「開発のプロとして」
静的解析ツールの徹底活用
～ PGRelief 2007 autumn ～

2008年1月31日

株式会社富士通ソフトウェアテクノロジーズ

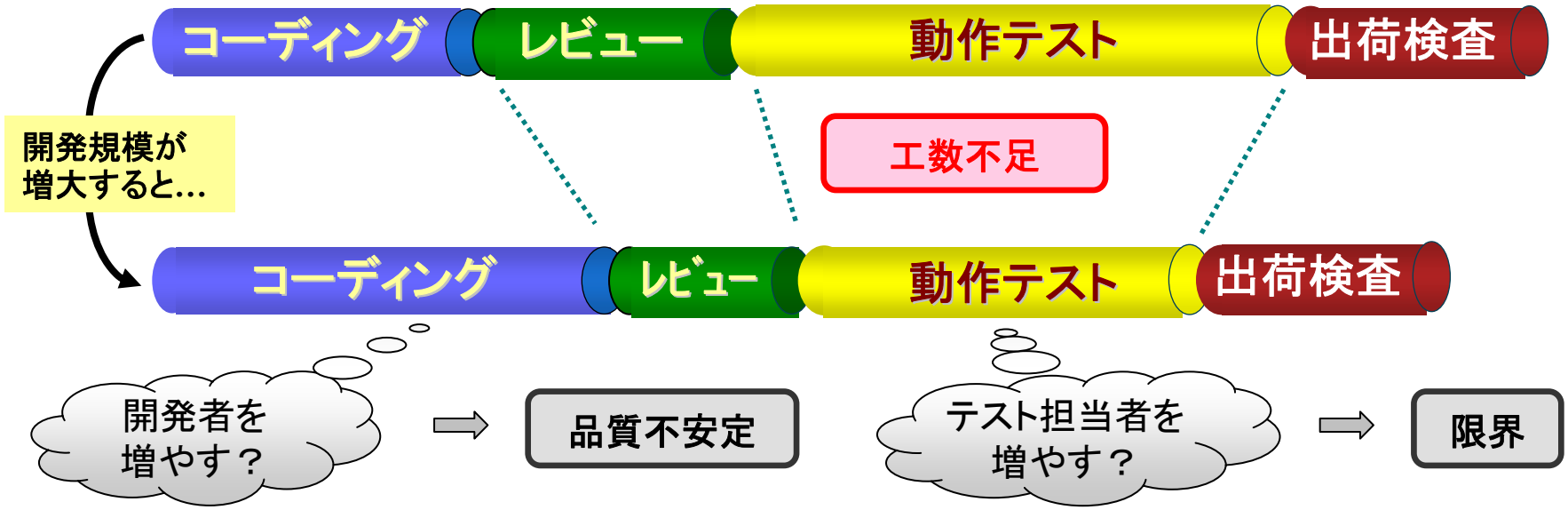
開発現場を取り巻く状況

FUJITSU

■ 課題

開発規模が増大・複雑化するなかで、如何にして
決められた期間内に品質を確保するか？

■ 悩み



高機能ソフトの短期開発に向けて

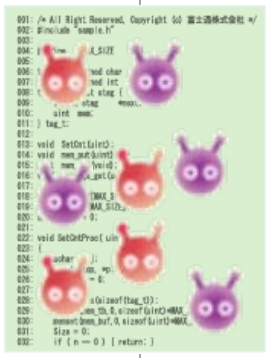
FUJITSU

コーディング・レビュー

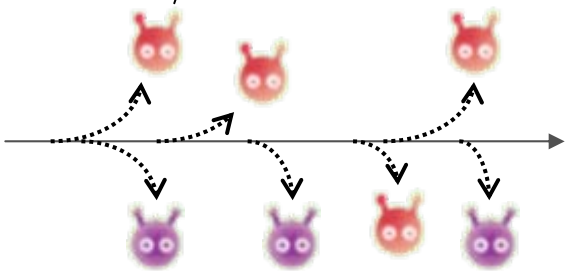
動作テスト・出荷検査

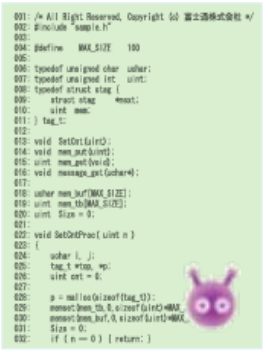
［従来］

目視レビュー



バグ原因の約4割はコーディングミス

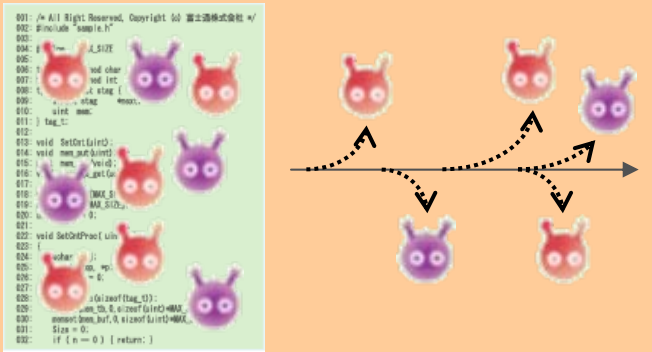




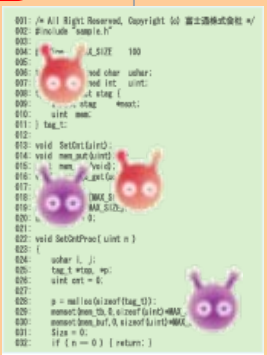
発生現象から原因追跡

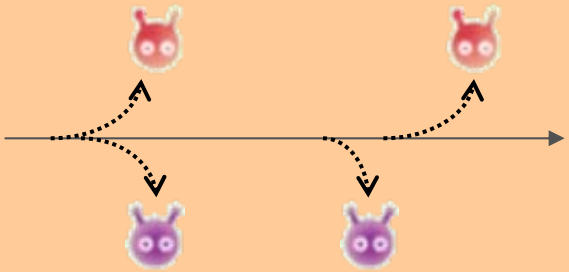
［今後］

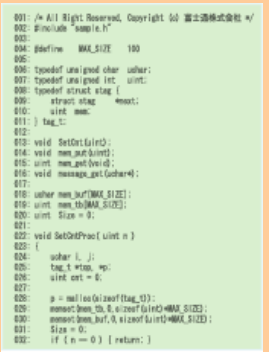
静的解析ツールを適用



コーディングミスを除去







2つのアプローチ

静的解析には、「ソースコードの書き方」を検証するアプローチと「論理的な誤り」を検証するアプローチがある。

チェック観点と品質向上効果		信頼性	移植性 保守性	その他	検証範囲	
書き方の検証	文法的な誤り	ー	ー	ー	PGRelief ※1	コンパイラ
	コーディング規約への適合性	○	◎	○		
論理的な検証	データ構造に対する検証	◎	ー	ー		
	関数内パス解析	◎	ー	ー		
	全パス解析	◎	ー	ー		

※1　将来対応を検討中

PGReliefとは

■ 概要

C/C++ソースコードを静的解析し、「データ構造」と「処理の流れ」に基づいて
バグを検出・指摘する品質向上支援ツール。

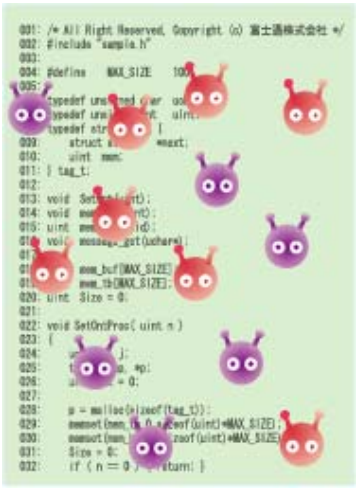
■ チェックの観点

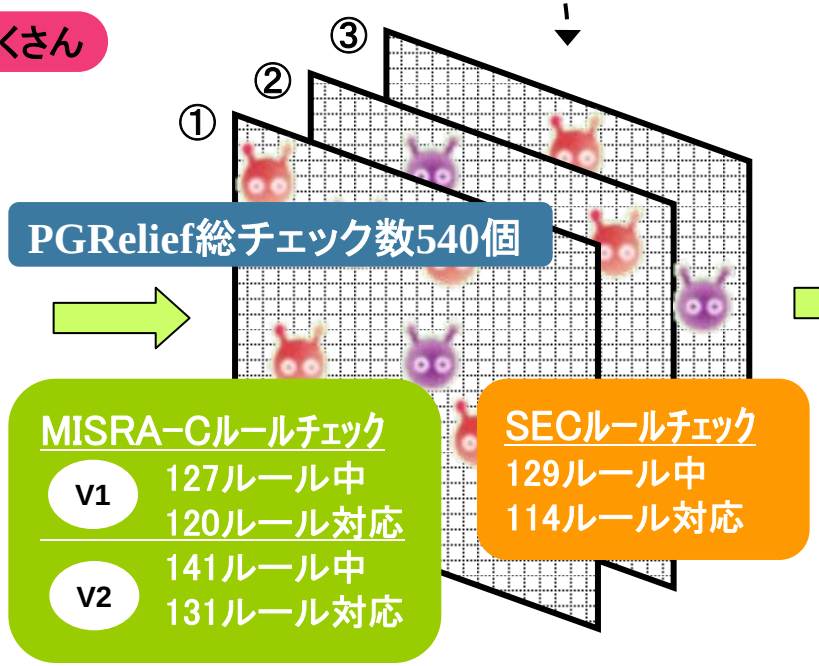
① 富士通社内の開発事例で培った独自ノウハウでチェック
② コーディング規約（SEC）への適合性をチェック
③ コーディング規約（MISRA-C）への適合性をチェック <オプション>

■ 特長

・ 問題箇所を的確に指摘
（冗長な指摘メッセージを出さない）
・ バグ検出率　平均2件／Kstep

ソースコードにはバグがたくさん





複数のチェックで品質アップ

指摘例(信頼性の向上)

FUJITSU

1) 領域破壊

```
1  struct  msg
2  {
3     char title[5];
4     char mode;
5   } *p;
～
9  strcpy (p->title, "12345");
```

pgr0532 strcpy (p->title, "12345")は領域 "p->title"を超えてコピーしてしまう可能性があります。(複写先：5, 複写元：6)

2) メモリリーク

```
1  void func (void)
2  {
3     char *p = malloc (10);
4     if (p == NULL) { return; }
5     err = process ( );
6     if (err == ERR) { return; }
7     free (p);
```

pgr0524 3行目の関数"malloc"で取得した資源が回収されていない可能性があります。

PGReliefの特長：関数内パス解析

論理的な意味も考慮し、問題箇所を的確に指摘。

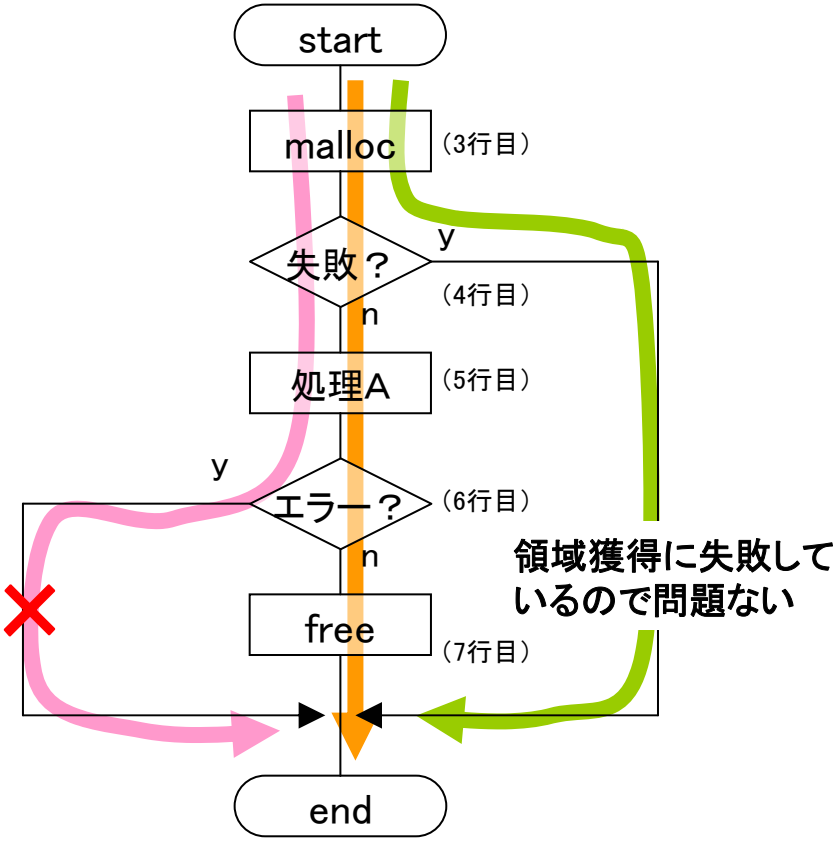
ソースコード例

```
1  void func (void)
2  {
3      char *p = malloc (10);
4      if (p == NULL) { return; }
5      err = process ( );
6      if (err == ERR) { return; }
7      free (p);
8  }
```

領域獲得に成功しているのに解放していない

pgr0524　3行目の関数"malloc"で取得した資源が回収されていない可能性があります。

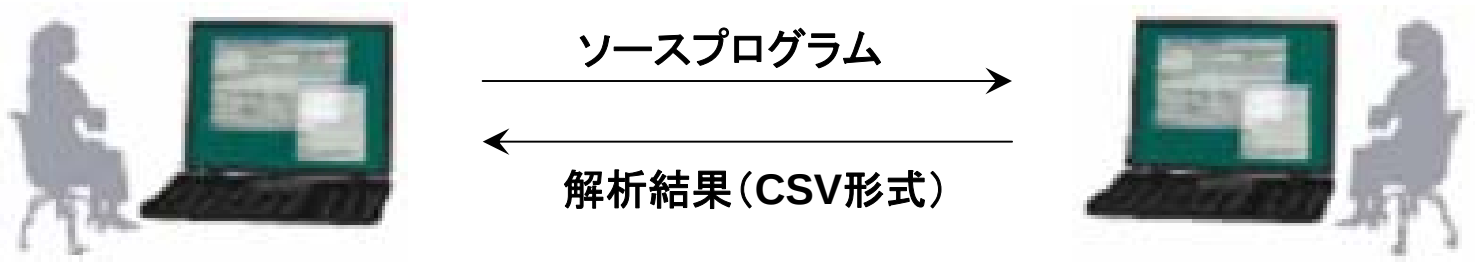
PGReliefの内部処理（ロジックフロー解析）



運用上の課題

開発者が自分で使うのが最も効果的。

でも一般的には、支援部門が静的解析ツールを使用し、
解析結果を開発者へフィードバックしている。



何故か？

- 解析するための設定が大変。
- メッセージが非常にたくさん出てくるので、とても開発時にやっていられない。

CSV形式での運用例

FUJITSU

支援部門の検証結果に基づいてソースを確認

［修正要の場合］　ソースを修正

［修正不要の場合］　ソース中にコメント「修正不要」を記述

［修正要否確認中の場合］

ソース中にコメント「後日検討」を記述

→ 2回目の解析時に行番号がずれるなど、
検証結果・対処結果の管理が困難。

→ 全体の品質状況が見えない。

例： ソースコード

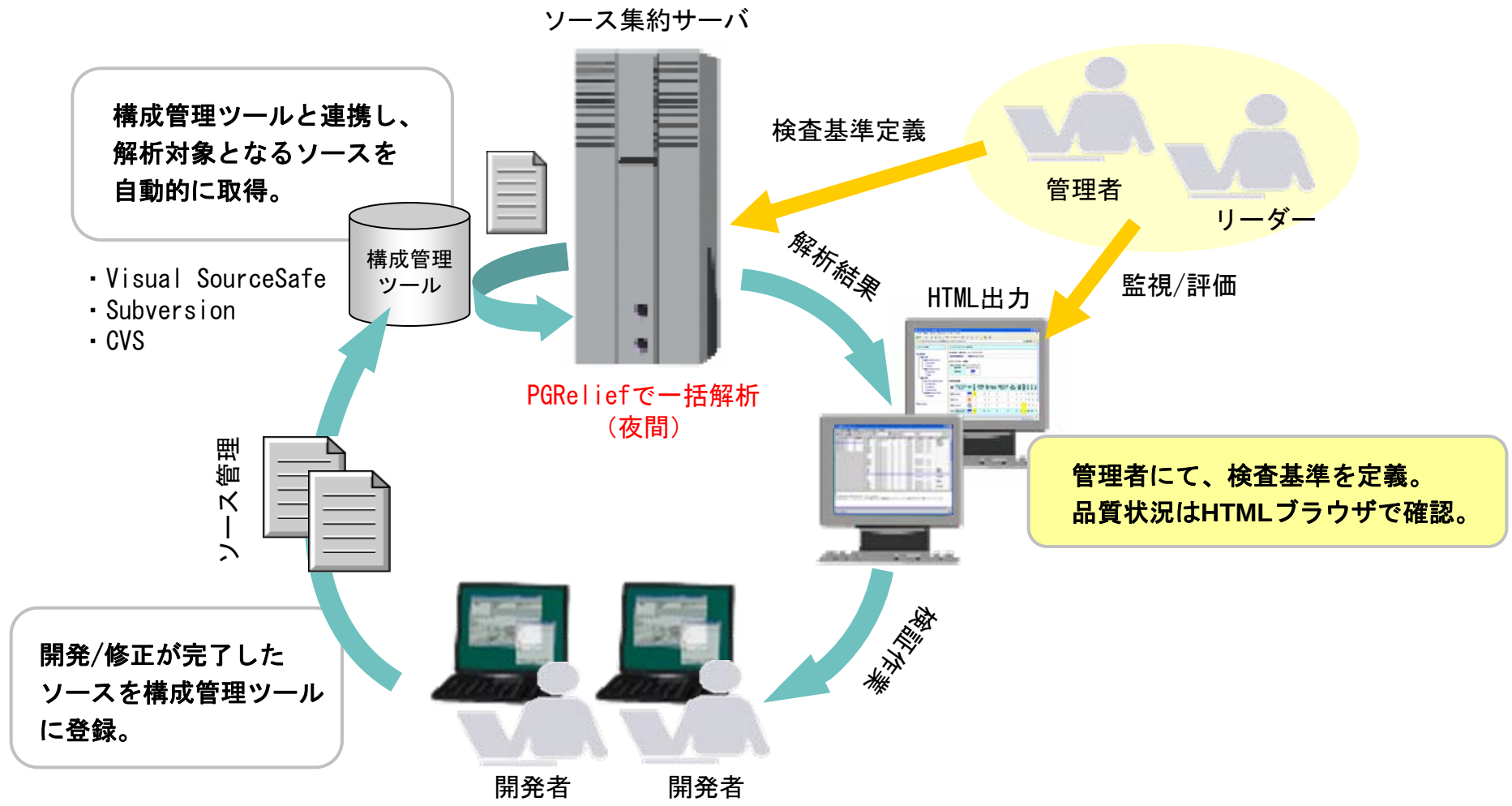
```
top = p;
for ( i = 0; i < n; i++ ) {  /* 修正不要 */
  p->mem = mem_get();
  g_Size += p->mem;  /* 修正不要 */
  p = p->next;
}
p->mem = 0;
p->next = NULL;

if ( g_Size > sizeof(mem_buf) ) {
  mem_put( p->mem ); /* 後日検討 */
}
```

例： CSV出力結果

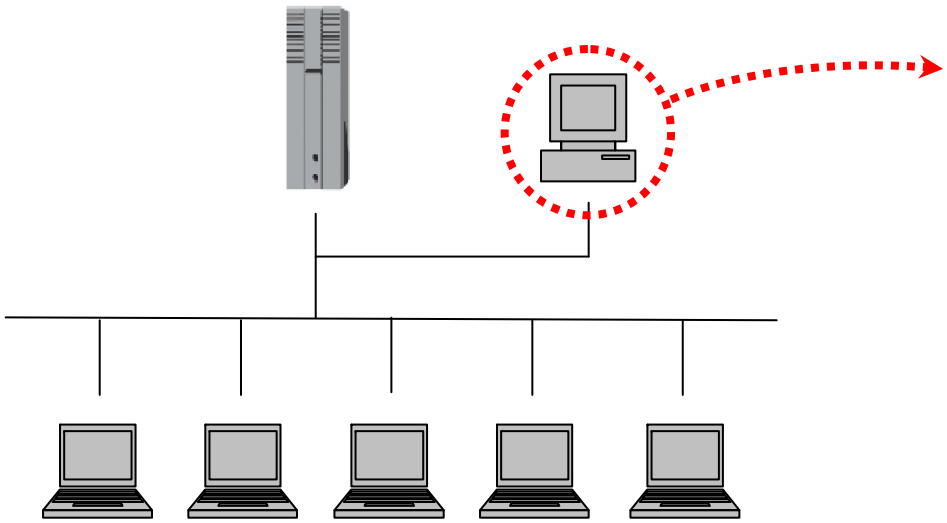
ファイル名	行番号	グループ	指摘ID	指摘メッセージ	検証結果	対処結果
sample.c	33	a	pgr0548	"memset(mem_buf, 0, sizeof(uint) * 100)"は領域 "mem_buf"の範囲を超え…	○	○
sample.c	36	a	pgr0524	28行目の関数"malloc"で取得した資源が回収されて…	○	―
sample.c	44	a	pgr0524	. . . . .	―	―
sample.c	50	a	pgr0522	. . . . .	○	
sample.c	63	a	pgr0695	. . . . .	○	○

PGRelief解決案：品質監視オプション



※　製品構成　：　フローティングライセンス　＋　品質監視オプション

サーバ側の運用イメージ



＜ 事前準備 ＞

1. デバイスドライバをインストール
2. ライセンスマネージャをインストール
3. フローティングライセンス版をインストール
4. 品質監視オプションをインストール
5. プロジェクト構成、解析オプションを定義

＜ 運用開始後 ＞

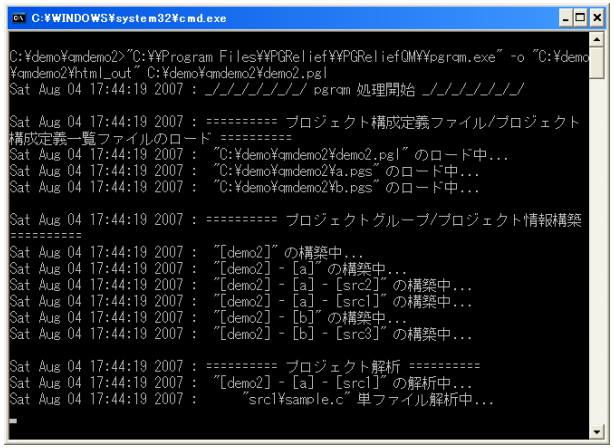
6. 夜間バッチ等で解析を実行
7. プロジェクト全体の品質状況を確認

（5. プロジェクト構成の記述例 ）

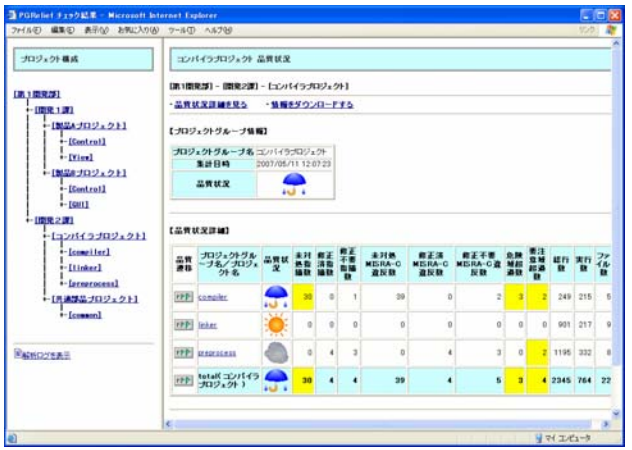
```
[SOURCES]
TARGET = C:¥demo2¥a
CODE = SJIS
CPLUSPLUSEXT = cpp

[OPERATECOMMONPRO…
OPTION = -KGNU/Solari…
RULEFILE = C:¥demo¥ch…
    :
    :
    :
```

（6. 夜間バッチの実行例 ）



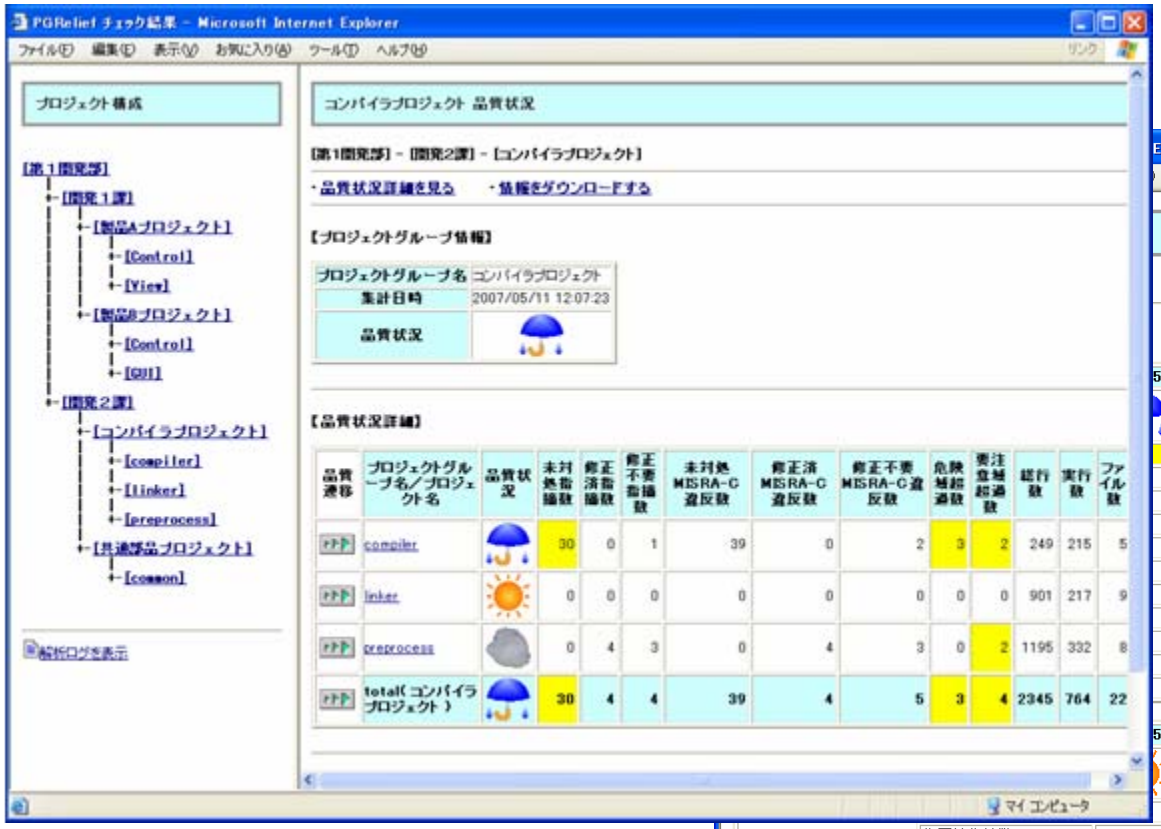
（7. 品質状況の確認例 ）



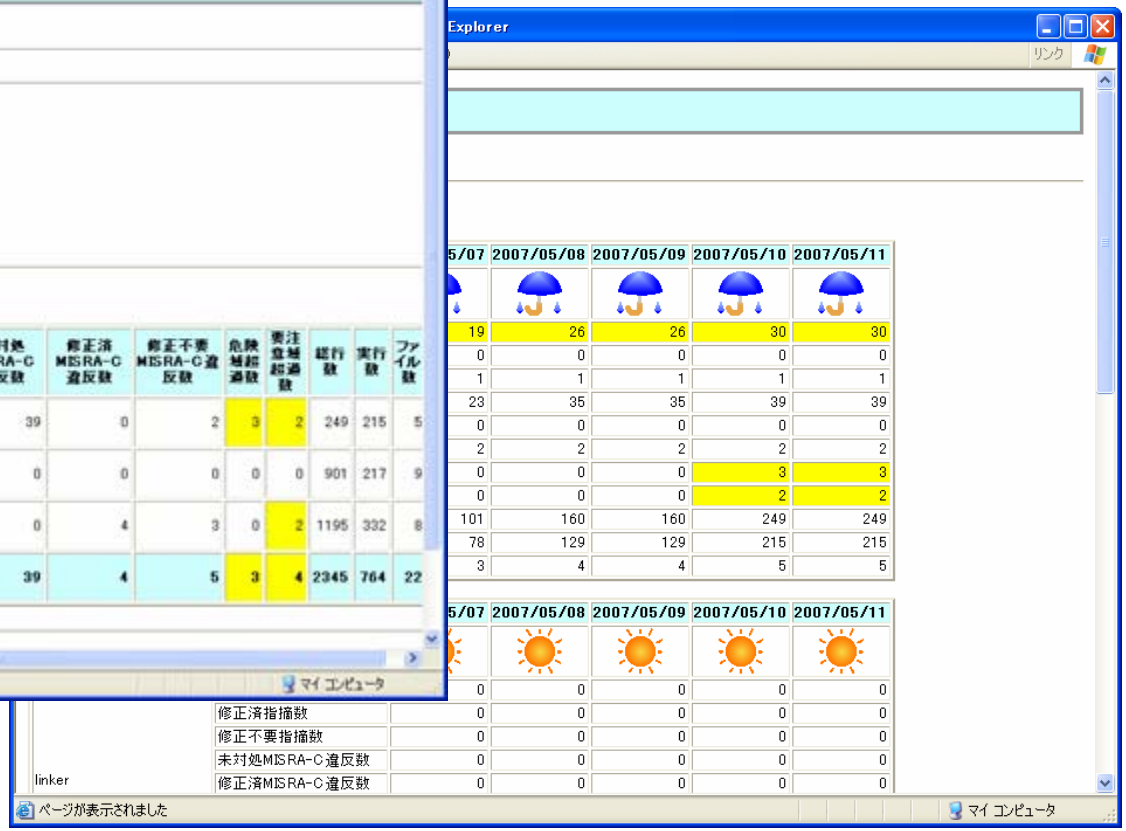
画面例（品質状況の確認）

プロジェクトごとの品質状況を可視化します。 問題箇所を容易に把握できます。

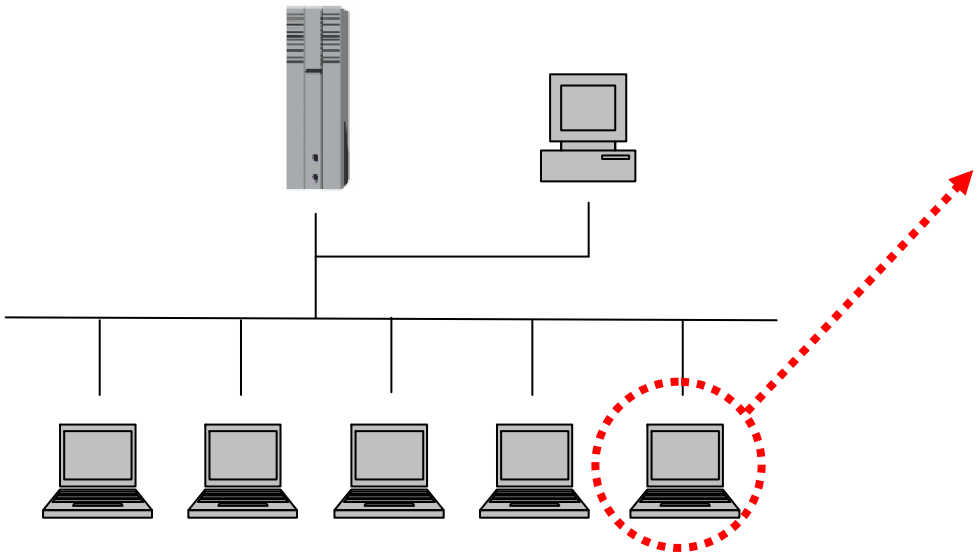
品質状況



品質遷移状況



開発者側の運用イメージ



＜ 事前準備 ＞

1. フローティングライセンス版をインストール

＜ 運用開始後 ＞

2. サーバ側での解析結果を確認
3. 対処結果を入力

（2. 解析結果の確認作業イメージ ）

（3. 対処結果の入力例 ）

画面例(解析結果確認、対処結果入力)

FUJITSU

ソースプログラムの問題箇所を確認し、対処結果を記録します。

公開情報、お問い合わせ

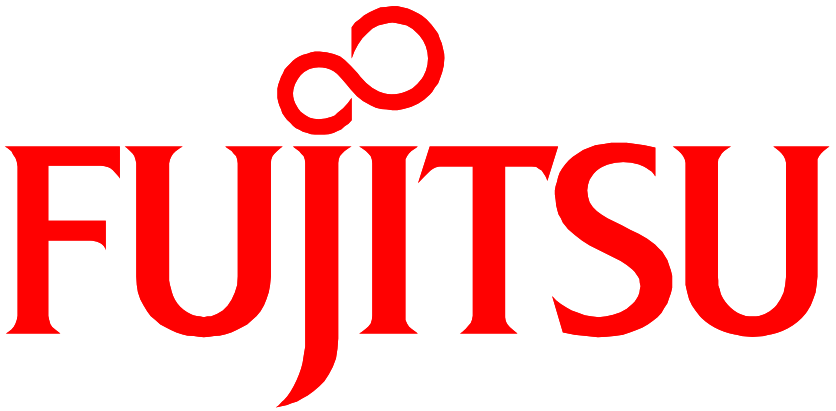
FUJITSU

公開Webサイト

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

技術的なご質問

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



THE POSSIBILITIES ARE INFINITE