1. Título: Evolución de la Arquitectura de Software y metodologías de desarrollo de software

2. Modalidad: grupal (4 estudiantes).

3. Resultado de aprendizaje: el estudiante analizará y comprenderá los elementos que motivan la evolución de la arquitectura de software e identificará los puntos críticos del proceso de desarrollo, donde la arquitectura de software juega un papel relevante.

4. Recursos: el estudiante cuenta con los diferentes materiales fundamentales y de apoyo de la Unidad 1. Adicionalmente, se puede apoyar de los foros, las asesorías y los encuentros sincrónicos para resolver dudas.

5. Indicaciones

5.1 Lea los materiales de estudio de la unidad, y consultar fuentes adicionales para complementar el tema.

5.2 Realice el enunciado en cada tema de la actividad.

5.3 Adjunte documento en la plataforma con el desarrollo de la actividad.

5.4 Prepare una breve exposición de los temas analizados, la cual se socializará en la sesión virtual.

Actividad

Evolución de la Arquitectura de Software:

Explique cómo ha evolucionado la arquitectura del software: describa las arquitecturas, explique cuáles han sido las motivaciones para evolucionar la arquitectura (necesidades, problemas, características), en qué forma evolucionaron y cuáles son las nuevas características y ventajas de la nueva propuesta arquitectónica.

Identifique tipos de sistemas que más se benefician con cada tipo de arquitectura. Brinde ejemplos.

Metodologías de desarrollo de software:

Estudie, analice y compare filosofías, principios, roles, actividades, fases, etc., en estos dos marcos metodológicos para el desarrollo de software: RUP y SCRUM.

Dé respuesta a los siguientes interrogantes:

1. ¿En dónde encaja la arquitectura de software en cada proceso de desarrollo?

2. ¿Cómo se desarrolla la arquitectura en cada marco metodológico?, ¿en qué momento? y ¿qué se hace?

3. ¿Quién lo hace? ¿Existe un rol específico para esto?

4. ¿Se documenta? o ¿qué se hace?

Formule sus propias preguntas y comparta su reflexión en la socialización.

EVOLUCIÓN DE LA ARQUITECTURA DE SOFTWARE

Explique cómo ha evolucionado la arquitectura del software:

En 1968 Edsger Dijkstra, científico computacional de los países bajos mencionó que se debería realizar una estructuración correcta de los sistemas de software antes de adentrarse en la codificación de estos. Fue quien promulgó la organización estructural de los sistemas a través de la ciencia computacional teniendo en cuenta conceptos matemáticos. Aludió a la división de los problemas, a los stocks, a las capas, al algoritmo del camino más corto y a otros conceptos computacionales para realizar la abstracción de los sistemas. [1]

Una de las ideas de Dijkstra sobre las metodologías de programación fue el movimiento de la programación estructurada, ya que en esos tiempos los ingenieros tenían dificultades con la organizanización del software; se encontraba que la mayor parte del código de esa época era complicado, difícil de leer, difícil de modificar y difícil de reutilizar. En 1967 Dijkstra cambiaría eso para siempre fundando las bases de la AS al publicar un artículo sobre cómo un software se podía construir en capas. [2]

Para describir esto Dijkstra usó THE OS. Los diseñadores impusieron la restricción de que las capas superiores sólo pueden depender de las inferiores para hacer que el razonamiento sobre el sistema (utilizando métodos cuasi-formales) sea más manejable y también para facilitar la construcción y prueba del sistema de forma incremental. Las capas se implementaron en orden, la capa 0 primero, con pruebas exhaustivas de las abstracciones proporcionadas por cada capa a su vez. [3]

La Capa 0 era la capa más baja, esta se encargaba básicamente del control, direccionar procesos al procesador, y activar los interruptores necesarios para que dichos procesos se hicieran de manera correcta. (Scheduler)

La Capa 1 es la encargada de el direccionamiento de memoria para cada proceso. (Pager)

La Capa 2 se ocupaba de la comunicación entre el sistema operativo y la consola.

La Capa 3 manejaba las entradas y salidas de los diferentes periféricos o dispositivos.

La Capa 4 constaba de programas de usuario, encargándose de la compilación, ejecución e impresión de los programas de los usuarios.

La Capa 5 era el usuario.

Se puede apreciar que esta arquitectura tiende a la arquitectura de la computadora ya que el software se trabajaba a bajo nivel.

Luego en 1969 en la conferencia de ingeniería de software (NATO) P.I. Sharp, basándose en las ideas de Dijkstra dijo que no se le estaba prestando la verdadera atención a la arquitectura

de software, que muchos de los que estaban allí ni siquiera tenían una idea clara en lo que consistía, por ende sus palabras radica primordialmente en las especificaciones funcionales del software teniendo en cuenta el diseño, la forma, la creación de algo, entre otros conceptos que abrieron el camino a lo que en realidad se debería hacer.

En el mismo año el ingeniero de software Fred Brooks y el informático canadiense Ken Iverson plantearon la primera definición de la arquitectura de software, se refirieron a ésta como la estructura conceptual de un sistema en la perspectiva del programador, o sea que ya no solo se centraban en el hacer, sino en el cómo, se tenía una idea más clara en la forma como se iba a estructurar los programas.

Llegaron los años 70s y en esta década más precisamente en 1971, el ex senador estadunidense C.R. Spooner realizó un ensayo sobre esta materia, el cual tituló “Una arquitectura de software para los 70s”, pero esto no tuvo mucha repercusión entre el gremio.

En 1972 David Parnas, ingeniero canadiense de software, fue quien desarrolló el concepto de ocultación de la información en la programación modular, que es una de las bases de la POO, además propuso otros principios como las estructuras de software y las familias de programas. Estas concepciones las dejó claras en el ensayo que publicó en dicho año.

Nuevamente hace aportaciones F. Brooks en este proceso, quien en 1975 utilizó el concepto de la arquitectura para diseñar la especificación detallada y completa de la interfaz de usuario. En los años 70s fue en donde hubo el advenimiento del diseño estructurado, las investigaciones académicas detalladas sobre la rama y la separación entre la implementación y el diseño.

Ya para 1980 existió un enfoque a la programación orientada a objetos, a las simulaciones computacionales y a los lenguajes informáticos.

El primer estudio en la que aparece la expresión “Arquitectura de Software” (AS) fue en 1982, introducido por Perry y Wolf, en la que se hacía una analogía entre la arquitectura de edificios y la de software, en las cuales se basaron algunos actores para determinar principios arquitectónicos, y otros no tanto, porque no estaban de acuerdo con dicha similitud. Perry y Wolf tenían la idea de desarrollar una intuición para la AS, presentando así un modelo basado en la forma, los elementos y la razón. Los elementos se referían a los datos, conexión, y al procesamiento. La forma hacía hincapié en las restricciones de dichos elementos y la razón tenía que ver con las restricciones del sistema las cuales se derivan de los requerimientos de estos.

Un gran aporte que tuvo la arquitectura de software fue el surgimiento de los patrones de diseño de software, estos son una descripción o plantilla sobre cómo resolver un problema que se puede utilizar en muchas situaciones diferentes. Los patrones de diseño son las “mejores prácticas” (best practices) formalizadas que el ingeniero puede utilizar para resolver problemas comunes al diseñar una aplicación o un sistema. [4]

De los patrones más importantes son los que se pueden encontrar contemplados en dos textos: la banda de los cuatro (Gang of Four o GoF) publicada en 1995 y la serie de POSA dada en 1996. En estos escritos se habla sobre la expansión de la POO y de la AS.

Finalmente, en el año 2000 Roy Fielding presentó su tesis sobre el modelo REST, el cual establece definitivamente el tema de las tecnologías de Internet y los modelos orientados a servicios y recursos en el centro de las preocupaciones de la disciplina. [5]

REST ha sido ampliamente utilizado y se compone de las siguientes características:

- Protocolo cliente / servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la solicitud. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de la comunicación entre mensajes. Sin embargo, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (REST no permite algunas de estas prácticas, como la reescritura de URL).
- Un conjunto de operaciones bien definidas aplicables a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD en bases de datos (CLAB en castellano: crear, leer, actualizar, borrar) que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.
- Una sintaxis general utilizada para identificar recursos. En el sistema REST, cada recurso solo puede ser direccionado por su URL.
- La hipermedia se utiliza para la información de la aplicación y las transiciones de estado de la aplicación: la representación de este estado en un sistema REST suele ser HTML o XML. Como resultado, es posible navegar desde un recurso REST a muchos otros recursos simplemente siguiendo el enlace, sin el uso de un registro u otra infraestructura adicional.

En el 2002 la AS se centró en la informática, en la implementación y la codificación, teniendo en cuenta conceptos como: reutilización, homogeneización de la terminología, desarrolló la tipificación de los estilos arquitectónicos y elaboró lenguajes de descripción de arquitectura. También se consolidó la concepción de las vistas arquitectónicas, reconocidas en todos y cada uno de los frameworks generalizadores que se han propuesto.

En el siglo XXI, la AS aparece dominada por estrategias orientadas a líneas de productos y por establecer modalidades de análisis, diseño, verificación, refinamiento, recuperación, diseño basado en escenarios, estudios de casos y hasta justificación económica, redefiniendo todas las metodologías ligadas al ciclo de vida en términos arquitectónicos.

Motivaciones de la Evolución de la Arquitectura de Software:

La arquitectura de software nació con el fin mejorar tanto los procesos de desarrollo como el producto final. A continuación se describen algunos problemas que resuelve y ventajas que ofrece:

- Proporciona una base para analizar el comportamiento del sistema de software antes de construir el sistema. La verificación de la capacidad de los futuros sistemas de software para satisfacer las necesidades de las partes interesadas sin la necesidad de construirlos puede ahorrar muchos costos y reducir los riesgos. Se han desarrollado varias técnicas para realizar dicho análisis, como ATAM o mediante la creación de representaciones visuales de sistemas de software.

- Proporcionar una base para la reutilización de proyectos y decisiones. Toda la arquitectura de software o parte de ella, como una decisión y estrategia de arquitectura única, se puede reutilizar en múltiples sistemas. Las partes interesadas de estos sistemas necesitan calidad, atributos o funciones similares, lo que ahorra costos y reduce los errores de diseño.

- Procura de forma temprana tomar las decisiones de diseño que afectan el desarrollo, la implementación y la vida útil de mantenimiento de un sistema. Ya que las decisiones tempranas de alto impacto, tomadas de forma correcta ayudan a evitar sobrepasar los plazos y el presupuesto.

- Facilita la comunicación con los interesados, contribuyendo a un sistema que satisface mejor sus necesidades. Comunicar sistemas complejos desde el punto de vista de los interesados les ayuda a comprender las consecuencias de los requisitos establecidos y las decisiones de diseño basadas en ellos. La arquitectura brinda la capacidad de discutir las decisiones de diseño antes de que se implemente el sistema, cuando aún son relativamente fáciles de adaptar.

De lo anterior la arquitectura de software ayuda a reducir riesgos y la posibilidad de fallas aún cuando se trabaja con proyectos complejos. [1]

Tipos de arquitecturas y sus usos:

Modelo Vista Controlador

Este patrón, también conocido como patrón MVC, divide una aplicación interactiva en 3 partes (capas), como modelo: contiene la funcionalidad y los datos básicos. Vista: muestra la información al usuario (se puede definir más de una vista). Controlador: maneja la entrada del usuario.

Esto se hace para separar las representaciones internas de información de las formas en que se presenta y acepta la información del usuario. Desacopla los componentes y permite la reutilización eficiente del código.

Es usada en Arquitecturas para aplicaciones World Wide Web en los principales lenguajes de programación. Marcos web como Django y Rails. Se ha usado en aplicaciones como Twitter y Github.

Cliente Servidor

Este patrón consiste en dos partes; un servidor y múltiples clientes . El componente del servidor proporcionará servicios a múltiples componentes del cliente. Los clientes solicitan servicios del servidor y el servidor proporciona servicios relevantes a esos clientes. Además, el servidor sigue escuchando las solicitudes de los clientes.

Se usa en aplicaciones en línea como correo electrónico, uso compartido de documentos y banca. Por ejemplo Gmail y Outlook.

Patrón Maestro Esclavo (también llamado de otras formas)

Este patrón consiste en dos partes; maestro y esclavos. El componente maestro distribuye el trabajo entre componentes esclavos idénticos y calcula el resultado final de los resultados que devuelven los esclavos.

Se usa en la replicación de la base de datos, la base de datos maestra se considera como la fuente autorizada y las bases de datos esclavas se sincronizan con ella. También en periféricos conectados a un bus en un sistema informático (unidades maestra y esclava). Algunas tecnologías que lo usan son Python, MySQL, Jenkins, Adobe y Linux.

Patrón de igual a igual (Peer to Peer)

En este patrón, los componentes individuales se conocen como pares. Los pares pueden funcionar tanto como un cliente, solicitando servicios de otros pares, y como un servidor, proporcionando servicios a otros pares. Un par puede actuar como un cliente, servidor o como ambos, y puede cambiar su rol dinámicamente con el tiempo.

Se usa en redes de intercambio de archivos como Gnutella y G2, y Protocolos multimedia como P2PTV y PDTP. Este patrón es usado por PayPal y Google Play. [6]

METODOLOGÍAS DE DESARROLLO DE SOFTWARE

1. Estudie, analice y compare filosofías, principios, roles, actividades, fases, etc., en estos dos marcos metodológicos para el desarrollo de software: RUP y SCRUM.

<table>
<tr><th></th><th>RUP</th><th>SCRUM</th></tr>
<tr><td>Principios</td><td>La filosofía de RUP se basa en 6 principios claves:
-Adaptar el proceso
-Equilibrar prioridades
-Demostrar valor iterativamente
-Colaboración entre equipos
-Enfocarse en la calidad
-Elevar el nivel de abstracción [7]</td><td>SCRUM se enfoca principalmente en:
-Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.
-Se basa más en la calidad del resultado en el conocimiento de las personas en equipos auto organizados, que en la calidad de los procesos empleados.
-Solapar las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o en cascada. [8]</td></tr>
<tr><td rowspan="2">Principales características</td><td>-Desarrollo iterativo.
-Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo).
-Pretende implementar las mejores prácticas en Ingeniería de Software.
-Administración de requisitos.
-Uso de arquitectura basada en componentes.
-Control de cambios.
-Modelado visual del software.
-Verificación de la calidad del software. [7]</td><td>-Desarrollo incremental de los requisitos del proyecto en bloques temporales cortos y fijos.
-Se da prioridad a lo que tiene más valor para el cliente.
-El equipo se sincroniza diariamente y se realizan las adaptaciones necesarias.
-Tras cada iteración (un mes o menos entre cada una) se muestra al cliente el resultado real obtenido, para que este tome las decisiones necesarias en relación a lo observado.
-Se le da la autoridad necesaria al equipo para poder cumplir los requisitos.
-Fijar tiempos máximos para lograr objetivos.
-Equipos pequeños (de 3 a 9 personas cada uno). [8]</td></tr>
<tr><td colspan="2">-No es un marco de trabajo estricto, en cada caso el método puede adaptarse según el contexto.
-Cada iteración produce un producto ejecutable. [7] [8]</td></tr>
<tr><td>roles</td><td>Analistas
Desarrolladores
Probadores
Directivos
otros [9]</td><td>-Product Owner.
-Scrum Master.
-El equipo de desarrollo. [8]</td></tr>
<tr><td>fases</td><td>Fase de Inicio
Fase de elaboración
Fase de Desarrollo
Fase de Cierre [7]</td><td>Sprint
Sprint Planning
Daily Scrum
Sprint Review
Sprint Retrospective [8]</td></tr>
</table>

RUP

El Proceso Racional Unificado o RUP (por sus siglas en inglés de Rational Unified Process) es un proceso de desarrollo de software creado por la empresa Rational Software, que actualmente es propiedad de IBM. igual que el Lenguaje Unificado de Modelado (UML). RUP es la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos.

El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. [7]

Roles

Analistas: Analista del proceso de negocios, diseñador de negocios, revisor del modelo de negocios, analista de sistema, especificador de requisitos, revisor de requisitos y diseñador de la interfaz usuario.

Desarrolladores: Arquitecto de software, revisor de la arquitectura, diseñador, diseñador de cápsula, diseñador de base de datos, revisor del diseño, programador, revisor del código, integrador.

Probadores: Diseñador de prueba, probador.

Directivos: Director de control de cambio, director de configuración, director de implantación, ingeniero de proceso, director del proyecto, revisor del proyecto.

otros: Stakeholder, cualquier rol, desarrollador de cursos, artista gráfico, administrador de sistema, documentador técnico, especialista en herramientas. [9]

Fases

Fase de Inicio: Tiene como propósito definir y acordar el alcance del proyecto con los patrocinadores o involucrados del proyecto en el cual se tienen que identificar los riesgos asociados, proponer una visión general de la arquitectura de software y desarrollar el plan de las siguientes fases e iteraciones posteriores.

Fase de elaboración: Se seleccionan los casos de uso que permitan definir la arquitectura base del sistema y se puedan desarrollar en la fase, además, se realiza la especificación de los casos de uso seleccionados, un análisis del dominio del problema y se diseña la solución preliminar.

Fase de Desarrollo: Su propósito es completar la funcionalidad del sistema, para ello se deben aclarar los requisitos pendientes, administrar los cambios de acuerdo a las evaluaciones realizados por los usuarios y realizar las mejoras para el proyecto.

Fase de Cierre o transición: Se asegura que el software esté disponible para los usuarios finales, se ajustan los errores y defectos encontrados en las pruebas de aceptación, se capacita a los usuarios y provee soporte técnico necesario. [7]

SCRUM

Roles

Product Owner: Se asegura de que el equipo Scrum trabaje de forma adecuada según el punto de vista del negocio. También, ayuda al usuario a escribir las historias de usuario, las prioriza, y las coloca en el Product Backlog.

Scrum Master: Es el responsable de que se siga la metodología de SCRUM y que sea entendida por la organización. Elimina los obstáculos que impiden que se desarrolle el objetivo de cada sprint, además, asesora y da la formación necesaria al propietario del producto y al equipo de desarrolladores.

El equipo de desarrollo: Está conformado por cada uno de los profesionales que realizan la entrega del incremento de producto generado en cada sprint. Es recomendable un pequeño equipo de 3 a 9 personas con las habilidades transversales necesarias para realizar el trabajo (análisis, diseño, desarrollo, pruebas, documentación, etc). [8]

Fases

Sprint Planning: Ocurre al comienzo de un sprint, en esta reunión el equipo de scrum planifica el sprint. Uno de los objetivos de la reunión es identificar y comunicar cuánto del trabajo es probable que se realice durante el actual Sprint.

Sprint: El Sprint es el período en el cual se lleva a cabo el trabajo en sí. Generalmente se recomienda que la duración de los sprints sea constante y definida por el equipo con base en su propia experiencia. Se puede ajustar con base en el ritmo del equipo.
Al final de cada sprint, el equipo deberá presentar los avances logrados, y el resultado obtenido es un producto que, potencialmente, se puede entregar al cliente. Con cada sprint este producto puede actualizarse y es llamado incremento.

Daily Scrum: Es una reunión que se realiza cada día durante un sprint. Su objetivo es que los miembros del equipo se mantengan actualizados unos a otros sobre el trabajo de cada uno desde el último Daily, qué problemas han encontrado o creen que se encontrarán, y qué planean hacer.
La reunión tiene una duración fija de entre 5 y 15 minutos.

Sprint Review: En esta reunión se presentan los trabajos completados.

Sprint Retrospective: Después de cada sprint, se realiza una retrospectiva en la cual todos los miembros del equipo dejan sus impresiones sobre el sprint recién finalizado. El propósito de la retrospectiva es realizar una mejora continua en el equipo y el proceso. [8]

2. Dé respuesta a los siguientes interrogantes:

1. ¿En dónde encaja la arquitectura de software en cada proceso de desarrollo?

En la metodología RUP, la arquitectura de software se encuentra sobre todo en las fases de Inicio, elaboración, construcción y transición. [10] Mientras que en SCRUM, la arquitectura se desarrolla al inicio del ciclo de desarrollo o durante el Sprint, ya que es en este momento donde se analiza y diseñan las generalidades del sistema, para que el mismo cumpla con los requisitos y sea comprensible para el equipo de desarrollo desde sus diferentes roles. [11]

2. ¿Cómo se desarrolla la arquitectura en cada marco metodológico?, ¿en qué momento? y ¿qué se hace?

En la metodología RUP se propone una visión general de la arquitectura de software y se define una arquitectura base para el proyecto, los momentos en los que la arquitectura se desarrolla en el proyecto son:

Fase de inicio : Esta fase tiene como propósito establecer una visión general de la arquitectura a usar en el proyecto, además de prever las interacciones posteriores.

Fase de elaboración: En esta fase se desarrolla la base de la arquitectura, se seleccionan los casos de uso que ayudan a definir la arquitectura base del sistema, se abarcan los flujos de trabajo de requisitos, el modelo de negocio, análisis, diseño y una parte de la implementación orientada a la base de la arquitectura, además en esta fase es donde se desarrollarán dichos casos de uso. [10]

Mientras que en SCRUM, la arquitectura se desarrolla durante el sprint que es a su vez las pequeñas reuniones a lo largo del proyecto donde se discuten lo que se ha alcanzado y se pretende alcanzar, así como si se está siguiendo la arquitectura de manera correcta y todo se está acoplando de la mejor manera.

3. ¿Quién lo hace? ¿Existe un rol específico para esto?

En Scrum no existe un rol específico para ello, como sabemos en la metodología existen 3 roles principales, y dentro de los mismos no hay una persona que se encargue de ello, de por si el Arquitecto hace parte de un rol externo llamado Cuerpo de asesoramiento de SCRUM (SCRUM Guidance Body), el cual se compone por diferentes expertos que ayudarán a definir los lineamientos y parámetros que se utilizarán para evaluar el valor del negocio, y son estos los que junto al equipo de desarrollo definen en qué momentos se desarrolla la arquitectura. [12]

En RUP tampoco existe un rol específico que se encargue de esto ya que todo el equipo junto al Arquitecto de software trabajan de manera conjunta en las etapas donde es necesaria la arquitectura. [13]

4. ¿Se documenta? o ¿qué se hace?

En el caso de Scrum efectivamente cuando termina el Sprint o el resultado es un documento inicial en el que se explica la arquitectura, en la mayoría de los casos este documento se basa en los pasos que define el método ADD (Attribute Driven Design), siendo este método el que mejor ha funcionado en proyectos previamente exitosos, además el mismo permite definir la arquitectura de software mediante una descomposición que se basa en atributos de la calidad del software [14]

Este documento le permite saber al equipo si la arquitectura cumple o no con los requisitos de calidad, para ello se hace uso de un método llamado ATAM (Architecture Tradeoff Analysis Method) el cual permite conocer si una arquitectura satisface o no los objetivos de calidad y da un vistazo de cómo dichos objetivos interactúan.

Para realizar esta evaluación, se propone el método ATAM (Architecture Tradeoff Analysis Method). El ATAM revela cuán bien una arquitectura satisface los objetivos particulares de calidad y provee una aproximación de cómo los objetivos de calidad interactúan.

En el caso de RUP también se documenta aunque esta metodología se centra en la producción de arquitectura básica en las primeras interacciones, lo que permite que la misma se vaya desarrollando a medida que el proyecto avanza hasta lograr convertirse en la arquitectura final del sistema, a medida que se termina cada fase se generan artefactos, los cuales se utilizan en las siguientes fases así como la documentación que se generó en las mismas para mejorar el seguimiento del proyecto. [13]

5. Formule sus propias preguntas y comparta su reflexión en la socialización.

¿Cuál es la mejor forma de aprender a usar la arquitectura de software?
¿El arquitecto de software debe asegurarse de que todos los desarrolladores implementen correctamente las arquitecturas propuestas?
¿Qué tan sólido sería el proyecto en caso de que se quiera llevar a escalas mucho mayores sin el acompañamiento de un arquitecto de software?

La arquitectura mejora la calidad de los resultados, la reutilización y facilita el mantenimiento de las aplicaciones, de esta forma se ahorran tiempo y costos, y se pueden mantener grandes proyectos con mínimas dificultades, siempre y cuando las arquitecturas sean definidas y usadas correctamente.

referencias

[1] "Software architecture - Wikipedia", En.wikipedia.org, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Software_architecture
[2] "Edsger W. Dijkstra - Wikipedia", En.wikipedia.org, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Edsger_W._Dijkstra
[3] "THE multiprogramming system - Wikipedia", En.wikipedia.org, 2021. [Online]. Available: https://en.wikipedia.org/wiki/THE_multiprogramming_system
[4] [5]"Software design pattern - Wikipedia", En.wikipedia.org, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Software_design_pattern
[5] "Representational state transfer - Wikipedia", En.wikipedia.org, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer
[6] "Los 10 patrones comunes de arquitectura de software", Medium, 2021. [Online]. Available: https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b
[7] "Proceso Unificado de Rational - Wikipedia, la enciclopedia libre", Es.wikipedia.org, 2021. [Online]. Available: https://es.wikipedia.org/wiki/Proceso_Unificado_de_Rational
[8] "Scrum (desarrollo de software) - Wikipedia, la enciclopedia libre", Es.wikipedia.org, 2021. [Online]. Available: https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)
[9] López Trujillo, Yucely, André Ampuero, Margarita ROLES EN EL PROCESO DE DESARROLLO DE SOFTWARE PARA LAS EMPRESAS CUBANAS. Ingeniería Industrial [en linea]. 2006, XXVII(1), 31-35. Avalible: https://www.redalyc.org/pdf/3604/360433560012.pdf
[10] I. Pabón, "SMARTSOFT - Metodología de Desarrollo Tradicional RUP", Smartsoftcolombia.com, 2021. [Online]. Available: https://smartsoftcolombia.com/portal/index.php/blog/49-rup
[11] "El Papel de la Arquitectura de Software en Scrum", SG Buzz, 2021. [Online]. Available: https://sg.com.mx/revista/30/el-papel-la-arquitectura-software-scrum#:~:text=Como%20ustedes%20saben%2C%20Scrum%20es,claramente%20su%20papel%20en%20scrum
[12] F. Melo, "Metodología SCRUM y el rol del arquitecto de software - Asesoftware", Asesoftware, 2021. [Online]. Available: https://asesoftware.com/es_co/metodologia-scrum-y-el-rol-del-arquitecto-de-software/
[13] "Metodología RUP", Métodos, 2021. [Online]. Available: https://metodoss.com/metodologia-rup/
[14] O. Soto & G.H. Alférez. “An Architecture Proposal for Academic Software in Adventist Universities”. Catalyst, Asia-Pacific International University, Vol. 4, num. 1. https://www.researchgate.net/publication/272791316_An_Architecture_Proposal_for_Academic_Software_in_Adventist_Universities