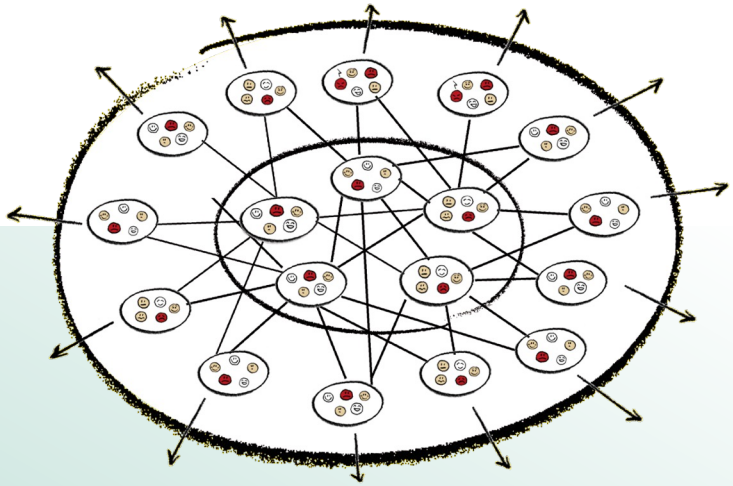
the betacodex network

since 2008



INTRODUCCIÓN AL **DESARROLLO DE SOFTWARE ORIENTADO AL TIEMPO (TOSD)**

Incorporando el flujo y la estructura del tiempo al desarrollo ágil de software – más allá de Scrum, Kanban y Lean

BetaCodex Network Associates

Niels Pflaeging | Sebastian Kubsch | Traducción: Ernesto R. Corona

BetaCodex Network White Paper Nº 26 | Octubre de 2025 | betacodex.org/white-papers

26

Dedicatoria

Este documento de investigación de BetaCodex
está dedicado a la memoria de

Ernst Weichselbaum

(13.04.1944 – 27.12.2024),

filósofo de la *orientación al tiempo*
y pionero de los *Sistemas de Trabajo Orientados al Tiempo*

Desarrollo de Software Orientado al Tiempo: Aprendiendo a usar la estructura definitiva en el trabajo y en las organizaciones

El tiempo está tan presente en la vida que a veces pasa inadvertido. Es una estructura poderosa. Sin embargo, el concepto de tiempo ha sido subestimado y poco aprovechado en el desarrollo de software y, en general, en las organizaciones. Las nociones de iteración y time-boxing de Scrum o Agile no llegaron lo suficientemente lejos para aprovechar de verdad el poder del tiempo en el desarrollo de software. Hoy, en la mayoría de equipos y organizaciones, el desarrollo de software sigue siendo afectado por mitos, secretismo, jerarquía y obediencia, a menudo atrapado en rituales ineficientes, roles innecesarios y procesos burocráticos.

Agile, por desgracia, no ha cambiado demasiado este panorama. Desde 2001, cuando se redactó el Manifiesto Ágil para el Desarrollo de Software, muchos debates clave dentro del movimiento han girado en torno a malentendidos: si planificar o no planificar, cómo planificar, cómo hacer retrospectivas, los roles de Product Owner, Scrum Master, Agile Coach, cómo escalar, cómo escribir user stories, cómo evaluarlas, etc. Sostenemos que estas discusiones estuvieron mal enfocadas desde el principio y estaban destinadas a aportar poco o nada; por ello, el movimiento Agile perdió tracción e ímpetu para el cambio.

Hoy está claro que Agile ha producido poco progreso real en la manera de organizar el desarrollo de software. En este documento se argumenta que la orientación al tiempo consistente es la mejor vía para superar el waterfall, el "haz-lo-que-quieras" y el *laissez-faire* en desarrollo de software. La disciplina colectiva, el enfoque y el flujo no se obtienen mediante escalado o gestión, ni con roles endebles o supuestos valores: deben abordarse mediante estructuras en el tiempo. **El tiempo es la estructura definitiva: solo hay que poner el tiempo a trabajar para impulsar de verdad el desarrollo de software.**

«Desde la perspectiva de lo viejo, lo nuevo siempre está mal. En la manera antigua de pensar, nada verdaderamente nuevo es posible.»
Ernst Weichselbaum

«Si se quiere cambiar el mundo, hay que empezar por los patrones en la cabeza: allí no hay nadie que se interponga.»
Ernst Weichselbaum

«Es mejor diseñar la puntualidad que gestionar los retrasos.»
Ernst Weichselbaum

«Si se necesita decidir, entonces no se sabe realmente qué se quiere. Comenzar consiste exclusivamente en comenzar.»
Ernst Weichselbaum

Orientación al tiempo:
Conecta con nuestras necesidades más humanas

Uno de los objetivos clave de todos los Sistemas de Trabajo Orientados al Tiempo (presentados en un documento aparte) es devolver un sentido de logro a las personas en el trabajo. Estamos hablando de ese sentimiento que puede experimentar una agricultora al final del día cuando piensa: «La tarea del día está hecha». Hablamos de logro en relación con el trabajo realizado –una necesidad humana fundamental –, distinto de discursos abstractos sobre propósito, objetivos o "estrellas polares - north stars" frecuentes en la gestión de Mando y Control.

Ese sentido de logro diario (o frecuente) se enraíza en el trabajo mismo y en la comprobación individual de que hay un cierre respecto a lo que se hizo, un final. El trabajo del conocimiento dirigido jerárquicamente, el waterfall o el laissez-faire privan a quienes lo realizan del derecho a experimentar esa relevancia y cierre con frecuencia.

Las consecuencias de esa privación y del desapego sistemático del sentir el logro en el trabajo son graves. La orientación al tiempo es la única vía para restaurar la realización del flujo en el trabajo del conocimiento y la unidad entre el trabajo propio y el beneficio que aporta a otras personas. **Ese sentimiento está en el corazón del TOSD.**

«"La jornada está cumplida", reza un viejo refrán campesino. Toda jornada empieza y termina; el día transcurre entre ambos. Hay un gran valor emocional en haber concluido: "Hoy hice mi parte". Quién está en una cadena de montaje no tiene eso: la producción en masa y la automatización lo impiden. Se rompe el sistema sociotécnico.»
Ernst Weichselbaum

«Al respetar acuerdos de costura (seam) con otras personas, se experimenta un valor emocional añadido que la dirección jerárquica no puede alcanzar.»
Ernst Weichselbaum

«Cuando el trabajo es menos monótono, aumentamos la cantidad total de sonrisas.»
Ernst Weichselbaum

Poner el tiempo y el flujo al mando del desarrollo de software, no a los jefes que lo frenan

El propósito de los inicios del movimiento Agile fue sustituir el Mando y Control – conocido entonces, por falta de mejor término, como waterfall. Hoy se sabe que ese intento fracasó. En su lugar, los patrones de Mando y Control en desarrollo de software y empresas tecnológicas fueron reafirmados por conceptos, herramientas y escalado "ágiles". Scrum, Kanban o Lean no han sido suficientes para romper esos patrones: roles, rituales y patrones propios de Agile han sido apropiados para fines jerárquicos. La sobrecarga y las actividades no productivas han aumentado desde los años 90, al tiempo que han crecido las barreras al flujo. TOSD debe ser diferente si quiere marcar la diferencia y cumplir la promesa de la agilidad.

En consecuencia, TOSD rompe con muchos patrones clásicos de waterfall y de Agile: por ejemplo, con la noción de equipos de desarrollo y el escalado, o con la idea de sprints -torpes e imprácticos desde una orientación al tiempo adecuada-.

Lo que hace a TOSD menos susceptible a la gestión coercitiva son sus principios: como el BetaCodex, del que deriva, TOSD se basa en principios significativos (las "siete leyes") donde la orientación al tiempo delinea lo que persigue y lo que rechaza (evitar los patrones de Mando y Control).

En la página siguiente se presentan las siete leyes; también aparecen en forma más concisa en la mirada general de la p. 23.

«La orientación al tiempo es conectora, no divisoria. Es puente, no muro. Es requisito para redes organizativas y autoorganización.»
Ernst Weichselbaum

«Si hay que dirigir, la culpa es propia. Diseñe bien la empresa; no la dirija.»
Ernst Weichselbaum

«Tener una opinión sobre lo que viene es valioso. Dirigir no lo es.»
Ernst Weichselbaum

«Donde mandan las excepciones, no puede emerger un sistema robusto.»
Ernst Weichselbaum

«La coacción es una declaración de guerra: sólo "triunfa" a costa de otras personas.»
Ernst Weichselbaum

La orientación al tiempo es la base para lograr auto-organización y colaboración sostenida en el desarrollo de software



Lenguaje/pensamiento asociado al volumen (que se abandona):

Escalamiento, backlog, story points, throughput, estimación, pronóstico, utilization, tamaño óptimo de lote, economías de escala, fusiones y adquisiciones, cuotas de mercado, reservas mínimas, cumplimiento de objetivos, programación, PI/Big Room/Quarterly Planning

Lenguaje/pensamiento asociado al tiempo (que se adopta):

Porción diaria, Punto OK, adherencia a time-slots, tiempo de Realización fijo, tiempo optimo, flujo, ritmo/pulso, "la tarea del día está hecha", espacio blanco, velocidad, fecha deseada, coordinación anticipada, puntualidad, entregas acordadas, acuerdos de costura entre personas e equipos, apretón de manos

Los 7 principios clave o “leyes” del Desarrollo de Software Orientado por el Tiempo

1. Orientación al tiempo. A diferencia de la orientación a la capacidad o al volumen, la orientación al tiempo actúa como un conector o un puente: habilita y refuerza la autoorganización.

2. Flujo y Punto OK. El flujo es una expresión clave del tiempo y el movimiento en el trabajo y en la creación de valor. Requiere eliminar las barreras entre el trabajo y el uso que los clientes hacen de los resultados (software, funcionalidades). Crear condiciones para el flujo permite que los problemas en los sistemas de trabajo se hagan visibles al generar restricciones.

3. Transparencia y Lista de Ítems. Proporciona los medios para el autocontrol y la autogestión. Siempre debe referirse a equipos o grupos de personas – nunca a individuos.

4. Autoorganización coherente. Es la única vía para liberar un alto compromiso y niveles elevados de motivación en el trabajo. Es una condición previa para el alto rendimiento. Requiere descentralización y no se lleva bien con la dirección o centralización.

5. El tiempo es fijo, la capacidad varía. No al revés.

6. HEU: Hablen Entre Ustedes *(Talk To Each Other).* Tener las conversaciones necesarias aquí y ahora es el corazón de la autoorganización y el flujo en los sistemas sociales. El diálogo, el discurso y el acuerdo deben estar en el centro del desarrollo de software contemporáneo. Las verdaderas conversaciones hacen innecesarias las reglas, la dirección y la centralización.

7. Coherencia gerencial. Sin ella, nada funciona. Es lo que los gerentes deben aportar para implementar TOSD y garantizar su continuidad.

«En orientación al tiempo, la capacidad no se programa (“flexible”): se diseña como sistema.»
Ernst Weichselbaum

«Dejar que las cosas fluyan es mejor que agruparlas. Un ítem grande no es más que cinco ítems pequeños con la misma dirección de entrega.»
Ernst Weichselbaum

«Diseñe el proceso para que no necesite ser monitoreado: es mejor ser puntual que administrar retrasos.»
Ernst Weichselbaum

«La autoridad no es propiedad de individuos: se genera en la interacción entre al menos dos personas; proviene del acuerdo.»
Ernst Weichselbaum

El Desarrollo de Software Orientado por el Tiempo no necesita estas cosas

Artefactos de desarrollo de software que deben abandonarse	Qué hacer en su lugar
Sprints/Cadencias. Forma de orientación por lotes: como pantalones demasiado holgados. No están realmente orientados al flujo; reducen la velocidad y limitan la autoorganización	Time-box por ítem individual; elimina la necesidad de cadencias
Iteraciones/MVP. Mitos del "ágil". El desarrollo iterativo genuino es raro en software y no constituye un caso distintivo del trabajo de desarrollo	Abandonar la idea y el término ("iteración/MVP")
Backlogs. Seducen a tener malas conversaciones sobre cosas equivocadas (p. ej., priorización, planificación de sprint, poder). Erosionan la aclaración, la conversación de calidad y el discurso. Al volverse "infinitos", minan la sensación de logro. Son el epítome de lo que está mal en Agile/Scrum	Entender los grandes emprendimientos a 30–90 días; mejorar la conversación sobre foco y urgencia
"Scrum teams/Development teams/Scalable teams". Para desarrollar, no se necesitan equipos per se: un "ítem" bien dimensionado lo resuelven 1 o 2 personas. Imponer equipos impide el flujo. Superar esta ilusión elimina roles disfuncionales como Scrum Master, Product Owner, Agile Coach	Desarrollo individual o en dúos ("tándems"); abandonar "equipos de desarrollo"
Estimación/Puntos de historia. Tópicos grandes en Agile, también engañosos y derrochadores. En TOSD sobran: la Realización está time-boxed y dura máximo 3 días	Sustituir por orientación al tiempo y time-boxing
Tableros Kanban. Herramientas de visualización que acentúan estructuras y conversaciones defectuosas, enfocadas en lo equivocado. Si son digitales y separadas de la plataforma de desarrollo, generan desperdicio, steering e intransparencia	Administrar listas de trabajo en GitHub/GitLab o equivalentes y visualizar en el espacio físico
Retrospectivas. Mejor elevar la interacción del proceso ("Hablen Entre Ustedes") que "revisar" en grupos fijos y a intervalos arrítmicos	Conversar cuando sea necesario, en cualquier momento (no esperar la "retro")

El Desarrollo de Software Orientado por el Tiempo no necesita estas cosas (continuación)

Artefactos de desarrollo de software que deben abandonarse	Qué hacer en su lugar
Marcos de escalado/"coordinación". Basados en suposiciones erróneas de steering y control mediante "equipos apilados"; jerarquizan, burocratizan y precipitaron la caída del movimiento Agile. Enemigos naturales de TOSD	Dejar de usarlos
Planificación (PI/Big Room/Quarterly). Se desvía de la realidad, del trabajo y del ahora; es lo opuesto al flujo y la orientación al tiempo	Usar OpenSpace DEV para discurso y solución y para crear unidad
Métricas "ágiles" y objetivos fijos. Obscurecen la realidad. Evitar fijar objetivos a individuos, equipos o grupos	Emplear Objetivos Relativos; P&L de equipo; juicio aplicado a la realidad del flujo
Mediciones de capacidad. La capacidad no debe medirse ni debatirse, sino cambiarse/influenciarse (acción, no métricas).	Hacer que la capacidad oscile según demanda con time-boxes fijos
Roles Lead/Scrum Master/Product Owner/Agile Coach. Desempoderan a desarrolladores, generan steering y desperdicio, crean impedimentos al flujo, y tienden a ocultar problemas	Aplicar Cell Structure Design; equipos pequeños (4–8 persones) e integrados funcionalmente más allá de desarrollo
Aprobaciones/"Production Readiness" reviews. Impedimentos al flujo	Eliminarlas y dejar de usar el término
User stories. Mito/ilusión ágil; sustituto pobre de la conversación grupal con el cliente	Escribir conceptos correctos; visualizar el flujo en físico; aportar claridad
Modelos de "madurez ágil"/categorizaciones de personas/"readiness". Inútiles y enjuiciadores de personas; lo que se requiere es actuar sobre el sistema	Dejar de juzgar personas; trabajar el sistema; abolir vocabulario de juicio
Financiación por proyectos/"pools" de recursos. Presupuestos, aprobaciones y asignaciones de desarrolladores.	Eliminar presupuestos; medir P&L de equipo

Una innovación clave del Desarrollo de Software Orientado por el Tiempo: el Punto OK

El Desarrollo de Software Orientado por el Tiempo se basa en una distinción temporal llamada Punto OK. Esta distinción permite separar con claridad la Conceptualización de la Realización del trabajo de desarrollo de software. Sin el Punto OK, la orientación al tiempo en desarrollo es inconcebible: solo puede acotarse la Realización en el tiempo (time-box) si antes se produjo un concepto que puede ser realizado. El Punto OK es el lugar donde Conceptualización y Realización se dan un apretón de manos – es una costura, una unión, no un separador (tal como muestra el diagrama de la página: "Una costura, no un separador").



Desarrollo de Software Orientado por el Tiempo: el Punto OK da estructura al proceso. La Realización puede acotarse en el tiempo



Conceptualización

Debe ser rápida

Puede realizarla 1 o 2 personas.

Describe el Qué, no el Cómo.

Involucra a personas autorizadas con la maestría adecuada para este trabajo

Es seguido por la Realización

Conduce a un concepto suficiente, que justifica un Punto OK para el ítem

Los conceptos se documentan por escrito y son visibles para todos

Incluye definiciones de requisitos, la ventana de Realización (*time-box*) y las personas involucradas

Pruebas y documentación son parte del proceso, no actividades separadas (TDD)

Punto OK

Es un "apretón de manos"

Sin él, ningún ítem puede avanzar.

Se basa siempre en acuerdo o decisión consultiva, no en mandato.

Realización

Está estrictamente acotada en el tiempo: 1, 2 o 3 días

Pueden ejecutarla 1 o 2 personas, dentro del time-box acordado (cumplirlo es una cuestión de honor)

Incluye o es seguida por el despliegue a clientes/usuarios

Puede involucrar personas distintas de la Conceptualización

Si hay duda sobre el tiempo de Realización, aún no se alcanzó el Punto OK

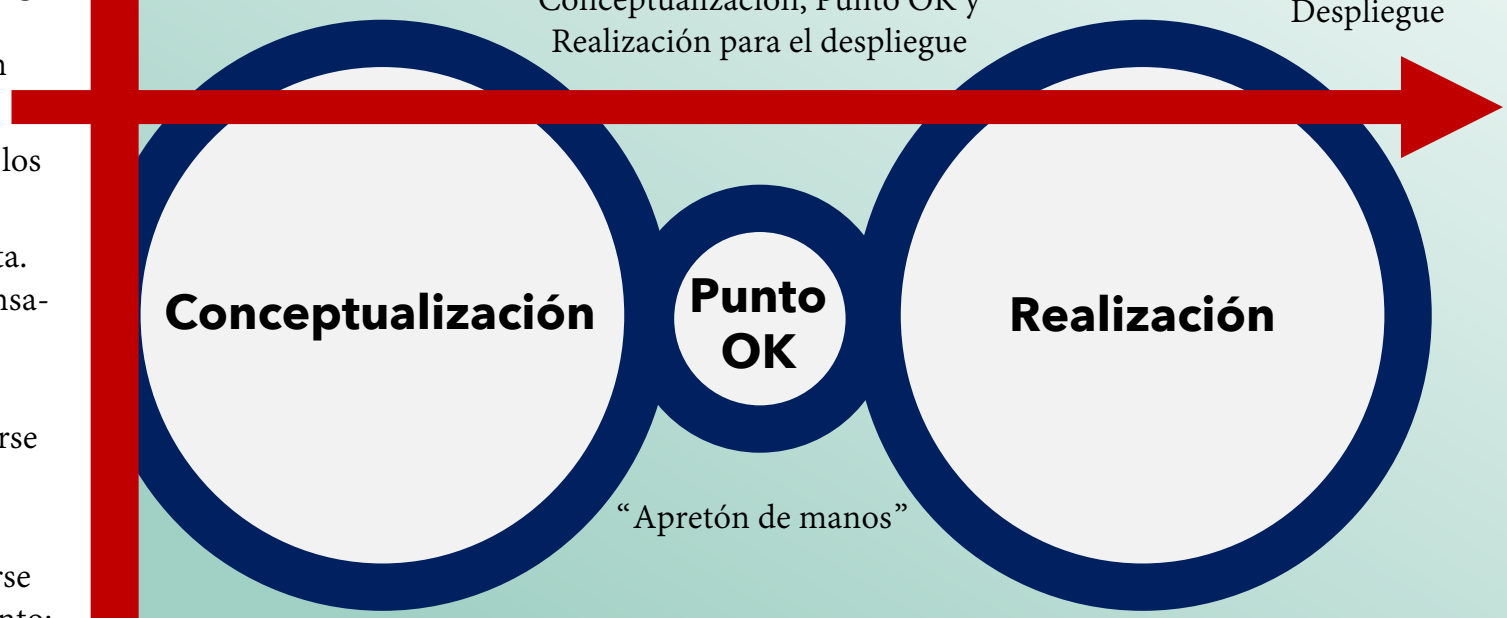
Pruebas y documentación son parte del proceso, no separadas (TDD)

Desarrollo de Software Orientado por el Tiempo: La manera de combinar estructura y autoorganización, flujo y disciplina

Lista de Items

- Se relaciona con un problema
- Contiene todos los ítems a trabajar
- El orden importa. Tiene un responsable (*dueño de la lista*)
- Puede actualizarse en cualquier momento
- Debe completarse en algún momento; luego se abre una nueva lista

Cualquier ítem (feature o proyecto) fluye una sola vez por las etapas de Conceptualización, Punto OK y Realización para el despliegue



Principio clave de TOSD para resolver problemas:

HEU: Hablen Entre Ustedes

La conversación puede darse en cualquier momento, no por escrito y preferiblemente cara a cara

En TOSD, el tiempo es fijo mientras la capacidad oscila

En el pensamiento de Mando y Control aplicado al desarrollo de software, se da por hecho que la capacidad es fija. Se tiende a creer que capacidad equivale a las «horas de trabajo de los recursos humanos» (léase, desarrolladores de software). Si la demanda o la dificultad de las tareas fluctúan, entonces el tiempo de entrega variará en consecuencia. Nociones defectuosas como esta han conducido a malentendidos numerosos sobre el desarrollo de software—por ejemplo, que el desarrollo de software es un arte; o, en el extremo opuesto, que debe ser dirigido y controlado desde arriba, restringido por presupuestos, *steering* y «personas al mando».

En TOSD, adoptamos la posición opuesta (y más realista). El tiempo de entrega debe ser fijo y la capacidad oscilar de forma natural (como de hecho ocurre siempre). Aquí se asume que el desarrollo de software es un oficio (no un arte) y que la autoorganización es natural en el desarrollo de software, como lo es en cualquier otro ámbito. La idea de que la capacidad oscila y el tiempo debe fijarse—para habilitar flujo, puntualidad, eficacia y un sentimiento humano de realización—fue articulada por primera vez por el austriaco Ernst Weichselbaum, pionero de los Sistemas de Trabajo Orientados al Tiempo. Su trabajo con empresas manufactureras se inspiró en el constructivismo y en el concepto de organizaciones fractales. Weichselbaum acuñó también el concepto de Punto OK, que aplicó sobre todo a la costura entre cliente y proveedor.



Amigos de TOSD: herramientas, conceptos y prácticas afines que combinan bien con la orientación al tiempo en software

Selección en orden alfabético

- Agentic AI
- Agentic IDEs
- Architecture as code
- Architecture Decision Records (ADR)
- Behavior-Driven Development (BDD)
- BetaCodex, descentralización
- Boy Scout Rule
- Cell Structure Design
- Chaos as code
- Chaos Engineering
- Clean Code (p. ej., YAGNI, KISS, DRY)
- Code Coverage
- Code Katas
- Collective Code Ownership
- Configuration as code
- Conway's Law
- Containerization
- Continuous Deployment (CD)
- Continuous Integration (CI)
- Defect Density / Escape Rate
- DevOps
- Distributed Version Control Systems (DVCS) — p. ej., Git
- Domain-Driven Design (DDD)
- End-to-End Testing
- Fail Fast
- Feature Branching
- Feature Flags
- GenAI
- Infrastructure as Code
- Inspect & Adapt
- Living Documentation
- Loose Coupling
- Mob Programming
- Modularization
- Mutation Testing
- Obeya Rooms
- Observability-Driven Development
- OpenSpace DEV
- Pair Programming
- Peer Reviews
- Pipeline as Code
- Relative Targets (Metas Relativas)
- Scaffolding
- Socio-Technical Systems
- SOLID — *Single Responsibility / Open-Closed / Liskov Substitution / Interface Segregation / Dependency Inversion*
- Static Code Analysis
- Story Boards
- Test-Driven Development (TDD)
- Vertical Slice Architecture
- Vibe Programming
- Workflow automation (p.ej., Jenkins, GitHub Actions)

El caso de Pradtke GmbH: cómo llegó TOSD a Pradtke

Desde mediados de los años 90, Pradtke GmbH (Bochum, Alemania) ha desarrollado y comercializado software para organizaciones como hospitales y servicios de bomberos. Tras años de resultados discretos, Pradtke emprendió con éxito una transformación a los principios BetaCodex mediante el enfoque OpenSpace Beta durante la segunda mitad de 2024. En noviembre de 2024, como parte de los 90 días del proceso de transformación OpenSpace Beta, la empresa puso en marcha su Diseño de Estructura Celular descentralizado con 8 células periféricas. La integración funcional permitió una simplificación sustancial de la administración y las operaciones. **La compañía se volvió visiblemente más ligera, alcanzó rentabilidad sostenida y mejoró con rapidez la satisfacción de clientes.** En el primer semestre de 2025, Pradtke simplificó y reformó su modelo de negocio, incorporando a cientos de clientes corporativos a un modelo nuevo. En palabras del propio equipo de Pradtke: entre el verano de 2024 y el verano de 2025, «pusimos en práctica como empresa todo aquello que antes habíamos intentado y que nos parecía imposible de lograr».

En el desarrollo de software, las cosas se veían menos halagüeñas. La empresa estaba en plena migración de su aplicación legada de escritorio hacia una Single Page Application. Eso implicaba cambiar de medio, de tecnología y de herramientas. Costaba generar impulso para el esfuerzo, con un desarrollo de software que seguía el patrón común de *Scrum-pero-no-tanto*…. Había un tablero JIRA, pero todas las personas involucradas sufrían los problemas de siempre. En con-junto, el des-pliegue seguía siendo demasiado lento, con estimaciones frágiles, sesiones de planificación ineficientes y rituales superfluos como retrospectivas y demos.

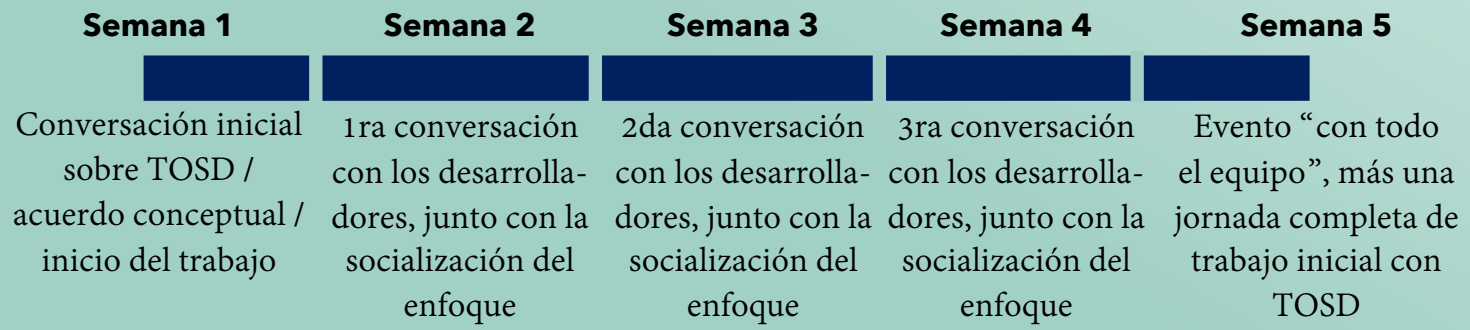
«Desde su fundación, Pradtke hizo un recorrido que resultará familiar a la mayoría de las firmas de software: acumulamos deuda técnica creciente y montones de código legacy del que nadie sabe bien qué hace; vivimos el síndrome del "no inventado aquí" mientras intentábamos equilibrar metas de arquitectura de largo plazo con necesidades de clientes de corto plazo. Íbamos malabareando nuestros "recursos" de desarrollo limitados con la esperanza de sacarles el máximo. En el camino, probamos cosas como waterfall, Scrum, JIRA, estimaciones y OKR—enfoques que nunca nos dieron lo que esperábamos. No nos volvíamos más rápidos y el software no mejoraba en gran cosa. Con perspectiva, fue porque no entendíamos realmente cuáles eran los problemas de fondo. Confiábamos en que podríamos "arreglar" los problemas si poníamos encima los procesos, marcos y modelos adecuados.»
Sebastian Kubsch

El caso de Pradtke GmbH: Implementación de la orientación al tiempo en solo 4 semanas

Dirigida por el CTO y junto con los gerentes generales Marc Otten y André Pradtke, la empresa comenzó su transición hacia TOSD en agosto de 2025. Mientras Pradtke implementaba TOSD, el enfoque se fue desarrollando aún más, yendo más allá de los conceptos iniciales como el Punto OK, la Lista de Ítems y el tiempo de realización fijo. La implementación se llevó a cabo en sincronía con los 14 desarrolladores de software de la empresa y 10 potenciales desarrolladores. Surgieron preguntas relevantes: **¿Cómo comenzar con TOSD involucrando a todos los desarrolladores al mismo tiempo?** ¿Cómo hacer transparente el trabajo en curso? ¿Cómo acelerar el trabajo sin que los desarrolladores tengan que trabajar más horas? ¿Cómo redistribuir las tareas operativas dentro de las células de negocio periféricas, para que los desarrolladores que forman parte de esos equipos puedan enfocarse lo suficiente en la Lista de Ítems compartida? ¿Cómo concebir el trabajo como ítems que puedan realizarse en uno a tres días? Tras un acuerdo conceptual en la primera semana, durante las tres semanas siguientes se realizaron tres rondas de conversaciones de 3 horas con los equipos de software. En la cuarta semana, se llevó a cabo el Go Live durante una primera sesión de trabajo TOSD para todos los desarrolladores. Toda la empresa fue invitada a la sesión de apertura conjunta de ese día.

«Con el nuevo diseño de estructura celular, la mayoría de los equipos encontró rápidamente satisfacción en la autoorganización y en la acción significativa y directa, libre de jerarquías artificiales y presupuestos. Las cosas avanzaban bien. Lo único que no vimos mejorar fue el proceso de desarrollo de software.»
Sebastian Kubsch

«Para una empresa de software como la nuestra, que existe desde los años 90, el desafío fue deshacerse de tres décadas de malos hábitos en solo cuatro semanas.»
Sebastian Kubsch



El caso de Pradtke GmbH: Cs y Rs – primeros pasos hacia el Desarrollo de Software Orientado por el Tiempo

El primer paso fue identificar y nominar a las personas desarrolladoras y especialistas autorizados para desarrollar conceptos o requisitos. La calidad del software y del flujo dependería en gran medida de la calidad de los conceptos que sostienen cada ítem. Ese grupo debía ser pequeño y cohesionado. Se revisó la Estructura Celular, identificando a quienes mejor podían convertirse en C (*conceptualizadores*, de *Konzeptionierer* en alemán). Además de la maestría, los criterios incluyeron una actitud abierta a la interacción y disponibilidad para conversar con toda persona. Internamente, desde entonces se les denomina Cs. El siguiente paso fue identificar a quienes pueden agregar valor como Realizadores. Este grupo resultó diverso en habilidades, maestría y madurez profesional. Algunas personas Cs también fueron identificadas como Rs (naturalmente, una persona C puede también ser R).

Con esto quedó listo el montaje inicial de TOSD: el CTO pasó a ser responsable de la Lista; Cs crean conceptos para las Rs; las Rs implementan esos conceptos dentro de time-boxes acordados. Los conceptos listos para el Punto OK deben ser, en principio, "sin más preguntas". El Punto OK exige acuerdo y apretón de manos entre Cs y Rs. Y algo crucial: **la conversación/consulta entre todas las personas puede darse en cualquier momento.**

En los días siguientes se organizó un taller de 3 horas con las Cs; luego, dos sesiones de 3 horas con todas las Rs. En cada sesión se expuso TOSD de punta a punta, partiendo de la necesidad del negocio. Nada se impuso: se discutió de forma genuina. El feedback fue excelente y permitió aclaraciones constructivas. Algunas ideas surgidas en las conversaciones se integraron al vuelo en el concepto. La facilitación externa ayudó a que los talleres fluyeran. Se acordó que la lista de migración, con varios cien-tos de ítems, podía despejarse en ~8 semanas. Todo el mundo quedó a bordo y con ganas de empezar.

«En agosto de 2025, Niels Pflaeging nos propuso una alternativa: la orientación al tiempo. Sonaba simple. Deshacerse de toda la tontería basada en cadencias. Abandonar los sprints. Eliminar roles innecesarios, como Product Owner y ScrumMaster. Y, sobre todo: dejar de intentar crear flujo alterando el tiempo de entrega. En su lugar, fijar el tiempo de entrega y dejar que la capacidad oscile. En el momento en que Niels nos lo planteó a mí y a uno de los gerentes generales, me enganché. Enganchado y preguntándome: ¿podría ser tan simple llegar al flujo?»
Sebastian Kubsch

«En TOSD, no hay juego de estimaciones. Nadie juega al póker de estimación. El ítem debe ser tan preciso que SEPA si tardará 1, 2 o 3 días en realizarse. Para las R, cumplir esa ventana de tiempo (ni un día más tarde) es una cuestión de honor.»
Sebastian Kubsch

El caso de Pradtke GmbH: Mejorando la efectividad a través de la orientación al tiempo, semana a semana

La primera semana trajo dificultades previstas: algunos conceptos todavía no eran lo bastante precisos, por lo que surgieron preguntas en la Realización; algunas R subestimaron el tiempo necesario; las C producían conceptos más rápido de lo que se realizaban, acumulándose en el Punto OK. En conjunto, los problemas se resolvieron fácilmente con mejores conversaciones y más frecuentes: las C ajustaron el grado de detalle en los conceptos; las R comenzaron a trabajar en parejas en tareas concretas y se guiaron por ítems ya listos para realizar; para lograr flujo más estable, se repriorizó la Lista; algunas personas fueron redesignadas como C o R, y se enfocó en el lado R.

La segunda semana trajo mejoras visibles: el rendimiento (throughput) se duplicó respecto de la anterior. La gente se acostumbró a la nueva forma de trabajar y empezaron a asentarse patrones. El impulso crecía.

En la tercera semana se produjo un punto de inflexión. Las R comenzaron a ver que TOSD funcionaba. Se hizo patente el buen progreso en entregar ítems a clientes. **Para la semana 4, se habían realizado casi tantos ítems como en los seis meses previos.** El flujo comenzó a emerger, aunque quedaron desafíos: la proporción de Conceptualización vs. Realización exigía atención constante; la capacidad todavía no oscilaba mucho ni en Conceptualización ni en Realización; algunos conceptos no se sostuvieron por falta de práctica y estándares insuficientes en Conceptualización. En respuesta, las C acordaron ciertos estándares. **Claves de TOSD: los problemas se resuelven en el acto;** los baches se atienden de inmediato; la inmediatez permite enfocarse en lo importante. En vez de reuniones ritualizadas de planificación por horas o días, TOSD obliga a la acción inmediata.

«Las C eligen ítems de la Lista, enfocandose en los de mayor prioridad e idealmente en los que caen en su dominio de conocimiento. Las C hacen el trabajo intelectual y de campo necesario para que las R tengan claro el QUÉ hay que hacer, sin decirles el CÓMO. Deben describirse interdependencias, comportamientos inesperados del sistema y otras particularidades. A veces hay que hurgar en el modelo de base de datos o en el código legado. El objetivo es claro: permitir que las R implementen el ítem lo más rápido posible, reuniendo de antemano la información necesaria. Las C deben identificar y acercarse a las R adecuadas. Las R pueden pedir ítems a las C, y se espera que lo hagan, especialmente cuando no tienen nada que hacer.»
Sebastian Kubsch

«Por ahora usamos un GitHub Project simple para seguir los ítems. No lo sobredimensionen, manténganlo simple. No construyan el próximo 'backlog'.»
Sebastian Kubsch

El caso de Pradtke GmbH: Reflexiones sobre dominar la orientación al tiempo

Si se quiere mayor efectividad en empresas de software y grupos de desarrollo, conviene preguntarse: ¿cuál es el mayor desperdicio en desarrollo de software? Con frecuencia es una combinación de vacilación, parálisis y sensación de soledad o *indefensión aprendida* entre quienes desarrollan. **Muchos grupos se inclinan a trabajar en modo "arranca-y-frena" y con inactividad parcial.** No porque lo deseen, sino porque los contextos creados por métodos de gestión comunes hacen inevitable esa modalidad: reuniones, rutinas, rituales, coacción, necesidad de aprobaciones jerárquicas y organización en silos inhiben la autoorganización. El flujo se interrumpe y la realización personal se reduce, limitando severamente el output. Esto fue así durante años en Pradtke. La conclusión: ese desperdicio y la falta de colaboración solo se eliminan rompiendo los patrones de gestión comunes (incluidos los de Agile) y superando el sistema convencional de desarrollo. Descentralización + TOSD proveen las estructuras para interrumpir patrones limitantes y derrochadores.

Luego está la cuestión de la capacidad: ¿cómo elevarla? ¿cómo lograr que oscile con la demanda? La respuesta apresurada suele ser contratar a corto plazo. Pero ese sesgo suele ser un error: a menudo ni es factible ni necesario contratar. Conviene considerar que la capacidad oscila en ambos sentidos: habrá momentos con capacidad alta y otros con capacidad baja. *El palanco principal para elevar capacidad no es contratar, sino colaborar mucho mejor,* liberar a las personas de aprobaciones, tiempos de espera, reuniones inútiles y atajos innecesarios; naturalizar la ayuda mutua, la consulta y el pair programming. Todo esto se aplicó en Pradtke. También, con justicia: trabajar un poco más en ciertos momentos de pico de demanda ayuda a lograr flujo. Si el trabajo se percibe como realizador y no coercitivo, dentro de un sistema que permite flujo, la capacidad puede oscilar.

«Comenzar deliberadamente con ítems más fáciles y de menor complejidad, para que todas las personas se concentren en aprender el método en lugar de lidiar con el contenido al principio. Esto ayuda a tomar velocidad.»
Sebastian Kubsch

«Hacia la semana tres, un desarrollador veterano se me acercó. Me dijo cuán contento estaba con el modelo nuevo: que por fin las cosas habían empezado a moverse otra vez; que estaba aliviado porque por fin trabajábamos en ítems de los que hablábamos hacía años, pero que no abordábamos. Me mostró una feature que acababa de finalizar y de la que estaba visiblemente orgulloso. Supe entonces que estábamos en el camino correcto.»
Sebastian Kubsch

El caso de Pradtke GmbH: Reflexiones sobre dominar la orientación al tiempo (continuación)

Un tercer aspecto evidente en la orientación al tiempo es la interacción de calidad entre quienes desarrollan. La falta de interacción de calidad reduce el output y daña la calidad del software. Facilitar/animar la interacción es una cosa; además, la buena comunicación se puede favorecer estructuralmente. TOSD provoca interacción constante de forma estructural: en torno a la Lista de Ítems (sus elementos y su secuencia); alrededor de las constelaciones/parejas en Conceptualización y Realización; durante la Conceptualización, en el Punto OK y durante la Realización. Se pueden realizar múltiples intervenciones para propiciar mejores conversaciones. En Pradtke, en las primeras semanas, se incrementaron los encuentros cara a cara con medidas como: almuerzo en la oficina todos los días; acuerdo para trasladar tiempo de trabajo desde casa a oficina; e instaurar sesiones voluntarias de trabajo y aprendizaje los sábados.

En este contexto, el principio clave de TOSD «Hablen entre ustedes» merece men-ción especial. Conversar de inmediato cuando aparece un problema es mejor que definir reglas, montar comités o reuniones periódicas, o instituir roles de steering y supervisión. Apostar por la conversación abierta e inmediata es un músculo que todas las personas deben entrenar, especialmente cuando, como en Pradtke, ese músculo no se había usado bien en años. La idea del principio es hacer inaceptable posponer la conversación hasta alguna reunión, y convertir en norma resolver en el acto los problemas/impedimentos. No todo se resuelve ad hoc, desde luego: lo que no se pueda resolver en el momento puede transformarse en nuevos ítems para la Lista actual o la siguiente. En desarrollo de software, «Hablen entre ustedes» es inseparable de la orientación al tiempo y una palanca potente para desplegar flujo y nuevos patrones de trabajo.

«En TOSD, posponer una conversación sobre un problema que acaba de surgir sería frívolo y absurdo.»
Sebastian Kubsch

«La autorización es un aspecto clave de la orientación al tiempo. La gente estaba acostumbrada a esperar legitimación antes de actuar. Eso hacía las cosas lentas. Dejamos muy claro desde el principio con TOSD que todas las personas estaban autorizadas a actuar, porque confiamos en ellas, porque sabíamos que harían las cosas bien. Y entonces lo hicieron. Porque quisieron, no por algún tipo de mandato.»
Sebastian Kubsch

«Elijan a la gente adecuada. Una selección descuidada de C dará malos resultados. Evalúen si las personas elegidas como C son las adecuadas para el trabajo a realizar. Es natural que los requisitos cambien con el tiempo.»
Sebastian Kubsch

Fortalezas del Desarrollo de Software Orientado al Tiempo (resumen)

1. **Puede iniciarse en cualquier momento.** La puesta en marcha requiere pocas semanas y pocos recursos externos; puede combinarse con OpenSpace Beta para transformar organizaciones completas (90 días).
2. **Es extremadamente directo.** No exige formaciones específicas para empezar: basta acordar proceso y principios. Elimina de inmediato sobrecarga, steering, reuniones y control conductual excesivos.
3. **Produce enseguida flujo y orientación al tiempo,** que habilitan autoorganización y colaboración—palancas de efectividad y desempeño.
4. **Genera foco y disciplina, sin jerarquía ni burocracia,** apoyándose en la Lista (y en el Responsable de la Lista y el Protector del Flujo), en el Punto OK, y en la ausencia de reglas, roles de steering y resquicios para la injerencia directiva.
5. **Acelera tanto la Conceptualización como la Realización** distinguiendo las dos maestrías que importan en el desarrollo de software efectivo.
6. **Escala de forma natural** a cientos o miles de personas; no depende de «equipos de desarrollo». Permite coexistencia de varias Listas de Ítems («iniciativas»).
7. **Incrementa drásticamente la calidad de las interacciones** gracias a la inmediatez de la conversación.
8. **Reintegra la conversación al proceso de desarrollo –** en vez de confinar el discurso a rituales o reuniones aparte. La conversación vuelve a ser parte del trabajo.
9. **Se combina fácilmente con más socio-tecnología para agilidad y auto-organización,** p. ej., formatos de interacción en grupos grandes (OpenSpace DEV), conceptos de estructuras descentralizadas (Cell Structure Design) y sistemas de desempeño relativo (Relative Targets).

«El Punto OK es el "big bang" diario del procesamiento de ítems. Como el traspaso es altamente ritualizado, el Punto OK es algo así como el latido de la empresa.»
Ernst Weichselbaum

«Dejar fluir las cosas es mejor que agruparlas.»
Ernst Weichselbaum

«Diseñe el proceso de manera que no necesite ser monitoreado. Es mejor ser puntual que administrar retrasos.»
Ernst Weichselbaum

«La autoridad no es propiedad de individuos, sino que se genera en la interacción entre al menos dos personas. La autoridad proviene del acuerdo.»
Ernst Weichselbaum

Desarrollo de Software Orientado al Tiempo: Acerca de la licencia de fuente abierta de tipo Attribution ShareAlike, 'CC-BY-SA'

Esta licencia te permite usar, comercializar, remezclar, adaptar y desarrollar a partir de *Time-Oriented Software Development*. Al hacerlo, aceptas cumplir con tres condiciones simples:

- Dar crédito a los autores originales, Niels Pflaeging y Sebastian Kubsch,
- Incluir el enlace especificado al material fuente, detallado más abajo,
- Licenciar tus creaciones derivadas bajo los mismos términos.

En concreto, deberás incluir y mostrar de forma destacada la siguiente frase y enlace en todos los trabajos originales o derivados, así como en cualquier texto o material gráfico relacionado que desarrolles:

> "Este trabajo se deriva de *Time-Oriented Software Development,* una tecnología social libre y de código abierto de Niels Pflaeging y Sebastian Kubsch, publicada bajo la licencia CC-BY-SA-4.0 de Creative Commons, disponible en: www.redforty2.com/tosd".

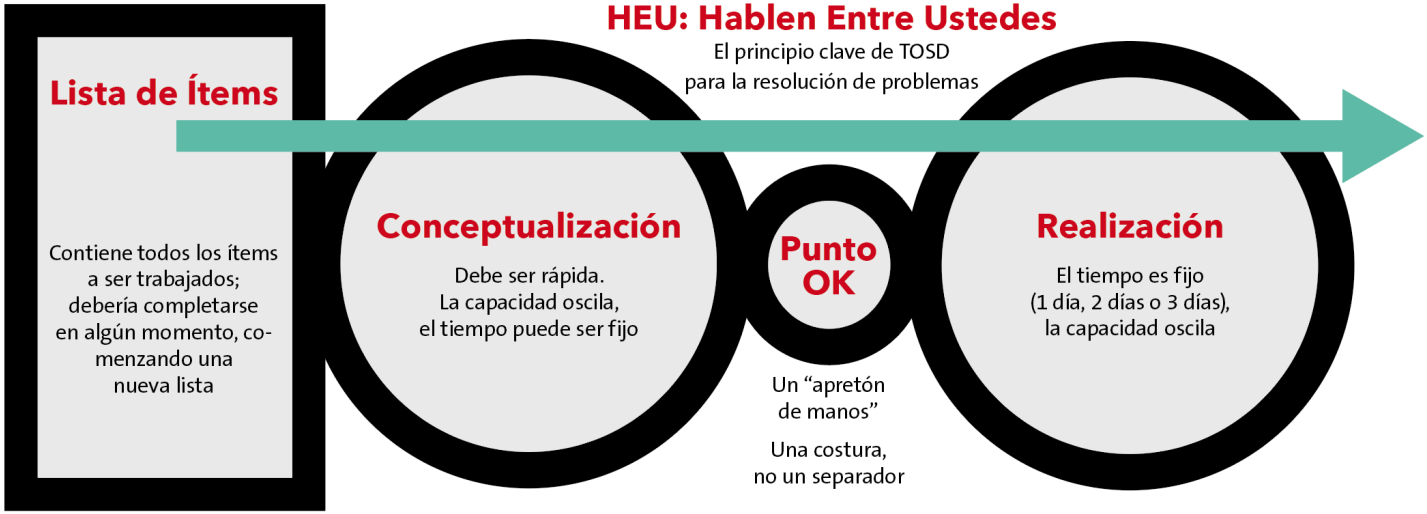


Te invitamos también a considerar las demás tecnologías sociales de código abierto creadas por el movimiento BetaCodex desde 2008: BetaCodex, Cell Structure Design, Relative Targets, OpenSpace Beta y OpenSpace DEV.

Desarrollo de Software Orientado al Tiempo (TOSD)

Concepto general | Una tecnología social de código abierto de Niels Pflaeging y Sebastian Kubsch

Resumen visual



7 leyes, o principios clave del Desarrollo de Software Orientado por el Tiempo (TOSD)

01. **Orientación al tiempo** – sin orientación al volumen/la capacidad
02. **Flujo y Punto OK –** sin experimentos superficiales, control centralizado y organización por lotes
03. **Transparencia y Lista de Ítems –** sin control jerárquico, sin mentalidad presupuestaria, sin objetivos fijos
04. **Autoorganización coherente –** sin escalar, ni programación y coordinación centralizadas
05. **El tiempo es fijo, la capacidad varía –** no al revés
06. **HEU: Hablen Entre Ustedes (Talk To Each Other) –** no uniformidad, reglas ni órdenes
07. **Coherencia gerencial –** sin burocracia, inconsistencia y falta de coherencia

Las 12 leyes del BetaCodex

The BetaCodex® | versión 2018 | www.betacodex.org

Ley		¡Haz esto! (Beta)	¡No esto! (Alfa)
01.	Autonomía del equipo	conexión con el propósito,	no con la dependencia
02.	Federalización	integración en células,	no división en silos
03.	Liderazgos	autoorganización,	no gestión
04.	Éxito rotundo	aptitud integral,	no mono-maximización
05.	Transparencia	fluye la inteligencia,	no obstrucción del poder
06.	Orientación al mercado	objetivos relativos,	no prescripción fija, de arriba abajo
07.	Ingreso condicional	participación,	no incentivos
08.	Claridad mental, presencia	preparación,	no economía planificada
09.	Ritmo	cadencia y fluidez,	no orientación al año fiscal
10.	Decisión basada en la maestría	consecuencia,	no burocracia
11.	Disciplina de recursos	conveniencia,	no orientación al estatus
12.	Coordinación de flujos	dinámicas de creación de valor,	no asignaciones estáticas

El Desarrollo de Software Orientado al Tiempo (TOSD) tiene tres componentes: Los 12 principios ("leyes") del BetaCodex, 7 principios ("leyes") de TOSD, y el Resumen visual | Desarrollo de Software Orientado al Tiempo (TOSD) de Niels Pflaeging & Sebastian Kubsch | versión 2025 | publicada bajo la licencia CC-BY-SA-4.0 | www.redforty2.com/tosd

Fuentes y lecturas recomendadas

Tiempo e organización

Bluedorn, Allen C. (2001) *The human organization of time – Temporal realities and experience*

Stalk, George (1990): *Competing against time – How time-based competition is reshaping global markets*

Schmenner, Roger W. (2012): *Getting and staying productive – Applying swift, even flow to practice*

Sistemas de Trabajo Orientados al Tiempo y descentralización

Pflaeging, Niels (2020): *Essays on Beta, Vol. 1 – What's now & next in organizational leadership, transformation and learning*

Pflaeging, Niels (2025): *Slave to the Rhythm – Adopt Time-Oriented Work Systems. Now*

Pflaeging, Niels (2025): *Relative Targets: Patterns in Practice*

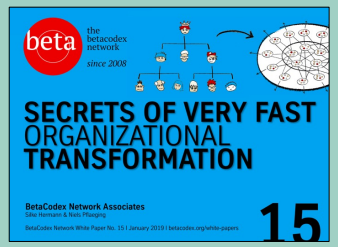
Pflaeging, Niels/Hermann, Silke (2023): *The laws of the BetaCodex*

Pflaeging, Niels/Hermann, Silke (2023): *Cell Structure Design Patterns, I*

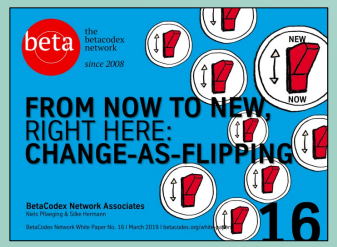
Pflaeging, Niels/Hermann, Silke (2024): *Cell Structure Design Patterns, II*

Weichselbaum, Ernst/Pflaeging, Niels (ed.) (2020): *In jedem Unternehmen steckt ein besseres* (solo en Alemán)

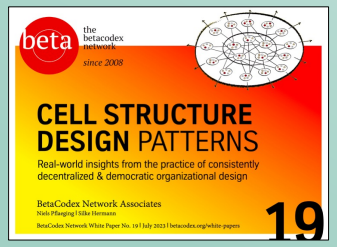
Papers de investigación de la BetaCodex Network, disponibles en www.betacodex.org



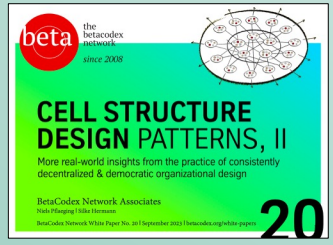
Secrets of Very Fast Organizational Transf., 2019



From Now to New, Right Here, 2019



Cell Structure Design Patterns, I, 2023



Cell Structure Design Patterns, II, 2024



Slave to the Rhythm, 2025

Papers de investigación de la BetaCodex Network

Los puede leer gratis en betacodex.org/white-papers



Libros relacionados de BetaCodex Press

Más de 160 citas directas del pionero intrépido de la orientación al tiempo, el arquitecto del Sistema de Producción de Toyota, Taiichi Ohno (1912–1990)



What would Ohno-san do?
Editado por Niels Pflaeging/ Stefan Schmeing
BetaCodex Press, 2025
También disponible como E-book

Sabiduría condensada del filósofo-practicante original de la orientación al tiempo: W. Edwards Deming (1900–1993)



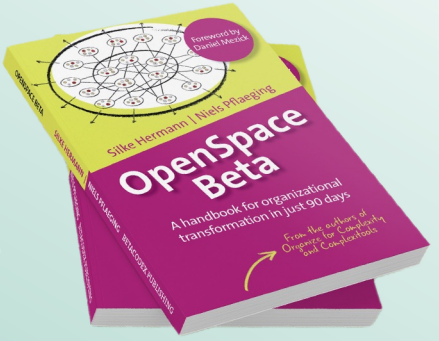
What would Deming do?
Editado por Niels Pflaeging
BetaCodex Press, 2023
También disponible como E-book

Conceptos y patrones de autoorganización coherente y orientación al tiempo, presentados por el fundador de BetaCodex



Essays on Beta, Vol. 1
Niels Pflaeging
BetaCodex Press, 2020
También disponible como E-book
También disponible en Alemán

¿Cómo lograr un cambio transformador de manera oportuna? Silke Hermann y Niels Pflaeging ofrecen ideas y conceptos específicos



OpenSpace Beta
Silke Hermann/Niels Pflaeging
3rd ed., BetaCodex Press, 2023
También disponible como E-book
También disponible en Alemán y Coreano

Visite la tienda en redforty2.com/shop

the betacodex network

since 2008

Ponete en contacto con los autores de este paper:
Niels Pflaeging, niels.pflaeging@redforty2.com
Sebastian Kubsch, sebastian.kubsch@pradtke.de
Concepto y diseño del paper: Niels Pflaeging

Ponete en contacto con los traductores del paper:
Ernesto R. Corona, ernesto@d-vops.com
y el equipo de d-vops.com

Sitio web de TOSD: timeoriented.dev


Niels Pflaeging


Sebastian Kubsch


Ernesto R. Corona

Aviso legal: No se utilizó inteligencia artificial en el proceso de redacción de este paper, salvo para investigación bibliográfica y fines de traducción.

Este trabajo se deriva de BetaCodex, una tecnología social libre y de código abierto del BetaCodex Network, publicada bajo la licencia CC-BY-SA de Creative Commons, disponible en: betacodex.org/licence. **Este trabajo también se deriva de Time-Oriented Software Development (TOSD),** una tecnología social libre y de código abierto desarrollada por Niels Pflaeging y Sebastian Kubsch, publicada bajo la licencia CC-BY-SA de Creative Commons, disponible en: redforty2.com/tosd

Estás en libertad de utilizar este trabajo, o partes del mismo, siempre que cites y/o menciones la fuente de forma coherente y adecuada. Todos los papers de investigación de BetaCodex están disponibles en: betacodex.org/white-papers



Aprenda más acerca del
Desarrollo de Software Orientado al Tiempo
en timeoriented.dev