派生開発カンファレンス

# エンジニアリング組織論
## ～不確実性に向き合う組織の現在と未来

株式会社レクター/日本CTO協会 広木大地

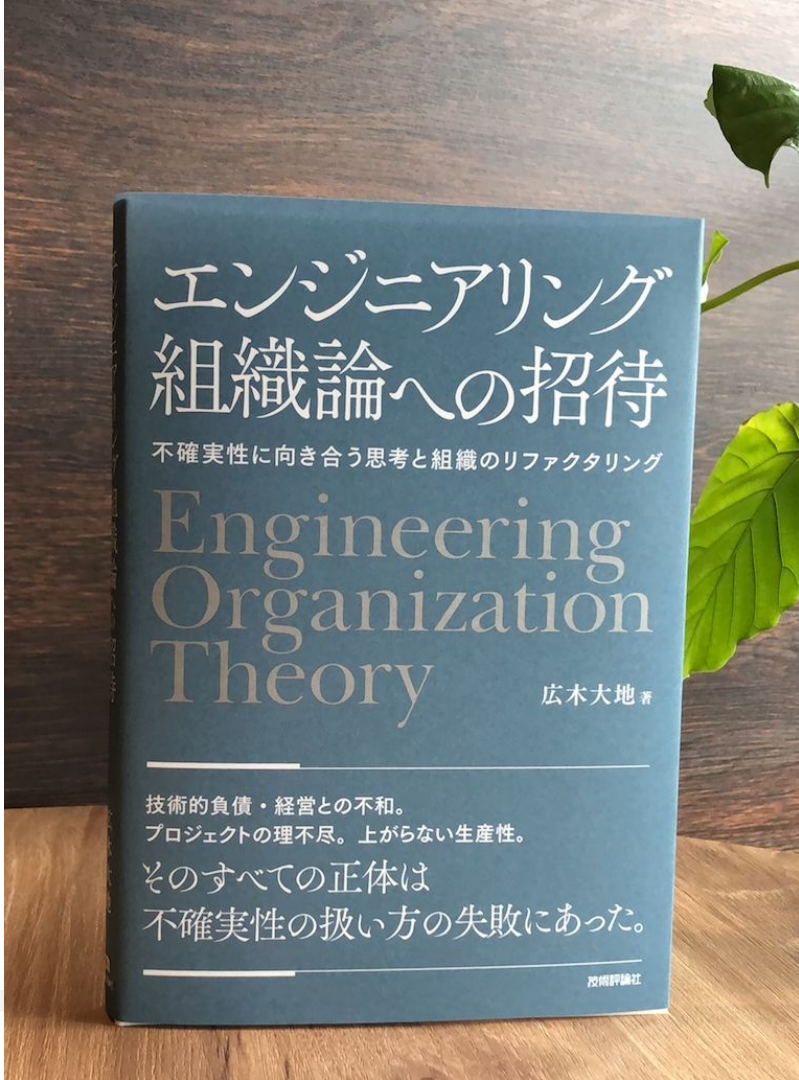
# 自己紹介

## 広木 大地

1983年生まれ。筑波大学大学院を卒業後、2008年に新卒第1期として株式会社ミクシィに入社。同社のアーキテクトとして、技術戦略から組織構築などに携わる。

同社メディア開発部長、開発部部長、サービス本部長執行役員を務めた後、2015年退社。現在は、株式会社レクターを創業し、技術と経営をつなぐ技術組織のアドバイザリーとして、多数の会社の経営支援を行っている。

著書『エンジニアリング組織論への招待～不確実性に向き合う思考と組織のリファクタリング』が第6回ブクログ大賞・ビジネス書部門大賞、翔泳社ITエンジニアに読んでほしい技術書大賞2019・技術書大賞受賞。一般社団法人日本CTO協会理事。朝日新聞社社外CTO。株式会社グッドパッチ社外取締役。



# 技術と経営の両面から
# ソフトウェアエンジニアリングを捉える

# INDEX

本日のアジェンダ

エンジニアリングと不確実性

不確実性に向き合う方法

エンジニアリング組織の過去と未来

皆さんは仕事に対して
どんなイメージを持っていますか


A
決められたタスクを
こなすイメージ


B
皆で話し合って物事を
具体化していくイメージ。

現在の仕事の多くは
何かを”具体化”していくこと

# 仕事は”曖昧”で始まり”具体”で終わる。

仕事の「はじめ」と「終わり」を考えてみる。



仕事のはじめは何も決まっていなくて、モヤモヤとしている。だんだんと方針が定まってきて、実現されてくる。



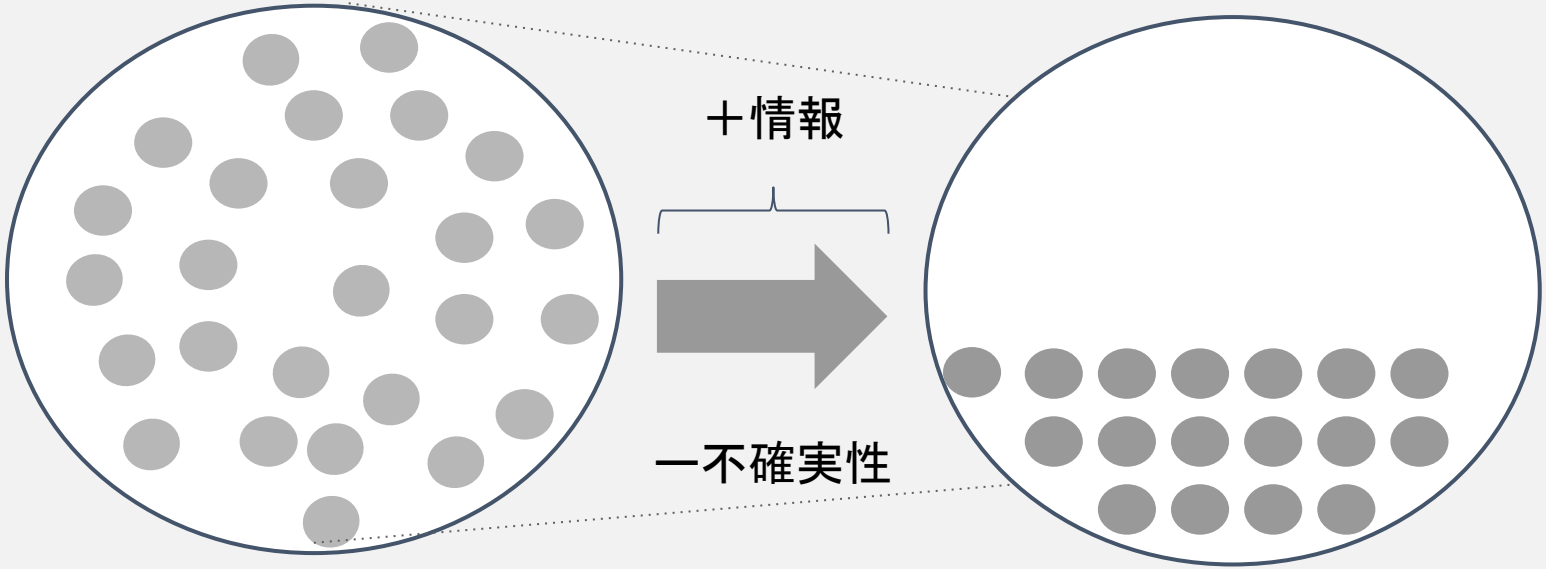
マニュアルでも契約でもソフトウェアでも、具体的で明確な表現物になって実現される。

この過程を「エンジニアリング」と
本書では定義しています。

この「エンジニアリング」の過程で、
減っているものはなんでしょうか。

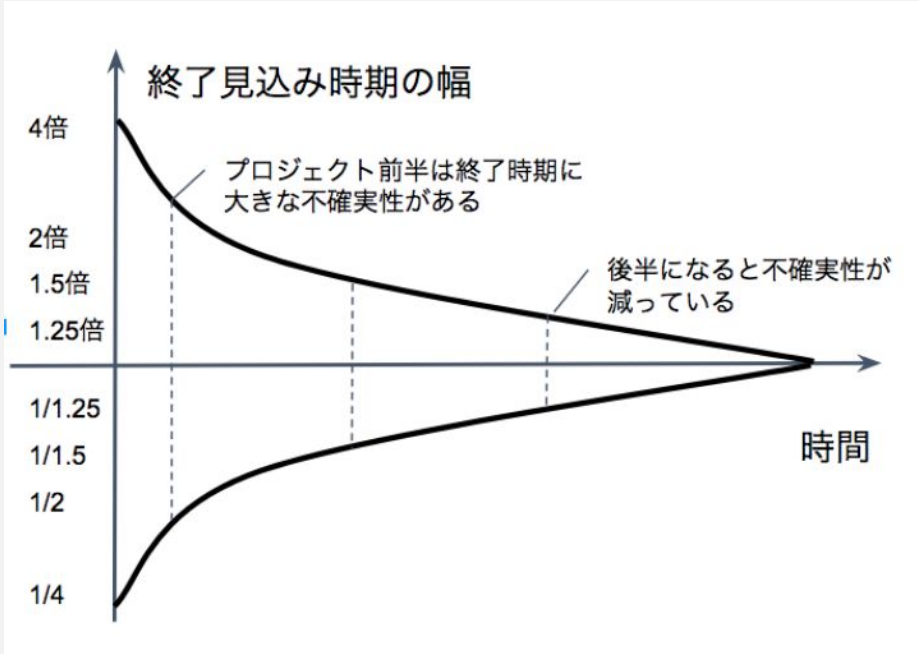
# 「不確実性」の削減こそが仕事の正体

その過程で行われるすべての行為を「エンジニアリング」と呼んでいる。



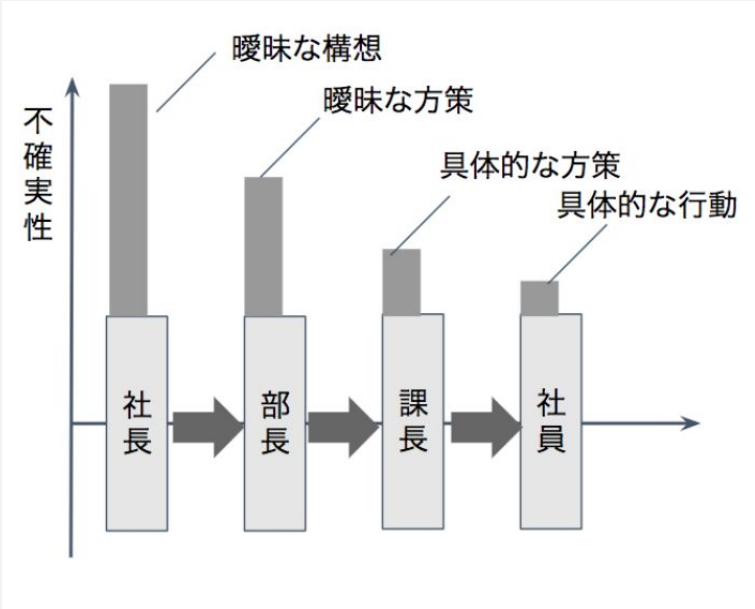
# プロジェクトも不確実性の削減過程

“不確実性コーン”はプロジェクトの本質的な意味を表現している。



# 組織も不確実性の削減するメカニズム

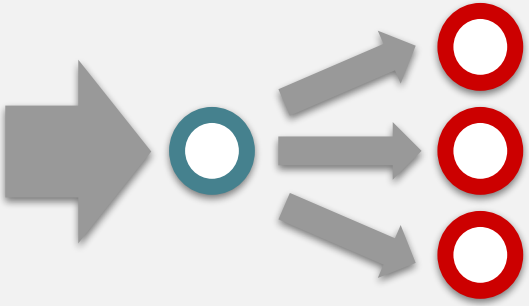
社会の不確実性を”喰って”成長する装置が株式会社という生命の構造



# 不確実性を大きく下げるチーム=生産的

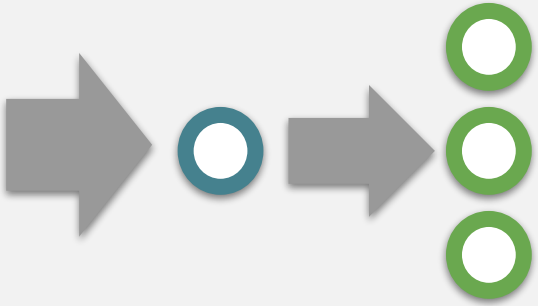
不確実性という補助線をひくと生産的なチームとそうでないチームの形が見えてくる。

## ❶マイクロマネジメント型



細かな指示まで上司が行う必要があるチーム

## ❷エンパワーメントチーム型

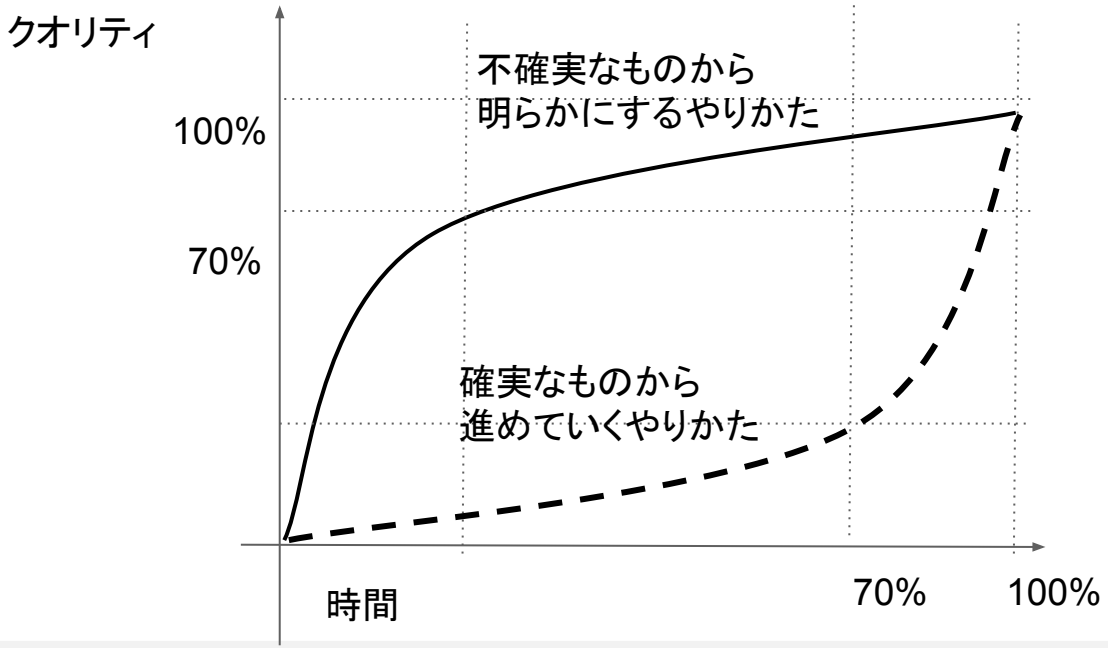


曖昧な状態でも行動でき、任せられるチーム

仕事で行っているのは
「不確実性の削減」だと捉えるとうまくいく

# より不確実なものから手をつける。

クライアントへの確認やイメージの共有、市場リスクなどより危ういものから確認していく



# では、仕事における
# “不確実性”の究極的な
# 発生源は何か？

# 人間が本質的にわからないものは2つ

**それが不確実性の発生源となっている。**

## 1 未来：環境不確実性

未来のことは誰も知り得ない。だから、不確実性を生み出す。マーケットの不確実性。

## 2 他人：通信不確実性

他人の頭の中を覗き込むことはできない。だから不確実性を生み出す。コミュニケーションの不確実性。

不確実性に人が向き合うとき
どう感じるか。

# 不確実性に対して”たたかう”か”にげる”

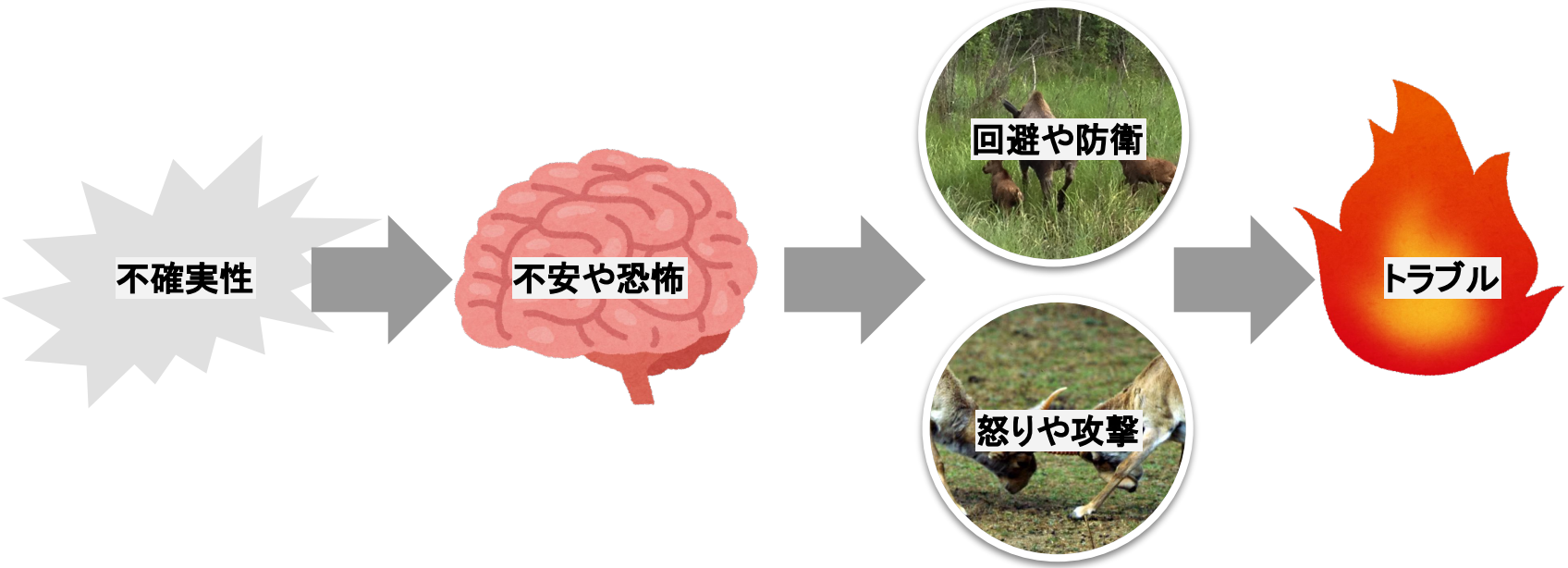
or

**攻撃的行動**
Fight

**回避的行動**
Flight

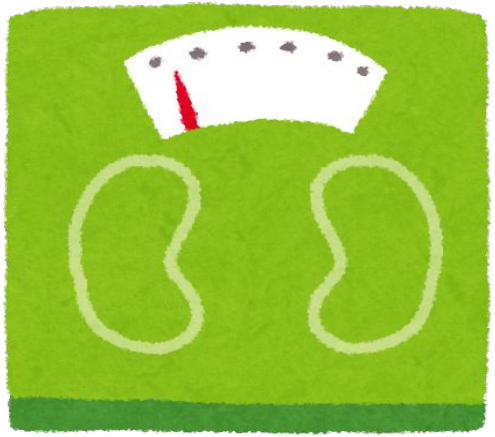
# 不確実性が脳にストレスをかける

不確実性に向き合おうとすると怒ったり、逃げたりする反応が生まれそこからトラブルが生じる



# 日常における不確実性との向き合い

現在地点の事実を知るということに対しての”忌避反応”



体重計に乗る

テストを受ける

# 仕事における不確実性への向き合い

早めにやった方がいいはずなのに、つい後ろ倒ししてしまうことはないか。

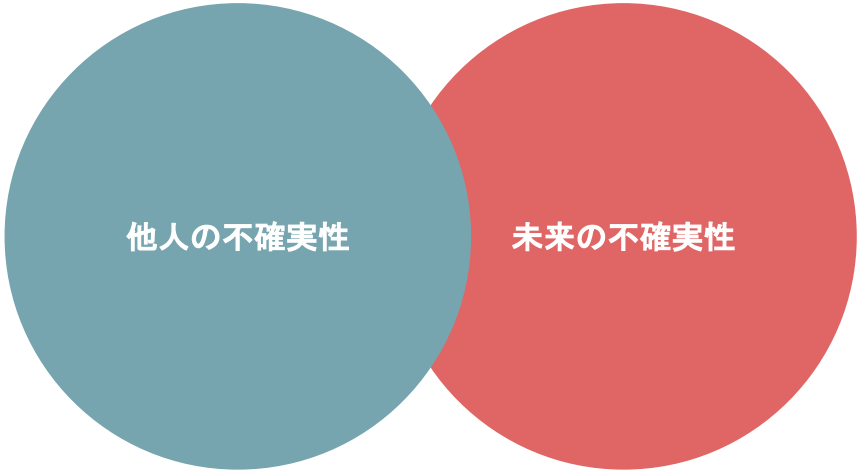
上司に報告をして
フィードバックを受ける

すれ違いのある部門との
会話をして懸念を払拭する

これらに向き合うとき、
人は「不安」を感じます。
いろんな方法で回避したり、
場合によっては怒りを覚えたりします。

ではどのようにして「不確実性」に
向き合うのか。

# 2つの不確実性とどのように向き合うか。



# 他人の不確実性への向き合い方

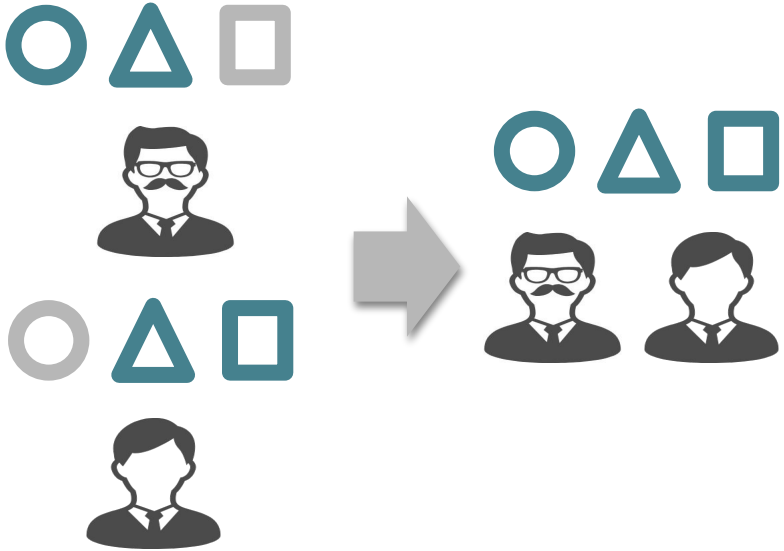

①心理的安全性のあるチーム

②疎結合なシステムと組織

③システムの不可視性の可視化と対話

# コミュニケーションとは何か？

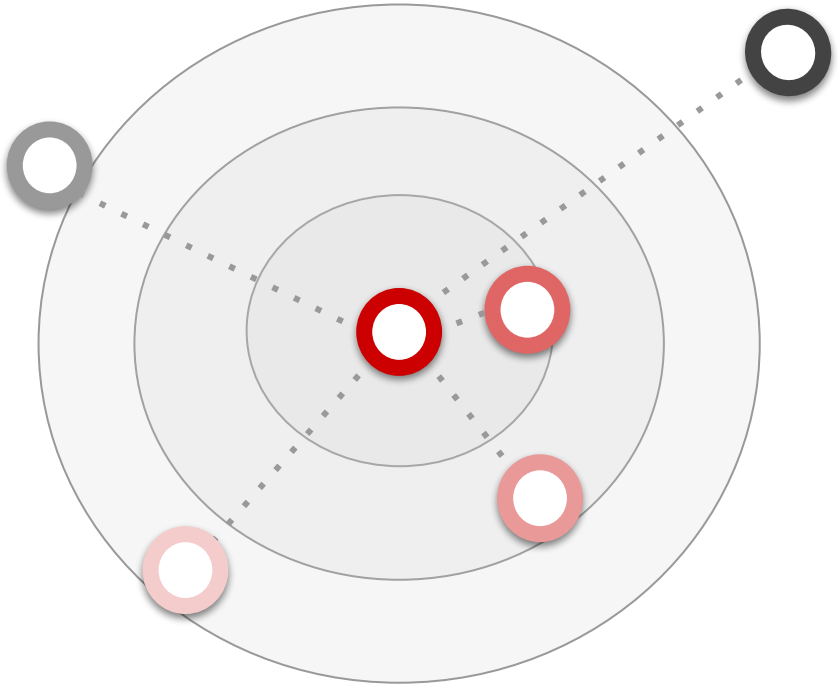
情報の非対称性の解消こそがコミュニケーション



# コミュニケーション能力とは

情報の非対称性を解消できる力のこと

常識の距離が近いほど、会話は成立しやすい。
遠いほど難しい。より多くの非対称性を乗り越える力がある人が「コミュニケーション能力」がある。



# ソフトウェアとコミュニケーション

ソフトウェアの人文的側面に着目すると立法プロセスのようなコミュニケーションそのもの

複数人の認識をそろえながら

機械的に処理できるほど明晰な文章を

将来の不確実な変化に耐えられるように
構成し、

共同で継続的に管理する営み

ソフトウェア開発

立法プロセス

# 誤解されがちな心理的安全性

“仲がよい”/”アットホームな”とは全く異なる概念

**心理的安全性**とは

*‘’対人リスクを取っても問題ないという信念がチームで共有されている状態*

OK! 「ここにいても」大丈夫

OK! 「意見」を言っても大丈夫

OK! 「間違い」を認めても大丈夫

OK! 「助け」を求めても大丈夫

# 心理的安全性は一人で創れるものではない

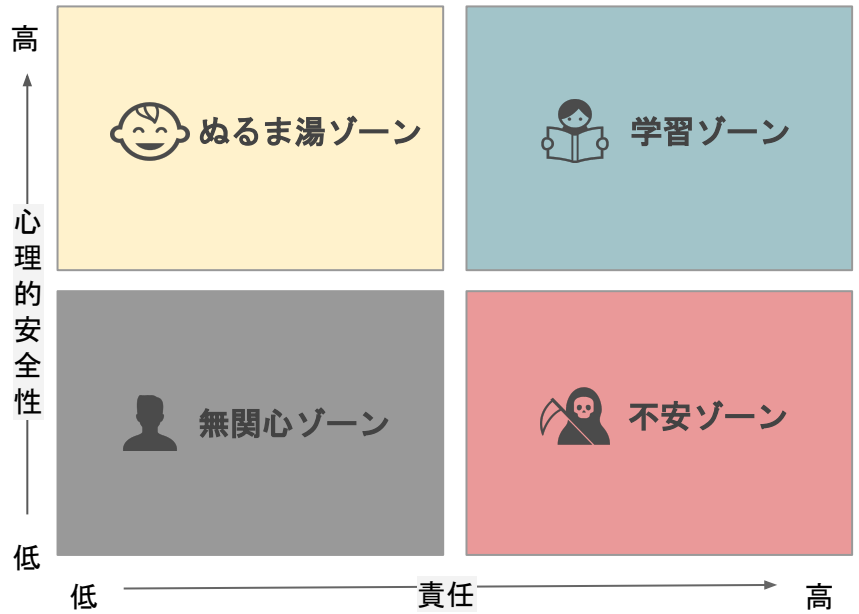
アサーティブな言語化能力と問題解決を適切にファシリテーションすることで生まれる。

優しすぎて意見を出せない上司

激しく捲し立てて
言語化できない部下

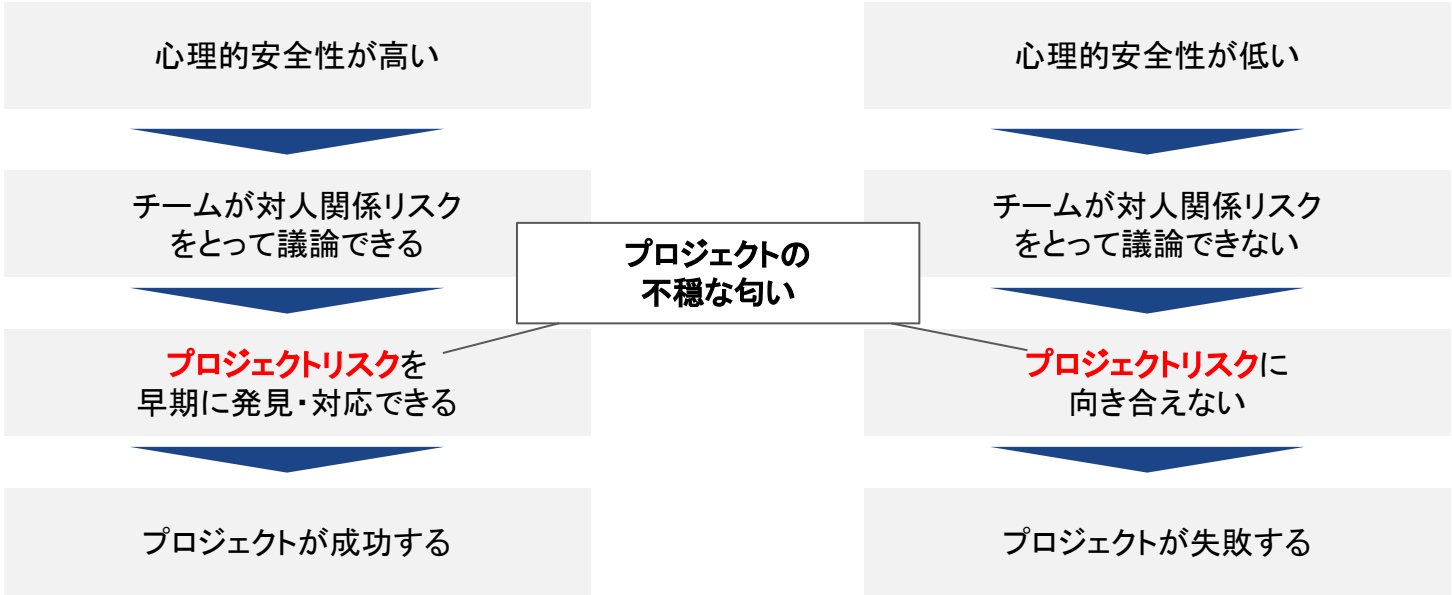
# 心理的安全性の四章限

如何に意見が言えても、無責任で自分本位の意見だと意味がない。



# 心理的安全性はなぜ生産性を上げるのか

心理的安全性はプロジェクトリスクを早期に曝露させる。



# コード修正の心理的安全性を高める

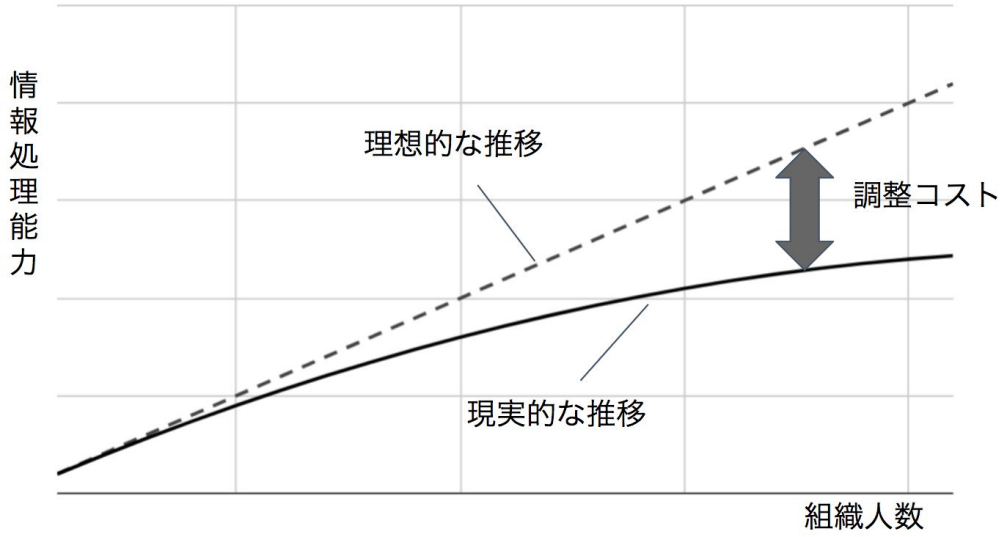
エンジニアの立場から見るとリリースで胃が痛くならない＝開発者体験の良さ

# 疎結合なシステムと組織

構造的に型をつくることで、そもそも必要なコミュニケーション量を下げる

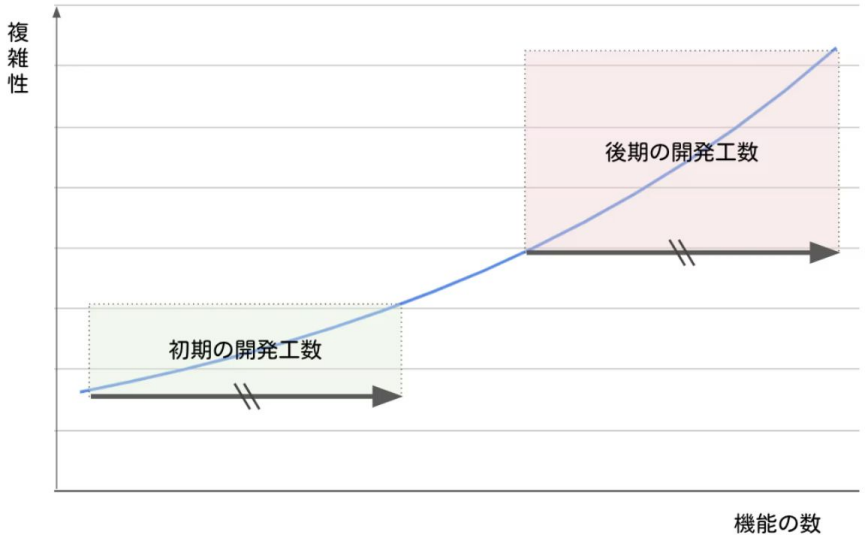
# 組織のコミュニケーション量

組織人数に対してリニアに情報処理能力はスケールしない



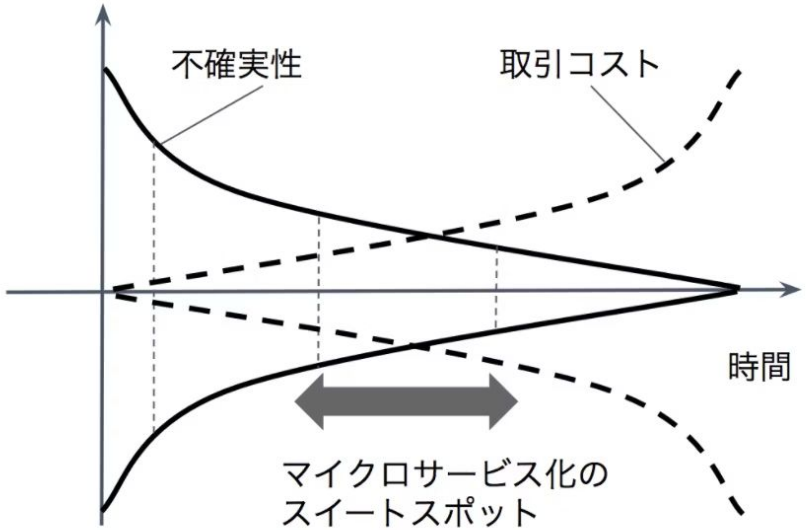
# 複雑性の増大に伴う開発工数の増加

ソフトウェアの複雑性は、機能の組み合わせの数に比例する。



# マイクロサービス化のスイートスポット

マーケットの不確実性が下がり、分割によって得られる効率が調整コストを超えるタイミング



ソフトウェアの本質的な困難性は
それが見えないことにある。

# ソフトウェア開発の本質的な難しさ

## ブルックスの名著「人月の神話—狼人間を撃つ銀の弾はない」より筆者によるまとめ

### 1 複雑性

ソフトウェアはその規模に対して複雑さが非線形に増大します。基本的にソフトウェアというものは一度作られたものであれば、再利用できます。そのため、どんな人工構造物よりも複雑になります。

### 2 同調性

ソフトウェアは、それ単独で意味を成すものではなく自分自身を動作させるハードウェアや、ネットワーク、OS、その他のシステムなどと連携しながら、同調することで初めて意味をなします。

### 3 可変性

ソフトウェアは、文化・業務・経済環境・商習慣、顧客行動などさまざまな点に連動して、変わり続ける必要があります。当初の計画通りのシステムが出来上がったとしてもそれは利用者の要望により変わり続けます。

### 4 不可視性

ソフトウェアは、目に見えません。抽象的な概念の相互関係であり、それは一般に技術者にしか理解できない言語で記述されます。そのため、ソフトウェアの構造を他の構造物とは違い見ることができません。

# 技術的負債の四象限

## ソフトウェアの性質を可視性と価値の正負によって4つに分類する

### 見えないことが技術的負債の性質

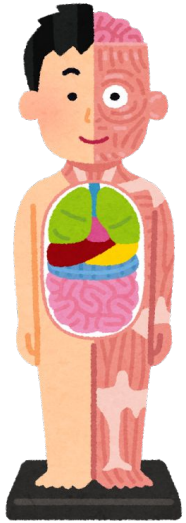
クルーシュテンの定義によると、技術

的負債の性質はソフトウェアにおいて「見えない」「マイナスの価値」のものとなる。


出典：フィリップ・クルーシュテン(2012)

# 見えないことによる情報の非対称性

エンジニアには悪いところがよく見える。非エンジニアには、表面しか見ることができない。



情報の非対称性が
コミュニケーションを不全にする。

# 見えてしまえば、非機能要件のリスト

『ブレーメンの音楽隊』の怪物の様なもの

影の時はよくわからないものバケモノ

見えてしまうとタスクの積み重ね

# 情報非対称性が技術的負債現象の原因

エンジニアは見える、非エンジニアは見えない。この情報非対称性：そのギャップを埋めるために生み出された言葉が技術的負債。このコミュニケーションギャップが技術的負債を「問題化」させるまで放置させてしまう原因。

# 未来の不確実性への向き合い方


①リアルオプション戦略と仮説検証

②経験主義的プロセス

③頻度がつくり出す質

# 仮説法とは何か

仮説検証という言葉が人口に膾炙しているが、仮説という考え方が理解されていない

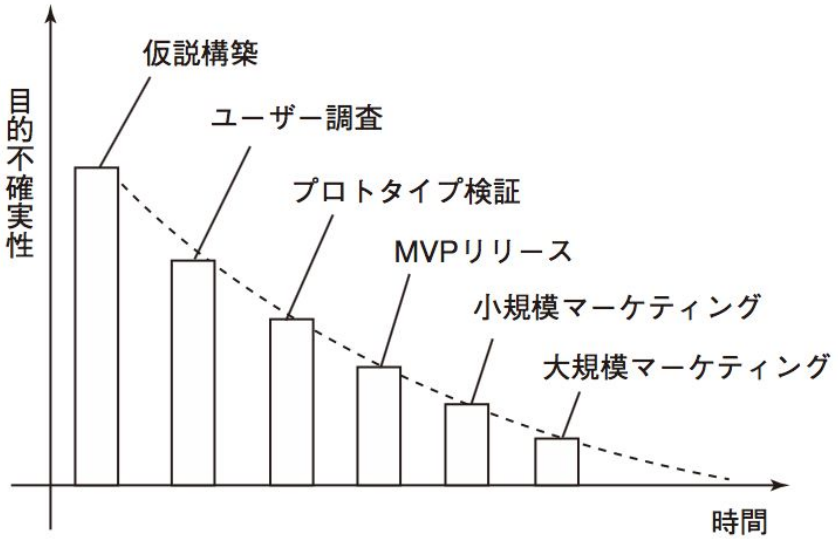
わずかな痕跡を元に

大胆な推測を行い

それが仮に真である場合
証拠をさがす行動をする。

# リアルオプション戦略

“わからない”部分を最小のコストで実験し、明らかにすることで後出しジャンケンする。



# 未来の不確実性への向き合い方


①リアルオプション戦略と仮説検証

②経験主義的プロセス

③頻度がつくり出す質

# 理性主義と経験主義

わからないことを行動を通して経験し、それを知識として徐々に突き止める経験主義

## 理性主義的な問題解決

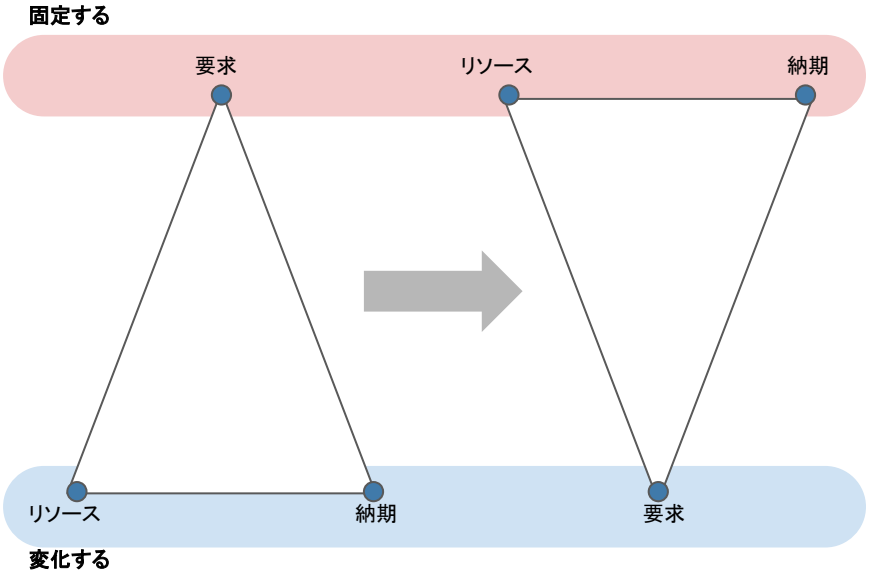
わからなかったので、もう一度考えてみよう！

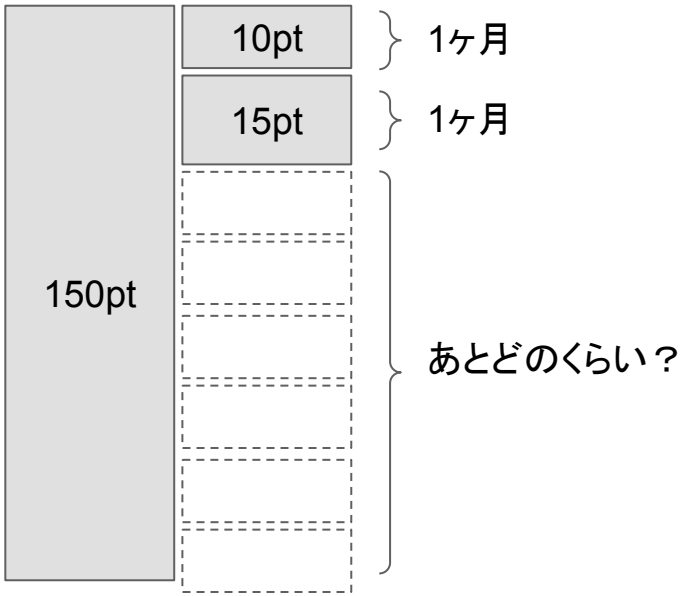
## 経験主義的な問題解決

わからなかったので、次に何をしたらわかるのか考えよう。

# 経験主義的プロセス

固定するもの/変化させるものを反転させて「経験」を得られるようにする。



# 経験主義的プロセス

机上の計算の積み上げによって予定するのではなく、実験から予測するプロセスに。

# 相対見積もりによってバイアスを取り除く

タイムスロットを固定し、消費ポイントから期日を予測することで相対見積もりが可能になる

スケジュールコミットが厳しい環境ほど悲観的に見積もる。

コミットが緩い環境や担当者の責任感が強いほど楽観的に見積もる。

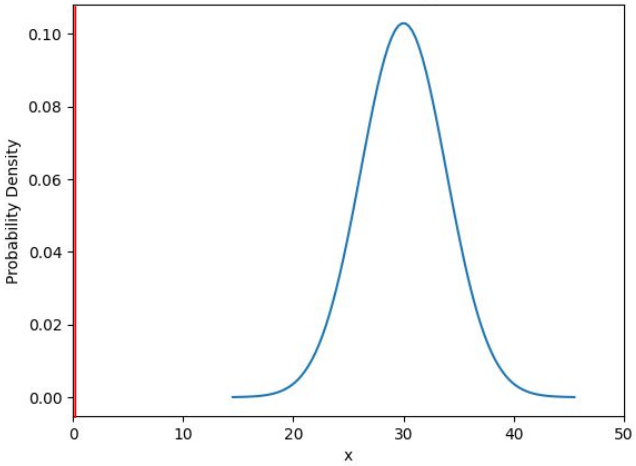
リンゴ1個分

あの猫は3個分くらい

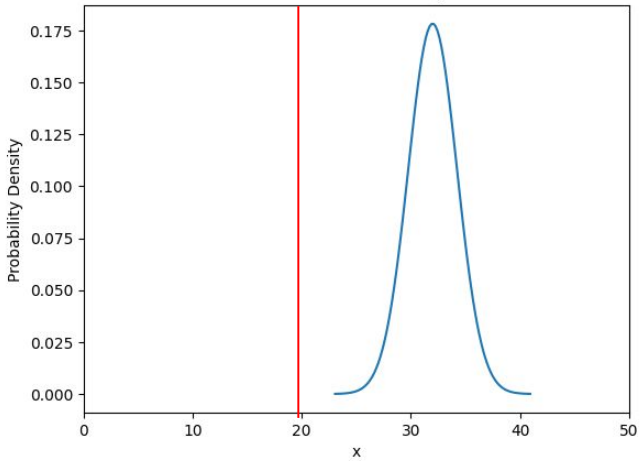
# 見積もりは”幅を持った予測”にする。

不確実性をファーストクラスとしてプロジェクト管理の対象にする

**プロジェクト初期**




**プロジェクト後半**



# 未来の不確実性への向き合い方


①リアルオプション戦略と仮説検証

②経験主義的プロセス

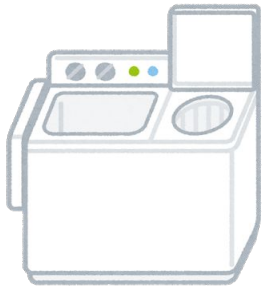
③頻度がつくり出す質

# ドラム式自動洗濯機の”良さ”

新しい「当たり前」となる習慣の価値は、使う前にはわからない。

手洗いの方が
よく落ちるよ

全自動は
信用できない

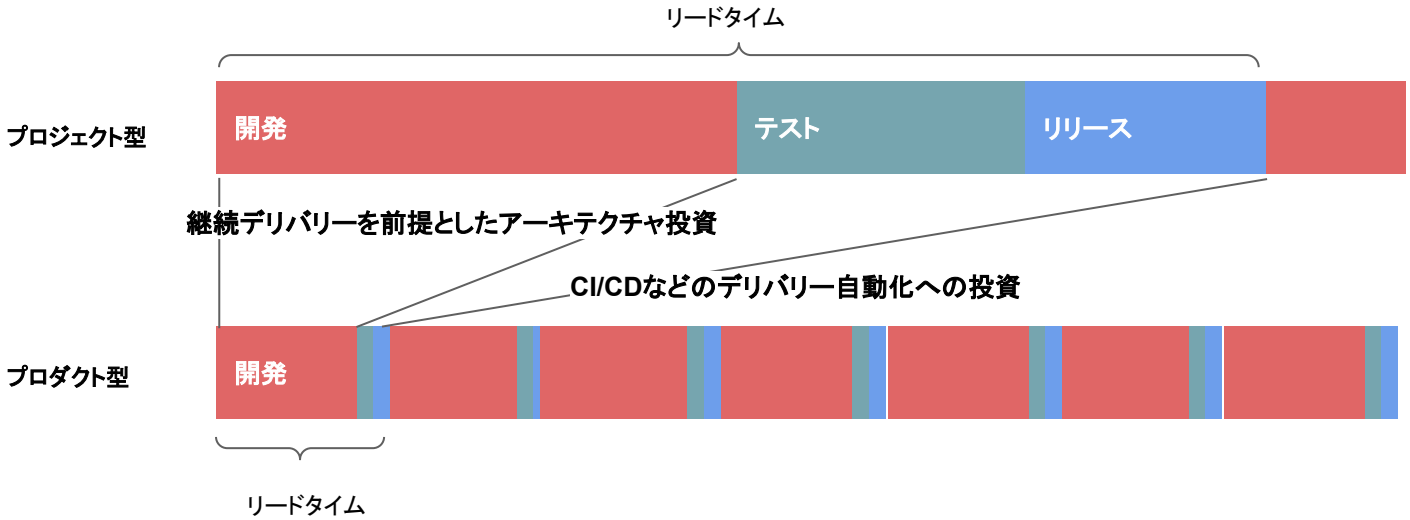


干すのは
手間じゃない

洗剤は
自分で入れる

# フロー効率を高める技術的投資

継続的デリバリーを前提に自動化や効率化を徹底的に進めることで品質と価値効率を最大化する




## 頻度は「質」に転化する。

# フェイルファストの原理

シリコンバレーでは、

**Fail Fast (早めに失敗しろ)**という言葉がよく言われる。

さまざまな試行錯誤があってこそ、イノベーションを生むのだという話として表層的に捉えられていることが多い。

モダンなソフトウェアプロダクト開発においては、まさにこの「早めに失敗する」という原理に基づいて様々な技術/文化が発達し、その結果強靭で強力、高品質なプロダクト開発が行われている。

# 失敗から学ぶ ≠ とりあえずやってみる

失敗するかも知れないことをやるというよりも、失敗そのものを積極的に生み出す

早く失敗しろ

失敗から学ぶ

何でもかんでも
闇雲にやる

早く失敗しろ

失敗から学ぶ

失敗が早期であれば
プロジェクトの被害は軽微

# フェイルファストの原理から導かれる文化

早期に”低コスト”で失敗を引き起こす仕掛けをつくる

失敗から”次の行動”が導けるような仮説思考に基づく

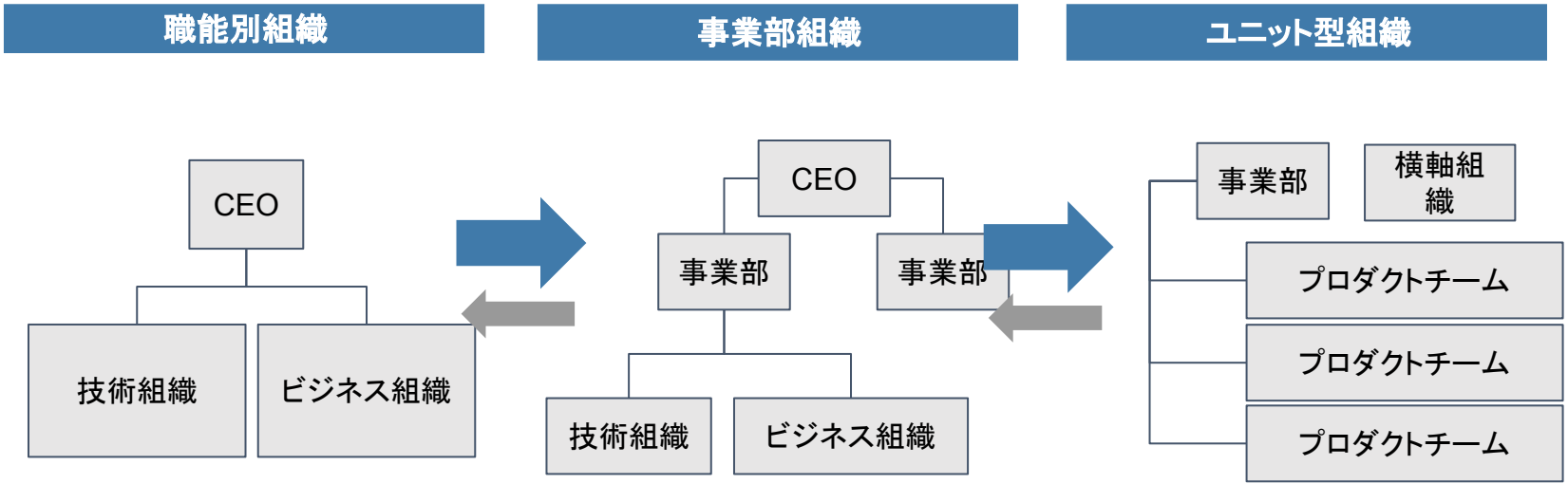
それらを組織的にスケールできるようにする

# フェーズ毎の「フェイルファストの原理」

| 経営と組織文化 | プロダクトマネジメント | 開発中 | リリース後 |
| --- | --- | --- | --- |
| 心理的安全性の投資 | 仮説法的なNSM | 静的型/型推論の発展 | コンテナ化 / k8s化 |
| リアルオプション戦略 | リードタイム最適化 | CodeClimate/ Linter | 継続的デプロイメント |
| 透明性あるOKR | システム思考 | Churn analysis/静的解析 | Blue Green  Deployment |
| 仕事の明確化 | データ駆動な仮説構築 | Security Shift Left | Feature Toggle |
| コアの内製化 | ユビキタス言語の管理 | DevOps / DevSecOps | Dark Launch |
| 失敗の予算化 | 仕様の直交性 | Microservices | サーキットブレイカー |
| 発信のオープン化 | 形式手法等の仕様検査技術 | ユニットテストとCI | フォルトインジェクション |
| フェイルファスト | 優先順位算出の明文化 | API駆動開発 | カオスエンジニアリング |
| マインドの育成 | | Trunk Based Development | Infra as Code |

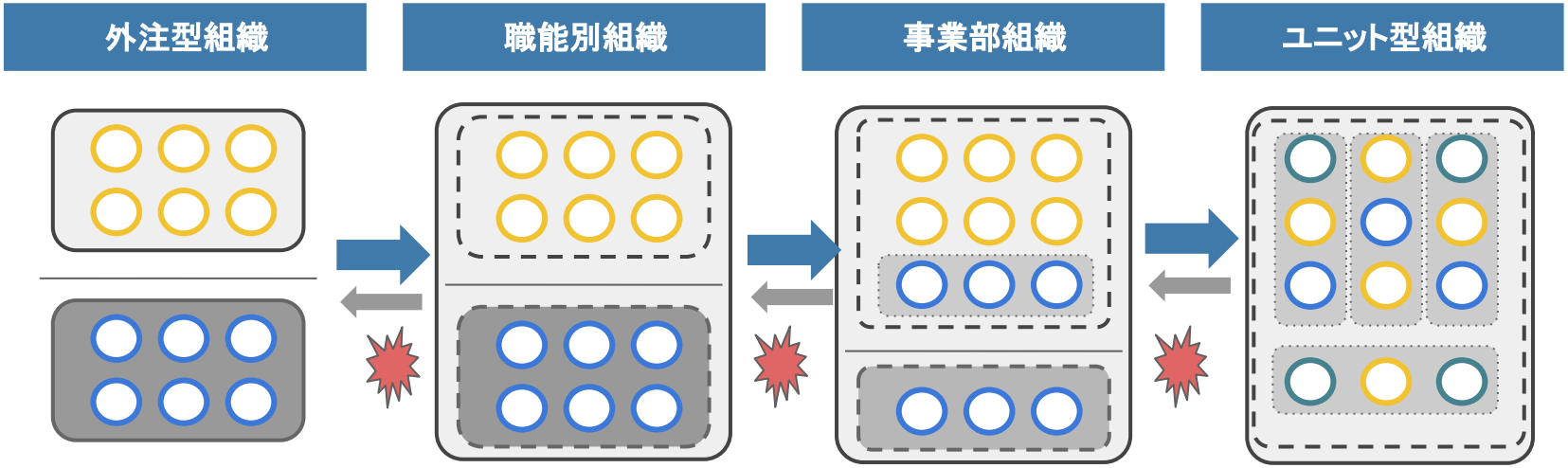
# エンジニアリングが導く未来の組織

# エンジニアリング組織の変化



# 組織の「エマルション」が進む

ソフトウェアを事業に生かしていくことが当たり前になるにつれて、組織の中で溶け合う



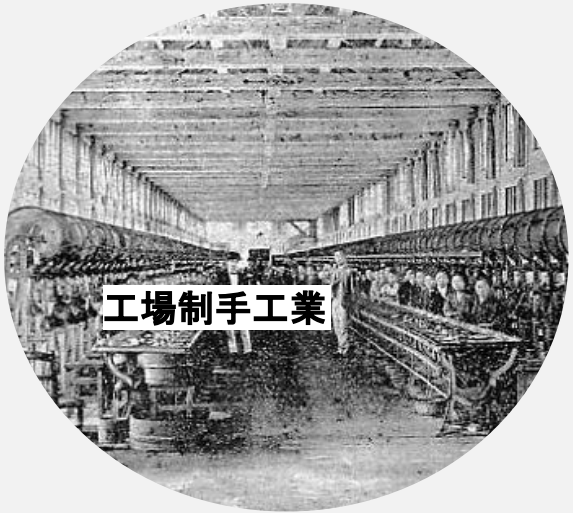
# 文字とコード。民主化するプログラミング

かつて、文字が書ける書記官は高給取りで人材不足と言われていた。


異なる要素の**境界面**に争いは生じる。
**溶け合った後**には、争いは生じない。

# 一昔前の働き方って「非人間的」？

「仕事が奪われる」そのとき、どんな仕事が奪われているのでしょうか。

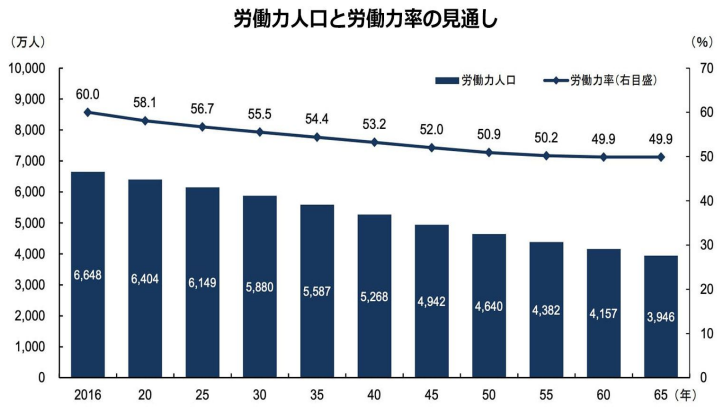



未来からみると
私達の働き方は「非人間的」。

# 働き方の変化はなぜ起きているか。

少ない労働力でより多くのことをコンピュータに任せて仕事をしていく必要がでてきている。

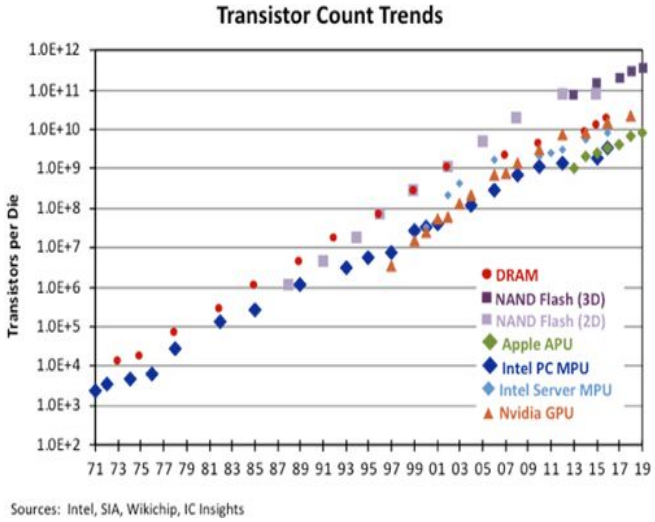
## 労働力が希少に




（注）2016年は実績。2020年以降は、男女別、年齢5歳階級別の労働力率を2016年と同じとして算出（75歳以上は、2016年の75歳以上の労働力率を75～79歳の労働力率とし、80歳以上はゼロとして算出）。

（資料）総務省「労働力調査年報」（2016年）、国立社会保障・人口問題研究所「日本の将来推計人口」（2017年4月推計）より、みずほ総合研究所作成

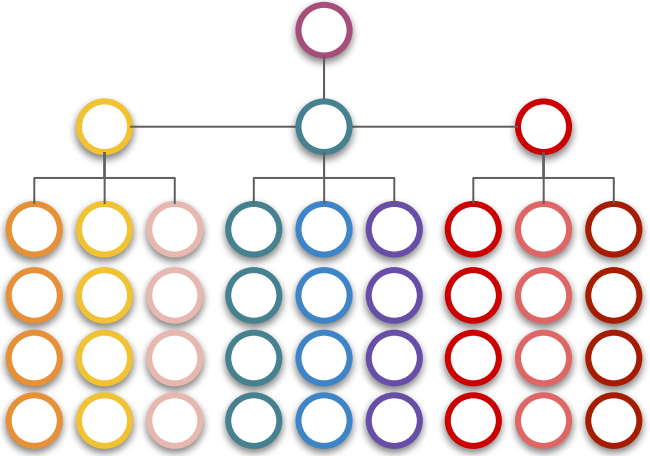
## コンピュータが安く大量に



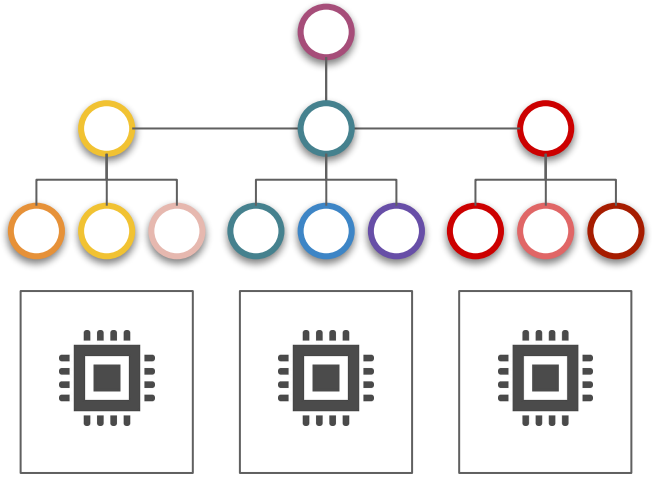
コンピュータに
働かせる領域を増やし
ヒトが働く領域の価値を上げる。

# 人からコンピュータに置き換わっている

旧来型組織

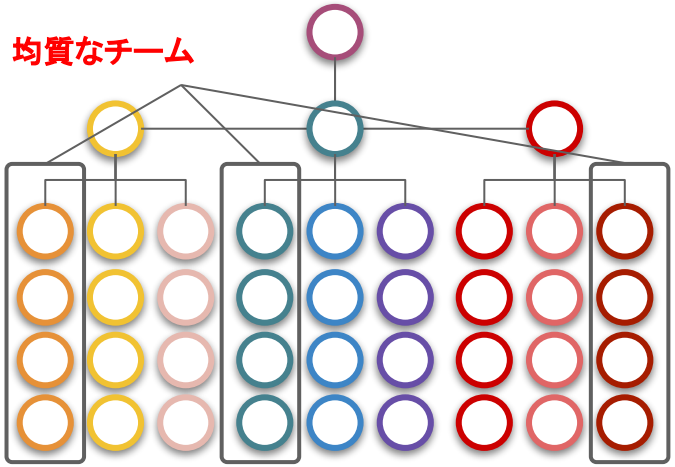


現代的ソフトウェア企業

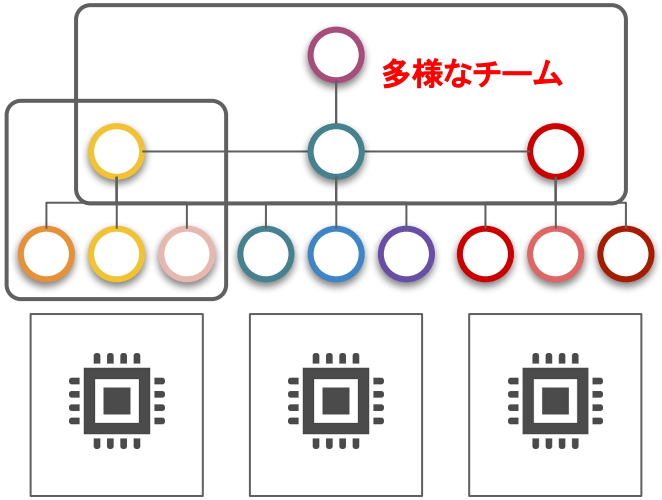


# 人からコンピュータに置き換わっている

## 旧来型組織

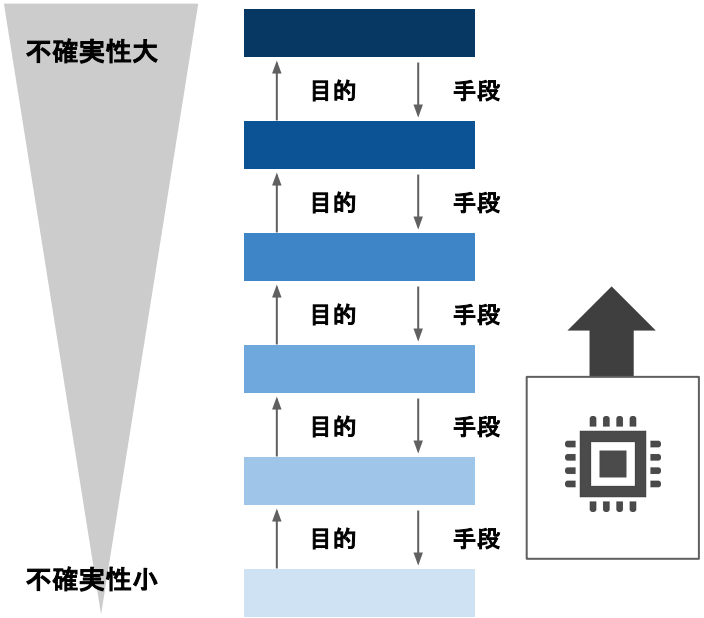



## 現代的ソフトウェア企業

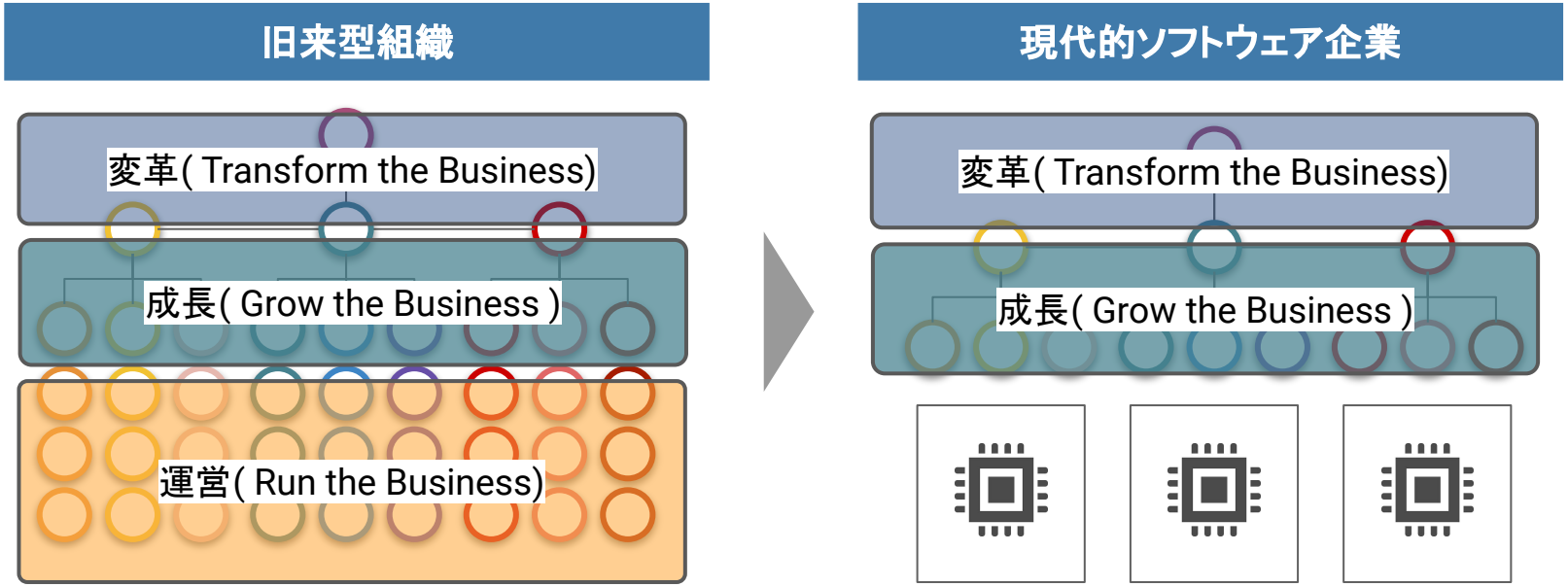


# 目的と手段のはしごが置き換わっていく

不確実性が小さい”手段”の領域からコンピューターが置き換えていく



# 人からコンピュータに置き換わっている



# 人からコンピュータに置き換わっている

| 旧来型組織 | 現代的ソフトウェア企業 |
| --- | --- |
| 官僚型職能別組織 | 全体論的多様性のあるチーム |
| コマンド＆コントロールの目標管理 | 権限委譲と透明性のある OKR型目標 |
| メンバーシップ型人事制度 | ジョブディスクリプション型人事制度 |

# 不確実性に適応する力が求められる。

オーディナリー・ケイパビリティとダイナミック・ケイパビリティの相違点[注16]

| | オーディナリー・ケイパビリティ | ダイナミック・ケイパビリティ |
|---|---|---|
| 目　　的 | 技能的効率性 | 顧客ニーズとの一致<br>技術的機会やビジネス機会との一致 |
| 獲得方法 | 買う、あるいは構築（学習）する | 構築（学習）する |
| 構成要素 | オペレーション、管理、ガバナンス | 感知、捕捉、変容 |
| ルーティン | ベスト・プラクティス | 企業固有の文化・遺産 |
| 経営上の重点 | コストコントロール | 企業家的な資産の再構成とリーダーシップ |
| 優先事項 | 「ものごとを正しく行う」 | 「正しいことを行う」 |
| 模倣可能性 | 比較的模倣できる | 模倣できない |
| 結　　果 | 効率性 | イノベーション |

2020年版ものづくり白書

# 経営学とソフトウェア工学を
# わけて考える時代の終わり。

社会のソフトウェア化が進むと
人間には本質的なコミュニケーション能力
が求められてくる。