JaSST'25 Kansai

変化する開発、進化する体系
時代に適応するソフトウェアエンジニアの
知識と考え方

～変わり続ける現場で生き抜くために～

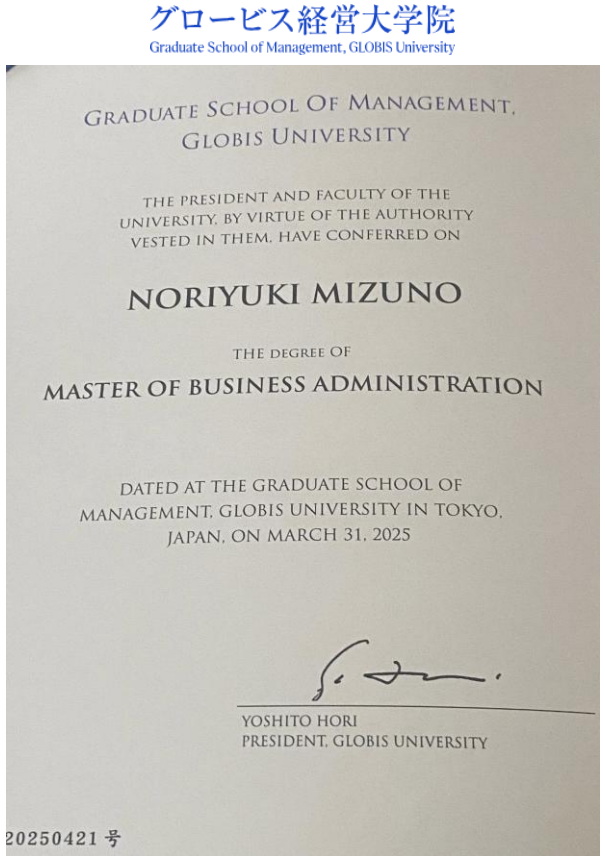
Software/System Engineering Practitioner
@NoriyukiMizuno
みずのり（みずののりゆき）
TOC/TOCfE北海道
RDRA MeetUp、TEF道など

JaSST関西のテーマは「QA Expo 2025」

そして世はまさに、生成AI時代！時代が変わるかも？
完璧に理解して活用しているでしょうか？

自己紹介

×



ソフトウェアエンジニアリング　　経営学修士（MBA）

自己紹介：ソフトウェアエンジニアリング

水野昇幸（みずののりゆき）

・Twitter（X）：@NoriyukiMizuno

ソフトウェアエンジニアリング関連

・（監訳/共訳）実践ソフトウェアエンジニアリング（第9版）、オーム社、2021
・8th長崎QDG（2025）：ビジネスと現場活動をつなぐソフトウェアエンジニアリング
・WACATE2024 Summer招待講演：モデリングのそだてかた
・ソフトウェア品質シンポジウム2022：演習で学ぶ実践ソフトウェアエンジニアリング
・JaSST22東京：（パネル）ソフトウェア開発にかかわる全てのエンジニアの一般教養
「ソフトウェアエンジニアリング体系」における品質技術
・ET＆IoT2021：米国修士課程ベストセラーに学ぶ
体系的ソフトウェアエンジニアリングの必要性

保有資格とか

・経営学修士（MBA）@グロービス経営大学院
・簿記3級、2級、JTSQB-FL、ALTM、ALTA、
・情報処理エンベデッド、プロマネ
・TOC-CCPMスペシャリスト（インプリメンター資格）
国際NPO TOC for Education, Inc 認定「ファシリテータトレーニング」
「思考及びコミュニケーションツールトレーニング」修了

コミュニティ系

・TOC/TOCfE北海道、TEF道、**JaSST北海道実行委員**
・ETロボコン実行委員、RDRA MeetUpなど

TOC/TOCfE Hokkaido

自己紹介：思考技術、TOC/TOCfEなど

TOC（制約理論：Theory of Constraints）関連発表

・TOCセミナー長崎（2024）：仕事でもコミュニティでも役立つTOC　約15年の活動から
・第11回教育のためのTOCシンポジウム（2022）：
500ページ超（原書650ページ超）の訳書の出版へ向けてATTで考えて活動したお話し
～TOCfEとソフトウェア開発技術の組合せ活用
・JaSST'18東京：SaPIDTOC～未来予想型チーム運営ワークショップ（チュートリアル）
・SQiP2014：CCPMの考え方を活用したテストフェーズにおける課題解決
～課題発見、解決までのケーススタディ～
・SQiP2013：システムテスト実施フェーズにおけるボトルネック/クリティカルチェーンを特定した残日数管理マネジメントとその効果

TOC関連の講演資料・ワークショップ

・プロジェクトマネジメントの概要とCCPMによる工期短縮の仕組み
・提案を検証する技術（TOC マフィアオファー（URO）活用技術）
・SaPIDTOC～未来予想型チーム運営ワークショップ
・CCPM折り紙ワークショップ　※2014年から12回くらい実施、300名以上が受講
・CCPMカレーワークショップ
・SaPIDTOC～未来予想型チーム運営ワークショップ

TOCおよびTOCfEイベント企画・運営

・TOC/TOCfE北海道立上げ（2016）
・TOCfE国際認定プログラム北海道開催（2018）

TOC/TOCfE Hokkaido

自己紹介：テスト/要件定義関連

ソフトウェアテスト/テスト自動化関連（海外）
- （共著）InSTA2019@西安：Coexistence of test execution efficiency and test
- （共著）InSTA2018@Sweden：Proposal for Enhancing UTP2 with Test Aspects
- （共著）InSTA2017@東京：Test Conglomeration - Proposal for Test Design Notation like Class Diagram
- （共著）6WCSQ@ロンドン（2014）：Stepwise Test Design Method

ソフトウェアテスト/テスト自動化関連（国内）
- JaSST'21東京：　（チュートリアル）価値につながる要件・仕様からテストを考える
- ET-WEST2018：なぜ組込みシステムにおけるテストは難しいのか
  ～ 2つのテストアーキテクチャによってテストを組立てる ～
- JaSST'17関西：納得できるテストをつくるアプローチ
- STAC2015：自動家は見た～自動化の現場の真実～
- テスト設計コンテスト優勝：2017年　STUDIO IBURI
- テストカタマリーの紹介/活用したテスト設計プロセス案
- **WARAI（関西ソフトウェアテスト勉強会）など立上げ**

要件定義関連
- JaSST'21東京：　（チュートリアル）価値につながる要件・仕様からテストを考える
- DevSumi関西2020：伝統的食品工場エンジニアリング会社が挑むDXへの
  ビジネスアイデアをRDRAによる要件定義でプロダクト開発へつなぐ～Side：要件定義
- RDRA+プロトタイピングおよび仕様化時に役立つ技術、事例紹介

自己紹介：おしごと

■過去
・某電機メーカーで宇宙・防衛系の
システム設計・プロマネなど
開発～保守まで全工程に対応
・大規模プロジェクトでの開発

■現在（2016～）
・流しのソフトウェアエンジニア（フリーランス）
ソフトウェア開発のなんでも支援
・支援企業は車・食品・旅行業など
・現状は企業内スタートアップ的な
プロダクト開発を支援中

※PjM/PdM、要件定義、実装/インフラ、テスト、保守

ちょっとした概要

今回のおはなし
・生成AIにかかわらず、開発状況は常に変わり続けている状況
・なんか変わったね、という話だけだとつまらない（つまらんかった）
　どう変わった・変わるのか、どう変化に適応するのか？を紹介

・新たに変化が発生しそうな時代に、どのように動くべきかのヒントを提示
　※個人的には楽しい時代になるなーと感じる状況

おことわり
・経歴上、QAってのはよくわかりません
・比較的開発の「現場」に近くにいる個人の体験と分析を中心に語る内容です
・細かい部分は資料見てもらおうかと→説明は端折りますので公開資料見てください
・かなりメタな内容にしてます
　ストレートに役立つ内容ではない可能性があるので注意ください

変化する開発
～過去と現在～

まずは過去と現在の開発（自分目線）を紹介します。
講演内LT×2で紹介。　※昔話トークはつまらない方もおりますので時間厳守予定

（JaSST Osakaが第1回開催した頃）
2005年～2010前後のおはなし

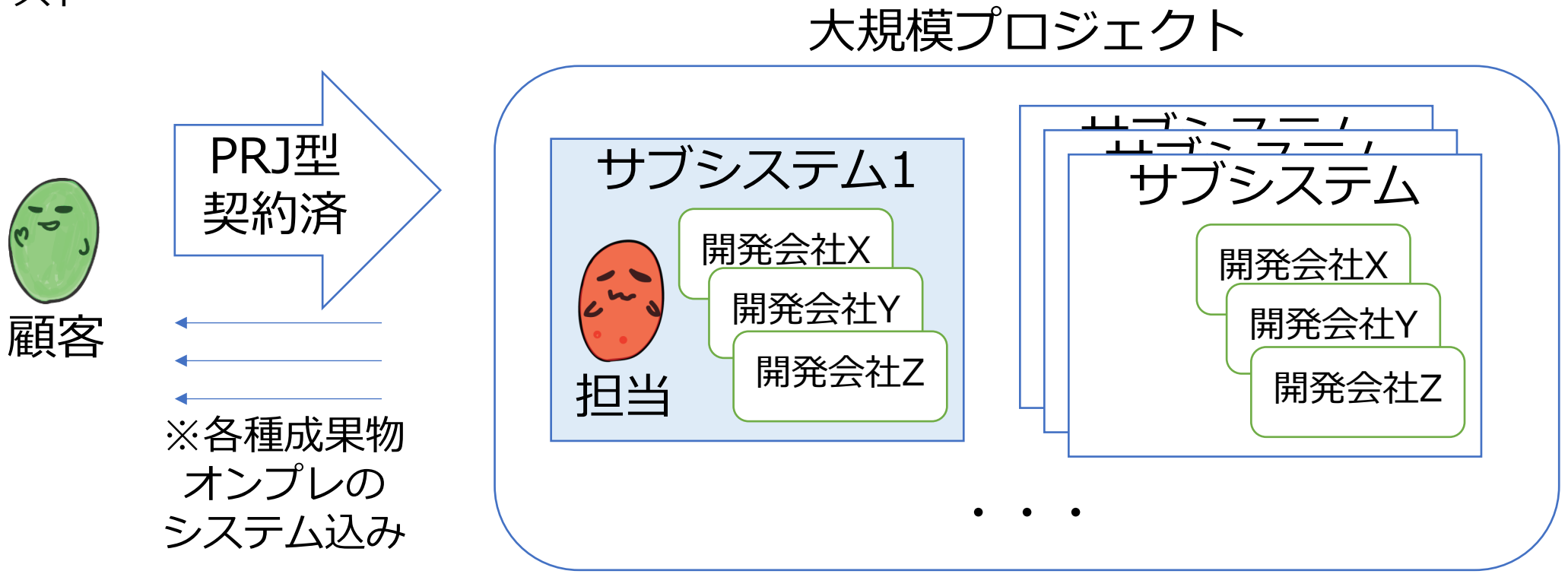
2020-2025年付近のおはなし

※こちらの参考
https://speakerdeck.com/mizunori/software-engineering-that-connects-business-and-operations-qdg2025

講演内LT1

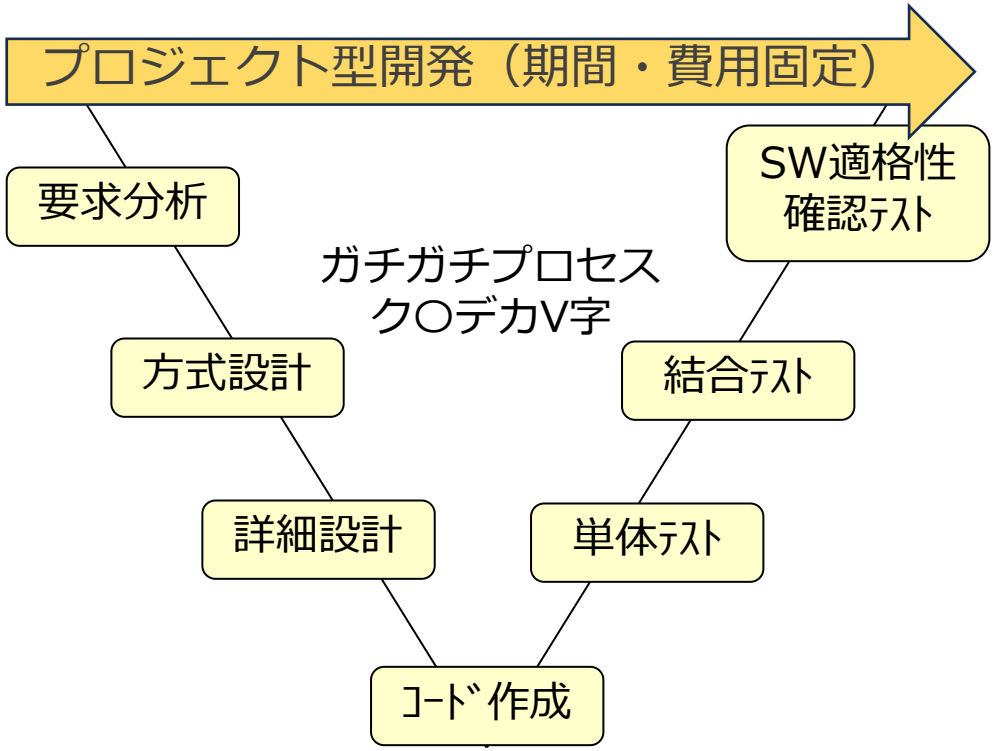

JaSST'25 Kansai：招待講演内LT1
（いにしえの）大規模SoSプロジェクトの
開発と運用の現場から
TOC/TOCfE Hokkaido
TEF道
RDRA MeetUp
Software/System Engineering Practitioner
@NoriyukiMizuno
みずのり（みずののりゆき）
TOC/TOCfE北海道
RDRA MeetUp、TEF道など
1

コンテキスト



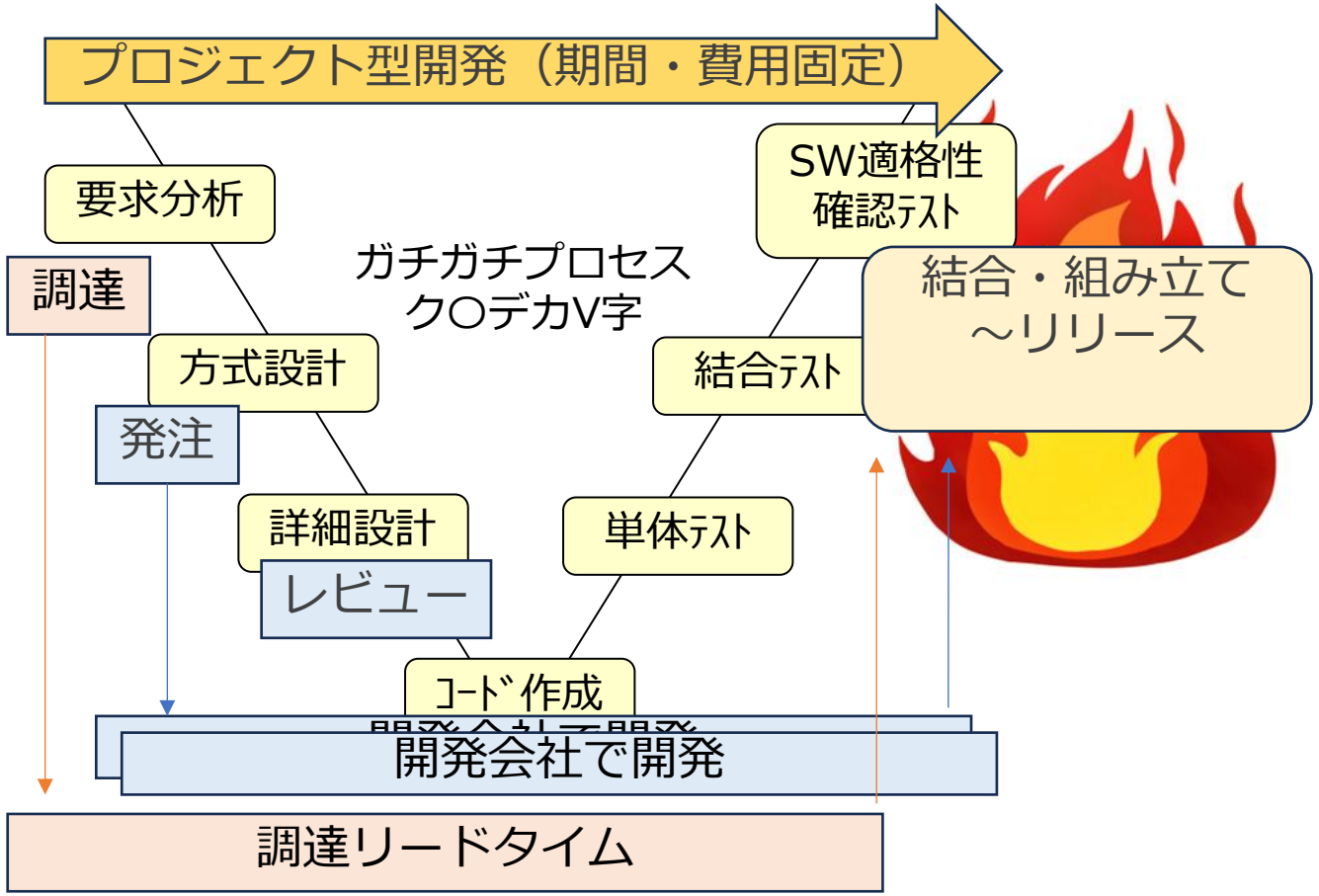
- **キーワード：System of Systems、通信、監視制御**
- **プロジェクト型**
  - ハードウェア調達も含めていくつもの開発会社へ依頼、サブシステムで億越えルーキーだらけ
- **やっていたこと**
  -PjM、システム/ソフトウェア設計、調達、テスト、運用保守（客先対応込み）などなど

いにしえのプロジェクトではスゴイ計画を定め、目標値に従い進めたとのことじゃ…



いにしえのプロセスありき

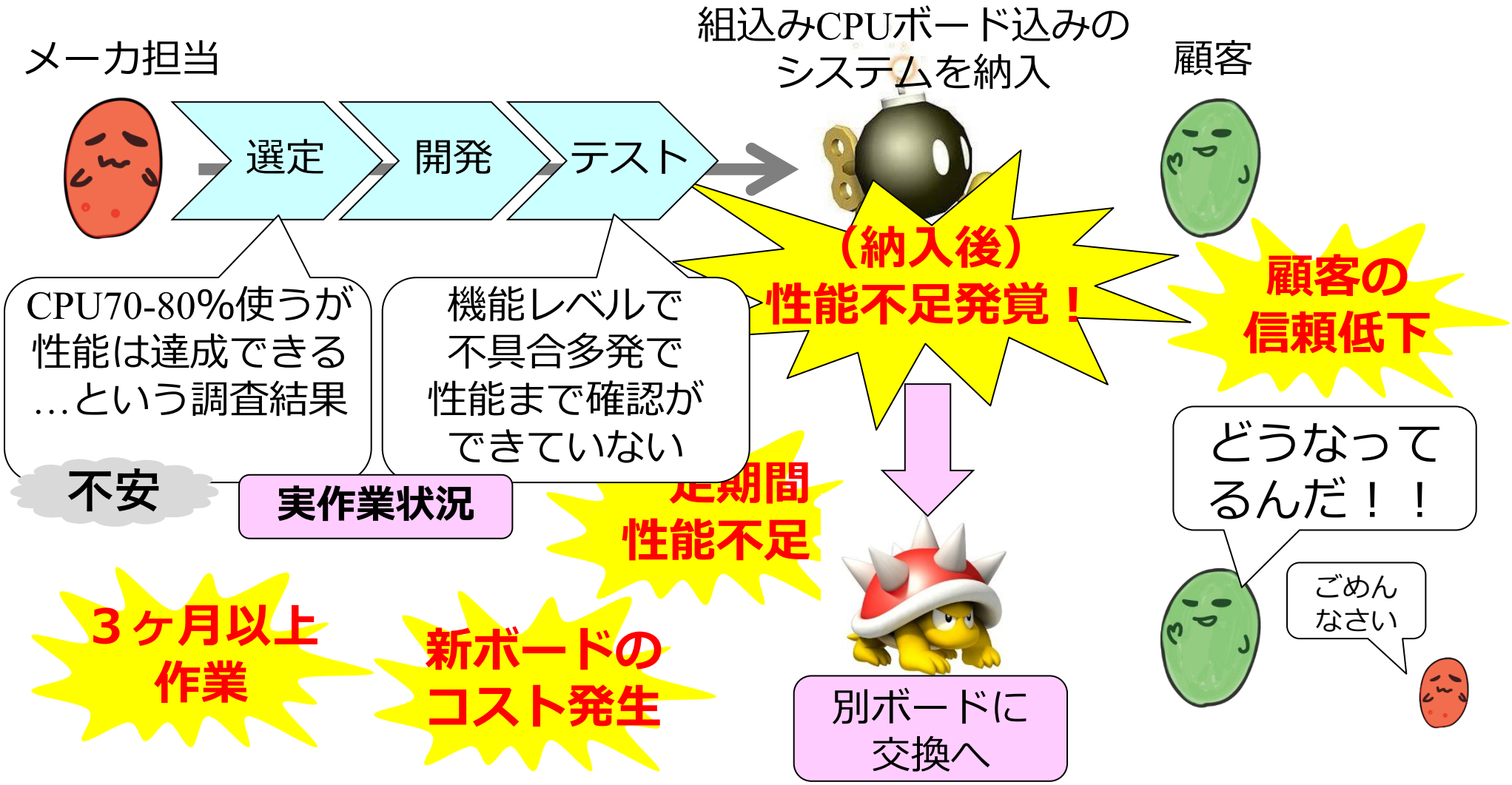
実際の現場は大変じゃった



世の中のフレームワーク・ライブラリ


炎上

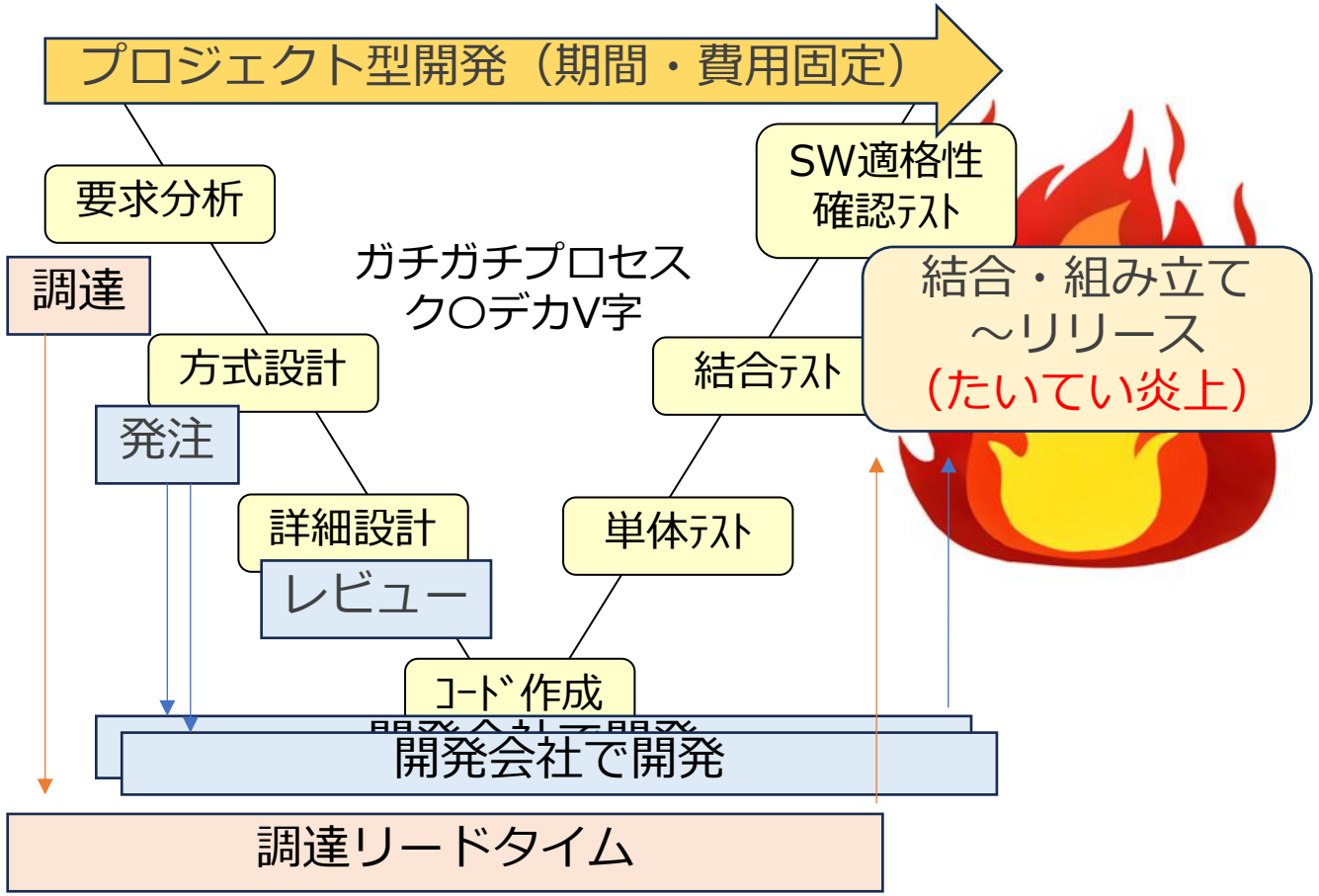
小説内の状況



JaSST'17 Kansai : https://speakerdeck.com/mizunori/approch-for-designing-convincing-test-case

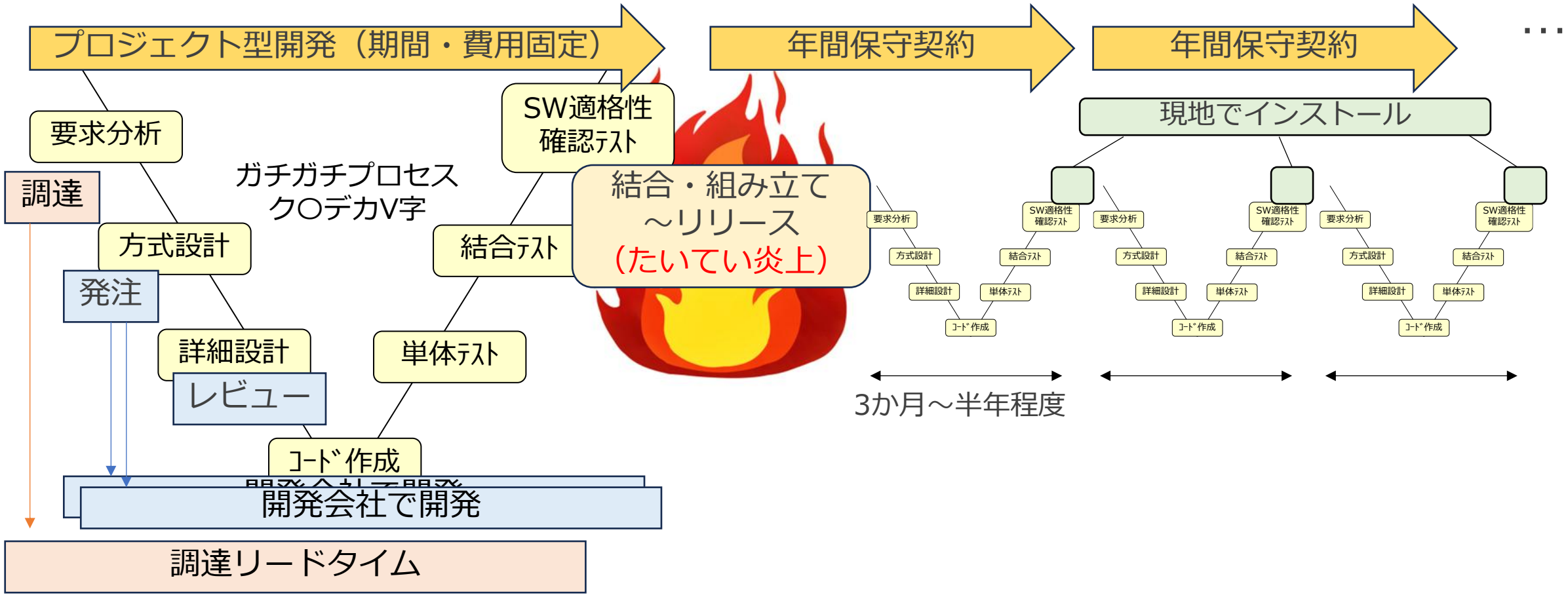
いにしえのプロセスありき

炎上後のおはなし



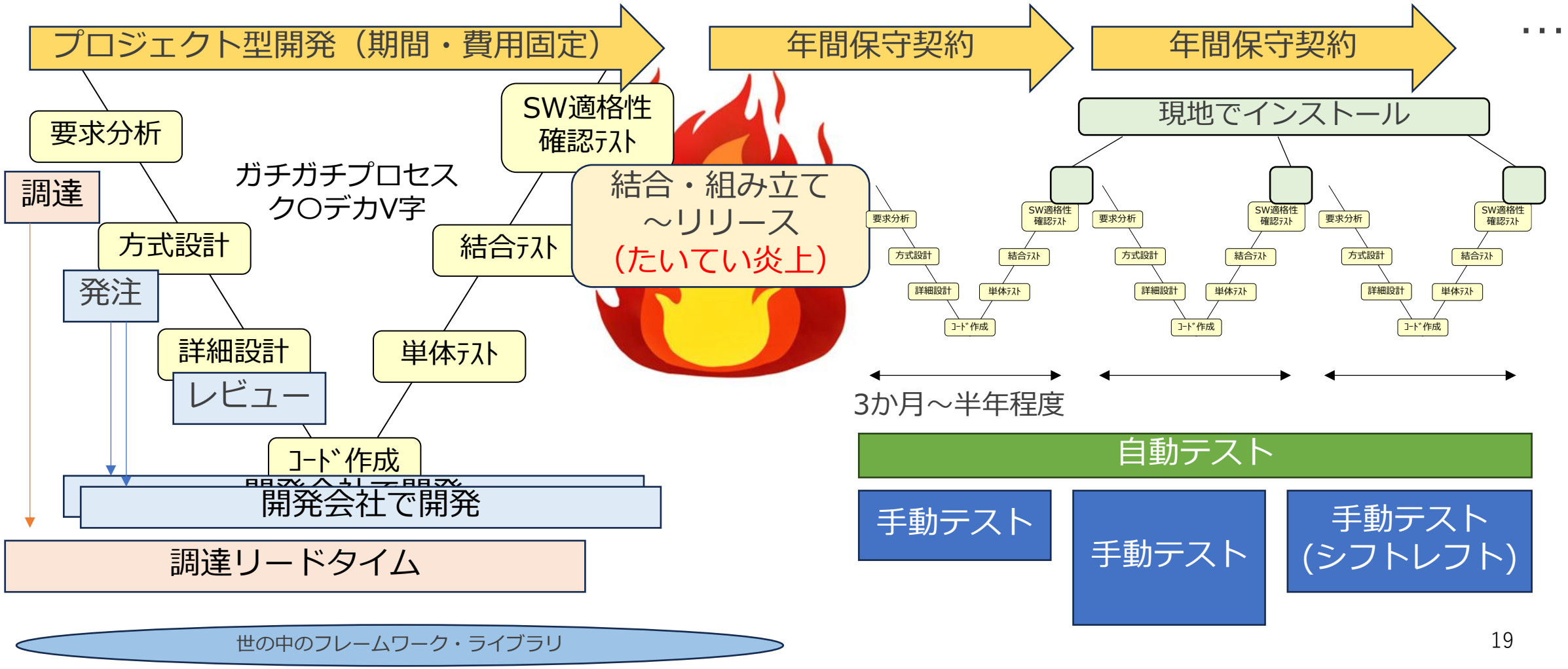
世の中のフレームワーク・ライブラリ

まっとうに運用されるシステムでは継続的＆インクリメンタルなリリースが続く
…とはいえ、PRJ完了後の（派生開発の）プロセスは定義なし→自由を得る



世の中のフレームワーク・ライブラリ

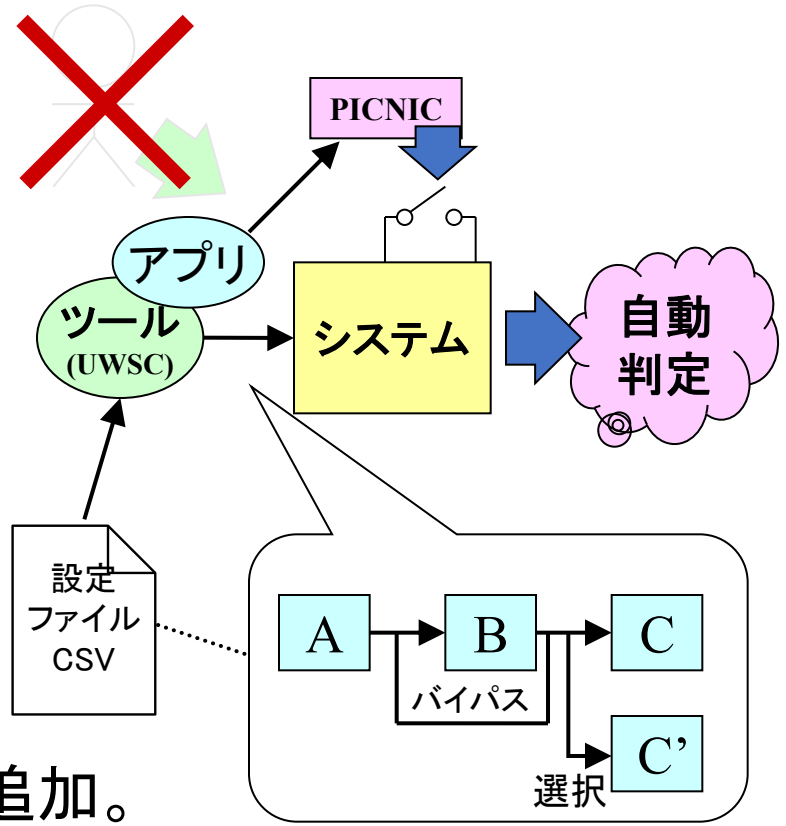
担当者が得られた知見に応じてテストも変化していく
テスト自動化、テスト設計の取り込みで変化



・多数のコンフィグ×時間のかかる操作を自動テスト化（2007～）

Windows自動化ソフト「UWSC」を軸に自動化環境を構築。

■フェーズ①
人が入力していたトリガを決め打ち。
同じ処理を繰返し再生し続ける。
■フェーズ②
CSVから設定パターンを変更可能に。
■フェーズ③
個別の制御の選択が出来るように。
■フェーズ④
周りの環境も出来る限り制御可能に。
■フェーズ⑤
結果判定がメンドイので、自動判定機能追加。



UWSC

日々の改善（テスト関係）　※多数実施している中から抜粋

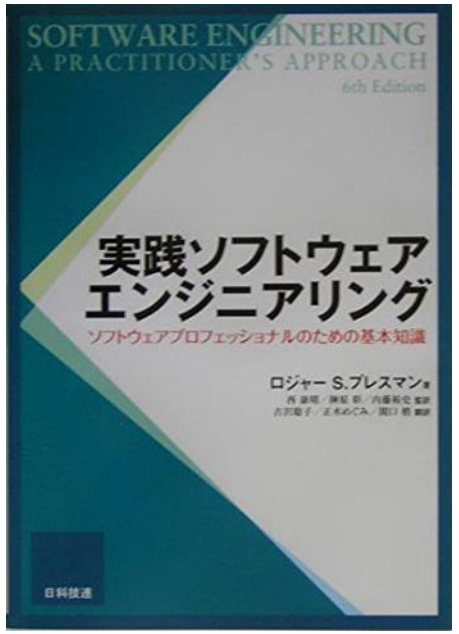
多数のツール、書籍、コミュニティとの関わりなどからも改善
・多数のコンフィグ×時間のかかる操作を自動テスト化（2007～）
・テスト設計技法、テストの知見を取り込んで定期リリースを大幅改善（2009～）

UWSC

TABOK kansai

warai 関西ソフトウェアテスト勉強会

引用：ソフトウェアテスト技法ドリル
秋山浩一、日科技連出版、2010年



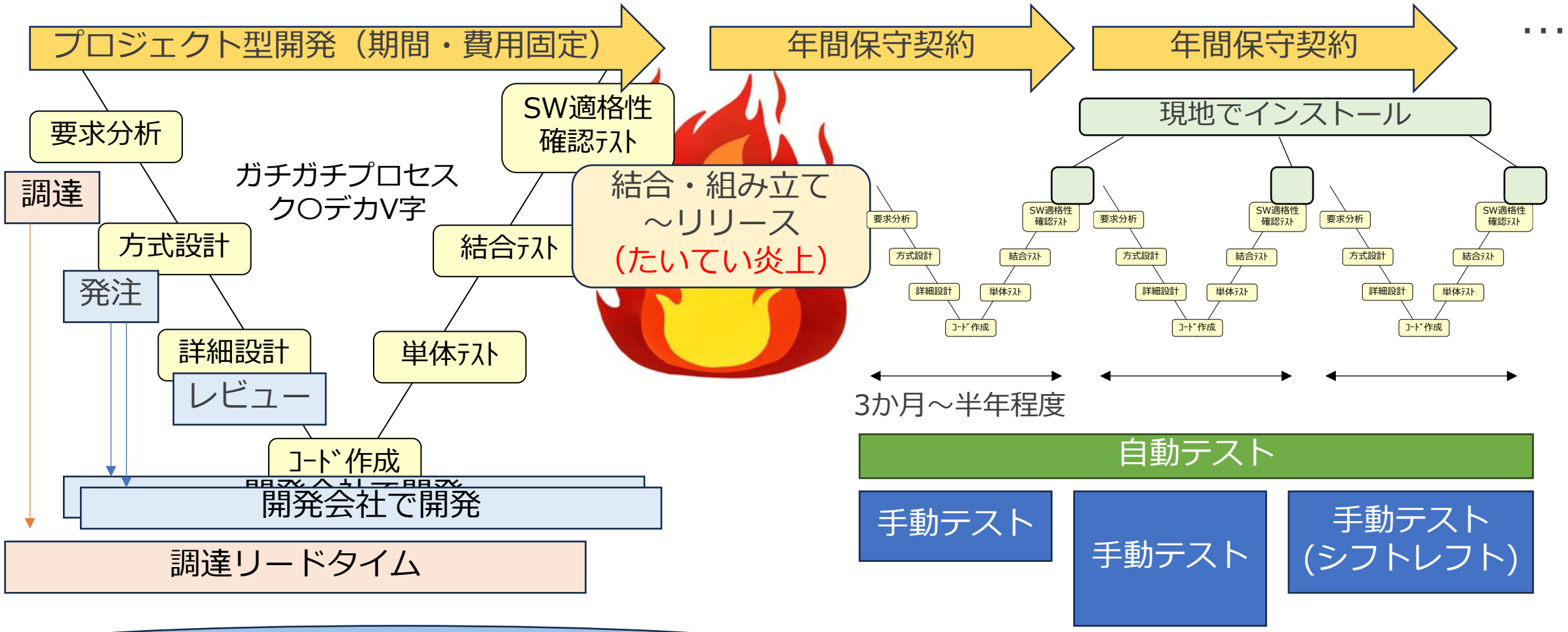
引用：実践ソフトウェアエンジニアリング（第6版）
Roger.S.Pressman、西 康晴 (翻訳)、日科技連出版、2005年

WACATE
Workshop for Accelerating CApable Testing Engineers

SQiP
Software Quality Profession
ソフトウェア品質シンポジウム

ソフトウェアテストシンポジウム
JaSST ージャスト
Japan Symposium on Software Testing

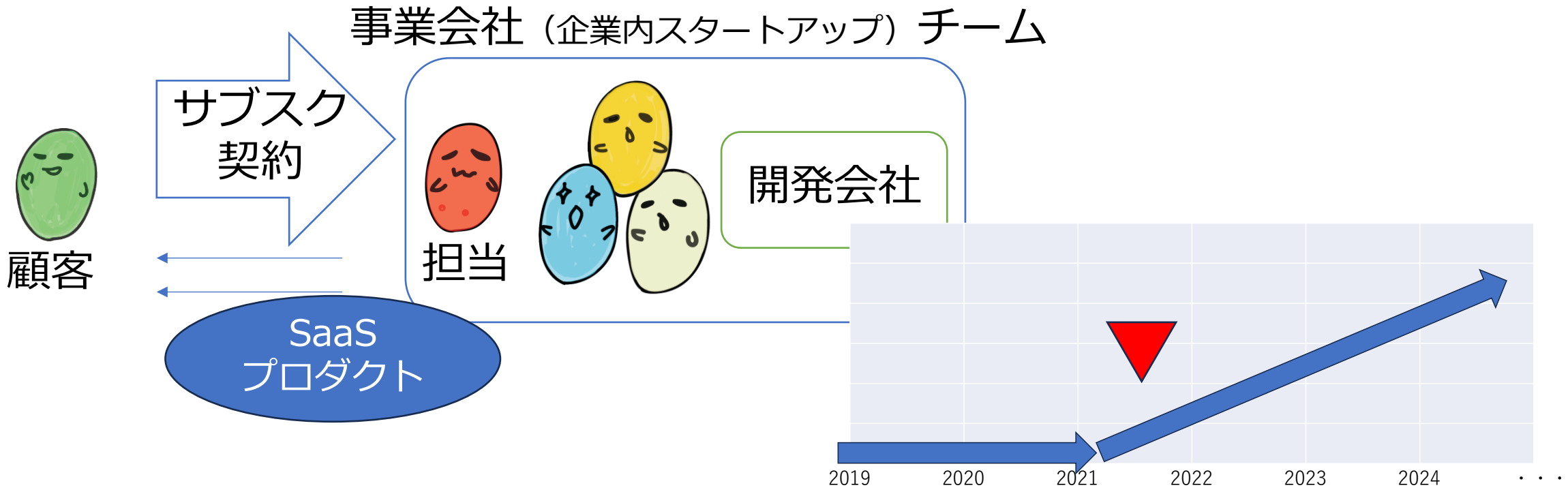
最終的には複数のサブシステムで最も安定したシステムとなったとのことじゃ…



講演内LT2


JaSST'25 Kansai：招待講演内LT2
新規事業SaaSプロダクトの
開発と運用の現場から
TOC/TOCfE Hokkaido
TEF道
RDRA MeetUp
Software/System Engineering Practitioner
@NoriyukiMizuno
みずのり（みずののりゆき）
TOC/TOCfE北海道
RDRA MeetUp、TEF道など

コンテキスト



- **キーワード：工場、設備保全・メンテナンス、SaaSプロダクト**
- **プロダクト型**
  - 顧客がいない中で企画からスタート、顧客提供はサブスク契約で継続メンテナンス
- **やっていたこと**
  -プロダクト・プロジェクトマネジメント、ソフトウェア設計、実装、
  - テスト、運用保守（顧客対応込み）

いけいけプロダクト開発？

最初は顧客無し
まったくプロセス等は定めず、デモや展示会等の必要に応じてリリース
※テストも無し、次ページ参照

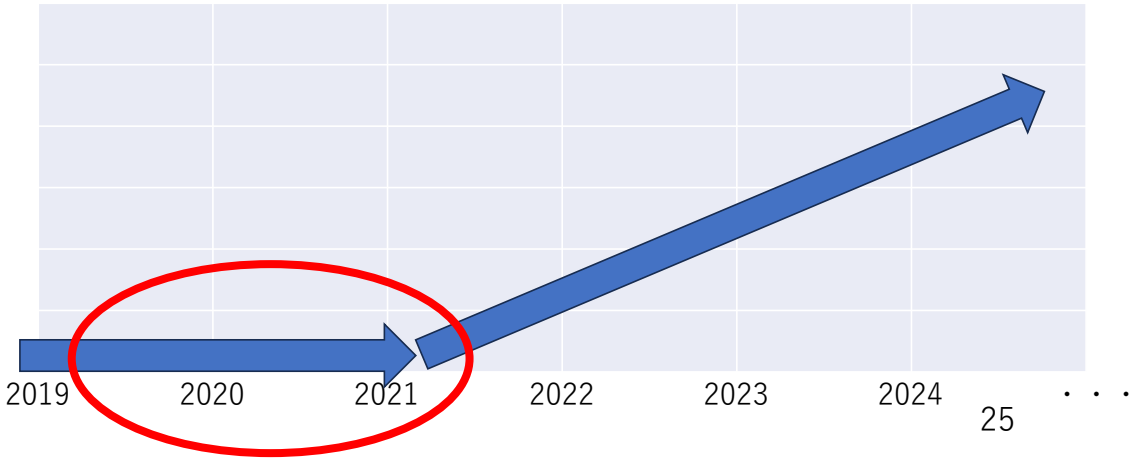
事業会社の自社開発

技術選定 | **適宜リリース** | **適宜リリース** | **適宜リリース**

クラウド/インフラ構築

アプリ開発とインフラ構築が並列で可能

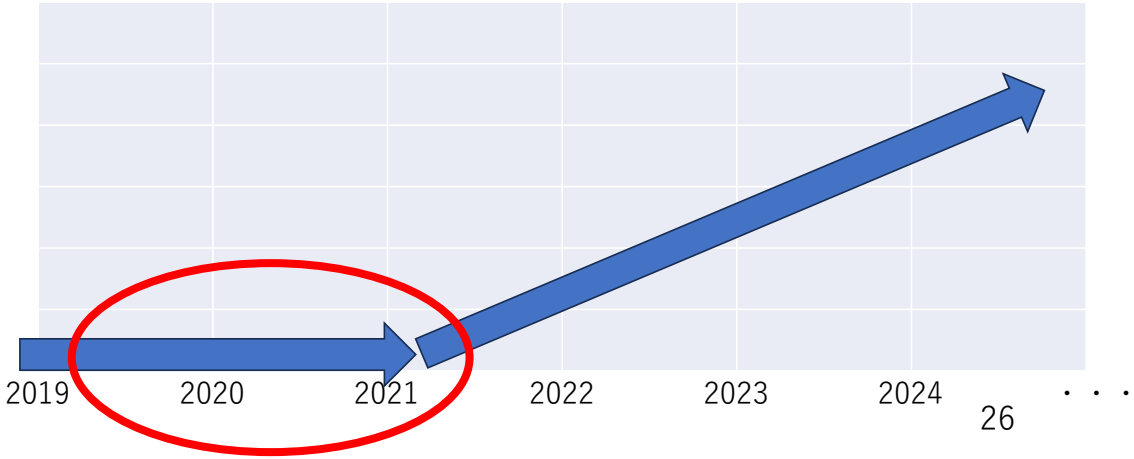
世の中のフレームワーク・ライブラリ



（参考）初期プロダクト開発での活動

（探索が必要な状況における）初期プロダクトでは、以下のような活動を繰り返す。

とにかくアイデアを試す	顧客に魅力的な要素を提供する必要あり 作るよりペーパープロトの方が速いことも多い 適切な顧客候補がいると効果的に進む ※CVP（顧客提供価値）が明確に把握できないと特に苦労する
高速にスクラッチ＆ビルド	試行錯誤の結果、高速に作り直すケースが多発 ※ある意味高速の犬小屋づくり
不具合は許容・テスト不要	よほどのことが無い限りテストはいらない 不具合はあっても使う人がいないので許容可能



いけいけプロダクト開発？

顧客獲得したけど？

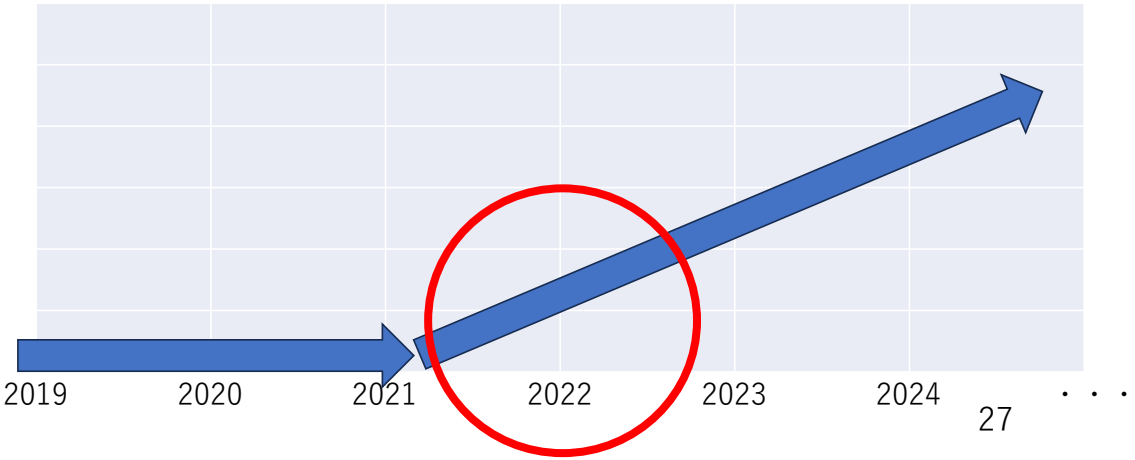
事業会社の自社開発

技術選定 | 適宜リリース | 適宜リリース | 適宜リリース

クラウド/インフラ構築

アプリ開発とインフラ構築が並列で可能

世の中のフレームワーク・ライブラリ




炎上

いけいけプロダクト開発、のつもりが？

ざっくり、「ゼロ」段階と「イチ」以降はまったく異なる開発が必要
継続的デグレ、インクリメンタルに開発が進まない（機能が消える？）など問題発生

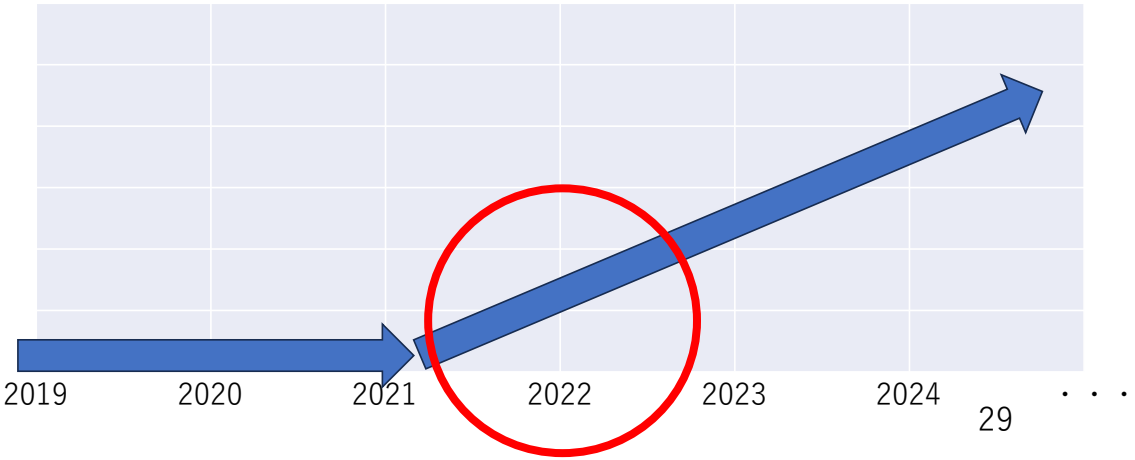
事業会社の自社開発

技術選定 | 適宜リリース | 適宜リリース | 適宜リリース

クラウド/インフラ構築

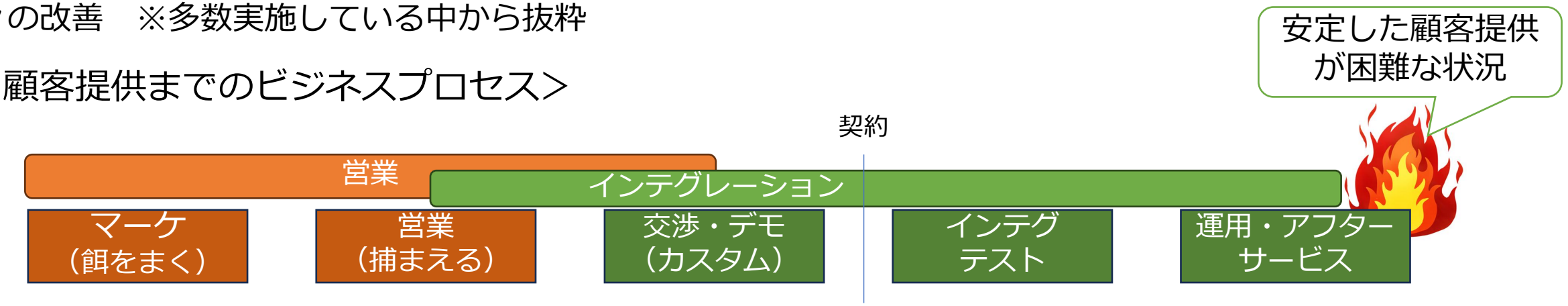
アプリ開発とインフラ構築が並列で可能

世の中のフレームワーク・ライブラリ

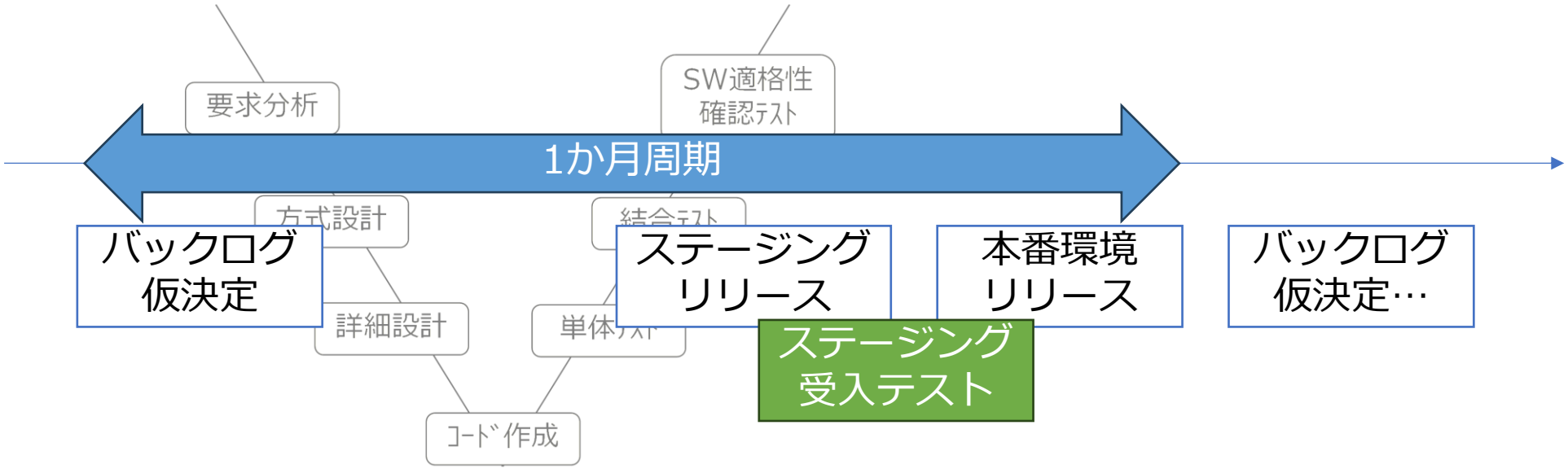


日々の改善　※多数実施している中から抜粋

<顧客提供までのビジネスプロセス>



・元々のカウボーイ的なリリース状況をプロセス整備、月1回リリースへ
・ステージング環境を整備してテストまでの流れを作る
　バックログに応じた内容に対し、テストを通過させてリリースへ



多少安定したが…

インクリメンタルな反復的な開発を行うことで多少安定する

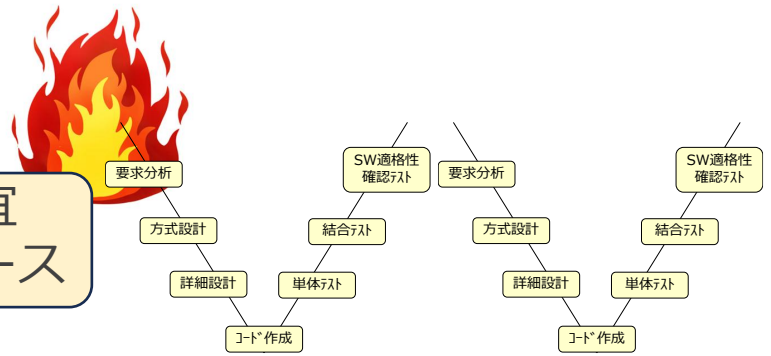
サブスク型契約 / 事業会社の自社開発

技術選定

適宜リリース

適宜リリース

適宜リリース



クラウド/インフラ構築

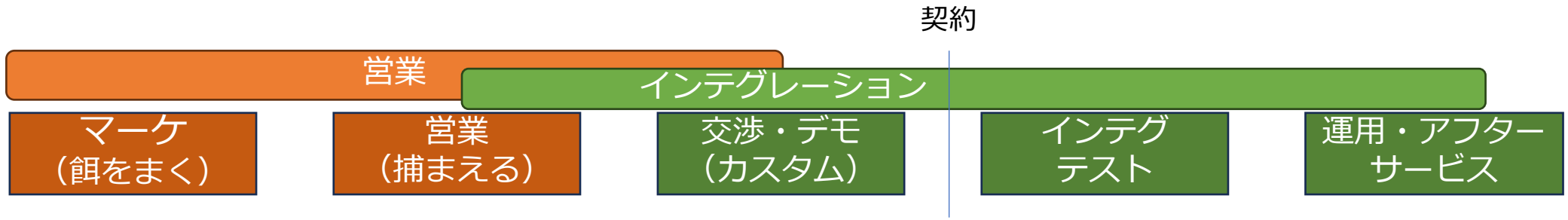
アプリ開発とインフラ構築が並列で可能

反復的、インクリメンタルな派生開発へ

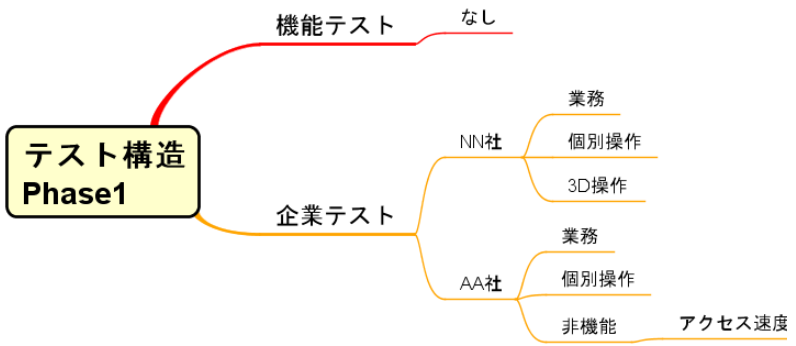
世の中のフレームワーク・ライブラリ

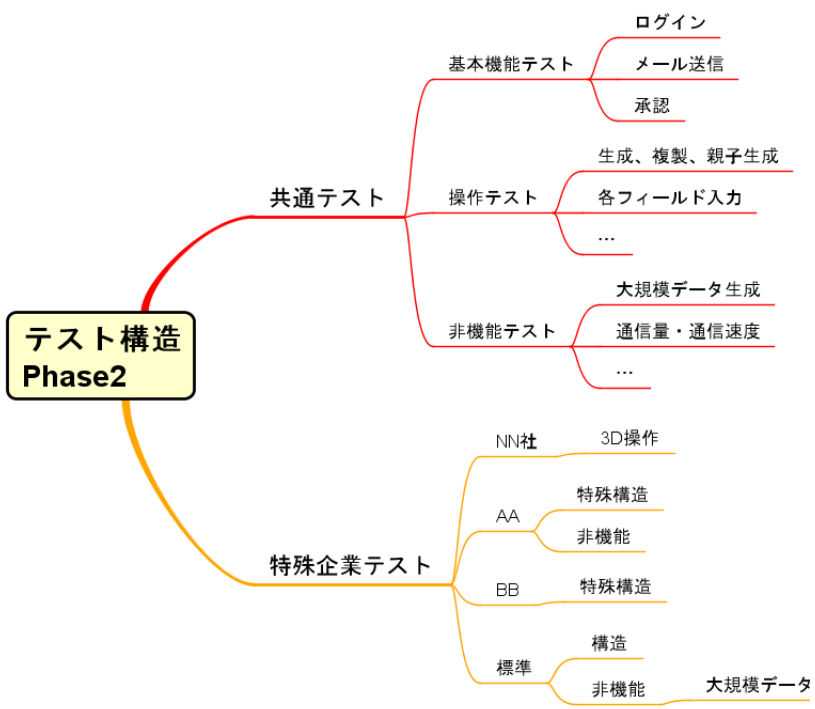
日々の改善　※多数実施している中から抜粋

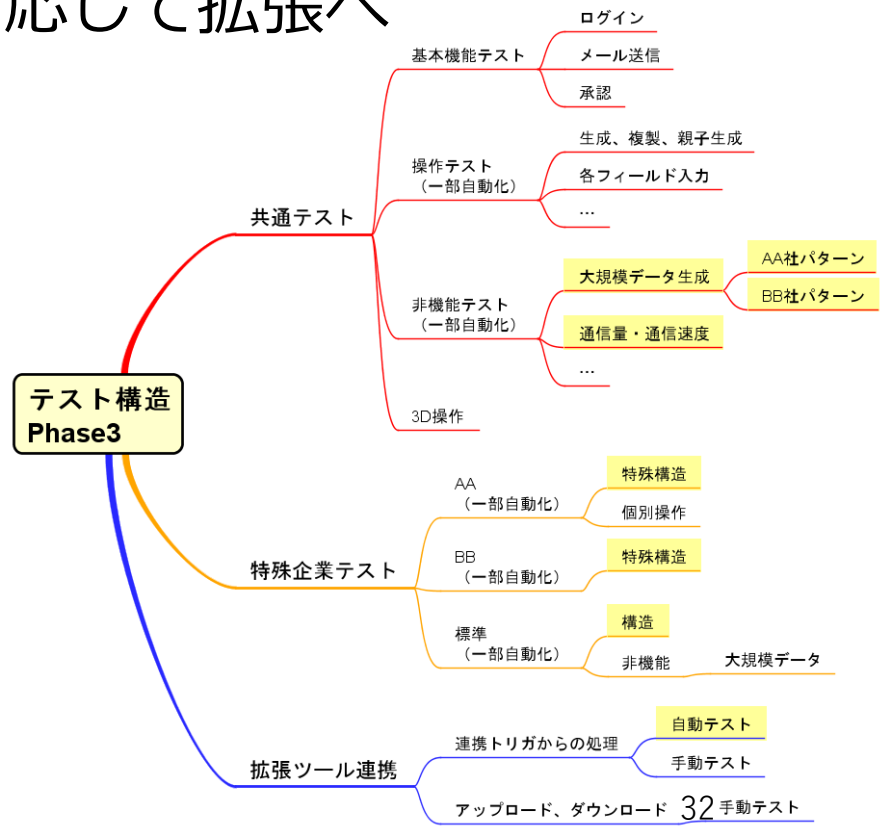
＜顧客提供までのビジネスプロセス＞



・リグレッションテストは最初は顧客対応のみ→規模に応じて拡張へ

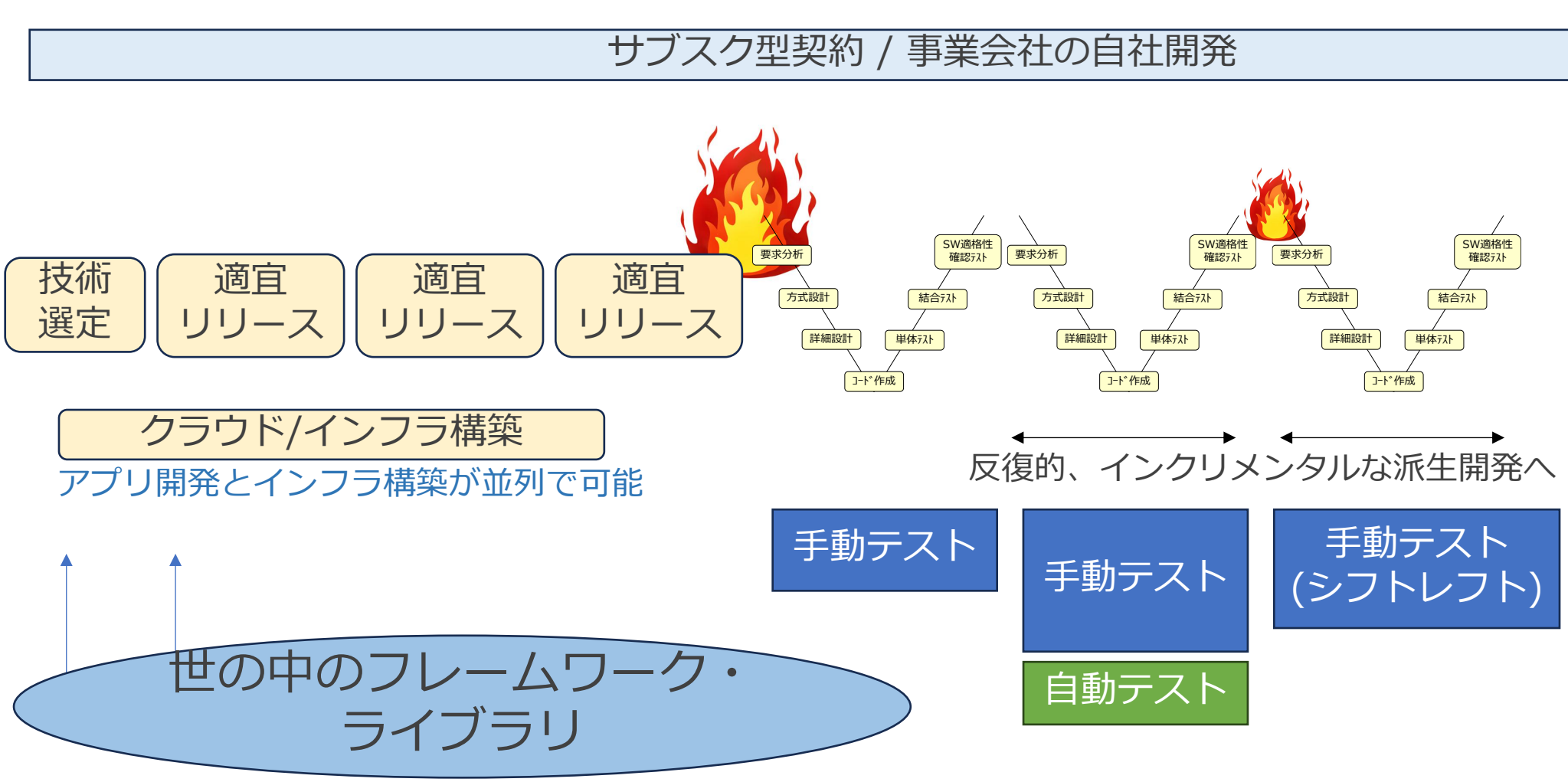






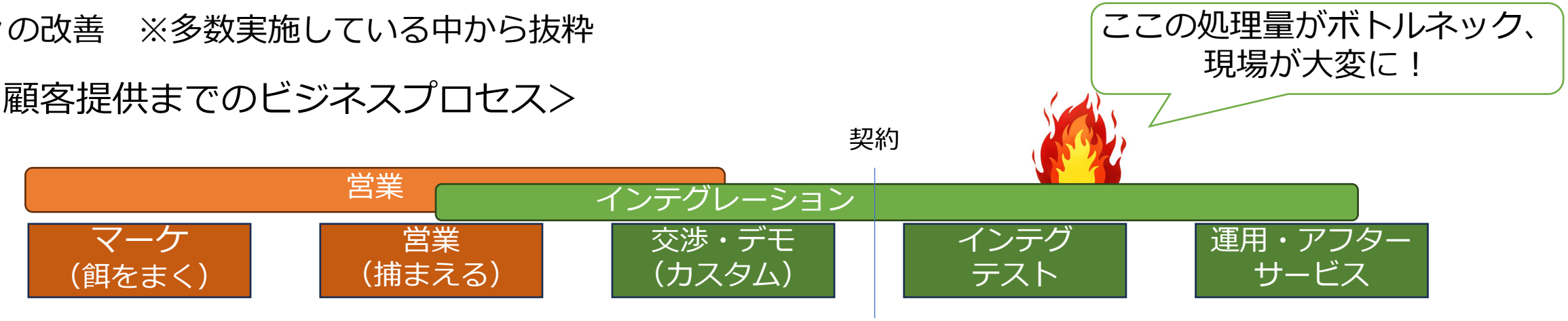
多少安定したが…

インクリメンタルな反復的な開発を行うことで多少安定するが、他にもボヤみたいな問題は発生する

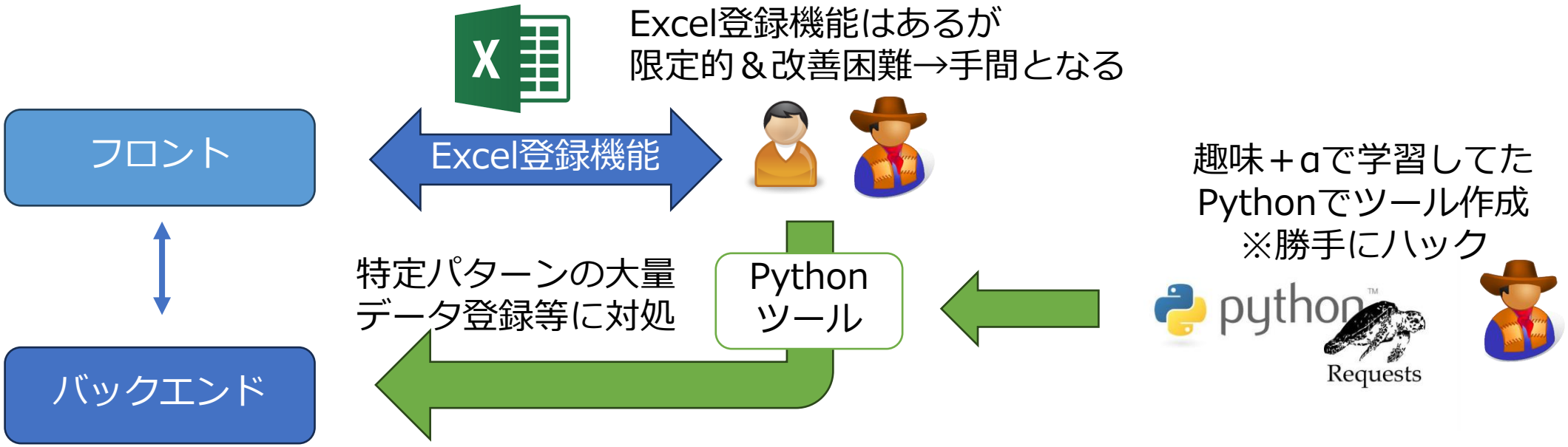


日々の改善　※多数実施している中から抜粋

<顧客提供までのビジネスプロセス>



・顧客単位のインテグ作業（事前データ登録）で特にチーム負担大
・対象のシステムはSPA、趣味でPython/requestsベースの支援ツール構築へ

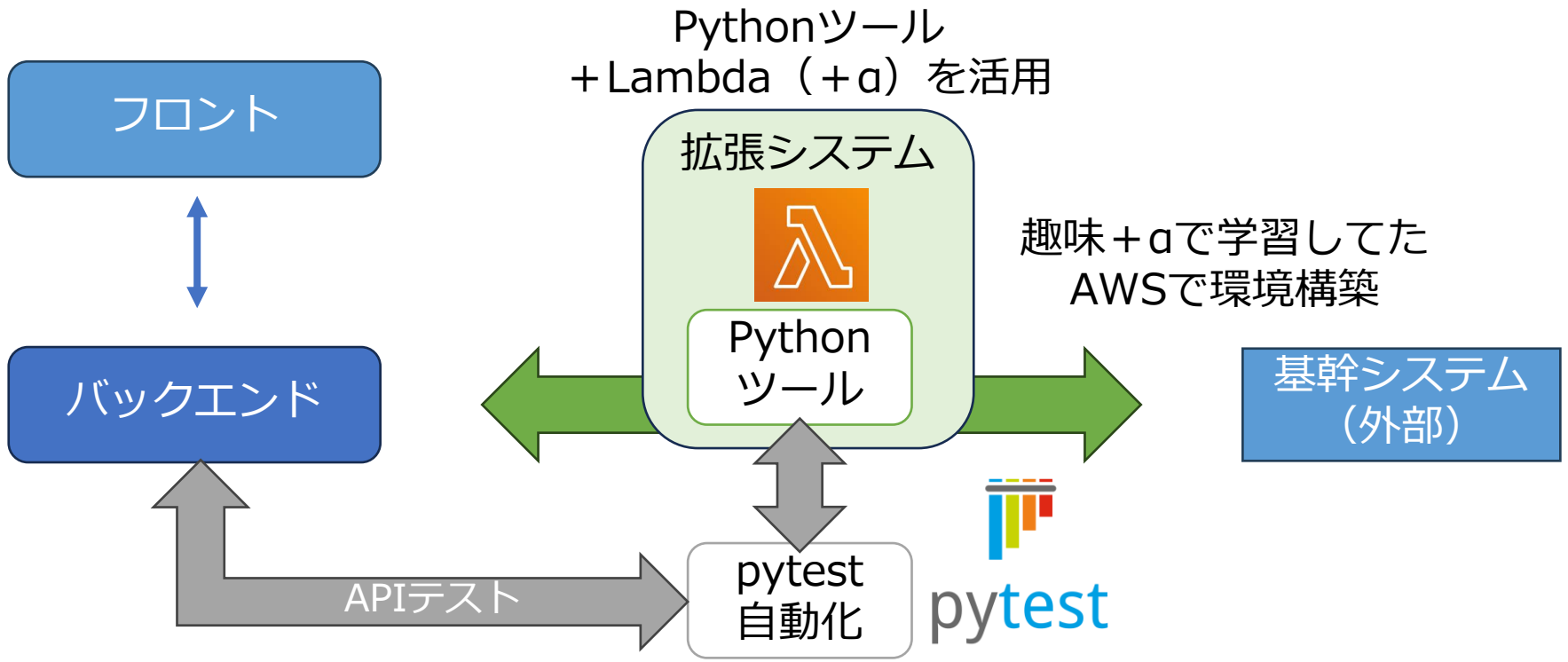


日々の改善　※多数実施している中から抜粋

鎮火、今の現場は
何事もなかったように作業中

＜顧客提供までのビジネスプロセス＞

・このツールが特定顧客の拡張機能として提供される方向へ…　※テストにも活用



多少安定したが…

自動テストも徐々に増えてきて、テストを育てて「資産化」していく
いちおう、システムも拡張しつつ、ビジネスとしては無難に成長中

サブスク型契約 / 事業会社の自社開発

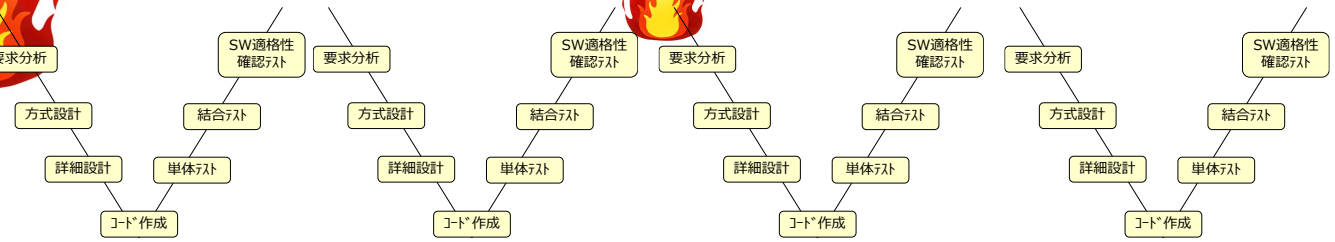
継続的な開発が前提となるため
リグレッションテストの資産化が重要となる

技術選定

適宜リリース

適宜リリース

適宜リリース



クラウド/インフラ構築

アプリ開発とインフラ構築が並列で可能

反復的、インクリメンタルな派生開発へ

手動テスト

手動テスト

自動テスト

手動テスト

自動テスト

手動テスト

自動テスト

世の中のフレームワーク・ライブラリ

だいたいやってた活動

※主張：QAやったことないのでよくわかりません

雑分類	過去：大規模SoSプロジェクト	現在：新規事業SaaSプロダクト
（事業提案）	なし/RFPからの提案書作成	プロダクト状況を考慮した提案
プロジェクトマネジメント	サブシステム単位、プロジェクト単位などでマネジメント実施	プロダクト開発を実施 顧客次第で小プロジェクトを実施
システム設計 要求分析/要件定義	システム全体設計と運用・移行設計 各サブシステムの要求分析など	継続的にバックログを作成 必要に応じて顧客の業務分析を実施
ソフトウェア設計	難易度の高い要素、問題をフォロー	仕様案作成、外部依頼＆内部で実装
（実装）	開発会社依頼、テストツール作成 C/C++中心	一部システムの実装、自動化など Python/JavaScriptなど
環境構築	PC調達、海外調達品交渉、NW構築、 OSセットアップ、アプリインストール	AWSでインフラ構築
テスト設計	設計に対してテスト設計	設計時にリグレッションテスト等検討
テスト実行	品質管理チームがテスト実行が多い	チーム or 自分でテストを実施
テスト自動化、ツール整備	現場改善でツール・システム構築	各種自動テスト構築
運用保守	顧客側の運用意見をベースに改善	プロダクト状況をモニタしつつ改善
不具合対応・管理	不具合解析、傾向分析など実施	不具合解析、傾向分析など実施

おまけ：過去と現在のテストの変化（運用時、継続的なテストの取り扱い）

雑分類	過去：大規模SoSプロジェクト	現在：新規事業SaaSプロダクト
リリース毎の周期（目安）	3か月に1回	1か月に1回
テスト設計の方法	＜仕様変更向け＞ 毎回のリリースに前にテスト設計 ＜リグレッション向け＞ 影響がありそうな部分について テスト設計→テストケース作成	＜仕様変更向け＞ 設計・仕様検討段階で検討 複雑な部分はテスト設計 ＜リグレッション向け＞ 定期にリグレッションテスト設計見直し ※構造、構成から見直しも実施
リリース毎のテスト実施方法	手動テスト（変更箇所中心） 簡易なリグレッションテスト （手動、自動双方実施）	手動テスト（変更箇所中心） 手動リグレッションテスト 自動リグレッションテスト
手動リグレッションテスト	影響範囲に応じて実施 ※品質管理チームが実行	毎回TestRail項目を実施（数時間程度） ※スタッフ兼務の方が実行
自動リグレッションテスト	同じUIテストパターンを毎回実行	APIテストを中心に蓄積、毎回実行
テストケース管理ツール	Excel様（差分をシート追加）	TestRail（常に更新）
蓄積テストケースの活用法	仕様の確認に使う 不具合発生時の改善に使う	仕様の確認に使う 不具合発生時の改善に使う リグレッションテストを資産として扱う

体感的変化まとめ：変化と影響の大きな技術

雑分類	過去	現在
開発の進め方	SIer：プロジェクト型 リリース後は定期リリース	事業会社：プロダクト開発型 継続的デプロイが前提
プロセス	ク〇デカV字、標準ガチガチ →初期リリース後はミニV字型	カウボーイリリース →混乱→月単位のミニV字型へ
リリース周期	初期リリース1年超 その後は3～6か月単位	顧客提供まではマイルストン型 顧客提供後は1か月単位
テスト実施方法	ほぼ手動テスト 一部自動テスト	一部手動テスト ほぼ自動テスト
デプロイ環境	オンプレ型（PC調達・保守込み）	クラウド型（構築も実施）
デプロイコスト	非常に大きい	小さい

技術的な変化要因（特徴的なものだけPickUp）

- クラウド
- 通信技術
- 定番フレームワーク（Ruby on Rails/Spring Bootなど）
- サブスク契約　※ビジネス上の影響は大きい

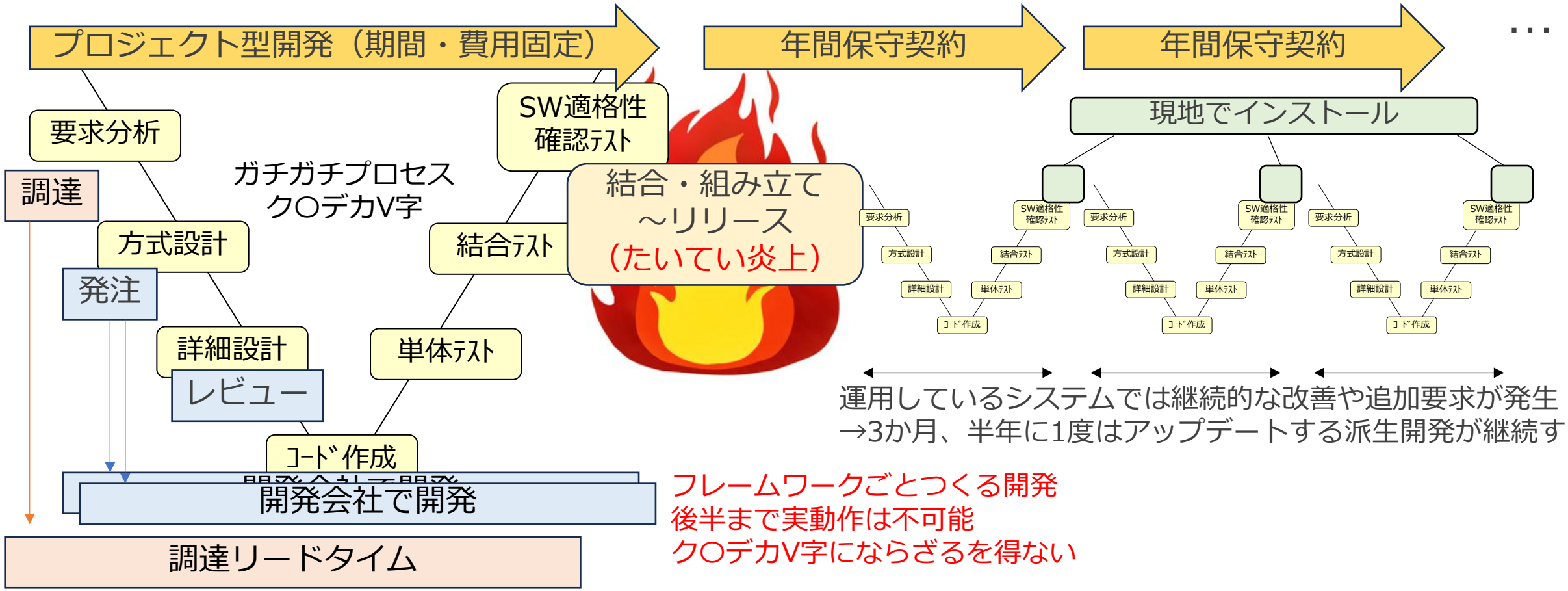
昔から変わったよね、だけでは
何の知見にもならないので、
どういう要因で変わっているか？を
考えてみます

変化する開発
～変化の要因・分析～

元々：開発現場も、ビジネスにおいてもリスク対策の難易度が高い状況だった

よほどの技術がないとリスクを後回しにせざるを得ない
（顧客期待、非機能のカバーなど）
リリースまでのリードタイムで外部環境が変わる

保守契約を超えるコスト発生の場合も
リリースのコストがかさむ状況
※手動テスト、現地インストール…

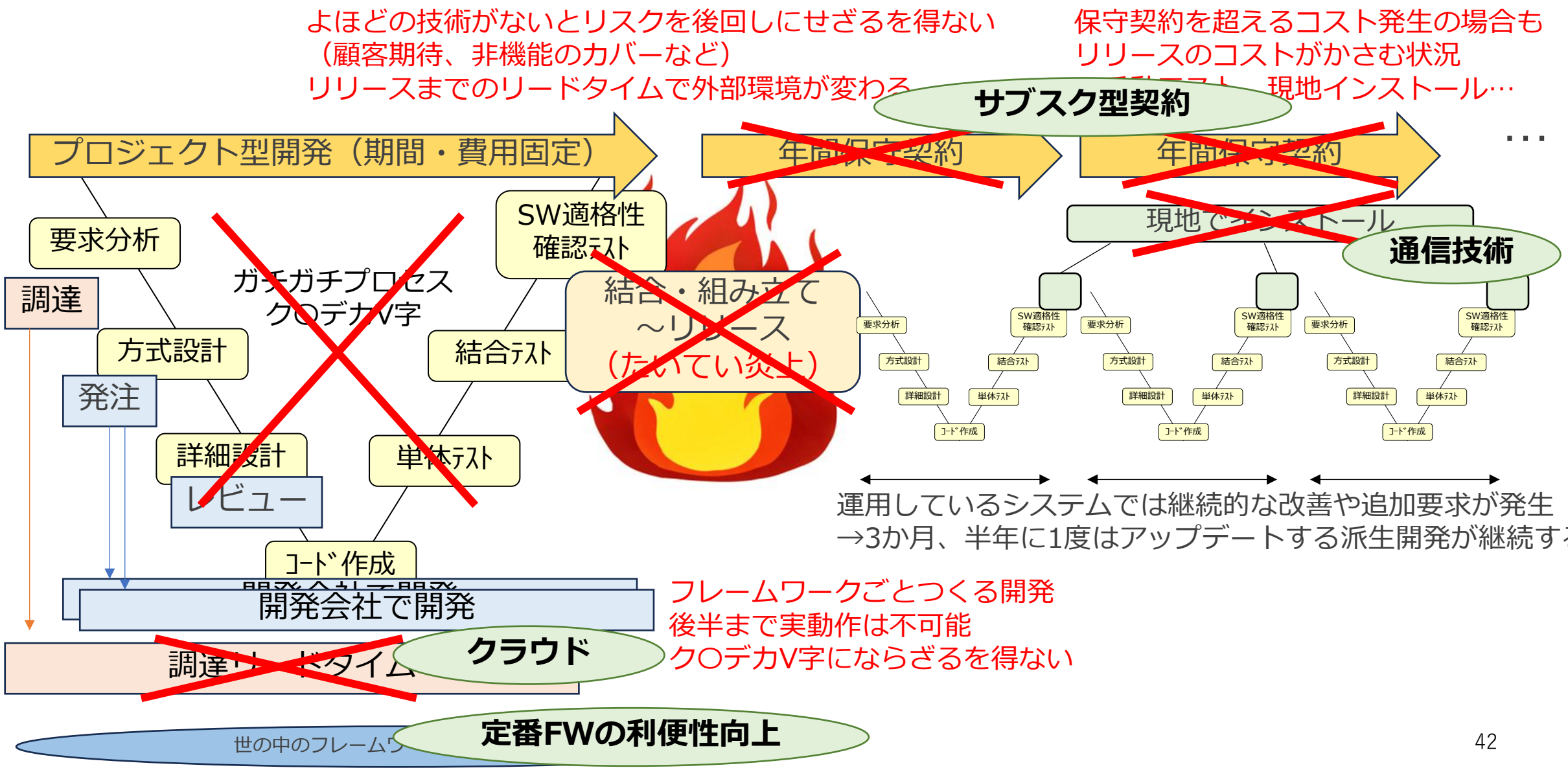


運用しているシステムでは継続的な改善や追加要求が発生
→3か月、半年に1度はアップデートする派生開発が継続する

フレームワークごとつくる開発
後半まで実動作は不可能
クソデカV字にならざるを得ない

世の中のフレームワーク・ライブラリ

開発・ビジネスにおけるペインを技術や新しい仕組みが解決していく



変化の要因：技術による解決の結果

多くの技術により小さい反復でのインクリメンタルな開発がやりやすくなった
開発初期から小さく開発できることで、ビジネスリスクも低減される

外部環境の変化にも即時対応ができる、ビジネスリスクは低減

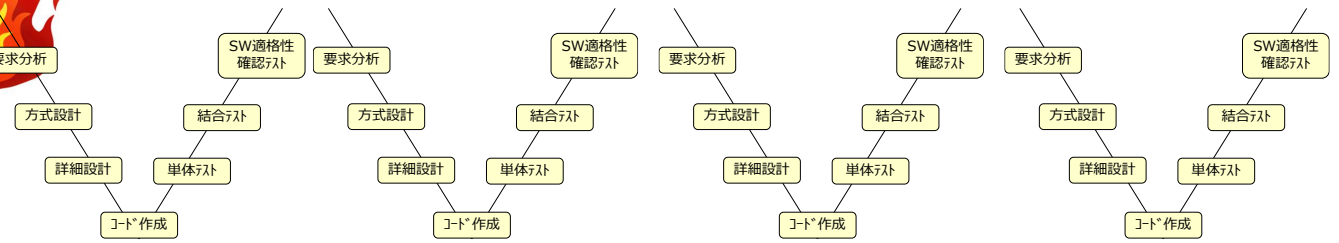
サブスク型契約 or 事業会社の自社開発

初期から小さく動作させ続けることが可能
その分、技術選定が重要な仕事となる
顧客に受け入れられるものを作らないと消える

注：炎上するときは
炎上する

継続的な開発が前提となるため
リグレッションテストの資産化が重要となる

技術選定 | 適宜リリース | 適宜リリース | 適宜リリース



反復的、インクリメンタルな派生開発が続くことは変わらない
※むしろ、開発初期からできるようになった

クラウド/インフラ構築

アプリ開発とインフラ構築が並列で可能
非機能リスクも札束で殴れば解決できるケースも

世の中のフレームワーク・ライブラリ

手動テスト

手動テスト
自動テスト

手動テスト
自動テスト

手動テスト
自動テスト

変化する開発：変化の要因まとめと、変化を拒む「慣性」

- ペインを解決する方向へ世の中は変化する、たいていは良い方向となる
  - クラウド、フレームワーク、通信技術、スマホ、あらゆる技術はペインの解決へ向かう
  - 生成AIの変化は？少なくとも開発の難易度がさらに低下しているのは事実
  - 最近は価値があると判断すると大規模の資金投入が発生→一気に拡大する傾向がある
    - Cursorの開発元のAnysphere社は創業3年で企業価値（調達などで決まる）が1.4兆円相当とか

- 最近の生成AIに限らず、世の中は常に変化している
  - 不安だとしても必然的に変化は発生する、外部環境はコントロールできない
  - 今までの変化に適応できているなら、これからの変化へ備えることもできるはず

- 特定の環境に適応していた組織では慣性が働き、即時の変化対応が困難
  - 組織レベルで即時に変化に適応できないことが要因で消えていくケースもある
    ※例：コダック（フィルム→デジカメ）、ノキア（携帯電話→スマホなど）
  - 企業側が変化を拒むも不正利用が発生→強制的に変わるケースも（音楽業界：Napster/Winny）
  - 個人レベルでは心持で対処できるだろうと考えている

世の中は必然的に変わる
変化に適応せず消えていく企業もある、個人も変化に適応する必要がある
でも、どう適応するとよいか？

進化する体系、時代に適応する知識
～実践ソフトウェアエンジニアリングより～

進化する体系：教えはどうなってんだ、教えは？

某書籍より、2000年頃の第4版から2019年の第9版への変化　※それ以前は除外

第４版（日本語版2000年）

第１部　製品とプロセス
　第１章　製品
　第２章　プロセス
第２部　ソフトウェアプロジェクトの管理
　第３章　プロジェクト管理の理念
　第４章　プロセスとメトリクス
　第５章　ソフトウェアプロジェクトの計画立案
　第６章　リスク管理
　第７章　プロジェクトのスケジューリングと追跡
　第８章　ソフトウェアの品質保証
　第９章　ソフトフェア構成管理
第３部　ソフトウェア工学の伝統的手法
　第１０章　システム構想設計
　第１１章　要求分析の基本概念と原則
　第１２章　要求分析モデル
　第１３章　設計の基本概念と原則
　第１４章　設計の手法
　第１５章　リアルタイムシステムの設計
　第１６章　ソフトウェアテスト技術
　第１７章　ソフトウェアテスト戦略
　第１８章　技術的なソフトウェアメトリクス
第４部　オブジェクト指向ソフトウェア工学
　第１９章　オブジェクト指向の基本概念と原則
　第２０章　オブジェクト指向分析
　第２１章　オブジェクト指向設計
　第２２章　オブジェクト指向テスト
　第２３章　オブジェクト指向システムの技術的メトリクス
第５部　ソフトウェア工学の進んだ話題
　第２４章　フォーマルメソッド
　第２５章　クリーンルーム開発
　第２６章　ソフトウェア再利用
　第２７章　リエンジニアリング
　第２８章　クライアント・サーバソフトウェア工学
　第２９章　CASE：コンピュータ支援ソフトウェア開発
　第３０章　進むべき道

第6版（日本語版2005年）

　第１章ソフトウェアとトウェアエンジニアリング
第１部　ソフトウェアプロセス
　第２章　プロセス
　第３章　規範的なプロセスモデル
　第４章　アジャイル開発
第２部　ソフトウェアエンジニアリングの実践
　第５章　プラクティス　－一般的な考え方
　第６章　システムエンジニアリング
　第７章　要求エンジニアリング
　第８章　要求分析モデリング
　第９章　設計エンジニアリング
　第１０章　アーキテクチャ設計
　第１１章　コンポーネントレベル設計
　第１２章　ユーザインタフェース設計
　第１３章　ソフトウェアテスト戦略
　第１４章　ソフトウェアテスト技術
　第１５章　成果物に関するソフトウェアメトリクス
第３部　Webエンジニアリングの適用
　第１６章　Webエンジニアリング
　第１７章　Webエンジニアリング他のための定式化と計画
　第１８章　Webアプリケーションのための分析モデリング
　第１９章　Webアプリケーションの設計モデリング
　第２０章　Webアプリケーションのテスト
第４部　ソフトウェアプロジェクトの管理
　第２１章　プロジェクトマネジメントの概念
　第２２章　プロセスとプロジェクトのメトリクス
　第２３章　ソフトウェアプロジェクトの見積もり
　第２４章　ソフトウェアプロジェクトスケジューリング
　第２５章　リスクマネジメント
　第２６章　品質マネジメント
　第２７章　変更管理
第５部　ソフトウェアエンジニアリングの先進トピック
　第２８章　フォーマルメソッド
　第２９章　クリーンルーム開発
　第３０章　コンポーネントベース・ソフトウェアエンジニアリング
　第３１章　リエンジニアリング
　第３２章　進むべき道

第9版（日本語版2021年）

　第１章ソフトウェアとソフトウェアエンジニアリング
第１部　ソフトウェアプロセス
　第２章　プロセス
　第３章　アジャイルとプロセス
　第４章　推奨のプロセスモデル
　第５章　ソフトウェアエンジニアリングの人間的側面
第２部　モデリング
　第６章　プラクティスの指針となる原則
　第７章　要求エンジニアリング
　第８章　要求分析モデリングの推奨手法
　第９章　設計の概念
　第１０章　アーキテクチャ設計の推奨手法
　第１１章　コンポーネント設計
　第１２章　ユーザエクスペリエンス設計
　第１３章　移動体端末におけるソフトウェアの設計
　第１４章　パターンに基づく設計
第３部　品質とセキュリティ
　第１５章　品質の概念
　第１６章　レビューの推奨手法
　第１７章　ソフトウェア品質保証
　第１８章　ソフトウェアセキュリティエンジニアリング
　第１９章　ソフトウェアテスト－コンポーネントレベル
　第２０章　ソフトウェアテスト－統合レベル
　第２１章　ソフトウェアテスト－移動体端末と…に対するテスト
　第２２章　ソフトウェア構成マネジメント
　第２３章　ソフトウェアメトリクスと分析
第４部　ソフトウェアプロジェクトのマネジメント
　第２４章　プロジェクトマネジメントの概念
　第２５章　実行可能で役立つソフトウェア計画
　第２６章　リスクマネジメント
　第２７章　ソフトウェアサポート戦略
第５部　先進的な話題
　第２８章　ソフトウエアプロセス改善
　第２９章　ソフトウェアエンジニアリングの新興トレンド
　第３０章　おわりに
付録１　UML入門
付録２　ソフトウェアエンジニアリのためのデータサイエンス

進化する体系：教えはどうなってんだ、教えは？

ファッションのように変化するものも多い

第４版（日本語版2000年）

第１部　製品とプロセス
　第１章　製品
　第２章　プロセス
第２部　ソフトウェアプロジェクトの管理
　第３章　プロジェクト管理の理念
　第４章　プロセスとメトリクス
　第５章　ソフトウェアプロジェクトの計画立案
　第６章　リスク管理
　第７章　プロジェクトのスケジューリングと追跡
　第８章　ソフトウェアの品質保証
　第９章　ソフトフェア構成管理
第３部　ソフトウェア工学の伝統的手法
　第１０章　システム構想設計
　第１１章　要求分析の基本概念と原則
　第１２章　要求分析モデル
　第１３章　設計の基本概念と原則
　第１４章　設計の手法
　第１５章　リアルタイムシステムの
　第１６章　ソフトウェアテスト技術
　第１７章　ソフトウェアテスト戦略
　第１８章　技術的なソフトウェアメ
第４部　オブジェクト指向ソフトウェア工学
**　第１９章　オブジェクト指向の基本概念と原則**
**　第２０章　オブジェクト指向分析**
**　第２１章　オブジェクト指向設計**
**　第２２章　オブジェクト指向テスト**
**　第２３章　オブジェクト指向システムの技術的メトリクス**
第５部　ソフトウェア工学の進んだ話題

オブジェクト指向がホット

第6版（日本語版2005年）

　第１章ソフトウェアとトウェアエンジニアリング
第１部　ソフトウェアプロセス
　第２章　プロセス
　第３章　規範的なプロセスモデル
**　第４章　アジャイル開発**
第２部　ソフトウェアエンジニアリングの実践
　第５章　プラクティス　－一般的な考え方
　第６章　システムエンジニアリング
　第７章　要求エンジニアリング
　第８章　要求分析モデリング
　第９章　設計エンジニアリング
**　第１０章　アーキテクチャ設計**
　第１１章　コンポーネントレベル設計
**　第１２章　ユーザインタフェース設計**
　第１３章　ソフトウェアテスト戦略
　第１４章　ソフトウェアテスト技術
　第１５章　成果物に関するソフトウェアメトリクス
第３部　Webエンジニアリングの適用
**　第１６章　Webエンジニアリング**
**　第１７章　Webエンジニアリング他のための定式化と計画**
**　第１８章　Webアプリケーションのための分析モデリング**
**　第１９章　Webアプリケーションの設計モデリング**
**　第２０章　Webアプリケーションのテスト**
第４部　ソフトウェアプロジェクトの管理
　第２１章　プロジェクトマネジメントの概念
　第２２章　プロセスとプロジェクトのメトリクス
　第２３章　ソフトウェアプロジェクトの見積もり
　第２４章　ソフトウェアプロジェクトスケジューリング

アジャイル開発アーキテクチャ、UI設計、Webへの取り組みが登場

第9版（日本語版2021年）

　第１章ソフトウェアとソフトウェアエンジニアリング
第１部　ソフトウェアプロセス
　第２章　プロセス
　第３章　アジャイルとプロセス
**　第４章　推奨のプロセスモデル**
**　第５章　ソフトウェアエンジニアリ**
第２部　モデリング
**　第６章　プラクティスの指針となる原**
**　第７章　要求エンジニアリング**
**　第８章　要求分析モデリングの推奨手**
**　第９章　設計の概念**
**　第１０章　アーキテクチャ設計の推奨**
**　第１１章　コンポーネント設計**
**　第１２章　ユーザエクスペリエンス設**
**　第１３章　移動体端末におけるソフトウェアの設計**
**　第１４章　パターンに基づく設計**
第３部　品質とセキュリティ
**　第１５章　品質の概念**
**　第１６章　レビューの推奨手法**
**　第１７章　ソフトウェア品質保証**
**　第１８章　ソフトウェアセキュリティエンジニアリング**
**　第１９章　ソフトウェアテスト－コンポーネントレベル**
**　第２０章　ソフトウェアテスト－統合レベル**
**　第２１章　ソフトウェアテスト－移動体端末と…に対するテスト**
**　第２２章　ソフトウェア構成マネジメント**
**　第２３章　ソフトウェアメトリクスと分析**
第４部　ソフトウェアプロジェクトのマネジメント
　第２４章　プロジェクトマネジメントの概念

推奨のプロセス
人間的側面が追加

モデリングが部としてまとまる
UX設計、移動体端末への取り組み

品質とセキュリティが部となる

情報技術（IT：Information Technology）による変化は非常に速く、しかも容赦がない。
後退すると元に戻れないため、企業は最新の技術をどんどん身につけていくか、
自らが消え去るかのどちらかを選ばざるを得ない。
〈中略〉モルモットが輪っかのなかを一生懸命走っているようなものだ。

進化する体系：教えはどうなってんだ、教えは？

とはいえ、長年かわらない分野もある。プロセスとマネジメントは安定した領域

第４版（日本語版2000年）

第１部　製品とプロセス
　第１章　製品
　第２章　プロセス
第２部　ソフトウェアプロジェクトの管理
　第３章　プロジェクト管理の理念
　第４章　プロセスとメトリクス
　第５章　ソフトウェアプロジェクトの計画立案
　第６章　リスク管理
　第７章　プロジェクトのスケジューリングと追跡
　第８章　ソフトウェアの品質保証
　第９章　ソフトフェア構成管理
第３部　ソフトウェア工学の伝統的手法
　第１０章　システム構想設計
　第１１章　要求分析の基本概念と原則
　第１２章　要求分析モデル
　第１３章　設計の基本概念と原則
　第１４章　設計の手法
　第１５章　リアルタイムシステムの設計
　第１６章　ソフトウェアテスト技術
　第１７章　ソフトウェアテスト戦略
　第１８章　技術的なソフトウェアメトリクス
第４部　オブジェクト指向ソフトウェア工学
　第１９章　オブジェクト指向の基本概念と原則
　第２０章　オブジェクト指向分析
　第２１章　オブジェクト指向設計
　第２２章　オブジェクト指向テスト
　第２３章　オブジェクト指向システムの技術的メトリクス
第５部　ソフトウェア工学の進んだ話題
　第２４章　フォーマルメソッド
　第２５章　クリーンルーム開発
　第２６章　ソフトウェア再利用
　第２７章　リエンジニアリング
　第２８章　クライアント・サーバソフトウェア工学
　第２９章　CASE：コンピュータ支援ソフトウェア開発
　第３０章　進むべき道

第6版（日本語版2005年）

　第１章ソフトウェアとトウェアエンジニアリング
第１部　ソフトウェアプロセス
　第２章　プロセス
　第３章　規範的なプロセスモデル
　第４章　アジャイル開発
第２部　ソフトウェアエンジニアリングの実践
　第５章　プラクティス　－一般的な考え方
　第６章　システムエンジニアリング
　第７章　要求エンジニアリング
　第８章　要求分析モデリング
　第９章　設計エンジニアリング
　第１０章　アーキテクチャ設計
　第１１章　コンポーネントレベル設計
　第１２章　ユーザインタフェース設計
　第１３章　ソフトウェアテスト戦略
　第１４章　ソフトウェアテスト技術
　第１５章　成果物に関するソフトウェアメトリクス
第３部　Webエンジニアリングの適用
　第１６章　Webエンジニアリング
　第１７章　Webエンジニアリング他のための定式化と計画
　第１８章　Webアプリケーションのための分析モデリング
　第１９章　Webアプリケーションの設計モデリング
　第２０章　Webアプリケーションのテスト
第４部　ソフトウェアプロジェクトの管理
　第２１章　プロジェクトマネジメントの概念
　第２２章　プロセスとプロジェクトのメトリクス
　第２３章　ソフトウェアプロジェクトの見積もり
　第２４章　ソフトウェアプロジェクトスケジューリング
　第２５章　リスクマネジメント
　第２６章　品質マネジメント
　第２７章　変更管理
第５部　ソフトウェアエンジニアリングの先進トピック
　第２８章　フォーマルメソッド
　第２９章　クリーンルーム開発
　第３０章　コンポーネントベース・ソフトウェアエンジニアリング
　第３１章　リエンジニアリング
　第３２章　進むべき道

第9版（日本語版2021年）

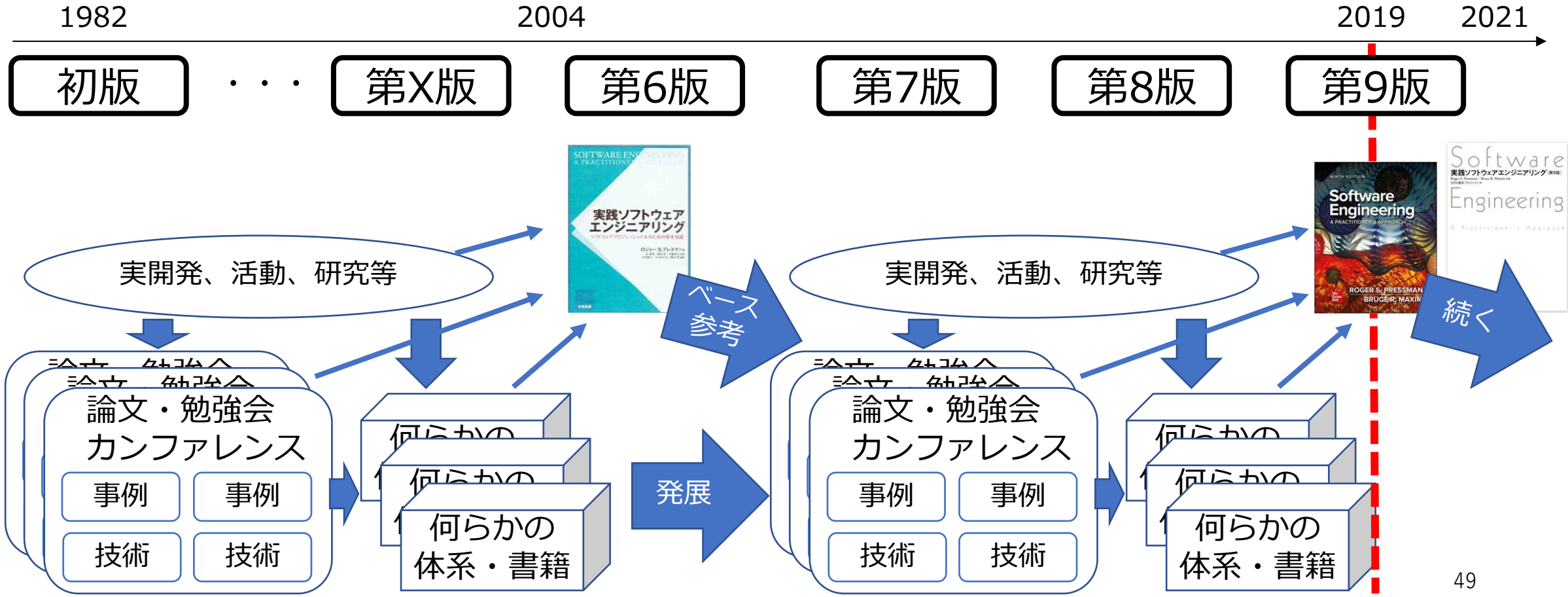
　第１章ソフトウェアとソフトウェア
第１部　ソフトウェアプロセス
　第２章　プロセス
　第３章　アジャイルとプロセス
　第４章　推奨のプロセスモデル
　第５章　ソフトウェアエンジニアリングの人間的側面
第２部　モデリング
　第６章　プラクティスの指針となる原則
　第７章　要求エンジニアリング
　第８章　要求分析モデリングの推奨手法
　第９章　設計の概念
　第１０章　アーキテクチャ設計の推奨手法
　第１１章　コンポーネント設計
　第１２章　ユーザエクスペリエンス設計
　第１３章　移動体端末におけるソフトウェアの設計
　第１４章　パターンに基づく設計
第３部　品質とセキュリティ
　第１５章　品質の概念
　第１６章　レビューの推奨手法
　第１７章　ソフトウェア品質保証
　第１８章　ソフトウェアセキュリティエンジニアリング
　第１９章　ソフトウェアテスト－コンポーネントレベル
　第２０章　ソフトウェアテスト－統合レベル
　第２１章　ソフトウェアテスト－移動体端末と…に対するテスト
　第２２章　ソフトウェア構成マネジメント
　第２３章　ソフトウェアメトリクスと分析
第４部　ソフトウェアプロジェクトのマネジメント
　第２４章　プロジェクトマネジメントの概念
　第２５章　実行可能で役立つソフトウェア計画
　第２６章　リスクマネジメント
　第２７章　ソフトウェアサポート戦略
第５部　先進的な話題
　第２８章　ソフトウエアプロセス改
　第２９章　ソフトウェアエンジニア
　第３０章　おわりに
付録１　UML入門
付録２　ソフトウェアエンジニアリのためのデータサイエンス

プロセスは常に存在
※拡大傾向

マネジメントも残る
※他領域へ分離も

体系の構築、変化のイメージ

・少し時間遅れのある情報がまとまられ、後続への高速道路化はされている
  - 最新付近まで追いつきたければ情報は存在している
・出版されたタイミングの、変化し続ける世界における「局面」がまとまっている



体系も進化し、昔は見向きもしなかった人も使うようにはなりつつある
しかし、ストレートに使えるものではない、各開発現場への「適応」が必要となる

・実践ソフトウェアエンジニアリングからの引用（第1章、序文）
～博士（Pressman氏）と青年（ゲーム（GTA？）開発＆AI機能の担当者）の会話

私は、世界有数の有名なファーストパーソン・シューディング（FPS：First-Person Shooter）ゲームの最新ビルドを見ていた。
（中略）
博士：（ 否定を予想しつつつつ）ソフトウェアエンジニアリングの手法は何か使っているのかね？
青年：（ ゆっくりと頷き）ニーズに応じてですが、もちろん使っていますよ。
（中略）
僕たちは IT部門や航空宇宙会社にいるわけじゃないから、博士が提唱するものをカスタマイズしないと使えないんです。でも肝心なところは同じで、高品質の製品を作らなければいけない。そして、繰り返し可能な方法で達成するには、独自のソフトウェアエンジニアリング技術のサブセットを作り、適応させないとダメなんです。

引用：実践ソフトウェアエンジニアリング（第9版）
Roger.S.Pressman他、SEPA翻訳プロジェクト、オーム社、2021年

昨今の品質系技術はビジネス傾向・開発の特徴により多様化の傾向あり
→実践ソフトウェアエンジニアリング 第8版、第9版より

＜第8版＞

ビジネスと特定技術で分けている

・・・
第３部　品質マネジメント
　第１９章　品質の概念
　第２０章　レビュー技法
　第２１章　ソフトウェア品質保証
　第２２章　ソフトウェアテスト戦略
　第２３章　従来のアプリ向けテスト
　第２４章　オブジェクト指向アプリ向けテスト
　第２５章　Webアプリ向けテスト
　第２６章　モバイルアプリ向けテスト
　第２７章　セキュリティエンジニアリング
　第２８章　フォーマルメソッドと検証
　第２９章　ソフトウェア構成マネジメント
　第３０章　プロダクトメトリクス

＜第9版＞

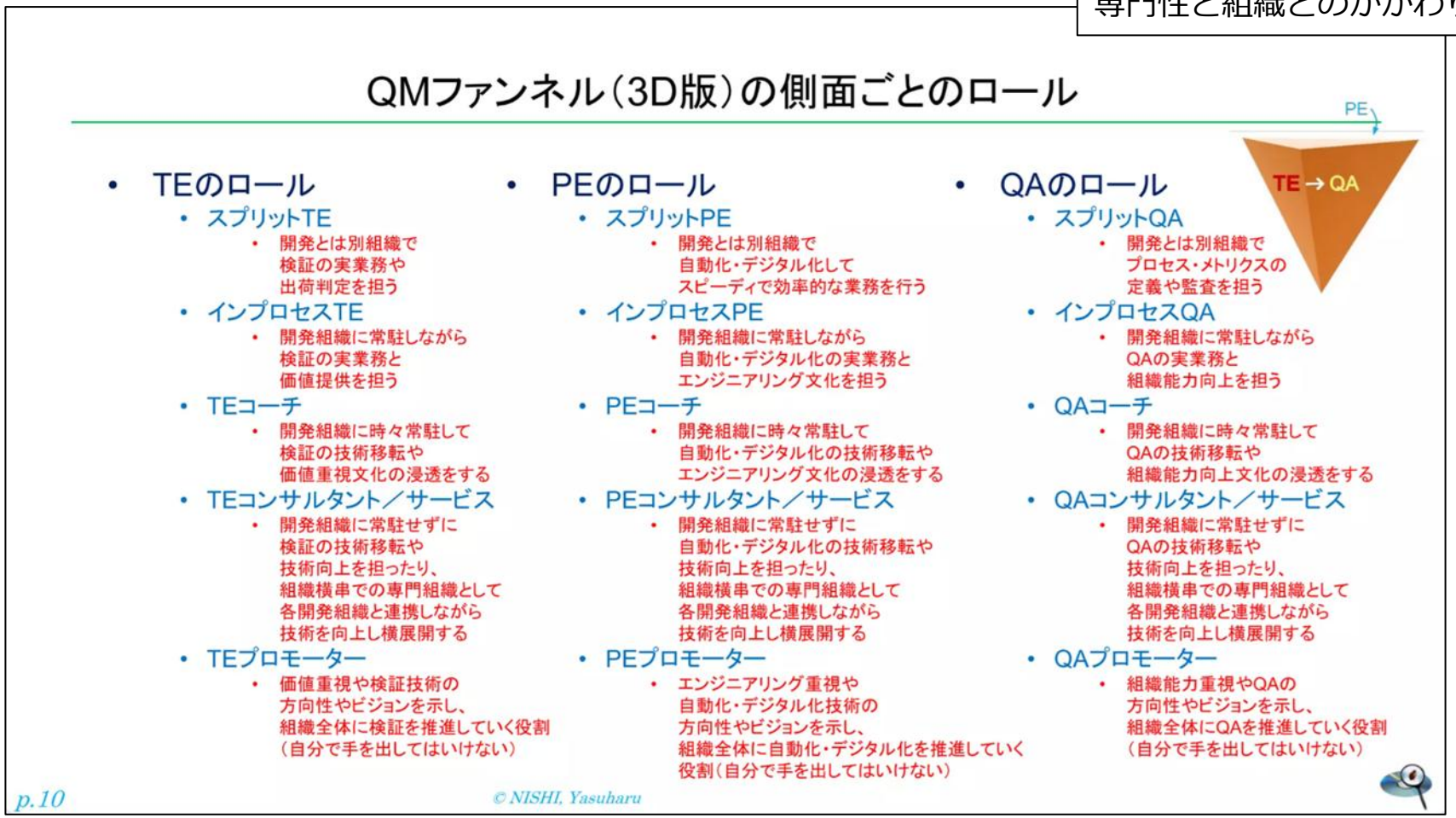
ビジネスと専門性で分けている

・・・
第３部　品質とセキュリティ（一部のみ記載）
　第１８章　ソフトウェアセキュリティエンジニアリング
　第１９章　ソフトウェアテスト－コンポーネントレベル
　第２０章　ソフトウェアテスト－統合レベル
　第２１章　ソフトウェアテスト－移動体端末と特定ドメインに対するテスト
　　21.1　モバイルアプリにおけるテストのガイドライン
　　21.2　モバイルアプリにおけるテスト戦略
　　21.3　UXを考慮したテストの課題
　　21.4　Webアプリのテスト
　　21.5　Webアプリにおけるテスト戦略
　　21.6　国際化
　　21.7　セキュリティテスト
　　21.8　性能テスト
　　21.9　リアルタイムテスト
　　21.10　AIシステムのテスト
　　21.11　ゲームとシミュレーションのテスト
　　21.12　ドキュメントとヘルプ機能のテスト
・・・

引用：実践ソフトウェアエンジニアリング（第9版）
Roger.S.Pressman他、SEPA翻訳プロジェクト、オーム社、2021年

昨今の品質系技術はビジネス傾向・開発の特徴により多様化の傾向あり
→にしさん資料：QMファンネル

専門性と組織とのかかわりで分けている



引用：品質を加速させるために、テスターを増やす前から考えるべきQMファンネルの話（3D版）
https://www.slideshare.net/slideshow/quality-management-funnel-3d-how-to-organize-qarelated-roles-and-specialties/249498558

昨今の品質系技術はビジネス傾向・開発の特徴により多様化の傾向あり
→ソフトウェアテスト徹底指南書、JSTQBの構成など

開発スタイルで分けている

Part Ⅱ テストの戦略とプロセス
- 第5章　定番のテスト戦略
- 第6章　アジャイル開発でのテスト戦略
- 第7章　継続的デリバリーでのテスト戦略
- 第8章　DevOpsでのテスト戦略
- 第9章　ソフトウェアプロダクトライン開発でのテスト戦略

引用：ソフトウェアテスト徹底指南書
井芹 洋輝、技術評論社、2025年

ビジネスと専門性で分けている
ISTQBにはさらに
Certified Tester Model-Based Tester
Certified Tester Gambling Industry Tester
などが存在

- Core Foundation
  - FLシラバス
- Core Advanced
  - テストマネージャ（ALTM）
  - テストアナリスト（ALTA）・テクニカルテストアナリスト（ALTTA）
- Specialist
  - FLシラバス（Specialist） AIテスティング
  - FLシラバス（Specialist）自動車ソフトウェアテスト担当者
  - FLシラバス（Specialist）モバイルアプリケーションテスト担当者
  - FLシラバス（Specialist） 性能テスト担当者
  - FLシラバス（Specialist） ゲームテスト
  - ALシラバス（Specialist）テスト自動化エンジニア
- Agile

  Advanced Level
  - -ALシラバス アジャイルテクニカルテスト担当者

  Foundation Level
  - FLシラバス（Extension）アジャイルテスト担当者

JSTQBサイトより（https://jstqb.jp/syllabus.html）

大きな単位は体系が役立つとして、
技術が多様化し、目まぐるしく変化
する状況に対してどう適応するか？
変わりやすい/変わりづらい部分を
分けながら、変化への適応について
考え方の一例を紹介します

時代に適応する
ソフトウェアエンジニアの考え方
～変わり続ける現場で生き抜くために～

・実践ソフトウェアエンジニアリングからの引用

> 情報技術（IT：Information Technology）による変化は非常に速く、しかも容赦がない。
> 後退すると元に戻れないため、企業は最新の技術をどんどん身につけていくか、
> 自らが消え去るかのどちらかを選ばざるを得ない。
> 〈中略〉モルモットが輪っかのなかを一生懸命走っているようなものだ。

・「速く、しかも容赦がない」世界を40年超見ており、適応方法もうかがえる

・以下、「ソフトウェアエンジニアリングの階層」を流用した考え方を紹介
- ツール
- 手法
- プロセス（＋問題解決とします）
- 品質に焦点を合わせる（Quality Focus）
  ※本発表ではQuality Focusの用語を使う

図1.3 ソフトウェアエンジニアリングの階層より流用した構成

・上層は目まぐるしく変わるが、下層の変化は少ない

引用：実践ソフトウェアエンジニアリング（第9版）
Roger.S.Pressman他、SEPA翻訳プロジェクト、オーム社、2021年

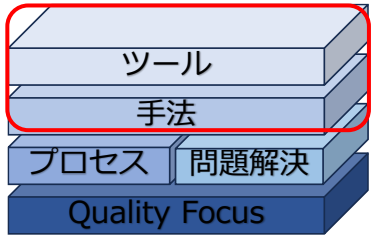
ソフトウェアエンジニアリングの階層：ツール・手法

この分野にて学んだことの1つは、ソフトウェアエンジニアリングの実践者は「流行に敏感」ということだ。先へ進む道には、誇大広告にもかかわらず実際にはうまくいかなかったエキサイティングな新テクノロジー（まさに最新の流行）が死屍累々と転がっている。

■ツール
・多数のツールが常に登場し続ける、変化が最も早い分野

■手法
・オブジェクト指向、DDD 、RDRA、VSTeP、テストカタマリー（？）など
・こちらも新しい提案が登場し続ける、適度に楽しく学び続けるのが一番

・新しい技術は背景を知ることができると適用判断に役立つ
- 例：カオスエンジニアリングはAWS黎明期の超絶な低品質環境に対する提案からスタート
- 適用箇所によりで役に立つ、立たないというものが存在する
→使い方を適切に理解することが大事

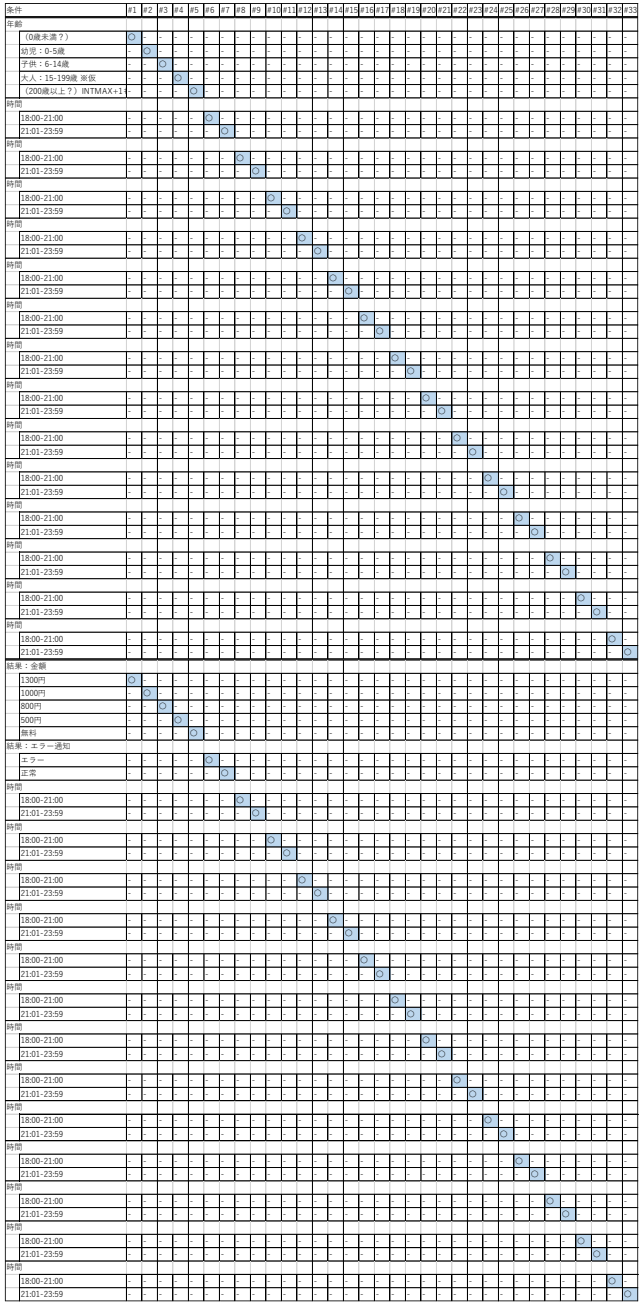


引用：実践ソフトウェアエンジニアリング（第9版）
Roger.S.Pressman他、SEPA翻訳プロジェクト、オーム社、2021年

使い方を適切に理解することが大事
→無理やりツールや手法を適用することはやめましょう

・うまくいかない場合、無理やり使わない
- 適切な適用場所・範囲がある、それを（体験も含め）知ることは大事
- 素振り、「やってみる」ことは重要、ただし個人で＆結果は分析しよう
- うまくいかなくても、時間をおいてあたらめてみると使えるものもある

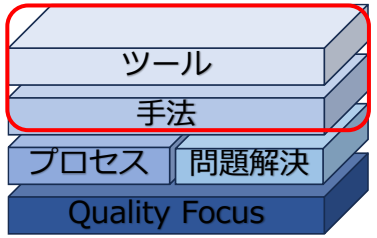
・チーム、組織への展開は特に丁寧にやること
- 使う人、作る人の役割は大きく異なる
- 自分ができても他の人はできないことも多い
- 特に他の人へ提案する際には、自分がしっかり理解している必要がある
- 教育、原理原則の説明とセットとした方がよい



「速く、しかも容赦がない」、とはいえ…

ソフトウェアエンジニアリング技術の変化は確かに「速く、しかも容赦がない」のだが、
その一方で普及の歩みは非常に遅い。新しいプロセス、手法やツールを採用することを決定するまでに、対象の理解に必要な教育や訓練を実施する。そして技術の導入により、ソフトウェア開発文化に何か新しい（そして素晴らしい）ことが少しずつ生まれる。そこから新しいプロセスが動きはじめる。
（中略）
過去の歴史から考察すると「人間自身は変わらない」と言わざるを得ない。しかし（中略）、コミュニケーション方法や働く環境、知識を得る方法、使用する手法やツール、適用する規律、さらにソフトウェア開発全体における文化は大きく変わっていくだろう。

- 結局「人間」がボトルネック、人が変わることを考慮するとすぐには変わらない
  - 周辺環境・一部の人から少しずつ変わっていく、イノベーター理論/キャズムを参考にするとよい
- （体感）2年くらい離れてもなんとかなった、とはいえ5年は無理か…



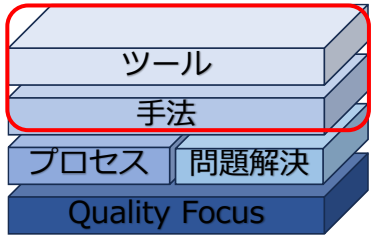
引用：実践ソフトウェアエンジニアリング（第9版）
Roger.S.Pressman他、SEPA翻訳プロジェクト、オーム社、2021年

今までの体験込みでのポイント紹介

・変化を楽しむマインド
- こんな時代に変化を恐れるものでもない、むしろ楽しもう！　タイパより楽しさ重視
- ただし、学んだ内容は正しく理解するようにつとめる、いきなり無理して適用しない

・別にイノベーター的に何でも飛びつかなくてもよい
- ボトルネックは人。人が変わるまで変化は起きない、タイミングは人依存（イノベーター理論）
- 学んでも当たるも八卦、当たらぬも八卦
- 自分にあうものを見つけて、取り込んでいく
- ある程度の時間投資を続けることは大事、読書時間の確保みたいなもの

・楽しく学び続ける環境をつくる
- 学びや試す行為を楽しむことができれば勝利（コミュニティ/検これ等の例を次ページに記載）
- 興味をもつ人が集まることで、新しいツール・技法の学習は楽しくできる
  - 成果を楽しく紹介しながら、他の人の紹介も聞きながら共有していく



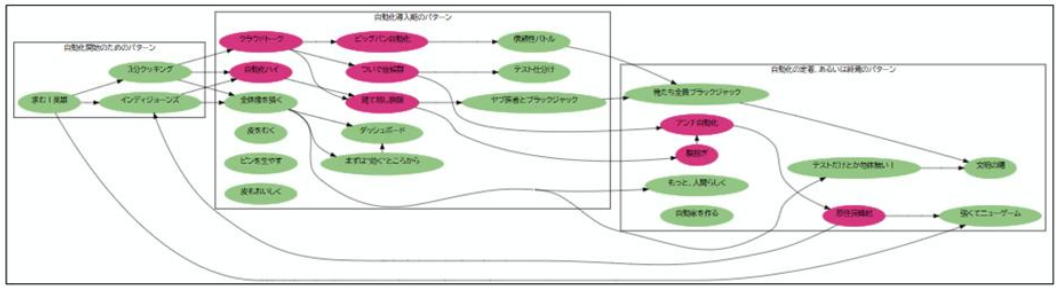
- 楽しく学び続ける環境をつくる　関西コミュニティでの暴走例
  - TABOK勉強会 関西→関西検証コレクション（検これ）→.reviewrc
  - 最初はTABOK（突っ込みどころの多いテスト自動化BOK）学習からスタート
  - テスト自動化パターンの作成、AAAなどのイベント開催
  - JaSST'13 Kansai はコミュニティメンバーに声掛けして発表を実施

TABOK kansai

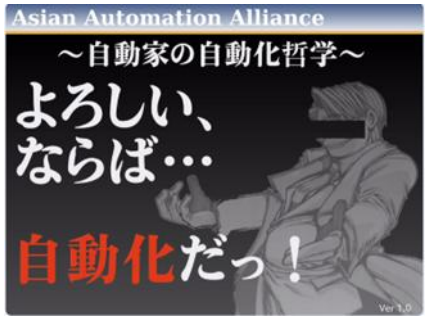
検これ

.REVIEWRC
#reviewrc

テスト自動化パタン言語 オーバービュー



https://kencolle.github.io/AutomationPatternLanguage/index.html



AAA（Asian Automation Alliance）
基調講演者（みうみう氏）の発表

https://www.slideshare.net/slideshow/aaa-all-forautomation/36401662

JaSST'13Kansai発表の一部

セッション　3-B1 ▶概要
「ようこそ、TABOKの世界へ」
前川 博志／井川 将　（TABOK（テスト自動化知識体系）勉強会関西）
Download 講演資料 (PDF ： 1,106KB)

セッション　3-B2 ▶概要
「システムテスト自動化環境を構築してみた！
～組込みシステムでの自動化の考え方～」
水野 昇幸　（WARAI 関西ソフトウェアテスト勉強会実行委員）
Download 講演資料 (PDF ： 1,216KB)

セッション　3-C1 ▶概要
「テスト設計コンテストから現場での実践へ
～実開発でやってみた～」
野中 成夫　（WARAI 関西ソフトウェアテスト勉強会実行委員）
Download 講演資料 (PDF ： 2,348KB)

セッション　3-C2 ▶概要
「Webツールを用いたテスト実施及びテスト管理の取組み（中）
～脱Excel 、、Webアプリ連携による有効活用～」
田中 学二　（デジタル・インフォメーション・テクノロジー）
Download 講演資料 (PDF ： 3,100KB)

セッション　3-D1 ▶概要
「ソフトウェアの品質向上に資する、開発・運用現場の情報管理」
赤羽根 州晴　（島津ビジネスシステムズ）
Download 講演資料 (PDF ： 9,325KB)

セッション　3-D2 ▶概要
「リスク駆動開発で、SPIN初実践とその成功
～チームソフトウェアプロセスとふりかえり～」
徳 隆宏　（オムロン）
山内 美絵子　（オムロンソフトウェア）

楽しく学び続けるマインドを持ち
楽しく学ぶ環境をつくる

■プロセス

・実践ソフトウェアエンジニアリングでは昔からプロセスを土台としている

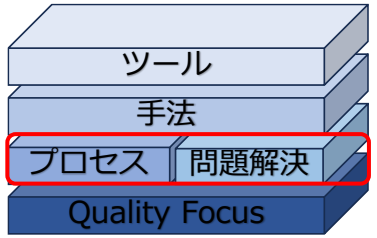
> ソフトウェアエンジニアリングの土台はプロセスである。プロセスは、技術の層を1つにまとめる役割を持ち、ソフトウェアが合理的かつ納期どおり開発できるようにする。

- 少なくとも第4版（2000年）にはその構成となっており、今も変わらない

■問題解決

・ソフトウェアエンジニアリングプラクティスの本質は問題解決（第1章）
- 問題解決を学ぶ際には一般書籍を推奨する

・なぜ必要か、どうすると効果的かを考える
・ツールや手法を適用する際の効果的な動き方（プロセス）を考える
・どうやって変化させるかを考える→活動の価値を伝える



引用：実践ソフトウェアエンジニアリング（第9版）
Roger.S.Pressman他、SEPA翻訳プロジェクト、オーム社、2021年

・プロセスは固定された決まり事ではなく、選択しカスタマイズするもの
- 書籍にいくつものプロセスが記載されるが、特徴を知って選択するものとして紹介されている
- ウォーターフォール VS アジャイルとか意味のない話はするもんじゃない
- （反例）いにしえのク〇デカV字の強制はジツニザンネン…

・新しいツールや手法が登場したらそれに合わせたプロセス調整をすることも大事
-プロセスを自在に選定・コントロールできる能力を身につけることが重要

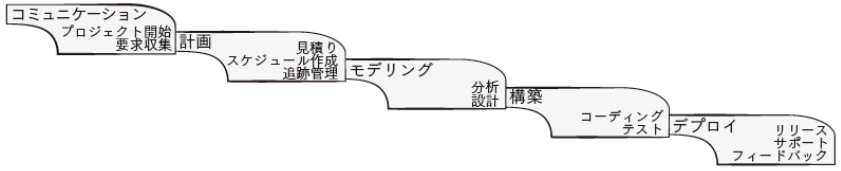


図 2.3　ウォーターフォールモデル

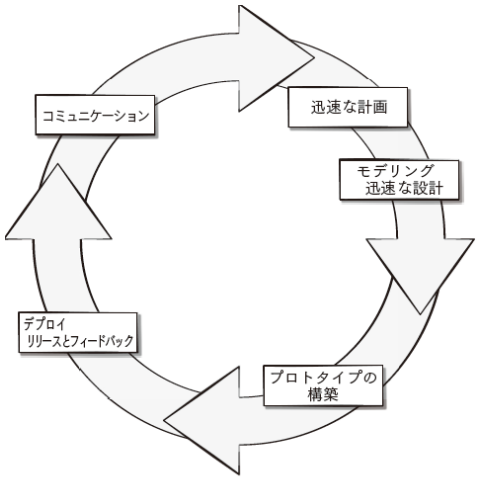


図 2.4　プロトタイピングパラダイム

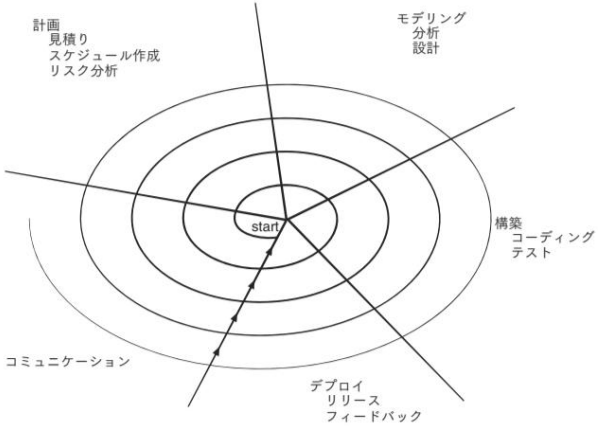


図 2.5　典型的なスパイラルモデル

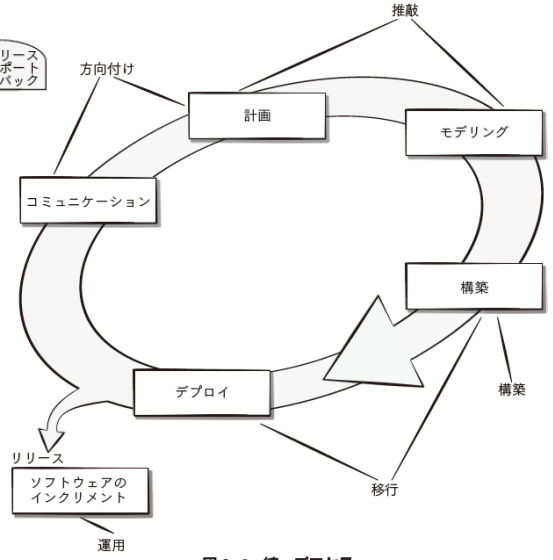


図 2.6　統一プロセス

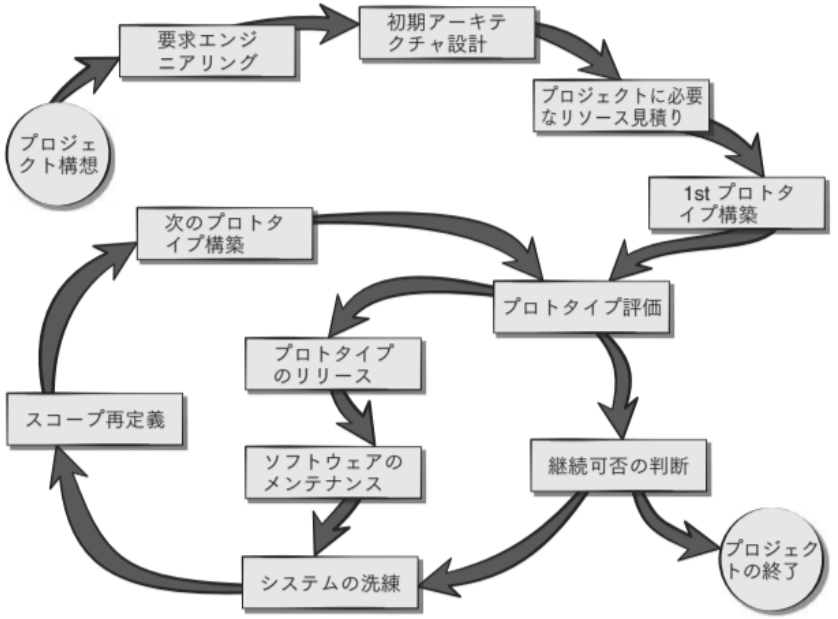


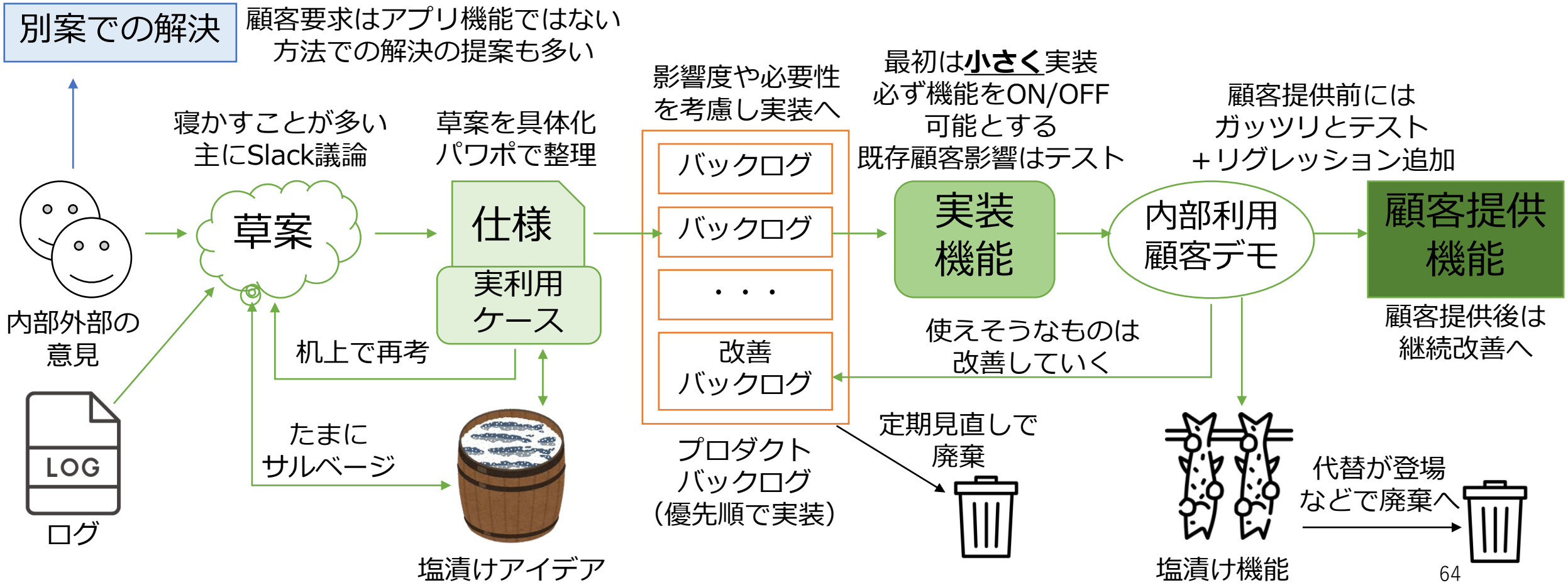
図 4.4　推奨のプロセスモデル

引用：実践ソフトウェアエンジニアリング（第9版）
Roger.S.Pressman他、SEPA翻訳プロジェクト、オーム社、2021年

プロセスを自分で描く力を身に着けることは、多くの領域で役立つ

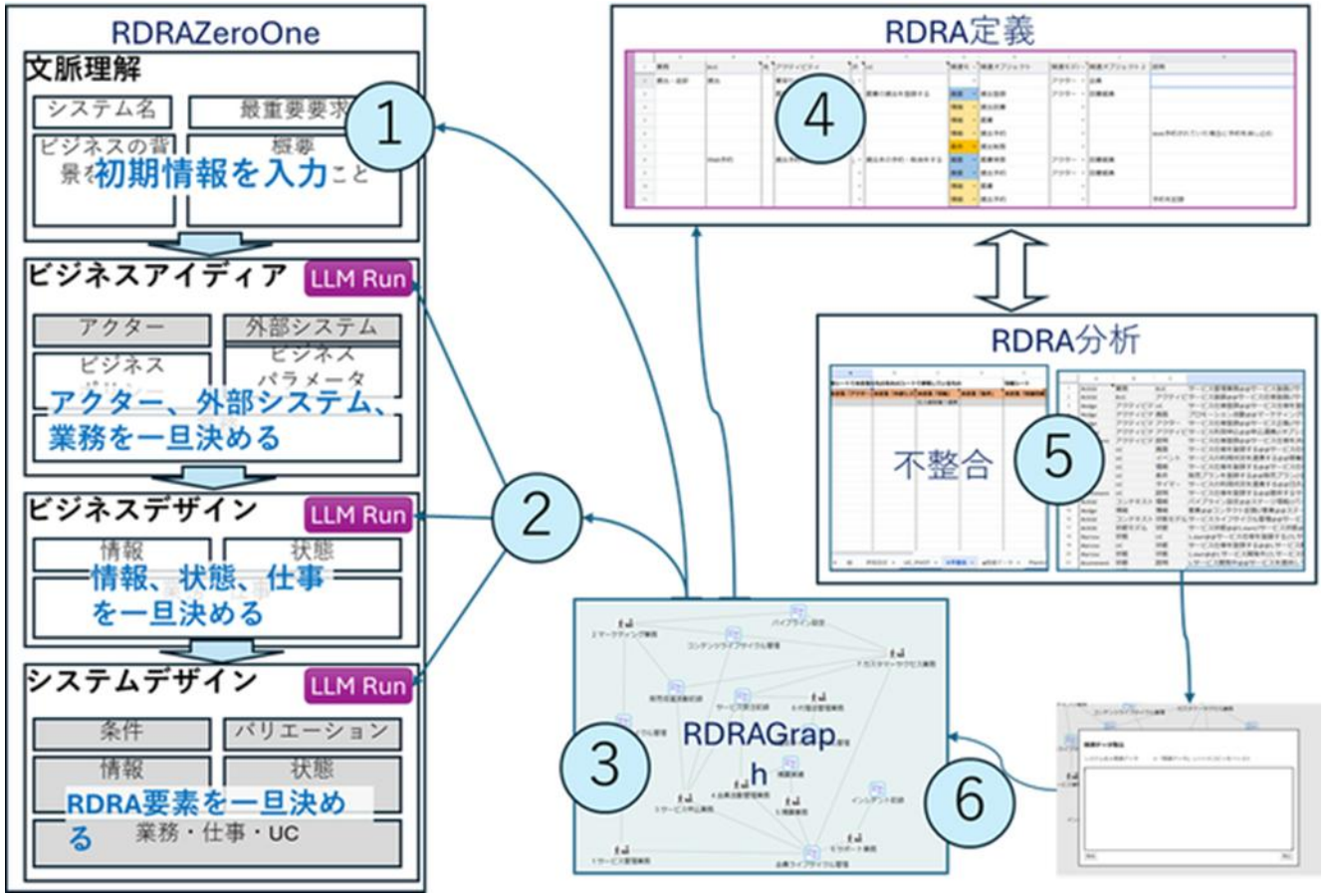
当たりハズレを
多数体験したうえで
パターン化したもの

例：運用中のB2BなSaaSシステムでの「比較的“当たり”が得やすい」新規機能の構築の流れ
→役立つアイデアを蒸留、試しながら提供へと向かう　※塩漬けも多いですが…



プロセスの知識が圧倒的な力になる可能性もある
～RDRAの進化より簡単に紹介～

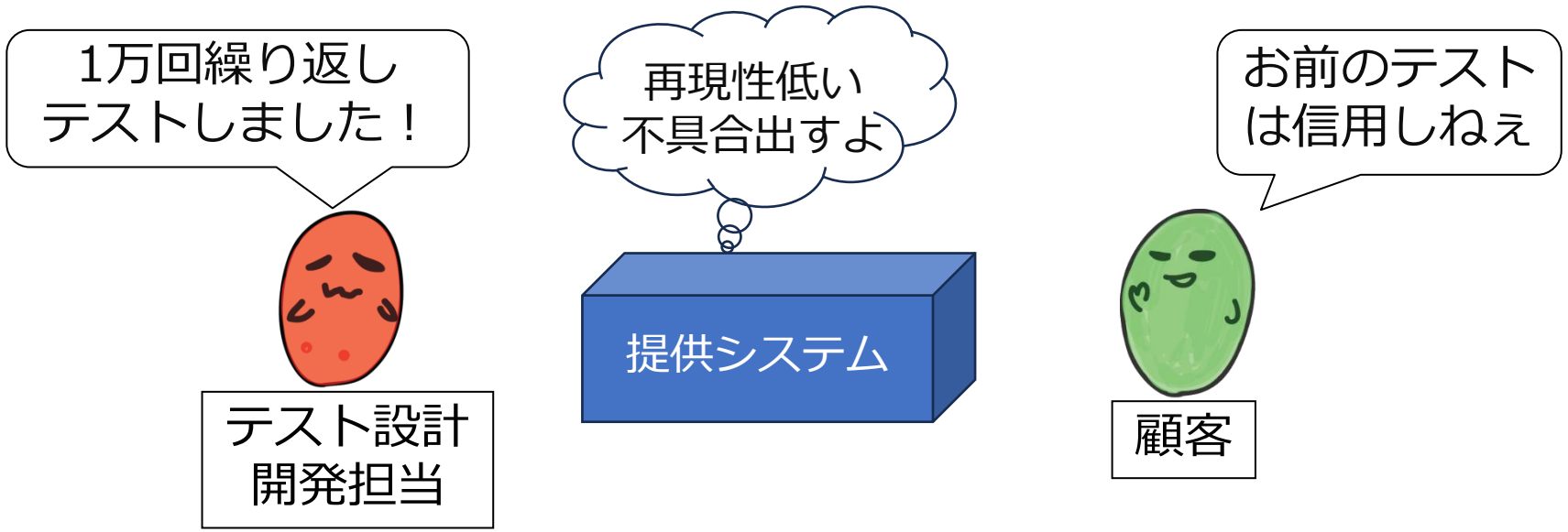
さらに知りたい方は
JaSST北海道へ！



https://rdra-meetup.connpass.com/event/352424/

・問題解決の実績を出せないツールや手法を主張しても信頼が得られない
- 自動化を活用しても、不具合が継続的に発生すれば信用はされない
  - いくら新しいツール・手法を勉強して試しても…ね
- 「（自動で）1万回繰り返しテストしました！」と伝えても…

・適切に問題解決ができない場合、ツールや手法まで疑われてしまうことすらある…
- 継続的デグレ状況で「UIテスト自動化をしている」と主張する開発組織
- UIテスト自動化自体の信頼も低下、敬遠される状況にもなる



問題解決力、考え方も学ぶことができる時代

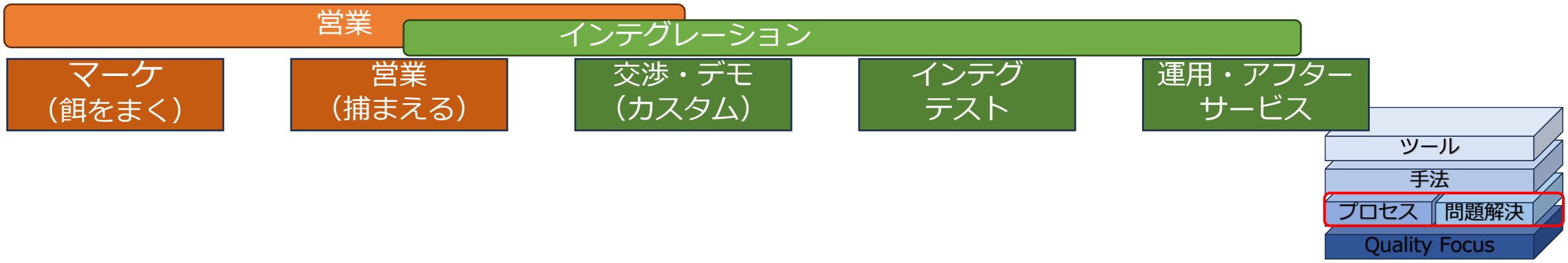
- XX思考、XXシンキングは問題解決の方法を教えている
  - 世の中の書籍を漁ればいくらでもよい本はある、新しいほどわかりやすいかも
- 専門性と組み合わせると圧倒的に強い（役立っている）

役割/専門性ベース or 問題解決ベース

- 特定の専門性に賭けるのはある意味ギャンブル
  - 特定の（習得に時間のかかる/人がいない）技術分野で需要の強いものは高価値とみなされる
  - 専門性が強い分野ではセクショナリズム的なふるまいも許容される（XXの仕事ではない発言）
  - インフラの構築、性能チューニングあたりは専門性だけでしばらくいける
  - とはいえ、時代の変化で価値が下がる可能性も。プログラム専門の人は今後の価値は下がるかと
- 専門性が弱い/低下した分野は組み合わせで価値を出す
  - テスト設計やテスト自動化はまあ習得しやすい範囲なのでそれだけではイマイチ
    - 生成AI側の支援を受けやすい、置き換わる可能性もある分野
  - 該当の専門性に加えて、組み合わせで価値を出すこともできる（幅が広がる）
    例：ドメイン知識のスペシャリスト、運用・保守、サービスのインテグレーション
- 問題を捉えて解決する活動を続けていれば仕事はなくならない

運用しているシステムに対しては提供先側の問題から解決していくのが王道
※わかりやすくリソースが確保できる、効果が明確

企画・要件　設計・開発　テスト　運用



学べ、しかして考えよ

※考え方までも学ぶことができる時代

ソフトウェアエンジニアリングの階層：Quality Focus

・実践ソフトウェアエンジニアリングからの引用

> すべてのアプローチは品質に対する組織的な責任をもって実施されなければならない
> …プロセスを常に改善し続ける…文化こそが、
> 究極的にソフトウェアエンジニアリングアプローチをさらに効果的にする。
> これがソフトウェアエンジニアリングを支える基盤となる

・Quality Focusの翻訳には以下の言葉が使われている
- 品質に焦点をあてる
- 品質中心の文化（第6版で翻訳に利用、某氏の訳だろうと予想）

・基盤ができていないと結局は価値を出すことができない
- ツールや手法の知識を有していても、現場で価値を出せない人がいる

※注：高度なレベルの話ではなく、ビジネスとして当たり前品質な話

引用：実践ソフトウェアエンジニアリング（第9版）
Roger.S.Pressman他、SEPA翻訳プロジェクト、オーム社、2021年

ソフトウェアエンジニアリングの階層：Quality Focus

・Quality Focusがないと、運用されるシステムの提供価値へはつながりづらい
- 普段の言動からにじみ出てしまうもの
- どうやら、なんとも思わない人もいるらしい…
  - 再現しない不具合には何も手を出さない、といったケースも存在した

・Quality Focusは組織に伝搬する　～そして「品質中心の文化」へ～
- 組織の文化から自然と身に付くケースも多い
- リーダーの一貫的な言動がQuality Focusを伝搬させることにつながる
- チーム構成・スキル、評価、チームの目標などの設定が組織文化につながる
- 特にカネ、イノチにかかわる組織は重過ぎるレベルで厳しい

・感性を高めることも大事
- シンプルに「**カッコ悪い**」と感じる心があればよい　※カッコ悪いとは何か？の感性を高めることも大事
  - 「UIの好み」みたいな話ではなく、実施している行動やプロダクトのあるべき姿を一考する
- 「？」の発生や心がざわめいたら、考えることができる人でありたいと願う
  - いわば「タウマゼイン」ともいえる（チ。-地球の運動について　参考）
- 「ソフトウェアエンジニアリング倫理綱領および専門職のプラクティス」も参考
- 真面目に仕事に取り組んでいれば自然と身につく、当たり前のものと思ってます

参考：リッツ・カールトンの採用基準

1. 職業倫理	**責任感、物事を遂行する能力があること**
2. 自尊心	**良くなろうとする願望を持っていること**
3. 説得力	ワンランク上の物を売る力があること
4. 関係拡大能力	顧客に受け入れられたいという願望を持っていること
5. チームワーク	仲良くしてお互いに協力する資質があること
6. 積極性	お客様に対して進んで働きかける、楽天的で明るい性格であること
7. サービス	人のために尽くすのが好き、人のことを第一に考える性格であること
8. 共感性	相手の感情を読み取る。顧客の願望を察知する能力があること
9. 気配り	顧客のために一歩進んだ何かをする気性を持っていること
10. 正確さ	時間を守る。タイムリーに物事を処理する心がけがあること
11. 向上心	知識力、解決策について革新的な案を生み出す力があること

参考：ソフトウェアエンジニアリング倫理綱領および専門職のプラクティス
（ACMとIEEE CS 協同によるタスクフォースが策定）

1. 公共性	ソフトウェアエンジニアは公益に沿う行動をとること
2. 顧客と雇用者	ソフトウェアエンジニアは顧客と雇用者、公共へ最大の利益をもたらすように行動すること
3. 製品	ソフトウェアエンジニアの開発する製品とその変更が、プロフェッショナルとして高い基準を満たしていること
4. 判断	ソフトウェアエンジニアはプロフェッショナルとしての判断の際に、整合性並びに独立性を保つこと
5. マネジメント	ソフトウェアエンジニアリングのマネージャとリーダーは、ソフトウェア開発とメンテナンスにおいて、理にかなったマネジメントの実施を誓うこと
6. 専門職	ソフトウェアエンジニアは専門職としての誠実さと評判を高めると共に、公益に努めること
7. 同僚	ソフトウェアエンジニアは同僚に対して公平かつ協力的であること
8. 自己	ソフトウェアエンジニアは生涯を通じて学習を続け、理にかなった専門職の遂行をすること

引用：実践ソフトウェアエンジニアリング（第9版）
Roger.S.Pressman他、SEPA翻訳プロジェクト、オーム社、2021年

「？」を感じ取ろう
（カッコ悪くないか問おう）

※？≒タウマゼイン/心のざわめき

まとめ

変化する開発
・世の中は変わるもの、開発現場は業界ごと変わり続けている
・大抵はビジネスや現場のペインを解決した、良い方向へ進んでいる

進化する体系
・体系が出たタイミングにおける知識を手早くまとめて得るために便利
・ファッション的要素も多いが、プロセスなど変わらない要素もある

変わり続ける現場で生き抜く→時代に適応する
・ツール、手法、プロセス・問題解決、Quality Focusでそれぞれ考えた
・楽しく学び続けるマインドを持ち、楽しく学ぶ環境をつくる
・学べ、しかして考えよ
・「？」を感じ取ろう（カッコ悪くないか問おう）