

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

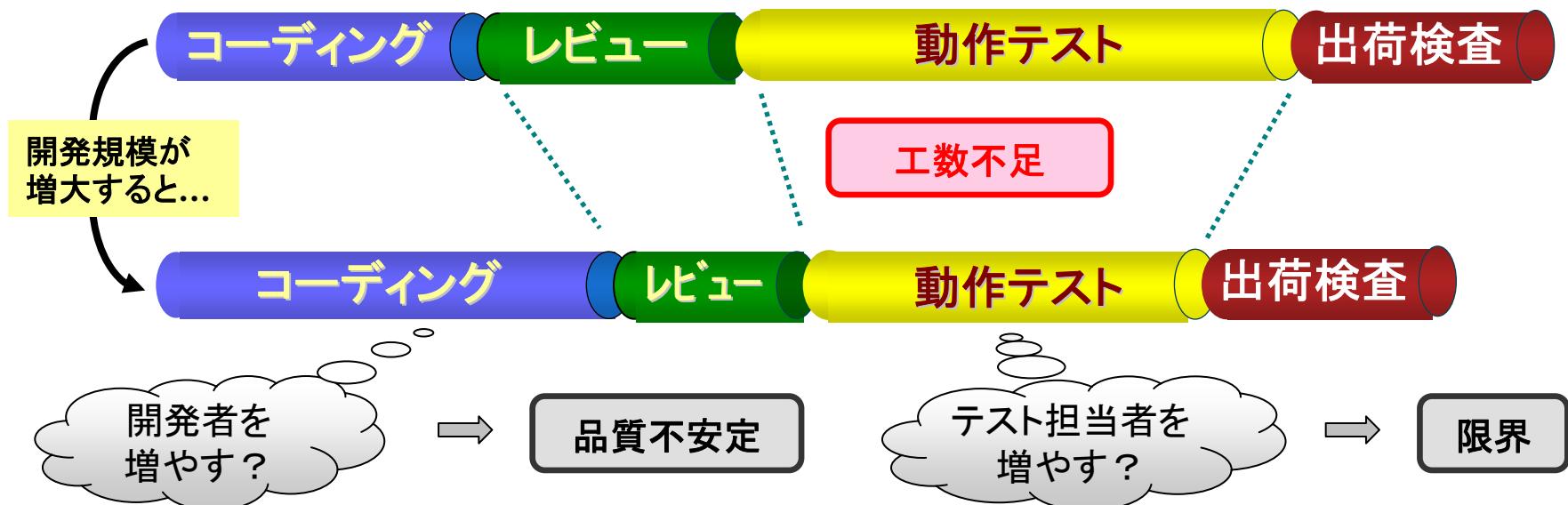
2008年1月31日

株式会社富士通ソフトウェアテクノロジーズ

■ 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

[従来]

```
001: /* All Right Reserved, Copyright (c) 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005: #define L_SIZE 10
006: typedef unsigned char uchar;
007: typedef unsigned int uint;
008: struct tag {
009:     uchar *tag;
010:     uint size;
011: } tag_t;
012:
013: void SetOrGet(uchar *tag, uint n);
014: void mem_set(uint);
015: uchar *mem_get();
016: uchar *mem_get(uint);
017: uchar *mem_getL(uint);
018: uchar *mem_getL(uint);
019: uchar *mem_getL(uint);
020: uchar *mem_getL(uint);
021: uchar *mem_getL(uint);
022: void SetOrGetL(uint n);
023: uchar *tag;
024: uchar *mp;
025: uint mp;
026: uint cmt = 0;
027:
028: p = m[0].mem_getL(tag, 0);
029: mem_set(mp, 0, cmt);
030: mem_set(mp, 0, cmt);
031: mem_set(mp, 0, cmt);
032: if (n == 0) { return; }
```

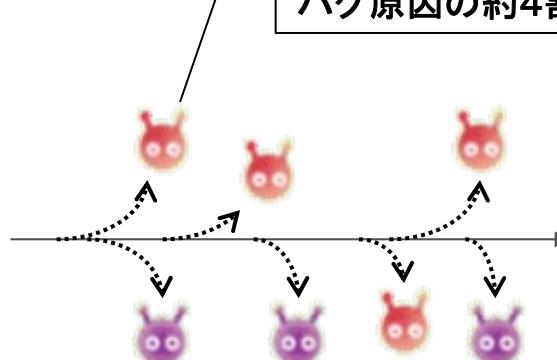


目視レビュー

```
001: /* All Right Reserved, Copyright (c) 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005: #define L_SIZE 10
006: typedef unsigned char uchar;
007: typedef unsigned int uint;
008: struct tag {
009:     uchar *tag;
010:     uint size;
011: } tag_t;
012:
013: void SetOrGet(uchar *tag, uint n);
014: void mem_set(uint);
015: uchar *mem_get();
016: uchar *mem_get(uint);
017: uchar *mem_getL(uint);
018: uchar *mem_getL(uint);
019: uchar *mem_getL(uint);
020: uchar *mem_getL(uint);
021: uchar *mem_getL(uint);
022: void SetOrGetL(uint n);
023: uchar *tag;
024: uchar *mp;
025: uint mp;
026: uint cmt = 0;
027:
028: p = m[0].mem_getL(tag, 0);
029: mem_set(mp, 0, cmt);
030: mem_set(mp, 0, cmt);
031: mem_set(mp, 0, cmt);
032: if (n == 0) { return; }
```

動作テスト・出荷検査

バグ原因の約4割はコーディングミス



```
001: /* All Right Reserved, Copyright (c) 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005: #define L_SIZE 10
006: typedef unsigned char uchar;
007: typedef unsigned int uint;
008: struct tag {
009:     uchar *tag;
010:     uint size;
011: } tag_t;
012:
013: void SetOrGet(uchar *tag, uint n);
014: void mem_set(uint);
015: uchar *mem_get();
016: uchar *mem_get(uint);
017: uchar *mem_getL(uint);
018: uchar *mem_getL(uint);
019: uchar *mem_getL(uint);
020: uchar *mem_getL(uint);
021: uchar *mem_getL(uint);
022: void SetOrGetL(uint n);
023: uchar *tag;
024: uchar *mp;
025: uint mp;
026: uint cmt = 0;
027:
028: p = m[0].mem_getL(tag, 0);
029: mem_set(mp, 0, cmt);
030: mem_set(mp, 0, cmt);
031: mem_set(mp, 0, cmt);
032: if (n == 0) { return; }
```

発生現象から原因追跡

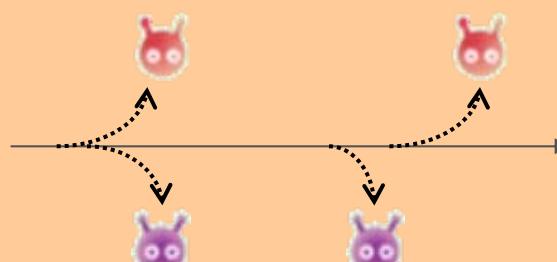
[今後]

静的解析ツールを適用

```
001: /* All Right Reserved, Copyright (c) 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005: #define L_SIZE 10
006: typedef unsigned char uchar;
007: typedef unsigned int uint;
008: struct tag {
009:     uchar *tag;
010:     uint size;
011: } tag_t;
012:
013: void SetOrGet(uchar *tag, uint n);
014: void mem_set(uint);
015: uchar *mem_get();
016: uchar *mem_get(uint);
017: uchar *mem_getL(uint);
018: uchar *mem_getL(uint);
019: uchar *mem_getL(uint);
020: uchar *mem_getL(uint);
021: uchar *mem_getL(uint);
022: void SetOrGetL(uint n);
023: uchar *tag;
024: uchar *mp;
025: uint mp;
026: uint cmt = 0;
027:
028: p = m[0].mem_getL(tag, 0);
029: mem_set(mp, 0, cmt);
030: mem_set(mp, 0, cmt);
031: mem_set(mp, 0, cmt);
032: if (n == 0) { return; }
```

コーディングミスを除去

```
001: /* All Right Reserved, Copyright (c) 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005: #define L_SIZE 10
006: typedef unsigned char uchar;
007: typedef unsigned int uint;
008: struct tag {
009:     uchar *tag;
010:     uint size;
011: } tag_t;
012:
013: void SetOrGet(uchar *tag, uint n);
014: void mem_set(uint);
015: uchar *mem_get();
016: uchar *mem_get(uint);
017: uchar *mem_getL(uint);
018: uchar *mem_getL(uint);
019: uchar *mem_getL(uint);
020: uchar *mem_getL(uint);
021: uchar *mem_getL(uint);
022: void SetOrGetL(uint n);
023: uchar *tag;
024: uchar *mp;
025: uint mp;
026: uint cmt = 0;
027:
028: p = m[0].mem_getL(tag, 0);
029: mem_set(mp, 0, cmt);
030: mem_set(mp, 0, cmt);
031: mem_set(mp, 0, cmt);
032: if (n == 0) { return; }
```



```
001: /* All Right Reserved, Copyright (c) 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005: #define L_SIZE 10
006: typedef unsigned char uchar;
007: typedef unsigned int uint;
008: struct tag {
009:     uchar *tag;
010:     uint size;
011: } tag_t;
012:
013: void SetOrGet(uchar *tag, uint n);
014: void mem_set(uint);
015: uchar *mem_get();
016: uchar *mem_get(uint);
017: uchar *mem_getL(uint);
018: uchar *mem_getL(uint);
019: uchar *mem_getL(uint);
020: uchar *mem_getL(uint);
021: uchar *mem_getL(uint);
022: void SetOrGetL(uint n);
023: uchar *tag;
024: uchar *mp;
025: uint mp;
026: uint cmt = 0;
027:
028: p = m[0].mem_getL(tag, 0);
029: mem_set(mp, 0, cmt);
030: mem_set(mp, 0, cmt);
031: mem_set(mp, 0, cmt);
032: if (n == 0) { return; }
```

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					検証範囲	
書き方の検証	文法的な誤り	—	—	—	PGRelief	コンパイラー
	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	◎	—	—	※1	
	関数内パス解析	◎	—	—		
	全パス解析	◎	—	—		

※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>

ソースコードにはバグがたくさん

```
001: /* All Right Reserved, Copyright (c) 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005:
006: typedef union tag_t {
007:     struct {
008:         uint32_t id;
009:         uint32_t next;
010:     } num;
011:     tag_t *tag;
012: } tag_t;
013:
014: void SetData(tag_t *t);
015: void move(tag_t *t);
016: void menu(tag_t *t);
017: void menu_get(char ch);
018:
019: const int MAX_SIZE;
020: const int MAX_SIZE;
021: uint32_t size = 0;
022:
023: void SetOrcras(tag_t *t);
024:
025: tag_t *ap;
026: int h = 0;
027:
028: ap = malloc(sizeof(tag_t));
029: menu(ap, h, 0);
030: menu(ap, h, 0);
031: size = 8;
032: if (h == 0) return;
```

PG Relief 総チェック数540個

MISRA-Cルールチェック	
V1	127ルール中 120ルール対応
V2	141ルール中 131ルール対応



SECルールチェック
129ルール中
114ルール対応

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

複数のチェックで品質アップ

```
001: /* All Right Reserved, Copyright (c) 善士株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005:
006: typedef struct {
007:     user;
008:     uint count;
009:     struc {
010:         uint max;
011    } tag_t;
012:
013: void SetCount(int);
014: void mem_alloc();
015: uint mem_free(void);
016: void message_get(char* );
017:
018: user mem_buf[MAX_SIZE];
019: uint mem_buf[Max_SIZE];
020: uint Size = 0;
021:
022: void SetCntrPrc( uint n )
023: {
024:     user i, j;
025:     tag_t *tmp, *ep;
026:     uint ent = 0;
027:
028:     p = malloc(sizeof(tag_t));
029:     memset(p, 0, sizeof(tag_t)*MAX_SIZE);
030:     memset(ep, 0, sizeof(tag_t)*MAX_SIZE);
031:     Size = 0;
032:     if ( n == 0 ) { return; }
```

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

FUJITSU

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

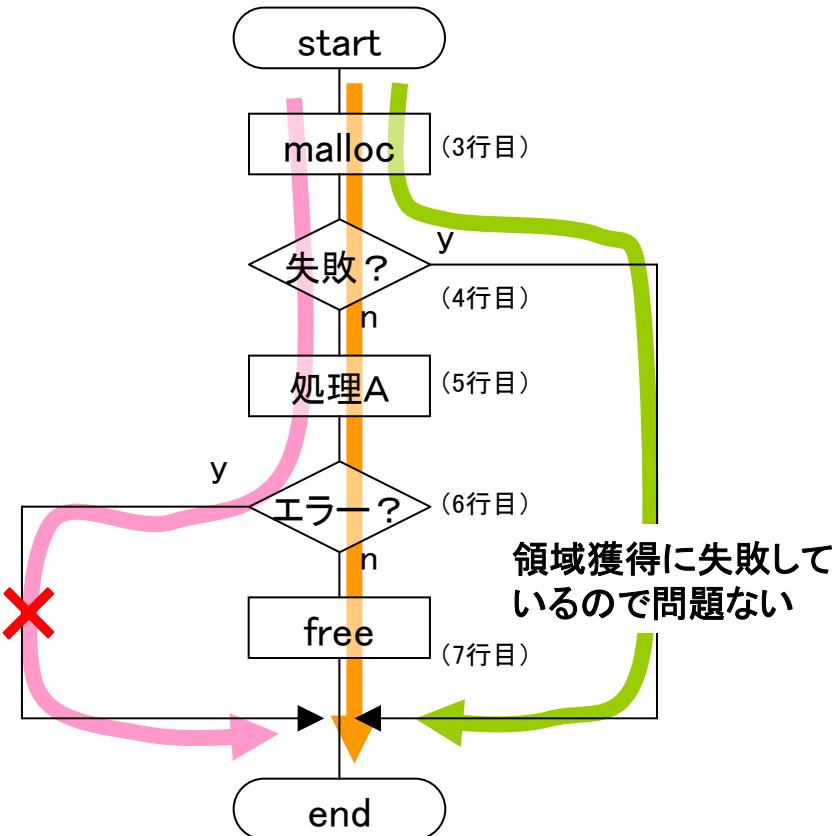
```
1 void func (void)
2 {
3     char *p = malloc (10);
4     if (p == NULL) { return; }
5     err = process ( );
6     if (err == ERR) { return; }
7     free (p);
8 }
```

領域獲得に成功している
のに解放していない



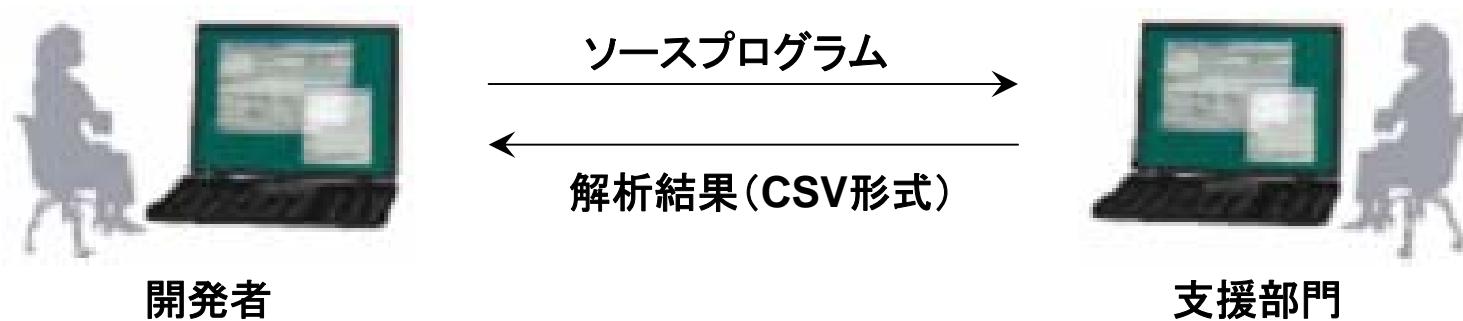
pgr0524 3行目の関数“malloc”で取得した資源が回収されていない可能性があります。

PGReliefの内部処理(ロジックフロー解析)



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

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;

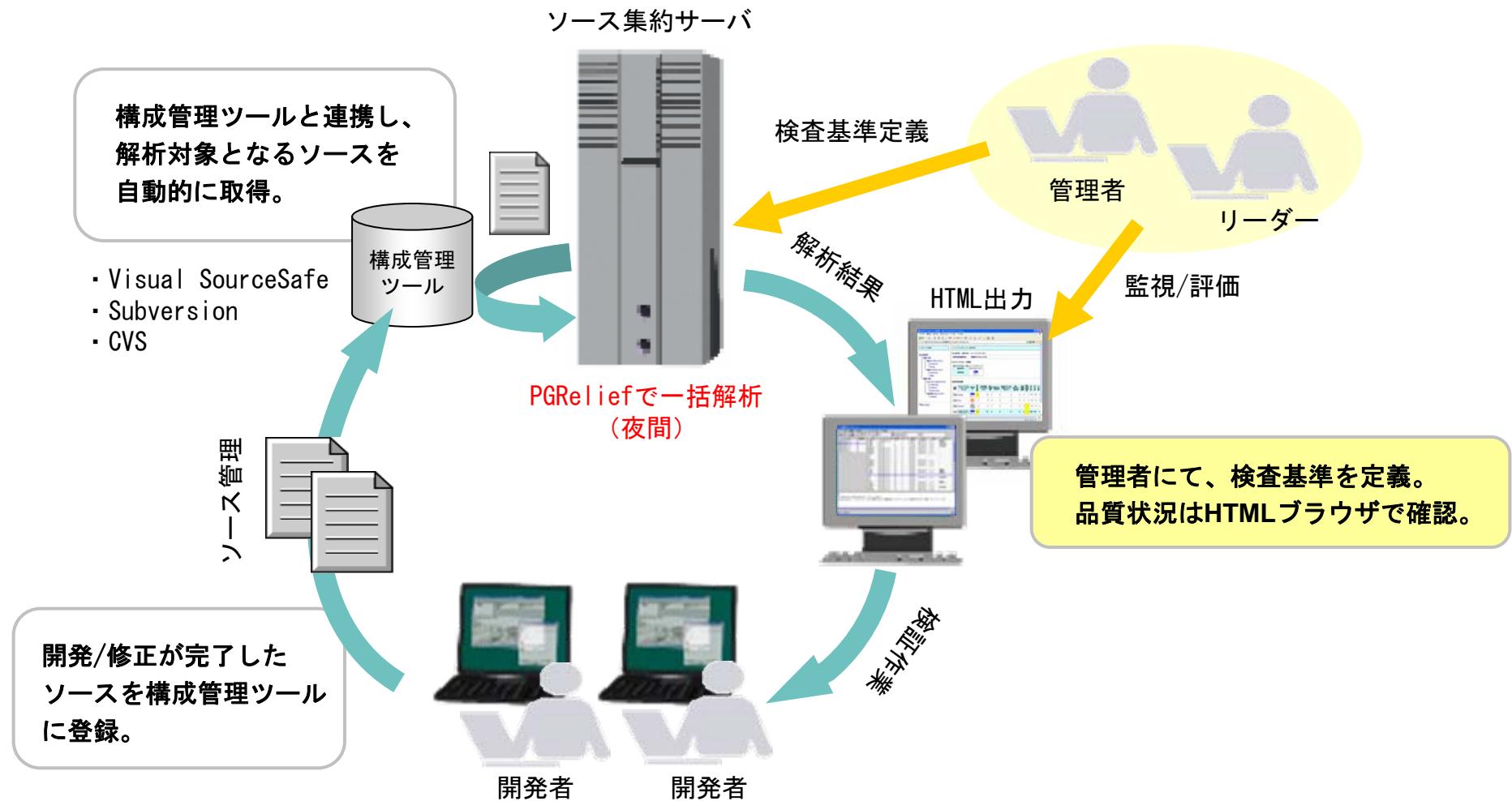
if ( g_Size > sizeof(mem_buf) ) {
    mem_put( p->mem ); /* 後日検討 */
}
```

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

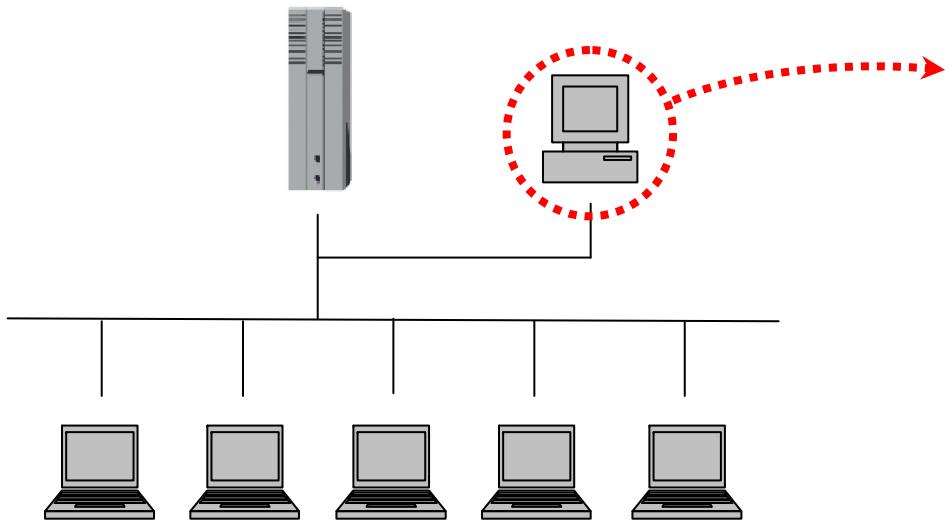
FUJITSU



※ 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

FUJITSU



＜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

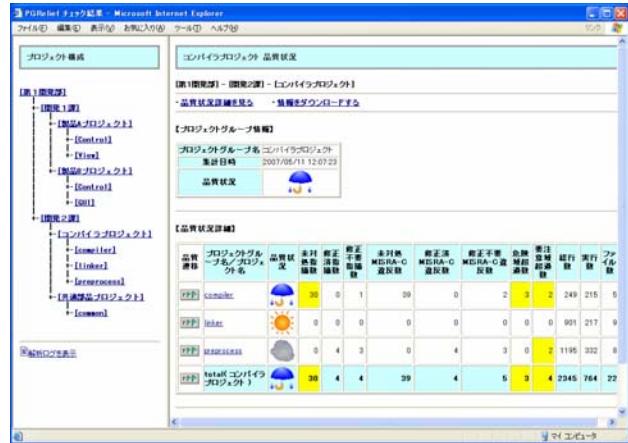
(5. プロジェクト構成の記述例)

```
[SOURCES]
TARGET = C:\demo2\aa
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

PGRelief チェック結果 - Microsoft Internet Explorer

ファイル(F) エディット(E) 表示(V) お気に入り(H) ツール(T) ヘルプ(H)

リンク

プロジェクト構成

【第1開発部】

- 【開発1課】
 - 【品質Aプロジェクト】
 - 【Control】
 - 【View】
 - 【品質Bプロジェクト】
 - 【Control】
 - 【GUI】
- 【開発2課】
 - 【コンパイラプロジェクト】
 - 【Compiler】
 - 【Linker】
 - 【Preprocessor】
 - 【普通部品プロジェクト】
 - 【Common】

コンパイラプロジェクト 品質状況

【第1開発部】 - 【開発2課】 - 【コンパイラプロジェクト】

- 品質状況詳細を見る - 故障をダウンロードする

【プロジェクトグループ情報】

プロジェクトグループ名	コンパイラプロジェクト
集計日時	2007/05/11 12:07:23
品質状況	

【品質状況詳細】

品質遷移	プロジェクトグループ名/プロジェクト名	品質状況	未対応指摘数	修正済指摘数	修正不要指摘数	未対応 MISRA-C 違反数	修正済 MISRA-C 違反数	修正不要 MISRA-C 違反数	危険指摘数	要注意指摘数	起行数	実行数	ファイル数
FTP	compiler		30	0	1	39	0	2	3	2	249	215	5
FTP	linker		0	0	0	0	0	0	0	901	217	9	
FTP	preprocess		0	4	3	0	4	3	0	2	1195	332	8
FTP	total(コンパイラプロジェクト)		30	4	4	39	4	5	3	4	2345	764	22

表示ログを表示

品質遷移状況

Explorer

5/07 2007/05/08 2007/05/09 2007/05/10 2007/05/11

19	26	26	30	30
0	0	0	0	0
1	1	1	1	1
23	35	35	39	39
0	0	0	0	0
2	2	2	2	2
0	0	0	3	3
0	0	0	2	2
101	160	160	249	249
78	129	129	215	215
3	4	4	5	5

5/07 2007/05/08 2007/05/09 2007/05/10 2007/05/11

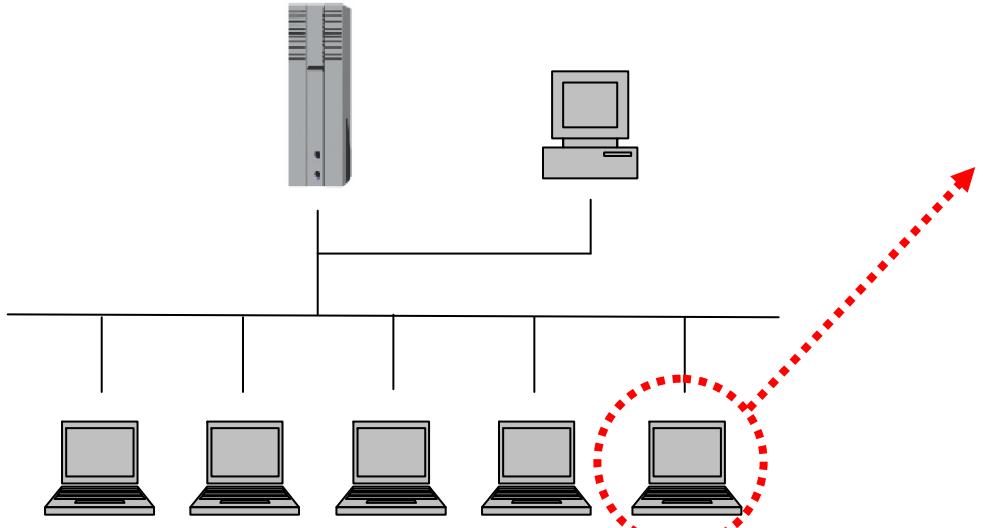
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
linker	修正済指摘数 修正不要指摘数 未対応 MISRA-C 違反数 修正済 MISRA-C 違反数	0 0 0 0	0 0 0 0	0 0 0 0

ページが表示されました

マイコンピュータ

開発者側の運用イメージ

FUJITSU



(2. 解析結果の確認作業イメージ)



＜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

(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

FUJITSU

THE POSSIBILITIES ARE INFINITE