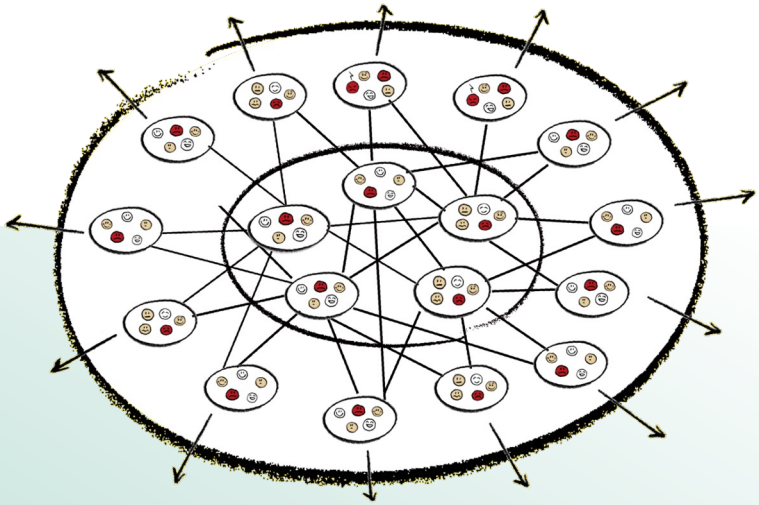
the betacodex network

since 2008



INTRODUCING **TIME-ORIENTED SOFTWARE DEVELOPMENT**

Bringing flow and the structure of time to Agile software development – beyond Scrum, Kanban and Lean

BetaCodex Network Associates
Niels Pflaeging | Sebastian Kubsch

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

26

Dedication

This BetaCodex research paper
is dedicated to the memory of

Ernst Weichselbaum

(13.04.1944 – 27.12.2024),

philosopher of time-orientation
and pioneer of Time-Oriented Work Systems

Time-Oriented Software Development (TOSD): Learn to use of the ultimate structure in work and organizations

Time: It is so ever-present in our lives we hardly notice it at all. Time is a powerful structure. But the concept of time has been grossly underappreciated and extremely underused in software development, and in organizations in general. Notions of *iteration* and *time-boxing* in Scrum or Agile did not nearly go far enough in making proper use of the *power of time* in software development. Today, in most teams and organizations, software development remains marred by myth, secrecy, hierarchy and obedience, often trapped in wasteful rituals, sloppy roles and bureaucratic processes.

Agile, sadly, hasn't made too much of a difference on this front. Since 2001, when the *Manifesto for Agile Software Development* was crafted, key debates within the movement have revolved around misconceptions, by and large: planning or no planning, how to run retrospectives, the roles of Product Owners/Scrum Masters/Agile Coaches, how to plan, how to scale, how to write user stories, how to evaluate user stories, and the like. We believe that these discussions (and others) have been misguided from the beginning and thus were destined to amount to little, or nothing. As a result, the Agile movement's traction and impetus for change has greatly suffered.

Today it is clear that *Agile* has produced little progress in how software development is organized. In this paper, we argue that *consistent time-orientation* is the best way to move beyond waterfall, "do-as-you-please", and laissez-faire software development. In software development, collective discipline, focus and flow cannot be produced through *scaling* or steering, feeble *roles* or supposed *values*, but that they must be tackled through *structures in time*. **Time is the ultimate structure – we just have to put time to good use if we want to propel software development forward.**

"From the perspective of the old, the new is always wrong. In the old way of thinking, nothing truly new is possible."
Ernst Weichselbaum

"If you want to change the world, start with the patterns in your head. Because there, nobody is standing in your way."
Ernst Weichselbaum

"It's better to design punctuality than to manage delays."
Ernst Weichselbaum

"If you need to decide, then you don't know what you really want. Beginning consists exclusively of beginning."
Ernst Weichselbaum

Time orientation:
It connects to our most inner needs as human beings

One of the key aims of all Time-oriented Work Systems (which we introduced in a separate research paper) is to devolve a *sense of accomplishment* to people at work. We are talking about the kind of feeling a farmer can tap into at the end of each day, when she can say to herself: *The day's deed is done.* We are talking about a *sense of accomplishment* in relation to the work performed – which is a fundamental human need.

A sense of daily (or frequent) accomplishment: that's a very different animal than abstract talk of *purpose, targets* or *north stars* – which is so common in today's command-and-control organizing. A sense of accomplishment *is grounded in the work itself* and in the individual realization that there is some closure to the work we have just performed.

Put differently, centrally and hierarchically-steered knowledge work, *waterfall* organizing, or *laissez-faire* software development deprives knowledge workers of their right to experience the frequent sense of relevance and closure they crave. The effects of such deprivation, and the systematic detachment from sensing accomplishment at work is dramatic. Time-orientation is the only way to restore the realization of *flow in knowledge work,* unity between one's work, and the benefit one's work brings to to other human beings. **That sense of accomplishment is at the heart of Time-Oriented Software Development.**

"'The day's work is done,' – an ancient saying of the rural population. A day's work has a beginning and an end, with the day in between. There is great emotional value in being done: *"I've done my part for today."* Completing the day's work gives those involved a sense of achievement. Someone on an assembly line does not have that: Mass production and automation prevent such experience. The socio-technical system breaks down."
Ernst Weichselbaum

"By adhering to seam agreements with others, people experience emotional added value that would be unattainable through hierarchical control."
Ernst Weichselbaum

"When the work is less monotonous, we increase the total amount of smiles."
Ernst Weichselbaum

Make time & flow the boss in software development – instead of having bosses stand in the way of flow

The intent of the early Agile software development scene was to supplant command-and-control management – referred to at the time, for lack of a better name, as *waterfall*. We know today that attempts to overcome the patterns of waterfall failed. Instead, command-and-control patterns among software development and tech-firms ended up being reaffirmed by ”agile” concepts, tools and scaling.

The problem appears to be that Agile approaches like Scrum, Kanban or Lean have been *insufficient* to break command-and-control patterns. Instead, the roles, rituals and patterns suggested in Agile concepts were highjacked, bent or deviated to serve hierarchical steering purposes. Overhead, and non-productive activities in software development have definitely increased, since the 1990s, and barriers to flow have increased as well, in spite of claims to the contrary. TOSD clearly needs to be different, if it is to make a difference and fulfil the promise of *agility*.

Consequently, TOSD breaks with most conventional waterfall patterns *and* Agile patterns: The notions of *development teams* and *scaling* for instance. Or the idea of sprints – which seem clunky and impractical, from the point of view of proper time-orientation. What makes TOSD less susceptible to coercive management is *principles*: Like the BetaCodex, which it is derived from, TOSD is based on meaningful principles (“seven laws”) that outline both *what time-orientation strives for*, and *what it is opposed to* – which are the patterns and design principles of command-and-control systems. On the next page, the seven principles of TOSD are outlined.
You will also find them in a more concise form, in the concept overview on page 23.

“Time orientation is a connector, not a divider. A bridge, not a wall. Time orientation is a prerequisite for organizational networks and self-organization.”
Ernst Weichselbaum

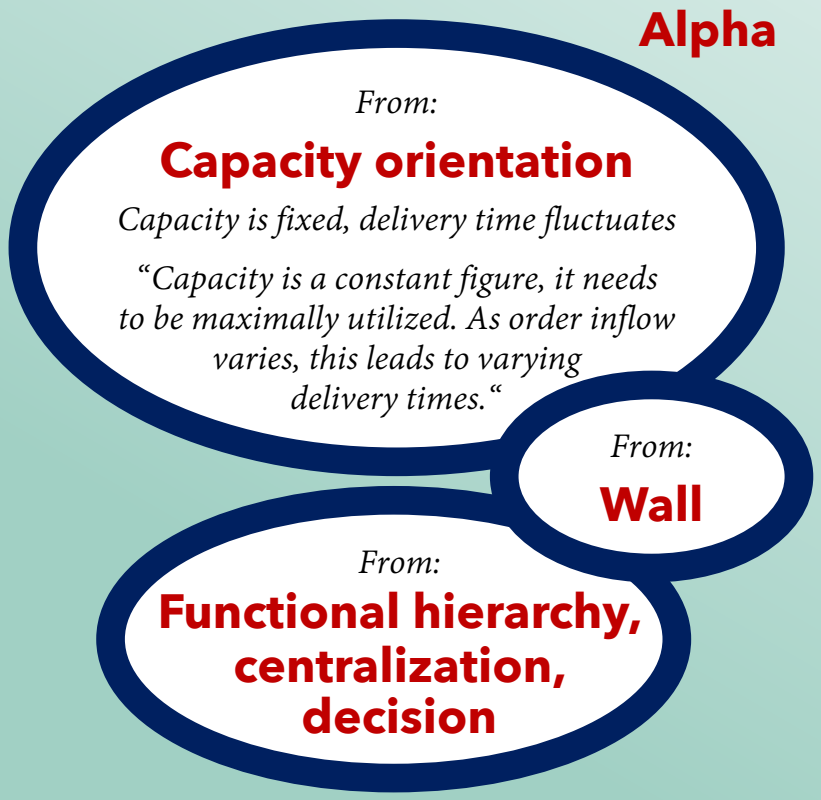
“If you need to steer, then you have yourself to blame. Set up the company well, don't steer it!”
Ernst Weichselbaum

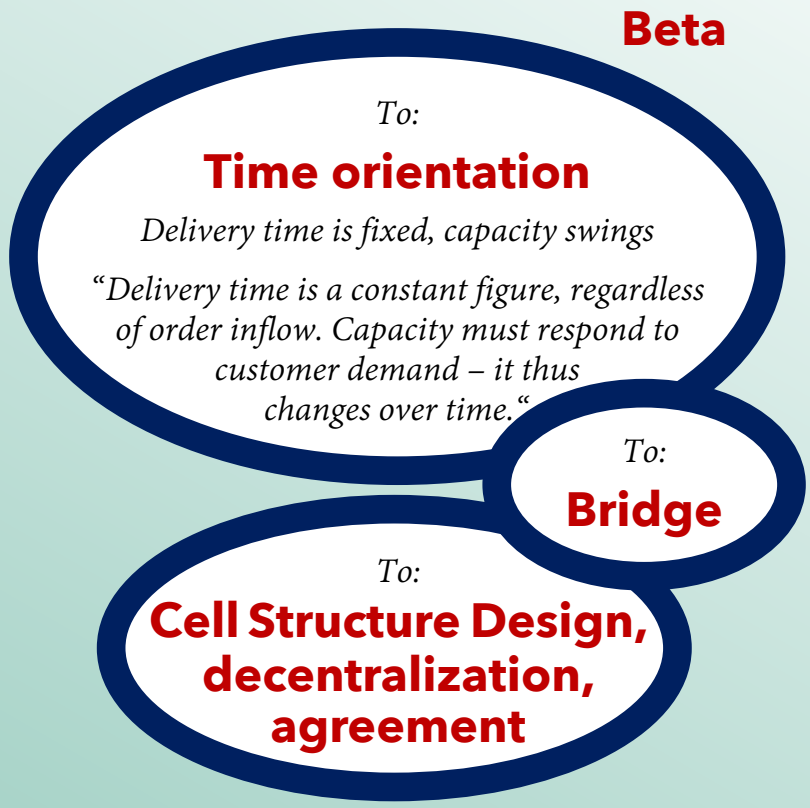
“Having an opinion about what's coming is valuable. Steering is not.”
Ernst Weichselbaum

“Where exceptions reign, no robust system can ever emerge.”
Ernst Weichselbaum

“Enforcement is a declaration of war: it only succeeds at the expense of others.”
Ernst Weichselbaum

In software development, time orientation is the prerequisite for consistent self-organization and collaboration





Thinking & language related to volumes:
Sprints, cadences, scaling, backlog, story points, batches, estimation, development teams, throughput,, capacity utilization, optimum production size, economies of scale, mergers & acquisitions, market share, minimum stock, utilization, target fulfillment, scheduling, forecasting, then, measures, PI/Big Room/Quarterly Planning

Thinking & language related to time:
Daily portion, OK Point, (adherence to) time slotting, fixed Realization time, optimum time, flow, rhythm, drumbeat, groove, fit, 'the day's work is done', white space, speed, desired date, anticipatory coordination, punctuality, agreed handovers, handshake, TTEO, now, seam agreements between actors and teams

The 7 key principles, or 'laws', of Time-Oriented Software Development

1. **Time orientation.** Different than capacity- or volume-orientation, time-orientation is a connector or a *bridge*: it enables and enforces self-organization.
2. **Flow and OK Point.** Flow is a key expression of time & motion in work and value creation, requiring the elimination of barriers between the work and the use of the work output (software, features) by clients. Creating conditions for flow has the effect of making problems in the systems of work become apparent by creating restraints.
3. **Transparency and List of Items.** Provides the means for self-control and self-steering. Should always refer to teams or groups of people – not individuals.
4. **Coherent self-organization.** It's the only way to unleash high engagement and high levels of motivation at work and is a precondition to high performance. It requires decentralization and doesn't combine well with steering and centralization.
5. **Timing is fixed, capacity swings.** Not the other way around.
6. **TTEO: Talk To Each Other.** *To have necessary conversations right here, right now* is the heart of self-organization and flow in social systems. Dialog, discourse and agreement must be at the center of contemporary software development. True conversation make rules, steering and centralization superfluous.
7. **Managerial consistency.** Without it, nothing works. It's what managers must provide, in order to put TOSD in place and make it persist.

"In time orientation, capacity is not scheduled (a.k.a. 'flexible'). Instead, capacity is designed as a system."
Ernst Weichselbaum

"Letting things flow is better than grouping them. A large [item] is nothing more than five smaller ones with coincidentally the same delivery address."
Ernst Weichselbaum

"Working hours are when there is work to be done. Thanks to the daily portion, every day becomes a season."
Ernst Weichselbaum

"If we don't put up any resistance to the flow towards customer demand, then value creation can swing."
Ernst Weichselbaum

"DIRFT. Do it right the first time."
Ernst Weichselbaum

Time-Oriented Software Development does not need this stuff

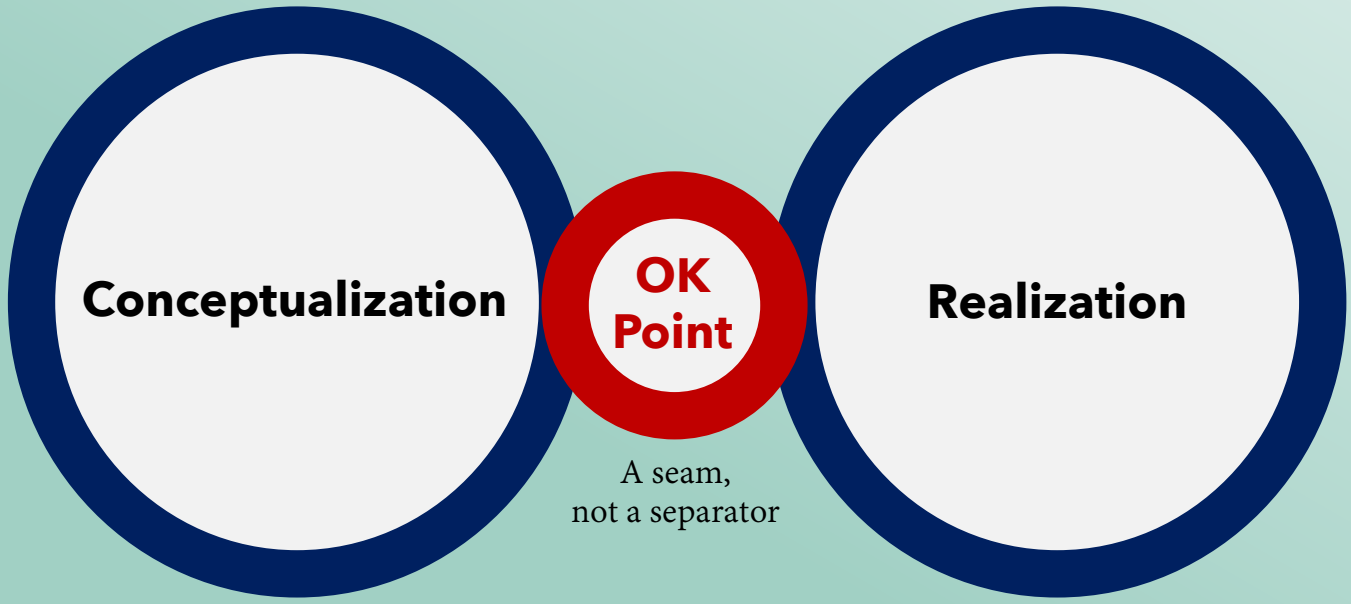
Software development artifacts to do away with, for good	What to do instead
Sprints/Cadences. A form of batch-orientation: Like trousers that are too baggy to use. Not actually flow-oriented, thus reducing speed and limiting self-organization	Time-box individual development items: It eliminates need for cadences
Iterations/MVP. Are Agile myths. Iterative product development is rare in software development. It does not constitute a distinctive case of software development work	Drop the idea and stop using the words
Backlogs. Seduce actors to have the wrong conversations about the wrong things, *like prioritization, sprint planning, power*. Backlogs erode clarification, quality conversation and discourse. Turning backlogs ever-evolving erodes the perception of accomplishment. Epitomizes everything that's wrong about Agile and Scrum	Develop an understanding of major undertakings to be accomplished within 30 to 90 days; improve conversation about focus and urgency
Scrum teams/Development teams/Scalable teams. To develop software, teams are not needed, per sé: Any well-sized software development job (”item”) is best accomplished by just one or two people, not a team. Enforcing existence of *development teams* is an impediment to flow. Overcoming the team illusion does away with dysfunctional roles like Scrum Master, Product Owner, Agile Coach	Software development is not accomplished within teams, but individually or within duos (“tandems”). Drop the notion/wording “software development team”
Estimation/Story points. Big topics in Agile. Misleading and wasteful ones, too. Estimates are dispensable in TOSD, as Realization is time-boxed and takes 3 days max.	Drop the idea and move to time-orientation and time-boxing, instead
Kanban boards. Visualization tools that accentuate flawed structures and flawed conversations. focused on the wrong things. If in digital form and separate from the development platform produces waste, steering and intransparency	Administer work lists/work items in Distributed Version Control System, Visualize the work in the physical space
Retrospectives. Better to make the process of work high on interaction (“Talk To Each Other!”) than „reviewing“ in fixed groups and in fixed, arhythmic intervals	Drop the idea and start having conversations at any time, when needed

Time-Oriented Software Development does not need this stuff (continued)

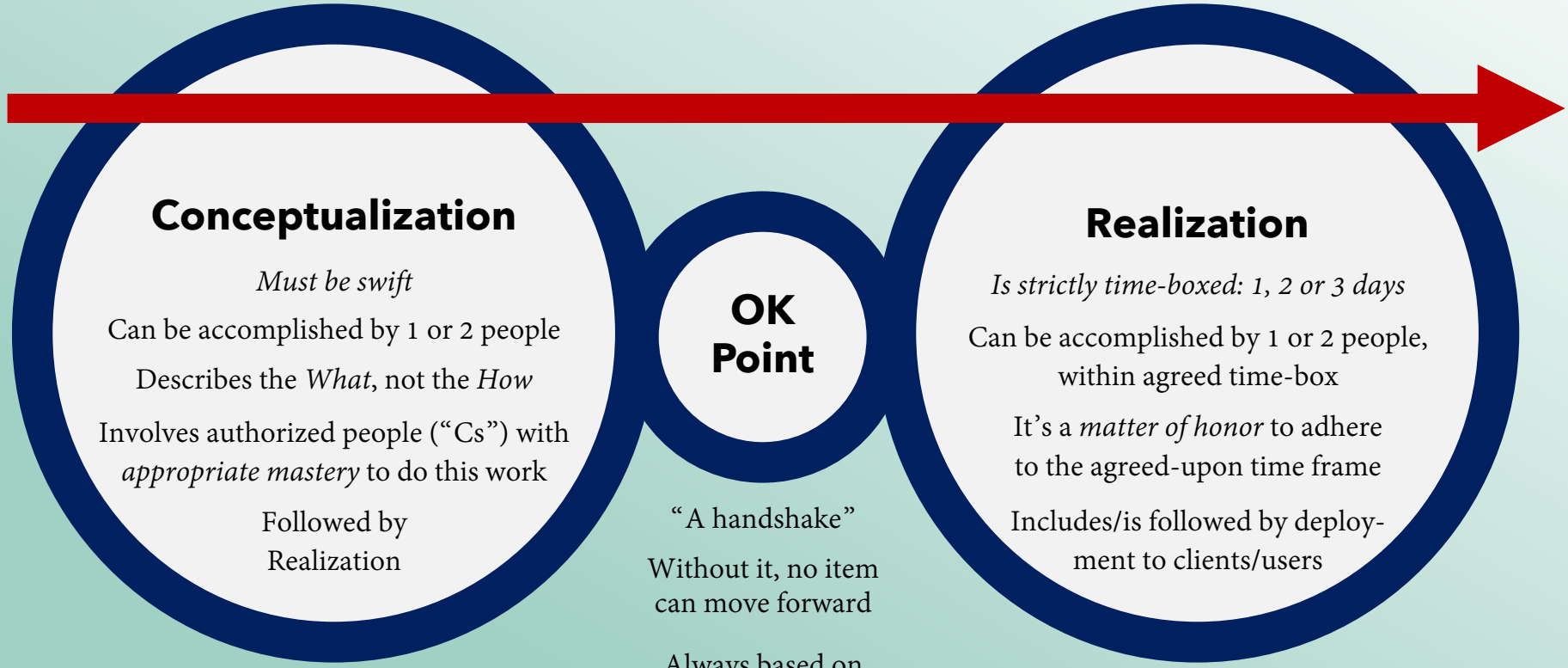
Software development artifacts to do away with, for good	What to do instead
Scaling frameworks/coordination. Based on flawed assumptions around steering and control of software development through "stacked teams", frameworks lead to hierarchization and bureaucracy, initiated the downfall of the Agile movement. Are natural enemies of TOSD	Just drop the idea and stop using the words
Planning, e.g. PI/Big Room/Quarterly Planning. Deviate from reality, work and now-orientation. Are the opposite of flow and time-orientation	Use OpenSpace DEV for discourse, problem-solving and creating unity
Agile measures, fixed targets. Eliminate them, as they obfuscate reality. restrain from setting/fixing targets for individuals, teams or groups	Use Relative Targets for team-based performance systems; Establish team P&Ls, apply judgement to the reality of flow
Capacity measures. Capacity should neither be measured nor debated, but changed and influenced instead. This requires action, not measures	Make capacity swing and fluctuate, according to demand, while fixing time-boxes
Lead/Scrum Master/Product Owner/Agile Coach roles. Disempower developers, produce steering and waste, create impediments to flow, tend to obfuscate problems	Apply Cell Structure Design principles; make teams small (4 to 8 people) and functionally integrated *beyond* software development
Approvals/Production Readiness reviews. Are impediments to flow	Eliminate them and stop using the word
User stories. An Agile myth or illusion. A poor substitute for group conversation with the client	Write proper concepts; visualize the flow of work in physical space, provide clarity
Agile maturity models, people categorizations, readiness measures. Unhelpful and judgmental about people, while acting on systems is required	Stop judging people, work the system, instead, abolish judging vocabulary
Funding of projects/Resource pools. Through budgets, approvals, allocations of developers ("pools")	Eliminate budgets, measure team P&Ls instead

A key innovation of Time-Oriented Software Development: The OK Point

Time-Oriented Software Development is based upon a distinction in time, which we call the OK Point.
The OK point allows us to distinguish between Conceptualization and Realization of software development work. It may seem odd, but it is also plausible that this distinction has been ignored by Scrum and Kanban, by and large. What is clear, however, is that without the OK Point distinguishing Conception from Realization, time-orientation in software development is inconceivable: Realization can be reliably time-boxed only when a concept for realization is produced in advance. The OK Point is where Conceptualization and Realization shake hands.



Time-Oriented Software Development: The OK Point gives the process structure. This allows Realization to be time-boxed



Conceptualization

Must be swift

Can be accomplished by 1 or 2 people

Describes the *What*, not the *How*

Involves authorized people ("Cs") with *appropriate mastery* to do this work

Followed by Realization

Leads to an OK Point-worthy concept for an item

Concepts are documented in writing and visible to all

Includes definitions of requirements, Realization time-box and people

Testing & documentation are part of the process, not separate (TDD)

OK Point

"A handshake"

Without it, no item can move forward

Always based on agreement or consultative decision-making, not mandating

Realization

Is strictly time-boxed: 1, 2 or 3 days

Can be accomplished by 1 or 2 people, within agreed time-box

It's a *matter of honor* to adhere to the agreed-upon time frame

Includes/is followed by deployment to clients/users

May involve different people than Conceptualization

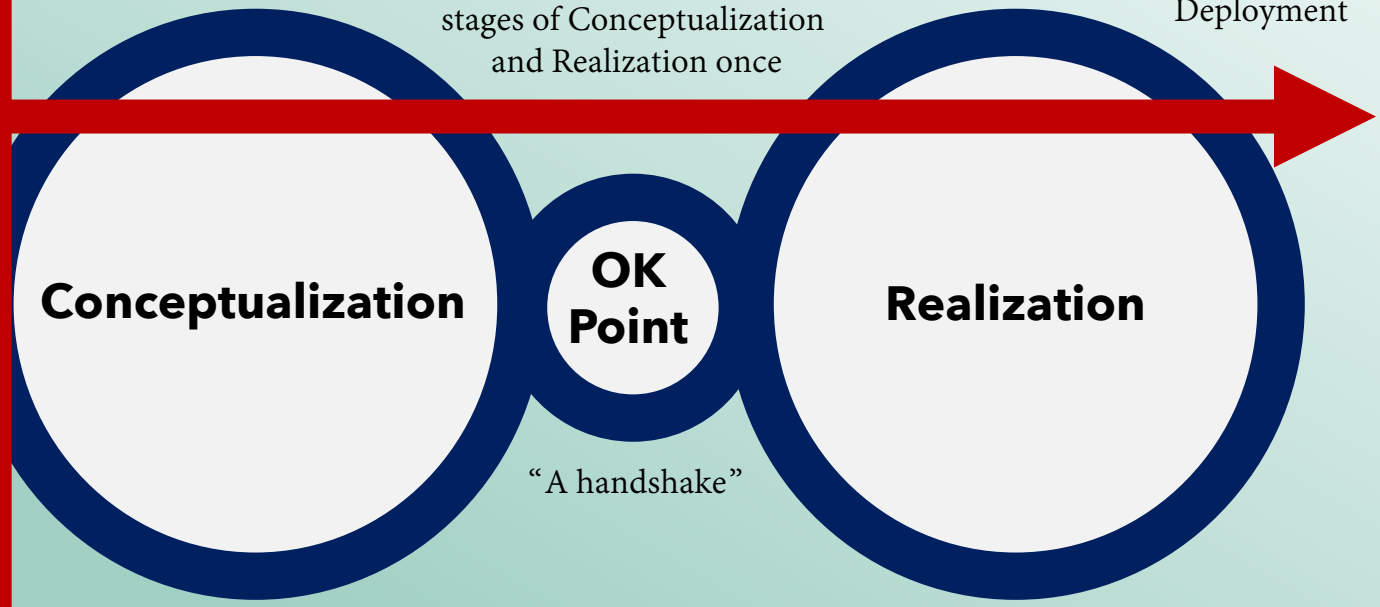
If there is doubt about timing, then you have not reached the OK Point yet

Testing & documentation are part of the process, not separate (TDD)

Time-Oriented Software Development: There is no simpler way to combine structure with self-organization, flow with discipline

List of Items

- Relates to a problem
- Contains all items to be worked upon
- Sequence matters
- There is a *list owner*
- Can be updated at any time
- Must at some point be completed, and a new list be started



The one TOSD key principle for problem-solving:

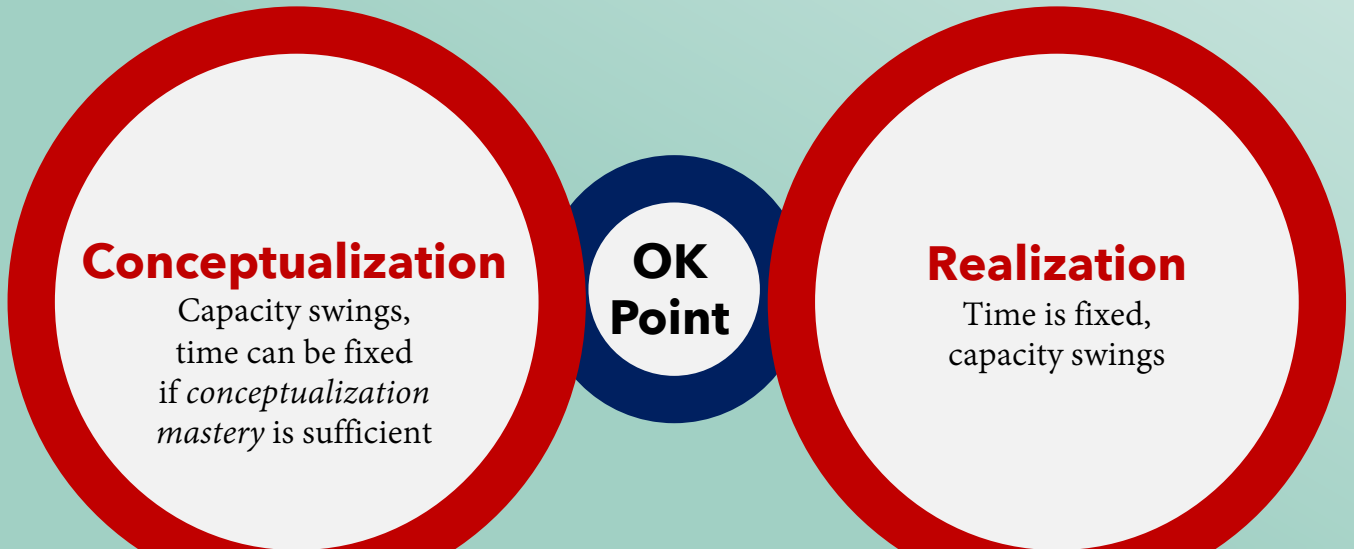
TTEO: Talk To Each Other

Conversation may occur at any time, not in writing, preferably face-to-face

In Time-Oriented Software Development, time is fixed while capacity swings

In command-and-control thinking about software development we take for granted that capacity is fixed. Capacity, we tend to believe, *equals* "working hours of the *human resources*" (a.k.a. software developers). If demand or difficulty of tasks fluctuate, then, accordingly, delivery time must vary and fluctuate accordingly. Flawed notions like this have led to many misconceptions about software development – that software development is an *art*, for example; or, at the other extreme, that it must be steered and controlled from above and constrained by budgets, steering and *people in charge.*

In TOSD, we take the opposite (and more realistic) stance. Delivery time should be fixed and capacity will naturally swing or fluctuate (which it does all the time). Here, we assume that software development is a craft (not an art), and that self-organization is natural in software development, as it is everywhere. The idea that capacity swings and time must be fixed to allow for flow, punctuality, effectiveness and human fulfillment was first articulated by Austrian Time-Oriented Work Systems pioneer Ernst Weichselbaum. His work with manufacturing companies was inspired by constructivism and the concept of fractal organizations. Ernst Weichselbaum also coined the concept of the OK Point, which he applied chiefly to the *seam* between client and supplier.



Friends of TOSD: Tools, concepts, and practices that go very well with time orientation in software development

selection, in alphabetical order

- Agentic AI
- Agentic IDEs
- Architecture as code
- Architecture Decision Records ADR
- Automatic Rollbacks
- Behavior Driven Development BDD
- BetaCodex, decentralization
- Boy Scout Rule
- Cell Structure Design
- Chaos as code
- Chaos Engineering
- Clean Code, e.g. 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 (e.g. 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
- Scaffolding
- Socio-Technical Systems
- Single Responsibility/Open-closed//Liskov substitution/ Interface segregation/Dependency inversion SOLID
- Static Code Analysis
- Story Boards
- Test-Driven Development TDD
- Vertical Slice Architecture
- Vibe Programming
- Workflow automation (e.g. Jenkins, GitHub Actions)

The case of Pradtke GmbH: How TOSD came about at Pradtke

Since the mid-1990s, Pradtke GmbH in Bochum/Germany has developed and commercialized software for organizations like hospitals and fire departments. After years of lackluster results, Pradtke successfully undertook a transformation to BetaCodex principles with the OpenSpace Beta approach during the 2nd half of 2024. In November 2024, as part of the 90 days of the OpenSpace Beta transformation process, the company went live with its decentralized Cell Structure Design with 8 periphery cells. Functional integration allowed substantial streamlining of administration and operations. **The company became noticeably leaner, becoming consistently profitable in the process, swiftly improving client satisfaction.** In the 1st semester of 2025, Pradtke successfully simplified and reformed its business model, bringing hundreds of corporate clients into a new model. The people at Pradtke would say: Between the Summer of 2024 and the Summer of 2025, we as a company put all those things into practice that we had previously tried and that we found impossible to accomplish.

In software development, things looked less rosy. The company was in the middle of a big migration project – working on transitioning its legacy desktop application to a Single Page Application. That meant change of medium, of technology, of tools and toys. The company was having a hard time trying to build up momentum for the undertaking, with software development following the rather common *Scrum but...* approach. There was a JIRA board, but everyone involved in software development struggled with the usual things. Overall, deployment remained far too slow, with flaky estimations, inefficient planning sessions, and superfluous rituals like retrospectives and reviews.

"Since its foundation, Pradtke made a journey that will seem familiar to most software firms: Accumulating growing technical debt and heaps of legacy code of which no-one knows what it does, living out 'not-invented-here' syndrome while trying to balance long-term architectural goals against short-term client needs. We were juggling our 'limited' developer resources in the hope of getting the most out of them. On the way, we tried stuff like waterfall, Scrum, JIRA, estimations and OKRs – approaches that at no point did for us what we hoped they would. We weren't getting faster and the software didn't get much better. With hindsight, that's because we didn't really understand what the underlying problems were. We hoped we could 'fix' problems if we just threw the right processes, frameworks and models at them."
Sebastian Kubsch

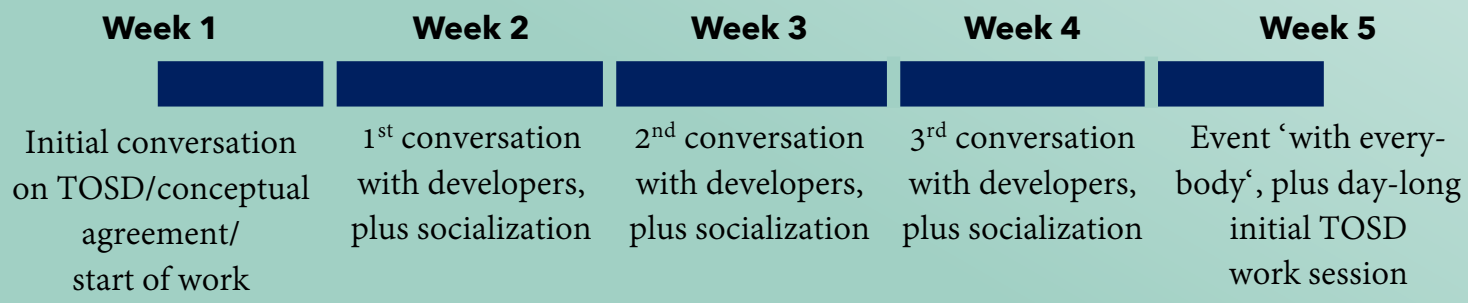
The case of Pradtke GmbH: Going live with time orientation in 4 weeks

Led by the CTO and together with general managers Marc Otten and André Pradtke, the company began the move towards TOSD in August 2025. While Pradtke was deploying TOSD, the approach was fleshed out further, beyond the initial concepts such as the *OK Point*, the *List of Items* and *fixed realization time*. Deployment took place in sync with the company's 14 software developers and 10 *potential* software developers. There were some valid questions: **How to get going in TOSD with *all* developers at once?** How to make the ongoing work transparent? How to accelerate the work, without necessarily having developers work more hours? How to redistribute operational duties within the business cells in the periphery, so that developers that are part of those teams could focus sufficiently on the joint List of Items? How to conceive the work as *items* that take just one to three days to realize?

Following the conceptual agreement in the 1st week, over the course of three weeks there were three rounds of 3-hour conversations with the software. In the following week, the Go Live took place during a first TOSD work session for all developers. Everyone in the company was invited to that day's joint opening session.

"With the new Cell Structure Design, most teams quickly found fulfillment in self-organization and meaningful, direct action, free of artificial hierarchies and budgets. Things were moving along well. The one thing we did not see improve was the software development process."
Sebastian Kubsch

"For a software company like ours that has existed since the 1990s, the challenge was getting rid of three decades of bad habits in four weeks."
Sebastian Kubsch



The case of Pradtke GmbH: Ks and Rs – the first steps to Time-Oriented Software Development

The first step was to identify and nominate the software developers and experts from within the company who would be authorized to develop concepts, or requirements. Software quality and flow would depend heavily on the quality of the underlying concepts for each item. Clearly, this group of *experts* had to be *small and cohesive*. The Cell Structure was reviewed, identifying those best suited to become *Ks* (for *conceptualizers,* or *Konzeptionierer* in German). In addition to mastery, selection criteria for individuals of this group was a forthcoming attitude towards interaction with others and availability for conversation with everyone. Within the company, this group is now consistently referred to as *Ks*. The next step was to identify those who could add value as Realizers. This became the group of *Rs*. As to be expected, this group of individuals was highly diverse and wide-ranging in their respective skill-sets, mastery and levels of professional maturity. Some *Ks* were also identified as *Rs:* Naturally, *Ks can also be Rs.*

With this, the initial TOSD setup was complete. The CTO become owner of *The List. Ks* create concepts for *Rs*. *Rs* realize these concepts within agreed-upon time-boxes. OK Point-ready concepts must be *no-further-questions-required,* in principle. The OK Point demands agreement, and a *handshake*, between Ks and Rs. Very importantly: **Conversation and consultation *among everyone* can *happen at all times.***

In the following days, a 3-hour workshop with the *Ks* was set up. Then two 3-hour conversation sessions with all the *Rs*. During each session, idea and concept of TOSD were laid out *from beginning to end, starting with the business need*. Things were not forced, but genuinely discussed. Feedback was great, and there was constructive clarification. Some ideas that arose during conversations were integrated into the overall TOSD concept on the spot. External facilitation helped to run these workshops smoothly. It was agreed that the list for the migration, which contained several hundred items, could be cleared away within around 8 weeks. Now everyone was on board and eager to start.

"In August 2025, Niels Pflaeging approached us with an alternative: Time-orientation. It sounded simple. Do away with all the cadence-based nonsense. Drop the idea of sprints. Throw out unnecessary roles, like Product Owner and Scrum Master. And, above all: stop trying to create flow by altering delivery times. Instead, we would fix delivery times and let capacity swing. The moment Niels discussed his proposition with me and one of our general managers, I was hooked. Hooked and left wondering: Could it be that simple to get to flow, after all?"
Sebastian Kubsch

"In TOSD, there is no estimation game. Nobody plays estimation poker. The item must be so precise that you KNOW if it will take 1, 2 or 3 days to realize. For Rs, to deliver within this window of time (not a day later, or two), is a matter of *honor*."
Sebastian Kubsch

The case of Pradtke GmbH: Improving effectiveness through time orientation, week by week

The 1st week of operating in TOSD saw some struggles with the concept, which was expected. Concepts for some items were still not precise enough, so questions arose during realization. Some Realizers underestimated the time they'd need. Ks were producing concepts faster than they were realized, so concepts began to pile up at the OK Point. All in all, the issues could mostly be handled in an easy way: by having better conversations more often. Ks talked about how to ramp up (or down) the grade of detail in concepts. Rs began pairing up on specific tasks, orienting themselves on already ready-to-realize items. To reach more steady flow, items in The List needed to be reprioritized as well. Some developers were re-designated K or R, and conversations were held to focus on the R side of things.

The 2nd week already brought noticeable performance gains. Throughput doubled, compared to the week before. People started to get accustomed to the new way of doing things, and some patterns began to settle in. Momentum was building.

During the 3rd week, a breakthrough occurred. Rs began to realize that things were working out in TOSD. Everyone could see that good progress was made on delivering items to clients. **By week four, almost as many items had been realized as in the six months before.** Flow began to emerge, but challenges remained. The flow ratio of *Concept to Realization* required constant attention. Capacity wasn't swinging much yet in Conceptualization or Realization. Some concepts failed to hold up, due to lack of practice and insufficient work standards in Conceptualization. Consequently, some standards were agreed upon among Ks. **This is a game changer in TOSD: Problems are solved on the spot.** Hiccups are dealt with right away; immediacy of the process allows for focusing on what's important. Instead of sitting through ritualized planning meetings for hours or days, TOSD compels one to take immediate action, right then.

"Ks pick items from The List, focusing on the ones with the highest priority and those, ideally, that fall into their individual domains of mastery. K's do the necessary brain- and footwork for the Realizers to give them a clear grasp of WHAT needs to be done. Without telling them HOW to do it. Interdependencies, unexpected system behavior and other quirks need to be described. Sometimes it's necessary to dig in the underlying database model or the legacy code. The goal is clear: To allow Rs to implement the item as fast as possible, by gathering the necessary information up front. Ks must identify and approach the right Rs for the job. Rs can ask Ks for items, and are expected to do so, especially when they have nothing to do."
Sebastian Kubsch

"At the moment we are using a simple GitHub Project to track items. This seems to work quite well for us. Don't overdo it, keep things plain. Don't build the next backlog."
Sebastian Kubsch

The case of Pradtke GmbH: Contemplations on mastering time orientation

If we want to achieve higher effectiveness in software companies and development teams, we should ask ourselves: *What's the biggest waste in software development?* We are convinced it is all too often a combination of hesitation, stalling and perceived loneliness, or *learned helplessness* among developers. **Most software development teams seem to gravitate towards stop-and-go, and partial inactivity.** Not because they want to, but because the contexts made of of common management methods make a stop-and-go working modality inevitable. Meetings, routines, rituals, coercion, need for hierarchical approvals and siloed organization prevent a sense of self-organization. Flow is disrupted and personal fulfillment at work is reduced, severely limiting output. For years, this was true for software development at Pradtke. We concluded that the resulting waste of development time and lack of collaboration is removable only by disrupting common management patterns (including *Agile* ones) and by overcoming the conventional system of software development. Combined, decentralization and TOSD provide the structures to disrupt limiting and wasteful patterns.

Then there is the overall question of *capacity*: How can it be raised? How can capacity be brought to swing with demand? The hasty answer to the 1st part of the question is that short-term hiring is the easiest, or perhaps the *only* option to increase capacity. But a bias for hiring would usually be a mistake. It is often neither feasible nor necessary to hire, or to add personnel in the short-term. Instead, it is helpful to consider that *capacity swings both ways.* There will be times when capacity is higher, and times when capacity is lower. *The main lever to raise capacity is not hiring, but much better collaboration,* liberating people from the burden of approvals, waiting times, pointless meetings and needless work-arounds. By making mutual help, consultation and pair programming natural. All of which are all steps taken at Pradtke. Also, to be fair: working a bit more, at certain points of time, when demand is high, is also needed to achieve flow. If the work itself is perceived as fulfilling and not coercive, within a system that allows for flow, then capacity can swing.

"Deliberately start out with easier items of lower complexity, so that everyone can focus on learning the method, instead of struggling with contents at first. This helps bringing things up to speed."
Sebastian Kubsch

"By week three a veteran developer approached me. He told me how happy he was with the new model: That finally things had started to move again, that he was relieved we were finally working on items we had been talking about for years, but hadn't tackled before. He showed me a feature he had just finalized, and that he was visibly proud of. I knew then that we were on the right track."
Sebastian Kubsch

The case of Pradtke GmbH: Contemplations on mastering time orientation (continued)

The third aspect that becomes apparent in time orientation is the importance of quality interaction among developers. Lack of quality interaction tends to reduce output and cripples overall software quality. Facilitating and encouraging interaction is one thing, but good communication can also be *structurally favored and provoked*. TOSD triggers constant interaction structurally. With regards The List of Items, the individual items and sequence of items on it; around constellations, or pairings in Conceptualization and Realization; during Conceptualization, at the OK Point and during Realization. Many interventions are possible in order for better conversations to happen: We believe that increasing face-to-face encounters among software develop-pers plays a crucial role in this regard. During the first weeks of time-oriented operating at Pradtke, interventions to increase face-to-face interaction included making *office lunch* available every day, agreement among developers about *shifting time from working from home to working from the office,* and instituting voluntary *work & learning* sessions on Saturdays.

In this context, the TOSD key principle 'TTEO: *Talk To Each Other*' deserves special mention. Talking to each other immediately when problems arise is better than defining rules, running committees or periodic meetings, or installing steering and supervisor roles. Opting for immediate, open conversation is a muscle that needs to be trained by everyone especially when, like at Pradtke, that muscle hadn't been well-used, for years. The idea behind the *Talk to each other* principle is to make delaying conversation until some meeting happens unacceptable, and to make solving problems and impediments "on the spot" the norm, at the very moment a problem emerges. Not all problems can be solved on the spot, for sure. What cannot be solved ad-hoc may lead to new items to be added to The List, or maybe to The List that will be tackled next. In software development, the Talk to each other principle is inseparable from time orientation, and a powerful lever to bring about flow and new patterns of work.

"In TOSD, to delay a conversation about a problem we just encountered would be frivolous and absurd."
Sebastian Kubsch

"Authorization is a key aspect of time orientation. People were used to wait for legitimation before taking action. That made things slow. We made it very clear from the beginning with TOSD that everyone was authorized to take action, because we trust them, because we knew they would do things well. And then they did. Because they wanted to, not because of some kind of command."
Sebastian Kubsch

"Choose the right people. Sloppy selection of Ks will yield poor results. Keep evaluating if the people you chose as Ks are the right ones for the work to be done. It's natural that requirements will shift over time."
Sebastian Kubsch

Strengths of Time-Oriented Software Development: A summary

1. **Can be started at any time.** Ramp-up requires no more than a few weeks, and few external resources. Can also be combined with the OpenSpace Beta approach to transformation for entire organizations, which takes 90 days.
2. **Is extremely straightforward:** Requires no specific training to get started – only agreement on process and principles. Immediately eliminates need for excessive overhead, steering, meetings and behavioral control.
3. **Immediately produces flow and time orientation,** which are enablers for self-organization and collaboration, which in turn are levers of effectiveness and performance.
4. **Produces focus and discipline, without need for hierarchy or bureaucracy,** through The List (and the *Keeper of the List and Protector of Flow*), through the OK Point and the absence of rules, roles of steering, or levers for meddling by management.
5. **Accelerates both Conceptualization and Realization** through the distinction between two different masteries that matter to effective software development.
6. **Scales naturally** to hundreds or thousands of people; does not rely on "development teams". Allows for several lists of items ("initiatives") to co-exist.
7. **Dramatically increases the quality of interactions** through immediacy of conversation.
8. **Reintegrates conversation into the process of software development,** instead of confining discourse to separate rituals or meetings. Conversation becomes part of the work again.
9. **Can be easily combined with further socio-tech for agility and self-organization,** e.g. large-group interaction formats (OpenSpace DEV), approaches to decentralized structure (Cell Structure Design) and relative performance systems (Relative Targets).

"The OK Point is the daily 'big bang' of [software item] processing. Since the hand-over is highly ritualized, the OK Point is something like the company's heartbeat."
Ernst Weichselbaum

"Letting things flow is better than bundling them."
Ernst Weichselbaum

"Design the process in a way that it does not need to be monitored. It's better to be punctual than to administer delays."
Ernst Weichselbaum

"Authority is not a property of individuals but is generated through interaction between at least two people. Authority comes from agreement."
Ernst Weichselbaum

Time-Oriented Software Development:
About the open source license Attribution ShareAlike, 'CC-BY-SA'

This license lets you use, commercialize, remix, tweak, and build upon Time-Oriented Software Development. In so doing, you agree to three simple things:

- credit the original authors, Niels Pflaeging and Sebastian Kubsch,
- and provide our specified link to the source material, as listed below,
- and license your derivative creations to others under the identical terms.



Specifically, you must provide and prominently display the following sentence and link with all original or derived works, and included as part of all related texts or graphics you might develop:

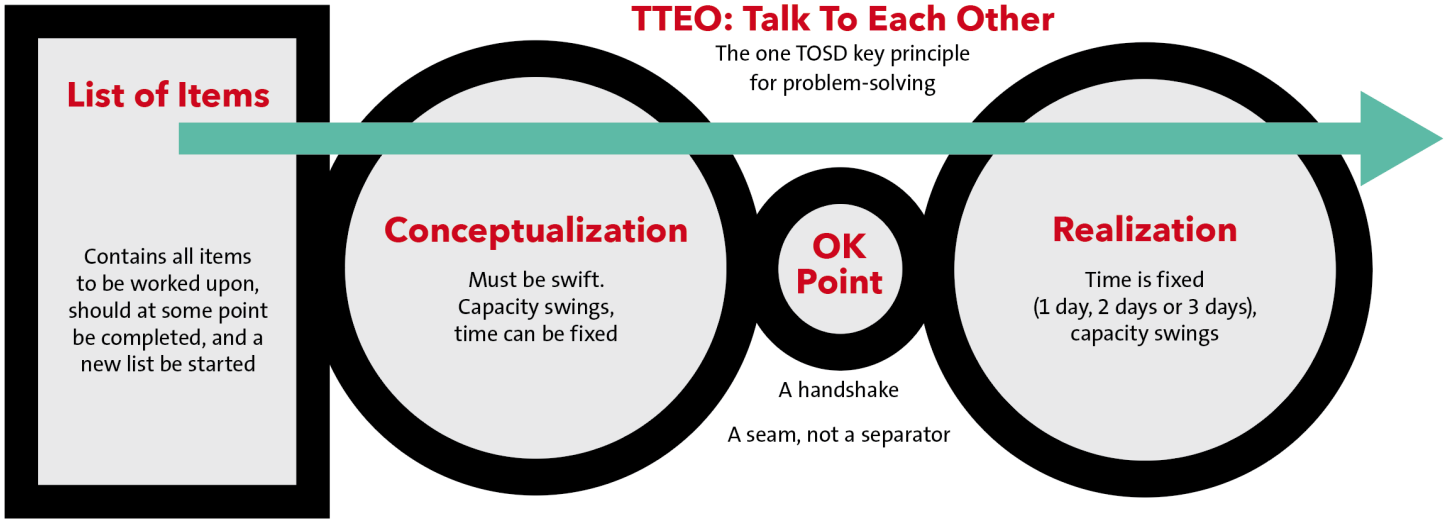
> "This work is derived from *Time-Oriented Software Development*, an open source, free social technology by Niels Pflaeging and Sebastian Kubsch, published under the CC-BY-SA-4.0 license from Creative Commons, and found here: www.redforty2.com/tosd"

Please also consider the other open source social technologies created by the BetaCodex movement, since 2008: BetaCodex, Cell Structure Design, Relative Targets, OpenSpace Beta and OpenSpace DEV.

Time-Oriented Software Development

Concept overview | An open source social technology by Niels Pflaeging and Sebastian Kubsch

Visual overview



7 laws, or principles of Time-Oriented Software Development (TOSD)

01. **Time-orientation** – not volume-/capacity-orientation
02. **Flow with OK Point –** not dabbling, centralized steering and batch organization
03. **Transparency and joint List of Items –** not hierarchical control, budget thinking, fixed targets
04. **Coherent self-organization –** not scaling, or centralized scheduling and coordination
05. **Timing is fixed, capacity swings –** not the other way around
06. **TTEO: Talk To Each Other –** not uniformity, rules or commands
07. **Managerial consistency –** not bureaucracy, inconsistency and lack of consequence

12 laws, or principles of the BetaCodex

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

Law	Do this! (Beta)	Not that! (Alpha)
01. **Team autonomy**	Connectedness with purpose,	not dependency
02. **Federalization**	Integration into cells,	not division into silos
03. **Leaderships**	Self-organization,	not management
04. **All-around success**	Comprehensive fitness,	not mono-maximization
05. **Transparency**	Flow intelligence,	not power obstruction
06. **Market orientation**	Relative Targets,	not top-down prescription
07. **Conditional income**	Participation,	not incentives
08. **Presence of mind**	Preparation,	not planned economy
09. **Rhythm**	Tempo & groove,	not fiscal-year orientation
10. **Mastery-based decision**	Consequence,	not bureaucracy
11. **Resource discipline**	Expedience,	not status-orientation
12. **Flow coordination**	Value-creation dynamics,	not static allocations

Time-Oriented Software Development (TOSD) has three components: The 12 principles ("laws") of the BetaCodex, 7 principles ("laws") of TOSD, and the Visual Overview | Time-Oriented Software Development by Niels Pflaeging & Sebastian Kubsch | version 2025 | Published under the open source license CC-BY-SA-4.0 | www.redforty2.com/tosd

Sources and recommended reading

Time and organizing

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*

Time-Oriented Work Systems and decentralization

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*

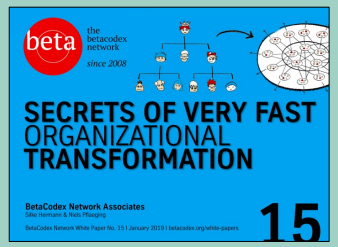
Pflaeging, Niels/Hermann, Silke (2023): *The laws of the BetaCodex*

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

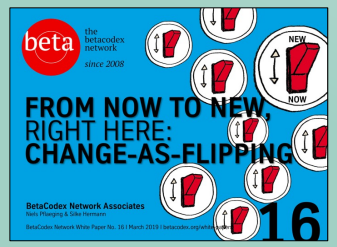
Pflaeging, Niels/Hermann, Silke (2024): *Cell Structure Design Patterns, II*

Weichselbaum, Ernst/Pflaeging, Niels (ed.) (2020): *In jedem Unternehmen steckt ein besseres* (German only)

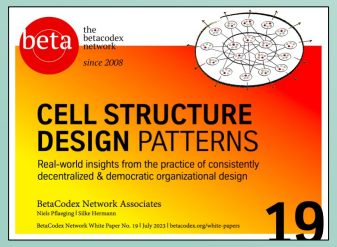
Related BetaCodex Network research papers, available on www.betacodex.org



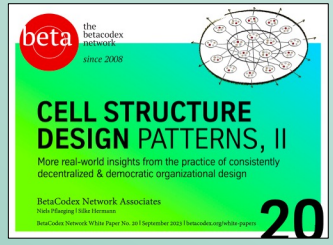
Secrets of Very Fast Organizational Transf., 2019



From Now to New, Right Here, 2019



Cell Structure Design Patterns, I, 2023



Cell Structure Design Patterns, II, 2024



Slave to the Rhythm, 2025

The BetaCodex Network research papers

Read them free of charge on betacodex.org/white-papers

the betacodex network since 2008



Related books from BetaCodex Press

beta codex press

160+ quotes straight from the fearless pioneer of time-orientation, Toyota Production System architect Taiichi Ohno (1912-1990)



What would Ohno-san do?
Edited by Niels Pflaeging/ Stefan Schmeing
BetaCodex Press, 2025
Also available as an E-book

Condensed wisdom from the original philosopher-practitioner of time-orientation: W. Edwards Deming (1900-1993)



What would Deming do?
Edited by Niels Pflaeging
BetaCodex Press, 2023
Also available as an E-book

Concepts and patterns of consistent self-organization and time-orientation, outlined by the BetaCodex founder



Essays on Beta, Vol. 1
Niels Pflaeging
BetaCodex Press, 2020
Also available as an E-book
Also available in German

How to make transformational change happen, in a time-oriented way? Silke Hermann and Niels Pflaeging deliver both insights and specific concepts



OpenSpace Beta
Silke Hermann/Niels Pflaeging
3rd ed., BetaCodex Press, 2023
Also available as an E-book
Also available in German and Korean

Visit the shop at redforty2.com/shop

the
betacodex
network

since 2008

Get in touch with the authors of this paper:
Niels Pflaeging, niels.pflaeging@redforty2.com
Sebastian Kubsch, sebastian.kubsch@pradtke.de
Paper concept and design: Niels Pflaeging

The TOSD website: timeoriented.dev





Disclaimer: No A.I. was used in the process of writing this paper, except for literature research and translation purposes.

This work is derived from the BetaCodex, an open source, free social technology by the BetaCodex Network, published under the CC-BY-SA license from Creative Commons, and found here: betacodex.org/licence

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

You are free to use this work, or parts of it, under the condition that you quote and/or mention the source consistently and appropriately. All BetaCodex research papers are available under betacodex.org/white-papers



Learn more about Time-Oriented Software Development:
timeoriented.dev