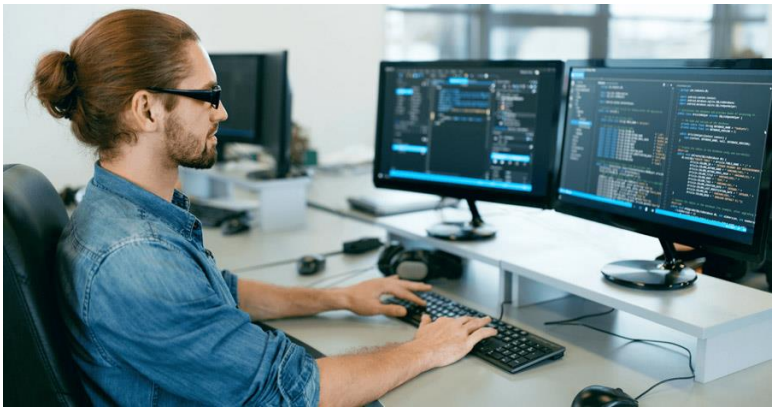
10 puntos para convertirte en un mejor arquitecto de software

Tomado de: https://www.abiztar.com.mx/articulos/10-puntos-para-convertirte-en-mejor-arquitecto-de-software.html



¿Qué hace un buen arquitecto de software? me preguntaba, y ¿Cómo puedo mejorar para convertirme en un mejor arquitecto de software? Leí artículos y libros y, por supuesto, hablé con mis compañeros.

Hoy, quiero compartir con ustedes una visión general de mis ideas, qué habilidades creo que son las más importantes y cómo mejorarlas para convertirme en un (mejor) arquitecto de software.

¿Qué es un Arquitecto de Software?

Antes de entrar en detalles, echemos un vistazo a dos definiciones.

Un arquitecto de software es un experto en software que toma decisiones de diseño de alto nivel y dicta estándares técnicos, incluyendo estándares de codificación de software, herramientas y plataformas. El experto principal es el arquitecto jefe. (Fuente: Wikipedia: Software Architect)

La arquitectura de software es la organización fundamental de un sistema, representada por sus componentes, sus relaciones entre sí y con el entorno, y los principios que determinan el diseño y la evolución del sistema. (Fuente: Handbook of Software Architecture)

Niveles de Arquitectura

La arquitectura se puede hacer en varios "niveles" de abstracción. El nivel influye en la importancia de las habilidades necesarias. Como hay muchas categorizaciones posibles, mi segmentación favorita incluye estos 3 niveles:

Nivel de aplicación: El nivel más bajo de arquitectura. Concéntrese en una sola aplicación. Diseño muy detallado y de bajo nivel. Comunicación normalmente dentro de un equipo de desarrollo.

Nivel de solución: El nivel medio de la arquitectura. Enfocarse en una o más aplicaciones que satisfacen una necesidad de negocio (solución de negocio). Algún diseño alto, pero principalmente de bajo nivel. Comunicación entre múltiples equipos de desarrollo.

Nivel Empresarial: El más alto nivel de arquitectura. Enfoque en múltiples soluciones. Diseño abstracto de alto nivel, que necesita ser detallado por arquitectos de soluciones o aplicaciones. Comunicación en toda la organización.

A veces los arquitectos también son vistos como "pegamento" entre diferentes partes interesadas. Aquí tenemos tres ejemplos:

- **Horizontal:** Puente de comunicación entre el negocio y los desarrolladores o los diferentes equipos de desarrollo.
- **Vertical:** Puente de comunicación entre desarrolladores y gerentes.
- **Tecnología:** Integrar diferentes tecnologías o aplicaciones entre sí.

Actividades típicas de los arquitectos de software

Para entender las habilidades necesarias que un arquitecto necesita, primero necesitamos entender las actividades típicas. La siguiente lista (no final) contiene desde mi punto de vista las actividades más importantes:

- Definir y decidir la tecnología y la plataforma de desarrollo
- Definir estándares de desarrollo, por ejemplo, estándares de codificación, herramientas, procesos de revisión, enfoque de pruebas, etc.
- Apoyo en la identificación y comprensión de los requisitos del negocio
- Diseñar sistemas y tomar decisiones en función de las necesidades
- Documentar y comunicar definiciones, diseño y decisiones arquitectónicas
- Comprobar y revisar la arquitectura y el código, por ejemplo, comprobar si los patrones definidos y los estándares de codificación se implementan correctamente.
- Colaborar con otros arquitectos y partes interesadas

- Asesorar y consultar a los desarrolladores
- Detallar y refinar el diseño de nivel superior a un diseño de nivel inferior.

Nota: La arquitectura es una actividad continua, especialmente cuando se aplica en el desarrollo ágil de software. Por lo tanto, estas actividades se realizan una y otra vez.

Habilidades importantes de los arquitectos de software

Para apoyar las actividades presentadas se requieren habilidades específicas. De mi experiencia, leyendo libros y discusiones pude reducir esto a las 10 habilidades que todo arquitecto de software debería tener:

Diseñar, Decidir, Simplificar, Codificar, Documentar, Comunicar, Estimar, Balancear, Consultar, Mercado.

Vayamos uno por uno. Para cada habilidad he presentado algunas acciones o ideas a seguir para mejorar en esa área.

(1) Diseñar

¿Qué es lo que hace un buen diseño? Esta es probablemente la pregunta más importante y desafiante. Haré una distinción entre la teoría y la práctica. Para mi experiencia, tener una mezcla de ambos es muy valioso. Empecemos con la teoría:

Conozca los patrones básicos de diseño: Los patrones son una de las herramientas más importantes que un arquitecto necesita para desarrollar sistemas mantenibles. Con los patrones puede reutilizar el diseño para resolver problemas comunes con soluciones probadas. El libro "Design Patterns: Elements of Reusable Object-Oriented Software" escrito por el "Gang of Four" (GoF) es una lectura obligada para todos los que están en el desarrollo de software. Aunque el patrón se publicó hace más de 20 años, sigue siendo la base de la arquitectura de software moderna. Por ejemplo, el patrón Modelo-Vista-Controlador (MVC) fue descrito en este libro, el cual se aplica en muchas áreas o es la base para un patrón más nuevo, por ejemplo, MVVM.

Profundiza en los patrones y anti-patrones: Si ya conoce todos los patrones básicos de GdF, amplíe sus conocimientos con más patrones de diseño de software. O investigue más a fondo en su área de interés. Uno de mis favoritos para la integración de aplicaciones es el libro "Enterprise Integration Patterns" escrito por Gregor Hohpe. Este libro es aplicable en varias áreas siempre que dos aplicaciones necesiten intercambiar datos, ya sea un intercambio de

archivos de la vieja escuela de algunos sistemas heredados o arquitecturas modernas de microservicios.

Conocer las medidas de calidad: Definir la arquitectura no es un fin. Tiene razones por las que se definen, aplican y controlan las directrices y normas de codificación. Esto se hace debido a los requisitos de calidad y no funcionales. Usted desea tener un sistema que sea mantenible, confiable, adaptable, seguro, comprobable, escalable, utilizable, etc. Y una pieza para lograr todos estos atributos de calidad es aplicar un buen trabajo de arquitectura. Puedes empezar a aprender más sobre las medidas de calidad en wikipedia.

La teoría es importante. La práctica es igualmente o incluso más importante si usted no quiere convertirse en un Arquitecto de la Torre de Marfil.

Pruebe y entienda los diferentes niveles de tecnología: Creo que esta es la actividad más importante si quieres ser un mejor arquitecto. Pruebe (nuevas) pilas de tecnología y aprenda sus altibajos. Diferentes o nuevas tecnologías vienen con diferentes aspectos de diseño y patrones. Lo más probable es que usted no aprenda nada simplemente hojeando diapositivas abstractas, sino probándolas usted mismo y sintiendo el dolor o el alivio. Un arquitecto no sólo debe tener un amplio conocimiento, sino que también debe tener un conocimiento profundo en algunas áreas. No es importante dominar todas las pilas de tecnología, sino tener un sólido conocimiento de las más importantes en su área. Pruebe también tecnología que no esté en su área, por ejemplo, si está muy familiarizado con SAP R/3, también debería probar JavaScript y viceversa. Aún así, ambas partes se sorprenderán de los últimos avances en SAP S/4 Hana. Por ejemplo, puedes probarlo por ti mismo y tomar un curso en openSAP de forma gratuita. Sea curioso y pruebe cosas nuevas. También pruebe cosas que no le gustaban hace algunos años.

Analizar y comprender los patrones aplicados: echa un vistazo a cualquier marco actual, por ejemplo, Angular. Puedes estudiar muchos patrones en la práctica, por ejemplo, Observables. Trata de entender cómo se aplica en el marco, por qué se hizo. Y si eres realmente dedicado, echa un vistazo más profundo al código y entiende cómo se implementó.

(2) Decidir

Un arquitecto debe ser capaz de tomar decisiones y guiar proyectos o a toda la organización en la dirección correcta.

Conoce lo que es importante: No pierdas tiempo con decisiones o actividades sin importancia. Aprende lo que es importante. Hasta donde yo sé, no hay ningún libro que tenga esta

información (si conoces alguno, por favor házmelo saber). Mis favoritos personales son estas 2 características que suelo tener en cuenta cuando evalúo si algo es importante o no:

- **(1) Integridad Conceptual:** Si decides hacerlo de una manera, ciñete a ella, aunque a veces sea mejor hacerlo de otra manera. Por lo general, esto conduce a un concepto general más sencillo, facilita la comprensión y el mantenimiento.
- **(2) Uniformidad:** Si, por ejemplo, se definen y aplican convenciones de nomenclatura, no se trata de utilizar mayúsculas o minúsculas (¡en realidad!), sino de que se apliquen en todas partes de la misma manera.

Priorizar: Algunas decisiones son muy críticas. Si no se toman con suficiente antelación, se acumulan soluciones que a menudo es poco probable que se eliminen más tarde y que son una pesadilla para el mantenimiento, o peor aún, los desarrolladores simplemente dejan de trabajar hasta que se toma una decisión. En tales situaciones, a veces es incluso mejor tomar una decisión "mala" en lugar de no tener ninguna decisión. Pero antes de llegar a esta situación, considera priorizar las decisiones futuras. Hay diferentes maneras de hacerlo. Sugiero que eches un vistazo al modelo Weighted Shortest Job First (WSJF) que se utiliza ampliamente en el desarrollo de software ágil. Especialmente las medidas de criticidad de tiempo y reducción de riesgos son críticas para estimar la prioridad de las decisiones de arquitectura.

Conoce tu competencia: No decidas cosas que no son de tu competencia. Esto es crítico ya que puede arruinar tu posición como arquitecto de manera significativa si no se tienen en cuenta. Para evitar esto, aclara con sus compañeros qué responsabilidades tienes y qué es parte de tu papel. Si hay más de un arquitecto, entonces debes respetar el nivel de arquitectura en el que te encuentras actualmente desplegado. Como arquitecto de nivel inferior, es mejor que hagas sugerencias para una arquitectura de nivel superior en lugar de tomar decisiones. Además, recomiendo verificar las decisiones críticas siempre con un compañero.

Evaluar múltiples opciones: Siempre establece más de una opción si se trata de tomar decisiones. En la mayoría de los casos en los que estuve involucrado, había más de una opción posible (buena). Ir con una sola opción es malo en dos aspectos: (1) Parece que no hiciste tu trabajo correctamente y (2) impide tomar las decisiones correctas. Mediante la definición de medidas, las opciones pueden compararse basándose en hechos en lugar de corazonadas, por ejemplo, el coste de la licencia o la madurez. Esto generalmente conduce a decisiones mejores y más sostenibles. Además, facilita la venta de la decisión a las diferentes partes interesadas. Si no tienes opciones evaluadas adecuadamente, puedes perder argumentos cuando se trata de discusiones.

(3) Simplificar

Ten en cuenta el principio de resolución de problemas de La Navaja de Occam (Occam's Razor) que dice que prefiere la simplicidad. Interpreto el principio de la siguiente manera: Si tienes demasiadas suposiciones sobre el problema para resolverlo, es probable que su solución sea errónea o lleve a una solución compleja innecesaria. Los supuestos deben reducirse (simplificarse) para llegar a una buena solución.

Agitar la solución: Para simplificar las soluciones, a menudo resulta útil "sacudir" la solución y mirarlas desde diferentes posiciones. Trata de dar forma a la solución pensando de arriba hacia abajo y de nuevo de abajo hacia arriba. Si tiene un flujo de datos o proceso, primero piensa de izquierda a derecha y de derecha a izquierda. Haz preguntas como: "¿Qué pasa con tu solución

en un mundo perfecto?" O: "¿Qué haría la compañía / persona X?" (Donde X probablemente no es tu competidor, sino una de las compañías de GAFA.) Ambas preguntas le obligan a reducir las suposiciones como lo sugiere la Navaja de Occam.

Da un paso atrás: Después de intensas y largas discusiones, a menudo el resultado son garabatos muy complejos. Nunca deberías ver esto como el resultado final. Da un paso atrás: Echa un vistazo de nuevo a la imagen completa (nivel abstracto). ¿Todavía tiene sentido? Luego, repasarlo en el nivel abstracto de nuevo y refactorizarlo. A veces ayuda detener una discusión y continuar al día siguiente. Al menos mi cerebro necesita algo de tiempo para procesar y encontrar soluciones mejores, más elegantes y sencillas.

Dividir y Conquistar: Simplifica el problema dividiéndolo en pedazos más pequeños. Luego, resuélvelas de forma independiente. A continuación, valida si las piezas pequeñas coinciden. Da un paso atrás para echar un vistazo al panorama general para esto.

La refactorización no es mala: Está totalmente bien empezar con una solución más compleja si no se puede encontrar una idea mejor. Si la solución está causando problemas, puedes repensar la solución y aplicar tu aprendizaje. La refactorización no es mala. Pero antes de empezar a refactorizar, ten en cuenta que debe tener (1) suficientes pruebas automatizadas que puedan asegurar el funcionamiento adecuado del sistema y (2) la aceptación de sus partes interesadas. Para saber más sobre refactorización te sugiero que leas "Refactorización. Improving the Design of Existing Code" por Martin Fowler.

(4) Programar

Incluso como *Enterprise Architect*, el nivel más abstracto de la arquitectura, debería saber lo que los desarrolladores están haciendo a diario. Y si no entiendes cómo se hace esto, puedes enfrentarte a dos problemas principales:

- (1) Los desarrolladores no aceptarán tus dichos y
- (2) Tú no entiendes los retos y necesidades de los desarrolladores.

Tener un proyecto paralelo: El propósito de esto es probar nuevas tecnologías y herramientas para descubrir cómo se hace el desarrollo hoy y en el futuro. La experiencia es la combinación de observaciones, emociones e hipótesis ("Experience and Knowledge Management in Software Engineering" de Kurt Schneider). Leer un tutorial o algunos pros y contras es bueno. Pero esto es sólo "conocimiento del libro". Sólo si pruebas las cosas por ti mismo puedes experimentar emociones y puedes construir hipótesis sobre por qué algo es bueno o malo. Y cuanto más tiempo trabajes con una tecnología, mejor será tu hipótesis. Esto te ayudará a tomar

mejores decisiones en tu trabajo diario. Cuando empecé a programar, no tenía ninguna finalización de código y sólo algunas librerías de utilidades para acelerar el desarrollo. Obviamente, con estos antecedentes tomaría decisiones equivocadas hoy. Hoy en día, tenemos toneladas de lenguajes de programación, frameworks, herramientas, procesos y prácticas. Sólo si tienes alguna experiencia y una visión general de las principales tendencias, podrás participar en la conversación y dirigir el desarrollo en la dirección correcta.

Encuentra las cosas correctas para probar: No se puede probar todo. Esto es simplemente imposible. Se necesita un enfoque más estructurado. Una fuente que descubrí recientemente es el radar de tecnología de ThoughtWorks. Categorizan tecnologías, herramientas, plataformas, lenguajes y frameworks en cuatro categorías: Adoptar, Juzgar, Evaluar y Retener significa "fuerte sentimiento de estar listo para el uso de la empresa", Prueba significa "la empresa debe intentarlo en un proyecto que pueda manejar el riesgo", Evaluar significa "explorar cómo afecta a su empresa" y Mantener significa "procesar con precaución". Con esta categorización es más fácil obtener una visión general de las cosas nuevas y su preparación para evaluar mejor qué tendencia explorar a continuación.

(5) Documentar

La documentación arquitectónica es a veces más y a veces menos importante. Los documentos importantes son, por ejemplo, las decisiones arquitectónicas o las directrices de código. La documentación inicial se requiere a menudo antes de comenzar la codificación y necesita ser refinada continuamente. Otra documentación puede ser generada automáticamente ya que el código también puede ser documentación, por ejemplo, diagramas de clase UML.

Código Limpio: El código es la mejor documentación si se hace bien. Un buen arquitecto debe ser capaz de distinguir entre código bueno y malo. Un gran recurso para aprender más sobre el código bueno y malo es el libro "Clean Code" de Robert C. Martin.

Generar documentación siempre que sea posible: Los sistemas están cambiando rápidamente y es difícil actualizar la documentación. Tanto si se trata de APIs como de paisajes de sistemas en forma de CMDBs: La información subyacente cambia a menudo demasiado rápido para mantener la documentación correspondiente actualizada a mano. Ejemplo: En el caso de las APIs, puedes generar automáticamente documentación basada en el archivo de definición si está dirigido por un modelo, o directamente desde el código fuente. Existen muchas herramientas para ello, creo que Swagger y RAML son un buen punto de partida para aprender más.

Tanto como sea necesario, tan poco como sea posible: Cualquier cosa que necesites documentar, por ejemplo, documentos de decisión, trata de enfocarte en una sola cosa a la vez e incluye sólo la información necesaria para esta única cosa. La documentación extensa es difícil de leer y de entender. La información adicional debe almacenarse en el apéndice. Especialmente para los documentos de decisión es más importante contar una historia convincente en lugar de limitarse a lanzar toneladas de argumentos. Además, esto les ahorra a ti y a tus compañeros de trabajo, que tienen que leerlo, mucho tiempo. Echa un vistazo a alguna documentación que hayas hecho en el pasado (código fuente, modelos, documentos de decisión, etc.) y hazte las siguientes preguntas: "Se incluye toda la información necesaria para entenderlo", "¿Qué información se necesita realmente y qué información podría omitirse" y "¿Tiene la documentación una línea roja?

Obten más información sobre los marcos de trabajo de arquitectura: Este punto podría aplicarse también a todos los demás puntos "técnicos". Lo pongo aquí, ya que marcos como TOGAF o Zachmann están proporcionando "herramientas" que se sienten pesadas en el sitio de documentación, aunque su valor añadido no se limita a la documentación. Obtener la certificación en un marco de este tipo le enseña a abordar la arquitectura de forma más sistemática.

(6) Comunicar

Si eres brillante en el diseño pero no puedes comunicar tus ideas, es probable que tus pensamientos tengan menos impacto o incluso no tengan éxito.

Aprende a comunicar tus ideas: Cuando se colabora en un pizarrón o rotafolio, es esencial saber cómo utilizarlo correctamente para estructurar tus pensamientos y los de tus compañeros. El libro "UZMO?-?Thinking With Your Pen" me pareció un buen recurso para mejorar mis habilidades en esta área. Como arquitecto, normalmente no sólo participas en una reunión, sino que también tienes que dirigirla y moderarla.

Dar charlas a grupos grandes: Presentar tus ideas a un grupo pequeño o grande no debería ser un problema para ti. Si te sientes incómodo con esto, empieza a presentarte a tu mejor amigo. Agranda el grupo lentamente. Esto es algo que sólo se puede aprender haciendo y abandonando la zona de confort personal. Sea paciente contigo mismo, este proceso puede tomar algún tiempo.

Encuentra el nivel de comunicación adecuado: Las diferentes partes interesadas tienen diferentes intereses y puntos de vista. Necesitan ser abordados individualmente a su nivel. Antes de comunicarte, da un paso atrás y comprueba si la información que quieres compartir tiene el nivel adecuado, en cuanto a abstracción, contenido, objetivos, motivaciones, etc. Ejemplo: Un desarrollador suele estar interesado en el mínimo detalle de la solución, mientras que un gerente prefiere saber qué le permite ahorra más dinero.

Comuníquate con frecuencia: Una arquitectura brillante no vale nada si nadie lo sabe. Distribuye la arquitectura de destino y las ideas que hay detrás de ella, regularmente y en todos los niveles (organizativos). Programa reuniones con promotores, arquitectos y gerentes para mostrarles el camino deseado o definido.

Sé transparente: La comunicación regular mitiga la falta de transparencia sólo parcialmente. Necesitas hacer que la razón detrás de las decisiones sea transparente. Especialmente, si la

gente no está involucrada en el proceso de toma de decisiones, es difícil de entender y de seguir la decisión y los fundamentos que la sustentan.

Mantente siempre preparado para dar una presentación: Siempre hay alguien con preguntas y tú quieres dar las respuestas correctas inmediatamente. Trata de tener siempre las diapositivas más importantes en un conjunto consolidado que puedas mostrar y explicar. Eso te ahorra mucho tiempo y te da seguridad.

(7) Estimar y evaluar

Conocer los principios básicos de la gestión de proyectos: Como arquitecto o desarrollador principal, a menudo te piden presupuestos para realizar tus ideas: ¿Cuánto tiempo, ¿cuánta gente, ¿cuántas personas, qué habilidades, etc.? Por supuesto, si planeas introducir nuevas herramientas o marcos de trabajo, necesitas tener una respuesta para este tipo de preguntas de "gestión". Inicialmente, debes ser capaz de dar un estimado aproximado, como días, meses o años. Y no olvides que no se trata sólo de implementar, hay más actividades a considerar, como ingeniería de requisitos, pruebas y corrección de errores. Por lo tanto, debes conocer las actividades del proceso de desarrollo de software utilizado. Una cosa que puedes aplicar para obtener mejores estimaciones, es usar datos anteriores y obtener tu predicción a partir de ellos. Si no dispones de datos anteriores, también puedes probar enfoques como COCOMO de Barry W. Boehm. Si estás desplegado en un proyecto ágil, aprende a estimar y planificar adecuadamente: El libro "Agile Estimating and Planning" de Mike Cohn proporciona una sólida visión general en esta área.

Evaluar la arquitectura "desconocida": Como arquitecto también deberías ser capaz de evaluar la idoneidad de las arquitecturas para el contexto actual o futuro. No es una tarea fácil, pero puedes prepararte para ello teniendo a mano una serie de preguntas que son comunes a todas las arquitecturas. Y no sólo se trata de la arquitectura, sino también de cómo se gestiona el sistema, ya que esto también te proporciona información interna sobre la calidad. Sugiero tener siempre algunas preguntas preparadas y listas para usar. Algunas ideas para preguntas generales:

- **Prácticas de diseño:** ¿Qué patrones sigue la arquitectura? ¿Se utilizan de forma consecuente y correcta? ¿El diseño sigue una línea roja o hay un crecimiento incontrolado? ¿Existe una estructura clara y una separación de preocupaciones?
- **Prácticas de desarrollo:** ¿Se han establecido y seguido las directrices del código? ¿Cómo se versiona el código? ¿Prácticas de despliegue?
- **Aseguramiento de la calidad:** ¿Cobertura de automatización de pruebas? ¿Análisis de código estático y buenos resultados? ¿Existen revisiones por pares?
- **Seguridad:** ¿Qué conceptos de seguridad existen? ¿Seguridad integrada? ¿Existen y se utilizan regularmente pruebas de penetración o herramientas de análisis de seguridad automatizadas?

(8) Equilibrar

La calidad tiene un precio: Si se excede con la arquitectura, se incrementarán los costes y probablemente se reducirá la velocidad de desarrollo. Es necesario equilibrar los requisitos arquitectónicos y funcionales. Se debe evitar la sobreingeniería.

Resolver objetivos contradictorios: Un ejemplo clásico de objetivos contradictorios son los objetivos a corto y largo plazo. Los proyectos tienden a menudo a construir la solución más simple, mientras que un arquitecto tiene en mente la visión a largo plazo. A menudo, la solución simple no encaja en la solución a largo plazo y corre el riesgo de ser desechada posteriormente (costes hundidos). Para evitar que la aplicación vaya en la dirección equivocada, hay que tener en cuenta dos cosas:

- (1) Los desarrolladores y las empresas necesitan entender la visión a largo plazo y sus beneficios para adaptar su solución y
- (2) Los gestores responsables del presupuesto deben participar para comprender el impacto financiero. No es necesario tener el 100% de la visión a largo plazo directamente en su lugar, pero la pieza desarrollada debe encajar en ella.

Manejo de conflictos: Los arquitectos son a menudo el pegamento entre múltiples grupos con diferentes orígenes. Esto puede conducir a conflictos en los diferentes niveles de comunicación. Para encontrar una solución equilibrada que también refleje objetivos estratégicos a largo plazo, a menudo corresponde a los arquitectos ayudar a superar el conflicto. Mi punto de partida en cuanto a la teoría de la comunicación fue el "modelo de cuatro orejas" de Schulze von Thun. Con base a este modelo se puede mostrar y deducir mucho. Pero esta teoría necesita algo de práctica, la cual debe ser experimentada durante los seminarios de comunicación.

(9) Ser proactivo

Ser proactivo es probablemente lo mejor que puede hacer cuando se trata de consultoría y coaching. Si se le pregunta, a menudo es demasiado tarde. Y la limpieza en el sitio de la arquitectura es algo que quieres evitar. Necesitas prever de alguna manera las próximas semanas, meses o incluso años y prepararte a tí mismo y a la organización para los próximos pasos.

Ten una visión: Si estás desplegado en un proyecto, ya sea en una cascada tradicional como enfoque o ágil, siempre necesitas tener una visión de tus objetivos a medio y largo plazo que deseas alcanzar. No se trata de un concepto detallado, sino más bien de una hoja de ruta para

que todos puedan trabajar. Como no se puede lograr todo a la vez (es un viaje) prefiero usar modelos de madurez. Proporcionan una estructura clara que se puede consumir fácilmente y dan el estado actual del progreso en todo momento. Para diferentes aspectos utilizo diferentes modelos, por ejemplo, prácticas de desarrollo o entrega continua. Cada nivel en el modelo de madurez tiene requisitos claros que siguen los criterios SMART para facilitar la medición de si lo ha logrado o no. Un buen ejemplo que encontré es para la entrega continua.

Construir una comunidad de práctica (CoP): El intercambio de experiencias y conocimientos entre un grupo de interés común ayuda a distribuir ideas y estandarizar los enfoques. Por ejemplo, podrías reunir a todos los desarrolladores y arquitectos de JavaScript en una sola sala, cada tres meses más o menos, y discutir los retos pasados y actuales y cómo se abordaron o las nuevas metodologías y enfoques. Los arquitectos pueden compartir, discutir y alinear sus visiones, los desarrolladores pueden compartir experiencias y aprender de sus pares. Esta ronda puede ser muy beneficiosa para la empresa, pero también para el propio individuo, ya que ayuda a construir una red más fuerte y a distribuir las ideas. También revisa el artículo Comunidades de Práctica del Marco de Trabajo de SAFe que explica el concepto de CoP en un entorno ágil.

Lleva a cabo sesiones de puertas abiertas: Una fuente de conceptos erróneos o ambigüedad es la falta de comunicación. Bloquea una franja horaria fija, por ejemplo, 30 minutos cada semana, para intercambiar temas candentes con tus compañeros. Esta sesión no tiene agenda, todo puede ser discutido. Trata de resolver las cosas menores en el acto. Programa seguimientos sobre los temas más complejos.

(10) Marketing

Tus ideas son grandes y las has comunicado bien, ¿pero aún así nadie quiere seguirlas? Entonces probablemente te faltan habilidades de marketing.

Motivar y convencer: ¿Cómo te convencen las empresas de comprar un producto? Demuestran su valor y beneficios. Pero no sólo con 5 puntos. Te envuelven bien y lo hacen lo más fácil posible de digerir.

- **(1) Prototipos:** Muestra un prototipo de tu idea. Hay muchas herramientas para crear prototipos. En el contexto de las empresas a las que les encanta SAP, visita build.me, en el que puedes crear aplicaciones UI5 de aspecto atractivo y en las que puedes hacer clic de forma rápida y sencilla.

- **(2) Muestra un video:** En lugar de "diapositivas aburridas", también puedes mostrar un vídeo que demuestre tu idea o al menos la dirección.

Pero por favor, no exageres con el marketing: A largo plazo, el contenido es el rey. Si tus palabras no se hacen realidad, esto dañará tu reputación a largo plazo.

Lucha por tus ideas y sé persistente: A veces a la gente no le gustan tus ideas o son demasiado perezosos para seguirlas. Si realmente estás convencido de tus ideas, deberías ir continuamente tras ellas y "luchar". Esto es a veces necesario. Las decisiones de arquitectura con objetivos a largo plazo a menudo no son las más fáciles: A los desarrolladores no les gustan, ya que son más complejos de desarrollar. A los directivos tampoco les gustan, ya que son más caros a corto plazo. Este es su trabajo ser persistente y negociar.

Tomado de: https://www.abiztar.com.mx/articulos/10-puntos-para-convertirte-en-mejor-arquitecto-de-software.html