派生開発ならではの品質確保

テクマトリックス株式会社
システムエンジニアリング事業部
ソフトウェアエンジニアリング営業部

アジェンダ

1. 新規開発 vs 保守･派生開発

2. 保守･派生開発における品質向上策

   2.1. テスト範囲を絞る
   - 影響範囲の分析
   - 意図しない影響の検出
   - 類似関係の確認

   2.2. テストの優先順位をつける
   - 現行システムのリスクを把握
   - 未検証部分のチェック

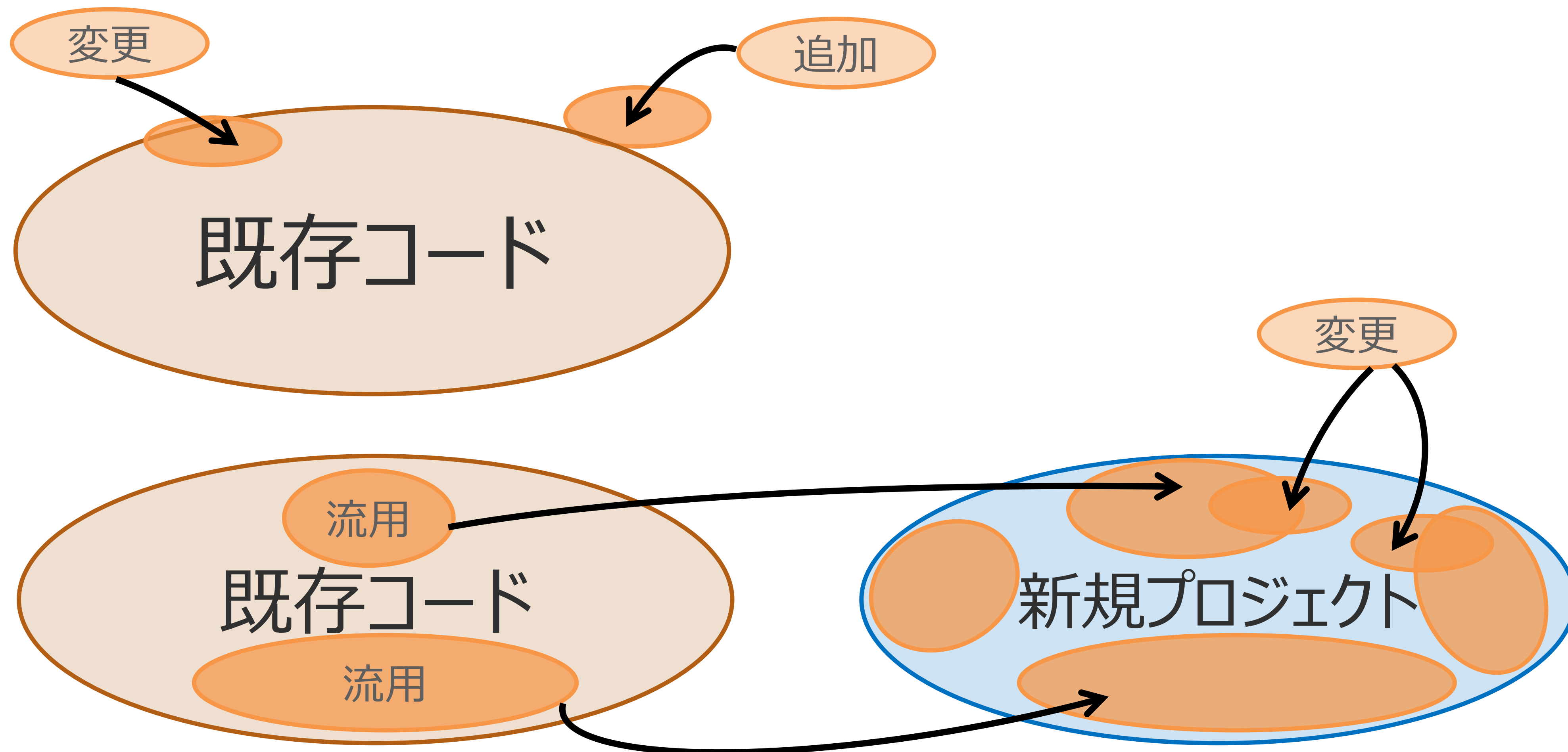
3. まとめ

Copyright © 2018 TechMatrix Corporation. All rights reserved.

新規開発 vs 保守･派生開発

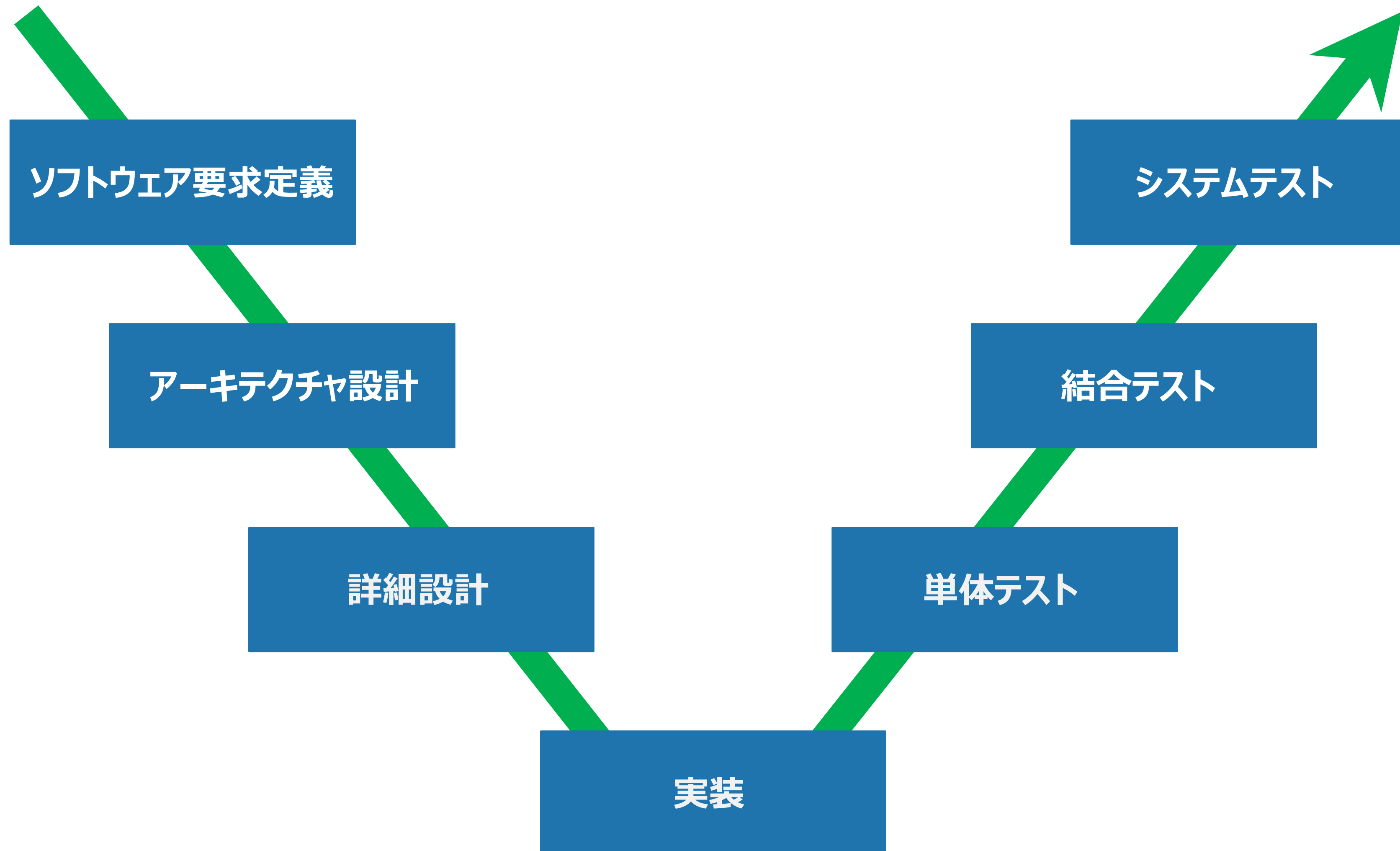
保守・派生開発とは？

リリース済みのソフトウェアに機能改善や不具合修正などの変更を加える、あるいは
リリース済みもしくは開発中のソフトウェアを流用し、新たなソフトウェアを開発すること。



Copyright © 2018 TechMatrix Corporation. All rights reserved.

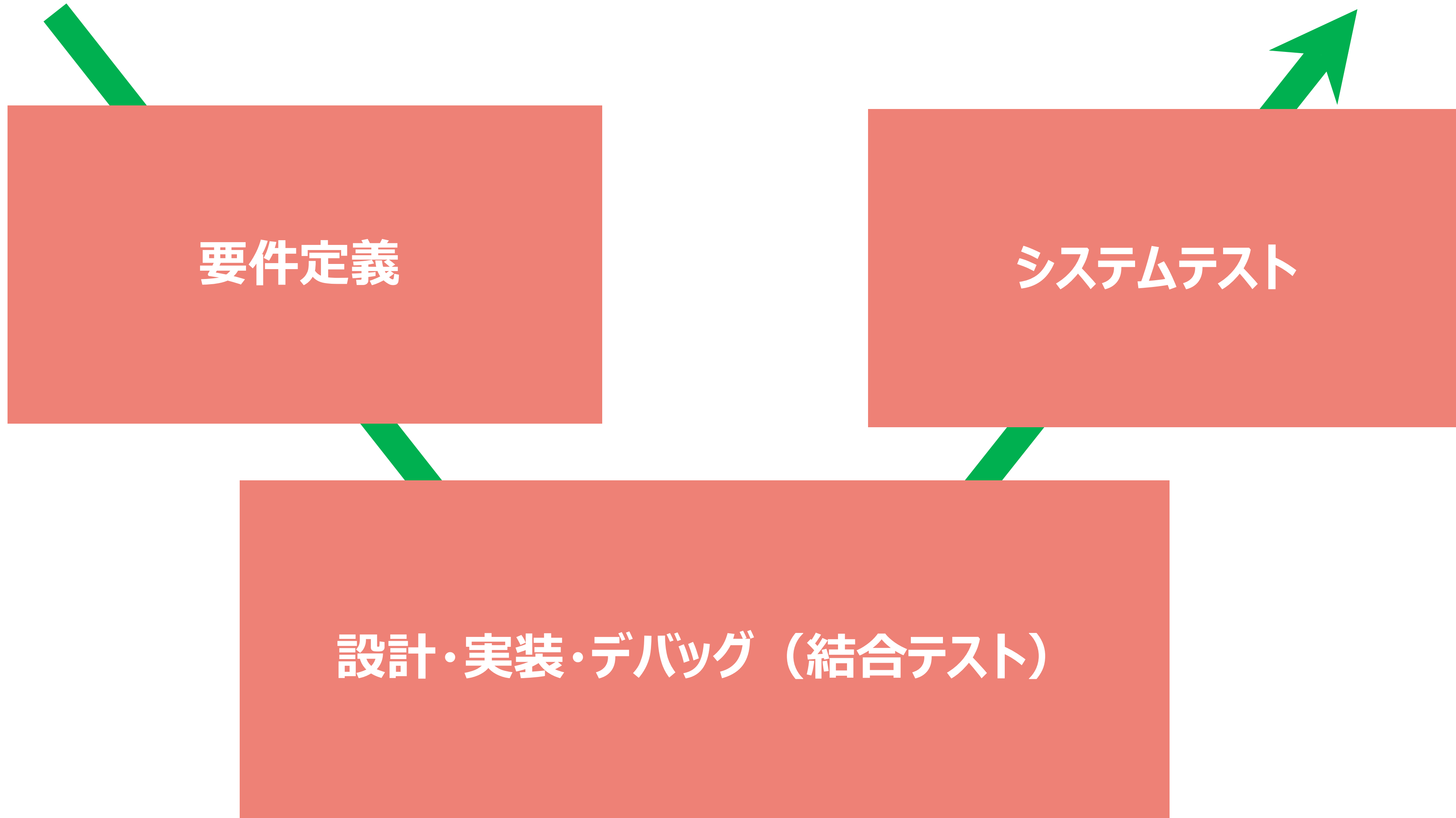
新規開発との違い



Copyright © 2018 TechMatrix Corporation. All rights reserved.

新規開発との違い

新規開発と比べて、保守・派生開発はプロセスが省略される傾向にある



Copyright © 2018 TechMatrix Corporation. All rights reserved.

新規開発との違い

新規開発

- 要件定義や設計は、十分な時間をかけて検討される
- システム全体にバグが含まれることが前提で、広範囲かつ開発の各工程でテストが実施される
- 検証も含め、各工程に十分な工数が割り当てられる

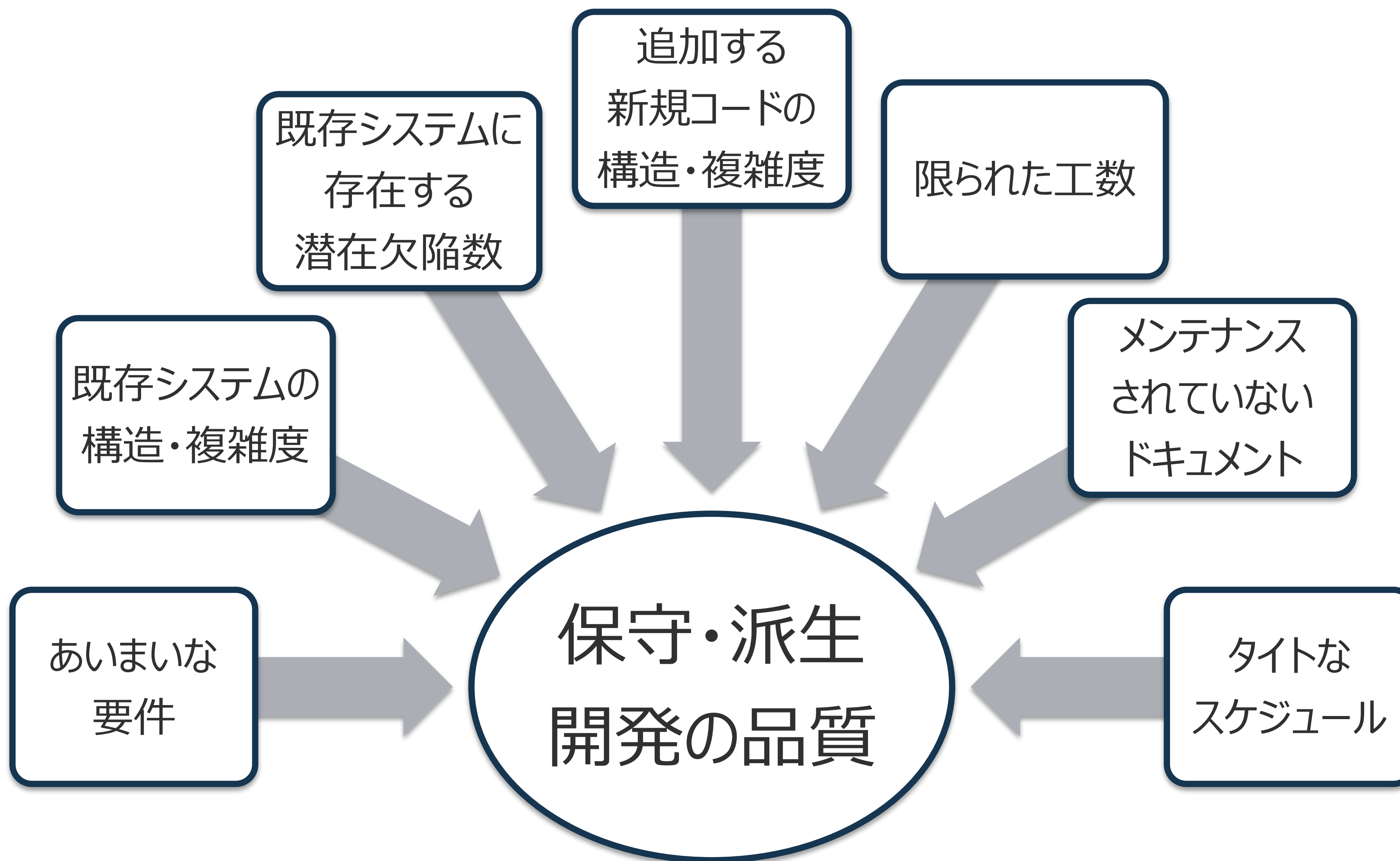
保守・派生開発

- 細部が決まっていない状況で計画が進められる
- 見積もりが不明確なため、工数や期間が十分に確保されない
- バグのないシステムをベースにすることが前提で、変更・流用時の検証は部分的に行われる

保守・派生開発はすぐに作業に着手できてしまう
曖昧な仕様のもと手探りの実装をして、勘に頼った部分的なテストをしてしまう

Copyright © 2018 TechMatrix Corporation. All rights reserved.

保守・派生開発の品質に影響を与える要因



Copyright © 2018 TechMatrix Corporation. All rights reserved.

派生開発の現実

ドキュメント

テストケース

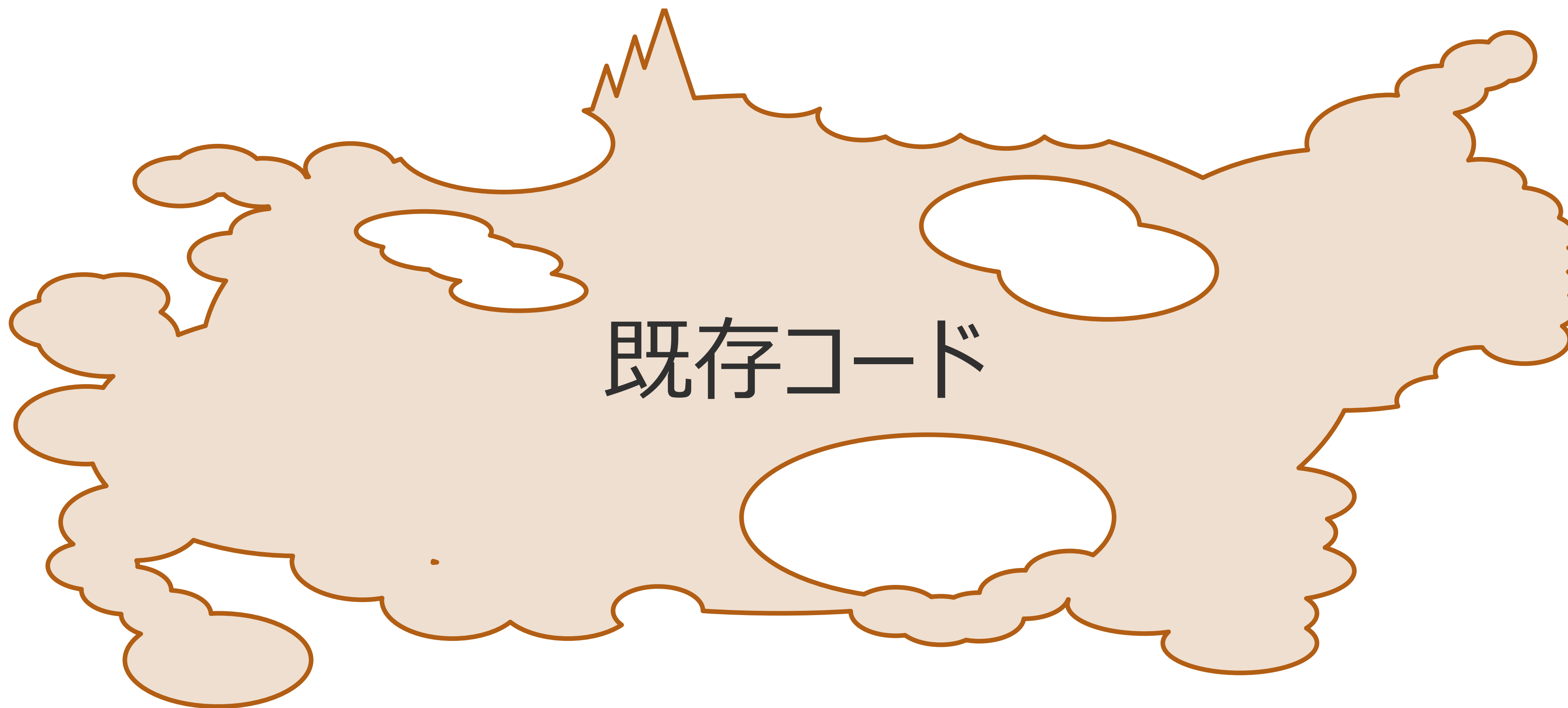
既存コード

Copyright © 2018 TechMatrix Corporation. All rights reserved.

派生開発の現実

ソースコードが古く

- ドキュメントが存在しない（あってもメンテナンスされていない）
- 有識者がいない（特に可読性の低いコードに限って）
- テストケースも存在しない



つぎはぎだらけの、誰も理解できていないソースコードが生まれる

Copyright © 2018 TechMatrix Corporation. All rights reserved.

保守・派生開発における品質向上策

保守・派生開発における品質向上策

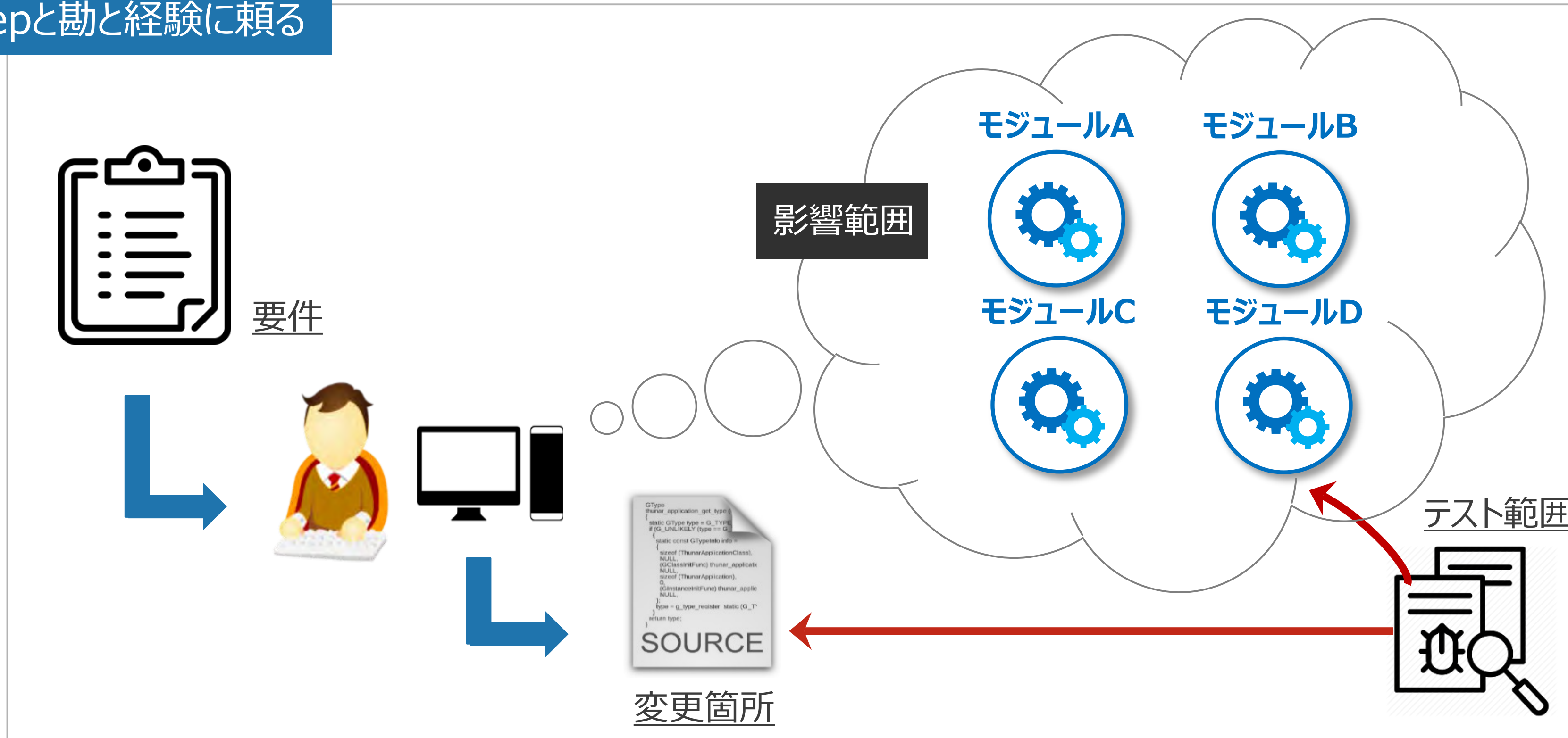
テスト範囲を絞る

テストの優先順位を付ける

Copyright © 2018 TechMatrix Corporation. All rights reserved.

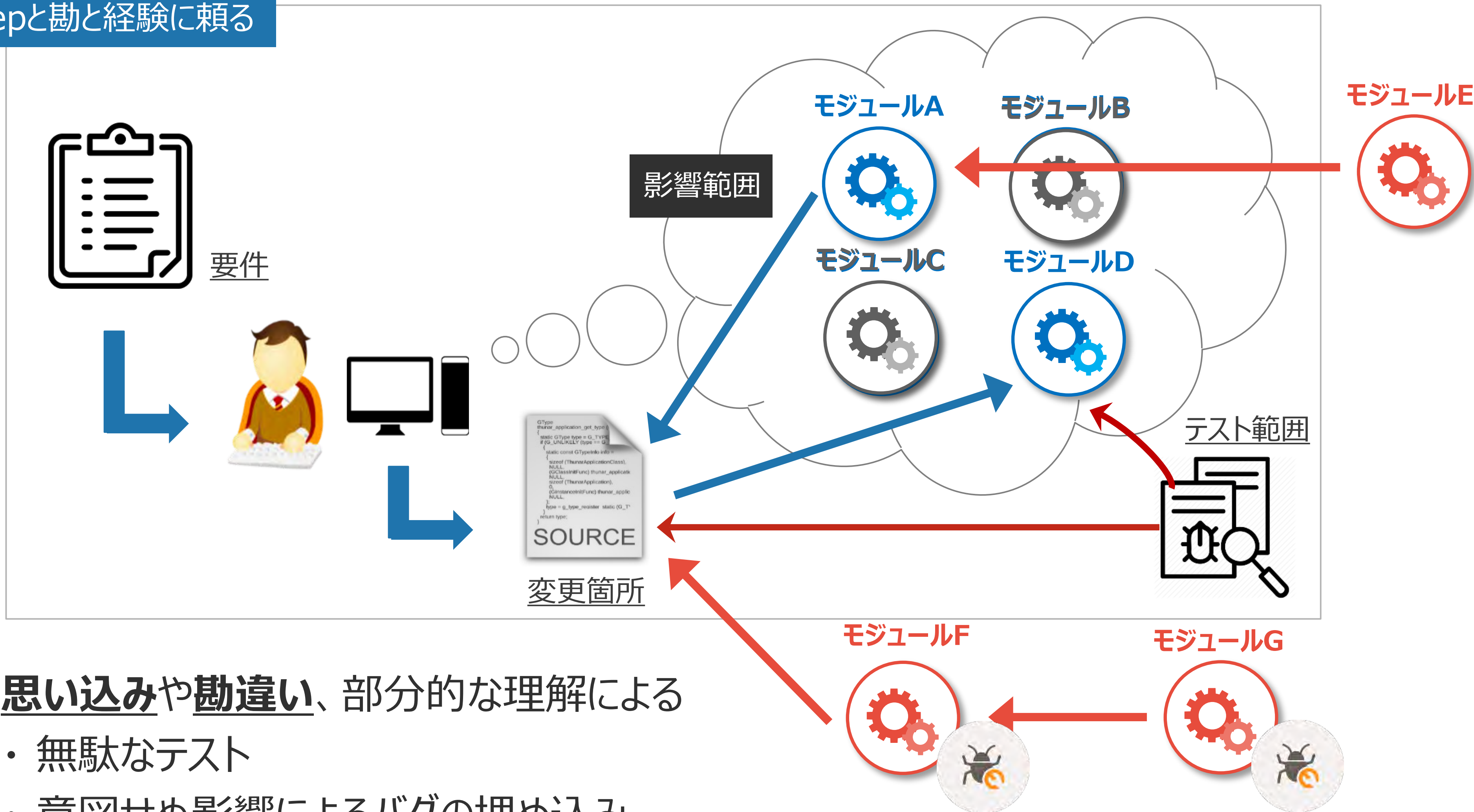
テスト範囲を絞る - 影響範囲の分析

影響範囲の分析 - 勘と経験に頼った推測



Copyright © 2018 TechMatrix Corporation. All rights reserved.

影響範囲の分析 - 勘と経験に頼った推測



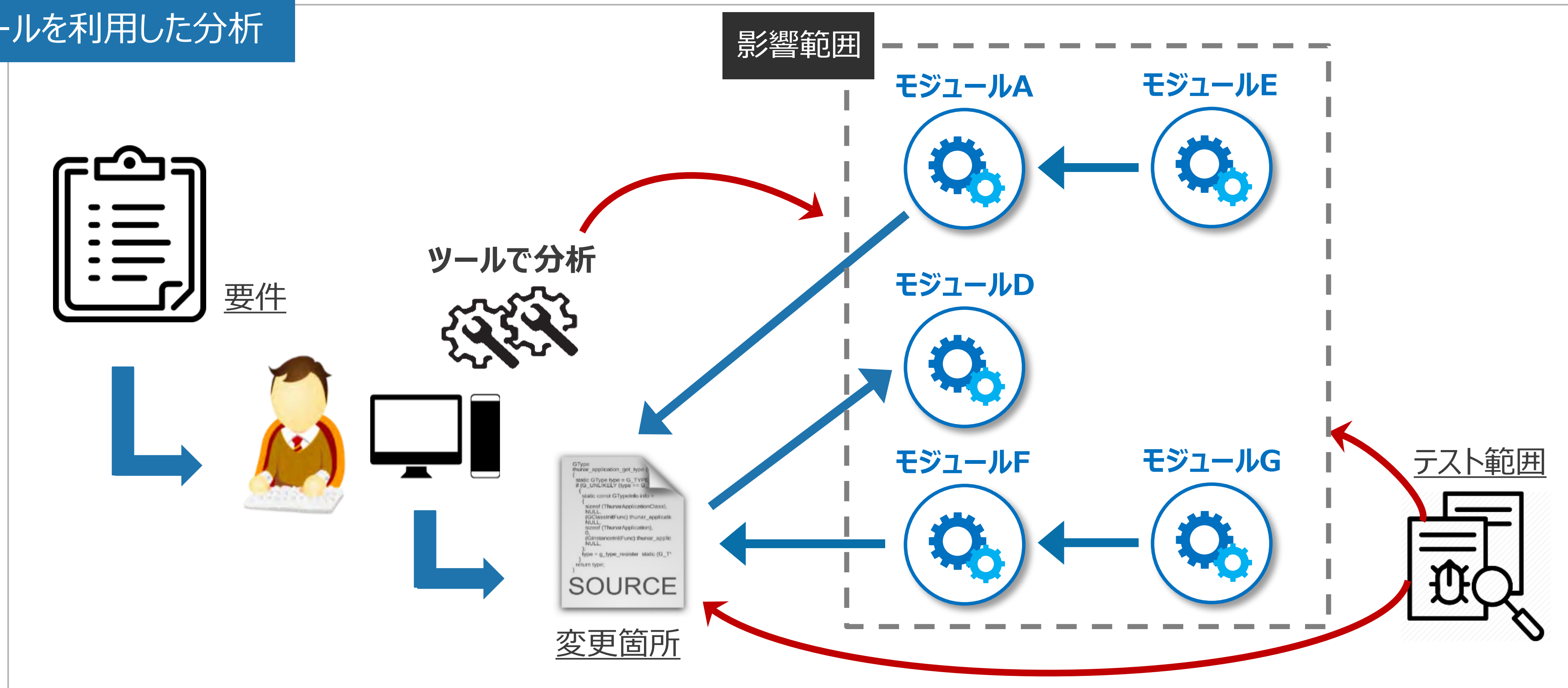
思い込みや**勘違い**、部分的な理解による

- 無駄なテスト
- 意図せぬ影響によるバグの埋め込み

が発生する

Copyright © 2018 TechMatrix Corporation. All rights reserved.

影響範囲の分析 - ツールを利用した分析



ツールを使って影響範囲を分析することで、以下を実現することができる

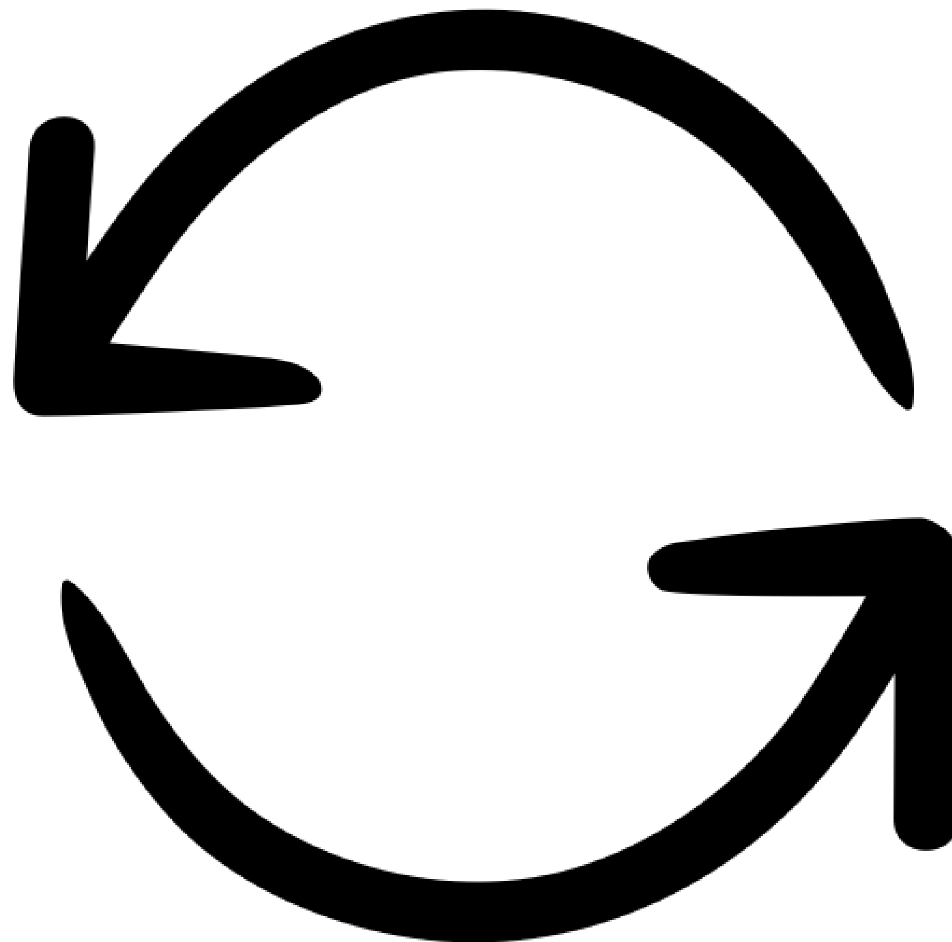
- 分析作業の正確性向上と工数削減
- 無駄なテストの削減
- 変更による意図せぬ問題を防止
- 影響を把握せずに変更を進めたことによる手戻り作業の防止

Copyright © 2018 TechMatrix Corporation. All rights reserved.

テスト範囲を絞る - 意図しない影響の検出

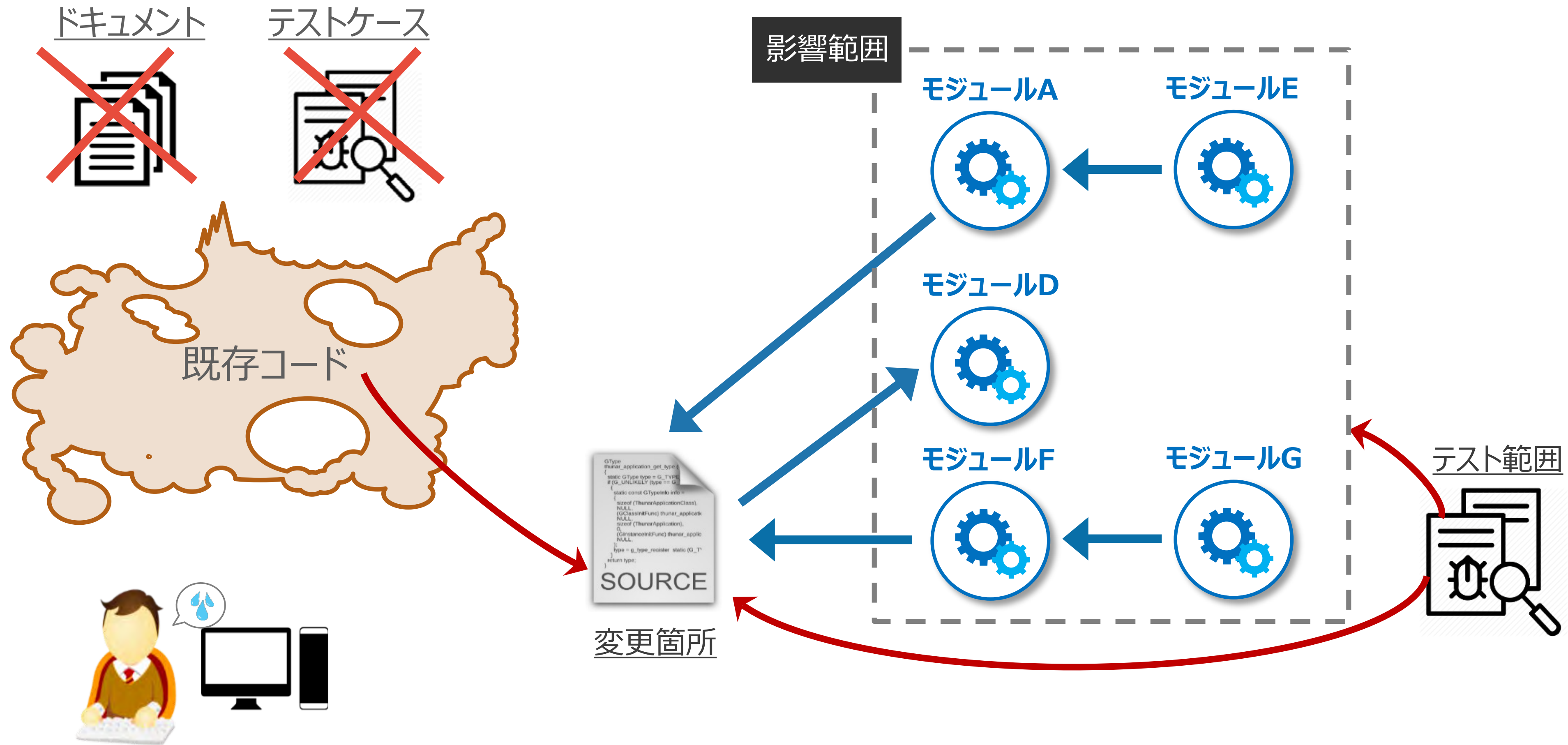
意図しない影響を検出 - 回帰テスト

回帰テストによって、変更による意図しない箇所への影響を検出できる



Copyright © 2018 TechMatrix Corporation. All rights reserved.

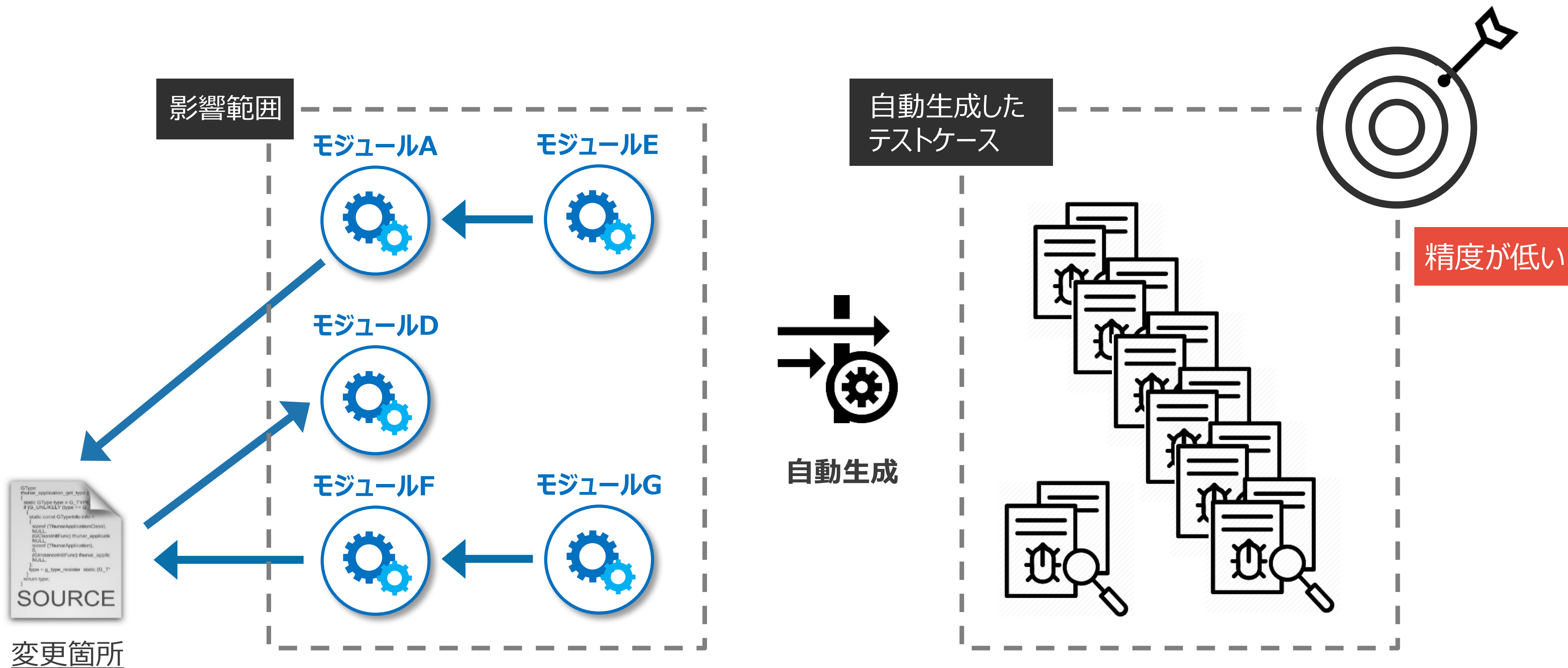
影響範囲の回帰テスト



手元にあるのは、誰も把握できていない既存コードだけ
影響範囲を特定できたのはいいけど、どうやってテストケースを作成するの。。？

Copyright © 2018 TechMatrix Corporation. All rights reserved.

影響範囲の回帰テスト - テストケースの自動生成



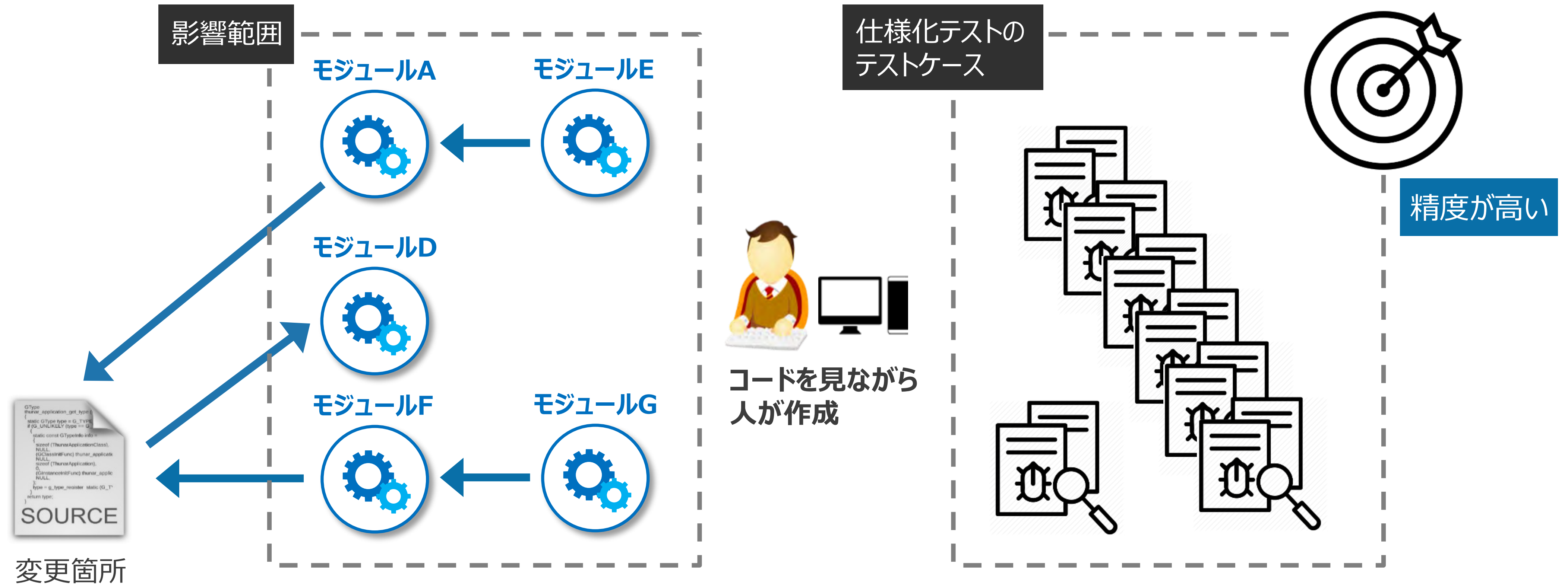
テストケースの精度が低く、開発者のコード理解も深まらない

おススメはしないが、やらないよりは良い

Copyright © 2018 TechMatrix Corporation. All rights reserved.

影響範囲の回帰テスト - 仕様化テストとは？

既存コードの振る舞いを明らかにするためのテスト。
コードに変更を加える前に作成し、現状の振る舞いを記録する。

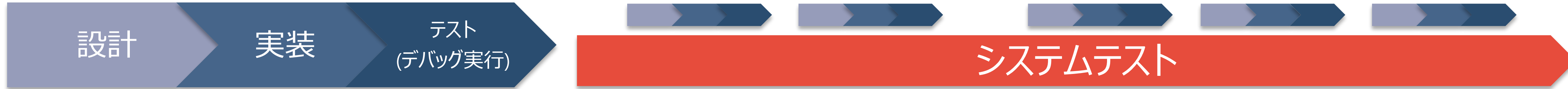


開発者のコード理解が深まるメリットがある
仕様化テストのテストケースは現状を記録したドキュメントとも言える

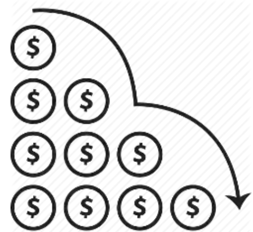
Copyright © 2018 TechMatrix Corporation. All rights reserved.

影響範囲の回帰テスト

- 手探りの実装をして、部分的なテスト(デバッグ)をした場合



- 影響範囲の分析をした上で実装に着手し、影響範囲の回帰テストを用意した場合



最初は工数・コストが余計に増えると感じるかもしれない
しかし、トータルでは工数・コストを抑えることができる
より工数を抑えるために、テスト準備は、**ツールを使った効率化**が必須と言える

Copyright © 2018 TechMatrix Corporation. All rights reserved.

Parasoft C++test - 単体テストの支援

静的テスト・動的テストを開発プロセスに組み込むことで開発の早い段階から品質への取り組みを行います。

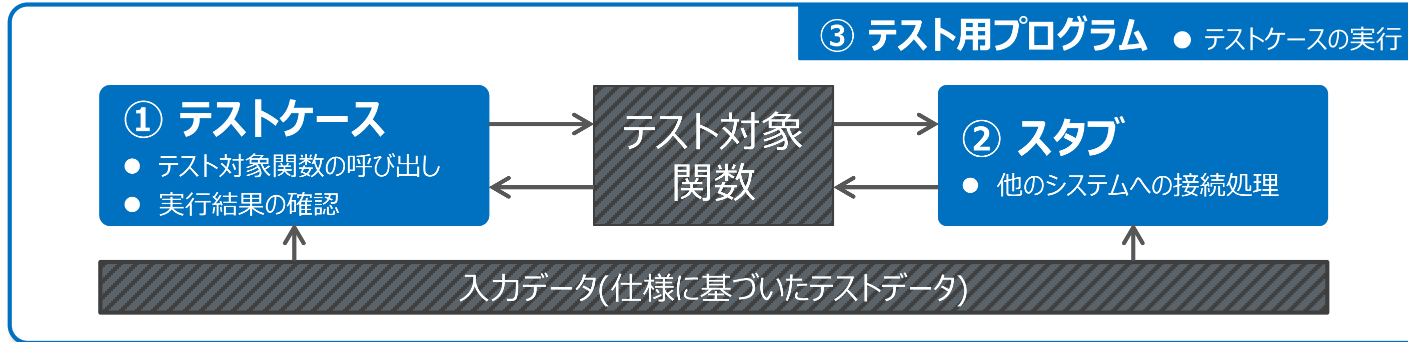
区分	項目	機能	説明
静的テスト	● バグを検出	フロー解析	開発の初期段階で結合レベルのバグを検出し、手戻り工数を削減する。
	● コーディング規約違反を検出	コーディング規約チェック	バグの発生しやすい実装や言語の仕様、セキュリティの規約などをチェックし、ソースコードの記述を平準化する。
	● ソフトウェア品質を計測	メトリクス計測	メソッド数やサイクロマチック複雑度、オブジェクトの結合度などを計測し、品質の低いコードを検出する。
動的テスト	● テストケース、スタブの作成 ● カバレッジの計測	単体テスト支援	コーディングを行うことなく、テストケースとスタブの作成が可能。単体テストにおいて、最大の課題とされるコーディング作業を大幅に削減する。
	● 機能・結合テストの可視化 ● メモリ関連エラーの検出	アプリケーションモニタリング	アプリケーションを実行し操作した際に実行された処理・実行されなかった処理を計測することで、テストの網羅性や整合性を確認する。また、メモリ関連のエラーを検出する。

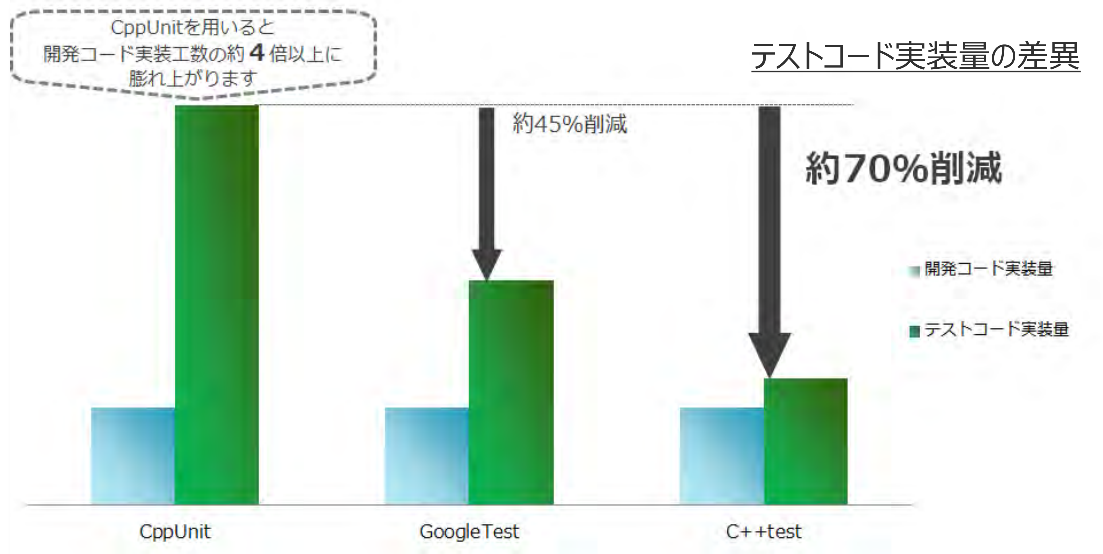
Copyright © 2018 TechMatrix Corporation. All rights reserved.

Parasoft C++test - 単体テストの支援

単体テスト･回帰テストの仕組みを構築･実行

- 単体テストの実行に必要なテスト資産の生成、単体テストの実行、結果のレポートを行います。
- テスト対象の関数を選定し入力データを準備すれば、他のテスト資産はC++testが生成します。
- 入力データを準備せず、テストケースの自動生成をすることも可能です。（推奨はしません）





カバレッジ計測結果の例

Copyright © 2018 TechMatrix Corporation. All rights reserved.

テスト範囲を絞る - 類似関係の確認

修正漏れの確認

類似コード

変更箇所

同様の変更は不要

同様の変更が必要

修正したコードと類似のコードが存在する場合、**「修正漏れ」**が存在する可能性がある
類似のコードも同じように修正する必要があるか確認しなければならない

Copyright © 2018 TechMatrix Corporation. All rights reserved.

Parasoft C++test - 重複コードの検出

静的テスト・動的テストを開発プロセスに組み込むことで開発の早い段階から品質への取り組みを行います。

静的テスト	● バグを検出	フロー解析	開発の初期段階で結合レベルのバグを検出し、手戻り工数を削減する。
	● コーディング規約違反を検出	コーディング規約チェック	バグの発生しやすい実装や言語の仕様、セキュリティの規約などをチェックし、ソースコードの記述を平準化する。
	● ソフトウェア品質を計測	メトリクス計測	メソッド数やサイクロマチック複雑度、オブジェクトの結合度などを計測し、品質の低いコードを検出する。
動的テスト	● テストケース、スタブの作成 ● カバレッジの計測	単体テスト支援	コーディングを行うことなく、テストケースとスタブの作成が可能。単体テストにおいて、最大の課題とされるコーディング作業を大幅に削減する。
	● 機能・結合テストの可視化 ● メモリ関連エラーの検出	アプリケーションモニタリング	アプリケーションを実行し操作した際に実行された処理・実行されなかった処理を計測することで、テストの網羅性や整合性を確認する。また、メモリ関連のエラーを検出する。

Copyright © 2018 TechMatrix Corporation. All rights reserved.

Parasoft C++test - 重複コードの検出

重複コードを自動で検出

- 重複コードと見なす条件をパラメータで柔軟に設定することができます。

パラメータ設定

CDD-DUPC: コードの重複を避ける

CDD-DUPC のパラメーター(詳細はルールの説明を参照してください):

重複トークンの最小数 20

☑ 文字列リテラルを無視する
☑ 数値リテラルを無視する
☑ ブーリアン リテラルを無視する
☑ 識別子を無視する

発見された重複コードをレポート

◉ テスト スコープの任意の場所
○ 同一プロジェクト中
○ 同一ファイル中

重複コード検出例

類似コード

sample.c　sample2.c　比較

重複コード ビューアー

sample.c (35-133)

```
int increase_cnt_ins(FILE *fp)
{
    int ins_line ;              /* 追加ライン数 */
    int part_line_tmp ;         /* 部分カウントライン数 */
    char buf[READ_SIZE + 1] ;   /* ファイル読み込みバッファ */
    char *ret_p ;               /* fgetsの戻り値 */

    /* 初期化 */
    ins_line = 0 ;
    memset(buf, '\0', sizeof(buf)) ;

    /* 部分カウント実行回数初期化 */
    sPart_count_no = 0 ;

    /* ファイル終端まで文字列を1行ずつ読み込む */
    for( ; (fgets(buf,READ_SIZE,fp)) != NULL ; )
    {
        /* 部分カウント開始文字列発見 */
        if ( strstr(buf, PART_COUNT_START) != NULL )
        {
            /* 部分カウント可能数オーバー */
            if ( sPart_count_no > PART_COUNT_MAX )
```

sample2.c (24-122)

```
int increase_cnt_chg(FILE *fp)
{
    int chg_line ;              /* 変更ライン数 */
    int part_line_tmp ;         /* 部分カウントライン数 */
    char buf[READ_SIZE + 1] ;   /* ファイル読み込みバッファ */
    char *ret_p ;               /* fgetsの戻り値 */

    /* 初期化 */
    chg_line = 0 ;
    memset(buf, '\0', sizeof(buf)) ;

    /* 部分カウント実行回数初期化 */
    sPart_count_no = 0 ;

    /* ファイル終端まで文字列を1行ずつ読み込む */
    for( ; (fgets(buf,READ_SIZE,fp)) != NULL ; )
    {
        /* 部分カウント開始文字列発見 */
        if ( strstr(buf, PART_COUNT_START) != NULL )
        {
            /* 部分カウント可能数オーバー */
            if ( sPart_count_no > PART_COUNT_MAX )
```

品質タスク　カバレッジ　抑制　タスク　コンソール　Find Duplicated Code...　スタブ

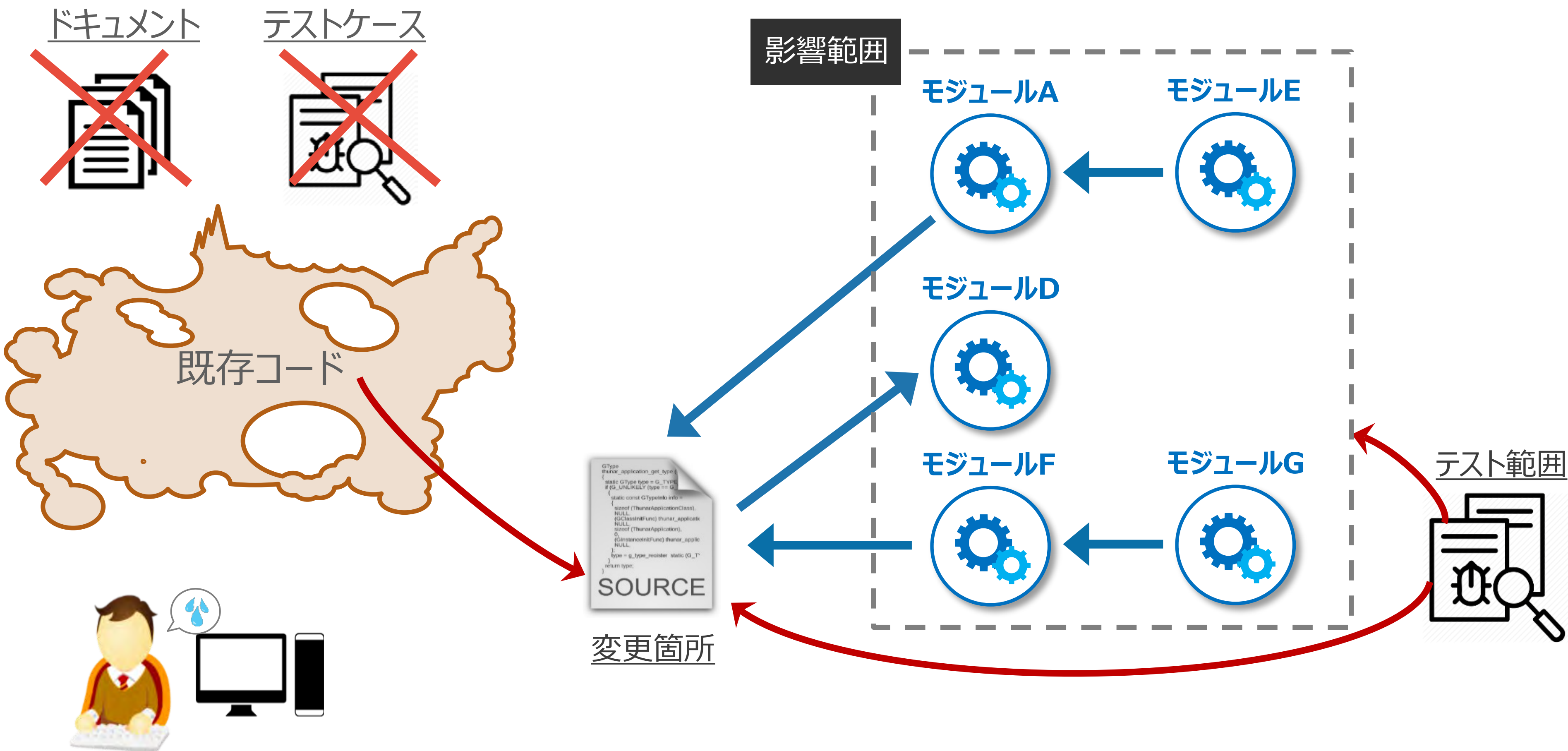
111 解析結果, 0 コード レビュー (フィルターは 111 タスクのうち 3 に一致しました)

- [3] >静的解析違反の修正
  - [3] >重複コードの検出
    - [2] >コードの重複を避ける (CDD-DUPC-3)
      - [2] >不明
        - [2] >sample.c
          - >[行 18] 重複コード: 'if (a < b) { r = b - a; } else { r = a - b;...'
            - >[行 18] ファイル 'sample.c' に重複コード
            - >[行 128] ファイル 'sample2.c' に重複コード
          - >[行 35] 重複コード: 'int increase_cnt_ins(FILE *fp){ int ins_line ; ...'
            - >[行 35] ファイル 'sample.c' に重複コード
            - >[行 24] ファイル 'sample2.c' に重複コード

Copyright © 2018 TechMatrix Corporation. All rights reserved.

テストの優先順位を付ける - 現行システムのリスクを把握

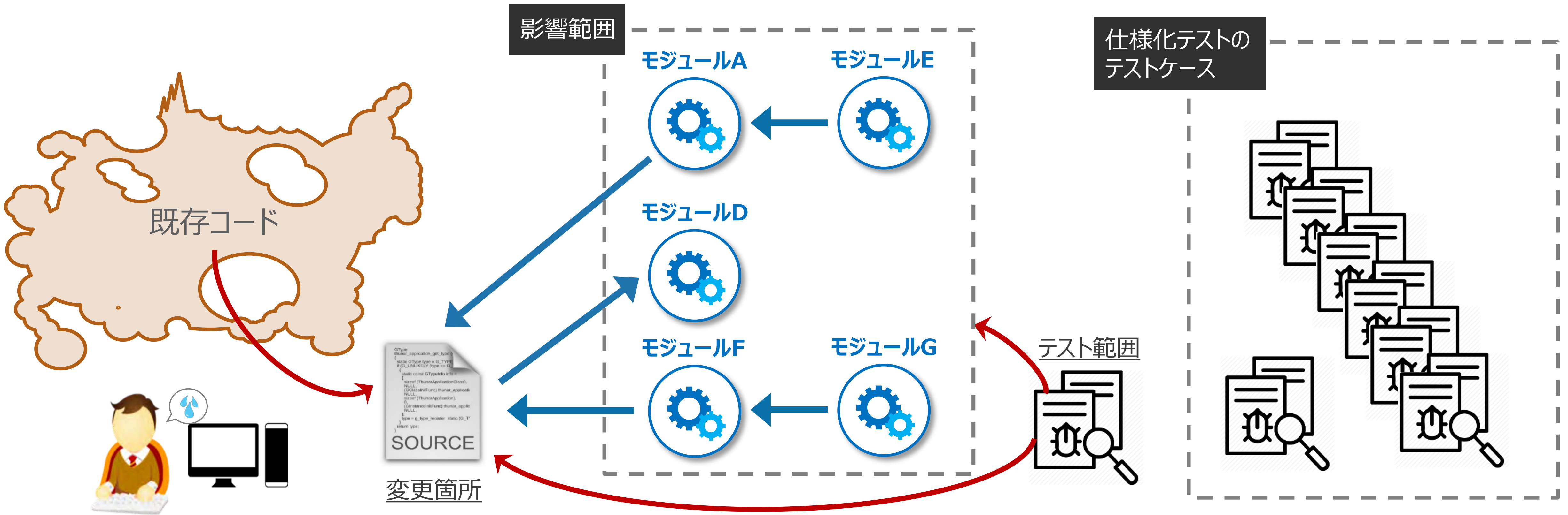
影響範囲を特定した後の状況を振り返る



手元にあるのは、誰も把握できていない既存コードだけ
影響範囲を特定できたのはいいけど、どうやってテストケースを作成するの。。？

Copyright © 2018 TechMatrix Corporation. All rights reserved.

限られた期間とリソース



テストすべき範囲とテストケースの作成方法は分かった
でも、限られた期間とリソースでどうやってテストケース作成の工数を確保するの。。？

Copyright © 2018 TechMatrix Corporation. All rights reserved.

欠陥多発モジュールの存在はきわめて一般的な現象であって、徹底した是正処置をとらない限り、大規模なシステムすべてで起こりうる事実であった。

『ソフトウェア開発の定量化手法（第3版）』
著者名:Capers Jones(著) / 富野 壽、小坂 恭一(監訳)
発売元:共立出版（株）
刊行日:2010.07.25
ISBN:978-4-320-09758-2

Copyright © 2018 TechMatrix Corporation. All rights reserved.

リスクの高いモジュールを優先的に

- コスト、リソースが限られている保守・派生開発では十分なテスト工数を確保できないケースもある。

- 優先順位を付けてテストすることも必要。欠陥が特定のモジュールに偏っているのであれば、全てを同じレベルでテストするのではなく、危険なモジュールはテストの優先度をあげる、より細かなテストをする、といったレベル分けをするということ。

- 変更のリスクが高い場合は、別の方法で実現する方法がないか検討する。要件だけで見積もりをすると、適切な見積もりができずにプロジェクト全体の工程に影響を及ぼすことも。

欠陥多発モジュール

Copyright © 2018 TechMatrix Corporation. All rights reserved.

Parasoft C++test - メトリクス計測

静的テスト・動的テストを開発プロセスに組み込むことで開発の早い段階から品質への取り組みを行います。

静的テスト

● バグを検出	フロー解析	開発の初期段階で結合レベルのバグを検出し、手戻り工数を削減する。
● コーディング規約違反を検出	コーディング規約チェック	バグの発生しやすい実装や言語の仕様、セキュリティの規約などをチェックし、ソースコードの記述を平準化する。
● ソフトウェア品質を計測	メトリクス計測	メソッド数やサイクロマチック複雑度、オブジェクトの結合度などを計測し、品質の低いコードを検出する。

動的テスト

● テストケース、スタブの作成 ● カバレッジの計測	単体テスト支援	コーディングを行うことなく、テストケースとスタブの作成が可能。単体テストにおいて、最大の課題とされるコーディング作業を大幅に削減する。
● 機能・結合テストの可視化 ● メモリ関連エラーの検出	アプリケーションモニタリング	アプリケーションを実行し操作した際に実行された処理・実行されなかった処理を計測することで、テストの網羅性や整合性を確認する。また、メモリ関連のエラーを検出する。

Copyright © 2018 TechMatrix Corporation. All rights reserved.

Parasoft C++test - メトリクス計測

ソフトウェアの品質を定量的に評価

- 開発したソースコードを、ソフトウェアの品質を定量的に評価できる数値として計測し、品質を測定します。
- 計測できるメトリクスの例
  - オブジェクト間の結合
  - McCabe Cyclomatic Complexity
  - コメントの割合/行数
  - ファンアウト
  - Halstead complexity
  - クラスの継承の深さ
  - 凝集性の欠如
  - 保守性インデックス
  - ネストの深さ
  - コード行数
  - メソッドのパラメータ数
  - クラス数
  - ファイル数
  - 空白行数

メトリクス計測の結果の例

概要（メソッド単位）

名前空間	ファイル	タイプ	メソッド	McCabe Cyclomatic Complexity
	csv_check.c		GetFilename	11
	csv_check.c		CheckCsvData	14
	csv_check.c		GetCsvData	6
	csv_check.c		CheckInfoSet	21

…テム数　1 - 60 / 60 アイテ…

メトリクス詳細

詳細

全般

メトリクス	値
McCabe Cyclomatic Complexity	14
メソッドの論理行数に対するコメントの割合	0.73
ネストされた if 文の深さ	2
Essential Cyclomatic Complexity	9
Halstead 難度	31.86
Halstead 心理的稼働時間	64875.03
Halstead インテリジェントコンテント	63.9
Halstead プログラム長	323
Halstead 抽象化レベル	0.03
Halstead 障害件数	0.54
Halstead プログラミング時間	3604.17
Halstead プログラム語彙数	79
Halstead プログラム容量	2036.12
Modified Cyclomatic Complexity	14
ブロックのネストの深さ	5
メソッドの空白行数	18

ファイル (メトリクス): csv_check.c

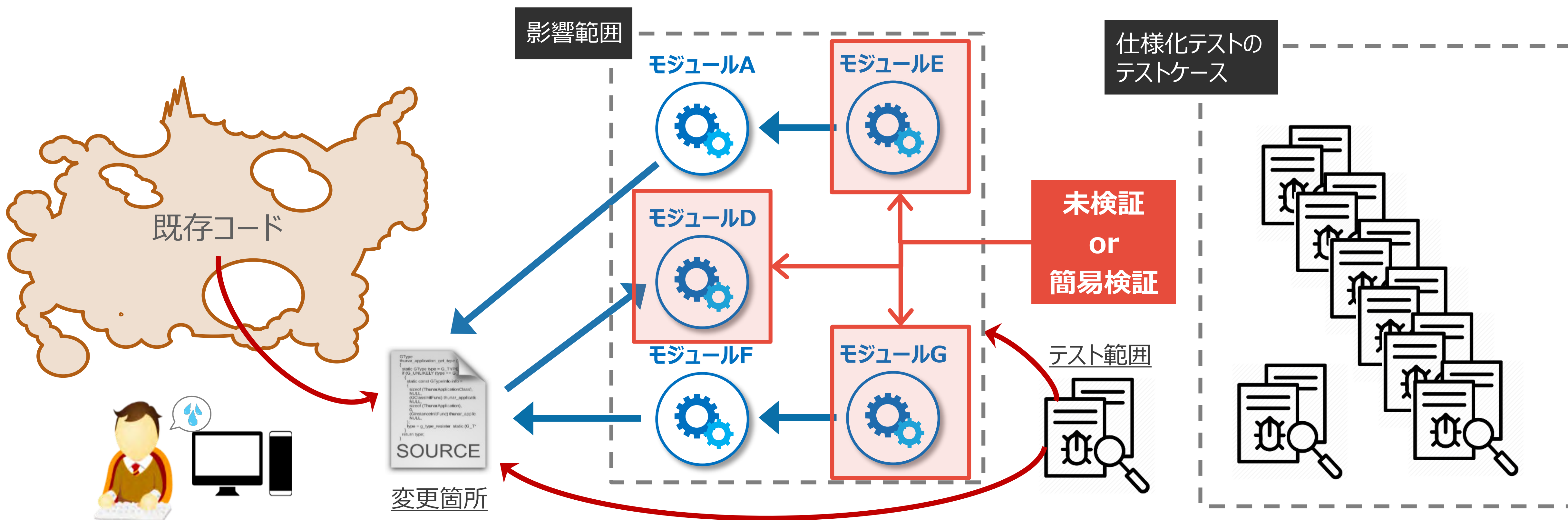
```
218  * NOTE          : 第1引数 : 読み込むファイル数  (IN)
219  * RETURN        : なし
220  ******************************************************************************/
221  void CheckCsvData(int read_cnt)
222  {
223      FILE *read_fp = NULL;       /* 読み込みファイルポインタ */
224      FILE *write_fp = NULL;      /* 書き込みファイルポインタ */
225      int i ;                    /* ループカウンタ */
226      int j ;                    /* ループカウンタ */
227      int ret ;                  /* 戻り値 */
228      int continue_menu ;        /* 続行メニュー選択番号 */
229      int element_cnt ;          /* 1行の要素数 */
230      unsigned char buf[READ_SIZE + 1] ;    /* ファイル読み込みバッファ */
231
232      /* 読み込むファイル数分ループ */
233      for ( i = 0; i < read_cnt; i++ )
234      {
235          /*********************************************/
236          /* 読み込みファイル確認                        */
237          /*********************************************/
238
239          /* 続行問い合わせループ */
240          while(1)
241          {
242              printf("\n----------------------------------------------------------------------\n") ;
243              printf("読み込むファイルは「%s」です \n", sRead_fname[i].fname) ;
244              printf("  チェック実行しますか？ \n\n") ;
245              printf("1. はい  2. いいえ\n") ;
246              printf(">");
247
248              scanf("%d", &continue_menu);
249              scanf("%*c") ;    /* バッファクリア */
250
251              if ( continue_menu == 1 )
252              {
253                  break ;    /* ループ抜ける */
254              }
```

Copyright © 2018 TechMatrix Corporation. All rights reserved.

テストの優先順位を付ける - 未検証部分のチェック

未検証部分・簡易検証部分をどうするか？

限られた時間とリソースの中で、優先順位を付けてテストを行う



テストで検証できなかった部分はどうする？

時間をかけずにできる検証があれば実施したい

Copyright © 2018 TechMatrix Corporation. All rights reserved.

Parasoft C++test - フロー解析

静的テスト・動的テストを開発プロセスに組み込むことで開発の早い段階から品質への取り組みを行います。

	項目	機能	説明
静的テスト	● バグを検出	フロー解析	開発の初期段階で結合レベルのバグを検出し、手戻り工数を削減する。
	● コーディング規約違反を検出	コーディング規約チェック	バグの発生しやすい実装や言語の仕様、セキュリティの規約などをチェックし、ソースコードの記述を平準化する。
	● ソフトウェア品質を計測	メトリクス計測	メソッド数やサイクロマチック複雑度、オブジェクトの結合度などを計測し、品質の低いコードを検出する。
動的テスト	● テストケース、スタブの作成 ● カバレッジの計測	単体テスト支援	コーディングを行うことなく、テストケースとスタブの作成が可能。単体テストにおいて、最大の課題とされるコーディング作業を大幅に削減する。
	● 機能・結合テストの可視化 ● メモリ関連エラーの検出	アプリケーションモニタリング	アプリケーションを実行し操作した際に実行された処理・実行されなかった処理を計測することで、テストの網羅性や整合性を確認する。また、メモリ関連のエラーを検出する。

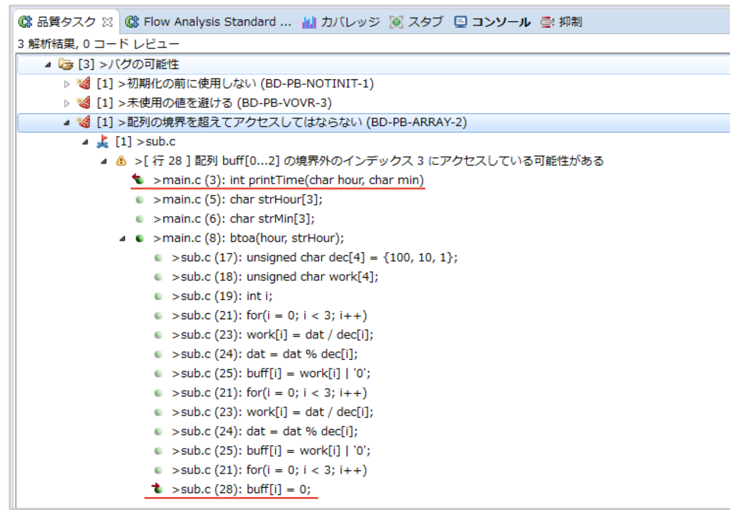
Copyright © 2018 TechMatrix Corporation. All rights reserved.

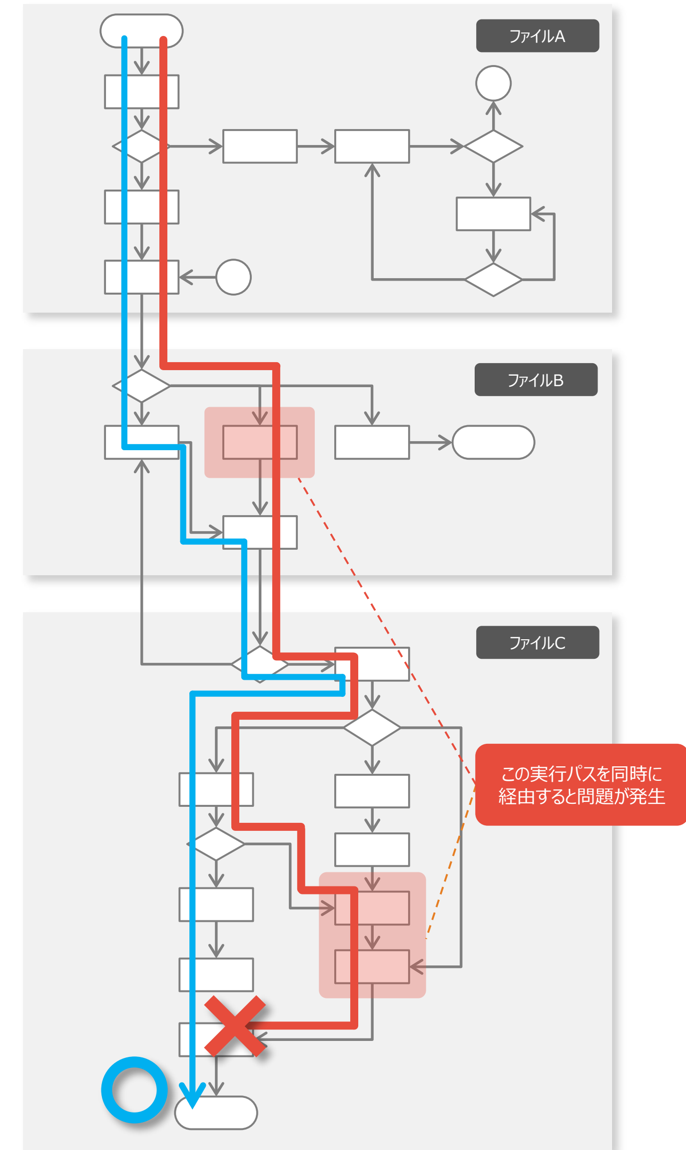
Parasoft C++test - フロー解析

モジュールのあらゆるパスをシミュレートし、バグを早期発見

- 見つかりにくいリソースリークなどの問題をソースコードを解析し、関数・ファイルをまたがったバグに至るまでの処理の流れをレポートします。
- すべての分岐の組み合わせを検証するため、人の手や目では発見しにくいバグを発見することができます。
- 主な検出可能な問題の種類
  - メモリリーク/リソースリーク
  - 未初期化変数の参照
  - 配列の範囲外アクセス
  - デッドロック/不適切な排他制御
  - バッファオーバーフロー
  - 整数オーバーフロー
  - イテレータ範囲外アクセス
  - NULLポインタの参照
  - ゼロによる除算
  - セキュリティ脆弱性

検出結果の例（関連する一連の処理の流れをレポート）



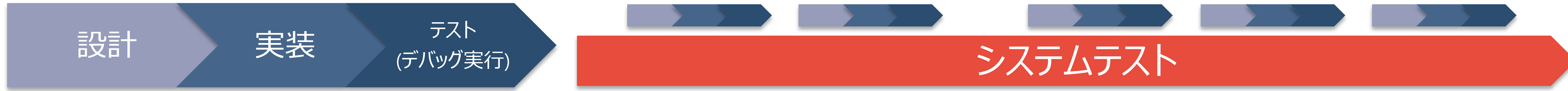


Copyright © 2018 TechMatrix Corporation. All rights reserved.

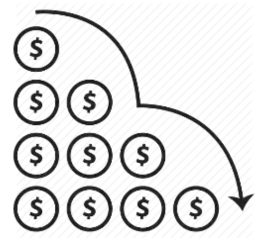
まとめ

まとめ

- 従来



- 対策適用後



保守・派生開発においても、**開発工程全体で品質を作り込む**ことができる
対策を行うことで部分的に見ると工数は増えるものの、全体では工数・コストを抑えることができる

Copyright © 2018 TechMatrix Corporation. All rights reserved.

Parasoft C++test

静的テスト・動的テストを開発プロセスに組み込むことで開発の早い段階から品質への取り組みを行います。

Copyright © 2018 TechMatrix Corporation. All rights reserved.

お問い合わせ先

テクマトリックス株式会社

システムエンジニアリング事業部
ソフトウェアエンジニアリング営業部