# ESPECIFICACIÓN TÉCNICA: METODOLOGÍA FASÉTICA

## Modelo de Ejecución por Fases Horizontales (MEFH)

**Versión:** 3.0

**Clasificación:** Estándar de Ingeniería de Software

**Autoría:** Andriy Lionel Pastrana Cajavilca

**Organización:** Studios TKOH!

**Estado:** Estándar Abierto / Propiedad Intelectual Originaria

## 1. RESUMEN EJECUTIVO

La **Metodología Fasética** (o Modelo de Ejecución por Fases Horizontales) es un marco de trabajo (*framework*) de ingeniería de software agnóstico, diseñado para resolver la crisis de inestabilidad y retrabajo en el desarrollo de aplicaciones modernas.

A diferencia de las metodologías ágiles tradicionales que priorizan la velocidad de entrega aparente, el modelo Fasético prioriza la **Certeza Técnica**. Se fundamenta en la estructuración del desarrollo mediante "filas de cajas" horizontales, donde la segregación de responsabilidades y la inmutabilidad de las fases garantizan una estabilidad residual absoluta.

## 2. DEFINICIÓN Y FILOSOFÍA

### 2.1 Definición Fundamental

La Metodología Fasética se define como un sistema de desarrollo basado en la **separación estricta de responsabilidades** y la **validación formal de hitos técnicos**. Su arquitectura lógica organiza la construcción del software en secuencias horizontales inquebrantables, eliminando la ambigüedad en la integración de componentes.

### 2.2 El Mantra Operativo

*"No avanzamos para terminar rápido; avanzamos para no volver atrás."*

Este principio establece que la velocidad real del proyecto es consecuencia de no tener que corregir el pasado (retrabajo), y no de la prisa en la escritura de código.

### 2.3 Los 4 Pilares Innegociables

Para que una implementación sea considerada "Fasética", debe adherirse a los siguientes axiomas:

1. **Certeza sobre Tarea:** El progreso no se mide por tareas marcadas como "hechas", sino por la validación técnica irrefutable de que la fase actual es sólida.
2. **Inmutabilidad de Fase:** Principio de congelamiento. Una vez cerrada una fase, sus reglas y funcionalidades son ley. No se admiten cambios retroactivos por capricho sin un proceso formal de reapertura.
3. **Sincronía de Espejo:** Los contratos de datos (Backend) y sus interfaces de consumo (Frontend) deben ser reflejos exactos. La asimetría es considerada un error crítico.
4. **Agnosticismo Arquitectónico:** El modelo es universal. Funciona sobre Arquitectura Hexagonal, Clean Arch, MVC o cualquier framework, bajo la **Ley de No Mezclar Responsabilidades**.

## 3. MODELO ESTRUCTURAL HORIZONTAL

La metodología visualiza el flujo de trabajo como una secuencia lineal de responsabilidades (cajas). Dependiendo de la naturaleza del proyecto, se selecciona uno de los dos **Vectores de Ejecución**:

### A. Vector Frontend-First (Diseño como Plano Maestro)

*Uso recomendado: Software a medida, Startups, Productos centrados en UX.*

1. **Caja 0 (UI/UX):** Diseño visual completo y validación funcional con el cliente (El contrato visual).
2. **Ciclo Backend:** Base de Datos $\rightarrow$ Contratos $\rightarrow$ Lógica/Servicios $\rightarrow$ Comunicación.
3. **Ciclo Frontend:** Contratos (Espejo) $\rightarrow$ Comunicación $\rightarrow$ Lógica de Negocio (Hooks) $\rightarrow$ Conexión (Fusión con la Caja 0).

### B. Vector Backend-First (Lógica como Estructura Maestra)

*Uso recomendado: Sistemas críticos, Fintech, Core Bancario, Alta Seguridad.*

1. **Ciclo Backend:** Base de Datos $\rightarrow$ Contratos $\rightarrow$ Lógica/Servicios $\rightarrow$ Comunicación.
2. Ciclo Frontend: Contratos (Espejo) $\rightarrow$ Comunicación $\rightarrow$ Lógica de UI $\rightarrow$ Construcción de UI.  
   (Nota: Aquí la interfaz es una consecuencia de la capacidad del sistema, no un requisito estético previo).

### 3.1 Representación Gráfica del Modelo Fasético

Para evitar ambigüedades, se establece la representación gráfica obligatoria de los flujos. Nótese cómo la **Fase 4 del Frontend** cambia drásticamente de función dependiendo de la existencia previa de la Caja 0.

#### DIAGRAMA A: FLUJO FRONTEND-FIRST

*El contrato visual (Caja 0) dicta la estructura, y la UI final es una integración.*

[ CAJA 0: UI / MOCKUPS ] <--(Solo existe en este camino)--+  
 │   
 ▼   
[ CICLO BACKEND ]   
Fase 1: Base de Datos → Fase 2: Contratos → Fase 3: Lógica → Fase 4: Comunicación  
 │  
 ┌────────────────────────────────────────────────────────┘  
 ▼  
[ CICLO FRONTEND ]  
Fase 1: Contratos (Espejo) → Fase 2: Comunicación → Fase 3: Lógica → [ FASE 4: INTEGRACIÓN ]  
 (Conecta Lógica a Caja 0)

#### DIAGRAMA B: FLUJO BACKEND-FIRST

*La estructura de datos dicta la función, y la UI final se construye desde cero.*

[ CICLO BACKEND ]  
Fase 1: Base de Datos → Fase 2: Contratos → Fase 3: Lógica → Fase 4: Comunicación  
 │  
 ┌─────────────────────────────────────────────────────────┘  
 ▼  
[ CICLO FRONTEND ]  
Fase 1: Contratos (Espejo) → Fase 2: Comunicación → Fase 3: Lógica → [ FASE 4: CONSTRUCCIÓN ]  
 (Dibuja UI basada en Lógica)

## 4. DESGLOSE TÉCNICO: CICLOS DE VIDA DIFERENCIADOS

El desarrollo no es lineal ni idéntico en ambos extremos. El orden de construcción varía drásticamente entre el Servidor (Backend) y el Cliente (Frontend) según el vector elegido.

### FASE 0: PRE-VISUALIZACIÓN (Exclusiva de Frontend First)

*Esta fase es el "Punto de Partida" en el camino Frontend First. En el camino Backend First, esta fase no existe al inicio y se absorbe al final.*

* **Objetivo:** Diseño visual estático, maquetación de pantallas y definición de experiencia de usuario (UX) sin lógica real.
* **Regla de Oro:** Todo es "fake". No hay conexiones reales. Se prioriza la aprobación visual del cliente antes de escribir lógica compleja.
* **Entregable:** Prototipos navegables o código de interfaz estático (Mockups).

### EL CICLO DEL BACKEND (El Cimiento)

*El Backend construye desde el dato hacia afuera.*

* **FASE 1: PERSISTENCIA (El Almacén):** Diseño de la arquitectura de datos, tablas, colecciones y relaciones. La integridad del dato es prioridad.
* **FASE 2: DEFINICIONES (El Contrato Interno):** Mapeo de la base de datos a código (Entidades/Modelos). Traducción estricta de tipos.
* **FASE 3: LÓGICA DE NEGOCIO (El Núcleo):** Algoritmos, validaciones y orquestación. Totalmente agnóstica a cómo se consumirá.
* **FASE 4: COMUNICACIÓN (La Ventanilla):** Exposición de la lógica (Endpoints, Sockets). Solo transporte, cero lógica de negocio.

### EL CICLO DEL FRONTEND (El Consumidor)

*El Frontend construye desde el contrato hacia la vista.*

* **FASE 1: CONTRATOS ESPEJO (La Ley):** Réplica exacta en el cliente de las estructuras definidas en el Backend (o Fase 0).
* **FASE 2: CONEXIÓN (El Puente):** Configuración de transporte y llamadas a endpoints. Solo mueve datos, no los procesa visualmente.
* **FASE 3: LÓGICA Y ESTADO (El Cerebro Local):** Procesamiento de datos en el cliente y gestión de estado. Testeable sin UI.
* **FASE 4: INTEGRACIÓN VISUAL / UI (El Reflejo):**
  + *Frontend First:* Conexión de la lógica (Fase 3) a la maqueta existente (Fase 0).
  + *Backend First:* Construcción de componentes visuales desde cero basados en las funciones disponibles.

## 5. GOBERNANZA, RIGOR Y NOMENCLATURA

### 5.1 Gatekeeping (Cierre de Fase)

El paso de una fase a otra no es automático. Requiere un acto formal de validación:

1. **Validación de Frontera:** Confirmación de que no existe fuga de lógica (ej. lógica de negocio en la capa de transporte).
2. **Pruebas de Integridad:** Verificación funcional mínima.

### 5.2 Semántica Narrativa (NDT)

Se exige el uso de **Nomenclatura Descriptiva Total**. El código debe narrar su ubicación en el modelo fasético.

* *Incorrecto:* updateUser()
* *Correcto (Fase 1/2):* UserSchema, UserDefinition
* *Correcto (Fase 3):* calculateUserTaxService

### 5.3 Documentación en Código

Los comentarios no explican *qué* hace el código, sino *por qué* existe esa regla en esa fase específica, reforzando la separación de preocupaciones.

## 6. JUSTIFICACIÓN ECONÓMICA Y ROI

La rentabilidad del Modelo Fasético no es uniforme; varía según el Vector de Ejecución elegido, permitiendo a la organización decidir **dónde** y **cuándo** asumir el riesgo de inversión.

### 6.1 Análisis Comparativo por Vector

| **Variable Económica** | **Vector Frontend-First (Enfoque Visual)** | **Vector Backend-First (Enfoque Estructural)** |
| --- | --- | --- |
| **Punto de Inversión Inicial** | **Fase 0 (Diseño/Mockups):** El gasto inicial es bajo en desarrollo técnico pero alto en definición de producto. | **Fases 1 y 2 (Datos/Estructura):** El gasto inicial es alto en ingeniería de software y arquitectura de datos. |
| **Riesgo Mitigado** | **"Rechazo del Cliente":** Se evita gastar en lógica compleja para funcionalidades que el cliente descartará al ver la UI. | **"Deuda Técnica":** Se evita construir una UI bonita sobre cimientos de datos podridos o no escalables. |
| **Costo de Integración** | **Casi Nulo:** Al llegar a la Fase 4, la UI ya existe y el Backend ya funciona. Solo es "conectar cables". | **Medio/Bajo:** La UI se construye "a medida" de los datos existentes, reduciendo la fricción. |
| **Velocidad de Entrega (Time-to-Market)** | **Percibida Rápida:** El cliente ve pantallas funcionales (aunque falsas) desde el inicio. | **Percibida Lenta (Efecto Túnel):** No hay nada visual hasta el final del ciclo, pero el sistema es robusto. |
| **Costo de Cambios (Change Request)** | **Bajo (en Fase 0):** Cambiar un diseño estático es barato. | **Alto:** Requiere refactorización desde la base de datos hacia arriba, pero garantiza integridad. |

### 6.2 El Dividendo de Estabilidad (Ahorro Común)

Independientemente del vector, el modelo elimina los dos "agujeros negros" presupuestarios del desarrollo tradicional:

1. **La Depuración Infinita:** Al estar las fases aisladas (Fase 3 Lógica vs Fase 4 UI), el costo de encontrar un *bug* se reduce drásticamente. Si el dato es correcto pero se ve mal, es Fase 4. Si el dato es incorrecto, es Fase 3. No hay ambigüedad.
2. **El Retrabajo por Descoordinación:** La regla de **Contratos Espejo** impide que Frontend y Backend desarrollen funcionalidades incompatibles, eliminando las semanas perdidas en integración.

## 7. VERSATILIDAD OPERATIVA DEL MODELO FASÉTICO

La Metodología Fasética no está limitada al desarrollo inicial de software. Su estructura horizontal y la separación estricta de responsabilidades la convierten en un modelo aplicable durante todo el ciclo de vida del sistema, incluyendo escenarios tradicionalmente problemáticos para otras metodologías.

### 7.1 Creación de Software (Greenfield)

En proyectos desde cero, el modelo permite seleccionar estratégicamente el Vector de Ejecución según el riesgo principal del negocio.

* **Decisiones Oportunas:** Reduce las decisiones prematuras que suelen "bloquear" arquitecturas futuras.
* **Base Validada:** Evita construir lógica sin contratos previos, garantizando que cada capa nace sobre cimientos estables.
* **Resultado:** Solidez estructural desde la primera línea de código.

### 7.2 Evolución y Mejora de Sistemas Existentes

El modelo permite introducir nuevas funcionalidades mediante **micro-ciclos faséticos**, sin comprometer la estabilidad del sistema principal.

* **Aislamiento de Features:** Cada nueva funcionalidad se trata como un ciclo independiente.
* **Reapertura Controlada:** Las fases se reabren solo para la feature específica, sin "contaminar" el núcleo existente.
* **Resultado:** Evolución progresiva y escalable sin acumulación de deuda técnica.

### 7.3 Mantenimiento Correctivo y Preventivo

La separación estricta por fases permite localizar fallos con precisión quirúrgica, eliminando la ambigüedad diagnóstica:

* **Errores de cálculo** $\rightarrow$ Se audita exclusivamente la **Fase 3 (Lógica)**.
* **Errores de visualización** $\rightarrow$ Se audita exclusivamente la **Fase 4 (UI)**.
* **Errores de integridad** $\rightarrow$ Se auditan las **Fases 1 o 2 (Persistencia/Contratos)**.
* **Resultado:** Reducción drástica del tiempo de depuración (MTTR) y de los costos de soporte.

### 7.4 Rescate de Proyectos Muertos o Moribundos

Uno de los usos más poderosos del modelo es la recuperación de sistemas "Legacy" o fallidos que sufren de código espagueti, lógica mezclada y ausencia de contratos. El rescate se realiza mediante la imposición progresiva del modelo, no mediante una reescritura total (*Big Rewrite*):

1. **Identificación de Contratos Implícitos:** Se extraen las reglas ocultas y se formalizan.
2. **Aislamiento de Lógica:** Se extrae la lógica de negocio atrapada en la UI o en la base de datos.
3. **Separación de Responsabilidades:** Se desacopla la comunicación de la presentación.
4. **Cierre Incremental:** Se bloquean las fases recuperadas para evitar regresiones.

* **Resultado:** Rehabilitación funcional del sistema, recuperando la inversión inicial.

## 8. PERFIL DEL INGENIERO FASÉTICO

La Metodología Fasética no es solo un flujo de trabajo, es una **disciplina mental**. No cualquier desarrollador encaja en este modelo; requiere un cambio de paradigma desde el "hacer que funcione" hacia el "hacer que perdure".

### 8.1 La Ley Suprema: No Mezclar Responsabilidades

Esta es la única regla de oro innegociable que define la validez del perfil, por encima del lenguaje o el framework utilizado.

* **El Mandato:** Bajo ninguna circunstancia se permite que una fase conozca, importe o dependa de los detalles de implementación de una fase ajena a su contrato inmediato.
* **Independencia de Arquitectura:** No importa la arquitectura elegida (**Hexagonal, DDD, MVC, Clean Architecture o una estructura personalizada**). Mientras se cumpla estrictamente la separación de responsabilidades, el perfil es válido y el proyecto es Fasético.
* **La Prueba Ácida:** Si al cambiar la base de datos (Fase 1) tienes que editar el controlador de la API (Fase 4), has violado la Regla de oro y has fallado en la implementacion de la metodologia.

### 8.2 Competencias Esenciales

Para operar bajo esta ley, el ingeniero debe cultivar tres virtudes explícitas y excluyentes:

1. **Paciencia Estratégica (Control de la Ansiedad Visual):**
   * *Definición:* La capacidad férrea de resistir la urgencia de ver "algo funcionando en pantalla" en los primeros días.
   * *Comportamiento:* El ingeniero entiende que una UI construida sobre una lógica inexistente es una alucinación costosa. Prefiere invertir tiempo validando esquemas de datos "invisibles" (Fases 1 y 2) sabiendo que esto acelerará exponencialmente el cierre del proyecto (Esto es en el caso de backend first).
2. **Imaginación Técnica (Abstracción Estructural):**
   * *Definición:* La habilidad para visualizar y manipular flujos de datos abstractos y contratos de interfaz en la mente, sin necesidad de renderizarlos gráficamente.
   * *Comportamiento:* Puede testear y validar una regla de negocio compleja (Fase 3) mirando solo un JSON de salida o un log de consola, confiando plenamente en que si el dato es correcto, la UI (Fase 4) lo representará fielmente después.
3. **Disciplina de Aislamiento (Rigor Quirúrgico):**
   * *Definición:* El rechazo absoluto a los "atajos" o "código espagueti" que violan la integridad de las fases, incluso bajo presión extrema de tiempo.
   * *Comportamiento:* Si detecta un error de datos en la UI, **jamás** lo "parcha" con un condicional en el Frontend. Rastrea el error hasta su origen (Fase 1 o 2 del Backend) y lo corrige quirúrgicamente allí. Mantiene la higiene de la separación por encima de la velocidad aparente.

## 9. ANÁLISIS COMPARATIVO CON MARCOS ESTÁNDARES

Este análisis sitúa a la Metodología Fasética (MEFH) en el espectro de la ingeniería de software, contrastándola con los marcos de trabajo predominantes bajo criterios estrictamente técnicos y operativos. El objetivo es identificar el dominio de aplicación idóneo para cada modelo, evitando juicios de valor subjetivos y centrando la atención en el "Trade-off" (costo-beneficio) estructural.

### 9.1 Contexto Conceptual y Origen de los Marcos Comparados

Para comprender las diferencias entre los marcos de trabajo analizados, resulta necesario examinar el contexto y propósito original de su creación. Las metodologías no son neutras: responden a los problemas que buscaban resolver en su momento de origen.

#### 9.1.1 Origen de Gestión vs. Ingeniería

Scrum, Kanban y Agile no surgieron originalmente como metodologías de ingeniería de software estructurada, sino como modelos de **gestión del trabajo humano** y logística.

* **Kanban:** Tiene su origen en la logística industrial y la optimización de flujos productivos (Toyota Production System).
* **Scrum:** Emerge como un marco para la gestión de equipos y la producción de conocimiento en entornos complejos.
* **Agile:** Como manifiesto, fue una reacción a procesos excesivamente rígidos, priorizando la adaptabilidad y la interacción humana.

La adaptación de estos marcos al desarrollo de software ha traído grandes beneficios organizacionales. Sin embargo, dicha adaptación introduce una tensión estructural: el foco principal permanece en la **gestión del tiempo, las tareas y las personas**, no necesariamente en la corrección formal de la arquitectura del sistema.

#### 9.1.2 Diferencia Fundamental de Enfoque

La distinción clave entre estos marcos y la Metodología Fasética no es ideológica, sino conceptual. Responden a preguntas distintas:

* **Scrum y Kanban responden a:** *¿Cómo organizamos a las personas, las tareas y el tiempo disponible?*
* **Metodología Fasética responde a:** *¿Cómo construimos software correcto, coherente y mantenible?*

Estas preguntas no son equivalentes, y sus respuestas conducen a estructuras de trabajo radicalmente distintas. Mientras que un marco ágil puede considerar una tarea "Hecha" (Definition of Done) al cerrar una iteración, el modelo Fasético puede bloquear el avance si existen inconsistencias estructurales, independientemente del tiempo transcurrido.

#### 9.1.3 Origen de la Metodología Fasética

La Metodología Fasética surge exclusivamente desde el dominio del software. Su diseño parte de una necesidad técnica recurrente en sistemas complejos: aislar responsabilidades, eliminar ambigüedades entre capas y priorizar la **integridad del sistema** incluso a costa de reducir el avance visible a corto plazo. No pretende reemplazar la gestión humana de Agile, sino imponer restricciones de ingeniería que garanticen la calidad estructural.

### 9.2 Matriz de Diferenciación Estructural

La siguiente tabla desglosa las diferencias arquitectónicas y de gestión entre los modelos de referencia.

| **Criterio de Análisis** | **Metodologías Ágiles (Scrum / Kanban)** | **Modelo en Cascada (Waterfall)** | **Metodología Fasética (MEFH)** |
| --- | --- | --- | --- |
| **Objetivo Primario** | Velocidad de entrega y adaptabilidad al cambio de requisitos. | Planificación exhaustiva y ejecución secuencial lineal. | **Certeza Técnica** y estabilidad estructural mediante aislamiento. |
| **Unidad de Progreso** | *User Stories* completadas por Sprint. | Fases temporales completadas (Análisis, Diseño, Implementación). | **Fases Horizontales** validadas técnicamente. |
| **Gestión de Riesgo** | Mitigación reactiva / iterativa (feedback rápido del usuario). | Mitigación preventiva (documentación inicial extensa). | Mitigación **arquitectónica** (confinamiento del error por capas aisladas). |
| **Integración Frontend/Backend** | Frecuentemente paralela y asíncrona, con riesgo de desalineación. | Secuencial (Backend primero, luego Frontend), con riesgo de "Efecto Túnel". | Sincronizada obligatoriamente por **Contratos Espejo** antes de codificar lógica. |
| **Deuda Técnica** | Aceptable a corto plazo para cumplir plazos del Sprint (*Time-boxing*). | Baja inicialmente, pero extremadamente costosa de corregir post-implementación. | **Controlada por diseño**; el sistema impide el avance si existen inconsistencias estructurales no resueltas. |
| **Escenario Ideal** | Productos con incertidumbre de mercado alta (MVP, exploración). | Proyectos industriales con requisitos inmutables (Construcción, Hardware). | **Sistemas de misión crítica**, Fintech, y software donde la integridad de datos es prioritaria. |

### 9.3 Conclusión del Análisis

Mientras que *Agile* optimiza la **flexibilidad** ante mercados cambiantes y *Waterfall* optimiza la **previsibilidad temporal** en entornos estáticos, el Modelo Fasético optimiza la **integridad estructural**.

La Metodología Fasética no es incompatible con la gestión ágil de equipos; sin embargo, impone restricciones de ingeniería que priorizan la solidez del "cómo se construye" por encima de la velocidad del "cuándo se entrega", **especialmente en contextos donde el fallo estructural tiene alto costo operativo o legal**. Es una metodología de **Arquitectura de Software**, no solo de Gestión de Proyectos.

## 10. ANTI-PATRONES EN LA METODOLOGÍA FASÉTICA

La Metodología Fasética impone restricciones explícitas de ingeniería. Cuando estas restricciones se ignoran o se relajan por conveniencia, surgen anti-patrones que erosionan la integridad estructural del sistema. Estos anti-patrones no representan fallos puntuales, sino desviaciones sistemáticas que anulan los beneficios del modelo.

La identificación temprana de anti-patrones es obligatoria para preservar la certeza técnica y evitar la regresión arquitectónica.

### 10.1 Mezcla de Fases (Phase Bleeding)

* **Descripción:** Implementación de lógica, validaciones o transformaciones pertenecientes a una fase dentro de otra fase distinta.
* **Ejemplos:**
  + Validaciones de negocio dentro de la capa de comunicación.
  + Transformaciones de datos en la UI.
  + Decisiones de negocio dentro de la base de datos.
* **Por qué es peligroso:** Rompe el aislamiento de responsabilidades, impide el cierre de fases y convierte cualquier cambio futuro en retrabajo transversal.
* **Señal de alerta:** *"No puedo probar esta parte sin levantar todo el sistema".*

### 10.2 Contratos Inestables o Implícitos

* **Descripción:** Uso de contratos no formalizados, incompletos o que cambian durante fases posteriores.
* **Ejemplos:**
  + Tipos inferidos "porque el frontend lo necesita así".
  + Campos opcionales sin definición clara.
  + Contratos que se ajustan para "hacer que funcione".
* **Por qué es peligroso:** El contrato deja de ser ley y se convierte en una sugerencia. La sincronía de espejo se rompe y la integración se vuelve frágil.
* **Señal de alerta:** *"El frontend y backend 'se entienden' sin un contrato explícito".*

### 10.3 Avance por Presión Temporal (Time-Driven Progress)

* **Descripción:** Cierre de fases por cumplimiento de fechas en lugar de validación técnica.
* **Ejemplos:**
  + "Cerramos esta fase y luego lo arreglamos".
  + "Funciona, no toquemos más".
  + "Eso lo vemos en el próximo ciclo".
* **Por qué es peligroso:** Reintroduce los defectos estructurales que la metodología busca eliminar. El tiempo sustituye a la certeza.
* **Señal de alerta:** *El criterio de cierre es el calendario, no la validación.*

### 10.4 Confusión de Autoridad de la UI

* **Descripción:** Asignar a la interfaz de usuario autoridad lógica o de datos que excede su rol dentro del modelo fasético, confundiendo el contrato visual con la fuente de verdad estructural.
* **Aclaración Clave:** En el Vector **Frontend-First**, la UI (Caja 0) es fuente de verdad visual y funcional esperada, pero **no** es fuente de verdad de negocio, reglas, ni persistencia.
* **Ejemplos:**
  + Reglas de negocio implementadas exclusivamente en componentes visuales.
  + Validaciones críticas presentes solo en la UI.
  + Estructuras de datos definidas o alteradas desde la interfaz sin respaldo contractual.
  + Duplicación de lógica entre UI y Backend para "hacer que coincida".
* **Por qué es peligroso:** La UI es una representación, no un núcleo. Convertirla en autoridad lógica rompe el aislamiento de fases y genera inconsistencias entre intención visual y comportamiento real del sistema.
* **Señal de alerta:** *"Eso funciona porque así lo definimos en la pantalla".*
* **Regla Fasética:**
  + La UI define qué debe ser **posible**.
  + La lógica define qué es **correcto**.
  + Los datos definen qué es **verdadero**.

### 10.5 Falsos Micro-Cierres de Fase

* **Descripción:** Declarar una fase cerrada cuando aún depende de decisiones o implementaciones futuras.
* **Ejemplos:**
  + Contratos definidos "a medias".
  + Lógica incompleta esperando nuevos endpoints.
  + Comunicación implementada sin errores controlados.
* **Por qué es peligroso:** El sistema aparenta avance, pero acumula deuda estructural encubierta.
* **Señal de alerta:** *"Está cerrado... salvo por estos detalles".*

## 11. EVOLUCIÓN Y MICRO-CICLOS FASÉTICOS

La Metodología Fasética no concibe el desarrollo como un proceso lineal cerrado, sino como una construcción evolutiva controlada. El cambio es inevitable; el desorden no.

### 11.1 El Concepto de Micro-Ciclo

Un Micro-Ciclo Fasético se define como la **re-ejecución controlada de una o más fases horizontales** sobre un sistema existente, sin invalidar las fases ya cerradas que no se ven afectadas.

* **Regla de Impacto:** Ningún cambio se implementa de forma directa. Todo cambio debe traducirse previamente en **impacto de fase**, determinando qué componentes se ven afectados y en qué orden deben ser re-ejecutados. Las fases no afectadas permanecen inmutables.

### 11.2 Clasificación de Cambios (Taxonomía de Evolución)

La metodología distingue tres tipos de evolución para garantizar que el remedio no sea peor que la enfermedad:

#### TIPO A: EVOLUCIÓN LOCAL (Micro-Ciclo Simple)

* **Definición:** Cambio que afecta una sola fase sin romper contratos existentes.
* **Ejemplo:** Optimizar una regla de negocio (Fase 3) o ajustar un color en la UI (Fase 4).
* **Proceso:** Reabrir fase afectada $\rightarrow$ Implementar $\rightarrow$ Validar $\rightarrow$ Cerrar.

#### TIPO B: EVOLUCIÓN ENCADENADA (Micro-Ciclo Compuesto)

* **Definición:** Cambio que afecta varias fases consecutivas de forma horizontal.
* **Ejemplo:** Añadir un nuevo campo en la base de datos que debe reflejarse en la UI.
* **Proceso:** Reapertura secuencial (fases reabiertas). El cambio fluye ordenadamente.

#### TIPO C: REVISIÓN ESTRUCTURAL MAYOR

* **Definición:** Cambio que invalida las decisiones base o el dominio del sistema.
* **Regla:** No se permite el "parche". Se reconoce como un nuevo ciclo mayor del proyecto.

**Regla Fasética de Evolución:** Todo cambio debe poder clasificarse antes de implementarse. Si no puede clasificarse, no puede ejecutarse.

### 11.3 La Prohibición de la Anti-Evolución

El modelo prohíbe expresamente los cambios informales ("hotfixes" que saltan fases) bajo la premisa de que **la degradación controlada no es evolución**. El mantenimiento y el rescate de proyectos se rigen por las mismas leyes de certeza técnica que la creación inicial. La Anti-Evolución constituye un **Anti-Patrón Crítico** (ver **Sección 10**).

## 12. GLOSARIO FASÉTICO

El presente glosario define de manera formal y no ambigua los términos propios de la Metodología Fasética (MEFH). Las definiciones aquí expuestas son normativas; cualquier reinterpretación fuera de este marco se considera una desviación del modelo.

### 12.1 Caja Fasética

* **Definición:** Unidad lógica de responsabilidad dentro del Modelo de Ejecución por Fases Horizontales. Una Caja no representa complejidad interna ni tamaño de implementación, sino una frontera semántica y técnica claramente delimitada.
* **Propiedades Clave:**
  + Posee una responsabilidad única.
  + Expone contratos claros hacia otras cajas.
  + Su implementación interna es irrelevante para las fases adyacentes.
  + No permite dependencias implícitas ni fugas de lógica.

### 12.2 Fase Horizontal

* **Definición:** Etapa estructural del modelo que agrupa Cajas con la misma responsabilidad funcional a lo largo de todo el sistema.
* **Característica:** Las fases son horizontales porque atraviesan el proyecto completo y no representan momentos temporales, sino capas de responsabilidad técnica. El avance entre fases está condicionado exclusivamente a validación técnica, no a cronogramas.

### 12.3 Vector de Ejecución

* **Definición:** Orden estratégico de construcción del sistema que determina el punto de partida y la dirección del desarrollo.
* **Vectores Válidos:**
  + **Frontend-First:** El diseño visual (Caja 0) actúa como plano maestro funcional.
  + **Backend-First:** La estructura de datos y la lógica actúan como cimiento primario.
  + *Nota:* El vector no altera las fases existentes, solo su orden de activación.

### 12.4 Caja 0

* **Definición:** Caja especial y opcional, exclusiva del Vector Frontend-First. Representa el contrato visual y funcional esperado del sistema, validado con el cliente antes de la implementación de lógica real.
* **Restricción:** La Caja 0 es fuente de verdad visual, pero nunca fuente de verdad de negocio, reglas o persistencia.
* **Características:**
  + Contiene UI estática o mockups funcionales.
  + No incluye lógica real ni conexiones productivas.
  + Sirve como referencia obligatoria para la integración final del Frontend.

### 12.5 Contrato

* **Definición:** Definición formal y explícita de las estructuras de datos, tipos y expectativas de intercambio entre fases o sistemas.
* **Norma:** Un contrato es ley dentro del modelo. No puede inferirse, improvisarse ni ajustarse informalmente sin reapertura de fase.

### 12.6 Contrato Espejo

* **Definición:** Réplica exacta de un contrato definido en una fase, implementada en la fase consumidora. Su objetivo es garantizar sincronía estructural entre productor y consumidor, eliminando interpretaciones implícitas.
* **Norma:** Cualquier divergencia entre contrato y contrato espejo constituye un error crítico.

### 12.7 Sincronía de Espejo

* **Definición:** Principio que establece que todo contrato debe existir en versiones equivalentes y sincronizadas en ambos extremos del sistema (Backend y Frontend). La sincronía no es conceptual, es estructural y verificable en código.

### 12.8 Gatekeeping (Cierre de Fase)

* **Definición:** Acto formal de validación que autoriza el paso de una fase a la siguiente. Sin Gatekeeping explícito, no existe avance legítimo.
* **Requisitos de Cierre:**
  + Validación de frontera (no mezcla de responsabilidades).
  + Integridad mínima funcional.
  + Ausencia de dependencias ocultas.

### 12.9 Ley de No Mezclar Responsabilidades

* **Definición:** Regla suprema de la Metodología Fasética. Establece que ninguna fase puede contener lógica, decisiones o conocimiento que pertenezcan a otra fase, más allá de su contrato inmediato.
* **Consecuencia:** La violación de esta ley invalida la implementación como Fasética.

### 12.10 Micro-Ciclo Fasético

* **Definición:** Re-ejecución controlada de una o más fases horizontales sobre un sistema existente, sin afectar las fases no impactadas.
* **Propósito:** El micro-ciclo permite evolución sin degradación estructural y constituye el único mecanismo válido de cambio.

### 12.11 Revisión Estructural Mayor

* **Definición:** Tipo de evolución que invalida decisiones base del sistema (modelo de datos, dominio o arquitectura fundamental).
* **Norma:** No se permite su implementación como parche. Debe tratarse como un nuevo ciclo mayor del proyecto.

### 12.12 Anti-Patrón Fasético

* **Definición:** Desviación sistemática del modelo que compromete la integridad estructural del sistema.
* **Naturaleza:** Los anti-patrones no son errores aislados, sino prácticas que anulan los beneficios del enfoque fasético (ej. mezcla de fases, contratos implícitos, avance por presión temporal).

### 12.13 Anti-Evolución

* **Definición:** Práctica explícitamente prohibida que introduce cambios saltando fases, contratos o procesos de validación. Incluye hotfixes informales, parches directos en UI o lógica fuera de micro-ciclos.
* **Efecto:** La anti-evolución degrada el sistema irreversiblemente, incluso si el cambio "funciona" en lo inmediato.

### 12.14 Cierre Técnico

* **Definición:** Estado en el cual una fase o micro-ciclo cumple todas las condiciones de validación y queda congelado hasta una reapertura formal.
* **Norma:** El cierre técnico es absoluto; no admite excepciones por conveniencia.

## 13. DECLARACIÓN DE USO, COMPATIBILIDAD Y MÉTRICAS TEMPORALES

La Metodología Fasética define qué puede construirse y en qué condiciones técnicas, pero no impone un modelo único de gestión humana, planificación ni cadencia temporal. Esta separación es deliberada y constituye una de las fortalezas del modelo.

### 13.1 Dominio de Aplicación

La Metodología Fasética opera exclusivamente en el dominio de la **ingeniería de software**, no en la gestión organizacional.

* **Define:** Cómo se construye un sistema, qué significa avance técnico real y cuándo una parte del sistema es estructuralmente estable.
* **No Define:** Horarios de trabajo, asignación de personas ni ritmos de entrega comerciales.

### 13.2 Compatibilidad con Metodologías de Gestión

La Metodología Fasética es compatible con marcos de gestión como **Scrum, Kanban u otros sistemas ágiles**, siempre que se respete la jerarquía de responsabilidad.

**Principio de Compatibilidad:** La gestión organiza el trabajo. La metodología fasética gobierna la arquitectura.

* **Ejemplos Válidos:**
  + Uso de Sprints como contenedores temporales de trabajo fasético.
  + Uso de tableros Kanban para visualizar el estado de fases.
  + Uso de ceremonias ágiles para coordinación humana.
* **Ejemplos Inválidos:**
  + Cierre de fases por finalización de Sprint (falso positivo).
  + Modificación de contratos por presión de WIP (Work in Progress).
  + Aceptación de deuda estructural para cumplir fechas (Time-Driven).

### 13.3 El Rol del Tiempo en el Modelo Fasético

En la Metodología Fasética, el tiempo no es una fuerza de decisión, sino una **métrica de observación**. El modelo distingue entre:

1. **Tiempo Aparente:** Velocidad percibida por entregables visibles.
2. **Tiempo Real de Ingeniería:** Tiempo necesario para cerrar fases sin retrabajo futuro.

La metodología prioriza el segundo, bajo el principio de que **el tiempo perdido corrigiendo errores estructurales supera ampliamente el tiempo invertido en prevenirlos.**

### 13.4 Métricas Temporales Faséticas

El avance temporal del proyecto se mide mediante indicadores técnicos objetivos:

1. **Tiempo de Cierre de Fase (TCF):** Duración desde la apertura de una fase hasta su validación y cierre técnico.
2. **Duración de Micro-Ciclo (DMC):** Tiempo requerido para implementar, validar y cerrar un micro-ciclo evolutivo completo.
3. **Estabilidad Post-Cierre (EPC):** Tiempo durante el cual una fase permanece cerrada sin reapertura.
4. **Índice de Reapertura:** Número de reaperturas por fase. Un índice bajo indica alta calidad estructural; un índice alto indica fragilidad en las fases precedentes.

Estas métricas permiten evaluar la eficiencia sin sacrificar la corrección técnica.

### 13.5 Declaración Final de Integración

La Metodología Fasética puede integrarse en cualquier entorno organizacional, ágil o tradicional, siempre que se respete su principio fundamental:

**Ningún objetivo temporal justifica la violación de la integridad estructural.**

Cuando este principio se mantiene, el tiempo deja de ser una amenaza y se convierte en una consecuencia natural de la solidez del sistema.

## 14. CHECKLIST DE CUMPLIMIENTO FASÉTICO (Instrumento de verificación técnica)

Esta sección no es teórica. Es operativa. Sirve para responder una sola pregunta incómoda: **¿Este proyecto está siendo construido bajo la Metodología Fasética o solo la está citando?**

### 14.1 Propósito del Checklist

El Checklist de Cumplimiento Fasético existe para:

* Detectar violaciones estructurales tempranas.
* Evitar "adopciones cosméticas" de la metodología.
* Establecer un estándar mínimo verificable.

**El cumplimiento no es subjetivo.**

### 14.2 Checklist General (Obligatorio)

Un proyecto cumple con la Metodología Fasética solo si **TODAS** las siguientes condiciones se satisfacen:

* **Fases y Cierre**
  + Cada fase tiene criterios explícitos de cierre.
  + No existe cierre parcial o "provisional".
  + Las fases cerradas no reciben cambios directos.
* **Contratos**
  + Cada transición de fase genera un contrato explícito.
  + Los contratos no son modificados retroactivamente.
  + Toda evolución respeta contratos o inicia un micro-ciclo.
* **Flujo de Cambios**
  + Todo cambio fue clasificado (Tipo A, B o C).
  + Ningún cambio saltó fases.
  + No existen hotfixes fuera de micro-ciclos.
* **Fuente de Verdad**
  + La fuente de verdad está claramente definida por fase.
  + La UI no persiste estado autoritativo.
  + El dominio no depende del frontend.
* **Anti-Patrones**
  + No existe mezcla de fases activas.
  + No se usan entregables visuales como evidencia técnica.
  + No se prioriza velocidad sobre estabilidad estructural.

### 14.3 Checklist de Auditoría Técnica

Este checklist se usa cuando el proyecto ya existe:

* Las fases cerradas coinciden con el código real.
* No hay lógica de negocio en capas no autorizadas.
* Los datos no dependen de comportamiento de UI.
* Los micro-ciclos están documentados.
* Las reaperturas están justificadas.

**Regla de Auditoría:** Si más del 20% falla, el sistema se considera estructuralmente degradado.

Nota sobre el Umbral de Degradación:

El valor del 20% no representa una constante matemática rigurosa, sino un umbral operativo de auditoría. Se utiliza como indicador práctico para distinguir desviaciones menores de una degradación estructural sistémica.

Importante: Cualquier violación de reglas críticas (ver 14.2) invalida inmediatamente el cumplimiento fasético, independientemente del porcentaje total.

### 14.4 Resultado del Checklist

* **Cumple** $\rightarrow$ El proyecto es faséticamente válido.
* **Cumple parcialmente** $\rightarrow$ Riesgo controlable.
* **No cumple** $\rightarrow$ El proyecto opera bajo anti-patrones.

*Este resultado no es negociable ni depende del contexto comercial.*

### 14.5 Declaración de Responsabilidad

Aplicar la Metodología Fasética implica aceptar que:

**El incumplimiento de una sola regla estructural invalida la integridad del sistema, aunque el software "funcione".**

## 15. CASOS DE ESTUDIO RESUMIDOS (BEFORE / AFTER)

Esta sección presenta estudios de caso cualitativos que ilustran la aplicación práctica de la Metodología Fasética en escenarios reales de desarrollo. El objetivo no es establecer métricas estadísticas universales, sino demostrar el impacto estructural del modelo sobre la estabilidad, claridad técnica y reducción de retrabajo.

Los casos se presentan como comparaciones entre ciclos de desarrollo no estructurados y ciclos faséticos, manteniendo el enfoque en la arquitectura del proceso y no en factores personales u organizativos.

### 15.1 Caso A: Sistema Web de Portafolio (Ciclo No Estructurado vs. Ciclo Fasético)

#### Contexto Inicial (Ciclo No Estructurado)

En un primer ciclo de desarrollo de un sistema web de tipo portafolio, el proyecto fue abordado sin una metodología de separación explícita de fases. El desarrollo priorizó la implementación directa de funcionalidades sin contratos formales ni cierre técnico de responsabilidades.

* **Características del enfoque inicial:**
  + Backend desarrollado sin contratos definidos de consumo.
  + Lógica escrita de forma reactiva, ajustándose a necesidades emergentes.
  + Integración Frontend/Backend realizada por prueba y error.
  + Ausencia de criterios claros de cierre de fase.
* **Resultado observado:**
  + Incremento del retrabajo durante la integración.
  + Dificultad para mantener coherencia entre capas.
  + Sensación de avance funcional acompañada de fragilidad estructural.
  + Complejidad creciente ante cambios menores.

#### Aplicación del Modelo Fasético (Ciclo Posterior)

En un segundo ciclo evolutivo del mismo sistema, se aplicaron principios fundamentales de la Metodología Fasética, especialmente bajo el Vector Frontend-First.

* **Cambios introducidos:**
  + Definición previa del contrato visual (Caja 0).
  + Formalización de contratos espejo entre Frontend y Backend.
  + Separación estricta entre lógica, comunicación y presentación.
  + Cierre técnico de fases antes de avanzar.
* **Resultado observado:**
  + Integración fluida entre Frontend y Backend.
  + Eliminación casi total del retrabajo correctivo.
  + Reducción significativa del tiempo de implementación.
  + Mayor coherencia entre intención visual y comportamiento del sistema.

#### Conclusión del Caso A:

La aplicación del modelo fasético permitió transformar un sistema funcional pero frágil en un sistema estructuralmente coherente, demostrando que la reducción de ambigüedad arquitectónica tiene un impacto directo en la eficiencia real del desarrollo.

### 15.2 Caso B: Sistema de Facturación con Backend Existente (Integración Fasética)

#### Contexto del Sistema

El segundo caso corresponde a un sistema de facturación con backend previamente implementado y operativo. El desafío principal consistía en integrar un frontend moderno bajo un cambio tecnológico de stack, manteniendo compatibilidad y estabilidad del sistema central.

* **El sistema presentaba:**
  + Backend funcional con lógica de negocio consolidada.
  + Necesidad de definición clara de contratos de consumo.
  + Riesgo de desalineación entre capacidades reales del backend y la implementación del frontend.

#### Aplicación de Principios Faséticos

La integración del frontend se realizó aplicando las fases iniciales del Ciclo Frontend bajo el modelo fasético:

1. **Fase 1:** Definición de contratos espejo estrictamente alineados al backend.
2. **Fase 2:** Implementación de la capa de comunicación sin lógica de negocio.
3. Aplicación de **Gatekeeping** antes de avanzar a lógica y UI.

En un módulo crítico (autenticación), se permitió una integración controlada de Fases 3 y 4 debido a la existencia de una referencia funcional previa, manteniendo el aislamiento de responsabilidades como criterio rector.

* **Resultado observado:**
  + Integración exitosa del módulo de autenticación en un intervalo reducido.
  + Eliminación de ambigüedad sobre responsabilidades entre capas.
  + Facilidad para extender funcionalidades posteriores sin refactorización estructural.
  + Mayor previsibilidad técnica del avance real del sistema.

#### Conclusión del Caso B:

El modelo fasético demostró ser especialmente eficaz como herramienta de integración y estabilización en sistemas existentes, incluso bajo condiciones de cambio tecnológico, al imponer claridad contractual y aislamiento de responsabilidades desde las fases iniciales del frontend.

### 15.3 Conclusiones Generales de los Casos de Estudio

A partir de ambos casos se desprenden los siguientes patrones consistentes:

* La separación estricta de fases reduce el retrabajo más que cualquier optimización puntual.
* La definición temprana de contratos elimina fricción en la integración.
* El avance estructuralmente validado supera al avance meramente visible.
* El modelo es aplicable tanto en creación desde cero como en evolución de sistemas existentes.

**Estos casos refuerzan la premisa central de la Metodología Fasética:**

**La estabilidad no es una consecuencia tardía del desarrollo; es una decisión temprana de ingeniería.**

## 16. APÉNDICES Y GUÍAS DE REFERENCIA

Esta sección final contiene material suplementario destinado a facilitar la adopción práctica de la metodología. A diferencia del cuerpo principal (normativo), estos apéndices son ilustrativos e informativos.

### 16.1 Apéndice A: Ejemplo Mínimo de Implementación (EMI)

El **EMI** es la demostración más pequeña posible de que la metodología se puede ejecutar sin contradicciones. No es un tutorial de programación, sino un "mapa de ruta" abstracto que valida el cumplimiento estructural.

* **Propósito:** Evitar interpretaciones creativas ("Yo lo entendí distinto") y ofrecer un anclaje técnico visual sin comprometerse con un stack tecnológico específico.

#### Sistema de Ejemplo: "Servicio de Autenticación" (Abstracto)

**A. Backend (Vector Backend-First)**

1. **Fase 1 – Persistencia**
   * Objeto: UserTable
   * Campos: id, email, password\_hash, created\_at
2. **Fase 2 – Contrato**
   * Objeto: UserDefinition
   * Tipos: id: UUID, email: string
3. **Fase 3 – Lógica**
   * Función: authenticateUser(credentials)
   * Retorno: UserDefinition | Error
4. **Fase 4 – Comunicación**
   * Endpoint: POST /auth/login
   * Input: credentials
   * Output: UserDefinition (JSON)

**B. Frontend**

1. **Fase 1 – Contrato Espejo**
   * Objeto: UserDTO
   * Tipos: id: string, email: string (Copia exacta de UserDefinition)
2. **Fase 2 – Comunicación**
   * Servicio: authService.login() (Transporte puro)
3. **Fase 3 – Lógica**
   * Hook/Store: useAuth() (Manejo de estado de sesión, sin UI)
4. **Fase 4 – UI**
   * Componente: LoginForm (Consume useAuth, renderiza visualmente)

**Nota:** Este ejemplo cumple al 100% con la metodología: no hay mezcla de fases, los contratos son explícitos y la UI no decide nada, solo muestra.

### 16.2 Apéndice B: Errores Comunes de Adopción

Lista de comportamientos que parecen faséticos pero violan sus principios nucleares (Anti-Patrones de Adopción).

1. **"Usar contratos pero no cerrarlos":** Definir interfaces (TypeScript/Java) pero cambiarlas cada vez que la UI lo "necesita", convirtiendo el contrato en una sugerencia.
2. **"Hacer Frontend-First sin Caja 0":** Empezar a programar el frontend sin tener un diseño validado, lo que obliga a improvisar lógica sobre la marcha.
3. **"Mezclar fases bajo presión":** Inyectar lógica de negocio en el controlador (Fase 4) "solo por esta vez" para llegar a la fecha de entrega.
4. **"Validación cosmética":** Creer que porque el código está en carpetas separadas, las responsabilidades están separadas (p.ej. una carpeta services que contiene código de UI).

### 16.3 Apéndice C: Mapa de Compatibilidad Arquitectónica

La Metodología Fasética no reemplaza arquitecturas de software, sino que las gobierna. Este mapa muestra cómo se alinea con los patrones más comunes de la industria.

* **Clean Architecture / Onion Architecture:**
  + Fase 1/2 $\approx$ *Domain / Entities*
  + Fase 3 $\approx$ *Use Cases / Application Services*
  + Fase 4 $\approx$ *Infrastructure / Controllers / Presenters*
  + *Alineación:* Natural y directa.
* **Arquitectura Hexagonal (Ports & Adapters):**
  + Fase 1 $\approx$ *Secondary Adapters (Persistence)*
  + Fase 2/3 $\approx$ *Hexagon (Domain + Ports)*
  + Fase 4 $\approx$ *Primary Adapters (API/Web)*
  + *Alineación:* Alta compatibilidad; las "Fases" actúan como fronteras del hexágono.
* **MVC (Model-View-Controller):**
  + Fase 1/2 $\approx$ *Model*
  + Fase 4 $\approx$ *Controller + View (API response)*
  + *Advertencia:* MVC tradicional tiende a mezclar lógica en el Controlador. La Metodología Fasética exige extraer esa lógica a una capa de Servicios (Fase 3) explícita, dejando el Controller "delgado" (solo transporte).

## 17. ESTATUTO DEL ESTÁNDAR (META-ESPECIFICACIÓN)

Esta sección final actúa como la capa de gobernanza del documento. **No introduce nuevas reglas operativas** para el desarrollo, sino que protege la integridad semántica y estructural de las secciones 1 a 16.

### 17.1 Declaración de Propósito (Canon Técnico)

Este documento constituye la definición canónica del **Modelo de Ejecución por Fases Horizontales (MEFH)**.

* **Naturaleza:** Es descriptivo en sus fundamentos y prescriptivo en sus restricciones estructurales.
* **Autoridad:** Ante cualquier discrepancia entre una implementación práctica y este texto, la definición aquí expuesta prevalece como la única fuente de verdad metodológica.

### 17.2 Jerarquía Normativa

Para efectos de adopción y auditoría, el documento se estructura en cinco niveles de autoridad decreciente pero complementaria:

1. **Fundamento (Secciones 1-8):** Innegociable. Define la filosofía y la estructura física.
2. **Control (Secciones 9-14):** Obligatorio. Define las reglas de validación y auditoría.
3. **Evidencia (Sección 15):** Ilustrativo. Provee contexto empírico.
4. **Guía (Sección 16):** Didáctico. Facilita la comprensión.
5. **Canon (Sección 17):** Meta-regla. Protege el estándar de interpretaciones subjetivas.

### 17.3 Política de Modificación y Versionado

El estándar Fasético es inmutable en sus **4 Pilares (Sección 2.3)**. Cualquier modificación que altere, relaje o elimine estos pilares constituye la creación de una metodología distinta y no debe utilizar el nombre "Metodología Fasética".

* **Versiones Menores (x.1):** Aclaraciones y expansiones de apéndices.
* **Versiones Mayores (X.0):** Revisiones profundas que no rompen los pilares fundamentales.

### 17.4 Cláusula de Cierre (La "Piedra Angular")

La Metodología Fasética no es un dogma, es una estructura de defensa técnica. Su adopción implica la aceptación de que la **Certeza Técnica** es el activo más valioso del software.

**"Lo que no está estructuralmente definido, está técnicamente endeudado."**

## 18. RESUMEN TÉCNICO Y DECLARACIÓN FINAL

Esta sección final consolida el propósito, los límites y la promesa de valor de la Metodología Fasética, sirviendo como epílogo técnico y declaración de alcance definitivo.

### 18.1 Síntesis del Modelo

El Modelo de Ejecución por Fases Horizontales (MEFH) no es un sustituto de la gestión humana (Agile, Scrum), sino un estándar de ingeniería que responde a la pregunta **"¿Cómo se construye correctamente?"**.

* **Problema que resuelve:** La inestabilidad estructural, el "código espagueti" y el retrabajo infinito causado por la mezcla de responsabilidades.
* **Solución que propone:** La segregación estricta de responsabilidades en fases horizontales inmutables, validadas mediante contratos espejo.
* **Diferenciador clave:** Prioriza la **Certeza Técnica** sobre la velocidad aparente de entrega.

### 18.2 Alcance y Límites

* **Lo que SÍ cubre:** Arquitectura de software, flujo de datos, definición de contratos, criterios de validación técnica y estrategias de evolución de código.
* **Lo que NO cubre:** Gestión de equipos, asignación de salarios, cultura organizacional, herramientas de comunicación (Slack/Jira) ni cronogramas comerciales.

### 18.3 Condiciones de Validez

Un sistema puede denominarse legítimamente "Fasético" únicamente si cumple con los **4 Pilares Innegociables** y supera el **Checklist de Cumplimiento (Sección 14)** sin violaciones críticas.

**Advertencia de Invalidez:** Cualquier implementación que permita la mezcla de fases, la ausencia de contratos explícitos o la priorización sistemática de la velocidad sobre la estructura, pierde automáticamente la denominación de "Fasética", independientemente de la intención del equipo.

### 18.4 Declaración de Cierre

La adopción de este modelo exige una madurez técnica que acepta una verdad incómoda: **el orden cuesta tiempo al principio, pero regala velocidad al final.**

La Metodología Fasética no promete ser la forma más rápida de escribir la primera línea de código. Promete ser la forma más segura de escribir la última.

**"La certeza no es negociable."**

**Nota Normativa Final:** Este documento completo (Secciones 1 a 18) representa la especificación oficial de la Metodología Fasética. Su integridad debe mantenerse para garantizar los resultados prometidos por el modelo.

**Autoría y Propiedad:** Esta metodología es una creación intelectual original de **Andriy Lionel Pastrana Cajavilca**, desarrollada en el seno de la organización **Studios TKOH!**. Se comparte con la comunidad global de ingeniería como un estándar abierto para elevar la calidad del desarrollo de software.

*Documento confidencial. Propiedad intelectual del autor.*