Informe estrategias de entrega y corrección

Proyecto SILVASE

-

Eugenio Cano Muñoz

Este documento ha sido creado después de investigar el funcionamiento de algunas herramientas de *GitHub*, especialmente *GitHub Classroom* (herramienta especializada para la automatización de la corrección y el manejo de entregas de alumnos). La finalidad del mismo es reflejar algunas estrategias para la automatización de la corrección de las entregas de proyectos de código y ha sido diseñado pensando principalmente en la asignatura de *SDGII* del *GTSIT*, aunque también se puede aplicar a cualquier otra asignatura en la que se necesiten corregir multitud de proyectos de código.

a) GitHub Classroom:

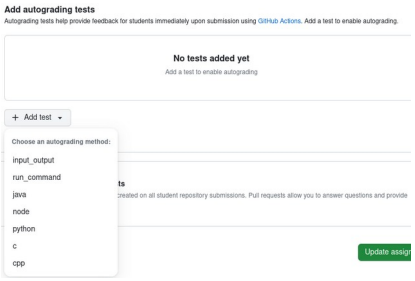
Esta estrategia pretende utilizar *GitHub Clasroom* para el manejo y corrección automática de los trabajos entregados por los alumnos, por lo que, una vez que se diseñen las pruebas se crea un documento CSV con los datos de los alumnos y sus calificaciones obtenidas.

El proceso de crear una tarea es bastante sencillo; simplemente hay que asignar un título, fecha de entrega, si lo queremos activar al momento y si el trabajo es individual o en grupo. Otro tema importante es que nos permite cambiar la visibilidad del repositorio, por lo tanto, los alumnos no tienen acceso de administrador al mismo. *GitHub* nos permite asignar la tarea a partir de una plantilla[1], por lo que los alumnos no parten de un proyecto en blanco y simplemente construyen a partir de ese código. Por último, podemos asignar métodos de corrección automática para asignar una nota a cada alumno automáticamente, aunque estas pruebas tienen bastantes limitaciones las cuales serán detalladas más adelante.

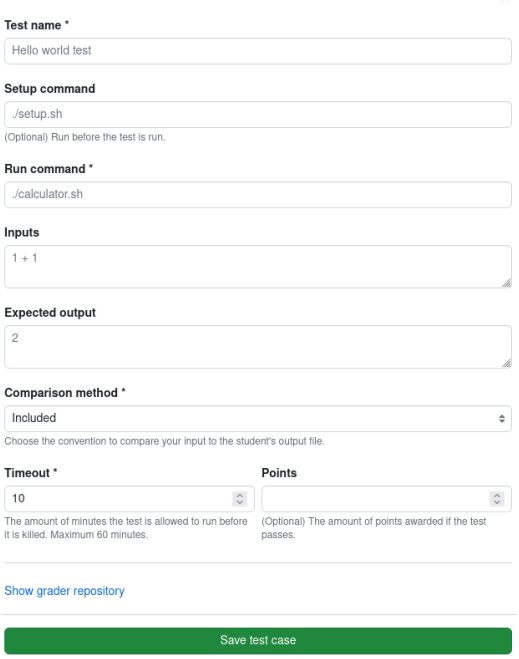
La mentalidad de desarrollo con esta estrategia es que el alumno desarrolla el proyecto desde su ordenador realizando *commits* a su base del proyecto siempre que lo vea oportuno y cada vez que se realiza un *push* a *GitHub* se entrega la tarea y se corrige automáticamente. Obviamente esta calificación no es definitiva y se va alterando cada vez que se sube el proyecto. Una vez que se pase la fecha de entrega, los alumnos pierden derechos de escritura de estos repositorios *tarea* y no pueden realizar más cambios, por lo que la nota de la última corrección pasa a ser la nota definitiva generada automáticamente. El profesor puede descargar el archivo CSV con las calificaciones en cualquier momento.

1 Se está actualizando para que las tareas de los alumnos sean *Forks* del proyecto, pero en el momento de escritura del documento este método da problemas

Esta estrategia es idónea para proyectos pequeños que no requieren alterar enormemente el entorno proporcionado por *GitHub* (Ubuntu Latest LTS), debido a que la plataforma proporciona diversas plantillas para compilar y ejecutar, como se puede observar en la siguiente captura:



El gran factor limitante es que no permite diseñar *actions* propias sino que deja escoger entre las opciones del menú desplegable. Las que más flexibilidad permite son *run_command* e *input_output*. Como su nombre indica, la primera ejecuta un comando y la segunda ejecuta un comando con parámetros y genera una salida, luego la compara con el texto que queramos. Por si esto no fuera poco, cada test es independiente de otros, por lo que únicamente permiten ejecutar un comando para preparar el entorno y otro para realizar la ejecución (Aunque pueden ser *scripts* de Bash). La siguiente imagen muestra los parámetros que se pueden configurar para un test *input_output*:



Una vez explicado el funcionamiento de *GitHub Clasroom* a grandes rasgos, se van a detallar los diferentes planes de acción que se han ideado para esta estrategia. Todos los planes hacen uso tanto del *script* de preparación como el de ejecución y utilizan la opción *input_output*:

a.1) Guardado de archivos en carpeta .github:

Este plan trata de guardar tanto los archivos de preparación como de ejecución dentro de la carpeta *.github* del proyecto, por lo que, aunque las pruebas y la asignación de notas se realice en la nube, no hay forma de saber si los *scripts* de instalación han sido modificados de alguna forma por el alumno. De hecho indagando en foros se ha comprobado que alterar las calificaciones y engañar al sistema parece bastante sencillo[2] ya que se ha comprobado que el alumno sigue teniendo acceso a esta carpeta aunque no aparezca la visualización en línea del repositorio.

a.2) Utilización de scripts remotos:

Para evitar los problemas comentados anteriormente, otra estrategia que se puede utilizar consiste ejecutar esos *scripts* en remoto, descargándolos de una fuente conocida. Este servidor tiene que estar preparado para soportar cierta carga simultánea, ya que cada vez que se ejecuta un test se tienen que descargar los dos archivos para ejecutarlos. En un principio, si se sigue esta estrategia, se intentaría que ambos *scripts* fuesen iguales para todas las pruebas a realizar y que se pasase el nombre de estas como argumento (Aún así sigue sin solucionarse el inconveniente del tráfico al servidor fuente).

a.3) Ejecución de tests en local y ofuscación de los resultados:

Con esta estrategia, los alumnos ejecutarían las pruebas individualmente en el ordenador y se guardarían los resultados ofuscados en el repositorio, así como la información personal del usuario, por lo que en este caso GitHub Classroom solo comprueba que la salida de los *tests* sea correcta y que la firma digital del alumno no haya sido alterada, todo esto para evitar que el alumno pueda alterar los resultados.

De todos los planes propuestos anteriormente, la tercera opción es la más sencilla de implementar y más óptima para la corrección de los *tests*, ya que al no poder hacer uso de las *actions* de *GitHub* se tienen que instalar todas las dependencias tanto de compilación como de ejecución para cada test, lo cual puede acabar repercutiendo en un el uso de un gran número de horas de ejecución dentro de *GitHub* (Entre 2 y 4 minutos por test), lo cual a la larga resulta bastante más costoso que el tercer plan de acción, que sería bastante más rápido y eficiente de ejecutar.

2 https://github.com/orgs/community/discussions/106545

b) Pull requests a un repositorio principal:

Esta estrategia se centra en realizar *forks* del proyecto principal. Es un método bastante más modular, ya que permite hacer uso al completo de las *actions* y que los usuarios trabajen de manera independiente con sus repositorios de la forma que quieran hasta que se realice una entrega.

La idea principal es que los alumnos trabajen con los *forks* del proyecto y cuando se quiera realizar una entrega, se haga una *pull request* con un nombre y una descripción específicos (ej: *Entrega v1 y v2 – XT23*). En un proyecto normal, cuando se realiza una *pull request*, se pueden configurar ciertas pruebas que por defecto sirven para comprobar si el proyecto proporcionado se puede combinar con el original. En nuestro caso, las utilizaríamos para poner una calificación automática al alumno, ya existen algunas automatizaciones para realizar justo esta tarea (véase: https://github.com/uhafner/autograding-github-action), además podríamos explorar la posibilidad de subir las notas a una base de datos centralizada por medio de *APIs* y controlar si se han alterado las *pruebas* con las funciones de *hashFiles.*

Algunos de los **beneficios** de esta estrategia son:

1. Control exacto de todos los cambios, ya que en teoría los usuarios deberían guardar los cambios con commits en git, los cuales aparecerían a la hora de realizar las pull requests.
2. Mayor facilidad para alterar el ambiente de trabajo así como de los componentes individuales (Posible migración a platformIO, alterar el comportamiento de qemu, como se ejecutan los tests…)

Como contraparte a esta estrategia, tenemos que los ***forks*** del proyecto principal deben ser **públicos**, lo cual puede llegar a generar un gran problema de copias entre alumnos, haciendo esta estrategia inviable. Además habría que almacenar las pruebas en el repositorio antes de realizar los forks (o los nombres en las automatizaciones que ejecutan las pruebas, si queremos utilizar los beneficios de paralelismo de la ejecución en matriz), aunque podríamos actualizarlos por medio de *scripts* con esto se pretende que los alumnos no alteren directamente ningún *action*.

c) Estrategia híbrida:

Debido al poco margen de maniobra que permite *GitHub Classroom* podemos optar por un enfoque híbrido que permita centralizar los repositorios para que los profesores se los puedan corregir semi-automáticamente así como utilizar al máximo los beneficios de las *actions.*

La estrategia consiste en utilizar *GitHub Classroom* para mandar las tareas y los usuarios las realizan como si estuviésemos en la primera estrategia, con el matiz de que no utilizamos la corrección automática proporcionada, sino que los alumnos utilizan las *actions* para ver que nota van a obtener y si pasan todas las pruebas. Llegado al fin del periodo de entrega, el profesor se descarga todos los repositorios de los alumnos haciendo uso del comando “gh classroom clone student-repos”[3] una vez hecho esto se ejecuta un *script* de corrección automática para corregir los trabajos y generar un fichero CSV con las notas.

3 Se debe instalar tanto *GitHub CLI* como la extensión de *GitHub Classroom*

Siguiendo esta estrategia, se podría hacer una única tarea e ir avanzando haciendo diferentes ramas para cada versión entregable (ej: v1, v2…)

Conclusiones:

Una vez exploradas todas las estrategias propuestas, pasamos a la determinación de las estrategias más optimas para cada tipo de proyectos y finalmente recomendamos la manera más óptima para realizar las entregas.

Como se comentó en el primer punto, una vez implementada la primera estrategia, es la más rápida y sencilla, aunque la complejidad para implementarla crece exponencialmente a medida que los proyectos se van especializando cada vez más, haciendo bastante complejo de implementar en proyectos que disponen de entornos más especializados (Como es el caso de MatrixMCU) además de ser bastante costosos respecto al tiempo de ejecución, ya que al solo poder ejecutar dos comandos no podemos utilizar ningún tipo de optimización propia de las *actions*, como es el caso de las *caches,* además esta estrategia es especialmente vulnerable a la manipulación de las pruebas ejecutadas.

La segunda estrategia dispone de bastantes más posibilidades, ya que se pueden utilizar todos los recursos propios de *GitHub,* como *actions*, *pull requests.*… El problema de esta estrategia es que los *forks* deben ser públicos, haciendo que esta estrategia sea altamente inviable para la calificación de proyectos originales.

Por último se propone una estrategia híbrida, que opta por combinar la centralidad y capacidad de gestión que otorga *GitHub Classroom* con el poder de automatización de los repositorios, en este caso las pruebas se ejecutan automáticamente cada vez que el alumno realiza un *push*, y cuando se finalice la entrega, el profesor descarga todos los repositorios en un zip, ejecutando las pruebas de todos los proyectos automáticamente. Esta estrategia es la más adecuada para los proyectos más complejos o que requieren un entorno específico, como es el caso de MatrixMCU y SDGII.