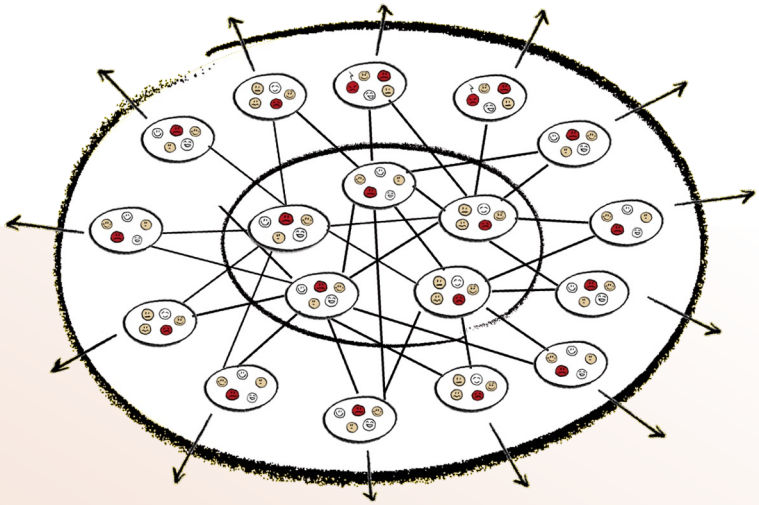
the betacodex network

since 2008



PATTERNS OF **TIME-ORIENTED SOFTWARE DEVELOPMENT**

Getting to rhythm, flow, swiftness and consistently high productivity, in an age of AI-aided work

BetaCodex Network Associates
Niels Pflaeging

BetaCodex Network White Paper No. 27 | April 2026 | betacodex.org/white-papers

27

Dedication

This BetaCodex research paper
is dedicated to the memory of

Taiichi Ohno

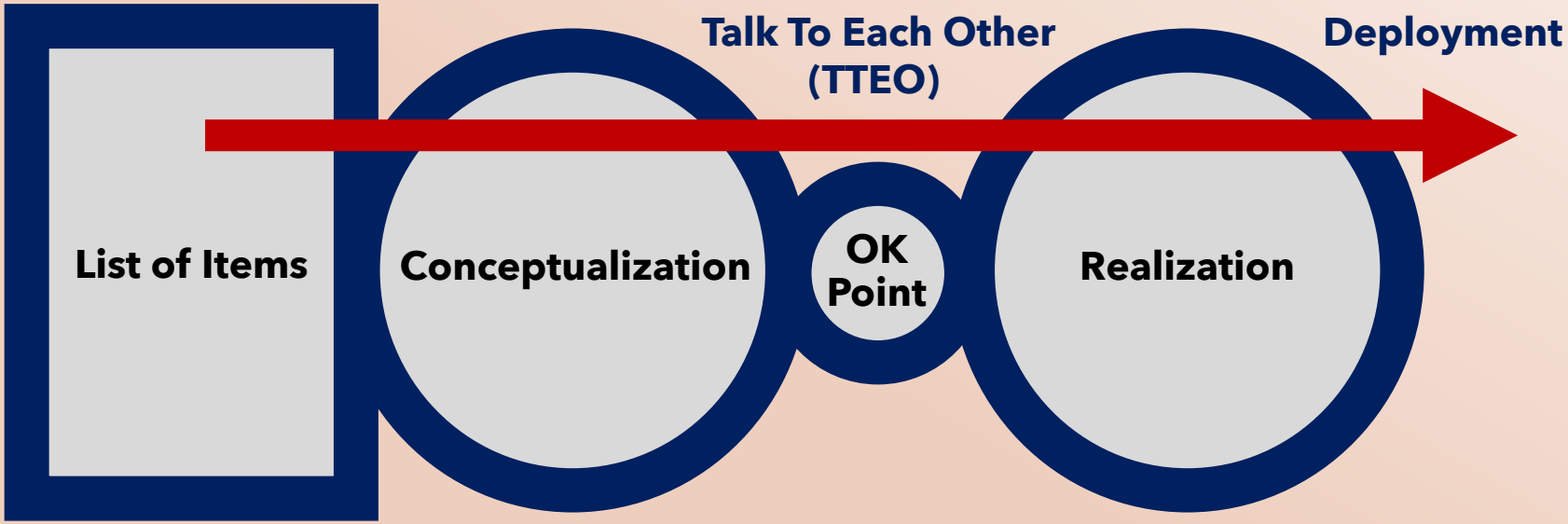
(29.02.1912, Dalian – 28.05.1990, Toyota City),

the daring pioneer of time-orientation who dedicated
three decades of his life to bringing
flow to the Toyota Motor Company ecosystem

This paper covers 18 patterns we discovered since making the TOSD approach available to the world, in late 2025

Reading our original research paper on Time-Oriented Software Development (TOSD), you may have noticed that, in TOSD, nothing is optional. The *OK Point,* the *TTEO principle*, *List curation*, replacing teams and support roles with *the distinction between Cs and Rs* – no element of this system (illustrated below) can be omitted without making it fall apart. TOSD is minimalistic. Accordingly, the 18 patterns described in this paper are not *additions* to TOSD, or somewhat *new features* of time orientation in product development. Instead, the patterns are part of the *fine print* that will make it easier for you to make TOSD work well in practice, or to bring about a new TOSD system, fast, regardless of the number of people involved in your system of software development.

The patterns in this paper cover aspects of TOSD we were not fully aware of when we published the first research paper on TOSD, in late 2025. Some are patterns we wanted to see confirmed in practice first. We discovered these patterns through work with pioneering companies, including Pradtke, extensive research into software development literature, and by concepts of time orientation like those outlined in the new book *What would Ernst Weichselbaum do?*



Patterns of Time-Oriented Software Development: An overview of this paper's topics

1. The **OK Point** is where handshakes happen in the morning
2. For Realization, the **Daily Portion** is the perfect batch size
3. A **List of Items** is like beautiful long hair: It needs a lot of combing
4. **Conceptualization** should also be time-boxed
5. How big is your organization's ***List Drying Rack*?**
6. **Strategy** must be integrated into the TOSD system. It cannot exist in isolation
7. The answer to the **How? of AI-aided development** is Who?
8. AI makes **(critical) thinking** more important, not less
9. **Let capacity swing**. No system of time orientation will last long without this
10. **Social density** within the group is good for high spirits & high performance
11. **Immediacy matters,** managerial consistency is critical!
12. There are two kinds of **White Space**: one is an enemy, the other is desirable
13. With TOSD, **sourcing development capacity externally** is a piece of cake
14. **Estimation** is a slippery slope
15. Address the **distinctions** between decisions & agreements, positional power & mastery
16. Moving to a TOSD system means **abandoning development teams**
17. Tackle **Organizational Hygiene** early
18. Adopt the **language of time orientation**, right from the start



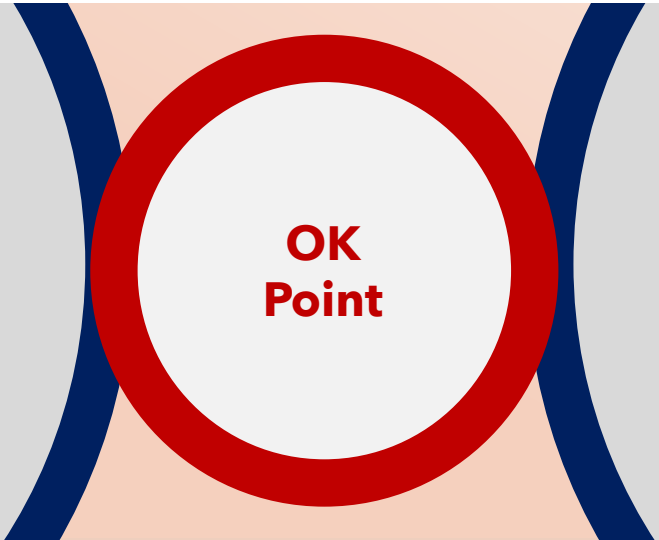
Also read BetaCodex Network research paper No. 26, Introducing Time-Oriented Software Development, published October 2025

Fore more conceptual information on Time-Oriented Software Development (TOSD), visit the web page www.timeoriented.dev

1. The *OK Point* is where handshakes happen in the morning

The OK Point is a brief daily gathering where handshakes happens between Conceptualizers and Realizers. Where transparency is produced, agreements made and Daily Portions set. Flow imbalances can be addressed, too, but they are better addressed ahead of the OK Point. At Pradtke, it seemed for a while that Realizers struggled with developing the discipline to take on new items daily, just by themselves. CTO Sebastian Kubsch felt he had to intervene and personally follow-up too much. Until, at some point, he established that the OK Point would happen every morning at 9.00 am, lasting up to half an hour. Cs and Rs would be present, and Rs would pick items to realize right then, right there, having the necessary conversations with the item's Conceptualizer(s), and shaking hands with them. Every Realizer would walk away with an item, reliably. This immediately led to easier oversight, and to higher commitment among Realizers to take on an item at 9.00am, on every day when they would take on a Daily Portion related to development (a.k.a. a new item).

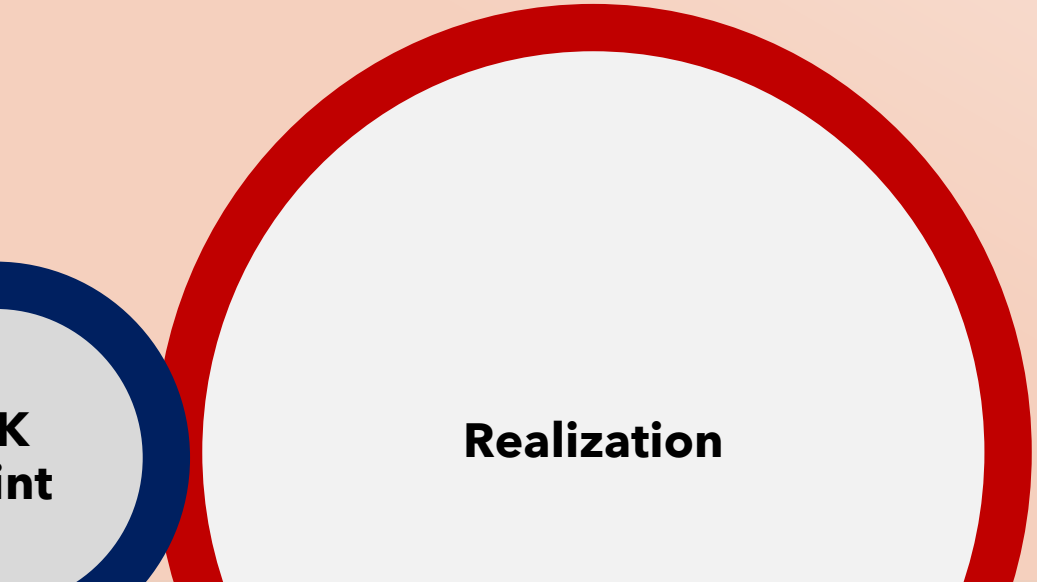
When the work consists of Daily Portions, then every Realizer will decide every day if she or he will dedicate their Daily Portion to software development, or to something else. At Pradtke, with its decentralized Cell Structure Design, developers are part of business cells in the periphery. Accordingly, this "something else" would be performing other client, or any kind of business-related work for the cell.



2. For Realization, the *Daily Portion* is the perfect batch size

Time orientation doesn't mean "faster", "as fast as possible", or "faster because of AI". Instead, time orientation establishes steady, reliable, even flow within a daily rhythm. In Realization, this means sizing the work items in a way that they are turned into Daily Portions. While one-day portions are ideal,. List items can be made up of 2 or 3 daily portions, according to required and available mastery, known interdependencies and novelty of technology or items. Avoid portions of less than 1 day, though, as this will reintroduce the need for scheduling and centralized coordination.

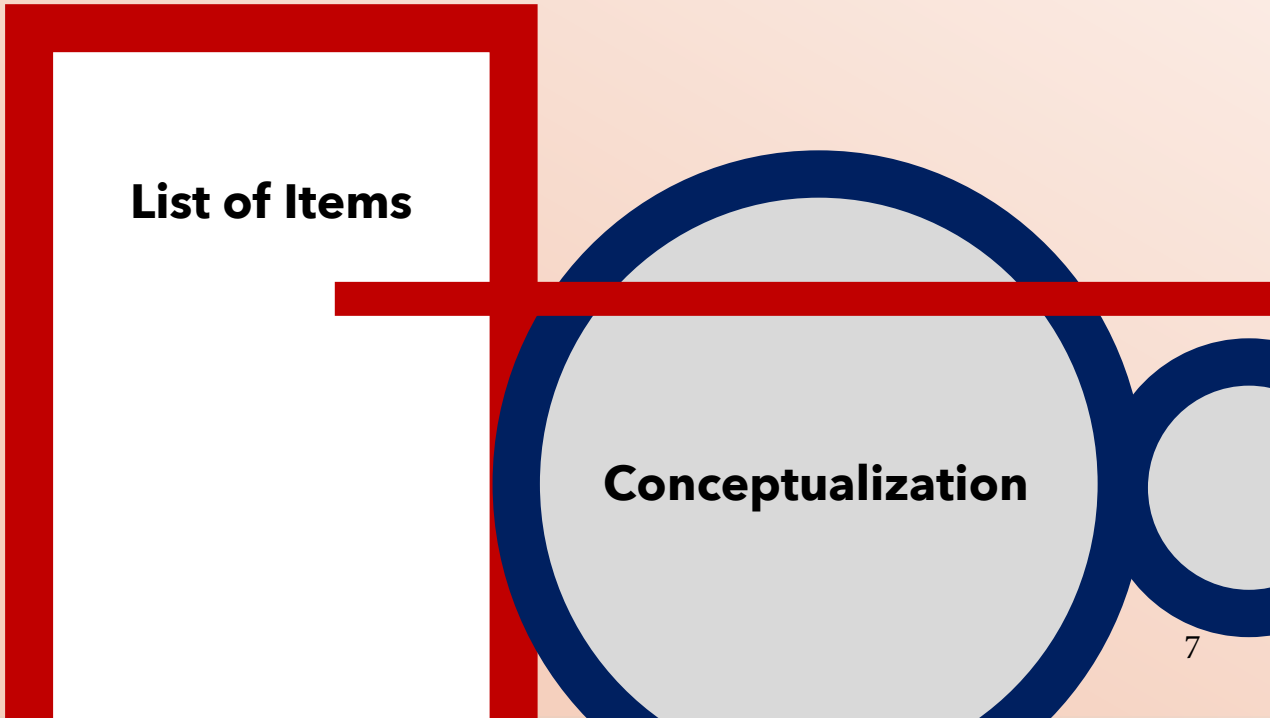
The day's work is done. That's an ancient saying of the rural population. Here, a day's work has a beginning and an end, with the day in between. There is great emotional value in being done for the day, in being able to say *I've done my part.* Completing the day's work provides a sense of achievement. People in software development never had that – which has long eroded socio-technical systems in tech. **TOSD restores the importance of the Daily Portion.** Daily rhythm is preferable to any other, as it matches our planet's most powerful rhythm – that of day and night. A day leaves people some slack. Daily rhythm allows for consistent self-organization, proper coordination within groups, and corrective measures.



3. A *List of Items* is like beautiful long hair: It needs a lot of combing

A List Owner should be somewhat in love with her or his List(s). It's not enough to have the *mastery* to curate a List. List Owners must also be somewhat *passionate* about solving the problem that underlies the List. The List Owner's work is about protecting the List's scope, curating the catalog of items, and about getting the size of items right: Items should be single-day items, ideally. 2-day and 3-day items are fine, when no other options seem available.

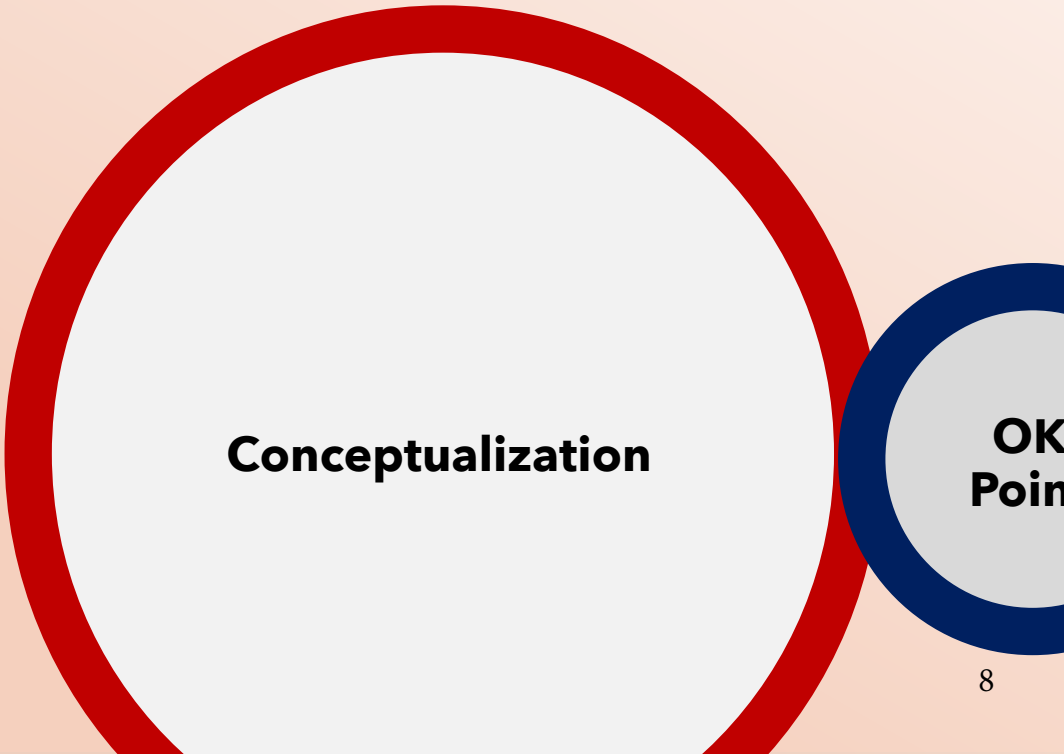
List Owners work daily with the Cs and Rs assigned to their Lists, but they also interact with other List Owners about reassigning Cs and Rs between Lists whenever needed or appropriate. List Owners can and should have other roles, apart from being List Owners. List Owners can act as Cs and Rs on their Lists. People can also switch into and out of the List Owner role when new Lists start or current Lists are exhausted. **List Owner, in short, is not a position, but a role.**



4. *Conceptualization* should also be time-boxed

Duration of the time-box for Conceptualization will usually be less than 1 day. Conceptualization may mean creating a verbal concept of 2 pages. In the age of AI-aided software development, it can also mean creating a *Proof of Concept* in the form of a functioning prototype. In this case, Realizers will ignore the AI-generated code from the prototype and develop the final code in a more systematic fashion (usually also aided by AI, in some form). The concept of XP (Extreme Programming) can serve as a yardstick for how to approach Conceptualization and Realization.

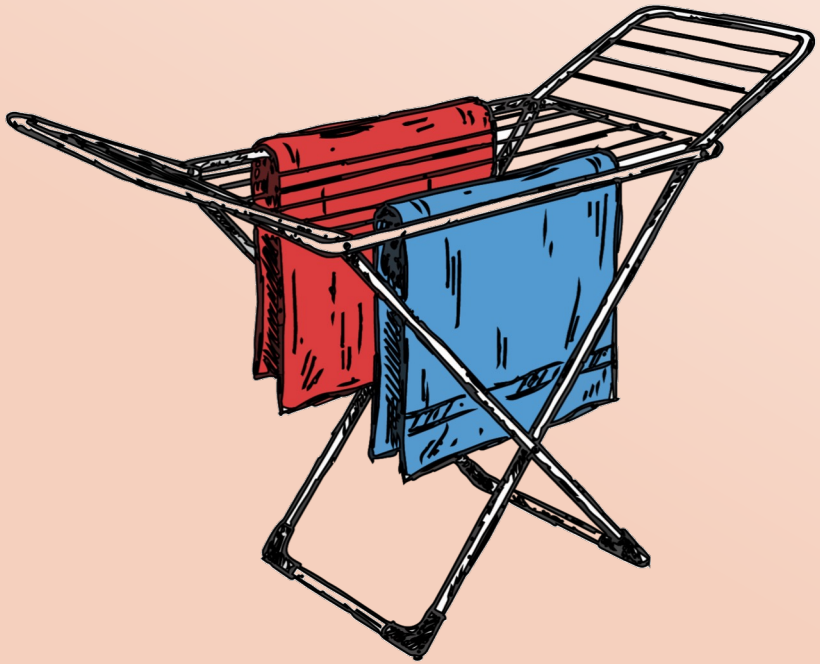
Remember that the OK Point is the antidote to shoddy development processes. It is important not to produce concepts "in bulk", so that they end up piling up ahead of the OK Point. Concepts such as these inevitably become obsolete or are of low quality. Always produce concepts in sync with the speed of realization.



5. How big is your organization's *List Drying Rack*?

You can only place a limited number of towels or clothes on a drying rack. How many pieces you can place on your rack at any given time depends on your rack's size. How big is your List Drying Rack? A company with 10 developers or more (Cs and Rs) will usually be able to place more than one list on its rack – these are the Lists it can actively work upon. **Never work on more Lists than your rack allows, or you will inevitably mess up flow.** Depending on the type and scope of the problem a List deals with, it should be exhausted within a period of 2 weeks to 2 months. One problem, one List. No List without a problem. Lists have a bounded scope. **Because remember: A List is not a backlog – it must exhaust itself.** Lists eventually dry out. They are done! A List *cannot stay on the rack forever*, but must give way to other Lists, after a while.

The more capacity you create, the more Lists you can work on at any given point in time, or the faster you get Lists done. In time orientation, capacity is not fixed. Capacity swings and is designed as a system.



6. *Strategy* must be integrated into the TOSD system. It cannot exist in isolation

In a TOSD system, strategy lives where the Lists of Items live. It's where choices and agreements around Lists are made. Definition of requirements, User Stories and the like, on the other hand, are more akin to making up rumors. In a system of time orientation, it is better to drop them, and develop *prototypes* and *Proofs of Concept* instead (with the help of AI), It is better to discuss prototypes and Proofs of Concepts and deploy them quickly, than to give in to baseless dreaming of so-called strategy experts, or even clients, as they may lack context to make educated recommendations.

In essence, strategy is a TTEO matter. When you truly Talk To Each Other (TTEO), in a timely and problem-related fashion, then strategic insights and agreements will come to life almost by accident. For a TOSD system to be created and be sustainable, you will have to remove discourse around strategy that is detached from conversation around TOSD Lists.



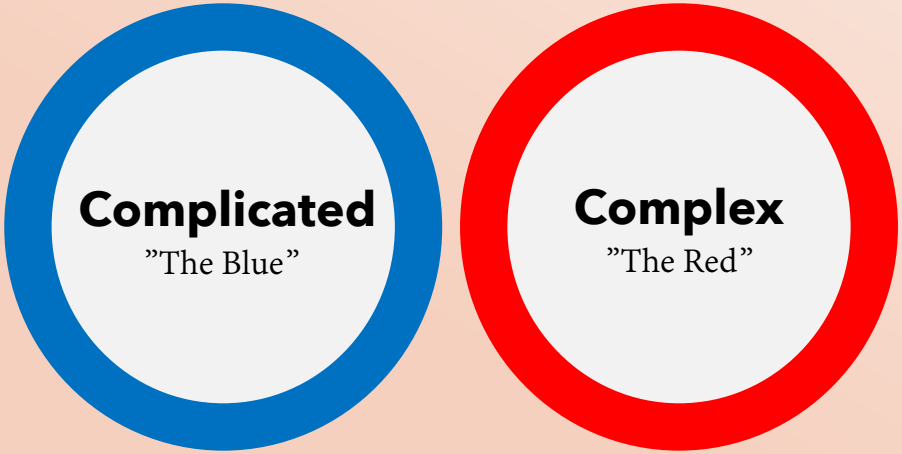
7. The answer to the *How?* of AI-aided software development is *Who?*

The real question with AI-aided development is not "How" – it's "Who"! To address the question of using AI at work, we should turn to a concept that's well-known in BetaCodex circles: The difference between the Blue and the Red. Large Language Models (LLMs) such as Anthropic's are among the most powerful and most widely usable technologies ever created. As a technology, AI is Blue, or complicated. However, dealing with this tech productively requires *Red smarts*. Problem-solving in complexity requires people with ideas – people with mastery. This is the reason why some people get a lot out of using AI, while others achieve nothing worth mentioning: The second group simply lacks the necessary ideas or mastery!

Conceptualization and Realization represent different work approaches and therefore require different tools for AI-support. A variety of different masteries will come into play. In times of AI-aided development, those stuck in *capacity-oriented thinking* will likely augment or scale the biggest problem they have always struggled with: Articulating requirements and boundaries properly (a.k.a. *Conceptualization*). *Precision* has been the problem, not *coding*, a.k.a. typing the code. AI cannot fix that problem for you, but proper distinction between Conceptualization and Realization, combined with time-boxing, can.

Blue systems
Controllable from outside
There is right/wrong

Problem-solving in the Blue
set up processes
use machines, AI
requires knowledge
turns simple with knowledge
ask How to do it?



Red systems
Not controllable from outside
Produce surprise

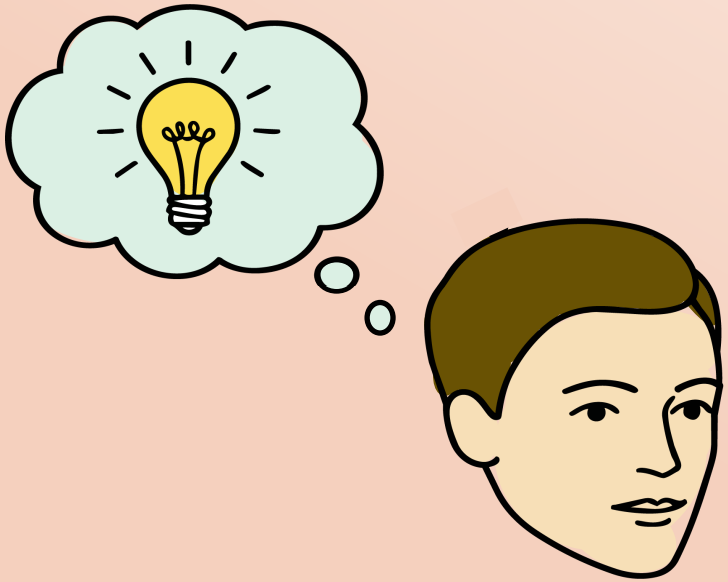
Problem-solving in the Red
set up projects
use people with ideas
requires ideas
same input, different outputs
ask Who can do it?

In work, the Blue and the Red shake hands: Work problems are "dual"!
Work problems usually contain both Blue parts & Red parts

8. AI makes (critical) thinking more important, not less. Time orientation provides the discipline

Illusions abound around the use of AI in software development. Some people loudly dream of AI being turned into team members, while others like to believe that AI will substitute developers. Such notions are absurd. Instead, AI technology will turn smart developers with appropriate mastery into better, more effective developers, while making developers without wit and without mastery look less effective than they ever seemed. Developers who lack mastery to work with AI tools effectively will simply be substituted by developers who have it. Sharp minds and people with ideas will be in more demand than ever, while others who are less inclined to work with AI (or to having ideas) may leave the field of software development, for good.

The use of AI will differ in the three work domains of a TOSD system: List curation, Conceptualization and Realization. Time-boxing the work into Daily Portions will help developers of all kinds avoid the trap of overthinking and otherwise losing themselves in the work. Sufficiently small items will keep the work manageable and thus help maintain transparency and flow.

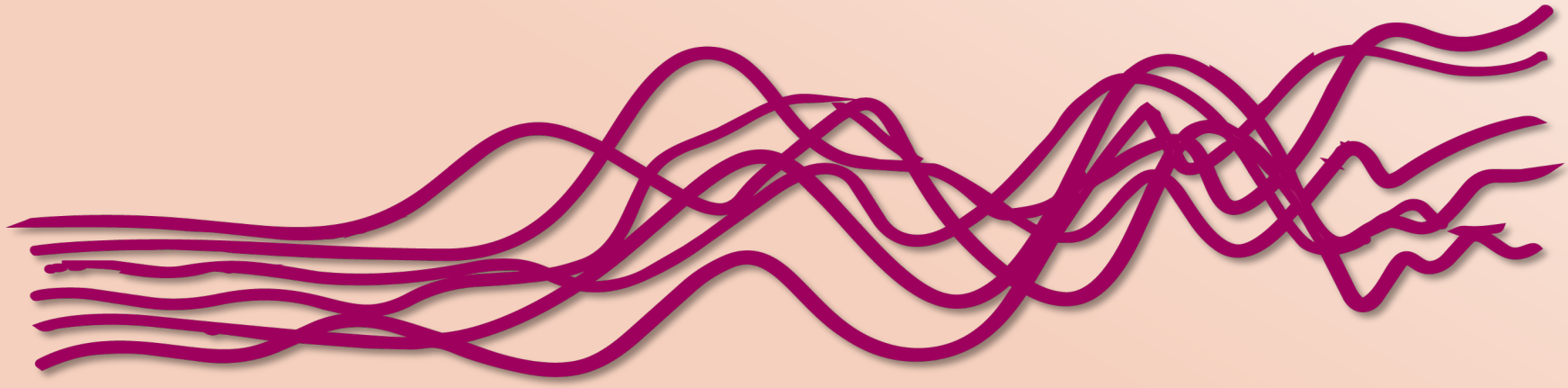


9. *Let capacity swing.* No system of time orientation will last long without this

Who sets the pace when a company resonates, or swings with the market? The question is not hard to answer: the whole world swings. The earth swings around the sun every year, the moon swings around the earth every month, and the earth swings around its own axis every day. The economy swings: we have economic cycles, seasonal influences, and recurring fashions. Whether cooking at home or farming, we naturally live in tune with the rhythm of life. In those contexts, we don't argue about *ideal batch sizes, setup times,* and *economies of scale*. Physics is the pacemaker. Life is the pacemaker. Ernst Weichselbaum concludes: **"If we don't put up any resistance to the flow towards customer demand, then value creation can swing."** In TOSD, throughput time is fixed, but the capacity is variable, so that production and value creation can swing.

Removing obstacles to swinging capacity and flow is essential. In time orientation, we remove bottlenecks, silos, the grouping of work, and internal steering. Beyond this, several options exist to let development capacity swing:

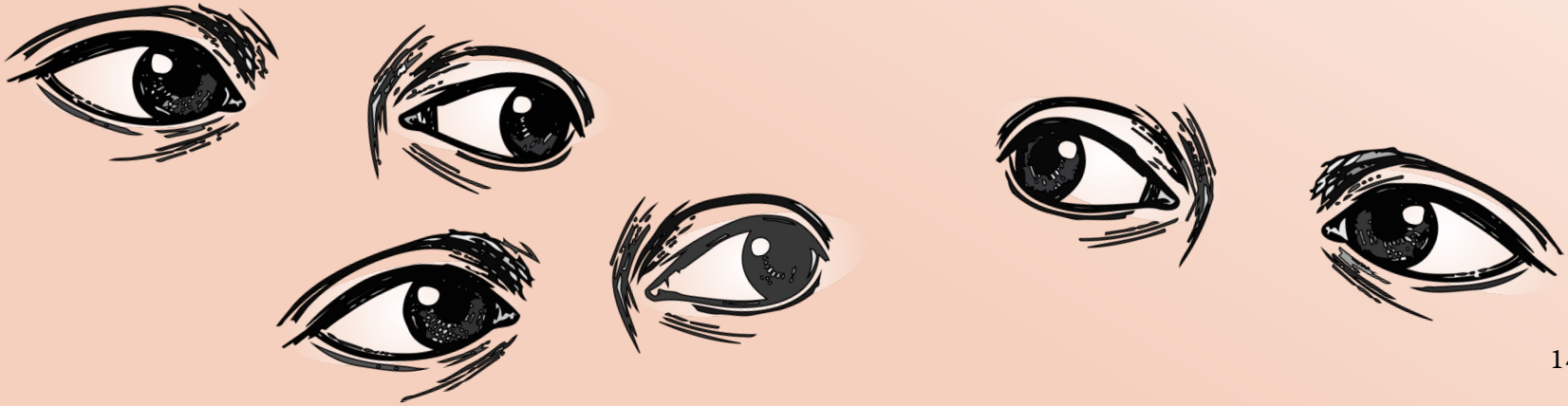
(1) relocate internal capacity to where it is needed most.
(2) develop multi-specialization in people, instead of mono-specialization. It will make capacity more readily available.
(3) increase internal capacity, by hiring, or by purchasing additional capacity through acquisitions.
(4) develop external capacity, by partnering or outsourcing.



10. Social density within the group is good for both high spirits and high performance

These days, developers often work in a highly dispersed fashion, collaborating virtually, most of the time. Often, groups of developers are geographically distributed, working up to 100% remote. Meeting in person may hardly ever happen. But regardless of the locations your developers may work in, the TTEO (Talk To Each Other) principle is paramount to making a TOSD system work. **TTEO implies that all good communication is instant and frictionless.** How the TTEO principle is enacted will take different shapes if developers all work in the same location, or if they hardly meet, if they are usually in the same office or work remotely most of the time. No matter how many people are co-located, what matters is that when problems arise, people consult each other and have discourse. Also make sure people gather for the OK Point every day, always.

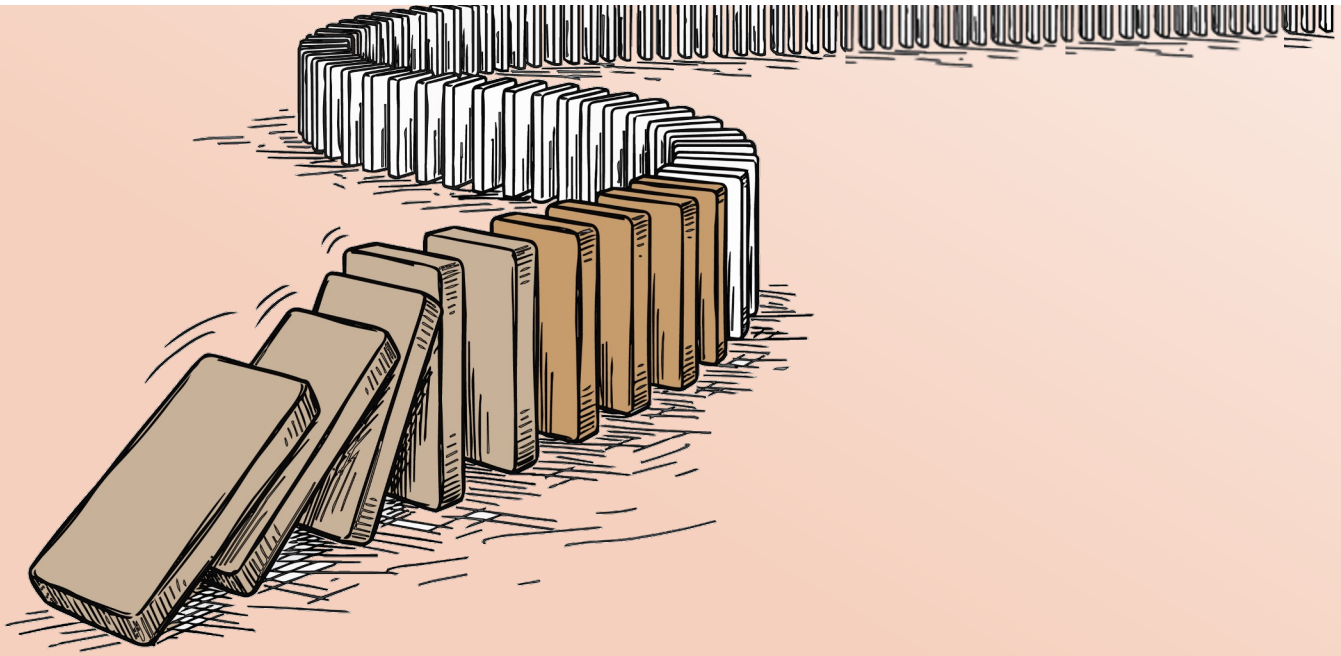
Making interactions instant and frictionless is not so much a matter of *specific tech* or *using tech differently*. Instead, it usually requires removal of bureaucracy and other barriers to conversation. In Berta, we call this Org Hygiene – a concept we will return to later in this paper. One often-overlooked concept in Beta and other time-oriented contexts is this: **Travel is cheap**. When serious problems arise, or conflict manifests, it is always *cheaper* to meet in person than trying to fix things remotely. It is an illusion that remote work alone is "good enough", in the long run. Because humans are social beings, meeting physically matters. Eating together as often as possible matters. Make generous use of *pair programming* – in-person and online. Use in-person *hackathons* when appropriate. Use office presence to help people mingle and observe each other at work and interact. And always remember: *Travel is cheap*.



11. Immediacy matters, managerial consistency is critical! Quality, finishing on time and learning are non-negotiable

Consistent, immediate action matters in a time-oriented system. **There are three things you cannot tolerate in TOSD: Sloppy quality, lack of punctuality, and lack of willingness to learn.** Everybody makes mistakes – but there is a difference between making mistakes and not bothering about quality. Everybody will experience that it's some-times hard to be punctual – but there is a difference between struggling with punctuality and not caring about punctuality. Everybody will experience moments when their mastery is insufficient – but there is a difference between having to learn something and not caring about one's own professional development. These three qualities - caring about quality, caring about punctuality and swiftness, and caring about one's own learning are non-negotiable in a sustainable time-oriented system.

Hire accordingly and act immediately on problems related to these three aspects of personal integrity. They are the cornerstones of consistent self-organization and time orientation in software development.



12. There are two kinds of White Space – one is an enemy, the other is desirable

When we talk about White Space, we are usually referring to waiting times inside a process: Times of inactivity, storage, delays and inaction during that add no value. In approaches like *Process Management,* *Lean Management*, or the *Theory of Constraints*, we aim at eliminating this kind of White Space, or we at least try to reduce it systematically. Time orientation routinely goes after this kind of White Space as well. But there is a second kind of White Space which is more benevolent. This kind of White Space needs to be observed and protected, to make a system of proper time orientation function. **It is White Space that appears at the end of a Daily Portion.**

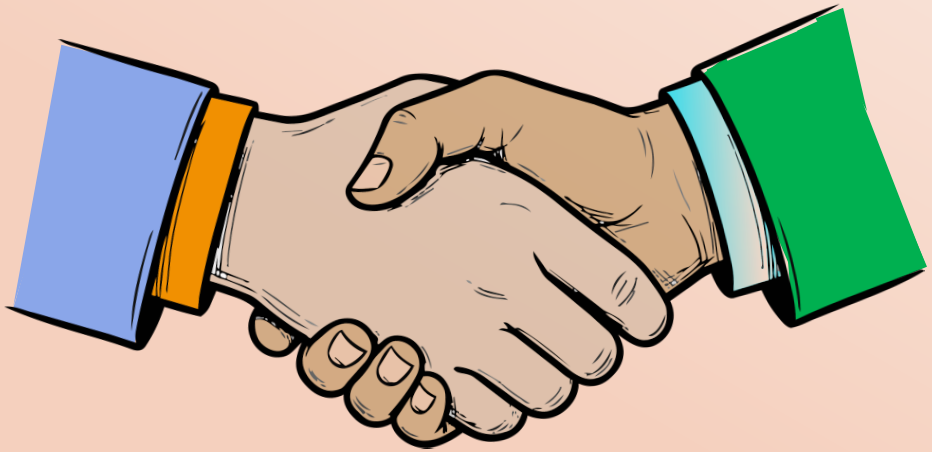
This kind of White Space is something like the opposite of *overtime*. It makes important and more subtle aspects of work noticeable. In TOSD, White Space at the end of a Realizer's Daily Portion means this person can either *call it a day*, or do something learning-related, or do other kinds of client work for the rest of the day. **But that Realizer is not allowed to pick up another development item until the next day's OK Point.** Protecting this White Space is necessary to gauge developer's mastery and ability, but also to assess if Items on the Lists are well-sized, and if items are well-conceptualized during Conceptualization. Only by respecting this type of White Space can you improve a TOSD system, over time.



13. With TOSD, sourcing & selling development capacity externally is a piece of cake

With time orientation, development capacity becomes a clearly recognizable in time. It also becomes modular and easily tradable. By structuring work strictly around short, explicit time-boxes and well-defined distinctions like Conceptualization and Realization, TOSD removes much of the uncertainty, hidden coordination effort, and dependency risk typically associated with external sourcing. External developers can be integrated seamlessly (and daily) into the same time-oriented flow as internal developers, delivering specific results within one to three days – eliminating the need for lengthy onboarding, micromanagement, or contractual complicatedness. This clarity makes it equally simple to buy development capacity from partners and to sell it outward as a reliable, high-quality service: transparent, predictable, and repeatable.

Billing between the partners will be by *Daily Portion*, setting prices can be agreed by asking *What is the value of the Daily Portion on this List?* In short: **In a TOSD setup, sourcing and selling development capacity is no longer a gamble – but rather a straightforward decision, and a solution to letting capacity swing, over time.**



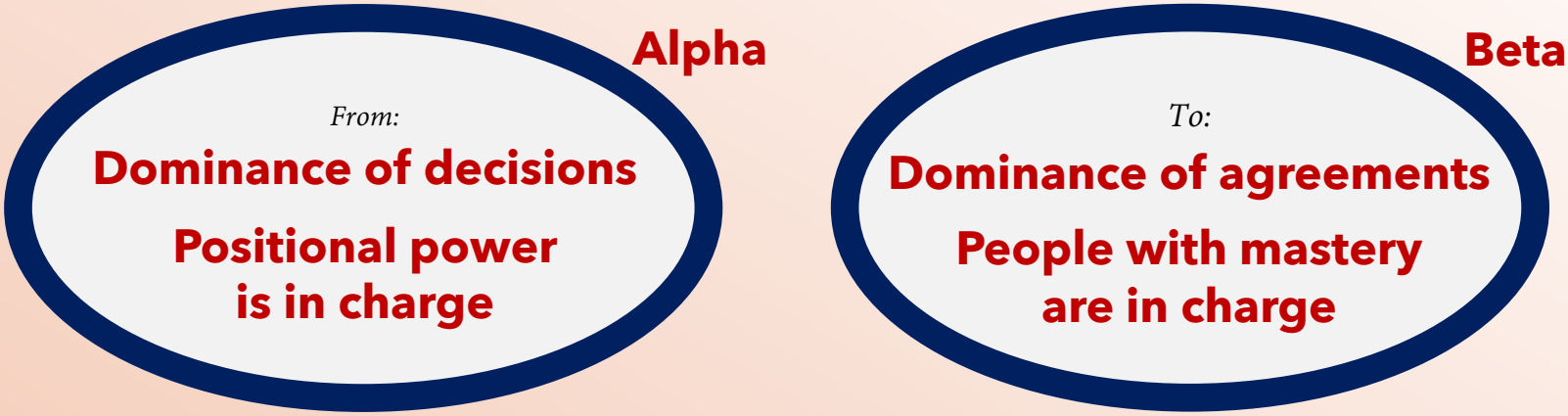
14. Estimation is a slippery slope

Estimating is usually the same as guessing, and guessing is the enemy of getting things done. The reason is complexity, which renders most forecasting and guessing useless. In particular, with software development, estimation is a slippery slope, as it is the starting-point of centralized steering and too much dependency on measurement. It is also the origin of too many meetings and too little action: Estimation keeps you from *designing capacity as a system.* It will become a substitute to improving things in the here and now, and a barrier to decisive action. Estimation itself also consumes enormous amounts of time, a.k.a. capacity.

In short: **Don't do it. Do not allow estimation to happen.** At Pradtke, since the start of the TOSD system, no estimation was necessary, 100% of discourse was about designing capacity, which is the ultimate job of managers in a time-oriented system. There were occasional calls for estimation, though, out of habit and preference. These calls had to be politely and systematically declined. You should do the same.



15. Address the distinctions between decisions & agreements, positional power & mastery



The question of *Who makes what decisions in TOSD?* is a command-and-control, or *Alpha*, kind of question. It will come up early, and it will come up repeatedly. Questions like this will likely be brought up during the first conversations about a time-oriented system. *Who will decide upon the Lists? Who decides who the List Owners are? Who decides this when people disagree? Who is responsible for strategy? But there must be someone!* You will have to make sure to address these questions correctly, or they will return repeatedly, and potentially create negative dynamics.

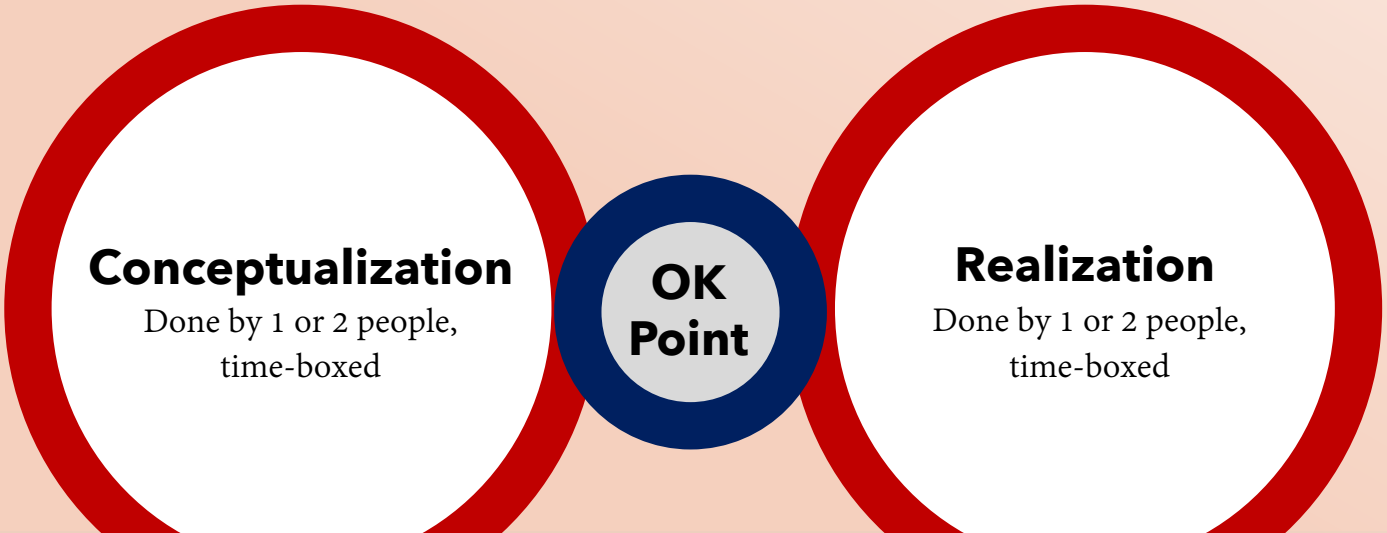
Such questions (even if some of them may be reflexes, not actual questions) should all be addressed in the same way. (1) In time orientation, it is not *decisions by bosses* that matter, but *agreements among relevant actors*. This will lead to the elimination of certain roles and positions, and to dropping committees and most regular meetings. (2) It is not *positional power* that matters most in TOSD, but *people with mastery*. This goes for the daily business *and* for creating the business of the future. Time orientation devolves power to people with mastery. (3) If decisions and positional power *worked*, the current mess within a not-so-effective development system wouldn't exist. But it does. It is better to acknowledge that hierarchy, centralized steering and "top-down" decision-making are sure to lead to failure. **There is no risk in adopting TOSD. The risk lies in leaving the system as it is.**

Ultimately, there is a question behind the questions: *Are you willing to act upon the system and change it – or not?*

16. Moving to a time-oriented system means abandoning the flawed notion of *software development teams*

The belief that *software is developed by teams* is an inheritance from a time when software development was always under-taken in the shape of large, long-term projects. Surprisingly, faith in the need for dedicated development teams survived the Agile upheaval. But look closer at the reality of development, and you will notice that actual software development activity is undertaken not by groups or teams, but by single individuals, or in pairs, at any given moment of time. So why do development teams still exist? Not because of the work itself, which requires individuals or pairs – but because of tradition and irrational beliefs. The result, however, is siloed organization, which will cause unnecessary need for cross-communication, stop-and-go-activity and White Space within processes. In organizations like this, flow is unattainable.

In TOSD, however, we can overcome the idea of development teams entirely, letting individuals or pairs work on either *Conceptualization* and *Realization*, with handovers between the individuals or pairs at the *OK Point*. When we abandon Sprints and overcome the fixation on dev teams, developers can self-organize (alone or in pairs), and make continuous, reliable flow a reality. AI-aided development makes this even more intuitive and achievable. With the help of AI, developers can be freed from the organizational silos, and integrated into business teams, becoming more attuned to client work, services and a variety of processes.



17. Tackle *Organizational Hygiene* early

The TOSD approach is simple. But adopting a TOSD system will usually require *overcoming a current system of capacity orientation and command-and-control steering*. This makes new, time-oriented beliefs, and mental models necessary, as well as the removal of a great deal of organizational practices, methods, roles, processes and rituals. With that comes the need to adopt new words and patterns of language, which we will turn to in the next pattern.

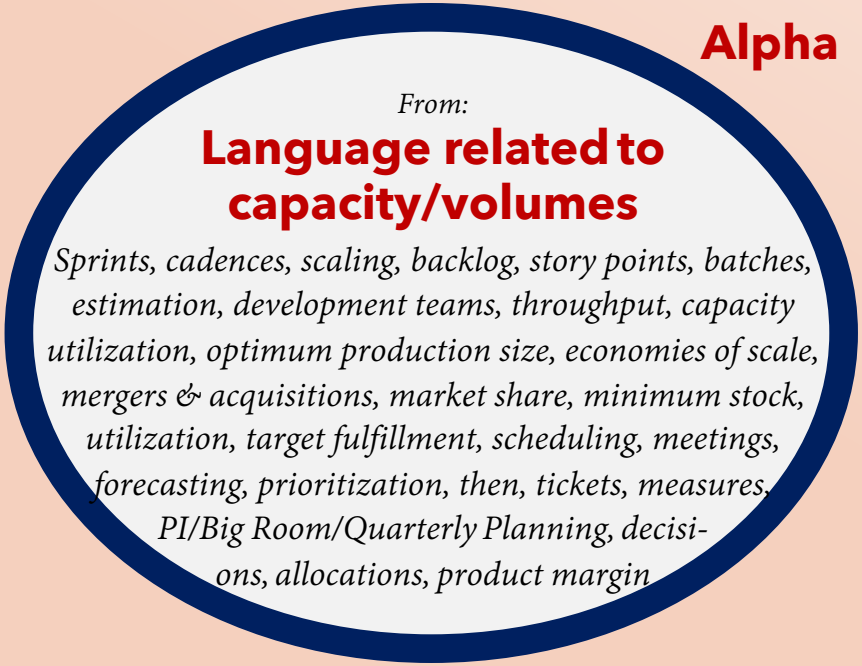
The list below features a selection of "anti-patterns" that don't get along well with a system of time orientation. You are probably already aware that some of the practices, methods, roles or processes below are dysfunctional and counter-productive. Others may still seem reasonable or even necessary to you. Once you start thinking about adopting a TOSD system, it helps to consider overcoming these practices, methods, roles and processes – *for good.*

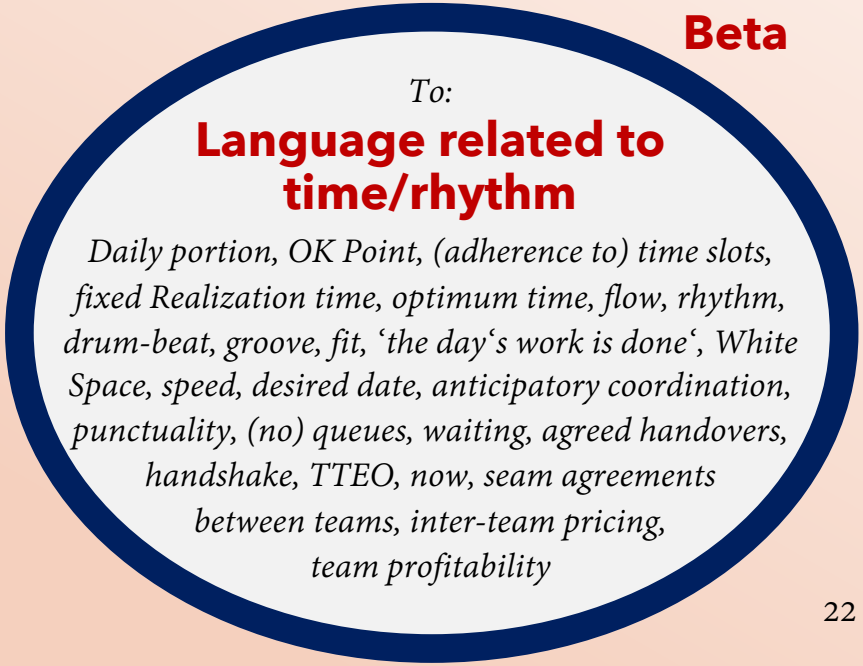
selection, in alphabetical order

- Agile Coaches
- Agile Scaling
- Approvals
- Architects (title)
- Backlogs
- Business Analysts
- Committees
- Dailies, Stand-ups
- Development Teams
- Estimates
- Handovers
- Individual targets
- Issue Tracking Software
- Kanban boards
- OKRs
- phases
- Prioritizations
- Product Manager/management
- Product Owner
- Production Readiness reviews
- Project Manager
- Requirements
- Resource pools
- Retrospectives
- Scrum Masters
- Sprints
- Sprint goals
- Sprint planning
- Story points
- Subject Matter Experts
- Testers
- Tickets, ticketing systems
- User stories
- Utilization measures
- UX Designers (title)
- …

18. Adopt the language of time orientation, right from the start

Language has a stabilizing function for any social system. Language is the way our beliefs around work, systems, problems and collaboration are expressed. **When starting with TOSD, assume that your current work language is part of the problem, not part of the solution.** In the previous TOSD paper, we outlined that adopting time orientation means overcoming the system, not merely optimizing it. Overcoming a system means overcoming beliefs, and our words and language are chained to those beliefs. The good news is that once you decide to change your words and patterns of language, which can be quite simple, you will see new options emerge and new interactions manifest themselves. **Over time, these words and related patterns of language will transform your own thinking, and that of your colleagues.** Changing the language will be part of every conversation around TOSD: before adopting it, while adopting it, and for months or maybe years after the "Go Live" date.





Sources and recommended reading

Time-Oriented Software Development, timeoriented.dev

Pflaeging, Niels/Kubsch, Sebastian (2025): *Introducing Time-Oriented Software Development*

Pflaeging, Niels (2026): *From Toyota to TOSD: For high performance, make time the boss*

Pflaeging, Niels (2026): *AI-aided software development needs Time orientation. Not yet another framework*

Time-Oriented Work Systems – books and research papers

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

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 (2023/2024): *Cell Structure Design Patterns, Part I and Part II*

Reinertsen, Donald (2009): *The Principles of Product Development Flow: Second Generation Lean Product Development*

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

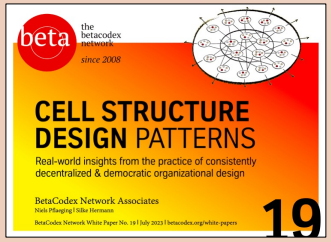
Schmenner, Roger W. (2012): *Getting and staying productive – Applying swift, even flow to practice*

Weichselbaum, Ernst/Pflaeging, Niels (ed.) (2026): *What would Ernst Weichselbaum do?*

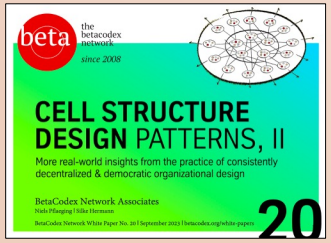
Related BetaCodex Network research papers, available on betacodex.org



From Now to New, Right Here, 2019



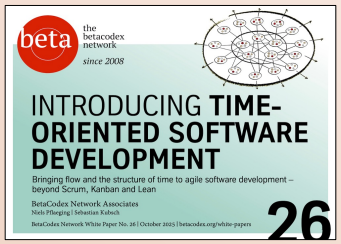
Cell Structure Design Patterns, I, 2023



Cell Structure Design Patterns, II, 2024



Slave to the Rhythm, 2025



Introducing TOSD, 2025

The BetaCodex Network research papers

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

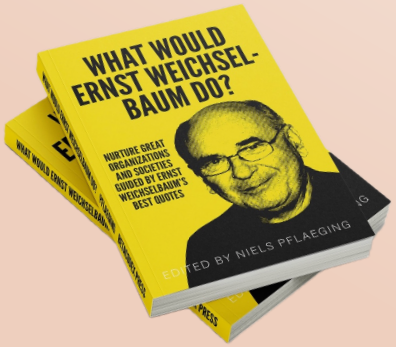
the betacodex network since 2008



Related books from BetaCodex Press

beta codex press

The 1st book in English on the work of the leading thinker of time orientation, Austrian consultant Ernst Weichselbaum (1944-2024)



What would Ernst Weichselbaum do?

Edited by Niels Pflaeging

BetaCodex Press, 2026

Also available as an E-book

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

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 author of this paper:
Niels Pflaeging, niels.pflaeging@redforty2.com

Paper concept: Niels Pflaeging
Design & illustration: Niels Pflaeging

The TOSD website: timeoriented.dev

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

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/license

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

Time-Oriented Software Development (TOSD)
is open source, so that you can make use of it.
Check out the open source license at
www.redforty2.com/tosd



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