MANIFIESTO

POR LA PROFESIONALIZACIÓN DEL DESARROLLO DE SOFTWARE

Editor
Edgar Serna M.

RedLatinaIS

Instituto Antioqueño de Investigación

Medellín, Antioquia, Colombia
2013

Serna M., Edgar (Ed.)
Manifiesto por la profesionalización del desarrollo de software / Edgar Serna M. --1a ed.
Medellín: Editorial IAI, 2013.
51 p. (Investigación Científica).

Copyright © 2013 Instituto Antioqueño de Investigación (IAI)™

Except where otherwise noted, content in this publication is licensed under the
Creative Commons Attribution, NonCommercial, ShareAlike 3.0 Unported License.
Available at http://creativecommons.org/licenses/by-sa/3.0.

Primera edición, diciembre 2013

Global Publisher: *Instituto Antioqueño de Investigación* (IAI)
Cover Designer: *Instituto Antioqueño de Investigación* (IAI), *Medellín, Antioquia, Colombia.*

Red Latinoamericana en Ingeniería de Software (RedLatinaIS) is trademarks of the *Instituto Antioqueño de Investigación* (IAI). All other trademarks are property of their respective owners.

The information, findings, views, and opinions contained in this publication are responsibility of the authors and do not necessarily reflect the views of *Instituto Antioqueño de Investigación* (IAI), and does not guarantee the accuracy of any information provided herein.

Diseño, edición y publicación: *Instituto Antioqueño de Investigación* (IAI)

Instituto Antioqueño de Investigación
http://fundacioniai.org/index.html
contacto(AT)fundacioniai.org

RedLatinaIS
http://fundacioniai.org/red.html
redlatinais(AT)lacrest.org

© 2013 Editorial IAI
Medellín, Antioquia, Colombia

ISBN: 978-958-46-3301-9

RED LATINOAMERICANA EN INGENIERÍA DE SOFTWARE

RedLatinaIS

MANIFIESTO

POR LA PROFESIONALIZACIÓN DEL DESARROLLO DE SOFTWARE

EDITOR

Edgar Serna M.

© Instituto Antioqueño de Investigación, IAI
Medellín, Antioquia, Colombia
2013

INTEGRANTES

RED LATINOAMERICANA EN INGENIERÍA DE SOFTWARE (RedLatinaIS)

Personas	Instituciones	Países
ALEJANDRO ARMANDO HOSSIAN	*Universidad Tecnológica Nacional*	Argentina
ÁLVARO RUIZ DE MENDAROZQUETA	*Fundación Sadosky + Taller Technologies + FuDePAN*	Argentina
FABIO GRIGORJEV	*Taller Technologies*	Argentina
FEDERICO SEBASTIÁN BOBBIO	*Instituto Nacional de Tecnología Industrial* (INTI), Córdoba	Argentina
MARÍA FLORENCIA POLLO-CATTANEO	*Universidad Tecnológica Nacional*	Argentina
PAOLA V. BRITOS	*Universidad Nacional de Río Negro*	Argentina
RAMÓN GARCÍA-MARTÍNEZ	*Universidad Nacional de Lanús*	Argentina
LUIZ FERNANDO MARTINS CALLADO	*IBM Latín América*	Brasil
ALBEIRO CUESTA MESA	*Ministerio de Tecnologías de la Información y las Comunicaciones (MINTIC)*	Colombia
ALEXEI SERNA A.	*Instituto Antioqueño de Investigación* (IAI), Medellín	Colombia
BRIGITTE NATHALIE ORTIZ LONDOÑO	*Institución Universitaria de Envigado*	Colombia
CARLOS ARTURO CASTRO CASTRO	*Universidad de San Buenaventura*, Medellín	Colombia
EDGAR SERNA M.	*Instituto Antioqueño de Investigación* (IAI), Medellín *Instituto Tecnológico Metropolitano* (ITM), Medellín	Colombia
EUGENIA VICTORIA MEJÍA DE R.	*Institución Universitaria de Envigado*	Colombia
GABRIEL PORRAS	*AxiomaX*	Colombia
GUILLERMO LINCE BONILLA MARIOTA	*Institución Universitaria de Envigado*	Colombia
HENRY MAURICIO VÁSQUEZ CARVAJAL	*Instituto Tecnológico Metropolitano* (ITM), Medellín	Colombia
HERNÁN DE J. SALAZAR ESCOBAR	*Instituto Tecnológico Metropolitano* (ITM), Medellín	Colombia
JESÚS ENRIQUE LONDOÑO SALAZAR	*Universidad de San Buenaventura*, Medellín *Universidad Católica del Norte*	Colombia
JORGE ENRIQUE MOLINA ZAMBRANO	*Universidad Piloto de Colombia*	Colombia
JORGE IVÁN BEDOYA RESTREPO	*Instituto Tecnológico Metropolitano* (ITM), Medellín	Colombia
JORGE MAURICIO SEPÚLVEDA CASTAÑO	*Corporación Universitaria Remington*	Colombia
JUAN CARLOS SOSA GIRALDO	*Outsourcing Desarrollos en Informática*	Colombia
JULY ANDREA CORREA SALAZAR	*Institución Universitaria de Envigado*	Colombia
LEONEL NOSSA ORTIZ	*Fundación Autónoma de Colombia*	Colombia
LEONEL VELÁSQUEZ TORRES	*Instituto Tecnológico Metropolitano* (ITM), Medellín	Colombia
LILIANA PATRICIA GUARÍN CARDONA	*Universidad de San Buenaventura*, Medellín	Colombia
LINA MARÍA TABORDA GIRALDO	*Corporación Intersoftware*, Medellín	Colombia
LUIS FERNANDO CAICEDO	*Assist Consultores*	Colombia
LUIS FERNANDO VARGAS CANO	*Universidad Cooperativa de Colombia*, Medellín	Colombia
LUIS MERCHÁN PAREDES	*Universidad de San Buenaventura*, Cali	Colombia
MARÍA EUGENIA MORALES MORA	*Institución Universitaria de Envigado*	Colombia
MARTHA LUCIA PALACIOS HUERTAS	*Fundación Autónoma de Colombia*	Colombia
PAULA ANDREA TAMAYO OSORIO	*Institución Universitaria de Envigado*	Colombia
PORFIRIO ÁLVAREZ ARANGO	*Corporación Universitaria Remington*	Colombia
Q-VISION S.A.	*Q-VISION S.A.*	Colombia
RAQUEL MARTÍNEZ MORALES	*Institución Universitaria de Envigado*	Colombia
RICARDO DE J. BOTERO TABARES	*Tecnológico de Antioquia (TdeA)*, Medellín	Colombia
SEGUNDO MANUEL BLANCO PALENCIA	*Instituto Tecnológico Metropolitano* (ITM), Medellín	Colombia
SOREY BIBIANA GARCÍA ZAPATA	*Fundación Avanet*	Colombia
WALTER HUGO ARBOLEDA MAZO	*Universidad EAFIT, Medellín*	Colombia
WILIAM A. ARÉVALO CAMACHO	*Corporación Cidenet*, Medellín	Colombia
GABRIELA SALAZAR BERMÚDEZ	*Universidad de Costa Rica*	Costa Rica
MARCELO JENKINS	*Universidad de Costa Rica*	Costa Rica
DAIRA PÉREZ SERRANO	*Universidad de las Ciencias Informáticas* (UCI)	Cuba
HENEY DÍAZ PÉREZ	*Centro Nacional de Calidad de Software* (CALISOFT)	Cuba
HEYDI MENÉNDEZ AVALOS	*Centro Nacional de Calidad de Software* (CALISOFT)	Cuba

Personas	Instituciones	Países
MAIRELIS QUINTERO RÍOS	*Centro Nacional de Calidad de Software* (CALISOFT)	Cuba
OSCAR PASTOR LÓPEZ	*Universitat Politècnica de València*	España
DIEGO VALLESPIR	*Universidad de la República, Montevideo*	Uruguay
LUCÍA CAMILLONI	*Universidad de la República, Montevideo*	Uruguay
JORGE L. DÍAZ-HERRERA	*Keuka College, New York*	USA

PRÓLOGO

El propósito del presente *Manifiesto por la Profesionalización del Desarrollo de Software*, organizado por la Red Latinoamericana en Ingeniería de Software (RedLatinaIS), es proporcionar una primera y única referencia completa, y asequible a estudiantes, profesores, profesionales y a la comunidad en general, de este importante tópico en América Latina. Con tal objetivo se ofrece información al día de una amplia gama de temas sobre las disciplinas de computación, y particularmente en Ingeniería de Software.

En los últimos 60 años, las disciplinas de la computación se han desarrollado rápidamente, al punto que ha sido necesario crear especialidades para ampliar y profundizar la cobertura de sus temas afines. Por ejemplo, en EE.UU., el crecimiento de esta disciplina académica, relativamente nueva, se describe en función de dos aspectos fundamentales: 1) las principales sociedades profesionales: The Association for Computing Machinery (ACM), IEEE Computer Society (IEEE-CS) y The Association of Information Systems (AIS), cada una preocupada de diferentes elementos disciplinares, como sus ciencias, ingenierías y tecnologías, y 2) la delimitación como familia de cinco sub-disciplinas diferentes: Ciencias Computacionales, Ingeniería de la Computación (*hardware*), Sistemas de Información, Tecnologías de la Información, e Ingeniería de Software. Los nombres de estas sub-disciplinas y las sociedades profesionales correspondientes, abarcan plenamente las diferentes tendencias profesionales y la variedad de programas de grado y posgrado que se han desarrollado en todo el mundo (ACM/IEEE, 2005), con excepción de América Latina donde se utiliza *informática* o *sistemas* como reemplazo generalizado.

Es interesante destacar que la palabra informática está cogiendo auge como la aplicación de computación en un contexto específico. Es decir, la informática es el uso contextual de la información, en relación a otro dominio, como la bioinformática cuando se trabaja con sistemas biológicos, la informática financiera, la astroinformática, y así sucesivamente. De esta manera, cada dominio tiene sus propios algoritmos y modelos para procesar esta información y generar conocimiento específico.

Este Manifiesto trata precisamente esta diferenciación de las sub-disciplinas de computación, así como el contraste que existe en Latinoamérica sobre las carreras correspondientes. Sin embargo, este documento se enfoca fundamentalmente en la Ingeniería de Software, y sobre todo en aportar para la búsqueda de la profesionalización del desarrollo de software, como dos tópicos íntimamente relacionados. Primero, y principalmente, se discute si es posible ciertamente catalogar a la Ingeniería de Software como una disciplina de ingeniería propiamente dicha, y en segundo lugar, se discute la necesidad de profesionalizar el desarrollo de software, independientemente de si se considera una ingeniería o no.

Pero una cuestión de suma importancia en este trabajo es determinar ¿cuál es la diferencia y por qué importa profesionalizar el desarrollo de software? Primero que todo es importante llamar a las cosas por su nombre, para saber y conocer lo que son y lo que representan; y en segundo lugar, se debe aclarar el panorama de las disciplinas afines a la computación, y sus enfoques y alcances. De esta manera, se puede factibilizar la creación de programas de estudios más adecuados y lograr profesiones mejores definidas.

Por un lado, el software es vital para el continuo desarrollo del mundo actual, y su importancia pública y para la Sociedad de la Información y del Conocimiento es cada vez más amplia y trascedente, no sólo para la seguridad y protección humana, sino también para su propio bienestar y beneficio. La complejidad en el desarrollo de software es creciente y la industria relacionada amplia y global, y crece a pasos agigantados. La importancia de los sistemas es cada vez más intensiva para la sociedad, y para áreas como las economías nacionales y mundiales, la salud, la aeronáutica entre otras, y por lo tanto, es imprescindible poder ofrecer software de alta calidad. Es por esto que en los últimos 50 años el acto de *crear* software se ha llegado a entender como extremadamente difícil, por lo tanto, se pensó que estructurarlo como una disciplina afín a las ingenierías más establecidas podría ser una salida. Esta búsqueda ha señalado sin duda dificultades insalvables para reconocerla como tal, y la historia de la Ingeniería de Software hasta la fecha es la historia del desarrollo de una disciplina práctica, en respuesta a las demandas del mercado y a la inherente y creciente complejidad del software. Un producto software defectuoso, mal diseñado y de baja calidad puede generar consecuencias catastróficas. Por ejemplo, los problemas que generó la implementación del sitio web del programa Affordable Health Care Act, que desestabilizaron el proyecto al punto que le puede negar al presidente Obama su más importante principio de gobierno.

Por otro lado, y a pesar de las preocupaciones manifestadas tempranamente por la OTAN, desde 1968, acerca de las consecuencias críticas de no considerar el desarrollo de software como una profesión más rigurosa, actualmente la Ingeniería de Software no se considera *una disciplina profesional completamente formada y madura*. Una cuestión particularmente problemática en este sentido es la *falta de un reconocimiento generalizado a los desarrolladores de software como profesionales en el sentido pleno de la palabra*. Entonces, la posibilidad futura de la profesionalización del desarrollo de software, aunque parezca un objetivo difícil de alcanzar, es imprescindible. Algunos nos preguntamos si el tratar de *someterla* al paradigma tradicional ingenieril haya dificultado su aceptación como una práctica profesional independiente. La verdad es que siempre fue muy tentador decir, durante el crecimiento de la computación, que la práctica de desarrollo de software se estaba convirtiendo en una disciplina de ingeniería. Sin embargo, y luego del recorrido y el trabajo realizado hasta ahora, la Ingeniería de Software *no es considerada como una disciplina de ingeniería,* por lo menos no en el sentido tradicional de la palabra.

La primera incógnita a descifrar aquí es si la Ingeniería de Software *es o no* ingeniería, para lo que debemos definir qué es ingeniería y cómo se aplica esa definición al desarrollo de software. Es decir, si el conjunto de actividades asociadas con la *creación* de software se considera ingeniería, entonces se deben establecer unas relaciones adecuadas, en términos de similitudes y diferencias, con las disciplinas convencionales de la ingeniería. Aunque existen algunas similitudes, también existen muchas diferencias importantes entre la Ingeniería de Software y la ingeniería tradicional, ya sea a nivel de objeto, de soporte científico o de procesos, entre otras (Díaz-Herrera, 2009).

Debido a su intangibilidad y plasticidad el software es diferente, como objeto, con respecto a los objetos tratados por otras ingenierías, y esto es lo más obvio. Por lo tanto, la Ingeniería de Software se basa en fundamentos diferentes, porque el producto que se diseña y se crea es un artefacto abstracto o lógico, en lugar de una entidad física y concreta. Estos fundamentos se encuentran principalmente en las Ciencias Computacionales —las ciencias de lo artificial—, y no en las Ciencias Naturales. De esta manera, la Ingeniería de Software *no tiene una ciencia bien establecida,* en la que puedan tener base la mayor parte de sus principios, como lo es el caso de la

ingeniería química, que se basa en gran medida en la química, la ingeniería civil, que se basa en las propiedades mecánicas conocidas de la física, o la ingeniería eléctrica, que se basa en las propiedades electromagnéticas de la física, y así sucesivamente.

A pesar de que algunas de las técnicas de ingeniería de software se basan en las matemáticas discretas —pero no en los sistemas de estado continuo como el cálculo, tradicionalmente materia fundamental de todas las ingenierías—, *no existe una matemática de Ingeniería de Software*. Es más, la larga búsqueda de un enfoque axiomático a la programación ha resultado vacía hasta hoy; e incluso, si se tiene éxito en la búsqueda de cómo probar programas correctos, muchos de los sistemas que se manejan actualmente son demasiado complejos para ser tratados por este análisis matemático. Entonces, es evidente que para *la Ingeniería de Software todavía no existe una "ciencia pura" con soporte matemático apropiado*.

Este es un problema importante, tanto en la teoría como en la práctica, que tiene que ver más específicamente con mediciones. En todas las ingenierías existe la noción de medidas y métricas, y esto es fundamental para los procesos y los productos de esta área del conocimiento, y probablemente sea el componente de la Ingeniería de Software que más difiere de la ingeniería tradicional. El desarrollo de software es inherentemente diferente, y sus indicadores métricos son mucho menos precisos, por lo que *el tipo de medidas precisas y suficientes, que se practican en otras disciplinas de ingeniería, no existen para la Ingeniería de Software*. Veamos. Los únicos tres aspectos que se pueden medir con respecto al software son: *tamaño*, *tiempo de desarrollo o esfuerzo*, y *defectos* (Humphrey, 1995). Estas mediciones, sin embargo, no pueden *predecir* exactamente el comportamiento de los productos software —un aspecto crítico para toda ingeniería[1]—. Es más, *no existen relaciones universales entre la calidad del software y estas mediciones y, en particular, el comportamiento del software no se puede medir de manera predecible*. No existen medidas que puedan predecir fallas y vulnerabilidades de seguridad, ni en qué medida se puede predecir la calidad de un producto software. Es decir, no se puede garantizar un software libre de defectos.

En cuanto a la práctica de la ingeniería, la diferencia primaria, nuevamente, es que el software es intangible e infinitamente maleable. Mientras que en la práctica todas las ingenierías resuelven problemas de gestión, éstos tienden a ser común a través de muchas áreas de la ingeniería, pero ninguno, en su formulación básica, se ocupa de las interacciones humanas con los artefactos producidos, como en los sistemas software. Otras diferencias prácticas incluyen la falta de normas, estándares y manuales correspondientes usados en la práctica, así como la normalización de la educación y la licenciatura de profesionales de software. Disciplinas más establecidas, como la física, la química y la arquitectura, tienen manuales organizados de referencia que son un compendio de tautologías y hechos indudables (como el calor específico de los materiales o el ancho estándar de un pasillo en un edificio de oficinas), o procedimientos (como las prácticas de seguridad industrial). Como se puede observar, debido a esto la Ingeniería de Software no se considera una disciplina completamente madura, y ciertamente *¡no se encuentra un conjunto de verdades y hechos universales sobre software! Y tampoco existen manuales ampliamente aceptados y reconocidos para desarrollar software*.

---

[1] Una excepción es la *ingeniería industrial,* que en gran parte es matemáticamente abstracta, y aunque su colección de técnicas y herramientas parece ofrecer algún prospecto al software, son medibles (Agresti, 1981). La ingeniería eléctrica o electrónica también parecen ser una excepción, porque tratan con algunos efectos físicos no visibles, y sin embargo se pueden ser medir, sus efectos son observables, y sus productos materializables en circuitos *físicos*.

Entonces, ¿qué es Ingeniería de Software? La Ingeniería de Software es una rama de la enseñanza y del aprendizaje que ha existido por casi 50 años, y de la que actualmente se tiene un conocimiento amplio. Se introdujo en la década del 70 como una especialización de las Ciencias Computacionales (Computer Sciences), y desde mediados de los 80 ha sido aceptada en la academia como un campo formal de estudio. En la década de los 90 proliferaron en USA, Canadá, Europa y Australia, programas de licenciatura con el título de Ingeniería de Software (Díaz-Herrera & Hilburn, 2004), y más recientemente han aparecido programas, aunque pocos, a nivel de doctorado. Sommerville (1999) introdujo la idea de que esta ingeniería quizás sea algo diferente, otro tipo de *ingeniería*: *In essence, the type of systems which we are interested in are socio-technical software-intensive system.* Además, afirman Sommerville y sus colegas que *no hay solución técnica a la complejidad del software* (Sommerville et al., 2012). De acuerdo con estas apreciaciones, el desarrollo de software, como producto, siempre será diferente de los de las ingenierías tradicionales, y especialmente en sus detalles técnicos, debido a su invisibilidad y maleabilidad.

Lo que queda es abordar seriamente el desarrollo de software como una *profesión* formalmente aceptada y reglamentada, y olvidarnos de si es o no *ingeniería*. Para lograrlo, se debe definir el cuerpo de conocimientos necesarios, la acreditación de los programas académicos, y la licenciatura y certificación de profesionales del software. Además, debemos tomar en cuenta las realidades políticas, y en este sentido parece claro que el futuro de la profesionalidad del desarrollo de software se verá impulsado por la industria y las entidades gubernamentales, aunque en la actualidad la absorción de credenciales ha sido asumida por los intereses particulares de los practicantes de software (IEEE, 2013).

¿Cuáles son entonces los elementos fundamentales del desarrollo de software? La naturaleza de evolución rápida de las tecnologías y de los principios del software significa que una lista de elementos fundamentales de la disciplina o de sus actividades principales en cualquier momento estaría incompleta y/o desactualizada. Sin embrago, los fundamentos incluyen principalmente métodos a seguir en las diferentes actividades de desarrollo, así como las correspondientes actividades de Verificación y Validación, y las herramientas de soporte. En este conjunto de categorías se observan dos características: 1) los métodos los llevan a cabo seres humanos, y 2) la categorización todavía es muy general. Pero en las últimas décadas se ha tratado de identificar el cuerpo de conocimiento necesario para resolver los problemas técnicos del desarrollo de software, lo mismo que sus aspectos económicos, legales, y sociales (Bourque & Dupuis, 2001). Concomitante con este adelanto en el reconocimiento del desarrollo de software como una actividad profesional se introduce códigos de ética, y a finales de 1999, la ACM e IEEE-CS, aprobaron el *código de ética y práctica profesional para el desarrollo de Software*.

Otra cuestión que se debe analizar es ¿qué se utiliza realmente en la práctica? Aunque no se conocen datos de encuestas, integrales y objetivas, sobre el tema, se conoce que la mayoría de productos software producidos actualmente todavía se desarrollan siguiendo métodos *ad-hoc*. De vez en cuando se encuentra información, aunque generalmente centrada en un lenguaje de programación o en una herramienta o metodología específica, pero muchas veces no son más que piezas de promoción. Es decir, necesitamos encontrar formas eficaces de evaluar conceptos innovadores, tanto en métodos como herramientas, que realmente aporten al avance del desarrollo de software (ver "Empirical and evidence-based software engineering" y resultados del Software) (CMM-CMM-I, 2013). Por otro lado, ¿podría existir una base *científica y matemática* para comprender la complejidad del software, de tal manera que pudiéramos predecir su calidad y

comportamiento? La respuesta a esta pregunta fundamental por ahora es desconocida. Por lo tanto, el desarrollo de software tiene que hacer frente a los retos de diseñar sistemas software-intensivos complejos usando más heurística que métodos analíticos. En este sentido, el desarrollo de software es, y posiblemente siempre será, experimental. En conclusión, el desarrollo de software todavía ni siquiera es una disciplina completa, ya sea en el sentido intelectual o práctico de la palabra e independientemente de si es ingeniería.

Sea cual sea el punto de vista y el interés en esta disciplina, la historia de los últimos 50 años puede ser esclarecedora y útil. La principal traba para considerar al desarrollo de software como una profesión ha sido el tratar de *encajarlo,* siendo una sub-disciplina fundamentalmente diferente, en el paradigma tradicional de ingeniería. El hecho de que la computación se esté convirtiendo en una rama distinta del conocimiento, es decir, una disciplina en su propio derecho, diferente de las ciencias y las ingenierías, se ajusta perfectamente a la noción de que las Ciencias Computacionales son un tipo diferente de ciencia, y por ende, la *ingeniería de software es un tipo diferente de ingeniería*. Tal vez lo que debemos hacer es explorar la forma de moldearla al interior de la computación como una disciplina madre diferente, que no es ciencia, ni ingeniería, ni tecnología, sino una disciplina propia y diferente de todas las que existen (Tucker et al., 2014).

La visión es entonces promover la computación como una rama distinta del conocimiento, una disciplina en su propio derecho, con principios y cánones particulares, y con su propia facultad académica. Aquí es donde la Ingeniería de Software pertenece. En la práctica, la profesión del desarrollo de software está tomando forma, pero es probable que estemos aún lejos de una definición disciplinar consistente. Este es un punto muy importante en las conclusiones de este manifiesto. El reto es convertirla en una profesión regularizada y bien formada, y no en una ingeniería a medias, y este *Manifiesto por la Profesionalización del Desarrollo de Software* es un paso agigantado en esa dirección.

Prof. Jorge L. DÍAZ-HERRERA, Ph.D.
President Keuka College
Keuka Park, NY USA

AGRADECIMIENTOS

Desde el Instituto Antioqueño de Investigación (IAI) queremos agradecerles a todas las personas e instituciones que hicieron aportes y nos brindaron sus opiniones para concretar este trabajo. Además de a quienes aparecen en listado de integrantes de la RedLatinaIS, sin cuyo valioso apoyo y decidida colaboración no se hubiese materializado esta iniciativa, queremos resaltar los aportes que nos brindaron: la profesora Raquel Anaya, quien hizo comentarios y propuso contenidos en la versión inicial de 2012; el impulso y difusión que obtuvimos desde el congreso LACREST; el apoyo logístico que nos aportó la Corporación Ruta N para la primera reunión en 2012; los espacios y recursos tecnológicos que para varios encuentros posteriores nos ofreció la Universidad de San Buenaventura Medellín; el apoyo de Albeiro Cuesta M. del programa de Fortalecimiento de la Industria de TI (FITI), Colombia, y los diferentes organismos estatales y privados que han hecho difusión de nuestro trabajo e iniciativas.

Muchas gracias a todos, y esperamos seguir contando con sus aportes y opiniones, para concretar y materializar los diferentes proyectos que continuaremos proponiendo y apoyando. El trabajo en equipo, la colaboración sincera, los aportes desinteresados y la crítica constructiva, serán los pilares que sustenten las diversas iniciativas en las que participemos. A todos los que, por una u otra razón, no integraron el equipo que llevó a cabo este proyecto les extendemos la invitación a que se unan, para incrementar cada día, en cantidad y calidad, el talento humano de la Red Latinoamericana en Ingeniería de Software (RedLatinaIS), y poder alcanzar los objetivos plasmados en nuestros Estatutos.

Si trabajamos juntos,
lo lograremos

CONTENIDO

Pág.
3 INTEGRANTES RedLatinaIS
5 PRÓLOGO
10 AGRADECIMIENTOS
12 PRESENTACIÓN

14 PRIMERA PARTE - CONCEPTUALIZACIÓN
16 1. DEFINICIONES BÁSICAS
16 1.1 Ingeniería
16 1.2 Ingeniero
16 1.3 Sistema
17 1.4 Software
17 1.5 Informática
17 1.6 Computación
18 1.7 Ciencias Computacionales
18 1.8 Sistemas de Información
18 1.9 Tecnologías de la Información
19 1.10 Ingeniería de Sistemas
19 1.11 Ingeniería Informática
20 1.12 Ingeniería de Software
20 1.13 Desarrollador
21 1.14 Programador
21 2. LA INGENIERÍA DE SOFTWARE COMO PROFESIÓN
22 2.1 Desarrollar software es un Arte y una Ciencia
24 2.2 Profesiones y Profesionales
25 2.3 Profesionalizar
28 2.4 Responsabilidad profesional de los ingenieros de software
29 2.5 Recomendaciones para alcanzar la profesionalización

31 SEGUNDA PARTE – LA PROFESIONALIZACIÓN DEL DESARROLLO DE SOFTWARE
33 INTRODUCCIÓN
35 1. SITUACIÓN ACTUAL DEL DESARROLLO DE SOFTWARE
35 1.1 Evolución de los enfoques de ingeniería
37 1.2 La Naturaleza del proceso software
38 1.3 La práctica de la Ingeniería de Software
39 1.4 Condiciones de contexto
39 2. LA FORMACIÓN DEL INGENIERO DE SOFTWARE
42 2.1 Competencias del Ingeniero de Software
43 2.2 Roles del Ingeniero de Software
45 2.3 Prácticas del Ingeniero de Software

46 MANIFIESTO POR LA PROFESIONALIZACIÓN DEL DESARROLLO DE SOFTWARE
47 REFERENCIAS

PRESENTACIÓN

El explosivo crecimiento de las Tecnologías de la Información ha generado desafíos particulares para el conocimiento, en relación con el uso de los términos *ingeniero*[2] *de software* e *Ingeniería de Software*. Actualmente, muchos de quienes desarrollan software se refieren a sí mismos como *ingenieros de software*, y a su trabajo como *Ingeniería de Software*, a pesar de no contar con una formación en ingeniería y de no estar reglamentados de alguna forma. Dadas la penetración e importancia que el software ha alcanzado en este siglo esta situación se ha convertido en un problema, porque induce a errores en los productos y una interpretación y valoración inadecuada de la profesión. Además, la dependencia social del software ha llegado a niveles amplios, y las personas delegan proyectos software a estos *seudo-profesionales*. Debido a que construir software se ha definido popularmente como una cuestión informal, la mayoría de ellos se sienten calificados para desarrollar programas, sólo por el hecho de conocer la semántica y la gramática de algún lenguaje. Esta cuestión ha hecho carrera, al punto que la industria se ha dejado influenciar y los contrata para *desarrollar* los sistemas, que luego ofrece al público. La práctica de esta ingeniería debe estar autorizada y regulada, casi al mismo nivel que la medicina o la ingeniería civil, porque si el desarrollo de Sistemas de Información no cumple con los criterios necesarios de integridad, eficiencia, fiabilidad y correctitud se podría poner en riesgo la vida de las personas y los intereses económicos de las empresas.

Por otro lado, la Ingeniería de Software proporciona el marco teórico, los métodos y las herramientas necesarios para desarrollar software de calidad, y ha impulsado la revolución de la Sociedad de la Información y del Conocimiento, porque sin sus aportes los computadores serían sólo herramientas sin utilidad específica. Además, a pesar de los avances en *hardware*, el impacto y la potencialización del desarrollo tecnológico sólo fue posible gracias a los productos *software*. Por otro lado, la actual sociedad se empieza a reconocer como *software-dependiente*, porque en este siglo el software hace parte de todos los dispositivos que se requieren para manipular información, y que las personas utilizan en sus actividades cotidianas.

El término *Ingeniería de Software* se conoció por primera vez en la NATO Software Engineering Conference de 1968, e inicialmente su objetivo era concientizar acerca de la *crisis del software* que se vivía en ese momento. Desde entonces, se ha desarrollado como una profesión y como un campo de estudio, cuyo objetivo es desarrollar productos de mayor calidad, más asequibles, fáciles de mantener y rápidos de construir. Dado que es un área del conocimiento relativamente joven, en comparación con sus campos ingenieriles afines, todavía se presenta debate en torno a lo que realmente es y si se ajusta a la definición clásica de ingeniería. Por ejemplo, las prácticas necesarias para crear un buen producto software se establecieron hace décadas, pero a pesar de algunos triunfos sobresalientes esta industria todavía no está a la altura de sus capacidades. Existe un amplio abismo entre las necesidades reales del medio y los productos desarrollados, e inclusive muchas de las prácticas de uso generalizado son anticuadas. Algunos profesionales utilizan métodos de desarrollo de software tratando de imitar los procesos que aplica la ingeniería civil, y otros han perdido la paciencia con antiguos modelos burocráticos, de cascada y con documentación abundante, que aunque útiles en su momento actualmente se olvidaron de, o apenas tiene en cuenta, conceptos fundamentales como calidad, accesibilidad, facilidad de mantenimiento, agilidad,

[2] En algunos países de América Latina, los términos Ingeniero y Licenciado tiene afinidad, por lo tanto en este documento se hará referencia a *Ingeniero* con el mismo significado de *Licenciado*.

criterio y orientación a resultados, que son características deseables en todo tipo de producto software para este siglo.

Además, un problema central del desarrollo económico actual y de la competitividad industrial, social y científica, es la *complejidad* de los grandes e intensivos sistemas software, y de los procesos para su desarrollo y aplicación. Esta complejidad se define por la cantidad y heterogeneidad de la interacción del hardware con los componentes software, de sus inter-relaciones, de la incorporación en los entornos técnicos y organizacionales, y de las interfaces para los seres humanos. El dominio de estos sistemas requiere de acciones y pensamientos jerárquicos y sistemáticos, y el éxito de los productos, los servicios y las organizaciones, está cada vez más determinado por la disponibilidad de productos software adecuados. Por lo tanto, se necesitan profesionales altamente cualificados, capaces de comprender y dominar los sistemas, de participar en todo el ciclo de vida de la Ingeniería de Software, y de adoptar diferentes roles durante el desarrollo Esta es la razón que guía el pensamiento de esta Manifiesto, cuyo objetivo es lograr la *Profesionalización del Desarrollo de Software*.

Este documento contiene una recopilación de ideas y aportes de diferentes actores de la comunidad latinoamericana del software, y tiene como objetivo constituir la base conceptual que sustente los principios de la Red Latinoamericana en Ingeniería de Software (RedLatinaIS). Este manifiesto no es el primer trabajo de este tipo, y esperamos que no sea el último, pero estamos convencidos que será una contribución importante para el logro de las iniciativas de la Red. Están todos cordialmente invitados a colaborar con sus aportes e ideas para lograr el establecimiento de un punto común de entendimiento alrededor de la necesidad de profesionalizar el desarrollo de software.

Debido a la amplia variedad de conceptos que se involucran en el contexto en el que se suscribe este documento, y para dar claridad a su intencionalidad en el contenido, en la primera parte se presentan las definiciones básicas de algunos de ellos y se hace una justificación de la necesidad de reconocer a la Ingeniería de Software como profesión. En la segunda parte se argumenta por qué es conveniente profesionalizar el desarrollo de software, iniciando con un acercamiento a la situación actual del desarrollo de software, y posteriormente con una descripción a la formación del ingeniero de software y de sus competencias, roles y prácticas, para terminar con el *Manifiesto por la Profesionalización del Desarrollo de Software*.

PRIMERA PARTE

CONCEPTUALIZACIÓN

La Ingeniería de Software es un área del conocimiento relativamente joven, y en muchos círculos se le considera como un área inmadura del conocimiento. En 1995, el Software Engineering Institute, de Carnegie Mellon University, llevó a cabo un proyecto para caracterizar y modelar la evolución y maduración de las profesiones, con el fin de comprender cómo desarrollar la profesionalización de esta ingeniería. La mayoría de proyectos e iniciativas que surgieron posteriormente llegaron a la conclusión de que era necesario reorganizar la visión futura y ajustar los principios rectores del área para lograr estos objetivos.

Dados los factores críticos emergentes de este siglo, sociales y económicos, se ha generado una situación en la que es vital que se discuta el progreso de la Ingeniería de Software como una profesión reconocida. Este juicio requiere acciones inmediatas y a corto plazo, que podrían comenzar con el análisis a cuatro áreas específicas:

1. Clarificar los aspectos ingenieriles de la Ingeniería de Software, lo que ayudará a establecer sus límites con disciplinas afines.

2. *Armonizar prácticas y consensuar estándares*, para construir una base repetible para el área, y en las que se tenga en consideración los contextos de implementación y las organizaciones que las llevan a cabo, evitando limitaciones de otras metodologías y estándares, pero sin caer en generalidades ambiguas.

3. *Identificar su cuerpo de conocimientos*, con el objetivo de proporcionar una plataforma para los procesos formativos.

4. *Identificar la línea básica de expectativas de rendimiento* para un profesional en Ingeniería de Software, que a su vez se convierta en punto de partida para lograr su certificación y acreditación.

Actualmente, uno de los enfoques más comunes para desarrollar software en el mundo es la programación mediante codificación y corrección, en el que un equipo de programadores inicia con una idea general de lo que quieren construir —puede que tengan una especificación formal, pero probablemente no—, y utilizan la combinación informal de diseño, codificación, depuración y metodologías de prueba que más les convengan, para luego escribir un poco de código y ejecutarlo para ver si funciona, y, si no funciona, lo modifican hasta lograr el objetivo. Pero este enfoque no concuerda con el actual estado del arte en desarrollo de software, porque en el proceso se evidencia una Ingeniería de Requisitos inadecuada, lo que genera productos de inferior calidad que los otros métodos, aunque su principal ventaja es que requiere poca formación técnica o de gestión. Desde hace décadas, las organizaciones líderes en el área conocen y utilizan prácticas eficaces de desarrollo de software, pero la brecha entre la práctica promedio y las mejores prácticas en software es amplia.

En este sentido, si uno de los extremos del espectro de competencias del desarrollo de software lo ocupan la codificación y la corrección, el otro lo está por la Ingeniería de Software, es

decir, un enfoque sistemático, sistémico, disciplinado y cuantificable, aplicado a la concepción, desarrollo, operación y mantenimiento de software. La comunidad, que trabaja en este sentido, ha sido testigo de acontecimientos alentadores relacionados con el establecimiento, soporte y difusión de amplios estándares de gestión para la práctica del desarrollo de software, y este documento es un aporte más a esos sucesos, con el objetivo de documentar, proponer y apoyar el movimiento por la *Profesionalización del Desarrollo de Software*. A continuación se describen algunos conceptos relacionados con la filosofía del este documento, con el objetivo de clarificarlos para la comunidad y para los diferentes lectores.

1. DEFINICIONES BÁSICAS

Debido a la amplia variedad de conceptos y términos que se involucran en el contexto en el que se suscribe este *Manifiesto*, a continuación presentamos algunas definiciones, construidas desde nuestra experiencia y conocimiento, para dar claridad a su intencionalidad en el documento.

1.1 Ingeniería

Esta área del conocimiento consiste en la aplicación práctica de la ciencia y las matemáticas, para resolver problemas mediante el planteamiento de soluciones innovadoras. Para hacerlo es necesario imaginar, inventar, crear y construir un mundo que, debido a su dinamismo se encuentra en permanente cambio. Pero no basta con presentar soluciones, también es necesario mantenerlas y mejorarlas, para lo que se deben aplicar las matemáticas y la ciencia en las actividades de diseño y creación. Esta disciplina utiliza el ingenio, combinado con las leyes conocidas de la ciencia, para diseñar y crear dispositivos y sistemas que mejoren la calidad de vida de la sociedad. En este proceso de conversión, de recursos naturales en dispositivos para el uso humano, se necesitan aplicar diferentes principios científicos, para fabricarlos con pleno conocimiento del diseño y para pronosticar su comportamiento futuro bajo condiciones específicas de operación.

1.2 Ingeniero

Es el profesional encargado de aplicar la ingeniería. Aunque en muchos contextos el término se ha convertido en algo genérico, que se aplica informalmente a la realización de cualquier actividad relacionada con tecnología, en el sentido tradicional se utiliza para designar a la persona que utiliza su formación y aplica sus capacidades para construir los componentes, físicos o no-físicos, que hacen parte de los diferentes sistemas. Para lograrlo requiere un amplio conocimiento científico y matemático, el cual aplica y utiliza de forma práctica, ingeniosa, creativa e innovadora, por lo que es un especialista en tecnología que diseña y produce soluciones a las demandas de la sociedad. También se basa en las ciencias aplicadas, aunque su trabajo en investigación y desarrollo tiene un enfoque diferente al que aplican los científicos, porque crean un enlace entre los descubrimientos científicos y su posterior aplicación a las necesidades humanas. Los ingenieros poseen algunas características diferenciadoras: son constructores, aventureros y solucionadores de problemas; buscan formas más rápidas, mejores y menos costosas de ejecutar proyectos; utilizan las fuerzas y materiales de la naturaleza para el beneficio de la humanidad, y ayudan a mejorar la vida de las personas con dispositivos que ahorran trabajo, como motores y computadores, a la vez que tecnologías para cuidar la salud, como riñones artificiales y máquinas corazón-pulmón. Por otro lado, colaboran para satisfacer necesidades sociales más amplias, como vivienda, alimentación, vestuario, transporte, agua, aire y energía. En términos generales, el ingeniero es una combinación de científico, matemático, inventor y planificador de proyectos, que toma el conocimiento de varios campos y lo aplica para resolver problemas, y por esto es *creador*.

1.3 Sistema

En su definición más amplia, un sistema es un conjunto integrado de elementos que cumplen un objetivo determinado, y desde diferentes disciplinas de la ingeniería se tienen diferentes perspectivas de lo que es. Por ejemplo, para la Ingeniería de Software es un conjunto integrado de programas informáticos, y para la Eléctrica se refiere a los circuitos integrados, o a un conjunto de unidades eléctricas, es decir, su significado depende de la perspectiva y el contexto en el que se

aplique. En su definición más interna, es una conjunción de recursos y procesos que operan para lograr un propósito común, y por lo tanto satisfacer alguna necesidad. Es decir, es un conjunto de componentes interactivos o interdependientes que forman un todo integrado.

1.4 Software

En términos generales se refiere a la información que se almacena en algún tipo de medio digital. El software es la interfaz y la lógica abstracta que vincula al ser humano con la tecnología tangible y otro software, permitiendo una mutua interacción y retroalimentación. En una visión más técnica, el término se utiliza para describir una colección de programas informáticos, algoritmos, procedimientos y la documentación necesaria que realizan algunas tareas en un sistema, como operar los equipos y dispositivos tecnológicos relacionados, y la gestión de la información para la toma de decisiones.

1.5 Informática

Es un término que genera diversas interpretaciones y por tanto tiene diferentes significados. En Europa es común que se utilice como referencia a las Ciencias Computacionales, y en otros escenarios reúne en un sólo programa a la Ingeniería de Software, a la interacción humano-computador y al estudio de las tecnologías de la información en las organizaciones. Para este documento se asume como el conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de computadores, por lo que se puede entender como la disciplina encargada del estudio de métodos, procesos, técnicas y desarrollos, y de su utilización en computadores, con el objetivo de almacenar, procesar y transmitir información y datos en formato digital. Se refiere al procesamiento automático de información mediante dispositivos electrónicos y sistemas computacionales.

1.6 Computación

Se acepta de forma general como cualquier tipo de cálculo o uso de tecnología computacional en el procesamiento de información. Para lograrlo aplica un proceso que sigue un modelo bien definido, comprendido y expresado en un algoritmo, un protocolo, una topología de red, u otros componentes necesarios. En la computación se investiga lo que se puede o no hacer *computacionalmente*. Además, el término no está vinculado a los números, acrónimos, puntuación o sintaxis, pero una de las cosas que lo hace interesante es que no está del todo claro lo que es en realidad. Generalmente se acepta que cuando alguien organiza, por ejemplo, el saldo de su cuenta corriente está haciendo cómputo, sin embargo, también se cuestiona si el cerebro es fundamentalmente un computador, y si eso es cierto significaría que ¿todos los pensamientos son computación? O que ¿toda la computación es pensamiento? Se conoce que los entornos *virtuales* en los video-juegos son en realidad simulaciones computacionales, pero también que el universo real efectivamente es un computador en el que la física cuántica se relaciona con la información, y que la materia y la energía son abstracciones construidas a partir de ella. Pero si el universo *sólo* es computación, entonces surge la cuestión de ¿qué no es computación? Por lo tanto es una idea en constante cambio, y la cultura actual está en proceso de renegociación de lo que realmente significa el término. Hace cien años se pensaba que era una operación mental que implicaba números, y como tal sólo la podían realizar las personas; hoy en día, la mayoría considera que la realizan las máquinas, pero al mismo tiempo y de alguna manera, todavía se asocia con pensamiento.

1.7 Ciencias Computacionales

Las Ciencias Computacionales no se refieren a la construcción de computadores o a la escritura de programas, ni a las herramientas utilizadas en la computación, se refieren a cómo utilizar estas herramientas y a interpretar lo que se encuentra en el proceso. La solución a muchos de los problemas de estas ciencias ni siquiera requiere el uso de computadores, sólo lápiz y papel, e incluso la mayoría se ha abordado y solucionado desde décadas antes que se construyeran las máquinas modernas. Estas ciencias se refieren al enfoque científico y matemático de las tecnologías de la información y sus aplicaciones, y al software y al hardware subyacente. Un científico computacional se puede especializar en la teoría computacional, o en la experimentación y diseño de componentes y nuevas tecnologías. Abarcan un amplio rango de conocimientos, desde los fundamentos teóricos y algorítmicos hasta los desarrollos de vanguardia en robótica, visión por computador, sistemas inteligentes, bioinformática, y otras áreas relacionadas. Los científicos computacionales pueden diseñar e implementar software, crear nuevas formas de utilizar los computadores y desarrollar métodos eficaces para resolver problemas computacionales, y en ocasiones, los planes de estudios de área de formación son criticados por no preparar a los estudiantes para un trabajo específico, porque mientras otras disciplinas lo hacen en habilidades para el trabajo inmediato, éstos ofrecen una formación integral de adaptación a nuevas tecnologías.

En términos generales, estas ciencias se ocupan del estudio sistemático de la viabilidad, la estructura, la expresión y la mecanización de los procesos metódicos o algoritmos, que subyacen a la adquisición, representación, procesamiento, almacenamiento, comunicación y acceso a la información, cuando está codificada en bits y bytes en un computador, o transcrita en los genes y en las estructuras proteínicas en una célula humana. La cuestión fundamental subyacente es ¿qué procesos computacionales se pueden automatizar e implementar eficientemente? Para responder esta pregunta, aparentemente simple, los científicos computacionales trabajan en diversas áreas complementarias, como la naturaleza misma de la computación, para establecer cuáles problemas son o no computables; comparan diversos algoritmos, para determinar si ofrecen una solución correcta y eficiente a un problema; diseñan lenguajes de programación, para especificar y expresar algoritmos; diseñan, evalúan y construyen sistemas computacionales que ejecuten eficientemente esas especificaciones, y aplican los algoritmos al dominio de aplicaciones importantes.

1.8 Sistemas de Información

Esta disciplina hace hincapié en las tecnologías como instrumento para generar, procesar y distribuir información. Los especialistas integran las soluciones tecnológicas y los procesos de negocio para satisfacer las necesidades de información de las empresas, lo que les permite alcanzar sus objetivos de manera eficaz y eficiente. Los profesionales en esta área se ocupan principalmente de la información que los sistemas computacionales proporcionan, para ayudarles a las empresas en la definición y el logro de sus objetivos, y a seleccionar qué procesos pueden implementar o mejorar con el uso de las tecnologías. Deben comprender tanto las técnicas como los factores organizacionales y ser capaces de orientar a la organización a determinar cómo la información y la tecnología pueden habilitar procesos de negocio que le proporcionen ventajas competitivas.

1.9 Tecnologías de la Información

El término se utiliza comúnmente para referir toda una industria, pero actualmente se define como el uso de hardware y software para gestionar información. En algunos entornos se conoce como

servicios de gestión de la información, o simplemente como servicios de información, encargados de almacenar, proteger, procesar y transmitir la información, y posteriormente recuperarla cuando sea necesario. Otros puntos de vista las etiquetan dentro de dos significados: 1) en un sentido amplio, se utiliza para referir toda la informática, y 2) en el ámbito académico, se refiere a los programas de pregrado que preparan a los estudiantes para satisfacer las necesidades tecnológicas computacionales de las organizaciones.

En un sentido más técnico, se define como la rama de la ingeniería que se ocupa del uso de los computadores y las telecomunicaciones para almacenar, recuperar y transmitir información. Sus campos son adquirirla, procesarla, almacenarla y difundirla, ya sea verbal, en imágenes, en texto o numérica, mediante combinación micro-electrónica, la informática y las telecomunicaciones. Algunos de sus campos nuevos y emergentes son: la siguiente generación de las tecnologías web, la bioinformática, *cloud computing*, los sistemas globales de información y las bases de conocimiento a gran escala. Sus avances se deben principalmente al campo de las Ciencias Computacionales.

1.10 Ingeniería de Sistemas

Es un área del conocimiento que en diversos países latinoamericanos no tiene una definición de consenso, aunque tradicionalmente se define como un campo multidisciplinar para construir grandes cosas complejas, y para lograrlo aplican métodos ingenieriles. Es un campo inter y multi-disciplinar de la ingeniería, que se centra en cómo diseñar y gestionar los ciclos de vida de los proyectos ingenieriles, y se ocupa de los procesos de trabajo y de las herramientas para gestionar los riesgos en este tipo de proyectos. Muchas veces se confunde con disciplinas técnicas y centradas en lo humano, como la ingeniería de control, la ingeniería industrial, los estudios organizacionales y la gestión de proyectos. Cuestiones como la logística, la coordinación de los diferentes equipos y el control automático de las máquinas se hacen más difíciles cuando se trata de proyectos grandes y complejos, pero para lograrlo, esta ingeniería integra varias disciplinas y grupos de especialistas en un esfuerzo de equipo para conformar un proceso de desarrollo estructurado, que va desde el concepto de producción hasta el de operación. Por lo sus practicantes deben considerar al negocio y a las necesidades técnicas de todos los clientes, con el objetivo de ofrecer un producto de calidad que satisfaga las necesidades generales.

La Ingeniería de Sistemas es un área que se ha desarrollado desde hace relativamente poco tiempo, que aplica técnicas y métodos de la Teoría General de Sistemas y de la matemática aplicada para dar solución a problemas complejos en diferentes campos del conocimiento. Su principal trabajo es asumir a los sistemas de ingeniería como conjunto y no como un componente en particular. Su objetivo surge del hecho de que casi todo lo que nos rodea y que utilizamos a diario es un sistema relativamente complejo, conformado por una serie de diferentes componentes mecánicos, electrónicos, hardware, software, y posiblemente otros tantos. Abarca el diseño de modelos matemáticos y el análisis de sistemas, centrándose especialmente en cómo encajan entre sí los distintos componentes, y en garantizar un diseño en el que todos interactúen de manera eficiente y eficaz. Muchas universidades de la región orientan este objeto de formación al desarrollo de proyectos software, aunque sus planes curriculares no logren este cubrimiento.

1.11 Ingeniería Informática

Es un campo del conocimiento que tiene que ver con el diseño y la construcción de equipos y sistemas basados en computadores. Se trata del estudio de hardware, software, comunicaciones, y

la interacción entre ellos. Sus planes de estudio se centran en las teorías, los principios y las prácticas de la ingeniería eléctrica tradicional y de las matemáticas, que luego aplica para resolver los problemas de diseño de computadores y de dispositivos basados en ellos. En esta ingeniería se estudia el diseño de sistemas hardware digitales, incluyendo los de comunicaciones, los computadores y los dispositivos que contienen procesadores, y especialmente los dispositivos digitales y sus interfaces con los usuarios y otros dispositivos.

Los ingenieros informáticos analizan y diseñan el hardware, el software y los sistemas operativos para los sistemas informáticos. Para esto deben combinar los campos de las Ciencias Computacionales y la Ingeniería Eléctrica, por lo que a menudo se confunde con las primeras; pero los ingenieros informáticos también se forman en el diseño de software y en su integración con el hardware, y deben aplicar algoritmos y principios de diseño digital para diseñar, construir y probar los componentes utilizados para procesar, comunicar y almacenar la información, que normalmente se integra en grandes sistemas de ingeniería y en ambientes de redes distribuidas. Sus áreas de aplicación incluyen a las comunicaciones, la automatización, la robótica, la potencia, la energía, la salud, los negocios, la seguridad, el entretenimiento, y muchos otras.

1.12 Ingeniería de Software

Es la disciplina ingenieril que proporciona y aplica los métodos y herramientas necesarios para construir software de calidad, ajustado al presupuesto, en un plazo determinado y en un contexto de cambio constante de requisitos. Es la aplicación de un enfoque sistemático, disciplinado y cuantificable, al desarrollo, operación y mantenimiento de software; es decir, es la *aplicación de ingeniería al software*. En términos de costo y complejidad, en la mayoría de sistemas el software es el componente predominante, por tanto, las buenas prácticas de esta ingeniería y las herramientas pueden hacer una diferencia sustancial, incluso en la medida en que son la fuerza impulsora del éxito del proyecto. Se diferencia de las Ciencias Computacionales en que éstas tienen que ver con el desarrollo de aplicaciones científicas a gran escala, mientras que la Ingeniería de Software abarca no sólo los aspectos técnicos del desarrollo, sino también las cuestiones de gestión, como la dirección de equipos de diseño, de desarrollo, de pruebas, de presupuestos y de mantenimiento.

Actualmente, esta ingeniería ha evolucionado en respuesta a los factores del creciente impacto y costo de los grandes sistemas software en una amplia gama de situaciones, y a la importancia cada vez mayor del software en aplicaciones de seguridad crítica. Tiene un carácter diferente al de otras disciplinas ingenieriles debido a la naturaleza intangible de sus productos y a su discontinuidad operativa. Es un área del conocimiento que integra principios matemáticos y de las Ciencias Computacionales en las prácticas ingenieriles, para desarrollar el software que requieren los artefactos tangibles y físicos que utiliza la sociedad. En términos filosóficos, la Ingeniería de Software se encarga de la aplicación de diferentes estrategias y disciplinas para mejorar la capacidad de los seres humanos para enfrentar los retos de la actual Sociedad de la Información y el Conocimiento.

1.13 Desarrollador

Los desarrolladores escriben *código de calidad*. Hacerlo limpio, claro, bien factorizado y libre de errores son cuestiones importantes que tienen en cuenta, y además conocen el significado de *buen código* dentro de un dominio. Tienen adecuados conocimientos en matemáticas y en cómo buscar buenas soluciones para los problemas; poseen amplios conocimientos en algoritmia y buenas

habilidades en su área de experticia y las relacionadas; debido a que su trabajo se desenvuelve al interior de equipos multi-disciplinares, poseen buena capacidad de comunicación verbal y escrita, y de interacción con otras personas. Son profesionales que contribuyen de diversas maneras para que el producto software tenga éxito, y su trabajo consiste en aplicar principios científicos e ingenieriles para comprender, abstraer y moldear un problema, que se pueda resolver mediante un programa informático, para luego aplicar una metodología con el objetivo de llevar a la práctica la solución creada, tradicionalmente soportada en código de lenguaje de programación. En términos generales son responsables de aplicar procesos de calidad en todo el ciclo de vida del software.

1.14 Programador

Los programadores escriben *buen código*. Hacerlo bien y limpio es un factor importante, pero a menudo tienen prioridad otras cuestiones. Las habilidades matemáticas son opcionales, aunque poseerlas les ayuda a ser conscientes de los problemas comunes y de las soluciones relacionadas con el dominio, pero son primordiales las relacionadas con la comunicación y la interacción social. Son generalistas sin un tipo de especialización verdaderamente profundo; son capaces de encontrar caminos en torno a los problemas, y de conectar diversos componentes para cumplir con una serie de requisitos. Su trabajo consiste en aplicar el conocimiento que poseen, del lenguaje de programación, para escribir código *eficiente* y *eficaz*, a la vez que respetan los conceptos de calidad y de seguridad, y responden al diseño que se ha construido desde la Ingeniería de Software.

2. LA INGENIERÍA DE SOFTWARE COMO PROFESIÓN

Por décadas, los investigadores han tratado de encontrar la *bala de plata* necesaria para matar al monstruo de la generación de software ineficaz. Estos esfuerzos se reflejan en la continua búsqueda de metodologías, lenguajes, notaciones y otras fórmulas, casi *alquimistas*, cada una comprometida en resolver un problema, y aunque muchas de ellas han hecho aportes sustanciales no han logrado descubrir la bala faltante. Desde hace algún tiempo se inició un movimiento a nivel mundial que busca no encontrar una solución alquimista sino profesionalizar la labor de la Ingeniería de Software: el *Desarrollo de Software*.

La forma tradicional y ampliamente extendida de desarrollar software en diferentes industrias no se puede considerar una actividad profesional, porque en la mayoría de casos sus practicantes no aplican un modelo estructurado en el que deban emplear los niveles de destreza y las habilidades necesarios para lograr un producto con los niveles de calidad que la sociedad requiere. Este es el caso del software comercial, como los sistemas operativos y los ofimáticos, que al poco tiempo de salir al mercado deben ofrecer actualizaciones para corregir errores que se pudieron evitar desde el desarrollo. A diferencia de ingenierías como la Civil, en la que están claramente establecidos los requisitos formales para que sus practicantes se puedan considerar profesionales, la Ingeniería de Software parece no tener establecidas las reglas para que su ejercicio se considere profesional. Aunque algunas personas tienen instinto natural para *programar*, sin haber recibido capacitación formal para hacerlo, esto no quiere decir que tengan la formación necesaria para entregar productos fiables a la sociedad, sobre todo cuando se trata de sistemas críticos, como en las áreas aeroespacial, de la salud y la aviación. Tradicionalmente se acepta que un programador es bueno porque entrega el código esperado, pero hacen falta estándares de clasificación, aceptados de forma generalizada en la industria, la academia y el Estado, acerca de qué debería ser un *desarrollador profesional*, y de las características que se debe exigir para considerarlo como tal.

Para ilustrar esta cuestión basta con leer la declaración de la misión de Visual Basic para democratizar la *programación*: cualquiera que pueda arrastrar y soltar controles y comprender un mínimo de material técnico podría *juntar* una solución a un problema con un esfuerzo relativamente pequeño. Además, existen herramientas y servicios públicos que les permiten a los usuarios crear sofisticadas hojas de cálculo sin poseer conocimientos de programación práctica. Diversas personas e industrias han decidido apoyar iniciativas como las de la *Programación Ágil*, sin tener un conocimiento adecuado de ellas y de sus principios, sólo con el objetivo de encadenar servicios en línea sin los principios necesarios para un desarrollo de software fiable y de alta calidad. Estas interpretaciones, en muchos casos amañadas, perjudican y generan información incorrecta en contra de otros esfuerzos que buscan incrementar continuamente la calidad y la seguridad de los productos software.

La industria de TI es relativamente joven, de hecho sólo ha sobrevivido a un par de generaciones; pero también es una especie de mina de oro que, comparada con otras industrias y de acuerdo con creencias muy generalizadas, es relativamente bien pagada, no requiere esfuerzos físicos excesivos y se ejerce sin requisitos mínimos de entrada. Esto significa que economías enteras han crecido viendo a las TI como un juego de números: *si acumulas suficientes personas alrededor de un problema es posible que desaparezca, y si esa cantidad no genera costos elevados entonces se puede acumular un montón mayor*. Por esto es que, desde una perspectiva puramente demográfica, es posible afirmar que la mayoría de personas está en esta industria porque: 1) se trata de un trabajo bien remunerado, en comparación con otros empleos de carácter intelectual o incluso manual o 2) no hay incentivos para hacer algo diferente a jugar cómodamente a los números.

Claro que también existen aquellos que optan por sobresalir construyendo buenos productos software. Ellos entienden que desarrollar bien es una habilidad, de hecho, toda una gama de habilidades: comprender y modelar el dominio del problema, entender y aplicar los lenguajes de programación, las librerías, los paradigmas, los idiomas, elegir qué aplicar en una situación dada, aprender y desarrollar algoritmos, comprender y dominar las fases de la Ingeniería de Software, automatizar procesos, conocer las teorías esbeltas para el suministro, la producción y el desarrollo de productos, aplicar la concurrencia y el paralelismo, destacar las bondades y potencialidades de las metodologías recientes, y así podría seguir la lista. Estos profesionales se quieren diferenciar de alguna manera, pero existen algunos interrogantes: ¿cómo potencializar sus habilidades y capacidades? Y ¿cómo pueden ayudar a otros en la industria a que tengan verdadero aprecio por el software que escriben? La respuesta puede ser que se necesita algún tipo de modelo de formación y una manera de identificar estas fortalezas, tanto entre los principiantes como en los experimentados. La cuestión es que en muchos casos el software se valora por la utilidad que proporciona, y no importa lo *feo* que sea internamente, siempre y cuando se entregue a tiempo. Un desarrollador de software puede construir un producto que *enamora* desde sus interfaces de usuario, pero lo que verdaderamente importa es que su interior, el código, se haya construido de acuerdo con las necesidades del cliente y con la seguridad y fiabilidad necesarias. *Ese es el producto de un desarrollador profesional*.

2.1 Desarrollar Software es un Arte y una Ciencia

Desarrollar software es tanto una ciencia como un arte. Es ciencia, porque en los procesos se deben considerar los principios de las Ciencias Computacionales y de la Ingeniería, y es arte, porque son tantas y tan diversas las variables involucradas en el desarrollo de productos software que no se puede tener solamente un proceso totalmente prescriptivo y repetitivo. De acuerdo con Brooks

(1986), en el desarrollo de software se articulan aspectos esenciales, que se corresponden con los elementos inherentes y abstractos relacionados a la naturaleza del software, y aspectos accidentales correspondientes a las dificultades para llevar a cabo el desarrollo mismo, debido a las complejas relaciones humanas de comunicación y colaboración, es decir, a lo que Sommerville (2006) clasifica como sistemas socio-técnicos.

En el arte, los productos tienen una belleza intrínseca en sí mismos. Una catedral realmente es una cabaña grande para que las personas se reúnan a orar; generalmente se construye de piedra para que dure más que una cabaña de madera, pero ¿por qué todo el material decorativo es de lujo? Por supuesto que está ahí para generar un sentido de grandeza y de imponencia, y llama la atención de aquellos que aprecian la belleza y magnificencia, por lo que entran con reverencia y humildad, listos para el culto. Lo que la hace arte es el trabajo y la belleza estética, por encima de su utilidad intrínseca y de la belleza estética. Entendemos el arte como un proceso creativo individual o colectivo, una profesión disciplinada donde los autores impregnan en la creación más que su intelecto, suman su alma y su impronta.

Existe una diferencia entre la mentalidad de un escultor, que esculpe la expresión en la cara de una gárgola, y un maestro de obra que utiliza bloques básicos para construir un parqueadero de varios pisos. En este caso, lo que menos se necesita es que el maestro esculpa su personalidad en la obra, de tal forma que los bloques tengan diferentes tamaños y que no sean intercambiables, lo que interesa es la funcionalidad. Cuando esto sucede, es decir, cuando el maestro de obra pierde de vista el objetivo de lo que está construyendo y coloca su firma, puede que logre una magnifica representación artística, pero que no responde a las necesidades del cliente. Entendemos que la Ingeniería de Software necesita más que esto, porque por sobre todas las cosas el arte, como cierto y de calidad, implica un compromiso y un resultado que trascienda aspectos básicos y cotidianos. Esto no implica dejar de lado las cuestiones técnicas ni la predominancia de la funcionalidad, pero el arte es una manera de hacer las cosas a nuestro entender y no sólo la búsqueda de la belleza, que es un concepto temporal y altamente subjetivo.

En los años 60, la comunidad del software aplicaba metáforas mecánicas para el proceso de desarrollo de software, pero actualmente la Ingeniería de Software es un área académica aceptada y un campo activo de investigación para las universidades y la industria. La perspectiva para el desarrollo de proyectos de esta ingeniería se define hoy como la aplicación de un enfoque disciplinado, cuantificable y sistemático, para desarrollar, operar y mantener productos software, es decir, es la aplicación de la ingeniería al software (IEEE, 1990). Este enfoque ha demostrado ser eficaz en el desarrollo de sistemas críticos de seguridad.

El software suficientemente bueno es una extensión lógica de las ideas de la Ingeniería de Software y de quienes la practican, pero que tienen un componente artístico. Es una labor de ingeniería en la que se intercambian recursos, calendarios, características y defectos. Por ejemplo, en el software del transbordador espacial la seguridad es una cuestión crítica, por lo que se tienen que minimizar los defectos, y las cuestiones artísticas pasan a segundo plano; pero las aplicaciones comerciales, como navegadores y editores gráficos, requieren una cantidad de características que se deben desarrollar rápidamente, la seguridad no es tan incidente pero asumen protagonismo cuestiones como combinación de colores, tipos de letra y la distribución del escritorio, es decir, la parte artística, además, los recursos son limitados por la necesidad de obtener un beneficio, por lo que su desarrollo se realiza pensando en la reducción de costos y de tiempo. Ambos enfoques de trabajo generan un mismo producto, software, pero con niveles diferenciados de seguridad,

fiabilidad, consistencia, ciencia y arte. Por eso es que el desarrollo de software es tanto arte como ciencia, *es una profesión* con igual o mayores responsabilidades que en la Ingeniería Civil.

2.2 Profesiones y Profesionales

En este proceso lo primero es comprender qué es una profesión y qué es un profesional, aunque la búsqueda de una definición universalmente aceptada para la primera es en sí misma una empresa prácticamente imposible. Mientras que las sociedades occidentales apuntan a la medicina como el ejemplo más sólido de profesión, otras no tienen una visión unificada. El propósito no es entrar en discusiones sin sentido, sino aclarar algunos conceptos de acuerdo con la opinión generalizada.

El diccionario define *profesión* como una ocupación, especialmente una que requiere formación avanzada. Desafortunadamente, utilizar el término como sinónimo de *ocupación* es superficial y simplista, y no aporta beneficios para el objetivo de este documento. Otros la definen como una *vocación* cuya práctica se basa en la comprensión de la estructura teórica de algún área del saber o de la ciencia, y sobre las habilidades que la acompañan, mientras que otra definición dice que son grupos que aplican *conocimientos especiales* al servicio de un cliente. Estas últimas definiciones presentan deficiencias, porque casi todas las ocupaciones modernas basadas en servicios involucran algún conocimiento teórico, y al calificarlas como profesiones pierden su significado distintivo. Una profesión tiene que ser algo más que una reunión de proveedores de servicios.

En los años 70 se intentó definir el término con base en el control que tenían las asociaciones sobre sus miembros y el mercado, y creían que la autonomía profesional era un índice útil para valorar la situación relativa de las diversas ocupaciones profesionales. El consenso era que la distinción más estratégica reside en la autonomía legítima y organizada, y que una profesión era diferente de otras en que se le ha dado el derecho de controlar su propio trabajo. Posteriormente, esta cuestión tomó un enfoque diferente, cuando se discutió si la odontología podía ser considerada como una profesión, y entonces se argumentó que una característica común a todas las profesiones es la *promesa de altruismo que hacen con la sociedad*. La asociación de una profesión con la autoridad moral y el bien público se basa en que los profesionales tienen el deber, no sólo el derecho, de hablar sobre cuestiones relacionadas con su experiencia para alentar a los ciudadanos a tomar decisiones informadas.

Otra definición, que tiene amplia acogida, dice que una profesión es un colectivo de proveedores de servicios expertos, que conjunta y públicamente se comprometen a priorizar las necesidades existenciales y los intereses del público, al que sirven sobre los suyos propios, y que a su vez reciben la confianza de éste para que lo hagan. Cuando una ocupación es reconocida como una profesión se firma un contrato social entre sus miembros y el público en general, y se sustenta en la confianza, el respeto, la condición social, el monopolio de la práctica y el derecho de auto-gobierno. Otra manera generalizada de definir una profesión es mediante el estudio de las características de las ya reconocidas, y examinar si la ocupación que se evalúa posee los mismos rasgos. Este enfoque, también conocido como de los rasgos o inventarios, se ha ampliado y utilizado para clasificar las ocupaciones en profesiones o semi-profesiones mediante el análisis a la ausencia o presencia de ciertos atributos. Desafortunadamente, este método tiene una desventaja evidente: los atributos se pueden convertir en una cortina de humo que sólo permite observar la apariencia y no el fondo de la cuestión.

Se observa entonces que el término *profesión* tiene varias definiciones: 1) es una *vocación* que requiere conocimiento especializado y una larga e intensiva preparación académica, y su práctica se basa en una comprensión de la estructura teórica de alguna área científica y de las habilidades que acompañan esa comprensión, 2) es una *capacidad* principal, una vocación, o un empleo y 3) es todo el *cuerpo de personas* que participan en una vocación. Por desgracia, la segunda definición describe con mayor precisión lo que actualmente se acepta para la Ingeniería de Software. Sin embargo, un creciente número de organizaciones, personas e instituciones están convencidas de que la primera definición es la más apropiada, y están trabajando por lograr su aceptación como tal. Estos movimientos han descubierto que un enfoque ingenieril para el ciclo de vida del software es menos caótico, produce mejores productos y también es rentable.

Ahora bien, ¿qué pasa con el término *profesionales*? En pocas palabras, los profesionales son miembros de una profesión reconocida, independientemente de si individual o colectivamente se comportan profesionalmente. Pero una mirada más profunda a esta definición, aparentemente obvia, revela dificultades elementales: la medicina es una profesión, pero ¿todos los practicantes de la medicina son automáticamente profesionales? Los médicos que tienen sus licencias suspendidas, pero que continúan en ejercicio y diagnostican pacientes y prescriben medicamentos dejan serias dudas acerca de si encajan como profesionales. Hay quienes piensan que debe ser la sociedad quien decida qué ocupaciones califican como profesiones, pero en lo que tiene que ver con la responsabilidad social y el bienestar de las comunidades debe ser el Estado el que decida acerca de estas cuestiones. Esto se puede lograr a través de licencias o certificaciones que emita un organismo oficial o un consejo independiente, las cuales, dentro de una profesión reconocida, trazan la línea entre los profesionales de confianza y los practicantes.

Entonces, al igual que el término *profesión*, un profesional puede ser: 1) alguien que se dedica a una ocupación o actividad profesional, es decir, alguien que se ajusta a los estándares técnicos y éticos de una profesión y 2) persona que ejerce una actividad por la cual recibe un beneficio económico. Si se tiene alguna duda acerca de que la segunda definición es la que refleja con mayor precisión a la mayoría de profesionales del software de hoy, basta con preguntarle a alguno de ellos acerca de las normas técnicas o éticas de su *profesión*.

2.3 Profesionalizar

Típicamente se acepta que profesionalización es la organización de una ocupación dentro de la forma asumida por los paradigmas de profesiones establecidas, como la medicina. En otros términos, es el proceso para promover una ocupación a una profesión, en el que se busca mejorar las habilidades de una persona con el fin de hacerla más competitiva en asuntos relacionados con su área de conocimiento. Los beneficios se reflejan al trasladar una serie de recursos —especialmente conocimientos y habilidades— a otros, que originarán recompensas sociales y económicas —para los profesionales—. A medida que una profesión adquiere mayor movilidad social y control de mercado podrá reflejar sus habilidades, experiencias y normas éticas. Esto se logra con productos fiables y seguros, que les permiten a sus miembros lograr el reconocimiento de la sociedad y afianzar su labor como profesión reconocida.

El trabajo de los profesionales se basa en proyectos, lo que implica que deben tener un principio y un fin claros, y un proceso metodológico de acciones deliberadas, planeadas e implementadas por los integrantes de los equipos. Esto es precisamente lo que se propone desde este Manifiesto para el desarrollo de software, porque el fin es obtener las recompensas que

acompañan a la condición de ser profesión. Para lograrlo, el primer paso consiste en llegar a un consenso acerca de su cuerpo de conocimiento, trabajo que se viene desarrollando desde varias organizaciones. La legitimación de la autoridad profesional consta de tres peticiones distintivas: 1) que el conocimiento y la competencia de los profesionales hayan sido validados por una comunidad o por sus pares o iguales, 2) que este conocimiento validado descanse en fundamentos racionales y científicos y 3) que el juicio profesional y el asesoramiento estén orientados hacia un conjunto de valores sustantivos, como la salud en el caso de la medicina. Estos aspectos de legitimidad corresponden a los tipos de atributos, colegiales, cognitivos y morales, usualmente incorporados en el término *profesión*. En este sentido, la Ingeniería como profesión se caracteriza por:

1. Una formación profesional inicial en un programa de estudios, validado por la sociedad a través de un registro y una acreditación de calidad

2. Registro de la aptitud para la práctica, mediante una certificación voluntaria o licencia obligatoria

3. El continuo desarrollo de habilidades especializadas y de formación profesional

4. Apoyo público a través de una asociación profesional

5. Un compromiso con las normas de conducta prescritas en un código de ética.

Por otro lado, para el desarrollo de una profesión es necesario articular un conjunto de conocimientos, porque esto representa un amplio consenso respecto a lo que debe conocer un profesional, por ejemplo, de la Ingeniería de Software. Este consenso también es un requisito previo a la adopción del desarrollo de competencias coherentes y de programas de formación continua. La profesionalización se puede definir, aunque vagamente, como el establecimiento de, y la adhesión a, un modelo profesional que no existe o no se ha arraigado con firmeza. En este modelo los individuos, que se refieren a sí mismos como profesionales, exhiben la mayoría de, si no todas, las características de los profesionales del área específica. Además, existe una *atmósfera profesional* establecida conjuntamente para los miembros individuales y para las asociaciones profesionales existentes.

El proceso de profesionalización se realiza en una serie de fases que no siempre se ejecutan de forma secuencial, es decir, algunas pueden ocurrir simultáneamente y otras lo hacen fuera de orden, pero siempre aprovechando el trabajo previo. La siguiente es una lista de esas fases:

1. *Reconocimiento de la necesidad de la profesionalización*. Esto suele ocurrir cuando se tiene y se siente la necesidad de los individuos altamente calificados y uniformemente capacitados, es decir, cuando el trabajo que realizan quienes ya están en esa área del conocimiento se convierte en fundamental y crítico en la naturaleza de las sociedades, en este caso, el software.

2. *Definición formal de la profesión*. Esto incluye establecer una terminología normalizada para la profesión, creación de títulos y descripciones para los miembros, y la identificación de relaciones, por ejemplo, profesión con profesión, profesional con cliente y profesión con público en general. Algo en lo que se pueden encajar iniciativas como SWEBOK.

3. *Identificación de asociaciones profesionales clave*. Se espera que estas sociedades hayan alcanzado cierto grado de estatus formal, por ejemplo, el respeto de la comunidad profesional, la

publicación de buenas revistas y la realización de reuniones locales y nacionales. Además, serán muy valiosas para avanzar en el resto del proceso de la profesionalización. Asociaciones como ACM e IEEE ya han adelantado estas cuestiones.

4. *Establecer criterios mínimos de ingreso*. Esto debe incluir temas como cierto grado de formación formal, experiencia demostrable y una certificación. La mayoría de asociaciones en el mundo tienen sus reglamentos bien establecidos.

5. *Establecer y reconocer programas de formación y de entrenamiento*. Que incluyen programas de estudios universitarios existentes y programas profesionalizantes de formación continua aprobados. Los Ministerios de Educación y los organismos de control tienen esto cubierto.

6. *Establecer procedimientos formales de certificación y renovación*. El proceso de certificación se debe basar en la mínima cantidad de habilidades y conocimientos útiles que necesita un profesional tipo. El proceso de renovación de la certificación se debe dirigir hacia las trayectorias profesionales de los asociados. Obviamente, los procesos de certificación y formación se afectan directamente el uno al otro. Para la Ingeniería de Software, tal vez sea la fase que requiere mayor trabajo en este momento.

7. *Recolección y promulgación de estándares profesionales*. Esto estándares cubren temas como procedimientos, metodologías, métricas, niveles aceptables de rendimiento y herramientas. Actualmente, se tienen diversas propuestas y se requiere unificarlas y aceptarlas ampliamente.

8. *Identificar y divulgar un código de ética profesional*. Se establece un código de ética que deben respetar todos los miembros. Lo ideal sería que ese código se incorporará en los procesos de certificación. Para la Ingeniería de Software ya existe un código considerablemente aceptado.

9. *Establecer objetivos de rendimiento cuantificables para la profesión y los profesionales*. Se espera que los profesionales satisfagan y superen un conjunto de objetivos de rendimiento para mantener su estatus profesional, para lo que es necesario conocer lo bien que se están desempeñando en su área de conocimiento. La profesión se debe esforzar continuamente para mejorarse a sí misma como un todo, por ejemplo, disminuyendo la tasa de error promedio por miembro. Ya existen criterios como las buenas prácticas que se podrían estructurar para alcanzar este objetivo.

10. *Identificar los requisitos y procesos de formación continua*. Las profesiones no son estáticas, especialmente las relacionadas con TI, y la vida útil de los conocimientos profesionales continúa disminuyendo. Por ejemplo, se dice que el conocimiento técnico de la raza humana se duplica cada seis meses, por lo que un profesional está obligado a tomar continuamente cursos de actualización para mantener su estatus. El protagonismo aquí lo deben asumir las universidades, porque la Ingeniería de Software requiere especializaciones y cursos de actualización permanente.

11. *Identificar las responsabilidades y obligaciones profesionales*. Los profesionales deben ser conscientes de sus responsabilidades para con sus clientes, sus empleadores, otros profesionales y con la profesión en general. Además, deben ser consecuentes con las obligaciones que esas responsabilidades generan. Sólo las personas que están legalmente dementes no son responsables por sus acciones. Será el Estado quien legisle a este respecto.

12. *Establecer mecanismos legales de ingreso y de permanencia*. El estado de una profesión se ve disminuido por la inclusión de personas que no cumplen con las normas establecidas y por las violaciones que sus asociados cometen. El Estado debe reglamentar este ejercicio profesional, que a su vez influye para que las asociaciones delimiten los mecanismos.

13. *Establecer y mantener una imagen pública positiva*. Una profesión con una imagen pública positiva pueden conseguir mejores beneficios para sus miembros, incluidos niveles salariales más altos. Se debe hacer público el trabajo y los logros de los profesionales en Ingeniería de Software, tanto positivos como los errores, para que la sociedad mantenga una imagen reciente de los beneficios de esta profesión.

Obviamente, aunque lograr todo esto toma tiempo, se puede llevar a cabo en un período relativamente corto, por ejemplo, unos tres o cuatro años, tiempo que se puede reducir mediante un esfuerzo concentrado de parte de los propios profesionales. En el caso de la Ingeniería de Software, y como se evidencia en las fases, ya se han hecho importantes avances en pro de la profesionalización.

2.4 Responsabilidad profesional de los ingenieros de software

Como puede observarse, las fases para la profesionalización de la Ingeniería de Software han evolucionado, aunque no de la forma esperada; inclusive, actualmente, parecen estar en retroceso debido a la falta de unificación de criterios y la necesidad de desmontar los intereses individuales. Las consecuencias se reflejan en proyectos con sobrecostos, pérdidas por incumplimiento, pérdida de ingresos y mayores costos de oportunidad, baja moral del personal y mala calidad de software en general. Desafortunadamente, poco se está haciendo en términos generales para cambiar este *statu quo*, y la industria continua contratando *programadores*, poco cualificados y a bajos salarios, mientras que los *desarrolladores* experimentados pierden apreciación en la industria y la academia.

Diversas investigaciones han concluido que hasta el 80% de los proyectos software no llegan a terminarse o son rápidamente retirados y sustituidos por otro de *última y mejor* versión, sólo para continuar una tradición de oportunidades perdidas, expectativas no satisfechas y pérdidas monetarias. Se puede decir entonces que el negocio de desarrollar software se encuentra en crisis, pero estamos convencidos con un poco de orgullo y compromiso de parte los involucrados en esta industria será posible alcanzar su recuperación. De acuerdo con estas apreciaciones, a continuación se describen algunas responsabilidades profesionales de los ingenieros de software:

1. *Comprender los requisitos reales del proyecto y mantenerse centrado en ellos*. Pocos proyectos software tienen éxito sin antes definir las fronteras y los límites del entorno en el que se ejecutarán, y las necesidades de los usuarios que los utilizarán. En ausencia de esta base, se tomarán muchas y malas decisiones de parte del equipo de desarrollo, y como resultado, el producto tendrá prioridades inadecuadas y soluciones mal ajustadas. La comunicación abierta, honesta y frecuente con el usuario es esencial para recoger esta valiosa información desde el principio en el ciclo del proyecto.

2. *Comprender quién es el cliente*. El cliente para un proyecto dado no siempre es una persona o un grupo obvio, por lo que el ingeniero debe tener la habilidad necesaria para identificarlo y reconocerlo, en beneficio del proyecto y del futuro producto. Es importante tener en cuenta quién es el cliente a fin de mantener la perspectiva correcta de las prioridades del proyecto.

3. *Mantener comunicación abierta, frecuente y directa con el cliente*. El enfoque que aplican comúnmente los equipos para construir productos software es contratar el desarrollo con una empresa dada, del mismo modo que un inventor contrata a un fabricante para manufacturar su producto. Irónicamente, con el software, este enfoque falla regularmente, debido a una comunicación deficiente entre el cliente y el equipo de desarrollo. Los ingenieros de software profesionales tienen la experiencia y el conocimiento para hacer las preguntas correctas, para determinar los requisitos adecuados, para proporcionar información de calidad y oportuna, y para realizar el asesoramiento y la consultoría necesarias para mantener el proyecto en marcha.

4. *Producir un diseño para todos y no sólo para proyectos simples*. Un programa no especificado no puede ser incorrecto, sólo puede ser sorprendente.

5. *Generar un plan para implementar el diseño*. Sin un plan es posible que no se alcancen los objetivos, que se pierda el rumbo y que se elimine cualquier posibilidad de alcanzar pruebas significativas.

6. *Elegir las mejores herramientas para el trabajo, no las más populares, y de acuerdo con las necesidades del proyecto propuesto*. Las nuevas tecnologías se deben elegir por la oportunidad de aprender, no porque sean las mejores para el trabajo. A menudo, la tecnología subyacente que se utiliza para poner en práctica un proyecto se elige por razones equivocadas. Otras veces se eligen sólo por ser la más popular del momento. Estos criterios son comunes, pero perjudiciales para el éxito potencial del proyecto. No se debe elegir una herramienta sin una revisión exhaustiva de los requisitos.

7. *Mantener un alto nivel de profesionalismo*. La formación permanente es fundamental para lograrlo.

8. *Mantener la disciplina necesaria para alcanzar estándares consistentes de codificación y de prueba*. Los estándares de desarrollo y de prueba son necesarios para ahorrar tiempo y dinero a largo plazo, conducen a tener menos errores, los que generalmente son más fáciles de rastrear y de resolver y que se comienza a probar en las fases tempranas del ciclo de vida, por lo tanto acortan el tiempo de desarrollo global y el del ciclo de prueba.

9. *Ejecutar pruebas completas y adecuadas al código para asegurar los más altos niveles de fiabilidad y facilidad de mantenimiento en el producto.* Uno de los objetivos de la Ingeniería de Software es generar productos con la calidad y fiabilidad suficientes para que la sociedad los acepte y utilice, pero para lograrlo se debe implementar un adecuado plan de pruebas.

10. *Comprender que la fase de mantenimiento es la más cara de cualquier proyecto software*. Por lo que se deben seguir estándares que ayuden a reducir este costo a través de una buena documentación y de comentarios en el código.

2.5 Recomendaciones para alcanzar la profesionalización

La Ingeniería de Software se ocupa de la creación y de la aplicación de fundamentos ingenieriles para el desarrollo de software y del uso intensivo de productos técnicos, procesos que se logran mediante un análisis sistemático y trabajo en equipo. El concepto de la formación de ingenieros de software profesionales requiere sujetos, secuencias sistemáticas y flexibilidad para hacerle frente a

las necesidades del mercado, los requisitos de la industria y los rápidos y abundantes desarrollos tecnológicos. Uno de los objetivos de este Manifiesto es integrar conceptos como formación, formalización y profesionalización en todos los ámbitos relacionados con el desarrollo de software. Aunque no se puede aseverar que el simple seguimiento de un algoritmo sea suficiente para alcanzar la profesionalización de un área de conocimiento, a continuación se plantean unos pasos que pueden ayudar a lograrlo:

1. *Iniciar el proceso de una vez*, con el objetivo de alcanzar resultados positivos visibles en no más de dos años. Específicamente, las funciones del desarrollador y del administrador promedio se deben ver afectadas por el proceso de profesionalización antes de esa línea de tiempo.

2. *Invitar a muchos y diversos actores a participar y a opinar*. Incluyendo a desarrolladores, analistas, gestores, profesores, gobierno, sociedades profesionales y a profesiones establecidas, entre otros.

3. *Dar a conocer el progreso del proceso*, y mantenerlo visible y asequible para todos los interesados, de forma que con sus opiniones y aportes se enriquezca y perfeccione en todo momento.

4. *Establecer un comité de mantenimiento para la profesionalización*. El trabajo de este comité incluirá el seguimiento a los cambios en el proceso, introducir estos cambios de manera ordenada y actuar como un *tribunal supremo* de las controversias que surjan.

5. *Fomentar el establecimiento de planes de estudio* para el programa de Ingeniería de Software. Por otra parte, promover el concepto de profesionalización en todos los niveles de formación relacionados con el software.

Este documento queda abierto para todo tipo de opiniones, aportes y modificaciones racionales, que permitan integrar la base sobre la que se realizará el trabajo futuro en pro de alcanzar el objetivo de *profesionalizar el desarrollo de software*. Esperamos la retroalimentación de la comunidad, para enriquecerlo y llevarlo a las esferas necesarias, y para que los actores involucrados hablemos el mismo idioma en todas las áreas de nuestro accionar.

SEGUNDA PARTE

LA PROFESIONALIZACIÓN DEL DESARROLLO DE SOFTWARE

El propósito de este reporte es describir la situación, los objetivos y el alcance de la iniciativa de profesionalización del desarrollo de software. Este informe actúa entonces como un manifiesto para la colectividad, y aclara el papel que queremos desempeñar, tanto en la comunidad científica, como en la académica, la industrial y el Estado. Aquí es importante aclarar nuestra comprensión acerca de la Ingeniería de Software, ya que es un término con amplio uso pero que tiene diferentes significados para cada persona y situación. En pocas ocasiones la visión global de la Ingeniería de Software es completamente positiva, por lo que es un área temática y una profesión que sufre de una sobrecarga de tecnología y de falta de rigurosidad histórica.

Para nosotros, *la Ingeniería de Software tiene por objeto mejorar la práctica del desarrollo de software de alta calidad*. Esta área del conocimiento es el centro de las Ciencias Computacionales, porque generar software de alta calidad para correr de forma eficiente en el hardware es el objetivo principal de la investigación en este campo. Paradójicamente, y debido a su importancia, puede ser difícil identificar lo que es exactamente, y sobre todo, la forma en que se diferencia de otras áreas de investigación de estas Ciencias. Sin embargo, creemos firmemente que todas las áreas de las Ciencias, especialmente aquellas en las que el manejo de información es esencial, están altamente relacionadas con la Ingeniería del Software y sus productos. Además, la sociedad tiene una interacción directa con todo tipo de sistemas y software —de ahí la denominación de *software-dependiente*—, por lo que la calidad y la fiabilidad de estos productos son la piedra angular de un buen desarrollo de software. Pero la Ingeniería de software se ha convertido en el *patito feo* de nuestro oficio, ya que es invitada a las reuniones familiares pero nunca a una noche de paseo por la ciudad. Creemos que varias causas han originado y enfatizado esta situación, como el hecho de que gran parte de la industria y de los programadores de software han interpretado a su amaño el *Manifiesto para el Desarrollo Software Ágil*, y se olvidan de las ventajas que esta iniciativa le aporta al logro de productos a tiempo y en el presupuesto (Intelligent Tings, 2009). Algunas de estas interpretaciones son:

1. Lo mejor es saltar inmediatamente a escribir código y dejar de preocuparse por escribir requisitos y especificaciones de diseño completos. Esta es una *buena práctica* teniendo en cuenta que la mayor parte del código que escriben, al faltarles formación adecuada, no tiene el arte ni la esencia científica que requiere para la calidad y la fiabilidad. De todos modos, siempre quisieron hacer las cosas a su manera, e interpretan que existe un principio que les dice que está bien.

2. No tenemos que perder el tiempo escribiendo documentos para explicar lo que estamos haciendo. En todo caso, sino comprenden lo que necesitan, entonces sobra el material técnico.

3. Estamos seguros de que nuestros empleadores estarán encantados de recibir lo que producimos, porque les entregamos a tiempo, cuando antes no sabían si iban a conseguir nada en absoluto.

4. Siempre admiramos a aquellos raros *desarrolladores* ingeniosos que pueden *hackear* nuestras bases de código y escribir otro tan brillante que nadie siquiera lo puede comprender. Ahora tenemos un nuevo nombre para ellos: *codificadores ágiles* (como parodia).

En caso de que no haya quedado claro hasta el momento, estas interpretaciones son muy, muy perjudiciales para la calidad de los productos software. Las personas creen que la Ingeniería de Software trata sólo y exclusivamente de dominar la sintaxis y las *APIs* de un lenguaje, y que la calidad y la tasa de éxito de nuestros proyectos sigue siendo el mismo año tras año. Creemos que se debe considerar cada metodología y cada principio que aporte al logro de productos de calidad, pero dentro de los lineamientos de un profesión en proceso de maduración, y no como espejos para distraer del objetivo de desarrollar software serio, responsable y que cumpla los requisitos del cliente y de la sociedad.

Por todo esto estamos de acuerdo con lo que expresa McConnell (2003), acerca de que las prácticas necesarias para crear buen software fueron establecidas y están fácilmente disponibles desde hace más de 20 años, pero han sido manipuladas e interpretadas a la luz de intereses personales, y los productos resultantes, a pesar de algunos éxitos sorprendentes, no están a la altura de las necesidades. Existe un amplio abismo entre las prácticas normales y la mejor, y muchas de uso generalizado son peligrosamente obsoletas y de poca utilidad, por lo que el desempeño promedio de los proyectos software, que se trabajan bajo está lupa, deja mucho que desear y diversos desastres conocidos lo constatan. Por esto es que estamos trabajando para que la comunidad del software comprenda, aplique y experimente los principios de cada propuesta antes de decidir aplicarla sin razón, porque aunque las buenas prácticas están claras en la teoría desde hace años, pero la puesta en práctica, la contextualización y el compromiso con el que se llevan adelante son el problema principal, y es lo que en parte genera las crisis y las falencias de la industria Software, no las tendencias metodológicas en sí.

Por último, creemos que la Ingeniería de Software es una disciplina práctica, por lo que su investigación nos obliga a participar en el trabajo experimental. Por lo tanto, trabajamos con empresas y organizaciones en proyectos reales, a menudo como parte de proyectos de investigación financiados por la academia, la industria o el Estado, pero también en proyectos de consultoría y de formación, incluso dentro de las mismas empresas. Esto nos permite estar en sintonía con sus problemas y poner a prueba nuestras ideas en el campo, porque sólo la investigación que se valida en el mundo real se puede decir que demuestra su eficacia.

INTRODUCCIÓN

El software es el elemento clave y se ha involucrado en los principales desarrollos tecnológicos. La rápida evolución de la tecnología informática ha propiciado la definición de nuevos servicios y esquemas de trabajo en las organizaciones, y les ha permitido mejorar la calidad de servicio a sus clientes, definir nuevos servicios, conquistar nuevos mercados, y en general ser más competitivas. La operación de las compañías se encuentra cada vez más soportada en sistemas de información intensivos en software, que son fundamentales para apoyar su liderazgo estratégico en el mercado o, por el contrario, propiciar su fracaso (Anaya, 2006). Igualmente, el software ha incursionado en los campos del entretenimiento y el hogar, modificando el estilo de vida y abriendo posibilidades a nuevas formas de trabajo y nuevos modelos de negocio.

La Ingeniería de Software es una disciplina relativamente joven con respecto a otras ingenierías, pero al igual que para las demás, su reto es solucionar problemas en un mundo en constante cambio, y lograr una adecuada relación costo-beneficio, aplicando principios matemáticos y de las Ciencias Computacionales (SEI, 1990). Las características particulares del producto software, la complejidad y la dinámica del contexto, y la rápida evolución de la electrónica y estas Ciencias, le imponen una complicación particular a la disciplina, lo que la diferencian de las demás (Bruegge & Dutoit, 2002; Maibaum, 1997). Hoy en día, el software es reconocido como un sistema socio-técnico que comprende uno o más sistemas, pero crucialmente, también incluye conocimiento acerca de cómo utilizarlo para alcanzar un objetivo más amplio. Este tipo de sistemas incluye a las personas como partes inherentes del mismo, son gobernados por políticas y reglas organizacionales, y se pueden ver afectados por restricciones externas, como las leyes nacionales y las políticas reguladoras (Sommerville, 2006).

Al analizar el estado actual de la Ingeniería de Software se podría tomar dos posturas: 1) la *pesimista*, planteada desde la conferencia de la NATO en 1968, cuando se habló por primera vez de la *crisis del software*, y que, aún hoy, muchos consideran que no se ha superado debido los resultados poco halagadores que con frecuencia se observan en los proyectos: incumplimiento sistemático de los plazos de entrega, desfase en los tiempos y presupuestos, y baja calidad del producto final. Pressman (2005) define esta problemática como *aflicción crónica*, pero afortunadamente existen movimientos que buscan solucionarla, como el SEMAT. 2) La *optimista,* que sin pretender desconocer las problemáticas que actualmente se presentan, busca enfrentar la alta complejidad del desarrollo de software en entornos altamente cambiantes, entiende los desafíos propios de los sistemas socio-técnicos, y reconoce la evolución positiva y los resultados logrados por la Ingeniería de Software hacia su madurez. Tal como lo planteó Ambler (2010) en Zurich: "*Somos más éxitos de lo que pensamos; la definición de éxito de un proyecto debe evaluarse en función de su contexto; cada organización define sus criterios de éxito del proyecto que van más allá de los criterios de tiempo, presupuesto y alcance*".

Actualmente no es posible concebir el mundo moderno sin software. Las infraestructuras nacionales y todas sus utilidades están controladas por sistemas basados en computadores, y los productos eléctricos de control incluyen un procesador y el software necesario. La fabricación y distribución industrial está completamente informatizada, y la industria del entretenimiento, como la música, los video-juegos, el cine y la televisión, utilizan software intensivo. Por lo tanto, la Ingeniería de Software es esencial para el funcionamiento de la sociedad de este siglo. Pero, debido a que los sistemas software son abstractos e intangibles, a que no están limitados por las

propiedades de los materiales, ni gobernados por las leyes físicas o por los procesos de fabricación, se pueden convertir en extremadamente complejos, difíciles de entender, y costosos de modificar.

Aunque muchas personas en diferentes contextos escriben programas, como en los negocios para simplificar el trabajo, los científicos e ingenieros para procesar sus datos experimentales, y los aficionados para su propio interés y disfrute, esto no las califica como desarrolladores, o ni siquiera programadores en muchos casos. Además, la mayor parte del desarrollo de software es una actividad profesional en la que los productos se desarrollan para fines comerciales específicos, para incluirlos en otros dispositivos, o como productos de los sistemas de información, que están destinados a ser utilizados por alguien diferente a su creador, y que generalmente los desarrollan equipos en lugar de individuos.

Esta situación ha hecho que muchos piensen que *software* no es más que otra palabra que se relaciona con los computadores. Sin embargo, cuando se habla de Ingeniería de Software, el término no sólo se relaciona con programas, sino también con la documentación asociada y con los datos de configuración que se requiere para hacer que los programas funcionen correctamente. A menudo, un sistema software es la combinación de varios programas, porque suele consistir de una serie de componentes independientes y de archivos de configuración, utilizados para implantarlos en un entorno de trabajo. Es por esto, que si alguien escribe un programa por sí mismo, que nadie más va a usar, no tendrá por qué preocuparse de escribir las guías de uso, ni la documentación de diseño. Sin embargo, si está escribiendo software que otras personas utilizarán y otros ingenieros modificarán, por lo general tendrá que proporcionar información adicional, además del código del programa. Este es un principio de las actividades de Verificación y Validación, porque al hablar de la calidad de software, se deba tener en cuenta que el producto lo utilizan y modifican personas diferentes a sus desarrolladores. La calidad no sólo tiene que ver con lo que hace el software, sino que también incluye el comportamiento mientras se ejecuta, y la estructura y organización de los demás programas del sistema, lo mismo que la documentación asociada.

Generalmente, los desarrolladores de software adoptan un enfoque sistemático y organizado para realizar su trabajo, porque es la manera más eficaz para producir software de alta calidad. Sin embargo, los programadores parecen no acercarse a seleccionar un método adecuado, y a veces ni siquiera lo tienen en cuenta, lo que genera una serie de circunstancias a través de *interpretaciones* poco creativas, menos formales y más rápidas, de las propuestas existentes, que aplican en sus proyectos. El desarrollo profesional de software es importante por dos razones:

1. Los individuos y la sociedad dependen cada vez más de sistemas software avanzados, por lo que se requiere suficiente capacidad para producirlos de forma fiable, segura, eficiente y fidedigna.

2. Generalmente, a largo plazo es más económico aplicar métodos y técnicas de Ingeniería de Software para desarrollar sistemas en lugar de sólo escribir programas, como si fuera un proyecto personal. Para la mayoría de sistemas, los costos más elevados tienen que ver con las actualizaciones y modificaciones posteriores a la implementación.

Por todo lo expuesto hasta el momento, es que mantenemos nuestra posición de que se necesita profesionalizar el desarrollo de software, y porque, a diferencia de otras actividades, parece no tener una regulación adecuada en cuanto a requisitos de empleabilidad de sus practicantes, ni de las responsabilidades sociales, y mucho menos de una formación acorde con las exigencias de este siglo.

1. SITUACIÓN ACTUAL DEL DESARROLLO DE SOFTWARE

Como se ha mencionado en repetidas ocasiones en este texto, el software se relaciona con todos los aspectos de la vida cotidiana: manufactura, banca, viajes, comunicaciones, defensa, medicina, investigación, gobierno, educación, entretenimiento, leyes, entre otros. Es parte esencial de los sistemas militares y se utiliza en diferentes sectores civiles, incluyendo los de misión crítica. Por desgracia, muchos sistemas de educación superior parecen no seguirle el paso a los cambios que generan esta situación, y los programas de ciencia e ingeniería, de pregrado y posgrado, necesitan incorporar más capacitación en Ingeniería de Software.

El diseño y construcción de software debe seguir procesos y procedimientos que también se aplican en otras disciplinas ingenieriles (Sommerville, 2006; Glass, 2006). En primer lugar, definir cuidadosamente los requisitos para luego desarrollar la arquitectura del sistema, y una vez definidos comenzar el desarrollo de código, a la vez que paralelamente se diseña e implementa el plan de pruebas, y no dejarlo como la fase final antes de la entrega. De acuerdo con el tipo de modelo de ciclo de vida utilizado y el tipo de sistema desarrollado, existen diversas formas de llevar a cabo estos pasos, y en el proceso también es necesario considerar los costos, interactuar con las personas y monitorear las responsabilidades del equipo de desarrollo. Si bien este proceso sigue los mismos pasos generales que para diseñar y construir cualquier sistema complejo, por ejemplo, puentes, aviones y equipo informático, la naturaleza intangible, maleable y cambiante con alta dependencia del contexto de su producto, obliga a trabajar con una alta flexibilidad entre sus fases. En ese sentido, es difícil asimilar el desarrollo de software a una línea de producción, donde todo está controlado, porque la *fabricación* de cada producto software sigue una línea de desarrollo difícilmente repetible en otro proyecto (Sommerville, 2006).

De acuerdo con la National Academy of Science (Jorgenson & Wessner, 2006), el software no es más que un artículo de comercialización básico que, de hecho, incorpora en sí a la función productiva de la economía. A pesar de la gran capacidad de los productos generados, el National Institute of Standards and Technology (NIST) estima que sus errores tienen un alto costo para la economía de los países (NIST, 2010). Aun así, la formación en Ingeniería de Software en el mundo no recibe la atención que merece, a pesar de esta importancia manifiesta para la economía. Para ayudar a disminuir estos problemas se requieren ingenieros de software mejor formados, a través de programas profesionales rigurosos, que estén al mismo nivel de cualquiera de las ingenierías con mayor historia. El asunto del reconocimiento de la Ingeniería de Software como una disciplina no es nuevo, y en los últimos años los debates se han incrementado (IEEE & ACM, 2004; Vaughn, 2000). El Institute of Electrical and Electronics Engineers (IEEE) y la Association for Computing Machinery (ACM) la definen como la *aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento de software* (IEEE & ACM, 2004). Un buen resumen acerca de esta cuestión se puede encontrar en Cook y Dupaix (1999).

1.1 Evolución de los enfoques de ingeniería

La modularidad, la reutilización y la abstracción en ingeniería son los principios que han llevado a proponer aproximaciones metodológicas para analizar, comprender y modelar el problema, y para buscar una propuesta de solución. Esta evolución está estrechamente relacionada con el desarrollo de los lenguajes de programación y de especificación. Del enfoque estructurado, donde la definición de unidades modulares planteaba una solución elegante al problema de código tipo *spaguetti* y que favorecieron el desarrollo de aplicaciones orientadas a bases de datos, surgieron los enfoques:

1. *Orientado por objetos*, en el que se fortalece la modularidad alrededor de unidades con estructura y comportamiento bien delimitados (Meyer, 1998).

2. *Orientado a componentes*, que enfatiza la definición de modelos de componentes (CORBA, .NET, JEE) y el desacoplamiento de las unidades modulares, a través de contratos claramente definidos mediante interfaces provistas y requeridas, y que fomenta el desarrollo de un mercado de componentes (COTS) y la construcción de soluciones a partir de elementos desarrollados por terceras partes (Szyperski, 1998; Daniels & Cheesman, 2001).

3. *Orientado a aspectos,* que enfrenta el reto de la modularidad de intereses transversales, definiendo módulos que a la vez definen la manera de intervenir y extender una funcionalidad base (Clarke & Baniassad, 2004).

4. *Orientado a servicios,* que busca romper la barrera de la tecnología impuesta por los modelos de componentes para proponer intercambio de mensajes entre módulos débilmente acoplados basados en estándares multiplataforma (Papazoglou & Heuvel, 2006), que asociados a los enfoques BPM favorecen la definición de funcionalidades de alta granularidad que potencian los procesos de negocio (Bajwa et al., 2008).

Sin embargo, hoy en día es difícil hablar de una aproximación metodológica imperante, porque el desarrollador de software debe evaluar en un problema real cuál es la más adecuada para desarrollar el sistema, e incluso plantear soluciones que articulen adecuadamente varias de propuestas. Las soluciones que se demandan son tan complejas que plantean la necesidad de escalabilidad e interoperabilidad, integración de componentes desarrollados por terceros, y definición de servicios con integración de sistemas legados (Boehm, 2011). Esta complejidad creciente del software impone dos connotaciones diferentes en el diseño:

1. *La arquitectura* o diseño de alto nivel, que define la estructura en módulos o componentes con funcionalidad claramente establecida, lo mismo que su interrelación, y donde se toman las decisiones de diseño relevantes, como patrones o tácticas de arquitectura, para atender los requisitos no funcionales y buscar que se cumplan los acuerdos de niveles de servicio establecidos con el cliente.

2. *El diseño detallado,* que aplicando diferentes modelos especifica la estructura o comportamiento interno de cada una de las partes, y los algoritmos o estructuras de datos a utilizar para implementar la funcionalidad de cada uno de los módulos.

Estas dos perspectivas dan origen al rol del arquitecto de software, que es un líder técnico responsable de las decisiones de diseño de alto nivel, que debe buscar acuerdos entre los involucrados en el proyecto y que es responsable de la definición de arquitecturas de referencia para utilizar en una tecnología en particular. Por otro lado, el auge del UML y del desarrollo de tecnología alrededor del XML, como lenguaje de etiquetado para describir y transformar documentos, plantearon nuevos desafíos para la reutilización, los cuales buscan dar relevancia a los modelos como elemento central del desarrollo, y a la definición de transformaciones automáticas o asistidas que permitan general parcial o totalmente el código. Model Driven Arquitecture (MDA) es una de las aproximaciones más conocidas —liderada por la OMG—, que propone un proceso de transformación a través de modelos en diferentes niveles de abstracción, como CIM, PIM y PSM (Kleppe, et al., 2004). Actualmente, el enfoque de desarrollo dirigido por modelos (MDSD, por sus

siglas en inglés) está siendo ampliamente trabajado con el propósito de crear poderosos lenguajes de dominio específico, que permitan definir la semántica del dominio del problema y favorecer el desarrollo de una línea de productos software. Estos enfoques son apoyados por comunidades como Eclipse, que ofrecen entornos robustos orientados al desarrollo basado en modelos.

La evolución de la electrónica, la tecnología de dispositivos móviles y las comunicaciones, abre la posibilidad para nuevos tipos de aplicaciones software, que proponen formas innovadoras de enfrentar los problemas de la humanidad y que representan nuevos desafíos para el desarrollador de software, como la confiabilidad y la escalabilidad (Boehm, 2011). Para propósito de ilustración se mencionan ejemplos de algunos sistemas que caen en esta categoría: sistemas dependientes del contexto que monitorean la salud de un paciente, sistemas para el control ambiental, sistemas embebidos que controlan entornos de producción, redes sociales que favorecen un trabajo colaborativo, minería a grandes volúmenes de datos para ofrecerles a los usuarios servicios relacionados con su perfil, o entornos distribuidos de realidad virtual utilizados para entrenamiento de tareas altamente especializadas, entre otros.

1.2 La Naturaleza del proceso software

El estudio de esta cuestión se inició en los años 80, como una línea de trabajo autónoma, y actualmente es uno de los elementos de la Ingeniería de Software más complejos de entender y analizar. Existen diferentes variantes, niveles de abstracción y connotaciones para lograrlo:

- *Modelos de ciclo de vida*, que plantean guías generales de las maneras como puede evolucionar un producto durante su desarrollo.

- *Métodos o técnicas*, que definen formas de trabajo para tareas con un alcance controlado.

- *Marcos metodológicos*, que recopilan un conjunto de técnicas y definen roles, guías, lineamientos y artefactos para realizar diversas actividades relacionadas con el desarrollo de software.

- *Modelos de procesos de alto nivel*, que recopilan un conjunto de prácticas reconocidas por comunidades y expertos. En esta categoría se reconoce el esfuerzo, aunque con intencionalidades diferentes, de entidades como el Software Engineering Institute (SEI Pittsburgh, USA) y el European Software Institute (ESI, Bilbao, Spain), y de entidades de estandarización como la ISO, que definen modelos y estándares de calidad que representan referentes internacionales, como CMMI®, METRICA, ISO12207 e ISO15504.

En este sentido, uno de los hitos que marcaron un nuevo rumbo a la concepción de este proceso fue el *Movimiento Ágil*, que plantea una reacción justificada a los métodos robustos y a la burocracia de los marcos metodológicos existentes, y rescató el valor de las personas como los principales actores del desarrollo de software (Agile Alliance, 2001). Afortunadamente, las tendencias de modelos y estándares robustos y los enfoques ágiles, que parecían ir por rumbos diferentes e irreconciliables, han madurado y buscan acortar las brechas. Hoy se conocen resultados positivos de empresas que han adoptado enfoques ágiles y que a la vez están articulados al esfuerzo de madurez y mejora del proceso con CMMI, surgido inicialmente para atender las necesidades de las grandes empresas (Navarro & Garzás, 2010). También se realizan esfuerzos para escalar otros enfoques, desde los asociados con equipos y proyectos pequeños, a proyectos más complejos (Ambler, 2009).

Uno de los retos propuesto por Boehm (2011) está justamente asociado a dos características esenciales en el proceso software: 1) *capacidad para generar resultados a corto plazo* y 2) *capacidad para adaptarse a las condiciones particulares de un proyecto*. Surge entonces el interrogante de cómo darle al proceso las características de agilidad, flexibilidad y adaptabilidad sin perder el control sobre el objetivo final, para generar una solución software con los niveles de calidad y satisfacción esperados, y permitiendo a la vez una disciplina de trabajo orientada hacia la madurez del equipo y de las organizaciones de software. Para buscar una respuesta, los esfuerzos de investigación alrededor del proceso se orientaron entonces bajo la premisa de que el *proceso software también es software* (Fugetta, 2002). Es decir, el problema de modularidad y reusabilidad, que ha derivado nuevos enfoques de desarrollo, es el mismo que se debe ser enfrentar desde la perspectiva del proceso software. En lugar de definir metodologías pesadas y poco flexibles, que difícilmente se pueden aplicar en todos los proyectos, se requiere la definición de activos software que describan formas de trabajo con un nivel de granularidad controlado, y que se puedan integrar y adaptar a un proyecto particular.

Visto de esta manera, la línea base del proceso software, que se va a aplicar en un proyecto particular, surge como una estrategia de composición a partir de activos software, los cuales son definidos y almacenados en bibliotecas. Esta visión del proceso ágil y flexible a partir de componentes predefinidos, es el principio que rige líneas de trabajo como la Ingeniería del Método Situacional (Mirbel & Ralyté, 2006) y la Ingeniería del Proceso Software (Ruiz, 2007), y es la motivación básica sobre la cual se soportan (SEMAT).

1.3 La práctica en la Ingeniería de Software

Uno de los elementos centrales, que facilita esta nueva perspectiva del proceso y que promete la articulación de enfoques que parecían irreconciliables, es el concepto de las prácticas, concebidas como *formas de trabajo propuestas para atender una necesidad particular*. Los modelos de calidad, como CMMI, ISO-SPICE e ISO9000, han articulado un conjunto de prácticas consideradas útiles y reconocidas por la industria de software, que se consideran generalmente como *mejores prácticas* (*best practices*), y la comunidad ha realizado trabajos que buscan evaluar cuáles son la prácticas aplicadas en la industria de software y su incidencia en las condiciones organizacionales (Cater-Steel & Bus, 2004).

Una práctica proporciona una manera de abordar, de forma sistemática y verificable, un aspecto particular de un problema, con un inicio y un fin claramente definido y con mecanismos que permiten verificar si se han logrado los objetivos. Las prácticas pueden ser desarrolladas, aprendidas y adoptadas por separado, y utilizar en conjunción con otras para crear formas de trabajo comprensibles y coherentes (Jacobson et al., 2007). Desde los enfoques ágiles se prefiere el uso de *contextual practice* por encima de *best practice*. En este sentido, Scott Ambler + Associates (http://www.ambysoft.com/essays/bestPractices.html) afirma que como profesionales, los ingenieros de software se deben esforzar para comprender el contexto en el que se encuentran, para luego aplicar la práctica que mejor se adapte al mismo. En otras palabras, se deberían centrar realmente en *las prácticas contextuales* y no en las *mejores prácticas.*

Otro aspecto relacionado con las prácticas es su nivel de adopción. No es de extrañar que algunas propuestas en la literatura sean valiosas en sí mismas, pero no han llegado a ser aplicadas en contextos reales. Existen barreras para la adopción de ciertas prácticas, y algunas innovaciones son más complejas e imponen una carga mayor de conocimientos, por lo que la importancia

percibida de las prácticas afecta su tasa de adopción. Es aquí donde la disposición del equipo de trabajo y su apertura al cambio juegan un papel importante para la adopción adecuada de nuevas formas de trabajo.

1.4 Condiciones de contexto

Las condiciones de contexto en las que actualmente se desarrolla el software son altamente complejas, a la vez que tienen una influencia relevante en el éxito o fracaso de los proyectos. Continuando con la argumentación anterior, acerca de que la aplicación de las prácticas deben ser consecuentes con el contexto en el cual se van a aplicar, algunos de los factores que ayudan a contextualizarlas son el tamaño del equipo, la distribución geográfica, las normas regulatorias, la complejidad del dominio, la distribución organizacional, la complejidad técnica- organizacional y la disciplina de la empresa. Y entre ellos se considera de factor crítico a la *distribución organizacional*, que está asociada a la modalidad del software, y dentro de la cual se puede distinguir diversos tipos de desarrollo: en áreas internas de la organización, a la medida por terceros, y de soluciones genéricas estandarizadas para un dominio particular. En esquemas de desarrollo a la medida por terceros se crea una relación comercial, técnica y política entre el proveedor del software y el cliente, tan compleja de gestionar que, bien manejada, puede representar una alianza estratégica de beneficio mutuo que repercute en el crecimiento del sector, y que mal manejada puede representar el fracaso de la empresa proveedora o la pérdida de ventaja competitiva del cliente.

Actualmente, el Global Software Development (GSD) es otro de los retos de la globalización para el desarrollo de software (Herbsleb, 2007), porque en este esquema de trabajo los equipos se encuentran geográficamente distribuidos y es necesario implementar mecanismos de coordinación que garanticen la integridad del producto en construcción y la integración del equipo de trabajo en sus diferentes roles.

2. LA FORMACIÓN DEL INGENIERO DE SOFTWARE

El incremento de la importancia y la complejidad del software, combinado con el del conocimiento acerca de cómo *construirlo,* ha generado la necesidad de actualización permanente para algunos profesionales que, como los ingenieros, han recibido una formación centrada en cómo diseñar y fabricar productos fiables pero poco especializada en el diseño, la construcción, las pruebas y el mantenimiento de productos software. Las características de este contexto no facilitan el logro de productos de buena calidad, porque ya no es suficiente con recibir algunos de los cursos de desarrollo que se imparten en los programas de ingeniería tradicional o en los de Ciencias Computacionales (Mitchell, 2004).

A nivel internacional existen estudios realizados por ACM, IEEE, MIT, Carnegie Mellon University, y otros organismos privados y estatales, en los que se propone que la Ingeniería de Software se debería ofrecer como un programa independiente. Sin embargo, en el contexto latinoamericano el enfoque actual de la formación en esta ingeniería no es satisfactorio. Mientras que muchos consumidores denuncian la baja calidad de los productos software, las empresas desarrolladoras sufren por la escasez de personal cualificado (Thompson, 2001). Además, los mismos empresarios no tienen claro lo que un graduado en ingeniería, relacionada con el software, pueda o no conocer acerca del desarrollo (Serna, 2011).

En escenarios como las ofertas de empleo se utiliza el término *Ingeniero de Software* como un eufemismo para *programador*. La mayoría parece asumir que la única responsabilidad de estos ingenieros es escribir código. Estos supuestos reflejan desconocimiento acerca del significado histórico y jurídico del ser *Ingeniero*, es decir, un profesional responsable de construir productos aptos para su consumo y que, para lograrlo, debe comprender ampliamente el entorno en el que cada producto será utilizado. Consecuentemente, los ingenieros de software necesitan conocer y adquirir conocimientos que no hacen parte de la formación que se imparte en la academia, porque el software no se utiliza aisladamente de otros productos de ingeniería, hace parte de sistemas con componentes físicos, donde computa información de los mismos. Entre los objetivos de los programas en esta área se debería tener en cuenta: 1) desarrollar la *capacidad lógico-interpretativa y abstractiva* de los estudiantes, para que comprendan y modelen los problemas antes de presentarles una solución, 2) especializarlos en el diseño de software fiable y de calidad y 3) brindarles el conocimiento suficiente acerca de otras áreas relacionadas, porque las deben conocer para solicitarles ayuda a otros profesionales, y para conformar y trabajar en equipos armónicos (McConnell & Tripp, 1999). En términos generales, los ingenieros de software se deben formar en:

1. Lo que es cierto y útil en la especialidad elegida, es decir, su cuerpo de conocimientos

2. Cómo aplicar ese conjunto de conocimientos

3. Cómo adquirir y utilizar conocimientos necesarios desde otras áreas para construir productos completos, que deben funcionar en entornos reales

4. El diseño y el análisis disciplinar, que deben seguir para cumplir con las responsabilidades que adquieren quienes construyen productos para otros.

Es decir, los ingenieros se forman en ciencia, además de en los métodos ingenieriles necesarios para aplicarla. Este proceso comienza con una aceptación de las tareas que los profesionales son capaces de realizar (Serna, 2011a):

- Comprensión del entorno en el que hay necesidad de cerrar una brecha o aprovechar una oportunidad de negocio, y que requiere una solución basada en software

- Identificación de los requisitos de la solución y de las tecnologías de apoyo

- Diseño de los componentes de la solución, para determinar cuáles funciones se implementarán en el hardware y cuáles en el software, y para seleccionar los componentes básicos

- Análisis del desempeño del diseño propuesto, ya sea analíticamente o por simulación, para asegurar que el sistema cumple con los requisitos y necesidades establecidos

- Diseño de la estructura básica del software, para dividirlo en módulos y para diseñar las interfaces entre ellos y la estructura de los programas individuales, mientras documenta con precisión todas las decisiones

- Integración del nuevo software al sistema existente

- Realización de pruebas integradas y paralelas al proceso de desarrollo

- Análisis de la estructura del software, en cuanto a completitud, consistencia e idoneidad, para la solución propuesta

- Implementación del producto, como un conjunto de programas bien estructurados y documentados

- Revisión, mejoramiento y mantenimiento de los productos software, manteniendo su integridad conceptual y preservando los documentos completos y precisos.

Además, estos ingenieros son responsables por la usabilidad, seguridad y fiabilidad de sus productos, y deben ser capaces de aplicar matemática, lógica y ciencia para asegurar que el sistema que diseñan realice sus funciones cuando se entregue al cliente. En esta circunstancia, surge entonces el interrogante de si en América Latina se tienen procesos y principios estructurados para formar de esta manera a los futuros ingenieros de software, pero lo más importante, si se están aplicando en las instituciones de formación. En este sentido, la recomendación que la comunidad le hace a la academia y a la industria, relacionadas con la Ingeniería de Software, hace referencia a que tengan en cuenta, en el diseño de esfuerzos, planes de estudio y programas de formación, la experiencia de los profesionales que realizan el trabajo práctico de desarrollo.

Por otro lado, también existen entidades que otorgan licencias, universidades que diseñan planes de estudio, empresas centradas en mejorar la formación de su personal. Además, IEEE está llevando a cabo esfuerzos, como SWEBOK, para definir el cuerpo de conocimientos que los profesionales de software deben poseer. Pero, mientras que la mayoría de estos grupos basan sus decisiones acerca del plan de estudios de la Ingeniería de Software en las opiniones de *expertos* en el área, deberían estar más interesados en escuchar lo que los profesionales en la materia tienen que decir de lo que es vivir y desarrollar la profesión. Interesado en aportar en este tema, Lethbridge (2000) realizó una investigación en la que entrevistó a 186 desarrolladores profesionales, y les planteó interrogantes acerca de 75 temas relacionados con la Ingeniería de Software. Los resultados de este trabajo demuestran que es necesario entablar un diálogo entre la academia, la industria y estos profesionales, porque al parecer los objetivos formativos no logran satisfacer las necesidades de la vida real. De la revisión a los trabajos en esta temática se concluye que debido a que las instituciones que forman en Ingeniería de Software son responsables de *producir* profesionales que van a diseñar, construir y mantener sistemas para la sociedad, entonces deberían ampliar sus horizontes para atender de mejor forma como honrar esta responsabilidad.

El objetivo de las instituciones de educación superior debe ser el de garantizar que la formación en ingeniería sea eficaz, y de que se lleve a cabo dentro del contexto de un examen exhaustivo a todos los aspectos relevantes del sistema académico-industrial, interrelacionado los sistemas formativos con las prácticas y el sistema económico global. La formación en Ingeniería de software se debe reajustar para promover el logro de las características deseadas en la práctica profesional del desarrollo de software, y esto se debe hacer en el contexto de un mayor énfasis en la base de investigación de la conducta subyacente a la práctica de la ingeniería en general. Para ello será necesario que las partes interesadas ejecuten roles protagónicos, en particular las facultades y las asociaciones profesionales de ingeniería. Además, la academia debería fomentar y ejemplificar una ética profesional, e incentivar a modificar los estados corruptos al interior de los proyectos software —ya sea en empresas, organismos estatales, educativos u otros contextos—, con el objetivo de aportar a remediar un flagelo que lastima a la profesión de la Ingeniería en general, y que inclusive perjudica a la sociedad general.

2.1 Competencias del Ingeniero de Software

Históricamente, la forma más común para desarrollar competencias han sido los procesos formativos, pero no fue sino hasta 1916 cuando más del 50% de los ingenieros profesionales lograron algún tipo de título universitario (Ford & Norman, 1996). Actualmente, gran parte del desarrollo de competencias se adquiere durante la formación profesional inicial, y si bien el objetivo de los procesos formativos es impartir los conocimientos necesarios para el ejercicio de una profesión, cada curso tiende a introducir el desarrollo de habilidades a través del trabajo de laboratorios, proyectos de aula y competiciones internas (Ford & Norman, 1996; Mengel, 1998). Cal Poly hace hincapié en la filosofía de *aprender haciendo* (Bagert, 1998), donde los estudiantes se forman principalmente a través de laboratorios y otras actividades prácticas, que desarrollan competencias generalmente en periodos diversos de tiempo, y en todo caso el objetivo es lograrlo antes de finalizar la formación profesional inicial. Estos procesos les permiten a los estudiantes y jóvenes graduados desarrollar habilidades en diversos entornos, antes de adentrarse plenamente en la práctica profesional (Lethbridge, 2000).

Aunque se espera que las competencias se adquieran con anterioridad a la práctica profesional, la mayoría se desarrolla durante el ejercicio profesional en el lugar de trabajo (Ford & Norman, 1996). Los ingenieros de software recién graduados pasan algún tiempo como ingenieros en entrenamiento, antes de adquirir la experiencia o la certificación que los acredite como ingenieros profesionales (NSPE). Sólo entonces comienza su desarrollo profesional, que incluye todas las actividades destinadas a mejorar o mantener actualizados los conocimientos y las competencias de su profesión, luego de lograr el título universitario (Ford & Norman, 1996). Este proceso involucra actividades diversas, como la lectura de revistas profesionales y tomar parte de diversos programas de formación y de entrenamiento continuo y la asistencia a eventos científicos relacionados. Dado que el desarrollo profesional de software cubre una amplia cantidad de posibilidades y alternativas, no es fácil encontrar ejemplos coherentes comunes en las profesiones relacionadas (Ford y& Norman, 1996). Sin embargo, se pueden describir dos patrones generales:

1. El desarrollo profesional es más importante en profesiones que tienen un cuerpo de conocimiento tecnológico en rápida evolución, sobre el que se basa su práctica como profesión (Ford & Norman, 1996). Por ejemplo, la Ingeniería de Software se nutre constantemente de nuevos conocimientos sobre las bases de las diferentes áreas profesionales de su desempeño, de las necesidades sociales y del desarrollo tecnológico y científico (Ford & Norman, 1996), lo que supone una demanda constante por ingenieros de software con actualización permanente en conocimientos y competencias.

2. El desarrollo profesional tiende a centrarse en las actividades con pequeñas ganancias a corto plazo para proyectos concretos, en lugar de un desarrollo a largo plazo de la profesión (Ford & Norman, 1996). En el caso de la Ingeniería de Software es más común que los profesionales tomen cursos o entrenamientos cortos sobre una herramienta o técnica específica, que luego podrán utilizar para sus funciones laborales inmediatas, pero están poco interesados en tomar cursos largos en los avances científicos relacionados con su área (Ford & Norman, 1996).

Las competencias son habilidades, técnicas y atributos de rendimiento en el trabajo, que se asocian con la actualización permanente del conocimiento de los profesionales, y quizás sea el factor más importante para determinar el éxito del desarrollo de su trabajo, por lo que es necesario identificar las competencias profesionales esenciales para ellos. El proyecto Tuning Educational

Structures define competencia como *una combinación dinámica de atributos, en relación a procedimientos, habilidades, actitudes y responsabilidades, que describen los encargados del aprendizaje de un programa educativo, o lo que los estudiantes son capaces de demostrar al final de un proceso formativo* (Beneitone et al., 2007.). El mismo proyecto define dos tipos de competencias: 1) *genéricas,* que en principio son independientes del área de estudio y 2) *específicas,* para cada área temática. Las competencias se adquieren normalmente durante diferentes unidades de estudio, y por tanto pueden no estar ligadas a una sola unidad. En la Tabla 1 se detallan las competencias que debería demostrar un ingeniero de software, que son avaladas por diversas instituciones en el mundo.

Tabla 1. Competencias del ingeniero de software

Capacidad para trabajar en equipo
Capacidad para actualizar permanente su conocimiento
Habilidad para trabajar en contextos internacionales
Capacidad para resolver problemas de forma sistémica y sistemática
Habilidad para experimentar con nuevos métodos y herramientas
Habilidad para procurar el mejoramiento continuo de su trabajo y productos
Habilidad para planificar, combinar y adaptar
Capacidad de comunicación en un segundo idioma
Responsabilidad social y compromiso ciudadano
Capacidad para identificar, modelar y resolver problemas
Habilidades para buscar constantemente el mejoramiento de la calidad
Poseer sólidas habilidades analíticas
Capacidad para tomar decisiones
Habilidad para elaborar y utilizar prototipos
Capacidad para aplicar teorías, modelos y técnicas
Habilidad para planificar, combinar y adaptar
Capacidades para identificar, evaluar e implementar tecnologías
Capacidad para responder adecuadamente bajo presión
Capacidad para formular y gestionar proyectos
Habilidades para buscar, procesar y analizar información
Habilidades para la comunicación verbal y escrita
Capacidad para seleccionar especificaciones y crear modelos abstractos
Habilidades para dirigir y liderar recurso humano
Capacidad para diseñar y dirigir experimentos y para analizar e interpretar datos
Poseer disciplina, terquedad, compulsión, dedicación y voluntad de trabajo
Ser proactivo y tener iniciativa
Presentar y defender ideas creativas e innovadoras
Poseer habilidades de liderazgo, persuasión y de asesoría

2.2 Roles del Ingeniero de Software

Los ingenieros de software se especializan en el diseño y el desarrollo de aplicaciones software, y están formados principalmente en matemáticas y Ciencias Computacionales. Los conocimientos y habilidades adquiridas a partir de esta formación los aplican en el diseño, construcción y despliegue de estas aplicaciones. Los sistemas de información se han convertido en un componente muy importante de las organizaciones, y las aplicaciones que estos profesionales diseñan las utilizan una amplia gama de usuarios. También necesitan formación adecuada en hardware, además de los conocimientos teóricos en software, para poder hacer frente a los complejos problemas de aplicación. De acuerdo con diversas fuentes que investigan y difunden acerca de esta cuestión, en su desempeño profesional un ingeniero de software puede llevar a cabo los roles de la Tabla 2.

Tabla 2. Roles del ingeniero de software

Rol	Definición	Tareas
Ingeniero/Analista de requisitos (IEEE, 2004; SIT, 2009; ACM & IEEE, 2004; Wang, 2008)	Se encarga de aplicar la ingeniería de requisitos con el objetivo de descubrir, analizar y documentar el propósito del producto, mediante la identificación de las necesidades de las partes interesadas.	Elicitar, modelar, analizar, especificar, validar, gestionar, documentar, comunicar, integrar.
Arquitecto de software (IEEE, 2004; SIT, 2009; ACM & IEEE, 2004; Bredemeyer & Malan, 2006)	Razonar sobre si el producto cumple con sus requisitos en un contexto determinado, y sobre sus limitaciones y mejoras; debe encontrar el balance entre el contexto de las fuerzas y las limitaciones que moldean la solución.	Abstraer, visionar, conceptualizar, experimentar, crear, innovar, investigar, especificar, validar, documentar, tomar decisiones.
Desarrollador (Fuller, 2003; Acuña & Juristo, 2004; Laporte et al., 2007; IEEE, 2004)	Debe interpretar una especificación de diseño para implementar una solución, utilizando una arquitectura de referencia establecida y utilizando adecuadamente las librerías de un lenguaje o las funcionalidades provistas en un *framework*.	Investigar, desarrollar prototipos, modificar, reutilizar y mantener código, re-ingeniería, probar, integrar, prototipar, modificar, reutilizar y mantener código.
Analista de calidad (IEEE, 2004; SIT, 2009; ACM & IEEE, 2004; Wang, 2008)	Es el responsable de aplicar los principios y prácticas de aseguramiento de la calidad del software durante todo el ciclo de vida. Sus funciones están orientadas a asegurar la calidad del proceso de desarrollo del producto.	Gestión de QMS, aprobar los documentos, realizar auditorías de calidad, mantener y actualizar bases de datos de entrenamiento y auditoría, identificar problemas o deficiencias en los productos, resolver los problemas de QMS.
Ingeniero de pruebas (Vliet, 2006; Acuña & Juristo, 2004; Wang, 2008)	Responsable de realizar investigación y experimentación para proporcionarles a los interesados información acerca de la calidad del producto. Proporciona una visión objetiva e independiente del software para permitirle a la empresa apreciar y comprender los riesgos del producto antes de ponerlo en producción. Asume el desafío de detectar la mayor cantidad de fallas con el mínimo esfuerzo, y participa de todas las etapas del proceso de desarrollo, colaborando para asegurar la máxima calidad del producto.	Planificar, diseñar, ejecutar y administrar pruebas, conocer el negocio y su modelo, proyectar, analizar.
Gerente de proyectos (Acuña & Juristo, 2004; Wang, 2008)	Encargado de llevar a cabo la aplicación de las actividades de gestión para garantizar que el desarrollo y el mantenimiento de software sean sistemáticos, disciplinados y cuantificados.	Planear, coordinar, medir, monitorear, controlar, presentar informes.
Diseñador (IEEE, 2004; SIT, 2009; ACM & IEEE, 2004; Wang, 2008)	Debe realizar el proceso de definición de la arquitectura, los componentes, las interfaces y otras características de los productos. Describe cómo se descompone el software y cómo se organizan en componentes. Se refiere a la identificación de los principales componentes hardware y software de un producto, que proporcionan las características y atributos de calidad del mismo.	Abstraer, acoplar, cohesionar, descomponer, modularizar, encapsular, diseñar, integrar, verificar.
Analista de Sistemas (Wang, 2008; Kendall & Kandall, 2014)	Los analistas de sistemas utilizan la metodología matemática para obtener los detalles de los sistemas que analizan.	Analizar sistemas y sus interacciones resultantes. Las organizaciones los requieren para que mejoren sus sistemas e incrementen su eficiencia.

2.3 Prácticas del Ingeniero de Software

Las prácticas son formas de trabajo aceptadas por una comunidad y reconocidas como válidas, y se pueden entender como declaraciones de alto nivel que establecen guías generales, o como formas de trabajo que marcan ciertos *estilos*, de la misma forma que los grupos de trabajo realizan sus actividades. Desde la perspectiva de los modelos de calidad, como CMMI, las prácticas establecen guías generales —*qué*— sin entrar a definir estilos de trabajo —*cómo*—. Se podría afirmar que las prácticas de alto nivel son las más cercanas las competencias en la realización de tareas —competencias procedimentales—, las cuales se espera que realicen los ingenieros de software, mientras que las prácticas específicas definen tendencias de trabajo y están estrechamente asociadas al contexto en los cuales se aplican y donde marcan una tendencia. En la Tabla 3 se presenta un listado de prácticas que pueden realizar los ingenieros de software en la industria, y que sirven para identificar tendencias o estilos de trabajo en el sector, que respalda y comunica la misma industria.

Tabla 3. Prácticas del ingeniero de software

Documentar la toma de decisiones para la solución
Aplicar las diferentes etapas de la ingeniería de requisitos
Diseñar soluciones con base en los requisitos
Codificar soluciones siguiendo los estándares y especificaciones de la arquitectura
Verificar y Validar el cumplimiento de los requisitos en la alternativa de solución
Utilizar prototipos para validar que la solución propuesta cumple los requisitos
Utilizar esquemas gráficos para comprender y explicar requisitos
Documentar las soluciones
Utilizar herramientas unificadas para todos los procesos
Mantener un repositorio unificado para conservar y garantizar la integridad de la solución
Comprender, moldear y representar los problemas
Generar código a partir de modelos
Proponer diferentes alternativas de solución
Representar los componentes estructurales y dinámicos de la solución
Utilizar patrones de diseño
Interactuar con el equipo a través de herramientas para trabajo colaborativo
Diseñar y aplicar planes de pruebas
Diseñar, aplicar y validar casos de prueba
Verificar y Validar software
Gestionar cambios a través de interacción directa
Gestionar riesgos
Reutilizar componentes
Gestionar y administrar la calidad
Realizar capacitación y entrenamiento permanente
Diseñar, aplicar y divulgar mejoras al proceso
Gestionar los cambios
Realizar instructivos que especifiquen y guíen el despliegue de la solución
Hacer control de seguimiento
Asignar roles como líder de proyectos
Diseñar e implementar mecanismos eficientes de comunicación

MANIFIESTO

POR LA PROFESIONALIZACIÓN DEL DESARROLLO DE SOFTWARE

Debido a que

EL SOFTWARE ES UN PRODUCTO DEL INTELECTO HUMANO, QUE PARA SU PRODUCCIÓN REQUIERE INGENIERÍA, CIENCIA, GESTIÓN, IMAGINACIÓN Y ARTE

LA COMPLEJIDAD DEL SOFTWARE, DEL PROCESO PARA SU DESARROLLO Y DE LAS NUEVAS Y VARIADAS NECESIDADES DE LA SOCIEDAD RESPECTO A SUS PRODUCTOS SE INCREMENTA CONSTANTEMENTE

EL SOFTWARE TIENE CADA DÍA MAYOR PROTAGONISMO EN LAS ACTIVIDADES COTIDIANAS DE LAS PERSONAS, Y GRAN PARTE DE ELLAS DEPENDE DE SU CORRECTO FUNCIONAMIENTO

EL DESARROLLO DE SOFTWARE ES UNO DE LOS SECTORES ESTRATÉGICOS DE LA ECONOMÍA QUE LOS ESTADOS PROMUEVEN Y APOYAN

LA INDUSTRIA NECESITA PROFESIONALES ALTAMENTE CAPACITADOS PARA RESPONDER A LAS NECESIDADES DE LA SOCIEDAD DE LA INFORMACIÓN Y EL CONOCIMIENTO

EL INGENIERO DE SOFTWARE DEBE CONTAR CON CONOCIMIENTOS Y COMPETENCIAS, GENÉRICOS Y ESPECÍFICOS, QUE LE PERMITAN DESARROLLAR SOLUCIONES DE CALIDAD E INTEGRADAS EN CONTEXTO

Por lo tanto

SE NECESITA PROFESIONALIZAR EL DESARROLLO DE SOFTWARE

Estableciendo claramente las responsabilidades, capacidades, conocimientos, habilidades y formación de los ingenieros de software

REFERENCIAS

- ACM & IEEE. 2004. Software Engineering 2004. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. National Science Foundation.
- ACM/IEEE. (2005). Computing Curricula. The Overview Report covering undergraduate degree programs in Computer Engineering, Computer Science, Information Systems, Information Technology, and Software Engineering. The Joint Task Force for Computing Curricula. http://www.computer.org/portal/c/document_library/get_file?p_l_id=2814020&folderId=3111026&name=DLFE-57601.pdf.
- Acuña, S. T. & Juristo, N. 2004. Assigning people to roles in software projects. Software-Practice and experience, 34(7), 675-696.
- Agile Alliance. 2001. The agile alliance. Online [Jul. 2012].
- Agresti, W. 1981. Software Engineering as Industrial Engineering. ACM SIGSOFT Software Engineering Notes, 6(5), 11-13.
- Ambler, S. 2009. The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments. Online [Jul. 2012].
- Ambler, S. 2010. Positions Paper. Proceedings of the Semat Zurich Workshop. Online [Jul. 2012].
- Anaya, R. 2006. Una visión de la enseñanza de la Ingeniería de Software como apoyo al mejoramiento de las empresas de software. Revista Universidad EAFIT, 42(141), 60-76.
- Bagert, D. 1998. The Challenge of Curriculum Modeling for an Emerging Discipline: Software Engineering. Proceedings 28th Annual Frontiers in Education Conference FIE '98, 910-915.
- Bajwa I. S. et al. 2008. SOA and BPM Partnership: A paradigm for Dynamic and Flexible Process and I.T. Management. Proceedings of World Academy of Science, Engineering and Technology, 35, 16-22
- Beneitone, P. et al., 2007. Reflexiones y perspectivas de la educación superior en América Latina. Proyecto Tuning. Spain/impreso.
- Bohem, B. 2011. Some Future Software Engineering Opportunities and Challenges. Online [May 2012].
- Bourque, P. & Dupuis, R. (2001). Guide to the Software Engineering Body of Knowledge (SWEBOK). IEEE CS Press, Los Alamitos, CA. (http://www.computer.org/portal/web/swebok).
- Bredemeyer, D. & Malan, R. 2006. The Role of the Architect. White paper. Bredemeyer Consulting. Online [Feb. 2012].
- Brooks, F. 1986. No Silver Bullet - Essence and Accident in Software Engineering. Proceedings of the IFIP Tenth World Computing Conference, 1069-1076.
- Bruegge, B., Dutoit, A. 2002. Ingeniería de software orientado a objetos. Prentice Hall.
- Cater-Steel B. & Bus, A. 2004. An Evaluation of Software Development Practice and Assessment-Based Process Improvement in Small Software Development Firms. PhD Thesis. School of Computing and Information Technology Faculty of Engineering and Information Technology, Griffith University.
- Clarke, S. & Baniassad, E. 2004. Aspect-Oriented analysis and design: The theme approach. Object Technology Series. Addison-Wesley
- CMM-CMM-I. Online: http://en.wikipedia.org/wiki/Capability_Maturity_Model [Oct. 2013].
- Cook, D. A. & Dupaix, L. 1999. A Gentle Introduction to Software Engineering. Software Technology Support Center.

- Daniels, J. & Cheesman, J. 2001. UML Components: A Simple Process For Specifying Component Based Software. Addison-Wesley Professional.
- Díaz-Herrera, J. L. (2009). The 'engineering' of software, a different kind of engineering. ACM SIGSOFT Software Engineering Notes, 34(5), 1.
- Díaz-Herrera, J.L. & Hilburn, T.B. (2004). Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. http://www.computer.org/portal/c/document_library/get_file?p_l_id=2814020&folderId=3111026&name=DLFE-57602.pdf.
- Ford, G. & Norman, G. 1996. A Mature Profession of Software Engineering. Technical Report. CMU/SEI-96-TR-004, ESC-TR-96-004. Carnegie Mellon University.
- Fugetta, A. 2000. Software Process: A Roadmap. In Finkelstein, A. (Ed.), The Future of Software Engineering. ACM Press.
- Fuller, P. D. 2003. Roles en el desarrollo de software. Apuntes de Taller de Ingeniería de Software. Online [Feb. 2012].
- Glass, R. L. 2006. Facts and Fallacies of Software Engineering. Addison-Wesley.
- Herbsleb, J. D. 2007. Global Software Engineering: The Future of Socio-technical Coordination. Proceedings Future of Software Engineering FOSE'07, 188-198.
- Humphrey, W. (1995). A Discipline for Software Engineering. Addison-Wesley.
- IEEE & ACM. 2004. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. National Science Foundation.
- IEEE. 1990. Standard Computer Dictionary.
- IEEE. 2004. Guide to the Software Engineering. Body of Knowledge SWEBOK®. Computer Society.
- IEEE. Certified Software Development Professional (CSDP). Online: http://www.computer.org/portal/web/certification/csdp [Sep. 2013].
- Intelligent Tings. 2009. The New Software Engineering Manifesto. http://intelligentthings.com/content/new-software-engineering-manifesto, [April 2013].
- Jacobson, I.; Wei, P. & Spence, I. 2007. Enough of Processes - Lets do Practices. Journal of Object Technology, 6(6), 41-66.
- Jorgenson, D. W. & Wessner, C. W. 2006. Measuring and Sustaining the New Economy, Software, Growth, and the Future of the U.S. Economy. National Academies Press.
- Kendall, K.E. & Kendall, J.E. 2014. Systems Analysis and Design. Prentice Hall.
- Kleppe, A.; Warner, J. & Bast, W. 2004. MDA Explained. The model driven architecture: practice and promise. Addison Wesley
- Laporte, C. Y. et al. 2007. Utilization of a Set of Software Engineering Roles for a Multinational Organization. Münch, J. & Abrahamsson, P. (Eds.), PROFES 2007, LNCS 4589, 35-50. Springer-Verlag.
- Lethbridge, T. C. 2000. What knowledge is important to a software professional? Computer, 33(5), 44-50.
- Maibaum, T. S. E. 1997. What We Teach Software Engineers in the University: Do We Take Engineering Seriously? ACM SIGSOFT Software Engineering Notes, 22(6), 40-50.
- McConnell, S. & Tripp, L. 1999. Professional Software Engineering: Fact or Fiction? IEEE Software, 16(6), 13-18.
- McConnell, S. 2003. Professional Software Development: Shorter Schedules, Higher Quality Products, More Successful Projects, Enhanced Careers. Addison Wesley.

- Mengel, S. A. 1998. Guidelines Proposal for Undergraduate Software Engineering Education. Proceedings 28th Annual Frontiers in Education Conference FIE '98, 916-919.
- Meyer, B. 1998. Construcción de Software Orientado a Objetos. Prentice Hall.
- Mirbel, I. & Ralyté, J. 2006. Situational Method Engineering: Combining Assembly-based and Roadmap-driven Approaches. Requirements Engineering 11(1), 58-78 (2006).
- Mitchell, W. 2004. Is software engineering for everyone? Proceedings of second annual conference on Mid-south College computing, 53-64.
- Navarro, J. M. & Garzás, J. 2010. Experiencia en la implantación de CMMI-DEV v1.2 en una micro-pyme con metodologías ágiles y software libre. REICIS Revista Española de Innovación, Calidad, 6(1), 6-15.
- NIST. 2010. Software Testing Metrics Portal. Online [Jul. 2012].
- NSPE. National Society of Professional Engineers. Continuing Professional Competency. Online [Jun. 2012].
- Papazoglou, M. & Heuvel, W. 2006. Service-Oriented Design and Development Methodology. Int. J. of Web Engineering and Technology, 2(4), 412-442.
- Pressman, R. 2005. Software Engineering: A Practitioner's Approach. McGraw-Hill Science.
- Ruiz, F. 2007. Software process engineering. De una gestión de procesos Contemplativa a una Productiva. Online [Jul. 2012].
- SEI. 1990. SEI Report on Undergraduate Software Engineering Education. Online [Jul. 2012].
- SEMAT. Software Engineering Method and Theory. http://semat.org/ [Jul. 2012].
- Serna, M. E. 2011. Systems engineering for the XXI century: A proposal from the academy. Proceedings Ninth LACCEI Latin American and Caribbean Conference (LACCEI'2011), Paper 5. August 3-5, Medellin, Colombia.
- Serna, M. E. 2011a. 'Software Engineering´ is Engineering. RACCIS, 1(1), 34-43.
- SIT - Stevens Institute of Technology. 2009. Graduate Software Engineering 2009(GSwE2009). Integrated Software & Systems Engineering Curriculum (iSSEc) Project.
- Sommerville, I. (1999). Systems engineering for software engineers. Annals of Software Engineering, 1-4, 111-129.
- Sommerville, I. 2006. Software Engineering. Addison-Wesley.
- Sommerville, I.; Cliff, D.; Calinescu, R.; Keen, J.; Kelly, T.; Kwiatkowska, M.; Mcdermid, J. & Paige, R. (2012). Large-scale complex IT systems. CACM, 55 (7), 71-77.
- Szyperski, C. 1998. Component software – beyond Object Oriented Programming. Addison Wesley.
- Thompson, J. B. 2001. A Long and Winding Road (Progress on the Road to a Software Engineering Profession). Proceedings of 25th International Computer Software and Applications Conference on Invigorating Software Development, 39-45.
- Tucker, A.; González, T.; Díaz-Herrera, J.L. & Topi, H. (2014). The Computing Handbook Set. In press.
- Vaughn, R. 2000. Software Engineering Degree Programs. CROSSTALK, 13(3), 7-9.
- Vliet van, H. 2006. Reflections on software engineering education. IEEE Software, 23(3), 55-61.
- Wang, Y. 2008. Software Engineering Foundations - A software science perspective. Auerbach Publications.

Copyright © 2013, Instituto Antioqueño de Investigación IAI™

SI TRABAJAMOS JUNTOS, LO LOGRAREMOS

Uno de los problemas centrales del desarrollo económico actual y de la competitividad industrial, social y científica, es la *complejidad* de los grandes e intensivos sistemas software, y de los procesos para su desarrollo y aplicación. Esta complejidad se define por la cantidad y heterogeneidad de la interacción del hardware con los componentes software, de sus inter-relaciones, de la incorporación en los entornos técnicos y organizacionales, y de las interfaces para los seres humanos. El dominio de estos sistemas requiere de acciones y pensamientos científicos, jerárquicos y sistemáticos; además, el éxito de los productos, los servicios y las organizaciones, está cada vez más determinado por la disponibilidad de productos software adecuados. Por lo tanto, se necesitan profesionales altamente cualificados, capaces de comprender y dominar los sistemas, de participar en todo el ciclo de vida de la Ingeniería de Software, y de adoptar diferentes roles durante el desarrollo. Esta es la razón que guía el pensamiento de este Manifiesto, cuyo objetivo es lograr la

Profesionalización del Desarrollo de Software

Red Latinoamericana en Ingeniería de Software (RedLatinaIS)
Instituto Antioqueño de Investigación (IAI)
©2013