EL "DESARROLLO DE SOFTWARE" COMO "INGENIERÍA DE SOFTWARE"

Scott F. Schaull
Columbia University, NY
scott.schaull@columbia.edu

(Tipo de artículo: **REFLEXIÓN**. Recibido el 08/06/2011. Aprobado el 20/09/2011)

RESUMEN
El desarrollo de software puede ser predecible y controlable, económico y manejable pero, actualmente, los proyectos software no se suelen ejecutar de esa manera, aunque sea posible lograrlo. En este artículo se trata al desarrollo de software como una emergente y necesaria "Ingeniería de Software" y se describe la necesidad de procesos formativos adecuados desde las universidades para reconocer la práctica del desarrollo de software como una profesión ingenieril.

Palabras clave
Ingeniería, Ingeniería de Software, Desarrollo de Software.

"SOFTWARE DEVELOPMENT" CONSIDERED AS "SOFTWARE ENGINEERING"

ABSTRACT
Software development can be predictable and controllable, economic and manageable, but at present time software projects usually are not run that way, although it is possible to achieve it. In this article software development is approached as an emerging and necessary "Software Engineering", and is described the need of proper learning processes from universities to recognize software development practice as an engineering profession.

Keywords
Engineering, Software Engineering, Software Development.

"DÉVELOPPEMENT DE LOGICIEL" COMME "GÉNIE LOGICIEL"

RÉSUMÉ
Le développement de logiciel peut être prévisible et contrôlable, économique et maniable mais, actuellement, l'ordinaire est que les projets des logiciels ne se réalisent pas de cette manière, en dépit de cette possibilité. Dans cet article on aborde le développement de logiciel comme une émergente et nécessaire « Génie logiciel» et on décrit la nécessité d'avoir des processus éducationnels appropriés aux universités pour reconnaitre la pratique du développement de logiciel comme une profession d'ingénierie.

Mots-clés
Génie, Génie logiciel, Développement de logiciel.

1. INTRODUCCIÓN

Cuando entrevisto a candidatos para puestos de trabajo en programación, una de mis preguntas favoritas es: "¿Cómo describiría su enfoque del desarrollo de software?" Yo les presento ejemplos como el de carpintero, el bombero, el arquitecto, el artista, el escritor, el explorador, el científico y el arqueólogo y los invito a que construyan sus propias respuestas. Algunos candidatos tratan de adivinar lo que quiero oír y usualmente me dicen que se ven a sí mismos como "científicos". Los "programadores" con amplia experiencia me dicen que se ven como soldados o como miembros de un equipo SWAT. Mi respuesta favorita provino de un candidato que respondió: "Durante el diseño del software, soy un arquitecto; cuando estoy diseñando la interfaz de usuario, soy un artista; durante la construcción, soy un artesano; y durante las pruebas unitarias soy, como mínimo, un ¡hijo de puta!".

Me gustaría plantear las siguientes preguntas porque constituyen una cuestión fundamental en nuestro campo de investigación: ¿Cuál es la mejor manera de pensar acerca del desarrollo de software? ¿Es ciencia? ¿Es arte? ¿Es un oficio? ¿Es algo completamente distinto? Preguntas a las que trato de responder en el resto del contenido de este trabajo.

2. "ES" vs. "DEBERÍA"

En las Ciencias Computacionales tenemos una larga tradición debatiendo acerca de si el desarrollo de software es arte o ciencia. Hace más de cuarenta años, Donald Knuth [1] comenzó a escribir una serie de siete volúmenes acerca de esta cuestión. Los primeros tres volúmenes contienen alrededor de 2200 páginas, lo que sugiere que el total de los siete podría ascender a más de 5000. Si eso es a lo que se parece el "*arte*" del desarrollo de software, ¡no estoy seguro de querer saber a qué se parecerá como "*ciencia*"!

Las personas que defienden al desarrollo de software como "*arte*" lo hacen desde el punto de vista de sus aspectos estéticos y argumentan que la ciencia no permite esta inspiración y libertad creativa. Las personas que lo defienden como "*ciencia*" lo hacen desde el punto de vista de que la mayoría de los programas tiene altas tasas de error y argumentan que esa baja confiabilidad es una libertad creativa intolerable que debe ser condenada. Ambos puntos de vista son incompletos y ambos hacen la pregunta equivocada. El desarrollo de software es arte, es ciencia, o es un oficio como la arqueología, el derecho, la sicología, la sociología, la comunicación social y otras actividades. Es no-profesional en algunos sectores y profesional en otros. Tiene que ver con muchas y diferentes cosas, ya que existen muchas personas diferentes desarrollando. Pero la pregunta correcta no es "¿Qué *es* actualmente el desarrollo de software?" sino "¿Qué *debería* ser el desarrollo profesional de software?" En mi opinión, la respuesta a esa pregunta es clara: "El desarrollo profesional de software debería ser *ingeniería*". ¿Actualmente es así? No, pero ¿debería serlo? Sin lugar a dudas que Sí.

3. INGENIERÍA vs. CIENCIA

Con sólo un 40% de los desarrolladores de software con títulos en Ciencias Computacionales y prácticamente ninguno en Ingeniería de Software, no debería sorprendernos encontrar personas confundidas acerca de la diferencia entre Ingeniería de Software y Ciencias Computacionales. La distinción entre estos campos es la misma que en otros campos [2]. Los científicos aprenden lo que es verdadero, cómo probar hipótesis y cómo ampliar el conocimiento en su campo; los Ingenieros aprenden lo que es verdadero, lo que es útil y cómo aplicar el conocimiento bien comprendido para resolver problemas prácticos. Los científicos deben estar actualizados con las últimas investigaciones; los ingenieros deben estar familiarizados con el conocimiento que ha demostrado ser fiable y eficaz. Cuando el científico hace ciencia puede darse el lujo de ser específico y especializado; cuando se hace ingeniería es necesario tener un amplio conocimiento de todos los factores que afectan al producto que se diseña. Los científicos no tienen que ser regulados, porque ellos deben rendirles cuentas a otros científicos; los ingenieros tienen que ser regulados, porque ellos tienen que rendirles cuentas a la sociedad. Una formación científica de pregrado prepara a los estudiantes para continuar sus estudios; una formación en ingeniería de pregrado prepara a los estudiantes para su inmediata incorporación al mercado laboral, después de "*terminar*" sus estudios.

Las universidades otorgan títulos en Ciencias Computacionales y normalmente esperan que sus egresados se desempeñen en puestos de trabajo en desarrollo de software, en el que iniciarán de inmediato la resolución de problemas del mundo real. Sólo una pequeña fracción de los estudiantes en Ciencias Computacionales continúa sus estudios de postgrado, en entornos de investigación acerca de los nuevos avances en el estado del conocimiento en el campo del software y/o los computadores. Esto pone a los estudiantes de Ciencias Computacionales en una tierra tecnológica de nadie. Son llamados científicos, pero en su trabajo realizan funciones que tradicionalmente las ejecutan los ingenieros, pero sin el beneficio de una formación en ingeniería. El efecto sería más o menos el mismo que si se le asignara a un Ph.D. en física el diseño de equipos eléctricos para venta comercial. El físico puede comprender mejor los principios eléctricos que los ingenieros con los que está trabajando, pero su experiencia en construcción de equipos se reduce a la creación de prototipos que se utilizan para progresar en el estado del conocimiento en un laboratorio. Él no tiene experiencia ni está capacitado en el diseño de los equipos robustos y económicos que ofrecen soluciones prácticas en entornos reales. Es de esperar que el equipo diseñado por el Ph.D. funcione, pero tal vez carezca de la solidez que haga posible su utilización por fuera del ambiente seguro de un laboratorio; o el equipo puede usar materiales que son aceptables para un prototipo, pero que son un extravagante derroche cuando las unidades se fabrican por miles.

Situaciones parecidas a este simple ejemplo de física se producen por montones en lo que tiene que ver con el software. Cuando los empleados, formados como científicos computacionales, comienzan a trabajar en sistemas de producción, a menudo diseñan y construyen software que es demasiado frágil para usar en producción, o que es inseguro. Se concentran estrecha y profundamente en consideraciones de menor importancia y excluyen otros factores que son más importantes. Pueden invertir días ajustando a mano un algoritmo de ordenación en lugar de horas usando una librería de código o copiando un algoritmo adecuado de un libro. El típico graduado de Ciencias Computacionales suele necesitar varios años de entrenamiento en el puesto de trabajo para acumular suficiente conocimiento práctico para, mínimamente, desarrollar software de producción satisfactorio.

La falta de un desarrollo profesional no es culpa sólo de los desarrolladores de software. El mundo del software se ha convertido en víctima de su propio éxito. El mercado laboral del software ha estado creciendo más rápido que la infraestructura formativa necesaria para apoyarlo, por lo que más de la mitad de las personas que ocupan puestos en desarrollo de software han sido formados en otros campos relacionados. Los empleadores no les pueden exigir a estas personas que, en sus horas libres, obtengan un título equivalente a un pregrado en ingeniería. Incluso si pudieran, la mayor parte de los cursos disponibles son para Ciencias Computacionales, no para Ingeniería de Software. La infraestructura formativa se ha quedado a la zaga de las necesidades de la industria.

4. MÁS QUE UNA MODA

Algunas personas piensan que "Ingeniería de Software" es sólo una palabra de moda que significa lo mismo que "programación de computadores". Es cierto que la Ingeniería de Software ha sido usurpada, pero un término puede ser objeto de abuso y todavía tener un significado legítimo. La definición del diccionario para "Ingeniería" es: “Aplicación de principios científicos y matemáticos con fines prácticos”, y es lo que la mayoría de los programadores tratan de hacer. Los ingenieros de software aplicamos algoritmos desarrollados científicamente y definidos matemáticamente, métodos de diseño funcional, métodos de aseguramiento de calidad y otras prácticas para desarrollar productos software y servicios [3]. Como David Parnas [4] señala, en otros campos técnicos de la profesión ingenieril se inventaron y asignaron personerías jurídicas y certificaciones para que sus clientes conocieran que estaban calificados para construir productos técnicos. Los clientes del software no se merecen menos.

Algunas personas piensan que tratar como ingeniería al desarrollo de software significa que todos tendremos que usar métodos formales para escribir programas como pruebas matemáticas. El sentido y la experiencia común nos dice que eso es demasiado para muchos proyectos. Otros objetan que el software comercial es demasiado dependiente de las cambiantes condiciones del mercado como para darse el lujo de invertir tiempo en ingeniería. Estas objeciones se basan en una idea estrecha y errónea acerca de la ingeniería. La ingeniería es la aplicación de principios científicos con fines prácticos y, si no se hace así, es mala ingeniería. Tratar de aplicar métodos formales a todos los proyectos de software es tan mala idea como tratar de aplicar *code-and-fix* para desarrollar todos los proyectos.

Tratar al desarrollo de software como ingeniería clarifica la idea de que para diferentes proyectos son apropiados diferentes objetivos de desarrollo. Cuando se diseña un edificio, los materiales de construcción deben ser adecuados para el propósito de la construcción. Es posible construir una bodega amplia para guardar vehículos agrícolas con un metal delgado y sin hoja de aislamiento, pero una casa no se construiría de la misma manera; pero, a pesar de que la casa sea resistente y cálida, de ninguna manera nos referimos a la bodega como de inferior calidad que la casa. La bodega se diseñó adecuadamente para un fin previsto y, si hubiera sido construida de la misma manera que la casa, incluso podría criticarse por tener "más ingeniería de la necesaria" –un juicio según el cual los diseñadores despilfarran recursos en las construcciones y que por lo tanto no aplican la “ingeniería necesaria”.

En software, un proyecto bien ejecutado se gestiona para que cumpla alguno de los siguientes objetivos del producto:

- Defectos mínimos
- Máxima satisfacción de usuarios
- Tiempo de respuesta mínimo
- Buena mantenibilidad
- Buena extensibilidad
- Robustez
- Alta correctitud

Cada equipo del proyecto software debe definir explícitamente la importancia relativa de cada característica; luego, el equipo completo, debe conducir el proyecto de forma que logre sus objetivos.

Los proyectos software son diferentes de los proyectos de ingeniería que utilizan materiales físicos. En otro tipo de ingeniería, el costo de los materiales puede llegar al 50% o más del costo total del proyecto. Algunas empresas de ingeniería reportan que consideran automáticamente como de alto riego a los proyectos cuya mano de obra constituye más del 50% de su costo [5]. En un proyecto típico de software, los costos de mano de obra pueden llegar casi al 100% del costo total. La mayoría de proyectos de ingeniería se centran en optimizar los objetivos del *producto* y los costos de diseño son relativamente insignificantes. Debido a que los costos de mano de obra constituyen gran parte del total de los costos del ciclo de vida del software, los proyectos software necesitan enfocarse más en optimizar los objetivos del *proyecto* que lo que

hacen otros tipos de ingenierías. Por lo tanto, además de trabajar en pro de los objetivos del producto, un equipo de software también debe trabajar para lograr alguno de los siguientes objetivos del proyecto:

- Calendario corto
- Fecha de entrega predecible
- Bajo costo
- Equipo pequeño
- Flexibilidad para ejecutar los proyectos aunque los requisitos cambien

Cada proyecto software debe encontrar un equilibrio entre los diferentes objetivos del proyecto y los del producto. No queremos pagar US$5000 por un procesador de texto, ni queremos que se bloquee cada 15 minutos.

¿En cuáles de estas características específicas del producto y del proyecto hace hincapié un equipo de trabajo para determinar si un proyecto es o no verdadera "Ingeniería de Software"? Algunos proyectos necesitan producir software con defectos mínimos y correctitud casi perfecta –software para equipos médicos, aviación, viajes espaciales y así sucesivamente. Muchas personas estarían de acuerdo en que estos proyectos son un dominio apropiado para la Ingeniería de Software a gran escala. Otros proyectos necesitan entregar su software con una fiabilidad adecuada, pero con bajos costos y calendarios cortos. ¿Pertenecen estos proyectos al dominio de la Ingeniería de Software? Una definición informal de ingeniería es "hacer con un centavo lo que cualquier persona puede hacer con un dólar". Muchos de los actuales *programadores* de software están haciendo con un dólar lo que cualquier buen ingeniero de software puede hacer con un centavo. El desarrollo económico también es dominio de la Ingeniería de Software.

La actual dependencia generalizada del desarrollo *code-and-fix* y el exceso en costos y calendario que implica, no es el resultado de una estimación de la Ingeniería de Software, sino de poca formación y entrenamiento en las prácticas de la Ingeniería de Software.

5. CONCLUSIONES

El desarrollo de software, como se practica comúnmente hoy en día, no se parece mucho a ingeniería, pero podría lograrse. Una vez que dejemos de hacer la pregunta equivocada de "¿Qué *es* actualmente el desarrollo de software?" y empecemos a hacer la pregunta correcta de "¿El desarrollo de software *debería* ser ingeniería?" podremos empezar a responder las preguntas realmente interesantes: ¿Cuál es el cuerpo base de conocimiento de la Ingeniería de Software? ¿Qué necesitan hacer los desarrolladores profesionales de software antes de que puedan utilizar ese conocimiento? ¿Qué tanta es la recuperación de la inversión desde la práctica de desarrollo de software como una disciplina de ingeniería? ¿Cuáles son las normas de conducta profesional apropiadas para los desarrolladores de software? ¿Para las organizaciones de software? ¿Se debe reglamentar a los desarrolladores de software? Si es así, ¿en qué medida? Y, tal vez la pregunta más interesante de todas: ¿Cómo será la industria del software después que todas estas cuestiones hayan sido resueltas?

REFERENCIAS

[1] D. Knuth. "The Art of Computer Programming: Volumes 1-3". USA: Addison-Wesley Professional. 1998.

[2] D. L. Parnas. "Software Engineering Programs Are Not Computer Science Programs". *IEEE Software,* Vol. 16, No 6, pp. 1-16. 1999.

[3] E. Serna M. (2010). "Métodos Formales e Ingeniería de Software". *Revista Virtual Universidad Católica del Norte*, No. 30, pp. 1-20.

[4] D. L. Parnas. "Software Engineering: An Unconsummated Marriage". *Communications of the ACM*. Vol. 40, No. 9, pp. 128. 1997.

[5] R. Baines. "Across Disciplines: Risk, Design, Method, Process, and Tools". *IEEE Software*, Vol. 15, No. 4, pp. 61-64. 1998.