**📘 CLASE 7 – El Rol del Programador en la Gestión Ágil de Proyectos**

**Unidad IV – Metodologías Ágiles**  
**Duración: 4 horas**  
**Objetivos de la clase:**

* Analizar el rol del desarrollador dentro de equipos ágiles.
* Aplicar herramientas y rutinas típicas de un programador ágil (daily meetings, pair programming, commits).
* Simular una jornada de trabajo en equipo ágil.
* Usar GIT para organizar trabajo colaborativo.

### 🧱 BLOQUE 1 – Introducción al Rol del Programador en Ágil (Duración sugerida: 40 minutos)🎯 Propósito del bloque:

Introducir al estudiante en el rol que cumple un **programador/desarrollador dentro de un equipo ágil**, diferenciándolo del enfoque tradicional (cascada). Se busca que comprendan tanto las **responsabilidades técnicas** como las **competencias blandas** que requiere el trabajo en marcos ágiles como Scrum.

### 🎥 Video disparador (5 minutos):

**¿Qué hace un programador en metodologías ágiles?**  
<https://www.youtube.com/watch?v=drI4Gz-JHkI&t=490s> (aprox 16min)

### 🧠 Desarrollo teórico (20 minutos)

#### 📌 ¿Qué hace un programador en Scrum?

En Scrum, el programador es parte del **Development Team**, un grupo multidisciplinario y autoorganizado que:

* **Implementa historias de usuario** seleccionadas para cada Sprint.
* **Participa activamente en todas las ceremonias Scrum**: Sprint Planning, Daily Stand-Up, Review y Retrospective.
* **Se compromete colectivamente** a entregar incrementos funcionales del producto.

A diferencia del modelo tradicional, en Scrum:

* No hay un jefe directo: la gestión es horizontal.
* Las tareas no se “asignan” por imposición, sino por **autoselección colaborativa**.

#### 📌 Responsabilidades clave

1. **Desarrollo incremental**:
   * Trabajar por ciclos (sprints) para entregar funcionalidades completas y testeadas.
   * Enfocarse en objetivos pequeños, medibles y funcionales.
2. **Participación en Daily Meetings**:
   * Comunicación diaria y breve (15 min) para compartir avances, próximos pasos e impedimentos.
3. **Estimación en equipo**:
   * Participar en el Planning Poker para asignar esfuerzo relativo a cada historia.
4. **Testing e integración continua**:
   * Ejecutar pruebas automatizadas y validar entregables.
   * Mantener el código integrable y desplegable (CI/CD).

#### 📌 Habilidades blandas esenciales:

* **Comunicación efectiva**: Expresar claramente problemas, dudas y avances.
* **Colaboración**: Trabajar con otros roles (QA, PO, SM) y ayudarse mutuamente.
* **Adaptabilidad**: Cambiar tareas o prioridades según las necesidades del sprint.
* **Autonomía y compromiso**: Cumplir con las tareas asumidas dentro del sprint.
* **Escucha activa**: Fundamental en retrospectivas y refinamientos.

### 📚 Bibliografía sugerida:

* **Sommerville, I.** (2011). Ingeniería de Software. Pearson Educación. Capítulo 4: "Procesos Ágiles".
* **Pressman, R.** (2010). Ingeniería del Software: Un enfoque práctico. McGraw-Hill. Capítulos 3–6: "Equipos de desarrollo y roles en proyectos".

### 💡 Actividad inicial (15 minutos): Tormenta de ideas colaborativa

**Plataforma sugerida**: [Jamboard](https://jamboard.google.com), [Miro](https://miro.com), o Padlet

**Consigna para el grupo:**

* En una pizarra colaborativa virtual o física, cada estudiante debe responder a las siguientes dos preguntas:
  1. ¿Qué tareas creés que hace un “dev” en un equipo ágil?
  2. ¿En qué se diferencia su rol respecto a un desarrollador en cascada?

**Guía para reflexión posterior:**

* ¿Qué tareas se mencionaron más?
* ¿Notan más tareas técnicas o de comunicación?
* ¿Creen que en ágil el desarrollador tiene más o menos presión?
* ¿Cuál rol sienten que les resultaría más cómodo?

**💬 BLOQUE 2 – Simulación de una Daily Meeting (Duración: 40 minutos)**

**🎯 Objetivo del bloque:**

Que los estudiantes **practiquen una rutina Scrum real**: la **Daily Stand-Up Meeting**, donde cada integrante del equipo comunica sus avances, próximos pasos e impedimentos. Esta práctica mejora la coordinación, el compromiso y la responsabilidad compartida.

**📦 Escenario de simulación**

**Proyecto:** Desarrollo de una app para gestionar **reservas de turnos en un gimnasio**  
**Sprint:** semana 1 de desarrollo (sprint simulado de 5 días)  
**Historias del backlog (ya estimadas y priorizadas en clase 6):**

1. Como usuario, quiero ver clases disponibles.
2. Como usuario, quiero reservar un turno.
3. Como instructor, quiero bloquear horarios no disponibles.
4. Como administrador, quiero consultar historial de reservas.
5. Como sistema, quiero enviar recordatorio antes de cada clase.

**🧑‍🤝‍🧑 Asignación de roles y tareas**

Antes de la simulación:

1. Dividí a los estudiantes en equipos de 4–5 personas.
2. A cada estudiante se le asigna un **rol técnico ágil**:
   * Frontend developer
   * Backend developer
   * QA/tester
   * Product Owner (puede estar representado por vos si se prefiere)
   * UX/UI designer (opcional)
3. Asigná una o dos **tareas concretas** por persona desde el Trello (o usá esta plantilla que puedo crearte si querés):

| **Historia** | **Rol** | **Tarea específica** |
| --- | --- | --- |
| Ver clases | Frontend | Maquetar vista de clases |
| Ver clases | Backend | Implementar endpoint de clases |
| Reservar turno | QA | Validar reservas inválidas |
| Enviar recordatorio | Backend | Programar envío automático por hora |
| Historial | PO | Revisar criterios de aceptación |

**🗓️ Desarrollo de la Daily (25 minutos)**

**1. Organización de la dinámica**

* Actividad sincronizada por vos como “Scrum Master”.
* Cada equipo tiene su propio espacio (virtual o en círculo si es presencial).
* Se abre el **tablero Trello** donde están cargadas las historias y tareas.
* Se les pide responder en ronda, una por una, las 3 preguntas:

a) ¿Qué hiciste ayer?  
b) ¿Qué vas a hacer hoy?  
c) ¿Tenés algún impedimento?

📌 Pautas:

* Tiempo por persona: 1–2 minutos.
* No se discuten soluciones en la daily (solo se detectan problemas).
* Anotá impedimentos que aparezcan para tratarlos luego.

**2. Ejemplo de intervención**

🎙️ *Ana – Backend Dev*:  
Ayer terminé el endpoint para listar clases disponibles.  
Hoy empiezo con la lógica para confirmar turnos.  
Como impedimento, necesito definir cómo se notifican los usuarios.

**✍️ Actividad escrita de cierre (10 minutos)**

Cada estudiante completa una tabla reflexiva breve:

| **Nombre** | **Rol asignado** | **Mayor desafío** | **¿Te resultó natural comunicarte? ¿Por qué?** |
| --- | --- | --- | --- |
| Ana | Backend Dev | Falta de info del cliente | Sí, porque sabía lo que hice y lo que me faltaba. |

**🧠 BLOQUE 3 – Práctica Técnica: GIT en Equipos Ágiles (Duración: 60 minutos)**

**✅ Objetivo del bloque:**

Que los estudiantes comprendan, practiquen y experimenten cómo se trabaja en **entornos de desarrollo colaborativos con Git y GitHub**, tal como lo hacen los equipos ágiles reales: con ramas, commits, issues y colaboración asincrónica.

**🧪 Actividad práctica: Taller guiado de Git**

**🛠 Requisitos previos**

* Git instalado (<https://git-scm.com>)
* Cuenta en GitHub
* Editor de código instalado (VSCode recomendado)
* (Opcional) GitHub Desktop si se prefiere interfaz gráfica

**📌 PASO A PASO DEL TALLER**

**🔹 1. Crear un repositorio en GitHub (10 min)**

**Opciones:**

* El docente crea un repositorio base y da acceso como colaborador a los estudiantes.
* O bien, cada estudiante crea su propio fork del proyecto.

Repositorio sugerido: https://github.com/ematevez/turno-app-demo

**🔹 2. Clonar localmente y crear un archivo README.md (5 min)**

git clone https://github.com/tuusuario/turno-app-demo.git

cd turno-app-demo

echo "# Turno App" > README.md

git add README.md

git commit -m "docs: agrega archivo README inicial"

git push origin main

**🔹 3. Crear una rama (feature) y hacer commits (10 min)**

git checkout -b feature-bloquear-turno

# Simular tarea: descomentar o modificar una función

git add .

git commit -m "feat: agrega función para bloquear turno en instructor.js"

git push origin feature-bloquear-turno

**✍️ EJEMPLO DE CÓDIGO BASE – archivo instructor.js**

// Función para bloquear turnos (tarea pendiente)

// Descomentar para implementar

// function bloquearHorario(horario) {

// if (!horario) return false;

// console.log("Bloqueando horario:", horario);

// return true;

// }

💡 **Tarea propuesta para el Issue:**

“Descomentar la función bloquearHorario() en el archivo instructor.js y probarla con un console.log() de ejemplo.”

**🧩 USO DE ISSUES EN GITHUB (10 min)**

**¿Qué son los issues?**

* Herramientas para documentar tareas, bugs o ideas.
* Se usan para asignar responsabilidades en entornos ágiles (como Trello pero en GitHub).
* Cada historia o tarea técnica debe tener un issue vinculado.

**Pasos:**

1. Ir al repositorio > pestaña "Issues" > "New Issue"
2. Título: Agregar función de bloqueo de horarios
3. Descripción:

Descomentar e implementar la función bloquearHorario() en instructor.js. Asegurarse de que retorne true si el horario es válido.

1. Asignar el issue al estudiante
2. Usar “Fixes #n” en el commit para cerrar el issue automáticamente.

**🔹 Ejemplo:**

git commit -m "fix: descomenta función bloquearHorario() [Fixes #2]"

**🔄 4. Hacer merge (fusión) a la rama principal (10 min)**

1. Ir a GitHub > Comparar & pull request
2. Crear un Pull Request
3. Revisar, comentar y aprobar (puede hacerlo el docente o compañeros)
4. Hacer merge a main

**📺 Tutorial de apoyo (en español)**

* **Curso Git para principiantes – Fazt**  
  🔗 <https://www.youtube.com/watch?v=HiXLkL42tMU>

**📊 Cierre del bloque (10 min)**

**Reflexión escrita individual:**

* ¿Qué aprendiste sobre Git?
* ¿Qué parte te resultó más desafiante: clonar, hacer commits, usar ramas o trabajar con issues?
* ¿Creés que esta práctica se parece al trabajo real de un programador en un equipo ágil?

**📘 Evaluación del bloque (criterios)**

| **Criterio** | **Indicador** |
| --- | --- |
| Uso de Git y ramas | Crea y trabaja en su propia rama |
| Gestión de commits | Mensajes claros, uno por cambio |
| Participación en colaboración | Usa issues y pull requests |
| Comprensión del flujo ágil | Relaciona el uso de Git con la colaboración en equipo |

**✍️ BLOQUE 4 – Pair Programming Simulado (Duración: 50 minutos)**

**🎯 Objetivo del bloque:**

Experimentar la dinámica de **programación en pareja (pair programming)**, una práctica común en metodologías ágiles (especialmente en **Extreme Programming - XP**) que favorece la colaboración, la mejora de calidad del código y el aprendizaje entre pares.

**💡 ¿Qué es Pair Programming?**

Es una técnica donde **dos personas trabajan juntas en una misma computadora** (o entorno remoto compartido):

* 🧑‍💻 **Driver**: quien escribe el código.
* 🧠 **Navigator**: quien revisa, propone mejoras, detecta errores y piensa estratégicamente.

🧪 Se alternan roles cada 10 minutos o al completar una tarea.

**📦 Contexto de trabajo (igual que en clases anteriores)**

**Proyecto:** App de reservas para un gimnasio  
**Funcionalidad:** Validar si un turno reservado tiene una fecha válida

**👥 Organización de la actividad (10 minutos)**

1. Formá **parejas de estudiantes** (si hay número impar, un grupo puede ser de tres).
2. Cada pareja elige roles: Driver / Navigator.
3. Se les entrega el mismo fragmento de código incompleto.
4. El objetivo es:
   * Completar y probar la función
   * Mejorarla si es posible
   * Validarla con distintos inputs
5. Luego de 15 minutos, se cambian los roles.

**📄 Código base propuesto (lenguaje: JavaScript)**

**Archivo: validarTurno.js**

javascript

CopiarEditar

// Validar si el turno es en una fecha futura

function validarTurno(fecha) {

const hoy = new Date();

const fechaTurno = new Date(fecha);

// TODO: retornar true si la fecha es futura, false si es inválida

// Descomentar esta línea y completar

// return ???;

}

// Ejemplo de uso

console.log(validarTurno("2025-07-10")); // Esperado: true

console.log(validarTurno("2023-01-01")); // Esperado: false

🔧 **Tarea técnica:**

* Implementar la función validarTurno
* Validar múltiples casos
* Hacer commit con mensaje: "feat: valida correctamente fechas de turnos"

✅ Opcional: usar console.log para ver los valores intermedios

**🔁 Dinámica de trabajo (25 minutos)**

| **Tiempo** | **Actividad** |
| --- | --- |
| 10 min | 1. Driver codifica, Navigator guía |
| 2 min | 2. Refrescan roles |
| 10 min | 3. Nuevo Driver continúa trabajo |
| 3 min | 4. Guardan cambios y hacen commit |

🧠 **Sugerencia para Navigator:**  
Guiar con preguntas como:

* ¿Validaste qué pasa si el input es inválido?
* ¿Qué pasa si el turno es hoy?

**💬 Cierre y reflexión (15 minutos)**

**Espacio grupal guiado:**

* ¿Cómo se sintieron en cada rol?
* ¿Fue difícil guiar o ser guiado?
* ¿Qué aprendieron del otro?
* ¿Qué errores detectaron gracias a la colaboración?

📋 Opcional: completar un formulario con estas preguntas o debatir en grupo.

**🎥 Video sugerido para introducir el concepto (opcional):**

**¿Qué es Pair Programming? – CodeStack (subtitulado en español)**  
🔗 <https://www.youtube.com/watch?v=vgkahOzFH2Q>

**📊 BLOQUE 5 – Cierre Reflexivo + Evaluación (50 min)**

🧩 **Debate guiado (puede hacerse en Miro, Padlet o foro):**

1. ¿Qué tareas te resultaron más naturales del rol de programador?
2. ¿Qué ventajas aporta trabajar con GIT y Trello en equipo?
3. ¿Qué cosas aprendiste hoy que cambiarían tu forma de trabajar en proyectos?

📌 **Evaluación individual breve:**

Entregar por escrito (o en Word/Google Doc):

* 3 tareas clave del programador en ágil.
* 1 diferencia clave con un programador en método cascada.
* Screenshot de commits hechos o captura del Trello usado.
* 1 reflexión personal sobre la experiencia de la daily o de GIT.

**📚 Recursos adicionales**

* **Scrum Guide (en español)**: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish.pdf
* **Trello**: <https://trello.com>
* **GitHub**: <https://github.com>
* **Plataformas colaborativas**: Miro, Google Docs, Notion