

# Shibumi

José Galaviz





# Programar

¿Ingeniería o  
artesanía?



¿Es programar una labor de ingeniería o una labor artesanal? El título de una de las obras más célebres sobre el tema de la programación parece sugerir que se trata de una labor del segundo tipo. Me refiero a la serie de libros que escribe Donald Knuth desde 1962: *The Art of Computer Programming* y cuyos primeros tres volúmenes aparecieron en 1968, 1969 y 1973. Es este un debate interesante que pareció haberse resuelto a favor de la ingeniería, pero que desde hace algunos años ha recuperado vigencia.

# Ingeniería = Progreso



1933



2013

A principios de la década de los 30 en el siglo pasado, mi abuelo, que era chofer, conducía el Packard de su jefe por la ciudad de México. Hasta sus últimos días mi abuelo recordó ese automóvil. Para él había sido un privilegio conducir lo que él consideraba una obra de arte. Con un tablero de madera esmeradamente esculpido por un ebanista y el ajuste mecánico perfecto logrado por manos expertas. Hoy en día, 80 años después, yo transito en un Ford Focus. ¿Cuál es la diferencia?



## La diferencia...

De haberse estrellado mi abuelo en el Packard a 60 km/h probablemente habría salido lesionado, o... a lo mejor no. El Packard, con sus 12 cilindros, consumía una cantidad enorme, aunque desconocida, de combustible y nunca sabremos cuántos gases nocivos emitía. La diferencia, luego de 80 años, es esa: GARANTÍAS. Hoy yo sé que estrellándome de frente a 60 Km en el Focus, muy probablemente no quedaría lesionado; que corro un 20% de probabilidades de que eso ocurra si el choque es lateral, que cada litro de combustible rinde unos 12 Km, puedo conocer mis emisiones. Esas son las ventajas de que el automóvil sea producto de un estricto y bien regulado proceso de ingeniería.

# Ingeniería de software

1968 y 1969 Conferencias convocadas  
por el *NATO Science Committee*.

En 1968 y 1969, la OTAN organizó sendas conferencias a propósito del proceso de elaboración de software. La situación había entrado en una crisis. Al menos así fue bautizada la situación en la primera de estas conferencias.

# Crisis del software



Desempeño del hardware.

Desempeño del software.



Mientras el hardware, apegado a la *Ley de Moore*, poseía un desempeño exponencialmente creciente, la situación con el software, esa segunda, pero igualmente indispensable parte del binomio de un sistema de cómputo útil, era completamente contraria. Habían ocurrido incidentes desagradables. Como muestra, el 22 de julio de 1962 el cohete que transportaba la sonda Mariner I destinada a explorar Venus tuvo que ser destruido en el aire luego de que el centro de control perdió el control de la nave.

Software

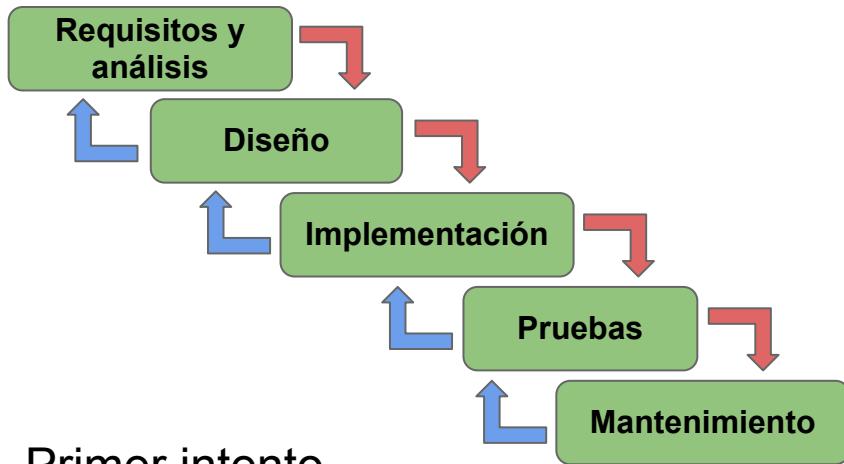
**EXCESO**

**de tiempo  
y de costo**

**SIN GARANTÍAS**

Los proyectos de software no se terminaban en plazo.  
No se ajustaban al presupuesto.  
El producto no poseía la calidad necesaria.  
No cumplía con las especificaciones.  
El código era difícil de mantener, de extender y de corregir.

# Proceso de ingeniería



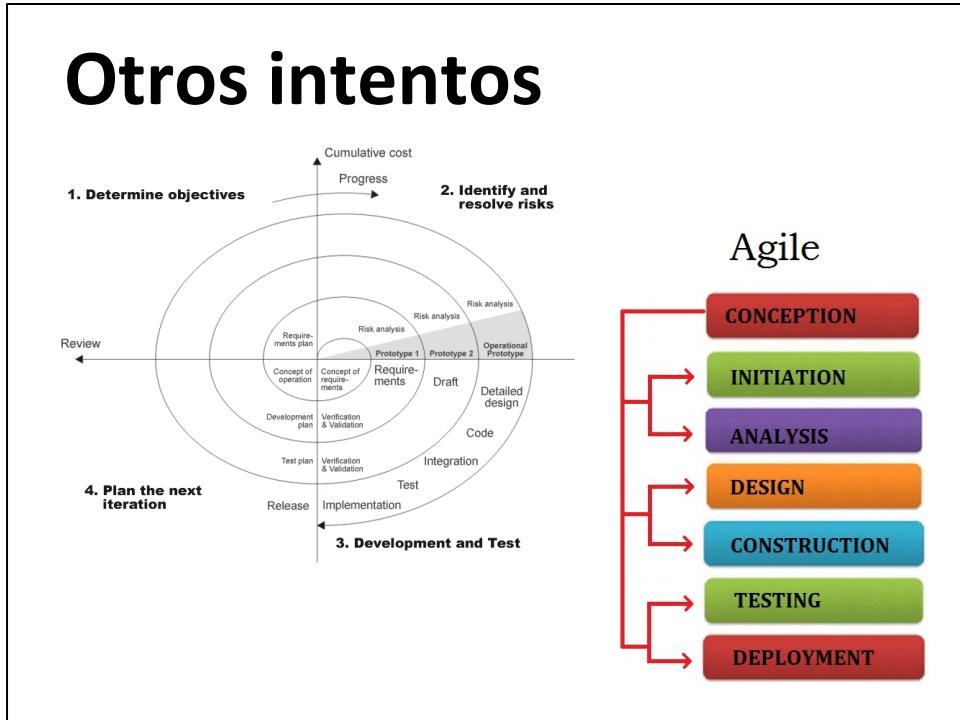
Primer intento

En respuesta a esta situación surgió la *Ingeniería de Software*. Lo primero que habría que hacer es regular el proceso de manufactura. Hacerlo como el de cualquier producto de ingeniería. Así surge el modelo del *Ciclo de Vida del Software*. Primero como una secuencia lineal de pasos a seguir (flechas rojas en la diapositiva), lo que resultó demasiado rígido. Luego, en una segunda versión, como una secuencia con retornos a etapas previas (flechas azules de la diapositiva), para hacer posible la retroalimentación, para corregir lo hecho en etapas previas.

Un **modelo** que,  
de seguirlo  
**estrictamente**,  
nos lleve  
**indefectiblemente**  
al resultado esperado.

La intención era establecer claramente cuales deben ser los pasos del proceso de elaboración del software y lo que debe hacerse en cada uno para obtener al final un producto de *calidad, confiable* y elaborado en *tiempo y forma*. Sin embargo el modelo resultó muy poco útil, aún con la adecuación de los retornos. Tratar de aplicarlo resultaba, salvo en el caso de los programas más simples, en la imposición de una metodología tan artificial que en los hechos no se utilizaba.

# Otros intentos



Surgieron entonces muchos otros modelos diferentes, ya sea para tratar de modelar lo que realmente se hace o para tratar de establecer lo que debería de hacerse. Hoy en día hay varios más o menos exitosos en cuanto los resultados que arroja su aplicación, pero no hay ninguno que pueda decirse *el mejor*, o el que pueda aplicarse *siempre*.

# **Pero seguir el camino...**



**no garantiza llegar adonde queremos.**

Sobre todo, hasta ahora, no hay una metodología tal que, de aplicarla a pie juntillas, indefectiblemente nos lleve a obtener un producto de la más alta calidad.

**La ingeniería no necesariamente**



**produce calidad**

Pero bueno, además en el resto de la industria, tampoco es garantía de calidad el que la elaboración de un producto esté respaldada por un sólido proceso de ingeniería. Lo que ocurre, realmente, es que la aplicación de un proceso ingenieril a la elaboración de un producto, hace que este tenga propiedades garantizables, a lo mejor no es extremadamente durable o fuerte, pero tiene una durabilidad y una fortaleza mínimas garantizables y probablemente con cotas superiores bien establecidas. La buena calidad y la ingeniería no están indefectiblemente enlazadas. Las *garantías* de calidad y la ingeniería sí lo están. Para poder asegurar el nivel de calidad de un producto es necesario que tenga detrás el respaldo de un proceso medible, reproducible y confiable.

Entonces...  
¿Es cierto que  
**programar** es  
una labor  
**puramente**  
de **ingeniería**?

Así que entonces ¿cuál podríamos decir que es la verdadera naturaleza de la actividad de programar: ¿es una labor puramente ingenieril? Veamos.  
No es infrecuente escuchar que la programación se aprende programando. Esto es cierto y además es una de las características esenciales de las labores artesanales y no es la única de ellas que la programación comparte con la carpintería o la herrería.

observar,  
aprender,  
hacer



Precisando:

1. Se aprende a hacer haciendo. En efecto, poco se puede avanzar en la labor de programación si no se practica asiduamente. Si regresamos a un programa hecho por nosotros mismos cuando comenzamos a programar, nos damos cuenta de lo que hemos aprendido. No parece mal hecho, desordenado, hasta sucio.
2. Se aprende mucho viendo lo que un experto hace. Todos los que programamos hemos aprendido leyendo el código escrito por alguno de nuestros gurús; les copiamos recetas, expresiones, técnicas.
3. Existen situaciones frecuentes en las que el artesano debe, sin mayores consideraciones metodológicas, hacer algo guiado sólo por su intuición o por un cierto tipo de juicio estético. ¿Cantas veces no nos ha ocurrido que terminamos una rutina (método, función) y cambiamos su código hasta que nos parece “bonito”? , ¿cuantas veces no nos ha pasado que aquella función que nos parecía fea es la que tenía el error?



La calidad...

está en los **detalles**.

Sólo un programador experto, como un carpintero experto, sabe cómo cuidar los detalles que hacen de su trabajo un gran trabajo, un trabajo de calidad. El ensamblaje mostrado en el cajón de la diapositiva es llamado en español “cola de pato” o “dovetail” en inglés, es considerado una muestra de experiencia y buena calidad en el trabajo de un carpintero; nadie lo ve, está normalmente oculto en los lados del cajón. Es un detalle que, aunque permanezca invisible, muestra el cuidado puesto por el artesano en la obra completa.

# Sólo un experto artesano



## logra la perfección

Es este el *quid* de la calidad verdadera. El que un artesano experto sepa traducir su habilidad y conocimiento formal y empírico de su disciplina, en pequeños detalles que la transforman en una obra útil, perdurable, confiable y elegante.

**“La belleza es verdad,  
la verdad es belleza.  
Eso es todo lo que  
sabemos  
en la tierra,  
y es todo lo que  
necesitamos saber”**

- John Keats



Cuando ingresé a la Facultad de Ciencias, como físico en 1987, rápidamente aprendí algo muy útil. Cuando nos dejaban una tarea que consistía en obtener la expresión analítica de la fuerza que actuaba sobre..., o la magnitud del campo que se producía por... o el periodo de tiempo que transcurre entre... y como resultado obtenía una larga expresión algebráica, desagradable de leer y de interpretar, sabía que estaba mal. La verdadera respuesta tenía que ser la más simple, la más simétrica, la más *bonita*, podríamos decir.

El nautilus es un molusco céfalo-podo que ha permanecido casi sin cambios evolutivos desde hace unos 500 millones de años. Su diseño es eficiente, perfectamente adaptado a su medio, elegantemente simple, bello. La forma de su concha obedece una muy simple ecuación llamada espiral logarítmica.

Algo similar ocurre con los objetos hechos por el hombre. Casi siempre los más útiles, los que mejor cumplen su cometido, los que no requieren mayor cambio o adecuación para funcionar, son también los más sencillos y los más bellos.

# ¿Qué es un **buen** programa?

Desde el punto de vista artesanal ¿qué es un buen programa?, ¿qué características debe poseer?

Para nombrarlas usaré los términos que designan las cualidades de la estética japonesa zen.

## **Entidad inacabada**

**Fukinsei**



Fukinsei: la característica de la imperfección, la asimetría o lo no acabado. En efecto, los grandes programas suelen nunca estar acabados. constantemente se renuevan se liberan nuevas versiones de ellos. Nunca son perfectos, siempre se perfeccionan. Generaciones de programadores pasan por ellos (por ejemplo Linux, Emacs, el compilador de GNU). Como la Mona Lisa, en la que Da Vinci trabajó a lo largo de su vida sin jamás decidir que había terminado.



**Austero y simple**

## Kanso

Kanso: austerdad, simplicidad.

Un buen programa no desperdicia recursos, es austero, tanto como sea posible, en su uso de memoria, disco, procesador. Nada en él es superfluo o innecesario, tiene sólo el código necesario, sólo las estructuras de datos necesarias. Cuando obtenemos una primera versión funcional de un programa, procedemos a quitar código, a minimizar funciones, a simplificarlas. El mejor código no se obtiene agregando, sino quitando.

Kanso es máxima efectividad con mínimos recursos.

**Con el tiempo no  
pierde utilidad. Se  
adapta.**



**Koko**

Koko: atemperamiento, ganar dignidad con el tiempo.

Un buen programa debe ser fácil de adecuar a nuevas circunstancias, se va adaptando con el tiempo. No pierde utilidad, gana respetabilidad. Como el viejo Emacs o Awk.

El descorchador que usamos en la familia es muy antiguo, sigue sirviendo mucho mejor que los modernos, la pátina que el tiempo le ha concedido lo hace único.

**Sirve para lo que  
fue hecho, sin  
pretensiones ni  
adornos.**



**Shizen**

Shizen: naturalidad sin pretensiones ni adornos, expresa con naturalidad aquello para lo que existe.

Al leer un buen programa se comprende el proceso que lleva a cabo. Expresa el algoritmo que implementa de manera natural, sin rodeos inútiles ni exageraciones rimbombantes.

La mesa es... una mesa, no pretende ser nada más. Logra ser una mesa perfecta sin requerir nada más.

No programamos soluciones rebuscadas, no exageramos. Sí el problema se resuelve con un breve programa en *shell script*, no hacemos ocho clases en Java.

# Profundidad no evidente



## Yugen

Yugen: profundidad más allá de lo evidente, la simplicidad oculta una sutil complejidad.

Un buen programa se ve simple, sus abstracciones ocultan la complejidad mediante encapsulación. En cada nivel de abstracción e igualmente simple, en cada uno se han cuidado los detalles y se esconde la complejidad.

Como una katana japonesa, su proceso de elaboración exige cuidar muchos y muy complejos detalles que permanecen ocultos al que observa la perfección simple de su hoja y el patrón sobre el acero.

Yugen es la profundidad envuelta en el misterio.

**Libertad, sus partes son  
versátiles.**



**Datsuzoku**

Datsuzoku: desapego, libertad, no se atiene a las ataduras de lo convencional. El programa está constituido de módulos bien hechos, que pueden, con libertad ser usados de manera independiente o en diferentes contextos, no están atados indisolublemente al programa, pero cuando trabajan juntos lo hacen como unidad homogénea. En el diseño del programa se procuró obtener la mejor solución, acorde al contexto, sin hacer uso de recursos subóptimos sólo por el hecho que estaban disponibles. El programa, por otra parte, es una obra original, la solución es creativa. Las navajas del ejército suizo son un conjunto de módulos independientes, cada uno de ellos funcional *per se*, que se conectan en una unidad de funcionalidades relacionadas: todo lo necesario para ir de campamento, o para abrir una computadora o para la vida cotidiana en la oficina. Pero cada una de ellas podría estar en otro modelo. Es una solución creativa hecha de muchas pequeñas soluciones independientes.

**Confiable**



**Seijaku**

Seijaku: confiabilidad, tranquilidad.

El buen programa ha sido probado concienzudamente, es confiable. Se puede depender de él, proporciona tranquilidad a sus usuarios.

Casi cualquier instrumento de escritura puede fallar... pero un lápiz, NO y mucho menos sin remedio, bastará con sacarle punta.

# Lo realmente cierto...

es que en la programación, la ingeniería y la artesanía

- No se contraponen.
- Para lograr la mejor calidad se requieren ambas.

Podemos sintetizar lo dicho declarando que:

- Ingeniería y artesanía no son dos conceptos contrapuestos. Ambos se benefician de la existencia del otro. Los mejores resultados se obtienen de la sinergia entre ambos.
- El potencial de la programación como labor de cuidadosa artesanía, se libera a nivel personal. Cada programador debe aspirar a ser un artesano experto.

## Ingeniería



## Artesanía



**Nivel macro**

**Nivel micro**

- El potencial de la programación como producto de un proceso de ingeniería, se libera a nivel de equipo. Es impensable hacer un sistema más o menos complejo sin la colaboración de múltiples programadores apegados a una metodología, y con la cuidadosa planeación típica de una obra ingenieril.
- En el plano individual cada artesano-programador aspira a producir programas simples, confiables, que perduren, que crezcan, se extiendan y sigan siendo útiles con los años.
- Estas últimas son las cualidades de la estética zen.

**¡Gracias!**