Los Detalles de Implementación en la Era de las Abstracciones

UNA CHARLA ENTRE YO JEKYLL Y YO HIDE

Por Gunnar Wolf

– Me llamo Gunnar, y soy programador.
– ¡Hola, Gunnar!
– Tengo que contarles hoy respecto a mi abuso de las abstracciones y de la automatización. Por aprovechar las facilidades de un framework completo y poderoso, me he ido olvidando de las interacciones reales. Me engolosiné con un ORM que me presenta los datos como objetos, y me olvido de la realidad de su representación entidad-referencial. Mis métodos son todos cortitos y asumo que el ciclo de atención a solicitudes es corto, olvidando que dejo referencias en memoria. Hago llamadas a procedimientos remotos en proveedores SaaS y no considero que pueden fallar.
– Pero ahora estás consciente de ello. ¡Has dado el Primer Paso! ¡Ese es el camino para ser un programador!
– Sí, pero... ¿Por dónde comienzo? Hay tantas abstracciones, tanto por reaprender. A todos lados a donde volteo a buscar, me encuentro con más abstracciones. ¡Me siento perdido! ¿Dónde quedó la simple realidad del cómputo que teníamos antes?

Gunnar Wolf es administrador de sistemas para el Instituto de Investigaciones Económicas de la UNAM y desarrollador en el proyecto Debian GNU/Linux.
http://gwolf.org

Sí, no hay ni cómo esconderlo. Tiene ya varias décadas que me apasionó comprender lo que pasa dentro de una computadora, cómo opera su aparente magia. Y cuando aprendí, cuando muchos de nosotros aprendimos, incluso un niño de diez años podía comprender cómo operaban ciertos procesos, si bien tal vez no a nivel electrónico, sí a partir de una visión muy cercana. Piensen, por ejemplo, en qué tan a profundidad podía un niño de los ochenta conocer a una magnífica Commodore 64, e incluso cuánto puede haber aprendido con una PC. Recuerdo largas horas de leer documentación, modificar binarios a mano con un editor hexadecimal para cambiar las cadenas que presentaban al usuario, buscando comprender todo lo que veía —simplemente, porque era posible hacerlo.

Con el tiempo, el mundo ha ido cambiando. Y yo también. Al volverme un miembro útil a la sociedad, como les habrá pasado a ustedes también, fui eligiendo mi nicho ecológico: desarrollo de aplicaciones web y administración de sistemas. Primero, exprimiendo los detallitos que me ofrecía cada pedacito del protocolo... Aunque recuerdo muy bien una discusión con un colega: yo era bastante reacio a emplear frameworks de desarrollo precisamente porque, al automatizar las tareas repetitivas, esconden del programador su funcionamiento real, y dificultan tener verdadero control de lo que hacen, pero él me mostró las ventajas que conllevan.

Pero bueno, acortando la historia ... de desarrollar mis aplicaciones completas con el lenguaje Perl, cabalgando a pelo sobre el API de Apache, me subí al tren de Ruby on Rails. Y disfruté muchísimo de las posibilidades que me brindó esta experiencia: una programación más limpia, orientada a objetos. Configuración por convención. Muchas muy buenas prácticas de desarrollo, y un marco de desarrollo con opiniones propias que me marcó cómo estructurar mi desarrollo. Y sí, las aplicaciones resultantes eran comprensiblemente más rápidas de desarrollar, y al tener el comportamiento base resuelto, me topé con muchos menos errores lógicos en las capas más bajas.

Pero la vida sobre un framework también trae sus desencantos. En mi caso, me topé con los primeros al encontrar la cantidad de código que había que reescribir cuando Rails pasó de su versión 1 a 2, de 2 a 3. Como es de esperarse, los cambios que introducen las versiones mayores no son compatibles hacia atrás, y buena parte de mi código histórico iba emitiendo advertencias por usos desaconsejados — O rompiéndose por completo. Y entre más sistemas desarrollaba, menos tiempo tenía para mantenerlos a todos al día.

La respuesta de la comunidad Rails a mis cuitas es relativamente simple. Un sistema que ya está en producción no requiere ser actualizado. El programador puede congelar el sistema base y las gemas (bibliotecas) que emplea, y convivir

> "Los frameworks no únicamente son cómodos, sino que son necesarios para la realidad del desarrollo de software hoy en día".

fácilmente en el mismo sistema con otras aplicaciones Rails — Incluso si estas usan otras versiones de prácticamente todo en el sistema.

En ese momento, comenzó una lucha interior, entre mi Dr. Jekyll, un administrador de sistemas que prepara las cosas y, con la cabeza fría, mantiene una visión consistente y coherente del conjunto, y mi Mr. Hyde, un programador que quiere vivir al límite probando las últimas versiones del último grito de la moda, cambiando de arquitectura subyacente, abandonando a FCGI por Mongrel, a Mongrel por Thin, a Thin por Passenger, incorporando a Rack... Y claro, encarnando a esa aberración de la que hablaremos en otra ocasión que hoy deambula libremente: el temido DevOps, criatura que encarna lo más obscuro tanto de desarrolladores como de administradores.

Y ojo, me centro en este aspecto porque mi Dr. Jekyll es administrador de sistemas. Pueden imaginar lo que diría uno con formación de DBA al ver el caos resultante del ORM: ¿Cómo se crea mi esquema de datos? ¿Dónde están las verificaciones de integridad? ¿Cómo se generan las consultas para cada una de las operaciones que el buen ActiveRecord realiza?

En fin, podemos recordar esa máxima de la ciencia de la computación, que he visto atribuída tanto a David Wheeler como a Butler Lampson: Todos los problemas en la ciencia de la computación pueden resolverse con un nivel adicional de indirección — a excepción del de tener demasiados niveles de indirección.

Mientras esa pelea ocurría en mi yo desarrollador, muchos otros importantes cambios se presentaron en mi vida. Uno de ellos, me convertí en profesor en la Facultad de Ingeniería de la UNAM. Imparto una materia que siempre me emocionó: Sistemas Operativos. He disfrutado muchísimo con la preparación del material y con el dictado de las clases. En esta materia estudiamos las funciones básicas de todo sistema operativo — Comunicación entre procesos, multiplexión de recursos, organización de sistemas de archivos, gestión de memoria, etc.

Conforme estudio y repito mi material, y conforme comento al respecto con mis colegas, vuelvo a uno de los planteamientos de origen que siempre hago a mis alumnos: no espero que de mi clase salgan autores de sistemas operativos; sería un gran orgullo que así fuera, pero no es lo que la materia persigue. Una de las principales razones para estudiar esta materia es que, si no se comprenden los fundamentos de lo que estamos haciendo, nuestros programas van a resultar de muy baja calidad.

Si nos olvidamos que a fin de cuentas todo nuestro código se traduce a instrucciones de muy bajo nivel, si nos olvidamos de cuidar la eficiencia de los cachés y de reducir accesos a disco, si no tenemos en cuenta el costo de cada llamada al sistema, si no consideramos la necesaria sincronización en problemas que enfrentan concurrencia... Vamos a terminar creando código lento y frágil. Vamos a terminar siendo malos desarrolladores.

Entonces bien, el motivo de esta columna no es llamar a que abandonemos las herramientas que facilitan nuestra tarea. Los frameworks no únicamente son cómodos, sino que son necesarios para la realidad del desarrollo de software hoy en día. Pero debemos mantener en mente, siempre, la importancia de comprender qué pasa. Si hay un comportamiento dado que parece mágico, ese es el punto donde debemos revisar el código fuente del framework y averiguar cómo está haciéndolo. Porque sólo de esa forma podremos sacar el provecho correcto del sistema — y escribir mejor código, que a fin de cuentas, por eso nos hacemos llamar profesionales.

Entonces, vestidos de investigadores privados y lupa en mano, ¡vamos a investigar implementaciones! ¡Disfruten el viaje!

Si alguno de ustedes se siente identificado con este disfrute histórico, no puedo dejar de invitarlos a una de las lecturas que más he disfrutado en los últimos meses. Un libro titulado sencillamente «10 PRINT CHR$(205.5+RND(1)); : GOTO 10», y que pueden descargar gratuitamente de *http://10print.org*. Este libro aborda muy distintos aspectos que se desprenden de la línea de BASIC que lleva por título; un análisis técnico, social, cultural, incluso artístico de esa era que a tantos nos atrapó convirtió en lo que hoy somos.

Como segunda nota recomendación literaria, el libro «Lauren Ipsum: A story about computer science and other improbable things» de Carlos Bueno (*http://laurenipsum.org*). Aprender los fundamentos del cómputo siendo aún niño definió mi vida. Este libro presenta, a través de un cuento inspirado en Alicia en el país de las maravillas, conceptos fundamentales de la ciencia de la computación: Problemas clásicos, estructuras de datos, conceptos fundamentales, presentados con el cuento de una niña perdida en el bosque de los árboles rojo-negros, buscando volver a casa.