SPI Japan 2013 in 東京

Software Product Line の実践
~ テスト資産の構築 ~

住友電工情報システム株式会社
QCD改善推進部
品質改善推進グループ
服　部　悦　子

2013.10.17

目次

- 1. テスト資産構築に至る 背景
- 2. テスト資産の構築 ～自動テストの実現～
- 3. 結果と評価

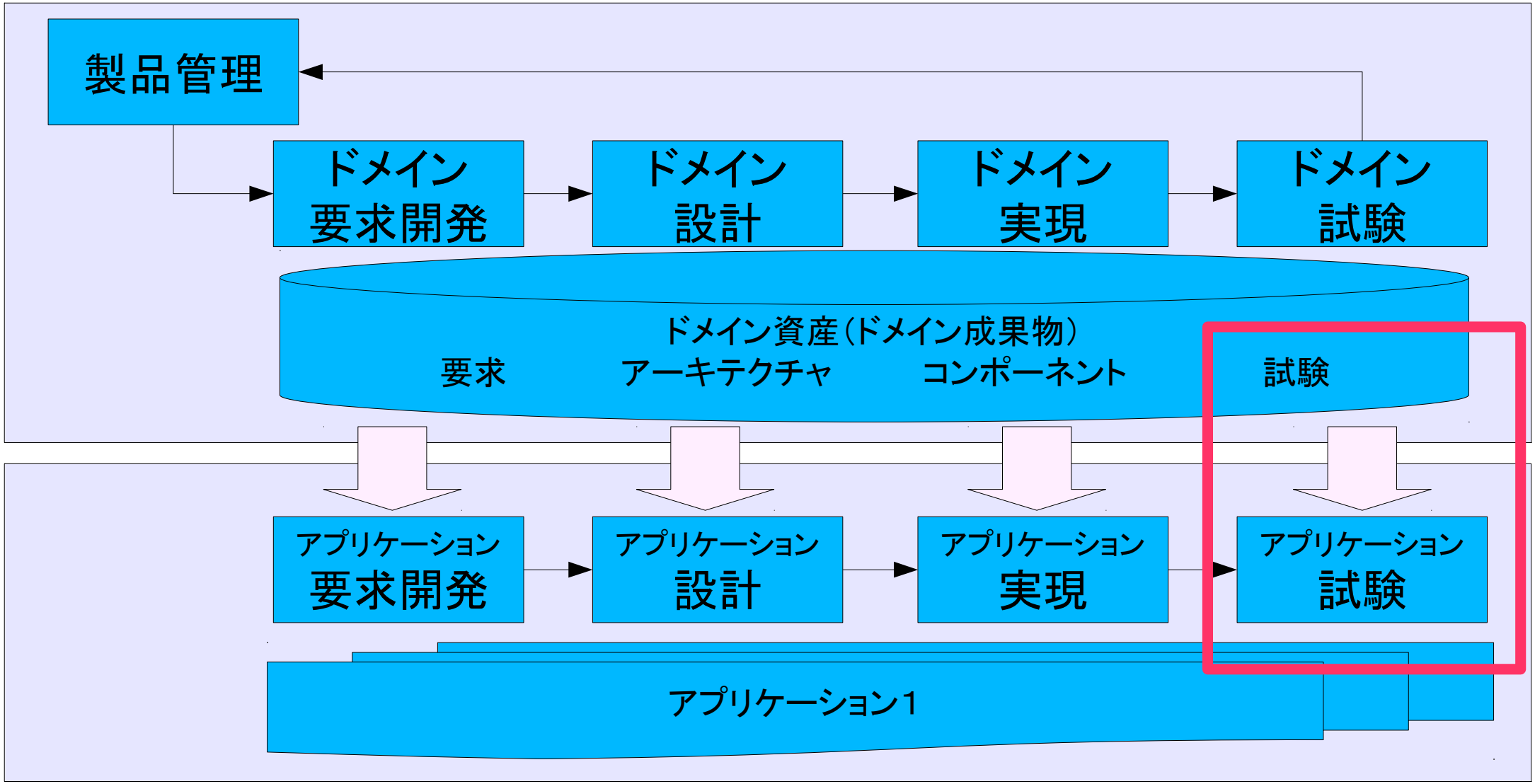
テスト資産構築に至る

背　景

背景

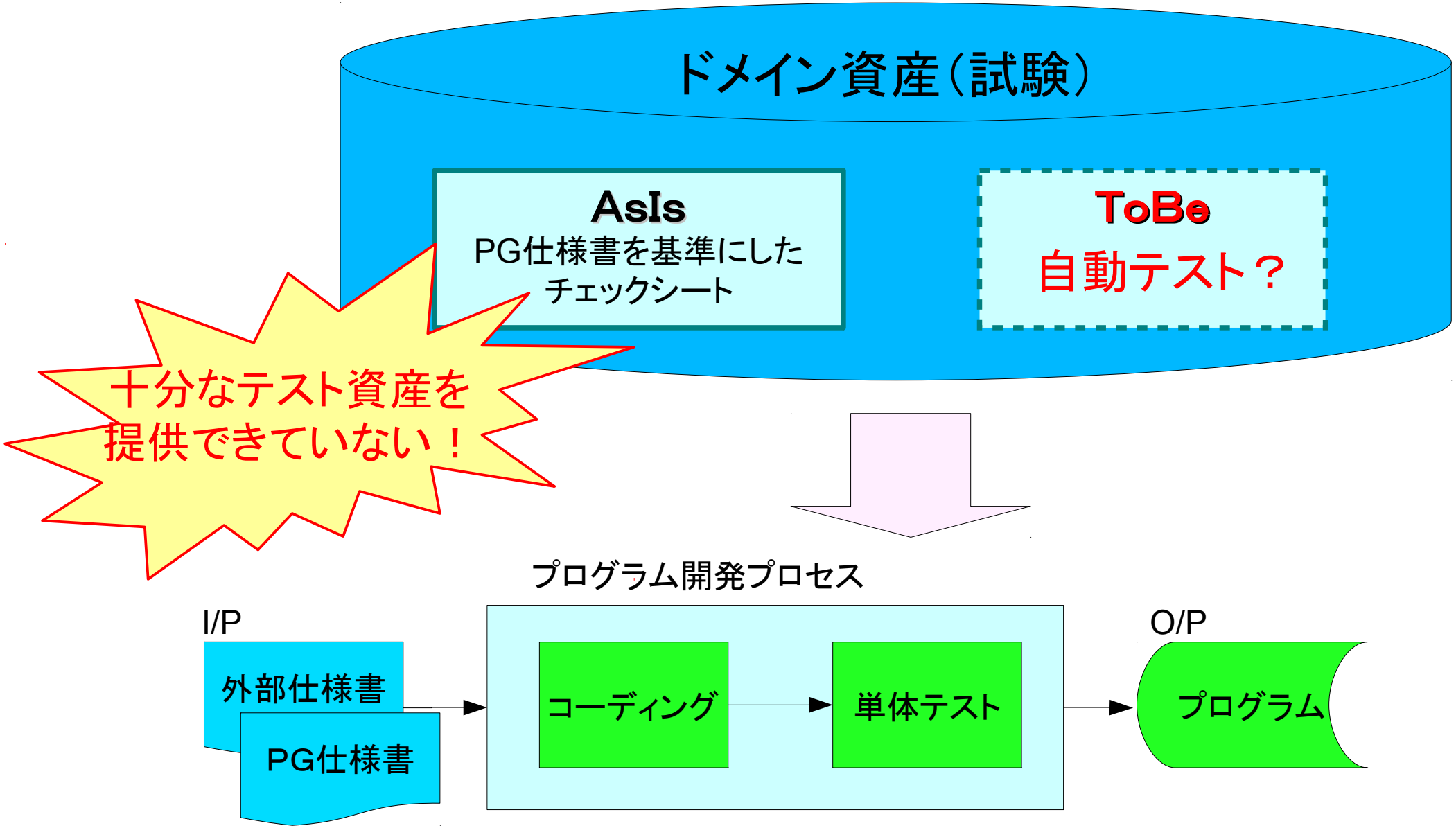
- 品質改善活動
  →継続して行っている
- 生産性向上活動(特にコーディング生産性)
  →開発フレームワークのバージョンアップ
- ソフトウェアプロダクトライン
  ＜テスト資産の提供＞
  →何を提供できるか？
  何を提供できる可能性があるか？

『ソフトウェアプロダクトラインエンジニアリング』より



引用元:『ソフトウェアプロダクトラインエンジニアリング ― ソフトウェア製品系列開発の基礎と概念から技法まで』
クラウス・ポール (著), ギュンター・ベックレ (著), フランク・ヴァン・デル・リンデン (著), 林 好一 (翻訳), 吉村 健太郎 (翻訳), 今関 剛 (翻訳)

単体テストの問題


ドメイン資産（試験）
AsIs
PG仕様書を基準にした
チェックシート
ToBe
自動テスト？
十分なテスト資産を
提供できていない！
プログラム開発プロセス
I/P
外部仕様書
PG仕様書
コーディング
単体テスト
O/P
プログラム

テストに関するあるべき姿

自動テスト

デグレを防止できる
QCD

ToBe
実現すべき仕様が
テストとして実装されている

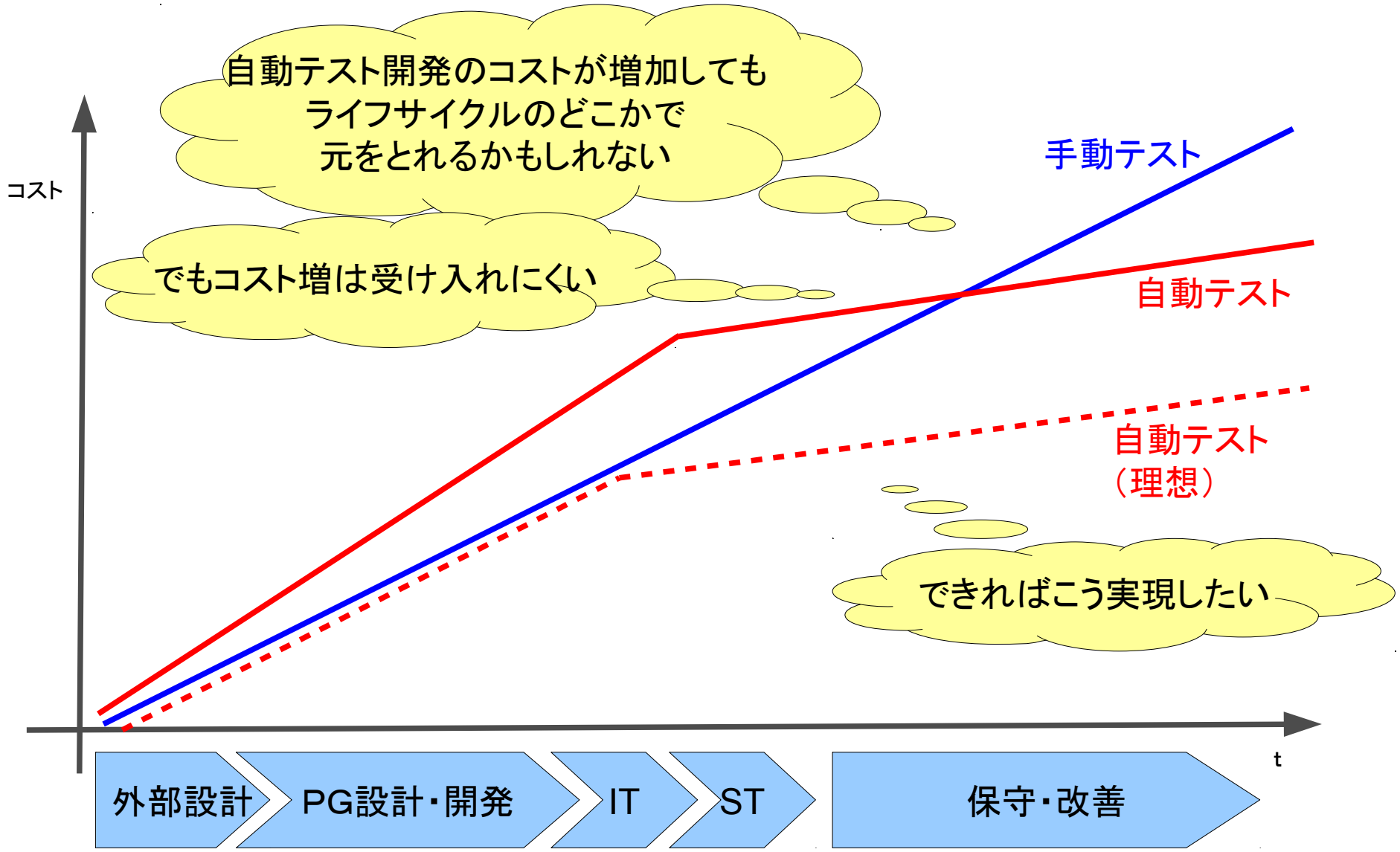
AsIs
実現すべき仕様を
全て理解できないので
関係しそうな部分だけをテストする

IT・ST、保守コストを
低減できる
QCD

ToBe
テストが実装されているので
実行するだけでよい

AsIs
テストをする為に
テストデータを用意しないといけない
テストの実行手順を
理解しないといけない

自動テスト導入の課題 ～何故今までできなかった？～



目指す姿と課題

目指す姿

全プロジェクトが自動テストを開発し保守業務が改善される

保守業務の改善
○開発者
　修正時テスト工数減
　デグレ検出率向上
○利用者
　トラブル減
　保守コスト減

従来のコストで開発したい

課題1．テストプログラム開発工数の捻出

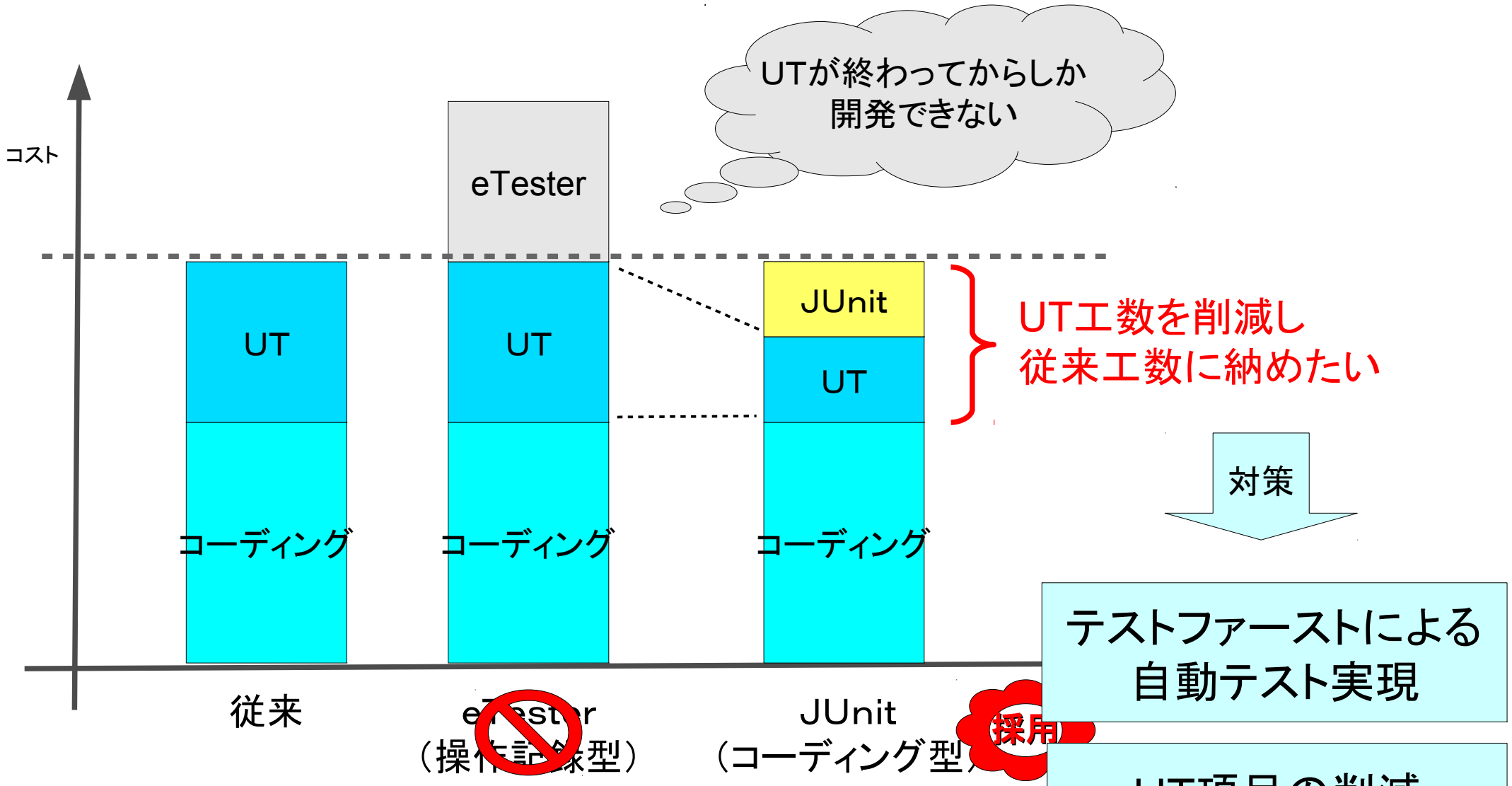
課題2．教育コストを抑えたい

全プログラマーが自動テストを開発できる

テスト資産の構築

～自動テストの実現に向けて～

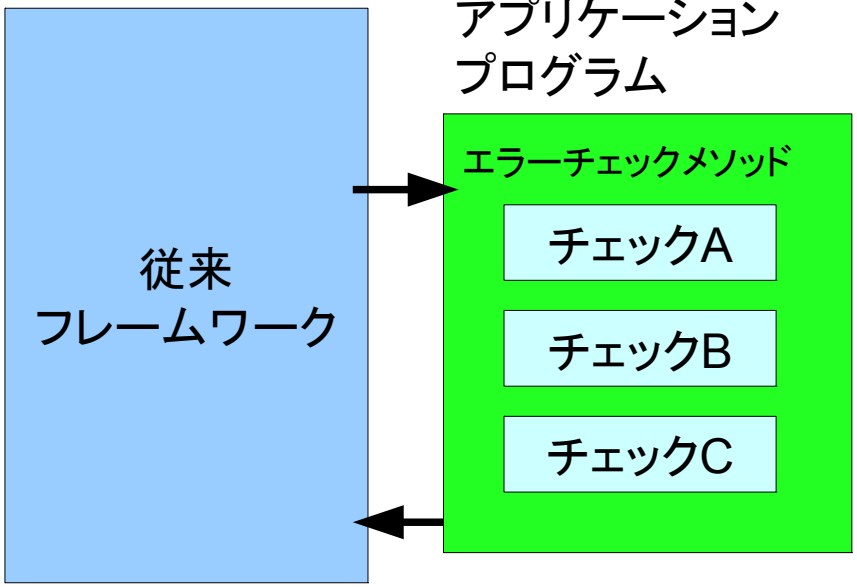
課題1.「テストプログラム開発工数の捻出」



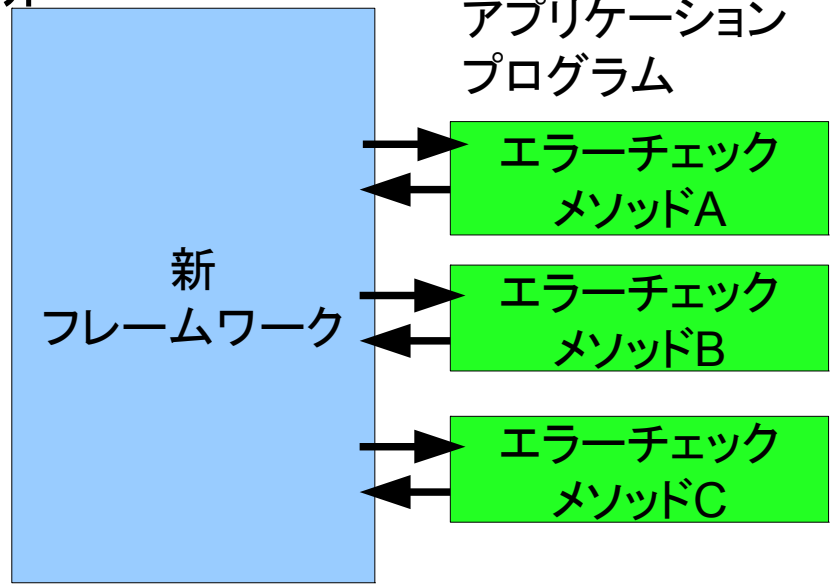
課題1.「テストプログラム開発工数の捻出」

対策. テストファーストによる自動テスト実現

従来



新



テストケース

チェックA	事象A1	YYYY ・・・・
	事象A2	---- ・・・・
チェックB	事象B1	YY-- ・・・・
	事象B2	--YY ・・・・
チェックC	事象C1	Y-Y- ・・・・
	事象C2	-Y-Y ・・・・

メソッドAのテストケース

チェックA	事象A1	Y-
	事象A2	-Y

メソッドBのテストケース

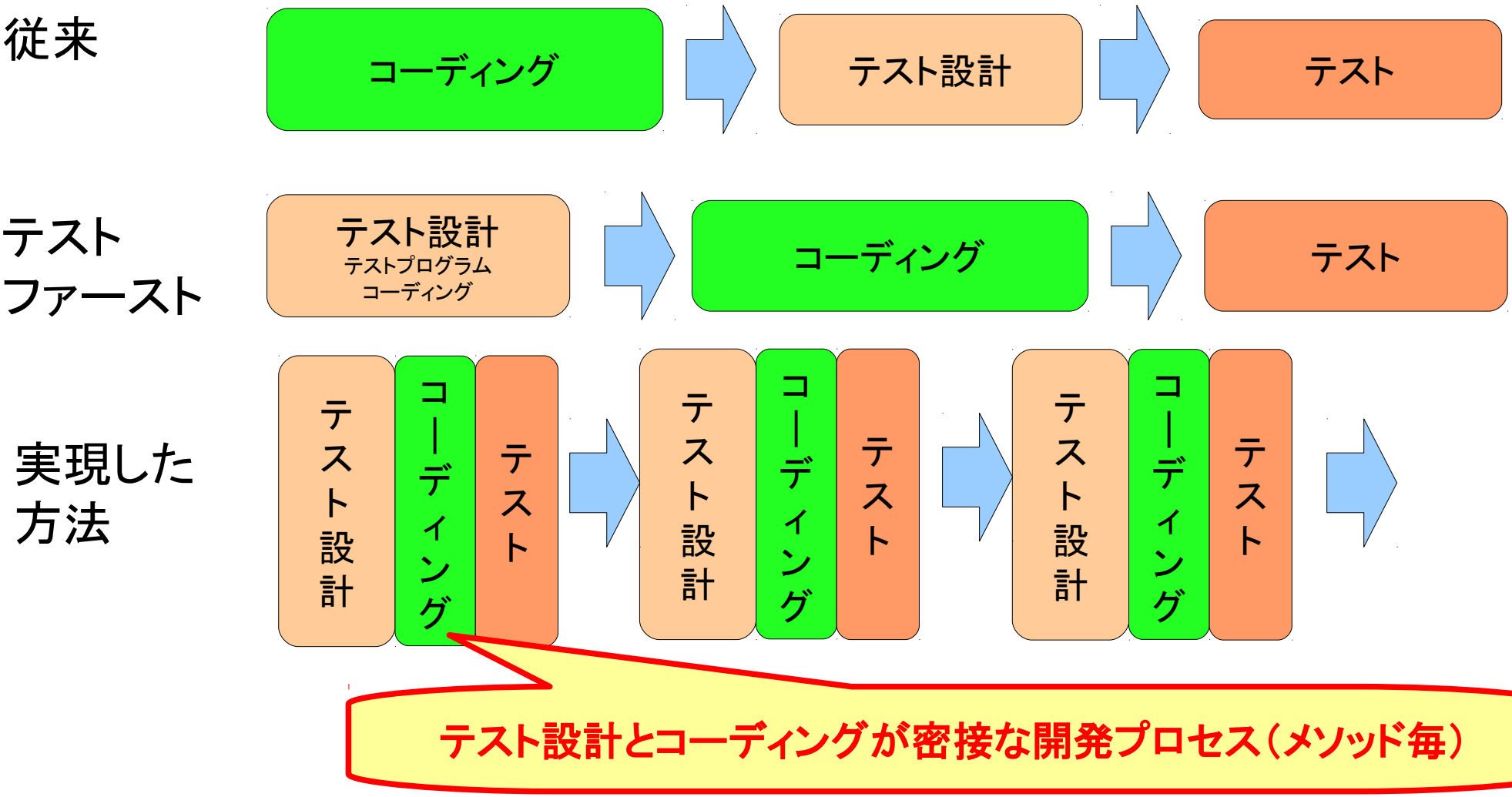
チェックB	事象B1	Y-
	事象B2	-Y

メソッドCのテストケース

チェックC	事象C1	Y-
	事象C2	-Y

課題1.「テストプログラム開発工数の捻出」

対策. テストファーストによる自動テスト実現



課題1.「テストプログラム開発工数の捻出」

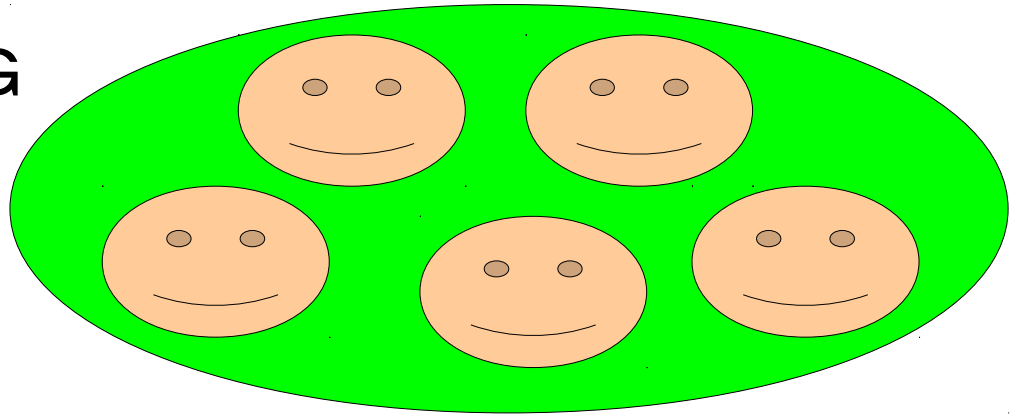
対策. UT項目の削減

- 単体テスト抽出基準　を作成
  テストをする、**しない**　を明確に設定

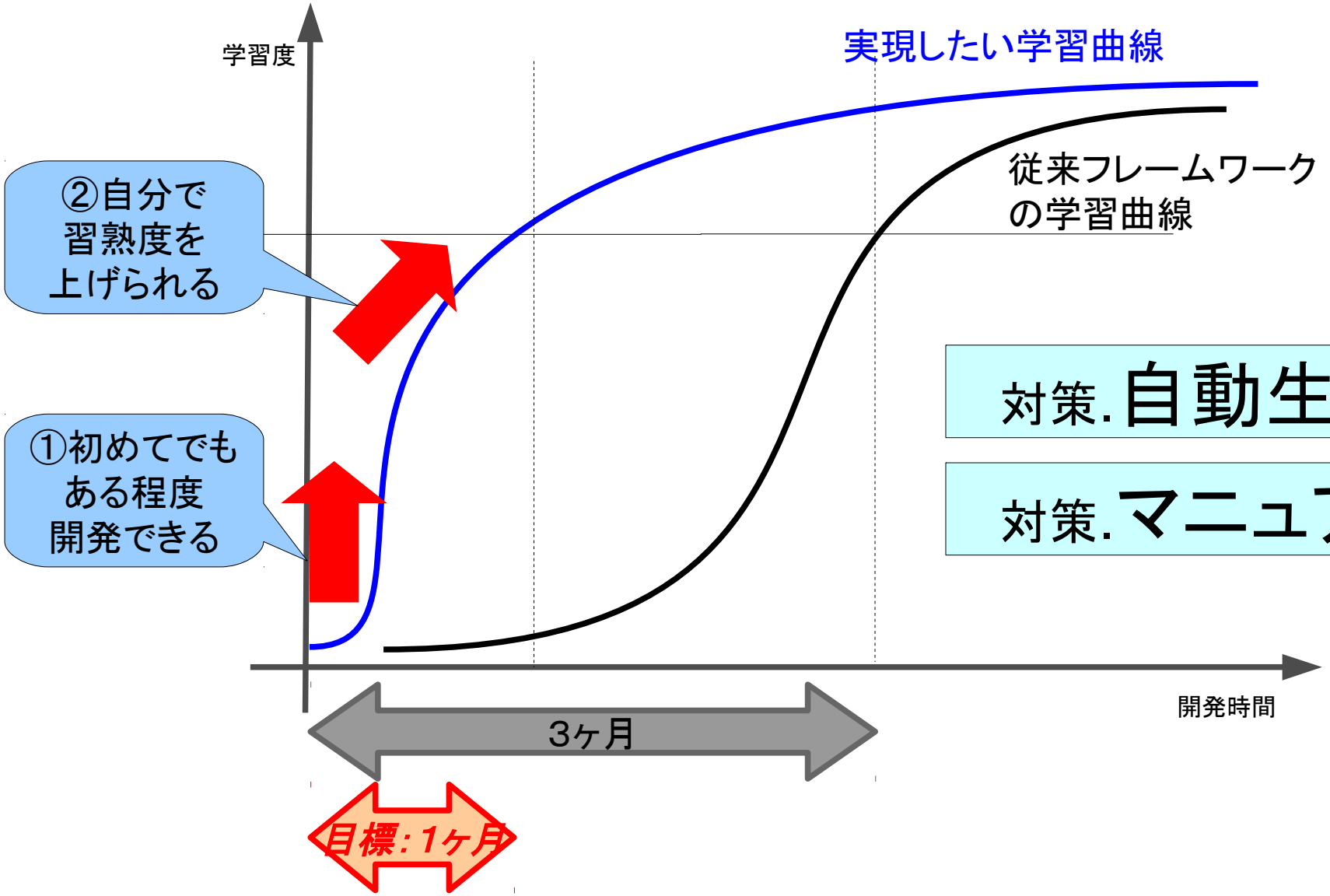
標準テスト 82項目中40項目は
UT不要と決めた

例
- 「画面のタイトルが正しいか」 →外部設計レビュー　PGはテスト不要
- 「画面遷移が正しいか」 →手動テスト
- 「管理者の場合はXXX、管理者以外の場合は〇〇〇」 →自動テスト

単体テストWG
SE（PLクラス）2名
PG　3名
改善　2名



課題2.「教育コストを抑えたい」

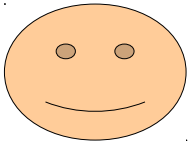


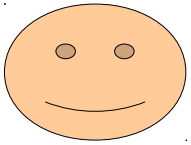
対策.自動生成

対策.マニュアル整備

課題2.「教育コストを抑えたい」

対策.自動生成


改善担当
（私）


FW開発者

- テストプログラムの自動生成
  - 機能の設計情報をデータベース化（リポジトリ）
  - プログラムのインタフェース（メソッド名、引数）も保持
  - インタフェースは15種類
  - 種類に応じたテストプログラムの雛形を自動生成することが可能
  - プログラマーは自動生成された雛形に必要なロジック（テストケース）を追加することでテストプログラムを開発できる

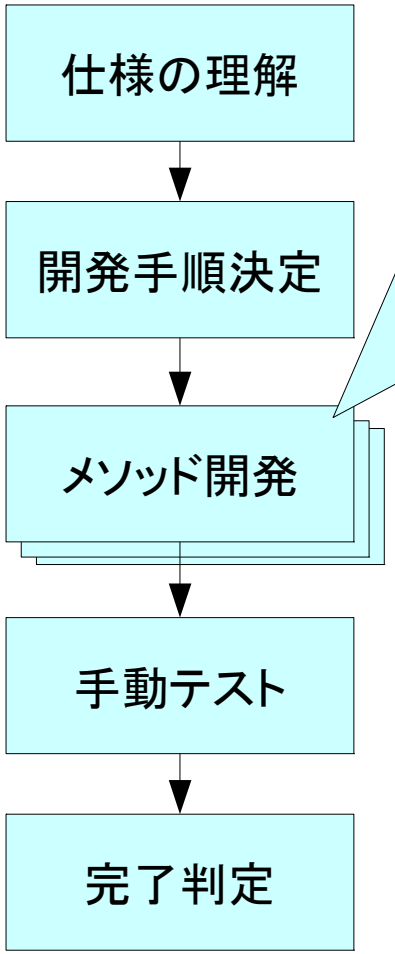
- テストデータの自動生成
  - リポジトリに画面項目を保持
  - データの雛形を生成
  - テストデータはテストプログラムから切り出しCSVファイル
  - テストケースに応じて簡単にコピーして作成することが可能

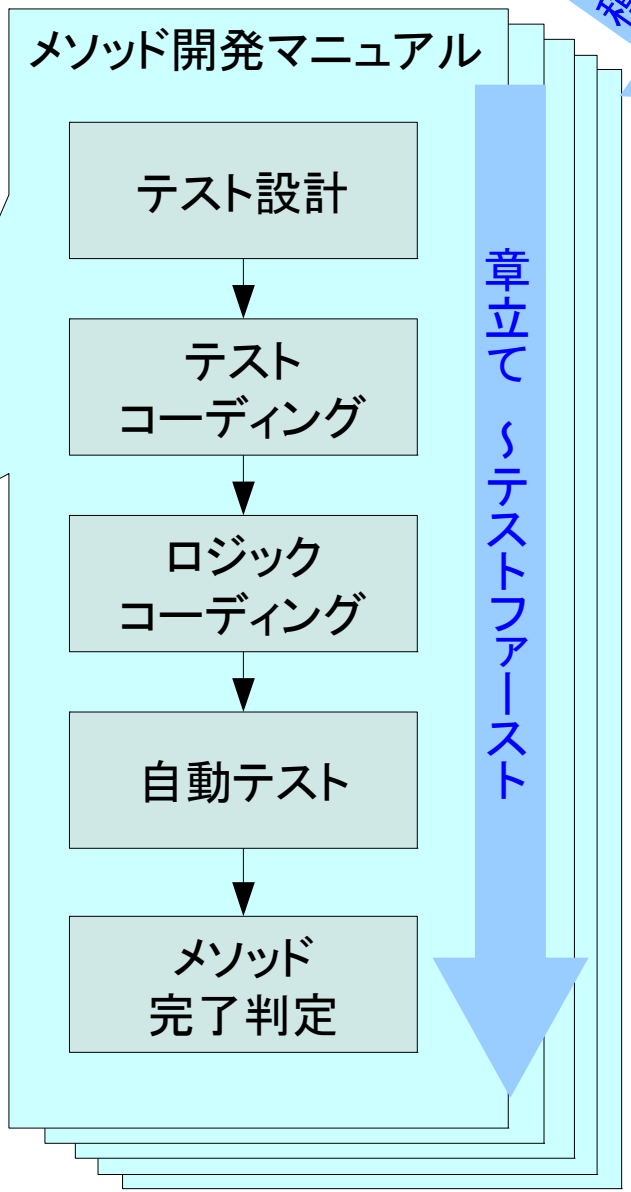
課題2.「教育コストを抑えたい」

対策.マニュアル整備

①章立ての工夫

PG開発手順





種類

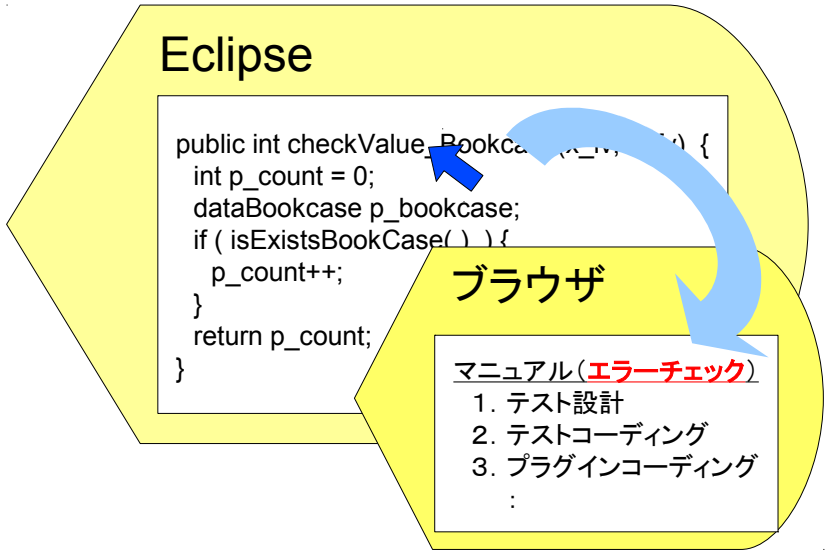
②種類の工夫

15のパターンに分類し
それぞれにマニュアル作成

- エラーチェック
- 更新項目値の加工
- 画面表示属性の設定　など

③参照の工夫

必要な時に**必要な種類**を参照できる
コーディング中　（Eclipse）



課題と対策 まとめ

課題1. テストプログラム開発工数の捻出

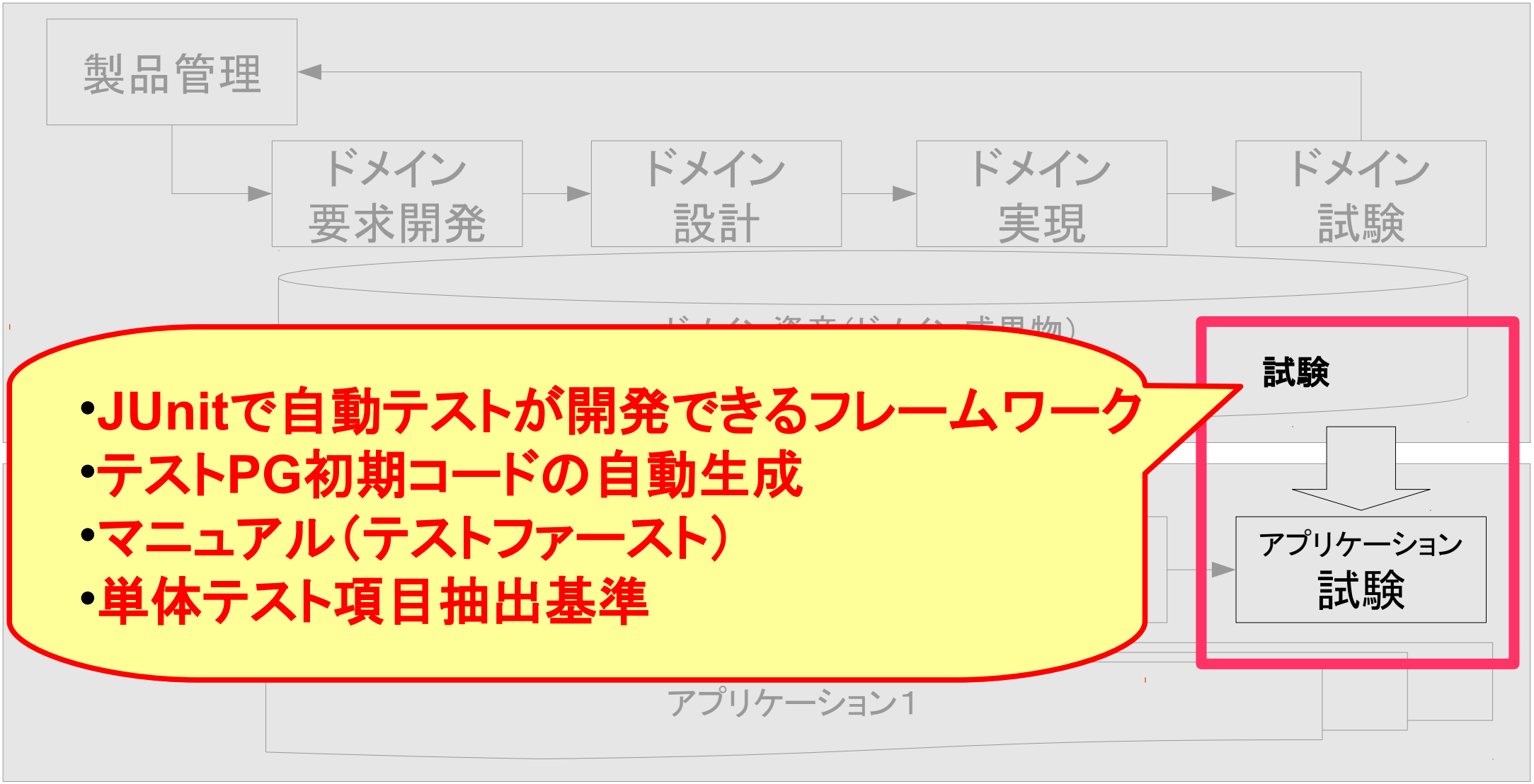
- 対策. テストファーストによる自動テスト実現
- 対策. UT項目の削減

課題2. 教育コストを抑えたい

- 対策. 自動生成
- 対策. マニュアル整備

結果 と 評価

構築したテスト資産



- JUnitで自動テストが開発できるフレームワーク
- テストPG初期コードの自動生成
- マニュアル（テストファースト）
- 単体テスト項目抽出基準

引用元:『ソフトウェアプロダクトラインエンジニアリング　―　ソフトウェア製品系列開発の基礎と概念から技法まで』
クラウス・ポール (著), ギュンター・ベックレ (著), フランク・ヴァン・デル・リンデン (著), 林 好一 (翻訳), 吉村 健太郎 (翻訳), 今関 剛 (翻訳)

試行プロジェクト 開発結果

- 開発結果

	本体コード A	テストコード B	テスト実装率 B/A	自動テスト カバレッジ(C0)
平均	116	348	2.99	96.3%
最小	65	135	2.01	82.3%
最大	224	488	4.73	100%

約96%が自動テストできている

- UT実施時のクリック回数

従来　１．９９回/JaX

今回　１．０３回/JaX

半分以下
→UT工数減

(*)JaX・・・弊社の規模指標（ライン数）

課題、対策の評価

課題1. テストプログラム開発工数の捻出

- 対策. テストファーストによる自動テスト実現
- 対策. UT項目の削減

実装したコードの96%が自動テストできている ◎

UTクリック数が従来と比べて半減している ○

課題2. 教育コストを抑えたい

- 対策. 自動生成
- 対策. マニュアル整備

開発者の大半は予定工数内でテストファーストできている ◎

若手プログラマーの一部は予定工数を超過している △

今後の課題

- 習熟スピードの向上
  - プログラム初心者でも1ヶ月習得を実現したい

- 自動テスト範囲の拡大
  - Seleniumの活用
    - 画面遷移、JavaScript
    - 1機能の中の業務シナリオ

      業務機能はサブメニューが多く、標準シナリオが活用できない

      →リポジトリに個別シナリオを登録することで実現できるのでは？

- テストデータ自動生成（特にデータベース）
  - テストデータまで構成管理していない
  - いつでも同じ自動テストができる

終わり

ご清聴ありがとうございました。