Evolución de la Arquitectura de Software y metodologías de desarrollo de software

Sebastián Bedoya Restrepo
Jose Andrés López Castaño
Andrés Eduardo Rossi Herreno
Luis Felipe Salazar Rios

Arquitectura de software

Ingeniería de Sistemas

Universidad de Antioquia

Ude@

Evolución de la Arquitectura de Software:

1. Basado en los requerimientos actuales socioeconómicos y la demanda tecnológica para el crecimiento económico mundial, la arquitectura de software ha sido pieza clave en el desarrollo de código capaz de agilizar y estandarizar procesos de cómputo rutinarios, así como digitalizar actividades, gustos, comportamientos y tendencias, permitiendo la creación de ambientes personalizados de acuerdo a la necesidad y gusto del cliente.

Apoyándose de herramientas como las redes neuronales y la programación orientada a objetos, la arquitectura de software se ha convertido en el eje de la nueva revolución industrial a nivel mundial, orientando el desarrollo hacia el internet de las cosas y la interconexión mundial a través de las redes 5G.

2. Los tipos de arquitectura de software más conocidos son los siguientes:

a. Arquitectura Spaghetti: Genera líneas de código interconectadas entre sí, pero sin una secuencia lógica que permitiera una trazabilidad óptima y consistente, lo cual al final generaba múltiples problemas para la detección y corrección de errores, así como la actualización del código en sí.

b. Arquitectura por capas: Permite que el código este dividido por capas que a su vez está clasificada de acuerdo a las funcionalidades, permitiendo diversos ambientes de desarrollo, mayor seguridad y mejor versatilidad a la hora de editar código, pues solo se accede a la capa y/o funcionalidad requerida; esto a su vez permite que el usuario final no tenga acceso al código fuente, evitando fraudes, hackeos modificaciones.

c. Arquitectura hexagonal: dada a conocer por Alistair Cockburn — y también conocida como arquitectura de puertos y adaptadores — , tiene como principal motivación separar nuestra aplicación en distintas capas o regiones con su propia responsabilidad. De esta manera consigue desacoplar capas de nuestra aplicación permitiendo que evolucionen de manera aislada. Además, tener el sistema separado por responsabilidades nos facilitará la reutilización. [1]

d. Domain Driven Design: es un enfoque para el desarrollo de software con necesidades complejas mediante una profunda conexión entre la implementación y los conceptos del modelo y núcleo del negocio.

El DDD no es una tecnología ni una metodología, este provee una estructura de prácticas y terminologías para tomar decisiones de diseño que enfoquen y aceleren el manejo de dominios complejos en los proyectos de software.[2] El término fue

acuñado por Eric Evans en su libro "Domain-Driven Design - Tackling Complexity in the Heart of Software".

La evolución de arquitecturas de software es consecuencia de cambios como la redefinición de requerimientos, o mejoras en la infraestructura/tecnología del sistema. Es necesario que la introducción de cambios arquitectónicos sea realizada de manera sistemática, a fin de evitar la erosión en el diseño arquitectónico y la pérdida de información vital para la comprensión del diseño obtenido. Los cambios aplicados y las decisiones tomadas deben ser documentados adecuadamente, para que se puedan recuperar posteriormente las soluciones aplicadas y conocer su impacto en la arquitectura.

Las exigencias normativas implantadas desde las administraciones públicas han ido unidas al desarrollo de programas que cada vez ofrecen más y nuevas posibilidades a los técnicos. Diseños y cálculos que hace años eran complejos e incluso imposibles, hoy en día son viables gracias a las mejoras introducidas por las empresas desarrolladoras de software.

Esta evolución ha facilitado y ampliado las opciones de los proyectistas, siendo posible la realización de diseños y cálculos de una gran complejidad en menos tiempo, permitiendo que el proyectista dedique más tiempo a la creatividad y pueda diseñar trabajos más innovadores. Gracias a la utilización e implementación de nuevas tecnologías, los softwares son capaces de hacer, cada vez, cálculos y modelados más precisos.

Ejemplos de la evolución de arquitectura de software:

- Robots especializados capaces de tomar decisiones basados en el análisis del ambiente, sin necesidad de seguir una línea de código rígida.
- Buscadores inteligentes que muestran resultados basados en los intereses y visitas frecuentes del usuario final.
- Descentralización de las aplicaciones de una plataforma específica, incrementando la independencia y disminuyendo la vulnerabilidad a ataques masivos.

Sistemas que más se benefician con la arquitectura de software:

- Programación de robots para desarrollo de tareas industriales que requieran secuencias lógicas de ambientes variables.
- Desarrollo de aplicaciones web basadas en los gustos e intereses del usuario final, generando experiencias personalizadas
- Aplicaciones bancarias y de seguridad de la información.

2) Metodologías de desarrollo de software:

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

En SCRUM: la arquitectura de software en scrum se encarga de plantear las bases del proyector, proveer soluciones, mejorar la comunicación dentro del equipo a través de modelos y técnicas, definen la interacción de todos los componentes del proyecto, así como otras tareas; si bien tienen tareas específicas en los proyectos también están inmersos en las demás fases de los mismos.

En RUP: aquí, la arquitectura se basa en tener una visión clara para todos los participantes del proyecto, entregando funcionalidades basadas en los casos de uso e implementando el proyecto en relación a las fases de la metodología

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

En SCRUM: Al iniciar el ciclo de desarrollo el principal objetivo de la arquitectura es analizar y diseñar las bases y generalidades del proyecto, que satisfagan los requisitos y sean entendibles por los demás miembros del equipo durante todo el proceso de desarrollo, es clave reutilizar artefactos y componentes de software creados anteriormente para satisfacer las necesidades de desarrollos o productos específicos.
En el Sprint 0 de forma iterativa se inicia la construcción de la arquitectura del proyecto después de un análisis preliminar de los requisitos y de un estudio de factibilidad del proyecto, El resultado es un documento inicial de la arquitectura que se debe evaluar con el fin de saber si la arquitectura cumple con los requisitos de calidad y satisface los objetivos del proyecto.
“Si en la evaluación de la arquitectura se encuentra que se deben realizar cambios a la misma, entonces ésta se debe refinar hasta llegar a tener como resultado del Sprint 0 el documento pulido de la arquitectura junto con el sistema esqueleto. Finalmente, la arquitectura creada en el Sprint 0 beneficiará el desarrollo en los siguientes sprints.”[3]

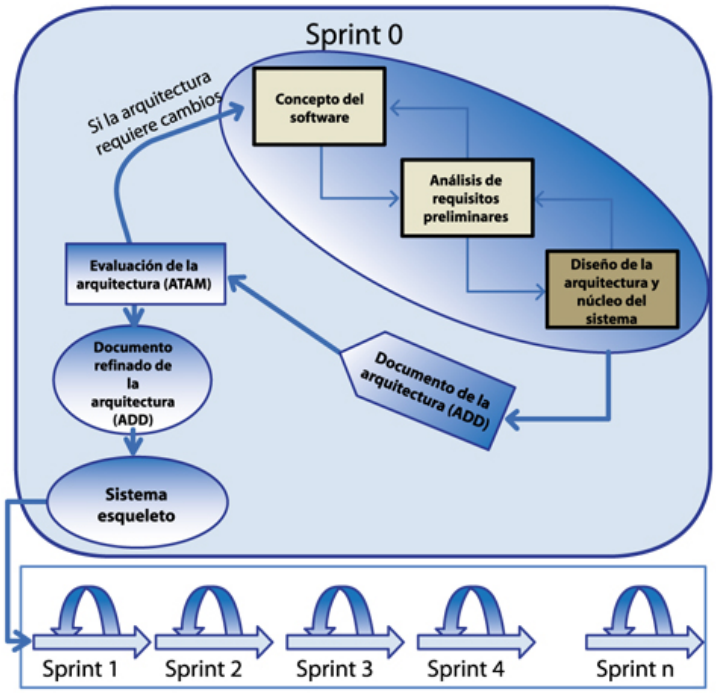

Figura 1. Propuesta de gestión de la arquitectura de software en Scrum.

En RUP: para hablar de cómo se desarrolla la arquitectura en esta metodología es necesario comprenderla.
En la metodología RUP se habla de fases y la primera de ellas es la de incepción, donde se establece el sistema y se delimita el proyecto, se identifican los actores del sistema y se habla en alto nivel. En la fase de incepción se establecen la evaluación de riesgos y estimación de recursos necesarios, y un plan de fase que muestre las fechas de las etapas principales; además, los entregables para esta fase son principalmente: Requerimientos generales, características del proyecto y restricciones, modelos de casos de usos iniciales, glosario, contexto del negocio, proyecciones financieras y uno o varios prototipos.

La segunda fase es la de elaboración, donde se desarrolla el plan del proyecto y se decide sobre la arquitectura del proyecto teniendo en cuenta los requisitos funcionales y no funcionales, además, se vigila que los costos del proyecto estén yendo en dirección adecuada. Aquí, los casos de uso deberán estar modelados hasta en un 80%, identificando a todos los actores que tendrá el sistema; además, el manual de usuario preliminar. Para asegurarse que el desarrollo del proyecto va en buen camino se responden preguntas como: ¿la visión del producto es estable?, ¿la arquitectura es estable?, ¿los gastos actuales y los gastos planeados son aceptables?, ¿los elementos de mayor riesgo han sido direccionados o resueltos?, entre otras.

La tercera fase de la metodología es la de construcción, aquí, los componentes y características del proyecto faltantes se desarrollan y se integran al producto y estas

se prueban; los avances pueden ser paralelos o uno a la vez y cuando se llega al momento de finalización de la construcción se debe evaluar si el software y los usuarios están listos para operar y por lo generar se genera una versión beta y luego se versiona el producto. En este punto, se entrega el producto, manuales de usuario y se describe la versión actual del software. Para asegurarse que se han cumplido los requisitos, se deben responder preguntas como ¿la versión del producto es lo suficientemente estable para desplegarse entre los usuarios?, ¿Están los usuarios listos para realizar la transición hacia el producto?, etc.

Por último, se encuentra la fase de transición, donde justamente se pretende hacer la transición del producto entregado a los usuarios finales, aquí se realiza el mantenimiento de la aplicación, solución de errores no encontrados, versionamiento de la aplicación con la solución de estos errores, se evalúan las expectativas del usuario vs características entregadas, desarrollo de nuevas características, etc. En este punto se tiene muy en cuenta la opinión de los usuarios. Para la terminación del proyecto se preguntan si el usuario está satisfecho o si los costos del proyecto fueron aceptables, etc.

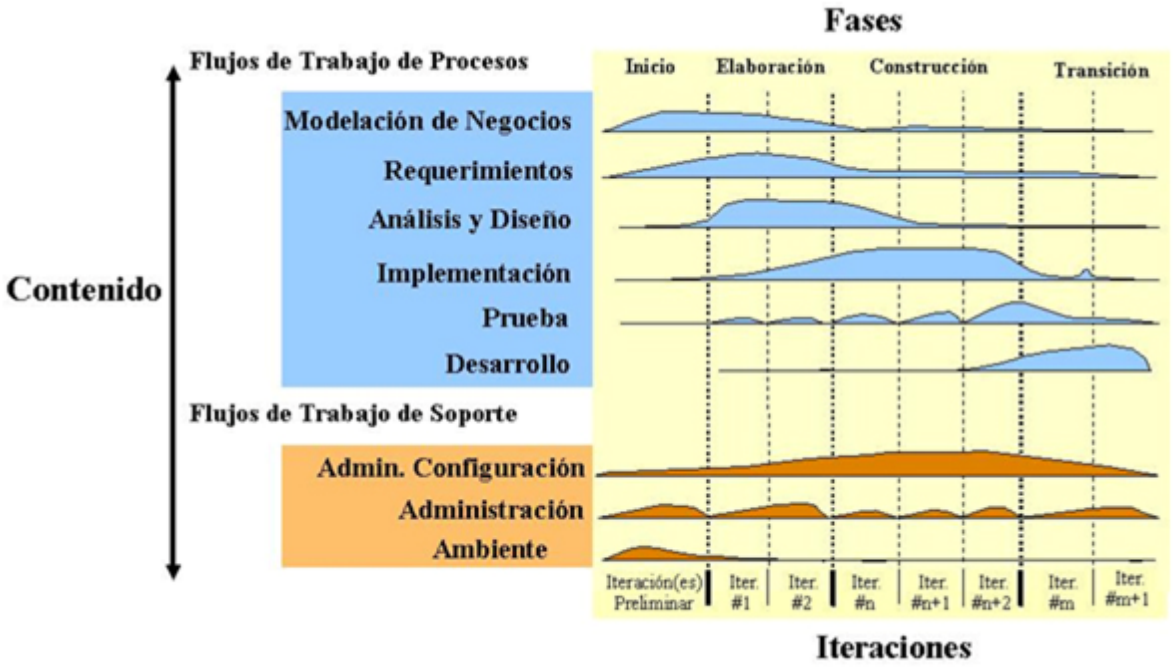

Figura 2. Fases del proyecto en la metodología RUP

Respondiendo a la pregunta anterior, la arquitectura en la metodología RUP se desarrolla por fases (anteriormente descritas), aquí, el equipo de trabajo tiene funciones específicas y a comparación de scrum, esta metodología tiene más roles y no hace tantas ceremonias, la ventaja de RUP es que trata de adelantarse a futuros requerimientos teniendo como base los requisitos funcionales y no funcionales, eliminando elementos que no son imprescindibles.

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

En SCRUM: En el sprint 0 en un trabajo en conjunto con los demás miembros del equipo; El arquitecto tiene la función de la creación y mantenimiento de la arquitectura del proyecto.
A partir del Sprint 1 el rol del arquitecto de software cambia un poco, allí se encarga de que no se pierda el enfoque y de cada reto arquitectónico que se desprenda en cada sprint, las entregas en scrum son construcciones cortas pero que deben estar guiadas por las estrategias de arquitectura.
Se encarga de por ejemplo de:
-priorizar requisitos que deban ser implementados en cada sprint.
-Orienta sobre el enfoque arquitectónico del proyecto cuando es necesario.
-Coordina a los miembros del equipo para conseguir que se adapten a la arquitectura prevista.
-ayudar a preparar el Product Backlog.
Entre otras funciones.

En RUP: Al inicio se habla con el cliente para para empezar con la estructuración del proyecto, presupuestos, funciones, roles, etc, existe un rol específico llamado “Arquitecto”, y es quien se encarga de soportar las decisiones técnicas en cuanto a la estructura del proyecto, es decir, Esto habitualmente incluye la identificación y la documentación de los aspectos arquitectónicamente significativos del sistema, que incluye las "vistas" de requisitos, diseño, implementación y despliegue del sistema.

2.4. ¿Se documenta? o ¿qué se hace?

En SCRUM: La documentación relativa a la arquitectura, desarrollo, mantenimiento, operación y despliegue del sistema desarrollado en scrum es la siguiente:

Product Backlog: Es el documento inicial, se almacena toda la información que es necesaria para el desarrollo del proyecto En este documento se explica todo lo que será desarrollado en su totalidad, describiendo todos los requerimientos, prioridades, estimaciones de inversión, esfuerzo esperado por parte del SCRUM team, período de tiempo y valor del proyecto.

Sprint Backlog: Allí se describen los requisitos y cómo cumplirlos durante el sprint, es decir, se exponen las asignaciones según las prioridades y, sobre todo, con un número de horas de trabajo por cada tarea.

Burndown chart: Permite medir el avance del proyecto de una manera gráfica. Es el gráfico en el cual se muestran la cantidad de requisitos pendientes del sprint, cómo están conectados y lo que se va logrando

Definition of done: contiene los criterios que determinan cuándo una tarea está completada para poder emprender otra.

Definition of ready: Define cuándo una tarea está lista para ser presentada y que el resto del SCRUM team pueda entenderla y así, pueda ser incluirla en una planificación de sprint.

En RUP: Se hace crean los casos de uso, el arquitecto del proyecto entonces, a partir de ahí empieza con la creación de diagramas en UML de la parte lógica y estructural del proyecto; también, se acostumbra realizar documentación sobre la arquitectura del software con los siguientes elementos:

- **Introducción**

[La introducción del Documento de Arquitectura de Software proporciona una descripción general de todo el Documento de Arquitectura de Software. Incluye el propósito, el alcance, las definiciones, las siglas, las abreviaturas, las referencias y la descripción general del Documento de arquitectura de software.]

- **Representación arquitectónica**

[Esta sección describe qué arquitectura de software es para el sistema actual y cómo se representa. De las vistas de caso de uso, lógica, proceso, implementación e implementación, enumera las vistas que son necesarias y, para cada vista, explica qué tipos de elementos de modelo contiene].

- **Metas y limitaciones arquitectónicas**

[Esta sección describe los requisitos y objetivos de software que tienen un impacto significativo en la arquitectura; por ejemplo, seguridad, protección, privacidad, uso de un producto estándar, portabilidad, distribución y reutilización. También captura las restricciones especiales que pueden aplicarse: estrategia de diseño e implementación, herramientas de desarrollo, estructura del equipo, programación, código heredado, etc.]

- **Vista de casos de uso**

[Esta sección enumera los casos de uso o escenarios del modelo de caso de uso si presentan alguna funcionalidad central significativa del sistema final, o si tienen una gran cobertura arquitectónica — ejercitan muchos elementos arquitectónicos o si enfatizan o ilustran un específico, delicado punto de la arquitectura.]

- **Realizaciones de casos de uso**

[Esta sección ilustra cómo funciona realmente el software dando algunas realizaciones de casos de uso (o escenarios) seleccionados y explica cómo los diversos elementos del modelo de diseño contribuyen a su funcionalidad.]

- **Vista lógica**

[Esta sección describe las partes arquitectónicamente significativas del modelo de diseño, como su descomposición en subsistemas y paquetes. Y para cada paquete significativo, su descomposición en clases y utilidades de clase. Debe introducir

clases de importancia arquitectónica y describir sus responsabilidades, así como algunas relaciones, operaciones y atributos muy importantes.]

- **Resumen**

[Esta subsección describe la descomposición general del modelo de diseño en términos de su jerarquía de paquetes y capas].

- **Paquetes de diseño arquitectónicamente significativos**

[Para cada paquete significativo, incluya una subsección con su nombre, su breve descripción y un diagrama con todas las clases y paquetes importantes contenidos en el paquete.
Para cada clase significativa en el paquete, incluya su nombre, una breve descripción y, opcionalmente, una descripción de algunas de sus principales responsabilidades, operaciones y atributos.]

- **Vista de proceso**

[Esta sección describe la descomposición del sistema en procesos ligeros (subprocesos únicos de control) y procesos pesados (agrupaciones de procesos ligeros). Organiza la sección por grupos de procesos que se comunican o interactúan. Describir los principales modos de comunicación entre procesos, como el paso de mensajes, las interrupciones y las reuniones.]

- **Vista de implementación**

[Esta sección describe una o más configuraciones de red física (hardware) en las que se implementa y ejecuta el software. Es una vista del modelo de implementación. Como mínimo para cada configuración debe indicar los nodos físicos (computadoras, CPUs) que ejecutan el software y sus interconexiones (bus, LAN, punto a punto, etc.). Incluir también un mapeo de los procesos del Proceso. Ver en los nodos físicos.]

- **Vista de implementación**

[Esta sección describe la estructura general del modelo de implementación, la descomposición del software en capas y subsistemas en el modelo de implementación y cualquier componente arquitectónicamente significativo].

- **Resumen**

[Esta subsección nombra y define las distintas capas y su contenido, las reglas que gobiernan la inclusión en una capa dada y los límites entre capas. Incluya un diagrama de componentes que muestre las relaciones entre capas.]

- **Capas**

[Para cada capa, incluye una subsección con su nombre, una enumeración de los subsistemas ubicados en la capa y un diagrama de componentes].

- **Vista de datos (opcional)**

[Una descripción de la perspectiva de almacenamiento de datos persistentes del sistema. Esta sección es opcional si hay pocos o ningún dato persistente, o la traducción entre el modelo de diseño y el modelo de datos es trivial.]

- **Tamaño y rendimiento**

[Una descripción de las principales características de dimensionamiento del software que impactan en la arquitectura, así como las limitaciones de rendimiento objetivo].

- **Calidad**

[Una descripción de cómo la arquitectura del software contribuye a todas las capacidades (distintas de la funcionalidad) del sistema: extensibilidad, confiabilidad, portabilidad, etc. Si estas características tienen un significado especial, como implicaciones de seguridad, protección o privacidad, deben estar claramente delineadas.]

A partir de este documento y gráficos en lenguaje UML es que el grupo de trabajo empieza a trabajar en la implementación del software

Bibliografía:

[1] Salguero, Edu (22 de mayo de 2020). «Arquitectura Hexagonal». Medium (en inglés). Consultado el 18 de junio de 2020.

[2] Evans, E., Domain-Driven Design - Tackling Complexity in the Heart of Software, 2004, Addison-Wesley.

[3] O. Soto & G.H. Alférez. “Scrum, ¿Un Paradigma de Administración de Proyectos que Cumple lo que Promete?” Software Guru, Agosto 2009. http://bit.ly/sg30r3

[4] C. Alfred. “Scrum and Architecture Partners or Adversaries”. http://bit.ly/sg30r4

[5] L. Bass, P. Clements, & R. Kasman. Software Architecture in Practice. Addison Wesley, 2da ed., 2003.

[6] O. Soto & G.H. Alférez. “An Architecture Proposal for Academic Software in Adventist Universities”. Catalyst, Asia-Pacific International University, Vol. 4, num. 1. http://bit.ly/sg30r5

Otras fuentes:

[7]https://core.ac.uk/download/pdf/95874162.pdf

[8]https://xn--zoraidaceballosdemario-4ec.info/scrum/los-5-documentos-del-scrum/

[9]https://metodologiascrum.readthedocs.io/en/latest/

[10]https://cgrw01.cgr.go.cr/rup/RUP.es/SmallProjects/core.informal_resources/guidances/templates/software_architecture_document_informal_146E1F0.html

[11]https://cgrw01.cgr.go.cr/rup/RUP.es/SmallProjects/core.base_rup/guidances/concepts/software_architecture_4269A354.html

[12]http://ima.udg.edu/~sellares/EINF-ES2/Present1011/MetodoPesadesRUP.pdf

[13]http://repositorio.puce.edu.ec/bitstream/handle/22000/11264/Documento%20Disertaci%C3%B3n%20Wendy%20Jaramillo.pdf?sequence=1

[14]https://www.usmp.edu.pe/publicaciones/boletin/fia/info49/articulos/RUP%20vs.%20XP.pdf

[15]https://core.ac.uk/download/pdf/76486415.pdf