**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

| PROGRAMA DE FORMACIÓN | Estructura de arquitectura para un proyecto software |
| --- | --- |

| COMPETENCIA | 220501116- Diseñar la arquitectura del software de acuerdo con arquetipos y requisitos técnicos. | RESULTADOS DE APRENDIZAJE | 220501116-02- Implementar la arquitectura de acuerdo con el arquetipo definido.  220501116-03- Verificar la operatividad de la arquitectura según las especificaciones técnicas. |
| --- | --- | --- | --- |

| NÚMERO DEL COMPONENTE FORMATIVO | CF002 |
| --- | --- |
| NOMBRE DEL COMPONENTE FORMATIVO | Implementar la arquitectura de *software* |
| BREVE DESCRIPCIÓN | La arquitectura de *software* es la base donde se cimienta la estructura de un componente o aplicación de *software* a desarrollar, por esta razón, está orientada a la comprensión de los conceptos más importantes para tener en cuenta en la construcción de dicha estructura y pretende resolver los requerimientos no funcionales de un *software*. |
| PALABRAS CLAVE | diseño, arquitectura, estructura, *software* |

| ÁREA OCUPACIONAL | 2 - Ciencias naturales, aplicadas y relacionadas |
| --- | --- |
| IDIOMA | Español |

1. **Tabla de contenidos**

**Introducción**

**1. Conceptos del ciclo de vida del *software***

**2. Patrones de diseño definición y clasificación**

**3. Estilos arquitectónicos**

**4. Diagramas UML**

**5. Documentación de la arquitectura**

**6. Verificar atributos de calidad**

**7. Seguridad –OWASP**

**Síntesis**

1. **Introducción**

La arquitectura de *software* es la base donde se cimienta la estructura de un componente o aplicación de *software* a desarrollar. Le damos la bienvenida y le invitamos a ver el siguiente video:



**1. Conceptos del ciclo de vida del *software***

Según Ungoti (2021), el ciclo de vida del desarrollo de *software* (*SDLC – Systems Development Life Cycle*) “es la estructura que contiene los procesos, actividades y tareas relacionadas con el desarrollo y mantenimiento de un producto de *software*, abarcando la vida completa del sistema, desde la definición de los requisitos hasta la finalización de su uso, y en general, el ciclo de vida del *software* comprende los siguientes ítems:”.



Observe, a continuación, las definiciones de los ítems que conforman el ciclo de vida del desarrollo de *software* (SDLC) y que, a su vez, mantienen un orden de realización lógico:



Ahora, antes de continuar, es importante complementar con las fases del ciclo de vida del *software*, las mismas son necesarias para la validación del desarrollo de *software.* Lo invitamos a escuchar la siguiente audio nota:

**2. Patrones de diseño definición y clasificación**

Según Canelo (2021), “Por definición, un patrón de diseño es una solución a un problema de diseño cuya efectividad ha sido comprobada por haber sido empleada para resolver problemas similares en ocasiones anteriores. Otra característica fundamental es que debe ser reusable, lo que significa que sea aplicable a diferentes problemas de diseño en distintas circunstancias”.

Los patrones de diseño pretenden:

* Proporcionar catálogos de elementos reusables en el diseño de sistemas *software*.
* Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
* Formalizar un vocabulario común entre diseñadores.
* Estandarizar el modo en que se realiza el diseño.
* Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Según su enfoque, los patrones de diseño se agrupan en tipos creacionales, estructurales y de comportamiento.

**Patrones creacionales**

Para Canelo (2021), “Los patrones de creación proporcionan diversos mecanismos de creación de objetos, que aumentan la flexibilidad y la reutilización del código existente de una manera adecuada a la situación. Esto le da al programa más flexibilidad para decidir qué objetos deben crearse para un caso de uso dado”.

Observe los patrones creacionales a continuación:



**Patrones estructurales**

Para Canelo (2021), los patrones estructurales “facilitan soluciones y estándares eficientes con respecto a las composiciones de clase y las estructuras de objetos. El concepto de herencia se utiliza para componer interfaces y definir formas de componer objetos para obtener nuevas funcionalidades”.

Observe los patrones estructurales a continuación:



**Patrones de comportamiento**

Según Canelo (2021), “El patrón de comportamiento se ocupa de la comunicación entre objetos de clase. Se utilizan para detectar la presencia de patrones de comunicación ya presentes y pueden manipular estos patrones. Estos patrones de diseño están específicamente relacionados con la comunicación entre objetos”.



**Arquitectura de capas y de eventos**

Arquitectura de capas

Según Bertucelli (2019), “Esta arquitectura se destaca por ser fácil de implementar en el comienzo del proyecto y fácil de entender para principiantes en el desarrollo de *software*. Lo que propone la arquitectura de capas es pensar nuestro sistema en capas y cada capa debe exponer en forma clara las operaciones que puede realizar. Estas operaciones se deben exponer mediante un api que nos diga qué servicio ofrece esa capa y cuál será su retorno, sin importar cómo esté implementado”.

Por ejemplo, una URL es un api que me dice qué datos necesito ingresar y qué retornará, o también puede ser una clase con métodos públicos, etc. Pueden existir “n” capas, pero cada una debe tener una responsabilidad única. Una separación muy utilizada es la de tres capas “presentación”, “lógica de negocio” y “acceso a datos”.

“Una capa es un conjunto de “cosas” que tienen cierta responsabilidad. Por ejemplo, una capa puede ser un conjunto de clases, agrupadas en un paquete, dentro de nuestro programa, que representa cierta responsabilidad, o también puede ser un jar que se comunica con otros y cada jar representa una capa, o puede ser un sistema y cada sistema es en sí una capa. Cuando nos referimos a capas, es una abstracción de responsabilidades” (Bertucelli, 2019).

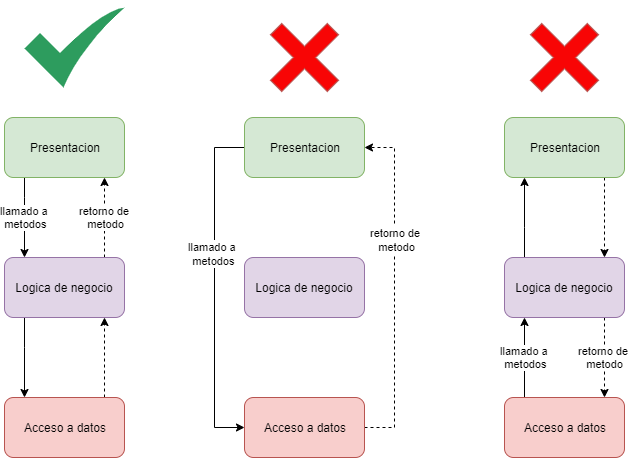
Además, la arquitectura establece dos reglas de cómo se deben comunicar las capas así; observe cómo las define Bertucelli:

**Primera regla:** “Cada capa debe tener una responsabilidad única. Es decir, las capas deben tener perfectamente delimitado de qué se ocupa cada una de ellas; por ejemplo, podemos tener una capa de “presentación”, que será la encargada de atender los eventos del cliente y encargada de representar la información para el mismo. Por otro lado, podemos tener la capa de “acceso a datos”, que será la encargada de guardar y acceder a los datos”.

**Segunda regla:** “Las capas deben respetar una estructura jerárquica estricta. Quiere decir que cada capa puede comunicarse solo con la que está debajo suyo, pero NO al revés. Por ejemplo, una clase ubicada en la capa de presentación puede llamar a un método ubicado en la capa de acceso a datos, pero nunca la capa de acceso a datos puede llamar a un método de la capa de presentación. Y cuando nos referimos a la próxima más baja significa que no se puede saltar capas”.

**Figura 1**

*Arquitectura de capas*



Nota. Adaptada de <https://somospnt.com/blog/118-arquitectura-de-capas>

Arquitectura de eventos

SegúnRed Hat (2021), “Este tipo de arquitectura puede basarse en un modelo de publicación/suscripción o en uno de flujo de eventos, así:

**Modelo de publicación y suscripción:** es una infraestructura de mensajería que se basa en suscripciones a un flujo de eventos. Con este modelo, una vez que se genera o publica un evento, este se envía a los suscriptores que necesitan estar informados al respecto.

**Modelo de flujo de eventos:** con este modelo, los eventos se escriben en un registro. Los consumidores no se suscriben a un flujo de eventos, sino que pueden leerlo desde cualquiera de sus partes y unirse a él en cualquier momento”.

Hay varios tipos de flujo de eventos:

* El procesamiento de flujos de eventos utiliza una plataforma de transmisión de datos, como Apache Kafka, para incorporar los eventos y procesar o transformar su flujo. Este procesamiento se puede utilizar para detectar patrones significativos en los flujos.
* El procesamiento de eventos simple surge cuando un evento desencadena inmediatamente una acción en el consumidor.
* El procesamiento de eventos complejo requiere que un consumidor de eventos procese una serie de ellos para detectar patrones.

Beneficios de la arquitectura basada en eventos

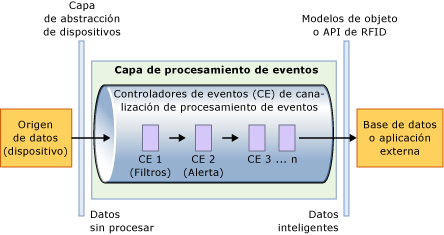
ParaRed Hat (2021), “Con una arquitectura basada en eventos, las empresas obtienen un sistema flexible que se adapta a los cambios y toma decisiones rápidamente. Conocer la situación de inmediato favorece la toma de decisiones empresariales, ya sean automatizadas o manuales, en función de todos los datos disponibles que reflejan el estado actual de los sistemas”.

Los eventos se captan a medida que ocurren desde sus fuentes, como las redes, las aplicaciones y los dispositivos del Internet de las cosas (IoT), lo cual permite que los productores y los consumidores compartan información sobre el estado y la respuesta de forma inmediata.

Las empresas pueden incorporar la arquitectura basada en eventos a sus sistemas y aplicaciones, para mejorar la capacidad de ajuste y de respuesta de las aplicaciones, y para acceder a los datos y al contexto necesarios para tomar mejores decisiones.

**Figura 2**

*Arquitectura de eventos*



Nota. Adaptado de http://arquitecturaorientadaaeventos.blogspot.com/2015/10/arquitectura-orientada-eventos\_22.html

**3. Estilos arquitectónicos**

Un aspecto muy importante en arquitectura de *software* son los patrones, tanto de diseño como de arquitectura, que son complementados por los estilos arquitectónicos. Los estilos favorecen el tratamiento estructural, la investigación y la teoría, mientras que los patrones están orientados o están más cerca del diseño, la práctica, el proceso o el código. Un estilo describe una clase de arquitectura o partes de esta, los estilos pueden ser reutilizados en situaciones semejantes, a futuro.

Un estilo arquitectónico es una abstracción de tipos de elementos y aspectos formales a partir de arquitecturas específicas o un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer el sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo.

**Componentes:** sistemas encapsulados indicando la forma de empaquetado y la forma en que interactúan con otros componentes.

**Conectores:** los procesos interactúan por medio de protocolos de transferencia de mensajes o por flujo de datos, etc.

**Estilo arquitectónico:** es una descripción del patrón de los datos y la interacción de control entre los componentes, ligada a una descripción informal de los beneficios e inconvenientes del uso del estilo.

**Estilos:** son entidades que ocurren en un nivel sumamente abstracto y puramente arquitectónico.

**Evolución:** Estructurado, Orientado a objetos, Orientado a componentes, Orientado a servicios.

**Catálogo de estilos arquitectónicos:**

Hay diferentes tipos de arquitectura, por lo cual le presentamos el catálogo de los estilos arquitectónicos más comunes, a través del siguiente video:

****

**4. Diagramas UML**

Con el nacimiento de la Programación Orientada a Objetos (POO), se establecieron una variedad de lenguajes, métodos y simbologías usadas para detallar el desarrollo de los diferentes sistemas. Indudablemente, esta situación generó un desconcierto en los programadores de *software*. Tratando de darle solución a este inconveniente, aparece el Lenguaje de Modelado Unificado, o UML por sus siglas en inglés (*Unified Modeling Language*).

El Lenguaje de Modelado Unificado proporciona un grupo de herramientas que facilitan realizar el modelado (análisis y diseño) de los sistemas que son orientados a objetos. Indicando una serie de normas técnicas que representan de manera gráfica el camino y la estructura de un sistema determinado a los programadores, analistas, desarrolladores, diseñadores, etc.

“El UML (Lenguaje Unificado de Modelado) es una de las herramientas más emocionantes en el mundo actual del desarrollo de sistemas. Esto se debe a que permite a los creadores de sistemas generar diseños que capturen sus ideas en una forma convencional y fácil de comprender para comunicarlas a otras personas”. (Schmuller, 2000).

En UML, existen dos tipos principales de diagramas que son: diagramas de estructura y diagramas de comportamiento. Entre los diagramas más relevantes (y más utilizados) que se encuentran en UML, están los diagramas de casos de uso, por lo que a continuación se realiza una definición más detallada de ellos.

**Diagrama de casos de uso**

Según Larman, los casos de uso permiten la descripción de cada una de las funciones de las aplicaciones a desarrollar, desde el punto de vista del usuario. Su utilización radica en expresar las funcionalidades que el producto de *software* debe efectuar y poder definir las personas que serán las encargadas de ejecutar estas funcionalidades. Es una técnica que consiste en hacer un diagrama con su respectiva notación.

El caso de uso representa el comportamiento del *software* en la interacción con el usuario, para que este alcance un objetivo; la notación de un diagrama de caso de uso se compone principalmente de tres (3) elementos que ayudan a representar simbólicamente las funcionalidades, personas involucradas y relación entre estas, y son: los actores, las relaciones y los casos de uso.

**Componentes de los diagramas de casos de uso**

La notación de un diagrama de caso de uso se compone principalmente de tres (3) elementos que ayudan a representar simbólicamente las funcionalidades, personas involucradas y relación entre estas, que son:

* Los actores
* Las relaciones
* Los casos de uso

A partir del siguiente ejemplo, observe en qué consiste cada uno de estos elementos:

“Una empresa solicita tener un sistema que permita hacer el proceso de generación de factura de venta a un comerciante, la cuál debe ser enviada por medio de un sistema externo que se encargará del proceso de la factura”.

Observe a continuación la explicación de cada elemento y cómo sería la representación de cada uno basado en el ejemplo indicado:



Además de la relación simple de comunicación que se muestra en el ejemplo trabajado, existen más relaciones en los casos de uso que especifican más detalladamente las acciones entre estos y que se mencionan a continuación:

Relaciones de casos de uso

En el proceso de desarrollo, pueden definirse diferentes casos de uso y estos no ser independientes, ya que se pueden establecer relaciones que dependan, uno del otro, entre los casos de uso.

Las principales relaciones consideradas por UML son:

* Inclusión (*include*)
* Extensión (*extend*)
* Generalización (*generalization*)

Estos tres tipos de relaciones que son soportadas por el estándar UML, se encuentran explicadas a continuación:

| Relación (concepto) | Notación (imagen a rediseñar) |
| --- | --- |
| Inclusión (*Include* o *use*): Se usa para extraer comportamientos comunes, un caso de uso puede incluir otro. |  |
| Extensión (*Extend*): Se utiliza para extender a otro caso de uso, es útil para nuevos requerimientos del sistema. |  |
| Generalización (*Generalization*): Relación entre un caso de uso general y uno específico que hereda propiedades del caso de uso base. |  |

Teniendo en cuenta lo anterior y con el propósito de perfeccionar el diagrama de casos de uso, se pueden agregar unos nuevos casos de uso donde se visualicen las relaciones mencionadas anteriormente, con sus respectivos ejemplos:

**<<*extend*>>:** se define esta relación cuando se cuenta con un caso de uso que es muy parecido a otro, pero que realiza mucho más.

Por ejemplo, en el caso a continuación, se puede ampliar el caso de uso envía factura hasta envía factura por correo electrónico. Ambos casos establecen la misma acción, la diferencia es que el caso de uso, además de enviar la factura al sistema externo, también envía un correo electrónico con la factura, y se representaría de la siguiente manera:

**Figura 3**

*Ejemplo diagrama de caso de uso venta con relación extend*











La relación de la figura anterior se lee de la siguiente manera: el caso de uso enviar factura por correo extiende del caso de uso enviar factura.

**<<*uses o include*>>:** se define esta relación cuando una parte del comportamiento o acción es equivalente en más de un caso de uso y no se requiere duplicar la representación de su actuación.

Por otro lado, tomando el ejemplo anterior, el mismo se amplía con el siguiente requerimiento funcional: antes de que el usuario envié la factura, puede realizar una *búsqueda de facturas* para seleccionarla y enviarla, haciendo necesario incluir la relación <<include>>. Por lo que se vería así:

**Figura 4**

*Ejemplo diagrama de caso de uso venta con relación include*













**Especificación de casos de uso**

Cuando se habla de especificación de casos de uso, se hace referencia al proceso de descripción textual de cada uno de los casos de uso, detallando flujos y eventos que interactúan con el sistema y los actores que participan en él; usualmente, el analista, que es la persona encargada del diseño de los casos de uso, a medida que va identificando los requerimientos, los va documentando.

La especificación de casos de uso busca:

* Realizar la descripción de la interacción de un actor con el sistema.
* Narrar el rol desempeñado por los actores en la interacción con el sistema y sus flujos.

A continuación, en la siguiente tabla, se describe una plantilla donde se indican las partes y las indicaciones básicas para la documentación de casos de uso.

**Tabla 1**

*Plantilla de casos de uso*

| DESCRIPCIÓN DE UNA ESPECIFICACIÓN DE CASO DE USO | |
| --- | --- |
| Nombre del caso de uso | Indica el nombre del caso de uso. Normalmente, el nombre expresa el resultado objetivo y observable del caso de uso, como, por ejemplo, "Imprimir factura", en el caso de un de un sistema de ventas. |
| Autor | Persona que diseña el caso de uso. |
| Fecha | Aquí se diligencia la fecha en la que se realizó el caso de uso. |
| Descripción | Se realiza una corta descripción del caso de uso. |
| Actores | Describe los nombres del actor o actores del caso de uso. |
| Precondiciones | Son aquellas condiciones que deben existir para que se cumpla el caso de uso. |
| Flujo normal | Son los pasos normales de ejecución del caso de uso. |
| Flujo alternativo | Flujos alternativos de ejecución de casos de uso. |
| PosCondiciones | Son aquellas condiciones que se cumplen cuando finaliza un caso de uso. |

A continuación, se muestra un ejemplo de especificación de casos de uso para el caso de uso “**Crear foro”** utilizando la plantilla propuesta en la tabla anterior.

**Tabla 2**

*Ejemplo especificación de caso de uso foro*

| DESCRIPCIÓN DE UNA ESPECIFICACIÓN DE CASO DE USO | |
| --- | --- |
| Nombre del caso de uso | Crear foro |
| Autor | Mario Vara Cuz |
| Fecha | 20/09/2021 |
| Descripción | Crea un nuevo mensaje. |
| Actores | Alumno/Docente |
| Precondiciones | Al iniciar, el usuario debe estar autenticado en el sistema. |
| Flujo normal | 1. El actor da clic en la opción nuevo mensaje.  2. Se muestra un área de texto para escribir el mensaje.  3. El actor introduce el mensaje.  4. El sistema verifica datos y los almacena.  5. El profesor acepta y publica. |
| Flujo alternativo | A. Se hace una comprobación de los datos, si no son correctos, muestra mensaje de corrección.  B. El profesor rechaza el mensaje, de modo que no es publicado, sino devuelto. |
| PosCondiciones | Se publica y almacena el mensaje satisfactoriamente. |

**Casos de uso reales**

Su objetivo principal es detallar el proceso de un sistema de información que se describe por medio de un caso de uso, incluyendo la interacción de objetos, así como definir las transacciones de las interfaces y clases de los diferentes procedimientos secundarios de diseño. De este modo, cuando se reconozcan las clases que participan internamente de un caso de uso, es fundamental que los entornos que se recolectan del análisis se completen, incorporando las correspondientes clases de diseño sin dejar a un lado las restricciones del ambiente tecnológico.

Los casos de uso reales representan el diseño preciso del caso de uso desde una tecnología en específico. Así pues, si se trata de un diseño gráfico para el usuario, se incluirá el diseño de ventanas de este y una descripción de la interacción con los objetos de la interfaz. En resumen, los casos de uso reales brindan diseños de pantalla y explicación de la interacción entre los objetos de esta.

Para aclarar las definiciones anteriores, observe un ejemplo de un sistema de ventas donde se realiza el diseño del caso de uso real y se hace la descripción de interacción entre los artefactos del diseño:

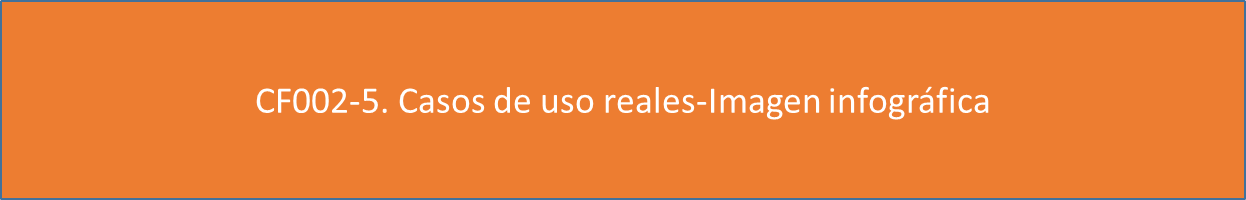
Caso de uso: Adquirir productos

Actores: cliente (quien inicia el proceso), cajero

Tipo: primario, real

Descripción: en el almacén de ropa, el cliente se dirige a la caja registradora con los productos a comprar. El empleado-cajero hace el registro de los productos que el cliente compra y recibe el dinero en efectivo. Al finalizar el proceso, el cliente se retira con los artículos comprados.

El inicio de este caso de uso se da cuando el cliente se acerca a la caja registradora con los artículos que desea llevar; observe las interacciones entre objetos en el diseño:

**

**5. Documentación de la arquitectura**

Según Clements, la documentación de la arquitectura son los documentos que se generan durante el proceso de desarrollo de la arquitectura de *software*; esta cumple varios propósitos significativos, como son:

* Informar a los distintos involucrados o *stakeholders*, el diseño y las decisiones que permitieron llegar a este.
* Facilitar la realización del análisis y la evaluación del diseño.
* Facilitar el soporte de las actividades de mantenimiento.

Para documentar un diseño de arquitectura, se requiere representar las distintas estructuras del diseño arquitectural, estas se documentan de forma separada, a través de distintas vistas. Cada vista modela una parte del diseño arquitectural desde distintas perspectivas, que pueden ser dinámicas, lógicas o físicas.

Para saber qué tantos documentos se llevará en el proyecto de *software*, se deben considerar diversos aspectos, que incluyen a los *stakeholders*, el tipo de proyecto y la experiencia del equipo de desarrollo.

Por lo general, los documentos más comunes que se generan en este proceso son:

* Requisitos de sistema.
* Requisitos de *software.*
* Requisitos de *hardware.*
* Diseños.
* Diagramas.
* Pruebas en todas sus versiones “planificadas, superadas, test, validación, etc.”.

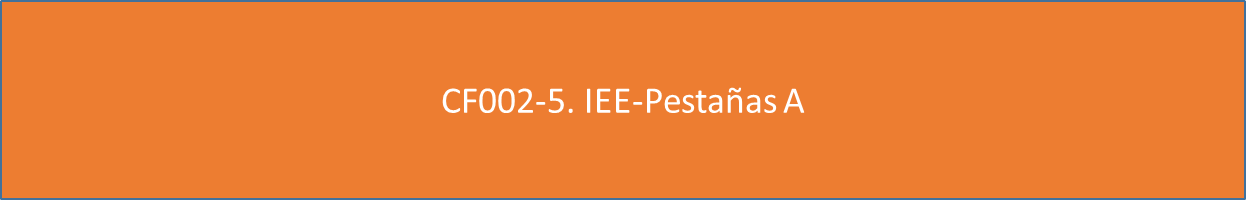
Varios estándares para la realización de dicha documentación de la arquitectura:

IEEE

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, por sus siglas en inglés) es la sociedad técnica profesional más grande del mundo, se dedica a divulgar los avances científicos en las áreas de ingeniería eléctrica, electrónica, energética, informática y afines.

Esta organización publicó el estándar ISO/IEC/IEEE 29119. Según ISO (2022), las pruebas de *software* son un estándar que agrupa procesos, documentación, vocabulario, técnicas y un modelo de evaluación de procesos para realizar las pruebas, se puede usar dentro de cualquier ciclo de vida de desarrollo de *software*.

Está conformado principalmente por la documentación de prueba y,como su nombre lo indica, se centra en la documentación de las pruebas de *software*; se divide en tres tipos de documentación:



ISO/IEC 25000

También se la conoce con el nombre *SQuaRE* (*System and Software Quality Requirements and Evaluation)* “Evaluación y requisitos de calidad del *software* y del sistema”.

El objetivo de este estándar es la creación de un marco o guía de trabajo para evaluar la calidad del producto de *software* o aplicación, esta norma es la evolución de las normas ISO/IEC 9126 e ISO/IEC 14598; se compone principalmente por: conceptos, modelos o patrones de referencia y definiciones, divididos en cinco áreas, así:

* ISO 2500n: gestión de calidad
* ISO 2501n: modelo de calidad: compuesto por fiabilidad, seguridad, mantenibilidad y usabilidad.
* ISO 2502n: medición de calidad
* ISO 2503n: requisitos de calidad
* ISO 2504n: evaluación de calidad

Dentro de este estándar, se destaca una de las normas más usadas para verificar la calidad de *software*: el modelo ISO/IEC 25010, el cual se encuentra compuesto por las ocho características de calidad que se muestran en la siguiente figura:

**Figura 5**

*Representación modelo ISO/IEC 25010*



Tenga en cuenta, a su vez, que, de sus ochos características, cada una cuenta con algunas subcaracterísticas relacionadas con su funcionamiento, como se puede ver en la siguiente tabla:

**Tabla 3**

*Subcaracterísticas relacionadas con el proceso de calidad ISO/IEC 25010*

| Característica | Subcaracterística |
| --- | --- |
| Adecuación funcional | Completitud funcional |
| Corrección funcional |
| Pertinencia funcional |
| Eficiencia de desempeño | Capacidad |
| Comportamiento temporal |
| Utilización de recursos |
| Compatibilidad | Coexistencia |
| Interoperabilidad |
| Usabilidad | Accesibilidad |
| Aprendizaje |
| Estética |
| Inteligibilidad |
| Operabilidad |
| Protección frente a errores de usuario |
| Fiabilidad | Capacidad de recuperación |
| Disponibilidad |
| Madurez |
| Tolerancia a fallos |
| Seguridad | Autenticidad |
| Confidencialidad |
| Integridad |
| No repudio |
| Responsabilidad |
| Mantenibilidad | Analizabilidad |
| Capacidad de ser modificado |
| Capacidad de ser probado |
| Modularidad |
| Reusabilidad |
| Portabilidad | Adaptabilidad |
| Capacidad de ser reemplazado |
| Facilidad de instalación |

Modelo 4+1

Este modelo fue creado por Hilippe Kruchten, ofrece una técnica para organizar la descripción de una arquitectura de *software* utilizando un conjunto de cuatro "vistas" concurrentes, que son las que se muestran en la figura:

**Figura 6**

*Modelo 4+1*



Según Kruchten (1995), además, los casos de uso o escenarios seleccionados se utilizan como la vista 'más uno' para mostrar el diseño. Como resultado, el modelo tiene 4+1 vistas, de ahí que el modelo se llame “The 4+1 Architectural View”.

Ahora, observe los tipos de vistas de forma individual:

* **Vista lógica**: se ocupa de la funcionalidad del sistema en lo que respecta a los usuarios finales. Los diagramas de clase y los diagramas de estado son ejemplos de [diagramas UML](https://medium.com/javarevisited/5-best-uml-books-and-courses-for-java-programmers-13c551a9235d) que se utilizan para representar la vista lógica.
* **Vista de desarrollo:** se enfoca en el comportamiento del tiempo de ejecución del sistema y se ocupa de los elementos dinámicos del sistema. Explica los procesos del sistema y cómo estos se comunican. [La concurrencia](https://medium.com/javarevisited/8-best-multithreading-and-concurrency-courses-for-experienced-java-developers-8acfd3b25094) , la distribución, el integrador, el rendimiento y la escalabilidad se abordan en la vista del proceso. El diagrama de secuencia, el diagrama de comunicación y el diagrama de actividad son todos diagramas UML que se pueden usar para describir una vista de proceso.
* **Vista de proces**o**:** describe un sistema desde el punto de vista de un programador y se ocupa de la administración del *software*, también se lo conoce como vista de despliegue, describe los componentes del sistema utilizando el diagrama de paquetes UML.
* **Vista física:** representa el sistema desde la perspectiva de un ingeniero de sistemas. La capa física se ocupa de la topología de los componentes de *software*, así como de las conexiones físicas entre estos, también se la conoce como vista de implementación, aquí se utiliza el diagrama de implementación de UML para representar la perspectiva física.

**6. Verificar atributos de calidad**

Es necesario tener en cuenta el concepto de calidad; según la RAE, es el grado en el que un conjunto de características propias de un objeto, llámese “servicio, proceso, producto, sistema, organización, o recurso, cumple con los requisitos, estándares o métricas definidas por una entidad de estandarización.

Para los términos de calidad de producción de *software*, se pueden verificar utilizando las siguientes normas o estándares, de las cuales ya ha comenzado a conocer algunas en el punto previo, siendo las más conocidas:

* ISO: Organización Internacional de Normalización
* OWASP: *Open Web Application Security Project* (“Seguridad del *software*”)
* IEC: Organización de normas internacionales para todas las tecnologías eléctricas, electrónicas y relacionadas
* IEEE: *Institute of Electrical and Electronic Engineers* (Instituto de Ingenieros en Electricidad y Electrónica)

Tanto los estándares como los modelos garantizan la calidad tanto del producto como del proceso, su implementación depende de las necesidades y enfoques definidos.

Algunos modelos y estándares de calidad clásicos han sido la base para los estándares de calidad más recientes y han permitido que los modelos actuales se consoliden como los más completos, para así optimizar con criterios o estándares que respaldan la calidad.

Performance y escalabilidad

De acuerdo con Lovelle (1999), cuando se habla de *performance*, se abordan los términos de rendimiento de un producto; la escalabilidad, por su parte, son las propiedades que se espera de un *software* o solución informática, lo cual indica su estabilidad para adaptarse, crecer y manejar más volumen de datos, sin perder la calidad en los procesos que este realiza, a lo que se llama rendimiento.

Se distingue dos tipos de escalabilidad, que son:

* Escalabilidad vertical: tiene que ver con la migración del sistema a un *hardware* de mayor potencia.
* Escalabilidad horizontal: consiste en optimizar el rendimiento del *software* desde un aspecto de mejora global, utilizando una red de servidores, los cuales se los conoce como clúster; en resumen, “entre más usuarios, más clúster”.

Disponibilidad y modificabilidad

La disponibilidad es la capacidad de una aplicación o solución informática de ser accesible y utilizable por los usuarios autorizados cuando estos lo necesiten. Por otra parte, la modificabilidad es la capacidad que tiene el *software* para admitir cambios que son necesarios por nuevos requerimientos o por posibles errores que se puedan presentar.



Estos son atributos de calidad que son indispensables al momento de desarrollar una solución informática, también se los conoce como requisitos no funcionales.

**7. Seguridad –OWASP**

**“***Open Web Application Security Project*” es una fundación sin ánimo de lucro que trabaja para mejorar la seguridad de las soluciones informáticas, los miembros de esta fundación trabajan de forma voluntaria y por medio de proyectos, todos los proyectos cuentan con código abierto para su libre uso.

Uno de los proyectos más importantes es el *OWASP Guide Project*, es una guía para el analista de seguridad donde se identifican los principales controles a auditar en cualquier aplicación web, estos se agrupan en las siguientes categorías:



Esta es una guía completa, la cual cuenta actualmente con un total de 87 controles diferentes de seguridad, todos relacionados con las categorías antes mencionadas y que permiten generar un estado global y actual de la seguridad de la página web.



Como dato importante, OWASP, de forma periódica, realiza un informe haciendo la recopilación de las vulnerabilidades más frecuentes dentro de las aplicaciones web; este informe es un insumo importante para comprobar si la aplicación que está en desarrollo tiene alguna vulnerabilidad mencionada en el informe; estos informes se los puede encontrar en la página oficial: <https://owasp.org>

Para aplicaciones móviles, esta fundación cuenta con [OWASP Mobile Security Testing Guide](https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide#tab=Main) MSTG, ya que una aplicación móvil posee una serie de características diferentes a una aplicación web estándar o página web; esta guía se enfoca en auditar aplicativos móviles Android e iOS, la conforman 3 apartados o documentos relacionados entre sí, que son:

* *OWASP Mobile App Security Verification Standard (MASVS):* son los requisitos de seguridad, los cuales se verifican por el auditor.
* *Mobile Security Testing Guide (MSTG):* son instrucciones para verificar cada requisito del MASVS (Guía de buenas prácticas para que el auditor testee el requisito de seguridad).
* *Mobile App Security Checklist:* es un listado de comprobación de los requisitos del MASVS.

Después de hacer la respectiva revisión de los controles, el cliente final puede obtener un estado de la seguridad de su aplicación de forma gráfica, gracias a las tablas y gráficas que se producen en el proceso de la auditoría.

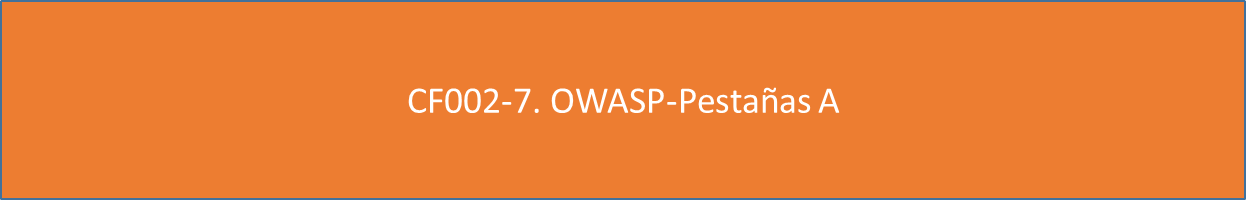
El término "aplicación móvil" se refiere a cualquier programa que se ejecuta en un dispositivo móvil.

Hay algunos tipos específicos de aplicaciones móviles de la siguiente manera:

* **Aplicación nativa***:* interactúa estrechamente con el sistema operativo del dispositivo móvil y accede a componentes del dispositivo, como una cámara, sensores, etc.
* **Aplicación web:** interactúa con los servicios web y no interactúa con los componentes del dispositivo.
* **Aplicación híbrida:**  Se ejecuta como una aplicación nativa, pero una parte de la aplicación se ejecuta en un navegador web integrado.
* **Aplicación progresiva:**Son páginas web normales, pero tienen la ventaja adicional de permitir que los desarrolladores trabajen sin conexión y obtengan acceso al *hardware* del dispositivo móvil.

Pruebas de seguridad

Si se desea tener un programa de pruebas, es necesario saber cuáles son los objetivos de estas. Estos objetivos son especificados por los requisitos de seguridad. A continuación, verá cuáles son los objetivos de las pruebas derivados de las normas y reglamentos aplicables de la guía OWASP. También, cómo los requisitos de seguridad conducen efectivamente a gestionar los riesgos de seguridad del *software*:

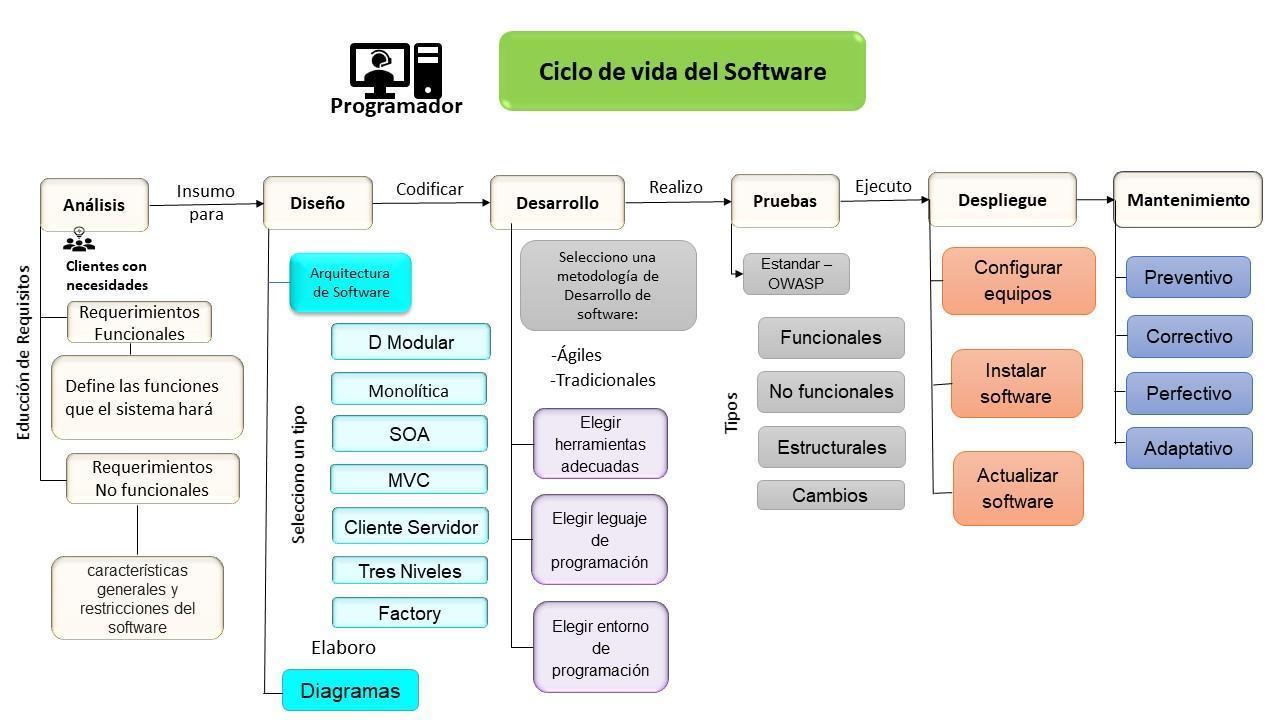


Por último, otro de los puntos clave de la guía *OWASP* es la autenticación.

Aquí, se encuentran las credenciales de usuario, como contraseña o PIN, información confidencial como SIM, *token* de *hardware*, generador de contraseña, biometría (voz, retina, huella digital, etc.). Aquí, se debe tener en cuenta lo siguiente:

* La autenticación de nombre de usuario/contraseña se debe realizar en el punto final remoto.
* Se debe aplicar una política de contraseñas.
* El segundo factor de autenticación se aplicará a las aplicaciones confidenciales.
* El usuario debe estar informado de las actividades recientes de la cuenta.

**D. SÍNTESIS**



**E. Actividades didácticas**

| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| --- | --- |
| Nombre de la Actividad | Conceptos sobre patrones de diseño |
| Objetivo de la actividad | Afianzar los conceptos de patrones de diseño. |
| Tipo de actividad sugerida | Selección múltiple. |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | Anexo: Actividad didáctica 1 |

1. **Material complementario**

| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| --- | --- | --- | --- |
| 3. Estilos arquitectónicos | Cubic Factory. (s. f.). *Editor HTML Online · Editor CSS Online · Editor JavaScript Online*. <https://www.cubicfactory.com/jseditor/> | Página web. | <https://www.cubicfactory.com/jseditor/> |
| 3. Estilos arquitectónicos | CodeSandbox: Online Code Editor and IDE for Rapid Web Development. | Página web. | <https://codesandbox.io> |
| 5. Documentación de la arquitectura | Normas ISO 25000 | Página web. | https://iso25000.com/ |
| 7. Seguridad –OWASP | OWASP Foundation. (s. f.). *Guía de pruebas OWASP*. <https://owasp.org/www-pdf-archive/Gu%C3%ADa_de_pruebas_de_OWASP_ver_3.0.pdf> | Página web | <https://owasp.org/www-pdf-archive/Gu%C3%ADa_de_pruebas_de_OWASP_ver_3.0.pdf> |
|  |  |

1. **Glosario**

| TÉRMINO | SIGNIFICADO |
| --- | --- |
| Apache Kafka | *software* manejador de eventos o también conocido como motor de eventos. |
| SDLC | Ciclo de vida del desarrollo de *software*. |
| SOA | Arquitectura Orientada a Servicios. |
| PMV | Producto mínimo viable. |
| Capa | conjunto de clases. |
| RPC | Llamada a procedimientos remotos. |
| IoT | Internet de las cosas. |

1. **Referencias bibliográficas**

Bertucelli, M. (2019). *Arquitectura de capas.* Somospnt. <https://somospnt.com/blog/118-arquitectura-de-capas>

Clements, P., Bachmann, F., Bass, L., Garlan, D. y Ivers, J. (2010). *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional.

Cueva, J. (1999). *Calidad del Software.* Universidad Nacional de la Pampa.

Departamento de Ciencia de la Computación e Inteligencia Artificial. (2019). *Arquitectura orientada a servicios (SOA)*. Universidad de Alicante. <http://www.jtech.ua.es/j2ee/2007-2008/restringido/int/sesion02-apuntes.html>

International Organization for Standardization [ISO]. (2022). *ISO/IEC/IEEE 29119-1:2022*

*Software and systems engineering — Software testing — Part 1: General concepts.* ISO Store*.* <https://www.iso.org/standard/81291.html>

Kruchten, P. (1995). Planos Arquitectónicos: El Modelo de “4+1” Vistas de la Arquitectura del Software. *IEEE Software*, *12*(6), p. 42-50. <http://materias.fi.uba.ar/7510/practica/zips/Modelo4_1Krutchen.pdf>

Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* Pearson.

Martínez, M. (2020). *¿Qué son los patrones de diseño de software?* Profile.<https://profile.es/blog/patrones-de-diseno-de-software/>

Node.js (2022). *Node.js assessment of OpenSSL 3 May 2022 security releases.* <https://nodejs.org/en/>

React.js. (2021). *React – A JavaScript library for building user interfaces*. <https://reactjs.org/>

Red Hat. (2020). *¿Qué es la arquitectura orientada a los servicios (SOA)?* <https://www.redhat.com/es/topics/cloud-native-apps/what-is-service-oriented-architecture>

Schmuller, J. (2000). *Aprendiendo UML en 24 horas.* Prentice Hall. <https://www.academia.edu/17124068/Aprendiendo_UML_en_24_Horas_Joseph_Schmuller>

Ungoti. (s. f.). *Ciclo de vida del desarrollo de software*. <https://ungoti.com/es/soluciones/desarrollo-de-software/sdlc/#:%7E:text=El%20ciclo%20de%20vida%20del,definici%C3%B3n%20de%20los%20requisitos%20hasta>

1. **Control del documento**

|  | Nombre | Cargo | Dependencia | Fecha |
| --- | --- | --- | --- | --- |
| Autor(es) | Mario Fernando Meneses Calvache | Experto Temático | Regional Cauca - Centro de teleinformática y producción industrial. | Abril 2022 |
| Zvi Daniel Grosman | Diseñador Instruccional | Regional Distrito Capital – Centro de Gestión Industrial | Abril 2022 |
| Silvia Milena Sequeda Cárdenas | Asesor Metodológico | Regional Distrito Capital - Centro de Diseño y Metrología | Abril 2022 |
| Rafael Neftalí Lizcano Reyes | Responsable del Equipo de Desarrollo Curricular | Regional Santander - Centro Industrial del Diseño y la Manufactura | Abril 2022 |
| Darío González | Corrector de Estilo | Regional Distrito Capital - Centro de Diseño y Metrología | Mayo 2022 |

1. **Control de cambios**

|  | Nombre | Cargo | Dependencia | Fecha | Razón del cambio |
| --- | --- | --- | --- | --- | --- |
| Autor(es) |  |  |  |  |  |