

# Guía de Ayuda — Segunda Entrega (Avance 2)

**Asignatura:** Taller de Proyecto de la Especialidad  
**Carrera:** Técnico en Programación y Análisis de Sistemas

Docente: Víctor M. Valderrama M.  
Semestre/Año: 2025-2  
Versión: 1.1 (AIEP)

# Índice

<b>1. Unidad 1: Requerimientos y su traducción al diseño</b>	<b>4</b>
1.1. Objetivo . . . . .	4
1.2. Conceptos y clasificación . . . . .	4
1.3. Redacción moderna de requerimientos . . . . .	5
1.4. Calidad y trazabilidad . . . . .	6
1.5. Checklist de validación . . . . .	6
1.6. Actividad práctica . . . . .	6
1.7. Rúbrica de validación de requerimientos . . . . .	7
1.8. Anexos . . . . .	8
<b>2. Unidad 2: Métodos modernos de desarrollo individual</b>	<b>10</b>
2.1. Comparativa de métodos ágiles actuales . . . . .	10
2.2. Metodologías y técnicas combinadas . . . . .	11
2.3. Justificación metodológica . . . . .	11
2.4. Presentación visual del método . . . . .	12
<b>3. Unidad 3: Diseño de la Base de Datos</b>	<b>14</b>
3.1. De requerimientos al modelo conceptual . . . . .	14
3.2. Normalización y calidad del modelo . . . . .	16
3.3. Transformación a modelo lógico . . . . .	17
3.4. Buenas prácticas y errores comunes . . . . .	20
3.5. Integración con el proyecto . . . . .	21
3.6. Actividad guiada de integración . . . . .	22
<b>4. Unidad 4: Diseño funcional y experiencia del usuario</b>	<b>23</b>
4.1. Del análisis al diseño funcional . . . . .	23
4.2. Diagramas de flujo de procesos y de actividad . . . . .	25
4.3. Diseño de experiencia del usuario (UX/UI) con wireframes . . . . .	28
4.4. Integración visual y consistencia . . . . .	31
4.5. Documentación en el Informe 2 . . . . .	32
4.6. Actividad guiada final . . . . .	33
<b>5. Unidad 5: Gestión técnica del repositorio (Git y GitHub)</b>	<b>34</b>
5.1. Instalación y configuración inicial . . . . .	34
5.2. Crear proyecto local y subirlo a GitHub . . . . .	35
5.3. Flujo básico de trabajo (local → remoto) . . . . .	36
5.4. Cuadro resumen de comandos esenciales . . . . .	36
5.5. Ramas (branches) y versiones etiquetadas (tags) . . . . .	37
5.6. Estructura profesional del repositorio . . . . .	37
5.7. Cuándo hacer commit, push o crear una versión . . . . .	38
5.8. Actividad guiada . . . . .	38
5.9. Conceptos clave de control de versiones . . . . .	40
5.10. Trabajo práctico con ramas (branches) . . . . .	40
5.11. Versiones con tags y publicaciones (releases) . . . . .	41
5.12. Cuadro resumen: acciones, propósito y riesgos . . . . .	42

5.13. Recuperación de errores: restore, reset y revert . . . . .	42
5.14. Guía rápida de comandos avanzados (resumen) . . . . .	43
5.15. Actividad guiada . . . . .	44
5.16. Organización profesional del repositorio en GitHub . . . . .	45
5.17. Archivo README.md: identidad del proyecto . . . . .	46
5.18. Uso de GitHub Projects (Kanban) . . . . .	47
5.19. Cuadro resumen: organización y seguimiento . . . . .	48
5.20. Actividad guiada final . . . . .	48
<b>6. Unidad 6: Integración del Informe de Análisis y Diseño</b>	<b>49</b>
6.1. Estructura general del informe . . . . .	49
6.2. Redacción técnica profesional . . . . .	50
6.3. Integración de evidencias . . . . .	51
6.4. Control de calidad del informe . . . . .	51
6.5. Entrega formal y presentación final . . . . .	52
6.6. Complementos visuales y checklist técnico . . . . .	53
6.7. Control de versiones y evidencia GitHub . . . . .	54
<b>7. Unidad 7: Reflexión, buenas prácticas y proyección al desarrollo</b>	<b>56</b>
7.1. Reflexión sobre el proceso . . . . .	56
7.2. Buenas prácticas técnicas y éticas . . . . .	56
7.3. Preparación para el desarrollo (Avance 3) . . . . .	57
7.4. Errores frecuentes y lecciones aprendidas . . . . .	58
7.5. Cierre y proyección profesional . . . . .	58

# Introducción

## Información Importante

Esta guía tiene como objetivo orientar al estudiante en la elaboración de la segunda entrega del Taller de Proyecto de la Especialidad: el **Informe de Análisis y Diseño**.

En esta etapa, el alumno transforma el análisis conceptual del problema en un diseño técnico completo, estableciendo la base para el desarrollo del prototipo funcional.

## Atención

La entrega 2 representa la mitad del trabajo total del proyecto. **Un diseño bien estructurado reduce errores, tiempos de desarrollo y asegura coherencia con los objetivos iniciales.**

# 1. Unidad 1: Requerimientos y su traducción al diseño

## 1.1. Objetivo

### Información Importante

Aprender a transformar las necesidades del usuario en **requerimientos claros, medibles y trazables**, que sirvan como base sólida para el **diseño del sistema** y luego para el desarrollo del prototipo.

## 1.2. Conceptos y clasificación

### Información Importante

**Requerimiento Funcional (RF)**: acción observable que el sistema debe realizar (p. ej., *crear, leer, actualizar, eliminar, calcular, emitir*).

**Requerimiento No Funcional (RNF)**: condición de calidad o restricción: *rendimiento, seguridad, disponibilidad, compatibilidad, usabilidad*.

Relación con los **objetivos SMART**<sup>a</sup>: un buen RF/RNF debe derivarse de objetivos SMART para evitar ambigüedad y sobrediseño.

<sup>a</sup>SMART: Específico, Medible, Alcanzable, Relevante y con Tiempo definido.

### Ejemplo Correcto

#### Ejemplos correctos

- **RF**: El sistema *permite registrar* un producto con código, nombre, precio y stock.
- **RF**: El sistema *emite* un reporte PDF de ventas por rango de fechas.
- **RNF**: El listado de productos *debe visualizarse en < 2 s* con 1.000 registros en hardware de laboratorio.

### Error Común

#### Qué evitar

- Ambigüedad: “El sistema será rápido y moderno”. (no es verificable; mezcla RF y RNF)
- Solapamiento: “Gestionar productos y productos del almacén” (duplicación conceptual).
- Omisiones: “Registrar producto” *sin* campos ni validaciones mínimas.

## Refuerzo

**Método aplicado:** se usarán **Design Thinking** (fase de Definición) y **User Story Mapping** para capturar necesidades desde la perspectiva del usuario y traducirlas a RF/RNF concretos.

### 1.3. Redacción moderna de requerimientos

#### Información Importante

Plantilla de **historia de usuario**: “*Como [rol], quiero [función] para [beneficio]*”. A partir de ella, deriva un **RF medible** y su **RNF** asociado con **criterios de aceptación** (*Definition of Done*).

#### Ejemplo Correcto

##### Ejemplo completo

- Historia: “Como cajero, quiero registrar cada venta para mantener actualizado el inventario”.
- RF: “El sistema *registra* una venta con fecha, productos (id, cantidad, precio) y monto” .
- RNF: “El registro *se confirma en*  $\leq 3$  s y valida stock disponible” .
- Criterios de aceptación: “Dado un producto con stock, cuando se registra la venta entonces el stock disminuye y se genera id de comprobante” .

#### Error Común

##### Conversión defectuosa

- Historia genérica: “Como usuario, quiero que sea fácil”. (no genera RF medible)
- RF incompleto: “Registrar venta”. (faltan campos, validaciones y condición de aceptación)

## Refuerzo

**Método aplicado:** integración de **Agile Modeling** con **Acceptance Criteria Driven Design** para que cada RF tenga evidencia verificable.

## 1.4. Calidad y trazabilidad

### Información Importante

**Criterios de calidad:** claridad, medibilidad, consistencia, verificabilidad.

**Trazabilidad:** vínculo RF → Caso de Uso → Módulo (y luego a pruebas). Esta práctica corresponde a la **Requirements Traceability Matrix (RTM)** simplificada, alineada con ISO/IEC 29148 e IEEE 830 (adaptadas al nivel técnico del taller).

ID RF	Descripción (resumen)	Caso de Uso	Módulo
RF-01	Registrar productos con código, nombre y precio	CU–Registrar Producto	Inventario — Alta
RF-02	Listar productos por categoría y búsqueda	CU–Consultar Inventario	Inventario — Consulta
RF-03	Actualizar stock tras una venta	CU–Actualizar Stock	Ventas — Sincronización

Cuadro 1: Matriz de trazabilidad RF → Caso de Uso → Módulo.

### Refuerzo

**Técnica usada:** *Requirements Traceability Matrix (RTM)* para detectar RF huérfanos (sin caso de uso) o funciones no justificadas.

## 1.5. Checklist de validación

- ¿Cada RF tiene **verbo de acción** y **datos** involucrados?
- ¿Existe **criterio de aceptación** verificable (tiempo, formato, regla de negocio)?
- ¿Está vinculado a un **caso de uso** y a un **módulo** (trazabilidad)?
- ¿Evita ambigüedad (términos vagos) y solapamiento con otros RF?

## 1.6. Actividad práctica

### Refuerzo

#### Tu tarea para el Informe 2

1. Redacta **3 RF** y **1 RNF** con criterios medibles (usa historias de usuario si ayuda).
2. Deriva **criterios de aceptación** para al menos **2 RF**.
3. Completa la **matriz de trazabilidad** con 3 filas (RF → Caso de Uso → Módulo) y súbelas en Markdown a GitHub.

## Atención

Si un requerimiento *no puede probarse*, no está listo. Ajusta redacción o añade criterios de aceptación hasta que sea verificable.

## 1.7. Rúbrica de validación de requerimientos

### Información Importante

Esta rúbrica te permitirá autoevaluar tus requerimientos antes de entregarlos. Cada criterio se evalúa con una escala de 0 a 4 puntos.

**Objetivo:** garantizar que tus requerimientos sean claros, verificables, trazables y coherentes con el propósito del sistema.

Criterio	Descripción del logro esperado	Puntaje (0–4)
Claridad	Redacción precisa, sin ambigüedad ni tecnicismos innecesarios.	
Verificabilidad	Puede ser probado mediante una acción observable o un resultado medible.	
Trazabilidad	Se vincula a un Caso de Uso y a un Módulo concreto.	
Consistencia	No contradice otros requerimientos; mantiene coherencia con los objetivos SMART.	
Completitud	Incluye datos, condiciones y restricciones esenciales.	
<b>Total sugerido</b>		/20

Cuadro 2: Rúbrica de validación de requerimientos.

### Refuerzo

Un resultado de 16/20 o más indica que tus requerimientos están bien definidos para pasar a la etapa de diseño.

## 1.8. Anexos

### Información Importante

Estos diagramas complementan la comprensión del proceso de análisis y diseño de requerimientos, relacionando la creatividad de la fase conceptual con la rigurosidad técnica del desarrollo.

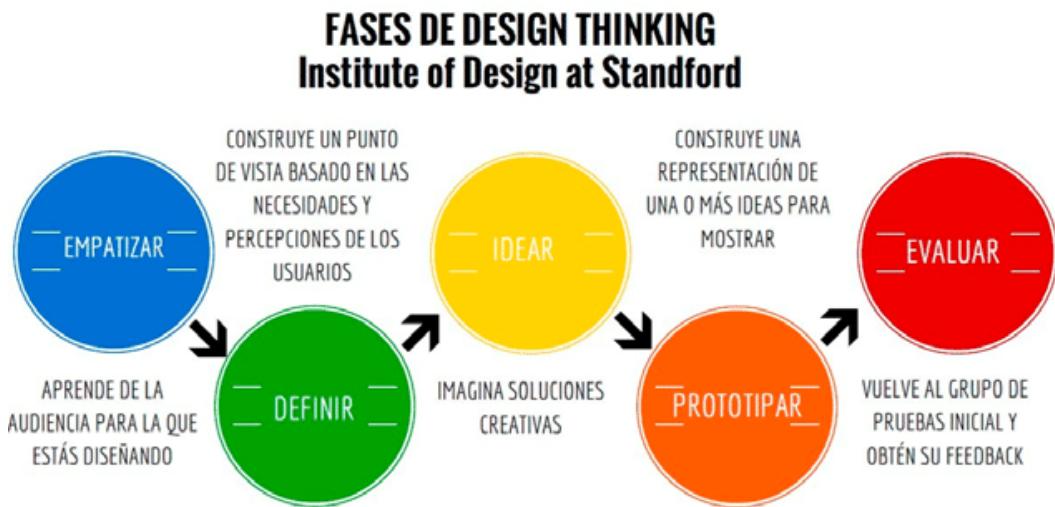


Figura 1: Fases del Design Thinking aplicadas al desarrollo de software.

### Refuerzo

En el contexto de este taller, se aplican principalmente las fases de **Definir** (problema y usuarios) e **Idear** (soluciones y requerimientos) para construir la base del análisis técnico.

## Matriz de trazabilidad extendida

ID RF	Descripción	Caso de Uso	Prueba	Evidencia
RF-01	Registrar productos con código, nombre y precio	CU-Registrar Producto	PR-Registrar Producto	Captura “Producto agregado”
RF-02	Listar productos por categoría	CU-Consultar Inventario	PR-Listar Productos	PDF “Lista categorías”
RF-03	Actualizar stock tras una venta	CU-Actualizar Stock	GIT-Stock	Log “Stock actualizado”

Cuadro 1: Matriz de trazabilidad extendida

Figura 2: Matriz de trazabilidad extendida: RF → Caso de Uso → Prueba → Evidencia.

### Atención

En la práctica profesional, la trazabilidad completa permite saber exactamente qué requerimiento falló si una prueba no se aprueba. En esta etapa, basta con mantener el vínculo RF–Caso de Uso–Módulo en tu proyecto.

*“Un requerimiento claro hoy evita un problema costoso mañana.”*

## 2. Unidad 2: Métodos modernos de desarrollo individual

### Información Importante

En esta unidad se presenta como seleccionar y aplicar una metodología moderna adaptada al trabajo individual del programador-analista.

El objetivo es trabajar de forma ágil, organizada y reflexiva, aprovechando herramientas de control visual (Kanban) y prácticas de desarrollo iterativo (Lean, XP, TDD, Prototipado rápido).

Cada uno de Uds. documentará su método elegido en el apartado “Metodología de desarrollo” del Informe 2.

### 2.1. Comparativa de métodos ágiles actuales

#### Información Importante

Los métodos ágiles comparten principios comunes: iteración, colaboración y mejora continua. A continuación se presenta una síntesis de los más aplicables al trabajo individual.

Método	Ventajas principales	Limitaciones o riesgos
<b>Lean Development</b>	Enfoca el esfuerzo en entregar valor rápidamente, evitando tareas sin impacto.	Requiere autoevaluación constante para no caer en improvisación.
<b>XP (Extreme Programming)</b>	Fomenta la simplicidad del código, la retroalimentación rápida y las pruebas frecuentes.	No siempre se comprende bien sin experiencia previa; puede parecer “caótico”.
<b>Kanban personal</b>	Visualiza el flujo de trabajo; fácil de implementar en GitHub Projects o Trello.	Si no se actualiza a diario, pierde utilidad como herramienta de control.
<b>Prototipado rápido</b>	Permite validar ideas y obtener feedback antes de programar completamente.	Riesgo de quedarse en versiones inacabadas si no se planifica el cierre.
<b>TDD (Test Driven Development)</b>	Garantiza calidad: primero se escribe la prueba y luego el código.	Aumenta el tiempo inicial de desarrollo, pero reduce errores posteriores.

#### Refuerzo

**Conclusión:** Ningún método es absoluto. Lo ideal es combinarlos de forma pragmática: Lean (eficiencia), XP (disciplina técnica) y Kanban personal (control visual).

## 2.2. Metodologías y técnicas combinadas

### Información Importante

Para este taller de Proyecto se recomienda una combinación equilibrada:

- **Lean Personal Kanban:** visualizar el flujo de tareas (“Por hacer”, “En progreso”, “Hecho”) y autoevaluar tu productividad semanal.
- **XP adaptado a un solo programador:** mantener el código simple, refactorizar continuamente y probar pequeñas unidades.
- **TDD y prototipado rápido:** escribir una pequeña prueba o validar un flujo antes de implementar funcionalidades completas.

### Refuerzo

**Flujo sugerido:** Planificar → Desarrollar → Probar → Ajustar → Documentar → Publicar en GitHub.

## 2.3. Justificación metodológica

### Información Importante

En el Informe 2, deben justificar su metodología de desarrollo. Este texto debe ser claro, técnico y demostrar comprensión. A continuación se presentan dos ejemplos contrastivos.

### Ejemplo Correcto

#### Ejemplo de redacción profesional:

El desarrollo del prototipo se realizará siguiendo un enfoque Lean Personal Kanban, apoyado en principios de Extreme Programming. Esta combinación permite priorizar tareas de mayor valor y mantener la calidad del código mediante iteraciones cortas y revisiones continuas. El control del avance se gestionará en un tablero Kanban de GitHub Projects con etiquetas por estado (To Do, Doing, Done).

### Error Común

#### Ejemplo de redacción superficial:

Se usará una metodología ágil porque es moderna y rápida. Se trabajará según se avance y se irán haciendo pruebas cuando sea necesario.

*Problemas:* No define método, no explica por qué es adecuado ni cómo se aplicará.

Criterio	Lean	XP	Kanban	TDD	Más adecuado si...
Simplicidad	4	5	3	4	Buscas mantener código limpio.
Retroalimentación	3	5	4	4	Necesitas detectar errores rápido.
Planificación visual	3	2	5	3	Requieres control del avance.
Aprendizaje iterativo	4	4	4	5	Deseas mejorar en cada ciclo.

Cuadro 3: Matriz de decisión: comparación de métodos por criterio.

### Refuerzo

La elección metodológica debe argumentarse en función de los objetivos del proyecto, el tiempo disponible y la experiencia técnica personal.

## 2.4. Presentación visual del método

### Información Importante

Los métodos ágiles son más comprensibles cuando se representan gráficamente. Usa un diagrama de flujo simple que muestre los ciclos iterativos: planificación → desarrollo → prueba → entrega.

### Ciclo de desarrollo iterativo individual sugerido



Figura 3: Ciclo de desarrollo iterativo individual sugerido.

### Atención

Pueden usar herramientas como draw.io, diagrams.net o Canva para crear su propio diagrama. Recuerden incluirlo como figura en el Informe 2.

*“El mejor método es aquel que puedes mantener con disciplina y claridad.”*

### 3. Unidad 3: Diseño de la Base de Datos

#### Información Importante

El diseño de la base de datos es el puente entre los requerimientos del sistema (Unidad 1) y el diseño funcional (Unidad 4).

Su propósito es estructurar los datos de manera lógica y eficiente, garantizando integridad, consistencia y escalabilidad.

Deben desarrollar el proceso completo desde la identificación de entidades hasta la generación del modelo conceptual.

#### 3.1. De requerimientos al modelo conceptual

#### Información Importante

El punto de partida son los **requerimientos funcionales** definidos anteriormente. De ellos se extraen los elementos que representan información que el sistema debe almacenar o procesar.

Para construir el **Modelo Entidad–Relación (MER)**, identifica:

- **Entidades**: objetos o conceptos del mundo real (Cliente, Producto, Venta, Usuario).
- **Atributos**: características de cada entidad (nombre, dirección, precio, fecha).
- **Relaciones**: vínculos entre entidades (un Cliente realiza una Venta, una Venta incluye Productos).

#### Refuerzo

**Método aplicado:** enfoque “**Conceptual–Lógico–Físico (CLF)**” de diseño de bases de datos.

En esta parte trabajaremos el nivel conceptual, es decir, la representación gráfica e independiente del motor SQL.

#### Ejemplo Correcto

##### Ejemplo práctico: a partir de un requerimiento funcional

RF-01: El sistema debe registrar las ventas realizadas, indicando cliente, productos vendidos, cantidad, precio y fecha.

##### Análisis:

- Entidades: Cliente, Venta, Producto.
- Relaciones: un Cliente realiza muchas Ventas; una Venta incluye muchos Productos.
- Atributos clave: idCliente, nombre, correo, idVenta, fecha, total, idProducto, descripción, precio.

## Error Común

### Errores comunes:

- Tratar un campo como entidad: “Fecha” o “Monto” no son entidades, sino atributos.
- Mezclar datos y procesos: “Emitir boleta” es una función, no una entidad.
- Usar nombres genéricos: “Datos1”, “TablaA” impiden comprensión del modelo.

## Refuerzo

### Reglas básicas para nombrar entidades y atributos:

- Usa sustantivos en singular: “Cliente”, “Producto”, “Venta”.
- Evita abreviaciones poco claras (usar “idCliente” mejor que “cliID”).
- Los atributos deben ser únicos dentro de su entidad.

## Atención

Recuerda: el modelo conceptual debe ser entendible incluso por quien no programa. Su objetivo es comunicar la estructura del sistema, no el código SQL.

## Refuerzo

### Actividad de esta parte:

1. Revisa tus requerimientos funcionales y selecciona al menos 4 que involucren almacenamiento de datos.
2. Identifica las entidades, atributos y relaciones derivadas de ellos.
3. Dibuja tu MER inicial y compáralo con los ejemplos de esta guía.
4. Guarda el diagrama en formato PNG o PDF y súbelo a tu repositorio GitHub.

*“Un buen modelo de datos es la columna vertebral de un sistema confiable.”*

## 3.2. Normalización y calidad del modelo

### Información Importante

La **normalización** es un proceso que garantiza la calidad del diseño de datos, evitando duplicaciones, inconsistencias y dependencias innecesarias.

Consiste en dividir la información en varias tablas relacionadas, de modo que cada dato exista una sola vez y tenga un propósito claro.

En esta parte se aplican las tres primeras formas normales: **1FN**, **2FN** y **3FN**.

### Refuerzo

**Método aplicado:** *Data Quality by Design*. Antes de crear las tablas SQL, verifica que cada relación cumpla con criterios de integridad y dependencia funcional.

### Ejemplo Correcto

**Primera Forma Normal (1FN):** todos los campos deben contener valores atómicos (no repetidos ni listas).

**Ejemplo:**

**Incorrecto:**

```
Cliente(id, nombre, telefonos)
```

**Correcto:**

```
Cliente(id, nombre)
Telefono(id, idCliente, numero)
```

### Ejemplo Correcto

**Segunda Forma Normal (2FN):** todos los atributos dependen completamente de la clave primaria.

**Ejemplo:**

**Incorrecto:**

```
DetalleVenta(idVenta, idProducto, cantidad, nombreProducto)
```

**Correcto:**

```
DetalleVenta(idVenta, idProducto, cantidad)
Producto(idProducto, nombreProducto)
```

## Ejemplo Correcto

**Tercera Forma Normal (3FN):** ningún atributo no clave debe depender de otro atributo no clave.

**Ejemplo:**

**Incorrecto:**

```
Producto(idProducto, nombre, idCategoria, nombreCategoria)
```

**Correcto:**

```
Producto(idProducto, nombre, idCategoria)
```

```
Categoría(idCategoria, nombreCategoria)
```

## Atención

La normalización mejora la consistencia, pero si se aplica en exceso puede dificultar el rendimiento. En proyectos pequeños o académicos, llegar hasta 3FN suele ser suficiente.

### 3.3. Transformación a modelo lógico

#### Información Importante

Una vez definido el modelo conceptual y normalizado, se transforma en el **modelo lógico**, donde cada entidad pasa a ser una tabla SQL con tipos de datos definidos.

El modelo lógico ya puede implementarse en un sistema gestor de base de datos (como SQL Server, MySQL o PostgreSQL).

#### Refuerzo

##### Reglas básicas para el modelo lógico:

- Cada entidad del MER se convierte en una tabla.
- Cada atributo se convierte en una columna con tipo de dato apropiado.
- Las relaciones se representan mediante claves foráneas.

## Ejemplo Correcto

### Ejemplo de transformación (entidad → tabla SQL)

**Entidad:** Cliente(idCliente, nombre, correo, telefono)

```
CREATE TABLE Cliente (
    idCliente INT PRIMARY KEY IDENTITY(1,1),
    nombre VARCHAR(100) NOT NULL,
    correo VARCHAR(100) UNIQUE NOT NULL,
    telefono VARCHAR(20)
);
```

## Ejemplo Correcto

### Ejemplo de relación: Venta–Cliente

```
CREATE TABLE Venta (
    idVenta INT PRIMARY KEY IDENTITY(1,1),
    fecha DATE NOT NULL,
    total DECIMAL(10,2) NOT NULL,
    idCliente INT NOT NULL,
    FOREIGN KEY (idCliente) REFERENCES Cliente(idCliente)
);
```

## Error Común

### Errores comunes:

- No declarar la clave primaria o dejarla como campo genérico (“id”).
- Usar tipos de datos inapropiados (ej: texto largo para números).
- Duplicar información en varias tablas.

## Atención

Evita el uso de nombres reservados del sistema (como “User”, “Order”, “Group”) y mantén una convención clara de prefijos (ej: “tblCliente”, “tblVenta”) si el proyecto lo requiere.

## Refuerzo

### Actividad de esta parte:

1. Aplica las tres primeras formas normales a tu MER.
2. Genera el modelo lógico completo en SQL.
3. Comprueba la consistencia entre claves primarias y foráneas.
4. Sube el script SQL a tu repositorio GitHub en la carpeta `/database`.

*“Una base de datos bien normalizada es sinónimo de precisión y confianza.”*

### 3.4. Buenas prácticas y errores comunes

#### Información Importante

El diseño de una base de datos no termina con la creación de tablas; la calidad del modelo depende también de seguir buenas prácticas que garanticen su estabilidad, seguridad y facilidad de mantenimiento.

A continuación se presentan recomendaciones basadas en estándares profesionales y errores comunes detectados en proyectos técnicos.

#### Refuerzo

##### Buenas prácticas esenciales:

- Definir siempre claves primarias únicas y estables.
- Mantener coherencia en la nomenclatura: usar minúsculas y nombres descriptivos.
- Incluir restricciones (NOT NULL, CHECK, DEFAULT) para asegurar la integridad.
- Usar tipos de datos adecuados: INT para identificadores, VARCHAR para texto, DATE para fechas.
- Comentar las tablas o columnas cuando sea necesario (ej: COMMENT ON TABLE).
- Revisar dependencias entre tablas antes de eliminar o modificar estructuras.

#### Error Común

##### Errores comunes a evitar:

- No definir claves foráneas, rompiendo la integridad referencial.
- Repetir información en varias tablas (“datos huérfanos” o duplicados).
- No usar índices en campos de búsqueda frecuente.
- Olvidar registrar fechas de creación o actualización de registros.
- Falta de respaldo o control de versiones del esquema SQL.

#### Atención

Una base de datos puede funcionar aunque esté mal diseñada, pero eso afectará el rendimiento y complicará el mantenimiento futuro.

**El buen diseño no es opcional: es una inversión.**

### 3.5. Integración con el proyecto

#### Información Importante

En el Informe 2, el diseño de la base de datos debe integrarse con los demás componentes del sistema: requerimientos, casos de uso y diseño funcional.

Esto garantiza trazabilidad y coherencia entre las distintas etapas del proyecto.

#### Refuerzo

##### Estructura sugerida para documentar el modelo de datos:

1. **Descripción general:** propósito del modelo y su relación con el sistema.
2. **Diagrama MER final:** incluir imagen exportada del diagrama validado.
3. **Tabla resumen:** entidades, atributos principales y relaciones.
4. **Justificación del diseño:** por qué se eligieron esas estructuras.
5. **Anexo:** script SQL inicial (creación de tablas y claves).

#### Ejemplo Correcto

##### Ejemplo de fragmento del Informe 2:

El modelo de datos del sistema “Gestión de Taller Automotriz” se diseñó en base a los requerimientos RF01–RF05.

Cada cliente puede registrar múltiples vehículos, y cada vehículo puede tener múltiples órdenes de trabajo.

El modelo sigue las tres primeras formas normales, y su estructura fue verificada mediante pruebas de integridad referencial.

El script SQL se encuentra disponible en el repositorio GitHub: carpeta /database.

#### Atención

Recuerda incluir la referencia al commit o enlace directo del script SQL en tu informe. Esto demuestra dominio técnico y control del versionamiento en GitHub.

### 3.6. Actividad guiada de integración

#### Refuerzo

##### Actividad final del avance 3:

1. Completa tu modelo lógico y revisa las claves primarias y foráneas.
2. Exporta tu MER y guárdalo como imagen (PNG o PDF).
3. Agrega el diagrama y una tabla resumen al Informe 2.
4. Verifica que todos los requerimientos con datos estén representados en el modelo.
5. Sube tu script SQL final a GitHub con el mensaje de commit: “Versión estable de base de datos – Avance 2”.

#### Atención

Este entregable será parte del Informe de Análisis y Diseño (Avance 2).

Debe estar correctamente documentado y sin errores de sintaxis en el script SQL.

*“El diseño de datos es la huella técnica del pensamiento lógico del programador.”*

## 4. Unidad 4: Diseño funcional y experiencia del usuario

### Información Importante

Esta unidad enseña a transformar los requerimientos definidos en la primera etapa en un **diseño funcional**, que representa las funciones del sistema y cómo los usuarios interactúan con ellas.

Aquí el alumno aprenderá a elaborar diagramas de casos de uso, identificar actores, y garantizar la coherencia entre los requerimientos funcionales (RF) y los módulos del sistema.

### Atención

El diseño funcional permite que un tercero (docente o evaluador) entienda el funcionamiento general del sistema sin necesidad de revisar el código.

Es el mapa visual del proyecto.

### 4.1. Del análisis al diseño funcional

#### Información Importante

Cada requerimiento funcional debe poder asociarse a una o más funciones del sistema. El conjunto de todas esas funciones conforma el diseño funcional.

Para estructurarlo, se identifican los **actores** (personas o sistemas externos) y las **acciones** que ejecutan dentro del sistema.

#### Refuerzo

**Regla de trazabilidad:** Todo RF debe tener un caso de uso que lo represente. Si un caso de uso no está asociado a un RF, probablemente no sea necesario.

#### Ejemplo Correcto

##### Ejemplo práctico — Sistema “Gestión de Taller Automotriz”

**Requerimiento funcional RF01:** El sistema debe registrar las órdenes de trabajo asociadas a cada vehículo.

**Caso de uso derivado:** “Registrar Orden de Trabajo”.

**Actor principal:** Mecánico.

**Flujo básico:**

1. El mecánico ingresa los datos del vehículo y cliente.
2. Añade la lista de servicios realizados.
3. Guarda la orden, que queda registrada en la base de datos.

## Error Común

### Errores comunes:

- Confundir un caso de uso con una pantalla del sistema.
- Incluir detalles de la interfaz (botones, colores) en lugar de la función.
- Crear casos de uso sin un requerimiento que los justifique.

## Refuerzo

### Buenas prácticas:

- Usa nombres verbales: “Registrar venta”, “Generar informe”, “Actualizar cliente”.
- Mantén la simplicidad: cada caso de uso debe tener un único objetivo.
- Representa los actores externos con claridad (Usuario, Administrador, Cliente, etc.).

**Diagrama de Casos de Uso – Sistema Gestión de Taller Automotriz (UML formal)**

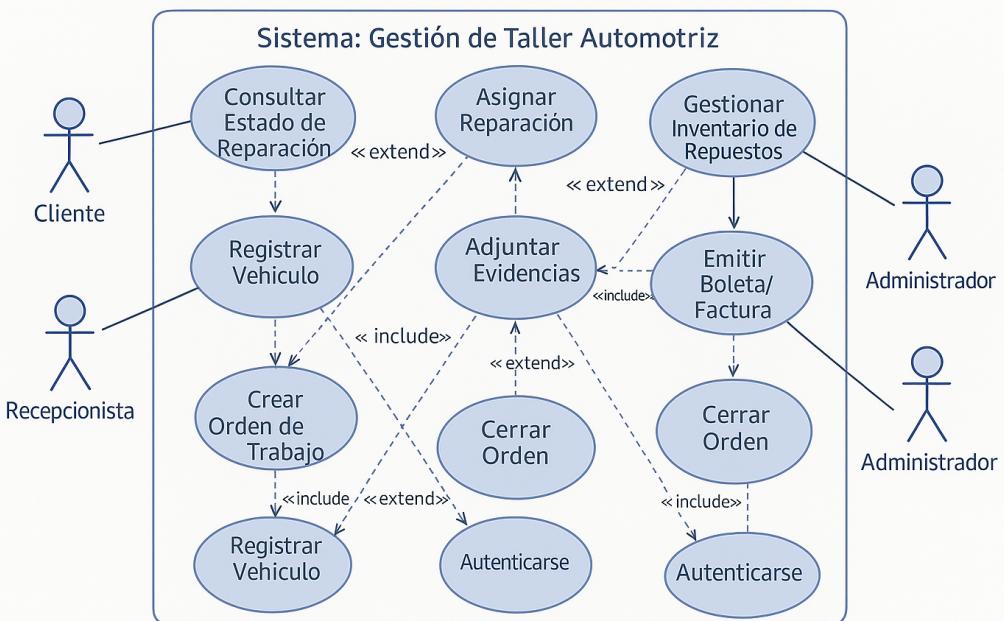


Figura 4: Ejemplo de diagrama de casos de uso del sistema “Gestión de Taller Automotriz”.

## Atención

Los diagramas de casos de uso se dibujan habitualmente en herramientas como Draw.io, Lucidchart o StarUML.

**Nota:** UML (\*Unified Modeling Language\*) es un estándar gráfico para modelar sistemas, y el diagrama de casos de uso es su nivel más accesible y visual.<sup>a</sup>

<sup>a</sup>UML (Unified Modeling Language): lenguaje gráfico estandarizado para modelar, visualizar y documentar sistemas de software.

## Refuerzo

### Actividad de esta parte:

1. Revisa tus requerimientos funcionales (RF01–RF05).
2. Crea un diagrama de casos de uso donde cada RF tenga su representación.
3. Incluye actores y relaciones (asociaciones o extensiones si aplica).
4. Exporta la imagen (PNG o PDF) y súbelo a tu repositorio GitHub en la carpeta /design.
5. Inserta el diagrama en el Informe 2 dentro del apartado “Diseño funcional”.

*“Diseñar funciones claras es la mejor forma de evitar errores en la programación.”*

## 4.2. Diagramas de flujo de procesos y de actividad

### Información Importante

Los diagramas de **flujo de procesos** (DFD simplificado) y **actividad** (UML) describen el paso a paso de una operación del sistema: qué hace el usuario, qué valida el sistema y qué datos se mueven.

Son el puente entre el *qué* (casos de uso) y el *cómo* (interacciones y validaciones).

## Refuerzo

### Cuándo usar cada uno:

- **Diagrama de Flujo (DFD simplificado):** para visualizar el movimiento de información entre pasos.
- **Diagrama de Actividad UML:** para detallar las *ramas*, decisiones, bucles y estados.

## Ejemplo Correcto

### Ejemplo — “Crear Orden de Trabajo” (actividad)

- Inicio → Ingresar patente → Buscar vehículo → [¿Existe?] → (Sí) Cargar datos / (No) Registrar vehículo → Ingresar servicios → Calcular total → Guardar → Fin.
- Validaciones: campos obligatorios, stock de repuestos, formato de fecha.

## Error Común

### Errores comunes:

- Mezclar pantallas con procesos (los botones pertenecen a wireframes, no al flujo).
- Omitir decisiones críticas (¿qué pasa si no hay stock?).
- Usar nombres poco claros para actividades (“Proceso 1”, ”Tarea A”).

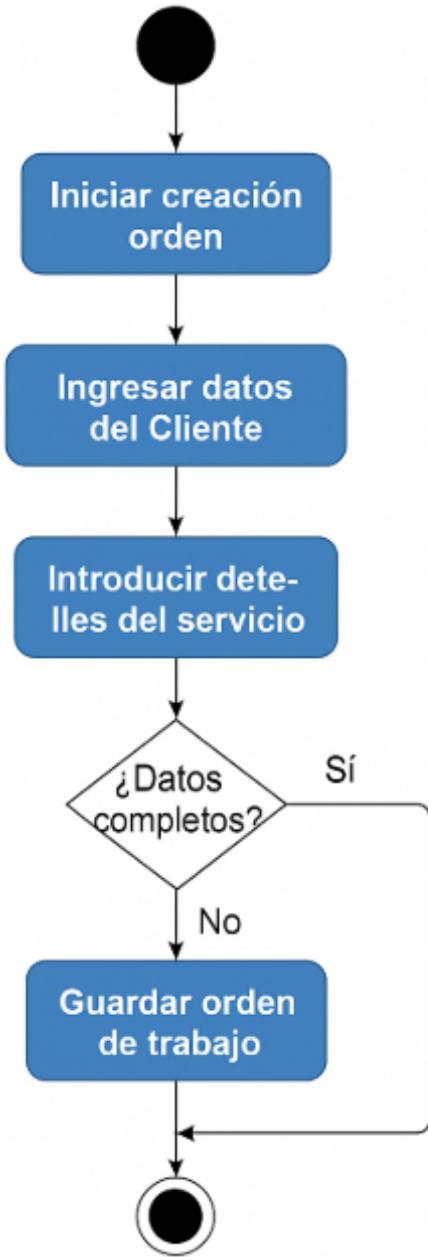


Figura 5: Ejemplo de diagrama de actividad para “Crear Orden de Trabajo”.

### Atención

Herramientas sugeridas: draw.io / diagrams.net, StarUML, Visual Paradigm (comunidad), PlantUML. Exporta como PNG o PDF e incluye la figura en el Informe 2.

## 4.3. Diseño de experiencia del usuario (UX/UI) con wireframes

### Información Importante

Un **wireframe** es un boceto estructural de una pantalla que define contenido, jerarquía y navegación sin entrar en colores o estilos finales.<sup>a</sup>

Se diseñan primero las pantallas críticas y se valida la coherencia con el modelo de datos y los casos de uso.

<sup>a</sup>**Wireframe:** bosquejo de baja fidelidad para decidir estructura y flujo antes del diseño visual.

### Refuerzo

#### Principios de buen UX (adaptados de Nielsen):

- Visibilidad del estado del sistema (mensajes claros y oportunos).
- Correspondencia con el mundo real (nombres conocidos, no códigos internos).
- Control y libertad del usuario (cancelar, deshacer, volver).
- Consistencia y estándares (mismos nombres y ubicaciones para acciones).
- Prevención de errores (validaciones antes de guardar).

### Ejemplo Correcto

#### Wireframes sugeridos (mínimo 2):

- **Login:** correo + contraseña, feedback de error, enlace “¿Olvidó su contraseña?”.
- **Gestión principal:** menú lateral (Órdenes, Clientes, Vehículos, Reportes), tabla con filtros y acción “Crear Orden”.

The wireframe shows a login form with the following components:

- Title:** Login
- Text:** Username
- Form:** An empty input field.
- Text:** Password
- Form:** An empty input field.
- Text:** Remember me
- Image:** A checkbox icon.
- Text:** Log in
- Text:** Forgot password?

Figura 6: Wireframe de ejemplo: pantalla de Login.

The wireframe illustrates a management interface with the following structure:

LOGO	CUADRO PRINCIPAL	ICONO (U)																					
BARRA LATERAL	ENLACE	MESSAJE 1 / MESSAJE 2 / MESSAJE 3																					
	ENLACE																						
	ENLACE																						
	TABLA RESUMEN	<table border="1"> <thead> <tr> <th>Columna 1</th> <th>Columna 2</th> <th>Columna 3</th> </tr> </thead> <tbody> <tr> <td>Dato</td> <td> Lorem ipsum dolor seclex</td> <td>20</td> </tr> <tr> <td>Dato</td> <td> Lorem ipsum dolor seclex</td> <td>15</td> </tr> <tr> <td>Dato</td> <td> Lorem ipsum dolor seclex</td> <td>30</td> </tr> <tr> <td>Dato</td> <td> Lorem ipsum dolor seclex</td> <td>12</td> </tr> <tr> <td>Dato</td> <td> Lorem ipsum dolor seclex</td> <td>4</td> </tr> <tr> <td>Tetsc</td> <td>Descripción</td> <td></td> </tr> </tbody> </table>		Columna 1	Columna 2	Columna 3	Dato	Lorem ipsum dolor seclex	20	Dato	Lorem ipsum dolor seclex	15	Dato	Lorem ipsum dolor seclex	30	Dato	Lorem ipsum dolor seclex	12	Dato	Lorem ipsum dolor seclex	4	Tetsc	Descripción
Columna 1	Columna 2	Columna 3																					
Dato	Lorem ipsum dolor seclex	20																					
Dato	Lorem ipsum dolor seclex	15																					
Dato	Lorem ipsum dolor seclex	30																					
Dato	Lorem ipsum dolor seclex	12																					
Dato	Lorem ipsum dolor seclex	4																					
Tetsc	Descripción																						

Figura 7: Wireframe de ejemplo: panel principal de gestión.

## Error Común

### Malas prácticas frecuentes en UI:

- Usar abreviaturas internas (“IDVeh”) en vez de nombres entendibles (“Vehículo”).
- Sobrecargar la pantalla con campos irrelevantes.
- No indicar estados: cargando, guardado, error.

## Atención

Herramientas libres recomendadas: **Penpot** (open source), Figma (plan educativo), Balsamiq (trial). Exporta las pantallas y guárdalas en `/design/wireframes` del repositorio.

## Refuerzo

### Actividad de esta parte:

1. Construye el diagrama de flujo / actividad para una operación completa (por ejemplo, “Crear Orden de Trabajo”).
2. Diseña al menos 2 wireframes (Login + Gestión principal) y vincúlalos al caso de uso.
3. Sube las imágenes (PNG/PDF) al repositorio y *cita* el commit en el Informe 2.

*“Primero el flujo correcto, luego las pantallas claras.”*

## 4.4. Integración visual y consistencia

### Información Importante

Una buena experiencia de usuario depende no solo de la apariencia, sino también de la **coherencia entre los elementos visuales y los datos**.

La integración visual une el diseño de pantallas con la estructura de la base de datos y los casos de uso, garantizando que cada elemento tenga un propósito funcional.

### Refuerzo

#### Principios de consistencia visual:

- Los nombres de campos deben coincidir con los atributos de la base de datos (ej: campo “Correo” → columna **correo**).
- Los botones deben reflejar acciones reales del sistema (**Guardar** ejecuta una inserción, **Eliminar** una eliminación lógica o física).
- Mantén un esquema visual uniforme (tipografía, alineación, colores funcionales).
- Usa iconografía coherente: el mismo ícono para la misma acción en todas las pantallas.

### Ejemplo Correcto

#### Ejemplo de correspondencia entre interfaz y modelo:

Pantalla	Campo visible	Atributo en BD
Formulario Cliente	Nombre completo	nombre
Formulario Cliente	Correo electrónico	correo
Órdenes	Fecha de ingreso	fechaIngreso
Órdenes	Total	total

### Error Común

#### Errores frecuentes:

- Usar nombres distintos en BD y formularios (**email** vs. “correo”).
- Crear campos visuales sin respaldo en el modelo de datos.
- Duplicar elementos visuales para la misma acción.

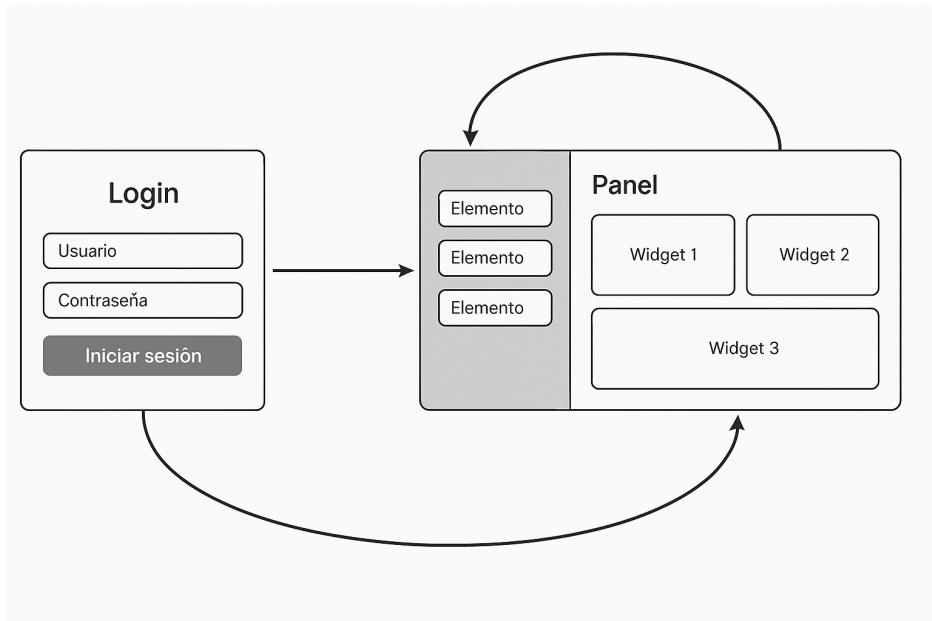


Figura 8: Ejemplo de wireflow: integración entre flujo y diseño visual.

### Atención

El **wireflow** combina el diagrama de flujo con los wireframes, mostrando cómo el usuario se desplaza por las pantallas según sus decisiones.<sup>a</sup>

<sup>a</sup>**Wireflow:** técnica visual moderna que une la navegación de pantallas (wireframes) con la lógica de flujo de usuario.

## 4.5. Documentación en el Informe 2

### Información Importante

El diseño funcional y de interfaz debe documentarse claramente dentro del Informe de Análisis y Diseño (Avance 2). Esto permite demostrar la relación entre requerimientos, base de datos, procesos y pantallas.

Sigue esta estructura recomendada para la entrega.

## Refuerzo

### Estructura recomendada para el apartado de diseño:

1. Descripción general del diseño funcional (qué partes del sistema se modelaron).
2. Diagramas de casos de uso y de flujo o actividad.
3. Wireframes o pantallas principales.
4. Explicación de decisiones de diseño (UX y consistencia visual).
5. Relación entre pantallas, casos de uso y base de datos.
6. Anexos: imágenes y referencias al repositorio GitHub.

## Atención

El documento debe mostrar coherencia entre los entregables previos. No se trata de crear pantallas bonitas, sino de demostrar un **diseño funcional y lógico que responda a los requerimientos definidos**.

Usa lenguaje técnico pero claro, y referencias cruzadas con capturas o figuras.

## 4.6. Actividad guiada final

## Refuerzo

### Actividad integradora:

1. Revisa los requerimientos y casos de uso principales.
2. Verifica que cada caso tenga al menos una pantalla o flujo representado.
3. Asegura la coherencia entre nombres, acciones y atributos.
4. Prepara el documento final del Informe 2 con diagramas, wireframes y tabla de correspondencia.
5. Realiza una autoevaluación usando la rúbrica entregada por el docente.

## Atención

Entrega: el avance debe subirse al repositorio institucional o GitHub, incluyendo todos los archivos gráficos y el Informe 2 en formato PDF.

Usa mensajes de commit descriptivos (ejemplo: "Diseño funcional completado { Avance 2 }").

*“Un diseño consistente transforma la lógica en experiencia.”*

## 5. Unidad 5: Gestión técnica del repositorio (Git y GitHub)

### Información Importante

Esta parte enseña a trabajar con **Git** y **GitHub** desde **Windows** usando **CMD o PowerShell**. Aprenderás instalación, configuración inicial, comandos esenciales, buenas prácticas y el flujo correcto para registrar avances del proyecto y publicarlos en GitHub.

*Nota:* los ejemplos usan la ruta C:

AIEP

TallerProyectoEspecialidad y el repositorio:

<https://github.com/usuario/TallerProyectoEspecialidad>.

Cada uno de Uds. Debe reemplazar **usuario** por su **nombre real de su GitHub**.

### Atención

**Importante:** C:

AIEP

TallerProyectoEspecialidad es **sólo un ejemplo**. Usen su propia carpeta y su **cuenta personal de GitHub**. No creen repositorios con nombres genéricos como “prueba” o “MiProyecto”.

### 5.1. Instalación y configuración inicial

### Información Importante

- 1) Descarga e instala **Git para Windows** desde [git-scm.com](http://git-scm.com).
- 2) Abre **CMD o PowerShell** y configura tu identidad (aparece en los commits).<sup>a</sup>

<sup>a</sup>**Commit:** registro de una versión local del proyecto con un mensaje asociado.

```
# Configurar nombre y correo (aparecen en el historial de commits)
git config --global user.name "Nombre Apellido"
git config --global user.email "tu_correo@ejemplo.com"

# (Opcional) Establecer editor por defecto
git config --global core.editor "code --wait"

# Verificar la configuración
git config --list
```

### Refuerzo

**Autenticación con GitHub:** al primer `git push`, GitHub pedirá iniciar sesión. Usa *Sign in with browser* (token) o configura una **Personal Access Token (PAT)** si se solicita.

## 5.2. Crear proyecto local y subirlo a GitHub

### Información Importante

Crea una carpeta para el taller, inicializa Git, realiza tu primer commit y conecta el repositorio remoto en GitHub.

```
# 1) Crear carpeta y entrar
mkdir C:\AIEP\TallerProyectoEspecialidad
cd C:\AIEP\TallerProyectoEspecialidad

# 2) Inicializar repositorio local
git init

# 3) Agregar archivos y primer commit
echo "# Taller Proyecto de la Especialidad" > README.md
git add .
git commit -m "Version inicial - estructura base"

# 4) Conectar con GitHub (crea antes un repo vacio con ese nombre)
git remote add origin https://github.com/usuario/TallerProyectoEspecialidad.git

# 5) Publicar rama principal en remoto
git branch -M main
git push -u origin main
```

### Error Común

#### Errores frecuentes:

- Crear un repositorio diferente para cada avance. *Solución:* usa el mismo repositorio todo el modulo.
- Subir todo desde la web con “Upload files”. *Solución:* trabaja con `git add`, `commit`, `push`. !!!!!!
- Borrar el repositorio remoto para “empezar de cero”. *Solución:* usa commits y tags para versionar.

### 5.3. Flujo básico de trabajo (local → remoto)

#### Información Importante

El ciclo del día a día es: editar archivos → git add → git commit → git push.<sup>a</sup>

<sup>a</sup>**Push:** envío de tus commits locales al repositorio remoto (GitHub).

```
# Ver estado de cambios  
git status  
  
# Agregar cambios al area de preparacion  
git add .  
  
# Registrar versin local con mensaje claro  
git commit -m "Agrega diagrama MER final y script SQL base"  
  
# Publicar cambios en GitHub  
git push
```

#### Refuerzo

##### Buenas prácticas de mensajes:

- ”Agrega formulario de Orden de Trabajo”
- ”Refactoriza validaciones en crear-cliente”
- ”Sube Informe 2 (Análisis y Diseño) a /docs”

Evita mensajes como: “update”, “cosas varias”, “final”.

### 5.4. Cuadro resumen de comandos esenciales

Comando	Qué hace	Cuándo usar
git init	Inicia un repositorio local	Al comenzar el proyecto
git status	Muestra cambios pendientes	Antes de agregar/commitear
git add .	Prepara archivos para commit	Tras editar/crear archivos
git commit -m "mensaje"	Registra una versión local	Al cerrar un avance claro
git push	Sube commits a GitHub	Al finalizar una sesión de trabajo
git pull	Trae cambios remotos	Si trabajas en 2 PCs o colaboras
git log --oneline	Historial compacto	Para revisar versiones
git remote -v	Remotos configurados	Para verificar origen (origin)

## 5.5. Ramas (branches) y versiones etiquetadas (tags)

### Información Importante

**Branch (rama):** línea de desarrollo paralela a `main`. En este modulo, al ser proyectos individuales, **se recomienda trabajar en una sola rama** para no complicar el flujo.<sup>a</sup>

<sup>a</sup>**Branch:** puntero a una serie de commits. Útil para trabajo paralelo o features, pero puede complejizar si se abusa.

### Refuerzo

Usa tags (versiones) para avances oficiales:

```
# Crear etiqueta de versión (p. ej., Entrega Avance 2)
git tag -a v2.0 -m "Entrega Avance 2 completada"
```

```
# Publicar la etiqueta en GitHub
git push origin v2.0
```

Acción	Se usa para	Riesgo si se abusa
Trabajar en <code>main</code>	Flujo simple de un solo desarrollador	Ninguno relevante
Crear muchas ramas	Features complejas en paralelo	Conflictos, integración difícil
Crear tags (v1.0, v2.0)	Marcar entregas oficiales	Ninguno (recomendado)

## 5.6. Estructura profesional del repositorio

### Información Importante

Organiza tu proyecto en carpetas estándar. Sube tus informes, diagramas y scripts SQL junto con el código.

```
/docs      # Informes (PDF, fuentes LaTeX), imágenes de diagramas
/design    # Wireframes, casos de uso, flujos (PNG/PDF)
/database  # Scripts SQL (DDL/seed)
/src       # Código fuente del prototipo
/tests     # Pruebas básicas (si aplica)
README.md  # Descripción del proyecto
.gitignore # Archivos a excluir del control de versiones
```

## Ejemplo Correcto

### Ejemplo mínimo de .gitignore

```
# Windows y editores
Thumbs.db
*.log
.vscode/
__pycache__/
*.aux
*.out
*.toc
```

## 5.7. Cuándo hacer commit, push o crear una versión

### Información Importante

**Commit local:** cada avance significativo (diagrama listo, sección de informe terminada, módulo probado).

**Push:** al final de la sesión o cuando un avance esté probado.

**Versión (tag):** al completar una entrega oficial (*Avance 2 completo*).

```
# Ejemplos de mensajes profesionales
git commit -m "MER final y normalizacion a 3FN"
git commit -m "Casos de uso y wireframes base en /design"
git commit -m "Informe 2 listo en /docs"
```

### Error Común

**Evita:** "final", "listo ahora sí", "varios cambios", "cosas". Mensajes vagos dificultan la revisión y la trazabilidad.

## 5.8. Actividad guiada

### Refuerzo

Tu entrega de esta parte debe incluir:

1. Repositorio en GitHub con estructura de carpetas estándar.
2. Al menos 3 commits con mensajes claros.
3. Un push exitoso a `main`.
4. Un tag de versión (v2.0 para Avance 2).
5. README con descripción breve y cómo ejecutar el prototipo.

### Atención

Crea también un tablero **GitHub Projects (Board)** con columnas: *Por hacer, En progreso, Terminado*. Agrega 3 issues mínimos (por ejemplo: “MER final”, “Wireframes base”, “Subir Informe 2 PDF”). Incluye capturas en el Informe 2.

“*Versionar con disciplina es construir confianza en tu propio proceso.*”

## 5.9. Conceptos clave de control de versiones

### Información Importante

**Branch (rama):** línea paralela de trabajo que apunta a una secuencia de commits.<sup>a</sup> **Tag:** marcador inmutable que identifica una versión específica (por ejemplo, v2.0). **Release:** publicación en GitHub basada en un tag e incluye notas y archivos adjuntos.

<sup>a</sup>**Commit:** registro de una versión local con mensaje.

### Atención

En proyectos **individuales de estudiante**, prioriza un flujo simple en `main`. Usa ramas *sólo* para experimentos puntuales que luego fusionas (*merge*) y eliminás.

## 5.10. Trabajo práctico con ramas (branches)

### Información Importante

Comandos desde **Windows CMD/PowerShell**. Para alumnos que prefieren sintaxis moderna, se incluye también `git switch`.

```
# Crear una rama nueva a partir de la actual (sintaxis clásica)
git branch diseno-ui
# Cambiar a esa rama
git checkout diseno-ui

# Sintaxis moderna equivalente (recomendada)
git switch -c diseno-ui

# Ver ramas y la rama actual (marcada con *)
git branch

# Volver a main (rama principal)
git switch main

# Fusionar cambios de la rama diseno-ui dentro de main
git merge diseno-ui

# Eliminar rama local ya fusionada
git branch -d diseno-ui

# (Opcional) Publicar una rama en remoto
git push -u origin diseno-ui
```

## Error Común

### Errores comunes:

- Crear muchas ramas y no fusionarlas nunca (historial caótico).
- Trabajar semanas en una rama sin integrar con `main` (conflictos grandes al final).
- Mezclar trabajo de informe y de código en una rama que no corresponde.

## Refuerzo

**Recomendación del curso:** usa una única rama `main`. Crea ramas cortas solo para pruebas grandes; fusiona y elimina al terminar.

## 5.11. Versiones con tags y publicaciones (releases)

### Información Importante

Marca las **entregas oficiales** con **tags anotados** y, si corresponde, crea un **release** en GitHub con descripción y adjuntos (por ejemplo, el PDF del informe).

```
# Crear un tag ANOTADO (recomendado) para Avance 2
git tag -a v2.0 -m "Entrega Avance 2 completada"

# Enviar el tag al remoto (GitHub)
git push origin v2.0

# Ver tags disponibles
git tag

# Moverse a una versión (modo detached HEAD: solo para revisar)
git checkout v2.0
# o bien (moderno):
git switch --detach v2.0
```

### Ejemplo Correcto

**Release en GitHub:** En el repositorio → pestaña *Releases* → *Draft a new release* → seleccionar tag (v2.0), escribir notas (cambios incluidos) y adjuntar el PDF del Informe 2.

### Atención

**Tag vs Release:** El *tag* es el marcador técnico en Git; el *release* es la presentación visible en GitHub (con notas y archivos). Ambos deben corresponderse.

## 5.12. Cuadro resumen: acciones, propósito y riesgos

Acción	Propósito	Cuándo	Riesgo si se abusa
<i>Commit</i>	Guardar avance local	Cambio coherente listo	Historial confuso si el mensaje es vago
<i>Push</i>	Publicar avances en GitHub	Fin de sesión o hito probado	Subir errores si no se probó
<i>Tag</i>	Marcar versión estable	Cerrar un Avance o entrega	Ninguno (recomendado)
<i>Branch</i>	Trabajo paralelo aislado	Prueba grande o feature puntual	Conflictos o pérdida de cambios si no se fusiona
<i>Release</i>	Publicación con notas y archivos	Entrega formal evaluable	Duplicar si no coincide con el tag

## 5.13. Recuperación de errores: restore, reset y revert

### Información Importante

Cuando algo sale mal, **no borres el repositorio**. Usa estas herramientas en orden de menor a mayor impacto.

### Refuerzo

#### Escenario A — Deshacer cambios NO confirmados (sin commit)

```
# Quitar archivos del rea de preparacin (staging)
git restore --staged .

# Descartar cambios locales en archivos (volver al ltimo commit)
git restore .
```

### Refuerzo

#### Escenario B — Deshacer el último commit (aún local)

```
# Mantener cambios en el directorio de trabajo (solo deshacer el commit)
git reset --soft HEAD~1

# Deshacer commit y staging (deja cambios en archivos)
git reset --mixed HEAD~1

# PELIGRO: Descartar commit y cambios (no recuperable)
git reset --hard HEAD~1
```

## Atención

**Precaución:** `reset --hard` borra cambios. Úsalo solo si estás 100% seguro. No lo uses si ya hiciste `push`.

## Refuerzo

### Escenario C — Revertir un commit ya publicado (historial limpio)

```
# Ver historial compacto para identificar el hash
git log --oneline

# Crear un commit inverso que revierte uno anterior
git revert <HASH>

# Subir el revert a GitHub
git push
```

## Ejemplo Correcto

**¿Cuál usar?** `restore` para archivos sin commit; `reset` para commits locales; `revert` para deshacer en remoto sin reescribir historial.

## 5.14. Guía rápida de comandos avanzados (resumen)

Acción	Comando
Crear y cambiar a rama nueva	<code>git switch -c nombre-rama</code>
Volver a main	<code>git switch main</code>
Fusionar rama en main	<code>git merge nombre-rama</code>
Eliminar rama local	<code>git branch -d nombre-rama</code>
Listar ramas	<code>git branch</code>
Crear tag anotado	<code>git tag -a vX.Y -m "mensaje"</code>
Publicar tag	<code>git push origin vX.Y</code>
Revertir commit publicado	<code>git revert &lt;HASH&gt;</code>
Ver historial compacto	<code>git log --oneline</code>

## 5.15. Actividad guiada

### Refuerzo

Para completar esta parte debes:

1. Crear una rama corta (por ejemplo, `diseno-ui`), realizar un cambio de prueba y fusionarla en `main`.
2. Etiquetar la versión de Entrega (`v2.0`) y publicarla en GitHub.
3. Crear un *release* basado en ese tag y adjuntar el PDF del Informe 2.
4. Simular un error y revertirlo correctamente (`git revert`).
5. Agregar en el Informe 2 capturas del release y del historial con la etiqueta.

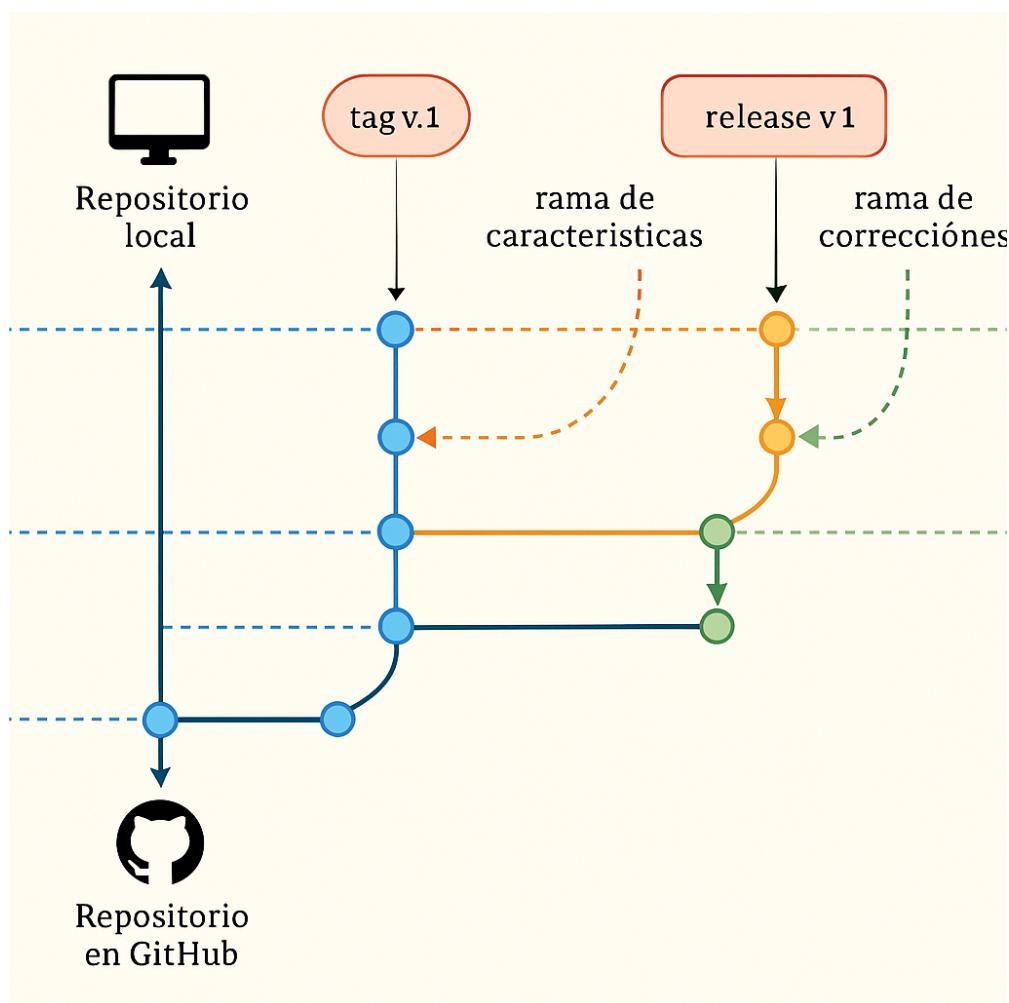


Figura 9: Esquema sugerido del flujo de control de versiones: trabajo en `main`, ramas cortas, tags y releases.

## 5.16. Organización profesional del repositorio en GitHub

### Información Importante

En esta parte se verá a organizar correctamente el repositorio del proyecto, garantizando orden, claridad y trazabilidad. La estructura debe reflejar las áreas principales del trabajo: código, documentación, diseño y pruebas.

```
/docs      # Informes (PDF, fuentes LaTeX), imágenes de diagramas  
/design    # Wireframes, casos de uso, flujos (PNG/PDF)  
/database  # Scripts SQL (DDL/seed)  
/src       # Código fuente del prototipo  
/tests     # Pruebas básicas (si aplica)  
README.md  # Descripción del proyecto  
.gitignore # Archivos a excluir del control de versiones
```

### Refuerzo

**Recomendación:** Cada avance debe incluir evidencias dentro del repositorio. Por ejemplo, el Informe 2 en formato PDF dentro de /docs, y las capturas de diagramas dentro de /design.

### Error Común

#### Errores comunes:

- Subir archivos sueltos en la raíz del repositorio sin carpetas.
- Incluir archivos temporales o personales (como ‘.vscode’ o ‘Thumbs.db’).
- No actualizar el ‘README.md’ con los avances del proyecto.

## 5.17. Archivo README.md: identidad del proyecto

### Información Importante

El archivo **README.md** es la portada del repositorio en GitHub. Describe el propósito del proyecto, tecnologías utilizadas, autor y pasos para ejecutar el prototipo.

```
# Ejemplo de README.MD para que Matias lo haga bien
# Taller Proyecto de la Especialidad

## Autor
Nombre del estudiante

## Descripcion
Aplicacion web para la gestion de ordenes de trabajo en un taller automotriz.

## Tecnologias
- SQL Server
- HTML / CSS / JavaScript
- Python Flask (o lenguaje elegido)

## Estructura del proyecto
/docs   informes
/design  diagramas
/database base de datos
/src    cdigo fuente

## Ejecucion
1. Clonar este repositorio.
2. Configurar la base de datos con los scripts en /database.
3. Ejecutar la aplicaci&on desde /src.
```

### Atención

Mantén el **README.md actualizado** en cada avance. Esto facilita la revisión del docente y mejora la presentación profesional del trabajo.

## 5.18. Uso de GitHub Projects (Kanban)

### Información Importante

GitHub Projects incluye un sistema visual tipo **Kanban**, útil para organizar tareas y hacer seguimiento de avances. Permite ver de forma clara qué está pendiente, en progreso y terminado.

1. Entra a tu repositorio en GitHub → pestaña *Projects*.
2. Crea un nuevo tablero (*New Project*) → selecciona **Board**.
3. Agrega columnas: **Por hacer**, **En progreso**, **Terminado**.
4. Crea **Issues** para cada tarea del proyecto (por ejemplo: “Diseñar MER”, “Crear prototipo CRUD Clientes”).
5. Mueve las tarjetas conforme avances.



Figura 10: Ejemplo de tablero Kanban creado en GitHub Projects.

### Refuerzo

Cada alumno debe tener su tablero con al menos 3 tareas en ejecución y reflejar su progreso real semanal. Captura el tablero y adjunta la imagen en el Informe 2.

## 5.19. Cuadro resumen: organización y seguimiento

Elemento	Propósito	Recomendación
Carpetas /docs, /src, /design	Separar documentación, código y diseño	Nombrar con claridad y mantener actualizadas
README.md	Describir el proyecto	Actualizar en cada entrega
.gitignore	Excluir archivos temporales	Revisar antes de cada commit
GitHub Projects (Kanban)	Visualizar progreso	Mantener actualizado semanalmente
Releases	Publicar versiones oficiales	Subir entregas formales (v2.0, v3.0)

## 5.20. Actividad guiada final

### Refuerzo

Para completar esta parte, el estudiante debe:

1. Organizar correctamente el repositorio en carpetas.
2. Crear y mantener actualizado su archivo README.md.
3. Configurar un tablero **GitHub Projects (Board)** con al menos 3 columnas y 3 issues.
4. Adjuntar capturas del tablero y del repositorio en el Informe 2.
5. Publicar un **release v2.0** con el Informe 2 adjunto.

*“La organización es parte del código: sin estructura, no hay progreso.”*

## 6. Unidad 6: Integración del Informe de Análisis y Diseño

### Información Importante

Esta unidad corresponde a la **entrega formal del Avance 2**, donde el estudiante debe consolidar todo el trabajo técnico y documental realizado durante el taller: requerimientos, diseño, repositorio y gestión de versiones.

El objetivo es elaborar un **informe técnico-profesional completo y coherente**, aplicando criterios de claridad, estructura y evidencias verificables.

### Atención

Este informe consolida el trabajo técnico y documental del Avance 2. Debe reflejar la madurez del proyecto y la capacidad del estudiante para analizar, diseñar y comunicar su solución. No se trata de copiar pasos, sino de **demostrar comprensión y coherencia global**.

### 6.1. Estructura general del informe

#### Información Importante

El informe debe presentar una estructura profesional que permita comprender de forma rápida el propósito, el desarrollo y las conclusiones del trabajo. Se recomienda seguir este orden:

Sección	Contenido recomendado
<b>Portada</b>	Nombre del estudiante, asignatura, carrera, docente, semestre, institución, fecha y número de avance.
<b>Índice</b>	Generado automáticamente. Incluye tablas y figuras.
<b>Introducción</b>	Breve descripción del proyecto, contexto y propósito del informe.
<b>Objetivos</b>	Específicos y medibles (tipo SMART). Reflejan lo que se busca demostrar en el análisis y diseño.
<b>Metodología de desarrollo</b>	Explica el método utilizado (Lean, XP, TDD, etc.) y justifica su elección.
<b>Análisis de requerimientos</b>	Requerimientos funcionales y no funcionales actualizados, clasificados y trazados a los diagramas.
<b>Diseño del sistema</b>	Diagramas MER, Casos de Uso, Actividad o Flujo, y Wireframes. Cada uno numerado y con título descriptivo.
<b>Repositorio y gestión técnica</b>	Capturas del repositorio GitHub, estructura de carpetas, tags, releases y tablero Kanban.
<b>Conclusiones y mejoras</b>	Reflexión sobre la experiencia de desarrollo y oportunidades de mejora.

## Refuerzo

Cada sección debe estar **claramente identificada con título y numeración**. No se aceptan informes sin estructura ni coherencia entre análisis, diseño y repositorio.

## 6.2. Redacción técnica profesional

### Información Importante

La redacción técnica debe ser **objetiva, precisa y coherente**. Se recomienda usar un tono impersonal y activo: describir lo que se hace, no quién lo hace.<sup>a</sup>

<sup>a</sup>**Technical Writing:** estilo de escritura profesional usado en documentación de ingeniería y tecnología.

### Ejemplo Correcto

#### Ejemplos de buena redacción:

- “El sistema permite registrar clientes y generar órdenes de trabajo de forma automática.”
- “El modelo MER fue diseñado considerando la normalización hasta 3FN.”
- “El proceso se implementó siguiendo el método Lean para reducir iteraciones innecesarias.”

### Error Común

#### Errores comunes:

- “Yo hice un programa que...” → Evitar primera persona.
- “El sistema está súper bueno.” → Evitar lenguaje coloquial.
- “Ver diagrama más abajo.” → Usar referencias: “Como se muestra en la Figura ??.”

## Refuerzo

Usa recursos de LaTeX para mantener profesionalismo: referencias cruzadas (6.2), numeración automática, y en tablas y figuras.

### 6.3. Integración de evidencias

#### Información Importante

Cada afirmación o diseño debe tener una **evidencia visual o técnica**. Esto incluye diagramas, fragmentos de código o SQL, capturas de repositorio, tablero Kanban y releases.

```
-- Ejemplo de fragmento SQL a incluir --
CREATE TABLE Cliente (
    id_cliente INT PRIMARY KEY,
    nombre VARCHAR(50),
    telefono VARCHAR(15)
);
```

#### Atención

Toda evidencia debe tener un título y número de figura o tabla. No insertar imágenes sin contexto ni explicación.

#### Refuerzo

Incluye también enlaces al repositorio y releases. Ejemplo: <https://github.com/usuario/TallerProyectoEspecialidad/releases/tag/v2.0>

### 6.4. Control de calidad del informe

#### Información Importante

Antes de entregar, realiza una revisión técnica, visual y de coherencia. Usa esta lista de control:<sup>a</sup>

<sup>a</sup>**Checklist:** lista estructurada de verificación para asegurar la calidad del documento.

Aspecto a revisar	Qué verificar
Estructura general	Todas las secciones presentes y en orden lógico.
Claridad y coherencia	Los diagramas coinciden con los requerimientos descritos.
Redacción técnica	Lenguaje formal, sin errores ortográficos ni coloquiales.
Numeración y referencias	Todas las figuras y tablas numeradas y referenciadas.
Evidencias visuales	Imágenes legibles, bien explicadas y con fuente.
Repositorio	Tag de versión correcto (v2.0), estructura clara y README actualizado.

## Error Común

### No entregar el informe si:

- Faltan diagramas o están incongruentes con los requerimientos.
- No hay capturas del repositorio o tablero Kanban.
- El informe está incompleto o con secciones vacías.

## 6.5. Entrega formal y presentación final

### Información Importante

La entrega incluye tres elementos integrados:

1. Informe PDF completo en carpeta `/docs` del repositorio.
2. Repositorio GitHub actualizado con tag `v2.0` y release publicado.
3. Presentación oral breve (**pitch**) de 5 minutos.<sup>a</sup>

<sup>a</sup>**Pitch:** exposición breve y enfocada para comunicar el valor y funcionalidad del proyecto.

### Refuerzo

Durante la presentación oral, el estudiante debe:

- Mostrar el prototipo funcionando (mínimo una función CRUD completa).
- Explicar en 1 minuto el problema y la solución propuesta.
- Describir brevemente la arquitectura y el método usado.
- Destacar una decisión técnica relevante o mejora implementada.

### Atención

Para aprobar el Avance 2, el informe debe cumplir con los cuatro ejes principales:

1. **Presentación:** formato limpio, títulos y numeración clara, ortografía correcta.
2. **Estructura:** secciones completas, coherentes y ordenadas.
3. **Coherencia técnica:** consistencia entre requerimientos, diseño y evidencia en código/repositorio.
4. **Evidencias:** diagramas, capturas, commits y releases verificables en GitHub.

*“Documentar bien no es un trámite, es la prueba visible de tu competencia técnica.”*

## 6.6. Complementos visuales y checklist técnico

### Información Importante

Para reforzar la comprensión global del Avance 2, se agregan a continuación tres elementos de apoyo: un mapa integrador del informe, un subapartado dedicado al control de versiones y evidencia GitHub, y una figura resumen que muestra el flujo completo del trabajo desde los requerimientos iniciales hasta la publicación del release final.

## Mapa integrador del informe

### Información Importante

Esta figura resume la relación entre los distintos componentes del Informe 2. Cada etapa produce entregables que se vinculan con la siguiente, asegurando coherencia entre el análisis, el diseño y la implementación.

### Información Importante

La siguiente figura sintetiza la estructura completa del Informe 2, mostrando cómo cada componente del trabajo —desde los requerimientos iniciales hasta la gestión técnica en GitHub— se relaciona con las etapas de análisis, diseño y documentación. Este esquema permite visualizar de forma clara la trazabilidad del proyecto y cómo cada entregable depende de los resultados obtenidos en las unidades anteriores.

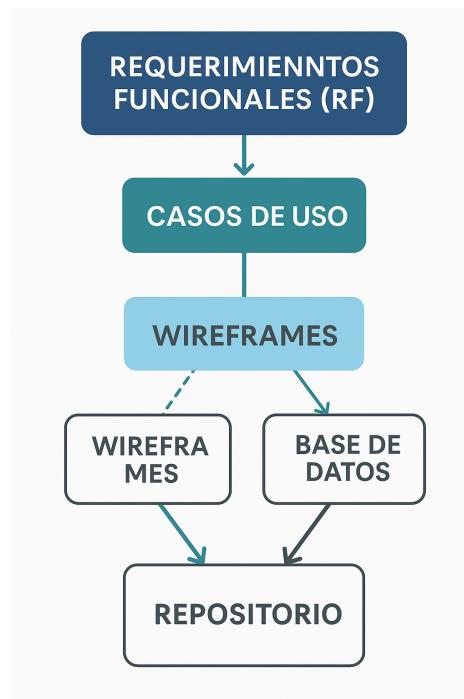


Figura 11: Mapa integrador del Informe 2: conexión entre entregables.

## Refuerzo

El flujo muestra la secuencia lógica del desarrollo:

- **Requerimientos Funcionales y No Funcionales (RF/RNF)** → definen qué hará el sistema.
- **Casos de Uso y Diagramas de Actividad** → representan cómo el usuario interactúa.
- **Wireframes y Diseño Visual** → anticipan la experiencia de uso.
- **Base de Datos (MER y SQL)** → implementan la estructura lógica.
- **Repositorio GitHub** → centraliza los resultados, control de versiones y entregas.

## 6.7. Control de versiones y evidencia GitHub

### Información Importante

El informe debe contener un apartado específico que evidencie el uso profesional de Git y GitHub como parte del proceso de trabajo. Este elemento será evaluado dentro del checklist final del Informe 2.

- Captura y enlace del **repositorio en GitHub**.
- Evidencia de uso de **tags, commits y ramas cortas**.
- Captura del **tablero Kanban de GitHub Projects**.
- Registro de **releases oficiales** (por ejemplo: v2.0, v3.0).
- Breve reflexión sobre cómo el control de versiones *facilitó* el trabajo individual.

### Ejemplo Correcto

**Ejemplo correcto:** incluir enlace al repositorio con releases y capturas del Kanban en la sección “Gestión técnica y control de versiones” del informe.

### Error Común

**Ejemplo incorrecto:** solo mencionar que “se usó GitHub” sin evidencias ni estructura de carpetas o releases visibles.

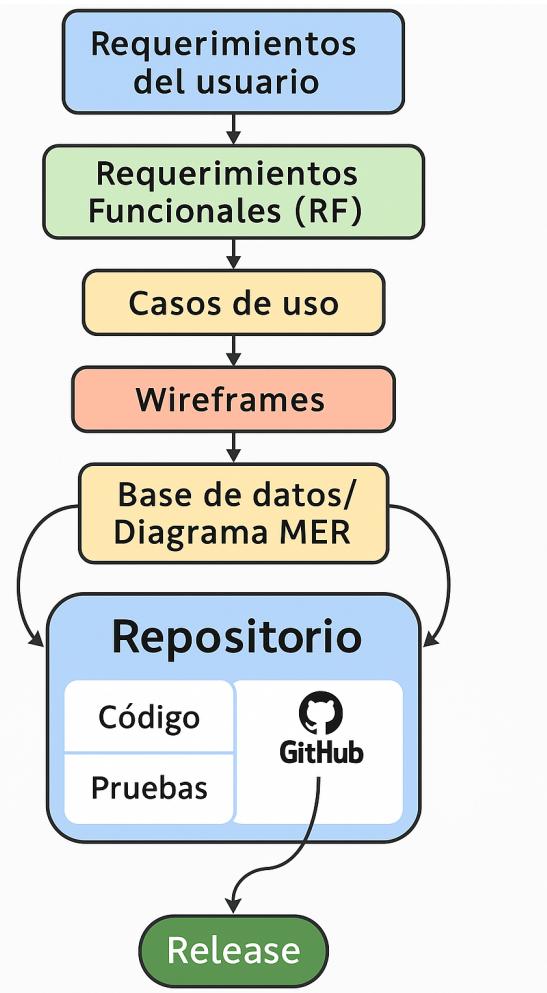


Figura 12: Flujo general del Avance 2: de los requerimientos a la publicación del release.

### Refuerzo

Esta figura puede usarse como guía de revisión antes de entregar el Informe 2. Si todos los pasos están cubiertos, el trabajo cumple con la trazabilidad técnica esperada para un proyecto profesional de análisis y diseño.

## 7. Unidad 7: Reflexión, buenas prácticas y proyección al desarrollo

### Información Importante

Esta unidad marca el cierre del proceso de análisis y diseño del proyecto, y prepara al estudiante para iniciar el desarrollo del prototipo funcional (Avance 3).

Su propósito es consolidar los aprendizajes adquiridos, reflexionar sobre la experiencia vivida y reforzar las buenas prácticas que acompañarán al futuro profesional técnico en programación y análisis de sistemas.

### 7.1. Reflexión sobre el proceso

#### Información Importante

Reflexionar sobre el propio trabajo es una práctica fundamental en la formación técnica. Permite reconocer logros, errores y aprendizajes, fortaleciendo la autonomía profesional.

- ¿Qué aspectos del análisis y diseño fueron los más desafiantes?
- ¿Qué decisiones técnicas resultaron más acertadas?
- ¿Qué errores no volverías a cometer en un proyecto real?
- ¿Cómo influyó la organización del tiempo y el trabajo individual?

#### Refuerzo

Redacta una reflexión personal (media página) respondiendo estas preguntas. Esta autoevaluación será parte de la introducción del Informe 3 (prototipo).

### 7.2. Buenas prácticas técnicas y éticas

#### Información Importante

Un buen profesional técnico no solo sabe programar, sino que también comprende la responsabilidad de su trabajo. Las buenas prácticas son hábitos que garantizan calidad, transparencia y sostenibilidad en el desarrollo de software.

- Mantén la trazabilidad del código y los documentos.
- Usa nombres descriptivos en archivos, carpetas y variables.
- Evita eliminar el historial del repositorio (nunca usar `--force` sin razón).
- Respeta las licencias y créditos de herramientas o librerías utilizadas.<sup>1</sup>

<sup>1</sup>Basado en el pensamiento de Eric Raymond en \*La catedral y el bazar\*: el conocimiento y la colaboración abierta fortalecen el software libre.

- Aplica principios de **Software Craftsmanship** y **Clean Code**.<sup>2</sup>

### Ejemplo Correcto

#### Ejemplo de práctica responsable:

- Subir cada avance al repositorio con mensajes claros de commit.
- Documentar las funciones principales en el código.
- Respetar los plazos y la estructura de carpetas del proyecto.

### Error Común

#### Ejemplo de práctica incorrecta:

- Subir versiones incompletas sin descripción.
- Copiar código sin citar su origen.
- Modificar el historial del repositorio para ocultar errores.

## 7.3. Preparación para el desarrollo (Avance 3)

### Información Importante

El Avance 3 corresponde al **prototipo funcional**, donde se implementa el sistema diseñado. Es fundamental mantener la coherencia entre el diseño aprobado y el código desarrollado.

- Revisa los requerimientos finales aprobados en el Informe 2.
- Define los módulos iniciales a implementar (CRUD, autenticación, reportes, etc.).
- Organiza subcarpetas dentro de `/src` según los componentes del sistema.
- Planifica commits semanales con objetivos claros (por ejemplo, “Implementa CRUD Clientes completo”).
- Usa el tablero Kanban de GitHub para registrar el avance de cada módulo.

```
/src
  clientes/      # módulo de gestión de clientes
  vehiculos/    # módulo de vehículos y mantenimientos
  ordenes/       # módulo CRUD de órdenes de trabajo
  static/        # recursos (CSS, JS, imágenes)
  templates/    # vistas HTML o plantillas
```

<sup>2</sup>**Software Craftsmanship:** movimiento profesional que enfatiza la calidad, la ética y la mejora continua en el desarrollo de software.

## Refuerzo

Define una meta de desarrollo semanal. No esperes al final del semestre: el progreso constante asegura un prototipo funcional y estable.

## 7.4. Errores frecuentes y lecciones aprendidas

### Información Importante

Aprender de los errores es parte del proceso de madurez técnica. Evitar repetirlos es lo que diferencia a un principiante de un profesional responsable.

Error común	Consecuencia y solución
Falta de coherencia entre requerimientos y código	El sistema no cumple lo diseñado → Revisar trazabilidad entre RF/RNF y módulos.
Diseño incompleto o inconsistente	El prototipo queda sin base sólida → Validar diagramas antes de programar.
Prototipo improvisado sin planificación	Causa errores y sobrecarga de trabajo → Usar commits y Kanban.
No documentar ni versionar	Dificulta evaluación y mantenimiento → Mantener README y mensajes de commit claros.

## Refuerzo

Actividad: redacta una lista personal titulada “*Tres errores que no repetire*” y adjúntala al final de tu Informe 2 como anexo breve.

## 7.5. Cierre y proyección profesional

### Información Importante

El trabajo realizado en este taller refleja las competencias reales de un Técnico en Programación y Análisis de Sistemas. La calidad, organización y claridad del proceso son indicadores directos de desempeño profesional.

- Mantén tus hábitos de documentación en futuros proyectos.
- Usa GitHub como portafolio profesional para mostrar tu trabajo.
- Refuerza tu autonomía: investiga, mejora y comparte tus conocimientos.
- Aplica estas prácticas en entornos laborales y proyectos personales.

## Atención

El éxito del prototipo dependerá directamente de la solidez del diseño, la disciplina en la ejecución y la capacidad de comunicar los resultados.

*“El desarrollo no comienza con código, sino con la comprensión de lo que se quiere construir.”*

*“Diseñar bien es prever los problemas antes de que existan.”*