Ingeniería de Software en México:
Educación, Industria e Investigación

Raúl Antonio Aguilar Vera

Editor

ACADEMIA MEXICANA DE COMPUTACIÓN, A.C.

Ingeniería de Software en México: Educación, Industria e Investigación

Editor: Raúl Antonio Aguilar Vera.

En colaboración con la Academia Mexicana de Computación

Coordinador: Luis Enrique Sucar Succar.

Segunda edición 2019

Academia Mexicana de Computación, A.C.

ISBN: 978-607-97357-8-4

Corrección de estilo: Raúl Antonio Aguilar Vera.

Diseño de portada: Mario Alberto Vélez Sánchez.

Cuidado de la edición: Luis Enrique Sucar Succar.

Este libro se realizó con el apoyo del CONACyT, Proyecto I1200/28/2019.

Impreso en México

Printed in Mexico

Ingeniería de Software en México:
Educación, Industria e Investigación

Autores:
Jorge Rafael Aguilar Cisneros,
Raúl Antonio Aguilar Vera,
María Karen Cortés Verdín,
María de León Sigg,
Julio César Díaz Mendoza,
Carlos Alberto Fernández y Fernández,
Brenda Leticia Flores Ríos,
Omar Salvador Gómez Gómez,
María Guadalupe Elena Ibargüengoitia González,
Reyes Juárez Ramírez,
Jezreel Mejía Miranda,
Enzo Molino Ravetto,
Mirna Adriana Muñoz Mata,
Jorge Octavio Ocharán Hernández,
Hanna Jadwiga Oktaba,
Oscar Mario Rodríguez Elías,
Juan Pablo Ucán Pech,
Francisco Valdés Souto,
Sodel Vázquez Reyes
Carlos Antonio Zozaya Gorostiza.

La Academia Mexicana de Computación, A.C. agradece al CONACyT su apoyo para la creación de esta obra.

Prólogo

Transcurridas las primeras cinco décadas de la concepción de una nueva disciplina Ingenieril, la cual, a partir de la denominada "Crisis del Software" adoptó como propósito, el desarrollo software de calidad, nos encontramos hoy día con una profesión reconocida internacionalmente, que cuenta con un amplio consenso en cuanto a su cuerpo de conocimientos, y que para avanzar en el grado de madurez en su quehacer cotidiano, recurre a la Investigación, particularmente la empírica, para generar nuevos métodos, técnicas, herramientas y buenas prácticas que abonen a dicho cuerpo de conocimientos.

La presente obra surge de la preocupación de la Academia Mexicana de Computación, A.C. (AMEXCOMP), a través de su Sección Académica de Ingeniería de Software, por difundir el desarrollo en Investigación, en la Industria del Software, así como en el grado de consolidación de los Programas Educativos creados exprofeso para la Formación de Recursos Humanos en esta nueva disciplina, lo anterior, en el contexto del desarrollo de México.

El libro, en su segunda edición, incorpora un primer capítulo introductorio en el cual se describe brevemente el origen de la disciplina

—naciente para finales de la década de los 60´s— así como las áreas de desarrollo y gestión que conforman el cuerpo de conocimientos reconocido para la misma. A medio siglo de su concepción, se identifican nuevos retos y tendencias que son discutidos el final del capítulo. Los ocho capítulos de la primera edición, fueron revisados por un grupo de coordinadores de sección de la AMEXCOMP, y actualizados por sus autores, a efecto de mantener la pertinencia del contenido de los mismos en esta obra.

El capítulo dos presenta un par de estudios en los que se analiza el estado guarda la Educación Superior en México —en el ámbito de la Ingeniería de Software— en cuanto a programas reconocidos por su calidad. El tercer capítulo presenta una visión del estatus de la Industria del Software en México, y para ello describe de manera cronológica las principales iniciativas promovidas para su desarrollo.

En cuanto al ámbito de la Investigación desarrollada en México, los capítulos cuatro, cinco, seis, siete y ocho, abordan las temáticas de Requisitos de Software, Diseño de Software, Mejora de Procesos Software, Métodos y Modelos para la Ingeniería de Software, así como la de Métricas de Software; si bien dichas temáticas no engloban a todas las vinculadas con el cuerpo de conocimientos reconocido para la disciplina, con la información plasmada por los autores de dichos capítulo, es posible identificar el interés particular en dichas áreas, por parte de los grupos de investigación en nuestro país.

Finalmente, una de las estrategias más recurridas para abonar al cuerpo de conocimientos de la disciplina, ha sido el uso de métodos empíricos para investigación, particularmente, el desarrollo de experimentos controlados que permiten a los investigadores, confirmar o rechazar hipótesis de investigación en torno a las relaciones que pueden guardar los diferentes factores inmersos en el proceso de desarrollo de software; el capítulo nueve presenta un panorama general del paradigma experimental, así como las áreas de conocimiento en las que éste ha sido utilizado en nuestro país.

Resulta necesario reconocer que el presente texto no hubiese sido posible sin el esfuerzo y dedicación de los autores de cada uno de los capítulos, los cuales, a pesar de las múltiples actividades que tienen al interior de sus instituciones, han destinado parte de su tiempo en la preparación y redacción de dichos capítulos, por ello, es menester agradecer, a nombre de la AMEXCOMP, su desinteresada colaboración en la presente obra.

Raúl Antonio Aguilar Vera

Abreviaturas

ACM	Asociación de Maquinaria Computacional (*Association for Computing Machinery*)
AMITI	Asociación Mexicana de la Industria de Tecnologías de Información
AMMS	Asociación Mexicana de Métricas de Software
ANIEI	Asociación Nacional de Instituciones de Educación en Tecnologías de la Información
ANOVA	Análisis de Varianza
CA	Cuerpo Académico
CANIETI	Cámara Nacional de la Industria Electrónica, de Telecomunicaciones y Tecnologías de la Información
CMMI	Integración de Sistemas Modelos de Madurez de Capacidades (*Capability Maturity Model Integration*)
COSMIC	Consorcio Internacional de Medición de Software en General (*Common Software Measurement International Consortium*)
FP	Puntos de Función (*Function Points*)
IEEE	Instituto de Ingeniería Eléctrica y Electrónica (*Institute of Electrical and Electronics Engineers*)
IES	Instituciones de Educación Superior
IR	Ingeniería de Requisitos
IS	Ingeniería de Software

LGAC	Líneas de Generación y Aplicación del Conocimiento
LOC	Líneas de Código (*Lines of Code*)
MDA	Arquitectura Dirigida por Modelos (*Model-Driven Architecture*)
MVC	Modelo Vista Controlador
PSP	Proceso Personal de Software (*Personal Software Process*)
RF	Requisitos Funcionales
RNF	Requisitos No Funcionales
SEI	Instituto de Ingeniería de Software (*Software Engineering Institute*)
SOA	Arquitectura Orientada a Servicios (*Service Oriented Architecture*)
SWEBOK	Cuerpo de Conocimientos de la Ingeniería de Software (*Software Engineering Body of Knowledge*)
TI	Tecnologías de la Información
TSP	Proceso de Software en Equipo (*Team Software Process*)
UML	Lenguaje Unificado de Modelado (*Unified Modeling Languaje*)

Índice general

Prólogo…….. v

Abreviaturas…….. viii

Índice General……... x

1. Introducción a la Ingeniería de Software……..…… 1

1.1 Origen de la Disciplina…….……... 1

1.2 Áreas de Conocimiento…….……... 3

1.2.1 Áreas de Desarrollo…… 3

1.2.2 Áreas de Gestión…..…… 5

1.3 Retos de la Disciplina ……... 7

1.3.1 El software desde su estructura interna……..…… 7

1.3.2 La formalización en producto y el proceso…… 8

1.3.3 Adopción de técnicas inteligentes……...……..…… 9

1.4 Tendencias y Paradigmas Emergentes…..……... 11

Referencias……...……...……... 14

2. Formación de Recursos Humanos.....… 19

2.1 La Educación en Ingeniería de Software…….…... 19

2.2 Organismos y Eventos Promotores de la Ingeniería de Software 26

Referencias……...……...……... 30

3. La Industria del Software en México…….……. 31

3.1 Introducción…….. 31

3.2 Programas gubernamentales para el uso de la computación y el fomento a la industria de software……..……... 33

3.3 Impactos de la adopción de modelos de calidad en la industria de software……... 39

3.4 Mapa de rutas para impulsar la industria de software……..……... 41

3.5 Conclusiones……..……... 47

Referencias……...……... 48

4. Investigación en el área de Ingeniería de Requisitos…… 49
4.1 Introducción…….. 49
4.2 Marco Referencial……...…….. 50
4.2.1 Perfil y Responsabilidades……..…… 51
4.2.1 Atributos de calidad de requisitos……...…… 51
4.3 Proceso de Ingeniería de Requisitos.……..…….. 52
4.3.1 Desarrollo de requisitos…….....…… 54
4.3.2 Administración de requisitos……...…...…… 60
4.4 Análisis de resultados……..……...…….. 61
4.5 Conclusiones…….……...……...…….. 66
Referencias……...……...……... 66
5. Investigación en el área de Diseño de Software……...…… 73
5.1 Introducción…….. 73
5.2 Fundamentos de Diseño de Software…...……...….. 77
5.2.1 Principios de Diseño de Software…....…… 77
5.3 Cuestiones clave en el Diseño……...…….…….. 77
5.3.1 Seguridad…….....……..…… 78
5.4 Estructura del Software y Arquitectura....…….…….. 79
5.4.1 Estructuras Arquitectónicas y Puntos de Vista……...… 79
5.4.2 Estilos Arquitectónicos……...…… 81
5.4.3 Decisiones de Diseño Arquitectónico…… 82
5.4.4 Familias de Programas y *Frameworks*…….... 84
5.5 Análisis y Evaluación de la Calidad del Diseño del Software…..... 90
5.6 Notaciones de Diseño de Software……...…… 91
5.7 Métodos y Estrategias de Diseño de Software…….. 93
5.8 Herramientas de Diseño de Software…… 97
5.9 Conclusiones…….....……. 100
Referencias……...……..... 100

6. Investigación en el área de Mejora de Procesos de Software…..…… 111
6.1 Introducción…….. 111
6.2 Modelos y Estándares……...…… 113
6.2.1 CMM-DEV……. 113
6.2.2 MoProSoft...……... 116
6.2.3 ISO/IEC 29110…….. 120
6.3 Personas…… 123
6.3.1 Proceso Personal de Software…….……... 124
6.3.2 Proceso de Software en Equipos…….……... 125
6.3.3 *SCRUM*…….……..…….……... 128
6.4 Herramientas y Equipo…….……...……. 129
6.5 Grupos de Investigación en México…...…….. 131
6.5.1 CENIDET…….…….. 131
6.5.2 CICESE…...…….……... 132
6.5.3 CIMAT…...…….…….. 133
6.5.4 UNAM…….……... 136
6.5.5 UAZ…….……... 136
6.6 Conclusiones…….…...……. 137
Referencias……...……..... 138
7. Investigación en el área de Métodos y Modelos en Ingeniería de Software…… 149
7.1 Introducción…….. 149
7.2 Métodos en la Ingeniería de Software……...…….…….. 150
7.2.1 Métodos Heurísticos…..…….…… 150
7.2.2 Métodos Formales…..…….……... 151
7.2.3 Métodos de Prototipado.…….…… 152
7.2.4 Métodos Ágiles……..…….…… 153
7.2.4 Investigación en el área de Métodos en México…….... 154

7.3 Modelos en la Ingeniería de Software……..…….. 155
7.3.1 Modelo de Contexto…..…… 157
7.3.2 Modelo de Comportamiento…….…….. 157
7.3.3 Modelo de Estructura…….…… 158
7.3.4 Modelo de Interacción…….……... 161
7.3.5 Investigación en el área de Modelos en México…… 163
7.4 Conclusiones…….....……. 164
Referencias……...……..... 165
8. Investigación en el área de Métricas de Software…… 173
8.1 Introducción…….. 173
8.2 Mediciones de elementos técnicos TSP/PSP…… 178
8.3 Mediciones de elementos funcionales……...……... 179
8.4 Adopción de Mecanismos de Medición en México…...…….. 183
8.5 Conclusiones…….....……. 199
Referencias……...……..... 200
9. Experimentación en Ingeniería de Software……...……. 205
9.1 Introducción…….. 205
9.2 El Paradigma Experimental...……... 207
9.2.1 Definición……..…….…… 210
9.2.2 Diseño…..……..…….…… 210
9.2.3 Ejecución…..……..…….…… 211
9.2.4 Análisis……..…….…… 211
9.3 Ejemplo de aplicación del Paradigma Experimental…….…….. 211
9.3.1 Antecedentes del experimento a realizar…….…….... 213
9.3.2 Definición……..…….…… 214
9.3.3 Diseño…...……..…….……... 216
9.3.4 Ejecución……..…….…… 217
9.3.5 Análisis……..…….…… 219

9.4 La Experimentación en Ingeniería de Software en México…..…… 222
9.5 Conclusiones……..……..…...…….. 225
Referencias……...……..... 226

Capítulo 1.
Introducción a la Ingeniería de Software

Raúl Antonio Aguilar Vera, *Universidad Autónoma de Yucatán,*

Hanna Jadwiga Oktaba, *Universidad Nacional Autónoma de México,*

Reyes Juárez Ramírez, *Universidad Autónoma de Baja California.*

La Ingeniería de Software tiene como propósito el desarrollar soluciones automatizadas a necesidades reales expresadas por personas u organizaciones con intereses en común; para tal fin, dispone de un conjunto de técnicas, herramientas, métodos y procesos que son utilizados para el desarrollo, operación y mantenimiento de sistemas software.

1.1 Origen de la Disciplina

A finales de la década de los años sesenta, en respuesta a un conjunto de problemáticas —costos imprecisos, plazos de entregas incumplidos y requisitos no satisfechos— vinculadas con el proceso de construcción del software, y ante el crecimiento en las capacidades de los equipos de cómputo, se comenzó a reconocer la necesidad de contar con un nuevo enfoque —sistemático, disciplinado y cuantificable— para el desarrollo del software.

A finales de 1967, en una reunión del Comité de Ciencia de la Organización del Tratado del Atlántico Norte (OTAN), Friedrich Ludwic Bauer, en referencia a la crisis que atravesaba el proceso software, argumentó la urgente necesidad de aplicar la Ingeniería al proceso de desarrollo del software, dicha propuesta generó un impacto mediático, el cual dio lugar a un par de conferencias, las cuales fueron celebradas en Alemania en 1968 (Naur & Randell, 1969) y en Italia 1969 (Buxton & Randell, 1970); en dichas conferencias se analizaron aspectos relevantes para el proceso software, como son: el diseño, la especificación, así como la calidad del software, entre otros temas. A partir de entonces, la naciente disciplina fue acumulando un conjunto de métodos, técnicas, metodologías y buenas prácticas, tales que permitieron que para principios del siglo XXI, se dispusiera de un Cuerpo de Conocimientos lo suficientemente maduro, como para disponer y/o formar profesionistas especializados en el proceso de desarrollo de software.

La sociedad profesional del Instituto de Ingenieros Eléctricos y Electrónicos (*Institute of Electrical and Electronics Engineer-Computer Society: IEEE-CS*) y la Asociación de Maquinaria Computacional (*Association for Computing Machinery: ACM*) en su proyecto denominado SWEBOK (Alain et al., 2004), compilaron el conocimiento generalmente aceptado en la disciplina hasta 2004 y establecieron dos subconjuntos de áreas de conocimiento: cinco vinculadas con los procesos de desarrollo y otras seis con las áreas de

gestión. Dicho documento fue actualizado en 2014 manteniendo básicamente los mismas áreas (Bourque & Firley, 2014).

1.2 Áreas de Conocimiento

1.2.1 Áreas de Desarrollo

El desarrollo de software, analizado desde la óptica de la dualidad Proceso-Producto, se circunscribe en cinco fases —requisitos, diseño, codificación, pruebas y mantenimiento— que integran un conjunto de actividades y tareas organizadas para un proyecto específico (Aguilar *et al*, 2007). La Figura 1.1 ilustra las cuatro fases vinculadas al desarrollo, así como la tarea de mantenimiento del software.

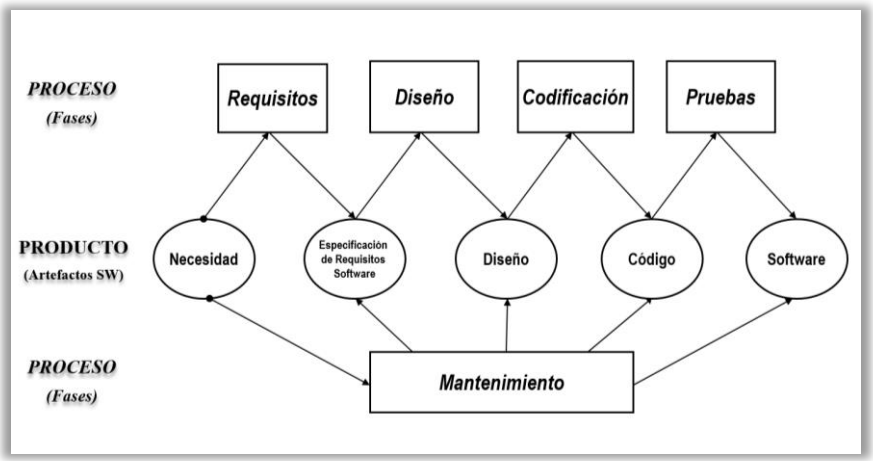

Figura 1.1 Dualidad Proceso-Producto en la Ingeniería de Software

Desarrollo de Requisitos. Se entiende por requisitos de software, como el conjunto de necesidades y restricciones expresadas respecto de un producto de software; esta área de conocimiento integra el conjunto de procesos vinculados con la obtención, análisis, especificación, validación, así como los procesos de gestión de los requisitos durante el ciclo de vida de un sistema de software.

Diseño de Software. El proceso de diseño de software se refiere tanto al proceso de definir la arquitectura, componentes, interfaces, modelo de persistencia de los datos, así como al resultado del mismo.

Programación. La creación detallada del software a través de un conjunto de procesos vinculados con la codificación del software, haciendo uso de algún lenguaje de programación, es conocida como la fase de programación o codificación.

Pruebas. Las pruebas de software son un conjunto de procesos vinculados con la verificación dinámica del comportamiento esperado del software, con base en un conjunto finito de casos de prueba debidamente seleccionados. La estrategia de prueba del software generalmente comienza por evaluar componentes más pequeños en forma independiente —pruebas de unidad— seguida por el proceso de integrar todos los módulos o componentes que conforman el sistema desarrollado —pruebas de integración— y finalmente se realizan las pruebas de aceptación, para verificar el cumplimiento de los requisitos acordados con el cliente, concluyendo con las pruebas de aceptación en

presencia del cliente, a efecto de validar que el sistema cumple con sus expectativas..

Mantenimiento. El mantenimiento de Software tiene como propósito modificar el software existente y preservar su integridad; esta área se refiere a las actividades particulares vinculadas con los diferentes tipos de mantenimiento: correctivo, perfectivo, preventivo y adaptativo.

1.2.2 Áreas de Gestión

Gestión de la Configuración. El proceso software genera un conjunto numeroso de artefactos, que en algunos casos contienen características volátiles; esta área de conocimiento se refiere a los procesos de gestión vinculados con la identificación, documentación y control de todos los elementos de configuración acordados para un proyecto software.

Gestión de la Ingeniería de Software. El área se refiere a los procesos vinculados con la planeación, coordinación, medición, supervisión, control y generación de informes que garanticen que los productos y servicios de software se suministren de manera eficiente y efectiva.

Procesos de la Ingeniería de Software. Conocida también como "Procesos de Software", esta área se ocupan de las actividades de trabajo realizadas por ingenieros de software para desarrollar, mantener y operar software; particularmente aquel conjunto de actividades y tareas interrelacionadas que transforman los productos de trabajo de entrada en productos de trabajo de salida; resulta importante resaltar,

que esta área se vincula con las actividades de trabajo, y no con la ejecución de proceso para el software implementado.

Métodos y Modelos de la Ingeniería de Software. El SWEBOK en su versión 2014, incluyó esta área de conocimiento para hacer énfasis en aquellos métodos y modelos —particularmente aquellos que surgieron en las últimas dos décadas— que hacen referencia a tareas y actividades vinculadas con más de una de las fases del ciclo de vida software.

Gestión de la Calidad del Software. La calidad del Software ha sido definida como la capacidad del producto de software para satisfacer las necesidades declaradas e implícitas bajo condiciones especificadas; es un área de conocimiento particularmente importante para la Ingeniería del Software, se enfoca en la prácticas, herramientas y técnicas para definir la calidad del software, así como para evaluar su estado durante el desarrollo, mantenimiento y despliegue.

Práctica Profesional de la Ingeniería de Software. La versión publicada en 2014 del SWEBOK, reconoce, a través de esta área de conocimiento, la importancia de establecer un conjunto de conocimientos, habilidades y actitudes que deben poseer los profesionistas de esta disciplina, para el desempeño de una práctica profesional, responsable y ética.

Economía de la Ingeniería de Software. Se incorpora al SWEBOK en 2014, e incluye conceptos, herramientas y métodos de análisis económico para la toma de decisiones, con una perspectiva del negocio,

pero sobre todo considerando las características de los proyectos de software, en particular, aquellas vinculadas con el riesgo y la gestión de incertidumbre.

Fundamentos Computacionales, Matemáticos e Ingenieriles. La versión publicada en 2014 del SWEBOK, destaca la importancia del conocimiento de ciertas áreas relacionadas con la Ingeniería de Software, y aunque no intenta caracterizarlas, presenta los tópicos de las Ciencias Computacionales, de las Matemáticas y de la Ingeniería, que resultan relevantes para la disciplina.

1.3 Retos de la Disciplina

1.3.1. El software desde su estructura interna

Particularmente en México poco o nada se hace de investigación en la categoría "Teoría de la informática" (Juárez-Ramírez, et al, 2013). Esto trae por consecuencia que no se tenga la cultura de valorar la estructura interna del software desde una perspectiva de ciencia; esto es, no se pone atención en el enfoque algorítmico, en la arquitectura interna, en la calidad y rendimiento del código fuente. La atención está más enfocada en el software funcional, en la aceptación del usuario.

Es conveniente abrir líneas de investigación que atiendan esta vertiente; así mismo, hace falta promover el estudio de la ciencia de la computación y de la Ingeniería de Software desde esta perspectiva.

1.3.2. La formalización en producto y el proceso

La Ingeniería de Software ha sido aceptada como una disciplina de carácter práctico. Por ser relativamente nueva, comparada con otras disciplinas de ingeniería, le hace falta el carácter de formalización basada en fundamentos teóricos de las matemáticas y la ciencia de la computación. En el libro "*Basics of Software Engineering Experimentation*", Juristo y Moreno (2001) enuncian algunas de las observaciones más precisas sobre los problemas que estaban presentes en la Ingeniería de Software al final del siglo XX, mismos que continúan latentes a la fecha. Estas observaciones están relacionadas con la falta de rigurosidad formal utilizada en esta área, y sugieren que esta deficiencia hace ver a la Ingeniería de Software como un arte y no como una disciplina formal.

Con el fin de manejar formalización en la Ingeniería de Software, muchos esfuerzos ya han sido realizados, especialmente en ambientes académicos, y aunque se ha logrado progreso significativo (Juárez-Ramírez, 2008), queda mucho trabajo por hacer en varios temas de la Ingeniería de Software. Por ejemplo, parafraseando a Baroni (2005) "Las métricas están informalmente definidas (usando un lenguaje natural) o parcialmente formalizadas (usando un lenguaje formal o las matemáticas), pero los conceptos fundamentales que las sustentan están informalmente definidos. Dicho problema trae como consecuencia que se tengan definiciones ambiguas que pueden

propiciar diferentes interpretaciones, produciendo resultados divergentes sobre los mismos artefactos de software".

Por otro lado, Baroni (2005) también mencionaba que, contrariamente a la falta de formalización en muchos aspectos de la Ingeniería de Software, en algunos casos "el uso de formalismos complejos en la definición de las métricas puede no ser fácil de manejar por los practicantes y desarrolladores de software". Atendiendo a esta segunda observación, Juárez-Ramírez (2008) propuso modelos para un enfoque cuantitativo de valorar los artefactos software en la fase de análisis y diseño, los cuales están basados en conceptos matemáticos básicos que cualquier profesional de software puede dominar, tales como: la teoría de grafos, los diagramas de árbol, permutaciones y combinaciones.

1.3.3. Adopción de técnicas inteligentes

De acuerdo con Thayer, Pyster, & Wood (1981), los problemas típicos en Ingeniería de Software, son: (1) las especificaciones de los requisitos son frecuentemente incompletas, ambiguas, inconsistentes y/o inconmensurables; (2) los proyectos a menudo se retrasan y exceden el presupuesto; (3) no existen métodos para garantizar que el software entregado "funcionará" para el usuario; (4) falta de detección sistemática de defectos de software; (5) los procedimientos, métodos y técnicas para diseñar un sistema de control de proyectos que permitirán a los gerentes de proyectos controlar con éxito su proyecto no están

disponibles; (6) relación poco predecible de la duración del proyecto con la funcionalidad del programa; (7) no están disponibles mediciones o índices de "bondad" de código que pueden usarse como elementos de diseño de software; y (8) para la medición de la calidad del software, hay cientos de métricas propuestas y no hay suficientes pruebas de su validez y valor aportado.

Las técnicas de Inteligencia Artificial (IA) tienen la capacidad de mejorar el desarrollo de software (Basu, et al, 2017; Sorte, Joshi, & Jagtap, 2015) lo que ayuda a abordar los problemas mencionados anteriormente. Es conveniente aumentar el uso de las técnicas de IA para resolver problemas de la Ingeniería de Software. Hoy en día, "*Machine Learning*" (ML) está siendo más aceptado en este contexto (Garzoli, *et al*. 2013). Claramente, ML es un enfoque orientado a datos (Munoz, 2017; Mohri, Rostamizadeh & Talwalkar, 2012) y para aplicar algoritmos de ML en la resolución de problemas de Ingeniería de Software se debe abordar el siguiente desafío: generar y preparar datos del proceso de desarrollo de software y el producto de software para aplicar técnicas y algoritmos de ML.

Actualmente hay repositorios públicos que contienen datos sobre el proceso de software y el producto de software, tales como: (a) Repositorio de PROMISE: un repositorio de conjuntos de datos de investigación especializado en conjuntos de datos de investigación de ingeniería de software (Menzies, Krishna & Pryor, 2016); http://openscience.us/repo/, y (b) Repositorio ISBSG: una base de datos

funcional basada en el tamaño de proyectos de software. http://isbsg.org/project-data/. Teniendo en cuenta estos repositorios, se pueden descubrir las prácticas de generación y gestión de datos en el proceso de desarrollo de software. Además, el contenido de estos repositorios puede permitir a los investigadores abordar los desafíos de la preparación y el tratamiento de datos.

Existen pocos intentos de aplicar técnicas inteligentes en la solución de problemas de la Ingeniería de Software. Es conveniente generar una cultura de adopción de herramientas provenientes de otras disciplinas para resolver las problemáticas de la Ingeniería de Software.

1.4 Tendencias y Paradigmas Emergentes

Algunos pioneros de la Ingeniería de Software han proyectado el futuro de esta disciplina, tal es el caso de Barry Boehm (2006), quien enunció las principales características que presentan los sistemas software en la actualidad y las que presentarán en lo subsecuente, con lo cual, sustenta la necesaria evolución de esta disciplina: incremento considerable en tamaño, incremento en complejidad, diversidad en contenido, apertura a la interacción con otros sistemas.

Para 2020, Boehm (2006) augura tendencias computacionales muy variadas, tales como: nuevos tipos de plataformas inteligentes (materiales inteligentes, nanotecnología, dispositivos micro mecánico-eléctricos, componentes autónomos para censado y comunicación), nuevos tipos de aplicaciones (redes de sensores, materiales

configurables o adaptativos, adaptación de prótesis humanas) así como el desarrollo de la bioinformática.

En el contexto de la bioinformática, los sistemas a construir se vuelven complicados ya que las combinaciones de la biología y la informática incluyen: (1) Computación basada en la biología, la cual utiliza fenómenos biológicos o moleculares para resolver problemas computacionales más allá del alcance de la tecnología basada en el silicio; y (2) Incremento de las capacidades físicas o mentales del humano basadas en la computación, con dispositivos quizás integrados o conectados a los órganos humanos o sirviendo como hospedaje de los cuerpos humanos (o partes de éste).

La diversidad y complejidad computacional de los sistemas que se desarrollarán en los próximos años, implicarán un impacto muy fuerte sobre la Ingeniería de Software que tendrá que afrontar los nuevos problemas que se avecinan. Este análisis presentado por Boehm (2006) permite visualizar los campos a los que debe turnarse la Ingeniería de Software, ya que tradicionalmente se ha enfocado más en temas específicos de la misma disciplina y de la Ciencia de la Computación (Wang, Li & Ma, 2014).

La globalización de los sistemas extensos en tamaño y contenido es una realidad actual; nuevos paradigmas de computación han tomado auge, tal como lo enunció el propio Boehm. A continuación se describen tres de los principales paradigmas emergentes.

Computación en la nube (*Cloud Computing*). Es un tipo de computación basada en Internet que proporciona recursos de procesamiento compartidos y datos, para computadoras y otros dispositivos, en base a demanda. Es un modelo para habilitar acceso ubicuo bajo demanda a un conjunto compartido de recursos informáticos configurables (Mell & Grance, 2011); por ejemplo, redes de computadoras, servidores, almacenamiento, aplicaciones y servicios Los retos de este paradigma son: problemas en la identificación de la calidad de los servicios ofrecidos, las capacidades y herramientas para hacer esto son bastante pobres y los problemas en la concepción de sistemas y generación de soluciones.

Computación social (*Social Computing*). Tipo de computación que se refiere a los sistemas que permiten la obtención, representación, procesamiento, uso y difusión de la información que se distribuye a través de colectividades sociales tales como equipos, comunidades, organizaciones y mercados electrónicos (Schuler, 1994). Los retos son que los sistemas de computación social deberían poder acceder a las funciones móviles tales como correo electrónico, mensajes, conocimientos y soluciones de administración de contenido y tener acceso a las aplicaciones transaccionales y sistemas de información; esto involucra considerar arquitecturas complejas.

Datos masivos (*Big Data*). Es un término referente a conjuntos de datos que son muy grandes y complejos, tanto que las aplicaciones tradicionales de procesamiento son inadecuadas para manejarlos

(Snijders, Matzat & Reips, 2012). Este concepto hace referencia al almacenamiento de grandes cantidades de datos y a los procedimientos usados para encontrar patrones repetitivos dentro de esos datos. Los retos para tratamiento de "Big Data" incluyen el análisis, captura, minería de datos, búsqueda, intercambio, almacenamiento, transferencia, visualización, consultas, privacidad y actualización de la información.

Referencias

1. Aguilar, R., Oktaba, H., Juárez-Ramírez, R., Aguilar, J., Férnandez, C. Rodriguez, O. y Ucán, J. (2017) *Ingeniería de Software*. En Pineda, L (Editor) La computación en México por especialidades académicas. Academia Mexicana de Computación. ISBN: 978-607-97357-1-5. Capítulo V.
2. Alain, A. et al. (2004) *SWEBOK: Guide to the Software Engineering Body of Knowledge 2004 Version*, IEEE Computer Society, Los Alamitos, California, 2004.
3. Baroni, A. (2005). Quantitative Assessment of UML Dynamic Models. *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2005, Lisbon, Portugal, September 5-9, 2005, pp. 366-369.

4. Basu, T., Bhatia, A., Joseph, D. & Ramanathan L. (2017) Survey on the role of artificial intelligence in software". *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 5, no. 4, April 2017, pp. 7062– 7066.

5. Boehm, B. (2006) A view of 20th and 21st century software engineering. *Proceeding of the 28th International Conference on Software Engineering*, pp. 12-29, 2006.

6. Bourque, P. & Firley, R. (2014) *Guide to the Software Engineering Body of Knowledge. SWEBOK V3.0*. IEEE Computer Society Press, 2014.

7. Buxton, J. & Randell, B. (1970) *Software Engineering Techniques: Report of a conference sponsored by the NATO Science Committee*, Rome, Italy, 27-31 Oct. 1969, NATO.

8. Garzoli, F., Croce, D., Nardini, M., Ciambra, F. & Basili, R. (2013) *Robust Requirements Analysis in Complex Systems through Machine Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 44–58.

9. Juristo, N. y Moreno, A. (2001). *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers.

10. Menzies, T., Krishna, R. & Pryor, D. (2016) *The PromiseRepository of Empirical Software Engineering Data*. http://openscience.us/repo. North Carolina StateUniversity, Department of Computer Science, 2016. Available at http://openscience.us/repo, retrieved on June 30, 2017.

11. Mohri, M., Rostamizadeh, A. & Talwalkar, A. (2012). *Foundations of Machine Learning*. The MIT Press.

12. Munoz, A. (2017). *Machine learning and optimization*, Available at https://www.cims.nyu.edu/~munoz/files/ml_optimization.pdf, retrieved on June 30, 2017.

13. Naur, P. & Randell, B. (1969) *Software Engineering: Report of a conference sponsored by the NATO Science Committee*, Garmisch, Germany, 7-11 Oct. 1968, NATO.

14. Juárez-Ramírez, R. (2008). *Formalización de la rastreabilidad y la correspondencia entre artefactos de software aplicando teoría de grafos, axiomas, diagramas de árbol y permutaciones*. Tesis Doctoral de la Universidad Autónoma de Baja California, junio de 2008.

15. Juárez-Ramírez, R., Cortés, K., Toscano B., Oktaba, H., Fernández-y-Fernández, C., Flores, B., Angulo, F. (2013) Estado Actual de la Práctica de la Ingeniería de Software en México. *Memorias del Congreso Internacional de Investigación e Innovación en Ingeniería de Software*. Xalapa, Veracruz.

16. Schuler, D. (1994) Social Computing - Introduction to the special edition. *Communications of the ACM*, Vol. 37, Issue 1 (January 1994), pp. 28 – 108, 1994.

17. Sorte, B., Joshi, P. & Jagtap, V. (2015) Use of artificial intelligence in software development life cycle: A state of the art review. *International Journal of Advanced Engineering and Global Technology*, vol. 3, no. 3, March 2015, pp. 398–403.

18. Snijders, C., Matzat, U. & Reips, U. (2012) Big Data: Big gaps of knowledge in the field of Internet. *International Journal of Internet Science*. 7, pp. 1–5, 2012.

19. Thayer, R. Pyster, A. & Wood, R. (1981) Major Issues in Software Engineering Project Management. *IEEE Transactions in Software Engineering*, Vol. SE-7, No. 4, July 1981.

20. Wang, Z., Li, B. & Ma. Y. (2014) *An Analysis of Research in Software Engineering: Assessment and Trends*. Retrieved June 20, 2014; Available at https://arxiv.org/ftp/arxiv/papers/1407/1407.4903.pdf

Capítulo 2.
Formación de Recursos Humanos

Raúl Antonio Aguilar Vera, *Universidad Autónoma de Yucatán,*

Oscar Mario Rodríguez Elías, *Instituto Tecnológico de Hermosillo,*

Julio Cesar Díaz Mendoza, *Universidad Autónoma de Yucatán.*

2.1 La Educación en Ingeniería de Software

La formación de recursos humanos en el ámbito de la Ingeniería de Software ha sido un tema de análisis y discusión desde las primeras reuniones auspiciadas por la Organización del Tratado del Atlántico Norte (OTAN) en las que se discutieron los principales temas de interés de la naciente disciplina a finales de la década de los años sesenta (Naur & Randell, 1969; Buxton & Randell, 1970).

Apenas una década después se comenzaron a ofrecer en los Estados Unidos de Norteamérica los primeros programas formales —en el nivel de posgrado— en Ingeniería de Software; el primer programa comenzó a operar en 1978 en la Universidad Cristiana de Texas, y un año más tarde comenzó a operar el de la Universidad de Seattle. En el caso del nivel de licenciatura, Inglaterra fue el primer país en ofrecer en 1987 un programa en la disciplina en el Imperial College,

y poco después se incorpora a la oferta el de la Universidad de Sheffield; nueve años después, Estados Unidos de Norteamérica —en el Rochester Institute of Technology— ofrece un programa ingenieril en dicha área (Aguilar y Díaz, 2015).

En el caso de México, la llegada de la primera computadora en 1958 a la Universidad Nacional Autónoma de México (UNAM) —con el propósito apoyar la investigación— originó la oferta de cursos específicos en computación para físicos, ingenieros, químicos y matemáticos. Durante los años setenta se crearon diversos programas en el país con estudios en Computación y áreas afines y se comenzaron incluir asignaturas con tópicos de la Ingeniería de Software.

La Asociación Nacional de Instituciones de Educación en Informática (ANIEI), ante la ausencia de un núcleo básico de conocimientos para los profesionistas en Computación, comenzó a trabajar desde 1983 en un conjunto de Modelos Curriculares de Nivel Superior en Informática y Computación, dicho modelo incluyó, desde su primera edición en 1983, temáticas vinculadas con la Ingeniería de Software; cabe mencionar, que hasta el día de hoy, buena cantidad de Instituciones de Educación Superior (IES) utilizan dicho modelo como referencia para el diseño curricular de programas educativos vinculados con la Computación (García, Álvarez y Sánchez, 2015).

Los primeros programas documentados en México, bajo el nombre de Ingeniería de Software, comenzaron a operar en 2004; en el nivel de Licenciatura, en la Universidad Autónoma de Yucatán

(UADY), y en el de posgrado, en el Centro de Investigación en Matemáticas Aplicadas (CIMAT). Se tiene referencia de otros programas de posgrado que comenzaron a ofertarse entre 2004 y 2006, tal es el caso de la Universidad Tecnológica de la Mixteca, la Universidad Popular Autónoma del Estado de Puebla, así como la Universidad Veracruzana.

La proliferación de programas de licenciatura en Computación e Informática en nuestro país en las últimas dos décadas del siglo pasado, en particular, en áreas vinculadas con la Ingeniería de Software, fue tal que la ANIEI en 2006, adoptó como nombre del segundo de sus cuatro perfiles profesionales propuestos —conocido como perfil B, el de Licenciatura en Ingeniería de Software; dicho perfil es definido como: (1) "Profesional especialista en la producción de sistemas de software de calidad para la solución de diversas problemáticas del entorno. Es responsable de la formulación, planeación, implantación y mantenimiento de sistemas de información que garanticen la disponibilidad de altos niveles de servicio; y (2) Deberá tener una sólida formación en técnicas de análisis y diseño de sistemas de información y en la configuración de ambientes de servicios de cómputo y redes, así como dominio de herramientas de programación e ingeniería de software, con el fin de construir programas y sistemas de aplicación con características de productos terminados y competitivos; (3) Se trata también de un perfil de orientación profesional con amplias

posibilidades de continuación en niveles de especialización y posgrado" (García, Álvarez y Sánchez, 2015, p. xi).

El interés constante de las IES por mantener la pertinencia en sus programas educativos ante las transformaciones económicas, políticas y socioculturales de las últimas décadas, ha generado nuevos paradigmas educativos, entre los destaca el Modelo por Competencias. La ANIEI, en respuesta a los nuevos Modelos Educativos adoptados por la Instituciones, aprobó en la asamblea general de asociados en junio de 2017, un conjunto de competencias específicas para cada uno de sus cuatro perfiles profesionales propuestos. En el caso del perfil B, las competencias específicas se identifican como: (a) Realiza ingeniería de requisitos de software, (b) Diseña Software, (c) Construye Software, (d) Realiza Pruebas de Software, (e) Realiza mantenimiento de Software, (f) Administra proyectos de software, (g) Estima parámetros del proyecto de software, (h) Asegura la Calidad del Software, (i) Establece mecanismos de seguridad, (j) Emplea ciclos de vida, (k) Verifica calidad de soluciones de software, (l) Usa herramientas para creación de software.

Con el fin de establecer parámetros que nos ayuden a identificar la situación actual de la formación de recursos humanos en Ingeniería de Software en México, llevamos a cabo un primer análisis de los programas académicos de licenciatura y posgrado de algunas de las principales IES en el país; la elección de las instituciones revisadas se basó en el listado de la Guía Universitaria 2016-2017

(www.guiauniversitaria.mx). Cabe destacar, que en el caso de los Institutos y Centros pertenecientes al Tecnológico Nacional de México, se consideraron como una sola Institución, con 266 "campus", que incluyen Institutos Tecnológicos Federales, Descentralizados, así como Centros de Investigación pertenecientes al antiguo sistema de tecnológicos.

Adicionalmente a la lista de IES, también se incluyeron los centros CONACYT, lo cual nos dio un total de 121 instituciones analizadas, de las cuales 46 (38%) no cuentan con programas relacionados, mientras que las restantes 75 (62%) cuentan con al menos un programa. Para el análisis realizado de consideraron todos los programas de licenciatura y posgrado relacionados con la computación, y estos fueron clasificados en una de las siguientes cuatro categorías: (1) Programas que incluyen materias aisladas relacionadas con el desarrollo de software, por ejemplo programación, pero que no abordan temas específicos de la Ingeniería de Software; (2) Programas que incluyen materias organizadas en el área de la Ingeniería de Software, incluyendo materias específicas de la Ingeniería de Software; (3) Programas que cuentan con alguna línea de salida o especialización en algún área de la Ingeniería de Software; y (4) Programas específicamente enfocados en Ingeniería de Software o alguna de sus sub-áreas.

La gran mayoría de programas están en las categorías 1 y 2 (80%), es decir, programas que integran materias aisladas en el área del

desarrollo de Software (principalmente programación), o a lo más un grupo de materias relacionadas con la Ingeniería de Software (materias introductorias a la Ingeniería de Software).

Con respecto a los programas que cuentan con alguna línea de salida o especialidad relacionada con la Ingeniería de Software, se encontraron 31, que representan el 12%; de ellos, 16 programas son de licenciatura (6%), 11 de maestría (4%), y 4 de doctorado (2%). Con respecto a los programas totalmente enfocados en Ingeniería de Software, se encontraron que estos constituyen el 8% del total; siendo 15 programas de licenciatura (6%), uno de especialidad, y cuatro de maestría (2%).

Con lo descrito en el párrafo anterior, se identifica la importancia que tiene la Ingeniería de Software dentro del área de la computación en nuestro país, ya que si consideramos todos los programas que incluyen materias básicas específicas en la disciplina, aunque no sea el principal enfoque del programa —programas de las categorías 2, 3 y 4 representan el 66% de los programas identificados, siendo el 71% de los programas de licenciatura, 50% de las especialidades, 57% de las maestrías, y 61% de los doctorados.

Ante la diversidad de programas que se ofertan relacionados con el área de la Ingeniería de Software en nuestro país, así como la creciente demanda de reconocimiento a su calidad, se decidió realizar un segundo análisis con la información reportada por los cuatro organismos

nacionales con autoridad para otorgar dichos reconocimientos (ver Tabla 2.1). En este caso los organismos identificados fueron:

Los Comités Interinstitucionales de Evaluación de la Educación Superior (CIEES), organismo que dio nacimiento en 1991 al proceso de aseguramiento de la calidad de la educación superior mexicana (www.ciees.edu.mx). Se identificaron 21 programas pertenecientes a IES —8 Institutos Tecnológicos y 13 Universidades— de 14 estados de la República Mexicana, los cuales fueron evaluados por el Comité de Ingeniería y Tecnología.

El Consejo de Acreditación en la Enseñanza de la Ingeniería (CACEI) constituido en 1994, tiene como objeto de evaluación a los programas de corte Ingenieril (www.acei.org.mx). Se identificaron 79 programas educativos distribuidos en 29 estados del país, vinculados con el perfil del Ingeniero de software que corresponden al 9.12% de los programas acreditados por dicho organismo.

El Consejo Nacional de Acreditación en Informática y Computación (CONAIC) constituido en 2002 (www.conaic.net), evalúa programas del área de Computación e Informática, clasificados en cuatro perfiles profesionales. Se identificaron 33 programas en el Perfil B —Ingeniero de software— que corresponden al 23.08% de los programas acreditados por dicho organismo; dichos programas se ubican en 16 estados.

El Centro Nacional de Evaluación para la Educación Superior (CENEVAL) instauró desde el curso escolar 2010-2011 (www.ceneval.edu.mx), un reconocimiento que se otorga a los programas educativos con base en un Indicador de Desempeño Académico (IDAP) obtenido por perfil profesional. Para el período 2016-2017 se identificaron 7 programas reconocidos en el estándar 1 y 4 programas en el estándar 2.

En el informe del CENEVAL para 2017 (Ceneval, 2017), se reporta que se tuvieron un total de 2,939 sustentantes (77 % hombres) en el Examen General para el Egreso (EGEL-ISOFT) provenientes de 145 IES del país o planteles de ellas. El 6.84% (201 sustentantes) obtuvieron Testimonio de Desempeño Sobresaliente; el 41.82% (1,229 sustentantes) Testimonio de Desempeño Satisfactorio y el 51.35% (1,509 sustentantes) no obtuvo testimonio; solamente dieciocho estudiantes provenientes de ocho IES obtuvieron un Testimonio de Desempeño Sobresaliente en todas las áreas del examen.

2.2 Organismos y Eventos Promotores de la Disciplina

Una de las primeras asociaciones en reconocer la importancia de la Ingeniería de Software en el ámbito de la Computación, fue la Asociación Nacional de Instituciones de Educación en Informática (ANIEI) —actualmente denominada Asociación Nacional de Instituciones de Educación en Tecnologías de la Información— constituida desde octubre de 1982 (http://www.aniei.org.mx/ANIEI/).

Tabla 2.1. Programas de Ingeniería de Software reconocidos por su Calidad, de acuerdo con los cuatro organismos autorizados en México.

Organismo	# PE	# IES	Estados	Observaciones
Comités Interinstitucionales de Evaluación de la Educación Superior (CIEES)	21	21 Instituciones: 8 Institutos Tecnológicos y 13 Universidades	14 estados: Aguascalientes (2), Campeche, Guanajuato (2), Hidalgo, México, Michoacán, Nuevo León, Puebla, Querétaro (3), Quintana Roo, Sinaloa, Tabasco, Tamaulipas (3), Yucatán (2)	Programas con Reconocimiento Vigente en Nivel 1
Consejo de Acreditación en la Enseñanza de la Ingeniería (CACEI)	79	79 Instituciones: 67 Institutos Tecnológicos y 12 Universidades	27 estados: Baja California Norte (2), Baja California Sur, Campeche, Chihuahua (5), Ciudad de México, Coahuila (3), Colima, Durango, Estado de México (7), Guanajuato (2), Guerrero (3), Hidalgo (5), Jalisco (3), Michoacán (6), Morelos, Nayarit, Nuevo León (2), Puebla (5), Querétaro (2), Quintana Roo, San Luis Potosí (3), Sonora (6), Tabasco, Tamaulipas (4), Veracruz (8), Yucatán, Zacatecas (3).	Programas Acreditados Vigentes
Consejo Nacional de Acreditación en Informática y Computación (CONAIC)	33	25 Instituciones 12 Institutos Tecnológicos y 14 Universidades	16 estados: Baja California Sur, Campeche, Chiapas (2), Colima, Estado de México (2), Guanajuato (4), Hidalgo (2), Morelos, Nuevo León, Querétaro, Sinaloa (2), Sonora (4), Tabasco (2), Veracruz (5), Yucatán, Zacatecas (3).	Programas Acreditados Vigentes
Centro Nacional de Evaluación para la Educación Superior (CENEVAL)	8	8 Instituciones 5 Institutos Tecnológicos y 3 Universidades	7 estados Ciudad de México (2), Estado de México (4), Querétaro (3), Nuevo León (3), Veracruz (2), Michoacán (2) Sonora (1), Hidalgo (1).	Programas con Reconocimiento en el período 2016-2017 en los estándares 1 y 2

La ANIEI propuso un conjunto de Modelos Curriculares para el Nivel Superior de Informática y Computación, el cual incorpora cuatro perfiles profesionales en Informática y Computación, dichos perfiles son configurables a través de una matriz de ponderaciones respecto de ocho áreas de conocimientos vinculados con programas curriculares en Computación e Informática; dos de dichas áreas —Programación e Ingeniería de Software, Tratamiento de la Información— tienen relación directa con tópicos de la Ingeniería de Software.

Otro de los organismos que han reconocido la importancia de la Ingeniería de Software en el ámbito de la Computación, es el Consejo Nacional de Acreditación en Informática y Computación (CONAIC), constituida en enero de 2002 a propuesta de la ANIEI, que surge al identificar la necesidad de contar con una instancia reconocida —por el gobierno federal— que pudiese evaluar, y en su caso acreditar, programas educativos en Informática y Computación, en particular, en el perfil profesional de "Ingeniero de Software".

Un organismo de reciente creación, ha sido la propia Academia Mexicana de Computación (http://amexcomp.mx/), organismo que dese su creación en 2015, evidenció la necesidad de conformar una sección académica para el área de la Ingeniería de Software.

Entre las primeras reuniones relacionados con la Ingeniería de Software en México, podemos citar la reunión que se llevó a cabo en Acapulco en 1996, del grupo SPICE (*Software Process Improvement Capability Determination*) cuyo trabajo se convirtió posteriormente en

el estándar ISO/IEC 15504:1998; dicha actividad fue un hito importante para despertar el interés en esta materia.

La historia de eventos académicos en el ámbito de la Ingeniería de Software, se inicia en 1997 con el Encuentro Nacional de Computación ENC'97 organizado por la Sociedad Mexicana de Ciencias de la Computación; en este evento se crea el Taller de Ingeniería de Software, precursor de varios eventos importantes en la disciplina, entre ellos, el Coloquio Nacional de Investigación en Ingeniería de Software en 2010 (CONIIS) y desde 2012, los denominados Congresos Internacionales en Investigación e Innovación en Ingeniería de Software (CONISOFT) que año con año se celebran en una ciudad distinta en México. Otro de los eventos importantes en la disciplina realizados en territorio mexicano, ha sido las Jornadas Iberoamericanas en Ingeniería de Software e Ingeniería del Conocimiento (JIISIC) organizadas en Puebla en 2006 por la UPAEP y por la UADY en Mérida en 2010; dicho evento se organizó por primera vez en Buenos Aires (Argentina) en 2011. Por su parte, la Conferencia Internacional en Mejora de Procesos Software (CIMPS) organizada por Centro de Investigación en Matemáticas (CIMAT) desde 2012, ha sido también otro de los eventos que año con año ha venido creciendo en calidad en cuanto a sus trabajos y publicaciones generadas, sus primeras ediciones fueron realizadas en Zacatecas, sin embargo, desde la edición de 2015 ha sido celebrada en una ciudad diferente de nuestro país.

Referencias

1. Naur, P. & Randell, B. (1969) *Software Engineering: Report of a conference sponsored by the NATO Science Committee*, Garmisch, Germany, 7-11 Oct. 1968, NATO.
2. Buxton, J. & Randell, B. (1970) *Software Engineering Techniques: Report of a conference sponsored by the NATO Science Committee*, Rome, Italy, 27-31 Oct. 1969, NATO.
3. Aguilar, R. y Díaz, J. (2015) La Ingeniería de Software en México: hacia la consolidación del primer programa de licenciatura, *Revista de Tecnología Educativa*, Vol 2. Num. 2. pp. 6-17.
4. CENEVAL (2018). Informe Anual de Resultados 2017: Examen General de Egreso para la Licenciatura en Ingeniería de Software, Centro Nacional de Evaluación para la Educación Superior.
5. García, A., Álvarez, F. y Sánchez, M. (2015). *Modelos Curriculares del Nivel Superior de Informática y Computación*. Pearson.

Capítulo 3.
La Industria del Software en México

Hanna Jadwiga Oktaba, *Universidad Nacional Autónoma de México,*

María Guadalupe Elena Ibargüengoitia González, *Universidad Nacional Autónoma de México,*

Enzo Molino Ravetto, *Grupo Corporativo Informático,*

Carlos Antonio Zozaya Gorostiza, *Grupo Nacional Provincial.*

3.1 Introducción

Prácticamente casi todas las actividades humanas dependen actualmente del software que se ejecuta en múltiples dispositivos. Por eso la Ingeniería de Software es un área muy importante con el gran impacto en nuestra vida diaria. En México, no hemos logrado crear la tecnología para producir dispositivos de Tecnologías de la Información (TI) propios por falta de la inversión adecuada en la investigación y desarrollo tecnológico. Sin embargo, hemos logrado crecer la industria de desarrollo de software, que en gran medida depende de la investigación y la educación en Ingeniería de Software.

La presente sección resume el estatus en el que se encuentra actualmente la industria de software en México. La sección comienza por presentar brevemente la historia de los programas de apoyo, que se han tenido en nuestro país por parte del gobierno. Posteriormente, se describen los impactos que ha tenido la adopción de modelos de calidad en la industria de software en México. Todos estos logros no se hubieran podido alcanzar sin la organización de los interesados del ámbito académico y profesional, lo que se resume en la sección de eventos y asociaciones.

Finalmente, se explican las rutas de acción estratégicas que los expertos de la industria han propuesto para lograr que nuestro país cuente con 1,000 centros de desarrollo de calidad suprema en el año 2024.

Cabe señalar que el mapa y las rutas 2024 que se describe al final de esta sección será viable y estratégico para el logro de la Visión 2024 de Prosoft 3.0, en la medida que se formule un proyecto de industria que despliegue las 8 rutas estratégicas: Talento, Innovación empresarial, Mercado digital, Globalización, Financiamiento, Certeza jurídica, Regionalización inteligente, Gobernanza; para lograr los indicadores de calidad, de productividad y de competencias profesionales de excelencia que posicionen local y globalmente a la industria de software mexicana. Con ello se contribuirá al crecimiento interno del país, a sus exportaciones y a la atracción de inversión.

3.2 Programas gubernamentales para el uso de la computación y el fomento a la industria de software

Uno de los primeros antecedentes en el Gobierno de México, relacionados con la computación, fue la creación en 1971 de un programa para racionalizar el uso de las computadoras en el sector público. El alcance de este programa ha sido la generación de propuestas y estudios de viabilidad.

En 1981 la SECOFI formuló el "Programa para la promoción de la manufactura de sistemas electrónicos computacionales" que tenía como finalidad, la producción nacional de mini y micro computadoras pero en la década de los 90's se abrió el mercado a la importación de computadoras. Esto aceleró el crecimiento de la industria de software por el aumento del acceso a los equipos de cómputo.

Hacia 1983 se crea el Instituto Nacional de Estadística, Geografía e Informática con una Dirección General de Política Informática, cuyo logro es la publicación en 1996 del primer Programa de Desarrollo Informático. Este programa tuvo como objetivo, entre otros: Promover el uso de computadoras en los sectores: público, privado y social del país, así como impulsar la formación de recursos humanos y estimular la investigación científica en el área de computación.

En 2001 se creó el Sistema Nacional de e-México a cargo de la Secretaría de Comunicaciones y Transportes (SCT). Su finalidad era

desarrollar la conectividad a través de centros comunitarios digitales e incorporaba la participación de varias secretarías del gobierno. Este programa, más recientemente llamado México Conectado, había alcanzado 49,000 instalaciones públicas con acceso a Internet de banda ancha en 2013.

En 2002 la Secretaría de Economía (SE) empezó a organizar mesas de trabajo para definir las estrategias del programa para el desarrollo de la industria de software en México, conocido como PROSOFT. Se hicieron estudios para explorar mercados nacionales para la industria, indagar el perfil de la industria a nivel estatal y respaldar el desarrollo del "Modelo de Procesos de Software" mediante un convenio entre la Secretaría de Economía la Universidad Nacional Autónoma de México (UNAM) con el objetivo de promover normas de calidad dirigidas a las pequeñas empresas que no compiten en el mercado mundial. El resultado del proyecto, presentado al inicio de 2003, fue un Modelo de Procesos para la Industria de Software (Oktaba et al, 2003) conocido generalmente como MoProSoft; dicho modelo es un conjunto integrado de procesos de Gestión e Ingeniería de Software, compuesto por prácticas reconocidas. Una vez definido MoProSoft en 2003 se inició el proyecto para definir un método de Evaluación de Procesos de Software (Oktaba et al, 2004) conocido como EvalProSoft; y el tercer proyecto de Pruebas Controladas cuyo objetivo fue demostrar que, en un lapso de tiempo relativamente corto, las empresas pueden elevar sus niveles de capacidad adoptando MoProSoft.

Una vez confirmado el valor en la práctica de MoProSoft y EvalproSoft, desde el inicio de 2005 los esfuerzos se centraron en convertir los dos modelos en la norma mexicana. El trabajo se realizó dentro del Subcomité de Software del NYCE (Normalización Y Certificación en Electrónica) creado con este propósito debido a que éste fue el primer esfuerzo de normalización para la industria de software en México. El nombre completo de la primera norma mexicana para la industria de software es MNX-I-059-NYCE-2005 Tecnología de la Información-Software-Modelos de procesos y evaluación para desarrollo y mantenimiento de software.

A partir de 2006 la Secretaría de Economía a través del programa PROSOFT ha otorgado apoyos a las empresas para la adopción de la norma basada en MoProSoft. Hasta la fecha más de 500 empresas han adoptado el modelo y han demostrado los niveles de madurez de capacidades entre niveles 1 a 3 verificados por NYCE o CERTVER. Cabe destacar que MoProSoft se convirtó en un estándar internacional ISO/IEC para pequeñas organizaciones. En junio de 2005 el grupo ISO/IEC JTC1 SC7 *Software and System Engineering* reconoció que sus estándares no eran apropiados para los proyectos y organizaciones pequeñas, por lo que se creó un grupo de trabajo el WG 24 específico para definir procesos de software para empresas, organizaciones o proyectos de 1-25 personas. El Perfil Básico, basado en los procesos de operación de MoProSoft, fue la primera guía de procesos publicada en 2011 dentro de la colección de documentos que

conforman el nuevo estándar ISO/IEC 29110 dirigido a pequeñas organizaciones de desarrollo de software.

A finales de 2005, el Programa Iberoamericano de CYTED aprobó el proyecto denominado COMPETISOFT: Mejora de procesos para fomentar la competitividad de la pequeña y mediana industria de Iberoamérica, previsto para tres años. En dicho proyecto participaron 23 grupos académicos y de la industria de 13 países; la idea, a grandes rasgos, fue repetir a nivel iberoamericano lo que se había hecho en México para generar la norma. Los modelos de MoProSoft y EvalProSoft fueron la base para el proyecto. Los resultados de dicho proyecto fueron reportados en dos materiales con audiencias diferenciadas, el primero fue dirigido al mercado internacional (Oktaba & Piatinni, 2008) y el segundo material tenía como objetivo el mercado de hispanoparlantes (Oktaba, et al. 2009). En 2007 se revisaron y reorientaron las estrategias del PROSOFT con los objetivos de:

- Mejorar sus resultados, de lo cual derivó una estrategia para impulsar los clústeres de TI,
- Expandir el financiamiento a estas actividades,
- Crear mejores condiciones para la producción de software para medios interactivos,
- Llevar a cabo el diseño, construcción y operación de parques tecnológicos, y
- Ampliar los servicios del e-gobierno, entre otros.

Dentro del PROSOFT 2.0 (2008-2012), se creó la estrategia "México IT" cuyo fin era posicionar a México como proveedor de TI en el contexto global. También se fundó la iniciativa "*MexicoFIRST*" para facilitar el entrenamiento y la certificación del personal que trabaja en empresas de TI en los programas diseñados por las grandes empresas multinacionales de este sector y en la adopción de modelos de calidad. Asimismo, en 2008 la Secretaría de Economía creo la Iniciativa Nacional TSP™ (*Team Software Process*) para promover la calidad en la línea de producción del desarrollo de software.

El programa PROSOFT 3.0 surge en 2013 con una visión al 2024 y tiene un alcance mayor a sus antecesores. Sus principales objetivos son:

- Ampliar el mercado digital nacional,
- Impulsar la innovación empresarial,
- Mejorar el talento que requiere la industria de software, para cubrir el 90% de las necesidades de capital humano,
- Ampliar al doble el número de empresas que cuentan con certificaciones de calidad,
- Promover la inserción del sector en el mercado internacional,
- Mejorar las posibilidades de financiamiento y atraer más inversión extranjera directa (IED),
- Promover una regionalización "inteligente", favoreciendo nichos específicos de TI de alto valor agregado, y
- Mejorar la certeza jurídica y la gobernanza para este sector.

Los apoyos otorgados por el Fondo PROSOFT pueden dirigirse a las áreas de capacitación, certificación, habilitación y equipamiento tecnológico, normas y modelos, adopción y producción de TI, innovación, comercialización, estudios para desarrollar capacidades de negocio, entre otros. También hay un descuento de 30% sobre el pago anual del Impuesto sobre la Renta (ISR) para la Investigación y Desarrollo (I+D), aplicado a las industrias que desarrollen este tipo de actividad.

En junio de 2015 se decide fusionar el PROSOFT 3.0 con el Fondo Sectorial de Innovación (FINNOVA) de la Secretaría de Economía y el Consejo Nacional de Ciencia y Tecnología (CONACYT), el Fondo de Co-inversión de Capital Semilla (FCCS) y el Fondo de Capital Emprendedor (FCE), con miras a crear un nuevo programa llamado "Programa para el Desarrollo de la Industria del Software y la Innovación". La integración de estos programas pretende implementar las mejores prácticas de estos programas y generar sinergias de manera transversal en los sectores e industrias del país, dando prioridad a aquellos que se establecen en el Programa de Desarrollo Innovador (PRODEINN).

Por otro lado, entre 2011 - 2016 se han publicado en el Diario Oficial las diferentes versiones del Manual Administrativo de Aplicación General en materia de Tecnologías de la Información, Comunicaciones y de Seguridad de la Información (MAAGTISI) como

norma oficial obligatoria, que regula los desarrollos de software en el contexto del Gobierno.

3.3 Impactos de la adopción de modelos de calidad en la industria de software.

Un estudio reciente (Select, 2015) cuantificó el impacto que ha tenido en México la adopción de modelos de calidad en el desempeño de las empresas desarrolladoras de software. El estudio se realizó con 140 entrevistas sobre un inventario de 707 centros de desarrollo de software que en septiembre 2014 tenían una certificación y/o verificación, de alguno de los modelos CMMI, NMX-I-059/NYCE (NMX-MoProSoft), o ISO/IEC 29110. Los principales resultados de este estudio fueron los siguientes:

El 87% de las organizaciones son empresas micro y pequeñas y sólo el 13% de las organizaciones tienen un número mayor a 100 empleados. Asimismo, más de la mitad de las organizaciones (54%) son empresas de entre 1 y 25 empleados y otro 16% tienen de 26 a 50 empleados.

El 65% de las empresas se concentran en 3 estados del país: Ciudad de México (26%), Jalisco (25%) y Nuevo León (14%); el cuarto lugar lo tiene Sinaloa con un 9% de las empresas certificadas.

El 77% de las organizaciones elabora proyectos de desarrollo a la medida y el 48% producen software estándar; sólo un 3% de las

empresas señalaron que en su empresa realizan integración de soluciones.

El 56% de las empresas entrevistadas cuentan con una certificación de tipo CMMI, el 52% con una verificación NMX-MoProsoft, y sólo el 4% se encuentra certificada en ISO/IEC 29110. En lo que se refiere a otras certificaciones, sólo el 3% de las organizaciones cuenta con alguna certificación de tipo TSP o bien PSP.

Con respecto a los beneficios que tuvieron las empresas por haber logrado la acreditación / certificación en un modelo de calidad, un 53% de los entrevistados calificaron los beneficios de la certificación como altos (calificaciones de 8 a 10), un 19% como medios (calificaciones de 5 a 7), y un 29% como bajos (calificaciones de 1 a 4).

Sin embargo, un análisis del impacto en las variables financieras de las empresas muestra un crecimiento del 27% en utilidades, de 29% en la adopción de nuevos clientes, de 30% en ventas, y de 50% en la adopción de nuevos clientes. Asimismo, un análisis del impacto en las variables operativas arrojó reducciones del 8% en los costos de desarrollo, de 22% en los tiempos de entrega, y del 44% tanto en el re-trabajo como en el número de defectos identificados en pruebas.

Finalmente, el estudio demostró que existe una relación entre el nivel de madurez en la adopción de CMMI y NMX-MoProSoft y mejora en las variables financieras y operativas de las empresas.

3.4 Mapa de rutas para impulsar la Industria del Software

Atendiendo una convocatoria abierta por parte del gobierno, en 2015 se organizó un coloquio con el objetivo general de generar un Mapa de Rutas que sirva para guiar las acciones gubernamentales orientadas a fortalecer las capacidades requeridas para apoyar la formación de 1,000 centros de desarrollo de software de "calidad suprema", la cual se define en el Prosoft 3.0 (https://mapa-prosoft.economia.gob.mx/MapaProsoft/) como los niveles 4 y 5 de madurez del modelo CMMI, los niveles 3, 4 y 5 del modelo MoProSoft, o la obtención del TSP PACE (*Team Software Process Performance And Capability Evaluation*). Al evento denominado "Coloquio Software en México: Calidad, Innovación y Productividad hacia el 2024" asistieron 117 especialistas relacionados con labores profesionales en Centros de Desarrollo, así como en organizaciones demandantes de software, empresas de consultoría y academia.

Los resultados del Coloquio se plasmaron en un documento, que fue redactado por un grupo de reconocidos expertos, en el cual se describen ocho rutas estratégicas para alcanzar la meta de contar con estos 1,000 centros de calidad suprema para el año 2024 (ver Figura 3.1).

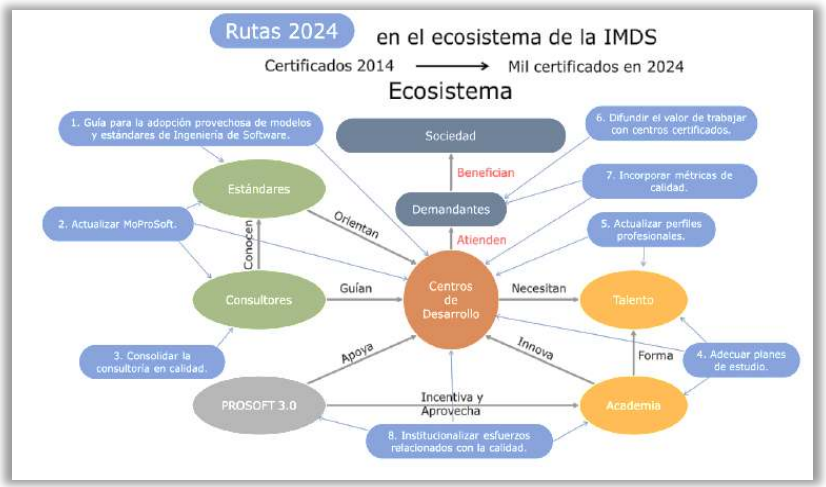

Figura 3.1. Mapa de rutas estratégicas para lograr el objetivo del Prosoft 3.0 de tener 1,000 centros de desarrollo de software de calidad suprema

Un resumen de estas ocho rutas estratégicas obtenido a partir de dicho documento, es el siguiente:

Guía para la adopción de modelos y estándares. Existen Centros de Desarrollo de software que han seleccionado modelos o normas de calidad y que han invertido en su adopción, algunos han recibido apoyos de PROSOFT y algunos han abandonado sus proyectos de mejora al no percibir suficientes beneficios. Ello se debe, entre otros factores, a que la elección de los modelos y normas acordes a las características de la organización (tamaño, antigüedad, ubicación, especialización), a su mercado (nacional, internacional), y a sus objetivos de negocio, no ha resultado una tarea fácil, debido la variedad de opciones y a la falta de información y conocimiento de beneficios en

casos de éxito. Esta ruta pretende elaborar una guía que apoye a los Centros de Desarrollo de software en la selección de modelos y normas en función de sus características, contexto, mercado y objetivos de negocio de las entidades.

Actualizar MoProSoft. MoProSoft fue creado en 2003, de modo que para 2015 es lógico pensar que requiera una actualización acorde con el avance de la Ingeniería de Software en este periodo y con la experiencia de su adopción en los Centros de Desarrollo. La definición actual de los niveles de madurez, dificulta el reconocimiento del avance real en la maduración de los centros con el tiempo. Por otra parte, el estándar internacional ISO/IEC 29110 *Basic Profile*, generado a partir de los procesos de operación de MoProSoft, contiene aportaciones de la comunidad internacional que deberían incorporarse a la norma mexicana. Ahora bien, los expertos consideraron que, dados los tiempos de desarrollo de los estándares internacionales, no se vislumbra que en los próximos 3- 5 años MoProSoft pueda ser completamente sustituido por ISO/IEC 29110, y por lo tanto resulta conveniente actualizarlo.

Consolidación de la Consultoría en Calidad. La Consultoría en mejora de procesos para la industria del software es una actividad profesional relativamente nueva; sus inicios pueden remontarse a no más de 30 años con la aparición de los primeros compendios de mejores prácticas relacionados con este sector. Desafortunadamente, los Centros de Desarrollo no tienen ninguna certeza de que el consultor contratado les pueda brindar un servicio de calidad, ni tampoco

disponen de un organismo independiente al que puedan recurrir para consulta y ante el cual puedan quejarse por un mal servicio prestado por una empresa consultora o un consultor particular. En el caso particular de la industria de TI, fuertemente subsidiada para la implementación de modelos de calidad, tampoco existe un mecanismo estandarizado para que un cliente mal atendido pueda presentar su reclamo ante las instancias que apoyan estas iniciativas, tales como organismos promotores, Cámara Nacional de la Industria Electrónica, de Telecomunicaciones y Tecnologías de la Información (CANIETI), Asociación Mexicana de la Industria de Tecnologías de Información (AMITI) o el mismo PROSOFT. Esta ruta implica adoptar un modelo de certificación para Consultores en normas y modelos para la industria de TI, que los habilite para brindar Consultoría, capacitación y evaluación de los Estándares nacionales e internacionales.

Adecuar planes de estudio. Los planes de estudio relacionados con disciplinas para desarrollo de software, que las instituciones de educación superior y sus docentes imparten, no están respondiendo a las necesidades de la industria de software. Al 2014 se contaba con múltiples carreras relacionadas con el desarrollo de software, pero los planes de estudio de muchas de ellas no siempre están alineados con las necesidades de la industria, ni con las prácticas de la ingeniería de software y las prácticas de calidad. Dichos planes de estudio deben ser actualizados con mayor frecuencia para ajustar su velocidad de respuesta a los cambios acelerados que caracterizan a la economía del

conocimiento. En particular, se considera necesario fortalecer la formación de habilidades blandas y la adopción de modelos de calidad. Esta ruta pretende: a) crear un proyecto de industria para analizar la situación actual y deseada de los planes de estudio y de los docentes, aprovechando los aciertos del proyecto IMPULSA (2015–2016), b) revisar y actualizar los perfiles de carreras, c) revisar y adecuar el examen CENEVAL, d) establecer un mecanismo de entrenamiento continuo de los docentes, y e) evaluar la satisfacción del nivel de preparación de los egresados.

Actualizar perfiles profesionales. En complemento a la adecuación de los planes de estudio, resulta necesario actualizar los perfiles profesionales para el talento en TI que está laborando en la industria. Si bien hay iniciativas en diferentes organizaciones, cámaras y asociaciones enfocadas al desarrollo de talento (por ejemplo: ANIEI, CONAIC, CONOCER, NYCE, CANIETI, AMITI, MexicoFIRST), hace falta revisar y redefinir los perfiles básicos profesionales, reforzando el conocimiento sobre los aspectos de calidad de procesos, productos y servicios, aspectos técnicos y "*soft skills*" requeridos en los diferentes roles de la ingeniería de software.

Difundir del valor de trabajar con centros certificados. Es importante difundir las características de los modelos de calidad y sus beneficios para los organismos demandantes, y a la vez generar criterios de evaluación y selección de sus proveedores. Debemos contar con nuevas métricas estandarizadas transversales a los modelos de calidad

que permitan criterios objetivos y confiables de selección, y generar modelos de desarrollo de proveedores que contribuyan al fortalecimiento de la cadena de valor y a ampliar el universo de proveedores de calidad suprema y alto desempeño. Esta ruta pretende formular y difundir un modelo de contratación de proveedores que se apoye en los modelos de calidad, y que cubra los procesos de aseguramiento de calidad, selección y gestión.

Incorporar métricas de calidad. Para que las métricas de calidad sean significativas, es importante que éstas sean básicas, transversales y trascendentes: a) básicas por estar alineadas a estándares probados internacionalmente; b) transversales porque sirven todos los actores de la industria de software; y c) trascendentes, porque permiten hacer comparaciones a lo largo del tiempo y entre diferentes modelos, prácticas y tecnologías.

Institucionalizar los esfuerzos vinculados con la calidad. Los diferentes esfuerzos realizados para el desarrollo de la industria de TI relacionados con la calidad del software y del talento en esta área, no están coordinados desde un punto de vista técnico, ni tampoco sus resultados son difundidos desde un solo punto de integración y comunicación. Con apoyo del programa PROSOFT, en años pasados se desarrollaron varios proyectos y estudios; sin embargo, muchos de estos esfuerzos fueron desarrollados de manera separada sin una evidente coordinación técnica. Esta ruta propone crear un organismo independiente, el Instituto Mexicano de Ingeniería de Software (IMIS)

para que coordine, promueva, vigile, evalúe y sancione los esfuerzos relacionados con la calidad de profesionales, empresas, productos y servicios de TI, apoyando el control y cumplimiento de las metas de PROSOFT 3.0.

3.5 Conclusiones

La industria de software es una industria basada en conocimiento, tiene fuerte impacto en el bienestar de la sociedad, no requiere de grandes inversiones y su impacto ambiental es mínimo comparando con otras industrias. Su breve resumen histórico, presentado en este capítulo, muestra el importante desarrollo que se ha logrado en México gracias a las políticas públicas y el esfuerzo de los emprendedores, académicos y diversos gremios profesionales. Aprovechando el reciente cambio de gobierno, sería recomendable revisar la situación actual de la industria de software y, con base en la propuesta del Mapa de rutas hacia 2024, proponer estrategias basadas en la sinergia de la sociedad y del gobierno. La tan anhelada transformación del país en gran medida tendrá que basarse en el uso extensivo de la TI y de los sistemas de software, sin ello va a ser difícil de lograrla.

Referencias

1. Oktaba, H., Alquicira Esquivel, C., Ramos, A. S., Martínez Martínez, A., Quintanilla Ozorio, G., Ruvalcaba López, M., López Lira Hinojo, F., Rivera López, M. E., Orozco Mendoza, M. J., Fernández Ordoñez, Y. y Flores Lemus, M. A. (2003). *Modelo de Procesos para la Industria de Software. Secretaría de Economía de México*.

2. Oktaba, H., Alquicira Esquivel, C., Ramos, A. S., Palacios Elizalde, J., Pérez Escobar, C. J. y López Lira Hinojo, F. (2004). *Método de Evaluación de Procesos para la Industria de Software*. Secretaría de Economía de México.

3. Oktaba, H. & Piattini, M. (2008) *Process Improvement for Small and Medium Enterprises: Techniques and Case Studies*. IGI Global.

4. Oktaba, H., Piattini, M., Pino, F.J., Orozco, M.J. y Alquicira, C. (2009). *Competisoft, Mejora de Procesos Software para Pequeñas y Medianas Empresas y Proyectos*. Ra-Ma.

5. Select. (2015). *Estudio de Retorno de Inversión (ROI), a partir de la Implementación de un Estándar, Norma o Disciplina de Calidad en Empresas de Desarrollo de Software y/o Prestadoras de Servicios de TI*. 4to. Entregable, Select, Ciudad de México, Julio 2015.

Capítulo 4.
Investigación en el área de Ingeniería de Requisitos

Brenda Leticia Flores Ríos, *Universidad Autónoma de Baja California,*

Jorge Rafael Aguilar Cisneros, *Universidad Popular del Estado de Puebla,*

Sodel Vázquez Reyes, *Universidad Autónoma de Zacatecas.*

4.1 Introducción

En este capítulo se presentan algunas de las investigaciones relacionadas con la disciplina de Ingeniería de Requisitos (IR) que se han reportado, exclusivamente en repositorios on-line, durante el periodo de 2002 al 2018, por practicantes e investigadores de universidades o centros de investigación mexicanas.

La metodología utilizada tiene un enfoque cuantitativo y se compone de dos etapas: En la primera etapa se identificaron y recolectaron las investigaciones, y en una segunda etapa se analizaron e interpretaron los resultados. Las cuatro variables consideradas fueron: (1) número de contribuciones, (2) asociación de las fases del proceso de IR contra las instituciones de adscripción de los autores, (3) instituciones de adscripción de los autores y (4) colaboración entre instituciones mexicanas y del extranjero.

Número de contribuciones: Se cuantificó el número de trabajos de investigación y artículos publicados en congresos o revistas nacionales e internacionales en idioma español o inglés, durante el periodo de 2002 a 2018.

Asociación de las fases del proceso de IR contra las instituciones de adscripción de los autores: Las investigaciones se estructuraron tomando como referencia las fases definidas en el proceso de IR (Wiegers, 2013) y la contribución que brindan ante una problemática o dificultad expuesta por la disciplina.

Institución de adscripción de los autores: Se identificaron los nombres de las instituciones, centros o universidades a las cuales están adscritos tanto el autor principal como el resto de los autores.

Colaboración entre instituciones mexicanas y del extranjero: Esta variable está asociada para trazar las colaboraciones de las instituciones mexicanas y/o la colaboración con las de procedencia extranjera.

4.2 Marco referencial

En esta sección se presentan algunas características deseables que un ingeniero de requisitos debería tener. Posteriormente, se enumeran atributos de calidad que los requisitos deben poseer.

4.2.1 Perfil y Responsabilidades

Un Ingeniero de Requisitos es una persona con habilidades de comunicación, trabajo en equipo, empatía, capacidad de análisis, sabe escuchar, entiende de los detalles, se le facilita la escritura. Es importante señalar que al ingeniero de requisitos también se le llama: Analista de requerimientos, Analista de sistemas de negocios, Arquitecto funcional, entre otros nombres.

La responsabilidad de un ingeniero de requisitos es la de hablar con el(los) cliente(s), con el fin de identificar sus requerimientos y redactarlos de manera precisa en un documento llamado: Documento de especificación de requisitos. Al redactarlo, será conveniente que tome en consideración los atributos de calidad de los requisitos.

4.2.2 Atributos de calidad de requisitos

Para que un requisito cumpla con los criterios de, éste de debe ser: necesario, claro, verificable y alcanzable (Hooks, 1993). Es necesario siempre y cuando, su omisión impacte de manera grave en el sistema, por lo que el ingeniero de requisitos se puede hacer la pregunta: ¿Qué es lo peor que podría pasar si el requisitos no se incluye?. El atributo "claro", se refiere a que el requisitos debe expresar una sola idea, esta debe ser concisa y simple con el fin de que el requisitos no sea mal interpretado. Por otro lado, un requisito es verificable si se le puede asignar un criterio de aceptación. Por último, un requisito es alcanzable,

si técnicamente es factible, además de que, por presupuesto, calendario y restricciones, puede ser considerado en el sistema.

Existen otros autores (Turk, 2005) que definen un buen requisito como aquel que presenta los siguientes atributos: uso de voz activa, uso de buena gramática, único, claro, entendible, no ambiguo, no contiene conectores como "y", "o", "también" entre otros; no contiene "etc", usan "deberá", no contiene términos que implican posibilidad: "podría", "quizá", "probablemente; no se usa palabras como: "excepto", "a menos que"; no existen términos ambiguos como: "no más grande que", "no menos que"; se evita el uso de términos indefinidos como: "máximo", "mínimo", "flexible", "amigable", "eficiente", "simple", "adecuado", entre otros; adicionalmente, no se debe presentar un diseño como si fuera un requisito, no se deben escribir requisitos contradictorios.

4.3. Proceso de Ingeniería de Requisitos

Según Sommerville, la IR se define como el proceso de desarrollar una especificación de software. Ésta representa las necesidades o requerimientos del usuario/cliente hacia los desarrolladores del sistema (Sommerville, 2005). Adicionalmente, indican las condiciones, atributos y propiedades que deben estar presentes en el sistema que se desarrollará con el fin de resolver un problema o alcanzar un objetivo.

La Figura 4.1 muestra cómo el proceso de IR se divide en las fases de: (1) Desarrollo de requisitos y (2) Administración de requisitos.

La primera fase se subdivide en las subfases: (1) Obtención, (2) Análisis, (3) Especificación y (4) Validación de requisitos. Ambas fases, al igual que la mayoría de las actividades de la Ingeniería de software, pueden ser iterativas, involucrar a varios roles con diferentes antecedentes, conocimientos y experiencias y requerir análisis más complejos o verificaciones exhaustivas y diversas (Terstine, 2015). Por ejemplo, el desarrollar cada una de las actividades de ambas fases, le permitirá al ingeniero de requisitos, incrementar la probabilidad de generar un documento de visión y alcance, un documento con la especificación de requisitos de software, completos y correctos. Adicionalmente, le permitirá, administrar las actividades de manera adecuada.

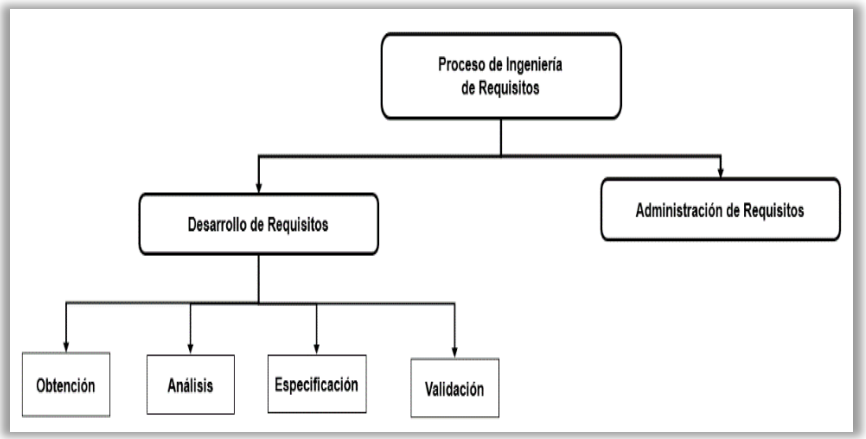

Figura 4.1. Proceso de Ingeniería de Requisitos. (Wiegers, 2013)

Cada una de las fases mencionadas del proceso de IR se han utilizado como referencia para la Ingeniería Web (Aguilar et al., 2016b). Específicamente, se ha investigado, por medio de una revisión sistemática de literatura, si la terminología es común para ambas disciplinas, cuáles son los métodos y técnicas de IR que se han adaptado para su utilización en Ingeniería Web y si existen herramientas que brinden cobertura tanto a las actividades de IR como al proceso de desarrollo de software o publicación sobre Web (Aguilar et al., 2016b).

Como se visualiza en la Figura 4.1, el proceso de IR está formado por las fases de Desarrollo de Requisitos y la Administración (Gestión) de Requisitos. En las siguientes secciones se describen cada una de dichas fases, identificando las respectivas contribuciones.

4.3.1 Desarrollo de requisitos

A continuación, se describirán las subfases que corresponden al Desarrollo de requisitos.

Obtención de requisitos. El inicio del desarrollo de un sistema de software tiene que ver con la obtención de requisitos de alto nivel. Durante esta actividad se deberá comprender el problema que el cliente necesita resolver mediante el sistema de software solicitado. En esta subfase, se deberá identificar y definir el(los) requisito(s) del negocio. Éste contiene el objetivo que expresa, por qué el sistema es necesario o en qué beneficiará al cliente. En esta subfase, se deberá acordar y lograr un completo involucramiento del cliente, con el fin de que el sistema

sea exitoso. Adicionalmente, se deberán identificar las funciones que el usuario realizará en el sistema. En la IR las funciones que se realizan con el sistema, se le conocen como Requisitos Funcionales (RF) y a las características de calidad de los sistemas de software se le conocen como Requisitos No Funcionales (RNF). En esta subfase se deberán identificar tanto RF como RNF.

El trabajo de investigación que aborda esta subfase del proceso de IR bajo un enfoque de metas organizacionales es el de Estrada et al. (2002). Los autores especifican que dichas metas son una buena base para establecer la relación entre los objetivos perseguidos por el negocio y los requisitos del sistema de información a desarrollar, debido a que los RF y RNF deben corresponderse con tareas que se desean desempeñar dentro de un proceso de negocios. Los procesos de negocio a su vez, permiten el cumplimiento o satisfacción de alguna o algunas de las metas del negocio. De esta forma, Estrada et al. (2002) presentan una propuesta para la obtención de requisitos de software a partir de modelos de negocios basado en metas que cada actor define para cumplir, a su vez con las metas generales del negocio. Se presentan los pasos necesarios para traducir el modelo de negocios en una especificación de casos de uso compatible con UML, y sus respectivos escenarios donde se describe la funcionalidad esperada del sistema de software.

Investigadores de la Universidad Tecnológica de la Mixteca (Fernández y Fernández et al., 2003) en su programa de maestría en

Ingeniería de software realizaron un proyecto de evolución de aplicaciones para el servicio de educación a distancia y virtual utilizando técnicas de obtención de requisitos, modelado de casos de uso con *Unified Modeling Language* (UML) y herramientas de administración de requisitos.

Buitrón et al., (2018) se centra en establecer las rutas de los diferentes flujos de transformación del conocimiento que se genera en esta fase de obtención de RNF al integrar diferentes componentes de Gestión de Conocimiento, una de ellos es el modelo conceptual del núcleo TCER.

Aguilar et al., (2015) enfatiza que un factor crítico para el éxito de un software es la subfase de obtención de requisitos porque ahí es donde se debe entender el problema que el cliente necesita resolver, identificar y definir el objetivo de negocio. Todo lo anterior, acordado con el cliente para que el software sea exitoso; es decir, proporciones valor y sea usado. Concluyen que no importa las técnicas y herramientas utilizadas en la subfase de obtención de requisitos mientras logres identificar RF y RNF.

Análisis de requisitos. En esta subfase se deberán analizar los requisitos de alto nivel que se obtuvieron en la subfase anterior (obtención de requisitos) y subdividir cada uno de ellos, en los RF y RNF que sean necesarios (Ver Figura 4.2). Es conveniente aplicar alguna estrategia para llevar a cabo el análisis, una alternativa sería la construcción de prototipos. Adicionalmente, es de utilidad asignar

prioridad a los requisitos identificados. La prioridad asignada a ellos, ayuda al administrador del proyecto en las actividades de: planeación de las entregas del sistema, definición de compromisos, asignación de recursos, entre otros.

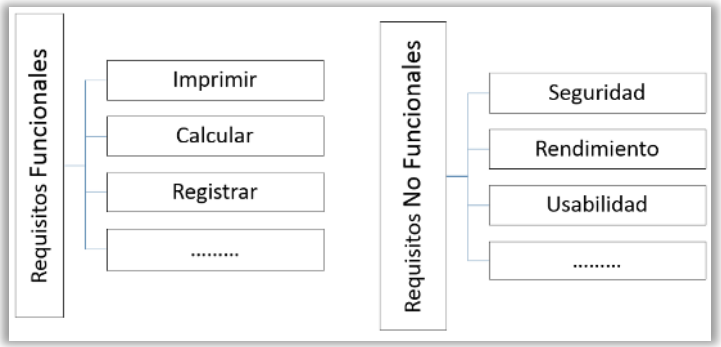

Figura 4.2. Ejemplos de Requisitos Funcionales (RF) y No Funcionales (RNF)

Existen diferentes propuestas de asignar prioridades o clasificar requisitos, por ejemplo, una de ellas (Sommerville, 1997), clasifica los requisitos como: Esenciales, útiles y deseables. Los requisitos esenciales deben ser incluidos en el sistema, los requisitos útiles, son aquellos que en caso que no se consideren, el sistema sería menos efectivo y, los requisitos deseables harán que el sistema sea más atractivo para el usuario, pero no pertenecen a la actividad principal del sistema de software.

Con respecto a los artículos identificados, existen algunos trabajos que abordaron esta subfase del proceso de IR, entre los que se encuentran la Universidad Autónoma de Sinaloa donde analizaron la

aplicación de técnicas y herramientas de la disciplina de IR en algunas empresas de su Estado (Aguilar et al., 2016). Adicionalmente, Zamudio et al. (2017), presentan una propuesta de clasificación de RF y RNF mediante técnicas de inteligencia artificial.

Especificación de requisitos. La subfase de especificación de requisitos, el objetivo es escribirlos de una manera que puedan ser entendidos de una manera clara y precisa, por lo que se deberá definir algún estilo homogéneo de redactarlos. Se deberán definir o considerar algunos atributos de calidad como (Turk, 2005): Necesario, correcto, no ambiguo, alcanzable, organizado, medible, entre otros aspectos.

Algunos problemas que dan como resultado requisitos de mala calidad son, entre otros: (1) Escribir la implementación de un sistema (¿Cómo?) en lugar de la necesidad (¿Qué?); (2) Usar términos de manera incorrecta; (3) Uso de la gramática de manera incorrecta; (4) Sobre-especificar; (4) Hacer suposiciones; (6) Utilizar términos no homogéneos; y (7) Escribir objetivos en lugar de requisitos

En esta subfase se debe desarrollar un documento que contenga los requisitos del sistema. Algunas propuestas para documentar requisitos se pueden encontrar en algunos estándares de IEEE 1233 (IEEE, 1998), estándar 830 (IEEE, 1998) o la plantilla creada por Karl Wiegers. En esta etapa, se deberá utilizar una herramienta automática para la escritura de requisitos. Actualmente, existen herramientas de tipo comerciales y de uso libre.

Con respecto a los artículos analizados, existen algunos trabajos que abordaron esta subfase. Estrada et al. (2002) presentaron los pasos necesarios para traducir un modelo de negocios en una especificación de casos de uso compatible con UML. Presentan una plantilla donde se muestran las responsabilidades del sistema que tiene como objetivo la validación de los datos introducidos por el usuario, aquellos casos donde los actores solicitan servicios o donde el sistema actúa como suministrador de información. Navarro-Almanza et al., (2017) presentan una propuesta para la clasificación automática de requisitos mediante *Deep Learning* (DL), con el fin de evitar la clasificación de manera manual. Otro trabajo es el de Aguilar-Cisneros y Fernández y Fernández, donde los autores muestran la manera de especificar requisitos para el sector automotriz. En este trabajo especifican que de acuerdo al estándar IEEE 26262-6, los requisitos automotrices deben ser especificados de manera semi-formal, mediante diagramas UML.

Validación de requisitos. La última subfase corresponde a la validación de requisitos. En esta fase se evalúa que los requisitos sean correctos y estén completos, con el fin de asegurar que los requisitos cumplirán las expectativas del cliente. Después de la validación, se podrá continuar con las siguientes etapas de construcción del sistema, en particular, se podrá proceder con el diseño. Esto no quiere decir que sea un proceso de construcción basado en el modelo de cascada. También en un modelo incremental se debe especificar, primero, los

requisitos y después continuar con el diseño. Regresando a la fase de Desarrollo de Requisitos el número de veces que sea necesario.

Con respecto a los artículos analizados, existen algunos trabajos que abordaron esta subfase del proceso de IR, entre los que se encuentran (Aguilar et al., 2016, Zamudio et al., 2017).

4.3.2. Administración de requisitos

La segunda fase del proceso de IR, llamada, Administración de requisitos es una actividad general que comprende una serie de actividades relacionadas a la evolución de los requisitos con el tiempo y a través de las familias de productos (Terstine, 2015). La administración de requisitos es una aproximación sistemática para la obtención, organización y documentación de los requisitos del sistema, y un proceso que establece y mantiene acuerdos entre el cliente y el equipo de desarrollo sobre los cambios de requerimientos que el cliente solicita para el sistema (Leffingwell y Widrig, 2000). Algunos autores (Aguilar et al., 2016) han contribuido en el área de la administración de peticiones de cambios en los requisitos.

Se han desarrollado las herramientas de software las cuales están orientadas a dar seguimiento a los requisitos que se van a controlar. Por ejemplo GIRMEX y ORMEX integran módulos de Administración y Configuración, Gestión de Documentos de la IR, de Trazabilidad entre artefactos o documentos de trabajo e Informes y estadísticas. Por lo tanto, las herramientas proporcionan el seguimiento

y el control de los requisitos para diversos tipos de proyectos (Vargas Pérez et al., 2010; Vargas Pérez et al., 2010b; Vargas Pérez et al., 2011; Vargas et al., 2014). Por otro lado, la herramienta de *Rational RequisitePro* permitió a los investigadores Fernández y Fernández et al., (2003) administrar toda la información de los requisitos de una forma eficiente, controlar las modificaciones y llevar el seguimiento y organización de los mismos. Aun cuando en la primera etapa se recolectaron las contribuciones en el proceso de IR por autores mexicanos, la revisión no es un estudio exhaustivo, sino una primera iteración bajo el propósito de identificar y concentrar las líneas de investigación actuales las cuales ayuden a comprender los desafíos para consolidar la disciplina de IR en nuestro país. Estos resultados corresponden a la variable 2. Asociación a las fases del proceso de IR.

4.4. Análisis de resultados

En la segunda etapa se analizan los contenidos de las contribuciones identificadas, dando respuesta a las cuatro variables definidas. La Tabla 3.1 presenta las 14 contribuciones (variable 1) y la asociación de las contribuciones con cada una de las fases del proceso de IR (variable 2).

Se identificó que los trabajos de Aguilar et al., (2016b), Zamudio et al., (2017) y Velázquez García, (2016) abarcan todo el proceso de IR, mientras que el de Vargas Pérez et al., (2010), Vargas Pérez et al., (2010b), Vargas Pérez et al., (2011), Vargas et al., (2014) y Aguilar et al., (2016) están orientados sólo a la fase de Administración

de requisitos. Las contribuciones para las subfases de la fase de Desarrollo de requisitos son muy puntuales a las actividades que en ellas se realizan.

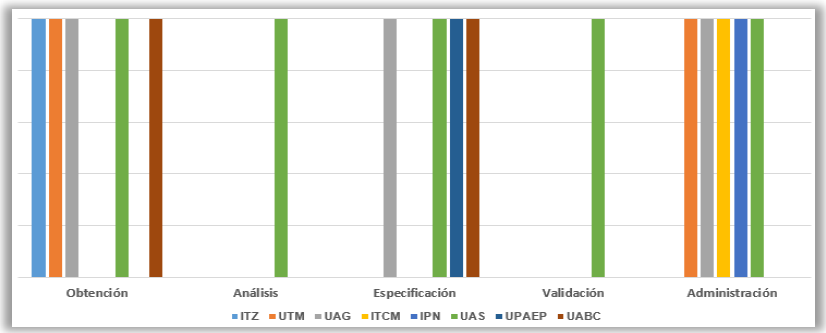

Figura 4.3. Instituciones identificadas vs. Fases de requisitos

La Figura 4.3 muestra de manera gráfica las diferentes fases del proceso de IR (eje x) con las siglas de las instituciones a las que pertenecen las contribuciones en IR de la Tabla 4.1.

Tabla 4.1. Relación de contribución con fases del proceso de IR.

Proceso de Ingeniería de Requisitos					
	Desarrollo de Requisitos				Administración de Requisitos
Contribución	**Obtención**	**Análisis**	**Especificación**	**Validación**	
Estrada et al., 2002	x		x		
Fernández y Fernández et al., 2003	x				x
- Vargas Pérez et al., 2010, 2010b, 2011 - Vargas et al., 2014 - Aguilar et al., 2016b					x
- Aguilar-Cisneros y Fernández y Fernández, 2015 - Navarro-Almanza et al., 2017			x		
Aguilar et al., 2015	x				
Velázquez García, 2016	x	x	x	x	x
Aguilar et al., 2016	x	x	x	x	x
Zamudio et al., 2017	x	x	x	x	x
Buitrón et al., 2018	x				

La Tabla 4.2 presenta los resultados de la variable 3. Institución de adscripción de los autores especificando el nombre de las instituciones, universidades o centros de investigación de procedencia de las contribuciones identificadas.

Tabla 4.2. Identificación de procedencia de las contribuciones

	Institución de Adscripción		
Contribución	**1er autor**	**2do autor**	**3ero o más**
Estrada et al. 2002	Universidad Politécnica de Valencia, CENIDET	Universidad Politécnica de Valencia, Instituto Tecnológico de Zacatepec	Universidad Politécnica de Valencia
Fernández y Fernández et al., 2003	Universidad Tecnológica de la Mixteca	Universidad Tecnológica de la Mixteca	Universidad Tecnológica de la Mixteca
- Vargas Pérez et al., 2010, 2011 - Vargas Pérez et al., 2010b	Instituto Tecnológico de Cd Madero	Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Cd de México	Instituto Politécnico Nacional, Instituto Tecnológico de Cd Madero
Salazar et al., 2012	Centro de Investigación en Matemáticas		
Vargas et al., 2014	Universidad Autónoma de Tamaulipas	Instituto Tecnológico de Cd Madero	Instituto Tecnológico de Cd Madero
Aguilar-Cisneros y Fernández y Fernández, 2015	Universidad Popular Autónoma del Estado de Puebla	Universidad Tecnológica de la Mixteca	
Aguilar et al., 2015	Universidad Autónoma de Sinaloa	Universidad Autónoma de Sinaloa	Universidad Autónoma de Sinaloa, Covenant University, ProTech I+D S.A. de C.V.
Velázquez García, 2016	Universidad Autónoma de Querétaro		
Valles Montalvo et al., 2016	Tecnológico Nacional de México	Instituto Tecnológico de Apizaco	
- Aguilar et al., 2016b - Zamudio et al., 2017	Universidad Autónoma de Sinaloa	Universidad Autónoma de Sinaloa	Universidad Autónoma de Sinaloa
Aguilar et al., 2016	Universidad Autónoma de Sinaloa	University of Alicante	University of Alicante
Navarro-Almanza et al., 2017	Universidad Autónoma de Baja California	Universidad Autónoma de Baja California	Universidad Autónoma de Baja California
Buitrón et al., 2018	Universidad del Cauca	Universidad Autónoma de Baja California	Universidad del Cauca

Para la variable 4. Colaboración entre instituciones mexicanas y del extranjero, se identificó una contribución donde el primer autor tiene doble adscripción a una universidad de España y un centro de investigación en México. Varios trabajos donde el primer autor es de una universidad de México y tiene publicaciones en coautoría con investigadores de México, España y Nigeria. Una contribución donde el primero y segundo autor es de una universidad de Colombia y el tercer autor de México. Los demás trabajos corresponden a colaboraciones entre autores de instituciones mexicanas. La Tabla 4.3 muestra la cobertura de las fases del proceso de IR en las publicaciones identificadas en colaboración entre instituciones mexicanas y del extranjero.

Tabla 4.3. Colaboración de instituciones mexicanas con grupos extranjeros en el proceso de IR

	Proceso de Ingeniería de Requisitos				
	Desarrollo de Requisitos				**Administración de requisitos**
Colaboración	Obtención	Análisis	Especificación	Validación	
México	x	x	x	x	x
España	x	x	x	x	x
Colombia	x				
Nigeria	x				

4.5. Conclusiones

La Ingeniería de Requisitos es un área de la Ingeniería de software sumamente importante, debido a que en algún momento pueden determinar el éxito o fracaso de un proyecto de desarrollo de software. Por lo que se requiere aumentar el universo de publicaciones en esta disciplina. Debido a que esta disciplina, representa un área de oportunidad para los investigadores tanto del área académica como industrial de México.

Cómo se puede observar en la Tabla 4.1, existen algunas fases del proceso de IR con pocas aportaciones por parte de los investigadores mexicanos. En particular, las subfases de análisis y validación de requisitos. Con esto, se resaltan las áreas de oportunidad o líneas de investigación que requieren de más esfuerzos o aportación entre las instituciones de los diferentes Estados de la República Mexicana o del extranjero para contribuir en la resolución de problemas específicos al proceso de IP.

Referencias

1. Aguilar, J. A., Garrigós, I. & Mazón, J.-N. (2016). Requirements Engineering in the Development Process of Web Systems: A Systematic Literature Review. *Acta Polytechnica Hungarica*. Vol. 13 (3). pp. 61-80.

2. Aguilar, J.A., Tripp, C., Zaldivar, A., García, V. & Zurita, C.E. (2016b) Evaluating a Requirements Change Request in a Goal-oriented Requirements Engineering Model. IEEE *Latin America Transactions*, vol. 14 (5), 2411-2417. DOI: 10.1109/TLA.2016.7530439.

3. Aguilar-Cisneros, J. R. & Fernández y Fernández, C. A. (2015). Especificación de requerimientos para el Desarrollo de software Automotriz en México. *Revista Latinoamericana de Ingeniería de Software*, Vol. 3 (6). pp. 250-258,

4. Aguilar, J. A., Zaldívar-Colado, A., Tripp-Barba, C., Misra, S., Bernal, R. & Ocegueda, A. (2015). An Analysis of Techniques and Tools for Requirements Elicitation in Model-Driven Web Engineering Methods. *Proceedings of International Conference on Computational Science and Its Applications*. (518-527). Springer, Cham.

5. Buitrón, S. L., Flores-Rios, B. L., & Pino, F. J.. (2018). Elicitación de requisitos no funcionales basada en la gestión de conocimiento de los stakeholders. Ingeniare. *Revista Chilena de ingeniería*, Vol. 26 (1), 142-156. https://dx.doi.org/10.4067/S0718-33052018000100142

6. Estrada, H., Martínez, A., Pastor, O., & Sánchez, J. (2002). Generación de Especificaciones de Requisitos de Software a partir de Modelos de Negocios: un enfoque basado en metas. *In V Workshop de Engenharia de Requisitos WER*. Vol. 2 (11). pp. 177 - 193.

7. Fernández y Fernández, C., Mendoza Vásquez, A., Martínez Guzmán, D., Mendoza Ortiz, E., & Sumano Ortega, P. (2003). Ingeniería de Requerimientos aplicada a la Universidad Virtual de la UTM. *In XVI Congreso Nacional y II Congreso Internacional de Informática y Computación*. ANIEI, México, Vol. 22.

8. Hooks, Ivy. (1993). Writing Good Requirements, A requirements Working Group Information Report. *Proceedings of the Third International Symposium of the INCOSE*, Vol 2.

9. IEEE Computer Society. Software Engineering Standards Committee, & IEEE-SA Standards Board. (1998). *Edition, IEEE Recommended Practice for Software Requirements Specifications*. Institute of Electrical and Electronics Engineers.

10. IEEE standards Software Engineering, (1999). *IEEE 1233*. Volume One.

11. IEEE 1233 Computer Society. Software Engineering Technical Committee. (1998). *IEEE guide for developing system requirements specifications*. IEEE.

12. Leffingwell, D., & Widrig, D. (2000). *Managing software requirements: a unified approach*. Addison-Wesley Professional.

13. Navarro-Almanza, R., Juárez-Ramírez R. & Licea, G. (2017). Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification, *5th International Conference in Software Engineering Research and Innovation*

(CONISOFT). Mérida, México. 116-120. DOI: 10.1109/CONISOFT.2017.00021

14. Salazar, M. G., Mitre, H. A., Olalde, C. L., & Sánchez, J. L. G. (2012). Proposal of Game Design Document from Software Engineering Requirements Perspective. 17th *International Conference on Computer Games (CGAMES 2012), IEEE*., pp. 81-85.

15. Sommerville, I. (2005). *Ingeniería del software*. Pearson Ed. Madrid: Pearson Educación S.A., pp. 1–677.

16. Sommerville, I., Sawyer, P. (1997). *Requirements Engineering: A good practice guide*. WILEY, England.

17. Terstine, M. El progreso de la investigación en Ingeniería de Requisitos. *Revista Antioqueña de las Ciencias Computacionales y la Ingeniería de software (RACCIS)*. Vol. 5 (1). 2015. pp. 18-24.

18. Turk, W., (2005). *Mission Possible...With Good Requirements. Acquisition Process Improvement*, Defense AT&L.

19. Valles Montalvo, D. A., Quintero Flores, P. M., & Nava Bautista, H. (2016). Implementación de una metodología de ingeniería de requerimientos en grandes proyectos de desarrollo de software. *Research in Computing Science*, 126, pp. 131-143.

20. Vargas Pérez, L. S., Gutiérrez Tornés, A. F., Felipe Riverón, E. M., Chavira Juárez, G., Peralta Escobar, J. & Mijares Fong, E. M. (2010). Gestor de Requerimientos para proyectos con técnicas de Ingeniería

de Requisitos. *VIII Congreso Internacional sobre Innovación y Desarrollo Tecnológico, CIINDET 2010*, IEEE Sección Morelos, IIEAt: Cuernavaca, Morelos, México.

21. Vargas-Pérez, L. S., Gutiérrez-Tornés, A. F., Felipe-Riverón, E. M., Peralta-Escobar, J. & Mijares Fong, E. M. (2010b). Organizador de requerimientos para proyectos con técnicas de competencias en Ingeniería de requisitos. *II Congreso Internacional de Gestión Tecnológica e Innovación*, GESTEC 2011.

22. Vargas Pérez, L. S., Gutiérrez Tornés, A. F., Felipe Riverón, E. M. & Peralta Escobar, J. (2011). ORMEX: sistema organizador de requerimientos para proyectos con técnicas de ingeniería de requisitos. *XXXVIII Conferencia Nacional de Ingeniería ANFEI 2011*. México.

23. Vargas,V., Vargas, L., Peralta J. & Gómez R. (2014). *Organizador de Requisitos de Proyectos Basado en los Estándares de Gestión de Proyectos.* M. Ramos., V.Aguilera., (eds.). Ciencias de la Ingeniería y Tecnología, Handbook ©ECORFAN.

24. Velázquez García, L. A. (2016). Gestión y tecnología para la ingeniería de requerimientos en servicios computacionales. *Revista Iberoamericana de las Ciencias Computacionales e Informática*, Vol. 5 (10), pp. 59-78.

25. Wiegers, Karl E. (2013). *Software Requirements*. 3rd edition. Microsoft Press.

26. Zamudio, L., Aguilar, J. A., Tripp-Barba, C., Zaldívar-Colado, A., Aguilar, P., & Zurita-Cruz, C.E. (2017). Software factories in Sinaloa: Requirements engineering application priority issues, *International Conference on Computing Networking and Informatics (ICCNI),* Lagos, 2017, pp. 1-5. doi: 10.1109/ICCNI.2017.8123817

Capítulo 5.
Investigación en el área de Diseño de Software

María Karen Cortés Verdín, *Universidad Veracruzana,*

Jorge Octavio Ocharán Hernández, *Universidad Veracruzana.*

5.1 Introducción

En el Estándar ISO/IEC/IEEE 24765 se define diseño como "el empleo de principios científicos, información técnica, e imaginación en la definición de un sistema de software para realizar funciones pre-definidas dentro de la máxima economía y eficiencia" (ISO/IEC & IEEE, 2017). El diseño es la etapa en la que, con base en los requisitos previamente establecido, se determina la estructura del software. Se definen los elementos o componentes que lo integrarán, la manera en que estos componentes se integrarán para formar el sistema de software. Un aspecto importante del diseño de software es que en esta etapa se sientan las bases para la calidad del software.

Para el abordaje de este capítulo se decidió seguir la estructura que indica el SWEBOK (*Software Engineering Body of Knowledge*) en su versión 3.0 (*IEEE Computer Society*, Bourque, & Fairley, 2014). El SWEBOK denomina área de conocimiento (*Knowledge Area*) al área de Diseño de Software y la divide en subtópicos. En la revisión

realizada de los trabajos realizados por investigadores mexicanos en esta área, no se encontraron trabajos para cada uno de los subtópicos del SWEBOK.

Para la búsqueda de los trabajos incluidos en este capítulo se realizó en primer lugar una búsqueda automatizada utilizando la base de datos de resúmenes Scopus (www.scopus.com) la cual incluye en su índice las principales bases de datos en donde se publican trabajos de Ingeniería de Software. Para facilitar el filtrado de los trabajos, se limitaron los resultados por afiliación, seleccionando únicamente a los institutos de investigación e instituciones de educación superior con sede en México. Posteriormente se realizó *snowballing* —Estrategia de búsqueda en la que se hace un rastreo de las referencias y citaciones de trabajos conocidos— de los trabajos identificados en el área de diseño de software con el objetivo de complementar los resultados de la búsqueda automatizada.

En lo que respecta a los cuerpos académicos (CAs) del país y su investigación en el área de Diseño de Software, se consultó el portal de la SEP. En la primera búsqueda se empleó la cadena "inge%software" obteniéndose un listado de 77 Cuerpos Académicos (CA). Se hizo una segunda búsqueda empleando la cadena "%software%" en la que se obtuvieron 209 CA. Se decidió entonces descartar el primer listado y trabajar con el segundo. Del segundo listado, se procedió a eliminar aquellos CA que en sus Líneas de Generación y Aplicación del

Conocimiento (LGAC) no incluyeran “desarrollo de software” o “ingeniería de software”.

Con la lista final, se procedió entonces a realizar la búsqueda de la página correspondiente al CA en la URL indicada de la institución. Se encontró que en la URL indicada, cuando existía, se mostraba información relativa a recursos asignados de PRODEP; no del CA mismos. Se procedió entonces a buscar directamente en el portal de la institución. En casi todos los casos, no se encontró información de los CA. Cuando pudo encontrarse algo, a lo más fue un listado de CA de la institución. En el caso de una universidad pudo encontrarse un listado de los CA asociados a un centro de investigación. Este listado contenía, para cada CA, los nombres de sus integrantes con correo electrónico. Para este caso pudo comprobarse que los trabajos de investigación que uno de los integrantes ya habían sido considerados en el área de Diseño de Software para el presente capítulo. Para otras dos universidades más, pudo encontrarse la página del CA relacionado con Ingeniería de Software mismo y, de esta manera, se comprobó que sus trabajos en el área de Diseño, ya habían sido analizados e integrados al presente capítulo. En total, se intentó obtener la información de 33 CAs de diversas universidades. Para los Institutos Tecnológicos, no fue posible encontrar la información correspondiente en el portal de la institución. Sin embargo, es probable que los trabajos desarrollados en estos institutos, correspondientes al área de Diseño de Software, ya han sido revisados, analizados y presentados en este capítulo.

En cuanto a los trabajos de pregrado y posgrado que se incluyen en este capítulo, para su identificación se recurrió a los repositorios institucionales de los Centros de Investigación e Instituciones de Educación Superior que cuentan con ellos. En dichos repositorios se realizó una búsqueda automatizada con las palabras clave "diseño" y "software" lo que arrojó diferentes trabajos en el área. Se pudo comprobar la existencia de publicaciones derivadas las cuales han sido analizadas en este capítulo.

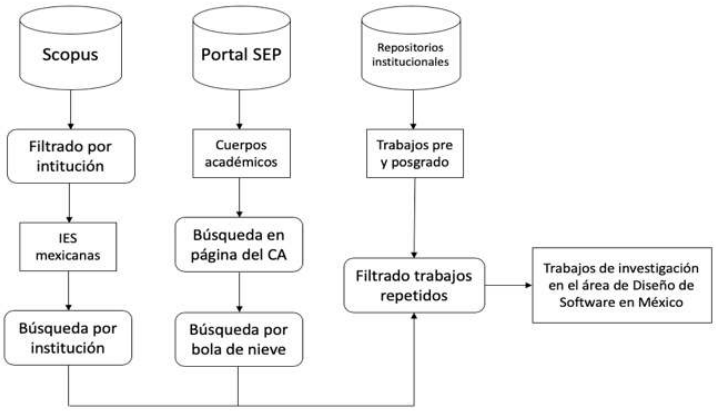

Figura 5.1. Búsqueda de trabajos de investigación en el Área de Diseño.

Finalmente, es importante comentar que de los trabajos de investigación buscados para el área de Diseño en México, sólo hubo cuatro (uno de ellos un capítulo de libro) que no pudieron conseguirse para su análisis. A continuación se presenta el análisis de los trabajos identificados divididos por tópicos y sub-tópicos.

Podemos decir que la investigación en ingeniería de software tiene como fin generar conocimiento nuevo en el ámbito de esta disciplina (y comprender mejor el ya existente) así como codificarlo de tal manera que éste pueda ser incorporado en la práctica diaria del desarrollo de software. Esto con el fin de mejorar los plazos de entrega, la ejecución del presupuesto asignado, así como mejorar la calidad del producto software a desarrollar o mantener.

5.2 Fundamentos de Diseño de Software

En este apartado se tratan los conceptos, notaciones y terminología que forman la base de la actividad de diseño de software.

5.2.1 Principios de Diseño de Software

Los principios de diseño son conceptos clave en el proceso de diseño como cohesión, acoplamiento y refinamiento suceso, por mencionar algunos. En esta subárea Acuña Cháirez (2012) hace una revisión de los principios de diseño de software propuestos por diversos autores, identifica buenas prácticas para cada principios y las aplica en el desarrollo de una biblioteca de árboles binarios de búsqueda, empleando para ello un diagrama de clases.

5.3 Cuestiones clave en el diseño

Este subtópico del diseño de software se refiere a los aspectos o cuestiones clave que deben ser considerados durante el proceso de

diseño. Podemos empezar por, en primer lugar, aquellos que se refieren a los también llamados atributos de calidad como la seguridad, el rendimiento y la facilidad de uso, por mencionar sólo algunos. Otros aspectos clave a considerar son la descomposición, organización y empaquetado de los componentes de software (IEEE Computer Society et al., 2014). Otros, finalmente, se refieren a los que se definen como Aspectos o intereses dispersos y entrelazados que dan lugar al enfoque orientado a aspectos

5.3.1 Seguridad

En su artículo "*Architectural Approches to Security: Four Case Studies*" de Cervantes et al (2016) proponen una estrategia basada en la arquitectura de software a fin de cumplir con requisitos de seguridad. Afirman que, en la mayoría de los casos, cuando de seguridad se trata, ésta se resuelve en la construcción. Sin embargo, esta estrategia puede complementarse de manera adecuada empleando tácticas arquitectónicas de seguridad. Los autores presentan cuatro estudios de caso en los cuales, las estrategias arquitectónicas se implementan mediante el uso de *frameworks* o mediante el uso de plataformas. Los autores sostienen que, los *frameworks* o plataformas, a su vez, implementan las tácticas de seguridad ya conocidas.

Velasco Elizondo, Barredo Hernández y Mitre (2014) proponen la agregación de la estimación del QoS a la composición de servicios. La estrategia de su trabajo se basa en el empleo de patrones

arquitectónicos que sustentan la integración de servicios. Su trabajo es un esfuerzo inicial hacia el ensamblaje predictivo (*predictable assembly*), que es una característica altamente deseable para enfoques de desarrollo de software basados en composición (*CBSE: Component-Based Software Engineering*).

Figueroa Gutiérrez (2015) se presenta el desarrollo de un modelo de calidad para arquitecturas de software, el cual toma como base el propuesto por Ryoo, Laplante y Kazman (2012). En el trabajo de Figueroa Gutiérrez, se identifican los principales mecanismos de seguridad y se categorizan en base a tácticas arquitectónicas relacionadas con seguridad. Posteriormente se desarrollan escenarios de prueba para su ejecución y cálculo de métricas. Más adelante, se presenta el catálogo de mecanismos de seguridad de dicho modelo (Figueroa-Gutiérrez, Cortés-Verdín, & Contreras-Vega, 2016).

5.4 Estructura del Software y Arquitectura

En este tópico se ubica lo relacionado con el diseño de los elementos que determinan la forma o estructura del software.

5.4.1 Estructuras Arquitectónicas y Puntos de Vista

El diseño de un sistema en particular puede verse desde varios puntos de vista o perspectivas, dependiendo de los intereses de los involucrados en el desarrollo de software. Estos puntos de vista también conocidos son como *viewpoints* y existen varias clasificaciones de ellos.

En este apartado se presentan los trabajos de investigación realizados en el área de Estructuras Arquitectónicas y Puntos de Vista.

La documentación de la arquitectura es un área importante dentro del tópico del tópico de Estructura de Software y Arquitectura ya que, al describir y documentar una arquitectura, se puede conocer el contexto, especificación y representación de la arquitectura de software, de sus elementos y de sus relaciones. En esta área se identifican dos trabajos que se centran en la documentación de arquitecturas orientadas a aspectos de líneas de productos de software (Ocharán-Hernández, 2009; Ocharán-Hernández & Cortes-Verdin, 2008). Posteriormente, Ramírez Mora (2016) presenta una guía para documentar arquitecturas de software que incluye el uso de ADLs (*Architecture Description Languages*) los cuales permiten, incluso, simular modelos arquitectónicos.

La incorporación del diseño de una arquitectura en métodos ágiles ha sido un tema de discusión dese hace tiempo. En la comunidad de investigadores mexicanos, se tiene el trabajo de Cortés Pichardo (2011). Este trabajo es una tesis de maestría que incluye una revisión sistemática del tema y hace una propuesta para integrar las prácticas correspondientes en métodos ágiles. Las prácticas incluyen no sólo el diseño de la arquitectura de software en métodos ágiles, sino también la documentación y evaluación de esta.

5.4.2 Estilos Arquitectónicos

Los estilos Arquitectónicos definen la estructura de alto nivel de un sistema. Existen diversos tipos de estilos arquitectónicos. La elección de uno o una combinación de ellos dependerá de los atributos de calidad que el sistema debe de cumplir. En esta Sección, se presentan los trabajos de investigación que se han realizado en Estilos Arquitectónicos.

En su subtópico de Estilos Arquitectónicos, se ubica a Ortega Arjona y Fernández (2008) introducen el estilo arquitectónico seguro Blackboard. Esta versión segura del patrón Blackboard al que se añaden componentes de seguridad y control para garantizar el manejo, transformación y movimiento de datos.

También en Estilos Arquitectónicos, se ubica el trabajo de Alor Hernández et al (2010) que consiste en una arquitectura híbrida que combina los estilos de arquitectura orientada a servicios (SOA, por sus siglas en inglés) y arquitectura dirigida por eventos (EDA por sus siglas en inglés) a fin de poder crear una arquitectura para cadenas de suministros. En el dominio de las cadenas de suministros es necesario responder a eventos que necesitan ser atendidos por alguno de los sistemas de la cadena mediante un gestor de eventos. El trabajo propone un middleware basado en componentes que combina características de SOA y EDA.

5.4.3 Decisiones de Diseño Arquitectónico

El proceso de diseño consiste en la toma de decisiones de diseño, referidas éstas, principalmente, a los atributos de calidad: como cumplirlos y como resolver conflictos entre ellos. En este apartado, se trata de las contribuciones realizadas en México en esta área.

Comentado ya en el subtópico de Estilos Arquitectónicos, se encuentra el trabajo de Ortega Arjona y Fernández (2008) que también se incluye en este apartado debido a que, para agregar seguridad en los aspectos de manejo, transformación y movimiento de datos proponen componentes de seguridad en el estilo arquitectónico *Blackboard*.

Cervantes, Velasco Elizondo y Kazman (2013) presentan una estrategia de diseño de arquitecturas de software que considera de manera específica el uso de *frameworks* de desarrollo. Los *frameworks* en software, consisten en código común que proporcionan funcionalidad, la cual puede ser extendida para un uso en particular. Los *frameworks* aumentan la productividad y reducen la carga cognitiva en el programador. La estrategia presentada por los autores incorpora la selección del *framework* dentro del proceso de ADD conectando los drivers arquitectónicos con la selección del *frameworks*. Los autores sostienen que los *frameworks* deben ser considerados entidades de primera clase ya que sus servicios implementan tácticas arquitectónicas.

Similar al trabajo de 2008, Fernández y Ortega-Arjona (2009) presentan el patrón *Secure Pipes and Filters*, que es, como ellos lo mencionan "una versión segura del patrón *Pipes and Filters*", en el que se añaden controles de seguridad en cada etapa de procesamiento. Estos controles sólo permiten el procesamiento previamente autorizado, así como aseguran el movimiento de los datos desde y hacia los filtros. Este trabajo tiene sus orígenes en el libro de *Security Patterns: Integrating Security and Systems Engineering* (Schumacher, Fernandez, Hybertson, & Buschmann, 2005).

En 2010, en el libro *Patterns for Parallel Software Design* (Ortega-Arjona, 2010) se presenta la manera de diseñar software paralelo con base en patrones arquitectónicos. Posteriormente, Matamoros, Savage y Ortega Arjona (2015) presenta la comparación de dos estilos arquitectónicos (*Blackboard y Peer-to-Peer*) con relación a tiempo de respuesta (performance) y el costo de construcción y modificación, en el contexto de robots de servicio móviles. Los resultados indican que, no hay diferencia en el caso del tiempo de respuesta mientras que, se reduce o queda igual el costo de construcción y modificación.

Por su parte Cervantes y Kazman (2016) desarrollaron un libro titulado *Designing Software Architectures: A Practical Approach*. En este libro presenta la nueva versión (3.0) de ADD (*Attribute-Driven Design*) un método de diseño de arquitecturas de software. En esta nueva versión, se cubren todo tipo de drivers arquitectónicos,

considerando también la integración del diseño de arquitecturas de software en la organización y con métodos ágiles.

El tratamiento del conocimiento arquitectónico es un área de investigación dentro del campo de las arquitecturas de software que trata de como almacenar y explotar las decisiones, supuestos, contexto, razonamientos y otros factores que determinan el diseño de una arquitectura. En esta área se encuentra el trabajo de Borrego, Morán, Palacio y Rodríguez (2016), el cual propone una ontología para conocimiento arquitectónico en el contexto de los medios electrónicos y textuales que se emplean en un ambiente de desarrollo de software ágil global (*Agile global software development*), identificando una brecha entre las metodologías ágiles y el desarrollo de software global en este contexto.

5.4.4 Familias de Programas y *Frameworks*

Las familias de programas o también llamadas líneas de productos de software sacan el máximo provecho de las similitudes en un conjunto de programas; es así como el desarrollo de familias o líneas de productos de software se logra mediante la gestión adecuada de las similitudes y variación, dando lugar a un enfoque basado en la reutilización.

En su trabajo de tesis de maestría Cabello Espinosa (2007) presenta la solución de un sistema experto para diagnóstico clínico que emplea arquitecturas orientadas a aspectos y técnicas de variabilidad

propias de líneas de productos de software. La solución sigue un enfoque de arquitectura conducida por modelos (MDA por sus siglas en inglés). Siguiendo esta línea de investigación, posteriormente, la misma investigadora, en su trabajo de tesis de doctorado Cabello Espinosa (2008) presenta un *framework* para la generación de aplicaciones siguiendo un enfoque de Líneas de Productos de Software. El *framework* se apega a MDA para construir los modelos de dominio (que incluyen una arquitectura de referencia) a partir de los cuales se generan las arquitecturas de las aplicaciones y, finalmente, se compilan las aplicaciones. En los trabajos anteriores se emplea como caso de estudio el de un sistema experto para el diagnóstico médico. Relacionado con el trabajo anterior, Cabello Espinosa y Ramos (2008) específicamente detalla la manera en que fue construido el baseline —conjunto de activos reutilizables que se emplean en la generación o construcción de los productos de la línea— del *framework*. Siguiendo con el dominio de los sistemas expertos para diagnóstico médico, en el trabajo de A. Gómez, Cabello y Ramos (2010) se introduce, además, el plan de producción correspondiente. En esta misma línea de investigación, Cabello, Ramos, Santana y Beristain (2014) presentan un proceso, un método y el *framework* para desarrollar una arquitectura de referencia de una línea de productos de software. Posteriormente, en (Cabello Espinosa & Ramos Salavert, 2016) se describe de manera detallada el proceso de ingeniería de aplicaciones que toma como base el *framework* propuesto. Ya para terminar con la investigación de Cabello Espinosa

en esta línea, basándose en todos los trabajos anteriores (Cabello Espinosa, Preciado Álvarez, Santana Álvarez, & Ramos Salavert, 2018) se presenta el proceso completo de generación de sistemas de sistemas expertos mediante el enfoque de Líneas de Productos ya discutido, incorporando, además, dos herramientas. La primera de las herramientas se emplea para elaborar y editar modelos de características (llamada *MODELER*); mientras que la segunda (llamada *GENERATOR*) toma los modelos de características generados con la primera herramienta y genera el código fuente y ejecutable del sistema experto.

En 2009 Cortés Verdín realizó un trabajo de tesis doctoral en el que propone un método para obtener una arquitectura orientada a aspectos para Líneas de Productos de Software (*Aspect-Oriented Product Line Architecture, AOPLA)* (Cortés Verdín, 2009). Este trabajo propone seguir, desde etapas tempranas del desarrollo de la PLA (*Product Line Architecture*), un enfoque orientado a intereses (*Concerns*) para su adecuado manejo e incorporación en la PLA ya sea, como aspectos o decisiones arquitectónicas. Entre estos intereses se encuentran considerados no sólo los requerimientos funcionales sino también los atributos de calidad. Este trabajo fue reportado posteriormente en otros dos artículos (Cortés Verdín, Lemus Olalde, van der Hoek, & Fernández Peña, 2009, 2010). A partir de este trabajo y debido a la necesidad de contar con una manera de documentar arquitecturas orientadas a aspectos para líneas de productos de

software, Ocharán Hernández (2009) propone una manera de documentar así como un perfil UML; este trabajo fue presentado previamente en otro artículo (Ocharán-Hernández & Cortes-Verdin, 2008). Relacionado con el trabajo de Cortés Verdín (2009), se detalla el modelo de calidad para línea se productos de software que soporta dicha propuesta (Cortés Verdín, 2011). Este modelo incorpora escenarios, definición de atributos de calidad y métricas correspondientes, así como los patrones arquitectónicos y las tácticas arquitectónicas conforme a la variabilidad de la línea de productos de software.

Otros trabajos que resaltan en el área o subtópico de Familias de programas o Líneas de Productos de Software son los de Díaz Velásquez (2016) y Hernández López (2016). El primer trabajo es una tesis de maestría que trata del desarrollo de una línea de productos de software para el área de la domótica —Técnicas empleadas para automatizar una vivienda integrando tecnologías de seguridad, energía y comunicaciones— que emplea el enfoque composicional de la programación orientada a deltas —diferencias de un producto a otro; mientras que, en el segundo, que también es un trabajo de tesis de maestría, trata del desarrollo de una línea de productos de software para el área de la inmótica —Conjunto de tecnologías aplicadas a la automatización inteligente de edificios no dedicados a la vivienda— empleando MDA, la orientación a aspectos y Scala —Lenguaje de programación multiparadigma. Asociado a este último trabajo, se

presenta en el trabajo de Hernández López, Juárez Martínez, Muñoz Contreras y Cortés Verdín (2015). Más adelante se presentó la automatización para la generación de productos derivados de la línea de productos de software dentro del dominio de la inmótica (Hernández-López & Juárez-Martínez, 2016). Posteriormente, Hernández López, Juárez Martínez e Ixmatlahua Díaz (2018) presentan el proceso de generación de la línea de productos de software, detallado el proceso basado en MDA, la correspondiente selección de características para la derivación de productos y, por último, la configuración de los mismos.

Se presenta un *framewok* para la derivación de arquitecturas de productos basada en ontologías (Duran-Limon, Garcia-Rios, Castillo-Barrera, & Capilla, 2015). Los autores proponen el uso de ontologías para derivar directamente de la PLA sin necesidad de configurar y personalizar (lo cual es propenso a errores), manualmente, los productos de la línea. El *framework* considera una ontología que contiene información de la PLA y del modelo de características, mientras que el motor de razonamiento correspondiente infiere los elementos arquitectónicos que cumplen la selección de características.

Ruiz, Durán Limón y Parlavantzas (2016) proponen un enfoque basado en Líneas de Productos de Software para solucionar el problema de las múltiples configuraciones de aplicaciones de IaaS en la nube. Los autores identifican dos inconvenientes importantes en la adaptación de aplicaciones en la nube: (1) son dependientes de la plataforma y (2) son estrategias complejas y propensas a errores debido a que son métodos

manuales. Es así, que en este trabajo se presentan los resultados preliminares de *framework* que sigue un enfoque LPS basado en desarrollo conducido por modelos (MDD por sus siglas en inglés) para la adaptación de este tipo de aplicaciones. Este enfoque incluye: (1) una capa para la interacción con el usuario, la cual ayuda a definir las configuración y reconfiguraciones de una aplicación de manera sencilla; y (2) una capa núcleo de LPS que incorpora, en primer lugar, un manejador de LPS que básicamente es un modelo o árbol de características de LPS general, a partir del cual se seleccionan la configuración de una aplicación en específico; un manejador de modelo que realiza las actividades de gestión del modelo; el manejador de adaptación que se encarga de gestionar las adaptaciones definidas por el usuario para las configuraciones de la nube y la capa de comunicación con la nube que, como su nombre lo indica maneja las comunicaciones con la plataforma de IaaS.

Por último, destaca la reciente inclusión de los enfoques de Multi Líneas de Productos a las Líneas de Productos de Software. Trujillo Tzanahua, Juárez Martínez, Aguilar Laserre y Cortés Verdín (2018) se explora la aplicación de estrategias de dicha área en el desarrollo de software así como se determinan los retos y problemas para las multi líneas de productos de software.

5.5 Análisis y Evaluación de la Calidad del Diseño de Software

Como una estrategia para la detección de defectos en el software existen diferentes enfoques para el análisis y evaluación de la calidad del software. Con este fin, se han propuesto diferentes técnicas que estudian la calidad del diseño y desarrollado herramientas de soporte asociadas con dichas técnicas. Entre las principales técnicas se pueden mencionar las revisiones de diseño, formales e informales, el análisis estático, la simulación y el uso de prototipos. A continuación, se presentan las contribuciones realizadas en este tópico en México.

Uno de los primeros trabajos que se analizaron fue el de López Martín (2008), en él se reporta un experimento en el que se investiga el efecto de las revisiones de diseño y de código en el número de defectos en un proceso de desarrollo personal (PSP) con lo que se contribuye en la generación de evidencia sobre la efectividad de este tipo de prácticas en la disminución de defectos. Por otro lado, en el trabajo de maestría de Núñez Reyna (2009) se propone un método de evaluación de diseño arquitectónico que se denominó como AEM y que está inspirado en ATAM (Kazman, Klein, & Clements, 2000). En esta propuesta, a diferencia de ATAM, la evaluación es realizada por evaluadores internos y tiene como principal objetivo asegurar que se satisfagan los conductores arquitectónicos (*architectural drivers*) identificados en fases anteriores de su propuesta completa la cual se describe más adelante en la Sección Métodos y Estrategias de Diseño de Software.

Existen diferentes atributos de calidad deseables en un producto de software. Un análisis del diseño permitirá conocer el grado en el que éste inhibirá o promoverán que se alcancen los requisitos de calidad del sistema. En este sentido y hablando específicamente de la rendimiento, disponibilidad y facilidad de escalar existen trabajos que inician con un análisis de los patrones de diseño que abordan el problema desde una orientada específicamente para la estimación de la composición de servicios (Velasco-Elizondo et al., 2014) y que han servido como base para el desarrollo de propuestas que buscan mejorar el desarrollo de sistemas basados en la composición y disminuir sus problemas asociados, como la selección de componentes, la detección de incompatibilidad entre componentes y la estimación de la calidad del servicio de la composición resultante, en aspectos de usabilidad (Jarvio Hernández, 2015).

5.6 Notaciones de diseño de software

El diseño de un software puede ser representado mediante diferentes notaciones las cuales auxilian en la descripción tanto de la estructura como el comportamiento de un sistema de software en específico. Las notaciones son conjuntos de símbolos se han adoptan y en algunos casos especializado para representar el diseño de alto nivel (la arquitectura) así como el diseño detallado del software y normalmente se encuentran asociadas a métodos de diseño específicos.

La representación del diseño del software se puede realizar utilizando múltiples notaciones, las cuales, pueden ser gráficas, textuales o una combinación de ambas. En el caso de la descripción de la estructura del software es posible encontrar artefactos como los lenguajes de descripción de arquitecturas, las tarjetas de clase-responsabilidad-colaborador, los diagramas entidad-relación, los lenguajes de descripción de interfaces y diagramas específicos de UML (clases, objetos, componentes, paquetes, estructura compuesta y despliegue). Por otro lado, para la descripción del comportamiento del software se pueden utilizar notaciones como los diagramas de flujo de datos, las tablas y diagramas de decisión, los diagramas de flujo, los lenguajes de especificación formales y, al igual que con la vista estructural, algunos diagramas de UML específicos, tal es el caso de los diagramas de actividad, comunicación, secuencia, interacción y máquina de estados.

En este ámbito, investigadores mexicanos han abonado principalmente en el desarrollo de lenguajes de descripción de arquitecturas a finales de la década de los años 2000. Es aquí cuando se desarrolla el *Wired Application Description Language* o WADL por sus siglas en inglés (Cervantes, Donsez, & Touseau, 2008) el cual se enfoca en la descripción de aplicaciones basadas en sensores dinámicos. De una forma declarativa, WADL permite la introducción y eliminación de productores y consumidores en tiempo de ejecución, da soporte a la conexión (*binding*) y la activación y desactivación de la aplicación de

acuerdo con la presencia de productores y consumidores. WADL logra de esta forma que el intérprete *WireAdminBinder* puede crear y destruir los conectores entre los productores y los consumidores a medida que se introducen o eliminan dinámicamente del entorno de ejecución sobre el *framework* OSGi (www.osgi.org/developer/architecture/).

Finalmente, Gómez y Cervantes (2013) proponen la notación de diseño *User Interface Transition Diagram* (UITD) la cual tienen como objetivo mejorar la comunicación entre los diferentes interesados en un proyecto de software (ingenieros de software, clientes y especialistas en interacción humano-computadora). El trabajo resulta relevante para el área de diseño de software debido a que dicha notación de modelado, de acuerdo con sus autores, puede ser un auxiliar en la especificación de requisitos, en el diseño de interfaces de usuario y en diseño de los módulos de un sistema.

5.7 Métodos y estrategias de diseño de software

Para guiar al desarrollar durante el proceso de diseño de software se han propuesto diferentes estrategias y métodos. En el caso de las estrategias, se proveen una serie de lineamientos o pautas generales que pueden ser aplicadas en la mayoría de los problemas de diseño. Entre ellas podemos encontrar estrategias como divide y vencerás, el refinamiento sucesivo, arriba abajo y de abajo a arriba. Otras estrategias hacen uso de diferentes heurísticas y patrones para asistir al diseñador. Por otro lado, los métodos brindan una descripción más detallada de los procesos

a desarrollar en la fase de diseño, normalmente proveen una notación para los diferentes artefactos y una serie de guías para la utilización del método dependiendo de las circunstancias en las que se encuentra el diseñador.

En esta área se han propuesto métodos para el diseño de un tipo de software específico. En diseño de alto nivel, es decir, a nivel arquitectónico encontramos propuestas de procesos generales para el desarrollo de software. Por ejemplo, Nuñez Reyna (2009) propone una adaptación de los métodos *Quality Attribute Workshop* (Barbacci et al., 2003), *Attribute Driven Design* (Wojcik et al., 2006), *Views and Beyond* (Clements et al., 2002) y *Architectural Tradeoff Analysis Method* (Kazman et al., 2000) para el diseño arquitectónico enfocado a equipos de desarrollo pequeños, sistemas de complejidad media a baja y de bajo riesgo para el negocio a través de una serie de guías, plantillas y listas de verificación para apoyar al proceso de diseño. En este mismo sentido, Serratos-Álvarez, Martínez-Martínez y Oktaba (2010) proponen un paquete de puesta en operación, el cual guía al desarrollador no únicamente en el diseño y documentación de la arquitectura de acuerdo con la Norma ISO/IEC 29110 y métodos de diseño del SEI (www.i.cmu.edu) sino también en el diseño detallado y la evaluación de los artefactos generados.

Inspirada en la arquitectura conducida por modelos, Estrada, Morales Ramírez, Martínez y Pastor (2010) proponen un método para la generación de servicios web a partir de un modelo de servicios de

negocio. En este trabajo se desarrolló un metamodelo orientado a servicios (*MOS Ecore*) para la creación de los servicios web dentro del contexto organizacional, una serie de reglas de transformación de modelos de negocio a servicios web y una herramienta para asistir a la propuesta.

Por otra parte, también basada en MDA y utilizando ontologías, Bartolo Espíritu, Sánchez López y Cava Rosales (2014) proponen un método que combina diferentes técnicas para el desarrollo de software pero con especial énfasis en su diseño. En la propuesta se hace uso de ontologías para la definición y especificación de la arquitectura en etapas tempranas del desarrollo.

Finalmente, el último trabajo que se revisó que considera a MDA para el proceso de desarrollo fue el propuesto por Morales et al (2016). En él, se presenta un lenguaje de dominio específico para la generación de código de una aplicación web a partir del lenguaje utilizado en organización de desarrollo de software en México y que abona en el área de la Ingeniería Web Conducida por Modelos.

Zamudio López, Santaolaya Salgado y Fragoso Díaz (2012) proponen en su trabajo 11 métodos para la reestructuración de *frameworks* orientados a objetos hacia una arquitectura conforme al patrón Modelo-Vista-Adaptador con el objetivo de mejorar la facilidad de mantenimiento y la reutilización de software manteniendo la misma funcionalidad del *framework*. También abonando en el diseño de alto nivel, Urquiza Yllescas (2013) propone una metodología que incluye

las tareas, actividades, productos a desarrollar así como guías para el diseño arquitectónico y la documentación de productos de software desarrollados con un enfoque ágil. De esta manera, la propuesta incorpora de manera explícita el diseño arquitectónico en las metodologías ágiles de una forma genérica que le permite adaptarse a los elementos específicos de cada una de dichas metodologías.

Por otro lado, en la comunidad de investigadores mexicana también existen propuestas para guía el diseño de software de propósito específico. En este sentido encontramos trabajos que abonan en el uso de meta-modelos, por ejemplo, Céspedes Hernández, Pérez Medina, González Caballero, Rodríguez y Muñoz Arteaga (2015) brindan un meta-modelo para el diseño de juegos serios que asisten a la rehabilitación auditiva. En dicha propuesta se consideran para el diseño tanto elementos propios de los juegos serios, como la mecánica del juego y acciones de los jugadores, así como elementos que especialistas en el área consideran clave describiendo las partes teórica y dinámica de la rehabilitación y el contexto de uso.

Finalmente, Gudiño Mendoza y López Mellado (2017) proponen una metodología de modelado que busca asistir al diseñador en el desarrollo de redes de agentes utilizando redes Petri híbridas temporizadas. En este trabajo se detalla la arquitectura interna de los agentes proporciona un marco para calcular y analizar sistemas de redes de agentes a través de la simulación.

5.8 Herramientas de diseño de software

Este tópico se refiere a las herramientas que se emplean para la creación de los artefactos de diseño. En este sentido, se presenta la investigación realizada que busca el desarrollo de este tipo de herramientas.

Riba Zárate (2007) propone una herramienta para apoyar el desarrollo de componentes, siguiendo un enfoque declarativo mediante un enfoque basado en MDA. Este es un trabajo de tesis a nivel maestría que: 1) define un lenguaje de modelado para aplicaciones basadas en componentes orientados a servicios, 2) desarrolla una herramienta MDA con un enfoque declarativo para apoyar en el desarrollo de componentes y, 3) define un proceso para la construcción de herramientas para cualquier otro dominio. Posteriormente, Riba y Cervantes (2007) presentan el trabajo de la herramienta anterior enfatizando la necesidad de tener un soporte que permita la disponibilidad dinámica de componentes de software ya que, mediante su herramienta, los desarrolladores pueden dedicarse a la lógica de la aplicación, sin necesidad de preocuparse por la lógica de las adaptaciones en tiempo de ejecución.

El trabajo de Marcial Palafox (2009) es una tesis que define un proceso basado en arquitectura conducida por modelos para definir esqueletos de arquitecturas de software ejecutables que, además, desarrolla una herramienta para este propósito. El proceso considera, principalmente tácticas arquitectónicas para atributos de calidad como: Disponibilidad, Desempeño, Seguridad, Usabilidad, Facilidad de

Prueba, Facilidad de modificación. El proceso y herramienta propuestos emplean *frameworks* de desarrollo como *Spring*.

El trabajo de Mauricio Zamarrón (2015) es una tesis que define un método y herramientas para detectar patrones de diseño en código Java empleando técnicas de PLN (Procesamiento de Lenguaje Natural, PLN) y métricas de centralidad. Este trabajo es un primer esfuerzo de intentar combinar técnicas de PLN y métricas de centralidad para detectar patrones de diseño que considera sólo los de tipo estructural en lenguaje Java. El autor afirma que este trabajo es un primer intento hacia lograr una herramienta que permita la reconstrucción de arquitecturas de software.

Por otra parte, Velasco Elizondo, Marín Piña, Vázquez Reyes, Mora Soto y Mejía (2016) proponen una estrategia automatizada para el análisis de descripciones de patrones arquitectónicos con respecto a atributos de calidad, que emplea técnicas de representación del conocimiento y extracción de información. En su trabajo, presentan un prototipo que considera el atributo de rendimiento (performance). El objetivo de su trabajo es ayudar al arquitecto de software en la selección de patrones arquitectónicos con respecto a los atributos de calidad que los patrones arquitectónicos promueven o inhiben. Comparan el comportamiento de su modelo con el comportamiento de arquitectos de software (tanto novatos como experimentados) realizando la misma labor. Sus resultados demuestran que, con su propuesta, se reduce el tiempo de análisis y aumenta la memoria al respecto.

El trabajo que presentan Velasco Elizondo, Castañeda Calvillo, García Fernández y Vázquez Reyes (2017), busca: 1) desarrollar una caracterización de *bad smells* más importantes en el patrón arquitectónico Modelo-Vista-Controlador (MVC) y 2) automatizar la detección de dichos *bad smells* empleando técnicas de análisis textual. Los autores obtienen una compilación de *bad smells* más significativos para MVC; mientras que, en el caso de la herramienta, reportan un buen grado de exactitud y un buen comportamiento. Los autores reportan que, al momento de su publicación, este trabajo aún continuaba.

El trabajo presentado por Méndez Luna (2012) consiste en la implementación de un algoritmo de búsqueda en profundidad adaptado para seleccionar los patrones arquitectónicos con base en atributos de calidad. Esta herramienta busca servir de apoyo al arquitecto de software en la toma de decisiones durante el diseño de la arquitectura. Para el desarrollo de la herramienta, se hizo un análisis del catálogo de Buschmann, Henney y Schmidt (2007) y, para los atributos de calidad, se consideran los definidos por el SEI (www.sei.cmu.edu/architecture/).

Relacionado con el trabajo de Cortés Verdín (2009) presentado en el subtópico de Familias de productos o *frameworks*, en dos artículos se presenta una herramienta para el modelado de intereses (*concerns*) basada en COSMOS (Sutton & Rouvellou, 2002) para Eclipse (Uscanga Castillo, 2013; Uscanga Castillo, Cortés, & Martínez, 2012). Dicha herramienta, además, incorpora el soporte necesario para el desarrollo de una Línea de Productos de Software, que incluye desde la

administración del portafolio de productos basado en intereses hasta la identificación temprana de aspectos.

5.9 Conclusiones

A lo largo de este capítulo se han expuesto las aportaciones en el área de diseño de software en México. La estructura del mismo capítulo estuvo inspirada en el SWEBOK y para la búsqueda de los trabajos de investigación se emplearon estrategias de búsqueda tanto automáticas como manuales. Cabe mencionar que, aunque en el SWEBOK se menciona el tópico de Interacción Humano Computadora, este no fue incluido en el análisis ya que constituye un área de investigación demasiado extensa para ser incluida en este capítulo.

Si bien en México no se trabajan en todos los tópicos señalados en el SWEBOK, se puede apreciar que existe una importante comunidad de investigadores que trabajan en el área de diseño de software, especialmente en el tópico de Estructura de Software y Arquitectura que es en la que más aportaciones fueron identificadas.

Referencias

1. Acuña Cháirez, E. (2012). *Principios de Diseño aplicados a Árboles Binarios de Búsqueda*. Centro de Investigación en Matemáticas A.C.
2. Alor-Hernández, G., A. Aguilar-Lasserre, A., Juárez Martínez, U., Posada-Gómez, R., Robles, G., Alberto Garcia Martinez, M.,González,

A. (2010). HYDRA: A Middleware-Oriented Integrated Architecture for e-Procurement in Supply Chains. *T. Computational Collective Intelligence*, 6220, 1–20. http://doi.org/https://doi.org/10.1007/978-3-642-15034-0_1

3. Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C., & Wood, W. (2003). *Quality Attribute Workshops (QAWs)* (Third). Pittsburgh, PA.

4. Bartolo Espirítu, F., Sánchez López, A., & Calva Rosales, L. J. (2014). Towards an improvement of software development process based on software architecture, model driven architecture and ontologies. *En CONIELECOMP 2014 - 24th International Conference on Electronics, Communications and Computers*, pp. 118–126, IEEE Computer Society. http://doi.org/10.1109/CONIELECOMP.2014.6808578

5. Borrego, G., Moran, A. L., Palacio, R., & Rodriguez, O. M. (2016). Understanding architectural knowledge sharing in AGSD teams: An empirical study. *En Proceedings - 11th IEEE International Conference on Global Software Engineering, ICGSE 2016,* pp. 109–118. Institute of Electrical and Electronics Engineers Inc. http://doi.org/10.1109/ICGSE.2016.29

6. Buschmann, F., Henney, K., & Schmidt, D. C. (2007). *A pattern language for distributed computing*. (4a ed.). John Wiley & Sons.

7. Cabello Espinosa, M. E. (2007). *Análisis y Diseño de un Generador Automático de Sistemas de Diagnóstico Basado en Líneas de Productos*. Universidad Politécnica de Valencia.

8. Cabello Espinosa, M. E. (2008). *Baseline-oriented modeling: Una aproximación MDA basada en Líneas de productos software para el desarrollo de aplicaciones*. Universidad Politécnica de Valancia.

9. Cabello Espinosa, M. E., Preciado Álvarez, F., Santana Álvarez, O. A., & Ramos Salavert, I. (2018). Una aproximación MDE para generar una línea de productos de sistemas expertos. *Revista de la Alta Tecnología y Sociedad,* 10(1), pp. 26–42.

10. Cabello Espinosa, M. E., & Ramos, I. (2008). The Baseline: The Milestone of Software Product Lines for Expert Systems Automatic Development. *En 2008 Mexican International Conference on Computer Science*, pp. 44–51. http://doi.org/10.1109/ENC.2008.42

11. Cabello Espinosa, M. E., & Ramos Salavert, I. (2016). *Ingeniería de aplicaciones dirigida por modelos y basada en técnicas de líneas de producción software*. Universidad de Colima.

12. Cabello, M. E., Ramos, I., Santana, O. A., & Beristain, S. I. (2014). A Generic Process for the Design and Generation of Software Product Line Skeleton Architectures. *International Journal of Software Engineering and Knowledge Engineering,* 24(09), pp. 1301–1335.

13. Cervantes, H., Donsez, D., & Touseau, L. (2008). An Architecture Description Language for Dynamic Sensor-Based Applications. *En 5th IEEE Consumer Communications & Networking Conference (CCNC 2008)*, Las Vegas, Nevada.

14. Cervantes, H., & Kazman, R. (2016). *Designing Software Architectures: A Practical Approach*. Addison-Wesley Professional.

15. Cervantes, H., Kazman, R., Ryoo, J., Choi, D., & Jang, D. (2016). *Architectural Approaches to Security: Four Case Studies*. *Computer*, 49(11), pp. 60–67. http://doi.org/10.1109/MC.2016.332

16. Cervantes, H., Velasco-Elizondo, P., & Kazman, R. (2013). A Principled Way to Use Frameworks in Architecture Design. *IEEE Software*, 30(2), pp. 46–53. http://doi.org/10.1109/MS.2012.175

17. Céspedes-Hernández, D., Pérez-Medina, J. L., González-Calleros, J. M., Álvarez-Rodríguez, F. J., & Muñoz-Arteaga, J. (2015). SEGA-ARM: A Metamodel for the Design of Serious Games to Support Auditory Rehabilitation. *En Proceedings of the XVI International Conference on Human Computer Interaction,* pp.10:1-10:8. New York, NY, USA: ACM. http://doi.org/10.1145/2829875.2829877

18. Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., … Merson, P. (2002). *Documenting Software Architectures: Views and Beyond*. Pearson Education.

19. Cortés Pichardo, D. (2011). *Arquitectura de Software en Métodos Ágiles*. Universidad Nacional Autónoma de México.

20. Cortés Verdín, M. K. (2009). *AOPLA: Aspect-Oriented Product Line Architecture*. Centro de Investigación en Matemáticas A.C.

21. Cortés Verdín, M. K. (2011). Modelo de Calidad para Líneas de Productos de Software. *En IX Congreso Internacional sobre Innovación y Desarrollo Tecnológico,* pp. 1–8. Cuernavaca.

22. Cortés Verdín, M. K., Lemus Olalde, C., van der Hoek, A., & Fernández Peña, J. M. (2009). AN ASPECT-ORIENTED APPROACH FOR THE DESIGN OF PRODUCT LINE ARCHITECTURES. *En 4o.*

Simposio Internacional en Sistemas Telemáticos y Organizaciones Inteligentes. Xalapa.

23. Cortés Verdín, M. K., Lemus Olalde, C., van der Hoek, A., & Fernández Peña, J. M. (2010). Diseño de Arquitecturas de Líneas de Productos de Software empleando AOPLA. *En VII Congreso Internacional sobre Innovación y Desarrollo Tecnológico*, pp. 1–8. Cuernavaca.

24. Díaz Velásquez, V. J. (2016). *Desarrollo de una LPS para Domótica utilizando programación orientada a Deltas*. Instituto Tecnológico de Orizaba.

25. Duran-Limon, H. A., Garcia-Rios, C. A., Castillo-Barrera, F. E., & Capilla, R. (2015). An Ontology-Based Product Architecture Derivation Approach. *IEEE Transactions on Software Engineering*, 41(12), pp. 1153–1168. http://doi.org/10.1109/TSE.2015.2449854

26. Estrada, H., Morales-Ramírez, I., Martínez, A., & Pastor, O. (2010). From business services to Web services: An MDA approach. *En CEUR Workshop Proceedings*, Vol. 586, pp. 31–35).

27. Fernandez, E. B., & Ortega-Arjona, J. L. (2009). The Secure Pipes and Filters Pattern. *En Proceedings of the 2009 20th International Workshop on Database and Expert Systems Application* (pp. 181–185). Washington, DC, USA: IEEE Computer Society. http://doi.org/10.1109/DEXA.2009.55

28. Figueroa-Gutiérrez, S., Cortés-Verdín, K., & Contreras-Vega, G. (2016). Development of a security mechanisms catalog. *Research in Computing Science*, 128.

29. Figueroa Gutierrez, S. (2015). *Modelo de calidad de seguridad para arquitecturas de software*. Universidad Veracruzana.

30. Gómez, A., Cabello, M. E., & Ramos, I. (2010). BOM-Lazy: A Variability-Driven Framework for Software Applications Production Using Model Transformation Techniques. *En SPLC Workshops*, pp. 139–146.

31. Gómez, M., & Cervantes, J. (2013). User Interface Transition Diagrams for customer-developer communication improvement in software development projects. J*ournal of Systems and Software,* 86(9), pp. 2394–2410. http://doi.org/10.1016/j.jss.2013.04.022

32. Gudiño-Mendoza, B., & López-Mellado, E. (2017). A modeling methodology for designing agents networks using timed hybrid Petri nets. *Simulation*, 93(4), pp. 323–333.

33. Hernández-López, J.-M., & Juárez-Martínez, U. (2016). Generación automatizada de aplicaciones inmóticas bajo el enfoque de LPS. *Research in Computing Science*, 126, pp. 73–83.

34. Hernández-López, J.-M., Juárez-Martínez, U., & Ixmatlahua-Díaz, S.-D. (2018). Automated software generation process with SPL. *En CIMPS 2017: Trends and Applications in Software Engineering,* Vol. 688, pp. 127–136. Springer. http://doi.org/10.1007/978-3-319-69341-5

35. Hernández-López, J.-M., Juárez-Martínez, U., Muñoz Contreras, H., & Cortés Verdín, M. K. (2015). Línea de productos de software combinando Scala, MDA y orientación a aspectos. *En Congreso Internacional de Robótica y Computación 2015*.

36. Hernández López, J. M. (2016). *Desarrollo de una LPS para Inmótica utilizando MDA, SCALA y ASPECTJ*. Instituto Tecnológico de Orizaba.

37. IEEE Computer Society, Bourque, P., & Fairley, R. E. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0* (3rd ed.). Los Alamitos, CA, USA: IEEE Computer Society Press.

38. ISO/IEC, & IEEE. (2017). *International Standard ISO/IEC/IEEE 24765: Systems and Software Engineering: Vocabulary* (Vol. 25021). International Standards Organization, IEEE. http://doi.org/10.1109/IEEESTD.2015.7106438

39. Jarvio Hernández, Y. (2015). *Uso de Modelos basados en Arquitectura de Software para el Desarrollo de Herramientas de Composición más Usables*. Universidad Veracruzana.

40. Kazman, R., Klein, M., & Clements, P. (2000). *ATAM: Method for Architecture Evaluation*. Pittsburgh, PA.

41. López-Martín, C. (2008). Quality improvement applying design and code reviews for developing small programs. *International Multi-Conference on Engineering and Technological Innovation, Proceedings (IMATI 2008)*, Vol. 2, pp. 153–158.

42. Marcial Palafox, P. A. (2009). *Modelado de tácticas de atributos de calidad para la generación de esqueletos de arquitecturas.* Universidad Autónoma Metropolitana.

43. Matamoros, M., Savage, J., & Ortega-Arjona, J. L. (2015). A comparison of two software architectures for general purpose mobile service robots. En M. R. A. L. Valente A. Marques L. (Ed.), *Proceedings - 2015 IEEE International Conference on Autonomous*

Robot Systems and Competitions, ICARSC 2015, pp. 131–136, Institute of Electrical and Electronics Engineers Inc. http://doi.org/10.1109/ICARSC.2015.10

44. Mauricio Zamarrón, J. M. (2015). *Evaluando el uso de técnicas de procesamiento de lenguaje natural y métricas de centralidad para la detección de patrones de diseño de software.* Centro de Investigación en Matemáticas A.C.

45. Méndez Luna, S. (2012). *Método-Asistente para la toma de decisiones de Diseño de Arquitecturas de Software (MATDDS).* Universidad Autónoma Metropolitana.

46. Morales, Z., Magańa, C., Aguilar, J. A., Zaldívar-Colado, A., Tripp-Barba, C., Misra, S., … Zurita, E. (2016). A baseline domain specific language proposal for model-driven web engineering code generation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9790, pp. 50–59. http://doi.org/10.1007/978-3-319-42092-9_5

47. Nuñez Reyna, J. I. (2009). *Adaptación de una Metodología de Desarrollo Arquitectónico al Contexto de Equipos de Desarrollo Pequeños.* Universidad Autónoma Metropolitana.

48. Ocharán-Hernández, J. O. (2009). *Documentación de Arquitecturas Orientadas a Aspectos para Líneas de Productos de Software*. Universidad Veracruzana.

49. Ocharán-Hernández, J. O., & Cortes-Verdin, K. (2008). Documentando Arquitecturas Orientadas a Aspectos para Líneas de Productos de Software. *Research in Computing Science*, 38, pp. 333–342.

50. Ortega-Arjona, J. L. (2010). *Patterns for Parallel Software Design* (1st ed.). Wiley Publishing.

51. Ortega-Arjona, J. L., & Fernandez, E. B. (2008). T*he Secure Blackboard Pattern. En Proceedings of the 15th Conference on Pattern Languages of Programs*, p. 22:1--22:5. New York, NY, USA: ACM. http://doi.org/10.1145/1753196.1753223

52. Ramírez Mora, S. L. (2016). *Guía para la descripción y documentación de arquitecturas de Software utilizando lenguajes de descripción de arquitectura*. Universidad Nacional Autònoma de México.

53. Riba, N., & Cervantes, H. (2007). A MDA tool for the development of service-oriented component-based applications. En Current Trends in Computer Science, 2007. *ENC 2007. Eighth Mexican International Conference on Computation*, pp. 149–156.

54. Ruiz, C., Duran-Limon, H. A., & Parlavantzas, N. (2016). Towards a Software Product Line-based Approach to Adapt IaaS Cloud Configurations. *En Proceedings of the 9th International Conference on Utility and Cloud Computing*, pp. 398–403. New York, NY, USA: ACM. http://doi.org/10.1145/2996890.3007893

55. Ryoo, J., Laplante, P., & Kazman, R. (2012). Revising a security tactics hierarchy through decomposition, reclassification, and derivation. *Proceedings of the 2012 IEEE 6th International Conference on Software Security and Reliability Companion, SERE-C* 2012, pp. 85–91. http://doi.org/10.1109/SERE-C.2012.18

56. Schumacher, M., Fernandez, E., Hybertson, D., & Buschmann, F. (2005). *Security Patterns: Integrating Security and Systems Engineering*. USA: John Wiley & Sons, Inc.

57. Serratos-Álvarez, E., Martínez-Martínez, A., & Oktaba, H. (2010). Desarrollo de Guías para el Diseño, Documentación y Evaluación de Arquitecturas de Software, con Base en la Norma ISO/IEC 29110 y en los Métodos del SEI. *En Memorias del Coloquio Nacional de Investigación en Ingeniería de Software y Vinculación Academia-Industria* (CoNIIS).

58. Sutton, S. M. J., & Rouvellou, I. (2002). Modeling of Software Concerns in Cosmos. *En 1st International Conference in Aspect-Oriented Software Development,* pp. 127–133. Enschede.

59. Trujillo-Tzanahua, G. I., Juárez-Martínez, U., Aguilar-Laserre, A. A., & Cortés-Verdín, M. K. (2018). Trends and Applications in Software Engineering. En J. Mejia, M. Muñoz, Á. Rocha, Y. Quiñones, & J. Calvo-Manzano (Eds.), *CIMPS 2017: Trends and Applications in Software Engineering*, Vol. 688, pp. 117–126. Springer. http://doi.org/10.1007/978-3-319-69341-5

60. Urquiza-Yllescas, J. F. (2013). *Modelo Genérico para el Desarrollo de la Arquitectura de Software en Metodologías Ágiles*. Universidad Autónoma Metropolitana.

61. Uscanga Castillo, M. (2013). *Desarrollo de una Herramienta para el modelado de Intereses (Concerns) para una Línea de Productos de Software*. Universidad Veracruzana.

62. Uscanga Castillo, M., Cortés, M. K., & Martínez, U. J. (2012). Herramienta para el Modelado de Intereses (Concerns) para una Arquitectura de Líneas de Productos de Software. *En CONISOFT* 2012, pp. 1–8. Guadalajara, México.

63. Velasco-Elizondo, P., Barredo-Hernandez, D., & Mitre, H. A. (2014). Aggregate QoS Estimation of Service Compositions - An Analysis of Pattern-Oriented Approaches. *En Proceedings of the 2014 IEEE World Congress on Services,* pp. 362–369. Washington, DC, USA: IEEE Computer Society. http://doi.org/10.1109/SERVICES.2014.70

64. Velasco-Elizondo, P., Castañeda-Calvillo, L., García-Fernandez, A., & Vazquez-Reyes, S. (2017). Towards Detecting MVC Architectural Smells. *En International Conference on Software Process Improvement*, pp. 251–260.

65. Velasco-Elizondo, P., Marín-Piña, R., Vazquez-Reyes, S., Mora-Soto, A., & Mejia, J. (2016). *Knowledge representation and information extraction for analysing architectural patterns. Science of Computer Programming,* 121, pp. 176–189.

66. Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R., & Wood, W. (2006). *Attribute-Driven Design (ADD*), Version 2.0. Pittsburgh, PA.

67. Zamudio López, S. A., Santaolaya Salgado, R., & Fragoso Díaz, O. G. (2012). Restructuring object-oriented frameworks to model-view-adapter architecture. I*EEE Latin America Transactions,* 10(4), pp. 2010–2016. http://doi.org/10.1109/TLA.2012.6272488

Capítulo 6
Investigación en el área de Mejora de Procesos de Software

Mirna Ariadna Muñoz Mata, *Centro de Investigación en Matemáticas,*

Jezreel Mejía Miranda, *Centro de Investigación en Matemáticas,*

María de León Sigg, *Universidad Autónoma de Zacatecas.*

6.1 Introducción

La capacidad de las organizaciones y de sus productos, sistemas y servicios que les permite competir, adaptarse y sobrevivir en un entorno altamente cambiante, depende cada vez más del software. El software facilita la adaptación rápida de productos y servicios a diferentes sectores del mercado, por tanto, es indispensable garantizar su calidad. Con base en la perspectiva de que la calidad del software está directamente relacionada con la calidad de los procesos utilizados para su desarrollo (Cuevas, 2002), las organizaciones necesitan establecer "el CÓMO" para definir y desplegar sus procesos. En este contexto, es necesario conocer modelos y estándares de mejora, herramientas y como el factor humano interviene en la implementación de procesos

dentro de las organizaciones, además, de la capacidad de seleccionar los más adecuados al entorno de la organización.

Por lo tanto, el objetivo de este capítulo es proporcionar una visión integral de los procesos, su importancia para la madurez y capacidad de las organizaciones y las tendencias de investigación desarrolladas en este contexto. Para lograrlo, en el capítulo se analiza la investigación en procesos desde el punto de vista de los elementos que intervienen en esta línea. Como se muestra en la Figura 5.1, se analiza la investigación de mejora de procesos de software desde tres aspectos: modelos y estándares utilizados como referencia para la mejora, herramientas y factor humano, necesarios para lograr una mejora de procesos exitosa.

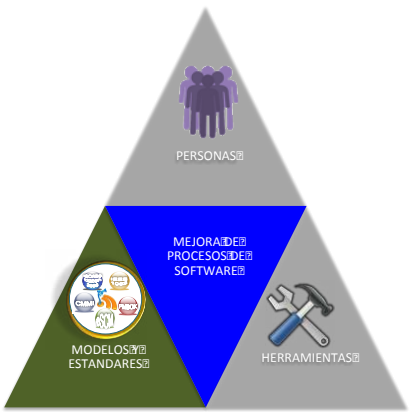

Figura 6.1. Aspectos analizados de la mejora de procesos de software

Además, el capítulo recopila la investigación realizada por grupos de investigación mexicanos en esta área.

6.2 Modelos y Estándares

Las organizaciones dependen cada vez más del software debido a que éste facilita la adaptación rápida de productos y servicios a diferentes sectores del mercado. Por lo tanto, asegurar la calidad del software se ha convertido en un aspecto crítico, siendo necesario para las organizaciones de desarrollo de software saber definir adecuadamente la calidad del software y cómo deber ser evaluada ésta (Chrissis, Konrad & Shrum, 2011). Además, para considerar que un software es de calidad deber ser analizada la seguridad, de lo contrario un software sin seguridad se considera un software sin calidad (Viega & McGraw, 2011).

En esta sección, se incluye un breve análisis de los principales modelos y estándares utilizados como referencia en México como son: CMMI-Dev, Moprosoft, ISO/IEC 29110.

6.2.1 CMM-DEV

CMMI (*Capability Maturity Model Integration*) es un modelo de mejora del desempeño de organizaciones y proyectos que reúne mejores prácticas útiles para elevar la calidad, la rentabilidad y la competitividad (*CMMI Institute, 2018*).

Propósito. El propósito de CMMI-Dev es brindar ayuda a las organizaciones para mejorar sus procesos tanto de desarrollo como de mantenimiento de productos y servicios (CMMI Institute, 2018).

Estructura. Cada área de proceso contiene definidos los siguientes elementos de proceso: objetivos específicos, objetivos genéricos, prácticas específicas, prácticas genéricas, productos típicos de trabajo y sub-prácticas.

Niveles de madurez/capacidad propuestos. CMMI utiliza un modelo de madurez de cinco niveles (CMMI Institute, 2018b), como se muestra en la Figura 5.2.

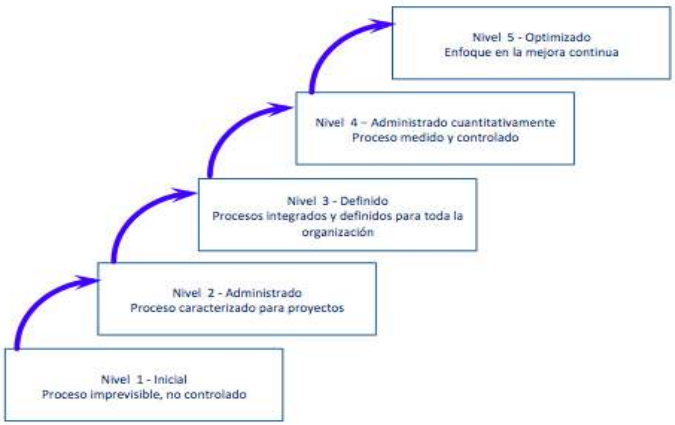

Figura 6.2. Niveles de Madurez CMMI

Procesos. CMMI-Dev contiene 22 áreas de proceso, clasificadas en cuatro categorías como a continuación se lista:

- Gestión de proyectos: gestión cuantitativa del proyecto (QPM), gestión integrada de monitorización y control del proyecto (PMC); planificación del proyecto (PP); gestión de requisitos (REQM); gestión de riesgos (RSKM) y gestión de acuerdo con proveedores (SAM)
- Soporte: análisis causal y resolución (CAR); análisis de decisiones y resolución (DAR); gestión de configuración (CM); medición y análisis (MA); aseguramiento de la calidad de proceso y de producto (PPQA).
- Ingeniería: Integración de producto (PI); desarrollo de requisitos (RD); solución técnica (TS); verificación (VER) y validación (VAL)
- Gestión de procesos: Definición de proceso de la organización (OPD); enfoque en proceso de la organización (OPF); gestión del rendimiento de la organización (OPM); rendimiento de procesos de la organización (OPP) y formación Organizativa (OT).

Estado actual en México del uso del estándar/modelo. En este capítulo se describe el modelo CMMI-Dev V1.3v, ya que es el que actualmente está siendo utilizado por las empresas mexicanas, sin embargo cabe resaltar que en el mes de marzo de 2018 se liberó la versión CMMI v2.0, que incluye modelos para desarrollo, servicios, adquisición y desarrollo de habilidades en las personas.

En México, existen 286 certificaciones de CMMI 1.3 (CMMI Institute, 2018c), divididas en 183 para el modelo CMMI-DEV v1.3, 97

para el modelo CMMI-SVC v1.3 y 6 para el modelo CMMI-SVC+SSD v1.3.

6.2.2 MoProSoft

En México, la mayoría de las empresas de desarrollo y mantenimiento de software entran en la clasificación de pequeñas y medianas empresas (PyMEs).

Entre las características de estas organizaciones se pueden distinguir las siguientes (Muñoz, Gasca-Hurtado & Valtierra, 2014):

- Capacidad limitada para implementar iniciativas en el área de mejora de procesos debido a que no cuentan con procesos definidos, no siguen ciclos de desarrollo de software y desconocen su importancia para la calidad del producto;
- Poca o nula experiencia en la adopción de modelos y estándares de mejora de procesos software;
- Recursos financieros limitados para invertir en la mejora de procesos; sus recursos humanos son mínimos y realizan varias funciones, además de que éstos no tienen conocimientos sobre mejora de procesos.

Lo anterior implica que la capacidad de estas empresas para competir se vea reducida significativamente, por lo que, en un esfuerzo para mejorar su competitividad, el gobierno de México a través de la Secretaría de Economía, creó el Modelo de referencia de Procesos para la Industria del Software (MoProSoft).

MoProSoft es un modelo de referencia apropiado a las características de las empresas mexicanas, basado en mejores prácticas de la industria, fácil de entender y aplicar y asequible para ellas (Oktaba, 2009). Este modelo se convirtió más tarde en la norma NMX-I-059/01-NYCE-2005 que ha evolucionado hasta la versión actual NMX-I-059-2-NYCE-2016.

Propósito. El propósito de MoProSoft es apoyar a las organizaciones a estandarizar sus prácticas, evaluar su efectividad y elevar su capacidad mediante la mejora continua, para que puedan ofrecer sus servicios con calidad y competir a nivel internacional (Oktaba, 2009). La norma está definida para que pueda aplicarse en organizaciones que tienen procesos establecidos así como en aquellas que no los tengan (Diario Oficial de la Federación, 2016).

Estructura. MoProSoft utiliza un patrón de procesos que contiene la definición general del proceso, prácticas y guías de ajuste.

La definición general del proceso incluye: proceso (nombre); categoría (nombre); propósito; descripción; objetivos; indicadores; metas cuantitativas; responsabilidad y autoridad; procesos relacionados; entradas (nombre, fuente); salidas (nombre, descripción, destino); productos internos (nombre, descripción) y referencias bibliográficas (normas, modelos de referencia, etc.)

Las prácticas están descritas por: roles involucrados y capacitación; actividades (rol, actividad, objetivo, tareas); diagrama de

flujo de trabajo; verificaciones y validaciones (actividad, producto, rol, descripción); incorporación a la base de conocimiento (producto, forma de aprobación); recursos de infraestructura (actividad, recurso); mediciones (ejemplo de medición por indicador); capacitación; situaciones excepcionales y lecciones aprendidas. Finalmente, las guías de ajuste contienen la descripción de las posibles modificaciones al proceso que no deben afectar los objetivos del mismo.

Niveles de madurez/capacidad propuestos. La evaluación de procesos de acuerdo con MoProSoft está basada en la norma ISO/IEC 15504, que propone cinco niveles de capacidad, más el nivel incompleto (ISO, 2011). Cada uno de estos niveles de capacidad se caracteriza por uno o dos atributos, como se muestra en la figura 5.3 (Oktaba, 2009).

Procesos. MoProSoft considera tres categorías de procesos llamadas Alta Dirección, Gerencia y Operación.

- La categoría de Alta Dirección incluye únicamente al proceso de Gestión de Negocio, que abarca las prácticas de gestión de la empresa como a continuación de describe (Oktaba, 2009).
- La categoría de gerencia integra los procesos de gestión de procesos, proyectos y recursos. La categoría de gestión de recursos está constituida a su vez por los subprocesos de "Recursos Humanos y Ambiente de Trabajo", "Bienes, Servicios e Infraestructura" y "Conocimiento de la Organización". Esta gestión se hace en términos de las directrices establecidas en la

categoría de Alta Dirección, de tal manera que se proporcionen los elementos necesarios para los procesos de la categoría de Operación.

- La categoría de operación contiene las prácticas de los procesos de administración de proyectos específicos y desarrollo de software y mantenimiento de software.

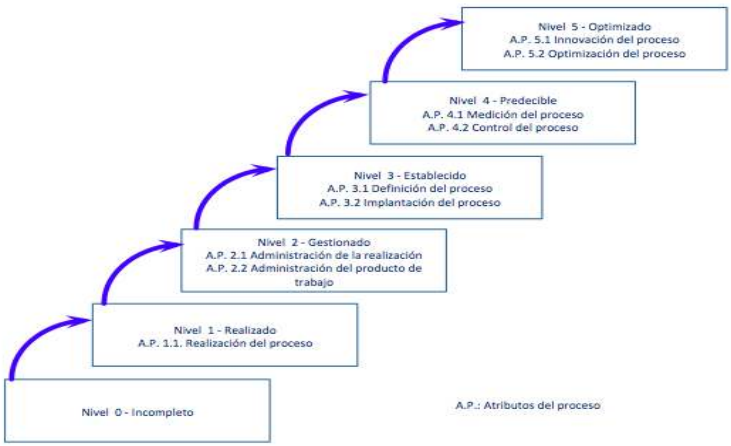

Figura 6.3. Niveles de Capacidad MoProSoft

Estado actual en México del uso del estándar/modelo. De acuerdo con datos de NYCE (NYCE, 2018), se han realizado en México 477 dictámenes en MoProSoft desde 2006 y hasta febrero 2018. Estos dictámenes incluyen 5 en el nivel 0, 212 en el nivel 1, 43 en el nivel 2, y 1 en el nivel 3 para la norma NMX-I-059/02-NYCE-2005.

Para la norma NMX-I-059/02-NYCE-2011 existen 1 para el nivel 0, 56 para el nivel 1, 124 para el nivel 2 y 11 para el nivel 3. Finalmente, para la norma NMX-I-059/02-NYCE-2016 existen veintiún dictámenes para el nivel 2, dos dictámenes para el nivel 3 y uno para el nivel 5.

6.2.3 ISO/IEC 29110

Este estándar surge para dar respuesta a la necesidad de reforzar organizaciones muy pequeñas mediante el desarrollo de estándares que las apoyen en la implementación de buenas prácticas de ingeniería de software para el desarrollo de productos de calidad, a la vez que mejoran su operación y sus procesos. Este estándar puede ser utilizado independientemente del enfoque de desarrollo o metodología utilizado en la empresa.

Propósito. ISO/IEC 29110 proporciona una serie de guías y directrices para mejorar el proceso de desarrollo de software de las organizaciones muy pequeñas, mediante la implementación de buenas prácticas para la obtención de beneficios como incremento en la calidad del producto y/o servicio, reducción en tiempos de entrega y reducción en costos de producción.

Estructura. La guía de gestión e ingeniería proporciona la siguiente información: alcance; referencias normativas; términos y definiciones; especificaciones y términos abreviados; visión general, proceso de

gestión de proyectos, proceso de implementación de software, roles, descripción del producto y requisitos para las herramientas de software.

Cada proceso está definido siguiendo un conjunto de elementos de proceso que facilitan su adopción como son: nombre del proceso, propósito, objetivos, productos de entrada, productos de salida, productos internos, roles involucrados, diagrama y actividad.

Además, cada actividad está definida utilizando los siguientes elementos del proceso: tareas, roles productos de entrada y productos de salida.

Niveles de madurez/capacidad propuestos. El estándar ISO/IEC 29110 está compuesto por 4 perfiles (perfil de entrada, perfil básico, perfil intermedio y perfil avanzado) que pueden ser usados por las entidades muy pequeñas de acuerdo a sus objetivos.

Procesos. Dependiendo del perfil el estándar proporciona un conjunto de procesos, como a continuación de describe y se muestra en la Figura 6.4.

- El perfil de entrada y básico: el proceso de gestión de proyectos y el proceso de implementación de software.
- El perfil intermedio: el proceso de gestión de proyectos, el proceso de implementación de software, proceso de administración de empresas y proceso de gestión de adquisiciones.

- El perfil avanzado: el proceso de gestión de proyectos, el proceso de implementación de software, proceso de administración de empresas, proceso de gestión de adquisiciones y proceso de transición y eliminación.

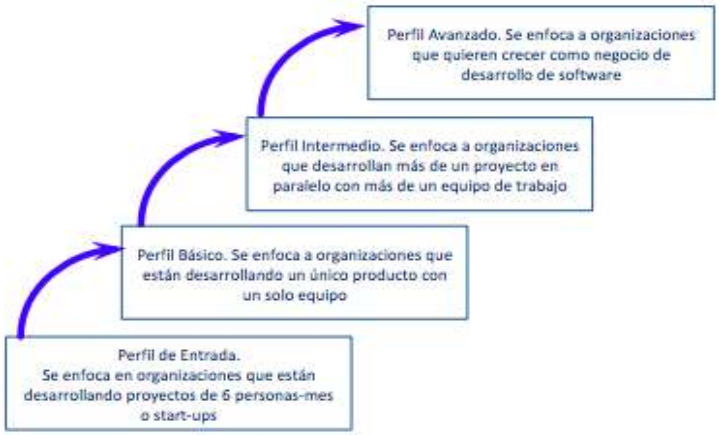

Figura 6.4. Perfiles de ISO/IEC 29110

Estado actual en México del uso del estándar/modelo. El perfil básico del estándar ISO/IEC 29110 proporciona un estándar aceptado en la industria mexicana del software, dato que se confirma al revisar el padrón de empresas certificadas en el estándar ISO/IEC 29110 publicado por NYCE, donde se menciona que 31 de las 38 empresas certificadas en el estándar son mexicanas.

6.3 Personas

En la actualidad el software es desarrollado por equipos de personas (CMMI Institute, 2018), por lo tanto, los profesionales en TI deben de estructurarse como equipos, lo que significa que deben comprender su propio rendimiento y aprender de su experiencia. Un aspecto clave para lograr ser un equipo de trabajo real es que sean capaces de establecer una comunicación adecuada, así como, tener la habilidad para planificar y estimar su trabajo, que se verá reflejado en el cumplimiento de sus compromisos y una mejora en su productividad y calidad.

En este contexto, se resaltan dos situaciones: 1) las empresas se encuentran compitiendo no solo por desarrollar los mejores productos y/o servicios, sino además por el talento requerido para producirlos y 2) de acuerdo al incremento del conocimiento requerido para la producción de productos y servicios, se hace más crítica la retención de empleados experimentados para mejorar la productividad y tiempos de liberación (Curtis & Hefley 2009).

Por lo anterior, es importante desarrollar una cultura del uso de procesos en las personas y, por otro lado, desarrollar y aplicar técnicas enfocadas en la gestión estratégica del capital humano.

Una forma de impulsar esta cultura de procesos es mediante el uso de metodologías que promueven el fortalecimiento de los dos aspectos antes mencionados. Entre las más comunes y más utilizadas

en México se pueden mencionar el Proceso Personal de Software (PSP), el Proceso de Software en Equipo (TSP) y Scrum.

6.3.1 Proceso Personal de Software

El Proceso Personal de Software (PSP, por sus siglas en ingles), es un proceso que ayuda al ingeniero de software a controlar, administrar y mejorar la forma en la que realiza las tareas propias del desarrollo (W. Humphrey, 2005).

La premisa principal de PSP es que la disciplina personal puede ayudar a mejorar la efectividad de los ingenieros de software (W. S. Humphrey, 1994), por lo tanto, para su diseño, Watts Humphrey se basó en los niveles del Modelo de Madurez de la Capacidad, CMM.

En este contexto, la premisa principal de PSP hace que el ingeniero progrese en el desarrollo de habilidades de planificación y seguimiento al plan, definición de procesos, revisiones personales, administración cuantitativa de procesos, administración de la calidad, prevención de defectos y administración de los cambios al proceso (W. Humphrey, 1995).

Propósito. El propósito principal de PSP es formar mejores ingenieros de software al ofrecer un marco de trabajo, una serie de formatos, guías y procedimientos para el desarrollo de productos. Con estas herramientas y métodos, el ingeniero de software obtiene información cuantitativa acerca de su proceso y de los productos que genera. Por ejemplo, con PSP es posible identificar por qué se cometen errores en

el desarrollo y cómo encontrar más fácilmente los defectos que se generan con esos errores, con los métodos que resultan más efectivos para ello (W. Humphrey, 2005).

Estado actual en México del uso del estándar/modelo. El Instituto de Ingeniería de Software (SEI, por sus siglas en inglés), mantiene un registro de los individuos que han obtenido un certificado PSP. En este registro aparecen los nombres de 240 mexicanos, de un total de 464 desarrolladores que han obtenido y tienen vigente la certificación. Esto implica que el 51.7% de los desarrolladores certificados en el mundo, son de México (*Software Engineering Institute*, 2018). Por otro lado, existen instituciones de educación superior mexicanas, tanto públicas como privadas, que cuentan, o han contado, entre sus profesores a instructores PSP autorizados por el SEI y que ofrecen PSP como parte del material de asignaturas de programas de Ingeniería de Software y afines. El Tec de Monterrey, en Monterrey y Guadalajara, la Universidad Regiomontana, la Universidad de Monterrey, la Universidad Autónoma de Zacatecas, la Universidad Autónoma de Yucatán, el Centro de Investigación en Matemáticas y la Universidad Autónoma de Nuevo León, son algunas de ellas (Gómez, 2014), (Nichols & Salazar, 2009).

6.3.2 Proceso de Software en Equipos

Los resultados obtenidos de proyectos grandes de desarrollo de software dependen en gran medida del desempeño de los equipos y de

los individuos que participan en ellos. Para mejorar esos resultados, una buena estrategia es enfocarse en el desempeño individual y de equipo y el Proceso de Software en Equipo. En este contexto, *Team Software Process* (TSP), guía a los ingenieros en el uso de métodos efectivos de trabajo en equipo (McAndrews, 2001).

Con la aplicación de la estrategia TSP, las organizaciones han logrado mejorar la productividad y reducir los costos, pero la mejora más significativa tiene que ver con la calidad del software producido, ya que existen proyectos en donde se han reportado cero defectos después de las pruebas de aceptación (W. S. Humphrey & Thomas, 2010).

Un equipo TSP es un grupo de hasta 15 integrantes, con un plan común, metas definidas y un único líder de equipo. Sin embargo, existen documentadas algunas variantes (W. S. Humphrey, Chick, Nichols, & Pomeroy-Huff, 2010):

- Equipos funcionales, donde todos las tareas de los integrantes son independientes unas de otras.
- Equipos integrados, son aquellos cuyos integrantes tienen diferentes especialidades, pero deben trabajar de manera cercana para producir un producto de calidad.
- Equipos distribuidos, cuyos integrantes trabajan en diferentes lugares físicos.

- Multi-equipo TSP, donde existen varios equipos TSP trabajando en producir elementos del mismo producto.
- Equipo TSP académico, es una variación introductoria de TSP utilizada para equipos pequeños de integrantes que, en un ambiente académico, simulan experiencias de proyectos reales.

Propósito. El propósito de TSP es ofrecer un contexto disciplinado para hacer ingeniería en equipos. Para ello, se basa en la convicción de que un equipo puede hacer trabajo extraordinario solamente si está correctamente formado, adecuadamente entrenado, dotado de integrantes capacitados y conducido de manera efectiva (W. S. Humphrey, 2000). El entrenamiento necesario para ser un integrante de un equipo bajo el enfoque TSP incluye la capacitación en PSP y en las disciplinas de ingeniería de software apropiadas para el desarrollo.

Estado actual en México del uso del estándar/modelo. Hasta ahora se ha hecho un esfuerzo importante por parte de la Secretaría de Economía y de instituciones como el Tecnológico de Monterrey para implementar una estrategia continua de capacitación y certificación TSP (Nichols & Salazar, 2009).

Como resultado de estos esfuerzos, existían hasta el 18 de abril de 2016, 28 centros de desarrollo TSP mexicanos evaluados en su desempeño y capacidad, con el proceso TSP-PACE (Secretaría de Economía, 2016). La evaluación TSP-PACE mide la capacidad para medir y articular el desempeño de una organización usando el TSP. Los

datos evaluados son los generados durante la realización del proyecto (Nichols, Kasunic, & Chick, 2013).

6.3.3 SCRUM

Es una metodología que forma parte de los métodos ágiles, por lo que se caracteriza por ciclos de desarrollo iterativos cortos, ejecutados por equipos auto organizados, que utilizan técnicas como diseños simples, refactorización de código, desarrollo basado en pruebas e involucración frecuente del cliente (Schwaber & Surtherland, 2017).

Scrum es un marco de trabajo que consiste de un equipo *Scrum* y sus roles, eventos, artefactos y reglas asociadas, donde cada componente del marco de trabajo sirve para un propósito específico y por lo tanto, es esencial para el uso y éxito de la metodología (Cohn, 2014).

Propósito. Scrum es un marco de trabajo de proceso utilizado para gestionar el trabajo de productos complejos, en el que se pueden utilizar varios procesos y técnicas (Cohn, 2014), (Schwaber & Surtherland, 2017).

Estado actual en México de su uso. Desde 2009 la empresa SCRUM México, ha sido un referente en capacitación y consultoría de *Scrum* en México certificando a cerca de 6,000 profesionales en *Scrum* de acuerdo a sus datos de finales del 2017 (*Scrum* México, 2018).

Esta empresa además colabora con empresas mexicanas y transnacionales para la adopción de *Scrum* como su metodología ágil para la gestión de sus proyectos.

6.4 Herramientas y Equipo

A pesar de que se ha reconocido la importancia de la mejora de procesos por varios autores como un mecanismo para impulsar la competitividad y eficiencia en la industria del software, la definición y uso de sus procesos, así como la mejora de los mismos, este puede convertirse en un camino lleno de obstáculos para la mayoría de las organizaciones, en especial en las pequeñas y medianas empresas (Muñoz et al., 2012).

Por lo tanto, hoy en día una necesidad primordial en esta área, es el desarrollo de herramientas software que soporten la implementación y uso de procesos. Una herramienta software puede ser un sistema, una plataforma, una aplicación o una wiki que apoye en la correcta realización de los procesos dentro de una organización.

En este contexto, en (Muñoz et al., 2012), se tienen considerados nueve requisitos a tener en cuenta para el desarrollo de herramientas de soporte enfocadas en apoyar a mejora de procesos.

- Evaluación del proceso: la herramienta debe proporcionar soporte para ejecutar una evaluación interna de sus procesos actuales.
- Instantánea del proceso: la herramienta debe permitir la obtención de una instantánea del proceso actual de la organización.

- Guía para la selección del proceso: la herramienta debe proporcionar una guía para selección del proceso a ser mejorado.
- Modelado del proceso: la herramienta debe proporcionar soporte tanto para el modelado como el almacenamiento del proceso actual y del proceso mejorado.
- Facilitar la implementación de la mejora: la herramienta debe proporcionar una estrategia y guía en las actividades que deben ser realizadas a través de la implementación de la iniciativa de mejora de procesos, proporcionando una definición clara de roles y sus actividades asociadas.
- Bajo costo: la herramienta no debe representar una gran inversión para la compañía.
- Autoformación: la herramienta debe proporcionar una formación esencial sobre proceso y mejora de procesos.
- Comunicación eficiente: la herramienta debe proporcionar asistencia y soporte a la organización durante todas las fases del ciclo de implementación de la mejora.
- Información útil: la herramienta debe proporcionar información útil y visible para la ejecución de la mejora de procesos de software, tal que la información clave esté disponible en tiempo y a la persona adecuada para que pueda ser utilizada en la toma de decisiones.

6.5 Grupos de Investigación en México

En esta sección se muestra una visión general de grupos de investigación que están desarrollando investigaciones en este campo.

6.5.1 CENIDET

La investigación en el área de mejora de procesos en el Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), se centra por una parte en personas y por otra parte en herramientas y equipo.

En la línea de personas, están realizando investigación sobre arquitectura de software, donde proponen el método *Architectural and Group Development* (AGD), método de desarrollo de software que es guiado por la arquitectura del producto, y su proceso es altamente guiado por un grupo dinámico (González-García & Martínez Enríquez, 2014).

En cuanto a herramientas y equipos, están realizando investigación sobre el uso de ontologías en repositorios de proyectos, donde proponen una solución al problema del manejo de metadatos y el problema semántico de la tarea de integración de datos mediante la técnica extraer, transformar y cargar (ETL) para pre-procesar almacenes de datos de proyectos de software mediante el uso de ontologías (González-García & Fernández Bonilla, 2014). Más recientemente, su investigación está enfocada en el uso de minería de datos en la ingeniería de software, donde proponen el ambiente AMDADS que soporte el proceso de minería de datos (*dotProject,*

RapidMiner, MantisBT y OpenRefine). Este ambiente permite que las herramientas de minería de datos trabajen de forma conjunta evitando incompatibilidad de las herramientas y tareas repetitivas, y facilitando la gestión de proyectos y el seguimiento de incidencias encontradas (Pérez-Luna & González-García, 2016).

6.5.2 CICESE

La investigación en el área de mejora de procesos en el departamento de ciencias de la computación del Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE), se centra en las personas, donde realizan investigación para facilitar la implementación de mejora de procesos, por una parte proponiendo técnicas gráficas que soporten el aprendizaje de modelos de referencias mediante un conjunto de diagramas para visualizar modelos de referencia de procesos (Espinosa-Curiel, Rodríguez-Jacobo & Fernández-Zepeda, 2010). Además, analizan factores que pueden afectar la implementación de iniciativas de mejora de procesos presentando un marco de trabajo de 123 factores agrupados en 6 categorías que pueden influenciar a las iniciativas de mejora en empresas pequeñas, además plantean una metodología para la evaluación y control de los factores (Espinosa-Curiel, Rodríguez-Jacobo & Fernández-Zepeda, 2011).

En esta línea están investigando además las competencias de los implicados (*stakeholders*) requeridas para llevar a cabo iniciativas de mejora de manera exitosa, en esta investigación proponen un marco de

trabajo que define competencias para siete roles involucrados en las iniciativas de mejora de procesos, así como el nivel de experiencia requerido por cada rol para cada competencia (Espinosa-Curiel,Rodríguez-Jacobo & Fernández-Zepeda, 2011b). Más recientemente su investigaciones se centra en el entendimiento de la mejora de procesos en empresas pequeñas (Espinosa-Curiel,Rodríguez-Jacobo & Fernández-Zepeda, 2016). Además de la investigación en mejora de procesos, este grupo de investigación está centrando sus esfuerzos en investigar sobre equipos, específicamente en cambios en factores como comunicación e interacciones sociales en equipos durante su transformación de equipos tradiciones a equipos ágiles (Espinosa-Curiel et, al., 2018).

6.5.3 CIMAT

La investigación en el área de mejora de procesos en la Unidad Zacatecas del Centro de Investigación en Matemáticas (CIMAT) abarca los tres aspectos considerados en este capítulo. En el aspecto de modelos y estándares se han centrado por un aporte en el desarrollo de soporte para la correcta implementación de mejoras en los procesos, definiendo métodos para la implementación de mejora de procesos bajo entornos multimodelo, línea en la que han propuesto metodologías y modelos que apoyan en las actividades relacionadas con la implementación de mejoras en los procesos de software, siempre enfocando en la reducción de la resistencia al cambio, por ejemplo: el análisis de ventajas en el uso de entornos multimodelo (Muñoz, et al.,

2011); el desarrollo de un método para evaluación del rendimiento de los procesos (Muñoz, et al., 2013), establecimiento de entornos multimodelo para mejorar los procesos de las organizaciones, donde proponen un método para el establecimiento de entornos multimodelo basado en los objetivos de negocio de la organización (Muñoz, Mejía & Gasca-Hurtado, 2014). Más recientemente están desarrollando investigaciones relacionadas con la aplicación de ingeniería de software en muy pequeñas organizaciones, donde están presentando tanto la estrategia seguida como un conjunto de experiencias en la implementación del estándar ISO/IEC 29110 en muy pequeñas organizaciones (Laporte et al., 2017), (Muñoz, Mejía & Laporte., 2018), (Muñoz, Mejía & Laporte., 2018b), y entornos DevOps (Muñoz & Díaz, 2017), así como la organización de áreas de proceso de CMMI v1.3 en su nivel 2 mediante el análisis de sus dependencias (Mejía, Muñoz & González, 2017) .

En el aspecto de personas, han desarrollado investigaciones enfocadas en el entendimiento de PyMEs para la implementación de mejoras, donde presentan un método que permite entender el entorno de las organización para poder identificar hallazgos que conduzcan a una mejora de procesos exitosa (Mejía, et. al, 2013); un método para dirigir el esfuerzo de las organizaciones en la implementación de mejoras (Muñoz, Mejía & Valtierra, 2015), desarrollo de estrategias para la implementación de mejoras de acuerdo al contexto de la

organización (Muñoz et al., 2016), y uso de TSPi para la gestión de proyectos en entornos *Outsourcing* (Mejía, García & Muñoz, 2013).

Más recientemente, están desarrollando investigaciones relacionadas con la formación de recursos humanos que cubran los requisitos de la industria del software al trabajar bajo modelos y/o estándares (Muñoz et al., 2016); (Muñoz et al., 2016b) (Muñoz et al., 2018), así como en la formación de equipos altamente efectivos mediante el uso de gamificación en donde proponen un nuevo modelo para la integración de equipos altamente efectivos mediante el uso de gamificación en entornos virtuales, TSP y estilos interactivos, brindando un camino atractivo a usuario para identificación de roles (Muñoz et al., 2017), (Muñoz et al, 2017b).

En el aspecto de herramientas y equipo, han desarrollado investigaciones enfocadas en la gestión del conocimiento para soportar entornos multimodelo en la mejora de procesos en donde introducen el uso de ontologías en la mejora de procesos (Muñoz et al., 2014), (Mejía, Muñoz & Muñoz, 2016). Más recientemente están proponiendo herramientas para la optimización de procesos mediante el uso de prácticas ágiles (Muñoz et al, 2016), implementación y uso de metodologías ágiles (Muñoz et al., 2017) y la implementación de análisis de datos para reforzar la mejora de procesos (Mejía, Íñiguez & Muñoz, 2017).

6.5.4 UNAM

La investigación en el área de mejora de procesos en la Universidad Nacional Autónoma de México, UNAM, ha dado como frutos importantes el Modelo de Procesos para la Industria del Software (MoProSoft), presentado antes en este capítulo. El grupo de investigadores también ha colaborado en el proyecto COMPETISOFT, orientado a fomentar la competitividad de la pequeña y mediana industria del software en Iberoamérica (Oktaba, Piattini, Pino, Orozco, & Alquicira, 2008).

El trabajo del grupo también ha dado como resultado el proyecto KUALI-BEH, que describe métodos y prácticas para el desarrollo, mantenimiento e integración de software y mecanismos para realizar proyectos de software (Oktaba, Morales, & Dávila, 2012), mismo que ha sido reconocido por el *Object Management Group* y que se integró a la propuesta de ESSENCE, liderada por Ivar Jacobson (SEMAT, 2014). Por último, el grupo ha trabajado fuertemente en la formación de recursos humanos mediante el desarrollo del Método Inicial de Desarrollo de Software, que se ofrece en cursos de Ingeniería de Software y que está basado en el método y prácticas propuestas por KUALI-BEH (Ibargüengoitia & Oktaba, 2014).

6.5.5 UAZ

La investigación en el área de mejora de procesos en la Universidad Autónoma de Zacatecas (UAZ), se ha enfocado en brindar capacitación

de recursos humanos de licenciatura y de maestría en PSP, TSP, el método Inicial de desarrollo de software, los perfiles de entrada y básico del ISO/IEC 29110, *Scrum* y *Kanban*. Además, se organiza la escuela de verano, cuyos temas se enfocan en los procesos de desarrollo de software de calidad, la arquitectura de software y los métodos ágiles.

6.6 Conclusiones

La mejora de procesos de software es un área dentro de la ingeniería de software que se enfoca en introducir una cultura de mejora continua en las organizaciones, mediante el uso de modelos y estándares de procesos como referencia. Sin embargo, se debe tener en cuenta al recurso humano, ya que los procesos por sí solos no son nada sin personas que los ejecuten; por tal motivo es necesario esta llevar a cabo la investigación del impacto de las personas en la implementación de procesos. Así mismo, se hace indispensable el desarrollo de herramientas y marcos de trabajo para facilitar la implementación de procesos en las organizaciones de desarrollo de software.

Hoy en día la mejora de procesos de software toma especial relevancia, ya que el software se está convirtiendo en el núcleo de la mayor parte de la industria, por lo tanto, es indispensable garantizar la calidad del mismo. En este contexto, se resalta que la calidad del producto está directamente relacionada con la calidad del proceso utilizado para su desarrollo.

A lo largo de este capítulo se han analizado tres elementos importantes en la mejora de procesos: modelos y estándares, personas y herramientas/equipo, que en conjunto permiten mejorar la competitividad y capacidad de las organizaciones. Además estos elementos caracterizan las áreas de investigación a desarrollar dentro de esta área y en las cuales se está reforzando con nuevas tecnologías emergentes como uso de modelos ontológicos, análisis de datos, gamificación, modelos matemáticos, entre otros.

Es por esto que existen en México grupos de investigación enfocados en estas temáticas, cuyo esfuerzo está enfocado en brindar apoyo a las empresas mexicanas para la resolución de problemas, tal que, se vea reflejada en una cultura de mejora continua así como en la calidad de los procesos que tienen impacto directamente en la calidad de los productos y servicios desarrollados u ofrecidos por éstas.

Referencias

1. Cuevas, G. (2002). *Gestión del Proceso Software*. Editorial Universitaria Ramón Areces 2002.
2. Chrissis M., Konrad M. & Shrum S. (2011) *CMMI for Development: Guidelines for Process Integration and Product Improvement* (3rd Edition) (SEI Series in Software Engineering).
3. CMMI Institute (2018). *CMMI Adoption and Transition Guidance V2.0 Abstract*. CMMI Institute.

4. CMMI Institute (2018b). *CMMI Levels of Capability and Performance.* [En línea]. Disponible: https://cmmiinstitute.com/learning/appraisals/levels.

5. CMMI Institute. Published Appraisal Results. [En línea]. Disponible: https://sas.cmmiinstitute.com/pars/pars.aspx. (2018c)

6. Cohn M. (2014). *Getting Agile with Scrum*. [En línea]. Disponible: https://www.mountaingoatsoftware.com/uploads/presentations/Getting-Agile-With-Scrum-Norwegian-Developers-Conference-2014.pdf

7. Curtis B. & Hefley W. (2009). *The People CMM: A Framework for Human Capital Management* (2nd Edition). Addison-Wesley Professional.

8. Diario Oficial de la Federación, (2016) DOF 23/09/2016. Declaratoria de vigencia de la Norma Mexicana NMX-I-059-2-NYCE-2016. http://dof.gob.mx/nota_detalle.php?codigo=5453674&fecha=23/09/2016.

9. Espinosa-Curiel I.E., Rodríguez-Jacobo J. & Fernández-Zepeda J.A. (2010) Graphical Technique to Support the Teaching/Learning Process of Software Process Reference Models, A. Riel et al. (Eds.): *EuroSPI 2010*, CCIS 99, pp. 13–24, © Springer-Verlag Berlin Heidelberg.

10. Espinosa-Curiel I.E., Rodríguez-Jacobo J. & Fernández-Zepeda J.A. (2011) A framework for evaluation and control of the factors that influence the software process improvement in small organizations. *J. Softw.: Evol. and Proc*. 2013; 25:393–406 Published online 23 September 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/smr.569.

11. Espinosa-Curiel I.E., Rodríguez-Jacobo J. & Fernández-Zepeda J.A. (2011b) A Competency Framework for the Stakeholders of a Software Process Improvement Initiative. *Proceedings of ICSSP'11*, May 21–22, 2011, Waikiki, Honolulu, HI, USA Copyright 2011 ACM 978-1-4503-0730-7/11/05.

12. Espinosa-Curiel I.E., Rodríguez-Jacobo J. & Fernández-Zepeda J.A. (2016) Understanding SPI in small organizations: a study of Mexican software enterprises. *J. Softw. Evol. and Proc*. 2016; 28: pp. 372–390 Published online 5 April 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/smr.1775.

13. Espinosa-Curiel IE, Rodríguez-Jacobo J, Vázquez-Alfaro E, Fernández-Zepeda JA, Fajardo-Delgado D. (2018) Analysis of the changes in communication and social interactions during the transformation of a traditional team into an agile team. *J Softw Evol Proc*. 2018;30:e1946. https://doi.org/10.1002/smr.1946.

14. Gómez, O., Aguileta, A., Gómez, G. y Aguilar R. (2014) Estudio del Proceso Software Personal (PSP) en un entorno académico. *ReCibe*, 3(2), 28. Retrieved from https://www.redalyc.org/pdf/5122/512251567001.pdf

15. González, M. & Fernández, O.D. (2014) Metodología ETL para el procesamiento de datos en repositorios de proyectos de software usando ontologías, *Encuentro Nacional de Ciencias de la Computación, enc2014*.cicese pp. 1-5.

16. González, M. & Martínez E. (2014). The Architectural and Group Development Method: an Experimentation, *Encuentro Nacional de Ciencias de la Computación (ENC 2014),* pp. 1-10.

17. Humphrey, W. S. (2000) *The Team Software Process* (TSP). Hanscom, MA.

18. Humphrey, W. (2005) *PSP: A Self-Improvement Process for Software Engineers. The complete PSP*. Addison-Wesley Professional.

19. Humphrey, W. S. (2006) *TSP Leading a Development Team* (1st.). Upper Saddle, NJ: Pearson Education.

20. Humphrey, W. S., Chick, T. A., Nichols, W., & Pomeroy-Huff, M. (2010) *Team Software Process Body of Knowledge.* Hanscom, MA.

21. Humphrey, W. S., & Thomas, W. R. (2010) *Reflections on Management: How to Manage Your Software Projects, Your Teams, Your Boss, and Yourself* (1st edition). Addison Wesley Professional.

22. Ibargüengoitia, G., & Oktaba, H. (2014) Identifying the Scope of Software Engineering for Beginners Course Using ESSENCE. In C. M. Zapata (Ed.), *Software Engineering: Methods, Modeling and Teaching*, Vol. 3, p. 107. Medellín, Colombia: Universidad Nacional de Colombia.

23. Laporte Y. C., Muñoz M., Mejía J. & O'Connor R.V. (2017). Applying Software Engineering Standards in Very Small Entities. *IEEE SOFTWARE*, January/February 2018, pp. 99-103.

24. McAndrews, D. (2000) *The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices* (No. CMU/SEI-2000-TR-015). Pittsburgh, Pa.

25. McAndrews, D. (2001) *The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices.* Hanscom, MA.

26. Mejía J., Íñiguez F., and Muñoz M. (2017) Data Analysis for Software Process Improvement: A Systematic Literature Review Springer International Publishing AG 2017 Á. Rocha et al. (eds.), Recent *Advances in Information Systems and Technologies, Advances in Intelligent Systems and Computing* 569, DOI 10.1007/978-3-319-56535-4_5.

27. Mejía J. García A. and Muñoz M. (2013) *TSPi to manage software projects in outsourcing environments*, New Perspectives in Information System and Technologies, Alvaro Rocha, Ana Maria Correia, Tom Wilson and Karla. Astro, SPRINGER-VERLAG BERLIN HEIDELB, Vol. 206.

28. Mejia J., Muñoz M. & Gonzalez M., (2017) Organization of the process areas of CMMI-Dev v1.3 level 2 through of its dependencies, *12th Iberian Conference on Information Systems and Technologies* (CISTI), 2017 pp. 2130-2136.

29. Mejía J., Muñoz M., Muñoz E., Calvo-Manzano J. A. (2013) Understanding the environment of small and medium enterprises to implement software processes improvement, *European System,*

Software & Service Process Improvement & Innovation EuroSPI 2013, Vol.2013, Pag.1-11. ISBN 978-87-7398-155-9.

30. Mejía J., Muñoz E., Muñoz M. (2016) Reinforcing the applicability of Multi-model Environments for Software Process Improvement using Knowledge Management. *Science of Computer Programming*, Elsevier, Vol. SCICO, pp. 1-13 Doi:10.1016/j.scico.2015.12.002.

31. Muñoz E., Muñoz M., Capón E., Mejía J. (2014) Knowledge management in process improvement and best practices sharing. ISSN: 1548-0992, *IEEE LATIN AMERICA TRANSACTIONS*, Vol.12, pp.469-474. ISSN: 1548-0992.

32. Muñoz M. & Díaz O., (2017) DevOps: Foundations and Its Utilization in Data Center. Springer International Publishing AG 2017 J. Marx Gómez et al. (eds.), Engineering and Management of Data Centers, Service Science. *Research and Innovations in the Service Economy*, DOI 10.1007/978-3-319-65082-1_10.

33. Muñoz, M., Gasca-Hurtado, G.P., Valtierra, C. (2014) Caracterizando las Necesidades de las Pymes para Implementar Mejoras de Procesos Software: Una Comparativa entre la Teoría y la Realidad. *Revista Ibérica de Sistemas y Tecnologías de Información*, Porto, N.spe1, p. 1-15, mar. 2014. Disponible en <http://www.scielo.mec.pt/scielo.php?script=sci_arttext&pid=S1646989520140001000002&lng=pt&nrm=iso>. Acceso: 07 ago. 2018. http://dx.doi.org/10.4304/risti.e1.1-15.

34. Muñoz M., Hernández L., Mejía J., Peña A., Rangel N., Torres C., and Sauberer G. (2017b) *A Model to Integrate Highly Effective Teams for*

Software Development. Springer International Publishing AG 2017 J. Stolfa et al. (Eds.): EuroSPI 2017, CCIS 748, pp. 613–626.

35. Muñoz M., Mejía, J., Calvo-Manzano J.A., Cuevas, G., San Feliu T. (2013) Method to Evaluate Process Performance Focused on Minimizing Resistance to Change. *International Journal of Human Capital and Information Technology Professionals*, 4(2), 1-15, April-June 2013.

36. Muñoz M., Hernández L., Mejía J., Gasca-Hurtado G.P. and Gómez-Álvarez M.C. (2017) State of the Use of Gamification Elements in Software Development Teams. Springer International Publishing AG. J. Stolfa et al. (Eds.): *EuroSPI 2017*, CCIS 748, pp. 249–258, (2017).

37. Muñoz M., Mejía, J., Alor G., Calvo-Manzano J.A., San Feliu T. (2011) Advantages of using a multi-model environment in software process improvement, 2011 *Electronics, Robotics and Automotive Mechanics Conference*, pp 397-402.

38. Muñoz M., Mejía, J., Miramontes J., Calvo-Manzano J.A. and Corona B. (2016) Establecimiento del estado del arte sobre el aligeramiento de los procesos de software. *Revista Ibérica de Sistemas y Tecnologías de Información*, No.17, Vol. 03/2016, ISSN: 16469895, pp. 16-25.

39. Muñoz M., Mejía, J., Calvo-Manzano J.A., San Feliu T., Corona B. and Miramontes J. (2017) Diagnostic Assessment Tools for Assessing the Implementation and/or Use of Agile Methodologies in SMEs: An Analysis of Covered Aspects. *Software Quality Professional*, 19(2), ISSN: 15220542, pp 16-27.

40. Muñoz M., Mejía J. & Laporte Y.C. (2018) Implementation of ISO/IEC 29110 in Software Development Centers from Mexican Universities: An experience of the Zacatecas State. *Revista Ibérica de Sistemas y Tecnologías de Información*, pp. 43-54.

41. Muñoz M., Mejia J., Laporte C.Y. (2019) Reinforcing Very Small Entities Using Agile Methodologies with the ISO/IEC 29110. In: Mejia J., Muñoz M., Rocha Á., Peña A., Pérez-Cisneros M. (eds) Trends and Applications in Software Engineering. *CIMPS 2018. Advances in Intelligent Systems and Computing*, vol 865. Springer, Cham

42. Muñoz M., Mejía, J., Gasca-Hurtado, G.P. A Methodology for Establishing Multi-Model Environments in Order to Improve Organizational Software Processes. International Journal of Software Engineering and Knowledge Engineering Vol. 24, No. 6 (2014) 909–933 #.c World Scientic Publishing Company DOI: 10.1142/S0218194014400038. (2014).

43. Muñoz M., Mejía J., Gasca-Hurtado G.P., Gómez-Álvarez M.C. & Durón B. (2016) Method to Establish Strategies for Implementing Process Improvement According to the Organization's Context, System, Software and Services Process Improvement C. KREINER ET AL. (EDS.), SPRINGER INTERNATIONAL PUBLISH, Vol. 633, Pps. 312-324.

44. Muñoz-Mata M., Mejía-Miranda J., Valtierra-Alvarado C. (2015) Helping Organizations to Address their Effort toward the Implementation of Improvements in their Software Process *Revista Facultad de Ingeniería*, Universidad de Antioquia, Vol.77, pp.115-126.

45. Muñoz M., Peña A., Mejía J. and Lara G. (2016) Coverage of the University Curricula for the Software Engineering Industry in Mexico ISSN:1548-0992, *IEEE Latin America Transactions*, Vol.14, pp.2383-2389.

46. Muñoz M., Peña A., Mejía J. and Lara G. (2016b) Actual State of the Coverage of Mexican Software Industry Requested Knowledge Regarding the Project Management Best Practices. *Computer Science and Information Systems* (ComSIS), Vol.13. pp.849-873.

47. Muñoz M., Peña A., Mejía J. and Lara G. (2018) ISO/IEC 29110 and curricula programs related to Computer Science and Informatics in Mexico: Analysis of practices coverage. Springer International Publishing AG 2018 J. Mejía et al. (eds.), *Trends and Applications in Software Engineering, Advances in Intelligent Systems and Computing* 688, https://doi.org/10.1007/978-3-319-69341-5_1.

48. Nichols, W. R., Kasunic, M., & Chick, T. A. (2013) *TSP Performance and Capability Evaluation* (PACE): Team https://doi.org/10.1007/978-3-319-69341-5_1Preparedness Guide. Hanscom, MA.

49. Nichols, W. R., & Salazar, R. (2009) *Deploying TSP on a National Scale: An Experience Report from Pilot Projects in Mexico.* Retrieved http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA501742.

50. NYCE. EMPRESAS DICTAMINADAS EN LA NORMA NMX-I-059/02-NYCE (MoProSoft). agosto 8, 2018, de NYCE Sitio web: https://www.nyce.org.mx/wp-content/uploads/2018/02/PADRON-DE-EMPRESAS-DICTAMINADAS-MoProSoft.pdf. (2018).

51. Oktaba Hanna, Piattini Mario, Pino Francisco J., Orozco, María J., Alquicira Claudia. (2009) *COMPETISOFT: Mejora de Procesos Software para Pequeñas y Medianas Empresas y Proyectos*. Alfaomega.

52. Oktaba, H., Morales, M., & Dávila, M. (2012) *KUALI-BEH: Software Project Common Concepts*. Recuperado de http://scholar.google.com.mx/scholar?hl=en&q=Kuali-Beh&btnG=&as_sdt=1%2C5&as_sdtp=#0.

53. Organisation Internationale de Normalisation (ISO) (2011). ISO/IEC 15504-9:2011. [En línea]. Disponible: https://www.iso.org/obp/ui/ - iso:std:iso-iec:ts:15504:-9:ed-1:v1:en.

54. Pérez-Luna E. & González-García M. (2016) Componentes de acoplamiento para la infraestructura de soporte de minería de datos de desarrollo de software. *Revista de Sistemas Computacionales y TIC's*. Diciembre 2016 Vol.2 No.6 99-111.

55. Schwaber K. & Sutherland J. (2017) The Scrum Guide. [En línea]. Disponible: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100. (2017).

56. Scrum México (2018). SCRUM MÉXICO. http://scrum.org.mx/.

57. Secretaría de Economía (2016). Centros de Desarrollo Certificados/Verificados Vigentes en Modelos de Calidad. Retrieved from https://prosoft.economia.gob.mx/doc/PADRON_CENTRO DE DESARROLLO VIGENTE_2016_abr-18.pdf.

58. SEMAT (2014). *Essence Standard*. Retrieved July 13, 2018, from http://semat.org/essence-standard.

59. Software Engineering Institute (2018). SEI-Certified Individuals. Retrieved from https://www.sei.cmu.edu/education-outreach/credentials/certified-individuals/index.cfm.

60. Viega J. & McGraw G. (2011) *Building Secure Software: How to Avoid Security Problems the Right Way*, Addison-Wesley Professional Computing Series.

Capítulo 7.
Investigación en el área de Métodos y Modelos en Ingeniería de Software

Carlos Alberto Fernández y Fernández, *Universidad Tecnológica de la Mixteca.*

7.1 Introducción

Esta área de conocimiento identificada en el SWEBOK 3.0 (IEEE, 2014) busca que los métodos y modelos de Ingeniería de Software apoyen a la realización sistemática de actividades, repetibles con el objetivo de lograr el éxito en el desarrollo del proyecto de software. Un método ofrece una aproximación sistemática y organizada para el desarrollo de software. Estos métodos pueden ser heurísticos, métodos formales, métodos basados en prototipos y métodos ágiles. Por otro lado, el modelado es una técnica de abstracción y representación que puede ser usado para representar aspectos del software. Un modelo puede servir como un medio tanto como un medio comunicación entre las partes, como en una forma de especificación y documentación. Los modelos se pueden clasificar en: modelos de información, de comportamiento, y de estructura.

Existen varios intentos de recopilar el trabajo en Ingeniería de Software en México. En 2013 se presentó un primer reporte por la entonces Red Temática Mexicana en Ingeniería de Software (REDMIS) (Juárez-Ramírez et al., 2013). En 2016 se hizo una recopilación de información trabajada por algunos grupos de investigación en Ingeniería de Software, la cual tomó como base el cuerpo de conocimientos establecido en SWEBOK antes citado. Se toma como base esta información, la cual ha sido enriquecida mediante información obtenida del repositorio nacional.

7.2 Métodos en la Ingeniería de Software

Cómo ya se mencionó, existen distintos categorías de métodos ya las clasificaciones pueden variar, pero el SWEBOK los organiza en: Heurísticos, formales, de prototipado y ágiles.

7.2.1 Métodos Heurísticos

Estos métodos de ingeniería de software están basados en la experiencia y son ampliamente ocupados. Se identifican tres categorías (Budgen, 2003; Somerville, 2011)

- Los métodos de análisis y diseño estructurados, -como el Método de análisis y diseño estructurado (Downs, Clare, & Coe, 1988),
- Métodos de modelado de datos – orientados a la representación de los datos que usará el software, con diferentes tipos de datos tales como jerárquico, relacional, y objeto relacional entre otros.

- Métodos orientados a objetos. - siendo el más conocido el Proceso Unificado de Desarrollo de Software (Booch, Rumbaugh, & Jacobson, 1999) y su versión reducida OpenUP (Balduino, 2007).

Los métodos heurísticos, en particular los orientados a objetos, han sido vistos por muchos como demasiado costosos de ser aplicados sobre todo para software mediano y pequeño, a pesar que dichos métodos pueden ser configurados para solo aplicar las etapas que se consideren relevantes para obtener los mejores resultados.

7.2.2 Métodos Formales

Son métodos basados en matemáticas que sirven para especificar, desarrollar y verificar el software (Wing, 1990). Los métodos formales proporcionan un dominio sintáctico (i.e., la notación o conjunto de símbolos del método), un dominio semántico (el universo de elementos), y un conjunto de reglas precisas definiendo como un objeto puede satisfacer una especificación (Spivey, 1989). Adicionalmente, una especificación es un conjunto de sentencias construidas usando la notación en el dominio sintáctico y como éste representa un subconjunto del dominio semántico (Fernandez-y-Fernandez, 2010). Pueden ser usados para la verificación de modelos de software.

Es posible identificar los siguientes tipos (Budgen, 2003; Somerville, 2011; Wing, 1990):

- Lenguajes de especificación;
- Derivación y refinamiento de programas;

- Verificación formal; e
- Inferencia lógica.

7.2.3 Métodos de Prototipado

Estos métodos parten de la actividad de ir creando prototipos inclusive no funcionales inicialmente; conforme dichos prototipos se van revisando y enriqueciendo se va llegando a una versión funcional con un cierto porcentaje de características deseables implementadas (Budgen, 2003; Somerville, 2011). Los métodos de prototipado han ido evolucionado desde su propuesta inicial en incicios de la década de los 70s (Grimm, 1998). Existen diferentes tipos de métodos de prototipado, aunque básicamente se reconocen dos:

- Prototipado desechable y
- Prototipado evolutivo.

El prototipado desechable generalmente parte de prototipos no funcionales, que pueden ser diseños en papel o imágenes estáticas sin funcionalidad. El prototipo es construido con la idea de que será descartado una vez que se tenga la idea del concepto y la aprobación del cliente (Crinnion, 1991).

Por otra parte, el prototipado evolutivo intenta partir de un robusto prototipo inicial y con una funcionalidad base e irlo refinando constantemente. El prototipo inicial se considera la base del resto de sistema que se expandirá gradualmente (Asimov & US, 1997).

7.2.4 Métodos Ágiles

A partir de una tendencia de métodos de desarrollo de software tradicionales considerados demasiado pesados y no adecuados para muchos tipos de desarrollos de software, surge una tendencia a lo que se conoce como métodos ágiles (Boehm & Turner, 2003). Aunque los métodos ágiles son considerados ligeros gracias a que, entre otras cosas, manejan ciclos de desarrollo cortos y un mayor involucramiento con el cliente, la mayoría de los métodos ágiles trabajan más sobre aspectos de administración del proyecto que con técnicas de desarrollo de software.

El término ágil fue popularizado por el Manifiesto para el Desarrollo de Software Ágil (Beck, Beedle, Van, & Cockburn, 2001), el cual incluyó una serie de valores y principios que forman la base de los métodos de esta categoría. El manifiesto propone los siguientes puntos:

- Individuos e interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.
- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio sobre seguir un plan.

El método ágil más popular en la actualidad es *Scrum* (Schwaber & others, 1995), pero existen también otros métodos tales como *Rapid Application Development* (RAD) (Graham, 1998), *eXtreme*

Programming (XP) (Beck, 2000), y *Feature-Driven Development* (FDD) (France, Ghosh, Dinh-Trong, & Solberg, 2006).

7.2.5 Investigación en el área de Métodos en México

En México, grupos en la UNAM han estado continuamente haciendo propuesta de mejores a los procesos de software, con un énfasis hacia las medianas y pequeñas empresas (Oktaba, 2003, 2008; Oktaba, Piattini, Pino, & Orozco, 2009).

Por su parte la Universidad Tecnológica de la Mixteca (UTM) ha realizado propuestas en esta área, principalmente en el área de métodos formales, proponiendo un método formal mediante un álgebra de procesos (Fernandez-y-Fernandez, 2010, 2012; Fernandez-y-Fernandez & Simons, 2014). Adicionalmente, impulsando prácticas que incluyan el uso de métodos formales en conjunto con las prácticas populares en la industria del software en el país (Fernandez-y-Fernandez, 2013).

Centros como el CIMAT tienen conjunto amplio de artículos orientados a los métodos para el desarrollo de software. Por mencionar algunos: han trabajado en un método de mejora para la selección de prácticas para el desarrollo de software (Sandoval, 2016); un proceso para el desarrollo de software de pequeñas y medianas empresas tomando las mejores prácticas desde tomando en cuenta el factor humano (Mirna, Jezreel, Brenda, & Claudia, 2014); una propuesta de mejora de los procesos en un organismo gubernamental (Miranda et al.,

2014); un método para la mejora del proceso de software tomando en cuenta el contexto de la organización (Muñoz, Mejia, Hurtado, Gómez-Álvarez, & Durón, 2016); un método para el desarrollo de videojuegos guiado por la experiencia del usuario (González-Salazar, Mitre-Hernández, & Lara-Alvarez, 2017); en metodologías ágiles proponen una herramienta de diagnóstico de la implementación y/o uso de las metodologías ágiles en pequeñas y medianas empresas (Muñoz et al., 2017) y el uso de prácticas ágiles para el desarrollo de un sistema ciber-físico (Hernández-Reveles, Sobrevilla-Dominguez, Velasco-Elizondo, & Soriano-Grande, 2016).

Es posible encontrar en la literatura propuestas en métodos para la Ingeniería de Software para usos atípicos, como la propuesta generada en el Instituto Nacional de Electricidad y Energías Limpias (INEEL) de donde surge una propuesta de métodos para el desarrollo de software para un centro de investigación (Popoca & Jiménez, 1999). Por otro lado en el CICESE se presenta un estudio donde se analiza el proceso de desarrollo de software para equipos distribuidos (García Mireles, 2000).

7.3 Modelos en la Ingeniería de Software

Un modelo es una simplificación de la realidad que describe completamente un sistema desde una perspectiva particular. El modelado es importante porque ayuda al equipo a visualizar,

especificar, construir y documentar la estructura y el comportamiento de la arquitectura del sistema.

El modelado de software ayuda a los ingenieros de software a entender y comunicar diferentes aspectos del mismo entre ellos, a los usuarios y a los que tienen algún interés en éste. Un modelo debe representar lo esencial, mostrando una perspectiva dependiendo del tipo de modelo y debe permitir la comunicación efectiva y sin ambigüedad (France et al., 2006; Mellor & Balcer, 2002; Somerville, 2011). De acuerdo a lo anterior, cada modelo es una descripción parcial para comunicar y dar una perspectiva específica. Un modelo puede estar formado de sub-modelos para disminuir la complejidad y cada modelo puede estar en diferentes notaciones.

Los modelos pueden ser textuales o visuales. El Modelado Visual es el modelado de una aplicación usando notaciones gráficas y puede comunicar de manera más efectiva una abstracción a un grupo de desarrolladores.

El estándar de facto actual es el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) (Booch, Rumbaugh, & Jacobson, 1998, 2005; OMG, 2015), pero existen otros lenguajes o variaciones propuestas. Aunque existen otros, la mayoría de los lenguajes pueden considerarse dentro de una de los siguientes tipos: modelo contextual, modelo de comportamiento, modelo estructural y modelo de interacción.

7.3.1 Modelo de Contexto

Un modelo de contexto sirve para representar las fronteras de un sistema de software. Se utilizan frecuentemente para establecer ideas iniciales del software y su posible interacción con otros sistemas (Somerville, 2011). Dependiendo del tipo de sistema de software a desarrollar, será más fácil o difícil identificar la frontera entre lo que se tiene que construir y el medio ambiente con el que tiene que interactuar. Esta interacción con otros sistemas es necesaria para evitar la duplicidad de actividades y datos. La desventaja es la posible dificultad y rigidez que puede venir de la implementación de dicha interacción.

7.3.2 Modelo de Comportamiento

Este tipo de modelo se usa para representar la funcionalidad del software. Se tienen de manera general modelos de flujo de control (del estilo de diagramas de flujo), modelos de flujo de datos y modelos de máquinas de estado (Budgen, 2003; Mellor & Balcer, 2002; Somerville, 2011). UML considera dentro de estos diagramas los siguientes (Booch, Rumbaugh, & Jacobson, 2004):

Diagrama de actividades. Representa flujos de trabajo por medio de una secuencia de actividades y acciones, pudiendo representarse estructuras de decisión, iteración y concurrencia de las acciones. Pueden ser usados para el modelado de actividades de casos de uso o para el diseño de sistemas embebidos.

Diagrama de máquinas de estados. Este diagrama se basa en el concepto matemático del autómata finito de estados, siendo una variante del diagrama de estados de Harel. La versión de UML ha sido adaptada y extendida representando los posibles estados del elemento representado (un objeto, componente u otro), y sus transiciones posibles. Es útil para expresar los posibles estados de objetos o componentes complejos donde es importante conocer los potenciales estados del elemento.

Diagrama de interacción. Estos últimos son considerados en la categoría de modelo de interacción que se menciona adelante.

7.3.3 Modelo de Estructura

Un modelo estructural trata de representar como está compuesto el software a partir de sus elementos físicos o lógicos. Se usa para representar conexiones relativamente persistentes en los modelos. Dependiendo el nivel del modelo dentro del proceso de desarrollo de software, puede mostrar mayor o menor información que corresponda con la implementación del software (Budgen, 2003; Page-Jones & Constantine, 2000; Somerville, 2011). Algunos de los diagramas usados para este tipo de modelo son de los más populares por los equipos de desarrollo debido a que es posible, si los modelos tienen suficiente información, transformar estos modelos en código en algún lenguaje de programación.

UML cuenta como parte de los modelos de estructura a los siguientes (Booch et al., 2004):

Diagramas de clases. Es quizá el más usado de los diagramas de UML y representa la estructura estática de un sistema mediante clases, atributos, métodos y sus relaciones de herencia, composición y asociación. Puede ser usado desde un nivel conceptual de análisis hasta el nivel de implementación donde represente entidades en un lenguaje de programación específico. En la Figura 7.1, es posible apreciar un ejemplo de este diagrama mostrando un conjunto de clases y sus relaciones de herencia, asociación y composición.

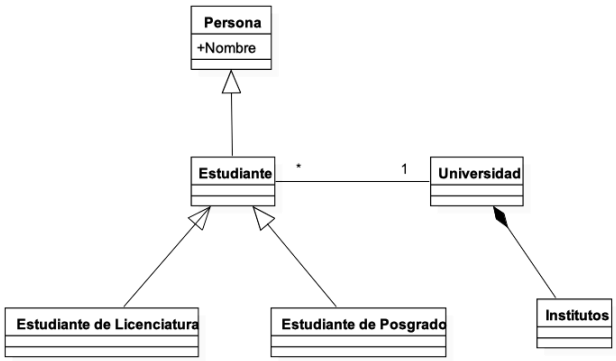

Figura 7.1. Ejemplo de Diagrama de Clases de UML

Diagramas de objetos. Este diagrama muestra una vista de los posibles objetos que pueden ser instanciados en un sistema de software, sus valores y relaciones. Un diagrama de este tipo es una instantánea del sistema en un momento particular y debe representar un estado válido del sistema en un momento determinado. La mayor dependencia del

diagrama de objetos es el diagrama de clases. Todo objeto representado en el diagrama de objetos debe estar definido mediante en el diagrama de clases con los valores correspondiendo a atributos definidos en las clases.

Diagramas de componentes. Se utiliza para mostrar como múltiples componentes están conectados y forman componentes más grandes en los sistemas de software. Un componente típicamente ofrece una o más interfaces de conexión mediante las cuales otros componentes se pueden comunicar con éste. La Figura 7.2 muestra un ejemplo del diagrama, mostrando interfaces y relaciones (dependencia y realizaciones) entre los elementos.

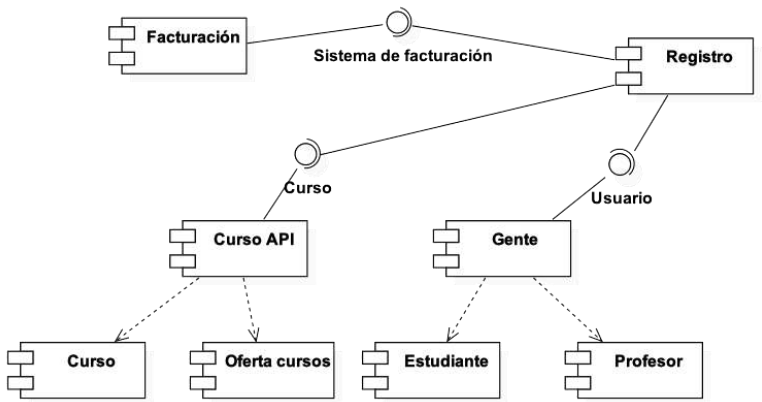

Figura 7.2. Ejemplo de Diagrama de Componentes de UML

Diagramas de paquetes. Dependiendo de la vista que se este representando, un diagrama de paquetes puede mostrar las dependencias entre diferentes paquetes para un modelo de software o la

organización interna de un paquete. Las conexiones entre paquetes son llamadas dependencias funcionales que implican un nivel de visibilidad hacia la dependencia. Los elementos internos de un paquete son normalmente clases, pero puede mostrar otros elementos donde se quiera mostrar la organización de estos.

Diagrama de despliegue. Este diagrama modela el despliegue físico de los elementos de un sistema de software (ver Figura 7.3). Se utiliza para mostrar el diseño de los nodos físicos y los elementos que se ejecutarán en cada nodo, mostrando adicionalmente como estos nodos se encuentran conectados.

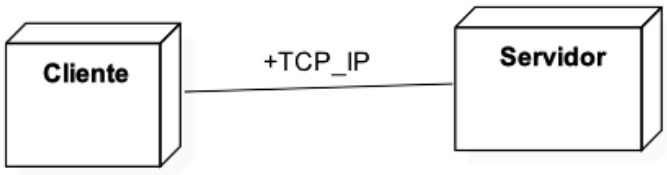

Figura 7.3. Ejemplo de Despliegue de UML

7.3.4 Modelo de Interacción

Los modelos de interacción representan cualquier interacción en el software. Esta interacción puede ser entre componentes del mismo sistema, entre el software y otros sistemas de software o interacción con el usuario (Somerville, 2011). El modelado de la interacción ayuda a identificar y optimizar la comunicación entre los elementos que se comunican, lo que puede ayudar a identificar si se están cumpliendo los requerimientos de usuario y los requerimientos de rendimiento y

confianza. En UML, el modelado de la interacción puede realizarse mediante un conjunto de diagramas (Booch et al., 2004), éstos son:

Diagramas de secuencia. Modelar software usando los diagramas de secuencia, implica modelar las interacciones entre objetos ordenados en el tiempo. La vista de este modelo representa los mensajes que se pasan entre si los objetos a través de líneas de vida paralelas que representan el paso de tiempo (ver Figura 7.4).

Diagramas de comunicación. En estos diagramas -antes llamados diagramas de colaboración- el énfasis del modelo es mostrar un conjunto de objetos y la relación de comunicación entre ellos mediante el paso de mensajes. Debido a que la ubicación de los objetos no indica un orden en particular, los mensajes tienen que ser numerados para entender la cronología de los mismos.

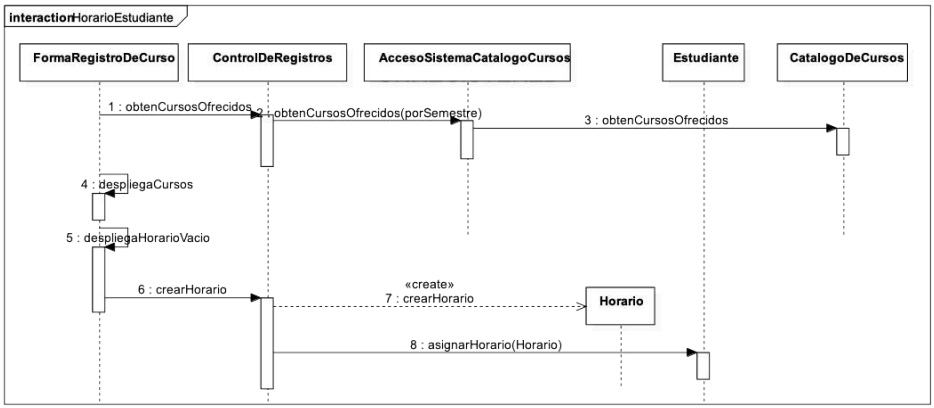

Figura 7.4. Ejemplo de Secuencia de UML

Estos diagramas modelan la interacción entre componentes internos y externos. Por otro lado, el diagrama de casos de uso de UML (ver 7.5), es un diagrama que puede modelar de manera general la interacción entre el sistema de software y otros actores – usuarios o sistemas externos- (Jacobson, 1992).

Figura 7.5. Ejemplo de Diagrama de Casos de Uso de UML

7.3.5 Investigación en el área de Modelos en México

Existe muy poco trabajo en esta área en México; sin embargo, trabajando en modelos de Ingeniería de Software, la Universidad Tecnológica de la Mixteca (UTM) ha diseñado y generado algunos productos relacionados. Por ejemplo, desde una herramienta CASE que genera código C++ a partir de diagramas de clase UML (Fernández-y-Fernández & Cruz Matías, 2003), herramientas para el modelado y verificación de especificación formal a partir de un álgebra de tareas (Fernández-y-Fernández, Quintanar Morales, & Fernández Santos, 2011; Fernández-y-Fernández & Simons, 2011, 2014; Quintanar Morales & Fernández-y-Fernández, 2015).

Existen también muestras del uso de modelado de software para la industria automotriz, un trabajo de la UPAEP donde la UTM ha

estado colaborando (Aguilar-Cisneros, Gulias Matas, Torreblanca, & Fernández-y-Fernández, 2017; Gulias, Torreblanca, Aguilar, & Fernández, 2016). Adicionalmente, la Universidad Autónoma de Aguascalientes (UAA) tiene trabajos hacia el modelado de software para personas con dificultades de aprendizaje (Muñoz-Arteaga, Esparza, Mendoza, & Reich, 2017), desarrollo dirigido por el modelo para ambientes de terapia ocupacional (Reyes, Muñoz-Arteaga, & González-Calleros, 2017), y de modelado de software mediante el uso de patrones, por mencionar algunas de sus aportaciones en esta área del conocimiento (Arteaga, Fuentes, Rodríguez, & Zezzatti, 2008).

7.4 Conclusiones

En este capítulo, se presentó una descripción general del área de conocimiento Métodos y Modelos en Ingeniería de Software identificada en el SWEBOK 3.0. Se hizo referencia a la clasificación de métodos heurísticos, formales, de prototipado y ágiles. Adicionalmente, se describieron los modelos de contexto, de comportamiento, de estructura y de interacción. Para ambos temas se mencionan ejemplos de investigación que realizados por académicos de Ingeniería de Software en México, con la intención, no de presentar una lista exhaustiva, sino de un esfuerzo para identificar el estado de la investigación en esta área de conocimiento.

Referencias

1. Aguilar-Cisneros, J., Gulias Matas, E., Torreblanca, L. F., & Fernandez-y-Fernandez, C. A. (2017). Automotive Safety Requirements Specification. In C. M. Zapata Jaramillo, C. E. Durango Vanegas, & W. Perdomo Charry (Eds.), *Software Engineering: Methods, modeling, and teaching*, pp. 225–243. Bogotá, Colombia: Editorial Bonaventura.
2. Arteaga, J. M., Fuentes, M. D. L. M., Rodriguez, F. Á., & Zezzatti, C. A. O. O. (2008). Extending Pattern Specification for Design the Collaborative Learning at Analysis Level. *In International Workshop on Hybrid Artificial Intelligence Systems*. pp. 543–550.
3. Asimov, I., & US, R. (1997). Evolutionary Rapid Development.
4. Balduino, R. (2007). *Introduction to OpenUP* (*Open Unified Process*).
5. Beck, Beedle, Van, B., & Cockburn. (2001). *The Agile Manifesto*.
6. Beck, K. (2000). *Extreme programming explained: embrace change*. Addison-Wesley Professional.
7. Boehm, B., & Turner, R. (2003). *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Professional.
8. Booch, G., Rumbaugh, J., & Jacobson, I. (1998). *The Unified Modeling Language User Guide* (1st ed.). Addison Wesley.
9. Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The unified software development process*. Reading, Mass: Addison-Wesley.

10. Booch, G., Rumbaugh, J., & Jacobson, I. (2004). *The unified modeling language reference manual* (2nd ed.). Boston: Addison-Wesley.

11. Budgen, D. (2003). *Software Design* (2nd ed.). Pearson Addison Wesley.

12. Crinnion, J. (1991). *Evolutionary systems development: a practical guide to the use of prototyping within a structured systems methodology.* Pitman London.

13. Downs, E., Clare, P., & Coe, I. (1988). *Structured systems analysis and design method: application and context*. Prentice Hall International (UK) Ltd.

14. Fernandez-y-Fernandez, C. A. (2010). *The Abstract Semantics of Tasks and Activity in the Discovery Method*, PhD Thesis, Department of Computer Science. The University of Sheffield.

15. Fernandez-y-Fernandez, C. A. (2012). Towards a New Metamodel for the Task Flow Model of the Discovery Method. In Tendencias en Investigación e Innovación en Ingeniería de Software: un enfoque práctico. *Congreso Internacional de Investigación e Innovación en Ingeniería de Software* (Conisoft 2012). Guadalajara, Jalisco, México.

16. Fernandez-y-Fernandez, C. A. (2013). *Integración de métodos formales en técnicas tradicionales de desarrollo de software con una perspectiva hacia los sistemas web en las empresas*.

17. Fernandez-y-Fernandez, C. A., & Cruz Matías, I. A. (2003). *UMLGEC ++: Una Herramienta CASE para la Generación de Código a partir de Diagramas de Clase UML*. Zacatecas, México: ANIEI.

18. Fernandez-y-Fernandez, C. A., Quintanar Morales, J. A., & Fernández Santos, H. (2011). An IDE to Build and Check Task Flow Models. *Advances in Computer Science and Applications*, (53), pp. 23–33. Retrieved from http://arxiv.org/abs/1107.2683

19. Fernandez-y-Fernandez, C. A., & Simons, A. J. H. (2011). An Algebra to Represent Task Flow Models. *International Journal of Computational Intelligence: Theory and Practice*, 6(2), pp. 63–74.

20. Fernandez-y-Fernandez, C. A., & Simons, A. J. H. (2014). An implementation of the task algebra, a formal specification for the task model in the Discovery Method. *Journal of Applied Research and Technology,* 12(5), pp. 908–918. Retrieved from http://www.jart.ccadet.unam.mx/jart/vol12_5/9_anImplementation.pdf

21. France, R. B., Ghosh, S., Dinh-Trong, T., & Solberg, A. (2006). Model-Driven Development Using UML 2.0: Promises and Pitfalls. *COMPUTER*, pp. 59–66.

22. Garcia Mireles, G. A. (2000). *Measurement process support tool for distributed software teams*. CICESE.

23. González-Salazar, M., Mitre-Hernández, H., & Lara-Alvarez, C. (2017). *Method for Game Development Driven by User-eXperience: A Study of Rework*, Productivity and Complexity of Use. Mechanics, 13, 14.

24. Graham, I. (1998). *Requirements engineering and rapid development : an object-oriented approach*. Harlow, England ; Reading, MA: Addison Wesley.

25. Grimm, T. (1998). The human condition: a justification for rapid prototyping. *Time Compression Technologies*, 3(3), 1–6.

26. Gulias, E., Torreblanca, L. F., Aguilar, J. R., & Fernandez, C. F. Y. (2016). Using SysML Modeling to Accurately Represent Automotive Safety Requirements. *In 2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT)* (pp. 21–26). Puebla, México: IEEE. http://doi.org/10.1109/CONISOFT.2016.12

27. Hernández-Reveles, J., Sobrevilla-Dominguez, G., Velasco-Elizondo, P., & Soriano-Grande, S. (2016). Adding agile architecture practices to a Cyber-Physial System development. *In Software Process Improvement International Conference (CIMPS)*, pp. 1–6.

28. IEEE. (2014). *Guide to the software engineering body of knowledge version 3.0*. IEEE Computer Society. http://doi.org/10.1234/12345678

29. Jacobson, I. (1992). Object-oriented software engineering : a use case driven approach. New York, Wokingham, Eng. ; Reading, Mass. *ACM Press*, Addison-Wesley Pub.

30. Juárez-Ramírez, R., Cortés Verdín, K., Toscano de la Torre, B. A., Okataba, H., Fernández-y-Fernández, C. A., Flores Ríos, B. L., & Angulo Molina, I. (2013). Estado Actual de la Práctica de la Ingeniería de Software en México. *In Congreso Internacional de Investigación e Innovación en Ingeniería de Software* (Conisoft 2013). Xalapa.

31. Mellor, S. J., & Balcer, M. (2002). Executable UML: A Foundation for Model-Driven Architecture. Addison Wesley.

32. Miranda, J. M., Mata, M. A. M., Cruz, G. R., Luna, H. E. R., Diaz, I. E. T., & Ramirez, J. M. G. (2014). Extracción del conocimiento tácito para

la mejora de procesos software: una experiencia en un organismo gubernamental. *Sistemas y Tecnologías de Información CISTI*, 1, 423–429.

33. Mirna, M., Jezreel, M., Brenda, D., & Claudia, V. (2014). Software process improvement from a human perspective. *In New Perspectives in Information Systems and Technologie*s, Volume 1, pp. 287–298. Springer.

34. Muñoz-Arteaga, J., Esparza, M. Á. O., Mendoza, J. E. G., & Reich, J. C. (2017). An Architectural Model to Design Graphical User Interfaces of Mobile Applications for Learning Problems in Basic Mathematics. In HCI for Children with Disabilities, pp. 31–51. Springer.

35. Muñoz, M., Mejia, J., Calvo-Manzano, J. A., FELIU, T. S. A. N., Corona, B., & Miramontes, J. (2017). Diagnostic Assessment Tools for Assessing the Implementation and/or Use of Agile Methodologies in SMEs: An Analysis of Covered Aspects. *Software Quality Professional*, 19(2).

36. Muñoz, M., Mejia, J., Hurtado, G. P. G., Gómez-Álvarez, M. C., & Durón, B. (2016). Method to establish strategies for implementing process improvement according to the organization's context. *In European Conference on Software Process Improvement*, pp. 312–324.

37. Oktaba, H. (2003). Ingeniería de Software: trayecto personal desde la programación a la producción de software. In Avances en Ciencias de la Computación*, IV Congreso Internacional de Ciencias de la Computación, ENC*, Vol. 3.

38. Oktaba, H. (2008). *Software Process Improvement for Small and Medium Enterprises: Techniques and Case Studies*: *Techniques and Case Studies*. IGI Global.

39. Oktaba, H., Piattini, M., Pino, F. J., & Orozco, M. J. (2009). *Competisoft: mejora de procesos software para pequeñas y medianas empresas y proyectos*. Alfaomega.

40. OMG. (2015). OMG Unified Modeling Language Specification. Version 2.5. Object Management Group.

41. Page-Jones, M., & Constantine, L. L. (2000). *Fundamentals of Object-Oriented Design in UML.* Addison Wesley.

42. Popoca, S. M., & Jiménez, J. T. (1999). Investigación de Métodos de Ingeniería de Software Para un Centro de Investigación.

43. Quintanar Morales, J. A., & Fernández-y-Fernández, C. A. (2015). Modeling software using temporary reductions to transform unstructured flow diagrams into a formal notation. *In Congreso Internacional de Investigación e Innovación en Ingeniería de Software (CONISOFT 2015)*, pp. 151–159. San Luis potosi, SLP, México: Universidad Autónoma de Baja California.

44. Reyes, H. C., Muñoz-Arteaga, J., & González-Calleros, J. M. (2017). Model-Driven Development of Interactive Environments for Occupational Therapy. *In HCI for Children with Disabilities*, pp. 91–113. Springer.

45. Sandoval, J. J. M. (2016). *Método para aligerar procesos de software mediante la optimización en la selección de prácticas de Ingeniería de software*.

46. Schwaber, K., & others. (1995). Scrum development process. *In OOPSLA Business Object Design and Implementation Workshop*, Vol. 27, pp. 10–19. Austin, TX. Retrieved from http://home.hib.no/ai/data/master/mod251/2009/articles/scrum.pdf

47. Somerville, I. (2011). *Software Engineering* (9th ed.). Addison-Wesley.

48. Spivey, J. M. (1989). An Introduction to Z and Formal Specifications. *Software Engineering Journal IEEE*, 4(1), pp. 40–50.

49. Wing, J. M. (1990). A Specifier's Introduction to Formal Methods. *IEEE Computer*, 23(9), pp. 8–24.

Capítulo 8.
Investigación en el área de Métricas de Software

Francisco Valdés Souto, *Universidad Nacional Autónoma de México.*

8.1 Introducción

En 1974 la conferencia que impartió el profesor Donald Knuth de la universidad de Stanford con motivo de la recepción del premio Turing comenzó de la siguiente manera (S. Sánchez, M. Á. Sicilia, 2012):

> "Si la programación de computadoras quiere llegar a ser una parte importante del desarrollo e investigación en las ciencias de la computación, deberá transitar desde la programación como arte a la programación como ciencia disciplinada…"

Después de tratar diferentes aspectos de los términos de ciencia y arte, la conferencia concluyó con lo siguiente:

> "La programación es un arte, por que aplica conocimiento acumulado, requiere habilidades e ingenio, y especialmente por que produce objetos bellos…"

La Ingeniería de Software es hoy en día una actividad de trabajo en grupo y casi nunca una actividad que se desarrolla de manera individual, la gente que trabaja con software se enfrenta a un proyecto

de desarrollo de software, trabaja sobre un marco de restricciones económicas, de plazos, costos, de personal y calidad.

El Instituto de Ingeniería Eléctrica y Electrónica (*Institute of Electrical and Electronics Engineers: IEEE*) define la ingeniería de la siguiente manera:

> "La ingeniería es la aplicación de un enfoque sistemático, disciplinado y cuantificable de estructuras, máquinas, productos, sistemas o procesos."

La Ingeniería de Software como ciencia es la aplicación del método científico a la teorización y creación de conocimiento sobre la propia Ingeniería de Software. Está dedicada al estudio de sus actividades, y centrada en generar teorías, modelos explicativos o enunciados descriptivos sobre la práctica de la ingeniería. La IEEE define la Ingeniería de Software como:

> "La Ingeniería de Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software."

En esta definición se mencionan tres calificativos que pueden aplicarse a la ingeniería, que son sistematicidad, disciplina y cuantificación, que pueden ser definidos de la siguiente manera (Macmillan Dictionary, 2018)

- Decimos que algo es sistemático cuando sigue completamente un método.
- Decimos que algo es disciplinado cuando está bien organizado y sigue reglas o estándares.
- Decimos que algo es cuantificable si es capaz de ser medido o descrito como una cantidad.

Una perspectiva más clara de visualizar estos elementos es a través de una analogía con un banco de tres (3) patas, el cual cada pata representa un calificativo.

Hoy en día, y sin lugar a duda, podemos decir que el calificativo menos desarrollado del software es el cuantitativo, y a esto se bebe que se considere aún inmadura la ingeniería de software, como lo mencionan algunos autores, que indican que la inmadurez es una peculiaridad de la ingeniería de software respecto de otras ingenierías clásicas, así como, otras disciplinas científicas. (Abran, 1998; Naji Habra, Alain Abran, Miguel Lopez, 2008)

De manera sencilla y utilizando la analogía descrita previamente podemos decir que:

- Un banco de dos (2) patas no se puede sostener y mucho menos sostener algo encima.
- Si una pata del banco no está desarrollada al mismo nivel que las otras, para que se pueda distribuir el peso, no va a poder sostener grandes cosas encima.

Por lo tanto, las acciones y decisiones de esta ingeniería deberían de tener una procedencia de la aplicación de métodos y técnicas para organizar los recursos de acuerdo con planes y objetivos definidos, tratando así al desarrollo de software como ciencia disciplinada, sometida al estudio científico y basada en técnicas y métodos aceptados por las personas que se dedican al desarrollo de software, dónde un aspecto cuantitativo es fundamental.

La mayoría de las personas involucradas en el desarrollo de software ya sea de un área interna o una fábrica de software reconocen que medir en el software es fundamental, la frase "Lo que no se mide no se puede mejorar" es dicha con mucha más frecuencia que lo que realmente se aplica.

Sin embargo, ¿qué es lo que se puede medir del software?, cada organización define elementos a medir del software que realiza en función de sus prácticas y experiencias, sin embargo, esto no permite comparaciones con otras organizaciones o industrias, en virtud de que las unidades de medición son distintas.

Básicamente, del software se pueden medir elementos técnicos (líneas de código (LOC), casos de uso, módulos, clases, etc.) y elementos funcionales, la diferencia radica en que los primeros, solo le sirven a las personas técnicas, es decir, si un integrante de un equipo de desarrollo menciona que programó 20,000 LOC, o 55 clases, a los usuarios no les importa, ni siquiera lo entienden, además los elementos técnicos no son estándares, es decir una LOC de Cobol, no

necesariamente es equivalente a una LOC en *Power Builder* o *Java*, menos una clase, adicional a esto no son portables, es decir una LOC o clase en algún lenguaje no es lo mismo en otro lenguaje, por último, estos elementos se conocen de manera precisa ya que se termina el desarrollo del software, por lo tanto, no funcionan de manera adecuada para temas de estimación o gestión de proyectos.

Por otro lado, existen los elementos funcionales, la funcionalidad le es útil a los desarrolladores, pero también a los usuarios, se conoce desde etapas tempranas, sirve de comunicación entre usuarios y técnicos, es portable, es decir, una funcionalidad desarrollada en Java puede ser desarrollada también en .NET o Power Builder, la funcionalidad es la misma, quizá el esfuerzo o costo sea diferente, pero la funcionalidad puede ser exactamente igual. Adicional a esto, la funcionalidad en el software si puede ser medida con un estándar.

En las siguientes secciones describiremos la historia de los esfuerzos para desarrollar este elemento cuantitativo en México, desde dos puntos de vista que están vinculados, aunque no siempre explícitamente, el primero es desde el punto de vista históricos de adopción en la industria y el segundo del punto de vista de investigación en la academia respecto de los elementos de medición de software adoptados en el país.

Cabe señalar que la parte de la industria se realizó con base en entrevistas comentarios de algunas personas que participaron en ella

durante el proceso, la parte de investigación solo se consideró lo que se tiene identificado del área, por lo que es posible que no se tenga todo el universo de investigación relacionada, las referencias de la investigación realizada se agregan por año, insertadas en la historia de la adopción en la industria.

Se agradece la colaboración de Cecilia Pérez Colín para la construcción de la parte de adopción de IFPUG en México.

8.2 Mediciones de elementos técnicos: TSP/PSP

Los modelos CMMI® (*Capability Maturity Model® Integration*) conforman colecciones de buenas prácticas que ayudan a las organizaciones a mejorar sus procesos, fueron desarrollados por equipos con miembros procedentes de la industria, del gobierno y del *Software Engineering Institute* (SEI), en particular, para el desarrollo de software se utiliza el modelo denominado CMMI para Desarrollo (CMMI-DEV), que proporciona un conjunto completo e integrado de guías para desarrollar productos y servicios.

En 1998, Watts Humphrey desarrolló estándares o modelos de procesos, buscando mejorar la calidad del software desarrollado en las empresas, estos modelos son: *Personal Software Process* (PSP) y *Team Software Process* (TSP).

Mientras CMMI-DEV se enfoca en plantear un modelo que sirva de referencia para avanzar en la madurez relativa al desarrollo de

software, el TSP es un conjunto de procesos que se pretende sea utilizada equipos de desarrollo.

En este sentido, CMMI es un modelo de referencia que considera aspectos e infraestructura organizacional, TSP es una implementación de procesos que se enfoca en desarrollar una disciplina de procesos, desarrollando al mismo tiempo una cultura de calidad a nivel de las personas dl equipo, que se trata de lograr a través de PSP.

8.3 Mediciones de elementos funcionales

En los años 70's Allan J. Albrecht's desarrolló una forma de medición funcional del software independiente de la plataforma de desarrollo y que consideraba el punto de vista del usuario, actualmente conocida como *Function Points* (método IFPUG)((IFPUG), 1999) , ésta técnica de medición considera conceptos vigentes para aquel tiempo cómo archivos lógicos, además tiene un alcance solo para aplicaciones de tipo MIS (*Management Information System*), en la actualidad existe una gran variedad de dominios de aplicaciones (no solo las MIS) como lo son en tiempo real, ERP, DWH, etc.

Más tarde a finales de los años 80's surgió un método llamado Mark II, este método estaba basado en transacciones lógicas y en el modelado de relaciones de entidades, populares en ese tiempo. Estos dos, y otros métodos como: *3D Function Points*, *Feature Points*, FISMA, NESMA, etc., surgieron con base en IFPUG, todos los métodos desarrollados hasta el año 2000, son considerados de la

primera generación de modelos de Medición de Tamaño Funcional (*FSM: Functional Size Measurement*).

La característica es que estos métodos se generaron a partir de estudios estadísticos realizados en alguna empresa (*Function Points* en IBM), el análisis expresa en sus resultados la manera de operar de la empresa para los tipos de proyectos analizados. Cuando se trata de utilizar estos métodos en alguna otra empresa o con otro tipo de proyectos, los resultados pueden no ser adecuados ya que la empresa y/o los proyectos operan de manera distinta.

Los criterios definidos como contexto de referencia inicial en el estudio realizado por Albrecht fueron (Abran, 2010):

- La organización incluye a 450 personas que desarrollan aplicaciones.
- El desarrollo está bajo contrato con clientes de IBM.
- Los desarrolladores y los clientes están dispersos por los Estados Unidos.
- En cualquier momento entre 150 y 170 contratos están abiertos.
- Un contrato medio implica dos o tres personas.
- Cada contrato se realiza en el contexto de un marco de desarrollo específico.
- La mayoría de los contratos se limitan a ciertas fases de la metodología.

- Con base en su experiencia, la fase de diseño representa el 20% de las horas de trabajo, la implementación del 80%.
- Es necesario medir todo el proceso, incluyendo el diseño y los costos incurridos durante el diseño.
- Los proyectos se completaron desde mediados de 1974 hasta principios de 1979.
- El tamaño de los proyectos varía de 500 a 105 000 horas de trabajo.
- Cerca de 1500 contratos para el período, sólo 22 cumplieron los criterios de selección.

Así mismo los criterios que se utilizaron para la aceptación de proyectos que dieron origen al estudio fueron (Abran, 2010) :

- El proyecto debe haber pasado por todas las fases de la metodología (desde la definición de los requerimientos hasta la implementación) y debe haber sido entregado al cliente.
- Todo el proyecto debe haber sido gestionado adecuadamente con definiciones consistentes de las tareas y los procesos de gestión.
- Todas las horas de trabajo de IBM y del cliente deben ser conocidas y deben haber sido cuidadosamente medidas.
- Los factores funcionales deben ser conocidos.

La segunda generación de métodos de FSM está representada por COSMIC (*Common Software Measurement International Consortium*), este estándar se basa en sólidos principios de ingeniería

de software y no está basado en estudios estadísticos, se generó considerando como base el estándar ISO/IEC 14143 y la experiencia de los métodos de la primera generación, lo que implica que resuelve gran parte de los problemas presentados en dichos métodos como:

- El manejo de conceptos actualmente no vigentes (la primera generación de estándares de medición se inició en los 70's).
- El alcance de aplicación de los métodos (el alcance del enfoque IFPUG son solamente sistemas para gestión de información.
- Una escala de medición mucho más práctica y homogénea.
- Operaciones matemáticamente válidas.

La figura 8.1 muestra la evolución de los Métodos de Medición de Tamaño Funcional (FSMM, por sus siglas en inglés) de primera generación y COSMIC como el primer FSMM de segunda generación.

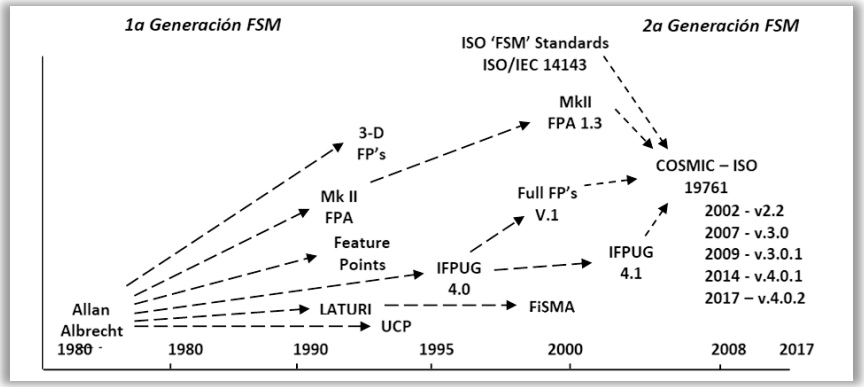

Figura 8.1. Evolución de los Métodos de Medición de tamaño funcional, adaptada de (Abran, 2010)

8.4 Adopción de Mecanismos de Medición en México

En el año de 1998 en InterSoftware, una pequeña empresa de software fue de las primeras en las que se inició a trabajar con la metodología de Function Points (IFPUG), como parte de su iniciativa de mejora de la calidad del software y alcanzar su certificación ISO-9001.

En esa época se estaban llevando a cabo en el Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas de la Universidad Nacional Autónoma de México (IIMAS-UNAM) seminarios organizados por el Círculo de Calidad del Software, donde se reunían interesados de la industria y la academia en la mejora de la calidad del software. Este Círculo de Calidad fue organizado entre otras personas por la Dra. Hanna Oktaba de la Facultad de Ciencias (FC).

Fue en IBM donde surgió esta metodología de Puntos de Función (IFPUG) en el año de 1979, sin embargo, no fue hasta alrededor de 1998 cuando ya se implementa como obligatorio en todos los proyectos de software de IBM en muchos países. El uso de esta métrica fue imprescindible como objetivo, medida comparativa que ayudaba en la evaluación, planificación, administración y control de producción de software.

En 1998, IBM de México crea un grupo de analistas de Puntos de Función. Este grupo fue el encargado de dar el servicio de Análisis de Puntos de Función a nivel mundial en sus proyectos, tanto internos como externos, en los que IBM estuviera involucrado. Curiosamente en

IBM de México no era obligatorio su uso, sin embargo, IBM de México ya contaba con un gran grupo de profesionales que implementa el uso de métricas de software no dependientes de Puntos de Función.

Aproximadamente en 1999, la empresa *Tecnosys*, estaba buscando implementar las buenas prácticas en ese momento para el desarrollo del software y poder así alcanzar en nivel 3 de CMM, Tecnosys fue comprada por IBM de México en ese mismo año, donde se inicia, para México, la formación de un grupo de profesionales dedicados a la Mejora en los Procesos de Software y, por lo tanto, la recolección e implementación de métricas de software para ese fin.

La investigación realizada en el año de 1999, fue una tesis de la maestría en Ciencia e Ingeniería de la Computación del IIMAS-UNAM, relativa la medición de software utilizando IFPUG, tema relativamente nuevo en México (Colín, 1999).

En 2001, IBM de México alcanza el nivel 3 de CMMI aplicando de manera representativa el método de Análisis de Puntos de Función.

En los años 2007 y 2008 en México la Secretaría de Economía da marcha a la Iniciativa Nacional TSP/PSP, la cual se trabajó directamente con el SEI y el Dr. Whatts Humphrey, el objetivo de esta iniciativa era crear en México la infraestructura humana que permitiera la introducción y expansión acelerada del uso de TSP, para que la industria de desarrollo de software en México alcanzara un desempeño superior al de su competencia internacional (Beatriz Velázquez Soto,

2008), con lo cual se generaron varios programas de apoyo para el fomento de estas prácticas, principalmente a nivel gobierno como tractor de la industria, ya en el año de 2008 México ocupaba el primer lugar a nivel mundial en personas certificadas en PSP, lo que aparentemente tenía ventajas.

Las prácticas de TSP/PSP son un conjunto adecuado, sobre todo, con una formalidad estadística y de calidad para el manejo de las métricas, sin embargo, dentro de las principales debilidades que se ha encontrado en la práctica se encuentran:

- Su compleja implementación,
- El uso de medidas basadas en elementos técnicos (LOC)

Durante mucho tiempo, casi 10 años el gobierno de México a través de la Secretaría de Economía, con el fondo PROSOFT, fomentó mediante recursos el uso de estas prácticas, ya que se encontró una buena alianza entre el SEI y México, ya que el primero buscaba un socio que le ayudara a cumplir el objetivo de impulsar TSP/PSP, y el segundo demostró que podía ser un buen aliado que permitiría continuar la evolución de las prácticas (Beatriz Velázquez Soto, 2008) (en virtud del tamaño del mercado, la cercanía a Estados Unidos) con la visión de que podría posicionarse como el país con mejor calidad y valor agregado de manera ágil, lo que México buscaba era adelantarse a las capacidades de sus competidores, contar con un método avalado por el SEI que permitirá demostrar objetivamente la calidad de los proyectos desarrollados por las empresas y que la calidad de los desarrollos con

talento mexicano sean mejores que aquellos con niveles de alta madurez de CMMI. Esto permitirá, supuestamente, hacer desarrollos en menor tiempo y mejor calidad, lo que se transforma en una ventaja de costo, convirtiéndose así en el primer jugador (*first mover*) para el mercado Americano de desarrollo de software, obteniendo ventajas sobre quienes le siguieran.

En el año 2006, mediante el Programa Nacional de Normalización, se realizó la traducción del estándar internacional ISO/IEC 19761 y se generó la Norma Mexicana NMX-I-119-NYCE-2006 TECNOLOGIA DE LA INFORMACION-INGENIERIA DE SOFTWARE-METODO DE MEDICION DEL TAMAÑO FUNCIONAL (COSMIC-FFP), en este contexto México fue si no el primero, de los primeros en adoptar el estándar como norma nacional.

En el año 2008, se crea la primera empresa especializada en la consultoría y capacitación en medición (dimensionamiento), estimación y evaluación de proyectos de Software en México (*SPINGERE*), enfocada principalmente al impulso del estándar de medición de tamaño funcional de 2ª generación: método COSMIC.

Fue hasta 2009 que se certificó el primer mexicano en el estándar internacional ISO/IEC 19761 o método COSMIC, al amparo del estudio del doctorado realizado en la *École de Technologie Superieure* (ETS) bajo la tutoría del Dr. Alain Abran, uno de los fundadores del método de medición COSMIC.

En 2010, IBM de México alcanza en nivel de madurez más alto al que una organización de software puede alcanzar, CMMi-5. Parte de sus logros reconocidos fueron: (1) La organización tiene una infraestructura de herramientas y métodos sólidos para soportar la mayoría de las actividades de las áreas de proceso, (2) Las prácticas de ingeniería de software en la empresa son robustas y utilizadas de forma consistente en los proyectos, (3) El uso de TSP (*Team Software Process*) /PSP(*Personal Software Process*), en algunos proyectos, que permite optimizar la calidad de las actividades de desarrollo a través del uso de prácticas avanzadas de diseño y construcción de software.

En todos estos logros, las métricas de software jugaron, sin duda, un papel crucial. La base de datos de métricas de IBM era muy completa y logró demostrar que el medir era una práctica consistente que ayudaba en gran medida a mejorar los procesos de calidad del software.

En relación con la adopción del método COSMOC, fue hasta 2012 en virtud de la obtención de la representación del Consorcio por parte de la empresa SPINGERE en el país que se creó el blog del *Special Interest Group in COSMIC*/México por el COSMIC *International Advisory Council* para México.

Como investigación académica realizada en el año 2011 y 2012, relacionada a medición de software, se tiene registro de las siguientes referencias (Valdés-Souto, 2011; Valdés-Souto & Abran, 2012).

En 2013 se pudo realizar el primer examen de certificación en México con la autorización del Consorcio de COSMIC y a través de la empresa representante del Consorcio COSMIC en México, a partir de aquí se dispararon las certificaciones, pasando de estar en los últimos lugares con solo 1 certificado en 2009, a estar en el tercer (3) lugar al 15 de agosto de 2019 con 173 certificados en COSMIC. Ver Figura 7.2.

Como investigación académica realizada en el año 2013, relacionada a medición de software, se tiene registro de la siguiente referencia (Valdés-Souto & Abran, 2013).

Figura 8.2. Profesionistas Certificadas en COSMIC de México por año al 15 de agosto de 2019

En el año de 2014, en el del ACUERDO que tiene por objeto emitir las políticas y disposiciones para la Estrategia Digital Nacional, en materia de tecnologías de la información y comunicaciones, y en la de seguridad de la información emitido el 8 de Mayo del 2014, se actualizó una versión del Manual Administrativo de Aplicación General en Materia de Tecnologías de la Información y Comunicaciones y de Seguridad de la Información (MAAGTICSI), que es de carácter obligatorio para las dependencias y entidades de la Administración Pública Federal (APF). En esta versión del Manual, se integró por primera vez el Apéndice IV.B Matriz de metodologías y mejores prácticas de TIC, en el cual se establecían las mejores prácticas para los procesos definidos en el Manual, y para el proceso de Administración de Proyectos, aparecía la Norma Mexicana NMX-I-119-NYCE-2006, equivalente al Método COSMIC o al estándar ISO/IEC 19761, lo que ratificaba como obligatorio para la APF.

A partir de este tiempo se tuvo mayor éxito con la difusión del método COSMIC; y se logró permear de buena manera en la industria de software mexicana, inicialmente en la APF, esto se ve en el incremento del número de personas certificadas, sin embargo, también se empezaron a observar algunas mala implementaciones del método de medición, desde el punto de vista de su uso, esto ya había pasado con CMMI, las empresas buscaban certificarse para obtener contratos, pero no lo aplicaban, lo mismo empezó a pasar con COSMIC, las entidades al ser obligatorio, pedían recursos certificados en el “Método de

estimación COSMIC", lo que denotaba su falta de entendimiento, ya que COSMIC es un método de medición no de estimación.

Como investigación académica realizada en el año 2014, relacionada a medición de software, se tiene registro de la siguiente referencia (Valdés-Souto & Abran, 2014).

En este año se realiza la primera licitación dónde se solicita explícitamente que se utilizaría para estimar el método COSMIC, esto lo solicitó el SAT, aunque en realidad no se aplicó, por el contrario, se hizo una mala práctica de implantación.

En el año 2015, la Secretaría de Economía (SE), a través de INFOTEC y de la empresa SELECT, desarrollaron el estudio denominado Mapa y Rutas 2024 (Select, 2015) como un plan estratégico de trabajo mediante el cual se pretende alcanzar la meta de "Contar con 1000 centros certificados en calidad suprema", meta establecida de facto dentro de la Visión 2024 que tiene PROSOFT 3.0 (Secretaría de Economía, 2014) para la Industria Mexicana de Desarrollo de Software (IMDS).

En el estudio Mapa y Rutas 2024 (Select, 2015), se establece una propuesta de ocho rutas estratégicas que pueden llevar al software producido en México desde 2014, a niveles de desarrollo que lo caractericen por una alta calidad mundial en 2024, contribuyendo así a la Visión 2024 de PROSOFT 3.0.

Resulta muy relevante que, por primera vez, en un estudio nacional, se identifica y determina formalmente la importancia de las métricas de software como un eje que contribuye a lograr que el software que se produce sea de una alta calidad mundial.

La Ruta "Determinar métricas básicas, transversales y trascendentes" definida en (Select, 2015), establece que para que las métricas sean significativas se asocian los calificativos siguientes:

BÁSICAS, porque son Estándares probados internacionalmente que permiten la generación de métricas derivadas;

TRANSVERSALES, porque sirven a todos los actores económicos de la IMDS, para sus funciones (desarrollo, autogestión, etc.) y transacciones (compra-venta, licitaciones, etc.); por último,

TRASCENDENTES, porque al ser básicas se pretende que permitan compararse a lo largo del tiempo y a través de diferentes prácticas, tecnologías, etc.

Definiendo las siguientes acciones a seguir:

- Definir métricas básicas, trascendentes y transversales.
- Medir Línea Base de Productividad País, que aplicará en APF e iniciativa privada por tipificación significativa (zonas, tecnologías, metodologías, clústeres, etc.).
- Generar masa crítica de capacitación y certificación de profesionistas en NMX (clústeres, universidades).

- Asociación de Métricas: Impulso y fomento de organización garante de la recolección, resguardo, estudio de las métricas para referencia de los agentes económicos de la industria, generación de aval para empresas de alta productividad, productos con base en métricas básicas.
- Difundir resultados y línea base de productividad país, por tipificación significativa.
- Desarrollar métricas complementarias con base en métricas básicas .
- Dar seguimiento al uso y utilidad a través de estudios que vinculen a universidades y generando más masa crítica (2017–2024, anual).

También en el año 2015 en el mes de septiembre, se realiza el 1er Congreso Nacional de Medición y Estimación de Software (CNMES15), que tuvo como objetivo acercar a los profesionistas, estudiantes, clústeres, empresas desarrolladoras y demandantes de productos de software, con las últimas tendencias en medición y estimación de software que provean soluciones formales y profesionales para administrar y controlar efectivamente la capacidad de desarrollo de software. El evento fue realizado en la Universidad Iberoamericana, asistieron como ponentes principales el Dr. Alain Abran, Chairman de COSMIC (Canadá), Frank Vogelezang Presidente de COSMIC (Holanda) y Mauricio Aguiar representante de IFPUG

(Brasil). (Congreso Nacional de Medición y Estimación de Software, 2015)

En el CNMES15, se presentó la Asociación Mexicana de Métricas de Software (AMMS) (Asociación Mexicana de Métricas de Software, 2015), que tiene como misión: Prestar servicios de promoción, recolección, referencia, y difusión de métricas relacionadas al software en las áreas que resulten de utilidad para los agentes económicos implicados en la Industria Mexicana de Software, dentro y fuera del territorio nacional. , alineadas a esta misión, se definieron cinco líneas de acción sobre las cuales inicia su operación la AMMS. Figura 8.3.

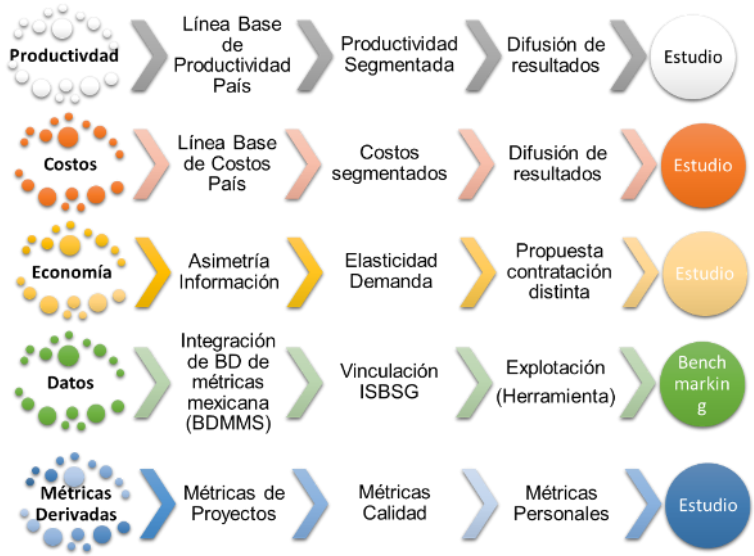

Figura 8.3. Líneas estratégicas de la Asociación Mexicana de Métricas de Software

En el CNMES, como parte de las líneas de acción establecidas de la AMMS, se informó del inició de la recolección de información de proyectos para desarrollar el "Estudio de Línea Base de Productividad y Costo de la Industria Mexicana de Desarrollo de Software", y se invitó a los asistentes a colaborar proporcionando información de proyectos. Con estos logros se da cumplimiento a las acciones definidas en (Select, 2015) como son:

- Generar masa crítica de capacitación y certificación de profesionistas en NMX (clústeres, universidades).
- Asociación de Métricas: Impulso y fomento de organización garante de la recolección, resguardo, estudio de las métricas para referencia de los agentes económicos de la industria, generación de aval para empresas de alta productividad, productos con base en métricas básicas.

Como investigación académica realizada en el año 2015, relacionada a medición de software, se tiene registro de la siguiente referencia (Valdés-Souto & Abran, 2015).

En 2016, IBM a nivel mundial decide dejar de utilizar los Puntos de Función como métrica de tamaño estándar; esto lo hace con la introducción de nuevas metodologías, en particular, las metodologías ágiles para construir software, además de que ya se cuenta con otras métricas más objetivas y adecuadas para asignarle un tamaño al software, como el caso del estándar de COSMIC.

Durante 2016, se continúa con la recolección de la información para el desarrollo del "Estudio de Línea Base de Productividad y Costo de la Industria Mexicana de Desarrollo de Software".

Es en este año que se logra la primera implantación exitosa en una entidad de gobierno del método COSMIC, tanto para estimar proyectos de software como para la gestión de los mismos, logrando una eficiencia de gasto de más de 20% del presupuesto.

Derivado del auge del movimiento ágil, desde aproximadamente hace 2 años se han mantenido pláticas dentro del SEI dónde se ha identificado que las prácticas de TSP/PSP van en decadencia, no son sostenibles de acuerdo con información que ha compartido uno de los mayores impulsores de estas en México, dando como resultado que a partir del 1 de octubre de 2018 el SEI menciona que se deja de dar soporte a estas prácticas.

Como investigación académica realizada en el año 2016, relacionada a medición de software, se tiene registro de la siguiente referencia(Valdés-Souto, 2016).

En mayo de 2017, se desarrolló en 2° Congreso Nacional de Medición y Estimación de Software (CNMES17), también se contó con la participación de especialistas internacionales como el Dr. Alain Abran, Chairman de COSMIC y Frank Vogelezang Presidente de COSMIC, como especialistas nacionales se contó con la participación del Dr. Francisco Valdés Souto.

En este evento (CNMES17) se presentaron los resultados preliminares del "Estudio de Línea Base de Productividad y Costo de la Industria Mexicana de Desarrollo de Software", logrando una aceptación y generando expectativa de los resultados finales.

Es en este año que se logra llegar por primera vez al tercer lugar en número de certificados en el estándar de COSMIC por país, también la AMMS se integra como Gold Partner del International Software Benchmarking Standards Group (ISBSG).

En 2017, como parte del impacto que se ha tenido el CNMES en las 2 ediciones anteriores, se recibe la invitación por parte del ISBSG, para que se realice en México su conferencia anual ITCONFIDENCE, y se propone hacer los 2 eventos en conjunto

Como investigación académica realizada en el año 2017, relacionada a medición de software, se tiene registro de las siguientes referencias (Valdes-Souto, 2017; Valdés-Souto, 2017b; Valdés-Souto, 2017a).

En septiembre del año 2018, se realizó el 3er CNMES y fue la sede de la sexta edición de la conferencia *ITCONFIDENCE* de ISBSG, por primera vez se logra el apoyo de una Universidad, en este caso de la Facultad de Ciencias de la UNAM en conjunto con el Centro Virtual de Computación (CVICOM), y se integra a los festejos de los 60 años de la computación en México.

En el CNMES18 se presenta la primera colección de material bibliográfico de la AMMS con alcance de industria, relacionado a métricas de software, basadas en el estándar internacional denominado como Método COSMIC (ISO/IEC 19761), la colección consta de 4 ejemplares:

- "Estudio de Línea Base de Productividad y Costo de la Industria Mexicana de Desarrollo de Software" (Asociación Mexicana de Métricas de Software, 2018c)
- Reporte de "Análisis de duración de proyectos de Software por Tamaño y Esfuerzo para la Industria Mexicana de Desarrollo de Software", (Asociación Mexicana de Métricas de Software, 2018a)
- Reporte de "Estimación de Software en la Industria Mexicana de Desarrollo de Software con base en Tamaño Funcional", (Asociación Mexicana de Métricas de Software, 2018b)
- Reporte de "Impacto del Tamaño Funcional de Software en la Productividad para la Industria Mexicana del Software". (Asociación Mexicana de Métricas de Software, 2018d)

La biblioteca de la AMMS fue desarrollada en colaboración con investigadores de la Facultad de Ciencias de la UNAM, dónde se desarrolla la línea de investigación en medición y estimación de software y de la Red de Tecnología de Información -Universidad Iberoamericana- Campus Ciudad de México, y representa el primer

esfuerzo en relación con las dos primeras acciones y la quinta que se establecieron en (Select, 2015):

- Definir métricas básicas, transversales y trascendentes.
- Medir Línea Base de Productividad País, que pueda ser considerada como referencia en Administración Pública Federal (APF) e iniciativa privada por tipificación significativa (tecnologías, metodologías, etc.).
- Difundir resultados y línea base de productividad país, por tipificación significativa.

Es importante señalar, que las acciones definidas se han logrado por esfuerzos independientes y particulares, no se ha recibido apoyo del gobierno en ningún sentido, esto atiende a la visión y trabajo de empresas del sector privado que identifican en las métricas de software una oportunidad de incrementar el éxito de los proyectos de software, así como de mejorar la industria.

Como investigación académica realizada en el año 2018, relacionada a medición de software, se tiene registro de las siguientes referencias (García-Floriano Andrés, López-Martín Cuauhtémoc, Yáñez-Márquez Cornelio, 2018; Valdés-Souto, 2018).

Cabe señalar del gran avance que se ha logrado en términos de investigación con la biblioteca de la Asociación Mexicana de Métricas de Software es muy valioso, en virtud de que son los primeros estudios realizados en el país a nivel industria de este tipo.

Hasta el momento, esto es lo que se tiene identificado en la industria y la academia en México, en relación con las métricas de software, probablemente exista más trabajo en relación con esta línea de investigación, aunque principalmente se desarrolla en la UNAM.

8.5 Conclusiones

Como se ha podido observar, la línea de investigación de métricas de software y la adopción de estas en la industria han estado separadas, mientras que por un lado son necesarias de forma práctica (en la industria), en la academia se avanza a pasos más lentos, este fenómeno se observa a nivel mundial, sin embargo, el avance ha sido bueno.

Actualmente en la industria ya se conoce más y se busca aplicar más la medición formal siendo la estimación su principal uso y el método COSMIC uno de los mejores posicionados hoy en día en el país.

El uso de la medición de software con el estándar COSMIC va más allá, dado las características de las mediciones obtenidas con este estándar, que son Básicas, Transversales y Trascendentes.

Por tal motivo, es importante fomentar el estudio y desarrollo de métricas significativas y formales en la academia, pero de manera más expedita, que cumplan los plazos que necesita la industria, es decir, con la finalidad de que se puedan adoptar más métricas formales en lugar de métricas intuitivas generadas al calor de la práctica profesional, que la mayoría de las veces no cumplen con los fundamentos mínimos y generan malos resultados, al mismo tiempo fomenta la inmadurez de la

Ingeniería de Software. Ya lo dijo Luis Pasteur - “Una ciencia es tan madura como sus herramientas de medición”-

La frase “Lo que no se mide no se puede mejorar”, es solo la parte que más se utiliza, sin embargo, la frase completa es de Lord Kelvin: William Thomson, Primer barón de Kelvin y dice “Lo que no se define no se puede medir. Lo que no se mide , no se puede mejorar. Lo que no se mejora, se degrada siempre” por tal motivo, es necesario dar atención a las métricas en el software, porque la situación de la degradación ya la estamos viviendo en una gran cantidad de desarrollos de software en casi todos los ámbitos.

Referencias

1. I. F. P. U. G. (1999). *Function Point Practices Manual*. Mequon, Wisconsin.
2. Abran, A. (1998). Software Metrics Need to Mature into Software Metrology (Recommendations). I*n NIST Workshop on Advancing Measurements and Testing for Information Technology* (IT). Gaithersburg Maryland.
3. Abran, A. (2010). *Software Metrics and Software Metrology*. Hoboken, New Jersey: John Wiley & Sons.
4. Asociación Mexicana de Métricas de Software. (2015). *Asociación Mexicana de Métricas de Software*. Retrieved from www.amms.org.mx

5. Asociación Mexicana de Métricas de Software. (2018a). *Análisis de Duración de Proyecto por Tamaño y Esfuerzo para la Industria Mexicana de Desarrollo de Software*. México, CDMX: Asociación Mexicana de Métricas de Software (AMMS).

6. Asociación Mexicana de Métricas de Software. (2018b). *Estimación Temprana de Software para la Industria Mexicana de Desarrollo de Software.* México, CDMX: Asociación Mexicana de Métricas de Software (AMMS).

7. Asociación Mexicana de Métricas de Software. (2018c). *Estudio Línea Base de Productividad y Costo de la Industria Mexicana de Desarrollo de Software*. (A. S. M. Francisco Valdés-Souto, Jorge Valeriano Assem, Ed.) (1a Edición). México, CDMX: Asociación Mexicana de Métricas de Software (AMMS).

8. Asociación Mexicana de Métricas de Software. (2018d). *Impacto del Tamaño de Software en la Productividad para la Industria Mexicana de desarrollo de Software.* México, CDMX: Asociación Mexicana de Métricas de Software (AMMS).

9. Beatriz Velázquez Soto. (2008, October). *Iniciativa Nacional TSP/PSP*. Software Gurú. https://doi.org/ISSN: 1870-0888

10. Congreso Nacional de Medición y Estimación de Software. (2015). C*ongreso Nacional de Medición y Estimación de Software*. Retrieved August 20, 2010, from www.cnmes.mx

11. García-Floriano Andrés, López-Martín Cuauhtémoc, Yáñez-Márquez Cornelio, A. A. (2018). Support Vector Regression for Predicting

Software Enhancement Effort. *Information and Software Technology*, (97), pp. 99–109. https://doi.org/10.1016/j.infsof.2018.01.003

12. Macmillan Dictionary. (2018). Macmillan English Dictionary.
13. Naji Habra, Alain Abran, Miguel Lopez, A. S. (2008). A framework for the design and verification of software measurement methods. *Journal of Systems and Software*, 81(5). https://doi.org/https://doi.org/10.1016/j.jss.2007.07.038
14. S. Sánchez, M. Á. Sicilia, D. R. (2012). *Ingeniería del Software. Un enfoque desde la guía SWEBOK* (Primera Ed). México: Alfaomega Grupo Editor, S.A. de C.V.
15. Secretaría de Economía. (2014). *Prosoft 3.0. AGENDA SECTORIAL PARA EL DESARROLLO DE TECNOLOGÍAS DE LA INFORMACIÓN EN MÉXICO 2014-2024*. CDMX, Mexico.
16. Select. (2015). *Resultados completos de la Consultoría " Estrategia de calidad para el crecimiento de la industria de software en México ", que contengan la realización de un coloquio a través del cual se define la estrategia de conversión de doble actualización e integ*r. CDMX, Mexico.
17. Valdes-Souto, F. (2017). Earned scope management: A case of study of scope performance using Use Cases as Scope in a real project. *Proceedings - 27th International Workshop on Software Measurement, IWSM 2017 and the 12th International Conference on Software Process and Product Measurement*, Mensura 2017, 53–64. https://doi.org/10.1145/3143434.3143448

18. Valdés-souto, F. (2018). Analyzing the Performance of Two COSMIC Sizing Approximation Techniques Using FUR at the Use Case Level. In M. Salmanoglu & A. Coşkunçay (Eds.), *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement* (IWSM-MENSURA) (pp. 70–108). Beijing, China.

19. Valdés-Souto, F. (2017). *Creating an Estimation Model from Functional Size Approximation Using the EPCU* Approximation Approach for COSMIC (ISO 19761). In C. Mario, Z. Jaramillo, C. Elena, D. Vanegas, & W. P. Charry (Eds.), Software Engineering : Methods, Modeling and Teaching, Volume 4 (Editorial, p. 468). Bpgotá, Colombia.

20. Valdés-Souto, F., & Abran, A. (2013). Using the ISO 19761 COSMIC Measurement Standard to Reduce "Information Asymmetry" in Software Development Contracts and Enable Greater Competitiveness. *Technology and Investment*, 04(04), 261–268. https://doi.org/10.4236/ti.2013.44031

21. Valdés-Souto, F., & Abran, A. (2014). COSMIC Approximate Sizing Using a Fuzzy Logic Approach: A Quantitative Case Study with Industry Data. In F. Vogelezang & M. Daneva (Eds.), 2014 *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement* (pp. 282–292). Rotterdam (Netherlands): Conference Piblishing Services (CPS). https://doi.org/10.1109/IWSM.Mensura.2014.44

22. Valdés-Souto, F., & Abran, A. (2015). Improving the COSMIC Approximate Sizing Using the Fuzzy Logic EPCU Model. In A. Kobylinski, B. Czarnacka-Chrobot, & J. Swierczek (Eds.), *Joint Conference of the 25rd International Workshop on Software Measurement & 10th International Conference on Software Process and Product Measurement - IWSM-MENSURA 2015* (pp. 192–208). Cracow (Poland): Springer International Publishing. https://doi.org/10.1007/978-3-540-71649-5

Capítulo 9.
Experimentación en Ingeniería de Software

Omar Salvador Gómez Gómez, *Escuela Superior Politécnica de Chimborazo,*
Raúl Antonio Aguilar Vera, *Universidad Autónoma de Yucatán,*
Juan Pablo Ucán Pech, *Universidad Autónoma de Yucatán.*

9.1 Introducción

En este capítulo introducimos al lector un panorama general del paradigma experimental aplicado a la Ingeniería de Software (IS), así como presentamos algunas investigaciones hechas en México que han utilizado este enfoque de investigación.

Si bien los esfuerzos en la adopción de enfoques ingenieriles en el desarrollo y mantenimiento de productos software datan desde la década de los años 60s, no fue sino hasta en la década de los 80s que la comunidad académica comenzó a adoptar y emplear enfoques de investigación para estudiar de manera más rigurosa los diferentes aspectos y problemáticas involucradas en el desarrollo de software (Glass et al., 2002).

Podemos decir que la investigación en ingeniería de software tiene como fin generar conocimiento nuevo en el ámbito de esta disciplina (y comprender mejor el ya existente) así como codificarlo de

tal manera que éste pueda ser incorporado en la práctica diaria del desarrollo de software. Esto con el fin de mejorar los plazos de entrega, la ejecución del presupuesto asignado, así como mejorar la calidad del producto software a desarrollar o mantener.

A día de hoy la comunidad académica dedicada a la investigación en ingeniería de software reconoce diferentes estrategias que un investigador puede utilizar en esta disciplina. Por ejemplo, Stol y Fitzgerald (2018) han identificado diferentes estrategias de investigación como son: estudios de campo, estudios muéstrales, simulaciones, experimentos, entre otras estrategias. A grandes rasgos las investigaciones hechas en la ingeniería de software suelen utilizar ya sea un enfoque deductivo (métodos formales, simulaciones), inductivo (encuestas, estudios de caso, experimentos) o una mezcla de ambos.

En el contexto nacional, la investigación en IS data de inicios de la década de los 2000 (Aguileta y Gómez, 2016). En este mismo ámbito nacional y tomando como referencia la versión más reciente del cuerpo de conocimientos de la Ingeniería de Software (en Inglés, *Software Engineering Body of Knowledge* o SWEBOK v3) (Bourque y Fairley, 2014), Aguileta y Gómez (2016) observan que las áreas de conocimiento KA02 (diseño de software) y KA08 (procesos) concentran el mayor número de investigaciones publicadas. Aunque en el ámbito nacional los investigadores han comenzado a utilizar diferentes estrategias de investigación, en el presente capítulo nos

centramos en presentar el paradigma experimental, por lo que quedan excluidos otros enfoques de investigación en esta disciplina.

El resto de apartados de este capítulo se encuentra organizado de la siguiente manera: en el segundo apartado se presenta de manera general el paradigma de experimentación, así como un proceso de experimentación aplicado a la ingeniería de software. En el tercer apartado se ejemplifica el paradigma experimental con un experimento realizado afín a la IS. En el cuarto apartado se presentan algunas publicaciones realizadas en el ámbito nacional que hacen uso de este paradigma. En el quinto apartado se discuten algunas consideraciones finales.

9.2 El Paradigma Experimental

Si bien el paradigma experimental tiene sus orígenes en la física o la química, en la actualidad diversas disciplinas emplean la experimentación como método de obtención de nuevo conocimiento. Este conocimiento resultante puede ser entonces aplicado de manera confiable por profesionales (por ejemplo, ingenieros químicos o físicos, etc.) para resolver problemas que ocurren en la práctica de su disciplina.

La experimentación aplicada a la IS cuenta ya con algunas décadas. Los primeros acercamientos de su aplicación se remontan a los años 60s y 70s, donde algunos investigadores evaluaban tecnologías que apoyaban la construcción de software; esto lo realizaban a través de aproximaciones científicas basadas en la observación (Weinberg y

Gressett, 1963; Grant y Sackman, 1967; Hanson y Ratsno, 1977). No obstante, no fue hasta en los 80s cuando se enfatizó la necesidad de aplicar la experimentación a la IS (Basili et al., 1986).

De manera general, el objetivo de la experimentación consiste en identificar las causas por las que se producen determinados resultados. Al aplicarla a la IS, la experimentación nos ayuda a identificar y comprender distintos aspectos así como conexiones involucradas en el desarrollo y mantenimiento de productos software. Esta identificación de aspectos y conexiones permite validar o refutar con hechos las creencias y prácticas que basamos en el desarrollo y mantenimiento de productos software.

Podemos decir que la meta de la experimentación en IS es poder prescribir soluciones que estén soportadas por un cuerpo de conocimientos validado por evidencia científica.

La unidad principal del enfoque experimental es el experimento. En un experimento se modelan las principales características de una realidad (en nuestro caso, el desarrollo de software) bajo condiciones que podemos controlar, permitiendo así estudiar y comprender mucho mejor esa realidad. Así pues, los experimentos son útiles para ayudarnos a confirmar o refutar teorías, creencias (juicios convencionales), o explorar relaciones causales.

La estructura básica de un experimento se conforma por dos tipos de variables conocidas también como factores (o variables

independientes) y variables de respuesta (o variables dependientes). Los factores son aquellas variables que podemos manipular o controlar en un experimento, mientras que las variables de respuesta son variables que analizamos para observar el efecto que producen los cambios en los factores de interés.

Por ejemplo, en un experimento donde se analiza la duración y esfuerzo que conlleva programar en pareja con respecto a programar de manera individual, el "tipo de programación" actúa como un factor que en este caso se conforma de dos niveles o tratamientos (programación en pareja y programación individual). Por otra parte, la duración y esfuerzo actúan como variables de respuesta. A través de estas variables podemos observar la presencia de un efecto ocasionado por estos dos tipos de programación.

Una vez presentado un breve panorama sobre lo que es la experimentación en IS, es momento de explicar cómo llevarla a la práctica. Así como en nuestra disciplina contamos con un proceso para desarrollar y mantener productos software, en la experimentación en IS se cuenta con un proceso para hacer experimentos (Gómez et al., 2013). Este proceso consta básicamente de las fases que se muestran en la Figura 9.1, y que a continuación se describen.

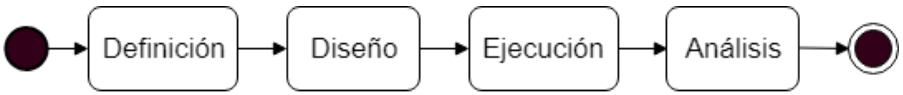

Figura 9.1. Proceso de experimentación genérico (Gómez et al., 2013).

9.2.1 Definición

Esta fase consiste en especificar de manera general los aspectos principales del experimento por realizar así como definir las hipótesis de trabajo. Para comenzar con esta fase es necesario tener ya alguna idea, creencia o teoría la cual se desee contrastar a través del experimento a realizar.

9.2.2 Diseño

Una vez que definimos el experimento y las hipótesis de trabajo, la siguiente fase de este proceso consiste en diseñar cómo llevaremos a cabo el experimento. Básicamente en esta fase se especifica cómo se asignan los tratamientos (organización de los factores) a los participantes, el tipo de participante a emplear así como la preparación de instrumentos o materiales para ejecutar el experimento. Respecto a cómo asignar los tratamientos a los participantes, en la literatura existen diferentes tipos de diseños experimentales que satisfacen propósitos particulares, revisar en Juristo y Moreno (2001) para profundizar en el tema.

9.2.3 Ejecución

En esta fase se lleva a cabo el experimento, el sitio donde se realizará el experimento así como los materiales y equipos a utilizar deben estar configurados y listos para su uso durante el experimento. Durante esta fase se les entrega a los participantes una serie de instrucciones con los materiales en los que trabajarán durante las sesiones del experimento.

9.2.4 Análisis

En la ejecución del experimento se generan diferentes mediciones las cuales se analizan con diferentes técnicas estadísticas. En esta fase se analizan e interpretan las mediciones generadas en el experimento. Usualmente en el diseño de experimentos se emplea el análisis de la varianza (en inglés ANOVA). Básicamente el ANOVA se basa en examinar la variabilidad de las mediciones recabadas y la variabilidad de otros elementos como pueden ser otras variables (o factores) así como el error experimental. El ANOVA proporciona una prueba estadística acerca de sí las medias o promedios de varios grupos son o no equivalentes.

9.3 Ejemplo de aplicación del paradigma experimental

De manera paulatina el uso de experimentos en IS ha ido en aumento. Como profesionales o académicos podemos llevar a cabo experimentos o leer acerca de experimentos realizados con el fin de tomar decisiones informadas basadas en evidencia científica y no en juicios o creencias. Por ejemplo, a través de los motores de búsqueda (como Google o

Google Scholar) es posible encontrar diferentes experimentos realizados en IS.

En el caso de las empresas de desarrollo de software, el uso de experimentos puede ser de utilidad para validar, por ejemplo, prácticas que quieran institucionalizar o tecnologías que deseen evaluar antes de ser implantadas. Imaginemos que en el área de procesos de cierta empresa han escuchado hablar sobre los beneficios de la programación en pareja y desean institucionalizar esta práctica. Sería sumamente arriesgado implantar esta práctica sin antes tener alguna evidencia de sus beneficios. En esta situación los responsables del área de procesos pueden tomar una muestra compuesta por varios programadores pertenecientes a distintos equipos de desarrollo y llevar a cabo un experimento. Imaginemos que los resultados del experimento indican una reducción del tiempo del 28% al emplear parejas, pero se observa un aumento en el esfuerzo (2 programadores) del 30%.

El personal de procesos tendría evidencia de que esta práctica les ahorrará un 28% de tiempo, pero esto implica un aumento en el esfuerzo del 30%. En base a estos números, el área de procesos o la alta gerencia cuenta con evidencia objetiva que puede usar de manera confiable para tomar una decisión sobre adoptar o no esta práctica en la empresa.

Tomando como referencia el proceso de experimentación descrito anteriormente, a continuación, se ejemplifican las fases de este proceso con un experimento realizado con alumnos de la carrera en IS

de la Universidad Autónoma de Yucatán (UADY). En este experimento se examina la duración y esfuerzo que conlleva la programación en pareja (este experimento se describe con mayor detalle en Gómez et al., 2013).

9.3.1 Antecedentes del experimento a realizar

Una de las doce prácticas principales de la programación extrema creada a finales de los 90s por Beck (1999) consiste en programar en pareja. Básicamente en esta práctica dos programadores trabajan en conjunto en una misma tarea usando una computadora. Uno de los programadores quien toma el rol de controlador escribe el código mientras que el otro programador identificado como observador revisa de manera activa el trabajo realizado por el controlador, esto con el fin de detectar posibles defectos. El observador puede hacer anotaciones o definir estrategias para llevar a cabo la tarea a realizar, también puede buscar por ejemplo referencias para ayudar a resolver alguna cuestión que se presente durante la realización de la tarea.

Se dice que el uso de esta práctica ayuda a producir programas más pequeños, ayuda a implementar mejores diseños y que los programas contienen menos defectos que aquellos escritos de manera individual. Se dice también que la duración en completar una tarea trabajando en pareja es menor que la duración de realizarla de forma individual.

9.3.2 Definición

Pensemos en realizar un experimento para contrastar nuestros hallazgos con respecto a los beneficios que hemos escuchado sobre esta práctica. En esta primera fase del proceso de experimentación podemos hacer uso del método GQM (*Goal-Question-Metric*) (Basili et al., 1994) que nos ayuda a establecer el objeto de estudio de nuestro experimento, el propósito, el enfoque de calidad, la perspectiva y el contexto. Usando el método GQM, la definición de este experimento podría describirse como:

> "Estudiar la programación en pareja y la programación individual (objeto de estudio) con el propósito de evaluar posibles diferencias entre estas dos formas de programación con respecto al esfuerzo (enfoque de calidad) que conlleva cada tipo de programación. El estudio se realiza desde el punto de vista (perspectiva) del investigador en el contexto de un laboratorio de cómputo donde alumnos de licenciatura codificarán en parejas o de manera individual dos especificaciones de programas".

La perspectiva y el contexto pueden variar dependiendo el entorno en donde se realice el experimento. Por ejemplo, imaginemos que en el área de procesos de una empresa de desarrollo de software han previsto institucionalizar esta práctica en los equipos de desarrollo, no obstante, antes de implementarla se realizará un experimento para conocer de manera más confiable si esta práctica puede ser o no beneficiosa para la empresa.

En este ejemplo el experimento se realiza desde la perspectiva del personal del área de procesos de la empresa y el contexto se conforma por una muestra de programadores pertenecientes a distintos equipos de desarrollo de alguna empresa.

Dos hipótesis que podemos definir para contrastar con las mediciones a recolectar de este experimento pueden ser:

- H0a: El tiempo requerido para codificar un programa utilizando programación en pareja es equivalente al tiempo requerido para codificarlo de manera individual o, la programación en pareja = programación individual con respecto a la duración.
- H0b: El esfuerzo requerido para codificar un programa utilizando programación en pareja es equivalente al esfuerzo requerido para codificarlo de manera individual o, la programación en pareja = programación individual con respecto al esfuerzo.

Para este ejemplo la duración es el tiempo medido en minutos, y el esfuerzo es el tiempo en minutos multiplicado dos veces para el caso de las parejas (dado que se asignan dos recursos para realizar una actividad o tarea).

El tipo de hipótesis antes planteada se le conoce como hipótesis nula y se especifica de tal modo que no tenga valor o efecto. Esta hipótesis se asume como verdadera hasta que las mediciones recolectadas tras ejecutar el experimento indiquen lo contrario, es decir la evidencia obtenida respalde alguna hipótesis alternativa, en este caso,

la existencia de una diferencia significativa entre la programación en pareja y la programación individual (con respecto a la duración y el esfuerzo).

9.3.3 Diseño

Dependiendo del número de aspectos que se quieran estudiar en un experimento, existen diferentes tipos de diseños para confeccionar nuestro experimento. Por ejemplo, podríamos asumir que en ciertas condiciones los programas a codificar y las herramientas para codificar los programas no inciden en los resultados (tiempo y esfuerzo) del experimento. En una primera versión de este experimento se podrían aislar o bloquear los programas a codificar así como las herramientas usadas y centrarnos sólo en la comparación del tipo de programación con respecto a la duración y el esfuerzo.

A este tipo de diseño experimental se le conoce como cuadrado Latino. La característica de este diseño es que usa dos tipos de variable conocidas como bloques las cuales ayudan a incrementar la precisión de las mediciones recabadas en el experimento. Es decir, ya que en un experimento existen diversos factores que pueden influir en los resultados, este diseño experimental ayuda a aislar fuentes de variación no deseada, por lo que se tienen resultados con una mayor precisión.

Tabla 9.1. Ejemplo de diseño cuadrado latino usando en el experimento

Programa / Herramienta	Con IDE	Con editor de texto
Programa 1	Individual	Pareja
Programa 2	Pareja	Individual

En la Tabla 9.1 se puede observar la organización de los tratamientos (en este caso el tipo de programación) que conforman este diseño experimental; en dicha tabla se observan también las dos variables de bloque que son: el programa a codificar y la herramienta para codificarlo, mientras que los tratamientos de interés son los referentes a la programación en pareja y la programación individual.

9.3.4 Ejecución

En esta fase se lleva a cabo el experimento en un entorno controlado, cabe señalar que es durante esta fase donde se generan las mediciones que posteriormente serán analizadas. El experimento descrito en este ejemplo se basa en un experimento real realizado en el año 2012 donde participaron 21 estudiantes universitarios inscritos en uno de los cursos de la carrera de IS de la Universidad Autónoma de Yucatán, UADY (Gómez et al., 2013). La mayoría de ellos, cursando su tercer año de carrera. Los participantes se asignaron de manera aleatoria a dos grupos que representan los tratamientos a examinar, el grupo de participantes que trabajó en parejas (siete parejas) y aquellos que trabajaron

individualmente (siete solos). A continuación se describe con mayor detalle la ejecución de éste.

El experimento reportado en Gómez et al., (2013) se dividió en dos sesiones, en cada sesión los participantes codificaron un programa diferente (variable de bloque). En la primera sesión los participantes que trabajaron individualmente utilizaron el entorno de desarrollo *NetBeans* para codificar el primer programa mientras que los participantes que trabajaron en pareja usaron como herramienta un editor de texto simple. En la segunda sesión se intercambió la herramienta de soporte (variable de bloque), los participantes que antes trabajaron individualmente con *NetBeans* ahora trabajaron con un editor de texto, mientras que los participantes que trabajaron en pareja codificaron el segundo programa con *NetBeans*. Para las sesiones del experimento se eligieron programas pequeños que los participantes pudieran codificar en cada sesión.

Para cada sesión del experimento se planificó una duración de 90 minutos. Las dos sesiones se llevaron a cabo en uno de los laboratorios del centro de cómputo de la Facultad de Matemáticas de la UADY. Al comienzo de cada sesión se les explicó a los participantes la especificación del programa a codificar. Los participantes registraron el tiempo de inicio y final que les llevó codificar el programa. La diferencia de estos tiempos (calculada en minutos) se usó como métrica para obtener la duración y el esfuerzo. Respecto a la métrica de esfuerzo, que mide la cantidad de trabajo gastado para realizar una tarea

(en este caso persona-minuto), se duplicó el tiempo usado por las parejas.

9.3.5 Análisis

Una vez ejecutado el experimento, se procede a pre-procesar y analizar las mediciones recolectadas. En nuestro caso, los grupos de mediciones recolectadas pertenecen a participantes codificando en parejas y participantes codificando de forma individual. Un ejemplo del tipo de análisis que suele realizarse en esta fase del proceso se muestra en la Tabla 9.2; en este caso se empleó el Análisis de Varianza (ANOVA).

Tabla 9.2 Ejemplo de valores p obtenidos luego de aplicar una prueba estadística

Factor	Métrica	Prueba F	valor p
Tipo de programación	Duración	2.98	0.09
Tipo de programación	Esfuerzo	2.89	0.10

El ANOVA utiliza un tipo de prueba estadística conocida como prueba F de Fisher que ayuda al investigador a determinar si hay o no una diferencia significativa entre los tratamientos examinados en el experimento. El valor p es la probabilidad de obtener un resultado (en este caso valor F) al menos tan extremo como el observado. Usualmente el investigador define un valor crítico llamado alfa (α) para contrastarlo con el valor p observado. Si un valor p es menor o igual al valor alfa entonces la hipótesis nula es rechazada. Por ejemplo, de acuerdo a los valores p observados en la Tabla 9.2, si establecemos un valor alfa de

0.1 se rechaza en ambos casos la hipótesis nula y se acepta la alternativa. ¿Qué significa esto? significa que, si este experimento se realizara 100 veces bajo las mismas condiciones, el 90% de las veces obtendremos una diferencia significativa entre estos dos tipos de programación con respecto a la duración y al esfuerzo.

En la Tabla 9.3 se muestra otro ejemplo de información estadística que suele utilizarse en esta fase del proceso de experimentación. En este caso, la Tabla 8.3 presenta un ejemplo de las diferencias obtenidas (parejas y solos) en minutos con respecto a la duración y al esfuerzo. Como se observa en esta tabla, se obtuvo una diferencia de 36 minutos a favor de la programación en pareja (28% decremento). Con un nivel de confianza del 95% esta diferencia puede variar de 6 a 66 minutos (decremento de 4% a 51%). Respecto al esfuerzo, se observa una diferencia de 56 minutos a favor de la programación individual (30% decremento). Este valor puede variar de 8 a 104 minutos (decremento de 4% a 55%).

Tabla 9.3 Ejemplo de diferencias entre parejas y solos con respecto a la duración y el esfuerzo

Métrica	**Diferencia entre parejas y solos (mins)**	**valor p**	**Límite de control inferior (95%)**	**Límite de control superior (95%)**
Duración	36.5	0.09	6.15	66.98
Esfuerzo	56.5	0.1	8.79	104.2

Por último, en la Figura 9.2 se ilustra un ejemplo del tipo de gráficos que suelen utilizarse también en esta fase del proceso de experimentación; en este caso, se muestra un histograma con los promedios de los tratamientos de interés (programación en pareja y programación individual) con respecto a la duración (medida en minutos).

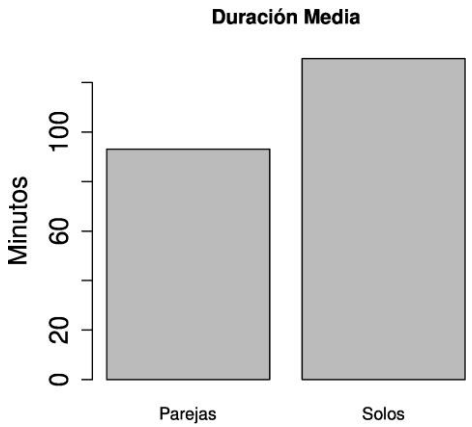

Figura 9.2 Duración media de los tratamientos de interés

Una vez concluidas las fases del proceso de experimentación, se suelen reportar los resultados del experimento ya sea como un informe técnico o como otras formas de publicación.

Los resultados de un primer experimento pueden servir de referente para seguir indagando sobre otros aspectos involucrados en éste. Por ejemplo, pueden configurarse otros tipos de diseños experimentales para estudiar nuevos factores de interés. El experimento

puede realizarse en otros sitios en otros entornos con el fin de corroborar y fortalecer los resultados previamente observados. Es decir, el experimento evoluciona como una familia de experimentos. El ejemplo del experimento sobre programación en pareja antes descrito es un ejemplo de un experimento que ha evolucionado, este experimento se ha realizado en otros contextos variando diferentes aspectos de interés como: el entorno de programación usado, los programas a codificar, el género, la duración, esfuerzo, calidad y productividad (Gómez et al., 2013; Gómez et al., 2017; Gómez et al., 2017a; Gómez et al., 2018).

9.4. La Experimentación en Ingeniería de Software en México

Si bien en el ámbito internacional la aplicación del paradigma experimental en la ingeniería de software tiene ya algunas décadas de presencia, en México comienzan a aparecer publicaciones en donde se reporta su aplicación. En este apartado describimos de manera general algunas publicaciones donde se ha aplicado este paradigma en el ámbito nacional.

Para identificar este tipo de publicaciones, seleccionamos la base de datos de literatura técnica y científica *Scopus* que alberga la mayor cantidad de publicaciones técnicas y científicas indexadas. Se confeccionó la cadena de búsqueda que incluyera los términos: "*controlled experiment*" y "*software engineering*" así como se filtró

aquellas publicaciones donde al menos uno de sus autores tuviese como filiación alguna institución nacional. La cadena de búsqueda confeccionada se muestra en la Tabla 9.4.

Tabla 9.4 Cadena de búsqueda especificada

Cadena de búsqueda
TITLE-ABS-KEY("controlled experiment" AND "software engineering") AND AFFILCOUNTRY(Mexico)

Tras ejecutar la cadena de búsqueda (búsqueda efectuada en septiembre de 2018), esta base de datos arrojó como resultado siete documentos (Lopez-Martin, 2008; Lopez-Martin et al., 2012; Mora et al., 2016; Raza et al. 2017; Gómez et al., 2017; Ucán et al., 2018; Gómez y Aguileta, 2018). De estos documentos 3 corresponden a publicaciones en revistas arbitradas (Mora et al., 2016; Gómez et al., 2017; Gómez y Aguileta, 2018) y 4 a publicaciones en memorias de congreso (Lopez-Martin, 2008; Lopez-Martin et al., 2012; Raza et al., 2017; Ucán et al., 2018). Con respecto a las instituciones de educación superior en las que al menos pertenece alguno de los autores de estas publicaciones, se destacan las mostradas en la Tabla 9.5.

Tabla 9.5. Autores pertenecientes a IES nacionales que han reportado experimentos en IS.

Autores	Institución	Publicaciones (#)
Gómez et al. (2017), Ucán et al. (2018), Gómez y Aguileta (2018),	Universidad Autónoma de Yucatán	3
Lopez-Martin (2008), Lopez-Martin et al. (2012)	Universidad de Guadalajara	2
Mora et al. (2016)	Universidad Autónoma de Aguascalientes	1
Gómez et al. (2017)	Universidad Veracruzana	1
Raza et al. (2017)	Tecnológico de Monterrey	1

Tomando como referencia la versión más reciente del SWEBOK (v3), podemos clasificar los experimentos identificados de9acuerdo a las áreas de conocimiento mostradas en la Tabla 8.6.

Tabla 9.6. Publicaciones clasificadas por áreas de conocimiento del SWEBOK.

Área de conocimiento del SWEBOK	Publicación
Diseño de software (KA 02)	Lopez-Martin (2008)
Pruebas de software (KA04)	Lopez-Martin (2008), Ucán et al. (2018)
Gestión en la Ingeniería de Software (KA07)	Raza et al. (2017), Lopez-Martin et al. (2012)
Proceso de la Ingeniería de Software (KA08)	Mora et al. (2016)
Práctica profesional de la Ingeniería de Software (KA11)	Gómez et al. (2017), Gómez y Aguileta (2018)

Si bien en el ámbito nacional el número de publicaciones que reportan experimentos es inferior al reportado en otros países, se observa que algunos investigadores han comenzado a adoptar el enfoque experimental en sus investigaciones. Con respecto a las áreas de conocimiento del SWEBOK, se observa que las áreas de conocimiento referentes a las pruebas de software (KA04), gestión (KA07) y práctica profesional (KA11), son las áreas en las que se han reportado el mayor número de experimentos.

9.5. Conclusiones

En este capítulo hemos presentado un panorama general sobre el paradigma experimental aplicado a la Ingeniería de Software. En el ámbito nacional también hemos presentado algunas investigaciones en donde se ha aplicado este paradigma. Para todos aquellos interesados en comenzar a realizar experimentos en IS, existen publicaciones que pueden ser de utilidad para este fin. Como punto de partida, dos publicaciones relevantes en este ámbito son el libro publicado por Juristo y Moreno (2001) en donde se detallan los diferentes tipos de experimentos que un investigador puede realizar en IS. Otra publicación de interés es la descrita en (Jedlitschka et al., 2008), en donde los autores presentan una guía sobre cómo reportar experimentos en ingeniería de software.

En el contexto nacional, existen instituciones de educación superior donde han comenzado a ofertar cursos sobre experimentación

en el ámbito de la Ingeniería de Software, como es el caso de uno de los cursos de la carrera de Ingeniería de Software de la Universidad Autónoma de Yucatán.

Referencias

1. Aguileta, A.A., Gómez, O.S. (2016) Software engineering research in Mexico: A systematic mapping study. *International Journal of Software Engineering and its Applications*, 10 (12), pp. 75-92.
2. Basili, V., Caldiera, G., Rombach, H. (1994) *Goal question metric paradigm.* In Encyclopedia of Software Engineering, New York, NY, USA: Wiley, pp. 528-532.
3. Basili, V., Selby, R. y Hutchens, D. (1986) Experimentation in software engineering. *IEEE Trans. Softw. Eng*., 12 (7), pp. 733–743.
4. Beck, K. (1999) Embracing change with extreme programming. *Computer*, 32 (10), pp. 70-77.
5. Bourque, P., Fairley, R. (2014) *Guide to the Software Engineering Body of Knowledge (Swebok(R)): Version 3.0* (3rd ed.). IEEE Computer Society Press, Los Alamitos, CA, USA.
6. Glass, R., Vessey, I., Ramesh, V. (2002) Research in software engineering: an analysis of the literature. *Information and Software Technology*, 44 (8), pp. 491 – 506.

7. Grant, E. E. y Sackman, H. (1967) An exploratory investigation of programmer performance under on-line and off-line conditions. *IEEE Transactions on Human Factors in Electronics*, 8 (1), pp. 33–48.

8. Gómez, O.S., Aguileta, A. A. (2018) Influence on the use of an IDE tool support in the pair programming: A controlled experiment. *IEEE Latin America Transactions*, 16 (3), pp. 948-956.

9. Gómez, O.S., Aguileta, A.A., Aguilar, R.A., Ucán, J.P., Rosero, R.H., Cortes-Verdin, K. (2017) An Empirical Study on the Impact of an IDE Tool Support in the Pair and Solo Programming. IEEE Access, 5, art. no. 7919168, pp. 9175-9187.

10. Gómez, O. S., Batún, J. L., Aguilar, R. A. (2013) Pair versus Solo Programming – An Experience Report from a Course on Design of Experiments in Software Engineering. *International Journal of Computer Science Issues*, 10 (1), pp. 734–742.

11. Gómez, O. S., Solari, M., Pardo, C. J. (2017) A controlled experiment on productivity of pair programming gender combinations: Preliminary results. *In: CIbSE 2017 - XX Ibero-American Conference on Software Engineering*, pp. 197-210.

12. Gómez, O. S., Ucán, J. P., Gómez, G. E. (2013) Aplicación del proceso de experimentación a la Ingeniería de Software. *Abstraction & Application*, 8, pp. 26–37.

13. Hanson, D. R. Ratsno. (1977) An experiment in software adaptability. Software, Practice and Experience, 7 (5), pp. 625–630.

14. Jedlitschka A., Ciolkowski M., Pfahl D. (2008) *Reporting Experiments in Software Engineering*. In: Shull F., Singer J., Sjøberg D.I.K. (eds). Guide to Advanced Empirical Software Engineering. Springer.

15. Juristo, N. y Moreno, A. M. (2001) *Basics of Software Engineering Experimentation.* Kluwer Academic Publishers.

16. Lopez-Martin, C., Chavoya, A., Meda-Campaña, M.E. (2012) Software size estimation of individual projects. *In: Proceedings of the IASTED International Conference on Software Engineering and Applications*, *SEA 2012*, pp. 402-408.

17. Lopez-Martin, C. (2008) Quality improvement applying design and code reviews for developing small programs. *In: IMETI 2008 - International Multi-Conference on Engineering and Technological Innovation*, pp. 153-158.

18. Mora, M., O'Connor, R.V., Rainsinghani, M., Gelman, O. (2016) Impacts of electronic process guides by types of user: An experimental study. *International Journal of Information Management*, 36 (1), pp. 73-88.

19. Raza, M., Faria, J.P., Salazar, R. (2017) Helping software engineering students analyzing their performance data: Tool support in an educational environment. *In: Proceedings of IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*, pp. 241-243.

20. Stol, K., Fitzgerald, B. (2018) The ABC of Software Engineering Research. *ACM Trans. Softw. Eng. Methodol*. 27 (3), 51 pages.

21. Ucán Pech, J.P., Aguilar Vera, R.A., Gómez, O.S. (2018) Software testing education through a collaborative virtual approach. *Advances in Intelligent Systems and Computing*, pp. 231-240, 2018.

22. Weinberg, G. M. y Gressett, G. L. (1963) An experiment in automatic verification of programs. *Communications of ACM*, vol. 6, pp. 610–613.

Ingeniería de Software en México: Educación, Industria e Investigación,
se terminó el 30 de septiembre de 2019.
A partir del 1 de diciembre de 2019 está disponible en formato PDF
en la página de la Academia Mexicana de Computación:
http://amexcomp.mx/