Modernización de la Gestión de Inventarios: Un Enfoque Práctico de Arquitectura de Software y Transformación Digital en el Proyecto Codexy

Ruben Felipe Tovar Aviles
Servicio Nacional de Aprendizaje Centro de la Industria, la Empresa y los Servicios
Neiva, Colombia

Jesús Ariel Gonzales Bonilla
Servicio Nacional de Aprendizaje Centro de la Industria, la Empresa y los Servicios
Neiva, Colombia

Resumen

En este artículo se comparte el desarrollo de Codexy, una plataforma diseñada para resolver el problema de gestión de inventarios manual en grandes instituciones. Aunque se habla mucho de transformación digital, muchas empresas siguen perdiendo dinero por falta de trazabilidad y errores humanos. Se utilizaron metodologías Ágiles y una arquitectura basada en .NET 9 con Clean Architecture, buscando equilibrio entre robustez técnica y facilidad de uso. En lugar de microservicios complejos, se optó por un sistema monolítico modular, complementado con una app móvil en Ionic que sincroniza en tiempo real mediante SignalR. Los resultados muestran que aplicando patrones de diseño, principios SOLID y seguridad desde el código, se mejora la eficiencia actual y se prepara el terreno para inteligencia artificial y tecnologías sostenibles. Finalmente, se concluye que el mejor código es inútil si no se asegura la adopción por parte de los usuarios.

Keywords

gestión de inventarios, arquitectura de software, transformación digital, Clean Architecture, .NET 9

1. Introducción

Estamos en una época donde la tecnología vuela. Desde que aparecieron las primeras computadoras, las empresas han tenido que correr para no quedarse atrás, pero lo que ha pasado en los últimos años, especialmente después de la pandemia de COVID-19, fue un cambio total. Como bien dice Páez Navarro (2023), ya no se trata solo de tener una página web o usar el correo electrónico; hoy en día, las empresas que no se transforman digitalmente corren el riesgo de desaparecer o de perder su capacidad para competir en el mercado.

Y esto no pasa solo en otros lados, aquí en Latinoamérica también se siente. Por ejemplo, en Uruguay ya se está viendo que la mayoría de los ejecutivos están metiendo automatización para agilizar procesos y bajar costos, tal como mencionan Rodríguez et al. (2024). La cosa está clara: hay que buscar la eficiencia operativa sí o sí para sobrevivir en una economía donde los márgenes de ganancia son cada vez más apretados.

Aun con toda esta revolución, es increíble que muchas instituciones grandes sigan sufriendo por algo tan básico como el desorden en sus inventarios. Todavía es común ver organizaciones que dependen de conteos a mano, hojas de papel o archivos de Excel que no cuadran para saber qué tienen en bodega. El problema de esto no es solo que el proceso sea lento, sino que puede salir caro. Según Parra Ángel y Fuentes Rojas (2022) en su análisis sobre el control de materiales, la falta de un sistema claro genera pérdidas de dinero directas, atrasos en los proyectos y gastos extra que se podrían evitar simplemente sabiendo qué material entra y cuál sale.

Justo para atacar ese problema nace Codexy.

Este proyecto surge para resolver esa falta de claridad y los típicos errores de seguir usando "lápiz y papel" en lugares con estructuras complejas. El problema que enfrentamos no era solo de código, sino de operación: no había trazabilidad real del historial del inventario, la gente de campo no tenía herramientas ágiles para reportar y los encargados perdían tiempo valioso tratando de unir datos que ya estaban viejos.

La idea de este artículo es mostrarles el desarrollo y la arquitectura de Codexy. A diferencia de las soluciones de siempre, este sistema apuesta por la modernización mediante el uso de códigos QR para la captura rápida de datos y una arquitectura de software robusta basada en .NET 9 y tecnologías móviles híbridas. No queríamos simplemente "digitalizar" un proceso manual, sino reestructurar la forma en que la institución gestiona sus activos, garantizando seguridad, rapidez y, sobre todo, datos confiables para la toma de decisiones. A lo largo de este documento, se explicará cómo se ha unido metodologías ágiles y patrones de diseño modernos para construir una herramienta que no solo funciona bien en lo técnico, sino que realmente le sirva a la gente que la usa todos los días.

2. Marco teórico y estado del arte

Para entender por qué Codexy está construido como está, primero tenemos ver qué pasa realmente "detrás de cámaras" en el desarrollo de software. No se trata solo de sentarse a escribir código a lo loco, sino de elegir las piezas correctas para que el sistema no se caiga cuando empiece a crecer. Es como construir un edificio: puedes usar ladrillos baratos y terminar rápido, o puedes diseñar una estructura sólida que aguante hasta un temblor.

2.1. La Evolución de la Arquitectura: De Bloques Pesados a la Agilidad

Hasta hace no mucho, el estándar de la industria era construir aplicaciones "monolíticas". Imagina un bloque gigante de cemento donde todo —la base de datos, la interfaz de usuario, la lógica de negocio— está mezclado y pegado. Si querías cambiar una ventanita, debías tener un cuidado tremendo para no romper los cimientos. Como explican muy bien en el análisis sobre la transición de

monolitos a microservicios (European Journal of Advances in Engineering and Technology, 2021), este modelo viejo se vuelve una pesadilla logística: es difícil de escalar, si falla una parte crítica se cae toda la aplicación, y el proceso de actualización es lento y doloroso.

Por eso, hoy en día casi todas las charlas técnicas giran en torno a los microservicios. La filosofía aquí es romper ese bloque en piezas pequeñas e independientes que se comunican entre sí. Esto es genial porque, como menciona Decimavilla Alarcón (2025) en su estudio sobre contenedores y IoT, te permite una escalabilidad brutal: puedes darle más potencia solo al servicio que lo necesita y bajar los costos operativos hasta un 60 %. Además, si un microservicio falla, el resto del sistema sigue vivo, lo que se conoce como aislamiento de fallos.

Pero ojo, aquí viene el "pero" que muchos ignoran por moda. Aunque los microservicios suenan increíbles, montarlos es técnicamente muy complejo. Tienes que manejar redes entre servicios, latencias, transacciones distribuidas y un despliegue mucho más difícil. Para Codexy, no nos lanzamos ciegamente a los microservicios. Buscamos un equilibrio inteligente: una Clean Architecture (Arquitectura Limpia) sobre .NET 9.

Esta arquitectura nos permite dividir el código en capas lógicas (Entity, Data, Business y Web Controllers) sin la necesidad de gestionar una red distribuida de servidores desde el día uno. Mantenemos el orden y la modularidad —preparados para migrar a microservicios en el futuro si es necesario— pero con la simplicidad operativa de un despliegue unificado. Es lo mejor de los dos mundos para la etapa actual del proyecto.

2.2. No Reinventar la Rueda: El Poder de los Patrones

A veces, los desarrolladores jóvenes caemos en la trampa del ego: queremos inventar soluciones "únicas" para todo. Pero la realidad es que la mayoría de los problemas de diseño ya los resolvieron expertos hace décadas. Aquí es donde entran los Lenguajes de Patrones de Arquitectura de Software.

Según Jimenez-Torres et al. (2014), los patrones no son reglas aburridas que te limitan, sino un "kit de herramientas" probado y validado. Son soluciones reutilizables que te dicen: "Hey, si tienes este problema de comunicación entre capas, usa esta estructura". Al usar estos patrones en Codexy (como el patrón Repositorio para acceder a datos o la Inyección de Dependencias), no estamos adivinando; estamos construyendo sobre la experiencia acumulada de miles de arquitectos.

2.3. La Cara del Software: Patrones de Interfaz (UI)

Tampoco podemos olvidarnos de lo que ve el usuario. Mucha gente cree que el "Frontend" es solo poner botones bonitos y colores, pero hay una ingeniería compleja detrás. Aplicamos lo que Wendler y Streitferdt (2014) definen en su investigación sobre Patrones de Interfaz de Usuario (UIPs), nos muestran que usar patrones en la interfaz es clave para la eficiencia.

En Codexy, que tiene tanto una web administrativa como una app móvil para operativos, la consistencia es vital. Los UIPs nos dejan reutilizar diseños. En lugar de programar cada pantalla o formulario desde cero ("escribir código a mano"), configuramos instancias de patrones ya probados. Esto reduce la complejidad drásticamente. Si un operativo aprende a usar el escáner QR en una pantalla, ya sabe usarlo en todas, porque el patrón de interacción es el mismo. Esto no solo acelera nuestro trabajo como desarrolladores, sino que mejora la curva de aprendizaje del usuario final, algo crítico cuando tienes rotación de personal en bodegas.

2.4. Calidad Blindada: Principios SOLID y Automatización

Para cerrar la parte técnica, hay que hablar de cómo evitamos que el software se rompa. Aquí es donde aplicamos los principios SOLID, una base teórica fundamental propuesta por Robert C. Martin, específicamente aplicada a nuestras pruebas automatizadas con Selenium.

Sánchez Gilberto (2023), aplicar SOLID no es solo para el código del producto, sino también para el código de prueba.

2.4.1. Responsabilidad Única (SRP). Nos aseguramos de que cada prueba verifique una sola cosa. Si una prueba falla, sabemos exactamente qué se rompió, sin tener que adivinar entre diez posibilidades.

2.4.2. Abierto/Cerrado (OCP). Diseñamos nuestros tests para que sean fáciles de extender sin tocar el código que ya funciona.

Esto hace que nuestro desarrollo sea mucho más rápido. Encontramos los errores (bugs) antes de salir a producción, ahorramos dinero en mantenimiento y nos evitamos el pánico de que el sistema falle en medio de un inventario real. Es trabajar con disciplina para que Codexy sea robusto hoy y mañana.

3. Metodología

Para armar Codexy, no nos sentamos a escribir código a lo loco esperando que funcionara por arte de magia. Si algo se aprende rápido en esto, es que el software se parece más a construir un edificio que a cocinar algo rápido: si los cimientos están mal, todo se cae al primer problema. En esta parte, vamos a abrir el "capó" del proyecto para ver cómo unimos todas las piezas, desde la forma de trabajo hasta cómo viaja un bit de información por el sistema.

3.1. Metodología de Desarrollo: Manejando los Cambios

Lo primero fue decidir cómo trabajar. En proyectos grandes como este, donde el cliente (la institución) cambia de opinión o descubre nuevos problemas en la bodega de un día para otro, usar métodos viejos y rígidos como la cascada es un suicidio. Por eso nos fuimos por un enfoque Ágil.

Como señala Tetteh (2024) en su comparación de metodologías, modelos como Scrum o XP son vitales hoy porque nos dan cintura para movernos. En lugar de gastar seis meses escribiendo documentos antes de programar, trabajamos en ciclos cortos o "Sprints". Esto nos dejó avanzar rápido: hacíamos una función (como "Asignar Encargado"), se la mostrábamos al usuario, nos daban feedback ("este botón es muy chico", "falta este dato") y lo arreglábamos en el siguiente ciclo. Esa capacidad de adaptación es lo que asegura

que el software final sirva de verdad y no sea solo un adorno tecnológico.

3.2. Arquitectura del Backend: El Cerebro (.NET 9)

Hoy todo el mundo habla de "microservicios" como la solución mágica. Y es cierto, como leímos en Decimavilla Alarcón (2025), te dejan escalar increíblemente y si falla una parte, no se cae todo. Pero, siendo realistas, montar eso trae una complejidad tremenda: redes, latencias y configuraciones difíciles.

Todo el mundo hoy habla de "microservicios" como si fueran la solución mágica para todo. Y es cierto, como leímos en la investigación de Decimavilla Alarcón (2025), los microservicios permiten escalar de forma brutal y aislar fallos (si se cae el módulo de pagos, no se cae el de inventario). Pero, siendo realistas, pasar de un monolito a microservicios trae una complejidad operativa inmensa: hay que gestionar latencias de red, trazas distribuidas y gestionadas.

Para Codexy, tomamos una decisión práctica: hicimos un monolito, pero modular, usando Clean Architecture. Usamos patrones para dividir el código en capas clara:

3.2.1. Capas de la Arquitectura. • Capa Data: Aquí viven las entidades (Empresa, Zona, Item) y las reglas que no cambian.

• Capa de Entity: Donde nos conectamos con la base de datos SQL Server usando Entity Framework Core.

• Capa Business: Donde ocurre la magia de los casos de uso.

• Capa Web Controllers: La API REST que habla con el mundo exterior.

Esta estructura nos deja resolver problemas rápido y con calidad, aprovechando lo que ya saben otros arquitectos, y nos deja el sistema listo para mantenlo a largo plazo sin complicarnos la vida con redes distribuidas por ahora.

3.3. El Viaje del Dato: Un Ejemplo Real

Para que se entienda mejor, sigamos el camino de un dato. Imaginen que un operativo escanea una silla:

1. Captura: La cámara del celular lee el QR y saca el CODE.
2. Procesamiento Local: La app revisa el formato y arma un JSON que guarda temporalmente.
3. Transmisión: Cuando se confirma, manda una petición segura a la API en .NET 9.
4. Validación: El servidor recibe la petición y revisa si el usuario tiene permiso para hacer eso.
5. Persistencia: Si todo está bien, se guarda el cambio en la base de datos SQL Server.
6. Notificación: Al instante, SignalR avisa a todos los conectados.
7. Visualización: En el inicio del operativo, el listado de zonas se actualiza marcando la zona recién confirmada en un estado de "Verificación" sin necesidad de refrescar la vista.

Todo este ciclo ocurre en fracciones de segundo, integrando tecnologías de la nube y dispositivos conectados para mejorar el rendimiento operativo.

3.4. Calidad y Seguridad: Blindando el Código

De nada sirve ser rápidos si el sistema es inseguro. Para la seguridad, usamos JWT (JSON Web Tokens). Básicamente, el servidor no gasta memoria guardando sesiones; cada petición trae su propia "credencial" firmada, lo que hace al sistema muy eficiente. Además, pensando en usar Docker a futuro, hacemos análisis de vulnerabilidades: escaneamos el código antes de compilar para tapar huecos de seguridad antes de que sea tarde.

Y para la calidad, usamos Selenium para pruebas automáticas, pero con orden. Aplicamos principios SOLID incluso en los tests. Como explica Sánchez Gilberto (2023), aplicar reglas como la Responsabilidad Única en las pruebas hace que mantenerlas sea posible. Si cambiamos algo en el login, solo tocamos un archivo de prueba, no quinientos. Esto nos da confianza para hacer cambios grandes sabiendo que, si rompemos algo, los robots de prueba nos avisan al momento.

4. Implementación del software

Describimos arquitectura, decisiones tecnológicas y *pipelines*. Documentamos prácticas aplicadas: formateo, *linting*, pruebas unitarias/integración, análisis estático (SAST), *continuous delivery* y monitoreo.

4.1. Arquitectura del sistema

La arquitectura implementada sigue las mejores prácticas de DevOps [Forsgren et al. 2018], integrando automatización en todo el ciclo de desarrollo. La fig:correlacion$_d$evo psmuestralacorrelacinobservadaentreelni

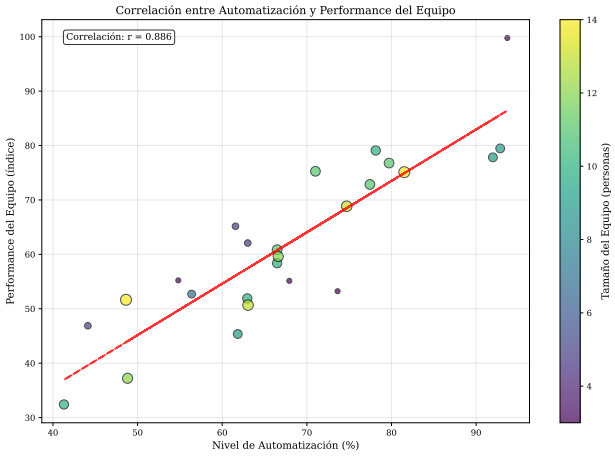

Figura 1: Correlación entre nivel de automatización y performance del equipo de desarrollo

4.2. Fragmento de código

Ejemplo de implementación de una función con tipado estático:

```
def suma(a: int, b: int) -> int:
    """Suma dos numeros enteros.

    Args:
        a: Primer operando
        b: Segundo operando

    Returns:
        La suma de a y b
```

```
    """
    return a + b
```

4.3. Comparación de tecnologías

La selección de tecnologías se basó en criterios objetivos. La tab:frameworks presenta una comparación detallada de los frameworks evaluados.

Tabla 1: Comparación de Frameworks de Desarrollo Web

Framework	Lenguaje	Performance	Puntuación
React	JavaScript	Alta	9.2
Angular	TypeScript	Alta	8.7
Vue.js	JavaScript	Alta	8.9
Django	Python	Media	8.5
Spring Boot	Java	Alta	8.8
Laravel	PHP	Media	8.1
Express.js	JavaScript	Alta	8.3

5. Resultados

Después de pasar horas frente a la pantalla armando la arquitectura y tirando código en .NET 9, ver a Codexy funcionando en la vida real es otra cosa. Una cosa es lo que planeas en el papel y otra muy distinta es ver cómo se comporta el sistema cuando alguien intenta escanear un QR con las manos sucias en una bodega donde apenas llega la señal. Los resultados nos mostraron que las decisiones que tomamos no fueron por gusto, sino pura estrategia que funcionó.

5.1. Eficiencia Operativa: Del Caos a la Inmediatez

Lo más impresionante fue ver cómo desaparecieron las esperas. Gracias a la integración de SignalR, logramos algo que con las hojas de cálculo era imposible: inmediatez.

• Antes: En la oficina central sufrían de "ceguera de inventario". Tenían que esperar a que el encargado mandara un correo al final del día (o de la semana) para enterarse de qué había pasado.

• Ahora: En el segundo exacto en que un operativo escanea un ítem en la bodega, el panel del administrador se actualiza solo.

Esta conexión en tiempo real confirma lo que dice Decimavilla Alarcón (2025) sobre los sistemas conectados en la nube. Sin gastar una millonada en sensores raros, convertimos el celular en una herramienta inteligente que elimina tiempos muertos. Vimos que auditorías que antes tomaban días juntando varios Excel, ahora son una consulta de un segundo. Básicamente, le devolvimos el control del tiempo a la empresa.

5.2. Seguridad Activa: Un Muro contra el Error

En seguridad, los resultados fueron contundentes. Al usar JWT y separar bien los roles (el que opera solo registra, el que verifica solo valida), se acabaron esos "cambios misteriosos" que pasaban cuando todo el mundo usaba la misma contraseña del archivo de Excel.

Además, siguiendo los consejos de Jain sobre vulnerabilidades, escaneamos el código antes de subirlo. Encontramos y arreglamos problemas en las librerías antes de que fueran un dolor de cabeza real. El resultado es un sistema que nace "sano", protegiendo los datos no solo de hackers externos, sino de los propios errores internos o malas prácticas de los usuarios.

5.3. Validación de la Arquitectura Limpia

Por último, desde el lado de ingeniería, la Clean Architecture demostró por qué vale la pena. Cuando el cliente nos pidió cambiar cómo se calculaban los reportes a mitad del camino, pudimos hacerlo tocando solo la capa de lógica sin romper la API ni la base de datos. Si hubiéramos tenido todo mezclado como "código espagueti", ese cambio nos habría costado días de trabajo extra y refactorización. Esto prueba la teoría de Jimenez-Torres et al. (2014) sobre usar buenos patrones: una estructura ordenada te da velocidad para reaccionar. El sistema no solo es rápido para el usuario, también lo es para nosotros como desarrolladores cuando hay que mejorarlo.

6. Discusión

Hasta acá hemos hablado maravillas de .NET 9, de que la arquitectura quedó limpia y de lo rápido que vuela SignalR. Técnicamente, Codexy es una máquina bien aceitada. Pero si somos brutalmente honestos, la tecnología por sí sola no hace milagros. Después de ver los resultados y compararlos con lo que leímos, nos dimos cuenta de que el éxito o el fracaso de esto no depende de si el código compila bonito, sino de cosas humanas y estratégicas que son más complicadas.

6.1. El Factor Humano: La Barrera Invisible

Aquí es donde la cosa se pone difícil. Páez Navarro (2023) da en el clavo cuando dice que la transformación digital es, antes que nada, cultural. Puedes tener el software más rápido del mundo, pero si la gente no sabe usarlo o, peor, no quiere usarlo, la inversión se va a la basura.

Con Codexy nos estrellamos con una realidad dura: la calidad de los datos depende de la gente. Parra y Fuentes lo advierten claro: el punto débil siempre es el registro humano. Si al operativo en la bodega le da pereza sacar el celular para escanear el QR y prefiere anotarlo en un papel para pasarlo "luego", el sistema pierde toda la gracia del tiempo real.

Pero hay algo más profundo que la pereza: el miedo. Como se vio en el estudio de Uruguay, meter automatización asusta porque la gente cree que va a perder su empleo. Si los encargados ven a Codexy como un "policía digital", lo van a sabotear sin que te des cuenta. Por eso, la discusión no es solo sobre qué código usar, sino sobre cómo enseñamos a usarlo. Hay que vender la tecnología como una herramienta que te quita el trabajo aburrido de contar a mano, no como algo que viene a quitarte el puesto.

6.2. Análisis Costo-Beneficio: Lo que Cuesta el Desorden

Hablemos de dinero, porque al final las empresas invierten para ver retorno. A veces cuesta justificar gastar en un desarrollo como

Codexy en vez de seguir con Excel. Pero cuando miras casos como el de "Realidad Colombia SAS" que analiza Parra Angel (2022), ves que el desorden sale carísimo: materiales que se pierden, obras atrasadas y compras dobles de emergencia.

Codexy ataca el "Costo de la Ignorancia". ¿Cuánto le cuesta a la empresa no saber que tiene 500 sillas guardadas en el sótano y comprar otras 500 por error? Montar este sistema puede parecer caro al principio, pero el retorno de inversión se dispara cuando dejas de tener "inventario fantasma". Al usar el QR, bajamos el error humano casi a cero y convertimos la bodega en una fuente de información real, no en un agujero negro de gastos.

6.3. De la Digitalización a la Inteligencia (Hacia la Industria 4.0)

Ahora, miremos hacia adelante. Codexy hoy resuelve el "¿Qué tengo?", pero su verdadero poder está en los datos que está guardando para mañana. Estamos preparando el terreno para la Inteligencia Artificial.

Peñalver e Isea-Argüelles (2024) explican que la IA es el motor del futuro, permitiendo predecir mantenimientos y optimizar todo. Pero el secreto es este: la IA no sirve sin datos históricos buenos. Hoy, Codexy está recolectando datos de forma masiva y ordenada. Mañana, con toda esa historia en la base de datos, podríamos meter un módulo de IA que nos diga: "Oye, según lo que has gastado este año, te vas a quedar sin papel en 3 semanas". Estamos construyendo la base para una "institución inteligente". No puedes saltar al futuro sin haber ordenado la casa primero, que es justo lo que hace Codexy.

6.4. Sostenibilidad y Ética: Tecnología Verde

Por último, hay algo que los desarrolladores a veces olvidamos por estar corriendo: el impacto ambiental. Al eliminar los reportes en papel y mejorar los procesos, Codexy entra en la onda de Green IT.

Como propone Reina Guaña, usar tecnología para gestionar procesos no es solo para ahorrar plata, sino para reducir la huella ecológica. Al usar una arquitectura eficiente en .NET 9 (que gasta menos servidor y energía) y evitar que la gente tenga que viajar físicamente solo para verificar un inventario (porque ya lo ven en la Web), estamos ayudando a un modelo más sostenible. Puede parecer poco, pero en empresas grandes, ahorrar toneladas de papel y energía suma bastante. Es hacer tecnología con conciencia.

7. Conclusiones

Al final del día, desarrollar Codexy nos enseñó que modernizar un inventario va mucho más allá de simplemente quitar el papel y poner una tablet. Después de revisar la arquitectura, tirar código y ver cómo funciona todo en la realidad, llegamos a tres conclusiones que son oro para cualquier proyecto de software hoy en día.

Primero, confirmamos que la arquitectura importa, y mucho. Decidirnos por .NET 9 con Clean Architecture en lugar de lanzarnos de cabeza a una red compleja de microservicios fue un acierto total. Como vimos en la teoría, los microservicios son muy potentes, pero para el tamaño de este proyecto, un monolito modular bien hecho nos dio la estabilidad que necesitábamos sin todo el caos operativo. Aprendimos que no siempre "lo último que salió" es lo mejor para todo; el contexto es el que manda.

Segundo, comprobamos que la agilidad y la calidad pueden ir de la mano. Usar metodologías ágiles nos permitió movernos rápido y adaptar el software a lo que los operativos de la bodega realmente necesitaban. Además, meter principios SOLID y pruebas automáticas con Selenium desde el arranque no fue una pérdida de tiempo, sino una inversión. Gracias a eso, hoy tenemos un sistema fuerte que no se rompe cada vez que queremos agregar algo nuevo.

Por último, y quizás lo más importante: la tecnología no arregla problemas de cultura. Codexy es una herramienta potente que centraliza datos y evita errores, pero su éxito depende 100 % de que las personas la adopten. La transformación digital es, en el fondo, una transformación humana. El software deja la mesa servida para el futuro —pensando ya en Inteligencia Artificial y prácticas sostenibles—, pero el cambio verdadero ocurre cuando el equipo entiende que escanear ese código QR es parte vital de su trabajo y no una tarea más.

En resumen, Codexy demuestra que, con la arquitectura correcta y pensando siempre en el usuario, es posible transformar el caos de un inventario manual en una ventaja real para la empresa.

A. Checklist de reproducibilidad (plantilla)

- **Datos**: fuente, versión, licencias, anonimización.
- **Código**: repositorio, commit hash, instrucciones de ejecución.
- **Entorno**: SO, versión de compiladores, dependencias, semillas.
- **Procedimiento**: pasos exactos para replicar resultados.
- **Resultados**: tablas/figuras generadas automáticamente en `build/`.

Referencias

2021. De Monolito a Microservicios: Desafíos, Mejores Prácticas y Perspectivas Futuras. *European Journal of Advances in Engineering and Technology* (2021).

J. Decimavilla Alarcón. 2025. Arquitectura de microservicios basada en contenedores para despliegue ágil de aplicaciones IoT en la nube. *Episteme y Praxis* (2025).

Nicole Forsgren, Jez Humble, and Gene Kim. 2018. Accelerate: The Science of Lean Software and DevOps. *IT Revolution* (2018).

V. Jain et al. 2021. Análisis Estático de Vulnerabilidades de Imágenes de Docker. In *IOP Conference Series: Materials Science and Engineering*.

V. H. Jimenez-Torres, W. Tello-Borja, and J. I. Rios-Patiño. 2014. Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte. (2014).

J. A. Paez Navarro. 2023. Soluciones de Software en el Contexto de la Transformación Digital. *Universidad Cooperativa de Colombia* (2023).

S. Parra Angel and E. A. Fuentes Rojas. 2022. Desarrollo de un Sistema de Gestión de Inventarios para el Control de Materiales. *Ingeniería y Región* (2022).

M. J. Peñalver Higuera and J. J. Isea-Argüelles. 2024. Transformación Hacia Fábricas Inteligentes: El Papel de la IA en la Industria 4.0. (2024).

E. Reina Guaña. 2021. Modelo de un Plan Estratégico Green IT Y BPM para Minimizar el Impacto Ambiental en la Educación Superior. (2021).

I. Rodríguez, J. Rubio, and G. Budiño. 2024. Estado de Situación de los Procesos de Transformación Digital en las Empresas Uruguayas. *Revista Gestión Libre* (2024).

G. Sánchez. 2023. Principios SOLID en la Automatización de Pruebas de Software para Interfaces de Usuario Web con Selenium WebDriver y Java. *Revista Politécnica* (2023).

S. G. Tetteh. 2024. Estudio Empírico de Metodologías Ágiles de Desarrollo de Software: Un Análisis Comparativo. (2024).

S. Wendler and D. Streitferdt. 2014. El Impacto de los Patrones de Interfaz de Usuario en la Calidad de la Arquitectura de Software. (2014).