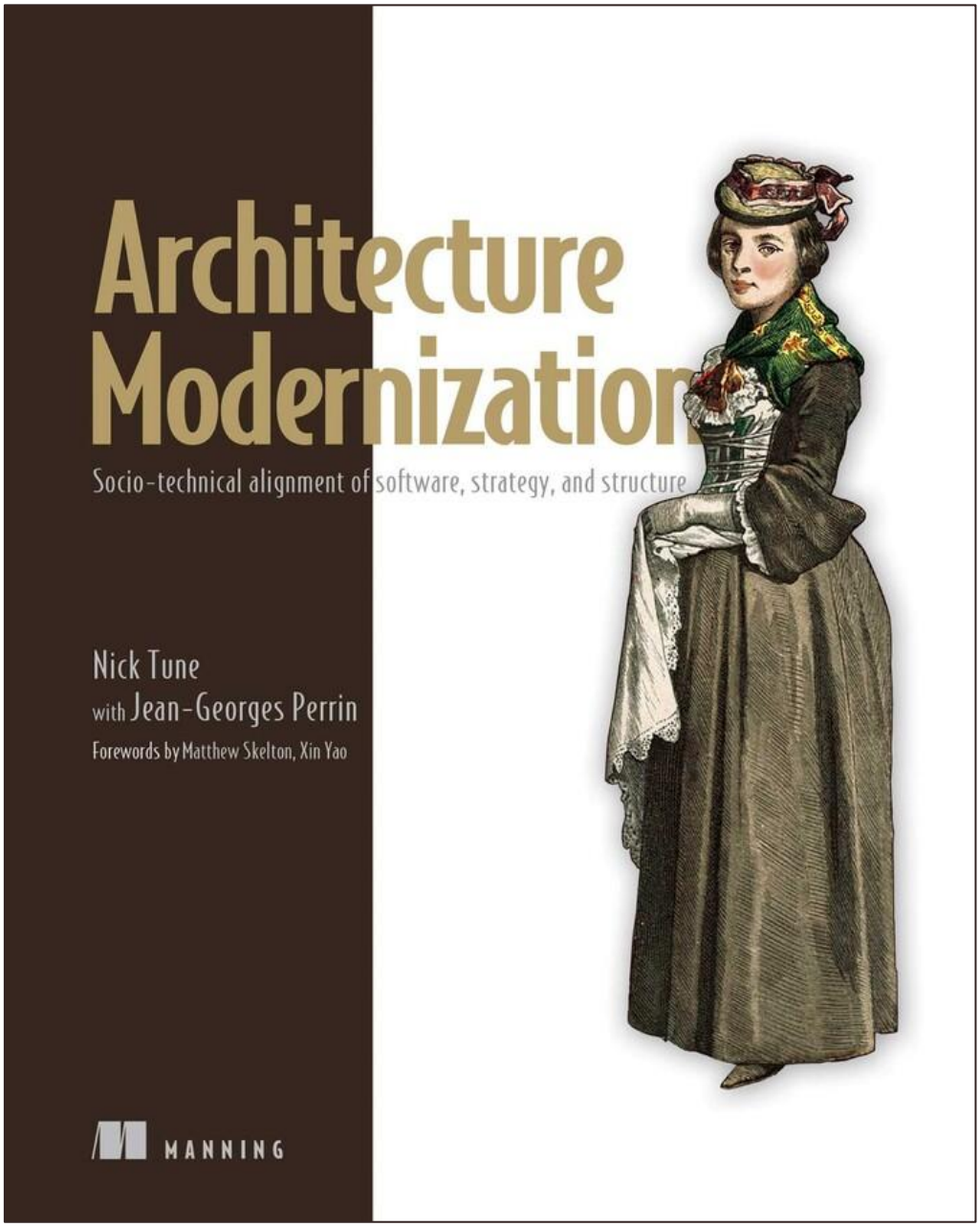
Architecture Modernization



Miguel Fernández Huerta | UO287577

Bruno Isla Sierra | UO293994

Adrián Mahía Loredo | UO289390

Índice

Los entrevistados

Los entrevistados en este podcast son **Jean-Georges Perrin** y **Nick Tune**. Ambos son expertos en sus respectivos ámbitos, relacionados con la arquitectura de *software*, como se ve a continuación.

Jean-Georges Perrin es:

- Director de innovación en ABI Data.
- Presidente del estándar Open Data Contract.
- Cofundador del grupo de usuarios IDA y autor de varios libros, entre ellos "*Architecture Modernization*", en colaboración con Nick Tune.
- IBM Champion de por vida.
- PayPal Champion.
- Y recientemente fue nombrado Data Mesh MVP.

Por otro lado, **Nick Tune**:

- Trabaja junto a líderes de producto y tecnología con el objetivo de:
  - Definir estrategias.
  - Modelar dominios.
  - Diseñar arquitecturas.
  - Y formar equipos de entrega continua.
- Además, es autor del libro "*Principles and Practices of Domain-Driven Design*".

Nick y Jean plantean la **modernización de la arquitectura** como solución al problema de que, mientras el software se mantiene estático…

- El mundo cambia constantemente.
- Surgen nuevas tecnologías, patrones y modelos arquitectónicos.
- Y el modelo de negocio de la empresa evoluciona.

Como consecuencia de todo esto, el software se vuelve obsoleto.

La modernización busca eliminar las desventajas de los sistemas heredados haciendo uso de prácticas, métodos y reflexiones modernas como:

- La migración a la nube.
- La adopción de arquitecturas basadas en microservicios.
- La integración y despliegue continuo.
- Etc.

Enfoque de la modernización de arquitectura.

Durante el podcast, surge la **pregunta** de si el objetivo de la modernización consiste en realizar **cambios a gran escala o mejoras incrementales**.

Nick responde diciendo que **depende del contexto**. Hoy en día, muchas empresas evitan proyectos de modernización que duren demasiado (por ejemplo: 3 o 5 años), ya que:

- Suponen un **coste elevado**.
- No garantizan resultados inmediatos ni a corto plazo.
- Y pueden **paralizar** servicios críticos, provocando la pérdida de clientes.

Sin embargo, **cuando la obsolescencia del sistema impide el crecimiento del negocio**, los cambios grandes son inevitables.

Estos cambios pueden abordarse de dos formas, dependiendo del nivel de deuda técnica:

- Si la deuda técnica es **alta**, hay que realizar un proyecto de modernización grande, lo que implica un gran esfuerzo inicial.
- Si la deuda técnica es **baja**, se puede llevar a cabo de manera más gradual.

La modernización es un proceso continuo, no un evento único.

Independientemente de si partimos con una alta o baja acumulación de deuda técnica, es importante que el proceso de modernización de la arquitectura se realice de forma continua.

En caso contrario, la deuda técnica volverá a acumularse, generando costos y obstáculos innecesarios en el futuro.

Además, un **problema** que hay que enfrentar al modernizar la arquitectura es que **continuamente las empresas quieren que su software tenga más funcionalidades que satisfagan más necesidades de los clientes**, lo que genera un **conflicto** entre:

- **Modernizar** el sistema.
- Y **seguir desarrollando** nuevas características.

Este equilibrio es **complejo**. Muchas veces, priorizar nuevas funcionalidades **retrasa** o **complica** la modernización, lo que acaba empeorando los problemas ya mencionados.

La arquitectura de software moderna es social y técnica.

En el podcast, Nick nos dice que la arquitectura de software moderna **es social y técnica**, ya que al final se deben tomar decisiones a nivel de:

- **Responsabilidades**: Decidir qué equipo se **encarga** de qué parte del sistema.
  - Esto influye en cómo diseñamos la arquitectura.
- **Fiabilidad**: Decidir cómo **dividir** la arquitectura y los equipos para minimizar los errores.
  - La segmentación incorrecta puede generar más problemas.

Para ello, recomienda seguir un enfoque basado en:

- **Diseño orientado al dominio (DDD)**: que nos ayuda a definir límites claros.
- **Identificación de transacciones críticas**: para saber qué datos deben actualizarse de forma atómica.
- **Distribución de responsabilidades**: en función de estos límites.

La importancia del aspecto social.

Jean también menciona que el aspecto social del proyecto puede percibirse de diferente forma en función de la etapa:

1. **En el inicio del proyecto.**
    - La modernización nace como una decisión estratégica.
    - Aquí el aspecto social se centra en **lograr aceptación**, es decir:
        - **Ganar apoyo inicial y preparar mentalmente** a la organización para lo que viene.

2. **Y durante la implementación**.
    - La arquitectura evoluciona, y con ella, la forma en la que trabajamos.
    - Aparecen **términos** como *microservicios* o *cloud-native*, que, aunque suenan innovadores, pueden generar **confusión, dudas o miedo** dentro del equipo.
    - En esta etapa, el aspecto social se centra en **facilitar adaptación**, es decir:
        - **Asesorar/guiar** a la organización **durante el cambio**.

¿Cómo decidimos qué modernizar en el sistema?

En el podcast se nos habla de **cómo decidir qué modernizar en el sistema**.

Este proceso se puede dividir en tres pasos:

1. **Analizar todo el sistema** con el objetivo de pensar en **cómo mejorar cada parte** del sistema para hacerlo **más eficiente** y no modernizar por modernizar.
2. **Identificar qué aspectos revisar**, como, por ejemplo:
    - Experiencia de usuario.
    - Modelo de dominio y datos.
    - Código mal diseñado.
    - O diferencias de vocabulario entre lo que habla la empresa y lo que aparece en el código.
3. **Pensar qué cosas mantener y qué cambiar**, para ello es importante:
    - **Entender el valor del cambio**, es decir, saber si realmente modernizar esa parte traerá una mejora tangible.
    - Y determinar qué **área nos ofrece el mayor valor con el menor esfuerzo posible para darle una mayor prioridad**, a esto último se le conoce como el mejor retorno de la inversión (ROI).

Cómo modernizar los sistemas

Se mencionan varias herramientas posibles para modernizar nuestros sistemas de *software*, como los '*wordly maps*' o el '*behavioral code analysis*', aunque no se hace hincapié en ninguna de ellas. En lo que sí que insisten ambos invitados es en que una modernización conlleva una serie de pasos, que incluyen siempre, al menos:

- ✓ Definir unos objetivos de negocio
- ✓ Comprender el sistema actual a modernizar
- ✓ Planificar el sistema futuro
- ✓ Diseñar la estrategia para la transición

Antes de elegir cualquier técnica para ser aplicada a nuestro proyecto, es crucial **entender** el problema que estamos resolviendo.

El **proceso de modernización** generalmente implica avanzar desde el porqué hasta el cómo. Esto es, tener una línea muy clara que una tu estrategia de negocio con la implementación que se llevará a cabo. Un ejemplo del mundo real ilustra cómo abordar la modernización centrándose en los objetivos de negocio, y después identificando las áreas claves para el cambio.

Ejemplo del UK Government

Nick nos habla de primera mano de uno de los mayores problemas que tienen los proyectos grandes, y que indican una alta necesidad de modernización, que es la separación entre las normas de negocio (*business rules*) y las normas de los equipos de implementación: el problema viene cuando una de las dos, normalmente la parte de negocio, es mucho más antigua que la otra, y dificulta la modernización del *software* y de las estructuras de implementación.

Este era el caso del Gobierno de Inglaterra, en el que los sistemas informáticos, pese a llevar relativamente poco tiempo en funcionamiento, se regían por las normas de negocio. Estas eran muy antiguas (probablemente tenían más de un siglo), y siguiéndolas, el trabajo se hacía imposible.

Todo esto indicaba que **era necesaria una modernización a gran escala**.

La importancia de los datos en la arquitectura

Ambos invitados, pero sobre todo Jean-Georges, insisten reiteradamente en la importancia que tiene el manejo de los datos en la arquitectura de *software*. Nos recuerdan que cualquier sistema es inútil sin datos, y que todas las aplicaciones modernas se mueven alrededor de ellos. Todo esto suele ser olvidado cuando se habla de arquitectura, y más de lo mismo con la modernización.

El principal problema con los datos es que su estructura ha cambiado mucho, pero la práctica de la ingeniería de datos, no. Esto crea equipos muy centralizados, lo que puede acarrear problemas de organización y de arquitectura de empresa.

Antiguamente, el único manejo necesario de los datos consistía en hacer *queries* simples. Los datos siempre se habían enfocado en simplemente hacer su trabajo y nada más, pero en la actualidad el panorama ha cambiado, ya que se hace muy necesario:

- El análisis de comportamiento de los usuarios
- La extracción de conclusiones a partir de los datos que dejan los usuarios
- Y muchas más cosas que antes ni si quiera se contemplaban

Las prácticas de ingeniería de datos **no han sabido actualizarse, y por tanto adaptarse** a estos nuevos requerimientos. No han sabido modernizarse.

Todo esto nos lleva a plantear la necesidad de **modernizar la estructura organizativa de los datos**.

En organizaciones grandes, es normal tener un "*data team*", el cual se centrará en la infraestructura de los datos, y por otro lado un "*architecture or software team*", centrado en lo que consisten los sistemas y aplicaciones.

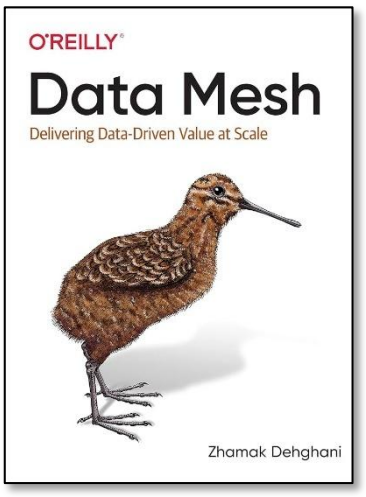
Estos equipos trabajan de manera separada, y esto puede llevar a la formación de cuellos de botella. Por tanto, esta estructura obstaculiza la modernización de la arquitectura general de un proyecto, ya que los equipos tienen formas diferentes de trabajar, y todo esto suele llevar también a un ambiente tóxico, lo cual generará aún más problemas a largo plazo.

Por otro lado, **dentro de los propios equipos de datos**, sus componentes se suelen centrar demasiado en mantener sistemas antiguos en vez de modernizarse: esto, como todos sabemos, los llevará a acumular deuda técnica.

Además, estos suelen decir que están siguiendo modelos ágiles, ya que siguen modelos de cascada comúnmente llamados "*mini waterfall methods*", los cuales, en realidad, **no** son escalables, ni tampoco son en la práctica ágiles. En el mundo de los datos no se están usando técnicas actuales de modernización.

Posible solución para los datos: Data Mesh

Jean-Georges sugiere que una solución que acabaría con la mayor parte de los problemas que nos crea la gestión de los datos, sería utilizar una técnica llamada *Data Mesh* para gestionar nuestros preciados datos.



Esta se basa en una gestión descentralizada de los datos que los centraliza en torno a dominios de negocio específicos. En vez de tener un equipo centralizado para la gestión de datos, se delega la propiedad y responsabilidad de los datos a los equipos de domino que los producen y consumen.

Con esto, si se hace correctamente, se lograría entonces tres grandes mejoras:

- Aumento de la **escalabilidad**
- Una mayor **agilidad** en el desarrollo
- **Alineación** entre los diferentes equipos

Ejemplo de modernización de datos

Un ejemplo que se nos plantea, y que es muy frecuente en entornos empresariales hoy en día, es el de trasladar el almacenaje de los datos a la nube. Esto es muy atractivo para las empresas, ya que este *outsourcing* conllevaría la delegación de la preocupación de tener que mantener unos servidores especializados en la conservación de datos y de los costes extras que ello podría implicar.

El principal inconveniente llega porque, a menudo, hay empresas que simplemente copian la **estructura de datos** que tenían localmente a la nube sin reestructurar la arquitectura. Esto se llama comúnmente "*lift and shift*", y mal hecho trae consigo varios inconvenientes:

- Si las tecnologías utilizadas son diferentes, tendremos esquemas no optimizados.
- Los modelos de facturación y de rendimiento cambiarán.
- Aumento de la deuda técnica por no enfocarse en adaptarse al cambio

El **enfoque correcto** para hacer este *lift and shift* sería, en vez de copiar la estructura actual, invertir tiempo en **reestructurar toda la gestión de los datos**, y, por tanto, **modernizarla**.

Esta migración y reestructuración es una muy buena oportunidad para ejecutar una modernización, pudiendo aplicar conceptos modernos de ingeniería de datos que no estuvieran en la versión *legacy* de nuestro sistema.

Por poner un ejemplo, si al hacer el *lift and shift* se cambia de tecnología, de *RedShift* a *SQL Server*, se debería aprovechar para optimizar los '*data flows*' a la nueva.

¿Cuándo modernizar la arquitectura?

Los dos invitados comparten su visión respecto a cuándo se hace necesario modernizar una arquitectura de *software*. Veamos ambas opiniones, una por una:

- **Según Jean-Georges**

Existen varios indicadores que nos dicen que este proceso se debería llevar a cabo. Sin embargo, hay tres de ellos que suelen ser los más habituales:

- Se pretende ganar más dinero
- Se quiere ahorrar costes (también, lógicamente, para ganar más dinero)
- Cumplir con normativas: si una aplicación de fuera de la Unión Europea se quiere poner a funcionar en alguno de sus territorios, se deberá reestructurar, por ejemplo, su gestión de los datos, para cumplir así con el RGPD, entre otras cosas. Muchas veces la adaptación de la gestión de datos 'obliga' a modernizar el resto de la arquitectura.

Nos comenta Jean-Georges que, aunque estos tres motivos son los más habituales para modernizar arquitecturas de datos, coinciden también con los motivos para comenzar una modernización a mayor escala de cualquier otra arquitectura de *software*.

- **Según Nick Tune**

Nick menciona varias razones para modernizar:

- Ahorrar dinero o ganar más: si el software actual requiere mucho esfuerzo en soporte y corrección de errores, modernizarlo puede ser una opción para reducir errores y por tanto, costes.
- Crecimiento y nuevas oportunidades: A veces, el sistema actual no permite innovar. Por ejemplo: un cliente pide una función "sencilla" (como unir datos de tres sistemas), pero como todo está desorganizado, implementarla lleva meses en vez de horas. Nick señala que "Si dices 'no' a cosas que deberían ser fáciles, es señal de que la arquitectura necesita un cambio."
- Acoplamiento excesivo: Al principio, tener todo conectado puede funcionar, pero cuando la empresa escala, se vuelve más difícil de mantener.

Modernizar la arquitectura no es solo por lo técnico, sino para que el negocio sea más ágil, barato y competitivo.

¿Vale siempre la pena modernizar?

Se lanza la pregunta sobre si alguna vez se han rechazado planes de modernización, y Nick, como consultor, dice que sí: esto ocurre con mayor frecuencia de lo esperado. Algunos de los factores que influyen son los siguientes:

1. Cuando el coste y el esfuerzo no se justifican

   Nick cuenta que muchas veces las empresas lo llaman con grandes ideas de modernización, pero cuando él les explica que eso puede llevar años y mucho dinero, se echan para atrás. A veces simplemente no pueden justificarlo al CEO o al CFO especialmente si lo que quieren es algo como pasar a microservicios y eso implica una reestructuración enorme.

2. Comprar en lugar de construir

   Aquí entra Jean-Georges, que tiene una intervención interesante: "A veces, en vez de modernizar, es mejor comprar una solución ya hecha." Por ejemplo, si tienes que conectar tu CRM con tu programa de fidelización, puede que te convenga comprar un software que ya lo haga, en vez de rehacer todo tu sistema.

3. Elegir las batallas con estrategia

   Giovanni añade que este tipo de decisiones son parte de la estrategia inicial, donde usas herramientas como *Wardley Mapping* para decidir qué partes del sistema son realmente importantes para el negocio y cuáles podrías reemplazar sin problema.

Nick nos vuelve a dar de nuevo un ejemplo real: en una empresa en Nueva Zelanda, estaban listos para arreglar todo su sistema antiguo. Pero al usar *Wardley Mapping*, se dieron cuenta de que, si se centraban en eso, no tendrían recursos para desarrollar proyectos nuevos con IA, que eran más prometedores. Finalmente decidieron comprar un sistema estándar y enfocar a su equipo en lo nuevo e innovador.

Sistemas que no necesitan modernización

Se hace la pregunta sobre si Nick ha presenciado un sistema tan bien diseñado que nunca necesitase modernización, a lo que responde que sí, aunque solo en un par de ocasiones en 15 años.

A continuación, Nick detalla los factores comunes que presentaban estos sistemas:

- Equipos autónomos y bien formados: Los equipos eran independientes y practicaban metodologías como TDD (Test Driven Development) y pair programming. Les importaba mucho la calidad y el código limpio.
- Cultura de mejora continua: Siempre estaban refactorizando, aprendiendo cosas nuevas, incluso tenían tiempo para formarse dentro del horario laboral.

- Apoyo desde arriba: No era solo cosa de los desarrolladores. Había CTOs que valoraban la calidad técnica y CEOs que confiaban en ellos. Esa confianza permitía a los equipos mantener un ritmo sano de trabajo sin tener que justificar cada cosa técnica que hacían.
- Resultados tangibles: Como el sistema estaba tan bien montado, podían hacer despliegues a producción todos los días, y si había un error, lo arreglaban en una hora. Eso generaba credibilidad con los líderes del negocio.

Modernización VS Otras Prioridades

Esta parte trata sobre cómo encajar la modernización en medio de todas las otras prioridades de una empresa, como nuevas funcionalidades o proyectos urgentes.

De nuevo, interviene Nick Tune, primero ilustra el conflicto común que surge a los equipos de trabajo: cuando reciben instrucciones tanto de comenzar la modernización como de avanzar en las nuevas funcionalidades, habitualmente acaban avanzando con las funciones, pues es algo más tangible. A continuación, señala una serie de recomendaciones, que incluyen:

- Un mensaje claro y alineado desde la dirección: Todos los líderes (CEO, CTO, CPO) deben mandar el mismo mensaje: “Modernizar es importante y nos traerá beneficios reales a medio/largo plazo.”
- Mostrar la conexión con objetivos de negocio. No basta con decir que algo es técnico. Hay que explicar que renunciar a una funcionalidad hoy puede dar 10 o 100 veces más valor en un año o dos.
- Hacer varios ‘*roadmaps*’ y decidir en conjunto: Nick recomienda crear tres escenarios de planificación: Uno con mucha modernización y pocas funcionalidades nuevas, uno equilibrado y uno con muchas funcionalidades y poca modernización. Luego, se discuten los pros y contras con los interesados y se decide.

Fracaso de los planes de Modernización

En la última parte de la entrevista, se centran en una pregunta algo pesimista pero muy realista: ¿con qué frecuencia fracasan los programas de modernización arquitectónica? Más importante aún, ¿por qué?

Nick Tune comienza con una intervención bastante clara: “Bastante seguido. A veces prometen que te dejarán tiempo para modernizar, pero en cuanto surge una nueva feature importante… la modernización se pospone indefinidamente.” A continuación, enumera una lista de los motivos más frecuentes de fracaso:

- Falta de una razón clara y fuerte: Sin un motivo convincente para modernizar, las prioridades volverán al trabajo de producto.
- Falta de conocimiento y habilidades: A veces hay una gran visión desde arriba (CTO), pero los equipos no saben cómo implementar bien la modernización. No entienden conceptos como: DDD (Domain-Driven Design), separación de capas (UI, lógica de negocio, etc.), desacoplamiento de sistemas

- Mal manejo del proceso de migración: Muchos se quedan atrapados a mitad de camino, con una mezcla de sistema nuevo y sistema viejo. Esto genera más complejidad que antes: Interfaces duplicadas, datos inconsistentes, UIs con valores diferentes (por ejemplo, dos precios distintos)
- Falta de planificación y disciplina: No anticipan los riesgos de quedar a medias o no tienen el coraje de decir “no” a nuevas funcionalidades cuando están en medio de una migración crítica.

Después, Jean-Georges Perrin comenta dos claves para tener éxito en el proceso de modernización:

- Modernización incremental: Aunque no siempre se cumple lo que se planeó al inicio, siempre se puede generar valor poco a poco, como en cualquier proyecto ágil.
- Evitar sistemas paralelos indefinidamente: Tener 2 o incluso 3 sistemas a la vez, como pasa en algunos bancos, es una probable fuente de problemas.

Para dar fin al *pódcast*, Nick zanja el tema con la siguiente frase: “O lo haces, o no lo haces. Pero lo peor es quedarte a medias y crear un sistema Frankenstein.”, haciendo referencia a los sistemas que se quedan a medias entre el antiguo y el modernizado, y que sufren errores e incoherencias por todas partes.

Preguntas.

Pregunta 1.

¿Qué enfoque pensáis que puede ser más efectivo para modernizar una arquitectura sin interrumpir demasiado las operaciones del negocio?

Buscar el equilibrio entre modernizar y desarrollar nuevas funcionalidades, priorizando lo que la empresa necesite de forma inmediata y realizando el proceso de modernización de forma incremental, es decir, con pequeños cambios, pero significativos.

Pregunta 2.

El episodio termina con advertencias sobre riesgos en modernizaciones. ¿Cuál creéis que es el error más común en proyectos de modernización y cómo mitigarlo desde el día uno?

En mi opinión, considero que el error más común es comenzar el proceso de modernización sin una razón sólida como base. Para evitarlo sería conveniente elaborar un plan de análisis que revele si realmente es necesario o merece la pena llevar a cabo el proceso, para así estar seguro de que se trabaja sobre razones coherentes. Para ello es clave la **comunicación** en la empresa.