Talleres de Autogestión + Pensamiento Computacional

Estudiante: Maximiliano Madrid Benavides

Una tabla para organizar mis reflexiones y aprendizajes a lo largo de los talleres.

Tabla de seguimiento de talleres

Taller	Descripción breve	Actividades realizadas	Evaluación / Reflexión	Comentarios / Recursos
Taller 1	Control de versiones en mi vida	Análisis de módulos personales + historial de cambios	Retrospectiva de hábitos y mejoras implementadas	📎 Dashboard personal, logs de actividades
Taller 2	Toolkit de habilidades personales	Documentación de métodos de aprendizaje y organización	¿Son transferibles? ¿escalables? ¿útiles para colaboración?	Sistema en Obsidian con metodologías
Taller 3	Interfaz de comunicación personal	Definí endpoints `/horarios`, `/modo_trabajo`, `/intereses`, `/notificaciones`. Establecí niveles de acceso y protocolos.	Mi interfaz es consistente pero necesita mejor gestión de interrupciones. Refleja fielmente mi estilo de trabajo y comunicación.	Documentación de protocolos en Obsidian, calendario sincronizado, sistema de estados en Discord.
Taller 4	Evolución de metodología de trabajo	Análisis v1.0 vs v2.0 vs v3.0 de mi sistema de gestión de proyectos	¿Incrementé productividad? ¿Reduje burnout?	Métricas comparativas con datos reales
Taller 5	Cuándo adaptar vs. crear desde cero	Analicé sistemas copiados de desarrolladores senior, metodologías ágiles, y herramientas como Obsidian. Evalué efectividad y necesidad de personalización.	Descubrí que la comprensión profunda es más valiosa que la implementación exacta. Crear híbridos personalizados funciona mejor que copiar completamente.	Framework de evaluación en Obsidian, comparativas antes/después, principios personales derivados.

Objetivo del Taller #1

> Aplicar principios de control de versiones para reflexionar sobre la gestión de mi tiempo y hábitos, organizando actividades en módulos independientes, documentando cambios significativos como commits, y realizando una evaluación periódica del progreso personal.

🧪 Actividades

Módulos de mi ecosistema personal:

- `desarrollo/` – programación, proyectos personales y aprendizaje técnico
- `academico/` – clases, tareas y preparación para exámenes
- `bienestar/` – ejercicio, alimentación y rutinas de autocuidado
- `social/networking/` – relaciones interpersonales y actividades grupales
- `creatividad/` – escritura, música y proyectos artísticos

Commits realizados (mejoras significativas):

- commit 1: Implementado sistema de bloques de tiempo de 90 minutos para deep work
- commit 2: Migración de notas dispersas a sistema centralizado en Obsidian
- commit 3: Establecida rutina matutina con 30 min de lectura técnica
- commit 4: Optimizado workflow de desarrollo con shortcuts y automatizaciones

`git log --oneline` (resumen semanal):

- a1b2c3d Completé 15 horas de programación enfocada sin distracciones
- b2c3d4e Mantuve consistencia en ejercicio 4/7 días de la semana
- c3d4e5f Terminé todas las entregas académicas con 2 días de anticipación
- d4e5f6g Dediqué 3 horas semanales a networking y colaboración

✅ Evaluación

¿Qué descubrí sobre mi funcionamiento?

- Que mi productividad aumenta significativamente con bloques largos de tiempo ininterrumpido
- El sistema centralizado de notas me permitió conectar ideas entre diferentes proyectos

¿Qué optimizaciones fueron más efectivas?

- El módulo `desarrollo/` con bloques de 90 minutos triplicó mi output de código limpio
- La rutina matutina de lectura mejoró mi capacidad de absorber conceptos complejos

¿Qué necesita refactoring?

- El balance entre `academico/` y `desarrollo/` aún genera conflictos de prioridad
- El módulo `bienestar/` necesita mayor automatización para mantener consistencia

---

Objetivo del Taller #2

- Estructurar mi conocimiento de manera que sea reutilizable y transferible
- Facilitar que otros puedan aprender de mis métodos probados
- Descomponer procesos complejos en funciones claras y modulares
- Evaluar la escalabilidad, claridad y facilidad de integración de mis metodologías

Mi Librería Personal: Sistema de Aprendizaje Técnico Acelerado

Descripción

Metodología sistemática para dominar nuevas tecnologías, frameworks o conceptos de programación de manera eficiente. Cada función representa una fase optimizada del proceso de aprendizaje.

Funciones de la Librería

Función	Propósito	Implementación	Ejemplo de uso
`MapearTerreno()`	Obtener visión general del ecosistema tecnológico	1. Investigar 3 fuentes confiables 2. Crear mapa mental de conceptos 3. Identificar prerrequisitos	Aprender React: JSX → Components → Hooks → State Management
`CrearLabPractico()`	Establecer entorno de experimentación controlado	1. Configurar proyecto base 2. Preparar casos de prueba 3. Documentar setup inicial	Repo de GitHub con ejemplos incrementales
`CicloFeynman()`	Validar comprensión mediante explicación simplificada	1. Explicar concepto sin jerga técnica 2. Identificar lagunas 3. Refinar comprensión	Explicar async/await como "esperar respuesta de servidor"
`IntegrarProyecto()`	Aplicar conocimiento en contexto real	1. Definir proyecto de aplicación 2. Implementar iterativamente 3. Documentar decisiones	Crear API REST usando el framework aprendido

Ejemplo Completo

Situación: *Necesito aprender Docker para un proyecto universitario*

1. **MapearTerreno:** Containerización → Images → Containers → Docker Compose → Orchestration
2. **CrearLabPractico:** Repo con Dockerfile básico, docker-compose.yml, y app simple
3. **CicloFeynman:** "Docker es como una caja que empaqueta tu app con todo lo necesario para funcionar en cualquier computadora"
4. **IntegrarProyecto:** Containerizar proyecto existente del curso, documentar proceso

Evaluación de mi Librería

Criterio	Preguntas clave	Puntuación (1-5)	Observaciones
Claridad	¿Puede alguien sin contexto seguir el proceso? ¿Los pasos son específicos?	4	El proceso es claro, pero podría beneficiarse de más ejemplos de debugging común
Reusabilidad	¿Funciona para diferentes tipos de tecnología? ¿Se adapta a distintos estilos de aprendizaje?	5	He usado esta metodología exitosamente para frameworks, lenguajes y herramientas DevOps
Integración	¿Se combina bien con otros métodos de estudio? ¿Escala para proyectos complejos?	4	Se integra bien con metodologías ágiles, pero necesita adaptación para conceptos muy abstractos

---

Objetivo del Taller #3

> Modelar mis patrones de comunicación, disponibilidad y colaboración como una API REST. Esto permite reflexionar sobre la consistencia de mis interfaces sociales y optimizar la forma en que otros interactúan conmigo académica y profesionalmente.

🧪 Actividades

- Definición de endpoints de mi interfaz personal:

```json
```

GET /horarios
→ {
  "lunes": {
    "deep_work": "08:00-11:30",
    "disponible_colaboracion": "14:00-17:00",
    "modo_aprendizaje": "19:00-21:00"
  },
  "martes": {
    "clases": "09:00-13:00",
    "proyecto_personal": "15:00-18:00"
  },
  "weekend": "flexible, confirmar 24h antes"
}

GET /modo_trabajo
→ {
  "estado_actual": "focused",
  "disponibilidad_interrupciones": "baja",
  "canales_preferidos": ["email", "slack"],
  "tiempo_respuesta_estimado": "2-4 horas",
  "contexto": "desarrollando feature crítica"
}

GET /intereses
→ {
  "tecnologias_actuales": ["React", "Node.js", "PostgreSQL"],
  "aprendiendo": ["Docker", "AWS"],
  "areas_colaboracion": ["backend", "arquitectura", "mentoria"],
  "proyectos_abiertos": ["biblioteca de utilidades JS", "blog técnico"]
}

POST /notificaciones
→ {
  "tipo": "deadline_proyecto",
  "prioridad": "alta",
  "canal": "slack_dm",
  "contexto_requerido": "detalles específicos del blocker"
}

- **Niveles de acceso y autenticación:**
  - `/horarios`: Visible para team members y compañeros de estudio
  - `/intereses`: Público para networking

- `/notificaciones`: Requiere colaboración activa
- `/modo_trabajo`: Visible para contexto de trabajo inmediato

- **Documentación estilo OpenAPI:**
  - Especificación completa en Obsidian con ejemplos de request/response
  - Rate limiting para diferentes tipos de interacción

Evaluación

- **¿Es mi interfaz predecible?** ✅ Mayormente sí. Mantengo horarios consistentes para deep work y respondo emails de manera regular. Los endpoints de `/horarios` son confiables.
- **¿Proporciona valor a los usuarios?** ✅ Definitivamente. Compañeros de equipo pueden planificar colaboraciones efectivamente, y mi `/modo_trabajo` reduce interrupciones innecesarias.
- **¿La experiencia de usuario es óptima?** ⚠️ Parcialmente. Mi `/modo_trabajo` a veces cambia abruptamente sin actualizar el estado, causando confusión en colaboradores.
- **¿Qué optimizaría?**
  - Implementar webhook automático cuando cambio de contexto
  - Crear endpoint `/energia_creativa` para proyectos que requieren brainstorming
  - Mejorar la documentación de mis procesos de code review

Objetivo del Taller #4

- **Documentar** la evolución histórica de mi metodología de gestión de proyectos personales
- **Cuantificar** las mejoras usando métricas objetivas como:
  - Tiempo de entrega vs. estimaciones
  - Calidad del código/resultado final
  - Nivel de estrés y sostenibilidad
  - Capacidad de manejar múltiples proyectos simultáneamente
- **Analizar** qué cambios generaron mayor impacto y planificar próximas iteraciones

Evolución de mi Sistema de Gestión de Proyectos

Rutina analizada: Desarrollo y entrega de proyectos de programación

Versión 1.0 – Enfoque caótico (`ProjectManager.v1()`)

- Comenzaba proyectos sin planificación detallada

- Codificaba features en orden aleatorio según inspiración
- Sin control de versiones adecuado (commits masivos)
- Testing manual solo al final
- Documentación inexistente
- Entregas de último minuto con alta tensión

Versión 2.0 – Metodología estructurada (`ProjectManager.v2()`)

- Análisis inicial con user stories y casos de uso
- Arquitectura definida antes del primer commit
- Desarrollo iterativo con branches por feature
- Testing unitario integrado en el workflow
- Documentación técnica durante desarrollo
- Buffer de tiempo para pulir y optimizar

Versión 3.0 – Sistema optimizado (`ProjectManager.v3()`) [Actual]

- Planning poker personal para estimaciones más precisas
- TDD (Test-Driven Development) como práctica estándar
- CI/CD pipeline automatizado con GitHub Actions
- Code review sistemático (incluso en proyectos personales)
- Métricas de performance y quality gates
- Retrospectivas post-proyecto para mejora continua

Comparación de Métricas

Métrica	v1.0 (Caótico)	v2.0 (Estructurado)	v3.0 (Optimizado)
Tiempo real vs. estimado	200% (siempre me pasaba)	120% (mejoré estimaciones)	105% (muy preciso)
Bugs en producción	8-12 por proyecto	3-5 por proyecto	0-2 por proyecto
Estrés nivel (1-10)	9 (crisis constantes)	5 (controlado)	3 (flujo natural)
Proyectos simultáneos	1 (máximo sin colapsar)	2-3 (con organización)	4-5 (sistema escalable)
Tiempo de onboarding	N/A (solo yo)	2-3 días para colaborador	30 min (docs + setup automatizado)
Satisfacción personal	6/10 (resultado vs. estrés)	8/10 (orgullo del proceso)	9/10 (craftsmanship)

Representación como API Evolution

```javascript
// v1.0 - Monolítico y frágil
function ProjectManager.v1(requirements) {
  return "Codificar hasta que funcione, rezar que no se rompa";
}

// v2.0 - Modular y predecible
function ProjectManager.v2(requirements) {
  return "Planificar → Desarrollar → Probar → Entregar";
}

// v3.0 - Automatizado y escalable
function ProjectManager.v3(requirements) {
  return "Estimate → TDD → CI/CD → Monitor → Iterate";
}
```

Conclusión y Roadmap v4.0

Mi sistema de gestión de proyectos evolucionó desde el caos hacia la excelencia técnica. La versión 3.0 no solo mejoró la calidad y predictibilidad, sino que redujo dramáticamente el estrés y me permitió escalar mi capacidad.

Próximas features para v4.0:

- Integración de AI para code review automático
- Métricas predictivas basadas en análisis histórico
- Sistema de alertas tempranas para riesgos de proyecto
- Templates automatizados para diferentes tipos de arquitectura

---

Objetivo del Taller #5

> Evaluar críticamente mis experiencias adaptando metodologías, herramientas y frameworks de desarrollo de otros programadores y equipos. Analizar cuándo la personalización agregó valor real versus cuándo la adopción directa fue más eficiente, para desarrollar criterio sobre cuándo innovar y cuándo seguir estándares establecidos.

Actividades

- **Identifiqué herramientas y metodologías que he adoptado** de otros desarrolladores, equipos open source, y recursos educativos: Ejemplos analizados:
    1. **Dotfiles y configuración de desarrollo de un senior developer**
        - Setup completo de VSCode, terminal, y aliases
        - Muy específico para su workflow, requirió adaptación significativa
    2. **Metodología Getting Things Done (GTD) para developers**
        - Sistema de captura y organización de tareas técnicas
        - Funcionó excelente con mínimas modificaciones
    3. **Architecture Decision Records (ADRs) de equipos enterprise**
        - Template muy formal y pesado para proyectos personales
        - Tuve que crear versión simplificada
    4. **Testing philosophy de Kent C. Dodds**
        - Pirámide de testing y best practices
        - Adopté directamente con gran éxito

Evaluación Detallada

Tabla de análisis comparativo:

Herramienta/Metodología	¿Reutilización directa viable?	¿Personalización necesaria?	Resultado final	Lecciones aprendidas
Dotfiles de senior dev	❌	✅ Adapté 60% del setup	Muy positivo	Los shortcuts son personales, pero la estructura organizacional es transferible
GTD para developers	✅	✅ Ajustes menores	Excelente	Los principios universales se adaptan bien a contextos específicos
ADRs enterprise	❌	✅ Simplificación radical	Positivo	El overhead debe ser proporcional al tamaño del proyecto
Testing philosophy	✅	❌	Transformador	Las buenas prácticas fundamentales son universalmente aplicables
Agile ceremonies para solos	❌	✅ Sprint planning personal	Moderado	Ceremonias grupales necesitan re-contextualización para trabajo individual
Clean Code principles	✅	✅ Adaptación gradual	Excelente	Los principios son sólidos, la aplicación debe ser progresiva

Framework de Decisión Desarrollado:

javascript

```
function shouldReuseOrAdapt(tool, context) {
  const factors = {
    fundamentalPrinciples: tool.hasUniversalPrinciples(),
    contextMatch: tool.targetContext === context.current,
    complexityBudget: context.timeAvailable >= tool.learningCurve,
    teamAlignment: context.isTeamWork && tool.requiresConsensus
  };

  if (factors.fundamentalPrinciples && factors.contextMatch) {
    return "REUSE_DIRECTLY";
  } else if (factors.fundamentalPrinciples && !factors.contextMatch) {
    return "ADAPT_PRINCIPLES";
  } else if (!factors.complexityBudget) {
    return "SIMPLIFY_OR_SKIP";
  } else {
    return "BUILD_FROM_SCRATCH";
  }
}
```

Conclusiones Estratégicas:

1. **Los principios fundamentales son transferibles**, las implementaciones específicas requieren adaptación
2. **El overhead de adopción debe ser proporcional al beneficio esperado** - no todas las herramientas "cool" valen el investment
3. **La personalización gradual es más efectiva** que la adopción masiva o el rechazo total
4. **El contexto (team vs. individual, pequeño vs. enterprise) determina qué aspectos adaptar**

Manifiesto Personal de Reutilización:

- **Entender antes de adoptar**: No copio configuraciones o metodologías sin comprender el razonamiento subyacente
- **Medir antes de comprometerse**: Pruebo en proyectos pequeños antes de adoptar en workflow principal
- **Principios sobre implementaciones**: Busco entender el "por qué" antes que el "cómo"
- **Iteración over perfección**: Prefiero adopción gradual con mejoras incrementales
- **Documentar decisiones**: Mantengo registro de qué adopté, qué adapté, y por qué, para futuras decisiones

La clave está en desarrollar criterio técnico para distinguir entre sabiduría reutilizable y preferencias personales, optimizando el balance entre standarización y personalización según el contexto específico.