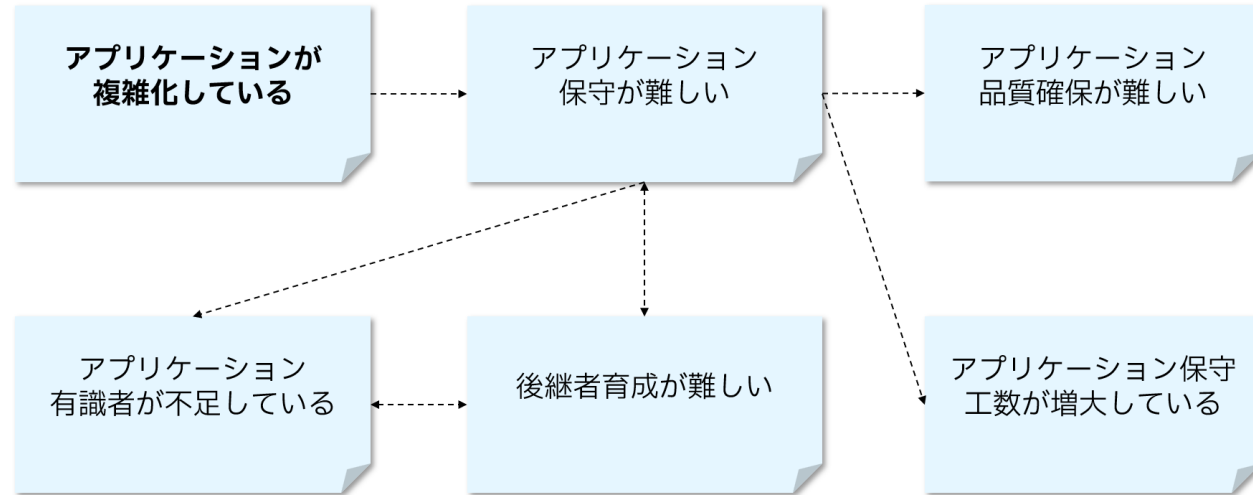


続・COBOLの課題を改めて考える

昨年の振り返り

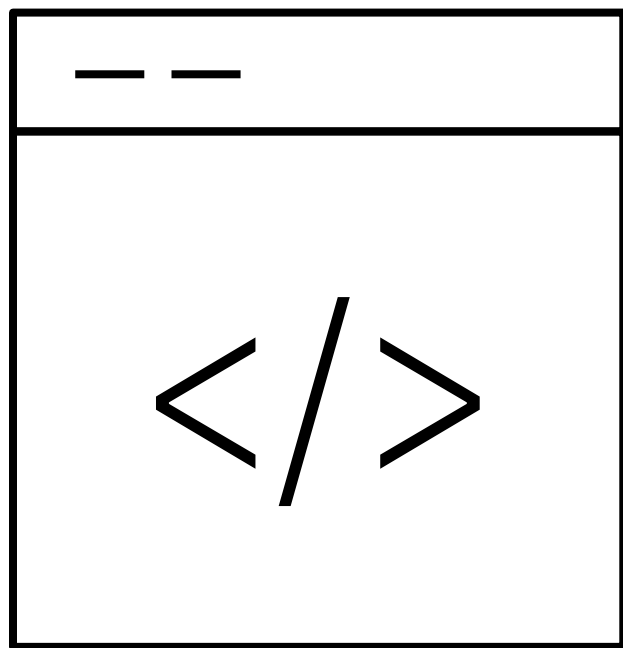
レガシーシステムの抱える課題と現状の相関



まとめ

- COBOL の現状課題解決（＝モダナイゼーション）には様々なアプローチが考えられるが、将来同じことを繰り返してしまうことを避けるために防止策も検討
- モダナイゼーションは一過性の活動ではなく継続的な活動（＝防止策）
- アプリケーション複雑化と技術者不足を同時に解決する重要なカギが**単体テスト**

COBOL アプリケーション開発向けの生成 AI 活用例



説明

生成

●プログラム仕様書

- _____
- _____
- _____
- _____

生成 AI に COBOL のコード説明をお願いしてみた

```
83      MAIN-RTN                                SECTION.  
84      IF TRN-KOKYAKU-BANGO = MST-KOKYAKU-BANGO  
85      THEN  
86          MOVE UKETSUKE-BANGO OF TRN-RECORD  
87          TO   UKETSUKE-BANGO OF EDIT-REC  
88          MOVE SHOHIN-BANGO OF TRN-RECORD  
89          TO   SHOHIN-BANGO OF EDIT-REC  
90          MOVE KONYU-SURYO OF TRN-RECORD  
91          TO   KONYU-SURYO OF EDIT-REC  
92          MOVE KOKYAKU-BANGO OF TRN-RECORD  
93          TO   KOKYAKU-BANGO OF EDIT-REC  
94          MOVE KOKYAKU-YUBINBANGO OF MST-RECORD  
95          TO   KOKYAKU-YUBINBANGO OF EDIT-REC  
96          WRITE OUT-REC FROM          EDIT-REC  
97          PERFORM TRNFILE-READ  
98      ELSE  
99          IF TRN-KOKYAKU-BANGO > MST-KOKYAKU-BANGO  
100         THEN  
101             PERFORM MSTFILE-READ  
102         ELSE  
103             DISPLAY 'エラー！KOKYAKU-BANGOがMSTFILEになし'
```

生成 AI に COBOL のコード説明をお願いしてみた

このCOBOLコードは、顧客番号（KOKYAKU-BANGO）をキーにして、トランザクションファイル（TRN）とマスタファイル（MST）を照合し、編集レコード（EDIT-REC）を作成・出力する処理です。

主な処理の流れ

1. 顧客番号の一致判定

`TRN-KOKYAKU-BANGO = MST-KOKYAKU-BANGO` の場合、

- トランザクションレコード（TRN-RECORD）から受付番号・商品番号・購入数量・顧客番号を編集レコード（EDIT-REC）にコピーします。
- マスタレコード（MST-RECORD）から顧客郵便番号も編集レコードにコピーします。
- 編集レコードを出力レコード（OUT-REC）として書き込みます。
- 次のトランザクションレコードを読み込みます（`PERFORM TRNFILE-READ`）。

2. 顧客番号の不一致判定

`TRN-KOKYAKU-BANGO > MST-KOKYAKU-BANGO` の場合、

マスタファイルの次レコードを読み込みます（`PERFORM MSTFILE-READ`）。

- それ以外（`TRN-KOKYAKU-BANGO < MST-KOKYAKU-BANGO`）の場合、エラーメッセージを表示し、次のトランザクションレコードを読み込みます。

注意点「落とし穴」

- 顧客番号の大小比較でファイルの進め方を制御しているため、両ファイルは顧客番号でソートされている必要があります。
- エラー時のメッセージは日本語で表示されますが、処理自体はトランザクションファイルの次レコードを読むだけです。
- `EXIT.` でセクションの終了を明示しています。

例

例えば、TRNファイルの顧客番号が1001、MSTファイルの顧客番号が1001の場合、編集レコードに必要な情報をコピーして出力します。
もしTRNファイルの顧客番号が1002、MSTファイルの顧客番号が1001なら、MSTファイルを進めて一致する顧客番号を探します。

改善案

- エラー時に処理を分岐させる場合、ログ出力やエラー件数のカウントなども追加すると運用面で便利です。
- ファイルの終端判定や例外処理も追加すると、より堅牢なプログラムになります。

ご不明点があれば、さらに詳しく解説できます。

コード説明やコード生成のお役立ち度

- **コード説明**

- COBOL になじみのないエンジニアや初心者にとっては有用
- 有識者やベテランにとってはコードレビューやコードの読み飛ばしに有用

- **コード生成**

- COBOL アプリケーションが稼働するシステムは保守が中心であり、新規に大量にコードを書く機会は多くないが、それでも有用か？
- テストコードをたくさん書く（＝テスト自動化に取り組む）ときには非常に有用

- **お役立ち度**

- 従来はヒトがやらなければならなかった開発作業のごく一部を生成 AI が支援するというものであり、開発全体の生産性向上によるコスト削減という目に見えるほどの[定量的な効果](#)を得るには至らない
- ただし、技術者の世代交代促進に寄与という[定性的な効果](#)はありそう

別のコードで生成 AI に COBOL のコード説明をお願いしてみた

```
83      主処理                      SECTION.  
84      IF   トラン顧客番号 = マスター顧客番号  
85      THEN  
86          MOVE  受付番号 0F   トランレコード  
87              TO   受付番号 0F   編集レコード  
88          MOVE  商品番号 0F   トランレコード  
89              TO   商品番号 0F   編集レコード  
90          MOVE  購入数量 0F   トランレコード  
91              TO   購入数量 0F   編集レコード  
92          MOVE  顧客番号 0F   トランレコード  
93              TO   顧客番号 0F   編集レコード  
94          MOVE  顧客郵便番号 0F   マスターレコード  
95              TO   顧客郵便番号 0F   編集レコード  
96          WRITE  出力レコード FROM 編集レコード  
97          PERFORM   トランファイル読み込み  
98      ELSE  
99          IF   トラン顧客番号 > マスター顧客番号  
100         THEN  
101             PERFORM   マスターファイル読み込み  
102         ELSE  
103             DISPLAY  'エラー！顧客番号がマスターファイルになし'  
104         PERFORM   トランファイル読み込み
```

別のコードで生成 AI に COBOL のコード説明をお願いしてみた

このCOBOLコードは「主処理」セクションで、トランザクションファイル（トランレコード）とマスターファイル（マスターレコード）の顧客番号を比較し、条件に応じて処理を分岐しています。

主な処理の流れ

1. 顧客番号の一致判定

`IF` トラン顧客番号 = マスター顧客番号

- 顧客番号が一致した場合、トランレコードとマスターレコードの情報を編集レコードに移し替えます。
- 編集レコードを出力レコードとして書き込みます（`WRITE`）。
- 次のトランファイルの読み込みを行います（`PERFORM` トランファイル読み込み）。

2. 顧客番号の不一致判定

- `ELSE` で分岐し、さらに `IF` トラン顧客番号 > マスター顧客番号 で判定します。
 - トラン顧客番号がマスター顧客番号より大きい場合は、マスターファイルの次レコードを読み込みます（`PERFORM` マスターファイル読み込み）。
 - それ以外（小さい場合）は、エラーメッセージを表示し、次のトランファイルの読み込みを行います。

ポイント・注意点

・編集レコードへの値の移動

`MOVE` 文で各項目を編集レコードにセットしています。

例:

```
MOVE 受付番号 OF トランレコード TO 受付番号 OF 編集レコード
```

・ファイル読み込みのタイミング

顧客番号の一致・不一致に応じて、どちらのファイルを読み込むかが変わります。

・エラーハンドリング

マスターファイルに顧客番号が存在しない場合、`DISPLAY` でエラーを表示します。

例

- 顧客番号が一致 → 編集レコード作成 & 出力、次のトランレコード読み込み
- トラン顧客番号が大きい → 次のマスターレコード読み込み
- トラン顧客番号が小さい → エラー表示、次のトランレコード読み込み

よくある落とし穴

- 顧客番号の比較ロジックが正しくないと、データの突合せに失敗します。
- ファイル読み込みの順序が重要です。間違えると無限ループやデータ漏れの原因になります。

他に知りたい部分があれば教えてください。

1 番目のコードと 2 番目のコードを比較してみる

```
MAIN-RTN                                SECTION.  
IF TRN-KOKYAKU-BANGO = MST-KOKYAKU-BANGO  
THEN  
    MOVE UKETSUKE-BANGO OF TRN-RECORD  
    TO UKETSUKE-BANGO OF EDIT-REC  
    MOVE SHOHIN-BANGO OF TRN-RECORD  
    TO SHOHIN-BANGO OF EDIT-REC  
    MOVE KONYU-SURYO OF TRN-RECORD  
    TO KONYU-SURYO OF EDIT-REC  
    MOVE KOKYAKU-BANGO OF TRN-RECORD  
    TO KOKYAKU-BANGO OF EDIT-REC  
    MOVE KOKYAKU-YUBINBANGO OF MST-RECORD  
    TO KOKYAKU-YUBINBANGO OF EDIT-REC  
    WRITE OUT-REC FROM EDIT-REC  
    PERFORM TRNFILE-READ  
ELSE  
    IF TRN-KOKYAKU-BANGO > MST-KOKYAKU-BANGO  
    THEN  
        PERFORM MSTFILE-READ  
    ELSE  
        DISPLAY 'エラー！KOKYAKU-BANGOがMSTFILEになし'
```

```
主処理                                SECTION.  
IF トラン顧客番号 = マスター顧客番号  
THEN  
    MOVE 受付番号 OF トランレコード  
    TO 受付番号 OF 編集レコード  
    MOVE 商品番号 OF トランレコード  
    TO 商品番号 OF 編集レコード  
    MOVE 購入数量 OF トランレコード  
    TO 購入数量 OF 編集レコード  
    MOVE 顧客番号 OF トランレコード  
    TO 顧客番号 OF 編集レコード  
    MOVE 顧客郵便番号 OF マスターレコード  
    TO 顧客郵便番号 OF 編集レコード  
    WRITE 出力レコード FROM 編集レコード  
    PERFORM トランファイル読み込み  
ELSE  
    IF トラン顧客番号 > マスター顧客番号  
    THEN  
        PERFORM マスターファイル読み込み  
    ELSE  
        DISPLAY 'エラー！顧客番号がマスターファイルになし'  
        PERFORM トランファイル読み込み
```

COBOL 最強のリファクタリング

プログラムで使用している半角シンボルを日本語で置き換える

COBOLは平易

最低限知っておくべき
命令はごく限定的

アルゴリズムは決まり
きった数パターンのみ

変数が日本語になれば
すらすら読める？

Java や Python のように業界全体にまたがる命名規則（クラス名はパスカルケース、メソッド名はキャメルケースなど）があればよいが、COBOLにはない

すらすら読めない COBOL は読めるようになるまでリファクタリング

ネストが深すぎる

結合度の高いプログラ
ム間インターフェース

過去の変更履歴が残存

すらすら読めるコードがあるならプログラム仕様書は不要なのでは？

必要なドキュメントもある

- “保守“だからこそ必要なドキュメント

- 案件対応の概要、対応方針、検討経緯、背景

✓ 例. オンライン会議用のソフトウェアに作ってもらった議事録を保管しておく あ、生成 AI だ（笑）

- 業務的なカタマリの単位での処理概要説明

- データと処理の流れの全体像を分かりやすく

- ✓ オンライン・トランザクション
- ✓ バッチ処理全体で重要なファイル（多数の処理に参照されるファイル）作成処理とその前提条件
- ✓ データベース更新処理とその前提条件
- ✓ 帳票や外部システムと受け渡すファイル作成処理とその前提条件 など

COBOL モダナイゼーションは開発プロセスの見直しから

- **開発プロセスの見直しとは**

- 工程完了基準や工程そのものの見直し
- ツールの見直し→生成 AI 活用のインターフェースには統合開発環境が当たり前になっています
- ドキュメントのファイル形式の見直し→用途の広いファイル形式 md、YAML、JSON

- **現在の開発プロセスは適切に見直されていますか？**

- 新規開発で必要なドキュメント ≠ 保守で必要なドキュメント
- 新規開発当時からの慣習で当たり前に無駄な作業をしていませんか？

- **生成 AI 活用が目的になっていませんか？**

- プログラム仕様書作成を目的とした活用 vs プログラムの理解を助けることを目的とした活用
- プログラム仕様書作成自体が不要というプロセス見直しが実現できればそもそも検討不要か？

COBOL 開発で生成 AI に期待すること

要件を与えたら
修正箇所を特定
してくれる

設計情報を与え
たらコード修正
してくれる

コードから設計
情報を生成して
くれる

コードの妥当性、
性能が悪いコー
ドや脆弱性を指
摘してくれる

最適なテスト
ケースを生成し
てくれる

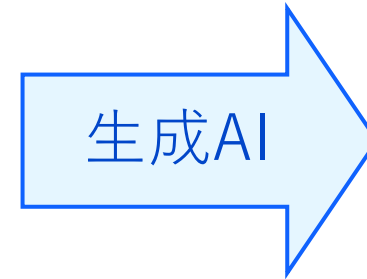
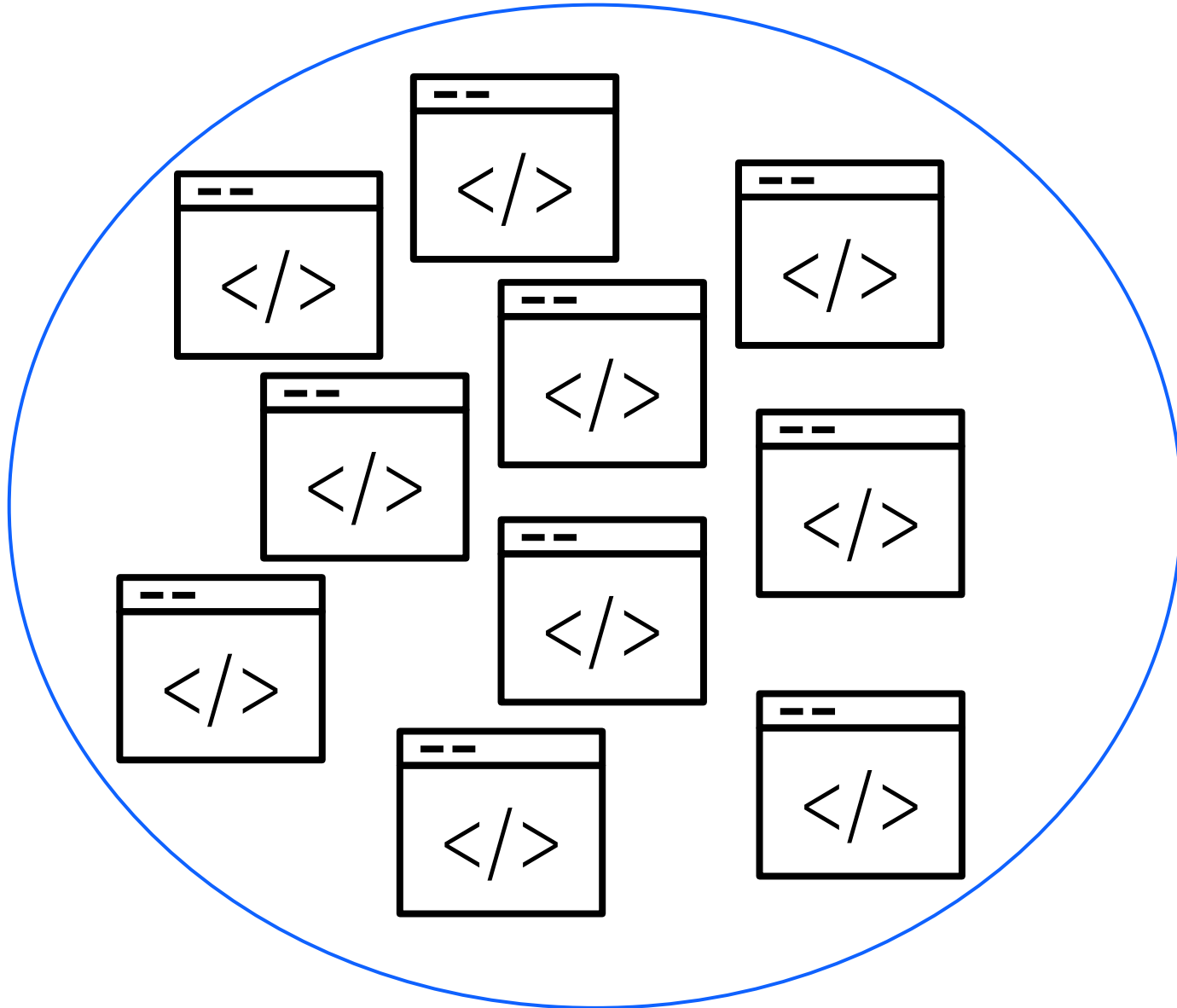
テストケースに
応じたテスト
データを生成し
てくれる

不具合の原因を
調査してくれる

不具合を修正し
てくれる

障害報告書を作
成してくれる

まずはコードからシステム全体のことを知りたい

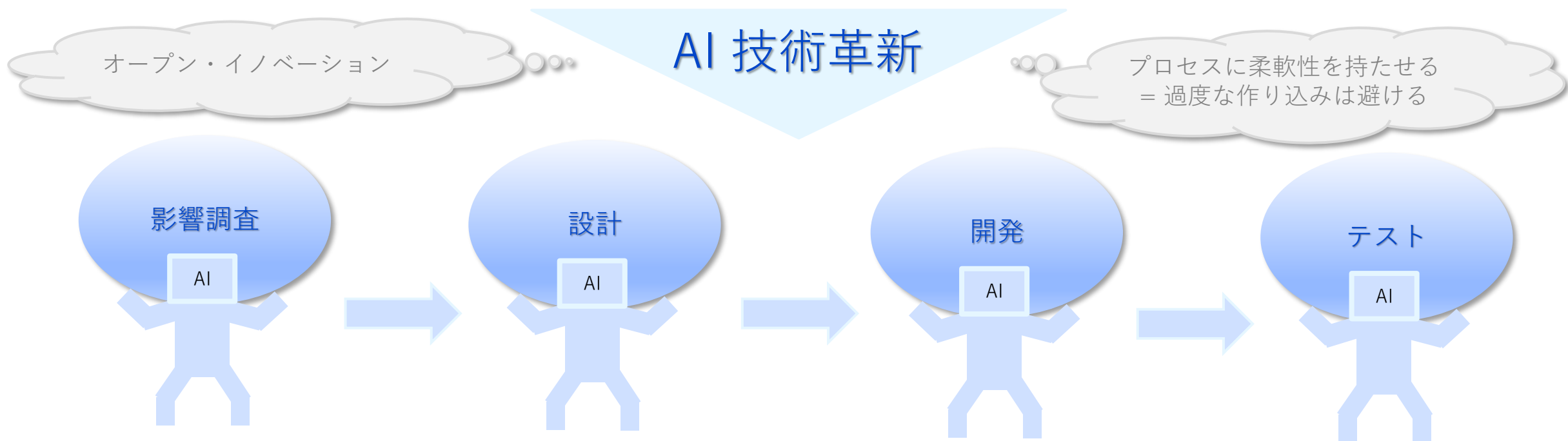
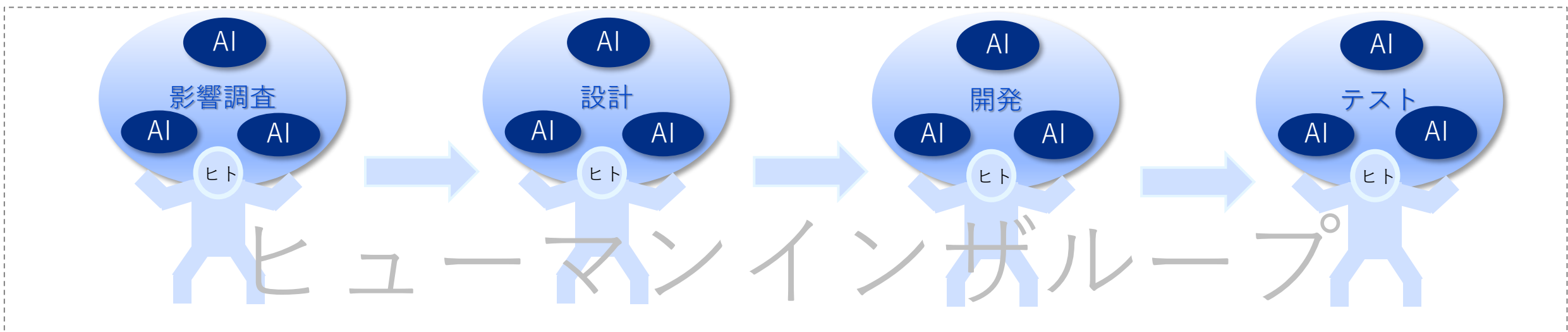


究極の読み飛ばし

AI の特性は「答えを確率的に返す」

	従来型プログラム	AI (基盤モデル)
	1010→ ←0101	
関数特性	$y = f(x)$ xの入力に対して「一意に」yを返す	$y = p(x)$ xの入力に対して「確率的に」yを返す
機能処理の作り方	機能処理をプログラミング 内部処理はホワイトボックス	機能処理を教師データで生成 内部処理はブラックボックス
品質の不変性	要件に対して不変 要件を満たさないものはバグ	教師データの品質や時間変化により 品質が変化
保守性	要件が変わる場合に変更	要件が変わらなくても、教師データや モデルが変わる場合に変更
求められるスキル	プログラムスキル	プロンプト・エンジニアリング モデル選定と評価

COBOL 開発の生成 AI 活用はどうなる？



まとめ

- COBOL モダナイゼーションは開発プロセスの見直しに優先的に着手
 - ✓ 生成 AI ありきではなく、プロセスを見直した結果として必要なものへの適用を検討
- 生成 AI 技術の進化スピードはとてつもなく早いことを念頭に
 - ✓ オープン・イノベーションは COBOL 開発以外の世界から起こる
 - ✓ 将来のあらゆる技術に柔軟に対応できるように過度な作り込みは避ける
活用したい技術の足枷にならないために
- 現時点で、COBOL 開発に関しては、生成 AI とは適度な距離感を保って、
つかずはなれず接していくのがよいかなあ、と思っています