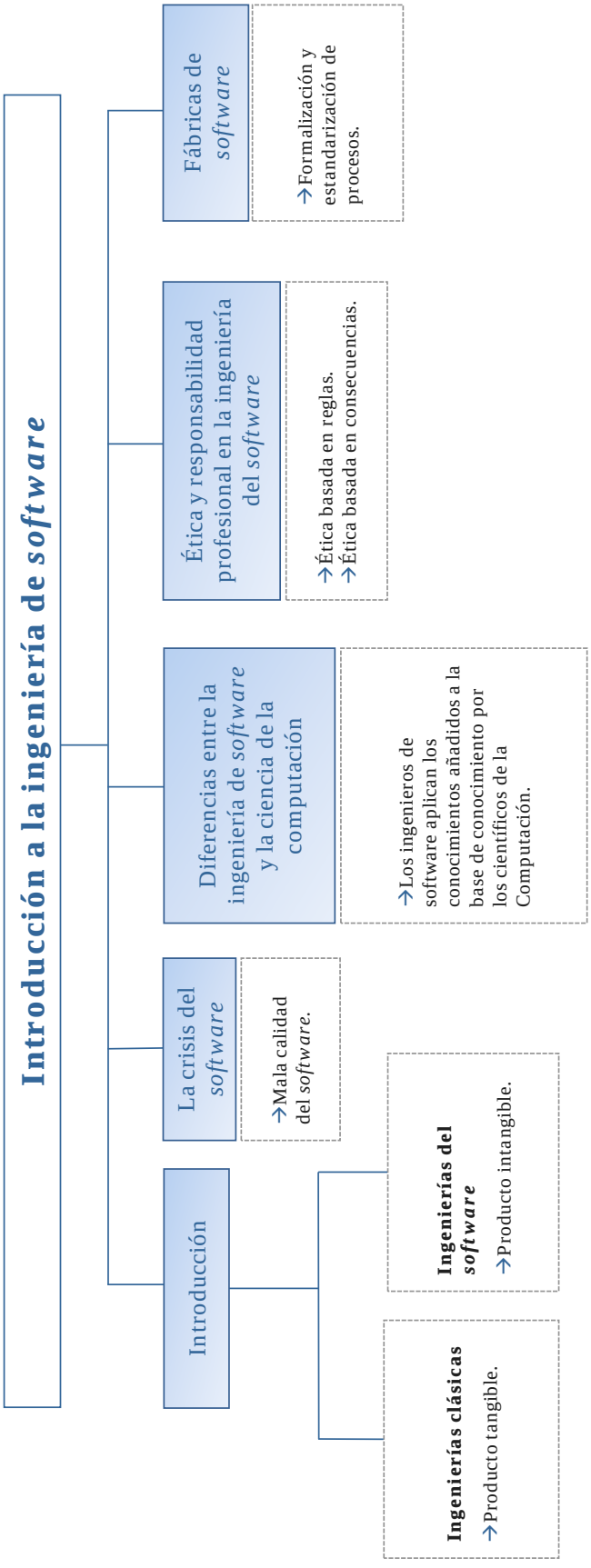
Introducción a la ingeniería de *software*

TEMA 1

Esquema



Ideas clave

1.1. ¿Cómo estudiar este tema?

Para estudiar este tema **lee las «Ideas clave»**, además del capítulo 1 (páginas 3-12), disponible en el aula virtual bajo licencia CEDRO, del libro:
Sommerville, I. (2005). *Ingeniería del Software*. Séptima edición. España: Pearson Addison-Wesley.

En este tema se estudian los conceptos básicos asociados a la Ingeniería de *software* y el motivo de su denominación como ingeniería. Con el estudio de este tema lo que se pretende es concienciar y entender sobre la necesidad de una disciplina que gestione y formalice el dominio del desarrollo de *software*, con el objetivo de implementar aplicaciones de calidad y de forma ética.

1.2. Introducción

La **ingeniería**, de manera genérica, se puede definir como «la **profesión** en la que el **conocimiento** de las ciencias naturales y matemáticas, ganado con estudio, experiencia y práctica, es **aplicado** con buen juicio para **desarrollar** formas de utilizar, económicamente, los materiales y las fuerzas de la naturaleza para el beneficio del género humano» (*Accreditation Board for Engineering and Technology*, 1996).

Por otro lado, el ***software*** es «la suma total de los programas de ordenador, procedimientos, reglas, documentación asociada y datos que pertenecen a un sistema de cómputo» (Lewis, 1994). Dicho de otra manera por el mismo autor, el ***software*** es «un producto diseñado para un usuario» (Lewis, 1994).

En este sentido, la **ingeniería de *software*** es entendida como «una **disciplina** de la **ingeniería** que comprende todos los aspectos de la **producción de *software***, y que abarca desde las etapas iniciales de especificación del sistema, hasta su mantenimiento una vez puesto en funcionamiento» (Sommerville, 2005, 6). A esta definición, cabría añadir, que la ingeniería de *software* ha de tener el objetivo de generar productos *software* de **calidad**.

Otra forma de entender la ingeniería de *software* es como el **estudio de métodos** y **herramientas** que se pueden utilizar para producir ***software* práctico** (es decir, *software* operacional desarrollado dentro de los límites económicos y organizativos de la empresa desarrolladora), **útil y de calidad** (es decir, *software* correcto y, si es posible, reutilizable).

Sin embargo, a pesar de estar clasificada y tratada como una ingeniería, la ingeniería de *software* tiene unas características muy particulares que le hacen distinguirse de las ingenierías clásicas y que se listan a continuación:

- El producto generado es intangible.
- Implica mucho desarrollo, que no es lo mismo que fabricar en un sentido clásico, por lo que no sigue las pautas ingenieriles.
- Cada solución *software* puede requerir (y normalmente requiere) un enfoque distinto, específico, con uso de diferentes técnicas, tecnologías, herramientas...
- Los requisitos del *software* cambian constantemente. No se puede esperar a que el diseño esté completo para empezar a codificar.
- En mayor o menor medida, requiere describir y documentar los artefactos que se van a generar.

El no tener en cuenta todas estas propiedades inherentes al desarrollo de *software* y tratar de fabricar *software* como si de un producto físico se tratase, hará que surjan inevitablemente problemas a la hora de planificar, gestionar y desarrollar la solución. En las ingenierías clásicas hasta que no se tiene el diseño completo no se pasa a la construcción, como ocurre por ejemplo, cuando se construye un edificio.

Sin embargo, en la ingeniería de software, debido a los frecuentes cambios, muchas veces el diseño y la codificación se acaban solapando. Por tanto, cuando se trata de desarrollar *software* no se puede ser tan tajante en la separación de diseño y codificación como ocurre en el resto de ingenierías en lo que respecta a la separación de diseño y construcción.

Otra diferencia fundamental entre las ingenierías clásicas y la ingeniería de *software*, es que los **productos fabricados en las ingenierías clásicas presentan un carácter continuo**, es decir, pequeñas modificaciones en el producto producirán pequeños cambios en el comportamiento. Sin embargo, los **productos *software*** suelen tener un **carácter discreto**, es decir, pequeñas modificaciones en un producto

software puede desencadenar en una gran cantidad de efectos colaterales que podrían producir cambios impredecibles e incontrolados en su comportamiento.

Además, las ingenierías clásicas se basan en unas teorías (normalmente matemáticas o físicas) que permiten verificar las propiedades por medio de cálculo, que no se da en la ingeniería de software. La Tabla 1 refleja esta situación. En la ingeniería de *software* se parte de muchos requisitos y a la hora de desarrollar una determinada aplicación no se sabe realmente qué es lo que se debería calcular para asegurarse de que el *software* va a estar bien construido.

Tabla 1: Propiedades inherentes de diferentes Ingenierías

Ingeniería	Producto	Teoría	Propiedades
Civil	Puente	Mecánica clásica	Carga máxima, resistencia lateral...
Aeronáutica	Avión	Dinámica de fluidos	Relación de planeo, turbulencias...
Software	*Software*	¿?	¿?

1.3. La crisis del *software*

Durante las últimas décadas la disciplina conocida como ingeniería del *software* se ha enfrentado a diferentes problemas relacionados con el desarrollo de *software*, englobados en lo que se ha denominado «**crisis del *software***»:

- Planificaciones imprecisas que difícilmente se llegan a cumplir, superando los plazos de entrega en la mayoría de los proyectos *software*.
- Los costes del proyecto suelen superar con creces el presupuesto inicial.
- Índices de productividad muy bajos.
- Clientes insatisfechos con las soluciones implementadas ya que no satisfacen sus necesidades.
- La calidad del *software* es inaceptable.
- El producto *software* desarrollado resulta muy difícil de mantener.
- El producto *software* no está integrado o ni siquiera alineado con el negocio de la compañía.
- El departamento de las Tecnologías de la Información (TI) suele verse como un freno al negocio.

Como consecuencia de la denominada «crisis del *software*», la ingeniería del *software* surge como concepto en 1968, en un congreso organizado por la Organización del Tratado del Atlántico Norte, OTAN (del inglés *North Atlantic Treaty Organization*, NATO) donde se intentaba explicar el porqué de la bajísima calidad del *software* (NATO, 1968). El *software* era visto como un añadido y su programación era totalmente 'artesanal', dependiente exclusivamente del programador, para la que no existía ningún tipo de metodología, ni se realizaba ningún tipo de planificación.

Como conclusión en el congreso, se dedujo que el *software* era un producto tecnológico y que, por tanto, había que tratarlo como una ingeniería más, aplicando sus principios, pero trasladada al software, acuñándose así el término de 'ingeniería de *software*'. De esta manera, se pretendía que el ingeniero de *software* aplicara los principios ingenieriles a la hora de construir los productos *software*, en base a un método más sistemático, disciplinado, cuantitativo y menos artesanal.

Con lo cual, se puede decir que la ingeniería de *software* es una disciplina relativamente joven. La ingeniería de sistemas, que no hace referencia al mismo dominio que la ingeniería de *software*, es anterior y comprende todos los aspectos del desarrollo y evolución de sistemas complejos, no solo del *software*, aunque bien es cierto que el *software* representa un papel principal en todo el conjunto.

A lo largo del tiempo, el porcentaje de *software* en los sistemas se ha visto incrementado de manera muy significativa (por ejemplo, los sistemas intensivos de *software* son sistemas en los que aproximadamente el 90% está constituido por *software*), y las técnicas empleadas en la ingeniería del *software* se están aplicando de manera equivalente en el proceso de ingeniería de sistemas (por ejemplo, el modelado de requisitos con casos de uso y la gestión de la configuración). Precisamente la evolución del *hardware* y su caída de precios es lo que está causando que los sistemas *software* se compliquen cada vez más, y se haga más difícil llegar al final de la mencionada «crisis del *software*».

Pero, ¿por qué es tan difícil desarrollar *software*? Si uno se pone a indagar, son muy pocos los proyectos *software* que pueden atribuirse el haber terminado el producto *software* en los plazos estipulados, dentro del presupuesto establecido inicialmente, y además cumpliendo con los requisitos del cliente. En la web se pueden encontrar diversas historias de proyectos que han fracasado de manera estrepitosa, con errores que han generado un coste demasiado alto, y no solo desde un punto de vista económico. Por

ejemplo, entre los errores informáticos más importantes y recientes en España se encuentran los que se muestran en la Tabla 2.

Tabla 2: Errores informáticos de gravedad en España.

Año	Entidad	Descripción	Nivel de gravedad
2015	Avión A400M	Un fallo en el *software* del ordenador que controlaba los motores fue la causa del accidente de que un avión A400M se estrellara en Sevilla y que provocara la muerte de cuatro tripulantes y heridas a otros dos.	Alta
2013	Juzgados de Madrid	Un fallo informático colapsa 46 juzgados de Madrid durante 12 días.	Media
2012	Gobierno de Canarias	Como consecuencia de un error informático, el gobierno canario pierde datos fundamentales acerca de la erupción volcánica de 'El Hierro'.	Media
2012	Gobierno de Navarra	Un fallo en el sistema informático de la red corporativa del Gobierno de Navarra interrumpe los servicios de Salud, Hacienda y Desarrollo Rural.	Media
2012	Aeropuerto de Málaga	Un fallo informático deja a oscuras la torre de control del aeropuerto de Málaga.	Alta
2011	Periódico 'El País'	El periódico 'El País' casi no sale a la calle un domingo por un fallo en el sistema informático y la ausencia de técnicos para poder solventarlo.	Media
2003	Vodafone España	La red de telefonía móvil de Vodafone en España sufre una caída de servicio dejando a más de 8 millones de usuarios sin cobertura.	Alta

Fuente:http://www.javiergarzas.com/errores-informaticos-en-espana-y-lantinoamerica y http://elpais.com/

Desde la perspectiva económica, se encuentra también un listado con los errores más costosos de la historia, todos ellos originados por fallos informáticos, y que se ilustran en la Tabla 3.

Tabla 3: Errores informáticos más costosos de la Historia

Año	Entidad	Descripción	Nivel de gravedad
2010	Toyota	Toyota tuvo que retirar más de 400.000 de sus vehículos híbridos, debido a un problema *software* que provocaba un retraso en el sistema anti-bloqueo de frenos. Se estima que, entre sustituciones y demandas, el error le costó a Toyota unos 2,8 billones de Euros.	Alta
1999	NASA	Los ingenieros de la NASA perdieron el contacto con la sonda *Mars Climate Orbiter* en un intento que	Alta

Año	Entidad	Descripción	Nivel de gravedad
		orbitase en Marte. La causa fue debida a un programa que calculaba la distancia en unidades inglesas, mientras que otro programa utilizaba unidades métricas.	
1996	Agencia Espacial Europea (ESA)	El cohete Ariane 5 explotó debido a que un número real de 64 bits en coma flotante, relacionado con la velocidad, se convirtió en un entero de 16 bits. Las pérdidas se estiman en 400 millones de Euros.	Alta
1994	Intel	Un profesor de Matemáticas descubrió e informó acerca de un fallo en el procesador Pentium de Intel. La sustitución de chips costó a Intel uno 443 millones de Euros.	Alta
1988	Internet	Un estudiante de posgrado, Robert Tappan Morris, fue condenado por el primer ataque con 'gusanos' a gran escala en Internet. El coste de limpiar el desastre ocasionado por Robert se cifra en unos 100 millones de dólares.	Alta
1978	Hartford Coliseum	Apenas unas horas después de que miles de aficionados abandonaran el estadio Hartford Coliseum (Estados Unidos), el techo se derrumbó por el peso de la nieve. La causa fue debida al cálculo incorrecto que se había introducido en el *software* CAD utilizado para diseñar el estadio.	Alta
1962	NASA	La sonda espacial Mariner I se desvió de la trayectoria de vuelo prevista con destino a Venus poco después de su lanzamiento. Desde control se destruyó la sonda a los 293 segundos del despegue. La causa de la desviación fue una fórmula manuscrita que se programó de manera incorrecta. Las pérdidas económicas se estimaron en unos 15 millones de Euros.	Alta

Fuente: http://www.javiergarzas.com/2013/05/top-7-de-errores-informaticos.html

La dificultad para satisfacer las necesidades del cliente (¿problemas de incomunicación?), unido a la complejidad del *software* y a sus constantes cambios y evolución, tanto en fase de desarrollo como una vez entregados, hacen de la tarea de desarrollar productos *software* un proceso arduo y muy costoso (una gran parte del presupuesto en informática se debe a las modificaciones en el *software* ya implementado y desplegado).

El intento de sistematizar la ingeniería de *software* para fabricar productos *software* de la misma manera que se construyen productos físicos en las Ingenierías clásicas ya se ha comprobado que no es la solución, ni el camino a seguir, como ya se comentó en el apartado anterior, por las particularidades características del *software*, pero el

desarrollo de *software* tampoco es ni mucho menos algo artesanal como en sus orígenes. Es por ello que hay que intentar encontrar el equilibrio, y la ingeniería de software es la disciplina que se esfuerza en conseguirlo.

Tal y como afirma Ince (1993), los sistemas *software* son el puro reflejo de la rapidez a la que se mueve el mundo y la aceleración en los cambios que se producen: sistemas contables que han de actualizarse porque cambian las leyes fiscales, sistemas de defensa que han de actualizarse a la últimas tecnologías en materia de seguridad, ampliaciones de los sistemas de información como consecuencia de fusiones entre compañías que modifican sus operaciones...

De cara a poder adaptarse a este cambio tan acelerado que se da en la sociedad, Ince (1993) enumera tres requisitos que se han de satisfacer en el desarrollo de productos *software* para que sean viables y de calidad:

- Utilización de diferentes técnicas de desarrollo que permitan minimizar la complejidad de un sistema *software*.
- Utilización de métodos y conceptos que permitan a las compañías desarrolladoras de *software* y a sus clientes (ya sean externos o internos de la organización) poder valorar la naturaleza y validez de la solución *software* lo antes posible una vez iniciado el proyecto.
- Utilización de técnicas de desarrollo que permitan minimizar los efectos colaterales que se producen como consecuencia de las modificaciones y evolución del *software*.

El enfoque de Ince (1993) claramente defiende el enfoque ingenieril del *software* a través del uso de herramientas y metodologías que permitan obtener un *software* de calidad, pero sin dejar de tener en cuenta de las características propias inherentes al *software* a través de métodos que permitan gestionar los cambios y evolución del *software* a lo largo del tiempo.

1.4. Diferencias entre la ingeniería de software y la ciencia de la computación

Aunque los conceptos de ciencia e ingeniería están relacionados, no son lo mismo. Hay que tener claro que, los **científicos** están más vinculados al **descubrimiento** de nuevos conocimientos y su dedicación se debe a la prueba continua de fórmulas y teorías. Por su parte, **un ingeniero** trata de darle **aplicación** a los principios, experiencias y juicios emitidos por los científicos para resolver problemas o fabricar cosas que supongan un beneficio para las personas y la sociedad en su conjunto. Todo ingeniero va a llevar a cabo sus funciones dentro de un conjunto de restricciones definido y ha de asegurar la usabilidad, seguridad, fiabilidad, eficacia y eficiencia de sus productos.

Con lo cual, los ingenieros suelen ser vistos como innovadores que intentan hacer del mundo un sitio mejor para vivir. En particular, la figura del ingeniero de *software* se podría definir como el profesional de la industria informática que aplica los conocimientos disponibles y las teorías científicas en el ámbito de interés para construir, en este caso, productos *software* que mejoren la vida de las personas. Las restricciones a las que se va a ver sometido un ingeniero de *software* serán, entre otras, un presupuesto, la duración del proyecto, tiempos de respuesta del sistema *software* y tamaño de la aplicación (Ince, 1993).

Los científicos de la computación buscan nuevos conocimientos en el área del *software* y la informática en general, para que los ingenieros de *software* puedan encontrar nuevas formas de resolver, con productos *software*, problemas de la vida real en base a los nuevos hallazgos.

Otro aspecto que también debe quedar muy claro es que el término 'ingeniero de *software*' no es sinónimo de 'programador', aunque en muchos casos se asume que la única responsabilidad de un ingeniero de *software* es la de escribir código bien estructurado (Parnas, 1999) y las ofertas de empleo van en esa línea cuando se demandan ingenieros de *software* (¿para qué se demanda un ingeniero de *software* cuando lo que se necesita es realmente un programador? ¿Es más *cool* decir 'ingeniero de software'?).

Además de la programación, un ingeniero de *software* ha de conocer el entorno en el que el producto *software* va a estar operativo, y entenderlo a la perfección para implementar soluciones que cumplan los requisitos y funcionen correctamente en el entorno real. Por

tanto, los ingenieros de *software* han de aprender aspectos científicos y los métodos necesarios para poder aplicarlos.

Parnas (1999) resume de manera muy precisa las responsabilidades de un ingeniero de *software*:

- Conocer el dominio de la aplicación para poder definir y documentar los requisitos que se han de satisfacer, de una manera precisa, bien organizada y que se pueda consultar con facilidad.
- Participar en el diseño de la configuración de los sistemas, para determinar qué funcionalidad será llevada a cabo a través del *hardware* y qué funcionalidad será implementada a través de *software*.
- Analizar el rendimiento del diseño propuesto (ya sea de manera analítica o a través de simulación), para asegurar que el sistema propuesto cumplirá con los requisitos del cliente.
- Diseñar la estructura básica del *software* a través de la división de su funcionalidad en componentes y las interfaces que los interconectan, además de documentar todas las decisiones de diseño que han sido adoptadas.
- Analizar la consistencia, disponibilidad y completitud de la estructura del *software* propuesta para implementar la solución.
- Implementar el producto *software* en base a un conjunto de programas bien estructurados y documentados.
- De ser necesario, integrar la nueva solución con aplicaciones ya existentes o comerciales.
- Llevar a cabo las pruebas necesarias para poder validar y verificar el *software*.
- Revisar y mejorar los sistemas *software* sin perder la integridad conceptual y manteniendo actualizada toda la documentación asociada al proyecto.

1.5. Ética y responsabilidad profesional en la ingeniería del *software*

Si hay una expresión que define la ética mejor que ninguna otra es la conocida frase: «Trata a los demás como te gustaría que te trataran a ti». Con este principio comienza este apartado, para tratar de expresar la necesidad de no olvidar la ética y tener una conducta moral cuando se desarrolla *software*.

Al igual que ocurre con las ingenierías clásicas, la ingeniería de *software* ha de estar dentro de un marco legal y social que debe limitar la libertad de los ingenieros a la hora de desarrollar *software*, para que puedan ser respetados profesionalmente (Sommerville, 2005, 12-13).

Tal y como afirma Génova et al. (2007): «Sin una sólida educación ética específica de la profesión, el ingeniero se convierte en un mero instrumento técnico y despersonalizado en las manos de otros». Lo que ocurre, es que muchas veces, por las características propias del *software* desarrollado, puede ser complicado determinar las consecuencias morales y daños a personas que pueden generar. En cualquier caso, el ingeniero de *software* siempre ha de partir de la premisa de construir *software* que respete y valore a las personas de acuerdo a una conducta ética (Génova et al., 2007).

En cualquier caso, independientemente de la tendencia ética con la que uno se pueda sentir identificado, existe una necesidad de establecer unos principios morales que guíen el comportamiento ético en el desarrollo de sistemas *software*. Esta necesidad de establecer unos principios morales en el campo de la ingeniería de *software* ha sido recogida por el *Código de Ética y Práctica Profesional de Ingeniería de Software de la ACM / IEEE Computer Society*, para su adopción como estándar en la enseñanza y práctica general de esta disciplina (ya sea empresarial o de otro tipo, con fines lucrativos o no). En este código se definen 8 principios básicos que deben seguir todos los profesionales de la ingeniería de *software* a la hora de ejercer su profesión:

- **Público.** Los ingenieros de *software* deberán actuar de manera consistente en bien del interés general.
- **Cliente y contratista.** Los ingenieros de *software* deberán actuar de tal modo que sea del mejor interés para sus clientes y contratistas, en coherencia con el interés general.

- **Producto**. Los ingenieros de *software* deberán garantizar que sus productos y las modificaciones asociadas a ellos cumplen con el mayor número posible de los mejores estándares profesionales.
- **Juicio**. Los ingenieros de *software* deberán mantener la integridad e independencia en su valoración profesional.
- **Gestión**. Los gestores y líderes en ingeniería de *software* suscribirán y promoverán un enfoque ético en toda gestión de desarrollo y mantenimiento de *software*.
- **Profesión.** Los ingenieros de *software* deberán realizar avances en lo que se refiere a integridad y reputación de la profesión, en coherencia con el interés general.
- **Compañeros.** Los ingenieros de *software* serán justos y apoyarán a sus compañeros de profesión.
- **Persona**. Los ingenieros de *software* deberán participar en el aprendizaje continuo de la práctica de su profesión y promoverán un comportamiento ético en la práctica de la misma.

1.6. Fábricas de *software*

Desde el comienzo de la crisis en España en el año 2008, bastantes sectores en la industria se vieron afectados. Muchos negocios que tuvieron mucho éxito en los años anteriores se vieron abocados al fracaso. No fue este el caso de las llamadas fábricas de *software* o factorías de *software*, como también se las conoce.

Tal y como revelaba el artículo *Inmunes a la crisis* del periódico global 'El País' en el año 2010 , las fábricas de *software* dedicadas al desarrollo y mantenimiento de aplicaciones no se vieron afectadas, sino que han tenido mayor proliferación si cabe durante estos años de crisis. La necesidad de ahorrar que han tenido y tienen las empresas en épocas de crisis, les lleva a contratar sus necesidades en materia tecnológica a terceros, ya sean empresas grandes como IBM o Accenture, o en el ámbito de la pequeña y mediana empresa (pymes).

Tal y como se afirmaba en el artículo, «a las empresas les sale mucho más barato que su *software* más básico y masivo lo desarrollen y mantengan estas fábricas que hacerlo en casa». Además, «nuestras factorías están muy reconocidas a nivel internacional. Estamos entre los países de Europa con más factorías certificadas con el estándar CMMI».

Efectivamente, tal y como se recoge también en Piattini y Garzás (2010), el desarrollo de productos *software* es uno de los sectores de mayor crecimiento a nivel mundial constituyendo una de las principales actividades económicas que mueven el sector industrial. El *software* representa el motor del progreso en toda sociedad y los avances en tecnología se producen a una velocidad vertiginosa, haciendo que funcionalidades impensables en el pasado ahora se puedan hacer realidad.

El concepto de fábrica de *software*, al igual que el término de ingeniería de *software* surge también en el año 1968, pero en este caso en el congreso *International Federation for Information Processing* (IFIP), en el que se declaraba (Piattini y Garzás, 2010): «Parece que tenemos pocos entornos específicos (instalaciones de fábrica) para la producción económica de programas (...) Una fábrica proporciona energía, espacio de trabajo, distribución del trabajo, controles financieros, etc.» Fue la compañía Hitachi la que al fundar su factoría *Hitachi Software Works*, en el año 1969, adoptara por primera vez este término de «fábrica de *software*» ideado en el IFIP, en base a dos objetivos principales (Piattini y Garzás, 2010: pág. 6):

- Mejorar la productividad y la fiabilidad de los productos *software* en base a la estandarización y control de los procesos implicados.
- Transformar el *software* visto como un servicio desestructurado en un servicio con un nivel de calidad óptimo.

Con la misma necesidad, la *Systems Development Corporation* (EEUU) fundó la segunda fábrica de *software* a nivel mundial entre los años 1975-1976. En este caso, los retos a los que se enfrentaban eran los que se indican a continuación (Bratman y Court, 1975):

- Adoptar disciplinas y estándares para sistematizar el proceso de desarrollo de *software*.
- Visualizar y controlar de manera efectiva el proceso de producción.
- Estandarizar la fase de diseño del *software*.
- Reutilizar componentes *software*.

Con el tiempo, las fábricas de *software* han ido evolucionando y adaptándose a las nuevas herramientas, tecnologías, enfoques y avances en la investigación en el campo de la ingeniería de *software*. Piattini y Garzás (2010) describen diferentes modalidades: Fábricas de software basadas en herramientas CASE (*Computer Aided Software*

Engineering), fábricas de *software* basadas en la experiencia para el desarrollo de componentes, fábricas de *software* basadas en la madurez de los procesos, fábricas de *software* basadas en la reutilización, fábricas de *software* para la renovación de *software* y fábricas de *software* enfocadas a técnicas específicas para la gestión de la calidad.

En definitiva, el término «fábrica de *software*» conlleva a desarrollar el *software*, de acuerdo a una organización específica del trabajo, formalizando y estandarizando los distintos procesos implicados (Piattini y Garzás, 2010). Las fábricas de *software* intentan adoptar el enfoque ingenieril e industrial en la fabricación de productos físicos en un intento de sistematización del proceso de desarrollo de *software*, pero teniendo en cuenta los aspectos particulares que presenta el *software*, lo que hace que se tenga que gestionar el proceso con un enfoque distinto.

Piattini y Garzás (2010) indican las tres necesidades principales que las fábricas de *software* intentan cubrir en el área del desarrollo y mantenimiento del *software*:

- Mano de obra cualificada: difícil de conseguir debido a la aceleración en los cambios tecnológicos o a los diferentes ciclos de la propia industria del *software*, que muchas veces se mueve por modas.
- Mejor gestión de los proyectos *software*: debido a la gran cantidad de fracasos que se han dado en la industria del *software*, muchas veces causados por la falta de madurez de la propia industria y otra veces debido a la complejidad, cada vez más mayor, de los sistemas a desarrollar.
- Adaptación a las nuevas tecnologías: debido a la evolución continua de las tecnologías, las empresas se ven obligadas a cambiar constantemente si quieren seguir siendo competitivas.

1.7. Referencias bibliográficas

Accreditation Board for Engineering and Technology (ABET) (1996). http://www.abet.org/

Bratman, H. y Court, T. (1975). *The Software Factory*. *Journal Computer*, 8(5), 28-37.

Génova, G., González, M.R. y Fraga, A. (2007). Ethical education in software engineering: responsibility in the production of complex systems [Educación ética en ingeniería de software: responsabilidad en producción de sistemas complejos]. *Journal on Science and Engineering Ethics*.

Ince, D.C. (1993). *Ingeniería de software*. Estados Unidos: Addison-Wesley Iberoamericana.

Laudon, K.C. (1995). Ethical Concepts and Information Technology. *Communications of the ACM*, 38 (12), 33-39.

Lewis, G. (1994). What is Software Engineering? *DataPro (4015)*. 1-10.

NATO Science Committee. (1968). *Software Engineering. Report of the NATO Software Engineering Conference 1968* sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 October 1968. Brussels, Scientific Affairs Division, NATO 1969. P. Naur and B. Randell, (Eds.).

Parnas, D.L. (1999). Software Engineering Programs Are Not Computer Science Programs. *Journal IEEE Software*, 16(6), 19-30.

Piattini, M.G. & Garzás, J. (2010). *Fábricas de Software: Experiencias, Tecnologías y Organización*. 2ª edición actualizada. Ra-Ma.

Sommerville, I. (2005). *Ingeniería del Software*. Séptima edición. España: Pearson Addison-Wesley.

Lo + recomendado

No dejes de leer...

Educación ética en ingeniería del *software*

Génova, G., González, M.R. y Fraga, A. (2007). Ethical education in software engineering: responsibility in the production of complex systems [Educación ética en ingeniería de software: responsabilidad en producción de sistemas complejos]. *Journal on Science and Engineering Ethics*.

En este artículo se trata la educación ética de los ingenieros informáticos, que los autores denominan «deontologismo moderado», la cual tiene en cuenta tanto las reglas como las consecuencias para valorar la bondad de las acciones y al mismo tiempo presta una adecuada consideración a los valores absolutos de la dignidad humana.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web: http://www.ie.inf.uc3m.es/grupo/docencia/reglada/Is1y2/Is1/ResponsabilidadIS.pdf

+ Información

A fondo

Software engineering

En este informe se recogen las actas de la *NATO Software Engineering Conference 1968* en la que el desarrollo de *software* pasó a clasificarse como Ingeniería, en un intento de sistematizar el proceso de desarrollo de *software* y acabar con la denominada «crisis del software».

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

Software engineering in the academy

Meyer, B. (2001). Software Engineering in the Academy. *Journal Computer*, 34(5), 28-35.

En este artículo se da una visión de la ingeniería de *software* enfocada a la formación de profesionales de *software*.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://www.inf.ed.ac.uk/teaching/courses/seoc/2006_2007/resources/meyer_teaching.pdf

Enlaces relacionados

JISBD

Jornadas de Ingeniería del Software y Bases de Datos (JISBD). Las JISBD son un foro de referencia en la investigación de la ingeniería del *software* y las *bases de datos* en el ámbito iberoamericano. Cita anual en la que los investigadores de España, Portugal y Latinoamérica pueden presentar sus trabajos y establecer una comunidad muy sólida alrededor de ambas disciplinas. Suelen dejar accesible en la web, de manera pública, las actas de las ediciones anteriores. Las JISBD constituyen una conferencia organizada bajo los auspicios de la *Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software* (Sistedes).

Accede a la página desde el aula virtual o a través de la siguiente dirección web:
http://www.istr.unican.es/sistedes2015/jisbd.html

SEI

Página web del *Software Engineering Institute* (SEI), instituto federal estadounidense de investigación y desarrollo, administrado por la Universidad Carnegie Mellon; una referencia en lo que se refiere a la Ingeniería de Software.

Accede a la página desde el aula virtual o a través de la siguiente dirección web:
http://www.sei.cmu.edu/

SEMAT

Página web de la comunidad *Software Engineering Method and Theory* (SEMAT), iniciativa creada en el año 2009 por Ivar Jacobson, Bertrand Meyer y Richard Soley con el objetivo de restablecer la Ingeniería de Software como una disciplina rigurosa para mejorar la calidad del *software*.

Accede a la página desde el aula virtual o a través de la siguiente dirección web:
http://semat.org/

SISTEDES

Página web de la *Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software* (SISTEDES), asociación sin ánimo de lucro, creada en 2005 con el principal objetivo de contribuir al desarrollo científico y tecnológico España en el área de la ingeniería del software y las tecnologías de desarrollo de *software*, así como de promover la investigación, la innovación y la transferencia de tecnología entre los distintos agentes involucrados en el avance de estas disciplinas.

Accede a la página desde el aula virtual o a través de la siguiente dirección web:
http://www.sistedes.es

Bibliografía

Brooks, F.P. (1975). *The Mythical Man Month and Other Essays on Software Engineering*. Addison-Wesley.

Mcconnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction, Second Edition*. Microsoft Press.

Piattini, M.G. & Garzás, J. (2010). *Fábricas de Software: Experiencias, Tecnologías y Organización*. 2ª edición actualizada. Ra-Ma.

Pressman, R. (2010). *Ingeniería del Software* (7ª ed.). Mcgraw-Hill.

Test

1. ¿Qué es la ingeniería de *software*?

A. Es una metodología para desarrollar *software* de calidad.

B. Es una disciplina que permite desarrollar *software* de calidad.

C. Es una disciplina que solo contempla el desarrollo del *software*, pero no su mantenimiento.

D. Es una metodología que contempla todo el proceso de producción del software.

2. ¿Qué es la ingeniería de sistemas?

A. Representa en detalle el sistema que va a ser desarrollado por el producto *software*.

B. Forma parte de la ingeniería de *software*.

C. Diseña la arquitectura de datos del producto *software*.

D. Permite entender el contexto en el que va a funcionar el producto *software*.

3. A la hora de desarrollar un producto *software*:

A. Hasta que no se termina el diseño jamás se empieza con la codificación.

B. Diseño y codificación pueden solaparse en el tiempo.

C. No se hace ninguna estimación.

D. Se tiene en cuenta el entorno en el que va a funcionar.

4. Las características de un producto *software* son:

A. Es algo intangible.

B. Es algo tangible.

C. No evoluciona con el tiempo.

D. Los cambios en el *software* pueden tener efectos colaterales.

5. Lo que diferencia la ingeniería de *software* de las ingenierías clásicas es:

A. No existe ninguna diferencia, funcionan igual.

B. Que la ingeniería de *software* se basa en las teorías matemáticas y las ingenierías clásicas en las teorías físicas.

C. Que en la ingeniería de *software* el producto que fabrica está sometido a muchos cambios.

D. Que en la ingeniería de *software* se estiman los costes del producto.

6. ¿Qué es la «crisis del *software*»?

A. Que las empresas ya no tienen tanta necesidad de productos *software* y hay crisis en el sector.

B. Que el *software* tiene una calidad muy baja.

C. Los proyectos *software* no cumplen con las planificaciones realizadas.

D. Que el *software* está alineado con el negocio.

7. La calidad del *software* siempre se mejora:

A. Utilizando técnicas de desarrollo que permitan minimizar la complejidad de un sistema *software*.

B. Contratando a los mejores programadores.

C. Disponiendo de un presupuesto muy alto.

D. Comprando las mejores herramientas CASE.

8. ¿Qué relaciona la ingeniería de *software* con la ciencia de la computación?

A. La ingeniería de *software* aplica los conocimientos añadidos por los científicos de la computación.

B. La ciencia de la computación aplica los conocimientos añadidos por los ingenieros de *software*.

C. La ciencia de la computación se enfoca en la programación del producto *software* ideado por el ingeniero de *software*.

D. No hay relación, se trata de lo mismo.

9. ¿Qué se entiende por responsabilidad ética del ingeniero de *software*?

A. Que siempre debe obedecer a sus superiores a la hora de desarrollar *software*.

B. Que construya el *software* sin pensar en el uso que se le va a dar.

C. Que considere las consecuencias que pueden producir la utilización del *software* que va a implementar.

D. El límite que tiene a la hora de desarrollar *software* bajo una conducta moral.

10. ¿Cuáles son los objetivos principales de las fábricas de *software*?

A. Conseguir que sus clientes ahorren en costes.

B. Visualizar y controlar de manera efectiva el proceso de producción.

C. Reutilizar componentes *software*.

D. Tener prestigio internacional.