UCAB UNIVERSIDAD CATÓLICA ANDRÉS BELLO

INGENIERÍA DE *SOFTWARE* Y EL PROCESO DE DESARROLLO

Tabla de contenido

Objetivos

Reconocer los modelos del proceso de desarrollo del *software* como mejores prácticas para la construcción de un *software* exitoso.

Caracterizar los diferentes modelos de los procesos de desarrollo del *software*.

Introducción

Estamos conscientes de cómo el *software* y la tecnología han cambiado (y siguen cambiando) nuestras vidas, nuestra forma de trabajar, de comunicarnos, etc. Con la pandemia, además, nos hemos visto forzados a adquirir más habilidades tecnológicas por su uso intenso.

Sin embargo, a pesar de todos sus avances, su calidad sigue siendo un tópico de investigación y de estudio. Por ejemplo, cuando buscamos una app en nuestro teléfono no nos preocupa si la habrá: seguro que encontraremos muchas; lo que quiere decir que no es su "funcionalidad" lo que influirá en la decisión sobre cuál descargar, sino su calidad (y el costo, por supuesto). ¿Qué hacemos? Recurrimos a la reputación. ¿Y qué es la reputación? La opinión de los consumidores/usuarios sobre su calidad. Cada vez tenemos usuarios/consumidores más exigentes y el *software* no siempre "da la talla".

Es por ello por lo que surgió la disciplina Ingeniería de *software*: para "disciplinar" el proceso de construcción de este; y como este evoluciona constantemente y el proceso de su construcción también, este proceso tiene nuevas exigencias cada día y tiende a ser muy complejo. Cuando el ser humano se enfrenta a procesos complejos los "modela" y ese es el tópico central de esta unidad: cuáles modelos del proceso de desarrollo del *software* existen, cuáles son sus beneficios, sus desventajas y cuál usar según el tipo de desarrollo.

Crisis de la ingeniería de *software*

El *Standish Group* es una organización que cuenta con una base de datos de más de 50.000 proyectos de Tecnologías de la Información (TI). Con esta data analiza sus tendencias y nos da un panorama de sus razones de éxito y de fracaso. Para hacer estas consideraciones se basa en el conocido "triángulo de hierro": costo, alcance y tiempo, tres variables sobre las cuales debemos hacer un equilibrio cuando se gestiona cualquier proyecto. En el reporte del 2018 introdujeron el concepto de éxito moderno: a tiempo, dentro del presupuesto y con resultado satisfactorio. (Portman, H., 2021).

Veamos las estadísticas de los últimos años (ver tabla 1):

	2011	2012	2013	2014	2015	2017	2020
Exitosos	29	27	31	28	29	36	31
Modificados	49	56	50	55	52	45	50
Fallaron	22	17	19	17	19	19	19

Fuente: Tabla 1. Estadísticas de los Proyectos, según los Reportes de Standish Group, desde los años 2011 hasta el 2020. Portman, H. (2021)

¿Sabes lo que significa un éxito que ha ido variando del 29 % al 36 %? Imagínate que vas a operarte con un cirujano y le preguntas: "doctor, ¿cuántas operaciones suyas han sido exitosas?". Y responde: "este año, un 31 %".

¿Te operarías con ese doctor?

¡Esto es grave! ¿Estás consciente de que por cada proyecto que inicies, tienes solo un 31 % de probabilidades de que termine de manera exitosa?

¡Ojo, estas son cifras del primer mundo! Es por ello por lo que nuestra profesión es altamente estresante**.** El mismo *Standish Group* nos da algunas pistas:

En este reporte (Portman, H., 2021) se cualifican las tres variables de éxito moderno:

- **Buen lugar** (*Good Place*): es donde el patrocinador y el equipo trabajan para crear el producto. Está compuesto por personas que apoyan tanto al patrocinador como al equipo. Estas personas pueden ser útiles o destructivas. Es imperativo que la organización trabaje para mejorar sus habilidades para que un proyecto tenga éxito. Esta zona es la más difícil de mitigar, ya que en cada proyecto participan cada vez más personas.
- **Buen equipo** (*Good Team*): es el caballo de batalla del proyecto. Ellos hacen el trabajo pesado. El patrocinador da vida al proyecto, pero el equipo toma ese aliento y lo usa para crear un producto viable que la organización pueda utilizar y derivar en valor. Dado que se recomiendan equipos pequeños, esta es la segunda área más fácil de mejorar.
- **Buen patrocinador** (*Good Sponsor*): es el alma del proyecto. El patrocinador da vida al proyecto y sin este actor no hay ningún proyecto. Mejorar las habilidades del patrocinador del proyecto es el factor de éxito número uno y también el más fácil de mejorar, ya que cada proyecto tiene solo uno.

En cada una de estas variables se proponen elementos a considerar para su mejora:

- **Buen lugar:** con madurez emocional, comunicación, participación del usuario/*sponsor*, negociación, sana competencia, optimización, ejecución rápida (horizontes de planificación cortos, muy cortos) y contar con una arquitectura empresarial.
- **Buen equipo:** atención plena, resolver los problemas, comunicación, respeto, confrontación (no aceptar las decisiones porque "son órdenes").
- **Buen promotor:** no retardar las decisiones, visión clara de producto (esto es muy difícil de lograr), influyente, apasionado, enfocado en el progreso del proyecto.

Es importante recordar que no estamos haciendo hamburguesas… Estamos construyendo un producto altamente complejo, que requiere de un trabajo intelectual y de la participación de personas que tienen diferentes perspectivas de lo que se debe construir: cliente, usuario, desarrollador, probador, líder del proyecto, etc.

Características del *software* como producto

- El *software* es abstracto e intangible. Es decir, no lo podemos "tocar".
- Complejo: está compuesto de muchas partes relacionadas y esto se acentúa con la interconectividad de múltiples dispositivos a través de internet.
- El *software* se desarrolla: no se fabrica en el sentido clásico.
- El *software* no se estropea: jamás vemos que una opción que usemos mucho, como "*copy*", por ejemplo, se borre o desaparezca. El *software* se obsoletiza muy rápidamente, lo cual significa que no es cuestión de cambiar una pieza por otra, como haríamos si el caucho de nuestro vehículo se daña o desgasta; se trata de que continuamente hay que "mejorarlo".
- La mayoría del *software* se construye a la medida, en lugar de ensamblarse por componentes.
- Ahora bien, durante estas siete décadas de desarrollo del *software* se ha concluido que, para construir uno exitoso, se deben seguir unas mejores prácticas. La intención y objetivo es actualizarse sobre algunas de ellas y ponerlas en práctica.

¡Eso es hacer ingeniería de *software*!

Mejores prácticas en el proceso de desarrollo de *software*

Las prácticas más fomentadas son:

- Desarrollar *softwares* iterativamente: tal como lo sugiere el *Standish Group*, hacer entregas a corto plazo. Esto implica hacer entregas que aporten valor en "rebanadas". Con ello se logra: detectar los malentendidos al inicio; se facilita el *feedback* entre los miembros del equipo (mejora la comunicación, pues se enfoca lo conversado sobre trozos pequeños del *software* que se está desarrollando); el equipo se concentra en lo esencial; se pueden hacer evaluaciones continuas del estatus del proyecto; las inconsistencias entre análisis, diseño e implementación se detectan tempranamente, lo que permite una mejor gestión de los riesgos. En la iteración siguiente se aplican las lecciones aprendidas y el promotor ve resultados a corto plazo.

- Usar arquitecturas probadas: propicia el modularidad, lo que hace que el *software* construido sea sostenible. También propicia el uso de *frameworks*, estilos y patrones arquitectónicos, que no son más que reutilización de diseño. A estos les dedicaremos varios temas porque son arquitecturas libres de errores.

- Representar/modelar el *software*: porque disminuye la ambigüedad, los detalles no necesarios se ocultan, se registran las decisiones arquitectónicas, se identifican los beneficios o desventajas de las decisiones arquitectónicas (esto se conoce como los *trade-off*), y porque un gráfico dice más que 1000 palabras.

- Verificación continúa de la calidad: porque se hace una evaluación objetiva de la calidad de lo que se está construyendo, se detectan inconsistencias entre análisis, diseño e implementación, gracias a las actividades de Aseguramiento de la Calidad (SQA). También se concentran en los aspectos de mayor riesgo: estos se identifican claramente y, por ende, se reducen los costos de su depuración.

Entonces es oportuno, en este momento, preguntarse: ¿cómo se desarrolla un *software* de calidad?

Calidad del *software*

La calidad del *software* se puede calificar, según Pressman y Maxim (2020), "...cuando un proceso eficaz de construcción de *software* se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan".

Revisemos mejor esta definición:

- **Proceso eficaz:** que todas las actividades realizadas aportan valor al objetivo
- **Producto útil:** aporta valor al promotor o a su organización
- **Proporciona valor medible a quienes lo producen y a quienes lo utilizan:** es decir, su construcción no solo beneficia al promotor, sino a todo el equipo de desarrollo.

Proceso de desarrollo del *software*

Un proceso:

- Es la colección de actividades de trabajo, acciones y tareas que se realizan cuando va a crearse un producto determinado.
- Es un conjunto de tareas relacionadas que generan un resultado de valor para el cliente.
- Aquí las palabras claves son: grupo, ya que en un proceso siempre participa más de una persona; relacionadas, ya que estas personas deben comunicarse, colaborar y cooperar durante la ejecución del proceso; resultado, ya que todo proceso debe generar algo (un documento, artefacto, producto, etc.); y cliente, ya que todo proceso es iniciado por alguien

(cliente/promotor/usuario) quien lo invoca. Por ejemplo, cuando entras en un banco para abrir una cuenta de ahorros, se inicia un proceso donde tú eres el cliente.

Para el caso del *software*:

- El proceso de desarrollo de *software* es complejo y, como toda actividad intelectual y creativa, depende mucho de las personas (Somerville, 2011).
- No hay proceso ideal de desarrollo de *software*: por regla general este es afectado por el tipo de *software* que se desarrolla, por quien lo desarrolla y para quien se desarrolla.

Por ello es por lo que, para su mejor entendimiento, se han desarrollado modelos de este proceso: el término "modelo" es polisémico (tiene más de un significado), por lo que da lugar a ambigüedades.

Algunos de esos significados son:

- Un objeto que se reproduce al imitarlo
- Es la muestra de un producto que se expone para su venta
- Búsqueda de una "perfección ideal" (los santos).

En la perspectiva epistemológica:

El modelo puede considerarse como una especie de descripción o representación de la realidad (hechos, situaciones, fenómenos, procesos, estructuras y sistemas, entre otros), que por lo general está en función de unos supuestos teóricos o de una teoría (suele presentarse en diferentes grados de abstracción).

Por tanto, el modelo es incompleto y nunca se da en el mundo real. El modelo se construye como un medio de ayuda para estudiar la realidad.

Volviendo al concepto de proceso, el que nos interesa en este curso es: el proceso de *software*. Un proceso de *software* es una secuencia de actividades que conducen a la producción de un producto de *software*. Este proceso se puede llevar a cabo de diferentes maneras.

A fin de optimizar esta secuencia y entenderla mejor, se formulan modelos de este proceso:

Modelos de proceso de desarrollo de *software*

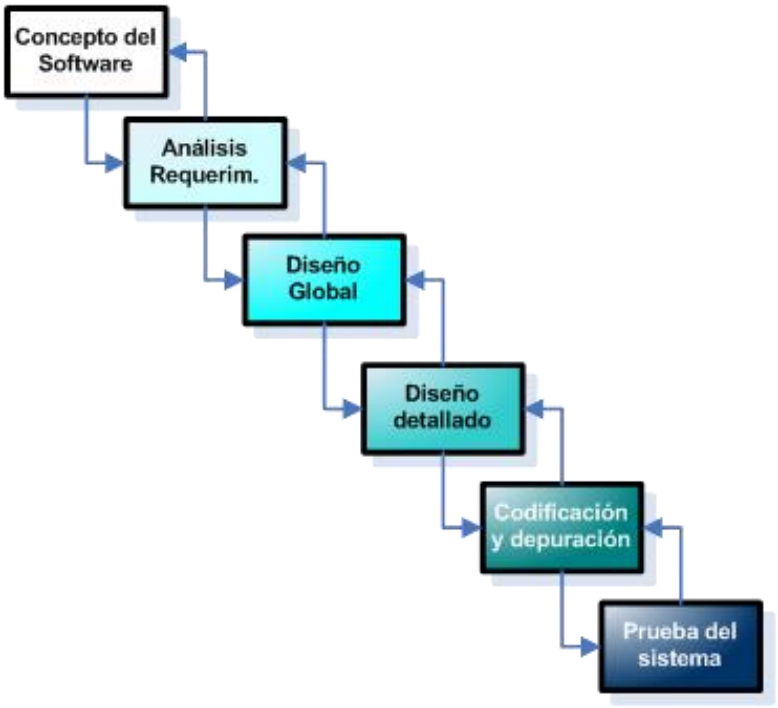
En la historia del desarrollo del *software* se han propuesto muchos modelos del proceso de desarrollo de este. Los podemos clasificar en:

- Lineal
- Iterativo
- Evolutivo
- Paralelo

¡Vamos a recordar cuáles son!

Lineal:

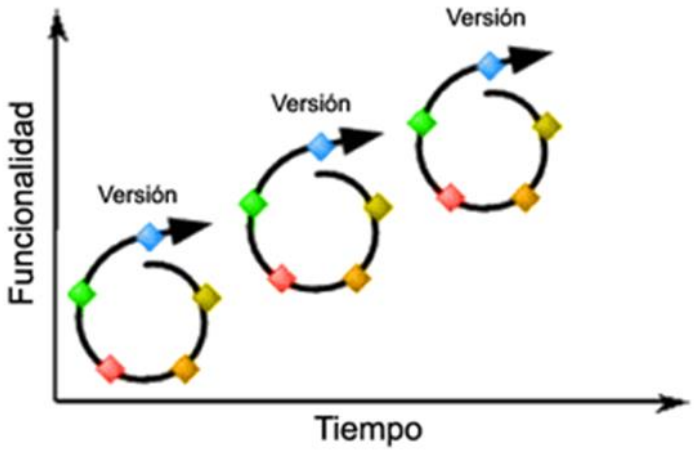
- Es el más conocido
- También lo llaman "cascada"
- Muy útil para proyectos pequeños, cuando se sabe muy bien qué es lo que se quiere construir y cuando se domina muy bien el lenguaje con el cual se va a construir.



Fuente: el modelo del desarrollo del *software*. Xavi (2013)

Iterativo:

- Son pequeñas cascadas
- Para proyectos grandes, rebanados en trozos
- Muy útil para:
    - cuando no se tiene claro lo que se va a construir.
    - poner en funcionamiento el *software* por partes.
    - cuando no se domina bien el lenguaje.
- Es la base del desarrollo ágil.
- Cada "cascadita" se puede conocer como un *Sprint* (*SCRUM*) o una historia (*XP*).



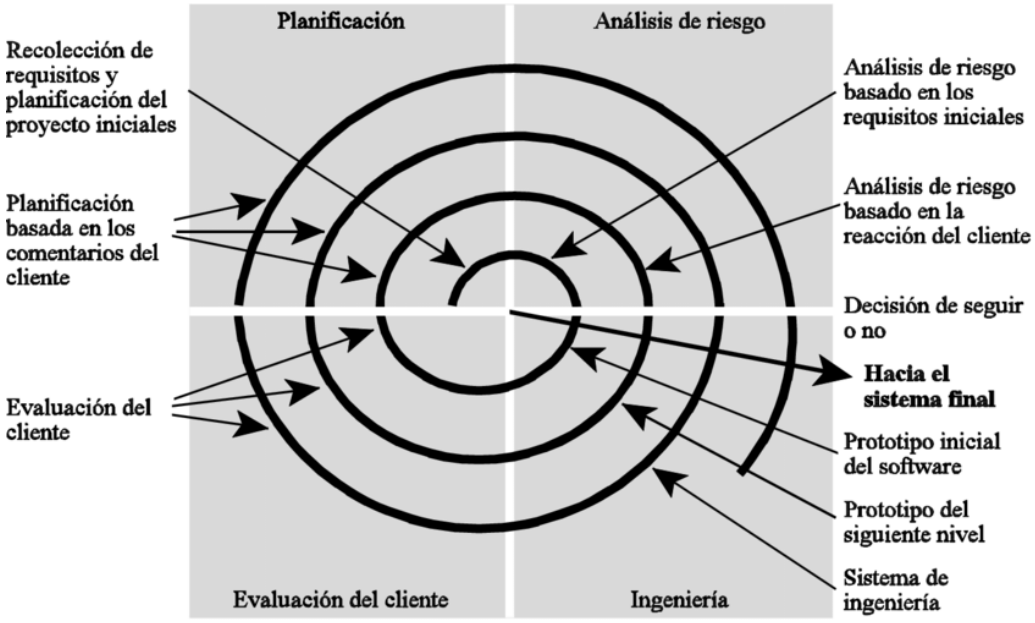
Fuente: STQB-Cap 2- Testing a través del ciclo de vida del *software*. Sarco (2012)

Evolutivo:

- Propuesto por Bohen en 1986. Plantea toda una revolución, porque:
    - Propone que se planifique el desarrollo del *software*, como cualquier otro proyecto. Esto era impensable en esos tiempos.
    - ¡Que se haga gestión de riesgos! Es una exigencia en todos los marcos metodológicos de gestión de proyectos actualmente, ¡pero no hace 40 años!
    - También se le conoce como espiral, pues para Bohen el proceso de desarrollo no termina nunca. ¿No es eso hoy día una realidad? ¿Cuándo los bancos "terminarán" el desarrollo de su portal?

- Finalmente, propuso una fase de evaluación con el cliente, la cual define si se continúa o no.

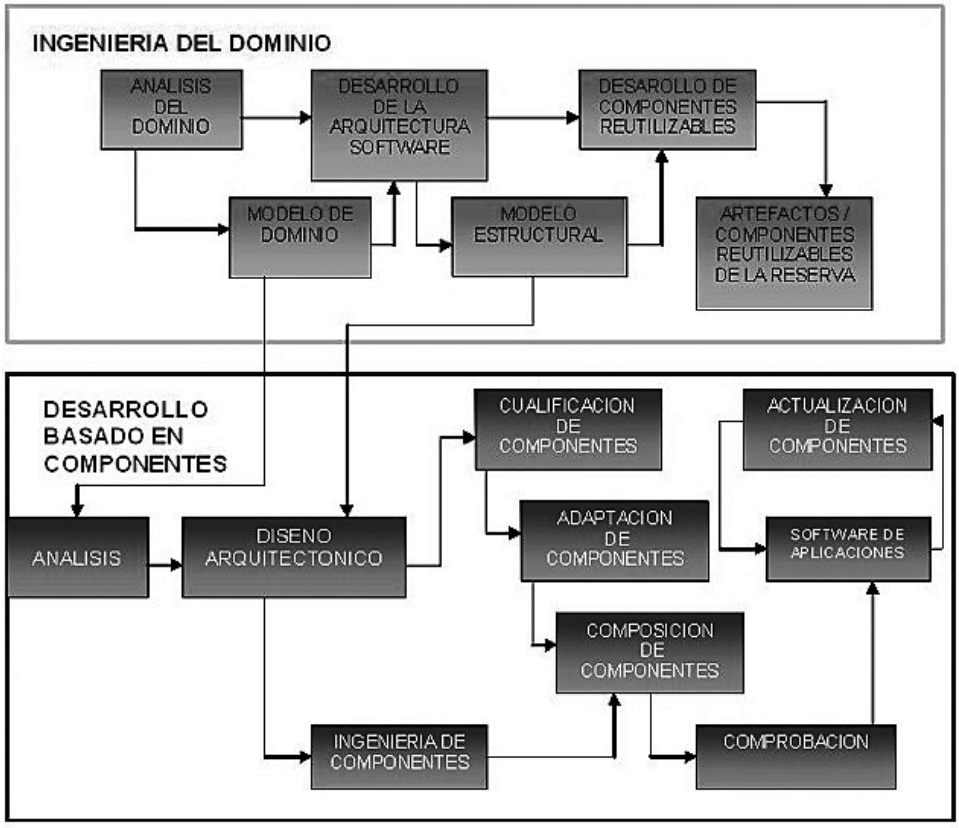
- Este modelo es altamente recomendado para:
  - Proyectos grandes
  - De alta incertidumbre
  - Con un retorno de inversión rápido.



Fuente: paradigmas en el desarrollo del software. Sulbaran (2014)

Paralelo:

- Modelo que se comienza a utilizar después de la llegada del paradigma de orientación a objeto.
- Se basa en el uso de Ambientes Integrados de Desarrollo (IDE), los cuales, a su vez, se basan en el uso de *frameworks*.
- Estos *frameworks* son jerarquías de clases desarrolladas a partir de un dominio.
- Por ello se le llama paralelo, pues se cuenta con dos grupos de desarrollo: los que desarrollan y ponen al servicio del otro grupo los ambientes de desarrollo, y los que desarrollan el *software* soportado en un IDE.
- Este último grupo de desarrollo debe conocer muy bien las bondades y limitaciones del IDE que utilizará para desarrollar su producto.



Fuente: ingeniería del dominio. Medina y Chaparro (2013)

Como puedes ver, cada modelo de proceso tiene sus beneficios y sus limitaciones. Por ello es muy probable que en tu organización se sigan algunos de ellos o alguna combinación. Aquí lo importante es estar conscientes de cuál es el que predomina, pues él nos estará imponiendo sus limitaciones.

También se puede dar el caso de que la organización no tenga muy claro o definido cuál es el proceso que se sigue, lo cual permite que cada equipo de desarrollo siga el proceso que considere; esto no es nada bueno, pues si cada equipo desarrolla *softwares* siguiendo el proceso que quieran, o no siguiendo ninguno, será muy difícil gestionarlo y, por ende, lograr eficiencia y calidad en el producto obtenido. Y menos aún, llevar alguna medición de éxito de los proyectos.

¿Y cuáles son las tendencias en el proceso de desarrollo del *software*?

- Una forma de trabajo más colaborativa entre las partes interesadas del proyecto: de esto ya precisamos algo al inicio con *Good Place* y *Good Sponsor*, equipos cada vez más integrados.

- **Agilidad:** esta es una tendencia muy fuerte, no solo en el proceso de desarrollo sino en su gestión como proyecto; horizontes cortos, muy cortos de planificación. De allí que el Marco Metodológico de Gestión de Proyectos más usado en los proyectos de desarrollo de *software* sea *SCRUM*, porque se basa en los *Sprint* que son cortas entregas; "calza" perfectamente con el modelo iterativo.
- **DevOps:** en muchas organizaciones de TI existe un "muro de confusión" entre el departamento/unidad responsable del desarrollo del *software* y el departamento/unidad responsable de su operación. Por un lado, la unidad de desarrollo quiere cambios, pero por el otro, la de operaciones quiere estabilidad. Entre ellos hay mucha confusión. Se propone un cambio para derribar este "muro de confusión", trabajando con una mentalidad DevOps. Los principios de esta son: los ingenieros de estos diferentes silos organizacionales están trabajando juntos para crear valor para el cliente; la mentalidad DevOps pone más énfasis en brindar calidad en un servicio, con responsabilidad de extremo a extremo (desde la identificación del requerimiento, hasta que se pone en operación, también llamado despliegue del *software*).
- **Mayores niveles de automatización:** ide todo! Gestión de los requisitos, desarrollo usando ambientes integrados de *software*, ambientes automatizados de pruebas y, si es posible, transición a producción de manera automatizada.
- **Integración continua (CI):** equipos integrados, es decir, que trabajan colaborativamente desde el inicio hasta el despliegue.
- **Formación con base al TModel:** esta es la tendencia en lo que a formación de los integrantes del equipo se refiere: formación tipo "T". Esto quiere decir: todos los miembros del equipo saben de todo. Cada uno se especializa, claro está, pero en cualquier momento uno puede sustituir a otro. Por ejemplo, el arquitecto podría eventualmente hacer las pruebas.

Cierre

- Se describió la situación de éxito de los proyectos de desarrollo del *software*, lo cual impulsa el estudio de la disciplina Ingeniería de *software*.

- Se especificaron las características que tiene el *software* como producto.

- Se describieron las cualidades de las mejores prácticas más fomentadas en el desarrollo del *software*.

- Finalmente, se actualiza la información sobre los modelos del proceso de desarrollo de *software*, recordando cuáles son sus tipos y sus ventajas.

Referencias

Portman, H. (2021). *Review Standish Group- CHAOS 2020 Beyond Infinity*. Henny Portman Blog. https://hennyportman.wordpress.com

Pressman, R. y Maxim, B. (2020). *Software Engineering. A Practitioner's Approach*. McGrawHill.

Sommerville, I. (2011). *Ingeniería de Software*. Editorial Pearson.

Referencias de las imágenes

Medina, S. y Chaparro, M. (2013). *Ingeniería del dominio* [Imagen]. Recuperado de Desarrollo de Aplicaciones WEB por Componentes – Código Libre. Publicaciones E Investigación, 7, pp. 33–51. https://doi.org/10.22490/25394088.1092

Sarco, J. (2012). *ISTQB- Cap 2- Testing a través del ciclo de vida del software* [Imagen]. Recuperado de Testing en español. https://josepablosarco.wordpress.com/2012/03/24/istqb-cap-2-testing-a-traves-del-ciclo-de-vida-del-software-i/