Portafolio del Mes - Noviembre

Autor: Vicente Fuenzalida Marín
Fecha de Entrega: 29 de noviembre de 2017
Tema: Frameworks en el desarrollo de software moderno.

Frameworks: los nuevos *lenguajes de programación*

El artefacto que escogí esta semana es el artículo *7 reasons why frameworks are the new programming languages* del sitio InfoWorld, que habla de la importancia de los frameworks en el desarrollo de software en la actualidad.

Artefacto

En el artículo se habla de los distintos cambios que han habido en el desarrollo de software con respecto a los desafíos que hay, las nuevas herramientas que apoyan al desarrollo, y los frameworks como principal soporte para desarrollar aplicaciones. El primer punto mencionado es la existencia de herramientas que hacen muchísimo más fácil programar, independiente del lenguaje utilizado, y que nos ayudan a corregir código, completarlo, ordenarlo, e incluso intercambiar lenguajes (cross-compilers), todo de forma casi automática. Con todas estas herramientas, ya no es necesario "dominar" un lenguaje de programación para poder desarrollar software con él, sino que solo hay que saber escoger correctamente las herramientas adecuadas (IDEs, linters, plugins, etc).

A pesar de lo anterior, siguen existiendo desafíos debido a que el software se vuelve cada vez más complejo, y es aquí donde aparecen los frameworks. El artículo plantea que hoy en día los frameworks son el *equivalente* moderno a los lenguajes de programación, ya que es imprescindible entenderlos y aprender a usarlos para desarrollar software "para hoy".

Para justificar la aseveración de que "los framewokrs son los nuevos lenguajesde programación" se dan algunos motivos:

La mayoría del desarrollo consiste en conectar APIs.

Se comenta que en algún momento escribir software significó demostrar todo el conocimiento que se tenía del lenguaje de programación (manejar punteros, scopes, funciones, etc), pero que ahora herramientas como el compilador o el garbage collector se encargarán de remover código muerto, espacio sin liberar, etc. En

resumen, ahora se trabaja a más alto nivel de abstracción que antes, se escribe menos código que interactúa con el hardware y más código que hace llamadas a otras APIs (de librerías o servicios).

Se puede depender y confiar en los *gigantes* (desarrolladores de los frameworks).

Aquí se argumenta que el uso de frameworks ahorra muchísimo tiempo y trabajo, a pesar de que a veces no son del todo compatibles con los proyectos. En resumen, resulta más fácil adaptarse al framework escogido, que desarrollar todo el código desde cero (para aplicaciones grandes).

Importa conocer la arquitectura más que la sintaxis correcta.

Al trabajar con frameworks, el código escrito dificilmente será complejo, en el sentido de tener que recurrir a los detalles específicos del lenguaje (funcionalidades "raras"). Por otro lado, aprender cómo funciona el framework de forma global puede ser muchísimo más provechoso para lograr resultados valiosos, y en cambio concentrarse en cómo se está ejecutando un algoritmo en una determinada librería puede no ser tan útil (en términos de tiempo).

Los algoritmos son lo difícil.

Los frameworks incorporan muchos algoritmos estándar y estructuras de datos que permiten evitar "reinventar la rueda", de forma que el programador se concentre únicamente en desarrollar la lógica de negocios más que los detalles "no funcionales".

Los compiladores e IDEs inteligentes corrigen tu sintaxis

Lo mismo que se señaló al principio. Con la automatización que entregan algunas herramientas, ya no es tan importante ser riguroso con la sintaxis del código (hasta cierto punto) ya que estas herramientas harán el trabajo por nosotros (completar ";", "{}", tipos, etc). Los IDEs incluso son capaces de señalar cuando faltan parámetros en una función o si hay referencias que no existen, variables no declaradas, etc, por lo que esto ha dejado de ser el problema que alguna vez fue (y que más de alguna vez ha provocado pérdidas muy grandes de dinero).

Reflexión

Si bien concuerdo en varios puntos con lo que se plantea en el artículo, hay varias cosas en las que difiero. En primer lugar, y basado en experiencia propia, me he dado cuenta de que los IDEs, plugins, compiladores, etc. permiten agilizar mucho el desarrollo y despreocuparse de los detalles de sintaxis y lenguaje. Existen herramientas que ayudan de varias formas:

- corrigen código
- lo dejan ordenado y bonito
- reemplazan funciones por otras más cortas y eficientes
- eliminan código muerto
- avisan cuando falta importar librerías o definir variables
- están integrados con los VCSs (git)
- sugieren y ayudan a autocompletar
- mucho más!

Aún con todo esto, creo que es fundamental entender el lenguaje que se está utilizando para aprovechar al máximo su funcionalidad (por ejemplo dependiendo de si es funcional u OOP), y además se debe tener algún conocimiento de teoría de computación si se desea elaborar componentes eficientes y efectivos (no tan avanzado, pero comprender estructuras de datos).

Con respecto a las tareas que realiza un desarrollador hoy, siento que hay una tendencia a utilizar APIs externas más que a desarrollar código propio debido a la gran cantidad de servicios y librerías que se encuentran disponibles (entendiendo que el uso de un paquete/librería cuenta como uso de una interfaz de aplicación).

Como todo el *trabajo de bajo nivel* lo realizan otras librerías (que uno simplemente importa y usa), el nivel de abstracción en el que se trabaja es muy alto, por lo que el código que el *consumidor* de estas APIs escribirá será más bien consiso y legible (si se utilizan correctamente). Como resultado, se incurre poco en el uso de sintaxis compleja y funcionalidades 'rebuscadas' del lenguaje de programación, por lo que no se requiere tanto conocimiento de este, sino del framework que orquestará todas las conexiones entre componentes.

Por último, y refiriéndome al punto de "confiar en los gigantes", me parece que debe ser sujeto a criterio del usuario, pero es relativamente cierto: podemos confiar en los frameworks (al menos los más desarrollados y respaldados). Esto obviamente dependerá del contexto y del proyecto a desarrollar, pero la mayoría de las veces se recurre a patrones y a arquitecturas conocidas, por lo que confiar en un framework puede ser una alternativa muchísimo más segura que intentar implementar dichos patrones/arquitecturas por cuenta propia (más propenso a errores, pero más adaptado al proyecto).

Concluyo diciendo que no estoy de acuerdo del todo con el artículo, debido a que se está encasillando el desarrollo de software a una única área particular (aplicaciones que conectan APIs) en vez de considerar otros tipos de proyectos que son mucho más *únicos* (que deben interactuar con hardware o proveer funcionalidades no cubiertas por ninguna librería existente). En lo que si concuerdo es en que hoy en día las herramientas de apoyo para desarrollar software son tan completas y automatizan tantos procesos que el desarrollo mismo requiere mucho menos conocimiento y prolijidad (en cuanto a sintaxis y dominio del lenguaje).