

Patrones de Diseño

Percy Taquila Carazas, Katerin Merino Quispe, Abraham Lipa Calabilla,
Edwart Balcon Coahila, Lisbeth Espinoza Caso

January 8, 2021

Abstract

The Test Management Tool is the tool that provides support for the test management and control of part of the testing process. Often it has multiple capabilities, such as managing test support products, test planning, results recording, process tracking, incident management, and test reporting. They have different approaches to testing and therefore have different sets of features. They are typically used to maintain and schedule manual tests, run or collect automated test execution data, manage multiple environments, and enter information on found defects.

Resumen

Las Herramienta de gestión de pruebas es la herramienta que proporciona soporte a la gestión de pruebas y control de parte del proceso de pruebas. A menudo tiene varias capacidades, tales como gestionar los productos de soporte de pruebas, planificación de pruebas, registro de resultados, seguimiento del proceso, gestión de incidencias y generación de informes de las pruebas. Tienen diferentes enfoques de prueba y, por lo tanto, tienen diferentes conjuntos de características. Generalmente se utilizan para mantener y planificar pruebas manuales, ejecutar o recopilar datos de ejecución de pruebas automatizadas, administrar múltiples entornos e ingresar información sobre defectos encontrados.

I. INTRODUCCION

Las herramientas de gestión de pruebas son aquellas que se utilizan para gestionar la información relativa a los «casos de prueba», normalmente los funcionales, para planificar actividades de testing, para gestionar los informes resultantes después de pasar dichos test, etc. Es fundamental para cualquier proyecto, salvo que sea muy pequeño, contar con alguna herramienta de gestión de pruebas. Hay herramientas que van por separado y otras que integran con herramientas complementarias, por ejemplo, con las de «bug tracking».

Probar no es solo buscar errores y regresiones. La gestión de pruebas también es vital. En cualquier empresa, es fundamental que pueda auditar la cobertura, la ejecución y los resultados de las pruebas. Aquí es donde entran en juego las herramientas de gestión de

pruebas.

II. DESARROLLO

```
package template;

public abstract class Comida {

    // template method
    public final void imprimir() {
        ingredientes();
        cocinar();
        comer();
    }

    public abstract void ingredientes();
    public abstract void cocinar();
    public abstract void comer();
}
```

i. GitHub Actions

Permite automatizar, personalizar y ejecutar flujos de trabajo de desarrollo de software directamente en tu repositorio. Así mismo, permite descubrir, crear y compartir acciones para

realizar incluso **CI/CD**, y combinar acciones en un flujo de trabajo personalizado. [1]

GitHub Actions ayuda a automatizar tareas dentro del ciclo de vida del desarrollo de software. GitHub Actions está controlada por eventos, lo que significa que puede ejecutar una serie de comandos después de que haya ocurrido un evento específico. Por ejemplo, cada vez que alguien crea *pull request* para un repositorio, puede ejecutar automáticamente un comando que ejecuta un script de prueba de software. [2]

i.1 Componentes

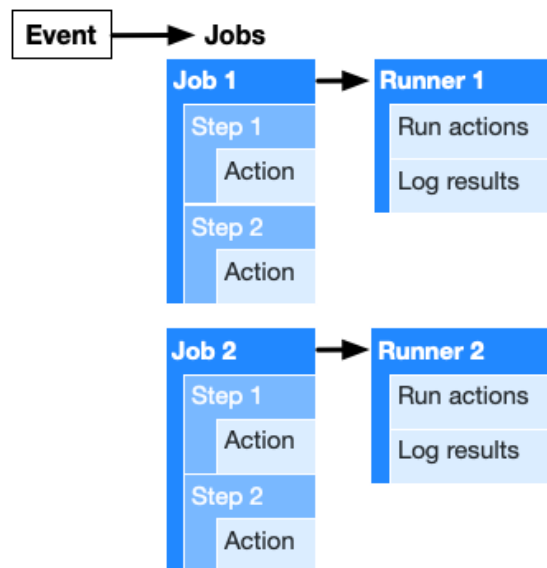


Figure 1: Diagrama de componentes en GitHub Actions

A continuación, se describirán los componentes de GitHub Actions. [2]

Workflows El flujo de trabajo (*workflow*) es un procedimiento automatizado que agrega a su repositorio. Los flujos de trabajo se componen de uno o más trabajos (*jobs*) y pueden ser programados o activados por un evento (*event*). El flujo de trabajo se puede usar para:

- Crear
- Probar
- Empaquetar

- Lanzar
- Implementar

Events Un evento (*event*) es una actividad específica que desencadena un flujo de trabajo. Por ejemplo, la actividad puede originarse en GitHub cuando alguien:

- Ejecuta un **push**
- Crea un problema (*issue*)
- Crea un **pull request**

Jobs Un trabajo (*job*) es un conjunto de pasos (*steps*) que se ejecutan en el mismo *runner*. De forma predeterminada, un flujo de trabajo con varios trabajos ejecutará esos trabajos en paralelo, pero también puede hacerlo de forma secuencial. Por ejemplo, un flujo de trabajo puede tener dos trabajos secuenciales que compilan y prueban código, donde el trabajo de prueba depende del estado del trabajo de compilación. Si el trabajo de compilación falla, el trabajo de prueba no se ejecutará.

Steps Un paso (*step*) es una tarea individual que puede ejecutar comandos en un trabajo. Un paso puede ser una acción o un comando de shell. Cada paso de un trabajo se ejecuta en el mismo *runner*, lo que permite que las acciones de ese trabajo compartan datos entre sí.

Actions Las acciones (*actions*) son comandos independientes que se combinan en pasos para crear un trabajo. Las acciones son el bloque de construcción portátil más pequeño de un flujo de trabajo. Es posible crear sus propias acciones o utilizar acciones creadas por la comunidad de GitHub. Para usar una acción en un flujo de trabajo, debe incluirla como un paso.

Runners Un runner es un servidor que tiene instalada la aplicación de ejecución de acciones de GitHub. Puede utilizar un runner alojado en GitHub, o puede alojar el suyo propio. Un runner escucha los trabajos disponibles, ejecuta un trabajo a la vez e informa el progreso, los registros y los

resultados a GitHub. Para los ejecutores alojados en GitHub, cada trabajo de un flujo de trabajo se ejecuta en un entorno virtual nuevo.

Los runners alojados en GitHub se basan en:

- Ubuntu Linux
- Microsoft Windows
- macOS

i.2 YAML

YAML es un lenguaje de serialización de datos diseñado para ser *leído y escrito por humanos*. Basa su funcionalidad en JSON, con la adición de líneas nuevas e indentación inspirada en Python. A diferencia de Python, YAML no permite tabulaciones literales. [3]

Las GitHub Actions utiliza la sintaxis YAML para definir los eventos, trabajos y pasos. Estos archivos YAML se almacenan en el repositorio, en el directorio `.github/workflows`. [2]

i.3 Ejemplo

Se puede crear un flujo de trabajo de ejemplo en un repositorio que activa automáticamente una serie de comandos cada vez que se realiza un **push**. En este flujo de trabajo, GitHub Actions verifica el código enviado, instala las dependencias del software y ejecuta **bats -v**.

Los pasos para crear este flujo de trabajo son los siguientes:

1. Crear un repositorio en GitHub.
2. Crear un nuevo archivo `learn-github-actions.yml` en el directorio `.github/workflows/` con el siguiente código:

```
name: learn-github-actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v1
```

```
- run: npm install -g bats
- run: bats -v
```

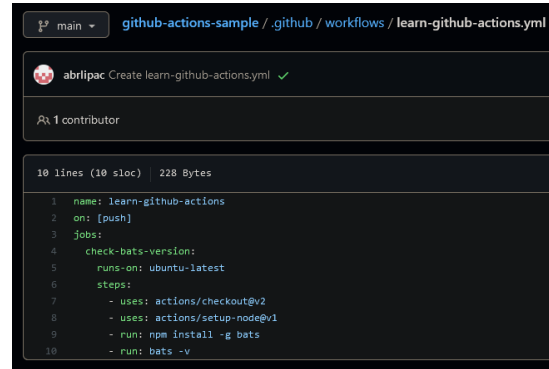


Figure 2: Captura de la ruta del archivo creado

3. Guardar los cambios (commit)

Cada vez que se haga *push* al repositorio, se ejecutará el flujo de trabajo creado.

Es posible ver la actividad de los flujos en la pestaña **Actions**. Al abrir un workflow se puede ver los pasos y los logs generados en cada uno.

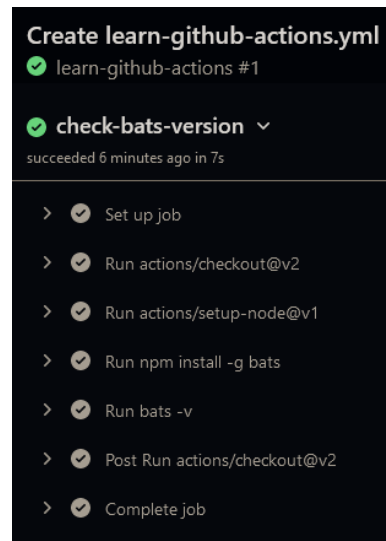


Figure 3: Captura del flujo de trabajo en la pestaña *Actions*

| Producto | Almacenamiento | Minutos | URL |
|---------------------------------|-----------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GitHub Free | 500 MB | 2,000 | https://docs.github.com/es/free-pro-team@latest/actions [2] |
| GitHub Pro | 1 GB | 5,000 | https://docs.github.com/es/free-pro-team@latest/actions/learn-github-team [3] |
| GitHub Free para organizaciones | 500 MB | 2,000 | https://docs.github.com/es/free-pro-team@latest/actions/learn-github-team [3] |
| GitHub Team | 2 GB | 5,000 | https://docs.github.com/es/free-pro-team@latest/actions/learn-github-team [3] |
| GitHub Enterprise Cloud | 50 GB | 5,000 | https://docs.github.com/es/enterprise-cloud@latest/actions/learn-github-team [4] |
| Sistema operativo | Tasa por minuto | Multiplicador | URL |
| Linux | \$0.008 | 1 | https://docs.github.com/es/free-pro-team@latest/github/setting-up-and-managing-billing-and-payments-on-github/about-billing-for-github-actions |
| macOS | \$0.08 | 10 | |
| Windows | \$0.016 | 2 | |

i.4 Precios

El uso de GitHub Actions es gratuito para los repositorios públicos. Para los repositorios privados, cada cuenta de GitHub recibe una cantidad determinada de minutos y almacenamiento gratuitos dependiendo del producto que se utilice con la cuenta.

Cada sistema operativo consume una cantidad diferente de minutos, siendo 1 minuto de trabajos ejecutados en Linux equivalente a 1 minuto de facturación mensual.

III. CONCLUSIONES

La conclusión

IV. RECOMENDACIONES

- Cuando se conoce el efecto colateral que conlleva el patrón de diseño y es viable la aparición de este efecto.
- Suministrar alternativas de diseño para poder tener un software flexible y reutilizable.

REFERENCIAS

- [0] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John(1995).Design Patterns: Elements of Reusable Object- Oriented Software. Reading,Massachusetts: Addison Wesley Longman, Inc.