Tarea 1: Sistema de Evaluaciones Basado en la Taxonomía de Bloom

Sebastián Márquez Fernando Quilaqueo Vicente Silva Matías Guajardo

Abril 2025

1. Introducción

Este taller presenta un sistema en C++ diseñado para facilitar la creación y gestión de evaluaciones escritas, basado en la Taxonomía de Bloom. Propuesta por Benjamin Bloom en 1956 y revisada por Anderson et al. en 2001, esta taxonomía organiza los objetivos educativos en seis niveles cognitivos: Recordar, Entender, Aplicar, Analizar, Evaluar y Crear [1, 2]. El sistema permite realizar operaciones CRUD (crear, consultar, actualizar, eliminar) sobre preguntas, generar evaluaciones filtradas por nivel taxonómico, calcular el tiempo estimado de resolución, y evitar la repetición de preguntas en años consecutivos.

La motivación es optimizar el proceso de evaluación para docentes, combinando rigor pedagógico con eficiencia técnica. Este informe detalla la solución implementada, describe el diseño orientado a objetos, presenta el diagrama de clases, verifica el cumplimiento de los objetivos, y reflexiona sobre los desafíos, limitaciones y posibles mejoras.

2. Descripción de la solución

El sistema está implementado en C++ con una interfaz por consola, garantizando simplicidad y accesibilidad. Utiliza un diseño orientado a objetos con clases modulares que encapsulan datos y comportamiento. A continuación, se describen los componentes principales.

2.1. Modelo de clases

El sistema se basa en las siguientes clases, ilustradas en el diagrama UML (Figura 1):

- **Clase `Pregunta`:** Representa una pregunta individual:

```
class Pregunta {
private:
    int id;
    string enunciado;
    NivelTaxonomico nivel;
    TipoPregunta tipo;
    string respuesta;
    int tiempo;
    bool usadaAntes;
public:
    Pregunta(int id, string enunciado, NivelTaxonomico nivel, TipoPregunta
        tipo, string respuesta, int tiempo);
    int getId() const;
    NivelTaxonomico getNivel() const;
    int getTiempo() const;
    bool getUsadaAntes() const;
    void setEnunciado(string e);
    void setRespuesta(string r);
```

```
    void setTiempo(int t);
    void marcarUsada();
    void mostrar() const;
};
```

El atributo `usadaAntes` evita la repetición de preguntas en evaluaciones consecutivas.

- **Clase `Evaluacion`:** Modela una evaluación:

```
class Evaluacion {
private:
    int id;
    string titulo;
    Pregunta* preguntas[MAX_PREGUNTAS];
    int cantPreguntas;
    int tiempoTotal;
public:
    Evaluacion(int id, string titulo);
    bool agregarPregunta(Pregunta* p);
    bool eliminarPregunta(int idPregunta);
    int calcularTiempoTotal() const;
    void mostrar() const;
};
```

Usa composición con `Pregunta`, ya que las preguntas son exclusivas de cada evaluación, almacenadas en un arreglo estático de tamaño $\texttt{MAX}_P REGUNTAS$.

- **Clase `GestorEvaluaciones`:** Administra el sistema:

```
class GestorEvaluaciones {
private:
    Pregunta* preguntasDisponibles[MAX_PREGUNTAS];
    Evaluacion* evaluaciones[MAX_EVALUACIONES];
    int cantPreguntas;
    int cantEvaluaciones;
    int contadorIds;
public:
    GestorEvaluaciones();
    Pregunta* crearPregunta(string enunciado, string nivelStr, string
        respuesta, int tiempo);
    bool actualizarPregunta(int id, string enunciado, string respuesta, int
        tiempo);
    bool borrarPregunta(int id);
    Pregunta* consultarPregunta(int id);
    int buscarPorNivel(string nivelBuscado, Pregunta* resultado[], int
        maxResultados);
    Evaluacion* generarEvaluacion(int cantPreg, string nivelBuscado, string
        titulo);
    void listarPreguntas() const;
    void mostrarMenu();
    ~GestorEvaluaciones();
};
```

Implementa agregación con `Pregunta` y `Evaluacion`, permitiendo su reutilización en múltiples instancias.

- **Clases `NivelTaxonomico` y `TipoPregunta`:** Definen clasificaciones:

```
class NivelTaxonomico {
private:
```

```
    int id;
    string nombre;
public:
    NivelTaxonomico(int id, string nombre);
    string getNombre() const;
    bool operator==(const NivelTaxonomico& other) const;
};

class TipoPregunta {
private:
    int id;
    string nombre;
public:
    TipoPregunta(int id, string nombre);
    string getNombre() const;
};
```

`Pregunta` depende de estas clases para clasificar nivel y tipo.

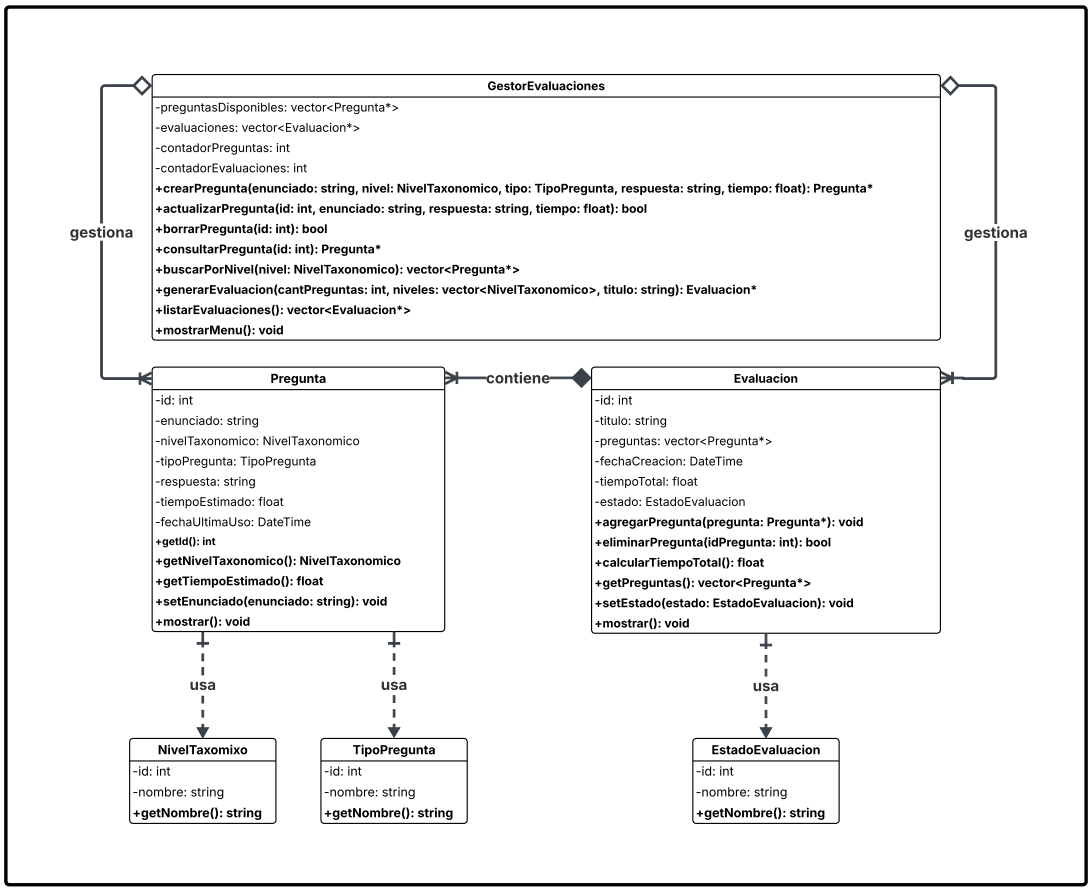

Figura 1: Diagrama UML del sistema, mostrando composición (`Evaluacion` → `Pregunta`), agregación (`GestorEvaluaciones` → `Evaluacion`/`Pregunta`), y dependencias (`Pregunta` → `NivelTaxonomico`/`TipoPregunta`).

2.2. Flujo del sistema

El sistema inicia con un menú interactivo (`GestorEvaluaciones::mostrarMenu`), que permite al usuario gestionar preguntas y evaluaciones. Un caso de uso típico, como generar una evaluación, sigue estos pasos:

1. El usuario selecciona la opción 5 ("Generar evaluación") e ingresa el nivel taxonómico (ej. .ªnalizar"), la cantidad de preguntas, y un título.

2. `GestorEvaluaciones::generarEvaluacion` invoca `buscarPorNivel`, que recorre `preguntasDisponibles` y selecciona preguntas no usadas (`getUsadaAntes == false`) con el nivel especificado.

3. Las preguntas se añaden a una nueva `Evaluacion` mediante `agregarPregunta`, que actualiza `tiempoTotal` y marca cada pregunta como usada (`marcarUsada`).

4. La evaluación se almacena en `evaluaciones` y se muestra con `Evaluacion::mostrar`, incluyendo el tiempo total.

La gestión de arreglos estáticos asegura que no se exceda $\texttt{MAX}_{P}REGUNTASni\texttt{MAX}_{E}VALUACIONES, verificando\texttt{cantPr}$

Para evitar la repetición, `Pregunta::usadaAntes` se establece en true al incluir una pregunta en una evaluación. La simulación de un reseteo anual permite al usuario marcar manualmente preguntas como no usadas, aunque no se implementa persistencia de fechas. Pseudocódigo ilustrativo:

```
void resetearUsadas(GestorEvaluaciones* gestor) {
    for (int i = 0; i < gestor->cantPreguntas; i++) {
        if (preguntasDisponibles[i]->esAntigua()) {
            preguntasDisponibles[i]->usadaAntes = false;
        }
    }
}
```

2.4. Cumplimiento de objetivos

Cuadro 1: Cumplimiento de los requisitos del enunciado.

Requisito	**Implementación**
Crear preguntas	`GestorEvaluaciones::crearPregunta`
Consultar preguntas	`GestorEvaluaciones::consultarPregunta`
Actualizar preguntas	`GestorEvaluaciones::actualizarPregunta`
Eliminar preguntas	`GestorEvaluaciones::borrarPregunta`
Buscar por nivel taxonómico	`GestorEvaluaciones::buscarPorNivel`
Generar evaluación	`GestorEvaluaciones::generarEvaluacion`
Calcular tiempo estimado	`Evaluacion::calcularTiempoTotal`
Evitar repetición	Atributo `Pregunta::usadaAntes`
Diagrama de clases	Figura 1

2.5. Resultados

Se realizaron pruebas con un banco de 10 preguntas, cubriendo niveles como Recordarz .ªnalizar". Un ejemplo generó una evaluación titulada "Prueba Analizarçon 3 preguntas de nivel .ªnalizar", sumando 15 minutos. Las capturas a continuación muestran el sistema en acción.

Link to this code: [copy]

options | compilation | execution

```
--- Menu del Sistema de Evaluacion ---
1. Agregar pregunta
2. Listar preguntas
3. Actualizar pregunta
4. Eliminar pregunta
5. Generar evaluacion
6. Marcar como usada
0. Salir
Opcion:
```

Figura 2: Menú principal en ejecución, mostrando opciones para gestionar preguntas y evaluaciones.

```
--- Menu del Sistema de Evaluacion ---
1. Agregar pregunta
2. Listar preguntas
3. Actualizar pregunta
4. Eliminar pregunta
5. Generar evaluacion
6. Marcar como usada
0. Salir
Opcion: 5
Ingrese nivel para la evaluacion: Analizar
Ingrese cantidad de preguntas: 3

--- Evaluacion generada ---
[1] ¿Porque 1+1 es igual 3? Responde segun lo visto en clases.
   Tiempo: 5 min
--------------------------
[2] ¿Quien fue Gauss y porque es el quien tiene la culpa de tus pes
   Tiempo: 5 min
--------------------------
[3] ¿Es posible aprobar un ramo con una nota en promedio de 3,95? F
   Tiempo: 5 min
--------------------------
Tiempo total estimado: 15 minutos
```

Figura 3: Ejemplo de evaluación generada, con 3 preguntas de nivel .ªnalizarz un tiempo total de 15 minutos.

3. Decisiones de diseño

Las decisiones clave reflejan un balance entre funcionalidad, claridad y alineación con POO:

- Enfoque orientado a objetos: Se usaron clases para encapsular datos y comportamiento. Por ejemplo, Pregunta::id es privado, accesible solo vía getId, garantizando integridad. La composición entre Evaluacion y Pregunta asegura exclusividad, mientras que la agregación con GestorEvaluaciones permite reutilización.
- Arreglos estáticos: Se emplearon arreglos de tamaño fijo(preguntasDisponibles[$\texttt{MAX}_P REGUNTAS$], eva
- Validación de entradas: Se implementaron verificaciones para:
  - Índices inválidos en consultarPregunta, retornando nullptr.
  - Tiempos negativos en Pregunta, ajustándolos a 0.
- Límites de arreglos en crearPregunta, verificando cantPreguntas <$\texttt{MAX}_P REGUNTAS$.
- Gestión de repetición: Pregunta::usadaAntes evita repetición inmediata. La simulación manual de reseteo anual permite flexibilidad, aunque requiere mejoras para persistencia.

4. Limitaciones

El sistema presenta las siguientes limitaciones:

- Capacidad fija: Los arreglos estáticos limitan el número de preguntas ($\texttt{MAX}_P REGUNTAS = 100) y evaluacion$

5. Conclusión

El sistema cumple todos los objetivos establecidos (Tabla 1), implementando un diseño orientado a objetos que facilita la gestión de evaluaciones. Refuerza conceptos de POO como encapsulamiento, composición, y agregación, alineándose con los aprendizajes del curso.

Los desafíos incluyeron gestionar arreglos estáticos sin exceder límites y diseñar relaciones entre clases para maximizar reutilización. Comparado con un enfoque estructurado, el diseño POO mejora la mantenibilidad y claridad.

Para mejoras futuras, se propone:

- Implementar almacenamiento en archivos para persistencia.
- Extender TipoPregunta para incluir preguntas de opción múltiple.
- Añadir una interfaz gráfica para mayor usabilidad.

El proyecto cumplió los requisitos académicos y proporcionó una experiencia valiosa en POO, destacando cómo la tecnología puede apoyar la enseñanza.

Referencias

[1] Bloom, B. S. (1956). *Taxonomy of educational objectives, handbook 1: Cognitive domain*. Addison-Wesley Longman Ltd.

[2] Anderson, L. W., Krathwohl, D. R., et al. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Pearson.