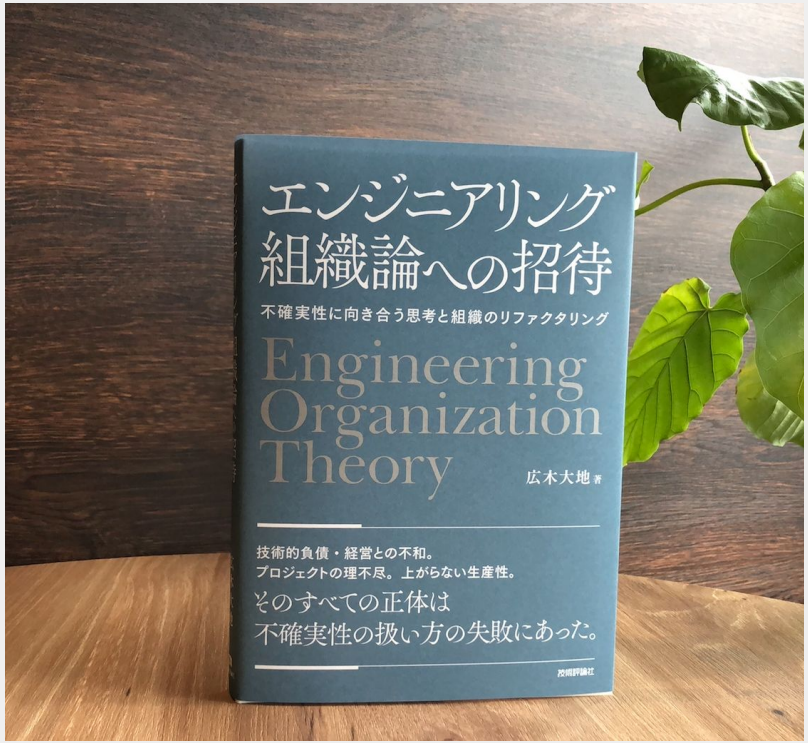
2つのDXとソフトウェアエンジニアリングの文化資本論

広木 大地 @hiroki_daichi

自己紹介



広木 大地(ひろき だいち)

2008年度に新卒第1期としてミクシィに入社。
メディア統括部部長、開発部部長、サービス本部長執行役員。組織改革の戦略・実行を行う。2015年同社を退社。

2016年「**CTOのノウハウを広く世の中に還元する**」ことを目指し、レクターを創業。

2018年2月エンジニアリング組織論への招待上梓。
第6回ブクログ大賞ビジネス書部門大賞受賞。
ITエンジニアに読んでもらいたい技術書2019大賞。

EOF2019

Engineering Organization Festival 2019

参加無料

会場

浅草橋ヒューリックホール&カンファレンス

時期

2019年10月31日

ビジョン

エンジニアリング組織を

もっとオープンに

主催 / 実行委員会

EM.FM パーソナリティ

湯前(アカツキ) /広木(レクター)

EM Meetup

大庭(Quipper)

ビジネス書大賞と技術書大賞の２つを
受賞した初めての書籍

ソフトウェアエンジニアリングも
組織設計もビジネスの一部。
この二つは**不可分**じゃないか。

私のテーマ： ２つのDX

=

DX (開発者体験)

Developer eXperience

DX (企業のデジタル化)

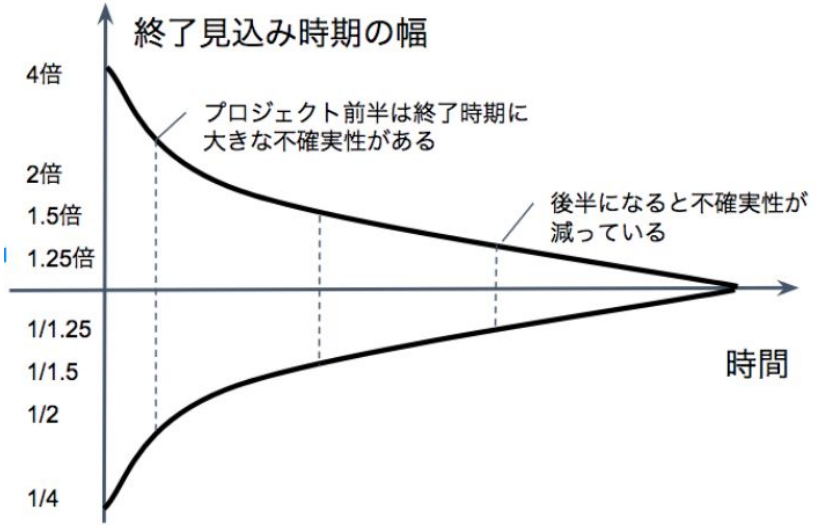
Digital transformation

エンジニアリング組織論の骨子

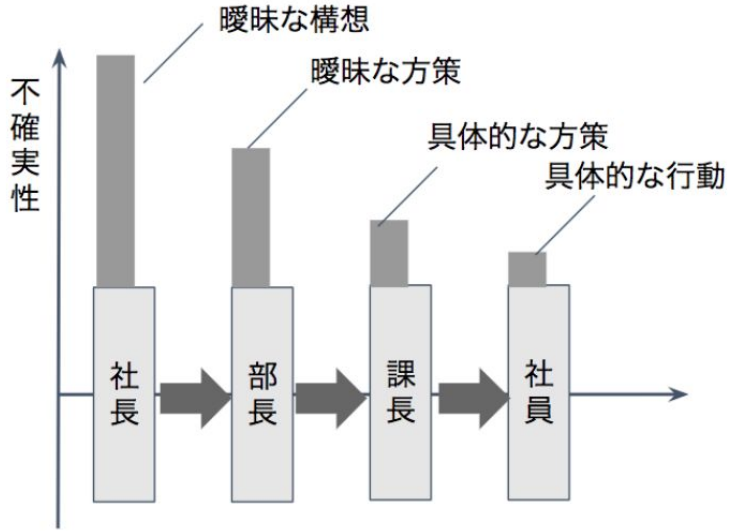
1. 何かを**実現**するとは、**不確実性を減らす**ことである
2. **不確実なもの**から人は**闘争・防御・回避**をしようとする
3. それらを**防ぐの思考と組織構造**を持つと**生産的**になる

エンジニアリング = 不確実性を減らす試み

プロジェクトも不確実性を減らす

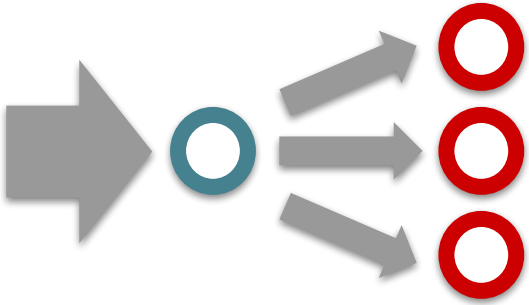


組織も不確実性を減らしていく



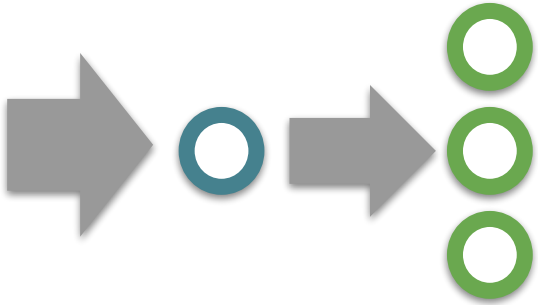
多くの不確実性に応えられるチームほど生産的

❶マイクロマネジメント型



抽象的で不確実な要求を、細かくて具体的な指示に分解しなければ、実施することができないチームは、上長の情報処理能力がボトルネックとなるため、高度に生産的なチームにならない。リーダーとメンバーの格差が大きすぎるとこのようなチームになりやすい。

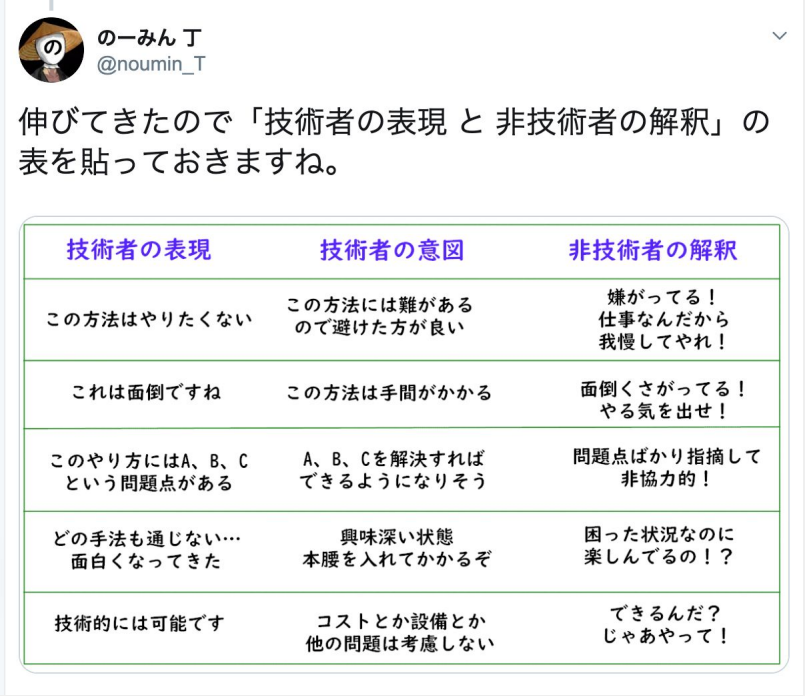
❷エンパワーメントチーム型



不確実な要求をそのまま受け取って具体的なアクションに展開できる組織が不確実性を処理するという観点では、生産的になる。チームとリーダーとの格差が減ってきて、権限が委譲されていること、チーム内のコミュニケーションが進んでいるとなりやすい。自己組織化と呼ばれる。

本日は、良いエンジニアリング組織が
つくられる**マクロな機序**について
お話しします。

エンジニアの言葉は、なぜ伝わらないのか。



のーみん 丁
@noumin_T

伸びてきたので「技術者の表現 と 非技術者の解釈」の表を貼っておきますね。

技術者の表現	技術者の意図	非技術者の解釈
この方法はやりたくない	この方法には難があるので避けた方が良い	嫌がってる！仕事なんだから我慢してやれ！
これは面倒ですね	この方法は手間がかかる	面倒くさがってる！やる気を出せ！
このやり方にはA、B、Cという問題点がある	A、B、Cを解決すればできるようになりそう	問題点ばかり指摘して非協力的！
どの手法も通じない…面白くなってきた	興味深い状態本腰を入れてかかるぞ	困った状況なのに楽しんでるの！？
技術的には可能です	コストとか設備とか他の問題は考慮しない	できるんだ？じゃあやって！

https://twitter.com/noumin_T/status/1135013683577335809

「じゃあ、そういえばいいじゃん」

実際には、これは言葉に変換する前に、ものごとを経験的・直感的に好ましい、好ましくないで捉える **心的性向（言語化されない選好）**。

何らかの職人は、この種の内面化した「心的性向」を持ち、これらが仕事を共にすることで、暗黙的な実践知を伝搬してきた。これをブルデューは **ハビトゥス**といった。

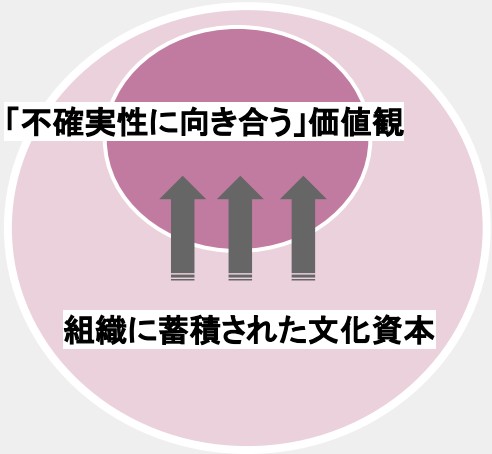
そのため、何も変換せずに思ったことを話してしまうと、**好き嫌い**を話しているように見える。

「好き嫌い」の思考の癖、これが習慣になり文化に変わる

良い思考の癖	悪い思考の癖
とりあえず、やってみよう。ダメならやめたらいいじゃん	前例は？失敗しないことを説明して。
うちの会社/ ぼくたち / おれたち	この会社/ あの部署 / あいつら
もっといいやり方がないか考え続けよう。	これまでのやり方を守っていこう
うまくいかなかったことを学習して次に生かそう。	うまくいかなかったことを隠そう。

これらから派生して、形作られる文化資本が生産性を産む

選好の文化資本/ハビトゥス



自然にやりたいと感じる

習慣行動/プラティーク

- アジャイル/XP
- TDD/CI/CD
- DevOps/IaC
- トランクベース開発
- 犠牲的アーキテクチャ
- 進化的・漸進的設計

経験を通じてさらに内面化された実践知

価値観が人を引き寄せる

「よい習慣」は合理的に説明できるわけではない

ユニットテスト・CI/CD

今では考えられないがかつては、**「ユニットテストを書くべきか、書いた方がコストがかかる」**という議論があった。

今でも、場所によってはそのような議論が存在する。

だが、現在では**「馬に乗って通勤する人がいない」**くらいには常識的なことになった。

これはユニットテストを書くことが常識になっていき、過剰なほどにテストに工数を費やさずに済むようになったことで体感的なメリットが理解されていった。

高スペックPCの貸与

今では考えられないがかつては、**「プログラマが使う端末が貧弱で、キーボードも選べない」**という環境があった。

この際に、総務やバックオフィス部門の管理コストの高さとエンジニアの生産性向上を比べて、合理的であることを説明する必要がった。

だが、現在では、ある程度のスペックのPCをエンジニアに与えるのは、**「陸軍に竹槍を持たせる人がいない」**くらいには常識的なことになった。

人的資源の価値が他の業種よりも生産性に劇的な差をもたらすため、**コストではなく競争優位で比較**するから。

新聞を”当たり前”に読む家・読まない家。

新聞が月額4000円として、それが合理的だという証明をしてください。

新聞を読む家

新聞を読まない家

これはエンジニア自身にとっても自明ではない。

- スクラム
- ペアプロ
- DevOps/IaC
- トランクベース開発
- 犠牲的アーキテクチャ
- 進化的・漸進的設計
- フィーチャーチーム
- OKR/1on1

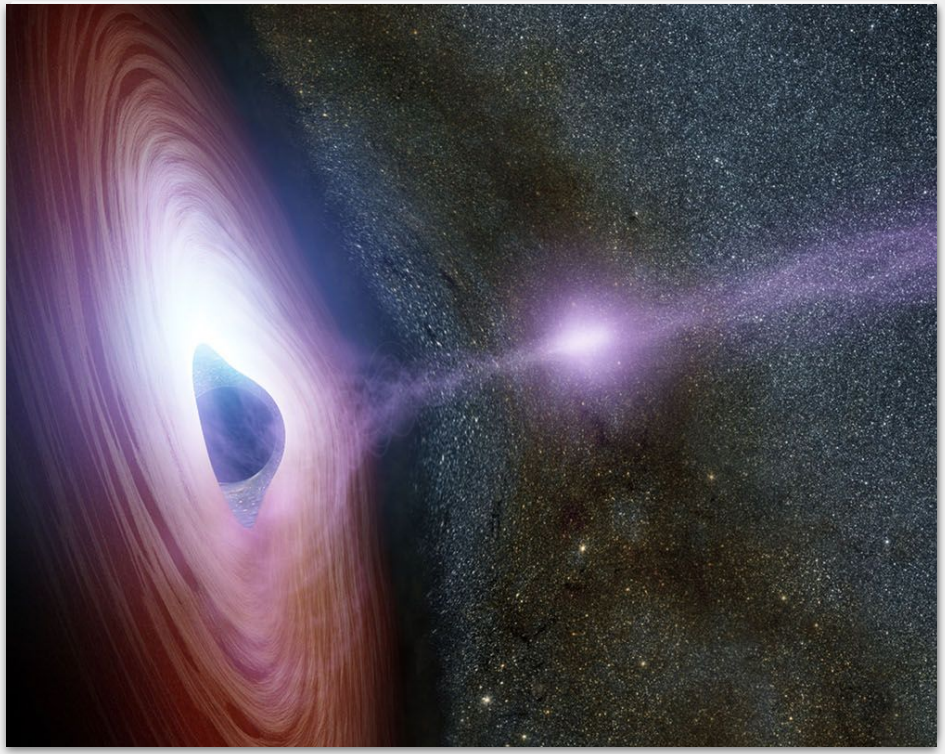
だから「形から」入るでも価値がある

実は、多く**プラティーク（習慣的行動）**は、それを形だけ真似てもすぐには意味がない。むしろ悪化することすらある。そのため、ソフトウェア工学では、ある習慣に価値があるかどうかを対照実験することが難しかった。

たとえば、TDDに慣れていない人に、TDDをやらせて、そうでない場合と比較しても優位な差が出ないどころかマイナスにもなることは想像に難くない。**自転車は乗り始めから、走るより速いわけではない。**

しかし、TDDに習熟したチームは生産性が高くなると経験的に知られている。これは、形から入ったとしても、目的を持ってやり続ければ習熟し、高いレベルになる

人材のブラックホール現象はこの原理によって起こる



場の持つ文化資本に人は惹かれる

良い組織には、良い文化資本（を持つ人）が集まる。

文化資本は、何らかの「習慣行動」の違いとなって、観測可能になる。

それが**オープンな開発者のコミュニティを通じて**、社外の開発者に漏れ伝わる。その結果、同様の習慣を持つ人が惹かれ合い、集まる。

一方、**良くない組織は、良くない文化資本**が集まり、悪い人材が集まっていく。これもまた雪だるま式に崩壊していく。

守・破・離とは懐疑的に考えつづけること

良い思考の癖

もっといいやり方がないか考え続けよう。

悪い思考の癖

これまでのやり方を守っていこう

思考停止は「守」ではない

守破離は千利休の言葉をまとめた利休百首から引用：

規矩作法 **守**り尽くして**破**るとも**離**るる
とても本を忘るな

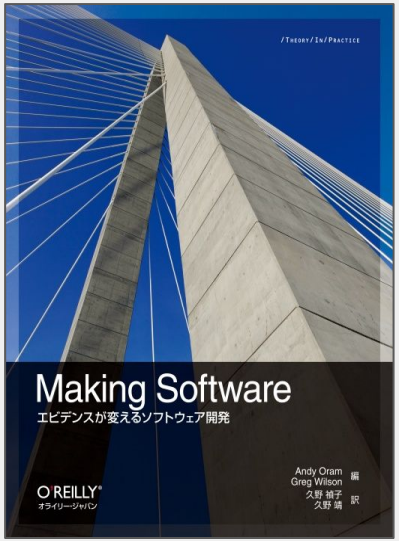
禅宗の影響をうける利休は、体感を大事にしつつも、思考停止に陥らず、目的を持って考え続ける哲学であった。

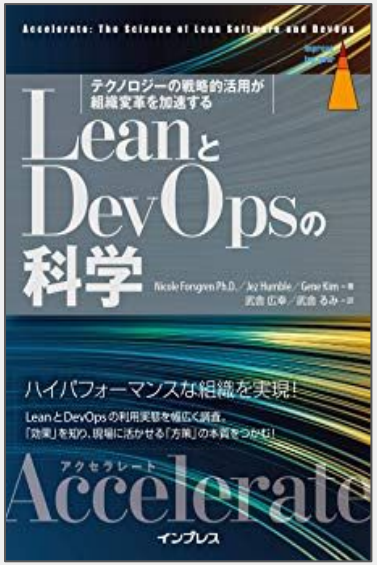
もとよりもなきいにしへの法なれど今ぞ極る本来の法

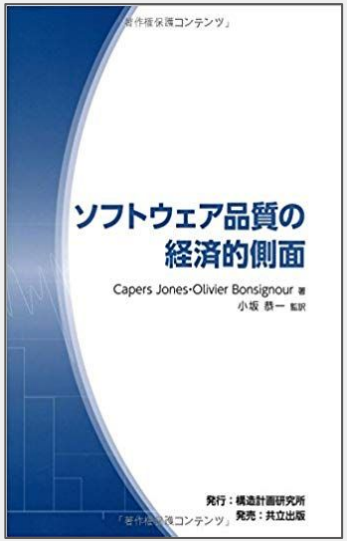
炭置くも習ひばかりに拘はりて湯のたぎらざる炭は消え炭

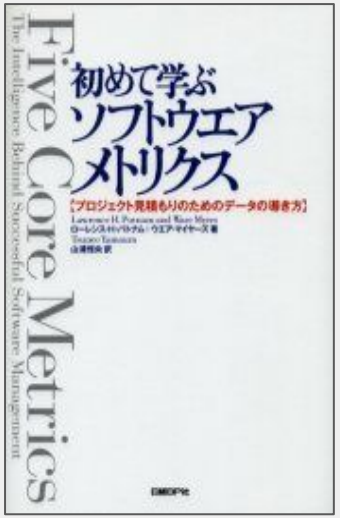
疫学的にエビデンスが集まってきている。

Systematic Reviewなども通じて、経験的に実証しようとしている。

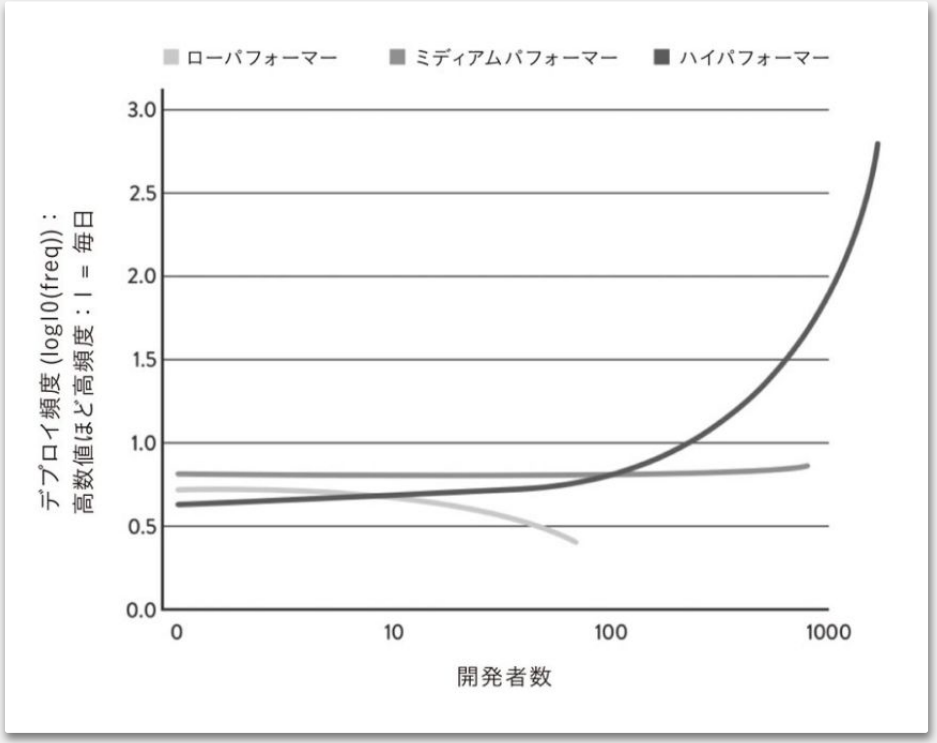








たとえば、サービスデプロイメントの数



継続的デプロイメントの心理的安全性

開発者数が増えるごとに、生産性の高い組織群はデプロイ頻度が増加する。

それに対して、低いパフォーマンスの組織は、開発者数が増えるごとにデプロイ数が低下していく。

そして、開発者数もスケールしなくなる。

そのため、マイクロサービスのようにシステムを疎結合に分解して、デプロイしやすい構造にしたり、自動テストの効率を上げていく。また、組織文化もミスに対して他罰的・権威的にならず、修正しやすい環境を用意している。

※ Lean と DevOpsの科学(p.78)

たとえば、サービスデプロイメントの数

DevOpsスコア

$$= \frac{\text{デプロイ数 (Deploys)}}{\text{開発者数 (Developers)}} \Big/ \text{営業日数 (Days)}$$

$$> 0.1$$

おおよその目安としてのスコア

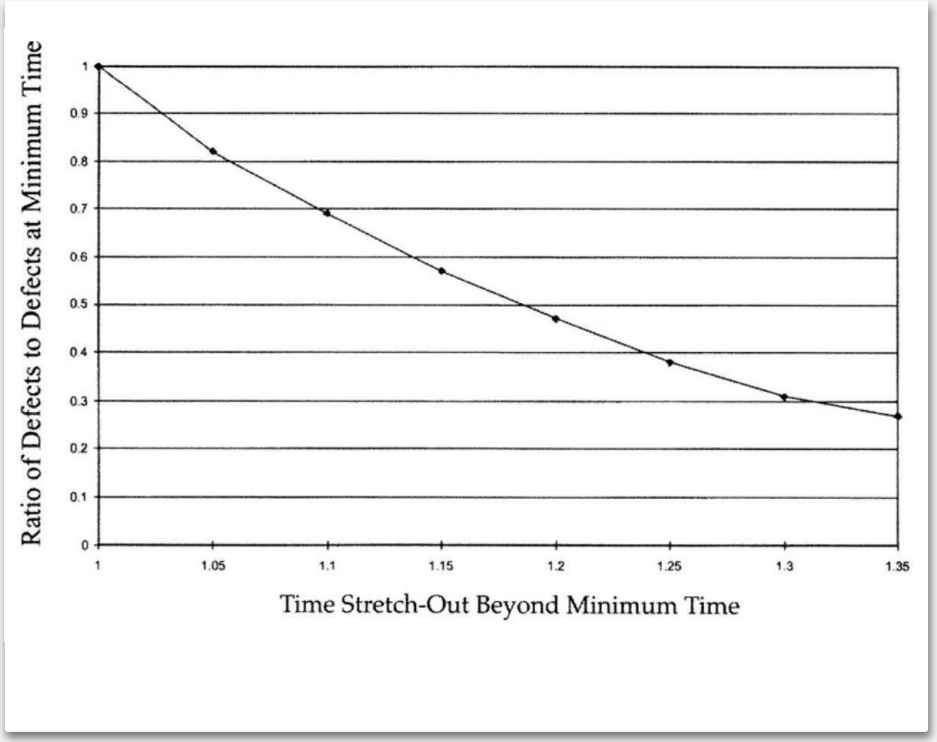
たとえば、60人のエンジニアがいて、1ヶ月（20営業日）あたり150回のデプロイ回数の会社を考えてみる。

150 / 60 / 20 = 0.125 > 0.1

と計算できる。およそ、一人のエンジニアが10営業日でつくった差分が、一度デプロイされる計算になる。

結合度の高いサービスやトランクベース開発のできていないシステムだとすぐにこのスコアが悪化する。

たとえば、開発時間の余裕と品質



20%程度の余裕が半分のバグを防ぐ

時間的余裕の欠如したプロジェクトにおいては、バグが多く練り込まれてしまう。

そのため、結局リリース後の障害や、テストフェーズでの手戻りが多くなる。

これらも経験的によく知られたことであるが、
実証されつつある。

たとえば、コンウェイの法則

コードそのものよりも高い精度でバグを予測する組織構造

表11-1　予測の正確さ

モデル	正確性	再現性
組織構造	86.2%	84.0%
コード変更量	78.6%	79.9%
コードの複雑さ	79.3%	66.0%
依存関係	74.4%	69.9%
テストカバレジ	83.8%	54.4%
リリース前のバグ	73.8%	62.9%

※Making Software (~p.190)

良い文化資本が蓄積する企業と
そうでない企業の差は
どのように生まれるか。

ソフトウェアエンジニアリングの文化資本

プラティーク(習慣行動)を獲得する際に、説明・説得に費やされるコストの差

当たり前のように
(クラウド|コンテナ|サービスメッシュ)
を使うことを決める会社

>

いちいち説明コストが
かかる会社

説明責任の選好性が文化資本を促進・阻害する権力

どちらに説明責任が求められるか

たとえば、PCのスペックであれば

- **「採用競合に比べて、悪くする理由」の説明を求めるか**
- **「去年に比べて、高くする理由」の説明を求めるか**

によって**無意識な選好の違い**が、上長が求める説明責任の方針に反映させてしまう。しばしば、合理的に説明を求めることに何の認知の歪みがないと錯覚している人がいるが、何に対し合理性を求めるかに権力構造が内包されている。

このような**まなざし(gaze)**の違いが、文化資本の形成を促進したり、妨げたりする。

コミュニケーションの難易度が
エンジニアリング組織構築の
難易度を決める

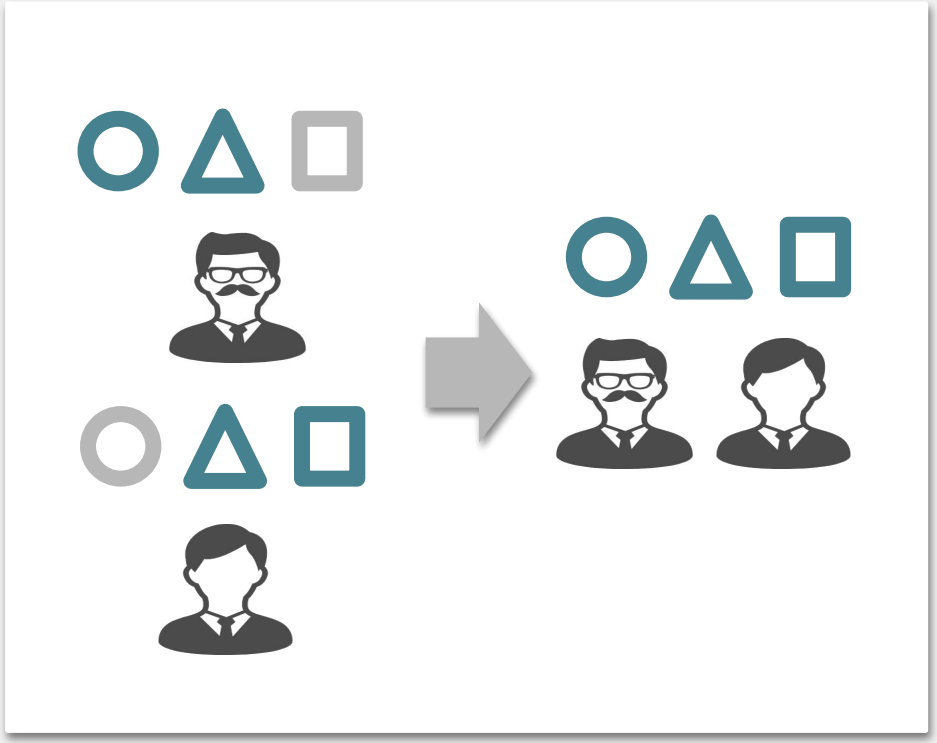
そもそもコミュニケーションとは？

情報の非対称性を削減すること

物事を実現していく過程で必要とされるコミュニケーションとは：

- **自分は知っているが相手は知らないこと**
- **相手は知っているが自分は知らないこと**

これらの差を解消していくことである。
このように、立場の違う二者で異なる情報を持っている状態を情報の非対称性と言います。シンプルにコミュニケーションをこの非対称性の解消だと定義します。

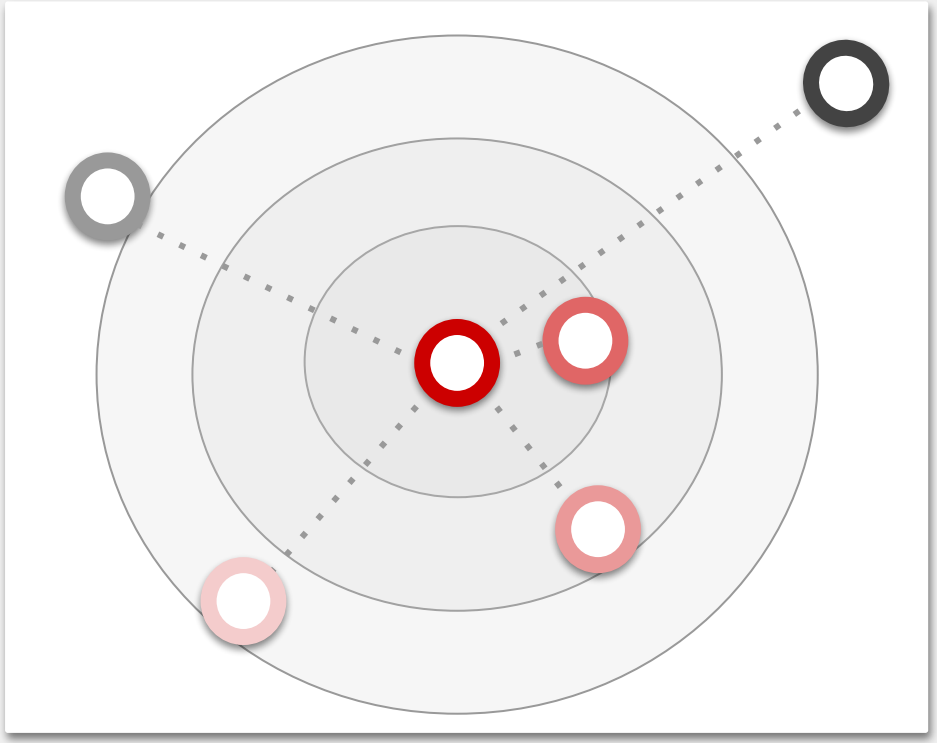


エンジニアはコミュニケーション能力が低い？

当たり前の距離が近い人ほど簡単

コミュニケーションを「**情報の非対称性の解消**」として定義すると、コミュニケーション能力とは、「**より非対称性が大きい状態でも解消できる力**」になる。

気の合う仲間と仲良くできるのは当たり前で、異なる常識を持つ集団（ダイバーシティの高さ）にどれだけ対応できるのかという能力がコミュニケーション能力だと理解できる。仕事で関わるメンバーどうしの認識に隔たりがあると、仕事は進捗しません。**この認識のズレをコミュニケーションの不確実性（通信不確実性）と言います。**



ソフトウェアづくりはコミュニケーションそのもの

機械が理解できるほど明晰な言語化

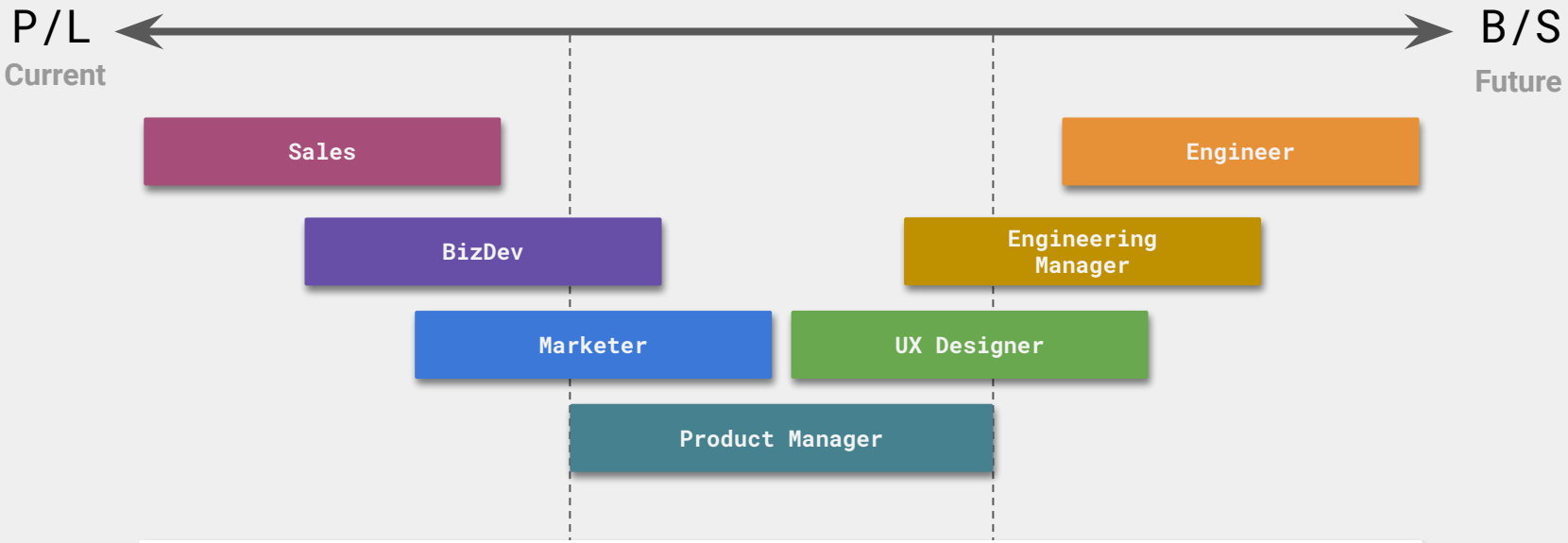
プログラミングは理科系的な数理的なという側面を受け取られがちだが、実際には

- 複数人の認識を揃えながら
- 機械が理解できるほどブレのない明晰な文章を
- 将来の不確実な要求に耐えられるように構成し、
- 共同で継続的に管理する

というプロセス。人文的な側面で言えば、立法行為に近いものだと考えると理解しやすい。コミュニケーションが大事どころか、**「コミュニケーションそのもの」**

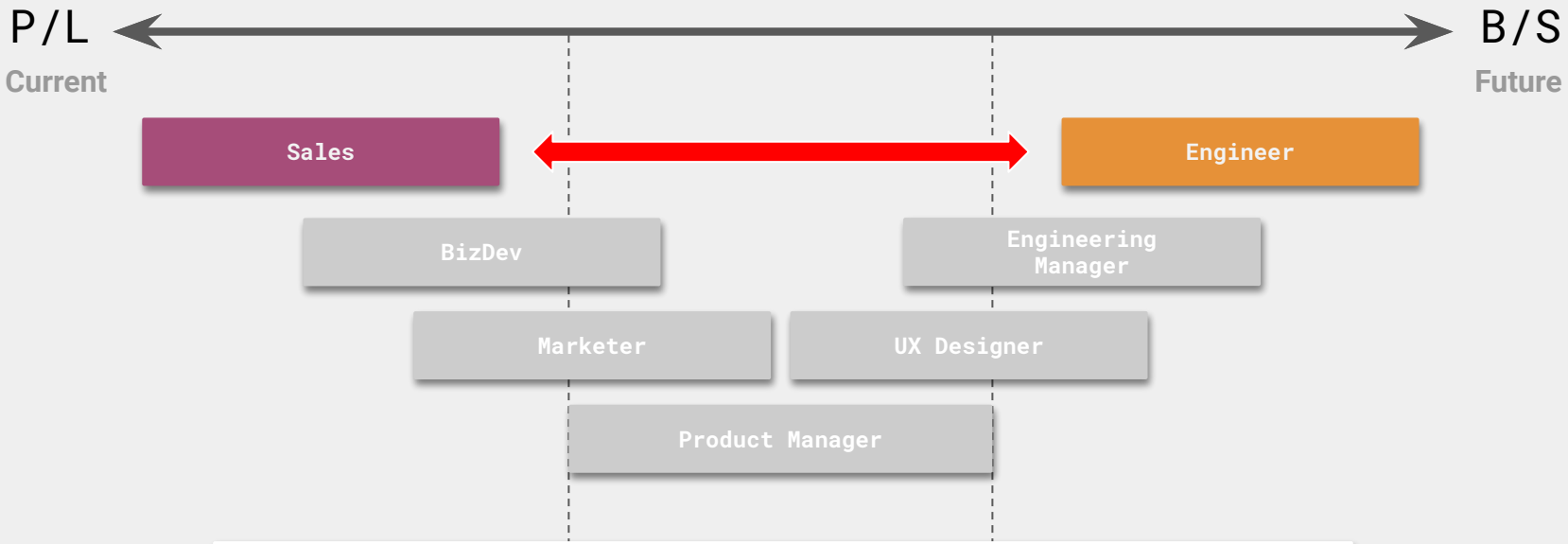
認識・常識のズレ
(コミュニケーションの不確実性)を
減らしていくとソフトウェアが
完成しやすくなる

Skill Spectrum: B2B SaaSの場合



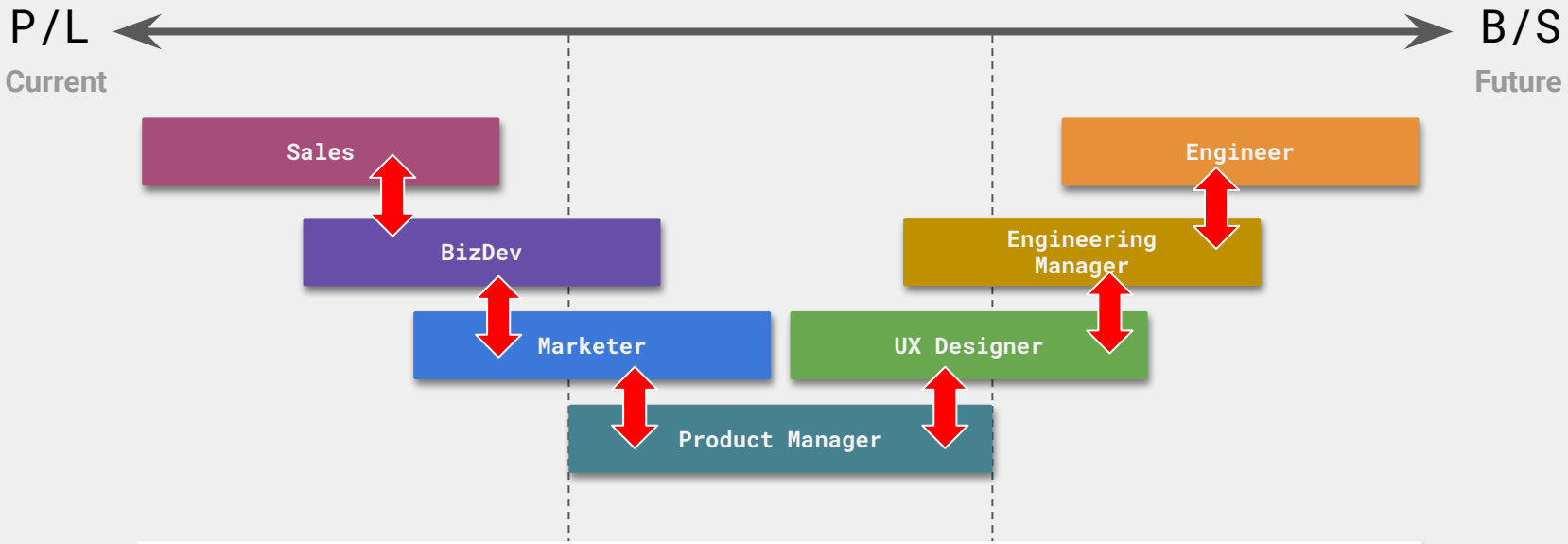
プロダクトの意思決定は、将来投資と現在のビジネスとの両端を持つスペクトラムになる。それぞれに持つべきスキルセットや常識が異なるので、隣接領域の知識を持つものとしかコミュニケーションが成立しない。

Skill Spectrum：B2B SaaSの場合



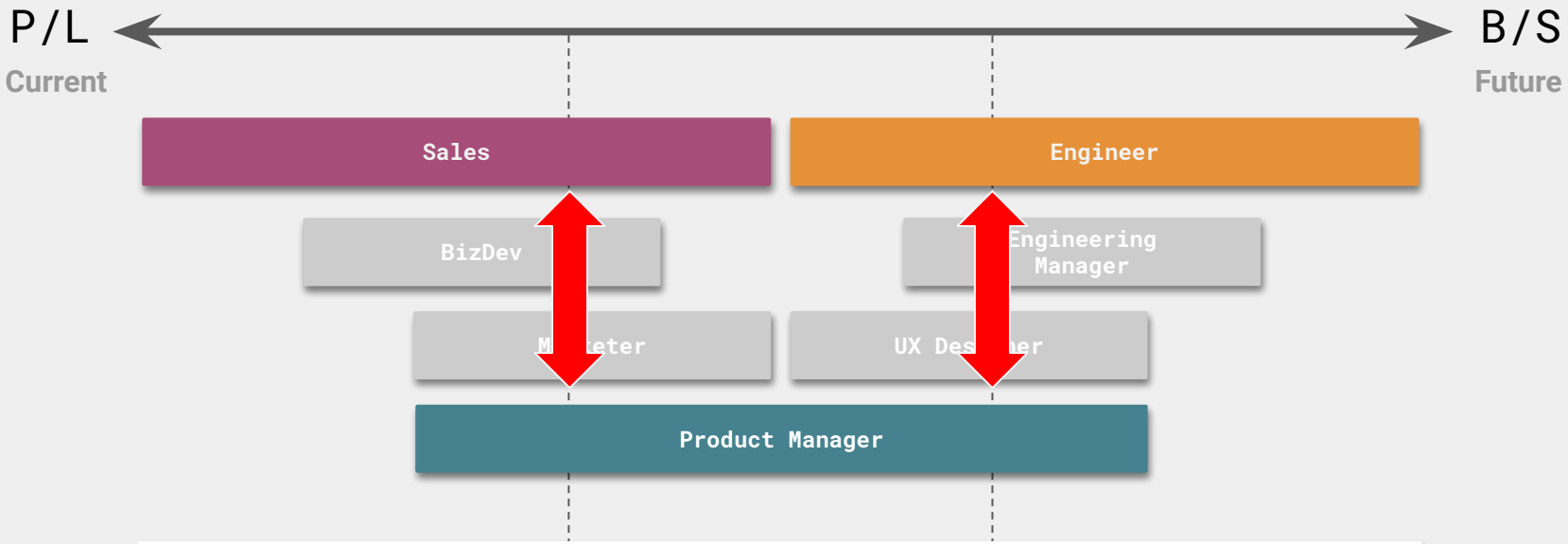
グルーとなる人材がいないほど、情報の非対称性が高くなり、
コミュニケーション上の問題が発生しやすい。

Skill Spectrum：B2B SaaSの場合



グルーとなる人材がいれば、コミュニケーションの健全性は保ちやすい。
しかし、人数が増える分コミュニケーションロスは発生しやすい。
チームとしての成熟に力を入れる必要がある。

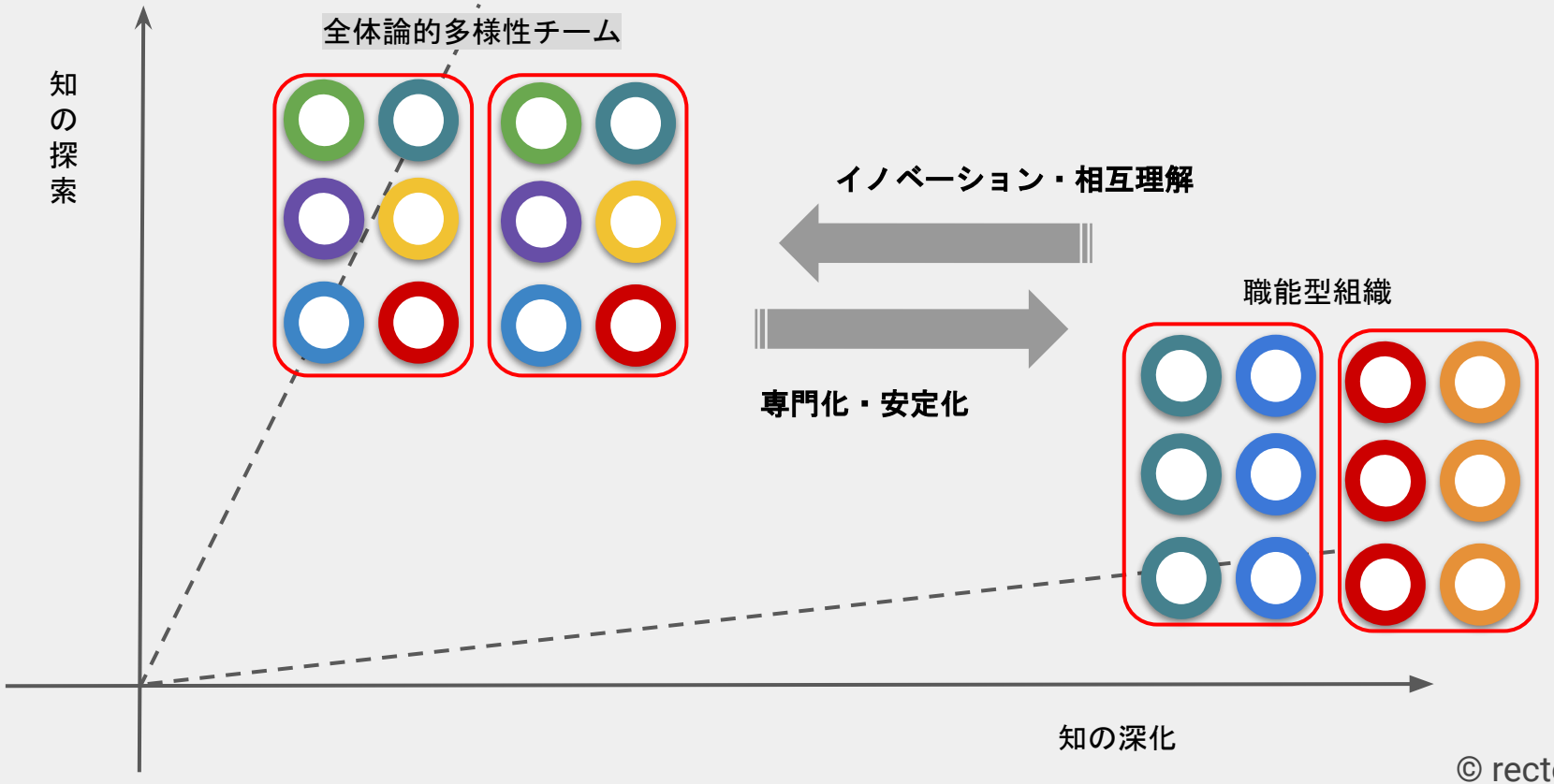
Skill Spectrum：B2B SaaSの場合



人数が少なくても、相互理解ができる領域が広い方が
コミュニケーションの齟齬は小さくなり、うまく行きやすい。
反面、広い範囲のスキルを持つ人材の採用は難しい。

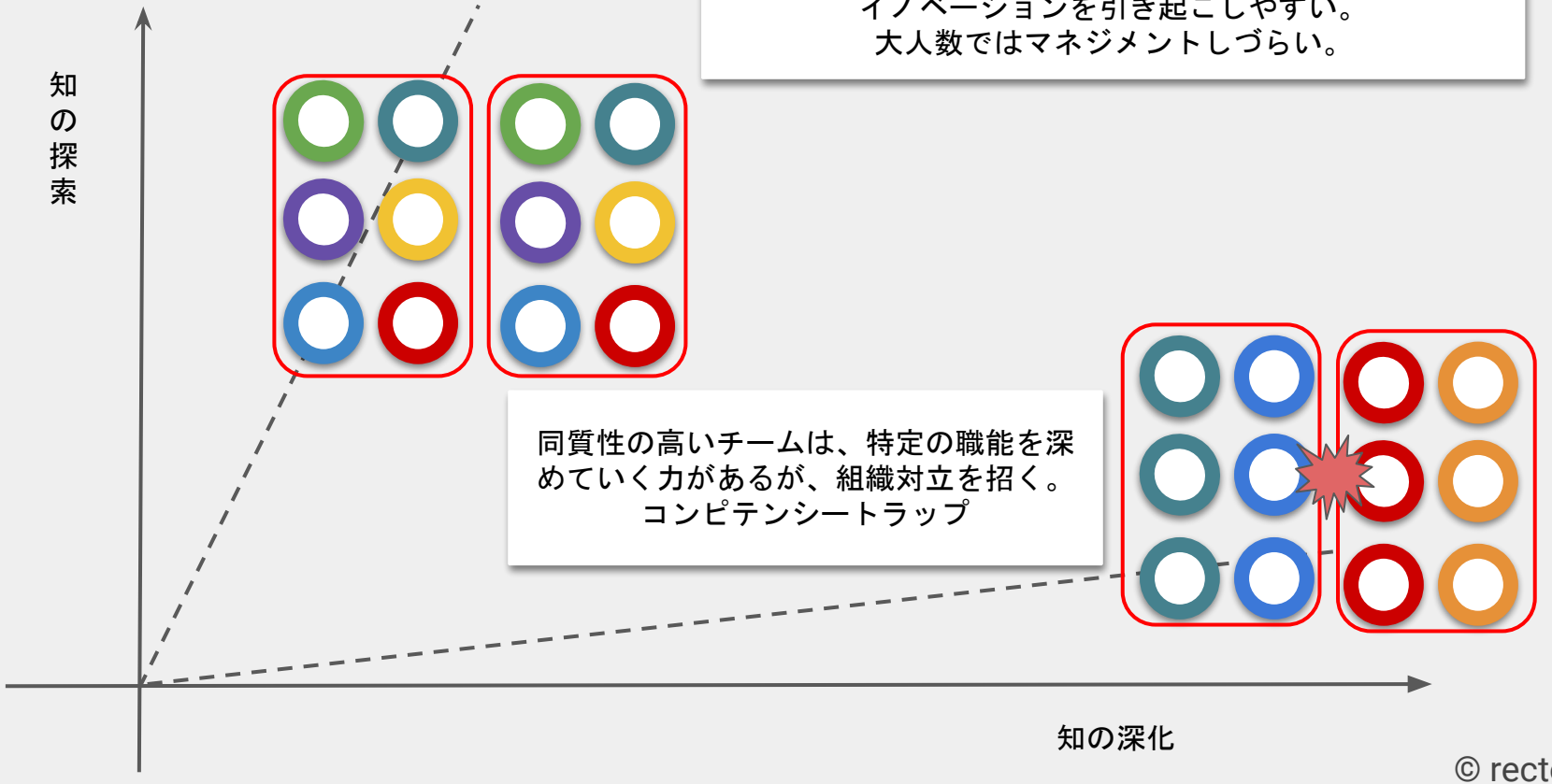
そうなると**一様な人々**でソフトウェアを
作ればよいのだろうか。

Ambidexterity：専門化とイノベーションのレバー



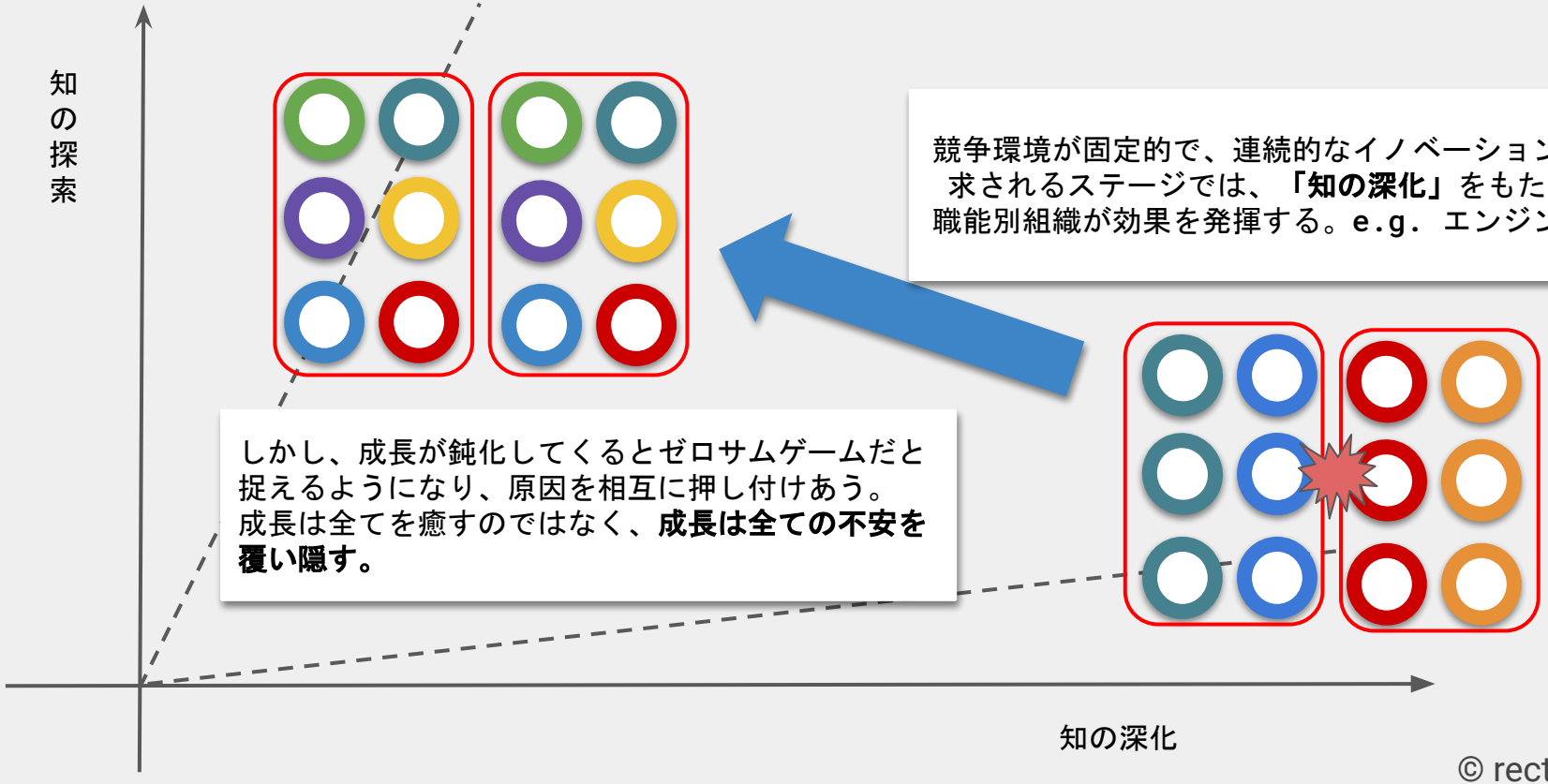
急な変化はコミュニケーション上の対立を産みやすいが、理解も進む

全体論的多様性のあるチームは
イノベーションを引き起こしやすい。
大人数ではマネジメントしづらい。



同質性の高いチームは、特定の職能を深めていく力があるが、組織対立を招く。
コンピテンシートラップ

専門化が進むと部門対立が生まれる。

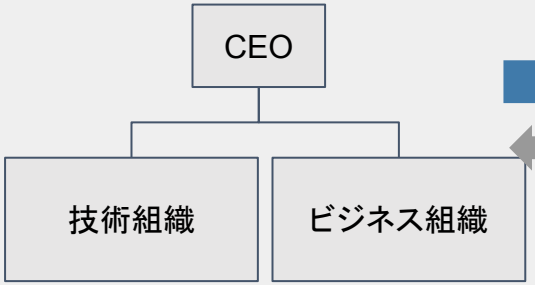


競争環境が固定的で、連続的なイノベーションが要求されるステージでは、**「知の深化」**をもたらす職能別組織が効果を発揮する。e.g. エンジン開発

しかし、成長が鈍化してくるとゼロサムゲームだと捉えるようになり、原因を相互に押し付けあう。
成長は全てを癒すのではなく、**成長は全ての不安を覆い隠す。**

エンジニア組織の変化には一定のパターンが存在する

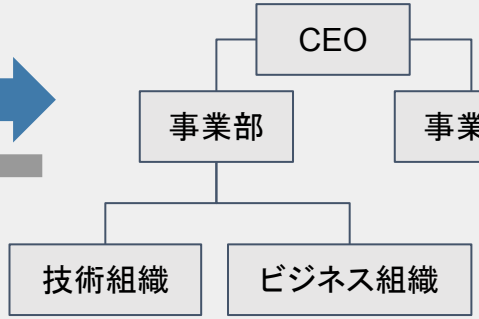
職能別組織



技術組織とビジネス組織を分離し、必要に応じてプロジェクトに振り分ける組織形態。

技術系人材の流動的運用が可能であるが、ビジネスとの間のコミュニケーションがうまくいかず、対立構造になることがある

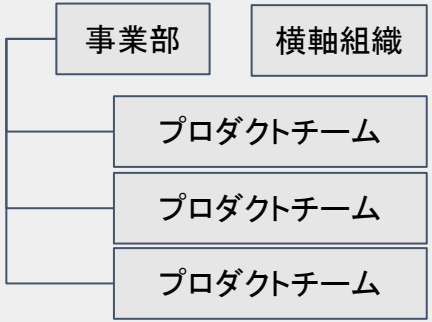
事業部組織



事業ごとに必要な技術組織を持つやり方。

技術人材の育成やキャリア形成に難がある。また、職能別組織と同じような問題も再現しやすい。

ユニット型組織

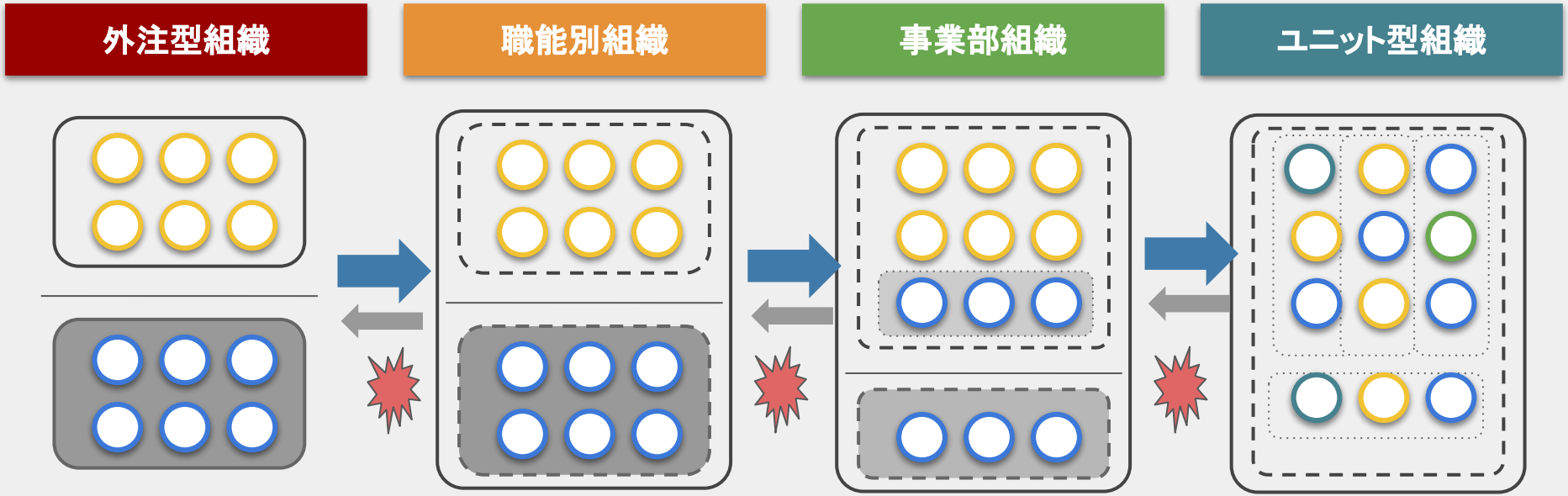


事業ごとに機能横断的なプロダクトチームを組成し、そのチームによって継続的なプロダクトマネジメントを行う。

2000年代後半から流行しはじめ、アジャイル開発の普及とともに日本でも盛んになった。

ソフトウェアエンジニアの「エマルション現象」

DXが進むとはLoB(Line of Business)へソフトウェアエンジニアが溶けていくこと



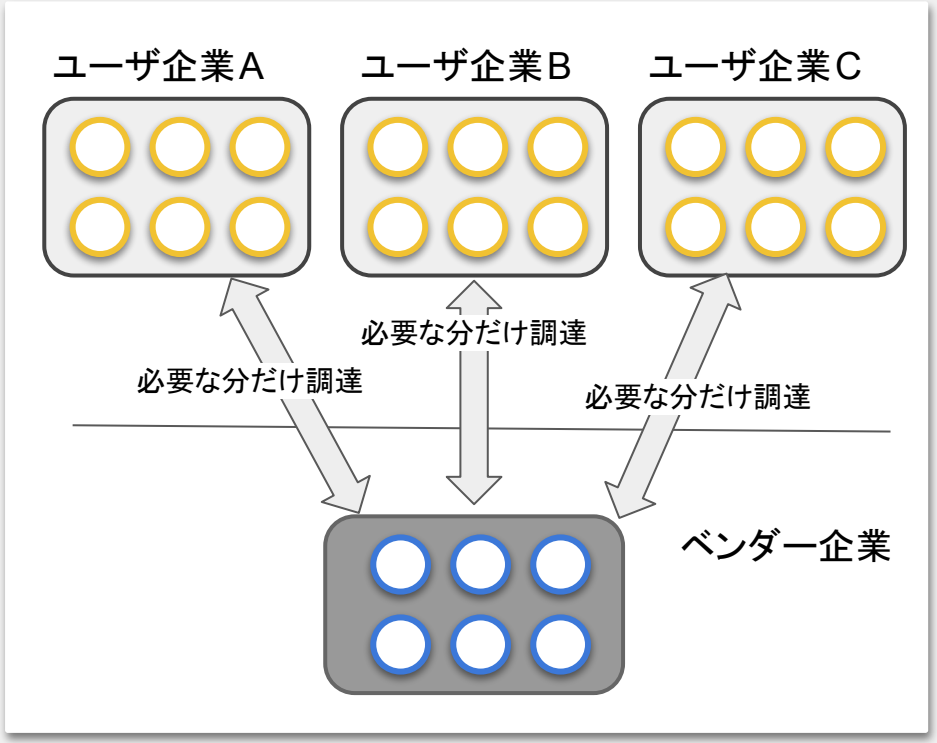
ソフトウェアエンジニアは製造の「原材料」だと思われた

日本のIT最大の錯誤の原点

日本の製造業の強い時代にIT化は進んだ。そのため、全ての新しい産業や技能は、製造業的文脈で捉えられた。製造業小売のバリューチェーンは非常に長く常識の差も大きい

ソフトウェアを作る技能は、「**製鉄**」などと同じく、各社のコアコンピタンスではなく、**調達するべき原材料**ととらえられた。

つまりソフトウェアは、一つの会社に集めて、各社のソリューションを提供する形の方がメリットがあると考えられたのだ。結果、各企業は発注能力もソフトウェアプロダクトのマネジメント能力も欠如してしまった。

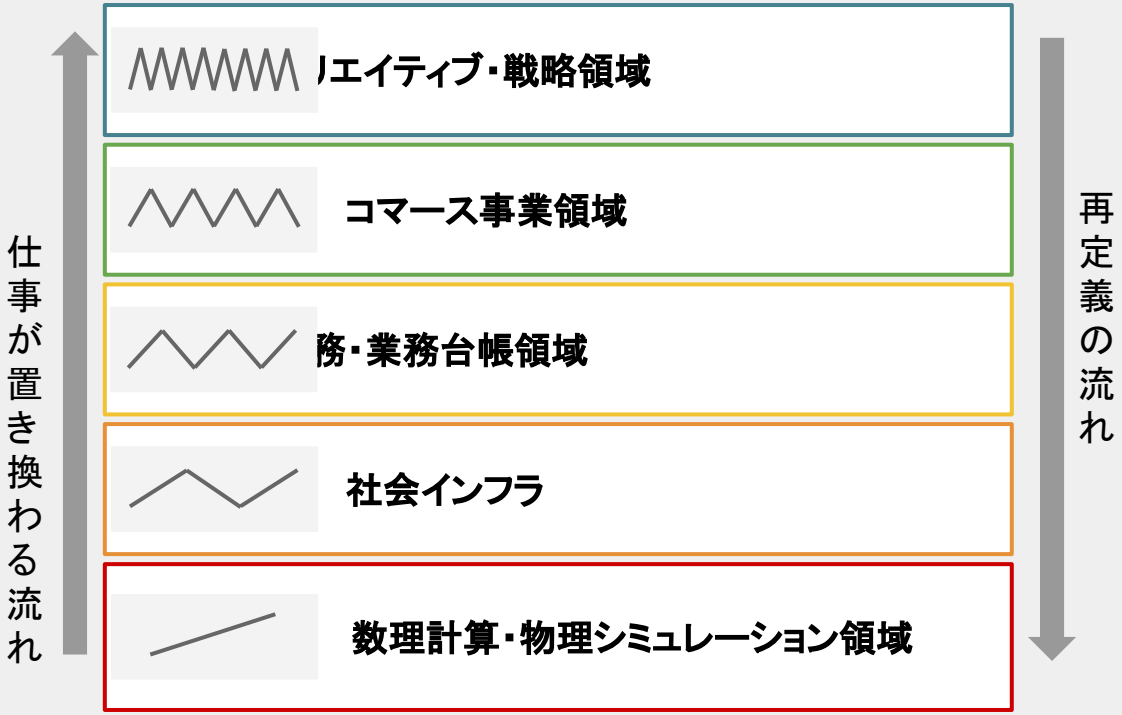


文字なんか書けなくても、専門職に任せればいいよね

古代エジプトの書記官

ソフトウェアエンジニア

何がコンピュータに置き換えられるか



コンピュータに置き換え可能で、かつ人間の単価の高いものから商業的にはコンピュータに置き換えられていく。

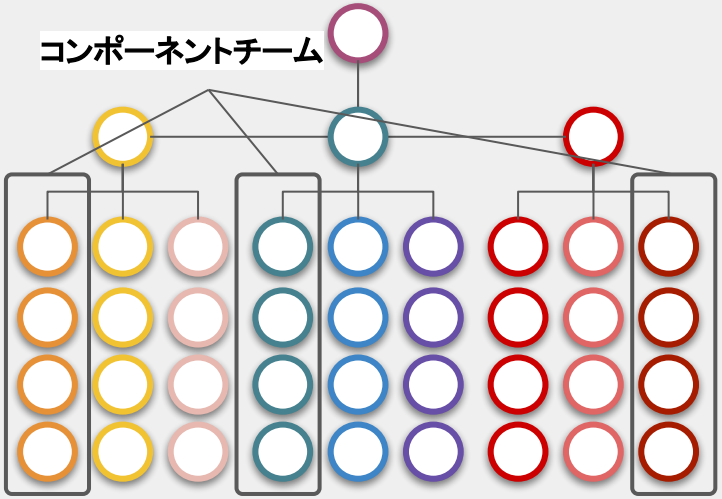
より、曖昧で定義がしにくく、変動の大きい仕事はコンピュータで置き換えるには、コンピューティングリソースや、ソフトウェアエンジニアリングリソースが膨大に必要になるため、置きかわりにくい。

科学技術領域からはじまり、
社会インフラ、ERP、マッチングサービス、
SNSなど。

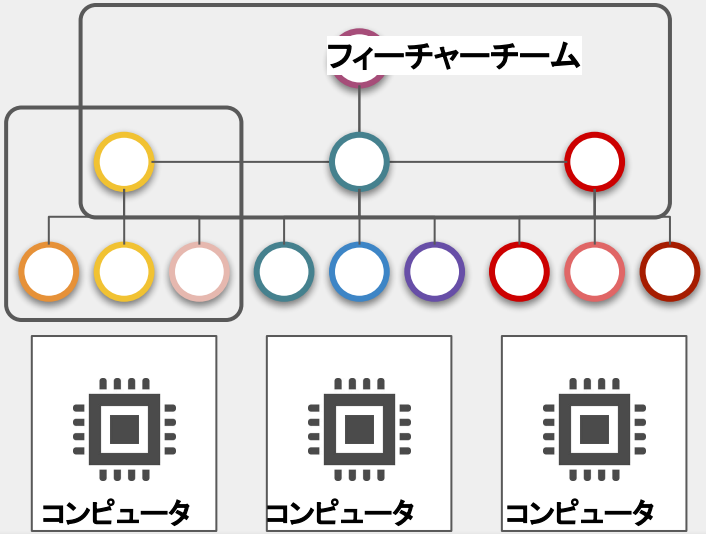
人的資源がコンピューティング資源に変換された企業

ソフトウェアは常に具体を自動化し、人に一段上の抽象を求めるようになる

旧来型組織

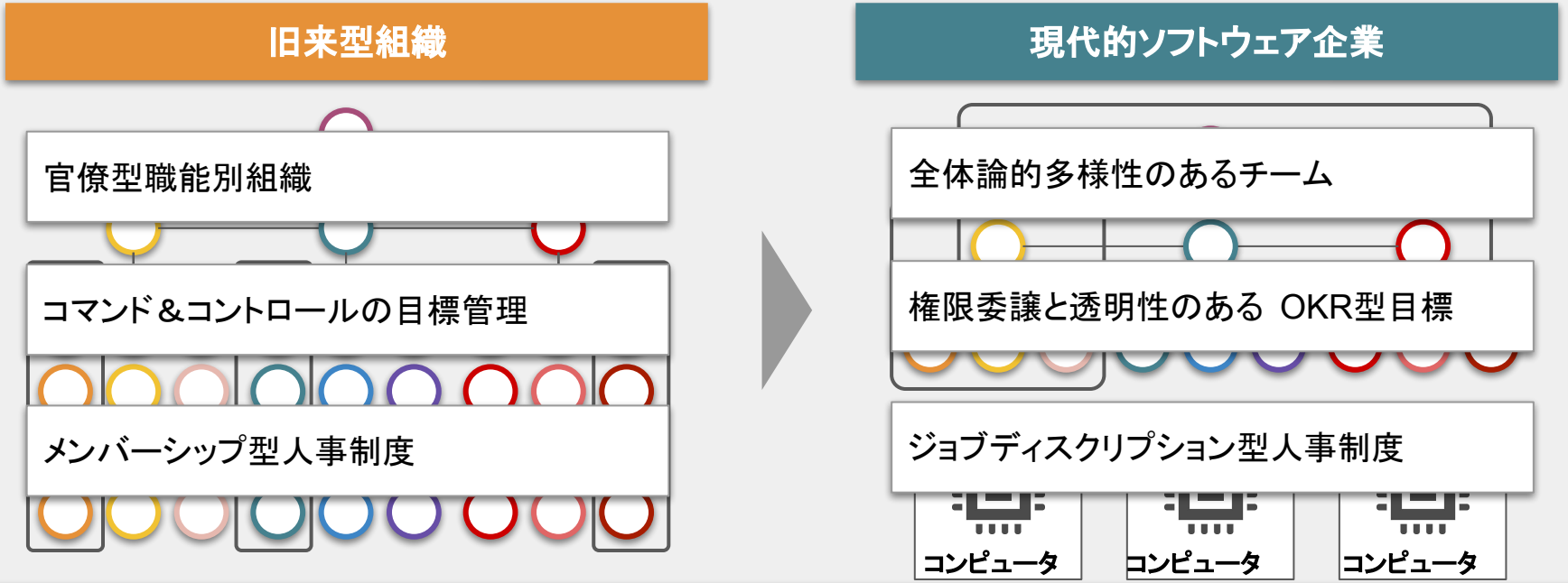


現代的ソフトウェア企業



人的資源がコンピューティング資源に変換された企業

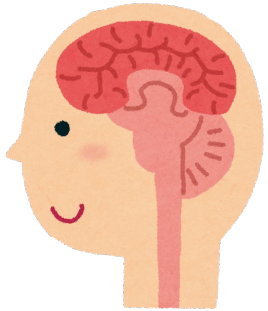
近年のワークスタイルのシフトは、この変換現象の一側面にすぎない。



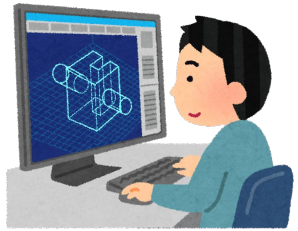
ソフトウェアとコンピューティングの
取り扱い方はもはや**経営学**。

ソフトウェアは
それを修正する
企業のウェットウェア
（文化資本）の顕現。

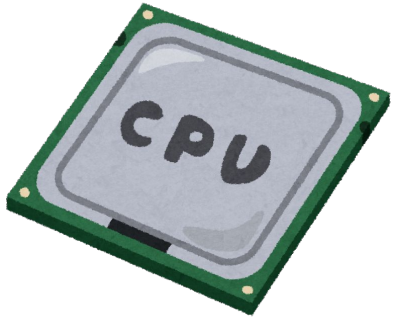
ハードウェアの付属物
ではない。



wetware



software



hardware

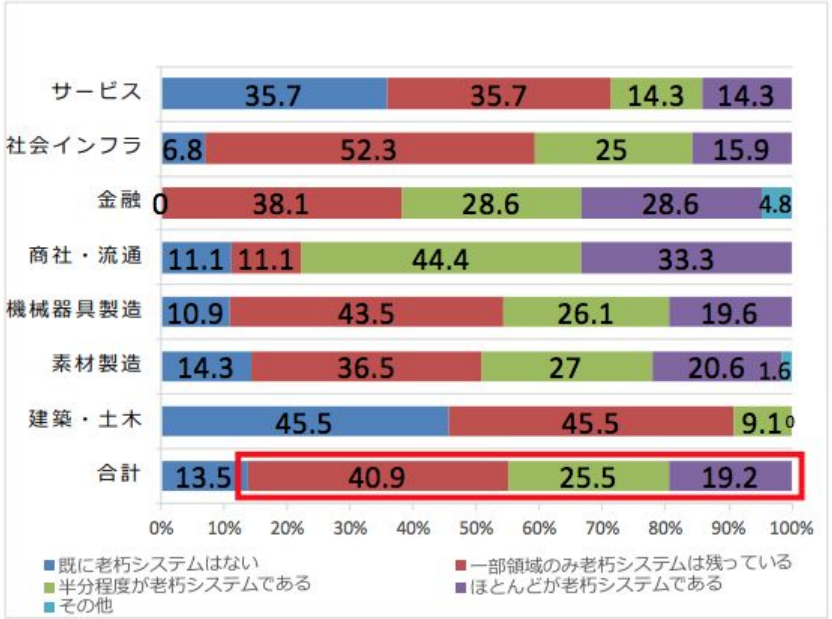
ウェットウェアが欠如すると
ソフトウェアはどうなるのか。

システムに対して、
組織が**コントローラビリティ**を
喪失すると技術的負債と呼ばれる

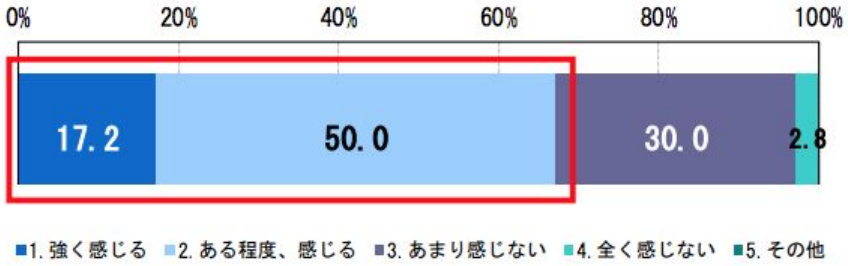
技術的負債。

毎年１２兆円の損害。

約８割の企業が老朽システムを抱えている



約７割の企業が、老朽システムが、DXの足かせになっていると感じている



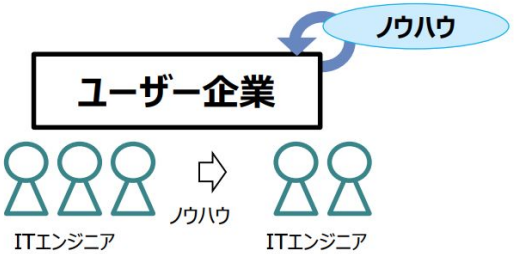
（出典）一般社団法人日本情報システム・ユーザー協会「デジタル化の進展に対する意識調査」（平成 29 年）を基に作成

経済産業省：DXレポートより

◆ ユーザ企業とベンダー企業の関係がレガシー化の一因

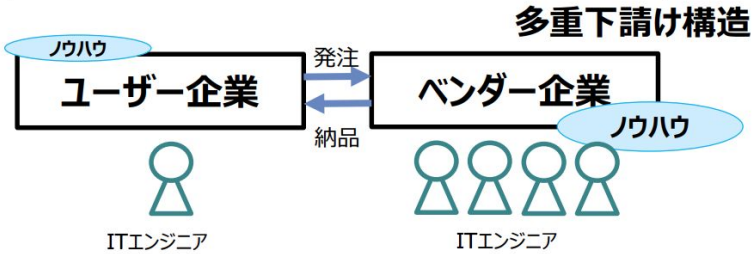
我が国では、ユーザ企業よりも**ベンダー企業の方にITエンジニアの多くが所属**している。

諸外国の場合



- ITエンジニアがユーザ企業に多い
- ノウハウが社内に蓄積しやすい
- 他のエンジニアへのノウハウの伝播が容易

我が国の場合



- ITエンジニアがベンダー企業に多い
- ノウハウがユーザ企業側に残りづらい
- 現場で作業をしている下請け企業にノウハウが蓄積

◆ 有識者の退職等によるノウハウの喪失

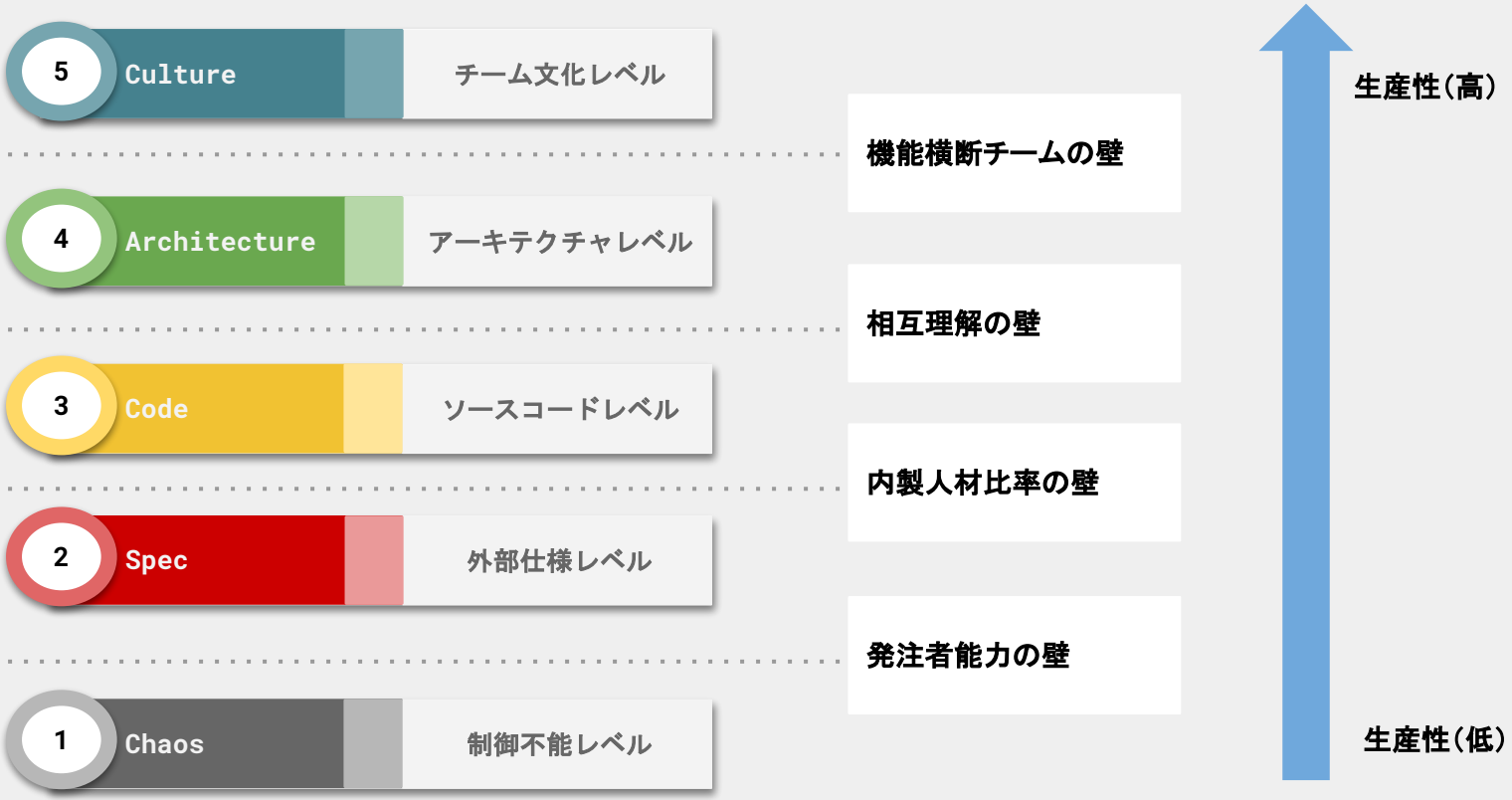
国内企業では、大規模なシステム開発を行ってきた**人材の定年退職の時期(2007年)が過ぎ、人材に属していたノウハウが失われ**、システムのブラックボックス化が進展している。

◆ 業務に合わせたスクラッチ開発多用によるブラックボックス化

国内には**スクラッチ開発**や汎用パッケージでも**カスタマイズを好む**ユーザ企業が多い。
このため、**個々のシステムに独自ノウハウが存在**するようになってしまう。
何らかの理由でこれが**消失したときにブラックボックス化**してしまう。

経済産業省：DXレポートより

System Control Level：システムのコントローラビリティレベル



ただ、古いシステムのことを
技術的負債と呼ぶのではない。

組織が**必要な速度で**
変更できる能力を失うことが**負債**

私のテーマ： ２つのDX

=

DX (開発者体験)

Developer eXperience

DX (企業のデジタル化)

Digital transformation

2つのDXを社会に広めていくため、一般社団法人を作りました。

- レクター代表取締役 松岡剛志氏
- DMM.com CTO 松本勇気氏
- GMOペパボ 取締役CTO 栗林健太郎氏
- VOYAGE GROUP CTO / レクター 顧問 小賀昌法氏
- カーディナル 代表社員 安武弘晃氏
- グリー 取締役上級執行役員CTO 藤本真樹氏
- クレディセゾン 取締役 CTO 小野和俊氏
- ビズリーチ 取締役CTO兼CPO / レクター 取締役 竹内真氏
- メルカリ 執行役員CTO 名村卓氏
- ヤフー 取締役 常務執行役員 CTO 藤門千明氏
- レクター 取締役 広木大地氏

説明しにくいことを説明し、
当たり前水準をあげていくための調査・発信を
していきます。

ご静聴ありがとうございました

広木 大地 @hiroki_daichi