Metodología para el análisis y medición automática de sistemas de software

Kevin Hernández Rostrán
Tecnológico de Costa Rica

25 de junio de 2020

Resumen

Este documento muestra el significado de estrategías accionables en el desarrollo de software, se analizan los métodos actuales para la representación visual del progreso del software sobre un proyecto de software. Se propone una metodología de visualización dado el fenómeno de *factor bus* en un proyecto, se muestra la herramienta realizada y la travesía a través de su desarrollo con la incorporación de *git hooks*.

Introducción

Una unidad fundamental de trabajo en la programación es la contribución del código "commit" que un desarrollador hace a la base del código de algún proyecto, sea individual o colaborativo. El desarrollo y mantenimiento de software implica cambiar código para agregar nuevas funciones, resolver defectos o mejorar el rendimiento. Dichos cambios tienen lugar de muchas formas: agregando nuevos métodos, implementando diferentes implementaciones de programación como concurrencia y manejo de excepciones, actualizando funciones de algún API, etc. Los administradores, desarrolladores e investigadores a menudo están interesados en analizar dichos datos evolutivos para comprender las características del desarrollo y mantenimiento del software. Por ejemplo, un administrador podría estar interesado en saber "¿Cómo la elección de un lenguaje de programación afecte la calidad del software?". Del mismo modo, un desarrollador podría preguntarse "Si uso muchas validaciones en mi código, ¿me ayuda a reducir errores?". Finalmente, los investigadores podrían tener curiosidad sobre "¿Cómo afecta la evolución de un API a la calidad del código?", O "¿Cuáles son las características de los diferentes errores del mundo real, por ejemplo, errores de concurrencia y rendimiento?". También los proyectos de desarrollo de software enfrentan muchos riesgos (como inflación de requisitos, mala programación, problemas técnicos, etc.) Subestimar esos riesgos puede poner en peligro el éxito del proyecto.

Metodología propuesta

Factor Bus

Por definición se conoce como el número de desarrolladores clave que necesitarían ser incapacitados, es decir, atropellados por un autobús, para que un proyecto no pueda continuar. Pero tal vez ser atropellado por un bus sea una situación muy extrema, podríamos cambiarlo por "tomar vacaciones al mismo tiempo" para tener la misma idea.

Este factor es un indicador de lo costoso que será reemplazar a personas específicas.

Alcanzar un factor bus de cero es posible si la base de código es lo suficientemente pequeña como para que pueda caber fácilmente en la mente de una persona.

En una situación ideal, todos en el equipo conocerán todas las partes del sistema para que la pérdida de cualquier persona tenga un impacto mínimo. En realidad, muchos proyectos dependen de uno o más "héroes" que son los únicos que entienden ciertas partes críticas del sistema. Cuando estos héroes se van (y se debe asumir que lo harán), el equipo debe estar preparado para recuperarte.

Si se tiene un héroe en el equipo, lo mejor que se puede hacer es reasignar a esa persona a una parte diferente del proyecto. Esto permitirá que el reemplazo aumente conocimiento mientras el héroe aún está disponible para dar asistencia.

El factor bus es una métrica rápida que resaltará los posibles problemas en el proyecto. Tener héroes en el equipo puede ser muy beneficioso, pero deben ser cuidados los que están a cargo de la administración del proyecto. El factor bus es una métrica que resaltará sus dependencias.

Git hook

Un *git hook* se puede usar para ejecutar uno o varios *scripts* cada vez que ocurre un evento en particular, como un "commit" o un "push" a su repositorio. Estos eventos generalmente se dividen en pasos y se puede conectar un script a cada uno de estos pasos. Los *hooks* pueden ser del lado del cliente, afectando su repositorio local o del lado del servidor, ejecutando scripts en el repositorio remoto.

Hay varios hooks disponibles (están normalmente ubicados en la carpeta `.git/hooks`) la diferencia entre todos esos hooks es cuando se llaman, o en otras palabras, la acción que los activa.

La mayoría de los ganchos git se dividen en una de dos categorías ("Extending Git Functionality with Git Hooks", 2016) :

- *Pre-hooks*: Se ejecutan antes de que se complete una acción, este tipo de hook se puede utilizar para aceptar, rechazar o modificar un cambio antes de que se aplique. Un ejemplo sería un *pre-commit hook* que verificaría que el mensaje contenga palabras que no están permitidas y, por lo tanto, las rechazará.
- *Post-hooks*: Estos se ejecutan después de que se completa una acción y se pueden usar para notificar a los usuarios o para comenzar a ejecutar una compilación. Un ejemplo sería un *post-update hook* que cierre un issue abierto si está etiquetado en el mensaje del commit.

Los git hooks no se inicializan en el repositorio local al clonar el repositorio remoto, lo que significa que debe configurarse manualmente para cada clon.

Para implementar un hook, todo lo que se tiene que hacer es ponerlo en la carpeta `.git/hooks`, nombrarlo correctamente (es decir, de acuerdo con el evento que debería desencadenarlo, por ejemplo, *pre-commit*. Por lo general, ya se tiene una muestra para cada posible trigger en el directorio `.git/hooks` después de la inicialización) después hay que hacer el script *ejecutable* (por ejemplo, `chmod+ x pre-commit`).

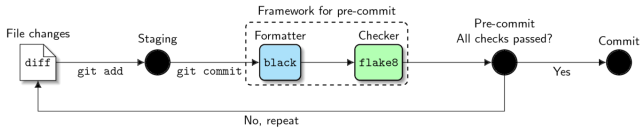

Figura 1: **Pre-commit workflow** con *black* y *flake8* para verificar archivos de python *.py*

La figura 1 muestra un ejemplo del flujo de trabajo de un pre-commit para verificar el formato de los archivos de acuerdo a los estándares predefinidos del proyecto.

Para ver un ejemplo de script para un flujo de trabajo *pre-release* tenemos el código del archivo release.sh.

```
#!/usr/bin/bash


current_version=‘git describe --abbrev=0 --tags‘
zip_file="Web␣API␣$current_version.zip"

files="\
API/bin/Release/Publish/bin␣\
API/bin/Release/Publish/Contexts␣\
API/bin/Release/Publish/Exporters␣\
API/bin/Release/Publish/Views␣\
API/bin/Release/Publish/favicon.ico␣\
API/bin/Release/Publish/Global.asax␣\
API/bin/Release/Publish/Web.config␣\
"


workdir=$(mktemp --tmpdir -d gitzip.XXXXXX)
```

```
cp -avr $files "$workdir" >/dev/null

pushd "$workdir"
git init
git add .
git commit -m "Web␣API␣$current_version"
popd


git archive --format=zip -o "$zip_file" --remote="$workdir" HEAD

echo 'zip file generated'
echo 'Removing temp file'
rm -rf "$workdir"
```

Código 1: release.sh

Para cumplir el objetivo de la herramienta se buscó la manera de representar visualmente la historia de acuerdo a la evolución con que se desarrolla el proyecto. Por esa razón se realizó la representación visual en Observable, para realizar un diagrama en D3 y a través de un *git hook,* de evento *post-merge* y generar el diagrama de acuerdo a los datos suministrados por el repositorio. Más adelante profundizaremos sobre estos conceptos.

La herramienta

Es un proyecto creado y mantenido por Georgios Gousios. Proporciona un mirror off-line de los datos de GitHub, en dos formatos: datos estructurados y datos sin procesar sobre eventos. Los eventos se recopilan mediante la API de eventos de GitHub y se complementan con otras solicitudes a la API REST de GitHub para proporcionar una representación estructurada. GHTorrent proporciona tres formas de acceder a sus datos: volcados de MySQL o MongoDB, servicio web proporcionado por GHTorrrent al acceder a los últimos volcados de MySQL o MongoDB, o Google Big Query al acceder desde MySQL.

Toda esta información es relevante al elegir una fuente de datos para una investigación, y puede afectar su resultado. GHTorrent y GitHub Archive son apropiados cuando es importante recuperar datos históricos sobre proyectos de acuerdo a un estudio comparativo realizado por Mombach y Valente (Mombach y Valente, s.f.).

Selección de herramienta

Actualmente existen herramientas al estilo de notebook que logran mostrar código a la hora de representar visualizaciones, son muy prácticos a la hora de probar y obtener resultados inmediatos sobre estadística de datos, análisis de información, etc.

Python Jupyter

Python Jupyter (Kluyver y cols., 2016), que es una aplicación moderna que emplea un paradigma de programación alfabetizada. El objetivo de Jupyter es lograr una narrativa computacional combinando el lenguaje humano, el código en vivo y sus resultados en un solo documento narrativo.

Observable HQ

Observable HQ (*The magic notebook for exploring data / Observable*, s.f.) proporciona notebooks de JavaScript para comentarios instantáneos, donde es muy sencillo agregar interacción y animaciones. Los notebooks se pueden compartir de manera web, supera las limitaciones de los documentos estáticos como archivos PDF al proporcionar la posibilidad de integrar contenido interactivo en documentos de texto. La facilidad de Observable es que permite la incorporación de frameworks como D3.js, Vega-Lite, en realidad cualquier biblioteca que exista para creación de diagramas.

Para la representación de esta metodología se decide utilizar la herramienta de Observable ya que permite una representación visual e inmediata de los datos de forma Web. Se prefiere el uso del framework D3.js debido a todas las formas de representación visual que permite a la hora de modelar datos.

La herramienta

Con Observable y D3.js podemos realizar representaciones visuales impresionantes (Meeks, 2015), como lineas en el espectro de colores de la tabla periódica, la jerarquización de la estructura de gobierno de un país, el índice de crecimiento de una enfermedad, en fin, un sinnúmero de ideas, pero nos vamos a enfocar en la representación en un proyecto que contenga control de versiones.

El código del archivo *post-merge* muestra el procedimiento que se siguió para generar la entrada de la herramienta.

```
#!/bin/bash
exec < /dev/tty

# Get the current branch name
branch_name=$(git branch | grep "*" | sed "s/\*␣//")

# Get the name of the branch that was just merged
reflog_message=$(git reflog -1)
merged_branch_name=$(echo $reflog_message | cut -d"␣" -f 4 | sed "s/://")

# if the merged branch was master - don't do anything
if [[ $merged_branch_name = "master" ]]; then
    exit 0
fi

# Begin output
echo "␣"
echo "You've␣just␣merged␣the␣branch␣\"$merged_branch_name\"␣into␣\"$branch_name\".␣"

# Ask the question
read -p "Do␣you␣want␣to␣create␣␣the␣graph?␣(y/N)␣" answer
```

```
# Check if the answer is a single lowercase Y
if [[ "$answer" == "y" ]]; then

    # Delete the local branch
    echo "Exporting␣your␣log␣in␣a␣file"
    git log --pretty=format:"%h%x09%an%x09%ad%x09%s" > git-log.txt

    # Delete the remote branch
    echo "Processing␣log␣with␣gitLogSummary.js"
    node gitLogSummary.js git-log.txt

    echo "Saving␣log␣in␣a␣csv"
    node gitLogSummary.js git-log.txt --csvOnly > git-log.csv
    exit 1
else
    exit 1
fi
```

Código 2: post-merge

Experimento

Para iniciar es importante dar a conocer el inicio de la representación, se hicieron varios prototipos, tomados de la galería de representaciones de Observable, creada por Mike Bostock [1] [2]. En esta galería se pueden encontrar ejemplos de: animación, interacción, análisis, jerarquías, redes, barras, líneas, áreas, puntos, mapas o secciones como *Just for fun*.

Cuando se inicia en Observable, se da una bienvenida una degustación de lo que es[3]; en conjunto con tres tutoriales bastante útiles[4]:

1. Lunch calculator
2. Dog pictures
3. Visualizing data

Todo este "tour" ayuda a entender la herramienta y familiarizarse con D3.js.

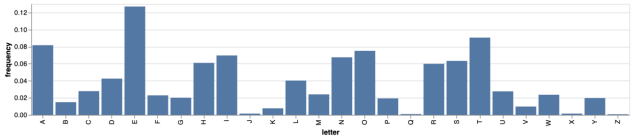

Figura 2: Frecuencia de las letras en el idioma inglés

La figura 2 muestra un gráfico de barras básico de un conjunto de datos: la frecuencia de las letras en el idioma inglés[5]. El notebook consistió en las líneas del código 2.

[1] Es uno de los cocreadores de Observable y se destacó como uno de los desarrolladores clave de D3.js

[2] Este es su perfil en Observable: https://observablehq.com/@mbostock

[3] Lectura recomendada: https://observablehq.com/@observablehq/a-taste-of-observable

[4] https://observablehq.com/@observablehq/tutorial

[5] El dataset se obtuvo del módulo @observablehq/alphabet

```
md`# Tutorial 3: Visualizing data`
alphabet = require('@observablehq/alphabet')
vegalite = require('@observablehq/vega-lite')
vegalite({
  data: { values:alphabet },
  mark: "bar",
  autosize: "fit",
  width: width,
  encoding: {
    x: {
      field: "letter",
      type: "ordinal"
    },
    y:{
      field: "frequency"
    }
  }
})
```

Código 3: tutorial

Después de continuar explorando Observable, se tuvo que pensar en Joins[6], una de las grandes áreas de D3. Y bueno, es importante destacar que D3 es una colección de módulos[7] diseñados para trabajar juntos. Se pueden usar los módulos de forma independiente o juntos como parte de la compilación predeterminada. Los principales módulos que se cree es para "novatos" son:

- **Arrays (d3-array)**
- Colors (d3-color)
- Number Formats (d3-format)
- Geographies (d3-geo)
- **Hierarchies (d3-hierarchy)**
- **Shapes (d3-shape)**

Ahora, si bien no son las visualizaciones mas elegantes, las diferencias de archivos son posiblemente las mas importantes debido a su ubicuidad. La figura 3 muestra como se veian las aplicaciones de diferencia de archivos hace 24 anos.

El siguiente experimento en Observable fue una visualización de diferencias de archivos con el módulo Join que se mencionó anteriormente. Este fue un "fork" del notebook de Steve Kasica[8] denominado "Software Engineering Visualization with GitHub's API"[9]

Despues de algunos intentos y fracasos la herramienta de visualizacion se pudo realizar a partir del fork de un notebook creado por Augustin Lacour[10] llamado "Git Project Summary" con modificaciones en cuanto al ingreso de los datos y su script de entrada. Este es el enlace[11].

[6] https://bost.ocks.org/mike/join/
[7] D3 API completo: https://github.com/d3/d3/blob/master/API.md
[8] Estudiante de doctorado de la Universidad de Columbia Britanica en el InfoVis Lab
[9] https://observablehq.com/@stvkas/software-engineering-visualization-with-githubs-api
[10] Ingeniero de Software, https://observablehq.com/@austil
[11] https://observablehq.com/d/8e6f52db94dbe55b

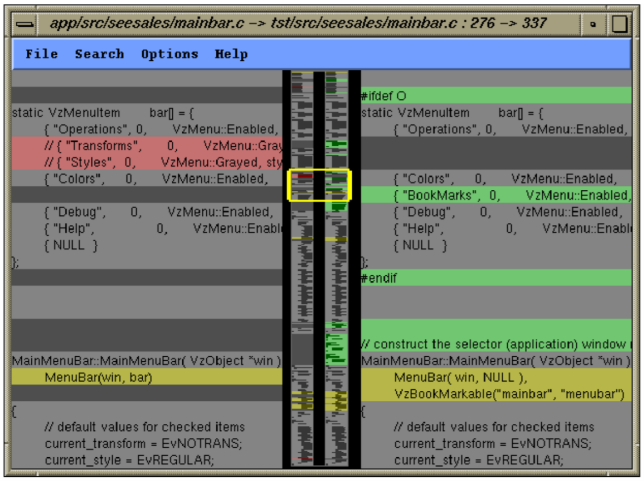

Figura 3: Ball T., Eick, S. (1996) Software Visualization in the Large. En IEEE Computer, Vol 29, No. 4. Abril 1996, 33-43

Resultados

De acuerdo a la figura 4 podemos ver el factor bus por desarrollador en periodo de tiempo, los colores de arriba significan (de acuerdo a la intensidad) que un Dev ("developer" o desarrollador) tiene alto indice de impacto en el proyecto, es decir su factor bus es muy elevado. Los datos de mas abajo van de acuerdo a convenciones de codificacion de los "commits" (*Commits Convencionales*, s.f.).

Se puede notar que los Dev 5 y Dev 9 son los de mas conocimiento en el proyecto, lo que hace que el proyecto este en peligro si alguno de ellos se incapacita.

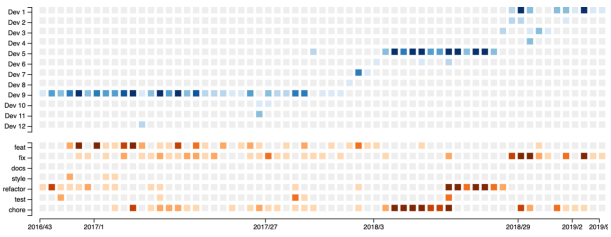

Figura 4: Factor bus por desarrollador en período de tiempo

El proyecto fue efectivo a razón no tanto de rendimiento, sino en probar una nueva metodología en cuanto a la evolución del software, que parte del flujo de trabajo de *git*, hasta la generación del gráfico en *D3.js*.

Trabajos relacionados

GiLA es una herramienta que proporciona información del factor de bus para cualquier archivo, directorio, rama y extensión de archivo en el repositorio de Git, destacando a los desarrolladores clave (es decir, los más expertos) para cada repositorio y para el proyecto *per se*.

En la Figura 1, se visualiza una demostración de la herramienta GiLA donde se encuentra el análisis del factor bus de un archivo de formato .png. La herramienta muestra una vista general del proyecto que consiste en 4 desarrolladores que contribuyeron activamente en 41 archivos donde el 24.39% son en formato *png*, el 17.07% son *javascript* y 12.2% son de tipo *java*. También muestra la relevancia de cada miembro del proyecto. Se puede notar que si *Belen* se incapacita el proyecto puede sufrir graves consecuencias.

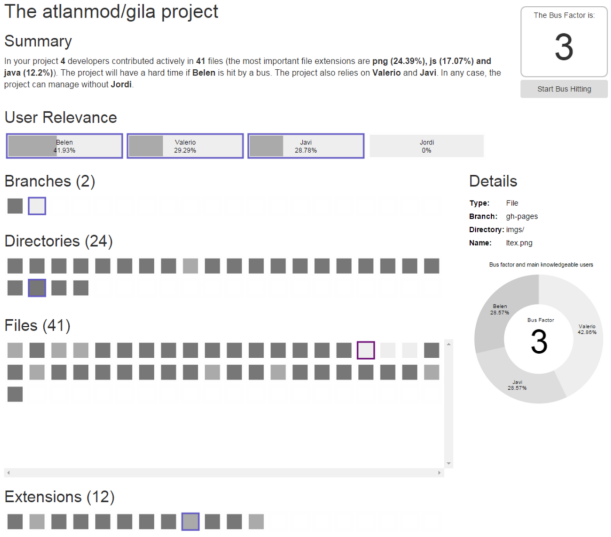

Figura 5: Análisis del factor de bus de un archivo en *GiLA*

Existen otras herramientas basadas en este principio como GitProc que se basa en expresiones regulares y bloques de código fuente, que descarga proyectos y extrae su historia de commits, incluyendo la información del código fuente de manera granulada y el tiempo de desarrollo en reparación de errores. Lo que hace esta herramienta es recuperar estadísticas globales de la evolución del proyecto que consisten en número de commits, fechas de commits, cantidad de autores asociados, y qué commits están involucrados en errores, además rastrea el cambios de elementos del programa de particular interés a lo largo del tiempo. Soporta cuatro lenguajes: C, C ++, Java y Python.

Conclusiones

Este documento ofrece ejemplos muy simples, creemos que esos ejemplos son lo suficientemente representativos como para aumentar la representación de la historia sobre proyectos y su control de versiones. Como se mencionó anteriormente, un criterio clave es definitivamente la velocidad de la respuesta. Si una verificación después de un commit lleva más de un par de minutos, definitivamente se vuelve poco interesante (incluso 1 minuto puede ser demasiado largo), por lo que el tiempo es el criterio de restricción para estas verificaciones. Afortunadamente, las operaciones de git son muy rápidas, por lo que hay muchas verificaciones útiles posibles.

Como parte de la herramienta se podría incluir una sumatoria del factor bus, tal y como sucede en GiLA, pero creemos que este es un buen avance en cuando a la representación visual del progreso del software sobre un proyecto de software y además es importante ver que el método es versátil, podría incluirse en los flujos de trabajo de Integración Continua (IC) del proyecto. Más adelante podría presentarse como una herramienta autónoma en las aplicaciones existentes de IC de plataformas como GitHub, GitLab o BitBucket.

De acuerdo al cumplimiento de los objetivos, por cuestión de tiempo y simplicidad, los prototipos y la herramienta como tal quedó bastante simple, pero es satisfactorio saber que cumple con el objetivo general del proyecto que pretendía construir un prototipo novedoso de un sistema de analítica visual orientado al análisis de la evolución de software. Se cumplieron a cabalidad los objetivos, desde el análisis de estrategias, hasta evaluar la efectividad del prototipo no a razón de rendimiento, como se menciona en los resultados, sino a nivel de metodología.

Referencias

Automate Python workflow using pre-commits: black and flake8. (s.f.). https://ljvmiranda921.github.io/notebook/2018/06/21/precommits-using-black-and-flake8/. Descargado de `https://ljvmiranda921.github.io/notebook/2018/06/21/precommits-using-black-and-flake8/` (Accessed on Thu, June 25, 2020)

Commits Convencionales. (s.f.). https://www.conventionalcommits.org/es/v1.0.0-beta.3/. Descargado de `/es/v1.0.0-beta.3/` (Accessed on Thu, June 25, 2020)

Cosentino, V., Izquierdo, J. L. C., y Cabot, J. (2015, mar). Assessing the bus factor of Git repositories. En *2015 IEEE 22nd international conference on software analysis evolution, and reengineering (SANER).* IEEE. Descargado de `https://doi.org/10.1109%2Fsaner.2015.7081864` doi: 10.1109/saner.2015.7081864

Extending Git Functionality with Git Hooks. (2016, nov). En *Professional git®* (pp. 423–441). John Wiley & Sons Inc. Descargado de https://doi.org/10.1002%2F9781119285021.ch15 doi: 10.1002/9781119285021.ch15

GH Archive. (s.f.). https://www.gharchive.org/. Descargado de https://www.gharchive.org/ (Accessed on Wed, June 24, 2020)

Gousios, G. (2013, may). The GHTorent dataset and tool suite. En *2013 10th working conference on mining software repositories (MSR).* IEEE. Descargado de https://doi.org/10.1109%2Fmsr.2013.6624034 doi: 10.1109/msr.2013.6624034

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., ... Corlay, S. (2016). Jupyter Notebooks-a publishing format for reproducible computational workflows. En *ELPUB* (pp. 87–90).

Kraetke, M. (s.f.). From GitHub to GitHub with XProc: An approach to automate documentation for an open source project with XProc and the GitHub Web API. En *Proceedings of balisage: The markup conference 2016.* Mulberry Technologies Inc. Descargado de https://doi.org/10.4242%2Fbalisagevol17.kraetke01 doi: 10.4242/balisagevol17.kraetke01

The magic notebook for exploring data / Observable. (s.f.). https://observablehq.com/. Descargado de https://observablehq.com/ (Accessed on Thu, June 25, 2020)

Meeks, E. (2015). *D3. js in Action.* Manning Shelter Island, NY.

Mombach, T., y Valente, M. T. (s.f.). GitHub REST API vs GHTorrent vs GitHub Archive.