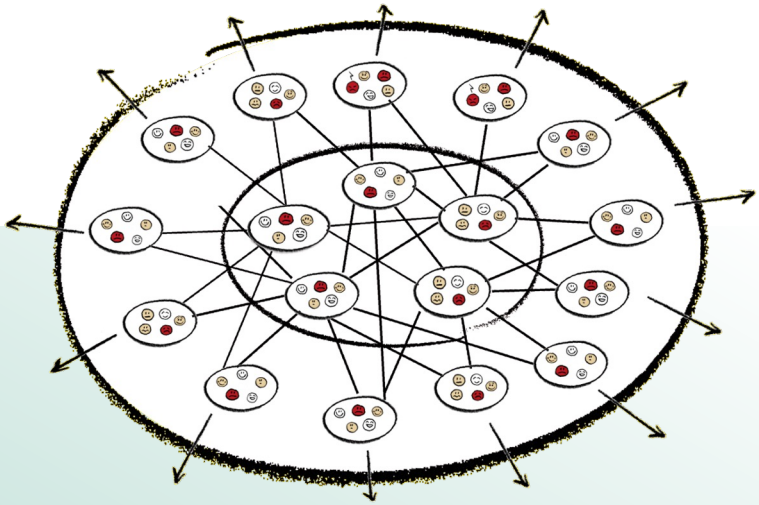
the
betacodex
network

since 2008



EINFÜHRUNG IN DIE **ZEITORIENTIERTE SOFTWAREENTWICKLUNG**

Flow, Rhythmus und Pünktlichkeit als Grundlage agiler Softwareentwicklung – jenseits von Scrum, Kanban und Lean

BetaCodex Network Associates
Niels Pflaeging | Sebastian Kubsch. Übersetzung: Peter Pröll

BetaCodex Network-Forschungspapier Nr. 26 | Oktober 2025 | betacodex.org/white-papers

26

Widmung

Wir widmen dieses BetaCodex Network-Forschungspapier dem Andenken an

Ernst Weichselbaum

(13.04.1944 – 27.12.2024),

Philosoph der Zeitorientierung
und Pionier zeitorientierter Arbeitssysteme

Zeitorientierte Softwareentwicklung: So lässt sich die ultimative Struktur in Arbeit und Organisationen nutzen

Zeit – sie ist in unserem Leben so allgegenwärtig, dass wir sie kaum bemerken. Zeit ist eine mächtige Struktur. Doch in der Softwareentwicklung – und in Organisationen generell – wurde das Konzept der *Strukturierung durch Zeit* bislang kaum wahrgenommen und so gut wie nie konstruktiv genutzt. Vorstellungen wie *Iteration* und *Time-Boxing* in *Scrum* und anderen Ansätzen der *Agilität* gingen längst nicht weit genug, um die Macht der Zeit wirklich für die Softwareentwicklung nutzbar zu machen. Auch heute noch ist die Organisation von Softwareentwicklung in Teams und Unternehmen geprägt von Mythen, Intransparenz, Hierarchie, Symptombekämpfung, Zwang und Gefolgschaft. Sie ist gefangen in verschwenderischen Ritualen, dysfunktionalen Rollen, bürokratischen Prozessen.

„Agile" hat an dieser Misere leider wenig geändert. Seit 2001, als das *Manifest für Agile Softwareentwicklung* verfasst wurde, haben sich die zentralen Debatten innerhalb der Bewegung im Wesentlichen um Missverständnisse gedreht: Schätzen oder nicht schätzen, wie führt man *Retrospektiven* durch, welche Rollen sollten *Product Owner, Scrum Master* oder *Agile Coaches* einnehmen, wie plant oder skaliert man, wie schreibt und bewertet man *User Stories* usw. Wir glauben: Diese und viele andere Diskussionen waren von Beginn an fehlgeleitet und mussten ins Leere laufen. Die Gewöhnung an falsche Debatten führte schnell dazu, dass der Agile Bewegung Schwung und Veränderungskraft ausgingen.

Heute ist klar: „Agile" hat kaum Fortschritte dahingehend gebracht, wie Softwareentwicklung organisiert wird. In diesem Papier vertreten wir die These, dass *konsequente Zeitorientierung* der wirksamste Weg ist, über *Wasserfall-*, *„pseudo-agil"-* und *Laissez-faire-*Ansätze der Entwicklung hinauszukommen. Denn kollektive Disziplin, Fokus, Flow und Freude am Gelingen entstehen in der Softwareentwicklung nicht durch *Skalierung* oder Steuerung, durch zahnlose Rollen oder postulierte Werte, sondern durch Strukturen in der Zeit. **Zeit ist die ultimative Struktur – wir müssen ihre Nutzung nur bewusst gestalten, wenn wir die Organisation der Softwareentwicklung wirklich voranbringen wollen.**

„Vom Standpunkt des Alten ist das Neue immer falsch. Im alten Denkrahmen ist nichts wirklich Neues möglich."
Ernst Weichselbaum

„Wenn du die Welt verändern willst, beginne bei den Mustern in deinem Kopf. Denn da steht dir niemand im Weg."
Ernst Weichselbaum

„Lieber Pünktlichkeit gestalten als Verspätung verwalten."
Ernst Weichselbaum

„Wer entscheiden muss, weiß nicht, was er wirklich will. Anfangen besteht ausschließlich aus Anfangen."
Ernst Weichselbaum

Zeitorientierung: Unsere grundlegendsten menschlichen Bedürfnisse hängen mit der Zeit zusammen

Eines der zentralen Ziele sogenannter Zeitorientierter Arbeitssysteme – deren Grundlagen wir in einem separaten BetaCodex Network Research Paper erstmals eingehender vorgestellt haben – besteht darin, Menschen bei der Arbeit ein Gefühl der Selbstwirksamkeit zurückzugeben. Wir sprechen von jener Empfindung, die eine Landwirtin am Ende eines jeden Tages verspüren kann, wenn sie sich sagen kann: *Das Tagwerk ist vollbracht.* Wir sprechen von einem tief empfundenen *Gefühl der Befriedigung* in Bezug auf die tatsächlich geleistete Arbeit, das mit einem grundlegenden menschlichen Bedürfnis gekoppelt ist.

Ein täglich (oder häufig) sich einstellendes Gefühl von Zufriedenheit – das ist etwas völlig anderes als der abstrakte Ruf nach Purpose, Zielen oder Nordsternen, wie sie in heutigen Weisungs-und-Kontroll-Organisationen verbreitet sind. Ein Gefühl der Erfülltheit *ist in der Arbeit selbst* verankert – durch die individuelle, persönliche Erkenntnis, etwas zum Abschluss gebracht zu haben.

Anders gesagt: zentral und hierarchisch gesteuerte Wissensarbeit, Wasserfall-Organisation oder Laissez-faire-Softwareentwicklung berauben Wissensarbeiter:innen ihrer Sehnsucht danach und ihrem Recht darauf, ein regelmäßiges Empfinden von Bedeutsamkeit und Abschluss zu erleben. Die Folgen eines solchen Entzugs und der systematischen Entkopplung vom Erleben von Selbstwirksamkeit bei der Arbeit sind dramatisch. *Zeitorientierung* ist der einzige Weg, das Erleben von Flow in der Wissensarbeit, die Einheit zwischen der eigenen Arbeit und dem Nutzen, den sie für andere Menschen stiftet, wiederherzustellen. **Dieses Gefühl der Selbstwirksamkeit steht im Zentrum der Zeitorientierten Softwareentwicklung – englisch: *Time Oriented Software Development, TOSD*.**

„‚Das Tagwerk ist vollbracht.' Ein uralter Ausspruch der ländlichen Bevölkerung. Ein Tagwerk hat einen Anfang und ein Ende, dazwischen liegt der Tag. Es hat einen hohen emotionalen Wert, fertig zu sein: ‚Für heute habe ich meinen Teil geleistet.' Das vollbrachte Tagwerk ermöglicht den Betroffenen das Erfolgserlebnis. Das hat jemand am Fließband nicht. Durch Serienfertigung und Automatisierung wird dieses Erlebnis verhindert. Das sozio-technische System geht kaputt."
Ernst Weichselbaum

„Durch Einhaltung der Nahtstellenvereinbarung erleben Menschen einen emotionalen Mehrwert, der durch Steuerung unerreichbar wäre."
Ernst Weichselbaum

„Wenn die Arbeit weniger eintönig ist, erhöhen wir die Summe des Lächelns."
Ernst Weichselbaum

Macht Zeit und Flow zum Chef in der Softwareentwicklung – statt Chefs den Flow behindern zu lassen

Die ursprüngliche Absicht der frühen Agile-Softwareentwicklungsszene bestand darin, Weisungs- und Kontroll-Management in der Entwicklung zu überwinden – das damals in Ermangelung einer besseren Bezeichnung als „Wasserfall" bezeichnet wurde. Heute wissen wir, dass die Versuche der Agilisten, Wasserfall-Muster zu brechen, scheiterten. Stattdessen wurden Weisungs- und Kontroll-Muster in Softwareentwicklungs- und Tech-Unternehmen durch „agile" Konzepte, Werkzeuge und Skalierungsansätze sogar bestätigt und vertieft. Das Problem dürfte darin liegen, dass *Agile*-Ansätze wie *Scrum, Kanban* oder *XP* nicht ausreichen, um Weisungs-und-Kontroll-Muster aufzubrechen. Die in agilen Konzepten vorgeschlagenen Rollen, Rituale und Muster wurden vielmehr vereinnahmt, verbogen oder umgeleitet, um hierarchischen Steuerungszwecken zu dienen. Overhead und unproduktive Aktivitäten in der Softwareentwicklung haben seit den 1990er-Jahren eindeutig zugenommen. Ebenso die Barrieren für Flow, trotz anderslautender Behauptungen.

Zeitorientierte Softwareentwicklung muss eindeutig *anders* funktionieren, wenn es einen Unterschied machen und die Versprechen der Agilität einlösen soll. **Entsprechend bricht TOSD mit den konventionellen Mustern von Wasserfall und Agile:** etwa mit gängigen Vorstellungen von *Entwicklungsteams* oder *Skalierung*. Oder mit der Idee von *Sprints* – die aus der Perspektive echter Zeitorientierung sperrig und unpraktisch wirken. Was TOSD weniger anfällig für zwanghaftes Management macht, sind seine *Prinzipien*: Wie der BetaCodex, von dem es abgeleitet ist, basiert TOSD auf sinnvollen Prinzipien („sieben Gesetzen"), die sowohl beschreiben, worauf Zeitorientierung abzielt, als auch *wogegen* sie steht – nämlich die Muster und Gestaltungsprinzipien von Weisungs- und Kontroll-Systemen. Auf der folgenden Seite sind die sieben TOSD-Prinzipien dargestellt.
In kompakterer Form finden sie sich auch in der Konzeptübersicht auf Seite 23.

„Zeitorientierung verbindet – sie trennt nicht. Sie ist eine Brücke, keine Mauer. Zeitorientierung ist Voraussetzung für Vernetzung und Selbstorganisation."
Ernst Weichselbaum

„Wer steuern muss, ist selber schuld. Die Firma einstellen, nicht steuern!"
Ernst Weichselbaum

„Eine Meinung über das Kommende zu haben ist wertvoll. Steuerung nicht."
Ernst Weichselbaum

„Wo Ausnahmen regieren, kann kein robustes System entstehen."
Ernst Weichselbaum

„Durchsetzen ist eine Kriegserklärung: Es gelingt nur auf Kosten anderer."
Ernst Weichselbaum

In der Softwareentwicklung ist Zeitorientierung Voraussetzung für konsistente Selbstorganisation & Zusammenarbeit



Mengenbezogenes Denken und Sprache:
Skalierung, Backlog, Story Points, Durchsatz, Forecast, Schätzung, Kapazitätsauslastung, Auslastungsgrad, Charge, optimale Produktionsgröße, Skaleneffekte, Mergers & Acquisitions, Marktanteil, Mindestbestand,, Zielerfüllung, Planung, PI Planning, Big-Room-Planning, Quarterly Planning

Zeitbezogenes Denken und Sprache:
Tagesportion, OK-Punkt, (Einhaltung von) Time Slots, feste Realisierungszeit, optimale Zeit, Flow, Rhythmus, Takt, Groove, Passung, „Das Tageswerk ist vollbracht", White Space, Geschwindigkeit, Wunschtermin, antizipative Koordination, Pünktlichkeit, vereinbarte Übergaben, Handschlagsqualität, Nahtstellen-Vereinbarungen zwischen Akteur:innen/Teams

Die sieben zentralen Prinzipien oder „Gesetze“ der Zeitorientierten Softwareentwicklung

1. **Zeitorientierung.** Anders als die Kapazitäts- oder Mengenorientierung ist Zeitorientierung etwas Verbindendes: eine Brücke. Sie ermöglicht und stärkt Selbstorganisation.
2. **Flow mit OK-Punkt.** Flow ist ein zentrales Phänomen des Ausdrucks von Zeit und Bewegung in Arbeit und Wertschöpfung. Flow erfordert, dass Barrieren zwischen der Arbeit und der Nutzung ihres Ergebnisses (Software, Features) durch Kund:innen beseitigt werden. Bedingungen für Flow zu schaffen hat den Effekt, dass Probleme im Arbeitssystem sichtbar werden – weil durch Restriktionen Verhältnisse entstehen, die Probleme offenlegen.
3. **Transparenz und gemeinsame Item-Liste.** Schaffen zusammen Voraussetzungen für Selbstverantwortung und Selbststeuerung. Beziehen sich stets auf Teams oder Gruppen von Menschen – nicht auf Einzelpersonen.
4. **Konsistente Selbstorganisation.** Ist der einzige Weg, um hohes Engagement und hohe Motivation bei der Arbeit freizusetzen und Voraussetzung für Höchstleistung. Erfordert Dezentralisierung und verträgt sich nicht mit Steuerung und Zentralisierung.
5. **Zeit ist fix, Kapazität schwingt.** Nicht umgekehrt.
6. **MER: MitEinander Reden.** Notwendige Gespräche genau hier und jetzt zu führen – das ist das Herz von Selbstorganisation und Flow in sozialen Systemen. Dialog, Diskurs und Vereinbarung müssen im Zentrum zeitgemäßer Softwareentwicklung stehen. Echte Gespräche machen Regeln, Steuerung und Zentralisierung überflüssig.
7. **Kohärenz im Managementhandeln.** Ohne sie funktioniert nichts. Manager müssen die Bedingungen dafür schaffen, das TOSD-System einzurichten und dauerhaft wirksam zu halten.

„In der Zeitorientierung wird Kapazität nicht verplant (‚flexibel‘), sondern als System gestaltet.“
Ernst Weichselbaum

„Fließen lassen ist besser als zusammenfassen. Ein [großes Item] ist nichts Anderes als fünf kleinere mit zufällig der gleichen Lieferadresse. “
Ernst Weichselbaum

„Arbeitszeit ist dann, wenn Arbeit ist. Dank Tagesportion wird jeder Tag zu einer Saison.“
Ernst Weichselbaum

„Wenn wir dem Fluss zur Kundennachfrage keinen Widerstand entgegensetzen, kann die Wertschöpfung schwingen.“
Ernst Weichselbaum

„MGR. Mach’s gleich richtig.“
Ernst Weichselbaum

Worauf Zeitorientierte Softwareentwicklung gut verzichten kann

Artefakte der Softwareentwicklung, auf die es sich zu verzichten lohnt	Was viel besser funktioniert
Sprints/Kadenzen. Form der Chargenorientierung. Wie Hosen, die zum Tragen viel zu weit sind. Nicht floworientiert, verringern Geschwindigkeit und Selbstorganisation	Individuelle Items *time-boxen:* Beseitigt Notwendigkeit von Kadenzen
Iterationen/MVP. Agile Mythen. Iterative Produktentwicklung ist in Softwareentwicklung selten. Sie stellt keinen eigenen Sonderfall von Entwicklungsarbeit dar	Idee fallen lassen und die Begriffe nicht mehr verwenden
Backlogs. Verleiten Akteure zu schlechten Gesprächen über falsche Themen – z.B. *Priorisierung, Sprint-Planung, Macht*. Backlogs untergraben Klärung, qualitativ gute Gespräche und Diskurs. Wenn Backlogs ewig fortdauern, untergräbt das die Empfindung von Erfülltheit. Verkörpern alles, was an Agile/Scrum problematisch ist	Ein Verständnis größerer Vorhaben entwickeln, die innerhalb von 30-90 Tagen zu erledigen sind; Diskurs zu Fokus und Dringlichkeit verbessern
Scrum-Teams/Entwicklungsteams/„Scalable" Teams. Entwicklung von Software macht nicht per sé Teams erforderlich: Jede sinnvoll zugeschnittene Softwareentwicklungsarbeit („Item") lässt sich am besten von einer oder zwei Personen erledigen, nicht von einem Team. Entwicklungsteams zu erzwingen behindert Flow. Überwindung der Team-Illusion macht Rollen *wie Scrum Master, Product Owner, Agile Coach* überflüssig	Softwareentwicklung findet nicht in Teams statt, sondern individuell oder in Duos („Tandems"). Begriff und Vorstellung von Softwareentwicklungsteams fallen lassen
Schätzung/Story Points. Große agile Themen – zugleich irreführend und verschwenderisch. Schätzungen sind in TOSD entbehrlich, da die Realisierung zeitlich begrenzt ist und maximal drei Tage dauert	Idee fallen lassen und stattdessen zu Zeitorientierung und konsequentem Time-Boxing übergehen
Kanban-Boards. Visualisierungswerkzeuge, die fehlerhafte Strukturen und schlechte Gespräche verstärken. Fokussieren auf die falschen Dinge. In digitaler Form/getrennt von der Entwicklungsplattform fördern sie Verschwendung, Steuerung, Intransparenz	Listen und Items in einfachen Listenwerkzeugen verwalten; Arbeit im physischen Raum sichtbar machen
Retrospektiven. Besser ein interaktionsintensiver Arbeitsprozess *(MER: MitEinander Reden!),* als in festen Gruppen und arhythmischen Intervallen zu „reviewen"	Idee fallen lassen, stattdessen dann, wenn nötig, Gespräche führen

Worauf Zeitorientierte Softwareentwicklung gut verzichten kann (Fortsetzung)

Artefakte der Softwareentwicklung, auf die es sich zu verzichten lohnt	Was viel besser funktioniert
Skalierungsframeworks/Koordination. Basieren auf fehlerhaften Annahmen über Steuerung und Kontrolle von Softwareentwicklung durch „Team-Hierarchien“. Frameworks erzeugen Hierarchisierung und Bürokratie, leiteten den Niedergang der Agile-Bewegung ein. Natürliche Feinde von TOSD	Idee fallen lassen und die Begriffe nicht mehr verwenden.
Planning, z. B. PI-, Big-Room- oder Quarterly Planning. Lenkt von Realität, Arbeit und Jetzt-Orientierung ab. Ist das Gegenteil von Flow/Zeitorientierung	OpenSpace DEV für Diskurs, Problemlösung und Herstellung von Einheit nutzen
Agile Kennzahlen, fixierte Ziele. Abschaffen, da sie die Realität verschleiern. Es ist besser, ganz darauf zu verzichten, Ziele für Einzelpersonen, Teams oder Gruppen festzulegen	Relative Ziele für teambasierte Performancesysteme; Team-P&Ls etablieren und Urteile an der Realität des Flow orientieren
Kapazitätskennzahlen. Kapazität sollte weder gemessen noch diskutiert, sondern gestaltet/beeinflusst werden. Das bedarf Handeln, nicht Kennzahlen	Kapazität entsprechend Nachfrage schwingen/variieren lassen, bei fixen Time-Boxes
Rollen Lead/Scrum Master/Product Owner/Agile Coach. Entmündigen Entwickler:innen, erzeugen Steuerung und Verschwendung, schaffen Barrieren für Flow und neigen dazu, Probleme zu verschleiern	Zellstrukturdesign- Prinzipien anwenden; Teams klein halten (4-8 Personen) und über Entwicklung hinaus funktional integrieren
Freigaben/Production-Readiness-Reviews. Sind Barrieren für Flow	Abschaffen und Begriffe nicht verwenden
User Stories. Ein agiler Mythos bzw. eine Illusion. Schlechter Ersatz für Gruppengespräche mit Kunden	Saubere Konzepte erarbeiten; Arbeit im Raum visualisieren, Klarheit schaffen
Agile Reifegradmodelle, Personen-Kategorisierungen, Readiness-Messungen. Nicht hilfreich und urteilend – erforderlich ist Arbeit am System	Beurteilung von Menschen stoppen und am System arbeiten, wertende Sprache meiden
Projektfinanzierung/Ressourcenpools. Durch Budgets, Freigaben und Zuweisungen von Kapazitäten (Entwickler:innen-„Pools“)	Budgets abschaffen, stattdessen zur Messung Team-P&Ls einführen

Eine zentrale Neuerung in Zeitorientierter Softwareentwicklung: der OK-Punkt

Die Zeitorientierte Softwareentwicklung beruht auf einer zeitlichen Unterscheidung, die wir den OK-Punkt nennen. Der OK-Punkt erlaubt es, *zwischen Konzeptionierung und Realisierung von Softwareentwicklungsarbeit zu unterscheiden.* Das mag zunächst ungewöhnlich erscheinen, ist aber fundamental. Zugleich wirft es die Frage auf, warum diese Unterscheidung in „Agile"-Ansätzen weitgehend ignoriert wurde. Klar ist jedoch: Ohne den OK-Punkt, der Konzeptionierung und Realisierung voneinander trennt, ist Zeitorientierung in der Softwareentwicklung nicht denkbar. Realisierung lässt sich nur dann verlässlich zeitlich begrenzen, wenn ein Konzept für die Realisierung im Vorfeld erarbeitet und abgeschlossen wurde. Der OK-Punkt ist der Ort, an dem Konzeptionierung und Realisierung sich die Hand geben.



Zeitorientierte Softwareentwicklung: Der OK-Punkt gibt dem Prozess Struktur. Das ermöglicht Timeboxing in der Realisierung



Führt zu einem OK-Punkt-tauglichen Konzept für ein Item

Konzepte werden schriftlich dokumentiert und sind für alle sichtbar

Enthält Definitionen von Anforderungen, Realisierungs-Timebox und beteiligten Personen

Testen & Dokumentation sind Teil des Prozesses, nicht separat (TDD)

“Ein Handschlag”

Ohne ihn kann kein Item weiterfliessen

Basiert immer auf Vereinbarung oder konsultativer Entscheidungsfindung –nicht auf Anweisung

Kann von anderen Personen als denen, die die Konzeptionierung erledigt haben, durchgeführt werden

Sollten Zweifel am Timing bestehen, ist der OK-Punkt nicht erreicht

Testen & Dokumentation sind Teil des Prozesses, nicht separat (TDD)

Zeitorientierte Softwareentwicklung: Der beste Weg, um Struktur mit Selbstorganisation, Flow mit Disziplin in Einklang zu bringen

Die Liste

- Hat mit einem Problem zu tun
- Enthält alle Items, die der Bearbeitung bedürfen
- Die Reihung ist von Bedeutung
- Es gibt einen *Eigentümer der Liste*
- Kann jederzeit aktualisiert werden
- Sollte irgendwann abgearbeitet sein, damit eine neue Liste starten kann

Jedes Item (Feature oder Projekt) wird einmal durch Konzeptionierung und Realisierung fließen



Das Eine TOSD-Prinzip zur Problemlösung:

MER: MitEinander Reden

Gespräch kann zu jedem Zeitpunkt stattfinden, nicht schriftlich, am besten Face-to-Face

In Zeitorientierter Softwareentwicklung ist die Zeit fixiert, während die Kapazität schwingt

Im kapazitätsorientierten Kontrolldenken im Zusammenhang mit Softwareentwicklung wird Kapazität als gegeben vorausgesetzt. Kapazität, so meinen wir, sei mit „Arbeitsstunden der Humanressourcen“ gleichzusetzen – also der Zeit der Entwickler:innen. Schwanken Nachfrage oder der Schwierigkeitsgrad von Aufgaben, so muss folgerichtig auch die Lieferzeit variieren und schwanken. Fehlannahmen wie diese haben zu vielen Missverständnissen über Softwareentwicklung geführt – etwa zu der Vorstellung, Softwareentwicklung sei *eine Kunst*; oder, im anderen Extrem, die Entwicklung müsse von oben gesteuert und kontrolliert, sowie durch Budgets, Steuerung und Verantwortliche (a.k.a. Manager) gebändigt werden.

In TOSD nehmen wir die entgegengesetzte (und realistischere) Haltung ein. Die Lieferzeit sollte fix sein, während Kapazität sich natürlicherweise schwingend anpasst oder variiert (was sie ohnehin ständig tut). Dabei gehen wir davon aus, dass Softwareentwicklung ein *Handwerk* ist (keine Kunst) und dass Selbstorganisation in der Softwareentwicklung natürlich ist – so wie überall sonst auch. Die Idee, dass Kapazität schwingt und Zeit fix sein muss, um Flow, Pünktlichkeit, Wirksamkeit und menschliche Erfüllung zu ermöglichen, wurde erstmals vom Pionier zeitorientierter Arbeitssysteme Ernst Weichselbaum formuliert. Seine Arbeit mit Industrieunternehmen war vom Konstruktivismus und dem Konzept fraktaler Organisationen inspiriert. Ernst Weichselbaum prägte zudem das Konzept des *OK-Punkts*, das er vor allem auf die *Nahtstelle* zwischen Kunden und Lieferanten anwandte.



Freunde von TOSD: Werkzeuge, Konzepte und Praktiken, die zur Zeitorientierung in der Softwareentwicklung passen

Auswahl, in alphabetischer Reihenfolge

- Agentic AI
- Agentic IDEs
- Architecture as code
- Architecture Decision Records ADR
- Behavior Driven Development BDD
- BetaCodex, Dezentralisierung
- Boy Scout Rule
- Zellstrukturdesign
- Chaos as Code
- Chaos Engineering
- Clean Code, z.B. 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 (z.B. 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 Ziele
- Scaffolding
- Socio-Technical Systems
- Single Responsibility/Open-Closed/Liskov Substitution/ Interface Segregation/Dependency inversion SOLID
- Static Code Analysis
- Storyboards
- Test-Driven Development TDD
- Vertical Slice Architecture
- Vibe Programming
- Workflow Automation (z.B. Jenkins, GitHub Actions)

Fallstudie Pradtke GmbH: Wie Pradtke zur Zeitorientierung kam

Seit Mitte der 1990er-Jahre entwickelt und vermarktet die Pradtke GmbH in Bochum Software für Organisationen wie Krankenhäuser und Feuerwehren. Nach Jahren eher durchwachsener Ergebnisse vollzog Pradtke in der zweiten Hälfte des Jahres 2024 erfolgreich eine Transformation hin zu BetaCodex-Prinzipien – mithilfe des OpenSpace Beta-Ansatzes. Im November 2024 ging das Unternehmen im Rahmen der 90-tägigen OpenSpace-Beta-Transformation mit seinem dezentralen Zellstrukturdesign mit acht Peripheriezellen live. Die funktionale Integration ermöglichte eine deutliche Verschlankung von Administration und Betrieb. **Das Unternehmen wurde spürbar schlanker, erreichte eine stabile Profitabilität und verbesserte die Kundenzufriedenheit zügig.** Im ersten Halbjahr 2025 vereinfachte und reformierte Pradtke sein Geschäftsmodell erfolgreich und überführte Hunderte von Unternehmenskund:innen in ein neues Modell. Die Menschen bei Pradtke würden sagen: Zwischen dem Sommer 2024 und dem Sommer 2025 haben wir als Unternehmen all jene Dinge in die Praxis umgesetzt, die wir zuvor immer wieder versucht hatten – und die wir für unmöglich hielten.

In der Softwareentwicklung sah die Lage jedoch weniger rosig aus. Das Unternehmen befand sich mitten in einem großen Migrationsprojekt: der Umstellung seiner Legacy-Desktop-Anwendung auf eine Single-Page-Application. Das bedeutete einen Wechsel des Mediums, der Technologie, der Werkzeuge – kurz: ein grundlegender Umbruch. Es fiel dem Unternehmen schwer, für dieses Vorhaben Momentum aufzubauen; die Softwareentwicklung folgte dem durchaus verbreiteten „Scrum-but"-Ansatz. Es gab ein JIRA-Board, doch alle Beteiligten in der Softwareentwicklung kämpften mit den üblichen Problemen. Insgesamt blieb das Deployment viel zu langsam – geprägt von wackeligen Schätzungen, ineffizienten Planungssitzungen und überflüssigen Ritualen wie Retrospektiven und Reviews.

„Seit seiner Gründung hat Pradtke eine Entwicklung durchlaufen, die den meisten Softwareunternehmen ähnelt: wachsende technische Schulden, Berge von Legacy-Code, von dem niemand mehr genau weiß, was er eigentlich tut, ausgeprägtes 'Not-Invented-Here'-Syndrom – und ständige Versuche, langfristige Architekturziele mit kurzfristigen Kundenbedürfnissen in Einklang zu bringen. Wir jonglierten mit den 'begrenzten' Entwicklerressourcen in der Hoffnung, möglichst viel aus ihnen herauszuholen. Auf diesem Weg probierten wir Dinge aus wie Wasserfall, Scrum, JIRA, Estimation und OKRs – Ansätze, die zu keinem Zeitpunkt das leisteten, was wir uns von ihnen erhofften. Wir wurden nicht schneller, die Software wurde auch nicht wesentlich besser. Rückblickend lag das daran, dass wir die zugrunde liegenden Probleme nicht wirklich verstanden hatten. Wir hofften, Probleme ließen sich 'reparieren', wenn wir nur die richtigen Prozesse, Frameworks und Modelle auf sie anwendeten."
Sebastian Kubsch

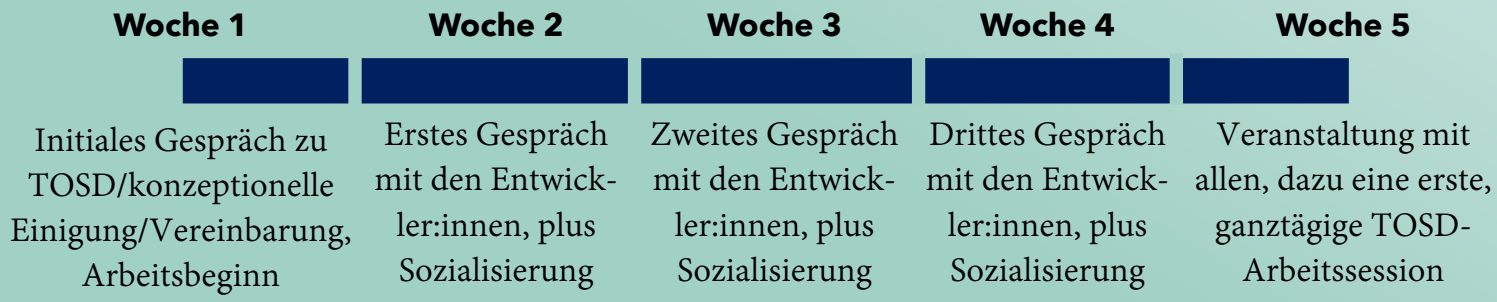
Fallstudie Pradtke GmbH: In vier Wochen live zur Zeitorientierung in der Entwicklung

Unter Führung des CTO und gemeinsam mit den Geschäftsführern Marc Otten und André Pradtke startete das Unternehmen im August 2025 die Realisierung von TOSD. Während Pradtke TOSD anging, arbeiteten wir den Ansatz weiter aus – über die ersten Konzepte wie *OK-Punkt, Die Liste* und *fixierte Realisierungszeit* hinaus. *Die Arbeit am System* erfolgte im Gleichklang mit den 14 Softwareentwickler:innen des Unternehmens sowie zehn potenziellen weiteren Entwickler:innen. Es stellten sich berechtigte Fragen: **Wie lässt sich TOSD mit allen Entwickler:innen gleichzeitig starten?** Wie wird laufende Arbeit transparent gemacht? Wie kann Arbeit beschleunigt werden, ohne dass Entwickler:innen mehr Stunden arbeiten müssen? Wie lassen sich operative Aufgaben innerhalb von Peripheriezellen („Businessteams") so verteilen, dass die Entwickler:innen, die Teil dieser Zellen sind, ausreichend fokussiert an Items der Liste arbeiten können? Wie lassen sich Items so fassen, dass sie ihre Realisierung stets ein bis drei Tage dauert?

Nach der konzeptionellen Einigung in der Startwoche folgten über drei Wochen hinweg drei Runden jeweils dreistündiger Gespräche mit den Entwickler:innen. In der darauf folgenden Woche fand der *Go Live* im Rahmen einer TOSD-Arbeitssession für alle Entwickler:innen statt. Zur gemeinsamen Eröffnungssession am gleichen Tag waren alle Mitarbeitenden des Unternehmens eingeladen.

„Mit dem neuen Zellstrukturdesign entwickelten die meisten Teams schnell ein Gefühl für Selbstorganisation und sinnvolles, unmittelbares Handeln – frei von künstlichen Hierarchien und Budgets. Vieles kam gut in Bewegung. Eine Sache verbesserte sich jedoch nicht: der Prozess der Softwareentwicklung."
Sebastian Kubsch

„Für ein Softwareunternehmen wie unseres, das seit den 1990er-Jahren besteht, lag die Herausforderung darin, drei Jahrzehnte schlechter Gewohnheiten in vier Wochen hinter sich zu lassen."
Sebastian Kubsch



Fallstudie Pradtke GmbH: Ks und Rs – erste Schritte zur Zeitorientierten Softwareentwicklung

Der erste Schritt bestand darin, die Entwickler:innen und Könner:innen im Unternehmen zu identifizieren und zu benennen, die autorisiert sein sollten, Konzepte bzw. Anforderungen zu entwickeln. Softwarequalität und Flow würden später in hohem Maße von der Qualität der zugrunde liegenden Konzepte für jedes einzelne Item abhängen. Offensichtlich musste diese Gruppe von Könner:innen *klein und kohäsiv* sein. Wir gingen das *Zellstrukturdesign* durch, um die Akteure zu identifizieren, die als *Ks* (für *Konzeptionierer*) geeignet waren. Neben fachlicher Könnerschaft waren Auswahlkriterien für die Personen dieser Gruppe eine offene Haltung gegenüber Interaktion mit anderen sowie die Verfügbarkeit für Gespräche mit allen. Der nächste Schritt war, diejenigen zu identifizieren, die als *Realisierer* Wertbeiträge leisten würden. Daraus wurde die Gruppe der *Rs*. Wie zu erwarten war, erwies sich diese Gruppe von Personen als hochgradig heterogen und breit gefächert in Fähigkeiten, Könnerschaften und professionellem Reifegrad. Einige *Ks* wurden auch als *Rs* identifiziert: *Natürlich können Ks auch Rs sein*.

Damit war das initiale TOSD-Setup abgeschlossen. Der CTO wurde *Eigentümer der Liste*. *Ks* erstellen Konzepte für *Rs*. *Rs* realisieren diese Konzepte innerhalb vereinbarter *Time-boxes*. Konzepte, die den *OK-Punkt* erreicht haben, müssen dem Prinzip nach *rückfragefrei* realisierbar sein: Der OK-Punkt verlangt Einvernehmen und einen Handschlag zwischen *Ks* und *Rs*. Wichtig: **Gespräche und Konsultation können jederzeit unter allen stattfinden.** Tage später führten wir einen dreistündigen Workshop mit den *Ks* durch. In den Wochen darauf dreistündige Gesprächsrunden mit allen *Rs*. In jeder Session wurden Idee und Konzept von TOSD *von Anfang bis Ende* dargelegt, *ausgehend vom Bedarf des Geschäfts*. Nichts wurde erzwungen, alles offen und ernsthaft diskutiert. Das Feedback war gut, es kam zu konstruktiver Klärung. Ideen, die während der Gespräche entstanden, wurden unmittelbar ins TOSD-Konzept eingearbeitet. Externe Moderation half, diese Workshops reibungslos durchzuführen. Es wurde diskutiert, dass Die Liste, die mehrere hundert Items enthielt, innerhalb von ca. acht Wochen abgearbeitet werden könne. Nun waren alle an Bord und motiviert zu starten.

„Im August 2025 trat Niels Pfläging mit einer Alternative an uns heran: Zeitorientierung. Das klang einfach. Weg mit dem kadenzbasierten Unsinn, Abschied vom Konzept der Sprints, weg mit unnötigen Rollen wie Product Owner und Scrum Master. Und vor allem: aufhören, Flow durch die Veränderung von Lieferzeiten erzeugen zu wollen. Stattdessen würden wir die Lieferzeiten fixieren und die Kapazität schwingen lassen. Als Niels seinen Vorschlag mit einem unserer Geschäftsführer und mir besprach, war ich gefesselt. Zugleich fragte ich mich: Konnte es am Ende so einfach sein, zu Flow zu kommen?" Sebastian Kubsch

„In TOSD gibt es kein Schätzspiel: Niemand spielt Estimation Poker. Items müssen klar aufzeigen, ob ihre Realisierung 1, 2 oder 3 Tage dauert. Für Rs ist es Ehrensache, innerhalb dieses Zeitfensters zu liefern – nicht einen Tag später oder zwei." Sebastian Kubsch

Fallstudie Pradtke GmbH: Wirksamkeit durch Zeitorientierung deutlich steigern – Woche für Woche

Erwartungsgemäß gab es in der 1. Woche in TOSD einige Schwierigkeiten. Die Konzepte einiger Items waren nicht präzise genug, sodass in der Realisierung grundlegende Fragen aufkamen. Einige Realisierer unterschätzten den eigenen Zeitbedarf. Ks produzierten Konzepte schneller, als sie realisiert wurden, sodass sich am OK-Punkt Arbeit aufstaute. Insgesamt ließen sich diese Themen auf einfache Weise handhaben: durch häufigere, bessere Gespräche. Ks sprachen darüber, wie der Detaillierungsgrad von Konzepten erhöht bzw. reduziert werden konnte. Rs begannen, sich bei konkreten Items zusammenzutun. Um gleichmäßigeren Flow zu erreichen, mussten Items in Der Liste zudem neu priorisiert werden. Einige Entwickler wurden als K oder R re-klassifiziert, und Gespräche geführt, um den Fokus stärker auf Realisierung zu legen. **Schon die 2. Woche brachte spürbare Leistungsgewinne.** Im Vergleich zur Vorwoche verdoppelte sich der Durchsatz. Alle begannen, sich an die neue Arbeitsweise zu gewöhnen. Erste Muster etablierten sich, Momentum entstand.
In der 3. Woche kam es zu einem Durchbruch. Rs begannen zu erkennen, dass TOSD funktionierte. Alle konnten sehen, dass bei der Auslieferung von Items an die Kund:innen Fortschritte erzielt wurden. **Bis zur 4. Woche waren nahezu so viele Items realisiert worden, wie in den sechs Monaten zuvor.** Flow begann sich einzustellen, doch es blieben viele Herausforderungen. Das für Flow wichtige Verhältnis von Konzepten zu Realisierung erforderte ständige Aufmerksamkeit. Die Kapazität schwang bislang weder in der Konzeptionierung noch in der Realisierung. Einige Konzepte hielten nicht stand, bedingt durch unzureichende Arbeitsstandards in der Konzeptionierung: In der Folge wurden unter den Ks einige Standards vereinbart.

Ein Game Changer in TOSD: Probleme werden unmittelbar gelöst. Stockungen werden sofort adressiert. Die Unmittelbarkeit des Ansatzes ermöglicht es, sich auf das Wesentliche zu konzentrieren. Statt Stunden oder Tage in ritualisierten Planungsmeetings abzusitzen, zwingt TOSD, sofort zu handeln: genau in diesem Moment.

„Ks wählen Items aus Der Liste aus, mit Fokus auf Items mit hoher Priorität – idealerweise solche Items, die sich mit ihren jeweiligen Könnerschaften decken. Ks leisten notwendige Denk- und Vorarbeit für die Realisierer, um ihnen zu vermitteln, WAS zu tun ist – jedoch ohne ihnen zu sagen, WIE sie es umzusetzen haben. Abhängigkeiten, unerwartetes Systemverhalten und andere Besonderheiten müssen in den Konzepten beschrieben werden. Mitunter ist es notwendig, sich dafür in das zugrunde liegende Datenbankmodell oder den Legacy-Code einzuarbeiten. Das Ziel ist klar: Rs sollen Items so schnell wie möglich realisieren können, indem die erforderlichen Informationen im Vorfeld zusammengetragen werden. Ks sollten die richtigen Rs für ihre Items identifizieren und ansprechen. Rs können Ks nach Items fragen – und es wird auch von ihnen erwartet." Sebastian Kubsch

„Derzeit nutzen wir ein simples GitHub Project, um Items zu verfolgen. Die Devise: Übertreib' es nicht, halte die Dinge schlicht. Bau' nicht das nächste Backlog." Sebastian Kubsch

Fallstudie Pradtke GmbH: Überlegungen zur Erreichung von Könnerschaft in der Zeitorientierung

Wenn wir in Softwareunternehmen und Entwicklungsteams höhere Wirksamkeit erreichen wollen, sollten wir uns fragen: Was ist in der Entwicklung die größte Verschwendung? Wir sind überzeugt, dass es allzu oft eine Kombination aus Zögern, Verzögern und wahrgenommener Isolation bzw. *erlernter Hilflosigkeit* unter Entwickler:innen ist. **Die meisten Softwareentwicklungsteams scheinen zu *Stop-and-Go* und partieller Inaktivität zu tendieren.** Nicht, weil sie es wollen, sondern weil die Kontexte, die durch gängige Managementmethoden entstehen, dieses *Stop-and-Go* unvermeidlich machen. Meetings, Routinen, Rituale, Zwang, hierarchische Freigaben und Silo-Organisation verhindern ein Gefühl von Selbstorganisation. Flow wird blockiert und persönliche Erfüllung bei der Arbeit nimmt ab, was den Output stark begrenzt. Das traf über Jahre hinweg auch auf die Softwareentwicklung bei Pradtke zu. Wir kamen zu dem Schluss, dass die daraus resultierende Verschwendung von Entwicklungszeit und der Mangel an Zusammenarbeit nur zu beseitigen wäre, wenn gängige Managementmuster (einschließlich *agiler* Muster) aufgebrochen und das konventionelle System der Softwareentwicklung überwunden würden. Zusammen stellen *Dezentralisierung* und TOSD Strukturen bereit, um begrenzende, verschwenderische Muster zu überwinden.

Dann stellt sich die Frage der Kapazität: Wie kann sie erhöht werden? Wie lässt sich Kapazität so zum Schwingen bringen, dass sie der Nachfrage folgt? Die vorschnelle Antwort auf die erste Frage lautet, dass kurzfristiges Recruiting der einfachste, vielleicht gar der einzige Weg sei, Kapazität zu erhöhen. Häufig ist es jedoch kaum praktikabel oder notwendig, schnell zu rekrutieren oder Personal aufzustocken. Stattdessen ist es wichtig zu verstehen, dass *Kapazität in beide Richtungen schwingt*. Es wird Zeiten geben, in denen die Kapazität höher ist, und Zeiten, in denen sie niedriger ist. *Der zentrale Hebel für mehr Kapazität ist nicht Recruiting, sondern viel bessere Zusammenarbeit:* Befreiung von der Last von Freigaben, Wartezeiten, sinnlosen Meetings und unnötigen Workarounds lässt gegenseitige Unterstützung, Konsultation und *Pair Programming* selbstverständlich. All dies sind Schritte, die bei Pradtke gegangen wurden.

„Fang' TOSD mit einfacheren Items geringerer Komplexität an, damit sich anfangs alle auf das Erlernen der Methode konzentrieren können, statt sich nur mit den Inhalten abzumühen. Das hilft, schneller Fahrt aufzunehmen."
Sebastian Kubsch

„In der 3. Woche kam ein erfahrener Entwickler auf mich zu. Er erzählte mir, wie glücklich er mit dem neuen Modell sei: dass sich die Dinge endlich bewegten. Er sei erleichtert, dass wir nun endlich an Items arbeiteten, über die wir seit Jahren gesprochen hatten, aber zuvor nicht angegangen waren. Er zeigte mir ein Feature, das er gerade fertiggestellt hatte und auf das er sichtbar stolz war. In diesem Moment wusste ich: wir waren auf dem richtigen Weg."
Sebastian Kubsch

Fallstudie Pradtke GmbH: Überlegungen zur Erreichung von Könnerschaft in der Zeitorientierung (Fortsetzung)

Der dritte Aspekt, der in der Zeitorientierung unübersehbar wird, ist die Bedeutung qualitativ hochwertiger Interaktion unter Entwickler:innen. Mangelnde oder qualitativ schlechte Interaktion neigt dazu, den Output zu reduzieren und die Gesamtqualität von Software zu beeinträchtigen. Gute Interaktionsqualität kann strukturell begünstigt und provoziert werden. TOSD löst strukturell Gesprächsbedarfe aus: in Bezug auf *Die Liste*, die einzelnen Items und ihre Reihung; rund um Konstellationen oder Paarbildungen in Konzeptionierung und Realisierung; während der Konzeptualisierung, am OK-Punkt und während der Realisierung. Es gibt in TOSD eine Menge Gesprächsanlässe. Wir sind überzeugt, dass die Erhöhung von Face-to-Face-Begegnungen unter Softwareentwickler:innen für Leistung eine zentrale Rolle spielt. In den ersten Wochen mit TOSD bei Pradtke umfassten Interventionen zur Steigerung der persönlichen Interaktion z.B. das tägliche Angebot eines gemeinsamen *Mittagessens im Büro*, Vereinbarungen mit Entwickler:innen, um *einen Teil der Arbeit vom Homeoffice zurück ins Büro zu verlagern*, sowie die Einführung *freiwilliger Arbeits- und Lern-Sessions* an Samstagen.

In diesem Zusammenhang verdient das zentrale TOSD-Prinzip „MER: *MitEinander Reden*" Erwähnung. Miteinander zu sprechen, wann immer Probleme auftreten, ist besser, als Regeln zu definieren, Ausschüsse einzusetzen, periodische Meetings abzuhalten oder Steuerungs- und Aufsichtsrollen zu installieren. Sich bewusst für das unmittelbare, offene Gespräch zu entscheiden, ist ein Muskel, der von allen trainiert werden muss – insbesondere dann, wenn dieser Muskel, wie bei Pradtke, über Jahre hinweg zu wenig genutzt wurde. Die Idee hinter dem Prinzip *MER* ist, das Aufschieben von Gesprächen bis zu einem späteren Anlass als inakzeptabel zu markieren und die Beseitigung von Hindernissen *on the spot* zum Normalfall zu erklären. Geredet wird genau in dem Moment, in dem ein Problem entsteht. Sicherlich lassen sich nicht alle Probleme sofort lösen. Was nicht ad hoc gelöst werden kann, kann zu neuen Items führen, die *Die Liste* ergänzen. In der Entwicklung ist das *MER-Prinzip* untrennbar mit Zeitorientierung verbunden.

„In TOSD wäre es leichtfertig und absurd, das Gespräch über ein gerade aufgetretenes Problem aufzuschieben."
Sebastian Kubsch

„Autorisierung ist ein zentraler Aspekt der Zeitorientierung. Unsere Leute waren es gewohnt, vor dem Handeln auf Legitimation zu warten. Das machte alles langsam. Mit TOSD haben wir von Anfang an geklärt, dass alle bereits autorisiert sind zu handeln, weil wir ihnen vertrauten und weil wir wussten, dass sie ihre Arbeit gut machen würden. Genau das taten sie. Weil sie es wollten – nicht wegen irgendeiner Art von Anweisung." Sebastian Kubsch

„Wähl' die richtigen Personen aus. Nachlässige Auswahl der Ks führt zu schlechten Ergebnissen. Prüf' kontinuierlich, ob die als Ks Nominierten die richtigen für die anstehende Arbeit sind. Selbstverständlich verändern sich Anforderungen im Lauf der Zeit."
Sebastian Kubsch

Stärken der Zeitorientierten Softwareentwicklung: Eine Zusammenfassung

1. **Kann jederzeit gestartet werden.** Der Ramp-up erfordert nicht mehr als ein paar Wochen und bedarf weniger externer Ressourcen. Er lässt sich zudem mit dem OpenSpace-Beta-Ansatz zur Transformation ganzer Organisationen kombinieren. Der dauert 90 Tage.
2. **Ist ausgesprochen geradlinig:** Erfordert keine speziellen Schulungen zu Beginn – nur Einigkeit über Prozess und Prinzipien. Beseitigt unmittelbar den Bedarf an ausuferndem Overhead, an Steuerung, Meetings und Verhaltenskontrolle.
3. **Erzeugt unmittelbar Flow und Zeitorientierung,** also Voraussetzungen für Selbstorganisation und Zusammenarbeit, die ihrerseits Hebel für Wirksamkeit und Leistungsfähigkeit sind.
4. **Erzeugt Fokus und Disziplin, ohne Bedarf an Hierarchie oder Bürokratie,** durch Die Liste (und den *Hüter der Liste* oder *Wächter des Flow*), durch den OK-Punkt und Verzicht auf Regeln, auf Steuerungsrollen oder Hebeln für Eingriffe durch Manager.
5. **Beschleunigt sowohl Konzeptionierung als auch Realisierung** durch die Unterscheidung der zwei entscheidenden Formen von Könnerschaft in der Softwareentwicklung.
6. **Skaliert auf natürliche Weise** auf 100e oder 1000e von Personen; stützt sich nicht auf „Entwicklungsteams". Erlaubt, mehrere Listen von Items parallel zu führen („Initiativen").
7. **Steigert durch Unmittelbarkeit von Gesprächen die Interaktionsqualität drastisch**.
8. **Integriert Gespräche in den Prozess der Softwareentwicklung,** statt Diskurs auf separate Rituale oder Meetings zu beschränken. Gespräche werden wieder Teil der Arbeit.
9. **Lässt sich problemlos mit weiteren soziotechnischen Ansätzen für Agilität und Selbstorganisation kombinieren,** z. B. mit Großgruppen-Interaktionsformaten (OpenSpace DEV), dezentralen Strukturen (Zellstrukturdesign) und relativen Leistungssystemen (Relative Ziele).

„Der OK-Punkt ist der tägliche Urknall der [Bearbeitung von Software-Items]. Da die Übergabe stark ritualisiert ist, ist der OK-Punkt so etwas wie der Herzschlag des Unternehmens."
Ernst Weichselbaum

„Disponieren ist eine Krankheit. Sie gehört zu den Nierenkrankheiten. Ebenso wie das Koordinieren."
Ernst Weichselbaum

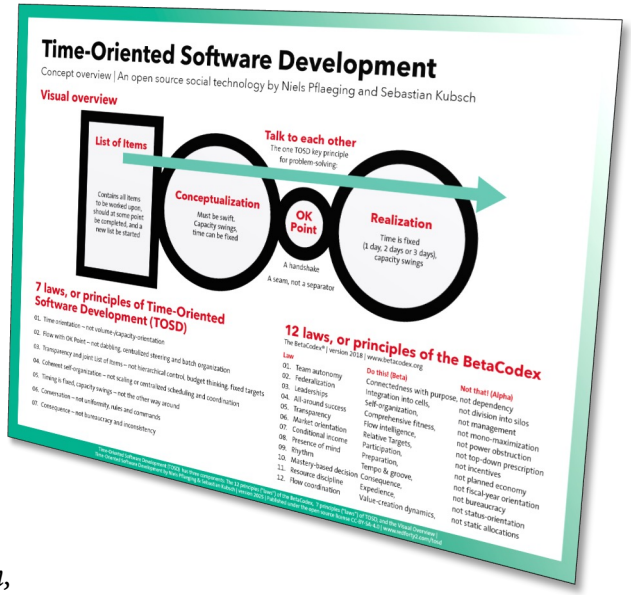
„Gestalte den Prozess so, dass er nicht Überwacht werden muss. Lieber Pünktlichkeit gestalten als Verspätung verwalten."
Ernst Weichselbaum

„Autorität ist keine Eigenschaft von Individuen, sondern wird durch Interaktion von mindestens zwei Menschen erzeugt. Autorität geht von Vereinbarung aus."
Ernst Weichselbaum

Zeitorientierte Softwareentwicklung (TOSD): Über die Open-Source-Lizenz Attribution ShareAlike (CC-BY-SA)

Diese Lizenz erlaubt es Dir, Zeitorientierte Softwareentwicklung (Time-Oriented Software Development, TOSD) zu nutzen, zu kommerzialisieren, zu remixen, anzupassen und darauf aufzubauen.
Dabei erklärst Du Dich mit drei einfachen Punkten einverstanden:

- Stets die ursprünglichen Autoren, Niels Pfläging und Sebastian Kubsch, zu nennen,
- den von uns angegebenen Link zum Quellmaterial wie unten aufgeführt bereitzustellen,
- und Deine abgeleiteten Werke, falls du welche kreierst, unter den identischen Bedingungen an andere weiter zu lizenzieren.



Konkret musst Du den folgenden Satz sowie den dazugehörigen Link bei der Nutzung der Originalwerke oder bei abgeleiteten Werken bereitstellen, d.h. gut sichtbar platzieren, sowie als Bestandteil sämtlicher zugehöriger Texte oder Grafiken aufnehmen, die Du entwickelst:

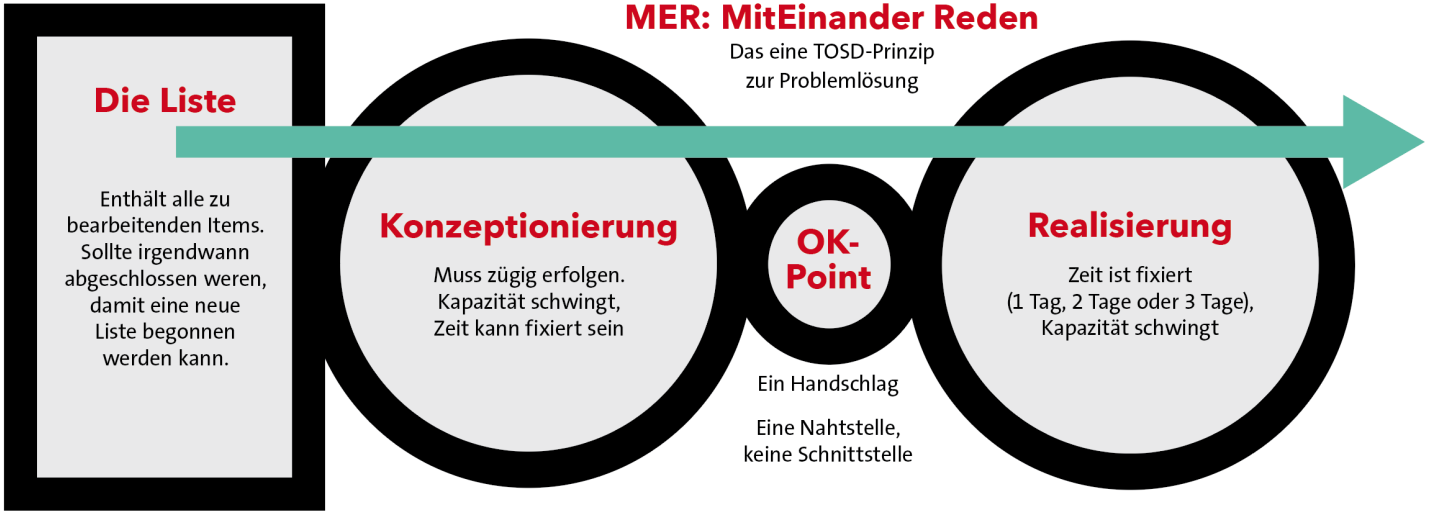
"This work is derived from Time-Oriented Software Development, an open source, free social technology by Niels Pflaeging and Sebastian Kubsch, published under the CC-BY-SA-4.0 license from Creative Commons, and found here: www.redforty2.com/tosd"

Schau' Dir auch die anderen Open-Source-Sozialtechnologien an, die seit 2008 aus der BetaCodex-Bewegung hervorgegangen sind: BetaCodex, Zellstrukturdesign, Relative Ziele, OpenSpace Beta und OpenSpace DEV.

Zeitorientierte Softwareentwicklung (TOSD)

Konzeptüberblick | Eine Open Source-Sozialtechnologie von Niels Pfläging und Sebastian Kubsch

Visueller Überblick



7 Prinzipien ("Gesetze") der Zeitorientierten Softwareentwicklung

01. **Zeitorientierung** – nicht Volumen-/Kapazitätsorientierung
02. **Flow mit OK Point** – nicht Herumprobieren, zentrale Steuerung und Chargenbildung
03. **Transparenz und gemeinsame Item-Liste** – nicht hierarchische Kontrolle, Budgetdenken, fixierte Ziele
04. **Konsistente Selbstorganisation** – nicht Skalierung, zentrale Ablaufplanung und Koordinierung
05. **Zeit ist fixiert, Kapazität schwingt** – nicht umgekehrt
06. **MER: MitEinander Reden** – nicht Uniformität, Regeln oder Anweisungen
07. **Kohärenz im Managementhandeln** – nicht Bürokratie, Inkonsistenz & mangelnde Konsequenz

12 Prinzipien ("Gesetze") des BetaCodex

Der BetaCodex® | Version 2018 | www.betacodex.org

Gesetz	Tu' dies! (Beta)	Nicht das! (Alpha)
01. Teamautonomie	Sinnkopplung	statt Abhängigkeit
02. Föderalisierung	Zellstruktur	statt abgeteilter Silos
03. Leaderships	Selbstorganisation	statt Management
04. Rundumerfolg	Passgenauigkeit	statt Monomaximierung
05. Transparenz	Fließintelligenz	statt Machtverstopfung
06. Marktorientierung	Relative Ziele	statt Chefvorgabe
07. Bedingtes Arbeitseinkommen	Teilhabe	statt Anreizung
08. Geistesgegenwart	Vorbereitung	statt Planwirtschaft
09. Rhythmus	Taktgefühl	statt Fiskaljahresorientierung
10. Könnerentscheidung	Konsequenz	statt Bürokratie
11. Ressourcendisziplin	Zweckdienlichkeit	statt Statusgedöns
12. Flowkoordination	Wertschöpfungsdynamik	statt Zuweisungsstatik

Zeitorientierte Softwareentwicklung (Time-Oriented Software Development, TOSD) hat drei Komponenten: Die 12 Prinzipien ("Gesetze") des BetaCodex, 7 Prinzipien ("Gesetze") der Zeitorientierten Softwareentwicklung, sowie den visuellen Überblick | Zeitorientierte Softwareentwicklung von Niels Pfläging & Sebastian Kubsch | Version 2025 | Veröffentlicht unter der Open Source-Lizenz CC-BY-SA-4.0 | www.redforty2.com/tosd

Quellen und empfohlene Lektüre

Zeit & Organisation

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

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

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

Zeitorientierte Arbeitssysteme & Dezentralisierung

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

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

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

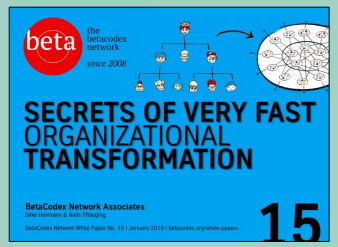
Pfläging, Niels/Hermann, Silke (2024): *Komplexithoden*, erweiterte Neuauflage

Pfläging, Niels/Hermann, Silke (2024): *Die Gesetze des Beta-Kodex*, 3. Auflage

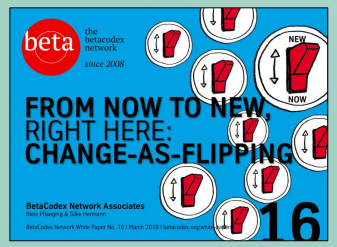
Pfläging, Niels/Hermann, Silke (2024): *Cell Structure Design Patterns, II*

Weichselbaum, Ernst/Pfläging, Niels (Hrsg.) (2020): *In jedem Unternehmen steckt ein besseres*

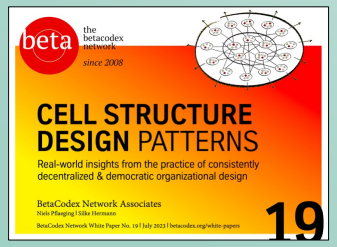
Ergänzende Forschungspapiere des BetaCodex Network, verfügbar unter www.betacodex.org:



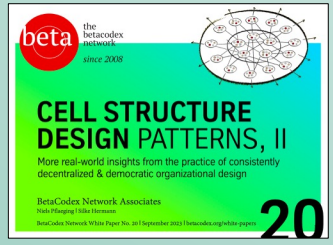
Secrets of Very Fast Organizational Transf., 2019



From Now to New, Right Here, 2019



Cell Structure Design Patterns, I, 2023



Cell Structure Design Patterns, II, 2024



Slave to the Rhythm, 2025

Die Forschungspapiere des BetaCodex Network

Kostenfrei abrufbar unter betacodex.org/white-papers

the betacodex network since 2008



Ausgewählte Buchempfehlungen

Das wundervolle Buch des großen Ernst Weichselbaum (1944-2024) – konzipiert, designt und herausgegeben von Niels Pfläging



In jedem Unternehmen steckt ein besseres
Ernst Weichselbaum/
Niels Pfläging (Hrsg.),
Vahlen, 2020
Auch als E-Book erhältlich

Über 160 Zitate vom Pionier der Zeitorientierung, dem Architekten des Toyota-Produktionssystems Taiichi Ohno (1912–1990)



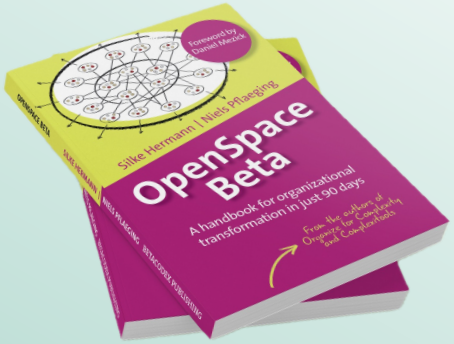
What would Ohno-san do?
Herausgegeben von
Niels Pfläging/Stefan Schmeing
BetaCodex Press, 2025
Auch als E-book erhältlich

Gebündelte Weisheit des ersten Philosophen und Praktikers der Zeitorientierung: W. Edwards Deming (1900–1993)



What would Deming do?
Herausgegeben von
Niels Pfläging
BetaCodex Press, 2023
Auch als E-book erhältlich

Wie lässt sich Transformation wirksam und auf zeitorientierte Weise gestalten? Der Ansatz liefert Einsichten und konkrete Konzepte. Vielfach bewährt



OpenSpace Beta
Silke Hermann/Niels Pfläging
Vahlen, 2020
Auch als E-Book erhältlich
Auch in Englisch und Koreanisch erhältlich.

Besuch den Shop unter redforty2.com/shop

the
betacodex
network

since 2008

Nimm Kontakt mit den Autoren dieses Papers auf:
Niels Pfläging, niels.pflaeging@redforty2.com
Sebastian Kubsch, sebastian.kubsch@pradtke.de
Konzept und Gestaltung: Niels Pfläging
Deutsche Übersetzung: Peter Pröll





Die TOSD Website: timeoriented.dev

Anmerkung: Beim Schreiben dieses Papers wurde keine KI eingesetzt, abgesehen von der Nutzung zum Zweck von Literaturrecherche und Übersetzung.

Diese Arbeit ist an den BetaCodex angelehnt, eine frei verfügbaren Sozialtechnologie des BetaCodex Network, veröffentlicht unter der Creative-Commons-Lizenz CC-BY-SA, und abrufbar unter: betacodex.org/licence

Diese Arbeit ist an Time-Oriented Software Development (TOSD) angelehnt, eine frei verfügbare Sozialtechnologie von Niels Pfläging und Sebastian Kubsch, veröffentlicht unter der Creative-Commons-Lizenz CC-BY-SA, und abrufbar unter: redforty2.com/tosd

Es steht Dir frei, dieses Werk bzw. Teile dieses Werks professionell zu nutzen, unter der Voraussetzung, dass Du die Quelle konsistent und angemessen zitierst und/oder nennst.
Alle BetaCodex-Forschungspapiere sind unter betacodex.org/white-papers verfügbar.



Lern mehr über Zeitorientierte Softwareentwicklung
(Time-Oriented Software Development, TOSD): timeoriented.dev