Guía de Ayuda — Primera Entrega (Avance 1)

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:	1.0 (AIEP)

Índice

Propósito de la guía **3**

1. Alcance del Avance 1 **3**
1.1. Resultados esperados . . . 3

2. Definición del problema **4**
2.1. 1. ¿Qué debe tener la descripción del problema? . . . 4
2.2. 2. Estructura recomendada . . . 4
2.3. 3. Ejemplos prácticos . . . 4
2.4. 4. Errores comunes a evitar . . . 5
2.5. 5. Actividad práctica . . . 6

3. Formulación de objetivos SMART **7**
3.1. 1. ¿Qué es un objetivo? . . . 7
3.2. 2. Los cinco criterios SMART . . . 7
3.3. 3. Ejemplos de redacción . . . 8
3.4. 4. Guía paso a paso para redactar tus objetivos . . . 9
3.5. 5. Actividad guiada . . . 9
3.6. 6. Errores frecuentes y cómo corregirlos . . . 10
3.7. 7. Cómo evaluar si tus objetivos se cumplieron . . . 12
3.8. 1. Indicadores cuantitativos y cualitativos . . . 12
3.9. 2. Ejemplos de relación entre objetivos e indicadores . . . 12
3.10. 3. Cómo crear tus propios indicadores . . . 13
3.11. 8. Autoevaluación y revisión de objetivos SMART . . . 14
3.12. 1. Tabla de autoevaluación SMART . . . 14
3.13. 2. Actividad de revisión entre pares . . . 14
3.14. 3. Revisión final y ajustes . . . 15

4. Innovación y valor agregado del proyecto **16**
4.1. 1. Qué significa innovar en este taller . . . 16
4.2. 2. Tipos de innovación aplicables a tu proyecto . . . 16
4.3. 3. Ejemplos de innovación (buenos y malos) . . . 17
4.4. 4. Cómo justificar tu innovación . . . 17
4.5. 5. Actividad práctica: define tu innovación . . . 18

5. Software libre, *La Catedral y el Bazar* y elección de licencias **19**
5.1. 1. La metáfora de la catedral y el bazar . . . 19
5.2. 2. Licencias de software libre . . . 19
5.3. 3. Cómo elegir tu licencia . . . 20
5.4. 4. Actividad práctica: redacta tu párrafo de innovación y licencia . . . 20

6. Diagramas de procesos: fundamentos y tipos **21**
6.1. 1. ¿Por qué modelar el proceso? . . . 21
6.2. 2. Diagrama de Flujo de Datos (DFD) . . . 21
6.3. 3. Diagrama de Actividades (UML) . . . 22
6.4. 4. Comparativa entre DFD y Diagrama de Actividades . . . 23
6.5. 5. Actividad práctica: identifica el proceso . . . 23

7. Creación del propio diagrama de procesos **24**
7.1. 1. Pasos para crear tu diagrama . . . 24
7.2. 2. Herramientas recomendadas . . . 24
7.3. 3. Ejemplo ilustrativo (DFD) . . . 25
7.4. 4. Ejemplo ilustrativo (Diagrama de Actividades UML) . . . 25
7.5. 5. Verificación del diagrama . . . 26
7.6. 6. Actividad práctica: crea tu diagrama . . . 26

8. Uso de GitHub en el proyecto **27**
8.1. 1. Conceptos básicos de GitHub . . . 27
8.2. 2. Estructura recomendada del repositorio . . . 27
8.3. 3. Buenas prácticas con commits y carpetas . . . 28
8.4. 4. Cómo evaluar tu propio repositorio . . . 28
8.5. 5. Actividad práctica: crear y configurar el repositorio . . . 29

9. Gestión visual del avance con Kanban **30**
9.1. 1. Estructura básica del tablero Kanban . . . 30
9.2. 2. Reglas básicas de Kanban . . . 30
9.3. 3. Kanban digital: GitHub Projects y Trello . . . 31
9.4. 4. Ejemplo práctico de tablero en GitHub Projects . . . 31
9.5. 5. Actividad práctica: crea tu tablero . . . 32

10. Checklist del Avance 1 y cierre **33**
10.1. 1. Elementos obligatorios del Avance 1 . . . 33
10.2. 2. Criterios de evaluación formativa . . . 33
10.3. 3. Recomendaciones antes de entregar . . . 34
10.4. 4. Cierre del bloque y preparación para el Avance 2 . . . 34

11. Glosario técnico y proyección al Avance 2 **2**
11.1. 1. Glosario técnico general . . . 2
11.2. 2. Consejos para el Avance 2 (diseño y prototipo) . . . 3
11.3. 3. Nota docente final . . . 3

Propósito de la guía

Esta guía te acompaña en la preparación de la **Primera Entrega (Avance 1)** del Taller. Al finalizar, habrás definido el **problema**, redactado **objetivos SMART**, justificado la **innovación**, comprendido el enfoque de **software libre** y configurado tu entorno de trabajo en **GitHub** (repo, README, Issues y tablero Kanban).

Refuerzo obligatorio en clase

La Primera Entrega **define el rumbo de todo tu proyecto**. Es clave que salgas de esta etapa con una base clara, coherente y documentada en tu repositorio.

1. Alcance del Avance 1

Objetivo del avance: Dejar establecida la base conceptual y técnica del proyecto: *problema*, *objetivos SMART*, *aporte innovador*, *reflexión de software libre + licencia* y *diagrama de procesos*; además del *entorno GitHub*.

1.1. Resultados esperados

Al cerrar el Avance 1, deberás:

- Redactar un **problema** claro, acotado y con contexto real.
- Formular **objetivos SMART** (1 general y 2–3 específicos).
- Describir el **aporte innovador** (mejora o valor agregado).
- Elaborar una breve **reflexión de software libre** y elegir **licencia** (MIT, GPL o Apache).
- Construir un **diagrama de procesos** (DFD o Actividades).
- Crear y configurar tu **repositorio GitHub**: README, Issues, tablero Kanban.

Atención

Error común: comenzar describiendo la *solución* sin haber definido el **problema**. En esta etapa se prioriza el *qué* y el *para qué*, no el *cómo programar*.

2. Definición del problema

Antes de programar o diseñar diagramas, todo proyecto comienza con la **definición del problema**. Un *problema* es una **necesidad real o dificultad observable** que requiere ser resuelta mediante un sistema de información[a]. Si el problema no está bien planteado, el resto del proyecto se volverá confuso o poco útil.

[a]Un *sistema de información* es un conjunto organizado de datos, procesos y recursos tecnológicos que permiten registrar, procesar y entregar información útil a los usuarios.

Refuerzo obligatorio en clase

El estudiante debe comprender que **no se trata de inventar una idea de software**, sino de **identificar una situación real que justifique su creación**. La claridad del problema es la base de todos los objetivos SMART y del diseño posterior.

2.1. 1. ¿Qué debe tener la descripción del problema?

- **Contexto:** dónde ocurre y a quién afecta (institución, empresa, persona).
- **Situación actual:** cómo se realiza hoy la tarea o proceso[1].
- **Dificultad o limitación:** qué aspectos generan pérdida de tiempo, errores o desorden.
- **Consecuencia:** qué problemas genera esta situación.
- **Necesidad o propósito:** qué se espera mejorar mediante la solución tecnológica.

2.2. 2. Estructura recomendada

El texto del problema puede escribirse en tres párrafos breves:

1. Describir el **contexto general**.
2. Explicar la **situación problemática concreta**.
3. Señalar la **consecuencia o impacto** y la necesidad de cambio.

2.3. 3. Ejemplos prácticos

Ejemplo incorrecto

Ejemplo mal redactado:

"Mi sistema permitirá registrar los alumnos y las notas."

Problema: No describe ninguna dificultad real. Se enfoca en la *solución*, no en el problema. No hay contexto, ni consecuencia, ni motivo.

[1]En análisis de sistemas, un *proceso* es una secuencia de pasos o actividades que transforman entradas en salidas.

Ejemplo positivo

Ejemplo correcto:

"En el Instituto San Luis, el registro de notas de los alumnos se realiza de manera manual en planillas de papel. Esto genera pérdida de tiempo y errores al calcular los promedios, además de dificultad para recuperar información de periodos anteriores. Se requiere un sistema que permita registrar, calcular y consultar las calificaciones de manera digital y segura."

Por qué está bien: presenta contexto, describe la situación, menciona consecuencias y sugiere la necesidad de mejora sin proponer aún la solución.

Ejemplo incorrecto

Ejemplo incorrecto 2:

"Crearé un sistema de inventario con base de datos."

Problema: Es una frase de propósito, no de diagnóstico. No dice qué falla, dónde, ni por qué se necesita ese sistema.

Ejemplo positivo

Ejemplo correcto 2:

"En el taller de la familia Paredes, los repuestos se registran en hojas sueltas. No existe control del stock ni fechas de ingreso o salida, lo que provoca pérdida de piezas y compras innecesarias. Se necesita una solución digital que organice los repuestos y permita controlar su disponibilidad."

Por qué está bien: se centra en la situación actual y en la consecuencia concreta que justifica el proyecto.

2.4. 4. Errores comunes a evitar

- Comenzar el texto con "El sistema que voy a hacer..." (ya sería solución).
- Describir un problema demasiado amplio o genérico.
- Repetir frases sin datos concretos.
- Escribir sin mencionar quién tiene el problema (actor principal[2]).

Atención

El problema debe ser **realista y verificable**. Evita situaciones imposibles de comprobar (por ejemplo, "mejorar la felicidad de las personas") o ajenas al alcance de un proyecto técnico individual.

[2]En modelado de sistemas, un *actor* es una persona o entidad externa que interactúa con el sistema, como un usuario o cliente.

2.5. 5. Actividad práctica

Ejercicio: Redacta tu problema en tres pasos.

1. Describe el contexto (¿dónde y quién?).
2. Explica qué falla o se hace de forma ineficiente.
3. Menciona qué consecuencias trae y qué se espera mejorar.

Escribe tu párrafo final en el informe del Avance 1 y guárdalo en tu repositorio GitHub.

Refuerzo obligatorio en clase

El docente debe revisar que cada estudiante:

- Presente un problema específico, no una idea general de sistema.
- Identifique un contexto y actores reales.
- Explique consecuencias concretas (tiempo, errores, pérdidas, duplicidad).

3. Formulación de objetivos SMART

Una vez que el problema está bien definido, el siguiente paso es establecer los **objetivos del proyecto**. Los objetivos representan lo que se desea *lograr* con el sistema o solución propuesta. Sin embargo, deben cumplir ciertas características para ser útiles: deben ser **SMART**[a].

[a]SMART es un acrónimo en inglés que resume cinco criterios de calidad para los objetivos: Specific (Específico), Measurable (Medible), Achievable (Alcanzable), Relevant (Relevante) y Time-bound (Limitado en el tiempo).

Refuerzo obligatorio en clase

Esta etapa es **clave para el éxito del proyecto**. La mayoría de los sistemas mal diseñados fallan porque los objetivos fueron vagos, irreales o imposibles de evaluar. Un objetivo mal definido genera un software que no resuelve nada concreto.

3.1. 1. ¿Qué es un objetivo?

Un **objetivo** es una meta que se busca alcanzar como resultado de las acciones del proyecto. Debe responder claramente a la pregunta:

¿Qué se quiere lograr y para qué?

En el Taller se utilizarán dos niveles:

- **Objetivo general:** expresa el propósito central del proyecto.
- **Objetivos específicos:** dividen el objetivo general en partes concretas, medibles y operativas.

3.2. 2. Los cinco criterios SMART

Criterio	Descripción y ejemplos
S – Specific (Específico)	Debe indicar claramente qué se va a lograr, en qué ámbito y con qué propósito. *Ejemplo: "Diseñar un sistema de control de inventario para el taller mecánico Paredes."*
M – Measurable (Medible)	Se debe poder comprobar si se logró. Incluye resultados observables o cuantificables. *Ejemplo: "Reducir los errores de registro de stock en un 80 %."*
A – Achievable (Alcanzable)	El objetivo debe ser posible de lograr con los recursos disponibles y dentro del tiempo del módulo (90 h académicas).
R – Relevant (Relevante)	Debe tener valor real: resolver el problema identificado y contribuir a la mejora del proceso[3].
T – Time-bound (Limitado en el tiempo)	Indica un plazo o etapa temporal para alcanzar el resultado. *Ejemplo: "Implementar el prototipo funcional al cierre del semestre académico 2025-1."*

Atención

Si un objetivo no puede ser medido ni comprobado, no es SMART. Evita frases como "mejorar", "optimizar" o "modernizar" si no explicas *cómo sabrás* que lo lograste.

3.3. 3. Ejemplos de redacción

Ejemplo incorrecto

Objetivo general incorrecto:

"Crear un sistema para registrar clientes."

Problema: No dice qué necesidad resuelve, no tiene contexto, ni se puede medir el éxito. **No cumple:** Específico (S), Medible (M), ni Relevante (R).

Ejemplo positivo

Objetivo general correcto:

"Desarrollar un sistema web que permita registrar y consultar los datos de los clientes del taller mecánico Paredes, con el fin de reducir los errores de registro y mejorar la atención al público durante el semestre 2025-1."

Cumple: Específico, Medible, Alcanzable, Relevante y Limitado en el tiempo.

Ejemplo incorrecto

Objetivos específicos mal redactados:

- Realizar un sistema de base de datos.
- Que tenga interfaz bonita.
- Usar PHP y MySQL.

Problema: Describen tareas o tecnologías, no resultados medibles.

Ejemplo positivo

Ejemplo de objetivos específicos correctos:

- Analizar los requerimientos del proceso de registro de clientes y ventas del taller.
- Diseñar la estructura de base de datos que permita almacenar los datos de manera consistente.
- Implementar formularios de ingreso y consulta para minimizar errores de digitación.
- Probar el prototipo con datos reales y documentar los resultados.

Por qué están bien: Cada uno es una acción concreta y medible que contribuye al objetivo general.

3.4. 4. Guía paso a paso para redactar tus objetivos

Pasos recomendados:

1. Revisa tu problema y subraya las palabras clave (errores, demoras, pérdidas, etc.).
2. Escribe una frase que exprese lo que quieres lograr: ese será tu objetivo general.
3. Divide esa meta en 3–4 pasos lógicos: esos serán tus objetivos específicos.
4. Asegúrate de que cada objetivo cumpla con los 5 criterios SMART.
5. Relee todo y verifica que los objetivos específicos realmente apoyen al general.

3.5. 5. Actividad guiada

Ejercicio práctico: reformular objetivos.
Lee los siguientes ejemplos y mejóralos para que cumplan con las características SMART:

- "Optimizar la atención al cliente."
- "Diseñar una aplicación para registrar pedidos."
- "Mejorar la gestión de inventario en la empresa."

Reescribe cada uno de manera que sea medible, alcanzable y con plazo definido. Incluye tus versiones en el informe de la primera entrega.

Refuerzo obligatorio en clase

Durante la revisión de avances, el docente debe comprobar que:

- Cada estudiante distingue claramente entre objetivo general y específicos.
- Todos los objetivos son medibles y alcanzables en el semestre.
- Ningún objetivo menciona tecnologías o lenguajes (eso pertenece a la etapa técnica).

3.6. 6. Errores frecuentes y cómo corregirlos

Muchos proyectos fracasan porque los objetivos se formulan de manera vaga o centrada en la tecnología. Aquí se muestran los errores más comunes y cómo corregirlos aplicando los criterios SMART.

Ejemplo incorrecto

Error 1: Objetivo demasiado general

"Mejorar el sistema de inventario."

Problema: No dice qué aspecto se mejorará, ni en qué plazo, ni cómo se medirá el resultado.

Ejemplo positivo

Corrección:

"Desarrollar un sistema de inventario para el taller mecánico Paredes que reduzca en un 70 % los errores de registro y controle el stock de repuestos durante el semestre 2025-1."

Por qué cumple: Es específico, medible, alcanzable y tiene límite temporal.

Ejemplo incorrecto

Error 2: Objetivo centrado en la tecnología

"Crear un sistema en PHP con base de datos MySQL."

Problema: Menciona herramientas, no metas. El lenguaje no es un objetivo, sino un medio.

Ejemplo positivo

Corrección:

"Implementar un sistema web para registrar las ventas y controlar los productos del almacén, utilizando tecnologías adecuadas que garanticen el registro seguro y consultas rápidas."

Por qué cumple: Describe resultados esperados, no solo las tecnologías empleadas.

Ejemplo incorrecto

Error 3: Objetivo no medible

"Optimizar la atención al cliente en la empresa."

Problema: No se puede comprobar cuándo "optimizar" se ha logrado.

Ejemplo positivo

Corrección:

"Diseñar un módulo de atención al cliente que reduzca en un 40 % el tiempo promedio de registro de pedidos, permitiendo un seguimiento digital de cada solicitud."

Por qué cumple: Incluye indicador cuantificable y resultado observable.

Ejemplo incorrecto

Error 4: Objetivo irreal o demasiado ambicioso

"Implementar una plataforma regional de ventas con inteligencia artificial."

Problema: Fuera del alcance de un proyecto individual y del tiempo del módulo.

Ejemplo positivo

Corrección:

"Diseñar un prototipo web de control de ventas y stock local, aplicando principios de usabilidad e integrando reportes automáticos básicos."

Por qué cumple: Es realista, ajustado al tiempo y a las competencias de un técnico en programación.

Atención

Evita confundir **tareas** con **objetivos**. Ejemplo: "Crear base de datos" o "Hacer interfaz" son actividades, no metas del proyecto.

Refuerzo obligatorio en clase

Durante la revisión en clase, muestra varios ejemplos (buenos y malos) y pídele a los alumnos que identifiquen cuál cumple con cada letra de SMART. Este ejercicio grupal refuerza la comprensión práctica.

3.7. 7. Cómo evaluar si tus objetivos se cumplieron

Un objetivo SMART solo tiene sentido si puedes demostrar que lo alcanzaste. Para eso se utilizan los **indicadores de logro**[a]. Los indicadores sirven para verificar, en la fase final del taller, que el sistema realmente resolvió el problema definido.

[a] Un *indicador de logro* es una medida cuantitativa o cualitativa que permite comprobar si un objetivo se cumplió. Por ejemplo, el tiempo promedio de registro, la cantidad de errores detectados o la satisfacción del usuario.

Refuerzo obligatorio en clase

Desde esta etapa inicial los estudiantes deben pensar **cómo sabrán que su sistema funcionó**. Cada objetivo debe tener una forma de comprobarse (un número, porcentaje o evidencia concreta).

3.8. 1. Indicadores cuantitativos y cualitativos

- **Cuantitativos:** se expresan con números, porcentajes o tiempos. Ejemplo: "Reducir en un 50 % los errores de registro."
- **Cualitativos:** se expresan mediante apreciaciones o niveles de satisfacción. Ejemplo: "Mejorar la usabilidad del sistema según encuesta aplicada a tres usuarios."

En este taller se recomiendan principalmente **indicadores cuantitativos**, porque son más fáciles de comprobar en un prototipo funcional.

3.9. 2. Ejemplos de relación entre objetivos e indicadores

Objetivo SMART	Indicador de cumplimiento
Reducir los errores de registro de stock en un 70 %.	% de errores antes y después del sistema.
Disminuir el tiempo promedio de atención al cliente en un 40 %.	Tiempo medio de registro de pedido (minutos).
Controlar digitalmente los préstamos de herramientas.	Cantidad de registros correctos generados / total de préstamos.
Implementar reportes automáticos de ventas.	Existencia y funcionamiento del módulo de reportes en el prototipo.
Aumentar la satisfacción del usuario.	Resultado de encuesta simple (1 a 5).

Atención

No todos los objetivos necesitan un número exacto, pero **todos deben ser comprobables**. Evita expresiones como "mejorar mucho", "hacer más rápido" o "optimizar", sin aclarar cómo se medirá.

3.10. 3. Cómo crear tus propios indicadores

1. Lee cada objetivo y piensa: "¿Qué podría medir para saber si lo cumplí?".
2. Anota una métrica posible (porcentaje, cantidad, tiempo o evidencia).
3. Verifica que la medición sea factible en tu proyecto individual.
4. Registra esos indicadores en tu informe del Avance 1 (al final de la sección de objetivos).

Ejemplo positivo

Ejemplo:
Objetivo: "Reducir el tiempo de registro de ventas." Indicador: "Tiempo promedio (en minutos) que tarda un usuario en registrar una venta antes y después del sistema." Forma de medición: "Cronometrar tres registros manuales y tres registros con el sistema."

Refuerzo obligatorio en clase

Los indicadores serán reutilizados más adelante en la fase de **plan de pruebas y evaluación final del prototipo**. Por eso es importante que queden definidos desde ahora en el informe de la primera entrega.

3.11. 8. Autoevaluación y revisión de objetivos SMART

Antes de continuar con el desarrollo de tu proyecto, es importante revisar tus objetivos con mirada crítica. Una buena autoevaluación evita errores de enfoque y asegura que tus metas sean realmente alcanzables dentro del tiempo del módulo.

Refuerzo obligatorio en clase

Los objetivos SMART son la brújula del proyecto: si apuntan mal, todo el trabajo se desviará. Revisarlos con método y sinceridad es parte del proceso profesional.

3.12. 1. Tabla de autoevaluación SMART

Completa la siguiente tabla para verificar si tus objetivos cumplen cada criterio SMART. Usa o y anota brevemente cómo podrías mejorarlos.

Objetivo	S	M	A	R	T	Comentario / Mejora
Objetivo general						
Objetivo específico 1						
Objetivo específico 2						
Objetivo específico 3						

S: Specific (Específico), M: Measurable (Medible), A: Achievable (Alcanzable), R: Relevant (Relevante), T: Time-bound (Limitado en el tiempo).

Atención

Un objetivo con en dos o más columnas debe reescribirse. Si el estudiante no puede explicar cómo sabrá que lo cumplió, el objetivo no es SMART.

3.13. 2. Actividad de revisión entre pares

Dinámica sugerida: En clase, intercambia tus objetivos con un compañero y evalúen mutuamente con la tabla anterior. Cada estudiante debe dejar un comentario escrito sobre:

- Qué objetivo está mejor formulado y por qué.
- Qué objetivo podría mejorarse.
- Si todos los objetivos apoyan el problema identificado.

Propósito: aprender a analizar críticamente, no solo redactar.

Ejemplo positivo

Ejemplo de retroalimentación entre pares:

"Tu objetivo general es claro y medible, pero los específicos no indican cómo comprobarás el éxito. Podrías agregar un porcentaje o un plazo para cada uno."

3.14. 3. Revisión final y ajustes

Luego de la revisión personal y por pares, reescribe tus objetivos definitivos y agrégalos al informe del Avance 1. Guarda una copia en tu repositorio GitHub junto con el archivo del informe (PDF).

Refuerzo obligatorio en clase

Tu docente revisará esta sección y podrá pedir correcciones antes de aprobar el Avance 1. Cada objetivo debe ser coherente con el problema y posible de comprobar durante el semestre.

4. Innovación y valor agregado del proyecto

En el contexto de la programación y análisis de sistemas, **innovar** no siempre significa crear algo completamente nuevo. Muchas veces, innovar consiste en **mejorar, simplificar o automatizar** un proceso que ya existe[a]. El valor agregado surge cuando el proyecto aporta una mejora real a la organización, al usuario o al entorno.

[a] En informática, se entiende por *proceso* un conjunto de pasos o actividades coordinadas que transforman entradas (datos) en salidas (información útil).

Refuerzo obligatorio en clase

El alumno debe comprender que la innovación no se mide por la complejidad técnica, sino por la **utilidad, originalidad y aporte práctico** que ofrece su sistema dentro de un contexto real.

4.1. 1. Qué significa innovar en este taller

- **Innovar** es introducir una mejora observable en la forma en que se realiza una tarea o proceso.
- Implica **resolver un problema** de manera más eficiente, rápida, segura o accesible.
- Requiere creatividad y análisis, no necesariamente nuevas tecnologías.
- Puede basarse en ideas existentes, siempre que se adapten o combinen de una manera distinta.

Ejemplo: si una empresa registra pedidos en planillas Excel y un alumno crea una aplicación que automatiza ese proceso, ya está innovando, aunque use tecnologías comunes.

4.2. 2. Tipos de innovación aplicables a tu proyecto

Tipo de innovación	Descripción y ejemplos
Técnica	Introducir nuevas herramientas, lenguajes o tecnologías en la solución. Ejemplo: usar una base de datos relacional bien diseñada o implementar validaciones de seguridad que antes no existían.
Funcional	Mejorar la manera en que el usuario realiza una tarea o accede a la información. Ejemplo: pasar de un registro manual a un sistema con reportes automáticos y alertas.
Social / Organizacional	Promover la accesibilidad, transparencia o colaboración mediante tecnología. Ejemplo: permitir que varios usuarios trabajen simultáneamente o que los datos estén disponibles desde distintos dispositivos.

Atención

No se considera innovación copiar una aplicación existente sin adaptarla o comprender su funcionamiento. El valor está en la **adaptación al contexto real** del problema que el alumno eligió.

4.3. 3. Ejemplos de innovación (buenos y malos)

Ejemplo incorrecto

Ejemplo incorrecto:
"Mi innovación será usar una base de datos MySQL." **Problema:** usar una herramienta común no implica innovación. Lo importante es qué mejora genera en el proceso.

Ejemplo positivo

Ejemplo correcto:
"Mi innovación consiste en incorporar validación automática de datos para evitar errores de digitación, lo que reducirá un 80 % los registros defectuosos." **Por qué cumple:** describe una mejora funcional medible y vinculada al problema real.

Ejemplo incorrecto

Ejemplo incorrecto:
"Haré un sistema igual al que ya tiene la empresa, pero con otros colores." **Problema:** cambiar la apariencia sin modificar el proceso no es innovación.

Ejemplo positivo

Ejemplo correcto:
"Haré un sistema que envíe notificaciones automáticas cuando un producto esté por agotarse." **Por qué cumple:** agrega una función nueva que mejora la toma de decisiones.

4.4. 4. Cómo justificar tu innovación

Para incluirlo correctamente en el informe del Avance 1, redacta un párrafo breve (5–8 líneas) respondiendo a estas preguntas:

1. ¿Qué problema o limitación concreta resuelve tu innovación?
2. ¿Qué mejora o valor aporta a los usuarios?
3. ¿Por qué tu solución es mejor que el método actual?
4. ¿Qué hace diferente a tu proyecto frente a una solución tradicional?

Ejemplo positivo

Ejemplo de redacción correcta:
"El proyecto aporta innovación al automatizar el registro y control de herramientas mediante un sistema web accesible desde dispositivos móviles. Esto permite reducir el tiempo de búsqueda de información y evita extravíos, mejorando la trazabilidad de los equipos."

Atención

Evita usar frases vagas como "mi innovación es hacerlo más moderno" o "mi sistema es diferente". Toda afirmación de innovación debe **fundamentarse en una mejora comprobable**.

4.5. 5. Actividad práctica: define tu innovación

Refuerzo obligatorio en clase

Ejercicio guiado:

1. Relee el problema que definiste en tu informe.
2. Identifica qué parte del proceso será diferente gracias a tu sistema.
3. Escribe una frase que empiece con "La innovación de mi proyecto consiste en..."
4. Revisa si tu idea se relaciona con alguno de los tres tipos (técnica, funcional, social).
5. Añade ese párrafo en el informe del Avance 1, sección "Innovación y valor agregado".

5. Software libre, *La Catedral y el Bazar* y elección de licencias

La innovación no solo depende de la tecnología, sino también de la **forma en que se comparte el conocimiento**. El movimiento del *software libre*[a] propone un modelo colaborativo que permite aprender, mejorar y crear de manera abierta. Esta filosofía es la base del pensamiento presentado por Eric S. Raymond en su ensayo ***La Catedral y el Bazar*** (1997).

[a]Se denomina *software libre* a aquel que puede ser usado, modificado y distribuido libremente, siempre que se respeten las condiciones de su licencia. No debe confundirse con "software gratuito", ya que lo libre se refiere a la libertad del usuario, no al precio.

Refuerzo obligatorio en clase

Comprender la diferencia entre los modelos de desarrollo cerrados y abiertos ayuda a valorar la innovación como un proceso colectivo. Usar **GitHub** en este taller replica ese mismo modelo de colaboración moderna.

5.1. 1. La metáfora de la catedral y el bazar

Modelo	Características principales
Catedral	Desarrollo cerrado, planificado por un grupo reducido. El código se libera solo cuando está "perfecto". Ejemplo: software comercial propietario.
Bazar	Desarrollo abierto y colaborativo, donde cualquiera puede aportar mejoras o detectar errores. El código se comparte desde el inicio y evoluciona con la comunidad. Ejemplo: proyectos de código abierto en GitHub o Linux.

Atención

El modelo **Bazar** no significa desorden, sino apertura: los proyectos funcionan gracias a reglas claras, comunicación constante y responsabilidad compartida.

Ejemplo positivo

Ejemplo aplicado a este taller: Cada estudiante desarrolla su propio sistema (trabajo individual), pero comparte el progreso, los commits y la documentación en GitHub. Así, el docente y los compañeros pueden revisar, comentar y aprender del trabajo ajeno: un bazar académico.

5.2. 2. Licencias de software libre

Cuando un desarrollador crea un programa, automáticamente posee derechos de autor. Para permitir que otros lo usen, debe aplicar una **licencia**, que indica las condiciones de uso, modificación y distribución. Las más comunes en proyectos académicos y colaborativos son MIT, GPL y Apache.

Licencia	Características principales y cuándo usarla
MIT	Muy permisiva. Permite usar, copiar, modificar y distribuir el código incluso con fines comerciales, siempre que se mantenga el aviso de autoría. Ideal para proyectos educativos y personales.
GPL (GNU Public License)	Requiere que cualquier derivado también sea libre (copyleft). Asegura que nadie cierre el código. Ideal para proyectos comunitarios o académicos que promueven colaboración.
Apache 2.0	Similar a MIT pero con cláusulas de protección de patentes. Muy usada en entornos empresariales.

Para este taller se recomienda usar la licencia **MIT** o **GPL**, por su claridad y compatibilidad con GitHub. El texto completo de cada licencia se puede agregar al repositorio como archivo `LICENSE`.

5.3. 3. Cómo elegir tu licencia

1. Si deseas compartir libremente tu código sin restricciones: usa **MIT**.
2. Si prefieres asegurar que siempre siga siendo libre: usa **GPL**.
3. Si te interesa permitir usos más empresariales: usa **Apache 2.0**.

Ejemplo positivo

Ejemplo: "El proyecto se publicará en GitHub bajo licencia MIT, permitiendo que otros estudiantes puedan revisarlo, modificarlo y reutilizarlo con fines académicos, siempre reconociendo su autoría original."

Atención

No aplicar una licencia equivale a mantener todos los derechos reservados por defecto. En ese caso, nadie podría legalmente usar o modificar el código, incluso dentro de la institución.

5.4. 4. Actividad práctica: redacta tu párrafo de innovación y licencia

Refuerzo obligatorio en clase

Agrega al informe del Avance 1 un párrafo que combine:

- **La innovación principal** de tu proyecto.
- **El tipo de mejora** (técnica, funcional o social).
- **La licencia elegida** y su propósito.

Ejemplo:
"El sistema propuesto introduce innovación funcional al automatizar la gestión de reservas mediante recordatorios por correo. Se distribuye bajo licencia MIT para permitir su uso educativo y su adaptación por otros estudiantes."

6. Diagramas de procesos: fundamentos y tipos

Antes de escribir una sola línea de código, es necesario comprender **cómo funciona el proceso que se desea automatizar**. Los **diagramas de procesos** permiten representar visualmente el flujo de información, las tareas y las decisiones que ocurren dentro de un sistema[a]. Estos modelos ayudan a analizar, comunicar y validar la idea antes del desarrollo.

[a]Un *flujo de información* describe cómo los datos se mueven entre los actores, procesos y almacenes dentro de un sistema.

Refuerzo obligatorio en clase

El modelado visual es una herramienta de análisis, no de diseño gráfico. Su propósito es entender el funcionamiento lógico del sistema y detectar errores o redundancias antes de programar.

6.1. 1. ¿Por qué modelar el proceso?

- Permite visualizar **entradas**, **procesos** y **salidas** de manera ordenada.
- Ayuda a detectar pasos innecesarios o tareas duplicadas.
- Facilita la comunicación entre el programador y los usuarios del sistema.
- Sirve como base para crear la base de datos y la interfaz del software.

En este taller se recomienda utilizar uno de los siguientes enfoques de modelado:

1. **Diagrama de Flujo de Datos (DFD)** — ideal para representar cómo se mueve la información.
2. **Diagrama de Actividades (UML)** — útil para visualizar las acciones y decisiones del proceso.

Ambos son válidos; el alumno puede elegir el que le resulte más comprensible.

6.2. 2. Diagrama de Flujo de Datos (DFD)

El **DFD** muestra cómo los datos entran, se transforman y salen de un sistema. Es especialmente útil para sistemas CRUD (Crear, Leer, Actualizar, Eliminar)[a].

[a]El acrónimo CRUD se refiere a las cuatro operaciones básicas que realiza un sistema sobre los datos: *Create*, *Read*, *Update*, *Delete*.

Elementos básicos del DFD:

- **Entidades externas:** personas u organismos que envían o reciben información (por ejemplo, cliente, proveedor, estudiante).
- **Procesos:** transformaciones que ocurren dentro del sistema (registrar pedido, calcular promedio, generar reporte).
- **Flujos de datos:** líneas con flechas que indican el movimiento de la información.
- **Almacenes de datos:** lugares donde la información se guarda o recupera (bases de datos, archivos, planillas).

Ejemplo positivo

Ejemplo básico:
En un sistema de biblioteca:

- Entrada: solicitud de préstamo de libro.
- Proceso: verificar disponibilidad.
- Almacén: base de datos de libros.
- Salida: préstamo autorizado o rechazado.

Ejemplo incorrecto

Error frecuente: confundir un DFD con un diagrama de flujo de programa. El DFD no muestra estructuras de control (if, while), sino el **movimiento de información**.

6.3. 3. Diagrama de Actividades (UML)

El **Diagrama de Actividades** pertenece al lenguaje UML[a] y muestra el flujo de acciones o tareas dentro de un proceso, destacando decisiones, bifurcaciones y paralelismos. Es ideal para representar la **secuencia lógica de tareas** del usuario o del sistema.

[a] UML (Unified Modeling Language) es un lenguaje estandarizado para modelar sistemas de software.

Elementos principales:

- Inicio y fin del proceso.
- Actividades (acciones o tareas).
- Decisiones (condiciones o bifurcaciones).
- Conectores (uniones de flujos).
- Swimlanes (opcional): separan las acciones por actor o responsable.

Ejemplo positivo

Ejemplo:

1. El cliente solicita una reserva.
2. El sistema verifica disponibilidad.
3. Si hay cupo, se confirma la reserva y se envía correo.
4. Si no, se informa la falta de cupo y se ofrece alternativa.

Este flujo puede representarse gráficamente como un diagrama de actividades.

Atención

Aunque UML es un estándar profesional, en este taller se busca claridad, no complejidad. El diagrama debe ser comprensible para cualquier persona, incluso sin conocimientos técnicos.

Aspecto	**DFD** vs. **Diagrama de Actividades**
Foco principal	Movimiento y transformación de datos (*qué información entra, se procesa y sale*). vs. Secuencia de acciones y decisiones (*qué tareas se ejecutan y en qué orden*).
Ideal para	Analizar entradas, salidas y almacenes de datos. vs. Visualizar flujos de trabajo y lógica de decisiones.
Símbolos	Entidades, procesos, flujos, almacenes. vs. Actividades, nodos de decisión, conectores.
Uso recomendado	Sistemas CRUD o administrativos simples. vs. Procesos con múltiples caminos o actores.

6.4. 4. Comparativa entre DFD y Diagrama de Actividades

6.5. 5. Actividad práctica: identifica el proceso

Refuerzo obligatorio en clase

Ejercicio:

1. Escoge tu proyecto y escribe las principales tareas que realizará el sistema.
2. Clasifícalas en tres categorías: **entrada**, **proceso** y **salida**.
3. Dibuja un borrador en papel o pizarra digital mostrando cómo se relacionan.
4. Decide si usarás DFD o diagrama de actividades para tu informe.

7. Creación del propio diagrama de procesos

Una vez que hayas comprendido los tipos de diagramas posibles, llega el momento de construir el modelo que representa tu proyecto. El objetivo no es dibujar "bonito", sino mostrar de manera lógica y ordenada cómo fluye la información o se ejecutan las tareas dentro del sistema.

Refuerzo obligatorio en clase

Un diagrama claro puede explicar tu idea incluso mejor que un texto largo. Debe ser posible entenderlo sin leer todo el informe, solo observando el flujo general.

7.1. 1. Pasos para crear tu diagrama

1. **Define el alcance del proceso.** Determina dónde comienza y termina. No intentes representar todo el sistema, solo el flujo principal. Ejemplo: "Desde que el usuario inicia sesión hasta que genera un reporte".
2. **Lista las entradas y salidas.** Identifica qué datos llegan al sistema y qué información entrega como resultado.
3. **Enumera las tareas o pasos intermedios.** Pueden ser procesos internos (calcular, validar, registrar).
4. **Identifica los actores.** Son las personas o entidades externas que interactúan con el sistema (cliente, empleado, docente).
5. **Elige el tipo de diagrama.** Usa DFD si te interesa mostrar el flujo de datos, o diagrama de actividades si el proceso tiene decisiones o caminos alternativos.
6. **Dibuja el borrador.** En papel, pizarra o software, ordena los elementos de izquierda a derecha (entrada → proceso → salida).
7. **Verifica coherencia.** Asegúrate de que cada proceso tenga entrada y salida, y que los flujos sean consistentes.

Atención

Un diagrama sin entradas o sin salidas no tiene sentido: todo proceso debe **transformar algo**. Revisa que cada flujo de datos lleve información útil y no quede "colgando".

7.2. 2. Herramientas recomendadas

Herramienta	Características principales
Draw.io / diagrams.net	Gratuita, en línea, sin registro. Permite exportar a imagen o PDF. Ideal para DFD y UML simples.
Lucidchart	Online, colaborativa. Buena para diagramas con varios actores. Tiene versión gratuita limitada.
Mermaid (Markdown)	Permite crear diagramas mediante texto, compatible con GitHub. Ideal para repositorios.
LibreOffice Draw / PowerPoint	Alternativa local sencilla para crear diagramas sin conexión.

Guarda siempre el diagrama como imagen (`.png` o `.jpg`) y súbelo a tu carpeta o repositorio GitHub junto con el informe. Nombralo con un identificador claro: `diagrama_proceso.png`.

7.3. 3. Ejemplo ilustrativo (DFD)

Ejemplo positivo

Ejemplo de DFD simplificado: Sistema de reservas

- **Entrada:** solicitud de reserva de cliente.
- **Proceso:** verificar disponibilidad y registrar reserva.
- **Almacén:** base de datos de reservas.
- **Salida:** confirmación enviada al cliente.

Visualmente, puede representarse con cuatro bloques conectados por flechas. El DFD nivel 0 debería incluir entre 3 y 5 procesos principales.

Ejemplo incorrecto

Ejemplo incorrecto:
Un DFD con 15 procesos distintos y sin relación entre ellos. **Problema:** demasiados elementos impiden ver el flujo principal; se pierde la función comunicativa del diagrama.

7.4. 4. Ejemplo ilustrativo (Diagrama de Actividades UML)

Ejemplo positivo

Ejemplo: Sistema de control de asistencia

1. El empleado marca entrada.
2. El sistema valida usuario.
3. Si es correcto, registra la hora y muestra mensaje de confirmación.
4. Si no, solicita verificación del supervisor.
5. Fin del proceso.

El diagrama mostrará estas actividades conectadas por flechas y decisiones (rombos) según la validación.

Atención

No se necesitan herramientas costosas ni diagramas complejos. La claridad y coherencia valen más que los íconos o colores. El objetivo es **entender y comunicar el proceso**, no decorarlo.

7.5. 5. Verificación del diagrama

Antes de entregar tu diagrama, verifica:

- ¿Está claramente indicado el inicio y el fin del proceso?
- ¿Cada proceso tiene entradas y salidas?
- ¿Se representa el flujo principal, no todos los detalles?
- ¿Coincide con el problema y los objetivos definidos en el informe?

Si todas las respuestas son "sí", tu diagrama está listo para incluirlo.

7.6. 6. Actividad práctica: crea tu diagrama

Refuerzo obligatorio en clase

Ejercicio guiado:

1. Elige el tipo de diagrama (DFD o Actividades).
2. Representa el flujo principal de tu sistema con 3–6 elementos.
3. Guarda la imagen e insértala en tu informe PDF (Avance 1).
4. Sube también el archivo fuente o imagen al repositorio GitHub.
5. Agrega una breve descripción debajo del diagrama: "El siguiente modelo representa el proceso de ... indicando las entradas, procesos y salidas principales."

8. Uso de GitHub en el proyecto

GitHub es una plataforma que permite almacenar, versionar y compartir código fuente en la nube. En este taller se utiliza como **evidencia de avance y organización del trabajo**, tal como ocurre en entornos profesionales de desarrollo. Cada estudiante debe mantener su repositorio personal actualizado, con toda la información del proyecto: informes, diagramas, código y documentación.

Refuerzo obligatorio en clase

El repositorio de GitHub no solo sirve para subir archivos. Es un registro histórico del esfuerzo, la constancia y la mejora del estudiante a lo largo del semestre.

8.1. 1. Conceptos básicos de GitHub

- **Repositorio:** espacio donde se guarda todo el contenido del proyecto (archivos, carpetas, historial de cambios).
- **Commit:** registro o "fotografía" de un cambio realizado en el código o documento.
- **Push:** acción de enviar los cambios locales al repositorio en la nube.
- **Branch:** rama del proyecto donde se pueden probar ideas sin alterar la versión principal.
- **README:** archivo que describe el propósito, instalación y uso del proyecto.
- **Issue:** nota o registro de problema, mejora o tarea pendiente.

Consejo: estos términos forman parte del lenguaje común de los equipos de desarrollo, incluso en proyectos individuales. Aprenderlos es parte de la formación profesional.

Repositorio obligatorio: Cada alumno debe crear un repositorio llamado:
`taller-proyecto-apellido-nombre`
Ejemplo: `taller-proyecto-perez-juan`
El repositorio debe ser **público** y contener al menos una carpeta por entrega.

8.2. 2. Estructura recomendada del repositorio

Carpeta / archivo	Contenido sugerido
`/documentos`	Informes en PDF (Avance 1, Avance 2, Avance Final).
`/diagramas`	Imágenes o archivos fuente de DFD o UML.
`/codigo`	Archivos del programa o prototipo desarrollado.
`/assets`	Imágenes, logotipos o recursos visuales del sistema.
`README.md`	Descripción general del proyecto (propósito, autor, tecnología, licencia).
`LICENSE`	Archivo de licencia (MIT o GPL).

Atención

El archivo `README.md` es **obligatorio** y forma parte de la evaluación. Debe incluir:

- Nombre del proyecto y autor.
- Breve descripción del problema y la solución propuesta.
- Tecnologías utilizadas.
- Instrucciones básicas de uso.
- Enlace al informe PDF y licencias.

8.3. 3. Buenas prácticas con commits y carpetas

Ejemplo positivo

Buen ejemplo de mensajes de commit:

- "Agregada tabla de clientes en la base de datos."
- "Actualizado diagrama de actividades."
- "Corrección de errores en formulario de registro."

Ejemplo incorrecto

Mal ejemplo:

- "cosas nuevas"
- "arreglo algo"
- "subiendo todo"

Estos mensajes no comunican claramente los cambios realizados y dificultan el seguimiento del progreso.

Realiza al menos un **commit y push por semana**. Esto demuestra constancia y permite ver la evolución del trabajo. Evita subir todo el proyecto en un solo día, ya que no se evidenciará el proceso.

8.4. 4. Cómo evaluar tu propio repositorio

Refuerzo obligatorio en clase

Antes de cada entrega, revisa tu repositorio y responde:

- ¿Los nombres de carpetas y archivos son claros y coherentes?
- ¿El `README.md` explica bien el proyecto?
- ¿Tienes commits frecuentes y descriptivos?
- ¿El informe, diagramas y código están actualizados y visibles?
- ¿Tu licencia está incluida correctamente?

Si todas las respuestas son "sí", tu repositorio está correctamente mantenido.

8.5. 5. Actividad práctica: crear y configurar el repositorio

Refuerzo obligatorio en clase

Ejercicio:

1. Ingresa a https://github.com y crea tu cuenta (si no la tienes).
2. Crea un nuevo repositorio llamado `taller-proyecto-apellido-nombre`.
3. Marca la opción "Public" y "Add a README file".
4. Sube al menos un archivo del proyecto (por ejemplo, el informe del Avance 1).
5. Crea un archivo de licencia usando la plantilla "MIT License".
6. Envía el enlace del repositorio al docente por formulario o correo.

Ejemplo positivo

Ejemplo de URL correcta:
https://github.com/juanperez/taller-proyecto-perez-juan

9. Gestión visual del avance con Kanban

El método **Kanban** permite organizar visualmente las tareas de un proyecto en columnas que representan su estado de avance. Su objetivo es hacer visible el flujo de trabajo y promover la mejora continua. En este taller, cada estudiante utilizará Kanban para planificar y controlar sus actividades, ya sea en GitHub Projects, Trello o una pizarra física.

Refuerzo obligatorio en clase

Kanban es parte de las **metodologías ágiles**[a], pero puede aplicarse incluso en proyectos individuales. Su poder está en la **simplicidad y visualización**: todos pueden ver de un vistazo qué está pendiente, en progreso o completado.

[a] Las *metodologías ágiles* son enfoques de trabajo que priorizan la entrega continua de valor y la adaptación rápida al cambio.

9.1. 1. Estructura básica del tablero Kanban

azulUbb!10**Por hacer**	grisClaro!30**En progreso**	verdeClaro!25**Terminado**
azulUbb!10Definir problema	grisClaro!30Redactar objetivos SMART	verdeClaro!25Problema aprobado
azulUbb!10Analizar requerimientos	grisClaro!30Diseñar diagrama DFD	verdeClaro!25Informe Avance 1 entregado
azulUbb!10Diseñar base de datos	grisClaro!30Crear repositorio GitHub	verdeClaro!25README completo

Cuadro 1: Ejemplo de tablero Kanban con tareas típicas del taller.

Atención

Cada tarea debe ser concreta y medible. Evita frases vagas como “trabajar en el sistema”. Prefiere enunciados específicos: “Crear formulario de registro”, “Probar inserción de datos”.

9.2. 2. Reglas básicas de Kanban

- Las tareas se escriben en tarjetas o post-its digitales.
- Cada tarjeta se mueve de izquierda a derecha según su progreso.
- Evita acumular demasiadas tareas en “En progreso”: indica sobrecarga.
- Una tarea “terminada” debe cumplir los criterios de calidad definidos (funciona, está probada o entregada).

Ejemplo incorrecto

Ejemplo incorrecto:
El estudiante llena la columna "Por hacer" con 20 tareas y no mueve ninguna durante semanas. **Problema:** el tablero deja de reflejar el progreso real.

Ejemplo positivo

Ejemplo correcto:
El tablero tiene entre 5 y 8 tareas activas. Cada semana, al menos dos cambian de columna. El flujo de trabajo es claro y progresivo.

9.3. 3. Kanban digital: GitHub Projects y Trello

GitHub Projects permite crear tableros vinculados al repositorio. Cada tarjeta puede asociarse a un archivo, *issue* o commit específico. Ideal para mantener todo en un solo entorno.
Trello, en cambio, es una herramienta externa, gratuita y más visual. Permite añadir colores, etiquetas y fechas límite, útil para quienes prefieren una interfaz gráfica simple.

Atención

Sea cual sea la herramienta, lo importante es la **disciplina de actualización**. Un tablero abandonado es tan inútil como un código sin versión. Debe reflejar el estado real del proyecto.

9.4. 4. Ejemplo práctico de tablero en GitHub Projects

azulUbb!10**Por hacer**	grisClaro!30**En progreso**	verdeClaro!25**Terminado**
azulUbb!10Subir diagrama DFD	grisClaro!30Redactar innovación	verdeClaro!25Crear README.md
azulUbb!10Agregar licencia MIT	grisClaro!30Ajustar commits	verdeClaro!25Publicar Avance 1 en repositorio

Cuadro 2: Tablero Kanban en GitHub Projects (vista simplificada).

Refuerzo obligatorio en clase

Cada tarjeta puede vincularse con un commit, un *issue* o un archivo específico. Esto permite trazar la relación entre el avance visible y el progreso técnico.

9.5. 5. Actividad práctica: crea tu tablero

Refuerzo obligatorio en clase

Ejercicio guiado:

1. Crea un tablero Kanban en GitHub Projects (o Trello).
2. Define tres columnas: **Por hacer**, **En progreso**, **Terminado**.
3. Agrega al menos 6 tareas de tu proyecto actual.
4. Mueve semanalmente las tareas según su avance.
5. Toma una captura del tablero y agrégala al informe o al repositorio.

Ejemplo positivo

Ejemplo de tareas iniciales:

- "Definir problema del sistema de inventario."
- "Redactar objetivos SMART."
- "Diseñar diagrama de procesos."
- "Crear estructura del repositorio GitHub."
- "Agregar licencia y README."
- "Registrar innovación en informe."

10. Checklist del Avance 1 y cierre

El **Avance 1** es la primera entrega formal del proyecto y representa la mitad del trabajo total del taller. Debe evidenciar el análisis, la planificación y la organización del estudiante, además de los primeros elementos técnicos. Este listado te ayudará a verificar que no falte ningún componente antes de entregar.

Refuerzo obligatorio en clase

El informe del Avance 1 equivale al 50 % de la nota final. El otro 50 % corresponde al prototipo funcional que se desarrollará en las siguientes etapas.

10.1. 1. Elementos obligatorios del Avance 1

azulUbb!10**Elemento a entregar**	**Verificado**	
Título y breve descripción del proyecto	[ ]	
Definición del problema y contexto	[ ]	
Objetivos SMART formulados correctamente	[ ]	
Párrafo de innovación y valor agregado	[ ]	
Diagrama de procesos (DFD o Actividades UML)	[ ]	
Repositorio GitHub configurado y activo	[ ]	
Archivo `README.md` completo	[ ]	
Licencia (MIT o GPL) agregada	[ ]	
Tablero Kanban actualizado (captura o enlace)	[ ]	
Informe en PDF subido a GitHub	[ ]	

Cuadro 3: Checklist de componentes del Avance 1.

Atención

Cada ítem marcado con [] debe tener evidencia real: un archivo, una imagen o un enlace verificado en GitHub. Si un elemento está "pendiente", no se considera completo.

10.2. 2. Criterios de evaluación formativa

Refuerzo obligatorio en clase

Este mismo cuadro puede utilizarse como pauta de evaluación docente o como herramienta de autoevaluación del alumno.

grisClaro!40**Criterio**	**Indicadores observables**
Claridad del problema	Se entiende el contexto, actores y necesidad.
Coherencia de objetivos SMART	Son medibles, alcanzables y relevantes.
Innovación y valor agregado	Presenta una mejora o aporte verificable.
Modelado del proceso	El diagrama es legible y corresponde al problema.
Organización en GitHub	Repositorio ordenado, commits frecuentes y mensajes claros.
Gestión visual (Kanban)	Tablero actualizado con tareas realistas.
Presentación del informe	Redacción clara, ortografía cuidada y formato coherente.

Cuadro 4: Criterios formativos sugeridos para el Avance 1.

10.3. 3. Recomendaciones antes de entregar

- Revisa ortografía y formato antes de exportar el informe a PDF.
- Verifica que todos los archivos estén visibles en GitHub.
- Actualiza el `README.md` con la descripción más reciente.
- Adjunta la captura del tablero Kanban (GitHub Projects o Trello).
- Comprueba que la licencia esté incluida correctamente.
- Realiza un último `commit` con el mensaje "Entrega Avance 1".

10.4. 4. Cierre del bloque y preparación para el Avance 2

Con este avance se completa la fase de análisis y planificación. El siguiente bloque se enfocará en el **diseño funcional y desarrollo del prototipo**. Los elementos elaborados hasta ahora (problema, objetivos, diagrama, GitHub y Kanban) serán la base del desarrollo posterior.

Taller de Proyecto de la Especialidad

Avance 1: Análisis, Innovación y Planificación

Carrera: Técnico en Programación y Análisis de Sistemas

Instituto Profesional AIEP

Estudiante: _____

Docente: _____

Semestre: _____

Fecha de entrega: _____

"El conocimiento técnico se fortalece cuando se comprende el propósito del sistema antes de programarlo."

El **Avance 1** del Taller de Proyecto de la Especialidad representa la primera gran etapa del proceso formativo: el paso desde la idea inicial hasta la definición clara y fundamentada del problema, sus objetivos y su alcance técnico. Durante este avance, el estudiante demuestra su capacidad para analizar, planificar y comunicar un proyecto de software con criterios profesionales.

Refuerzo obligatorio en clase

A lo largo de este bloque, el alumno ha:

- Identificado un problema real o una necesidad que puede resolverse mediante tecnología.
- Formulado objetivos **SMART** coherentes y medibles.
- Propuesto una innovación funcional, técnica o social vinculada al contexto.
- Representado gráficamente el proceso mediante un **DFD o diagrama de actividades**.
- Creado su repositorio en **GitHub** con estructura organizada, licencia y README.
- Gestionado sus tareas mediante un tablero **Kanban** digital o físico.

Cada elemento evidencia competencias de análisis, comunicación técnica y organización profesional.

Atención

El éxito del prototipo que se desarrollará en la siguiente etapa dependerá de la claridad, coherencia y realismo alcanzados en este análisis inicial. Todo sistema bien diseñado parte de un diagnóstico preciso.

"Todo gran sistema comienza con un buen análisis."

— Cierre del Avance 1 —

11. Glosario técnico y proyección al Avance 2

Este glosario resume los principales conceptos utilizados a lo largo del **Avance 1**. Sirve como referencia rápida para comprender los términos técnicos y metodológicos más relevantes del taller. La comprensión de este vocabulario es esencial para enfrentar la siguiente fase del proyecto: el desarrollo del prototipo funcional.

11.1. 1. Glosario técnico general

azulUbb!10**Término**	**Definición**
azulUbb!10**SMART**	Acrónimo que define objetivos *Específicos, Medibles, Alcanzables, Relevantes y con Tiempo definido*. Permite formular metas claras y verificables en proyectos de software.
azulUbb!10**CRUD**	Conjunto de operaciones básicas de un sistema: *Create, Read, Update, Delete* (crear, leer, actualizar, eliminar). Base de la mayoría de los sistemas de gestión.
azulUbb!10**DFD (Diagrama de Flujo de Datos)**	Representación gráfica del movimiento de la información dentro de un sistema, mostrando entradas, procesos, almacenes y salidas.
azulUbb!10**UML (Unified Modeling Language)**	Lenguaje estandarizado para modelar sistemas de software. En este taller se usa principalmente el *diagrama de actividades* para representar procesos.
azulUbb!10**Innovación**	Incorporación de mejoras funcionales, técnicas o sociales que aportan valor real al usuario o a la organización.
azulUbb!10**GitHub**	Plataforma de desarrollo colaborativo basada en control de versiones. Permite registrar avances, documentar código y compartir proyectos.
azulUbb!10**Commit**	Registro puntual de cambios realizados en un archivo o grupo de archivos dentro de un repositorio. Ayuda a llevar control y trazabilidad del trabajo.
azulUbb!10**Kanban**	Método visual de gestión de tareas basado en columnas (“Por hacer”, “En progreso”, “Terminado”). Facilita la organización y el seguimiento del proyecto.
azulUbb!10**Licencia MIT / GPL**	Documentos legales que definen las condiciones para usar, modificar y compartir un software. MIT es más permisiva; GPL exige mantener el código libre.
azulUbb!10**Software libre**	Modelo de desarrollo que promueve la colaboración, la transparencia y la libertad de uso y modificación del código.

Cuadro 5: Glosario técnico resumido del Taller de Proyecto de la Especialidad.

Refuerzo obligatorio en clase

Este glosario puede actualizarse en las siguientes etapas del taller para incluir nuevos términos relacionados con diseño de interfaces, bases de datos y pruebas de software.

11.2. 2. Consejos para el Avance 2 (diseño y prototipo)

El **Avance 2** se centrará en el diseño funcional y el desarrollo del prototipo del sistema. Para avanzar con éxito, se recomienda:

- Revisar el problema y los objetivos SMART para asegurar que el diseño responda a ellos.
- Comenzar por crear las tablas y estructuras de datos necesarias.
- Elaborar los primeros formularios o pantallas (wireframes) para validar el flujo del sistema.
- Mantener actualizado el repositorio GitHub con commits frecuentes y mensajes claros.
- Continuar usando el tablero Kanban para planificar las tareas semanales.
- Probar cada módulo antes de integrarlo al prototipo final.

Atención

Recuerda que el prototipo no necesita ser un sistema terminado, sino una demostración funcional de los principales procesos definidos en el Avance 1. La claridad y la coherencia son más importantes que la cantidad de funciones.

11.3. 3. Nota docente final

"Analizar antes de programar, planificar antes de ejecutar, y compartir para mejorar."

— Cierre general del Taller de Proyecto de la Especialidad —