CENTRO DE E-LEARNING UTN BA
Medrano 951 CABA, Buenos Aires Argentina
(1179) // tel +54 11 7078 – 8073 / fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Documento clase 15: Mantenimiento y Escalabilidad

1. Introducción: La Profesionalización del Desarrollo Ciudadano en la Era de la Automatización

La transformación digital en América Latina ha experimentado una aceleración sin precedentes en la última década, impulsada no solo por la adopción de nuevas tecnologías, sino por una reconfiguración fundamental en la manera en que las organizaciones conciben la creación de software. Tradicionalmente, el desarrollo de aplicaciones y la automatización de procesos eran dominios exclusivos de departamentos de Tecnología de la Información (TI) altamente especializados. Sin embargo, la emergencia y consolidación de plataformas *No-Code* (sin código) y *Low-Code* (bajo código) ha democratizado estas capacidades, dando lugar a la figura del "Citizen Developer" o desarrollador ciudadano.[1] Este cambio de paradigma, si bien reduce las barreras de entrada para la innovación, introduce desafíos críticos relacionados con la sostenibilidad, el mantenimiento y la escalabilidad de las soluciones generadas fuera de los entornos controlados de la ingeniería de software tradicional.

El presente documento complementario a la Clase 15 de la Diplomatura en Inteligencia Artificial para No Programadores de la UTN.BA tiene como objetivo profundizar en los aspectos técnicos y metodológicos necesarios para que estas automatizaciones sobrevivan al paso del tiempo. Contrario a la creencia popular de que las herramientas *No-Code* son soluciones de "configurar y olvidar", la evidencia académica y la práctica industrial demuestran que estas aplicaciones requieren un ciclo de vida de gestión riguroso, análogo al del software convencional.[4] La literatura especializada, incluyendo los trabajos seminales de Ian Sommerville y Roger Pressman, enfatiza que la fase de mantenimiento consume, en promedio, entre el 60% y el 80% del costo total del ciclo de vida de un sistema de software.[5] Ignorar esta realidad en el ámbito *No-Code* conduce inevitablemente a la creación de deuda técnica, fragilidad operativa y riesgos de seguridad conocidos como *Shadow IT*.[7]

En el contexto latinoamericano, donde la eficiencia operativa y la optimización de recursos son

imperativos económicos, la capacidad de escalar automatizaciones desde prototipos individuales hasta soluciones empresariales robustas se convierte en una ventaja competitiva. Este reporte examina exhaustivamente las estrategias de documentación, control de versiones, monitoreo de errores y arquitectura de datos necesarias para profesionalizar la práctica de la automatización inteligente, integrando principios de ingeniería con la agilidad de las herramientas visuales modernas.

2. Ingeniería de la Documentación en Entornos Visuales

2.1 El Rol Estratégico de la Documentación en la Mantenibilidad

La documentación en el desarrollo de software, y por extensión en la automatización *No-Code*, no debe interpretarse como un requisito burocrático posterior a la implementación, sino como un componente intrínseco de la calidad del sistema. La fragilidad inherente a las integraciones visuales radica en que la lógica de negocio suele estar dispersa entre múltiples servicios (SaaS) y conectores, lo que dificulta la trazabilidad mental del flujo de datos.[4] Un sistema que funciona perfectamente hoy puede colapsar silenciosamente mañana debido a un cambio trivial, como la modificación del nombre de una columna en una hoja de cálculo o la actualización de una API de terceros. Sin una documentación adecuada, el diagnóstico y la reparación de estos fallos se vuelven tareas exponencialmente costosas y propensas al error.

El concepto del "Factor Bus" (o *Bus Factor*) es fundamental en este análisis. Si el conocimiento sobre el funcionamiento, las credenciales y la lógica de una automatización reside exclusivamente en la mente de una sola persona, el riesgo operativo para la organización es inaceptable. Si ese individuo se ausenta, ya sea por vacaciones o desvinculación, el sistema se convierte en una "caja negra" inmanejable.[4] Por tanto, el objetivo primordial de la documentación es garantizar la transferibilidad del conocimiento, permitiendo que cualquier miembro del equipo pueda comprender, mantener y mejorar el sistema en ausencia de su creador original. Esto se alinea con los principios de mantenibilidad descritos por Sommerville, quien establece que la inteligibilidad del software es un requisito previo para su evolución.[5]

2.2 Metodologías de Documentación Ágil para No-Code

A diferencia de los enfoques tradicionales que abogan por extensos manuales en formato PDF que rápidamente quedan obsoletos, el ecosistema *No-Code* exige una documentación ágil, dinámica y contextual. La naturaleza visual de estas herramientas sugiere que los métodos de

documentación también deben aprovechar formatos visuales y auditivos para reducir la carga cognitiva del lector.

Nomenclatura Semántica y Auto-documentación

La primera capa de documentación ocurre dentro de la propia herramienta de automatización. La práctica de "Clean Code" (Código Limpio) debe traducirse a "Clean Flows". Esto implica el uso riguroso de una nomenclatura semántica para todos los componentes del sistema. Nombres genéricos generados por defecto, como "Módulo 1", "Webhook 3" o "Hoja 1", deben ser reemplazados sistemáticamente por identificadores descriptivos que expliquen la función y el contenido del componente. Ejemplos de buenas prácticas incluyen nombres como Trigger_Recepcion_Lead_Web, Iterador_Productos_Activos o Router_Tipo_Cliente.[4] Esta auto-documentación permite que la lectura del flujo visual revele la lógica de negocio sin necesidad de consultar documentos externos.

Micro-documentación Audiovisual

Dada la complejidad de las interacciones en interfaces gráficas, las descripciones textuales a menudo resultan insuficientes o excesivamente verbosas. La utilización de herramientas de grabación de pantalla, como Loom, para crear "micro-documentaciones" de 2 a 5 minutos ha demostrado ser altamente efectiva. Estos videos deben explicar no solo *cómo* funciona el flujo, sino *por qué* se tomaron ciertas decisiones de diseño, cuáles son las dependencias críticas y qué pasos manuales (si los hay) son necesarios.[4] Estos activos audiovisuales deben almacenarse en repositorios accesibles y vincularse directamente dentro de los escenarios de automatización mediante notas o comentarios.

Documentación Asistida por Inteligencia Artificial Generativa

La integración de Modelos de Lenguaje de Gran Escala (LLMs) ha revolucionado la capacidad de generar documentación técnica. Mediante técnicas de *Prompt Engineering*, es posible instruir a una IA para que actúe como documentalista técnica. Al proporcionar a la IA una descripción textual de la lógica del escenario, capturas de pantalla de los flujos (en modelos multimodales) o incluso el JSON de la estructura del escenario (disponible en herramientas como Make o Zapier), se pueden generar automáticamente manuales de usuario, diagramas de secuencia y descripciones de puntos de fallo potenciales.[4] Esta práctica no solo ahorra tiempo, sino que estandariza el formato de la documentación a través de la organización.

La siguiente tabla resume la transición recomendada desde prácticas documentales tradicionales hacia estrategias ágiles adaptadas al entorno No-Code:

Dimensión	Enfoque Tradicional (Legacy)	Enfoque Ágil (No-Code/Low-Code)
Formato Principal	Documentos de texto estáticos (DOCX, PDF) de gran extensión.	Videos explicativos cortos, wikis vivas (Notion, Confluence), anotaciones en la herramienta.
Actualización	Episódica, generalmente al final del proyecto; alta obsolescencia.	Continua, integrada en el ciclo de desarrollo; "Documentación como Código".
Nivel de Detalle	Exhaustivo, enfocado en especificaciones técnicas de bajo nivel.	Pragmático, enfocado en la lógica de negocio, resolución de problemas y dependencias.
Accesibilidad	Archivos almacenados en carpetas de red o sistemas de gestión documental separados.	Enlaces directos incrustados en el lienzo de trabajo (Canvas) de la herramienta de automatización.
Creación	Manual, intensiva en tiempo humano.	Híbrida, asistida por IA generativa para borradores y estructuración.

2.3 Listas de Verificación (Checklists) de Entrega

Para asegurar que ninguna automatización pase a un entorno productivo sin los estándares mínimos de mantenibilidad, se recomienda la implementación de una lista de verificación de entrega. Este mecanismo de control de calidad debe validar:

1. **Nomenclatura:** ¿Tienen todos los módulos, variables y archivos nombres descriptivos y consistentes?
2. **Explicación:** ¿Existe un video o documento breve que explique el propósito y funcionamiento del flujo?
3. **Credenciales:** ¿Están documentadas la ubicación y el propietario de las credenciales (sin exponer las claves sensibles)?
4. **Validación de Terceros:** ¿Ha probado el sistema alguien distinto al autor original para confirmar su inteligibilidad?.[4]

3. Gestión de Configuración y Estrategias de Versionado

El control de versiones es una disciplina central de la gestión de configuración del software, diseñada para mantener la integridad del sistema a lo largo del tiempo. En el desarrollo *No-Code*, la ausencia de archivos de texto plano que puedan ser gestionados por sistemas tradicionales como Git obliga a adoptar estrategias adaptadas a las capacidades de las plataformas SaaS.[6]

3.1 Separación de Entornos: Desarrollo vs. Producción

Uno de los riesgos más graves en la operación de automatizaciones es la modificación de flujos "en vivo" (Producción). Cualquier error introducido durante una edición en caliente puede tener consecuencias inmediatas, como la pérdida de datos, el envío de comunicaciones erróneas a clientes o la interrupción de servicios críticos. La metodología estándar debe ser estricta: **nunca se trabaja sobre el flujo que está en producción**.[4]

Se debe establecer un protocolo de copias de seguridad y entornos de prueba (*Staging*). Si se requiere implementar una mejora o corrección, el procedimiento correcto implica crear una copia del escenario actual (clonación), aplicar los cambios en este entorno seguro, realizar pruebas exhaustivas y, solo una vez validado, reemplazar la versión productiva o actualizarla replicando los cambios probados.

3.2 Nomenclatura de Versionado

Para mantener el orden y la trazabilidad, es vital adoptar una convención de nomenclatura de versiones clara. Se sugiere el uso de versionado semántico simplificado en los nombres de los archivos o escenarios:

- Proceso_Facturacion_v1.0_PROD (Versión estable, actualmente en uso).
- Proceso_Facturacion_v1.1_TEST (Versión en desarrollo, incorporando nuevas funcionalidades).
- Proceso_Facturacion_v1.0_BACKUP_20231027 (Copia de seguridad antes de cambios mayores).

Esta práctica permite identificar inequívocamente qué versión está operativa y facilita la recuperación ante desastres mediante la reversión a una versión anterior estable.[4]

3.3 "La Máquina del Tiempo": Historial de Versiones en Plataformas SaaS

Las plataformas líderes en el mercado *No-Code* han implementado funcionalidades nativas de historial de versiones que mitigan el riesgo de errores humanos destructivos.

- **Google Sheets:** Ofrece un "Historial de versiones" robusto (Accesible desde Archivo > Historial de versiones) que registra cada cambio realizado en la hoja de cálculo, identificando al usuario responsable y la marca de tiempo. Esta funcionalidad permite restaurar el documento completo a cualquier estado previo, actuando como una red de seguridad vital ante borrados accidentales de fórmulas o datos.[4]
- **Make (anteriormente Integromat):** En sus planes superiores, Make conserva un historial de versiones de los escenarios, permitiendo a los desarrolladores visualizar los cambios entre guardados y revertir a una configuración anterior si una actualización rompe la funcionalidad del flujo.
- **AppSheet:** Gestiona versiones de la aplicación cada vez que se guarda en el editor. Aunque la recuperación de versiones muy antiguas puede ser lenta debido al procesamiento de logs, es una herramienta esencial para la auditoría de cambios en la estructura de la app.[12]

3.4 Versionado de Prompts en Ingeniería de IA

Con la incorporación de la Inteligencia Artificial Generativa en los flujos de automatización, los *prompts* (instrucciones textuales) se convierten en componentes de software críticos. Un *prompt* no es un texto estático; es código en lenguaje natural que determina el comportamiento del sistema. Los modelos de IA (como GPT-4o, Claude 3.5, etc.) evolucionan, y un mismo *prompt* puede arrojar resultados diferentes en versiones distintas del modelo o tras actualizaciones opacas del proveedor.[9]

Por ello, se recomienda tratar los *prompts* con el mismo rigor que el código fuente. Se debe

mantener un "Documento Maestro de Prompts" o un repositorio centralizado que registre:

- **El Prompt exacto:** La cadena de texto completa, incluyendo variables y delimitadores.
- **Modelo Probado:** La versión específica del modelo con la que se validó (ej. gpt-4-0613).
- **Configuración:** Parámetros como temperatura, *top_p*, etc.
- **Fecha de Última Prueba:** Cuándo se verificó su funcionamiento.
- **Resultado Esperado (Golden Sample):** Un ejemplo de la salida correcta para usar como referencia en pruebas de regresión.[4]

La investigación sugiere que el *Prompt Engineering* se está moviendo hacia un enfoque de ingeniería dirigida por modelos, donde los prompts son versionados, probados y encadenados sistemáticamente para garantizar la determinabilidad de las respuestas de la IA.[13]

4. Confiabilidad del Sistema: Monitoreo y Gestión de Errores

La confiabilidad de un sistema de automatización no se define por la ausencia total de errores, sino por su resiliencia: la capacidad de detectar fallos, evitar que estos colapsen todo el proceso y notificar a los responsables para una pronta resolución. En arquitecturas *No-Code*, que dependen fuertemente de la orquestación de múltiples APIs de terceros, los errores son inevitables debido a factores exógenos como latencia de red, tiempos de inactividad de servicios o cambios en las políticas de seguridad.[4]

4.1 Taxonomía de Fallas en Automatización

Para diseñar estrategias de manejo de errores (*Error Handling*) efectivas, es necesario categorizar los tipos de fallas más comunes según su origen y comportamiento:

1. **Fallas de Autenticación y Autorización (Error 401/403):** Ocurren cuando las credenciales de conexión (tokens OAuth, API Keys) expiran, son revocadas o se cambian las contraseñas de las cuentas vinculadas. Estas fallas son críticas y detienen la ejecución inmediatamente, ya que el sistema pierde el permiso para interactuar con el servicio externo.[4]
2. **Fallas de Validación de Datos (Error 400):** Suceden cuando los datos enviados a un servicio no cumplen con el formato o esquema esperado. Ejemplos comunes incluyen enviar texto en un campo que requiere números, formatos de fecha inválidos (ej. "Mañana" en lugar de "2023-10-27") o direcciones de correo electrónico mal formadas. Estos errores indican problemas en la calidad de los datos de entrada o en la lógica de

transformación previa.[4]

3. **Fallas de Límite y Cuotas (Error 429):** Las APIs imponen límites de tasa (*Rate Limits*) para prevenir abusos. Si una automatización realiza demasiadas solicitudes en un periodo corto, el servicio responderá con un error 429 ("Too Many Requests"). Asimismo, servicios como Google Sheets tienen cuotas estrictas de lectura/escritura por minuto y por día.[17]
4. **Fallas de Infraestructura (Error 5xx):** Son errores del lado del servidor del proveedor (ej. OpenAI está caído, Google Drive no responde). Generalmente son transitorios y se pueden mitigar con estrategias de reintento (*retry policies*).[16]
5. **Fallas Silenciosas (Lógica de Negocio):** Son las más peligrosas y difíciles de detectar. El sistema se ejecuta técnicamente sin errores (código 200 OK), pero no realiza la acción deseada debido a una lógica defectuosa. Por ejemplo, un filtro mal configurado que descarta todos los registros entrantes, o un bot que cree haber enviado un mensaje pero la API no completó la entrega final. Estas fallas requieren monitoreo de resultados, no solo de ejecución.[4]

4.2 Estrategias de Manejo de Errores (Error Handling)

El manejo de errores proactivo implica diseñar "rutas de escape" o flujos alternativos dentro de la automatización para gestionar las excepciones de manera controlada. Las plataformas avanzadas como Make ofrecen directivas específicas para esto [19]:

- **Directiva IGNORE:** El error se ignora y el escenario continúa (o finaliza esa operación) como si nada hubiera pasado. Útil para errores no críticos donde la pérdida de un dato no afecta el resultado global.
- **Directiva BREAK:** Permite manejar errores transitorios (como fallos de conexión o timeouts). La plataforma almacena la entrada que falló y reintenta la ejecución automáticamente después de un intervalo de tiempo creciente. Es fundamental para lidiar con inestabilidades temporales de las APIs sin intervención manual.
- **Directiva ROLLBACK:** Detiene la ejecución inmediatamente y revierte las operaciones anteriores si es posible (aunque en APIs REST esto suele requerir lógica compensatoria manual).
- **Rutas de Manejo de Errores (Error Handler Route):** Se pueden añadir rutas específicas conectadas a un módulo que se activan solo cuando ocurre un error. Una práctica recomendada es configurar esta ruta para capturar los detalles del error y guardarlos en una base de datos de "Logs de Error" o enviar una notificación, transformando un fallo fatal en una tarea administrativa pendiente.[4]

Ejemplo Práctico de Resiliencia:
Si un módulo de envío de correo electrónico falla, en lugar de detener todo el proceso de

venta, el sistema debe derivar a una ruta de error que registre el incidente en una hoja de cálculo ("Envíos Fallidos") y envíe una alerta al equipo de soporte. De esta manera, el proceso principal no se rompe y el cliente puede ser contactado manualmente luego.4

4.3 Monitoreo Activo y Alertas

La visibilidad del estado del sistema es crucial. Los administradores deben enterarse de los errores antes que los usuarios finales.

- **Make:** Ofrece notificaciones por correo electrónico nativas ante desactivaciones de escenarios por errores acumulados. Sin embargo, se recomienda configurar alertas personalizadas (vía Email, Slack, Telegram o WhatsApp) dentro de las rutas de manejo de errores para tener visibilidad inmediata de fallos en procesos críticos de negocio.[19]
- **Zapier:** Proporciona notificaciones de fallos en los "Zaps". Recientemente, ha mejorado su gestión de errores introduciendo el estado "Held" (Retenido) para tareas que fallan por problemas de autenticación o límites, permitiendo su reejecución (*replay*) masiva una vez resuelto el problema, evitando la pérdida de datos.[22]
- **AppSheet:** Permite configurar alertas automáticas al creador de la aplicación cuando se registran errores en el *Audit History*, facilitando la detección proactiva de problemas de sincronización en los dispositivos de los usuarios.[23]

4.4 Auditoría y Trazabilidad (Logging)

Para el análisis forense y la mejora continua, es indispensable mantener un registro detallado de la actividad del sistema.

- **Historial de Ejecuciones:** Plataformas como Zapier y Make mantienen logs detallados de cada ejecución (*Zap History*, *Scenario History*), mostrando los datos de entrada y salida de cada módulo. Es vital conocer las políticas de retención de estos datos: Zapier, por ejemplo, retiene el historial entre 29 y 69 días dependiendo de la fecha y el plan, mientras que los planes empresariales permiten personalizar esta retención.[24]
- **AppSheet Audit History:** Registra cada sincronización, adición, edición o eliminación de datos, así como la invocación de bots y APIs. Debido al volumen de datos, la recuperación de estos logs puede tener latencia, pero es la fuente de verdad definitiva para entender el comportamiento de la aplicación.[23]
- **Logs Personalizados:** Dado que los historiales nativos tienen caducidad, se recomienda crear un sistema de *logging* propio (por ejemplo, enviando resúmenes de transacciones a BigQuery o una base de datos SQL) para garantizar la trazabilidad a largo plazo y cumplir con normativas de auditoría empresarial.[23]

5. Escalabilidad y Arquitectura de Datos: Preparándose para el Crecimiento

La escalabilidad es la capacidad del sistema para manejar un incremento en la carga de trabajo (usuarios, transacciones, datos) sin comprometer el rendimiento ni incurrir en costos prohibitivos. En el desarrollo *No-Code*, la escalabilidad presenta desafíos únicos debido a las limitaciones físicas de las herramientas SaaS y las estructuras de datos subyacentes.[4]

5.1 El Techo de Cristal de las Hojas de Cálculo

Google Sheets y Excel son herramientas omnipresentes y excelentes para el prototipado rápido (MVP) y la gestión de volúmenes bajos de datos. Sin embargo, su uso como base de datos transaccional (*backend*) es una de las principales barreras para la escalabilidad.

Aunque Google ha aumentado el límite técnico de Google Sheets a 10 millones de celdas 11, esto no implica que el rendimiento sea aceptable al acercarse a ese número. El rendimiento de una hoja de cálculo se degrada no solo por la cantidad de filas, sino por la complejidad de las fórmulas (especialmente funciones volátiles como AHORA() o GOOGLEFINANCE), el número de columnas y las interdependencias entre hojas.28
Los síntomas de colapso incluyen tiempos de carga superiores a 10 segundos, errores de "timeout" en las conexiones API (error 504 Gateway Timeout), duplicación de datos debido a condiciones de carrera cuando múltiples usuarios escriben simultáneamente y errores de cálculo ("Loading...") que nunca se resuelven.4
Recomendación Técnica: Se establece un límite práctico operativo de entre 10,000 y 20,000 filas para mantener un rendimiento óptimo en automatizaciones *No-Code*. Superado este volumen, o cuando se requiere concurrencia real, la migración es obligatoria.[4] Además, se recomienda eliminar periódicamente filas vacías y celdas no utilizadas, ya que estas cuentan para el límite total de 10 millones de celdas y consumen memoria de procesamiento.[30]

5.2 Estrategias de Migración de Datos

La arquitectura de datos en proyectos *No-Code* debe evolucionar escalonadamente para acompañar el crecimiento del negocio:

1. **Nivel 1 (Prototipado / Baja Escala):** Hojas de cálculo (Google Sheets, Excel Online). Ideal para validación de ideas, uso personal o equipos muy pequeños. Costo cero o muy bajo.
2. **Nivel 2 (Media Escala / Colaborativo):** Bases de datos "User-Friendly" o relacionales ligeras. Herramientas como **Airtable**, **AppSheet Databases** o **SmartSuite**. Estas

plataformas ofrecen integridad referencial (relaciones reales entre tablas), vistas personalizadas, tipos de datos estrictos y mejores capacidades de API que las hojas de cálculo, soportando cientos de miles de registros con buen rendimiento.[4]

3. **Nivel 3 (Alta Escala / Empresarial):** Bases de datos profesionales y Data Warehouses. Soluciones como **Google BigQuery**, **PostgreSQL**, **MySQL (MariaDB)**, **MongoDB** o **Xano**. Necesarias para manejar millones de registros, consultas SQL complejas, alta seguridad y concurrencia masiva. Las herramientas de automatización como Make y AppSheet tienen conectores nativos para estos motores, permitiendo que el *front-end* siga siendo *No-Code* mientras el *backend* es robusto y escalable.[4]

5.3 Gestión de Datos Históricos

Una estrategia clave para mantener el rendimiento sin eliminar información es la segmentación de datos históricos. No es eficiente mantener registros de años anteriores en la base de datos operativa "caliente". Se recomienda implementar procesos de archivado automático: mover datos antiguos (ej. ventas del año 2023) a un archivo de "Histórico" o un Data Warehouse, dejando en la base activa solo los datos del periodo corriente necesario para la operación diaria. Esto reduce la carga sobre las consultas y mejora la velocidad de la aplicación.[4]

5.4 La Economía de la Escalabilidad (Unit Economics)

La escalabilidad no es solo técnica, sino financiera. Las automatizaciones que dependen de APIs con modelos de precios por uso (Pay-as-you-go) pueden volverse económicamente insostenibles al escalar. El caso más paradigmático son los modelos de IA como los de OpenAI.
El costo se calcula por "tokens" (fragmentos de palabras). Una interacción simple puede costar fracciones de centavo, pero si se escala a miles de usuarios o ejecuciones diarias, el costo mensual puede dispararse.

- **Cálculo de Proyección:** Si una consulta cuesta $0.01 USD y se proyectan 100 consultas diarias, el costo mensual es de $30 USD. Si se escala a 1,000 usuarios, el costo salta a $300 USD o más, dependiendo de la complejidad del prompt.[4]
- **Optimización:** Es vital estimar estos costos antes de escalar e implementar técnicas de optimización, como el uso de modelos más baratos para tareas simples (ej. GPT-4o-mini en lugar de GPT-4o), el almacenamiento en caché de respuestas frecuentes (*Caching*) para evitar llamadas repetitivas a la API, y la limpieza de *prompts* para reducir el consumo de tokens de entrada.[32]

5.5 Transición a "Pro-Code"

Existe un punto de inflexión donde las plataformas *No-Code* pueden dejar de ser la solución más eficiente. Si un sistema se convierte en el núcleo crítico del negocio (*Core Business*), maneja transacciones de alto volumen o requiere lógica algorítmica extremadamente compleja, el desarrollo de software a medida ("Pro-Code") puede ofrecer mejor rendimiento, control total y menores costos operativos a gran escala. El *No-Code* es excelente para validar, iterar y operar hasta niveles significativos de madurez, pero la ingeniería de software tradicional escala mejor en escenarios de hiper-crecimiento. La decisión de migrar debe basarse en un análisis de costo-beneficio y deuda técnica.[4]

6. Gobernanza: El Reto del Citizen Developer en Latinoamérica

La proliferación de herramientas *No-Code* ha empoderado a los empleados no técnicos para resolver sus propios problemas, pero también ha generado desafíos de gobernanza. En Latinoamérica, a menudo caracterizada por estructuras jerárquicas y departamentos de TI con recursos limitados, el desarrollo ciudadano surge muchas veces como una respuesta a la necesidad de agilidad que la burocracia corporativa no puede satisfacer.[34]

6.1 Shadow IT vs. Desarrollo Gestionado

El fenómeno de Shadow IT (TI en la sombra) se refiere al uso de dispositivos, software y aplicaciones dentro de una organización sin el conocimiento o aprobación explícita del departamento de TI. Esto introduce riesgos severos: fugas de datos, brechas de seguridad, falta de cumplimiento normativo y silos de información incomunicados.7
La respuesta moderna no es prohibir el desarrollo ciudadano, sino gobernarlo. Las organizaciones deben transicionar del Shadow IT al Citizen Development gestionado. Esto implica establecer un marco de trabajo donde TI provee las herramientas aprobadas, la capacitación y las barreras de seguridad (guardrails), mientras que las unidades de negocio construyen las soluciones.8

6.2 Centros de Excelencia (CoE)

La literatura académica y las mejores prácticas de la industria recomiendan la creación de un "Centro de Excelencia" (CoE) para la automatización. Este órgano interno tiene la función de:

- Definir estándares de desarrollo y documentación.
- Evaluar y aprobar herramientas *No-Code* seguras.
- Proveer soporte y mentoría a los desarrolladores ciudadanos.
- Auditar las aplicaciones críticas para asegurar que cumplen con las políticas de seguridad

y privacidad de datos.[38]

6.3 Responsabilidad Ética y Continuidad del Negocio

El desarrollo de automatizaciones conlleva una responsabilidad ética. Automatizar un proceso defectuoso o sesgado solo amplifica el impacto negativo a una velocidad mayor. Es responsabilidad del desarrollador asegurar que los flujos diseñados respeten la privacidad de los datos, no introduzcan sesgos algorítmicos y mantengan la transparencia.40
Además, siempre debe existir un "Plan de Continuidad de Negocio" (BCP). La tecnología puede fallar; si OpenAI se cae o una API crítica deja de responder, el negocio no puede detenerse. Debe existir un proceso manual documentado que permita a la organización seguir operando, aunque sea a menor velocidad, durante la contingencia.4

7. Conclusión

La sostenibilidad de las automatizaciones *No-Code* no es un accidente, sino el resultado de un diseño deliberado que integra principios de ingeniería de software adaptados a un entorno visual y ágil. La documentación efectiva, el versionado riguroso, el monitoreo proactivo de errores y una planificación arquitectónica para la escalabilidad son los pilares que diferencian un script frágil de una solución empresarial robusta.

Para el profesional en formación, el dominio de estas competencias marca la transición de ser un usuario de herramientas a ser un arquitecto de soluciones digitales. En el contexto de América Latina, donde la agilidad y la eficiencia son vitales, la capacidad de construir sistemas mantenibles y escalables posiciona al *Citizen Developer* como un actor clave en la transformación digital de la región. El futuro no es solo *No-Code* o solo Código, sino una síntesis híbrida gobernada por buenas prácticas, rigor técnico y visión estratégica.

Tabla de Referencias y Recursos Citados

Recurso / Autor	Título / Tema	Relevancia para la Clase
Ablebits (2023)	*Google Docs & Google Sheets limits* [29]	Detalles técnicos sobre límites de rendimiento en

		hojas de cálculo.
AppSheet Help	*Monitor app activity using Audit History* [23]	Documentación oficial sobre auditoría y monitoreo en AppSheet.
Automation Anywhere	*¿Qué es la automatización robótica de procesos (RPA)?* [41]	Conceptos fundamentales de RPA y ciclo de vida.
Calaméo / Pressman	*Ingeniería de Software, Roger S. Pressman* [10]	Fundamentos teóricos de ingeniería de software y mantenimiento.
Google Developers	*Google Sheets API Limits* [18]	Especificaciones técnicas de cuotas de API para escalabilidad.
Google Workspace	*Google Sheets doubles cell limit* [11]	Actualizaciones oficiales sobre capacidades de almacenamiento de datos.
Kissflow	*How Citizen Development Help Combat Shadow IT* [35]	Relación entre Shadow IT y desarrollo ciudadano gestionado.
Make Help Center	*Error handling & Introduction to errors* [19]	Guías oficiales para la implementación de manejo de errores en Make.
OpenAI	*API Pricing & Managing Costs* [32]	Modelos de costos para escalar soluciones con IA generativa.

ProcessMaker (2021)	*Establishing Governance for the Citizen Developer* [37]	Estrategias de gobernanza organizacional.
Quixy	*Making Shadow IT a Frenemy with Citizen Development* [7]	Gestión de riesgos de Shadow IT.
Row Zero (2025)	*Google Sheets Limits* [30]	Análisis comparativo de límites de hojas de cálculo y alternativas.
Salesforce	*RPA: Qué es y cómo optimizar la automatización* [42]	Visión empresarial de la optimización de procesos.
ServiceNow	*What is a Citizen Developer?* [43]	Definición y contexto del rol del desarrollador ciudadano.
Sheetgo	*Google Sheets cell limit* [27]	Implicaciones prácticas de los límites de datos.
Sommerville, I. (2011)	*Software Engineering (9th ed.)* [5]	Libro de texto fundamental sobre ingeniería y mantenimiento de software.
UTN.BA	*Clase 15 - Mantenimiento y Escalabilidad* [4]	Material base de la cátedra y conceptos nucleares de la clase.
Zapier	*Data Retention, History & Limits* [24]	Políticas de retención de datos y manejo de historial en Zapier.

Works cited

1. What is Automation? - ServiceNow, accessed December 27, 2025, https://www.servicenow.com/latam/platform/what-is-automation.html
2. Plataforma low-code para el desarrollo ágil de aplicaciones | ManageEngine, accessed December 27, 2025, https://www.manageengine.com/latam/herramientas-desarrollo-aplicaciones-con-low-code.html
3. ¿Qué es un citizen developer? La democratización de la tecnología - ManageEngine Blog, accessed December 27, 2025, https://www.manageengine.com/latam/blog/general/que-es-citizen-developer-democratizacion-de-la-tecnologia.html
4. NO PRO - Clase 15 - Mantenimiento y Escalabilidad
5. Ingenieria de Software - Catálogo de recursos SCALA, accessed December 27, 2025, https://gc.scalahed.com/recursos/files/r161r/w25469w/ingdelsoftwarelibro9_compressed.pdf
6. Ingeniería de Software: Enfoque del Practicante - 9na Edición - Studylib, accessed December 27, 2025, https://studylib.net/doc/27891445/ingenieria-de-software-roger-s.-pressman--bruce-r.-maxim-...
7. 5 Steps to Tackle Shadow IT : Unleash its Potential as a Citizen Developer | Quixy, accessed December 27, 2025, https://quixy.com/blog/making-shadow-it-a-frenemy-with-citizen-development/
8. Governing Citizen Development to Address Low-Code Platform Challenges - AIS eLibrary, accessed December 27, 2025, https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1601&context=misqe
9. Unlocking the Potential of the Prompt Engineering Paradigm in Software Engineering: A Systematic Literature Review - MDPI, accessed December 27, 2025, https://www.mdpi.com/2673-2688/6/9/206
10. Ingeniería de Software, Roger S. Pressman - Calaméo, accessed December 27, 2025, https://www.calameo.com/books/000409775b2733e206ae1
11. Google Sheets doubles cell limit, accessed December 27, 2025, https://workspaceupdates.googleblog.com/2022/03/ten-million-cells-google-sheets.html
12. Audit History performance - AppSheet Help, accessed December 27, 2025, https://support.google.com/appsheet/answer/10104807?hl=en
13. Model-Driven Prompt Engineering - IEEE Xplore, accessed December 27, 2025, https://ieeexplore.ieee.org/document/10343974/
14. The Prompt Report: A Systematic Survey of Prompt Engineering Techniques -

arXiv, accessed December 27, 2025, https://arxiv.org/abs/2406.06608
15. Best practices for prompt engineering with the OpenAI API, accessed December 27, 2025, https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api
16. Error handling | Custom Apps Documentation - Make Developer Hub, accessed December 27, 2025, https://developers.make.com/custom-apps-documentation/app-components/base/error-handling
17. Google Sheets | Hightouch Docs, accessed December 27, 2025, https://hightouch.com/docs/destinations/google-sheets
18. Usage limits | Google Sheets, accessed December 27, 2025, https://developers.google.com/workspace/sheets/api/limits
19. Introduction to errors and warnings - Help Center - Make.com Help, accessed December 27, 2025, https://help.make.com/introduction-to-errors-and-warnings
20. Overview of error handling - Help Center, accessed December 27, 2025, https://help.make.com/overview-of-error-handling
21. Error handlers - Help Center - Make.com Help, accessed December 27, 2025, https://help.make.com/error-handlers
22. New: Don't Lose Any Data With This Safety Net for Your Zaps - Updates - Zapier, accessed December 27, 2025, https://zapier.com/blog/updates/1478/new-dont-lose-any-data-new-held-tasks
23. Monitor app activity using Audit History - AppSheet Help, accessed December 27, 2025, https://support.google.com/appsheet/answer/10104794?hl=en
24. Customize data retention in Zapier, accessed December 27, 2025, https://help.zapier.com/hc/en-us/articles/8496327478413-Customize-data-retention-in-Zapier
25. Data Privacy Overview | Zapier, accessed December 27, 2025, https://zapier.com/legal/data-privacy
26. Scalability and Performance Limitations of Low-Code and No-Code Platforms for Large-Scale Enterprise Applications and Solutions, accessed December 27, 2025, https://www.ijetcsit.org/index.php/ijetcsit/article/view/368
27. How to solve the 10 million cell limit in Google Sheets - Sheetgo, accessed December 27, 2025, https://www.sheetgo.com/blog/spreadsheets-tips/google-sheets-cell-limit/
28. What Google Sheets' 10 million cell limit means for you - Zapier, accessed December 27, 2025, https://zapier.com/blog/google-sheets-cell-limit/
29. Google Docs and Google Sheets limits – all in one place - Ablebits.com, accessed December 27, 2025, https://www.ablebits.com/office-addins-blog/google-sheets-limits/

30. Google Sheets Limits - Max row, column, and file size limits - Row Zero, accessed December 27, 2025, https://rowzero.com/blog/google-sheets-limits
31. The Google Sheets row limit? 10 million cells - Row Zero, accessed December 27, 2025, https://rowzero.com/blog/what-is-the-google-sheets-row-limit
32. API Pricing - OpenAI, accessed December 27, 2025, https://openai.com/api/pricing/
33. Managing costs | OpenAI API, accessed December 27, 2025, https://platform.openai.com/docs/guides/realtime-costs
34. Full article: Introduction: Looking for Governance: Latin America Governance Reforms and Challenges - Taylor & Francis Online, accessed December 27, 2025, https://www.tandfonline.com/doi/full/10.1080/01900692.2021.2020905
35. How citizen development help organizations combat shadow IT? - Kissflow, accessed December 27, 2025, https://kissflow.com/citizen-development/how-citizen-development-help-combat-shadow-it/
36. The Risks Of Shadow IT For Businesses - Forbes, accessed December 27, 2025, https://www.forbes.com/councils/forbestechcouncil/2022/12/26/the-risks-of-shadow-it-for-businesses/
37. Establecer la gobernanza para el ciudadano desarrollador - ProcessMaker, accessed December 27, 2025, https://www.processmaker.com/es/blog/establishing-governance-for-the-citizen-developer/
38. Is Citizen Development Shadow IT? - Quandary Consulting Group, accessed December 27, 2025, https://www.quandarycg.com/citizen-development-shadow-it/
39. Studio - Ciclo de vida de la automatización, accessed December 27, 2025, https://docs.uipath.com/es/studio/standalone/2020.10/user-guide/automation-lifecycle
40. Ingeniería del Software, accessed December 27, 2025, http://www.vc.ehu.es/jiwotvim/IngenieriaSoftware/Teoria/BloqueI/Transp-01IngSw-Pressman.pdf
41. ¿Qué es la automatización robótica de procesos (RPA)? - Automation Anywhere, accessed December 27, 2025, https://www.automationanywhere.com/la/rpa/robotic-process-automation
42. RPA: Qué es y cómo optimizar la automatización de operaciones - Salesforce, accessed December 27, 2025, https://www.salesforce.com/mx/blog/rpa/
43. ¿Qué es un desarrollador ciudadano? - ServiceNow, accessed December 27, 2025, https://www.servicenow.com/latam/workflows/creator-workflows/what-is-a-citizen-developer.html

44. ¿Qué es el desarrollo ciudadano? - ServiceNow, accessed December 27, 2025, https://www.servicenow.com/es/workflows/creator-workflows/what-is-a-citizen-developer.html
45. Data Retention/Deletion/Export - Zapier, accessed December 27, 2025, https://zapier.com/legal/data-retention-deletion