5. Ingeniería de Software

La Ingeniería de Software es una disciplina que tiene como propósito desarrollar programas de cómputo que brinden soluciones automatizadas a necesidades expresadas por personas con intereses en común; para tal fin dispone de un conjunto de técnicas, herramientas, métodos y procesos que se utilizan para la creación y mantenimiento de programas de cómputo.

5.1. Origen de la disciplina

Desde la aparición de las primeras computadoras en la década de los cincuenta, la necesidad de crear programas para solucionar problemáticas inicialmente vinculadas a la realización de numerosas operaciones de cálculo y, más tarde, al procesamiento de grandes volúmenes de datos, ha estado presente como parte de la actividad profesional de nuestra sociedad; dichos programas de computadora, conocidos como Programas de Software[1] consisten de un conjunto de instrucciones organizadas —de acuerdo a una sintaxis y con un estilo definido— que controlan y coordinan las operaciones que ejecuta el hardware de una computadora; para este efecto se utilizan los datos que se reciben a través de algún medio o que previamente se han almacenado en algún dispositivo, o incluso una combinación de ambos, con el objetivo de brindar solución a problemáticas previamente definidas u ofrecer algún tipo de servicio.

[1] Abran, A. et al. (2004). **SWEBOK: Guide to the Software Engineering Body of Knowledge Version,** IEEE Computer Society, Los Alamitos, California.

A finales de la década de los sesenta, ante el crecimiento en las capacidades de los equipos de cómputo, las exigencias de solución a problemas cada vez menos triviales, así como la proliferación de lenguajes de programación, se reconoció la necesidad de contar con un nuevo enfoque —sistemático, disciplinado y cuantificable— que permitiera a la programación evolucionar como disciplina profesional y dar respuesta a los nuevos desafíos del proceso de desarrollo de dichos programas de computadora; es decir, se comenzó a reconocer la necesidad de aplicar el enfoque ingenieril al proceso de desarrollo de software.

El término "Ingeniería de Software" se puede atribuir al menos a tres pioneros de la computación. Meyer[2] lo atribuye a Anthony Oettinger —Presidente de la ACM entre 1966 y 1968— quien al reflexionar sobre la actividad de los profesionistas en el emergente campo de la industria computacional, reconoce la existencia de nuevas profesiones de naturaleza ingenieril, como la Ingeniería del Hardware y la Ingeniería del Software. Por su parte, Margaret Hamilton —galardonada por el presidente de los Estados Unidos en noviembre de 2016— al ser entrevistada sobre el proyecto Apolo, comenta que durante el tiempo en que se desarrolló el software de navegación "*on-board*" comenzó a utilizar el término "Ingeniería de Software" para distinguirlo del hardware y de otras ingenierías; sin embargo, en aquel entonces se consideró una broma por lo divertido que les resultaba.[3] Aunque probablemente no es el primero en haber citado dicho término, Friedrich Ludwic Bauer es seguramente el más reconocido por haberlo utilizado al referirse a la necesidad urgente de aplicar la ingeniería al proceso de fabricación del software. Su intervención en la reunión del Comité de Ciencia de la Organización del Tratado del Atlántico Norte (OTAN) a finales de 1967 generó un impacto mediático que dio lugar a un par de conferencias celebradas en Alemania en 1968[4] y en

[2] Meyer, B. (2013). **The origin of software engineering**. https://bertrandmeyer.com/2013/04/04/the-origin-of-software-engineering/

[3] Hamilton, M. **The engineer who took the Apollo to the moon**. https://medium.com/@verne/margaret-hamilton-the-engineer-who-took-the-apollo-to-the-moon-7d550c73d3fa

[4] Naur, P., Randell, B. (1969). S**oftware Engineering: Report of a conference sponsored by the NATO Science Committee**. Garmisch, Germany, 7-11 Oct. 1968, NATO.

Italia en1969,[5] en las cuales se analizaron aspectos relevantes como el diseño, producción, especificación y calidad del software, entre otros temas.

5.2. Importancia del Software y necesidad de una disciplina ingenieril

En la actualidad resulta prácticamente imposible llevar a cabo cualquier tarea cotidiana sin el uso o la aplicación de software. La fabricación industrial, las infraestructuras y organizaciones nacionales, los servicios públicos y los sistemas financieros se controlan mediante sistemas de software. La industria del entretenimiento como la música, los videojuegos, la televisión y el cine utilizan también software de forma intensiva.

De acuerdo con Sommerville la Ingeniería de Software resulta especialmente importante por dos razones:[6] i) cada vez es más frecuente que las personas y las sociedades se apoyen en sistemas de software más avanzados y complejos por lo que es necesario producir software confiable, de manera económica y rápida; y ii) generalmente resulta más barato el uso de métodos y técnicas específicas de ingeniería de software a largo plazo que sólo diseñar y codificar software, como si fuese un proyecto de desarrollo personal.

Adicionalmente, diversos estudios han argumentado la necesidad de la disciplina ingenieril para los proyectos de Tecnologías de la Información (TI); por ejemplo *Standish Group*[7] señala que menos del 40% de los proyectos de TI resultan exitosos; McKinsey[8] señala que en promedio los grandes proyectos en el área de TI sobrepasan en un 45% los presupuestos estimados, utilizan

[5] 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.

[6] Sommerville, I. (2010). **Software Engineering** 9ª. Ed. Pearson.

[7] Standish Group International (2012). **CHAOS Manifesto 2012 Coll. Research Reports**. The Standish Group International, Inc.

[8] McKinsey&Company (2012). **Study on large scale IT projects**, McKinsey & Company and the University of Oxford.

un 7% de tiempo adicional para su ejecución y sus entregables poseen un valor por debajo del acordado hasta en un 56%; KPMG[9] reporta que el 70% de las organizaciones han sufrido al menos un fallo en sus proyectos de TI.

Por otro lado las tendencias en áreas tecnológicas como *Big Data* o *IoT* (*Internet of Things*) demandarán la generación de una mayor cantidad de software; por tanto, si no se fomenta el enfoque ingenieril en el software no se logrará la madurez requerida para explotar la información existente.

Por lo anterior la Ingeniería del Software resulta imprescindible para el desarrollo y mantenimiento de software de calidad y aunque aquellas problemáticas que dieron origen a la "crisis del software" —costos imprecisos, plazos de entregas incumplidos y requisitos no satisfechos— aún no se han resuelto totalmente, la disciplina apenas cumplirá medio siglo de existencia, por lo que es de esperarse que en el siglo XXI se logren importantes avances en la consolidación de su cuerpo de conocimientos.

5.3. El cuerpo de conocimientos de la Ingeniería de Software

El cuerpo de conocimientos que sustenta a la Ingeniería de Software comenzó a construirse desde finales de la década de los sesenta precisamente en las reuniones de la OTAN en las que comenzaron a analizarse los síntomas de la crisis del software.

El desarrollo de software, analizado desde la óptica de la dualidad Proceso-Producto, consiste de cinco fases: i) requisitos, ii) diseño, iii) codificación, iv) pruebas y v) mantenimiento. Estas fases integran las actividades y tareas organizadas para un proyecto específico, en función del ciclo de vida, como se ilustra en la Figura 5.1. Este ciclo establece el orden de las fases y procesos involucrados, así como los criterios de transición de una fase a otra. Las

[9] KPMG (2010). **Survey of 100 businesses across a broad cross section of industries**, New Zeland, 2010.

actividades se seleccionan de acuerdo a la problemática, la claridad de los requisitos por parte del cliente/usuario, la madurez del equipo de desarrollo y la novedad de la tecnología, entre otros aspectos. Dichas tareas y actividades transforman la descripción de las necesidades expresadas por el cliente o usuario en un conjunto de artefactos a saber: i) Especificación de Requisitos Software, ii) Diseño y iii) Código, que derivan finalmente en un software con ciertas funcionalidades y restricciones de operación acordadas.

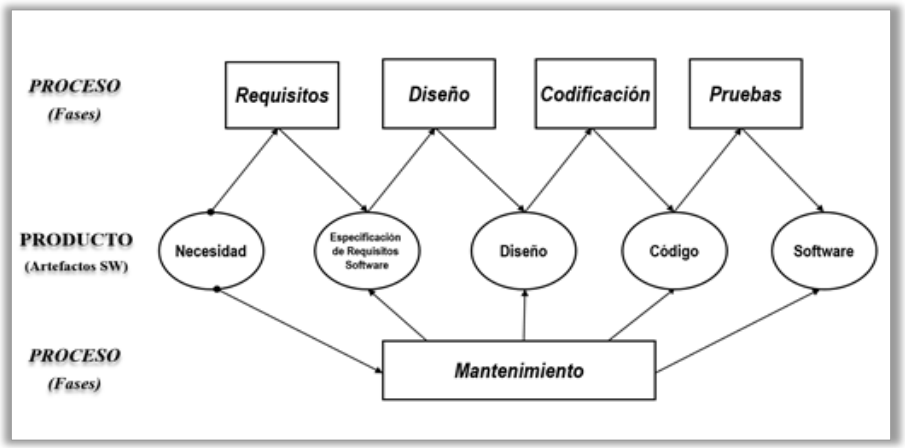

Figura 5.1. Dualidad Proceso-Producto en la Ingeniería de Software

El rápido desarrollo de la disciplina originó un acervo de conocimientos desde sus primeras tres décadas; la Sociedad de Computación del Instituto de Ingeniería Eléctrica y Electrónica (*IEEE-CS*, por sus siglas en inglés) y la Asociación para la Maquinaria Computacional (*ACM,* por sus siglas en inglés) en su proyecto denominado SWEBOK compilaron el conocimiento generalmente aceptado en la disciplina hasta 2004[10] y establecieron dos subconjuntos de áreas de conocimiento, cinco vinculadas con los procesos de desarrollo y otras cinco con las áreas de gestión. Junto con estas áreas se reconocieron otras disciplinas directamente relacionadas con la actividad de la disciplina;

[10] Ver nota 1.

una década después, dicho cuerpo se revisó y actualizó; el SWEBOK V3.0[11] representa el próximo paso en la evolución de la disciplina. En las siguientes subsecciones se describen dichas áreas así como el trabajo de investigación que se ha desarrollado en cada una de éstas por la comunidad en México.

5.3.1. Requisitos de Software

Se entiende por requisitos de software al conjunto de funcionalidades y restricciones expresadas respecto a un producto de software; los requisitos contribuyen a la solución de un problema, a la mejora de un servicio o a la automatización de un proceso específico; esta área de conocimiento integra un conjunto de procesos vinculados con la obtención de los requerimientos, el análisis, la negociación, la especificación y la validación.

La primera etapa del proceso, denominada *obtención*, tiene como propósito identificar aquellas necesidades reales y restricciones de operación —criterios de calidad— expresadas por un conjunto de personas o entidades que son afectadas por el sistema de información, servicio o proceso en cuestión que se desea automatizar; dichas personas o entidades denominadas *stakeholders* pueden ser usuarios, clientes, proveedores, decisores o reguladores externos; en fin, todo aquel que tiene algún interés y que, por ende, puede describir, desde una óptica distinta el funcionamiento actual, así como las oportunidades de mejora del sistema de información, servicio o proceso. Para la obtención de información, el Analista de Requisitos (AR)[12] —papel del especialista en el proceso de requisitos— diseña una estrategia en la que combina un conjunto de técnicas de educción como la entrevista, la encuesta estructurada/no estructurada, la observación de tareas habituales, los escenarios de uso, modelado del negocio, el uso de prototipos, por mencionar algunas de las más conocidas.

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

[12] Young, R. (2004). **The Requirements Engineering Handbook**, Artech House Inc., Capítulo 2, 3 y 7.

La etapa de *análisis y negociación* tiene como objetivo asegurar la calidad de los requisitos antes de incorporarlos al documento de especificación. En esta etapa se precisan los límites del sistema software y su interacción con el entorno y se traducen los requisitos del usuario a requisitos de software; es decir, se deben transformar las descripciones —desde la perspectiva del problema— obtenidas de los *stakeholders* respecto a los servicios que esperan del sistema, en descripciones detalladas —en términos de la solución— acerca de las funcionalidades y restricciones de operación de dichos servicios o prestaciones. Para logar lo anterior se realizan tres tareas principales con los requisitos: i) Clasificación, ii) Modelización y iii) Negociación. La tarea de Clasificación consiste en separar las descripciones de aquellos requisitos que expresan la funcionalidad o servicios que el sistema debe brindar —requisitos funcionales— de las descripciones que expresan las restricciones de operación o propiedades que debe cumplir el sistema —requisitos no funcionales. La segunda tarea, la Modelización, tiene como propósito identificar la arquitectura del sistema mediante una representación gráfica, típicamente semiformal, en la que se puedan identificar los subsistemas y requisitos asociados a cada uno de éstos. Finalmente, la tarea de Negociación consiste en identificar y eliminar los conflictos entre requisitos, con la intención de obtener finalmente un conjunto de requisitos factibles y mutuamente satisfactorios.

La etapa de *especificac*ión consiste en documentar los requisitos de software acordados, en un nivel apropiado de detalle; para ello se suele utilizar un modelo de documento redactado en términos comprensibles para los *stakeholders*; para este efecto el IEEE propuso el estándar 830-1998;[13] sin embargo, éste es sólo una recomendación. Un gran reto de la especificación consiste en expresar con claridad lo que los clientes desean y necesitan en términos de requisitos de software, por lo que se recomienda el uso de plantillas que permitan estandarizar el lenguaje utilizado; no obstante, en ocasiones la especificación textual es insuficiente por lo que es conveniente acompañar las

[13] IEEE Std 830-1998. **IEEE recommended practice for software requirements specifications**. *IEEE Computer Society,* Los Alamitos, 1998.

descripciones con modelos gráficos que ilustren algún aspecto que resulte de interés para el sistema; por ejemplo, para ilustrar la interacción del usuario con el sistema y el flujo de datos entre los procesos se suelen utilizar Diagramas de Casos de Uso y Diagramas de Flujo de Datos respectivamente; en ocasiones una aplicación (o parte de ella) se interpreta mejor si se piensa que está en uno o varios estados, para lo cual se suelen utilizar los Diagramas de Transición de Estado.

Finalmente, en la última etapa del proceso de requisitos, la fase de *validación*, se realiza una revisión cuidadosa de la consistencia, completitud y otros aspectos específicos de interés particular, vinculados con la calidad del documento; el objetivo es identificar problemas en el documento de especificación de requisitos de software antes de que sea usado como base para el diseño del sistema.

5.3.2. Diseño de Software

Se entiende por diseño al proceso de definición de la arquitectura, componentes, interfaces y otras características del sistema software, así como del producto de dicho proceso.[14] El diseño de software es una tarea que se lleva a cabo en una etapa temprana de los procesos vinculados con la construcción de una solución de software —diseño, codificación y pruebas— y a diferencia de la fase de requisitos en la que se define qué debe hacer el sistema, en el diseño se decide cómo debe hacerlo; para ello, el ingeniero de software utiliza como guía un conjunto de principios, métodos y técnicas.

Los principios como la abstracción, acoplamiento, cohesión, modularización, entre otros, representan nociones clave que proporcionan la base para muchos enfoques y conceptos de diseño de software diferentes; por su parte, los métodos y técnicas han ido evolucionando a la par con los diferentes paradigmas de desarrollo. A través de los métodos, se generan representaciones

[14] IEEE Std. 610.12-1990 (1990). **IEEE Standard Glossary of Software Engineering Terminology**.

—generalmente gráficas— de diferentes aspectos vinculados con el software, principalmente, aspectos relacionados con la estructura y el comportamiento que debe tener para satisfacer los requisitos acordados.

En una analogía con la ingeniería civil, para la construcción de una casa se debe pensar cómo la construcción podrá satisfacer las necesidades expresadas por sus futuros habitantes. Parte de la tarea de diseño consiste en elaborar los planos en los que se representan la ubicación y distribución de los espacios físicos, como la sala, el comedor y las habitaciones, y de los ductos y tuberías para el suministro de electricidad y agua. En el caso del diseño del software, los métodos y las técnicas permiten modelar los aspectos estructurales del software, como la distribución de los cuartos, y su dinámica de operación o comportamiento, como el servicio eléctrico y de agua, independientemente del paradigma utilizado.

En la sección correspondiente a Métodos y Modelos de la Ingeniería de Software se ofrece una mayor descripción de los métodos para diseño de software, los cuales se han desarrollado a la par del paradigma estructurado y del orientado a objetos. Desde la perspectiva del proceso de software, el diseño comprende tres actividades principales: i) la definición de la arquitectura, ii) la definición de sus componentes y iii) la definición de la interfaz del sistema.

La definición de la arquitectura tiene como propósito descomponer y organizar el software en componentes e interfaces, entendiendo por componente una clase, un módulo e incluso una base de datos. La arquitectura de alto nivel es un modelo que sirve como insumo a la fase de pruebas; se utiliza para diseñar la estrategia de integración de los diferentes componentes del sistema, así como para verificar su funcionamiento integral.

El objetivo de la definición de componentes o diseño detallado es especificar las estructuras de datos, algoritmos, características de la interfaz y mecanismos de comunicación con suficiente detalle para implementarlos. El proceso incluye una serie de tareas que permiten representar al software a

un nivel de abstracción cercano a la codificación. Uno de los componentes usualmente requeridos es el modelo de persistencia de los datos, mejor conocido como el modelo de la Base de Datos; para el diseño de dicho componente, independientemente del paradigma, se suele utilizar el modelo relacional.

La definición de interfaz de usuario tiene como propósito modelar el funcionamiento e interacción del usuario con el software, donde los modelos de usuario, mental y de implementación representan la interacción entre el usuario y el sistema.[15] Para que este proceso sea efectivo, el diseñador debe realizar un análisis de las tareas de los usuarios, del entorno de trabajo y cómo los usuarios interactúan con otras personas o con otro software y, de ser posible, desarrollar un prototipo; es importante realizar una evaluación de la interfaz propuesta, la cual permita retroalimentar al equipo de desarrollo, así como otros aspectos vinculados con la experiencia del usuario. En resumen, la interfaz de usuario comprende dos componentes principales: el lenguaje de presentación —conjunto de objetos como íconos, menús y formularios, entre otros— y el lenguaje de acción —como el lenguaje de comandos y tiempos de respuesta, por mencionar algunos. Ambos componentes se corresponden con la forma y el contenido en la definición de la interfaz de usuario.

5.3.3. Construcción de Software

Cuando se habla del proceso de construcción del software generalmente se piensa en las tareas vinculadas con las tres fases centrales del desarrollo: el diseño, la codificación y las pruebas. La codificación utiliza las salidas del diseño y proporciona el insumo de la fase de pruebas; no obstante, los límites entre el diseño, la codificación y las pruebas no siempre son claros y suelen variar en función del ciclo de vida seleccionado para el desarrollo de un proyecto

[15] Pressman, R. (2010). **Ingeniería de Software un Enfoque Práctico**, McGraw-Hill, Séptima Edición. Ver también el capítulo de Interacción Humano Computadora en este texto.

software. En este apartado nos enfocamos al proceso de codificación y más adelante se abordan la terminología y las actividades de diseño y pruebas.

La codificación, como actividad socio-técnica, involucra actividades dependientes del lenguaje, las cuales requieren del conocimiento de la sintaxis, la semántica y la pragmática del lenguaje de programación. Este es uno de los motivos por los que el aprendizaje de la programación es uno de los principales problemas en las primeras etapas de la formación de los ingenieros de software. Una de las formas de clasificar a los lenguajes de programación es en términos de su expresividad; si el lenguaje utiliza expresiones cercanas a las de una computadora, como el ensamblador, es de más bajo nivel que aquellos que permiten generar expresiones más cercanas al lenguaje humano, como Pascal. Por lo anterior, la elección del lenguaje de programación no es tan sólo la manera en la que se traduce el diseño detallado a código, sino que influye significativamente en la complejidad de la abstracción utilizada durante el proceso de construcción.

Los principios fundamentales que se utilizan en la construcción de software y que guían el proceso de codificación son: i) minimizar la complejidad, ii) anticiparse a los cambios, iii) construir para verificar, iv) reutilizar y v) utilizar estándares.

La complejidad se refiere al grado en que un sistema o componente tiene un diseño o código difícil de entender o de verificar.[16] Este primer principio consiste en generar código que sea fácil de entender, probar y mantener; para ello se han definido un conjunto de buenas prácticas, como técnicas de legibilidad o el uso estándares de documentación, que ayudan al ingeniero de software a simplificar el código. El segundo principio se relaciona con la utilidad del software a lo largo de su vida ya que éste se someterá a un proceso de mantenimiento en varias ocasiones por lo que es necesario anticiparse a los cambios desde el proceso de codificación; en este sentido es recomendable

[16] IEEE Std. 610.12-1990, **IEEE Standard Glossary of Software Engineering Terminology**

identificar y aislar[17] aquellas áreas que son más proclives de ser modificadas, como las reglas del negocio, las dependencias del hardware y las entradas y salidas, para que en lo posible los cambios afecten a módulos y/o rutinas ubicadas en el código de antemano. El tercer principio, construir para verificar, tiene como filosofía el uso sistemático de pruebas de unidad mediante la automatización y el uso de métodos estandarizados para las revisiones del código. La reutilización se refiere al uso de los activos existentes —módulos, bibliotecas, componentes— para la resolución de diferentes problemas, y en términos generales, para mejorar tanto la productividad como la calidad. Finalmente, el uso de estándares tiene como propósito brindar interoperabilidad entre artefactos desarrollados por diferentes equipos de desarrollo y/o por diferentes individuos asignados a un mismo proyecto.

5.3.4. Pruebas de Software

Antes de describir los procesos relacionados con la fase de pruebas es conveniente aclarar los términos que se utilizan en el proceso de verificación. En general los comportamientos que difieren de los que se especifican por el usuario son defectos o errores del software; en particular, cuando el defecto se manifiesta en tiempo de ejecución se dice que ha habido un fallo en el software, el cual se puede deber a una falta cometida por el equipo de desarrollo, ya sea en el código, en el diseño o en algún artefacto previamente elaborado.

Las pruebas al software son procesos dinámicos que tienen como propósito verificar la calidad del código y evaluar su comportamiento; para este efecto se generan casos de prueba específicos y se intenta generar fallos. La actividad inmediata es la depuración, la cual consiste en descubrir las causas del defecto del software —por ejemplo, la falta en el código que al momento de ejecución generó el mal funcionamiento— y modificarlo para generar el comportamiento deseado. Aunque la evaluación tiene como propósito

[17] McConell, S. **Code Complete, A Practical Handbook of Software Construction**, Microsoft Press, 2nd. Edition, 2004.

generar información relacionada con los defectos, éstos no siempre se deben a faltas en el código y las faltas no siempre generan fallos. En términos generales, las pruebas no tienen como propósito obtener un código libre de faltas, sino eliminar faltas que permitan reducir al mínimo el tiempo medio entre fallos.

Para el diseño de las pruebas se conocen dos estrategias generales: las pruebas de caja negra, que se enfocan en los resultados generados al ejecutar el código en función de los datos de entrada, y las pruebas de caja blanca, enfocadas a seguir puntualmente el comportamiento del software durante la ejecución del código (por ejemplo, una estrategia es generar casos de prueba que obliguen a utilizar todos los caminos posibles para el flujo de los datos).

Existen cuatro niveles o tipos de pruebas: i) pruebas unitarias: se ejecutan a nivel de módulos o componentes y verifican el funcionamiento correcto de cada elemento; ii) pruebas de integración: se centran en diseño de alto nivel o diseño de la arquitectura; el objetivo es verificar la interacción entre todos los módulos y diseñar una estrategia para integrar cada uno de los componentes previamente probados de manera individual; iii) pruebas del sistema: verifican el comportamiento del software como una unidad; generalmente se enfocan a la evaluación de los requisitos no funcionales acordados con el usuario —aspectos como rendimiento, fiabilidad, velocidad, etc.— ya que los funcionales se evalúan en los dos niveles de prueba previos; y iv) pruebas de aceptación: se ejecutan en conjunto con el usuario; el documento de especificación de requisitos se utiliza como guía y el objetivo es validar que el software dé respuesta a las necesidades de funcionalidad, así como las restricciones acordadas al inicio del proceso de desarrollo. En resumen, los tres primeros niveles permiten verificar que el software funcione correctamente y el cuarto validar que disponga de la funcionalidad acordada con el usuario.

5.3.5. Mantenimiento de Software

El estándar IEEE 610.12[18] define al Mantenimiento del Software como la modificación de un sistema software, o de un componente, después de que se ha entregado a los usuarios o clientes con el fin de corregir defectos, mejorar su rendimiento u otros atributos, o adaptarlo a un cambio en el entorno.

A diferencia del hardware, el software no se avería con el tiempo, por lo que no requiere reparación o reemplazo de sus componentes; por lo mismo el mantenimiento del software es diferente al del hardware o al realizado a otros artefactos físicos. En términos prácticos, el mantenimiento es un proceso que permite al software brindar la utilidad para la cual fue concebido, a pesar de la volatilidad de los requisitos del usuario y de los cambios de los entornos tecnológicos; dependiendo del tipo de modificación requerida, el mantenimiento puede clasificarse en una de cuatro categorías: i) correctivo, ii) perfectivo, iii) adaptativo y iv) preventivo.[19]

A pesar de la estrategia utilizada para la verificación en la fase de pruebas siempre existe la posibilidad de encontrar defectos una vez que se libera el software; por ende se tiene la necesidad de continuar con la fase de mantenimiento correctivo a lo largo de su ciclo de vida. Los defectos se pueden deber a faltas cometidas en alguna de las fases del proceso de desarrollo —requisitos, diseño, programación, pruebas— y en función de la urgencia, el mantenimiento correctivo se planifica y ejecuta en el tiempo. Otro motivo para hacer modificaciones al software es la volatilidad en las necesidades de los *stakeholders*; el mantenimiento perfectivo atiende solicitudes para ampliar las funcionalidades o mejoras en aspectos vinculados con la calidad del software (por ejemplo, eficiencia). Por su parte, el mantenimiento adaptativo se refiere a las modificaciones provocadas por cambios en el entorno tecnológico y de

[18] IEEE Std. 610.12-1990, **IEEE Standard Glossary of Software Engineering Terminology.**

[19] Piattini, M. *et al.* **Mantenimiento del Software**, Alfaomega, 2001.

los datos o en el de los procesos. Finalmente, el mantenimiento preventivo se refiere a las modificaciones al software que mejoran sus propiedades, pero sin alterar las prestaciones originales.

Independientemente del tipo de modificaciones realizadas, durante el mantenimiento se ejecutan procesos y actividades relacionadas con las fases correspondientes al desarrollo del software; no obstante, suelen usarse métodos específicos como la ingeniería inversa o la reingeniería.

5.3.6. Gestión de la configuración

Los procesos vinculados con el desarrollo de software generan un conjunto numeroso de artefactos que en algunos casos contienen características volátiles; esta área de conocimiento se refiere a las actividades de gestión relacionadas con la identificación, documentación y control de todos los elementos de configuración[20] acordados en un proyecto de desarrollo; en sentido estricto, es una actividad que se realiza a lo largo de todo el ciclo de vida del software, tanto en los procesos de desarrollo como en las actividades de mantenimiento.

La gestión de la configuración es una disciplina de control que tiene por objetivos establecer y mantener la integridad de los elementos de configuración generados a lo largo de la vida del software; evaluar y controlar los cambios sobre dichos elementos y facilitar la visibilidad del producto. Un concepto clave en estas actividades es el de "línea base", que se refiere a un punto de referencia en el procesos de desarrollo del software que queda marcado por la aprobación de uno o varios elementos de configuración mediante una revisión técnica formal.

Para el logro de sus objetivos, la gestión de la configuración establece cuatro actividades principales:[21] i) identificación de la configuración, ii) con-

[20] Un elemento de configuración es un artefacto software generado como parte del proceso de desarrollo; puede tomar la forma de un: i) código, ii) documento, o iii) conjunto de datos.

[21] IEEE Std. 828-1998, **IEEE Standard for Software Configuration Management Plans**.

trol de la configuración, iii) generación de informes de estado, y iv) auditorias de la configuración. La actividad i) consiste en identificar la estructura del software, sus componentes, el tipo de cada uno de éstos, además de hacerlos únicos y accesibles; ii) se refiere al control de versiones y entregas del software y a los cambios que se producen a lo largo de su ciclo de vida; iii) consiste en informar el estado de los componentes del software y de las solicitudes de cambio, y generar estadísticas de su evolución; finalmente, iv) consiste en validar la completitud de un producto software y la consistencia entre sus componentes y asegurar que el software que se entrega es el que el usuario requiere.

5.3.7. Gestión de la Ingeniería de Software

Para la obtención exitosa de un producto de software, los procesos de gestión son tanto o más importantes que los procesos orientados al desarrollo, es por ello que los primeros deben ser considerados incluso desde antes de la obtención del primer producto y hasta la liberación del producto final.

El área relacionada con la gestión de la Ingeniería de Software involucra las actividades de medición, estimación, planificación, seguimiento y control. La estimación tiene como objetivo determinar los recursos humanos, económicos, así como el tiempo requerido para el desarrollo de un proyecto de software.[22] Para esto es fundamental dimensionar apropiadamente el sistema de manera previa. Una de las primeras métricas claras y precisas para este efecto es el número de líneas de código; sin embargo, su utilización es inviable para la estimación ya que esta medida se conoce hasta una fase avanzada de la codificación. Otra manera de dimensionar el software consiste en determinar los puntos de función[23] y se obtiene desde la primera fase del proceso de desarrollo mediante el análisis de varios tipos de funcionalidad en el documento

[22] Con la asistencia de algún modelo matemático como, por ejemplo, COCOMO (*Constructive Cost Model: COCOMO*) propuesto por Barry Boehm a finales de los setenta.

[23] El método para el análisis de los puntos de función fue propuesto inicialmente por Allan Albrecht en 1979.

de requisitos. El dimensionamiento del software es la base para estimar los recursos necesarios para su desarrollo, el conocimiento de las características del cliente, la madurez del equipo de desarrollo, el conocimiento del dominio del problema, así como los riesgos relacionados con la tecnología considerada, entre otros aspectos; la primera decisión previa a la planificación de las actividades consideradas es la selección de un modelo apropiado de ciclo de vida de desarrollo de software y, consecuentemente, de la metodología de desarrollo.

La planificación genera una relación ordenada de las actividades del proyecto, incluyendo plazos, responsables y recursos necesarios; se identifican claramente las dependencias entre las actividades y los tiempos de holgura para su finalización. Por su parte, el seguimiento permite recolectar y acumular datos sobre los recursos utilizados, costes generados e hitos asociados; éstos a su vez permiten generar los informes de estado y, en su caso, tomar decisiones de control.

Finalmente, las actividades de gestión para la Ingeniería de Software se realizan en tres niveles: i) gestión organizativa y de infraestructura, ii) gestión a nivel de proyectos y iii) gestión del programa de medición.

5.3.8. Procesos de la Ingeniería de Software

El área de procesos se ocupa de las actividades realizadas por ingenieros de software para desarrollar, mantener y operar el software; particularmente es el conjunto de actividades y tareas interrelacionadas que transforman los productos de entrada en productos de salida. Esta área se relaciona con las actividades de trabajo y no con la ejecución del proceso; es decir, especifica lo que se debe hacer, preo no lo que realmente se hace.

Los tópicos de interés en esta área se relacionan con: i) la definición del proceso del software, su gestión y la infraestructura requerida; ii) el análisis

de los diferentes modelos de ciclos de vida; iii) los modelos y métodos de evaluación de procesos de software, iv) los modelos de mejora de procesos de software y v) la medición de procesos y productos de software, así como la calidad de los resultados de dicha medición.

5.3.9. Métodos y modelos de la Ingeniería de Software

Los métodos y modelos relacionados con los procesos de software les imprimen un carácter ingenieril ya que promueven que la actividad se realice de manera sistemática, repetible, pero sobre todo con una orientación al éxito. El uso de modelos proporciona un enfoque particular en la resolución de problemas, así como una notación y procedimientos para la construcción y el análisis del modelo; por su parte, los métodos proporcionan un acercamiento a los procesos relacionados con la especificación, diseño, construcción, prueba y verificación del software, así como con los artefactos generados en dichas fases.

En esta área de conocimiento se hace énfasis en cuatro temáticas: i) la práctica del modelado, ii) la tipología de los modelos, iii) las técnicas de análisis para el modelado y iv) los métodos asociados a los procesos del desarrollo de software.

La aplicación del enfoque ingenieril al proceso de desarrollo de software con el paradigma estructurado, orientado principalmente a sistemas de información, originó los primeros métodos, modelos y técnicas para los procesos de análisis y diseño, actualmente conocidos como procesos de requisitos y diseño; éstos generan representaciones tanto del aspecto algorítmico como del estructural de los datos.[24] En la actualidad, el paradigma orientado a objetos ha generado el desarrollo de métodos, modelos y técnicas que permiten generar software para diversos dominios —comercio, transporte, servicios financieros, etc.— en los que los aspectos estructurales y del comportamien-

[24] DeMarco, T. **Structured Analysis and System Specification**, Prentice Hall, 1979.

to asociado al software requieren mayor variedad de representación, como las que ofrece UML.[25]

5.3.10. Calidad del Software

La calidad del software se define como la capacidad del producto para satisfacer las necesidades declaradas e implícitas bajo condiciones especificadas;[26] esta área se enfoca a las prácticas, herramientas y técnicas para definir la calidad del software, así como para evaluar su estado durante el desarrollo, mantenimiento y despliegue.

Debido a que el software es un producto intangible difiere de la mayoría de los productos construidos bajo otras disciplinas ingenieriles; de hecho, el software no se construye sino más bien se desarrolla. Asimismo, no se avería a lo largo de su vida útil. Más bien puede dejar de ser útil, a pesar de los procesos de mantenimiento a los que se someta; por lo tanto, su curva de fallos tiene un comportamiento distinto a los dispositivos físicos. Ante estas características surge la interrogante de si es realmente posible encontrar un conjunto de propiedades en un producto de software que nos den un indicativo de su calidad. La respuesta se intenta dar a través de los modelos de calidad. En éstos la calidad se define de manera jerárquica: en el nivel más alto se encuentran los factores de calidad, los cuales representan las características que desde el punto de vista del usuario debe reunir el software; se les conoce también como atributos de calidad externos. Cada uno de los factores tiene un conjunto de criterios asociados, cuya presencia contribuye al aspecto de calidad que dicho factor representa. En el nivel más bajo se encuentran las métricas, las cuales indican el grado en que el software posee determinado atributo de calidad. Por ejemplo, un factor de calidad asociado al software es

[25] *Unified Modeling Language* (UML, por sus siglas en inglés) es uno los lenguajes de modelado de aplicaciones de software que se ha convertido en un estándar.

[26] ISO/IEC 25010:2011 **Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—Systems and Software Quality Models**, ISO/IEC, 2011.

la corrección, entendiéndola como el grado en que el software satisface las necesidades del usuario; un criterio asociado es la completitud, la cual evalúa que el software implemente todas las funciones requeridas y acordadas; finalmente, una métrica asociada a la completitud es el porcentaje de requisitos cubiertos, lo cual se puede determinar mediante una prueba de validación.

Otra interrogante es cómo es posible medir el grado de calidad de un producto software. Este problema se enfrenta con los procesos de evaluación; en términos generales se pueden identificar dos tipos de procesos de evaluación: verificación y validación.[27] La verificación permite determinar si los artefactos generados al final de una fase del proceso de desarrollo cumplen con los requisitos establecidos durante la fase anterior. La validación, por su parte, tiene como propósito asegurar el cumplimiento de las necesidades del cliente al final del proceso de desarrollo. Las técnicas asociadas a los procesos de verificación y validación se pueden clasificar en dos categorías: estáticas y dinámicas. Las primeras tienen por propósito buscar faltas sobre el sistema en reposo, por lo que se pueden aplicar al final de cada una de las fases del proceso de desarrollo (por ejemplo, revisiones y auditorias); por su parte, las técnicas dinámicas son aplicables al sistema en ejecución, es decir, una vez que el sistema se ha codificado. Las técnicas dinámicas se conocen también como pruebas del software (por ejemplo, unidad e integración) como se ha descrito previamente.

5.4. Aportaciones en investigación

La comunidad de Ingeniería de Software en México se ha orientado principalmente a consolidar la calidad en los programas de formación de recursos humanos,[28] a fortalecer a la industria del software[29] y, en tercer lugar, a la

[27] IEEE Std. 610.12-1990, **IEEE Standard Glossary of Software Engineering Terminology.**

[28] Aguilar, R., Díaz, J. (2015). **La Ingeniería de Software en México: Hacia la consolidación del primer programa de Licenciatura**. *Revista Tecnología Educativa.* 2(2):6–17.

[29] **Resultados completos de la Consultoría "Estrategia de calidad para el crecimiento de la industria de software en México"**, 4to. Entregable, Select, Cd. México, Marzo 2015.

investigación.[30] En esta sección se describen brevemente las principales temáticas que se han abordado en la investigación tomando como referencia las áreas de conocimiento establecidas en el SWEBOK.

En el área de requisitos de software (AC_01) se han desarrollado propuestas de especificación formal de las necesidades del software de manera precisa; destaca el desarrollo de una propuesta para el modelado formal de flujos de tareas,[31] la cual permite verificar, por ejemplo, los flujos detallados en los casos de uso; este modelo ofrece la posibilidad de corregir errores antes de que se llegue a etapas donde sería más costoso enfrentarlos. También se han explorado áreas no tradicionales donde la ingeniería de requisitos debe aplicarse, como en el software para la industria automotriz.[32]

En el área de diseño de software (AC_02) se ha utilizado la ingeniería Web para el diseño de aplicaciones colaborativas en entornos educativos; en particular, se ha explorado el uso de la Metodología UWE para modelar la funcionalidad y las características de dicho tipo de aplicaciones;[33] también en este ámbito se ha trabajado en plataformas de comunicación y educación ambiental soportada por una arquitectura de software para atender requerimientos no funcionales, particularmente, de seguridad y mantenimiento.[34] En el caso de la seguridad se ha elaborado un modelo de calidad que permite determinar las tácticas arquitectónicas y los mecanismos correspondientes para

[30] Juárez-Ramírez, R. et al, **Estado Actual de la Práctica de la Ingeniería de Software en México**. En R. Juárez-Ramírez et al (Eds.), Tendencias de la Práctica de la Ingeniería de Software en México en el ámbito Académico; Tijuana, México, Editorial UABC, pp. 3–14, 2013.

[31] Fernandez, C., Simons, A. (2014). **An implementation of the task algebra, a formal specification for the task model in the Discovery Method**. *J. Appl. Res. Technol.*, 12(5):908–918.

[32] Aguilar, J. Fernández-y-Fernández, C., (2015). **Especificación de Requerimientos para el Desarrollo de Software Automotriz en México**. *Rev. Latinoam. Ing. Softw.*, 3(6):250–258.

[33] Uicab, O., Ucán, J., Aguilar, R. (2016). **Una Herramienta para el Análisis de la Colaboración diseñada con UWE**. *Rev. Latinoam. de Ing. de Soft.*, 4(6):235–242.

[34] Íñiguez, F., Cortes, K., Pérez, J., Contreras, G., Maldonado, A. (2015). **The development of a software architecture for an environmental education platform**. *International Conference on Computing Systems and Telematics (ICCSAT).* Xalapa, Ver., México.

distintos escenarios de seguridad.[35] También se han realizado algunas investigaciones de tipo exploratorio acerca de SOA (*Service Oriented Architecture*) que han llevado a propuestas como la de una arquitectura orientada a servicios para una línea de productos de software.[36]

El área de pruebas de software (AC_03) ha sido poco atendida; se reportan algunos trabajos en los que se ha dado continuidad a la familia de experimentos propuesta inicialmente por Basili[37] en la que se pretende contrastar diferentes tipos de prueba para la detección de faltas en el código; se han realizado réplicas diferenciadas en ambientes académicos que incorporan factores como el uso de un entorno colaborativo virtual inteligente para asistir a los aprendices durante el proceso de identificación de faltas en el código.[38]

El área de construcción de software (AC_04) se ha enfocado a incorporar buenas prácticas así como a la inclusión de diversos patrones arquitectónicos y de diseño que permiten obtener software de calidad, incluyendo la investigación, evaluación y prueba de marcos de trabajo o *frameworks* y estilos arquitectónicos; por ejemplo, el desarrollo de una red social con el propósito de apoyar los estudios de seguimiento de egresados[39] con una aplicación móvil[40] apoyada precisamente en el estilo arquitectónico referido.

---

[35] Figueroa-Gutiérrez, S., Cortés, K., Contreras, G., Pérez, J. (2016). **Desarrollo de un Catálogo de Mecanismos de Seguridad**. *Research in Computing Science*. 128:57–67.

[36] Gómez, R., Cortés, K., Pérez, J., Arenas, A. (2014). **Desarrollo de una arquitectura orientada a servicios para un prototipo de una línea de productos de software**. *Research in Computing Science*, 79:75–86.

[37] Basili, V., Shull, F., Lanubile, F. (1999). **Building Knowledge through Families of Experiments**. *IEEE Transactions on Software Engineering*, 25 (4): 456–473.

[38] Ucán, J., Gómez, O., Aguilar, R. (2016). **Assessment of Software Defect Detection Efficiency and Cost through an Intelligent Collaborative Virtual Environment**. *IEEE Latin America Transactions*, 14(7):3364–3369.

[39] González-Jiménez, B., Contreras-Vega, G., Cortés-Verdín, K. (2012). **Redes Sociales para el Seguimiento de Egresados**. *Research in Computing Science*, 60:239–250.

[40] Zavaleta-Ibarra, L. A., Contreras Vega, G., Cortés Verdín, K., Pérez-Arriaga, J. C. (2014). **Desarrollo de la versión móvil para la red social FEIBook**. *Research in Computing Science*, 79:63–74.

En el área de procesos de software (AC_08) se ha implantado un modelo de referencia y evaluación de procesos en las normas mexicanas NMX-I-059/NYCE-2011 y NMX-I-1554-2010 que responden a las demandas de la industria de software.[41] Entre los resultados resaltan la identificación de factores de éxito y el desarrollo de herramientas para la autoevaluación de procesos. Asimismo, se han definido estrategias para la transferencia de conocimiento en equipos de mejora de procesos y desarrollo de software y se han identificado y caracterizado los tipos y flujos de conocimiento para promover procesos de innovación centrados en la mejora de procesos de software.[42] En el ámbito de la experimentación en entornos académicos se ha trabajado en la definición de métodos que incorporan una descripción ordenada, clara y concisa de lo que deben realizar los alumnos a lo largo de los cursos; cada método toma en cuenta los objetivos del curso y define su conjunto de prácticas descritas con Kuali-Beh;[43] dichos métodos tienen sus prácticas basadas en estándares como ISO/IEC 29110 y SCRUM.

En el área de métodos y modelos de la Ingeniería de Software (AC_09) se ha investigado la creación de modelos y estándares con el objetivo de ayudar a la industria de software a ser más competitiva. Uno de los proyectos más difundidos, con impacto a nivel internacional, ha sido la creación del MoProSoft[44] para las pequeñas organizaciones de software, el cual fue pu-

[41] Flores, B., Astorga, M., Rodríguez, O., Ibarra Esquer, J. E., Andrade, M. (2014). **Interpretación de las Normas Mexicanas para la implantación de procesos de software y evaluación de la capacidad bajo un enfoque de Gestión de conocimiento.** *Revista Facultad Ingeniería Universidad de Antioquia,* pp. 81–100, Colombia..

[42] Flores-Ríos, B., Pino, F., Ibarra-Esquer, J. E., González-Navarro, F. F., Rodríguez, O. (2014). **Análisis de Flujo de conocimiento en Proyectos de Mejora de Procesos software bajo una perspectiva multi-enfoque**. *Revista Ibérica de Sistemas y Tecnologías de la Información*, 14:51–66, Portugal.

[43] Ibargüengoitia, G., Oktaba, H. (2014). **Identifying the scope of Software Engineering for Beginners course using ESSENCE**. En *Software Engineering: Methods, Modeling, and Teaching, vol. 3*. Eds. C. M. Zapata, L. F. Castro. Universidad Nacional de Colombia. Medellín, Colombia 2014, pp. 67-76. lases.cidenet.org/index.php/es/descargas/libro-2014

[44] Oktaba, H. (2005). **Moprosoft: A Software Process Model for Smail Enterprises**. *Proceedings of International Research Workshop for Process Improvement in Small Settings,* 19-20 de Octubre, Pittsburg, EEUU, Special Report CMU/SEO-2006-SR-001, pp. 93-101.

blicado como norma mexicana MNX-I-059-NYCE; este modelo se aceptó por la ISO/IEC JTC1/SC7 *Software and System Engineering* como base para la creación del estándar ISO/IEC 29110 para *Very Small Entities* (VSEs) de la industria de software. Adicionalmente, se definió un método de evaluación de procesos de software denominado EvalProSoft y poco después se trabajó en un tercer proyecto orientado al diseño de pruebas controladas cuyo objetivo fue demostrar que las empresas pueden elevar sus niveles de capacidad adoptando MoProSoft en un tiempo relativamente corto. Otro proyecto de la comunidad en esta área fue el desarrollo de un nuevo estándar llamado *RFP A Foundation for the Agile Creation and Enactment of Software Engineering Methods*, con la propuesta de *KUALI-BEH: Software Project Common Concepts*,[45] la cual se integró en la propuesta de ESSENCE 1.0 publicada como estándar de OMG.

5.5. Tendencias de la Ingeniería de Software

Algunos pioneros de la Ingeniería de Software han proyectado el futuro de esta disciplina, como Barry Boehm,[46] quien enunció las principales características que presentan actualmente los sistemas software y las que presentarán en el futuro. Boehm sostiene que la evolución de esta disciplina se caracterizará por un incremento considerable del tamaño, complejidad, diversidad en contenido y apertura a la interacción con otros sistemas. También augura que para 2020 habrá tendencias computacionales muy variadas, como nuevos tipos de plataformas inteligentes (materiales inteligentes, nanotecnología, dispositivos micro mecánico-eléctricos, componentes autónomos para sensado y comunicación -MEMS) y nuevos tipos de aplicaciones (redes de sensores, materiales configurables o adaptativos, adaptación de prótesis humanas) así como el desarrollo de la bioinformática.

[45] Morales, M., Oktaba, H., Piattini, M. (2015). **Making-Of an OMG Standard**. *Computer Standards & Interfaces Journal*, 42:84–94, Elsevier.

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

En el contexto de la bioinformática, los sistemas a construir serán complicados ya que incluirán:

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.

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).

Todas estas tendencias computacionales, principalmente vistas como sistemas y aplicaciones a construir, presentan retos para la Ingeniería del Software tales como i) la especificación de configuraciones y comportamientos, ii) la generación de aplicaciones y sistemas con base en las especificaciones, iii) la verificación y validación de la capacidad, rendimiento y fiabilidad, y iv) la integración de los sistemas en otros aún más complejos (sistemas de sistemas).

La diversidad y complejidad de los sistemas que se desarrollarán en los próximos años impondrán grandes retos a la Ingeniería de Software. El análisis presentado por Boehm permite visualizar los campos a los que se deberá orientar la Ingeniería de Software, ya que tradicionalmente se ha enfocado más a temas específicos de la misma disciplina y de la Ciencia de la Computación.[47] La globalización de los sistemas es una realidad y hay al menos tres paradigmas emergentes que plantean grandes retos a la Ingeniería de Software, como sigue:

1. Computación en la nube ("Cloud Computing"): Éste es un tipo de computación basada en Internet para habilitar el acceso por demanda a recursos informáticos compartidos, configurables y ubicuos (por

[47] Wang, Z., Li, B., Ma, Y. (2106). **An Analysis of Research in Software Engineering: Assessment and Trends**. https://arxiv.org/ftp/arxiv/papers/1407/1407.4903.pdf.

ejemplo, redes de computadoras, servidores, almacenamiento, aplicaciones y servicios).[48] El reto es la identificación de la calidad de los servicios; sin embargo, las capacidades y herramientas para enfrentarlo son limitados y los problemas en la concepción de sistemas y generación de soluciones son complejos.

2. Computación social ("Social Computing"): Este tipo de computación se refiere a los sistemas que permiten obtener, representar, procesar, usar y difundir información que se distribuye a través de colectividades sociales, tales como equipos, comunidades, organizaciones y mercados electrónicos.[49] Estos sistemas deberán acceder a 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, lo que involucra considerar arquitecturas complejas.

3. Datos masivos ("Big Data"): El término "Datos Masivos" se refiere a conjuntos de datos complejos de grandes dimensiones cuyos requerimientos de procesamiento y almacenamiento superan las capacidades de las aplicaciones tradicionales.[50] Se requiere encontrar patrones repetitivos dentro de los datos. Los retos para el tratamiento de los datos masivos incluyen el análisis, captura, búsqueda, intercambio, almacenamiento, transferencia, visualización, consultas, minería, privacidad y actualización de la información.

---

[48] Mell, P., Grance, T. **The NIST Definition of Cloud Computing (Technical report)**. National Institute of Standards and Technology: U.S. Department of Commerce, September 2011.

[49] Doug, Schuler, D. (1994). **Social Computing, introduction to Social Computing.** *Special edition of the Communications of the ACM*, 7 (1):28 –108.

[50] Chris Snijders, Ch., Matzat, Reips, U. D**.** (2012). **Big Data: Big gaps of knowledge in the field of Internet**. *International Journal of Internet Science*. 7:1–5.

Con las tendencias antes citadas, seguramente dos temas propios de la Ingeniería de Software estarán involucrados:

1. Interfaces de usuario adaptativas. La diversidad actual de los paradigmas de computación, así como las capacidades de las tecnologías de la información y las redes de conectividad, junto con los datos recopilados por los sensores disponibles en los dispositivos inteligentes, posibilitan la creación de experiencias personalizadas e inmersivas para el usuario, así como rastrear las interacciones, registrarlas y analizarlas en tiempo real. Estas capacidades proporcionan una oportunidad para diseñar la adaptabilidad y ofrecer una mejor experiencia de usuario discreta y transparente. Todo sistema requiere una interfaz de usuario, la cual generalmente es gráfica. Una interfaz gráfica de usuario adaptativa (GUI, por sus siglas en Inglés) tiene el potencial de optimizar el desempeño y la satisfacción de usuario mediante la adecuación automática de la funcionalidad a las capacidades de cada usuario específico.[51] El desarrollo de software "Front-end" es tan importante como el software "Back-end", así que las compañías necesitan tener ingenieros de interfaces de software que puedan crear aplicaciones móviles y aplicaciones Web intuitivas y enfocadas al usuario. Ya sea para una aplicación empresarial o para usuarios individuales, los ingenieros de software, en conjunto con el diseñador de la interfaz, deben construir una experiencia satisfactoria para el usuario final. No sólo se requiere software funcional también intuitivo y fácil de utilizar.[52]

2. Administración de proyectos. Debido a la diversidad y complejidad de los sistemas y aplicaciones es necesaria la formación de equipos de trabajo, los cuales frecuentemente deben ser multidisciplinarios. Gestionar en forma adecuada un proyecto es un reto que seguirá presente en el campo de la Ingeniería de Software. Los principales problemas

[51] Findlater, L., Gajos, K. (2009**). Design Space and Evaluation Challenges of Adaptive Graphical User Interfaces**. *AI MAGAZINE*, 30(4):68–73.

[52] Ver capítulo de Interacción Humano-Computadora en este texto.

que se tienen en la administración de proyectos actualmente son la planificación incompleta de proyectos, la estimación de costos de software y la baja precisión en los criterios de selección de las mejores técnicas de análisis, diseño y pruebas.

Aunado a estos problemas y retos se tiene la apertura cada vez más grande al trabajo distribuido en tiempo y en espacio. La globalización conduce a una fuerza laboral diversa en términos de lengua, cultura, resolución de problemas, filosofía de gestión, comunicación de las prioridades y de interacción persona a persona. Para cuestiones de gestión en un contexto distribuido se requiere contar con dos elementos vitales: conectividad y colaboración.[53] La conectividad se debe habilitar por comunicación de alto ancho de banda y la colaboración se debe dar entre equipos de software que no ocupan el mismo espacio físico y que requieren de medios electrónicos y del empleo a tiempo parcial en un contexto local y no solamente en tareas pequeñas de trabajo por "pares".

[53] Vijiyan, G. (2015). **Current Trends in Software Engineering Research**. En *Proc. 3rd International Conference on Emerging Trends in Scientific Research,* At Peal International Hotel, Kuala Lumpur, Malaysia, https://www.researchgate.net/publication/282059722_Current_Trends_in_Software_Engineering_Research