Informe de Retrospectiva (Post-Mortem): Optimización de Transferencia de Archivos en VPN con Algoritmos Voraces

---

1. Propósito

Este documento resume las lecciones aprendidas durante el desarrollo del proyecto, identificando *qué funcionó bien*, *qué falló* y *cómo mejorar* en futuras iniciativas similares. El objetivo es convertir los hallazgos en acciones concretas para aumentar la eficiencia y calidad en próximos proyectos.

---

2. Resumen del Proyecto

El proyecto consistió en optimizar la transferencia de archivos en una VPN mediante algoritmos voraces (Dijkstra para rutas rápidas y Kruskal para árboles de expansión mínima). Se logró:

- Configurar una VPN funcional con 5 nodos.
- Diseñar un protocolo de transferencia con latencia reducida en un 40%.
- Implementar una GUI para transferencias seleccionables.

---

3. Lecciones Aprendidas

A. Qué salió bien (¡Repetir!)

- **Comunicación diaria**: Las *standups* mantuvieron al equipo alineado.
- **Herramientas ágiles**: GitHub Projects y Jira facilitaron el seguimiento de tareas.
- **Pruebas incrementales**: Validar cada algoritmo por separado (Dijkstra/Kruskal) evitó errores en cascada.

B. Qué falló (¡Mejorar!)

- **Dependencias externas**: Retrasos en la obtención de licencias para herramientas de medición de ancho de banda.
- **Documentación tardía**: El código se documentó *al final*, dificultando la onboarding de nuevos miembros.
- **Subestimación de tareas**: La GUI tomó un 50% más de tiempo del estimado por falta de experiencia previa en Qt.

C. Impedimentos críticos

- **Hardware limitado**: Los nodos de prueba tenían poca RAM, afectando las mediciones de latencia.
- **Cambios de alcance**: Se añadió last-minute la comparación de topologías (original vs Kruskal), forzando ajustes en el cronograma.

---

4. Acciones de Mejora

A. Planificación

1. **Realizar un *Proof of Concept* (PoC) temprano**:
    - Validar requisitos técnicos (ej: rendimiento de algoritmos en hardware real) antes del desarrollo.
2. **Buffer de tiempo para dependencias**:
    - Asignar un 20% adicional de tiempo a tareas con riesgos externos (ej: licencias).

B. Ejecución

3. **Documentar en paralelo**:
    - Usar herramientas como *Sphinx* o *Markdown* para generar docs automáticamente desde el código.
4. **Capacitación técnica previa**:
    - Organizar talleres de GUI (Qt/Tkinter) si el equipo no tiene experiencia.

C. Comunicación

5. **Revisión de alcance formal**:
    - Implementar un *change request* firmado por el PO para evitar cambios improvisados.
6. **Retrospectivas intermedias**:
    - Hacer mini-retros cada 2 sprints para ajustar procesos.

D. Herramientas

7. **Monitorización en tiempo real**:
    - Integrar *Prometheus* + *Grafana* para visualizar latencia/ancho de banda durante las pruebas.
8. **Entorno de pruebas escalable**:

    - Usar contenedores (*Docker*) para simular nodos con diferentes configuraciones de hardware.

---

5. Recomendaciones para Futuros Proyectos

- **Priorizar MVP**: Enfocarse primero en la funcionalidad básica (ej: Dijkstra) antes de features secundarias (GUI).
- **Involucrar stakeholders desde el inicio**: Presentar avances semanales para evitar malentendidos en requisitos.
- **Métricas de calidad**: Definir KRs no solo de funcionalidad (ej: "Reducir latencia"), sino también de mantenibilidad (ej: "Cobertura de tests > 80%").

---

6. Conclusión

Este proyecto demostró que la combinación de metodologías ágiles (Scrum) con algoritmos de optimización puede entregar resultados tangibles. Sin embargo, la falta de preparación para riesgos externos y la documentación tardía fueron cuellos de botella críticos. Implementando las acciones propuestas —especialmente *PoCs tempranos* y *docs en paralelo*—, futuros equipos podrán reducir incertidumbres y aumentar la eficiencia en un **30-40%**.