THE POSSIBILITIES ARE INFINITE FUJITSU

JaSST'10 Tokyo テクノロジーセッション資料

ソース解析ツールの活用による品質確保と全体品質の管理・統制方法

2010年1月28日

株式会社富士通ソフトウェアテクノロジーズ

目次

FUJITSU

FUJITSU

第１部 ソフトウェア品質の実態と課題

FUJITSU

1．1 ソフトウェアの品質の実態

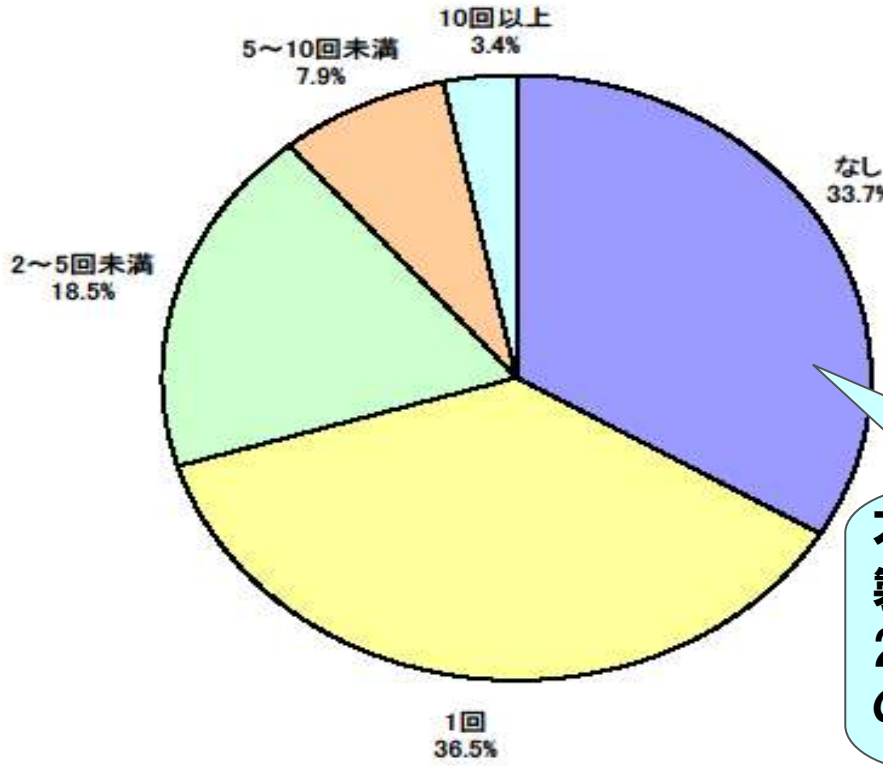
出荷後の製品品質

FUJITSU

Q8-1-5　1製品の不具合発生回数

METI

事業責任者



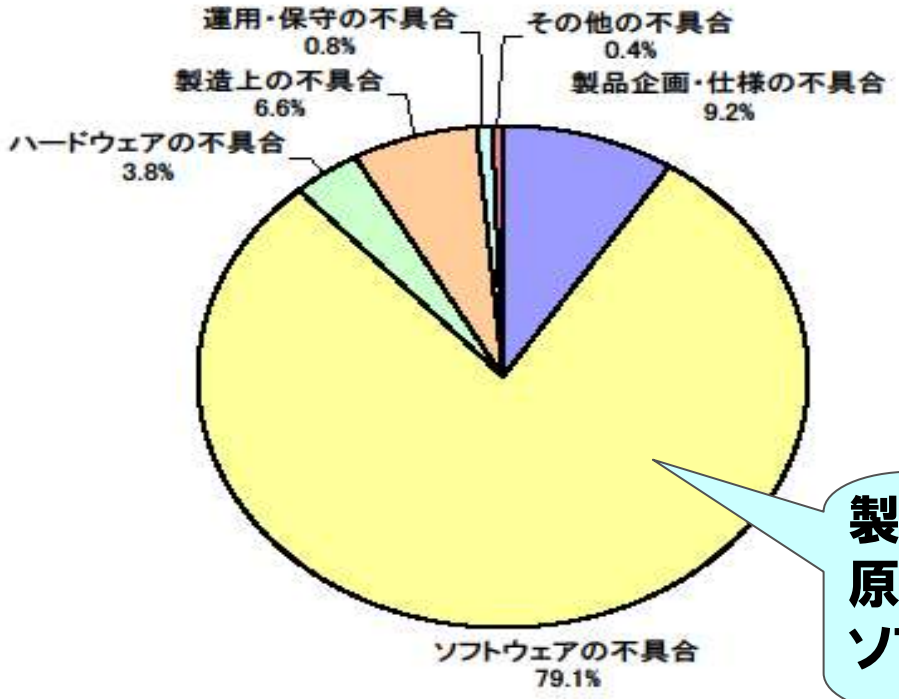
不具合を起こしていない製品は1/3だけ、2/3の製品で1回以上の不具合を起こしている

製品不具合の原因

FUJITSU

Q6-2 製品出荷後の不具合原因比率

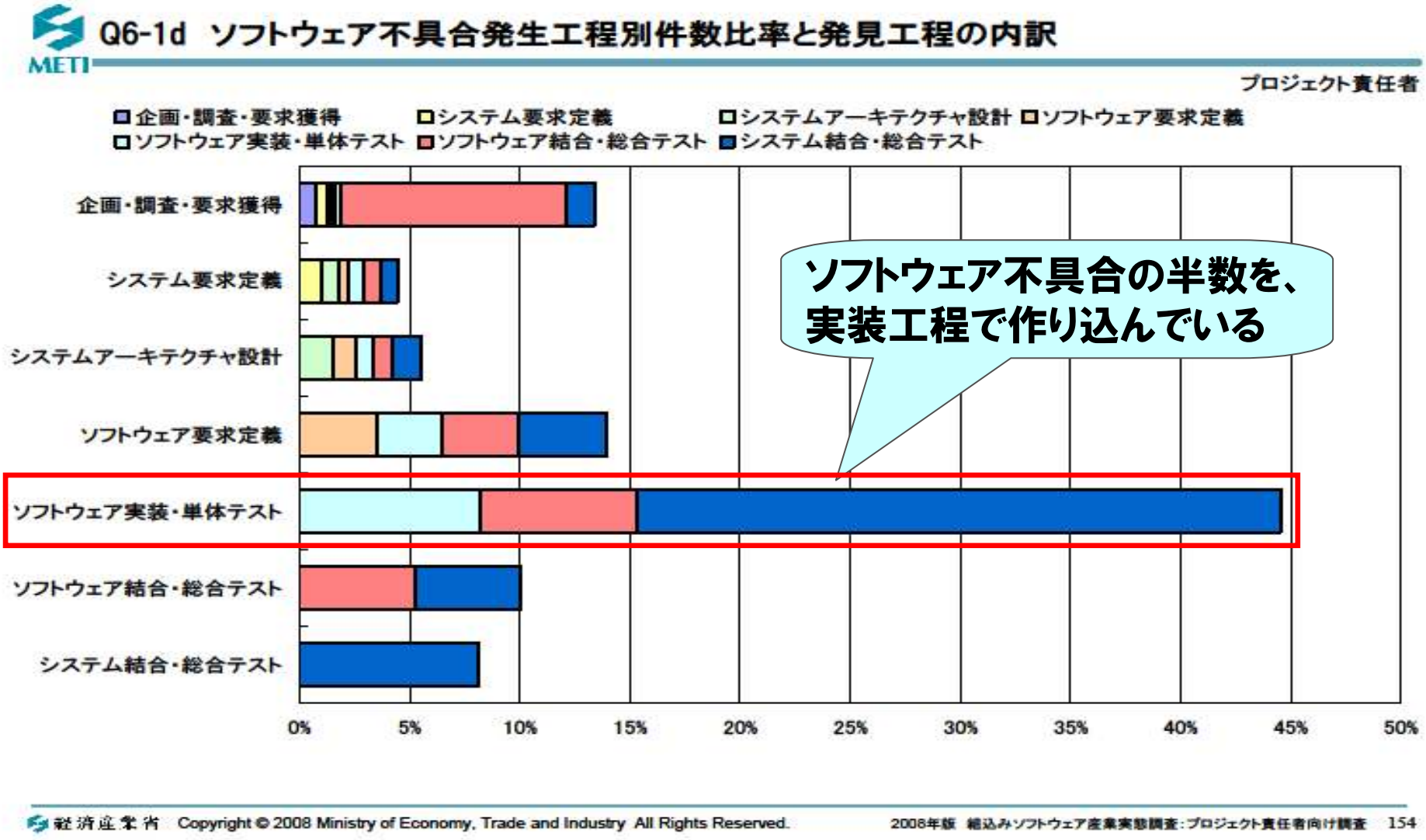
プロジェクト責任者



製品出荷後の不具合原因の8割は、ソフトウェアの不具合

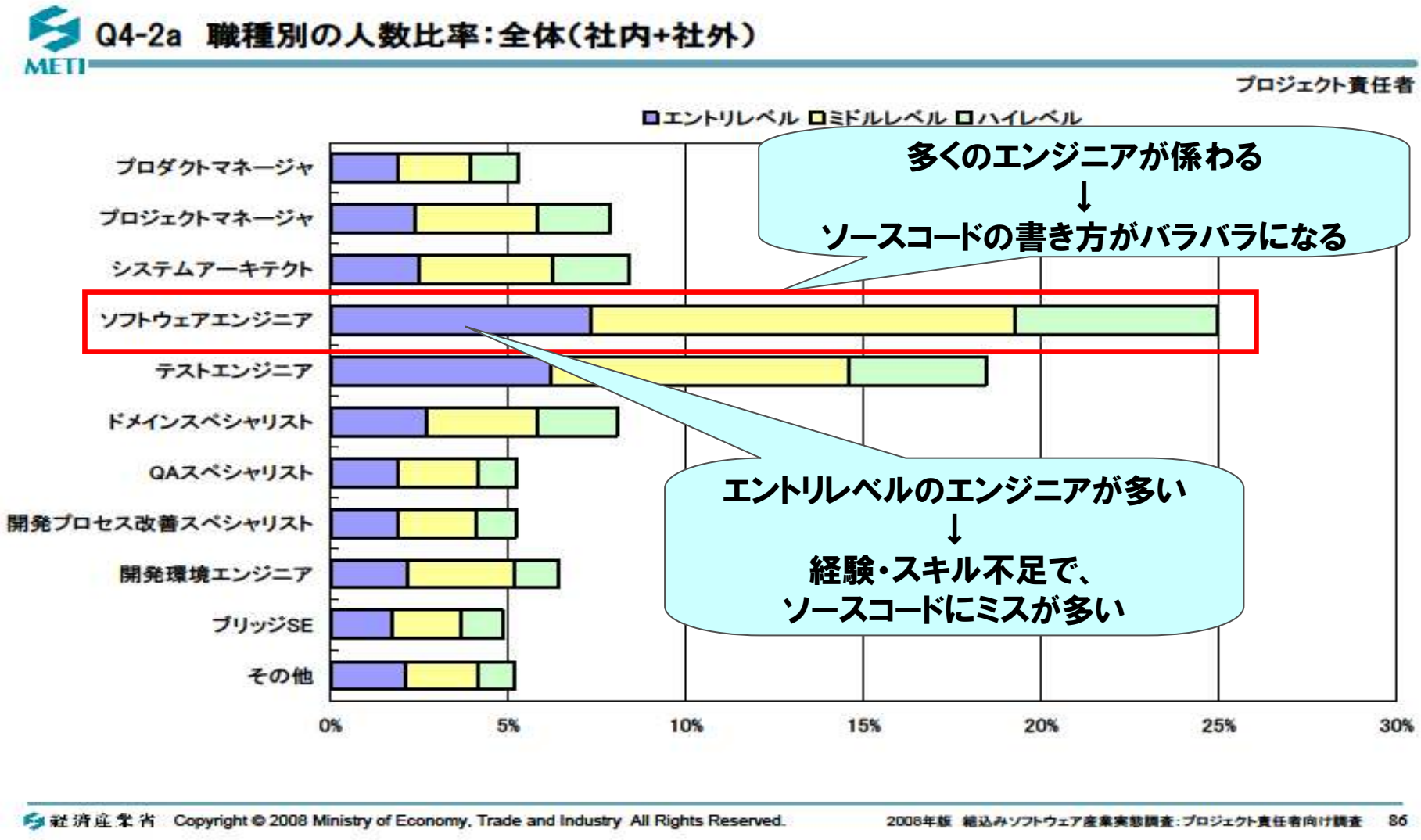
ソフトウェア不具合を作り込む原因工程

FUJITSU



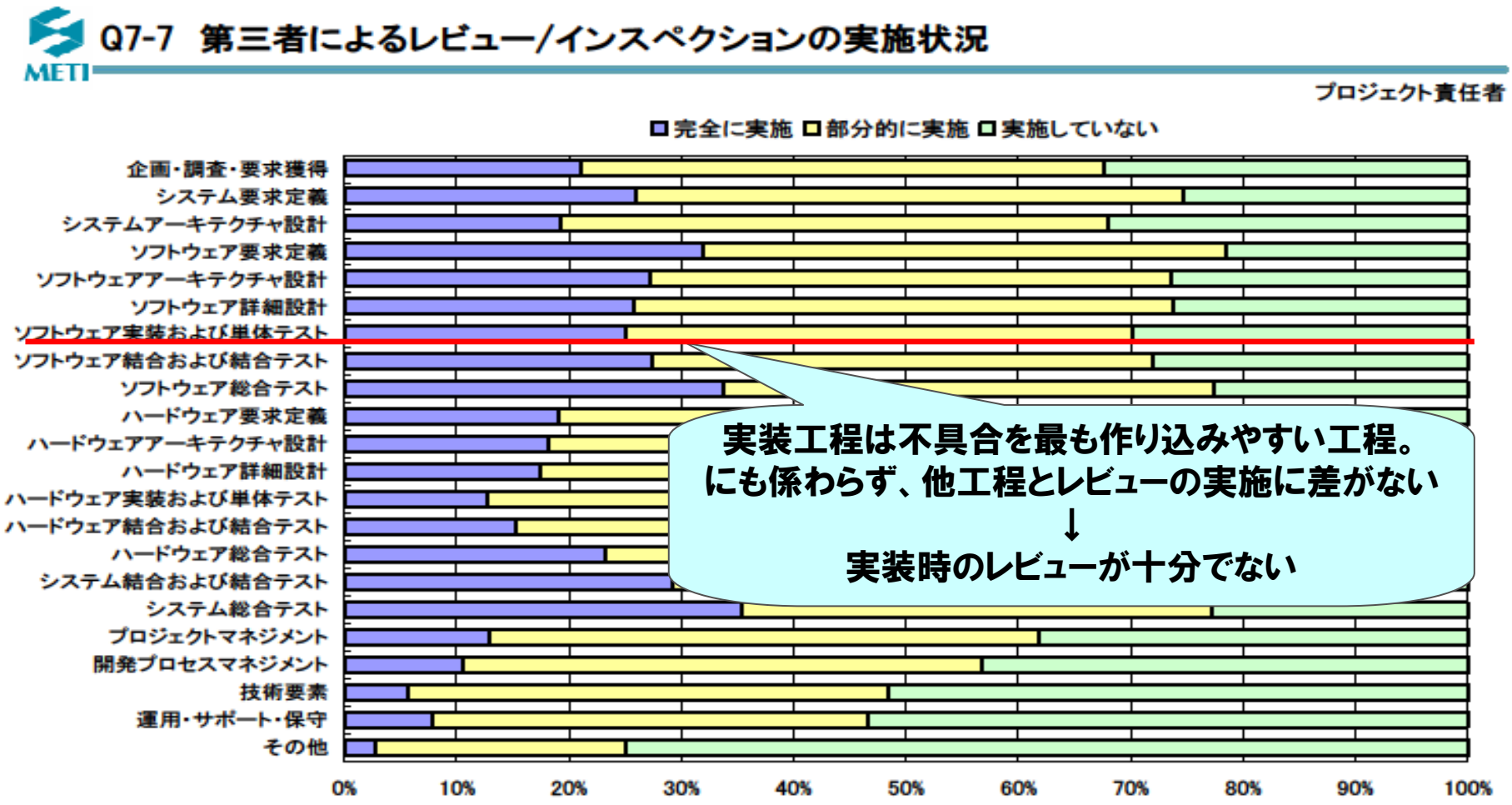
実装工程での問題(1/2)

FUJITSU



実装工程での問題(2/2)

FUJITSU



FUJITSU

1．2 品質確保のための課題

実装工程での問題と解決策

FUJITSU

問題点：

- ソースコードにミスが多い
- ソースコードの書き方がバラバラになる
- ソースコードのレビューが十分でない

課題：

- 経験者のノウハウの活用が必要
- レビュー効率化のために、統一した書き方が必要
- レビューを十分に行うルールや体制が必要

解決策：

- 経験者のノウハウや統一した書き方を規定したコーディング規約の作成
- コーディング規約の遵守をチェックする体制づくり

補足：実装工程での問題は、製品要求、設計、開発環境等にも起因するものがありますが、
今回は「ソースコードの作成」に着目しています。

FUJITSU

第2部 プログラム品質の確保とソース解析ツールの活用

FUJITSU

2．1 プログラム品質確保へのアプローチ

1）コーディング規約のあり方

コーディング規約の実態

FUJITSU

コーディング規約はあるが、運用が形骸化しているプロジェクトが多いのでは？

- 命名や書式に、こだわり過ぎている
- 守れないルールになっている
- チェックしづらいルールになっている
- 昔に作成したままで、現在の開発に合った更新がされていない
- 全社ルールのままで、個々のプロジェクトに合っていない

現状のコーディング規約は、遵守してもソースコードの品質向上につながらない

コーディング規約の目的

FUJITSU

① 要件・仕様との整合
プロジェクトの要件や設計仕様をプログラム中に具現化するため

② 信頼性・可読性・保守性の向上
プログラムの書き方を規定することによって、プログラムの信頼性・可読性・保守性を向上するため

③ スキル移転
コーディング規約の中に推奨する書き方や注意事項を記載することによって、経験の浅い開発者のスキルを向上するため

コーディング規約が保持すべき内容

FUJITSU

① 適用範囲などの全体情報
本コーディング規約の適用範囲を規定した情報、管理者情報
および改版履歴などの全体情報を記載する。

② コーディング規約・作法
命名規約やプログラムスタイルなど、プログラムを作成する際の
決め事を記載する。

③ 推奨・注意事項
推奨する書き方や誤りを未然に防ぐための注意事項を記載する。

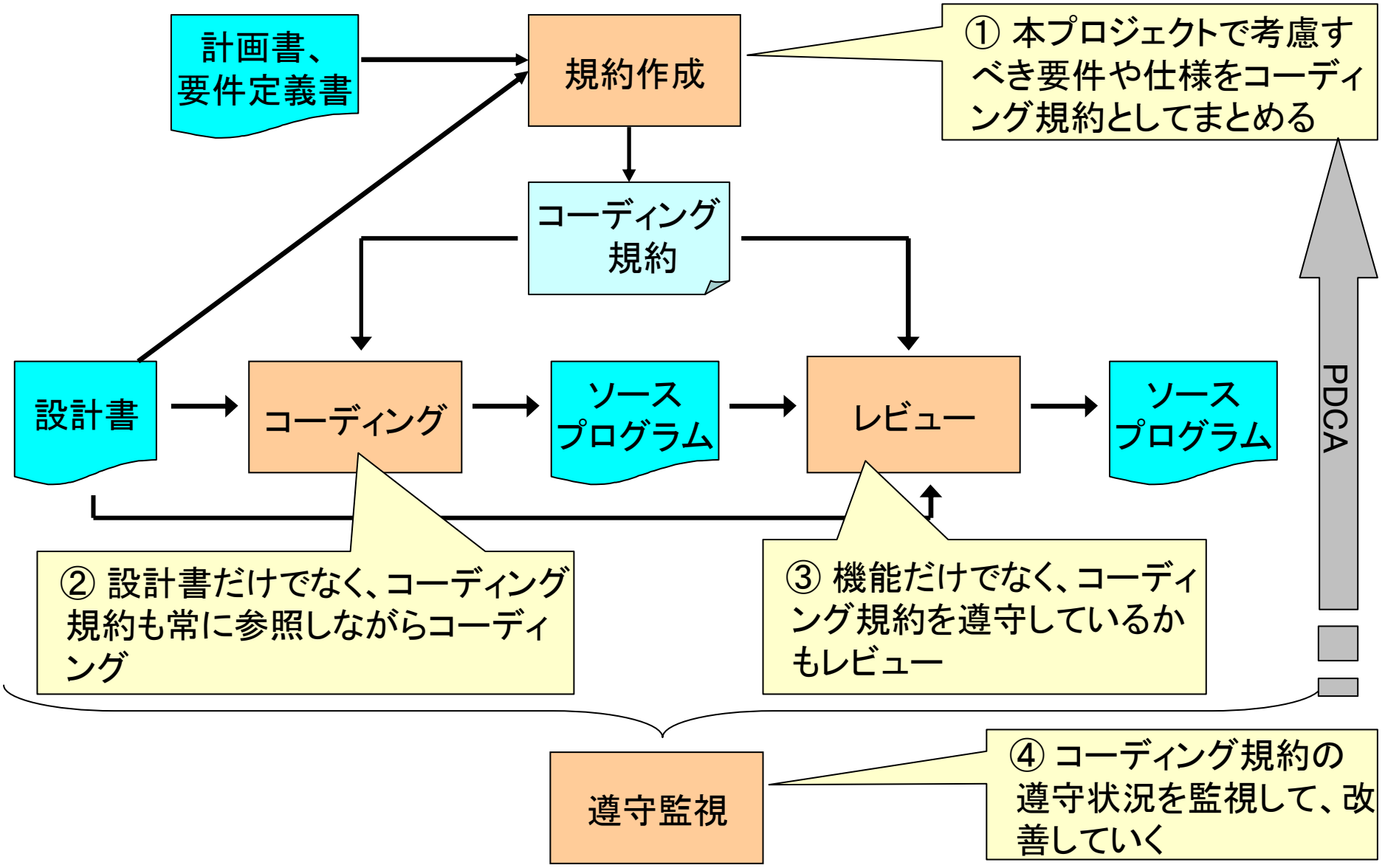
書き方が統一されたソースコードの利点

FUJITSU

書き方	利点
➢名前の付け方 ➢式や文の書き方	ソースコードを理解しやすい ➢処理を誤解しないため、誤った修正を防止できる。 ➢レビューが効率化され、バグ検出に集中できる。
➢ライブラリの使い方 ➢関数インタフェース	ソースコードを修正しやすい ➢処理がまとまっているため、修正漏れを防止できる ➢インタフェイス変更が容易のため、類似コードが増えない。

コーディング規約の適用

FUJITSU



FUJITSU

２．１ プログラム品質確保へのアプローチ

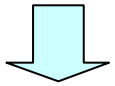
２）コーディング規約の作成

コーディング規約の作成手順

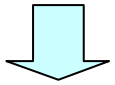
FUJITSU

①コーディング規約の作成方針の決定

②作成方針に沿ったルールの選択



③適用範囲の決定



④適用を除外するルールの選択

①作成方針の決定

FUJITSU

ソースコードがどう書かれるべきかの方針を、
プロジェクト要件から検討し、決定する

■製品特性

- 人命に係わる製品　⇒不具合発生時の被害を抑えているソースコード
- 24時間稼動製品　⇒異常停止させないソースコード

■開発体制

- 新人が多い　⇒バグ（過去の失敗）がないソースコード
- ベテランが多い ⇒引継ぎを意識した分かりやすいソースコード

■今後の展開

- カスタマイズ要求が多い ⇒機能拡張を考慮したソースコード

②作成方針に沿ったルールの選択（1/2）

FUJITSU

■既にコーディング規約がある場合

- 作成方針と照らし合わせて、ルールの変更・削除を実施
- 一般的なコーディング規約（※）と比較し、抜けているルールがあればルールを追加

■コーディング規約がない場合

- 作成方針に基づいて、一般的なコーディング規約（※）から、プロジェクトに合ったルールを選択

※一般的なコーディング規約（C/C++）

- IPA/SEC ESCR
- MISRA C/C++
- GNU Cogding Standards
- JSF Air Vehicle – C++ Coding Standards
- Effective C++、More Effective C++
- C++ Coding Standards

②作成方針に沿ったルールの選択（2/2）

FUJITSU

コーディング規約から、プロジェクトに合ったルールを選択

（例：IPA/SEC ESCR）

初期化漏れによる異常終了を防止したい

品質概念	作法	ルール	選択
信頼性	領域は、初期化してから使用する	自動変数は宣言時に初期化する。または使用する直前に初期値を代入する。	○
		const型変数は、宣言時に初期化する。	○
	初期化は過不足無いことがわかるように記述する	要素数を指定した配列の初期化では、初期値の数は、指定した要素数と一致させる。	○
		列挙型のメンバの初期化は、定数を全く指定しない、すべてを指定する、または最初のメンバだけを指定する、のいずれかとする。	×
保守性	構造化プログラミングを行う	(1)goto文を使用しない (2)goto文は、多重ループを抜ける場合とエラー処理に分岐する場合だけに使用する	○ (2)を選択
		continue文を使用してはならない	×

ソースコードを読みやすくしたい

③適用範囲の決定

FUJITSU

開発スタイルに合わせたルール決めが肝要。

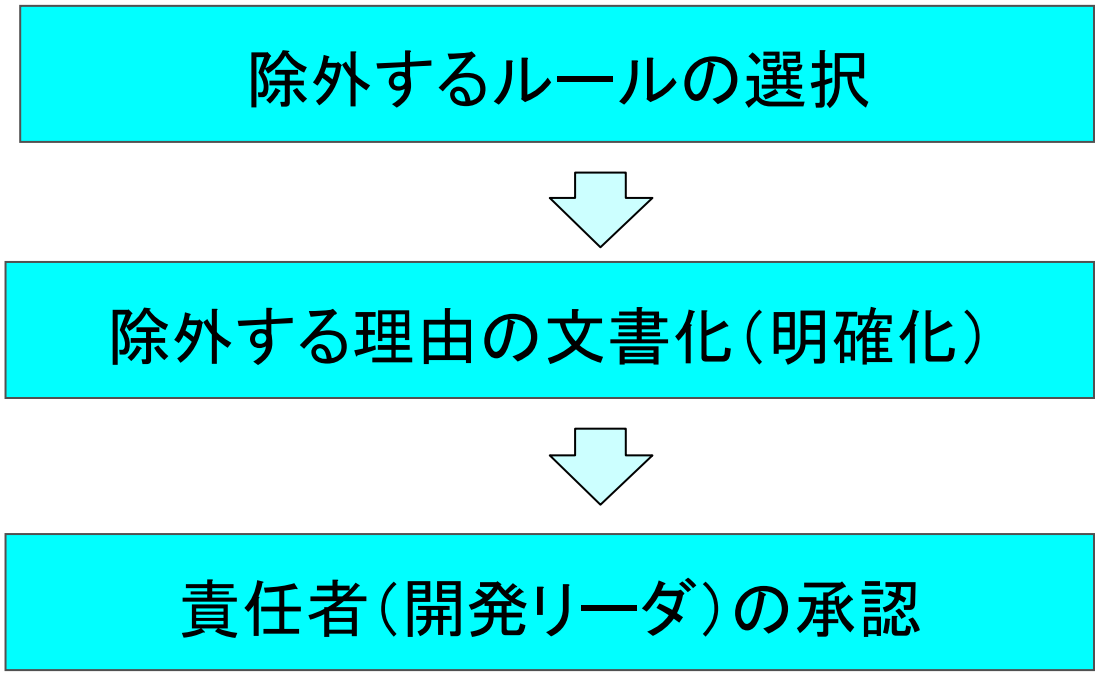
開発スタイル		適用方針
新規	スクラッチ開発	コーディング規約をしっかり決め、静的解析ツールをフルに利用します。
	流用開発	流用ソースのコーディングスタイルに注意。 規模大 ⇒ 流用ソースに従う 規模小 ⇒ 新規にスタイルを策定
保守	改造開発	スタイル系ルールは適用しない。 既存ソースを一括解析し、バグ検出系ルールへの対応方針を決める。 以降、改造部分のみのバグ検出系ルールに対応していく。

④適用を除外するルールの選択

FUJITSU

同一製品でも機能により、ソースコードに求める形が異なる。
機能単位に、ルールのカスタマイズを実施。

[手順]



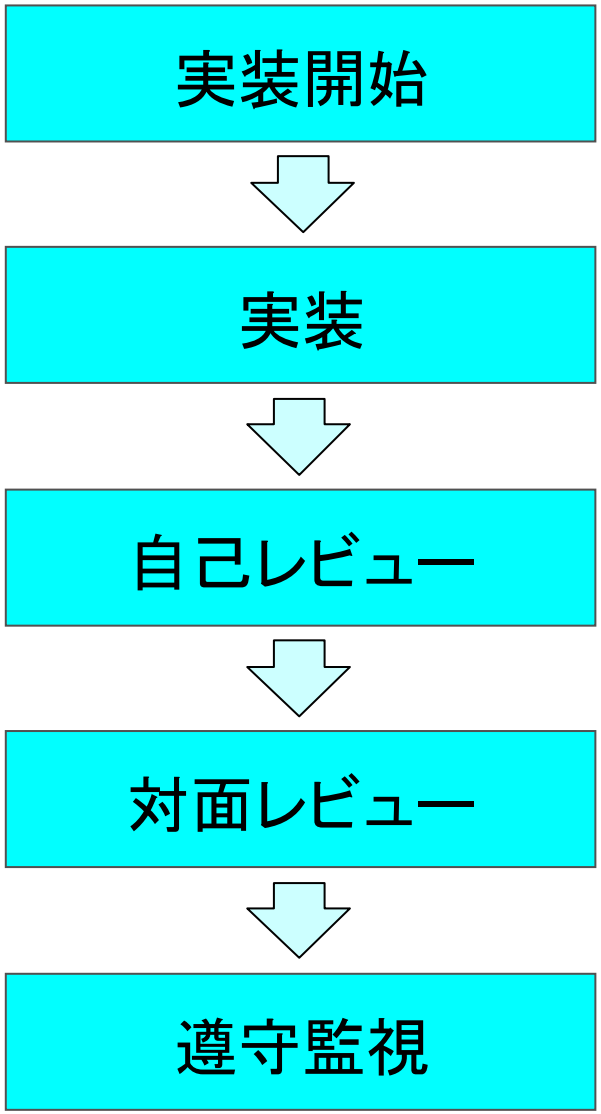
FUJITSU

2．1 プログラム品質確保へのアプローチ

3）コーディング規約の運用

運用方法

FUJITSU



- コーディング規約を理解する
- 実装者は、コーディング規約に沿って、ソースコードを作成する
- 実装者は、自身が作成したソースコードに、コーディング規約違反がないかチェックする
- レビューアは、実装者が作成したソースコードに、コーディング規約違反がないかチェックする
- リーダーは、開発者がコーディング規約違反を遵守しているかを監視する

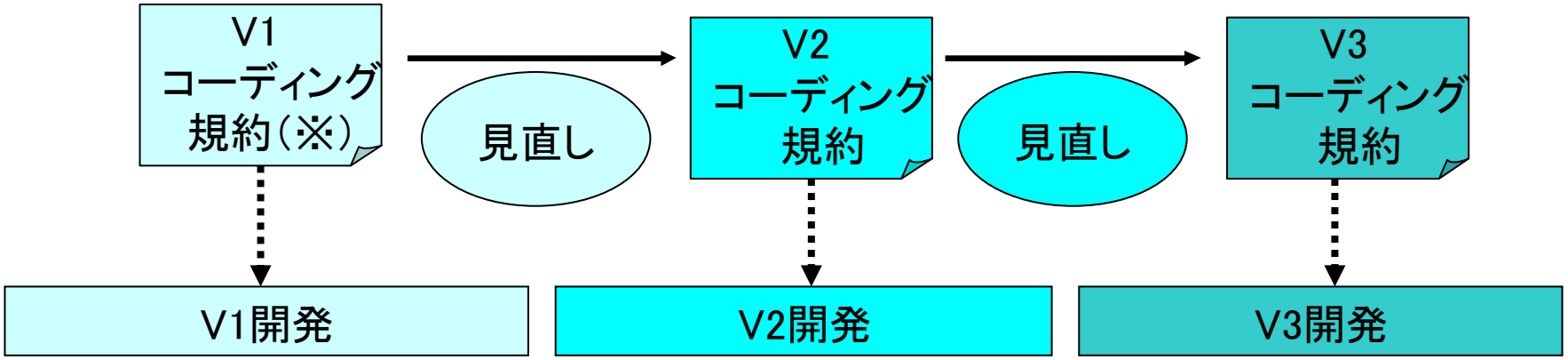
コーディング規約の充実

FUJITSU

コーディング規約の振り返りを実施し、次回への見直しが重要

振り返り項目:

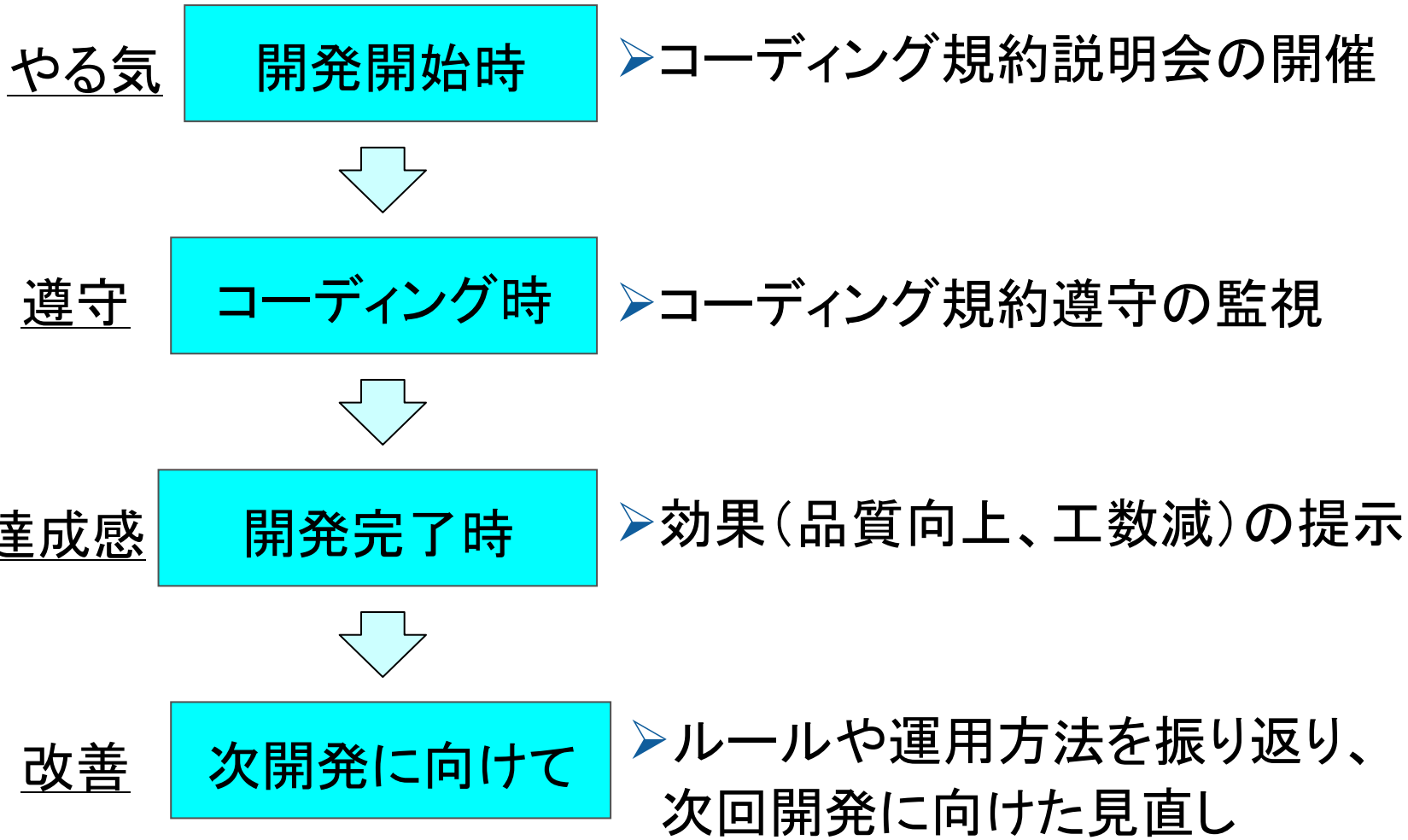
- 効果が低いルールや扱いにくいルールはないか
- テスト工程で検出されたバグをコーディング規約で検出できないか
- 運用時に負担に感じた作業はないか



※アドバイス:
初回のコーディング規約は、効果が高いルールのみを運用しましょう。

開発者に定着させる施策

FUJITSU



※定着には、推進役(開発リーダーや品証の方)の活動が重要です

FUJITSU

2．1 プログラム品質確保へのアプローチ

４）事例

A社様の事例（１／２）

FUJITSU

「実施タイミングの早さが負担とミスを防ぐカギ」とF社様。

開発者に負担を掛けず、やる気を起こす工夫を実施。

■工夫した点

- WGで選択ルールを協議、
  その際「ルール違反数／コード量」を測定し、対応可能なルールへ調整
- 導入モデルプロジェクトに、実現確率が高いプロジェクトを選定
- 「修正ファイルの違反ルールは、新規・既存に関係なく対処する」ルールで運用
- 推進役が、週１回、残違反ルール数を開発者へ広報
- 推進役が、よく発生するルール違反の修正方法を広報
- 推進役が、「開発時メトリクス」と「開発者アンケート」で効果を広報

A社様の事例(２／２)

FUJITSU

■効果

- コーディング効率が、キロステップあたり約2時間短縮
- コードレビューにおける問題点の指摘率が30%軽減
- 潜在的なバグを5000件以上消し込み
- 全テスト段階における問題点の指摘数を22.8%削減

2．2 ソース解析ツールの活用

1）活用メリット

ソース解析ツールのメリット

FUJITSU

人手によるコーディング規約のチェックは効率が悪く、漏れが発生しやすいため、静的解析ツールを活用する。

ソース解析ツールの活用による利点：

- チェック時間を短縮
- チェック漏れ（自己レビュー時、対面レビュー時）を防止
- チェック精度のバラつきを防止
- 違反数の把握が容易

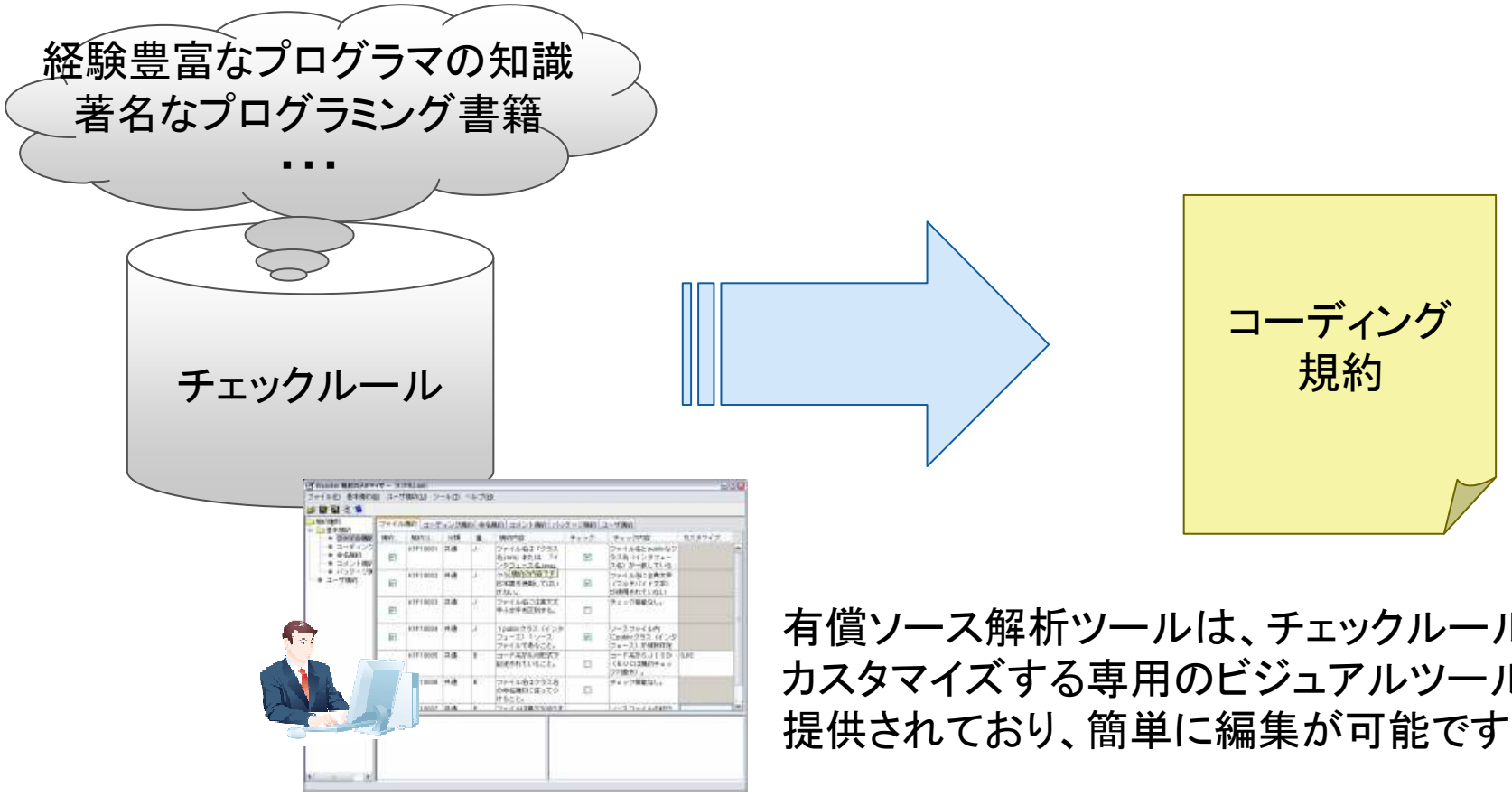
ソース解析ツールを活用した運用方法の改善：

- 自己レビューは、静的解析ツールでコーディング規約のチェックを実施
- 対面レビューは、機能漏れや実現方式についてのレビューに専念
- 遵守監視は、ソース解析ツールの結果を集計することで把握

コーディング規約の作成

FUJITSU

ソース解析ツールで用意されているチェックルールには、多くのプログラミング・ノウハウが詰まっている。



有償ソース解析ツールは、チェックルールをカスタマイズする専用のビジュアルツールを提供されており、簡単に編集が可能です。

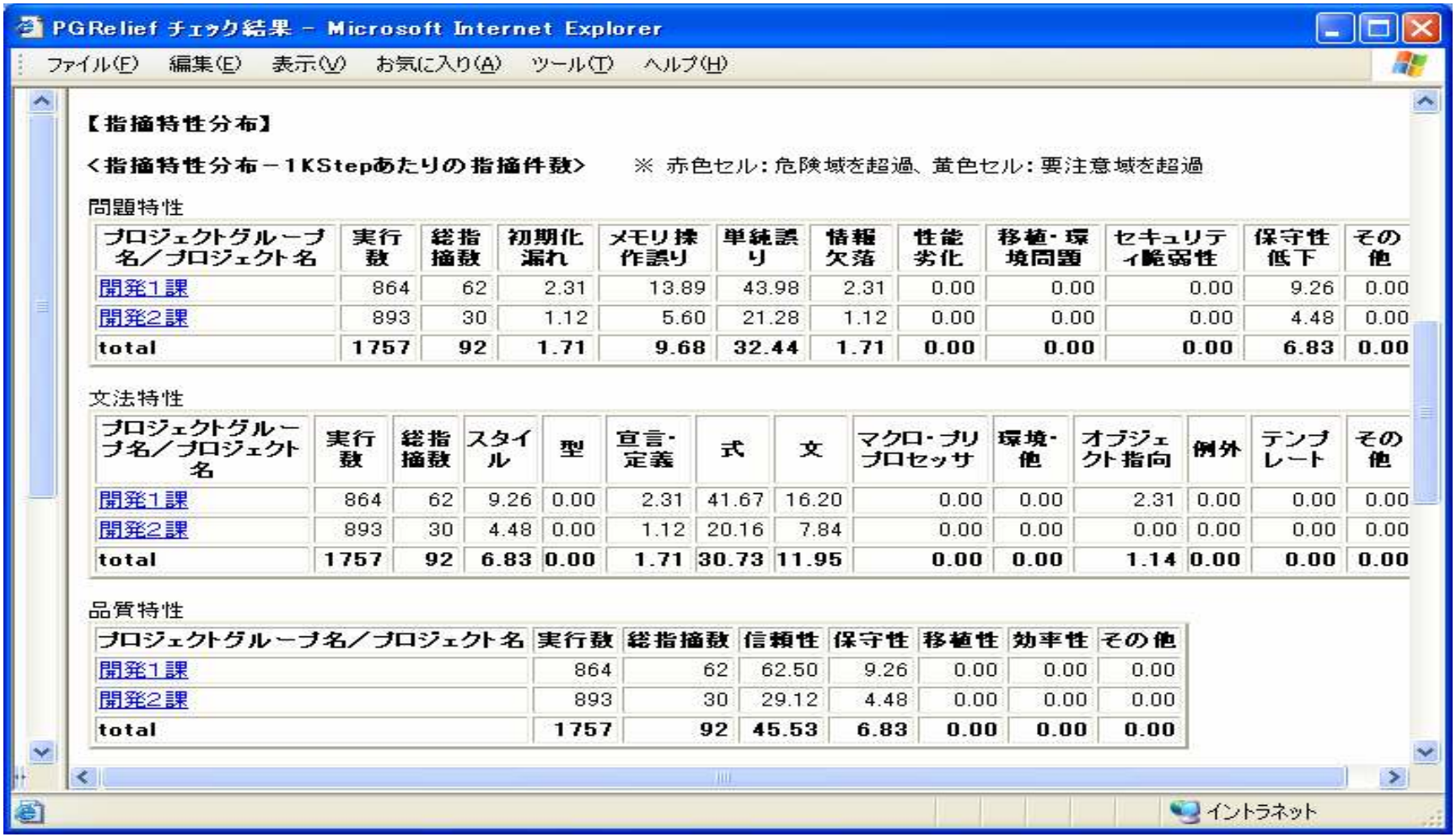
コーディング規約の運用

チェックルール（コーディング規約）を共有しながら、確実なコーディング規約の運用を可能にし、また、遵守・監視が低コストで実現。

品質状況の監視

FUJITSU

コーディング規約への遵守結果から、それを集計・分析することで現状の品質状況が容易に確認できる。

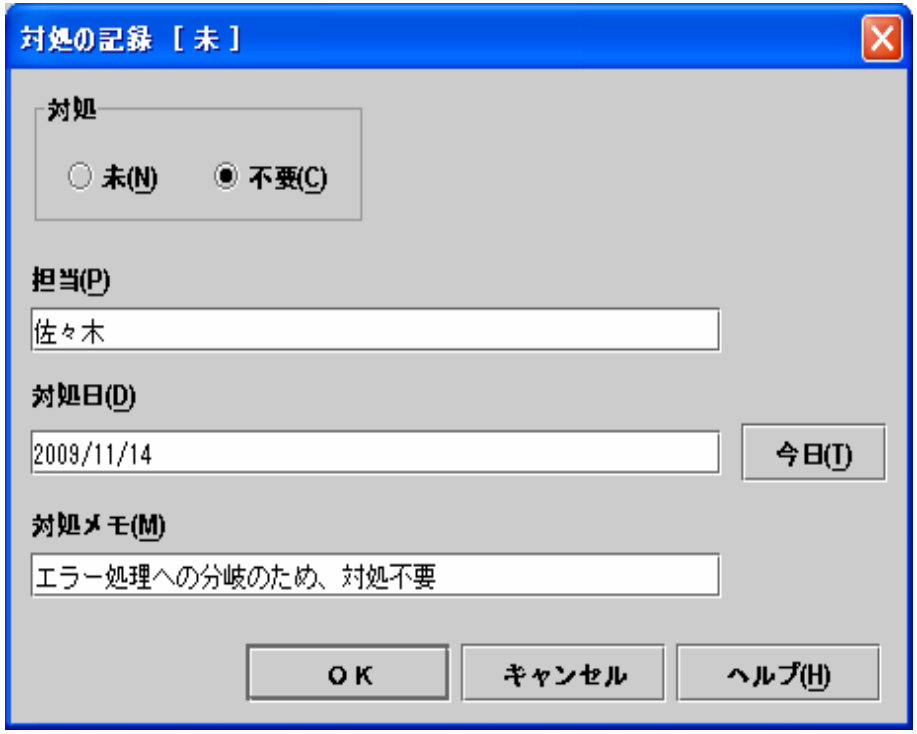


修正不要なメッセージへの対処

FUJITSU

ソース解析ツールには、次回以降の実行時からメッセージを抑止する機能が存在する。修正な不要な場合に使用すると便利。

①GUIで設定する方法



②ソースにコメントとして設定する方法

```
                    :
pfOpneFile = fopen( szFName,"R") ;
if( pfOpenFile == NULL ) {
      nErrCode = OPENERR;
      goto L_ERR; /* M3.1.2 */
}
                    :
L_ERR:
                    :
```

FUJITSU

2．2 ソース解析ツールの活用

2）活用アドバイス

ツール導入失敗パターン

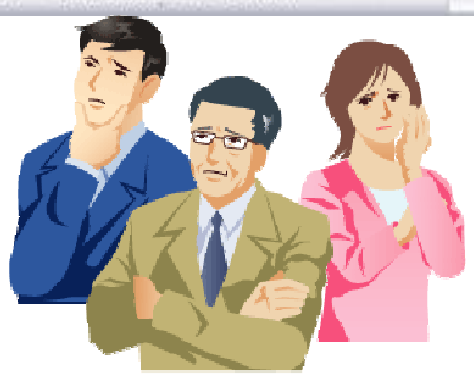
FUJITSU

「とりあえず導入してみようか！」では失敗します。

えっ！凄い指摘の数だ！

何で、こんなこと指摘するんだ！

これは、どう対処しますか？



結局、リーダも開発者もツールに
振り回されてしまう。

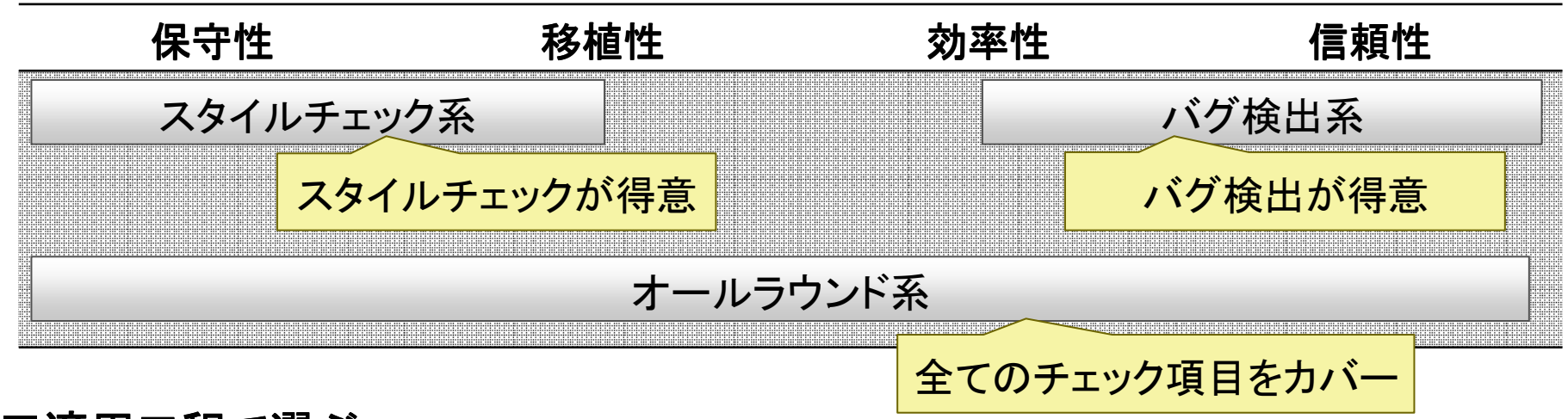
事前に「何のチェックをツールに任せるか（コーディング規約の作成）」と「どのように遵守・監視するか」を皆で合意することが重要。

目的にあったツールを選ぶ

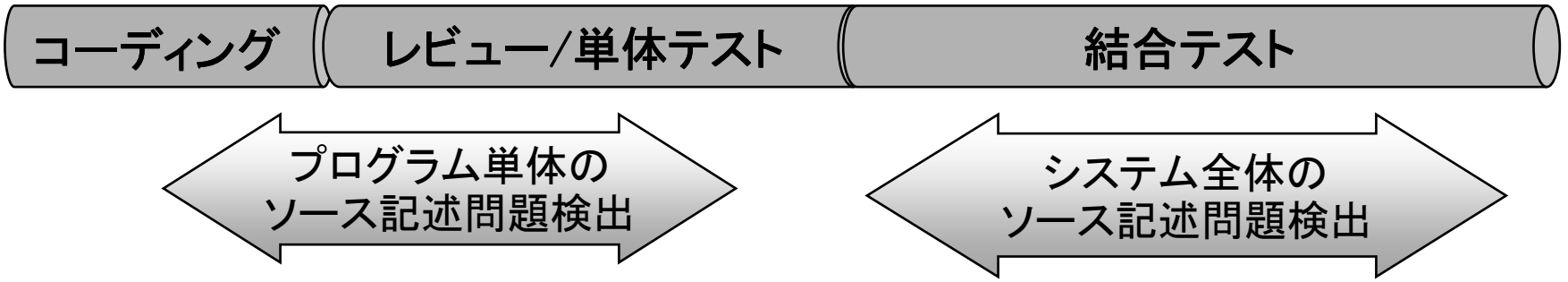
FUJITSU

ソース解析ツール達は、それぞれ特長を持っている。各ツールの特長を理解して導入すること。

■チェック観点で選ぶ



■適用工程で選ぶ



各プロセスにおける検出問題の明確化

FUJITSU

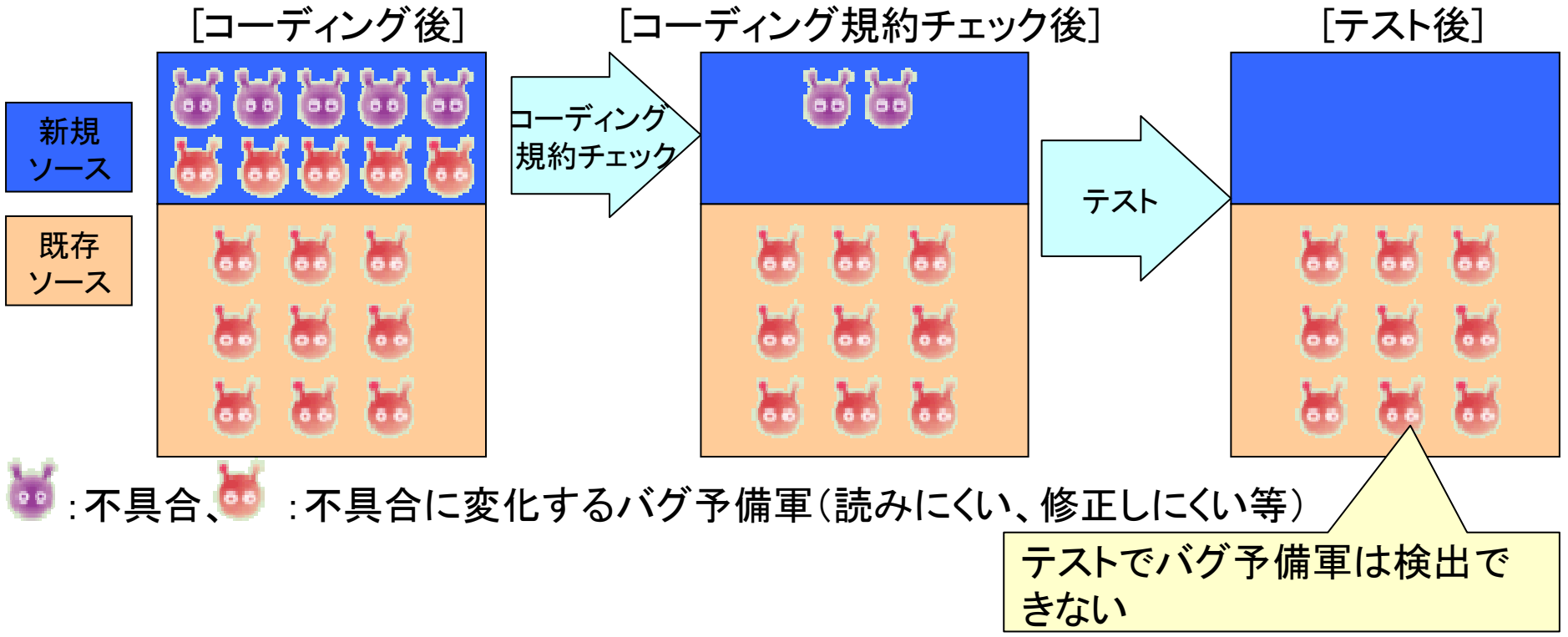
「ソース解析ツール」⇒「目視レビュー」⇒「動作テスト」の順で、重点的につぶす問題を明確にすること。

検出プロセス / 問題種類	ソース解析ツール	目視レビー	動作テスト
スタイル誤り	◎	×	×
単純誤り	◎	×	×
論理誤り	○	◎	×
インタフェイス誤り	×	○	◎
機能誤り	×	○	◎

既存ソースのバグをどうするか？

FUJITSU

既存ソースのバグ予備軍（潜在問題）は、いずれ不具合になる恐れあり。



バグ予備軍は、品質問題の火種です。
静的解析ツールは、新規も既存も関係なく検出します。
既存部までの対処を検討してみてはいかがでしょうか。

FUJITSU

第3部 プログラム品質の管理／統制

FUJITSU

3．1 管理／統制の必要性

サービスイン後のトラブルを未然に防ぐため

FUJITSU

システムのサービスイン後のトラブルを未然に防ぐには、単なるバグ取りだけでなく、「プログラム品質を見える化して、必要な対策を実施する」という、プログラム品質の管理／統制が必要

要件の変化：
ITシステムの安定稼動が社会的責任となり、開発プログラムの品質確保がますます重要になってきている

⇔

開発スタイルの変化：
プログラム開発の一括発注や過去資産の流用など、開発プログラムがブラックボックス化が増えている

ソフトウェア開発

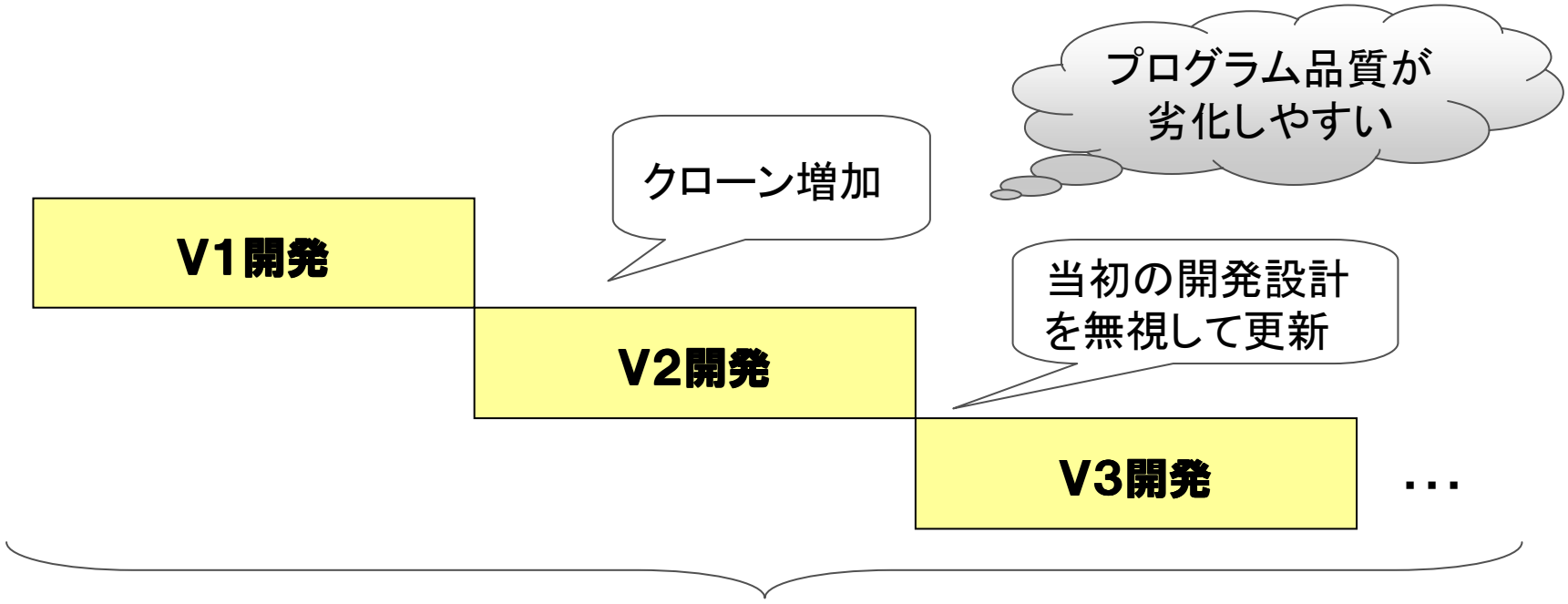
UI設計	構造設計	プログラミング	プログラムテスト	インタフェーステスト	システムテスト	運用テスト

品質の管理／統制が必要：
プログラム品質を見える化して、危険箇所に対して対策を実施する

品質劣化を把握するため

FUJITSU

V1, V2, …と複数世代に渡って開発が続く場合、各世代での改版に伴ってプログラム品質は劣化していく傾向にある。
そうした品質劣化は、保守コストの増加を招いたり、将来的トラブルの温床になったりするので、世代間での管理／統制が重要である。



FUJITSU

3．2 プログラム品質の評価と見える化

プログラム品質の評価

FUJITSU

■プログラム品質

プログラム品質は以下の6つの性質で構成される（JIS X0129-1）

機能性　使用性　信頼性　効率性　保守性　移植性

■プログラム品質の評価方法

以下の方法で開発プログラムに対する6つの性質を評価する

機能性	外部仕様とプログラムを突き合わせて十分性を評価
使用性	運用設計に基づくユーザビリティテストの結果を評価
信頼性	I/F違反や例外発生に対する処置などを評価
効率性	適切なAPIを使っているか冗長な処理がないかを評価
保守性	プログラムの文体や複雑度などを評価
移植性	システム依存や処理系依存のコードを評価

上記6つの評価結果をポイント換算する（換算方法は要件や特性で決定）

プログラム品質の見える化(１／２)

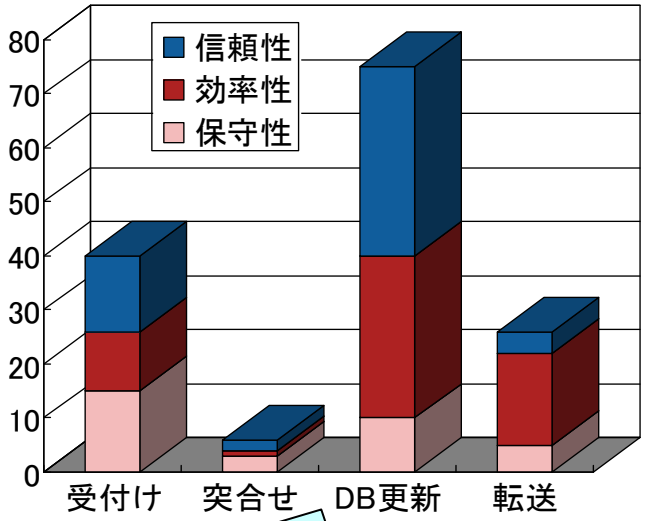
FUJITSU

プログラム全体を見える化したい観点でカテゴライズして、各プログラムの評価値を集計し、分析する。

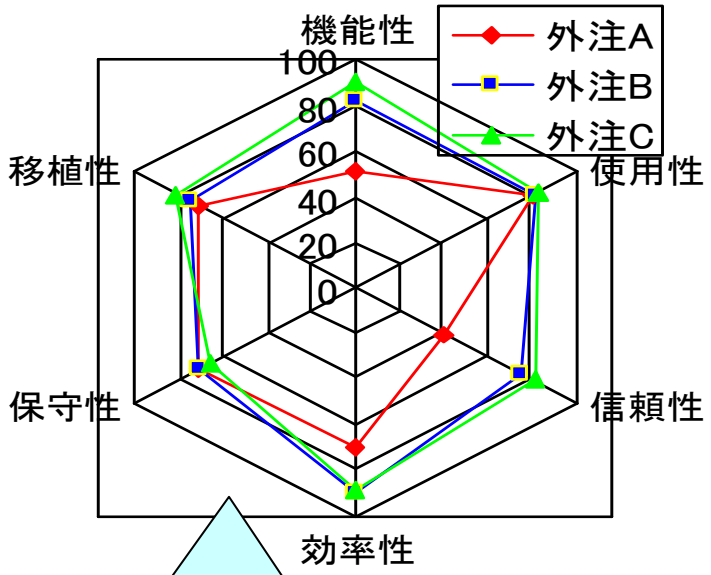
カテゴライズの例：

機能単位、開発メンバ単位、発注単位、新規開発/流用単位

品質分析の例：



品質が低い処理はどれか？
その理由は何故か？

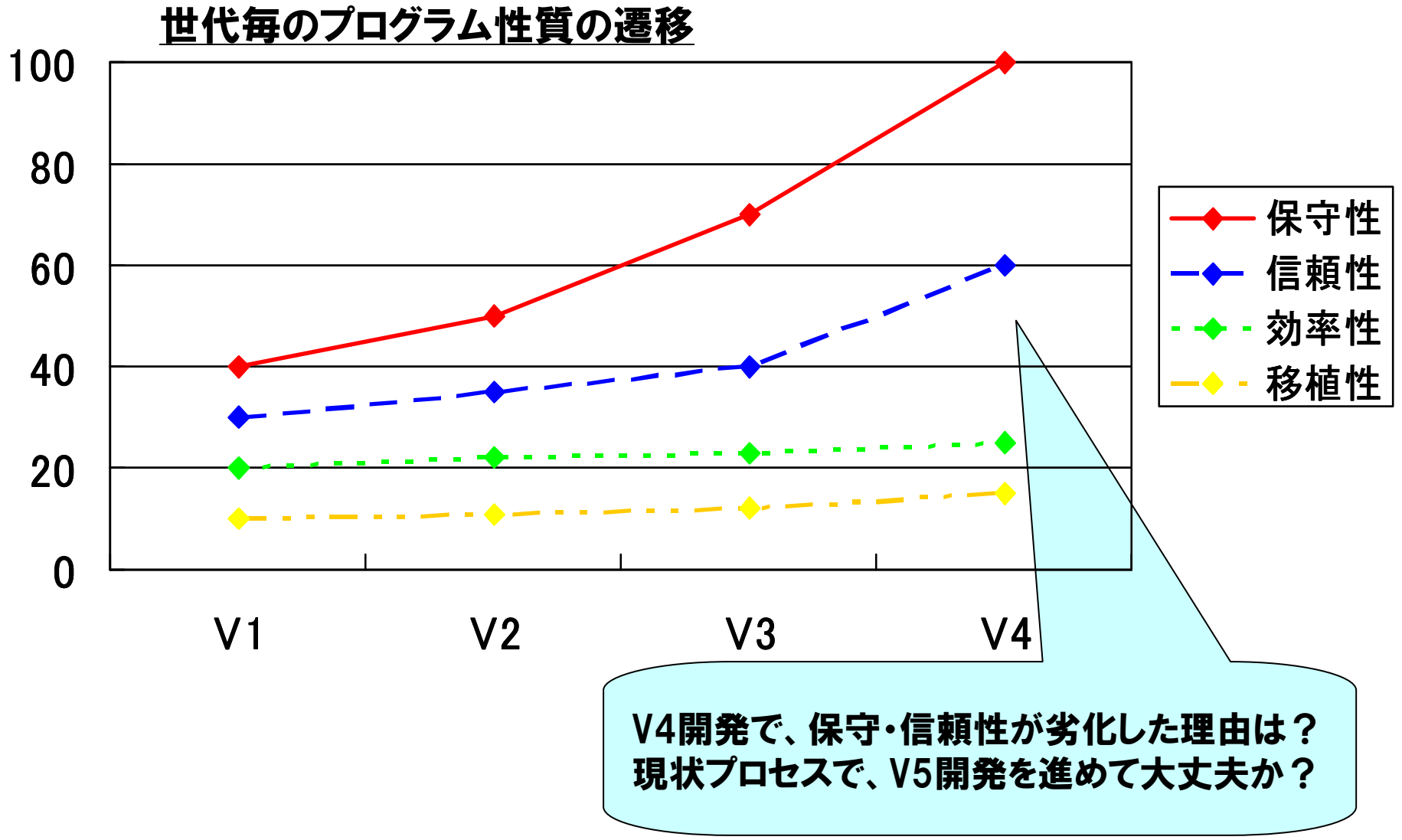


外注のスキルに問題ないか？

プログラム品質の見える化(2／2)

FUJITSU

品質分析の例:



ソース解析ツールの活用

FUJITSU

プログラム品質の評価は人手と時間がかかる。
また、人による評価はバラつきが出やすい。

⇒ 評価作業の効率化／平準化のためには、
ソース解析ツールを活用するとよい

＜ソース解析ツール活用のポイント＞

- ソースコード解析の特性から、品質の6つの性質のうち信頼性/効率性/保守性/移植性の4性質が評価可能
- ソース解析ツールの選定にあたっては、ツールに該当機能があるか否かを確認
- 自社内で評価作業ができない場合は、品質サービスなどの活用を検討

FUJITSU

3. 3 管理／統制の実施

管理／統制の実施

FUJITSU

■プログラム品質の管理

- ✓ プログラム品質は開発の節目ごとに定点観測し、品質見える化する
  例えば、各工程の終了時や納品受入れ時など
- ✓ V1, V2, …と何世代かに渡って開発が続く場合、各世代ごとの品質評価を
  を残しておき、世代間の劣化が確認できるようにする

■プログラム品質の統制

- ✓ 品質見える化によって、現在の開発プログラムに重大な欠陥が検出された
  場合、開発完了までに是正処置を講じる
  例えば、ある機能の再設計や他に類似問題がないかの見直しなど
- ✓ 次回開発で見直すべき課題について、次回開発での強化項目とする
  例えば、保守性に関する書き方統一や外注時に渡すべき資料見直しなど

第4部 製品・サービスのご紹介

FUJITSU

4. 1 PGReleifご紹介

PGReliefの特徴

FUJITSU

■ 高いヒット率

データフロー解析・ロジックフロー解析により、プログラム欠陥
の可能性を的確に指摘します。

■ 充実したドキュメント

各々の指摘メッセージについて、詳細な解説が用意されています。
簡単な例を交えて、そのメッセージが指摘された理由や対処例など
も記載されていますので的確に対処することができます。

■ 安心なサポート

開発およびサポートは、日本国内で行っています。
問題対応、お問い合わせの回答は、迅速に行っています。
また、ご要望もお聞きし、製品へ反映しています。

PGRelief C/C++ 関数間問題検出強化

FUJITSU

PGRelief C/C++ 2010(10年6月出荷予定)では、検出の範囲を関数間に広げ、関数間に潜んでいるメモリリークやバッファオーバーラン等を検出します。

関数：main

```
void main( ) {
    int array[10] ;
    sub( array );
}
```

10個の配列を渡す

関数：sub

```
void sub( int *p) {
   for( i=0 ; i < 50 ; i++ )
       p[ i ] = 0 ;
}
```

ﾊﾞｯﾌｧｵｰﾊﾞｰﾗﾝが発生！

チェック観点	性能
メモリリーク	△⇒○
バッファオーバーラン	△⇒○
0番地参照	△⇒○
初期化漏れ	△⇒○
データ欠落・データ型不整合	○
記述忘れ(break, return)	○
意味の無い演算、式、条件、文	○
副作用問題	○
命名規約違反	○
デットロック	×(2010以降)

今回強化

指摘例① メモリリーク(2010)

FUJITSU

例1

```
int* p;
void inip( ) { p=malloc( sizeof(int) * 10 ); }
void endp( ) { free(p); }
void proc()  {
    inip( ) ;
    inip( ) ;
    endp( );
}
```

PGRelief2010:
・pに格納されている資源が解放されない

例2

```
void endp( int * p ) {
    free(p);
}
void proc()  {
    int *p1 = malloc( sizeof(int)*10 );
    int *p2 = malloc( sizeof(int)*10 );
          :
    endp(p2);
}
```

PGRelief2010:
・p1に格納されている資源が解放されない

指摘例② 領域外アクセス(2010)

FUJITSU

例1

```
void array_sum( int * p ) {
    for( i = 0 ; i < 100 ; i++ )
        total += p[i];
}
void proc( ) {
    int array[10];
    array_sum( array );
}
```

PGRelief2010:
・array変数に対して、領域外アクセスが発生する

例2

```
int array[10];
int sub( int index ) {
    return array[index];
}
void proc( ) {
    for( i = 0 ; i < 100 ; i++ )
        total += sub(i);
}
```

PGRelief2010:
・array変数に対して、領域外アクセスが発生する

指摘例③ 0番地参照／初期化漏れ(2010)

FUJITSU

例(0番地参照)

```
void proc( int * p ) {
    int x = *p;
}
void func( ){
    int *p=NULL;
    int   i = 0;
    proc( p  );
    proc( &i );
}
```

PGRelief2010:
・proc関数内で0番地参照が発生する

例(外部変数の初期化漏れ)

```
int gnum;
void sub( ){
    int i = gnum ;
}
void main( ) {
    sub( );
}
```

PGRelief2010:
・外部変数gnumを初期化せずに参照している

4．2 ソース診断サービスご紹介

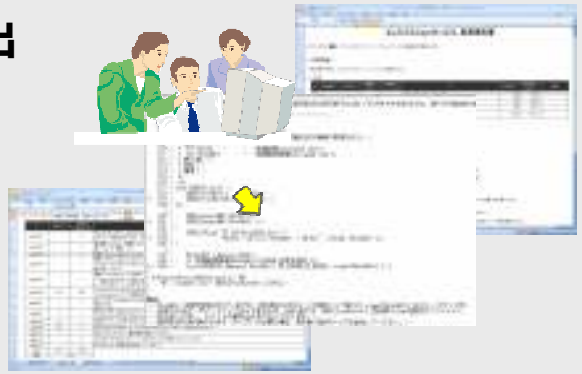
サービス実施内容

FUJITSU

静的解析ツール(主にPGRelief)と目視による調査

➢ ソースコードの問題点検出

- ✓ツールと目視調査でバグやバグの可能性が高い問題箇所を検出
- ✓コーディングルールで許容されているものは除外
- ✓バグの種類や重大度ごとに分析/評価し、改善案を提示



参考) ソースコードの問題点の具体例

➢ ソースコードの品質特性分析(オプション)

- ✓信頼性/保守性/移植性/効率性(性能)/機能性(セキュリティ)の観点で分析
- ✓お客様指定の観点(業務/チーム/コンポーネント/バグ種類)で分析
- ✓弱点や注力観点を洗い出し、改善案を提示

※)本サービスの対象は、C/C++およびJava言語で記述されたソースコードです

診断結果例(問題点検出)

FUJITSU

重大度を記号で表記
★:バグ
▲:バグの可能性
□:改善

問題点検出結果、問題傾向／評価

バグ箇所のソースを抜粋

バグの解説、改善指針を提示

バグおよびバグの可能性がある箇所の件数一覧

3. 診断結果

対象資産における指摘は以下の通りです。

a) 全体件数

指摘種別	指摘件数(件)	指摘率(件/ks)
バグ	325	0.24
バグの可能性	89	0.07
全体	414	0.31

評価)
総行数に対するバグ件数(0.24件/ks)は、PT完了時の弊社基準値(0.35件/ks)に対し0.69倍と低い検出率でし
しかし、有効行数のに対するバグ件数(0.41件/ks)は弊社基準値(0.44件/ks)と同等であり、標準的品質であ
考えられます。
また、レビューで指摘、修正されるべき問題が多く検出されましたので、レビューの強化を検討してください。

b) 観点別件数

No.	観点	バグ		バグの可能性		合
		件数(件)	占有率(%)	件数(件)	占有率(%)	件数(件)
1	メモリ、リソース関連の誤り	196	60.3	34	38.2	230
2	NULLポインタチェック漏れ	74	22.8	12	13.5	
3	変数の初期化忘れ	22	6.8	23	25.8	
4	意味のない演算、式、条件、文	21	6.5	0	0.0	
5	変数、演算子、型の使用誤り	11	3.4	9	10.1	
6	関数使用誤り	1	0.3	8	9.0	
7	データの欠落	0	0.0	3	3.4	
8	記述忘れ	0	0.0	0	0.0	
	合計	325	100.0	89	100.0	

評価)
・メモリ、リソース関連の誤りが、196件、60.3%を占めています。
メモリ操作(獲得、参照、破棄)について再確認してください。
また、上記のうち、new[] で獲得したオブジェクトを delete で破棄しているケースが64件
以下の原因が考えられますので、対処の検討をお願いいたします。
－レビュー観点不足
－delete[]により破棄することの忘却
－delete[]により破棄しなければならないことへの知識不足
・NULLポインタチェック漏れが、74件、22.8%を占めています。
NULL初期化後の値設定処理や、異常系の処理を見直してください。

```
★D:¥source¥c_src¥FZ1¥sub_sys1.c, 139, a, pgr0854, 配列形式の仮引数"RtnCdDtl"の大きさを求めるのは、誤りの可能性があ
- D:¥source¥c_src¥FZ1¥sub_sys1.c,, ----------
   106 :  /**
   107 :  * 関数名：SUBSYS_Init 初期処理 Seq:02
   108 :  * 機能概要：処理を行う上で必要な領域の初期化及び情報の取得を行う。
   109 :  * 出力パラメタ：
   110 :  *  (1) RtnCd           -   処理結果(unsigned char)
   111 :  *  (2) RtnCdDtl        -   処理結果詳細(unsigned char)
   112 :  * 戻り値：-
   113 :  * 例外：
   114 :  * 備考：
   115 :  *
   116 :  */
   117 : void SUBSYS_Init( 
   118 :      SB00_ProcRslt_t     RtnCd,
-> 119 :      SB00_ProcRsltDtl_t  RtnCdDtl 
   120 : ){
   130 :      SB00_AssertNN( RtnCd );
-> 131 :      SB00_AssertNN( RtnCdDtl );
   132 : 
   133 :      SB00_PTLog( "PT_02-01:LA3X0_Init "
-> 134 :                  "RtnCd = [%.1s], Rt      = [%.4s] ", RtnCd, RtnCdDtl );
   135 : 
   137 :      RtnCd[0] = Normal_50
   138 :      /* 処理結果詳細(R      tl)に0000:正常を設定 */
=> 139 :      (void)SUBSYS1_      cpy( RtnCdDtl, SX_FRPUBLIC_R0000, sizeo
- d:¥source¥include¥s      _sys1.h, 49
    49 : typedef char  SB00_ProcRsltDtl_t[(4)];

解説
- RtnCdDtl は配列宣言されていますが、仮引数のためポインタ変数として扱わ      of(RtnCdDtl) はポインタサイズが
返却されます。ポインタサイズは８であるため、139行目では、SB00_ProcRslt      型変数＝char型配列(サイズ＝４)
RtnCdDtlに対し、８バイトの書き込みを行なってております。
配列形式の仮引数の大きさ（サイズ）が必要な場合、仮引数に配列サイズを      してください。
```

No.	指摘カテゴリ	★+▲	ID	★	▲	指摘メッセージ例
1	メモリ、リソース関連の誤り	230	per0013	3		404行目のfor文で"i"には16がはいるので、"array_01[i]"で配列の範囲を超えます。
2			per0026	2		自動変数のアドレス"(pt_work)"を、関数の復帰値にするのは誤りです。
3			per0522	1	30	800行目で資源の回収をした変数"temp_shop_no"を参照している可能性があります。
4			per0524	8		1200行目の関数"fopen"で取得した資源が回収されていない可能性があります。
5			per0528	3		1500行目で代入した静的な資源"M"003""を演算子"delete"で回収しようとするのは誤りです。
6			per0532		3	strcpy(wk_pj.name, t_el_name)は領域"wk_pj_name"を超えてコピーしてしまう可能性があります。
7			per0548	9		memcpy(&shop, add_data->ShopNo, sizeof(add_data->Shop_No))は領域"shop"の範囲を超えてアクセスしてしまう可能性があります。
8			per0570	3		関数"sprintf"は、8バイトの"buff"に9バイトを入れます。
9			per0840	64	1	415行目の"new"で生成した資源が破棄されていない可能性があります。
10			per2225	74		new[]の形式で生成した変数"buff"を、別の形式deleteで破棄しています。
11			per2238	29		関数"memcpy"の引数に"class OptCls"型を渡しています。
12	NULLポインタチェック漏れ	86	per0060	8	12	1600行目で0番地を代入している変数"in_name"は、0番地を参照している可能性があります。
13			per0520	6		1250行目で0番地を代入する可能性のある変数"pt_work"は0番地を参照する可能性があります。
14			per0689	60		6105行目で0番地と比較している変数"msg"は、0番地を参照する可能性があります。
15	変数の初期化忘れ	45	per0000	16	23	変数"code_num"は参照の前に値の設定がない可能性があります。
16			per0039	6		変数"code_user_num"は参照の前に値の設定がありません。
17	意味のない演算、式、条件、文	21	per0017	5		復帰値"(idx_num++)"でのインクリメントやデクリメントは意味がありません。
18			per0037	1		i < 0 && i > 2は、決して真になることがありません。
19			per0040	3		条件式"(*err_No()) = 2"は意味がありません。=は == の誤りの可能性があります。
20			per0575	4		str_count > 10は、"UpdateFile.cpp"の1200行目に同じ条件式があるため、決して真になることがありません。
21			per0695	7		0は何もしない式です。
22			per1222	1		Error_code == FileDataException::ERROR_CODE_200 || Error_code == FileDataException::ERROR_CODE_200に、意味が無い条件式があります。

診断結果例(品質特性分析)

FUJITSU

［評価］

●全体
指摘率は標準値比83.8%と少なく、全体の品質は良いと言えます。
ただし、信頼性の指摘率は標準値比107.6%と若干多くなっていますので、信頼性に着目した見直しを推奨します。

業務ごとでは、「カード決済
回っていました。「カード決済
保守性、移植性の指
り、他業務と比較しても高い値でした。特に信頼性の指摘率
着目した見直しを重点的に行ってください。
れぞれ標準値よりも低い値であり、問題ないと言えます。

品質特性観点ごとの評価

●信頼性
品質特性の中で、信頼性の指摘率が最も高い値でした。
標準値よりも高くなっており(88.2件/KS、標準値比107.6%)、信頼性の考慮が不足している可能性があります。動作異常
が残っていないか確認することを推奨します。
「1.2 信頼性に
ついては、「1.6
いて分析します。また、信頼性の中で最も高い指摘率であった「カード決済
て」で分析します。

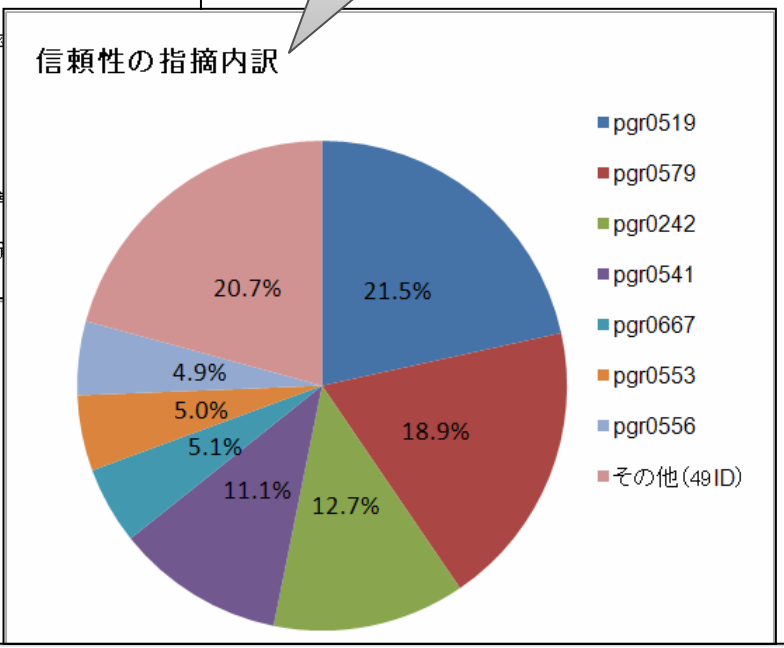
業務単位ごとの傾向グラフ

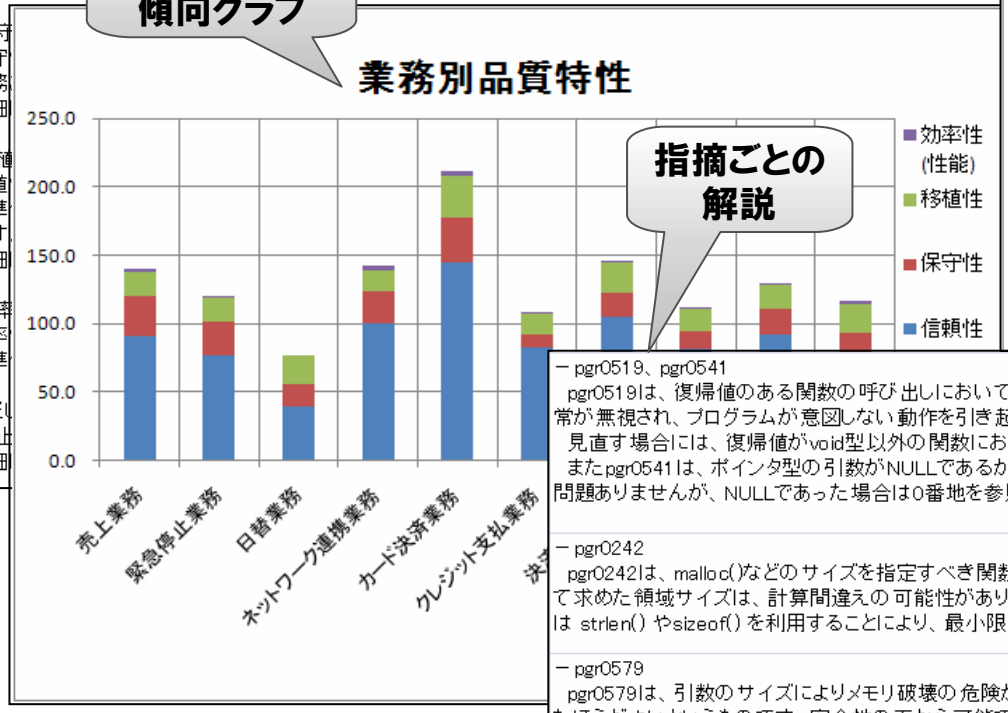
●保守
保守
業務
詳細

●移植
移植
標準
えます
詳細

●効率
効率
標準
ます。
ただし
急停止
詳細

品質特性ごとの指摘内訳グラフ





指摘ごとの解説

－ pgr0519、pgr0541
pgr0519は、復帰値のある関数の呼び出しにおいて復帰値を参照していないというものです。復帰値が異常を知らせるものであった場合、異常が無視され、プログラムが意図しない動作を引き起こす可能性があります。
見直す場合には、復帰値がvoid型以外の関数において、呼び出し結果を無視している箇所に着目してください。
またpgr0541は、ポインタ型の引数がNULLであるかチェックしないで引数を参照しているというものです。引数にアドレスが設定されていれば問題ありませんが、NULLであった場合は0番地を参照してしまいます。

－ pgr0242
pgr0242は、malloc()などのサイズを指定すべき関数の引数には定数ではなくstrlen() やsizeof() を利用すべきというものです。自分で計算して求めた領域サイズは、計算間違えの可能性があります。また、関連処理変更時にサイズの計算式を修正し忘れる可能性もあります。これらは strlen() やsizeof() を利用することにより、最小限に抑えることができます。

－ pgr0579
pgr0579は、引数のサイズによりメモリ破壊の危険がある関数(strcpyやsprintfなど)ではなく、危険性が低い関数(strncpyやsnprintf)を使ったほうがよいというものです。安全性の面から可能であれば修正を検討することを推奨します。

サービス基盤

FUJITSU

支えるのは・・・ コンパイラ開発で培った“プロの目”

◆ **確な指摘**

コンパイラ開発経験が豊富な、高い言語スキルを持つ専門技術者チームが診断

◆ **率的な解析**

静的解析ツール（PGRelief）開発経験者が、ツール活用ノウハウを駆使して解析
重要度が高く改善効果が大きい問題点を効率的に抽出

◆ **富な分析手法**

お客様指定観点と品質特性を絡め、さまざまな観点からわかりやすく解説
豊富なアプリ開発経験から、弱点や注力観点を分析

らに・・

サービス活用により、静的解析ツール導入費用、教育費用といったコストが不要
ツールセットアップ、解析、利用ノウハウ蓄積といった手間が不要

事例紹介

FUJITSU

対象プロジェクト

言語：C/C++言語
規模：1.4Mstep
工程：PT中

診断実施期間：11営業日
プロジェクト概要：金融営業系システム機能追加
開発形態：社外への委託開発

問題検出を効率化

問題の早期検出

弱点箇所の強化

ソースコード診断サービス

診断結果

バグ指摘　　：312件（27種）
バグの可能性：　63件（11種）
傾向分析　　：異常系処理に弱点
　　C++仕様理解不足の可能性

お客様の対処

全件修正（27種）
43件修正（7種）、既存資産分は保留
異常系処理の強化レビュー実施
指摘事項をPrj内に横展開

お客様のご感想

- ✓ PTで検出できなかったコーディングミスやレビュー漏れによるバグを効率的に検出できました
- ✓ バグを早期に検出できたことで、IT工程以降で手戻りが発生するといった品質リスクが減少しました
- ✓ テストでは見つけられない保守性、移植性の問題もわかり、今後の保守効率の向上が期待できます

公開情報、お問い合わせ先

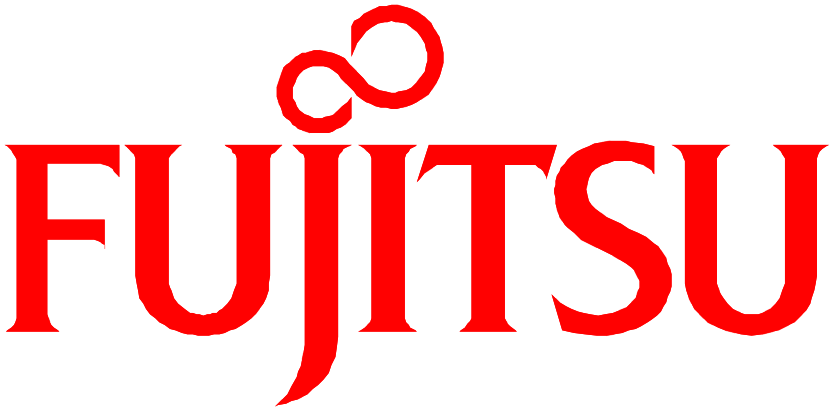
FUJITSU

公開Webサイト

http://jp.fujitsu.com/fst/services/pgr/

お問合せ先

株式会社富士通ソフトウェアテクノロジーズ
PGReliefお問い合わせ窓口
E-mail：fst-pgr-sales@cs.jp.fujitsu.com



THE POSSIBILITIES ARE INFINITE