Chapter 10. Beautiful Features

# Capítulo 10. Hermosos rasgos

Así, esperando tu respuesta, profano mis labios en tu pie, mis ojos en tu cuadro y mi corazón en cada una de tus partes. Tuyo, en el más querido designio de la industria. . .

-WilliamShakespeare, *Love's Labor's Lost*

El año pasado me invitaron a contribuir con un capítulo al libro de Andy Oram y Greg Wilson *Beautiful Code* (O'Reilly), una antología sobre el tema de la belleza expresada en programas informáticos. Quería escribir mi capítulo en JavaScript. Quería utilizarlo para presentar algo abstracto, potente y útil que demostrara que el lenguaje estaba a la altura. Y quería evitar el navegador y otros lugares en los que se encasilla a JavaScript. Quería mostrar algo respetable con algo de peso.

Inmediatamente pensé en el analizador sintáctico Top Down Operator Precedence de Vaughn Pratt, que utilizo en JSLint (véase [el Apéndice C](apc.html)). El análisis sintáctico es un tema importante en informática. La capacidad de escribir un compilador para un lenguaje en sí misma sigue siendo una prueba de la completitud de un lenguaje.

Quería incluir todo el código de un analizador sintáctico que analizara JavaScript, pero mi capítulo era uno de 30 ó 40, así que me sentí limitado en el número de páginas que podía consumir. Otra complicación era que la mayoría de mis lectores no tendrían experiencia con JavaScript, por lo que también tendría que introducir el lenguaje y sus peculiaridades.

Así que decidí subconfigurar el lenguaje. De ese modo, no tendría que analizar todo el lenguaje ni describirlo por completo. Llamé al subconjunto JavaScript simplificado. Seleccionar el subconjunto fue fácil: incluía sólo las características que necesitaba para escribir un analizador sintáctico. Así es como lo describí en *Beautiful Code*:

JavaScript simplificado es sólo lo bueno, incluido:

*Functions as first class objects*

Las funciones en JavaScript simplificado son lambdas con alcance léxico.

*Dynamic objects with prototypal inheritance*

Los objetos no tienen clases. Podemos añadir un nuevo miembro a cualquier objeto mediante una asignación ordinaria. Un objeto puede heredar miembros de otro objeto.

*Object literals and array literals*

Se trata de una notación muy cómoda para crear nuevos objetos y matrices. Los literales de JavaScript fueron la inspiración del formato de intercambio de datos JSON.

El subconjunto contenía lo mejor de las partes buenas. Aunque era un lenguaje pequeño, era muy expresivo y potente. JavaScript tiene muchas funciones adicionales que realmente no aportan gran cosa y, como verás en los apéndices que siguen, tiene muchas funciones con valor negativo. No había nada feo ni malo en el subconjunto. Todo eso desapareció.

JavaScript simplificado no es estrictamente un subconjunto. He añadido algunas características nuevas. La más sencilla fue añadir pi como constante simple. Lo hice para demostrar una función del analizador sintáctico. También demostré una mejor política de palabras reservadas y demostré que las palabras reservadas son innecesarias. En una función, una palabra no puede utilizarse como nombre de variable o parámetro y como característica lingüística. Puedes utilizar una palabra para una cosa o para la otra, y el programador puede elegir. Eso hace que el lenguaje sea más fácil de aprender, porque no tienes que estar pendiente de funciones que no utilizas. Y hace que el lenguaje sea más fácil de ampliar porque no es necesario reservar más palabras para añadir nuevas características.

También he añadido el ámbito de bloque. El ámbito de bloque no es una característica necesaria, pero no tenerlo confunde a los programadores experimentados. Incluí el alcance de bloque porque preveía que mi analizador sintáctico se utilizaría para analizar lenguajes que no son JavaScript, y esos lenguajes harían el alcance correctamente. El código que escribí para el analizador sintáctico está escrito con un estilo al que no le importa si el ámbito de bloque está disponible o no. Te recomiendo que tú también escribas así.

Cuando empecé a pensar en este libro, quería llevar la idea del subconjunto más allá, mostrar cómo tomar un lenguaje de programación existente y hacerle mejoras significativas sin hacer más cambios que excluir las características de poco valor.

Vemos muchos diseños de productos basados en características en los que el coste de éstas no se tiene debidamente en cuenta. Las funciones pueden tener un valor negativo para los consumidores porque dificultan la comprensión y el uso de los productos. Estamos descubriendo que a la gente le gustan los productos que simplemente funcionan. Resulta que los diseños que simplemente funcionan son mucho más difíciles de producir que los diseños que reúnen largas listas de características.

Las características tienen un coste de especificación, un coste de diseño y un coste de desarrollo. Hay un coste de prueba y un coste de fiabilidad. Cuantas más funciones haya, más probable será que una desarrolle problemas o interactúe mal con otra. En los sistemas de software, hay un coste de almacenamiento, que se estaba volviendo insignificante, pero que en las aplicaciones móviles está volviendo a ser significativo. Hay costes de rendimiento ascendentes porque la Ley de Moore no se aplica a las baterías.

Las funciones tienen un coste de documentación. Cada característica añade páginas al manual, lo que aumenta los costes de formación. Las funciones que ofrecen valor a una minoría de usuarios imponen un coste a todos los usuarios. Por eso, al diseñar productos y lenguajes de programación, queremos que las características principales -las partes buenas- sean las correctas, porque es ahí donde creamos la mayor parte del valor.

Todos encontramos las partes buenas en los productos que utilizamos. Valoramos la simplicidad, y cuando no se nos ofrece simplicidad, la creamos nosotros mismos. Mi horno microondas tiene montones de funciones, pero las únicas que utilizo son cocinar y el reloj. Y ajustar el reloj es una lucha. Afrontamos la complejidad del diseño basado en funciones encontrando y quedándonos con las partes buenas.

Estaría bien que los productos y los lenguajes de programación se diseñaran para que sólo tuvieran partes buenas.