertura de objetivos por parte de las herramientas empleadas en el curso

Nuestro segundo paso consistió en revisar la literatura sobre el uso de las anteriores herramientas en asignaturas de características similares a la nuestra. El uso de tests en prácticas de programación es un tema estudiado en la literatura, y cuyas ventajas y carencias son bien conocidas (ver por ejemplo [3–5]). Teniendo en cuenta que las prácticas de nuestra asignatura se desarrollan tanto en Java como en C++, dos elecciones naturales para la implementación y el uso de los tests parecían las librerías JUnit [16] y Catch [13].

El uso de servicios de integración continua en docencia en el año 2017, principio de nuestro experimento, había sido bastante menos analizado que el uso de tests, y solo pudimos encontrar ciertos trabajos seminales (ver por ejemplo [2, 6, 8, 11, 12, 14, 15]). En general, coincidían en señalar ciertos puntos comunes. El primero, la necesidad de usar sistemas de control de versiones como paso previo al uso de servicios de integración continua. Es necesario señalar que los estudiantes de nuestra asignatura no tienen conocimientos ni experiencia previa en el uso de sistemas de control de versiones. En segundo lugar, algunas de las elecciones naturales presentaban ciertas carencias. Por ejemplo, el servicio de integración continua Jenkins no dispone de un sistema de gestión de usuarios que permita separar los trabajos o proyectos de los estudiantes, con lo cual todos los usuarios pueden acceder al código y resultados de los demás. Para nuestro caso de uso, donde las prácticas son individuales, esta opción distaba de ser la ideal. En tercer lugar, la mayor parte de los trabajos citados hacen uso de los servicios de integración continua para enseñar conceptos de ingeniería del software o desarrollo dirigido por tests. En nuestro caso de uso particular, la asignatura de Programación III es anterior a Ingeniería del Software (que, de hecho, los estudiantes de Grado en Matemáticas ni siquiera cursan). Esto, y el hecho de que nuestra asignatura pertenezca al bloque de "Programación" nos obliga a intentar plantear soluciones "sencillas" y que los estudiantes puedan asimilar y adoptar rápidamente. Es importante señalar que, de los trabajos antes citados, el de Shaikh [14] ofrecía un entorno y soluciones tecnológicas que consideramos que cumplía todos nuestros requisitos. En el mismo se propone el uso de GitHub como sistema de control de versiones y de Travis CI

como servicio de integración continua con un número de estudiantes superior al que nosotros tenemos.

Finalmente, sobre el uso de sistemas de control de versiones para gestión de prácticas, de nuevo es interesante reseñar la diferencia entre las asignaturas cuyos contenidos incluyen el uso de gestores (ver por ejemplo los trabajos de López-Pellicer y otros [10] o de Vadillo y otros [18] en asignaturas sobre ingeniería del software) y aquellas en las que se pretenda que solo sea un medio para gestionar los trabajos o prácticas (como en el trabajo de Laadan y otros [9] para la enseñanza de sistemas operativos). En este segundo grupo de trabajos no suele quedar muy bien establecido cuál es el nivel de competencia previo de los estudiantes en el uso de sistemas de control de versiones. Nuestro caso es este segundo, en el que los estudiantes no tienen experiencia previa en el uso de sistemas de control de versiones y pretendemos que su uso sea lo más transparente posible para ellos.

# 4. Soluciones planteadas

A la hora de explicar las soluciones planteadas, es importante señalar que las mismas se han ido desplegando de manera gradual desde el curso 17/18 hasta el 19/20, donde podemos decir que las soluciones "definitivas" se han puesto ya en uso, con el fin de que la transición tanto para estudiantes como para profesores fuera lo más fiable y menos traumática posible.

En el curso 17/18, la primera novedad metodológica que incluimos fue el uso de tests, tanto por parte de los estudiantes como de los profesores. En cada práctica los estudiantes recibían un conjunto de tests que podían ejecutar para comprobar que su código funcionaba de manera adecuada al menos en los casos comprobados. Esos mismos tests eran usados por los profesores para comprobar funcionalidad básica del código entregado.

También en dicho curso, y ante la incertidumbre que nos generaba el uso de sistemas de control de versiones con los requisitos y limitaciones que hemos presentado en la sección anterior, lo que hicimos fue entregar a los estudiantes programas en Java que les permitían comprobar por sí mismos que la estructura de directorios y los nombres de ficheros que entregaban en cada práctica cumplían una determinada estructura propia de proyectos Java o C++. Estos programas tenían una doble ventaja; a los estudiantes les garantizaban que entregaban lo que los profesores les solicitábamos (como veremos en el análisis de datos de la sección siguiente, esto antes del curso 17/18 estaba lejos de ser la norma) y a los profesores nos permitían tomar el código de los estudiantes y ejecutarlo de manera automática.

Finalmente, también creamos un servidor en el que los estudiantes podían subir un fichero comprimido y el servidor hacía comprobaciones tanto de estructura (similar a las que hemos nombrado antes) como de ejecución de los tests. Los estudiantes recibían mensajes del servidor que les avisaban en caso de que la estructura o los tests hubieran producido algún fallo. Para este servidor tuvimos que adoptar algunas soluciones tecnológicas interesantes como el uso de Docker para hacer "ejecuciones aisladas" en el servidor.

La solución anterior, como mostramos en la Sección 5, mejoró la calidad de las entregas, pero todavía requería un esfuerzo importante tanto de estudiantes, para generar los "empaquetados" que debían entregar, descargar los programas de comprobación de la estructura, subirlos al servidor, y finalmente entregarlos a través del Aula Virtual, como de los profesores, que debíamos actualizar el servidor y los programas de comprobación, y repetir las comprobaciones que ya habían hecho los estudiantes (el servidor no disponía de un mecanismo de autenticación, y no quedaba constancia de las comprobaciones que hacía cada usuario).

En el curso 18/19 decidimos dar el salto a empezar a usar sistemas de control de versiones y servidores de integración continua. De las varias elecciones tecnológicas que hay disponibles, nos decantamos por usar GitHub (con licencia académica) y Travis CI (también con licencia académica). Las razones son varias: en primer lugar, era la elección de Shaikh [14] para un caso de uso bastante similar al nuestro, lo cual nos daba cierta seguridad. Además, estas opciones cumplían todos los requisitos que nosotros esperábamos obtener. Creando una "organization" de GitHub, obteniendo para la misma una licencia académica, y haciendo uso de la herramienta http://classroom.github.com, se nos permitía crear tareas con un repositorio de partida (en nuestro caso, con los tests), que al ser asignadas a los estudiantes daban lugar a repositorios privados con acceso solo para el estudiante y los administradores de la organización (es decir, los profesores). Además, los resultados de las operaciones llevadas a cabo en el servidor de integración continua (Travis CI) también eran accesibles únicamente para los usuarios con acceso al repositorio (esto, por ejemplo, no era posible conseguirlo con Jenkins CI). Otra opción interesante podría ser usar GitLab, aunque la desconocíamos en el momento de lanzar el proyecto. De los IDEs que usábamos en el curso 18/19 (Eclipse para Java, Code::Blocks para C++), Eclipse dispone de un plugin para Git que resulta bastante sencillo de usar, pero Code::Blocks no. Esto y la incertidumbre de que toda la infraestructura funcionara correctamente (hablamos de 90 estudiantes que solo en Java dieron lugar a más de 1500 repositorios, que sumados a los de C++ serían más de 2700) nos hizo adoptar la decisión de usar GitHub y Travis CI para las partes en Java, y seguir con la metodología del curso 17/18 para C++. Los resultados obtenidos, así como la comparación entre ambas metodologías,

están disponibles en la Sección 5. En la Sección 6 damos nuestra valoración sobre los resultados obtenidos. Sí que nos gustaría decir en esta sección que GitHub nos dio algunos problemas técnicos, en un porcentaje muy bajo de repositorios, a la hora de que los estudiantes se clonaran el repositorio de partida; el problema aparece de manera recurrente en los foros de usuarios de GitHub Classroom (ver por ejemplo https://github.com/education/classroom/issues/1537), pero en el curso 19/20 hemos sido capaces de evitarlo con un pequeño cambio que detallamos más adelante. Es importante decir que este tipo de fallos generan inseguridad en los profesores y desconfianza en los estudiantes. También nos parece importante reseñar que no dedicamos tiempo en clase a presentar ninguna de estas herramientas, sino que simplemente facilitamos a los estudiante dos vídeos de realización propia (de 9 y 3 minutos de duración), uno sobre cómo clonar una tarea, descargarla al IDE, y realizar un "commit + push", y otro sobre cómo generar una "release" a partir de un "commit"; cabe reseñar que a los estudiantes les pedimos que creen una "release" en su código de nombre "entrega", y de fecha anterior a la fecha de entrega, que para nosotros hace las veces de "versión entregada" de su código (esto les da a ellos la libertad de seguir introduciendo cambios sobre el mismo una vez entregado). Los vídeos están disponibles en https://github.com/POO1920/JENUI2020.

Finalmente, en el curso 19/20, y tras el buen funcionamiento (salvo pequeñas excepciones) obtenido con la solución GitHub + Travis CI, decidimos también adoptarla para las prácticas de C++, cambiando Code::Blocks por CLion (producto propietario de JetBrains, pero con licencia gratuita para uso académico) que sí dispone de una utilidad para trabajar con sistemas de control de versiones. Para evitar los problemas de clonado de los repositorios por parte de los estudiantes usamos la opción de generar repositorios que sean "*templates*"; de esta forma, los estudiantes no se clonan el repositorio de partida (y se copian toda su historia), sino que simplemente crean un repositorio nuevo a partir del último "commit" del repositorio de partida. Esto nos ha obligado a hacer cambios menores en la ejecución en Travis CI, ya que antes, para asegurar que los estudiantes no editaban los tests, recuperábamos los tests del propio repositorio de los estudiantes por medio de un "checkout" (de un "tag" especial que asignábamos de nombre "original" previo a su trabajo) y ahora debemos recuperar esos mismos tests del repositorio de partida (añadiendo un repositorio "remoto" en el código que se ejecuta en Travis CI, ver ejemplos en https://github.com/POO1920/JENUI2020). A cambio de esa mínima modificación hemos conseguido eliminar los problemas de clonado por parte de los estudiantes, que este año de nuevo empezaban a ser bastante habituales. De nuevo para el uso de CLion con GitHub facilitamos un vídeo a los estudiantes. Una de las lecciones aprendidas del curso 18/19, y que hemos tratado de aplicar en el curso 19/20, es que, si bien el uso de Travis CI es muy relevante para nosotros, pues nos evita compilar y ejecutar el código de los estudiantes y los tests, a los estudiantes no les resulta lo suficientemente amigable como para interpretrar los errores obtenidos (ver Valoración de los estudiantes, Sección 6), así que hemos tratado de dirigir menos su atención hacia esta herramienta, ya que además pueden obtener esos mismos resultados en local (en Eclipse o CLion).

Para reproducir esta experiencia hemos creado la página web https://github.com/POO1920/JENUI2020 donde se incluye una guía explicando de manera detallada el flujo de trabajo, repositorios de ejemplo y otros materiales que se han creado para este trabajo.

# 5. Calidad de las entregas

En esta sección valoramos la calidad de las entregas desde el curso 16/17 hasta el curso 19/20. Las prácticas en los cuatro cursos han sido similares, cubriendo los mismos conceptos. La diferencia radica en los flujos de trabajo y herramientas explicadas anteriormente.

A la hora de evaluar la calidad de las entregas hemos considerado tres bloques de prácticas (que eran los solicitados a los estudiantes) y tres parámetros: estructura, compilación y corrección. El primer parámetro, estructura, indica si los estudiantes han realizado la entrega siguiendo la estructura de ficheros pedida, es decir, que no falta ningún fichero, que no se incluyen ficheros adicionales, y que la organización de ficheros en carpetas es la solicitada. El segundo punto, compilación, sirve para medir el porcentaje de prácticas que tiene errores de compilación. Por último, corrección indica si el comportamiento de las entregas es el esperado. Durante el curso 16/17 la corrección se revisaba manualmente, mientras que en los otros cursos se ha hecho mediante tests. Los resultados obtenidos en los tres parámetros se presentan en la Figura 1.

Como se puede ver en la Figura 1, los cambios introducidos han permitido mejorar los resultados obtenidos con respecto a los tres parámetros medidos a lo largo de los cursos. Si nos centramos en la estructura de ficheros podemos ver la gran mejoría que se ha logrado. Durante el curso 16/17 eran pocos los estudiantes que realizaban las entregas con la estructura pedida, mientras que en el curso actual, la estructura de archivos entregada por los estudiantes siempre coincide con la solicitada. Esta tendencia ya se vio en los cursos anteriores. El servidor implementado en el curso 17/18, cuyo uso no era obligatorio, ya mejoró los resultados de estructura. El mismo servidor se empleó también

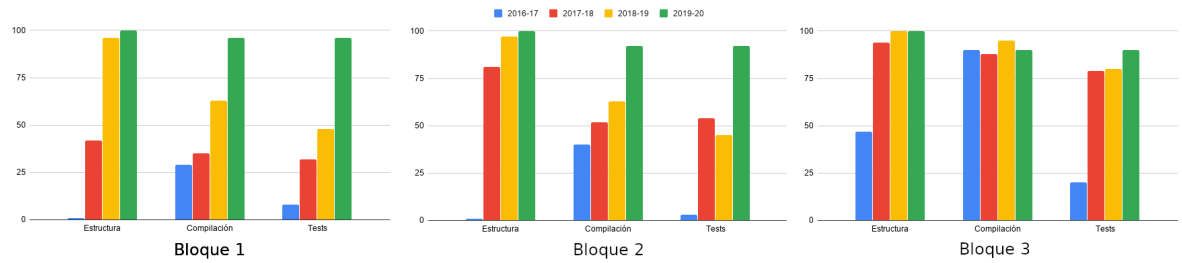


Figura 1: Porcentaje de entregas correctas.

para las entregas de C++ en el curso 18/19 (para las entregas de Java se utilizaban en el curso 18/19 los repositorios de GitHub, donde definimos una estructura inicial de ficheros). En el curso actual tanto para las entregas de Java como para las de C++ se utilizaban repositorios de GitHub donde definimos la estructura de ficheros.

La exigencia de los estudiantes respecto a compilación y corrección de sus entregas también ha aumentado. En el caso de la compilación, esto se debe, posiblemente, a que para ejecutar los tests se necesita que las prácticas compilen, y por lo tanto los estudiantes deben esforzarse en solucionar problemas que antes pasaban por alto. Del mismo modo, gracias a los tests, la corrección de las prácticas ha aumentado, ya que los tests fuerzan no solo a compilar los programas, sino también a ejecutarlos y comprobar que el comportamiento es el esperado. Durante este curso, y para todos los bloques de prácticas, al menos un 90 % de los estudiantes ha entregado prácticas que compilaban y superaban los tests, una diferencia considerable con respecto al curso 16/17 donde en menos de un 20 % de los casos las prácticas superaban estos filtros.

## 6. Valoración de los estudiantes

Con el propósito de conocer la opinión de los estudiantes con respecto a los objetivos O1E– O3E se elaboró en cada curso, utilizando Google Forms, una encuesta individual, anónima y voluntaria (estas encuestas están disponibles en https://github.com/POO1920/JENUI2020). En esta sección analizamos los resultados obtenidos en las mismas. Las encuestas consistían de una sección de información general, varias secciones de valoración que variaban dependiendo de las herramientas introducidas durante el curso, y una sección de comentarios. Las secciones de valoración consistían en una serie de afirmaciones que seguían una escala de Likert de 4 puntos que iban de 1 (muy en desacuerdo) a 4 (muy de acuerdo). Las preguntas eran del tipo: “Me ha resultado fácil usar los tests” o “El uso de GitHub me ayuda a gestionar mejor las prácticas”. Durante el curso 2017-18 participaron 42 estudiantes en la encuesta; en el 2018-19, 44 estudiantes; y, en el curso 2019-20, 70 estudiantes.

Empezamos analizando la valoración de los estudiantes sobre las mejoras introducidas para agilizar el proceso de gestión y entrega de prácticas (Objetivo O1E); en concreto, sobre el uso de Git y GitHub — estas herramientas fueron introducidas durante el curso pasado, por lo que solo fueron evaluadas en dicho curso y en el actual. En primer lugar, se preguntó a los estudiantes si preferían usar el Aula Virtual o repositorios de GitHub a la hora de iniciar y entregar las prácticas. En ambos casos y en ambos cursos, más del 80 % de los estudiantes indicaron que preferían el uso de GitHub. Además, más de un 90 % de los estudiantes opinan que les ayuda a gestionar las prácticas.

También se preguntó a los estudiantes del curso actual sobre su preferencia a la hora de gestionar el código. En concreto, se les pedía indicar si preferían usar GitHub para gestionar su código, o si por el contrario preferían emplear otros medios como USBs o sistemas de almacenamiento en la nube, como Dropbox o Google Drive. Un 93 % de los estudiantes respondió que prefería usar los repositorios de GitHub ante otras alternativas (un 3 % prefería usar USBs, un 3 % prefería usar sistemas de almacenamiento en la nube, y el resto prefería usar otros medios).

Para finalizar la valoración del primer objetivo, se preguntó a los estudiantes si creían que GitHub es una herramienta útil a la hora de comunicarse con el profesor para resolver dudas o cuestiones en el código. Un 83 % respondió afirmativamente. En realidad, las ventajas del uso de repositorios para la comunicación entre estudiantes y profesores es uno de los puntos que no éramos capaces de apreciar antes de empezar el proyecto, pero que a posteriori más valoramos.

Pasamos ahora a analizar la percepción de los estudiantes con respecto a los tests unitarios, que tenían como objetivo facilitar a los estudiantes herramientas que les permitan comprobar propiedades de su código, objetivo O2E. Dado que esta herramienta fue introducida durante el curso 2017-18, disponemos de encuestas de tres años, ver Figura 2. Lo primero que podemos notar en el gráfico de la Figura 2 es que, aunque la valoración

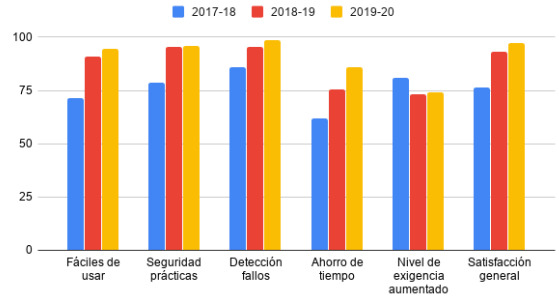


Figura 2: Valoración de los tests.

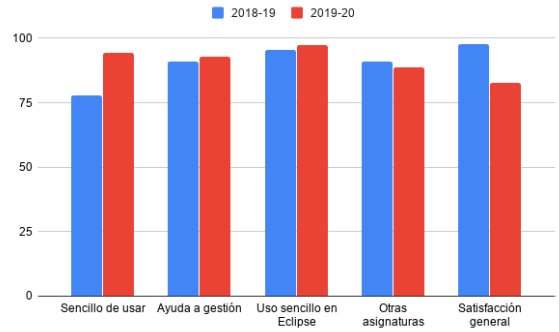


Figura 3: Valoración de Git y GitHub.

de los tests siempre ha sido positiva (desde los puntos de vista de usabilidad, seguridad con las prácticas, detección de fallos, y ahorro de tiempo), la opinión de los estudiantes con respecto a los mismos ha mejorado con el paso de los cursos. En la prueba piloto realizada durante el curso 2017-18 se ve en parte reflejada la inexperiencia del profesorado con el uso de tests, tanto a la hora de diseñarlos como en el momento de explicar su funcionamiento a los estudiantes. La destreza adquirida por el profesorado con el uso de tests durante estos cursos ha mejorado la visión de los estudiantes sobre los mismos. Una cuestión a destacar sobre los tests es que aumentan la exigencia de los estudiantes con las prácticas.

Por último, preguntamos a los estudiantes su opinión sobre las herramientas de gestión de control de versiones (Git y GitHub) y de integración continua (Travis CI) — ambas herramientas fueron introducidas en el curso 2018-19 con el objetivo O3E.

En el caso de Git y GitHub, la evaluación de estas herramientas es muy positiva (ver Figura 3). En ambos cursos más del 80 % de los estudiantes está satisfecho con ellas, y a maś del 90 % les gustaría usarlas en otras asignaturas. La mayoría de los estudiantes indica que usar Git y GitHub es sencillo, un 79 % el curso pasado y un 94.2 % en el actual. Esta diferencia puede deberse a varias razones. En primer lugar, los profesores tienen más experiencia con el uso de estas herramientas, y esto permite explicarlas mejor. El uso de un vídeo explicativo de cómo usar GitHub también ha sido valorado positivamente por los estudiantes. El hecho de que este curso tanto las prácticas de C++ como las de Java se hayan gestionado con GitHub posiblemente también haya mejorado la percepción del mismo. Además, hemos conseguido limar alguno de los problemas que encontramos el curso pasado, como el clonado inicial de tareas, con el uso de "*templates*". Por último, se preguntó a los estudiantes sobre la integración de GitHub con los IDEs Eclipse y CLion. Más del 95 % de los estudiante de ambos cursos respondieron que el uso de Git y GitHub desde Eclipse es sencillo; y un 84 % de los estudiantes dijeron lo mismo para CLion. Estos buenos resultados son debidos en parte a los vídeos que se produjeron para explicar dicha integración — todos los estudiantes que habían visto los vídeos indicaron que las explicaciones eran claras. La diferencia entre los resultados de Eclipse y CLion puede ser debida a que CLion fue introducido en el curso actual, por lo que los profesores no tienen la destreza suficiente con este IDE (en comparación con Eclipse, un entorno que se lleva utilizando desde hace más de 3 años). Finalmente, más del 80 % de los estudiantes opinaban que Git y GitHub les serían útiles en su futuro.

En el caso de Travis CI, su uso no era obligatorio, por lo que las siguientes valoraciones solo incluyen las respuestas de aquellos estudiantes que habían accedido a esta aplicación (un 57 % de los encuestados). Aunque un 71 % de los estudiantes está satisfecho con el uso de esta aplicación, solo el 55 % piensa que es sencilla de usar y que ofrece sugerencias útiles. A pesar de ello, al 73 % les gustaría disponer de herramientas similares en otras asignaturas.

Como conclusión a esta sección cabe resaltar que las mejoras introducidas han sido valoradas de manera muy positiva por los estudiantes. Aunque todavía hay margen de mejora, la experiencia adquirida durante estos cursos se ve también reflejada en la opinión de los estudiantes sobre las herramientas introducidas.

## 7. Valoración de los profesores y posibles mejoras

Pasamos ahora a valorar la perspectiva del profesorado con respecto a la consecución de los objetivos O1P– O3P, las carencias detectadas, y las posibles mejoras a introducir en cursos futuros.

En primer lugar, con las soluciones planteadas, y en concreto con el uso de Git y GitHub, se ha conseguido simplificar la gestión de las prácticas (objetivo O1P). En la actualidad, toda la gestión y almacenamiento de las prácticas se realiza a través de repositorios de GitHub que pertenecen a una "*organization*" común

(lo podríamos pensar como "la asignatura"), de tal forma que cada estudiante tiene acceso a sus repositorios, y los profesores, como administradores de la "*organization*", tienen acceso a todos los repositorios. El hecho de que los profesores tengan acceso a todos los repositorios permite automatizar tareas como la descarga del código de los estudiantes (anteriormente esto se tenía que hacer de manera manual práctica por práctica) o la modificación de tests (una tarea que requería modificar los ficheros, colgarlos en el aula virtual y pedir a los estudiantes que los descargaran y añadieran a sus proyectos; y que ahora se puede realizar de manera transparente para los estudiantes).

Un beneficio adicional obtenido gracias al uso de Git y GitHub es la mejora en la comunicación electrónica con los estudiantes. A la hora de resolver dudas vía correo electrónico nos encontrábamos de manera habitual con que los estudiantes enviaban solo algunos de los ficheros de código con los que tenían problema, o en el peor de los casos, capturas de pantalla. Esto hacía difícil resolver dudas y requería un cruce de correos hasta que se disponía del entorno necesario para reproducir los problemas del estudiante. Gracias al uso de GitHub, este problema se ha reducido ya que el profesorado tiene acceso al repositorio del estudiante. Para el futuro queda estudiar si otras funciones de GitHub pueden servir para mejorar la comunicación estudiante/profesor. Por ejemplo, para realizar aclaraciones mediante la creación de "*issues*" en los repositorios, o de comentarios en los "*commit*" de los estudiantes.

La posibilidad de introducir comentarios en el código de los estudiantes puede ser una manera adicional de proporcionar "*feedback*" a los estudiantes más allá del proporcionado por los tests, objetivo O2P. Como hemos ilustrado en la Sección 5, el aumento de la "calidad" (entendemos aquí por calidad que las entregas tengan la estructura adecuada y cumplan unos requisitos mínimos de compilación y paso de tests) de las entregas ha sido muy relevante, y esto es debido en gran parte al uso de tests que fuerzan a los estudiantes a ser más exigentes. También, el uso de tests unitarios facilita la labor de corrección de las prácticas ya que sirven para comprobar de manera automática si las entregas satisfacen ciertos requisitos. Además, gracias al uso de Travis CI es posible comprobar dichos requisitos sin necesidad de descargar o recompilar el código de los estudiantes. En el futuro planteamos integrar otras herramientas para automatizar la corrección como son MOSS [1], para detectar plagios, o funcionalidades de auto-evaluación proporcionadas por GitHub classroom (http://classroom.github.com).

Sin embargo existen varias limitaciones asociadas con los tests. Una limitación inherente al uso de tests unitarios es que con los mismos comprobamos el resultado de una acción, pero no cómo ha sido completada. En particular, en algunas prácticas, algunos métodos podían delegar en otros para completar acciones; independientemente de que los estudiantes los programaran delegando en los otros métodos o volvieran a reprogramar los métodos, los resultados de los tests eran satisfactorios; quizá el uso de programación orientada a aspectos [7] o de herramientas de análisis estático de código [17] podrían ayudar a realizar este tipo de comprobaciones de "más alto nivel". Otro efecto negativo asociado a los tests es la dependencia que se crea hacia los mismos; es decir, los estudiantes consideran que si sus programas superan los tests no necesitan hacer nada más. Además nos encontramos con la situación de que, tras cursar la asignatura, los estudiantes demandaban la disponibilidad de tests en otras asignaturas ya que sin ellos se sentían inseguros con su trabajo. Este problema se podría abordar pidiendo a los estudiantes que diseñen sus propios tests. En realidad, la elaboración y desarrollo de los tests unitarios es otro de los temas en que poco a poco hemos ido puliendo (y todavía nos falta por mejorar) pequeños errores o visiones simplistas que no siempre coinciden con las del desarrollador (en este caso, los estudiantes): por ejemplo, a veces asumimos en los tests que el código de los estudiantes va a producir una cierta salida por pantalla; si bien esto obliga a los estudiantes a cuidar estos aspectos, a veces el test les fuerza a hacer cierto uso de, por ejemplo mayúsculas o minúsculas, que sin duda peca de innecesariamente exigente (estos detalles se pueden corregir fácilmente, pero a veces es difícil preverlos en una primera versión de los tests). También hemos visto problemas parecidos con ciertas estructuras de datos donde comprobamos que los datos ocupan ciertas posiciones que no siempre coinciden con las que asignan los estudiantes. Si bien los estudiantes, en general, reconocen la utilidad de los tests, en casos como los anteriores (que no siempre son evitables) manifiestan cierta incomodidad con los mismos.

El desarrollo de tests es un concepto de Ingeniería del Software que se explica de manera específica en otra asignatura del Grado en Ingeniería Informática, pero los ejemplos proporcionados en esta asignatura facilitan su aprendizaje. Del mismo modo, las herramientas y técnicas utilizadas en la asignatura abren la puerta a profundizar en otras cuestiones relacionadas con la Ingeniería del Software como son los trabajos en grupo, o la gestión de versiones y ramas, que pueden ser introducidos en la asignatura mediante GitHub.

Por último destacar que la experiencia adquirida con estas herramientas y técnicas, objetivo O3P, ha sido el germen para su adopción en otras asignaturas del Grado en Ingeniería Informática donde se imparte la asignatura. En concreto, GitHub se utiliza para gestionar prácticas en otras 6 asignaturas del grado, tests unitarios en otras 3 asignaturas, y Travis CI en otras 2.

# 8. Conclusiones

En este trabajo se han presentado las conclusiones extraídas de una experiencia docente en la que se han introducido técnicas y métodos de integración continua a lo largo de tres cursos académicos. Los resultados han sido positivos tanto para estudiantes como profesores. Desde el punto de vista de los estudiantes se ha conseguido agilizar la gestión de sus prácticas, mejorar la calidad de las mismas, y aprender el uso de herramientas de gestión de control de versiones y de integración continua que les serán útiles en su vida profesional. Desde el punto de vista del profesorado se ha simplificado el trabajo de gestionar y comprobar las entregas de los estudiantes. Por último, lo aprendido ha podido ser extrapolado a otras asignaturas del grado, y servirá de base para introducir nuevas mejoras en el futuro. Terminamos con una breve reflexión personal que también aparecía en el título del artículo: es probable que hayamos llegado a donde pretendíamos llegar hace 4 cursos; es llamativo como, en cierto modo, sentimos que volvemos a estar al principio de otro camino.

# Referencias


[1] A. Aiken: *MOSS: A System for Detecting Software Similarity*, 2018. https://theory.stanford.edu/~aiken/moss/.

[2] J. Bowyer and J. Hughes: *Assessing undergraduate experience of continuous integration and test-driven development*. In *Proceedings of the 28th international conference on Software engineering*, pages 691–694. ACM, 2006.

[3] A. Cernuda del Río, M. García Rodríguez, N. García Fernández y M. González Rodríguez: *Uso de JUnit para evaluación en laboratorio de Estructuras de Datos*. En *Actas de las XX Jornadas de Enseñanza Universitaria de Informática (JENUI 2014)*, páginas 237–243, 2014.

[4] A. García Dopico, S. Rodríguez de la Fuente y F. J. Rosales García: *Automatización de prácticas en entornos masificados*. En *Actas de las IX Jornadas de Enseñanza Universitaria de Informática (JENUI 2003)*, páginas 119 – 126, 2003.

[5] P. P. Garrido Abenza: *Sistema de evaluación automatizada de prácticas para Tecnología de Computadores*. En *Actas de las XI Jornadas de Enseñanza Universitaria de Informática (JENUI 2005)*, páginas 138 – 144, 2005.

[6] S. Heckman, J. King, and M. Winters: *Automating Software Engineering Best Practices Using an Open Source Continuous Integration Framework*. In *Proceedings of the the 46th ACM Technical Symposium on Computer Science Education*, pages 677–677, 2015.

[7] G. Kiczales *et al.*: *Aspect-oriented programming*. In *Proceedings of the 11th European Conference on Object-Oriented Programming*, pages 220–242. Springer, 1997.

[8] C. Kästner: *Teaching software construction with travis ci*, 2014. https://www.cs.cmu.edu/~ckaestne/travis/.

[9] O. Laadan, J. Nieh, and N. Viennot: *Teaching operating systems using virtual appliances and distributed version control*. In *The 41st ACM technical symposium*, page 480. ACM, 2010.

[10] F. J. Lopez-Pellicer, R. Béjar, M. A. Latre, J. Nogueras-Iso y F. J. Zarazaga-Soria: *GitHub como herramienta docente*. En *Actas de las XXI Jornadas de Enseñanza Universitaria de Informática (JENUI 2015)*, páginas 66 – 73, 2015.

[11] R. T. Mason, W. Masters, and A. Stark: *Teaching Agile Development with DevOps in a Software Engineering and Database Technologies Practicum*. In *Proceedings of the 3rd International Conference on Higher Education Advances*, pages 1353–1362, 2017.

[12] C. Matthies, A. Treffer, and M. Uflacker: *Prof. CI: Employing continuous integration services and Github workflows to teach test-driven development*. In *Proceedings of the IEEE Frontiers in Education Conference*, pages 1–8. IEEE, 2017.

[13] P. Nash: *Catch2: C++ automated test cases in a header*, 2017. https://github.com/catchorg/Catch2.

[14] O. Shaikh: *Real-time feedback for students using continuous integration tools*, 2017. https://github.com/blog/2324-real-time-feedback-for-students-using-continuous-integration-tools.

[15] R. Sjodin and S. Barnes: *Teaching agile methodologies and devops/ci/cd in the classroom: concepts, techniques, modalities: panel discussion*. Journal of Computing Sciences in Colleges, 32(2):90–91, 2016.

[16] P. Tahchiev, F. Leme, V. Massol, and G. Gregory: *JUnit in Action*. Manning Publications, 2008.

[17] P. Trinidad, M. Resinas, S. Segura y A. Ruiz-Cortés: *Evaluación y seguimiento de trabajos en equipo de desarrollo desoftware a través de la calidad del código fuente*. En *Actas de las XVIII Jornadas de Enseñanza Universitaria de Informática (JENUI 2012)*, páginas 121–128, 2012.

[18] J. A. Vadillo, R. Arruabarrena y J. M. Blanco: *Uso de herramientas web en la asignatura Sistemas Web:facilitando el aprendizaje del alumnado y el proceso de evaluación*. En *Actas de las XXIII Jornadas de Enseñanza Universitaria de Informática (JENUI 2017)*, páginas 261 – 267, 2017.