Universidad de La Laguna

ghedsh: Un intérprete de comandos para GitHub Education
GitHub Education Shell: ghedsh.

Autor: Carlos de Armas Hernández
Director: Casiano Rodríguez León

Escuela Superior de Ingeniería y Tecnología
Departamento de Ingeniería Informática y de Sistemas
Universidad de La Laguna

12 de julio de 2018

Índice

Introducción I

¿Qué es "ghedsh"?

Es una gema Ruby que consiste en un intérprete de comandos desarrollado para integrar las metodologías de GitHub Education, viendo las organizaciones como aulas y los repositorios como las asignaciones de los alumnos.

Introducción II

En cuanto a herramientas similares, existen las siguientes:

- *Teachers Pet*.
- *ghi* (GitHub Issues).
- *ghs* (GitHub Search).

Introducción III

Desarolladas por *GitHub*:

- **Teachers Pet**. *CLI* desarrollado previamente a *Classroom*.
  - Inconveniente: los comandos se hacían excesivamente largos en determinados casos. Cayó en desuso y se dejó de desarrollar.

Desarrolladas por la comunidad:

- **ghi** (GitHub Issues). Permite gestionar SOLO las incidencias (issues) de los repositorios desde la terminal del usuario.
- **ghs** (GitHub Search). Permite realizar búsquedas de repositorios alojados en GitHub. Es poco ágil.

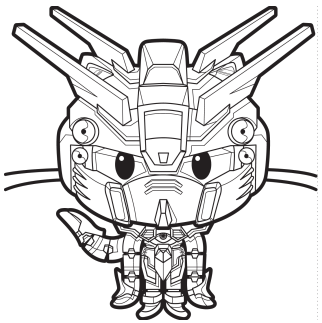
Objetivos

- Esta segunda versión de *ghedsh* busca mejorar el código fuente de la primera versión, teniendo en cuenta aspectos como:
  - La mantenibilidad del código
  - Facilitar la incorporación de nuevas funcionalidades por parte de terceros.
- Por otro lado, se han añadido funcionalidades que dan soporte al proceso de evaluación.

Tecnologías empleadas

Lenguaje de programación: **Ruby**

Gestión de dependencias: **Bundler**



API de GitHub: **octokit**

Testing: **RSpec**

Desarrollo del proyecto

Dividimos el desarrollo del proyecto en dos fases bien diferenciadas:

- Análisis. Identificar aquellas partes mejorables del diseño e implementación iniciales.
- Refactorización. Proceso llevado a cabo para solucionar las debilidades anteriores.

Primera fase. Análisis

Tras estudiar el código de la primera versión de la gema se han detectado diversos *code smell*.

¿Qué es un code smell?

Se define como cualquier característica del código fuente que, posiblemente, indica un problema más profundo. No son considerados como bugs.

Primera fase. Análisis

Switch Smell

USER=1
ORGS=2
USER_REPO=10
ORGS_REPO=3
TEAM=4
ASSIG=6
TEAM_REPO=5

```
case
  when op == "exit" then ex=0
    @sysbh.save_memory(config_path,@config)
    s.save_cache(config_path,@config)
    s.remove_temp("#{ENV['HOME']}/.ghedsh/temp")
  when op.include?("help") && opcd[0]=="help" then self.help(opcd)
  when op == "orgs" then self.orgs()
  when op == "cd .."
    if @deep==ORGS then t.clean_groupsteams() end ##cleans groups cache
    self.cdback(false)
  when op.include?("cd assig") && opcd[0]=="cd" && opcd[1]=="assig" && opcd.size==3
    if @deep==ORGS
      self.cdassig(opcd[2])
    end
  when op == "people" then self.people()
  when op == "teams"
    if @deep==ORGS
      t.show_teams_bs(@client,@config)
    end
  when op == "commits" then self.commits()
  when op == "issues"
    if @deep==ORGS_REPO || @deep==USER_REPO || @deep==TEAM_REPO
      @issues_list=r.show_issues(@client,@config,@deep)
    end
  when op == "col" then self.collaborators()
  when op == "forks" then self.show_forks()
```

Long Method se clasifica a nivel de método. Como su propio nombre indica, consiste en un método que ha crecido demasiado y dificulta saber qué es lo que realmente hace.

Large Class se clasifica dentro de los smells a nivel de clases. Indica que una clase ha crecido excesivamente en tamaño (God Object). Su funcionalidad puede descomponerse en clases más pequeñas.

Segunda fase. Refactorización

Esta fase se centra en eliminar las debilidades anteriormente comentadas. Gran parte de este proceso consistió en eliminar el *Switch Smell*, ya que era el más repetido a lo largo del código fuente.

Para eliminar el *Switch Smell* hemos aplicado el Patrón Estrategia (*Strategy Pattern*).

El propósito de este patrón es proporcionar una manera clara de definir familias de algoritmos y poder intercambiarlos fácilmente.

Segunda fase. Refactorización

Strategy Pattern

```
loop do
  begin
    input = Readline.readline(@shell_enviroment.prompt, true)
    # handle ctrl-d (eof), equivalent to exit (saving configuration)
    if input.nil?
      @shell_enviroment.commands['exit'].call(nil)
      exit!(0)
    end
    input = input.strip.split
    command_name = input[0]
    input.shift
    command_params = input
    unless command_name.to_s.empty?
      if @shell_enviroment.commands.key?(command_name)
        result = @shell_enviroment.commands[command_name].call(command_params)
      else
        puts Rainbow("-ghedsh: #{command_name}: command not found").yellow
      end
    end
  rescue StandardError => e
    puts e
  end
  break if result == 0
end
```

Segunda fase. Refactorización

Extract Method

Problema:

```
1    def print_owing(amount)
2      print_banner
3
4      # print details
5      puts "name: #{@name}"
6      puts "amount: #{amount}"
7    end
```

Solución (Extract Method):

```
1    def print_owing(amount)
2      print_banner
3      print_details(amount)
4    end
5
6    def print_details(amount)
7      puts "name: #{@name}"
8      puts "amount: #{amount}"
9    end
```

Segunda fase. Refactorización

Extract Class

Una clase debería tener una única responsabilidad.

- Paso 1. Determinar qué se va a extraer.
- Paso 2. Crear una nueva clase.
- Paso 3. Renombrar la clase antigua.

Resultados obtenidos

Tras la etapa de desarrollo, *GitHub Education Shell* incorpora las siguientes características:

- Autenticación con credenciales de GitHub (OAuth).
- Conjunto de comandos
  - Comandos del núcleo de *ghedsh*.
  - Comandos incorporados (*built-in commands*).
  - Comandos que dan soporte al proceso de evaluación.

Resultados obtenidos

Comandos del núcleo de *ghedsh*

- **cd**: permite movernos entre repositorios, organizaciones, equipos, etc.
- **! ó bash**: interpreta la entrada del usuario como un comando tipo Unix.

Resultados obtenidos

Comandos incorporados

```
clear
clone
commits
exit
files
invite_member
invite_member_from_file
invite_outside_collaborators
issues
new_issue
```

```
new_repo
new_team
open
orgs
people
repos
rm_repo
rm_team
teams
```

Resultados obtenidos

Comandos que dan soporte al proceso de evaluación

Tenemos los siguientes:

- **new_eval**
- **foreach**
- **foreach_try**

new_eval

Permite crear un repositorio de evaluación. Consiste en hacer uso de los submódulos de *git*, de manera que se crea un "súper repositorio" que contiene como submódulos los proyectos que se van a evaluar.

foreach

Ejecuta para cada submódulo el comando especificado (por ejemplo, `git pull`). Internamente realiza `git submodule foreach`. Si ocurre algún error en un submódulo durante la ejecución del comando, pasa al siguiente submódulo.

foreach_try

Realiza lo mismo que `foreach`. Sin embargo, `foreach_try` dentendrá su ejecución si ocurre algún error en un submódulo.

Resultados obtenidos

foreach y foreach_try

Hagámonos una pregunta:

¿Y si en lugar de ejecutar un comando simple, hacemos uso de una librería o paquete que realiza tareas más complejas?

Resultados obtenidos

ghedsh-grade-node

Explico el paquete de ejemplo [....]

Caso de uso I

Conclusions and future work lines I

Bibliografía I


"Node JS."
https://nodejs.org/es/docs/.


"Github REST API V3."
https://developer.github.com/v3/.


"Express JS."
http://expressjs.com/es/.


"Pug JS."
https://pugjs.org/api/getting-started.html.


"Mongoose JS."
http://mongoosejs.com/.


"Heroku."
https://devcenter.heroku.com/.


"NPM."
https://www.npmjs.com/.

Fin de la presentación

Gracias por su atención