Actas de las Jenui, vol. 4. 2019.
Páginas: 349-356

Uso de herramientas y técnicas DevOps para la gestión y corrección de 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 se presenta cómo diversas técnicas y herramientas utilizadas en metodologías DevOps han sido empleadas para la gestión y realización de prácticas de una asignatura de programación. En concreto, un gestor de tareas (`classroom.github.com`) ha sido usado para asignar los trabajos a los estudiantes, un gestor de versiones (GitHub) ha sido utilizado para que los estudiantes gestionen su código, tests unitarios (en Catch y JUnit) han sido usados para proporcionar *feedback* inmediato a los estudiantes, y el servidor de integración continua Travis CI, junto a los tests unitarios, ha sido usado para ayudar en la validación y corrección de prácticas. Esta aproximación para la gestión de las prácticas no solo facilita la tarea de corrección por parte de los profesores, sino que también sirve para inculcar a los estudiantes conceptos de Ingeniería del Software (como son la entrega continua y el uso de tests) e introducirles en el uso de herramientas que serán habituales en su vida profesional. Además de presentar la solución tecnológica usada para implantar este sistema de gestión de prácticas, en este trabajo se incluye una valoración basada en la observación de las mejoras en el aprendizaje de los estudiantes obtenidas a través del mismo, una valoración de los estudiantes del uso y aprendizaje obtenido de las herramientas, y se comparan los resultados con los obtenidos en cursos anteriores, donde se utilizaban otras herramientas.

Abstract

In this work we present how DevOps techniques and tools have been applied to manage labs in a programming course. Namely, a task manager (`classroom.github.com`) has been used to assign the tasks to the students, a version control system (GitHub) has been used by students to handle their code, unit tests (using Catch and JUnit) to provide immediate feedback to the students, and the continuous integration server Travis CI, together with unit tests to help in the validation and grading of assignments. This approach to handle assignments not only makes the correction task easier, but also serves to instill Software Engineering concepts (such as continuous delivery or the usage of tests) in the students. In this work, we present the technical solution to implant this lab handling system; and, in addition, we include an assessment based on the improvements of the students obtained thanks to this system. We also compare the results with previous years where other tools were employed.

Palabras clave

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

1. Motivación

En los últimos años conceptos de Ingeniería del Software como integración, entrega y despliegue continuos [4, 11] y en general metodologías y tecnologías asociadas con DevOps [12] se están imponiendo en la industria de desarrollo de software [18]. Esta familia de metodologías persigue varios fines, siendo uno de ellos el de incrementar la calidad de los productos (como el software) generados [11].

Algunas de las ideas que se aplican para incrementar la calidad incluyen:

- usar gestores y control de versiones en todo el código desarrollado;
- aplicar tests que permitan incrementar la calidad del producto;
- liberar código y ponerlo a disposición del cliente con asiduidad.

Otro de los fines de DevOps es aumentar la automatización de procesos. Un medio para ello es la definición de "*pipelines*" [12], que simplifiquen el proceso que va desde que se genera una nueva versión del código hasta que el mismo se pone a disposición de los

clientes finales; para ello, se recurre a software especializado que permite la gestión del proceso de integración continua.

Con las anteriores ideas en mente, nos enfrentábamos a una asignatura de programación de segundo curso de Grado en Ingeniería Informática y Grado de Matemáticas en la que por segundo año consecutivo el número de estudiantes superaba los 85. La asignatura no trata sobre Ingeniería del Software, y por supuesto entre sus objetivos no se encuentra enseñar DevOps a los estudiantes; más bien, la principal motivación por parte de los autores es agilizar el proceso de gestión de las prácticas de los estudiantes, incrementar la calidad de sus entregas, y todo ello a la vez que los mismos usan (de la manera más transparente posible) y se familiarizan con herramientas y técnicas propias de metodologías de desarrollo DevOps.

Simplemente por dar una idea del volumen de trabajo apuntado, los estudiantes deben entregar a lo largo del cuatrimestre 11 prácticas, cada una de ellas formada por uno, dos o tres ejemplos, y cada ejemplo puede estar en uno o dos lenguajes de programación; esto quiere decir que en algunas prácticas puede llegar a haber hasta seis "entregables", consistentes de varios ficheros describiendo varias clases o clientes de las mismas. Un estudiante que completa las prácticas entrega al final del semestre más de 200 ficheros de código.

El proceso de gestión de las entregas de los estudiantes hasta hace dos cursos consistía en descargar cada entrega (generalmente, cada una de ellas contenía de dos a cuatro prácticas, y era realizada por medio de un fichero comprimido) del Aula Virtual (disponible en la plataforma BlackBoard Learn [2]), descomprimir los ficheros, y comprobar las partes críticas del código (idealmente, se habría compilado y ejecutado cada uno de los ejemplos, además de revisado las partes críticas, pero la tarea era en la práctica inabordable).

En un proceso gradual, y con las motivaciones antes señaladas de aumentar la calidad de las entregas y simplificar su gestión a los profesores, el curso pasado se facilitó a los estudiantes con cada práctica un conjunto de tests que permitían a los mismos (y también a los profesores) comprobar que las prácticas cumplían unos ciertos criterios de calidad (de hecho, en algunos casos, estos criterios llegaban a ser bastante exigentes). Además, los profesores desarrollaron y pusieron a disposición de los estudiantes un servidor que permitía comprobar que los ficheros que entregaban eran los que se pedían en cada práctica, y que los mismos superaban los tests facilitados. Como mostraremos en la Sección 4, estas herramientas ya consiguieron aumentar ampliamente la calidad de las entregas con respecto al curso anterior.

Con el anterior objetivo ya cubierto, este curso se pretendía automatizar el anterior proceso de entrega de las prácticas (no en vano, las entregas seguían realizándose por medio de ficheros comprimidos a través del Aula Virtual), el proceso de comprobación de calidad de las mismas (que hasta la fecha se hacía por medio de scripts), y familiarizar a los estudiantes con herramientas de gestión de versiones y de integración continua ampliamente extendidas en la industria.

2. Objetivos

Los objetivos planteados en este trabajo para este curso son:

- al menos, mantener la calidad de las entregas alcanzada el curso anterior;
- simplificar el proceso de entrega de las prácticas por parte de los estudiantes;
- automatizar la comprobación de la calidad de las prácticas (en este caso, de comprobación de la compilación y de superación de los tests) por parte de los profesores;
- familiarizar a los estudiantes con el uso de herramientas DevOps (en nuestro caso, gestores de versiones y servidores de integración continua).

Otras consecuencias que planteamos inicialmente que podían tener nuestro trabajo incluyen:

- mejorar la comunicación "estudiante - profesor" a la hora de resolver dudas;
- aumentar la información de que dispone el profesor a la hora de evaluar el trabajo de los estudiantes;
- simplificar a los estudiantes el proceso de gestión de su código.

3. Metodología utilizada

3.1. Flujo de trabajo para los estudiantes

Pasamos a describir el flujo de trabajo que definimos para la gestión de las prácticas en este curso, así como las soluciones tecnológicas que nos han permitido implementarlo. En la asignatura se realizan prácticas en dos lenguajes: C++ y Java. Para poder tener una mejor comparativa de la utilidad de las herramientas, definimos dos flujos de trabajo diferentes, uno para las prácticas en C++ y otro para las de Java. El flujo de trabajo de C++ es el que ya usamos en el curso anterior para ambos lenguajes (Java y C++); el flujo de trabajo de Java es el que proponemos para integrar herramientas DevOps.

1. Para las prácticas de C++, en el Aula Virtual se pone a disposición de los estudiantes un fichero comprimido con una suite de tests (desarrollados

con Catch [17]) que el estudiante debe descargar e incluir en su proyecto. Una vez completada la práctica, el estudiante dispone de un servidor público desarrollado por los autores, al cual debe subir un fichero comprimido; el servidor genera un informe al estudiante en el que se detalla si la estructura de directorios y ficheros del proyecto es correcta y si se pasan los tests entregados. Este fichero comprimido constituye el entregable de los estudiantes, que deben subir al Aula Virtual.

2. Para las prácticas de Java, el flujo de trabajo para el estudiante comienza por aceptar una "tarea" generada por los profesores (podríamos definir una tarea como un repositorio de GitHub, en nuestro caso con los ficheros de tests, generados con JUnit [21], más una fecha de entrega) en `classroom.github.com`. Cuando el estudiante acepta la tarea (clickando en un enlace generado automáticamente para cada tarea) se hace un "*fork*" del repositorio. A partir de ahí comienza a trabajar cada uno en su repositorio privado. Los repositorios de los estudiantes están automáticamente integrados con Travis CI (un servidor de integración continua) [14], donde se realiza automáticamente el proceso de construcción de la práctica que hemos definido los profesores en un fichero ".travis.yml", que también forma parte de los repositorios. Cuando el estudiante considera que ha completado la práctica, le pedimos que genere un "*tag*" de nombre "entrega", el cual consideramos como su versión entregada del trabajo (aunque incidiremos en ello, no está de más notar que en este flujo de trabajo la dependencia del Aula Virtual es prácticamente nula, lo cual valoramos en este caso concreto positivamente).

3.2. Flujo de trabajo para los profesores

Pasamos a detallar la carga de trabajo adicional que para el profesor suponen los dos flujos de trabajo detallados en la sección previa. Dejamos para la Sección 5 una valoración (basada en encuestas, en el caso de los estudiantes, y subjetiva, en caso de los profesores) de cada uno de ellos.

El flujo de trabajo 1 requiere de los siguientes pasos:

- En primer lugar, los profesores deben disponer del servidor en que se realice la comprobación de las prácticas. En nuestro caso, este servidor estaba disponible del curso anterior, pero para cada bloque de prácticas se debe generar un fichero donde se especifiquen las mismas.
- En segundo lugar, para cada práctica se debe generar el conjunto de tests, así como ponerlos a disposición de los estudiantes en el Aula Virtual.
- Finalmente, a la hora de gestionar las entregas, el profesor debe habilitar la entrega en el Aula Virtual, descargar las entregas de todos los estudiantes y comprobar las mismas. Esto puede requerir repetir su proceso de construcción en el servidor (los estudiantes se supone que ya lo han realizado, aunque no están obligados a ello), o descomprimir las entregas para comprobar el código.

El flujo de trabajo 2 requiere de los siguientes pasos:

- En primer lugar, tanto para GitHub como para Travis CI solicitamos licencias académicas; en el primer caso, esto nos daba la posibilidad de disponer de repositorios privados, lo cual considerábamos relevante. En el segundo caso, era para poder integrar los repositorios privados en Travis CI (asumimos que soluciones parecidas se podrían plantear con otras herramientas de gestión de versiones y servidores de integración continua, aunque no sabemos si completamente idénticas; por ejemplo, descartamos Jenkins CI como servidor de integración continua porque cualquier usuario podía acceder a los procesos de ejecución de otros usuarios; también Travis CI es una herramienta que únicamente permite integrarse, hasta donde sabemos, con GitHub; quizás con GitLab[1], que dispone herramientas propias de integración continua, se podría implantar una solución similar, aunque en este caso GitLab no dispone de la noción de "organization" que nos permite acceder y gestionar simultáneamente todos los repositorios de la asignatura, sino de un concepto comparable conocido como "Group").
- La generación de cada tarea requiere por parte del profesor de generar un repositorio "de partida" (podría ser vacío, en nuestro caso incluye los tests y el fichero de configuración ".travis.yml") y, si se desea, una fecha límite. Esto genera un enlace que se debe facilitar a los estudiantes.
- Durante la elaboración de la práctica por el estudiante, el estudiante y el profesor pueden acceder al repositorio y realizar cambios en el mismo, lo cual consideramos que facilita la comunicación y el proceso enseñanza - aprendizaje.
- Finalmente, en la parte de gestión de las entregas, a partir de la fecha de entrega el profesor puede simplemente acceder a Travis CI, ver el resultado de la construcción del repositorio, acceder al repositorio (a través de la web) y comprobar el código del estudiante, o descargar los repositorios y trabajar con ellos de forma convencional.

[1] `about.gitlab.com/`

Curso	Estructura	Bloque 01 Compilación	Tests/Correctas
16/17	0 % (0/58)	29 % (17/58)	8 % (5/58)
17/18	42 % (36/85)	35 % (30/85)	32 % (27/85)
18/19	96 % (82/85)	63 % (54/85)	48 % (41/85)
Curso	Estructura	Bloque 02 Compilación	Tests/Correctas
16/17	1 % (1/53)	40 % (21/53)	3 % (2/53)
17/18	81 % (64/79)	52 % (41/79)	54 % (43/79)
18/19	97 % (82/84)	63 % (53/84)	45 % (38/84)
Curso	Estructura	Bloque 03 Compilación	Tests/Correctas
16/17	47 % (23/49)	90 % (44/49)	20 % (10/49)
17/18	94 % (72/76)	88 % (67/76)	79 % (60/76)
18/19	100 % (84/84)	95 % (80/84)	80 % (68/84)

Cuadro 1: Porcentaje de entregas correctas

4. Comparativa de la calidad en las entregas

En esta sección valoramos la calidad de las entregas en los cursos 16/17, 17/18, y 18/19. Las prácticas en los tres cursos han sido similares, cubriendo los mismos conceptos. La diferencia radica en los flujos de trabajo y herramientas explicadas en la sección anterior.

A la hora de evaluar la calidad de las entregas hemos considerado 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 cursos 17/18 y 18/19 se ha hecho mediante tests. Los resultados obtenidos con respecto a los tres parámetros se presentan en el Cuadro 1.

Como se puede ver en el Cuadro 1, los cambios introducidos han permitido mejorar, o al menos mantener, 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 en el curso 18/19. Durante el curso 16/17 eran pocos los estudiantes que realizaban las entregas con la estructura pedida, esto se ha invertido en el curso 18/19 y ahora son pocos los estudiantes cuya estructura no coincide con la solicitada. Esta tendencia ya se vio en el curso 17/18, y la razón se debía al servidor de validación que avisaba a los estudiantes de cuando su entrega no satisfacía el criterio de estructura. Durante el curso 18/19 se ha seguido utilizando dicho servidor para las entregas en C++, mientras que para las entregas de Java se utilizaban los repositorios de GitHub, en los cuales se define una estructura inicial de ficheros. Debido a que el uso del servidor de validación no era obligatorio todavía hemos encontrado entregas en C++ que no satisfacían la estructura; mientras que al ser obligatorio el uso de repositorios de GitHub para Java, todas las entregas en este lenguaje sí que satisfacían el requisito de estructura.

La exigencia de los estudiantes respecto a compilación y corrección también ha mejorado. 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.

Sin embargo, hay un efecto negativo asociado a los tests y es la dependencia que se crea hacia los mismos; es decir, los estudiantes consideran que si sus programas superan los tests, entonces no necesitan hacer nada más. Esto se pudo ver en el último bloque de entregas donde se introdujo un ejercicio control que no tenía tests asociados. El ejercicio consistía en utilizar un código proporcionado para hacer una llamada a una API Rest. Lo único que tenían que hacer los estudiantes era generar una clave para la API e incluir dicha clave en el código proporcionado. De los 84 estudiantes que realizaron la entrega con ese ejercicio, 37 de ellos (el 45 %) hizo mal este ejercicio. Esto contrasta con el resto de ejercicios de la entrega, para los cuales un 80 % dio una solución correcta.

5. Valoración de los estudiantes

Con el propósito de conocer la opinión de los estudiantes con respecto al uso de los tests y a las diversas herramientas puestas a su disposición se elaboró, utilizando Google Forms, una encuesta individual, anónima y voluntaria. En esta sección analizamos los resultados obtenidos a través de dicha encuesta. La encuesta consistía de 9 secciones (información general, valoración de los tests, valoración aplicación estructura de ficheros, valoración aplicación web, valoración de Git y GitHub, valoración de Travis, preferencia uso Aula Virtual y repositorios, y 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”. Por último la sección de comentarios permitía a los estudiantes introducir comentarios adi-

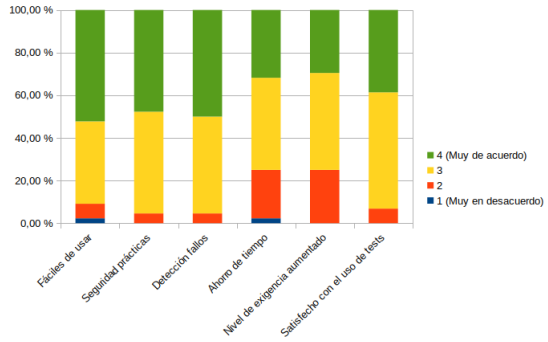

Figura 1: Valoración de los tests

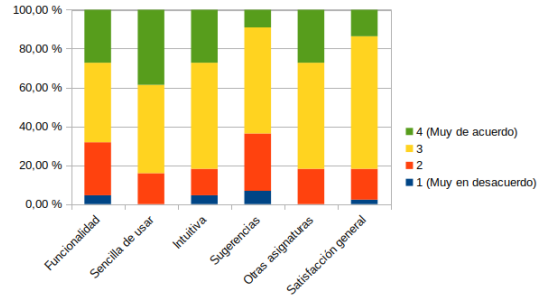

Figura 2: Valoración de la aplicación web

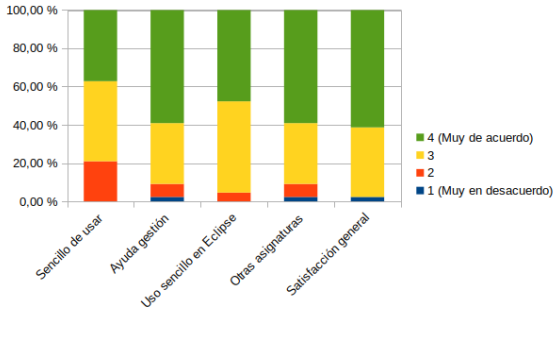

Figura 3: Valoración de Git y GitHub

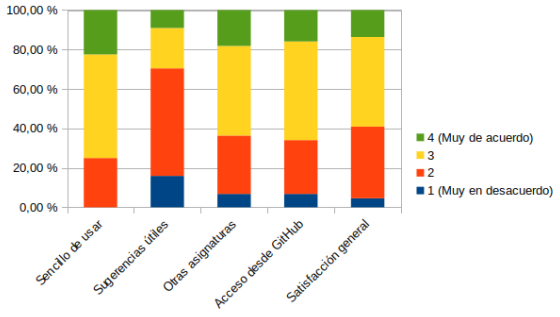

Figura 4: Valoración de Travis CI

cionales. En la encuesta participaron 44 estudiantes.

Empezamos analizando la opinión de los estudiantes sobre el uso de tests (ver Figura 1). La valoración de los tests es muy positiva en todos los aspectos evaluados, y en general un 93 % de los estudiantes están satisfechos con el uso de los mismos. Considerando la utilidad de los mismos, el 95 % de los estudiantes afirman que los tests les han servido para detectar fallos, el mismo porcentaje opina que los tests dan más seguridad a la hora de evaluar las prácticas, y un 75 % considera que el nivel de exigencia con respecto a las prácticas ha aumentado gracias a los tests. Por último, desde el punto de vista de la usabilidad, un 90 % de los estudiantes considera que son fáciles de usar y un 75 % opina que el uso de tests les ha ahorrado tiempo.

El servidor web para la validación de las entregas de C++ también recibe valoraciones positivas (ver figura 2). Un 81 % de los estudiantes está satisfecho con el servidor web. Además la mayoría de los estudiantes indica que es sencillo de usar (un 84 %) e intuitivo (un 81 %). De nuevo, al 81 % de los estudiantes les gustaría disponer de una herramienta similar en otras asignaturas. Una cuestión a mejorar son las sugerencias ante errores proporcionadas por el servidor ya que solo el 63 % las considera útiles.

La valoración del uso de Git y GitHub para gestionar las prácticas es también muy positiva (ver figura 3). Un 97 % de los estudiantes está satisfecho con dicha aplicación, y un 90 % piensa que ayuda a gestionar las prácticas. Además la mayoría de los estudiantes indica que usar Git y GitHub es sencillo (un 79 %) y que su uso desde el IDE Eclipse es también sencillo (un 95 %). Por último, al 90 % de los estudiantes les gustaría que se usaran Git y GitHub en otras asignaturas.

Al contrario que en los casos anteriores, la valoración de Travis CI no es tan positiva, ya que como se puede ver en la figura 4, solo un 59 % está satisfecho con el uso de esta aplicación. Aunque el 75 % piensa que es sencillo de usar, más del 70 % de los estudiantes piensan que las sugerencias que ofrece no les son útiles. A pesar de ello, al 63 % les gustaría disponer de herramientas similares en otras asignaturas.

Por último, se preguntó a los estudiantes sobre si preferían usar el Aula Virtual o los repositorios de GitHub a la hora de iniciar y entregar las prácticas. En ambos casos, más del 80 % de los estudiantes indicaron que preferían el uso de GitHub, ver Figura 5.

6. Discusión

Pasamos a enumerar algunas de las ventajas e inconvenientes de los flujos de trabajo que presentamos en la Sección 3; incluimos también en la comparación nuestro flujo previo de trabajo, en que no se usaban tests ni tampoco ninguna forma de comprobación de las entregas, y nuestras impresiones sobre las valoraciones de

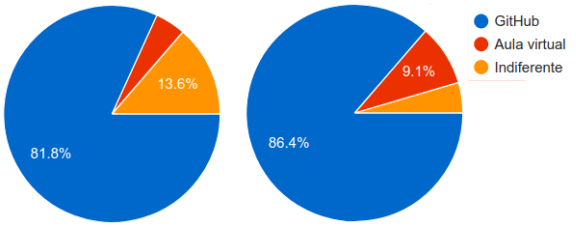

Figura 5: Valoración sobre el uso de GitHub o Aula Virtual para iniciar (derecha) y entregar (izquierda) las prácticas

los estudiantes.

- Como hemos ilustrado en la Sección 4, 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 con los flujos de trabajo 1 y 2 ha sido muy relevante.
- Entre los flujos de trabajo 1 y 2, y siendo que ambos requieren ciertos pasos manuales (asumimos que algunos son inevitables) por parte de los estudiantes, entendemos que el segundo es más sencillo y contiene mayor automatización, ya que comenzar una práctica para los estudiantes exige solo acceder a una url, y la entrega solo requiere etiquetar un punto de su repositorio. El profesor puede disponer del resultado de la comprobación del código del estudiante de manera automática (accediendo a su cuenta de Travis CI). No está de más aquí señalar algunas limitaciones o fallos comunes que hemos ido detectando con los tests. Una limitación inherente al uso de tests 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 necesarios, los resultados de los tests eran satisfactorios; quizá el uso de programación orientada a aspectos [13] nos podría ayudar a realizar este tipo de comprobaciones de "más alto nivel". Otro fallo que cometimos, por el cual algunos test erróneos no aparecían como tales a los estudiantes, tuvo que ver con la gestión de excepciones; para poder acceder a ciertos métodos de los estudiantes tuvimos que usar librerías de reflexión en Java [5], que podían generar ciertas excepciones si, por ejemplo, el método correspondiente no existía. Nuestra gestión de las excepciones no completaba ninguna acción, y por tanto los estudiantes no tuvieron ninguna información útil de esos tests (una vez detectado el fallo, lo solventamos lanzando un error de los tests, que los estudiantes veían como un fallo del test correspondiente). También al final de la Sección 4 hemos apuntado el resultado del ejercicio de control sobre los tests.
- El flujo de trabajo 2 "libera" a los estudiantes, y a los profesores, de realizar la gestión y almacenamiento de los ficheros, al encontrarse los mismos en un repositorio. Todos los repositorios pertenece 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 de todos los estudiantes. Esto permite agilizar la comunicación con los estudiantes a través de medios electrónicos, y resuelve los habituales problemas por parte de los estudiantes de pérdida de información.
- En el flujo de trabajo 2, el hecho de que los profesores sean administradores y puedan acceder a todos los repositorios de los estudiantes también nos permite automatizar ciertas tareas que en el flujo de trabajo 1 resultaban más complicadas. Por ejemplo, si es necesario incluir un cambio en los tests, el flujo de trabajo 1 requiere que generemos un nuevo fichero comprimido, lo pongamos en el Aula Virtual, y pidamos a todos los estudiantes que lo descarguen y lo incluyan en sus proyectos; con el flujo de trabajo 2, usando la API de GitHub, basta con descargar cada repositorio, realizar el cambio, y volver a subirlo (de manera transparente al estudiante, siempre que no se encuentre trabajando en el mismo). De hecho, en una de las primeras prácticas, y por ingenuidad nuestra, cometimos el error de ejecutar en el servidor de Travis CI lecturas desde teclado; estas lecturas no se completaban, y el proceso se quedaba detenido durante 10 minutos, dando un fallo de construcción del repositorio; la forma en que solventamos este error es como hemos mencionado antes (descargando y modificando con un script todos los repositorios de todos los estudiantes). Operaciones similares se pueden realizar para descargar los repositorios de todos los estudiantes, comprobar que cumplen el plazo de entrega, invocar a herramientas externas para medir la similitud del software, etc.
- El flujo de trabajo 2 requiere del uso de herramientas (repositorios de código y servidores de integración continua) que eran desconocidos para la gran mayoría de nuestros estudiantes. Siendo que entre los objetivos de nuestro curso no figura el enseñar a usar las herramientas, lo que hicimos

fue crear unos breves vídeos (entre 2 y 8 minutos) donde se describía el uso de cada uno de ellos. Si bien no hemos incluido el uso de los vídeos en nuestras encuestas de valoración, nuestra percepción es que los estudiantes las integraron rápidamente en su conjunto de herramientas (en menos de un mes de curso ya habían realizado su primera entrega de manera satisfactoria), aunque también fuimos detectando carencias puntuales en su comprensión (que no en su uso). Por ejemplo, algunos estudiantes aspiraban a "arreglar" un "commit" ya realizado cuyo proceso de construcción había dado resultado negativo, en lugar de simplemente hacer los cambios oportunos y realizar un nuevo "commit", corregido y cuya construcción fuera exitosa; como identificamos estas carencias como puntuales, las fuimos aclarando de manera individualizada.

- En términos generales, se puede afirmar que el flujo de trabajo 2 está formado por herramientas propias de las metodologías DevOps que se están aplicando en la industria, y cuyo uso está ampliamente extendido (y todo parece indicar que seguirá creciendo); en nuestro caso, creemos que nuestro trabajo confirma que estas mismas soluciones también se pueden aplicar en el ámbito académico de manera satisfactoria, mejorando la calidad de los trabajos entregados, y además reduciendo la cantidad de trabajo que estudiantes y profesores deben invertir. Incluimos aquí algunos números para ilustrar el volumen de trabajo del que hablamos (conviene recordar que este curso tenemos 86 estudiantes, y todos los que han completado las prácticas han generado 17 repositorios): el número total de repositorios que se han generado en GitHub es de 1490 (se incluyen los repositorios de partida de cada práctica y los generados por los profesores para completar ellos mismos las prácticas); el número de ficheros de código Java 16270, con 869517 líneas de código, 257457 líneas en blanco y 48407 de comentarios; también hay 1490 ficheros Maven (donde se detalla qué fases incluimos en la construcción del proyecto) y 1490 ficheros YAML, correspondientes a los ficheros ".travis.yml"; el tiempo de CPU que ha usado Travis CI para construir los repositorios de cada estudiante (que haya completado todas las prácticas) es de al menos 13 minutos (este tiempo se podría comparar con el tiempo que deberían invertir los profesores para simplemente compilar y ejecutar las prácticas); el número de commits que ha realizado cada estudiante entre todos sus repositorios (basado en el número de "builds" que se han realizado en Travis CI) es bastante dispar (aunque nuestra recomendación era que realizaran "commits" frecuentes, cada vez que completaran un ejercicio o superaran algún test), pero en todos los estudiantes que lo hemos comprobado es superior a 100 (algunos estudiantes han hecho más de 200).

7. Trabajos relacionados

Existen varios trabajos que fomentan el uso de técnicas y herramientas utilizadas en metodologías DevOps dentro de las aulas de Informática. En primer lugar, las ventajas y debilidades de usar tests unitarios para la autoevaluación de código por parte de los estudiantes, y también por parte de los profesores, han sido tratados en múltiples trabajos previos [3, 6–8]. En dichos trabajos, al igual que en el nuestro, se ha visto que los estudiantes valoran de manera positiva la introducción de los tests unitarios en las prácticas, y que además se eleva el nivel de autoexigencia de los estudiantes.

También existen varios trabajos que fomentan el uso de herramientas de integración continua dentro de las aulas. Dichos trabajos pueden separarse en dos categorías: los que usan las herramientas y técnicas de DevOps como un medio (como es nuestro caso) [1, 10, 14, 19]; y aquellos en el que el fin es enseñar esta metodología de desarrollo de software [15, 16, 20]. Debido a su simplicidad de uso y configuración, y a las posibilidades de gestión de repositorios que ofrece, la mayoría de los trabajos han utilizado como servidor de integración continua Travis CI. También nos gustaría destacar el trabajo de Heckmann y otros en Jenkins [9], ya que más allá del testing, también integra otras herramientas para evaluar la calidad y cobertura del código mediante plugins externos. Algunos trabajos([10, 16]) desarrollan herramientas propias, con la desventaja de que los estudiantes no se familiarizan con las usadas en el ámbito empresarial.

8. Conclusiones y trabajo futuro

En este trabajo hemos presentado una experiencia en una asignatura de programación donde se han introducido técnicas y herramientas DevOps como forma de simplificar la gestión de las prácticas. Los resultados obtenidos en esta experiencia han sido positivos tanto para los estudiantes como para el profesorado de la asignatura. Desde el punto de vista de los estudiantes, gracias a la introducción de los tests se ha mejorado la calidad de las entregas, y se han reducido los problemas relacionados con las entregas y pérdida de código. Desde el punto de vista del profesorado, el tiempo dedicado a corregir las entregas se ha reducido gracias a las herramientas como GitHub, `classroom.github.com` o Travis CI.

A partir de la experiencia de este curso, y el *feedback* recibido de parte de los estudiantes, hemos detectado una serie de puntos que pueden mejorarse para los próximos años. En primer lugar queremos seguir mejorando la calidad de los tests para cubrir la mayor cantidad de requisitos posibles; además el uso de técnicas como la programación orientada a aspectos puede servir para verificar requisitos que no se pueden comprobar con los tests unitarios. También sería interesante integrar las herramientas utilizadas hasta ahora con plugins que proporcionen información adicional (control de copias, estadísticas, etc.). Por último, sería interesante extender el uso de las herramientas y técnicas DevOps a otras asignaturas del grado.

Referencias

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

[2] P. Bradford, M. Porciello, N. Balkon y D. Backus: *The Blackboard Learning System: The Be All and End All in Educational Instruction?* Journal of Educational Technology Systems, 35(3):301–304, 2007.

[3] A. Cernuda del Río *y cols.*: *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] P. Duvall, S. Matyas y A. Glover: *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007.

[5] I. R. Forman y N. Forman: *Java Reflection in Action*. Manning Publications, 2004.

[6] 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.

[7] 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.

[8] M. A. Gómez Martín, G. Jiménez Díaz y P. P. Gómez Martín: *Test de unidad para la corrección de prácticas de programación, ¿una estrategia win-win?* En *Actas de las XVI Jornadas de Enseñanza Universitaria de Informática (JENUI 2010)*, páginas 51 – 58, 2010.

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

[10] J. H. Hill: *CUTS: a system execution modeling tool for realizing continuous system integration testing*. En *Proceedings of the 30th international conference on Software engineering*, volumen 2, páginas 309–310. ACM, 2010.

[11] J. Humble y D. Farley: *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010.

[12] M. Hüttermann: *DevOps for Developers*. Apress, 2012.

[13] G. Kiczales *y cols.*: *Aspect-oriented programming*. En M. Akşit and S. Matsuoka (editor): *ECOOP'97 — Object-Oriented Programming*, páginas 220–242, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[14] C. Kästner: *Teaching Software Construction with Travis CI*, 2014. `https://www.cs.cmu.edu/~ckaestne/travis/`.

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

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

[17] P. Nash: *Catch2: C++ Automated Test Cases in a Header*, 2017. `https://github.com/catchorg/Catch2`.

[18] J. Roche: *Adopting DevOps Practices in Quality Assurance*. Communications of the ACM, 56(11):38–43, 2013.

[19] 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`.

[20] R. Sjodin y 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.

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