¿ES LA INGENIERIA DEL SOFTWARE UNA INGENIERIA MADURA?

GUILLERMO LONDOÑO ACOSTA

Magíster en Ingeniería de Sistemas de la Universidad del Valle. Magíster en Física del Instituto Venezolano de Investigaciones Científicas. Físico de la Universidad del Valle. Profesor del ICESI.

INTRODUCCION

La construcción en 1946 del primer computador electrónico de uso general que funcionó satisfactoriamente, el ENIAC, se puede tomar como el punto de partida de la historia del hardware y el software, esto implica cincuenta años de desarrollo en estas dos áreas. Durante los primeros años del hardware, el principal reto era incrementarlo de forma que se redujera el costo de procesamiento y almacenamiento de datos, lo cual se logró a lo largo de la década de los años 80 con los grandes avances en microelectrónica. Hoy el problema es diferente, el principal objetivo es producir software de calidad, reusable, económico, fácil de mantener y confiable, pero aún no se ha logrado un avance comparable al de la microelectrónica que permita al software alcanzar los niveles del hardware.

Este desbalance entre los niveles de desarrollo del hardware y software se conoce en la literatura como «la crisis del software». Esta crisis se caracteriza por el hecho de que frecuentemente la calidad del software no es la adecuada y el usuario queda insatisfecho con el producto desarrollado, la productividad de la gente que lo desarrolla no se corresponde con la demanda de sus servicios y el mantenimiento consume demasiados recursos económicos y humanos. Cincuenta años después, muchas de las instrucciones de los programas se confeccionan todavía artesanalmente a mano con lenguajes de programación primitivos y usando técnicas que no se someten a evaluación ni pueden repetirse con consistencia; además, cada programador empieza prácticamente de cero en cada programa.

Frecuentemente, uno se pregunta por las causas que produjeron esta crisis y el objetivo de este artículo es hacer una reflexión y tratar de encontrarlas, comparando la forma de trabajo de un ingeniero de hardware y un ingeniero de software.

CAUSAS DE LA CRISIS

Para encontrar las causas de los problemas enunciados antes, veamos cuál es el ciclo de vida seguido tanto por un producto software como por un producto hardware y establezcamos la diferencia, en cuanto a la fundamentación, téc-

nicas y herramientas utilizadas tanto por los ingenieros de hardware como por los ingenieros de software.

Las fases del ciclo de vida que siguen ambos productos son:

Análisis de requisitos del producto: Se encuentran los requerimientos detallados del producto a desarrollar (hardware o software) como son las entradas, salidas, funciones a realizar, la información a manejar, el rendimiento y las interfases requeridas. Con esta información se construyen los modelos conceptuales de especificación del producto.

Diseño del producto: Con base en el modelo de análisis se hace el diseño arquitectónico, procedimental o modular y el diseño de las interfases con los demás elementos.

Prototipado: Con base en el diseño se construye un prototipo del producto final que facilite la creación de un modelo del producto a construir y que sirva para evaluar y refinar los requisitos. Se produce un proceso interactivo en el que se afina el prototipo para que satisfaga las necesidades del usuario.

Implementación: Se construye finalmente el producto y éste puede ser tangible, en el caso del hardware, o intangible, en el caso del software.

Prueba: Una vez construido el producto se hacen las pruebas a cada uno de sus módulos individuales, luego se prueba la interacción entre los módulos y por último se prueba el producto como un todo.

Mantenimiento: Los productos indudablemente sufren cambios después que se entregan al usuario, debido a que se encuentran nuevos errores, o a que el producto debe adaptarse a cambios de su entorno, o debido a que el usuario requiere una ampliación funcional o una mejora en el rendimiento del producto.

Para la fase de análisis, podemos decir que tanto el ingeniero de software como el de hardware utilizan técnicas gráficas apoyadas en texto y se fundamentan en modelos fenomenológicos o heurísticos para desarrollarlas. No habría entonces ninguna diferencia en esta primera fase.

En la fase de diseño empiezan a notarse diferencias drásticas en cuanto a la fundamentación utilizada por los dos tipos de ingenieros. El ingeniero de hardware utiliza la electrónica para diseñar sus circuitos y esto le permite construir modelos matemáticos precisos, los cuales a su vez le facultan para predecir y validar el comportamiento del circuito a construir, ya que la electrónica se fundamenta en la física moderna y la teoría electromagnética, lo que le da un piso científico de trabajo muy firme; o sea que desarrolla una labor de verdadera ingeniería, mediante la aplicación de principios científicos para la elaboración de un producto. Si miramos la forma de trabajo de un ingeniero civil al construir un puente, o la de un ingeniero mecánico al diseñar una caldera, todos se basan en principios científicos, o usan métodos basados en principios y leyes científicos para diseñar y construir sus productos.

¿Pero, en qué se basan muchos de los ingenieros de software cuando diseñan? Por lo general, en métodos informales no cuantificables, o en el peor de los casos, en el sentido común, en su capacidad lógica, en su experiencia, etc. lo cual implica que está limitado por sus habilidades personales y el alcance que le brinde la tecnología de la cual dispone en el momento. Por eso se dice muchas veces que el desarrollo de software es más una labor artística que ingenieril. Esta diferencia en la fundamentación de su trabajo marca mucho la calidad del producto que obtienen ambos tipos de

ingenieros. Además, las ingenierías maduras tienen manuales de soluciones bien probadas, de forma que incluso ingenieros poco experimentados pueden abordar los diseños rutinarios con garantía de éxito. No existen manuales semejantes para la programación por lo que los errores se repiten año tras año, en un proyecto tras otro.

En la fase de prototipado, también existen diferencias grandes. Para construir un prototipo y el producto final, el ingeniero de hardware toma un catálogo de partes y elige las que le den la funcionalidad buscada de acuerdo con el diseño. Algunas de estas partes pueden ser un circuito integrado que tiene una función e interfaz bien definida y un conjunto estándar de criterios de integración. Estos circuitos integrados pueden implementar una función compleja del sistema; o sea que el ingeniero de hardware construye su prototipo y su producto final de forma modular, empleando elementos verdaderamente complejos. Por último, en un protoboard conecta los diferentes elementos y empieza a validar el prototipo.

Por el contrario, el ingeniero de software no dispone de un catálogo de partes, ni de elementos complejos que le permitan ensamblar un prototipo rápidamente, el software se desarrolla como una unidad completa y no como componentes que puedan reensamblarse. Se pretende que la tecnología orientada a objetos logrará que los ingenieros de software puedan ensamblar partes complejas de diferentes fabricantes y obtener así el producto software deseado. Esto permitiría el desarrollo de software muy complejo a partir de elementos complejos y no de instrucciones y tipos de datos simples, de manera análoga a la forma de trabajo de los ingenieros de hardware, quienes no construyen sus circuitos transistor por transistor.

Sobre el prototipo del circuito se hacen ahora todo tipo de pruebas, tanto modulares como globales, para determinar si satisface los requerimientos exigidos en las fases de análisis y diseño. Validado el prototipo, el ingeniero de hardware lo pasa a una tarjeta optimizando las conexiones entre sus componentes, sus interfaces y todos los elementos que permitan obtener un producto terminado que cumpla con ciertos estándares de calidad. La fase de construcción del hardware puede introducir problemas de calidad que no existen, o son fácilmente corregibles en el caso del software.

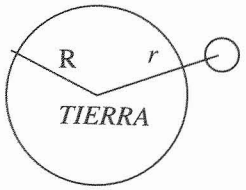
Cuando un componente de hardware se daña, se reemplaza, pero en el caso del software no hay piezas de repuesto. Cada fallo en el software indica un error en el diseño o en el proceso mediante el que se tradujo el diseño a código máquina ejecutable. Por tanto, el mantenimiento del software tiene una complejidad considerablemente mayor que el mantenimiento del hardware. Peor aún, cuando se realiza mantenimiento al software es probable que se introduzcan nuevos defectos haciendo que se deteriore más.

CARACTERISTICAS DEL SOFTWARE

Intangibilidad. El software tiene unas características que lo hacen muy especial y que dificultan su desarrollo. Para determinar estas características, comparemos su comportamiento con el de un sistema físico, mediante un ejemplo sencillo.

Si queremos modelar el comportamiento de un pequeño objeto esférico de masa m, cerca de la superficie de la tierra cuya masa es M, necesitamos apoyarnos en dos leyes físicas fundamentales, la ley de la gravitación y la segunda ley de Newton. La ley de la gravitación nos dice que todo objeto en el

universo atrae a otro objeto con una fuerza directamente proporcional al producto de sus masas e inversamente proporcional al cuadrado de la distancia entre ellos:



$$F = G\frac{Mm}{r^2}$$

donde G es la constante gravitacional y r la distancia entre los centros de los dos objetos. Si el radio de la tierra R es mucho mayor que la distancia entre el objeto y la superficie de la tierra, podemos decir que r es constante e igual a R durante toda la trayectoria del objeto, y por lo tanto la fuerza gravitacional se reduce a F = mg, donde $g = GM/R^2$ es la aceleración constante gravitacional que nos enseñaron en el colegio.

La segunda ley de Newton dice que el pequeño objeto responde a la fuerza que ejerce la tierra sobre él, acelerando en la dirección de la fuerza en una cantidad que es inversamente proporcional a la masa del objeto; o sea, fuerza es igual a masa por aceleración, lo que también nos repetían mucho en el colegio. Recordando que la aceleración es el cambio de la velocidad con respecto al tiempo, tenemos que:

$$m\frac{dv}{dt} = -mg$$

ecuación que representa el modelo matemático del presente problema. Podemos usar el cálculo integral para resolver esta ecuación y obtener así la solución que nos permite predecir el comportamiento del pequeño objeto en todo instante de tiempo:

$$\int_{vo}^{v} dv = -g\int_{to}^{t} dt \qquad \boxed{v(t) = vo - g(t\text{-}to)}$$

donde vo es la velocidad del objeto en el instante de tiempo to. Recordando que la velocidad es el cambio en la posición con respecto al tiempo, e integrando de nuevo, obtenemos:

$$\frac{dx}{dt} = vo - g(t\text{-}to); \qquad \int_{xo}^{x} dx = \int_{to}^{t} (vo - g(t\text{-}to))dt;$$

$$\boxed{x(t) = xo + vo(t\text{-}to)\text{-}(1/2)g(t\text{-}to)^2}$$

donde xo es la posición en el instante de tiempo to.

Veamos la fortaleza del método que hemos seguido. Basados en una ley física universal, la ley de la gravitación, en una ley física de comportamiento, la segunda ley de Newton, y utilizando el cálculo integrodiferencial como lenguaje de expresión y herramienta para la construcción de un modelo matemático, hemos logrado determinar el conjunto de posibles estados, posición y velocidad del objeto y su evolución en el tiempo. Este sistema físico tiene un espectro continuo de estados y su comportamiento también es continuo, a través de estos estados, en el tiempo.

¿Será que podemos seguir el mismo proceso en el caso del software? ¿Será que podemos construir un modelo matemático y a partir de él obtener una solución que nos permita determinar el estado del software en cualquier instante del tiempo?

Para responder estas preguntas, notemos que el código fuente es únicamente una imagen estática de un programa de computadora, y aunque los efectos producidos por el programa suelen ser visibles, el programa en sí no lo es. Si hacemos una comparación con una orquesta, podemos decir que los músicos y los instrumentos son el hardware, el director es el sistema operativo, la partitura es el código fuente y la música es el software. Por eso decimos

que el software es un elemento intangible, que además, no tiene masa, ni carga eléctrica, ni momento dipolar magnético, ni espín, ni volumen, ni color, ni olor, etc.; es decir, carece de propiedades físicas y por lo tanto no obedece a las leyes gravitacionales, ni electromagnéticas. No existen leyes de Newton o ecuaciones de Maxwell que guíen el desarrollo del software, su intangibilidad y falta de propiedades físicas dificulta la creación de directrices y lineamientos fundamentales para encauzar el diseño y la implementación del software. El diseño de programas es comparable con el diseño arquitectónico de edificios en ausencia de gravedad, esto no quiere decir que no existan principios fundamentales en esta rama de la ingeniería; sin embargo, dichos principios y guías deben contemplarse para cada situación en particular.

Comportamiento discreto. Los programas complejos están constituidos por complicadas estructuras lógicas interrelacionadas, a través de las cuales solamente pueden pasar datos de naturaleza específica. Un programa de apenas unos centenares de líneas puede contener decenas de decisiones y permite millares de distintas rutas de ejecución. El conjunto de valores, en un momento dado, de todas las variables de un programa y la dirección de ejecución de cada uno de los procesos del mismo, constituyen uno de sus estados. Al ejecutarse el software en computadores digitales, se tiene un sistema con estados discretos, sometido a transiciones entre estos estados producidas por eventos, que en el caso del software de gestión comercial, se dan asincrónicamente en el tiempo.

Es como si el objeto del ejemplo anterior, sólo pudiera estar en determinadas posiciones con determinada velocidad y no realizara una trayectoria continua en el espacio, a través del tiempo. Si el objeto tuviera este comportamiento discreto no podríamos utilizar el cálculo integrodiferencial para modelar el sistema y obtener una solución a su comportamiento. Es por esto que no podemos utilizar matemática continua para modelar y predecir el comportamiento del software. Necesitamos elementos de la matemática discreta, por ejemplo máquinas de estado finito o redes de Petri, para modelar su comportamiento, pero es difícil modelar el comportamiento de software complejo identificando todos sus estados, transiciones e invariantes en las transiciones, ya que su número es muy grande. Si construimos un modelo simplificado del software complejo, con pocos estados, transiciones, invariantes y eventos, no podemos garantizar ni predecir el comportamiento en el tiempo ante todos los posibles eventos que puedan presentarse. Es probable que varios eventos no tenidos en cuenta, lleven el software hacia un estado inconsistente. Todo esto dificulta la caracterización del comportamiento de los sistemas discretos. Además, la matemática discreta consistente de la teoría de conjuntos y el cálculo de predicados, es una especialidad mucho menos madura que la matemática continua. Pero aún así, es necesario el desarrollo de métodos formales y flexibles para la especificación de software complejo, que permitan representarlo en lenguaje matemático, donde pueda analizarse, validarse y probarse con instrumentos teóricos.

CONCLUSIONES

Entendemos por ciencia el conjunto de principios, leyes y teorías que sintetizan las investigaciones científicas. Se trata de un conocimiento lógico y racional, verificado por el sólo pensamiento lógico, o mediante experimentos contrastados muchas veces y ajustados a una realidad que siempre los verifica. Por otra parte, entendemos por ingeniería la aplicación del conocimiento científico a la creación de planes, diseños y

medios para lograr los objetivos deseados. Podemos decir entonces, que la ciencia es un cuerpo de principios y la ingeniería un cuerpo de métodos basados en los principios de la ciencia. Esto nos lleva a pensar que mientras los ingenieros de software se basan en su experiencia personal, los demás ingenieros se basan en la experiencia de toda la humanidad, representada en los métodos de su ingeniería.

La ingeniería del software se define como la aplicación de métodos sistemáticos, disciplinados y cuantificables al desarrollo, funcionamiento y mantenimiento de programas informáticos. Basados en las definiciones anteriores, vemos que la ingeniería del software no es todavía una ingeniería madura, puesto que la gran mayoría de sus métodos son sistemáticos y disciplinados pero pobres en principios teóricos formales cuantificables. La crisis del software seguirá agudizándose, a menos que se adopten las características de las ramas de la ingeniería, que están firmemente basadas en la ciencia y en las matemáticas. El pensamiento matemático es crítico para el desarrollo productivo de una disciplina de ingeniería.

Infortunadamente, la parte de ciencias de la computación correspondiente al estudio de los algoritmos como sus fundamentos teóricos, análisis formal de complejidad algorítmica, métodos formales para la especificación y validación de algoritmos y máquinas abstractas para su ejecución, no constituyen los fundamentos de los métodos y conjunto de herramientas utilizadas por una gran mayoría de desarrolladores de software; como tampoco lo son muchas veces el álgebra ni el cálculo relacional, cuando se usa tecnología de bases de datos relacionales. Y lo que es peor aún, su formación en la universidad les dejó la idea de que las ciencias de la computación son pura matemática; no se dieron cuenta que la matemática es el lenguaje que usa la ciencia para expresar sus principios y leyes. Nuestro medio está tan alejado de la ciencia que es muy común oír decir a nuestros ingenieros, que la matemática que les enseñaron en la universidad era un mecanismo de gimnasia mental y un medio para obtener capacidad de raciocinio. La verdadera formación de un ingeniero no está en el estudio de la matemática, sino en la comprensión y entendimiento de las leyes y principios de la ciencia en la cual se apoya su ingeniería, expresados en un lenguaje matemático.

Desde los años sesenta se han anunciado muchas innovaciones destinadas a resolver las crisis del software. Se proclamó que la programación estructurada era un cambio de paradigma, luego se han anunciado como solución los lenguajes de cuarta generación, las herramientas CASE y últimamente la tecnología orientada a objetos, pero los resultados no han sido muy halagadores. La pobreza teórica de esta diversidad de métodos y herramientas no les ha permitido lograr su objetivo. Todas estas propuestas impulsan la *construcción de mejores cosas,* pero lo que realmente se necesita para consolidar la ingeniería del software es una propuesta que impulse el *construir mejor las cosas.*

Lo que uno recibe por cada peso invertido en un computador se duplica cada año y medio. El desafío para los desarrolladores de software no sólo es grande sino que va en aumento y para no verse superados por semejante demanda, los ingenieros de software tendrán que cambiar sus métodos de trabajo. La historia nos muestra que cincuenta años no han sido suficientes para que la ingeniería del software alcance la madurez necesaria. No sabemos exactamente cuántos años más se necesiten, para que la ingeniería del software se convierta en la ingeniería madura que la sociedad informática requiere.

COMUNICACIONES Y SOCIEDAD
EL INGENIERO DE SISTEMAS FRENTE AL SIGLO XXI

CARLOS H. ARDILA

JUAN MANUEL MADRID*

Ingeniero de Sistemas del ICESI
Director Red Institucional. Profesor ICESI

El siglo XXI está cada vez más cerca. Los avances en los sistemas y en las telecomunicaciones ponen de manifiesto maneras cada vez más ágiles y eficientes para facilitar las comunicaciones entre los seres humanos. Ante esta gran ola de avances, ¿qué actitud y qué responsabilidades debe asumir el gremio de los ingenieros de sistemas? La intención de este artículo es presentar una visión de los servicios y tendencias que en cuanto a comunicación se están vislumbrando desde ahora, y concluir estableciendo algunas de las acciones que los ingenieros de sistemas debemos tomar para aprovechar esta revolución en beneficio del mundo entero.

LA COMUNICACION ENTRE SERES HUMANOS

De todos los inventos del hombre, los que han causado un impacto más grande, han sido aquellos que han incrementado de manera dramática el ancho de banda (es decir, la capacidad de transmitir información) entre los seres humanos. En un inicio, esta capacidad era muy reducida, puesto que la información sólo se podía transmitir por vía oral. La invención de la escritura ayudó, pero aún así, la producción masiva de material escrito sólo comenzó hasta la invención de la imprenta. Este invento fue el primer salto hacia la masificación de la información. Luego han seguido surgiendo medios de comunicación más ágiles, tales como la radio, el teléfono, el télex, la televisión y la fotocopiadora. Con la invención de los PCs y el fenómeno de la gran expansión de las redes de cómputo, surgen nuevas alternativas de comunicación como el correo electrónico y el web. Todos estos nuevos servicios ponen a disposición del hombre una cantidad cada vez mayor de información, a un costo cada día menor. Nos vamos acercando al día en que las comunicaciones sean virtualmente gratuitas.

El computador por sí solo ha sido un invento que ha cambiado completamen-

* Este artículo está basado en la conferencia del mismo nombre, dictada por el Ingeniero Carlos H. Ardila en el marco del Primer Encuentro de Egresados de Ingeniería de Sistemas del ICESI, el 15 de marzo de 1996. Juan Manuel Madrid compiló y complementó el material de dicha conferencia, para lograr como producto final este artículo.