

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

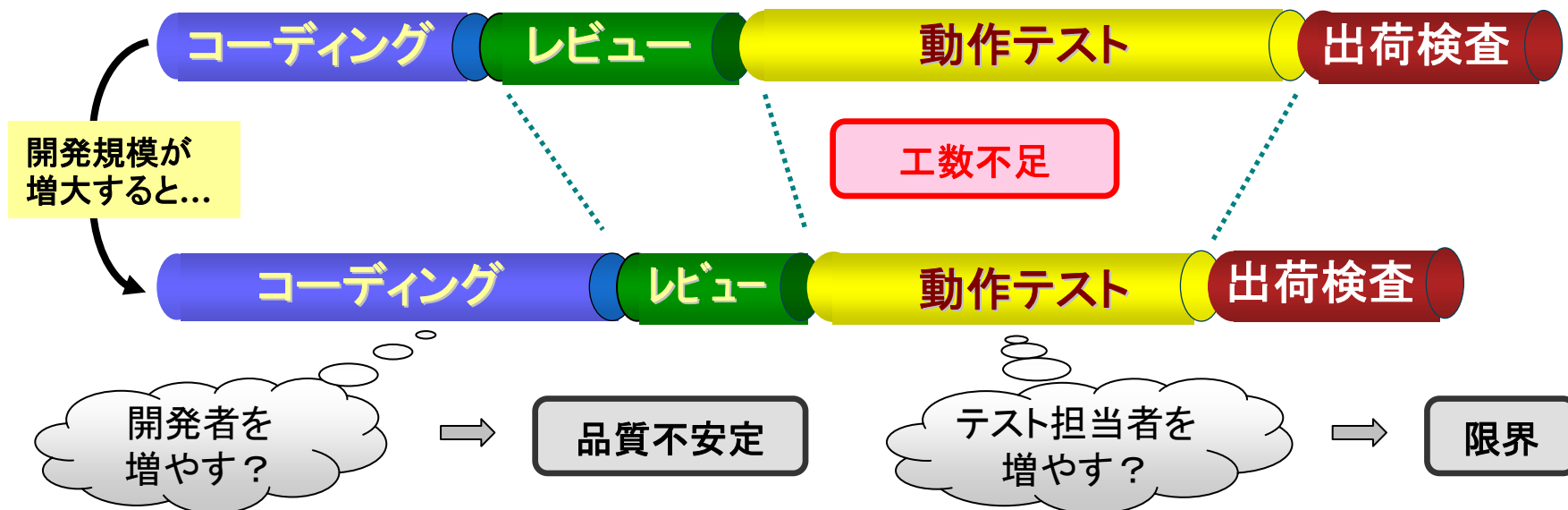
2008年1月31日

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



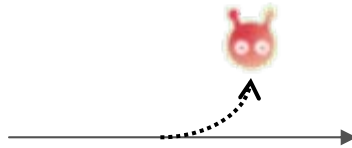
高機能ソフトの短期開発に向けて

コーディング・レビュー

動作テスト・出荷検査

[従来]

```
001: /* All Right Reserved, Copyright © 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005:
006: void read_char(char *str);
007: void read_int(int *i);
008: void read_string(char *str);
009: void write_message(char *msg);
010: void tag();
011:
012: void SetDefParam();
013: void MemGetParam();
014: void MemPutParam();
015: void MemGetParam();
016: void MemPutParam();
017:
018: int main()
019: {
020:     int i;
021:     SetDefParam();
022:     MemGetParam();
023:     MemPutParam();
024:     MemGetParam();
025:     MemPutParam();
026:     read_char(str);
027:     read_int(i);
028:     read_string(str);
029:     write_message(msg);
030:     tag();
031:     if (i == 0) { return; }
032: }
```



目視レビュー

```
001: /* All Right Reserved, Copyright © 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005:
006: void read_char(char *str);
007: void read_int(int *i);
008: void read_string(char *str);
009: void write_message(char *msg);
010: void tag();
011:
012: void SetDefParam();
013: void MemGetParam();
014: void MemPutParam();
015: void MemGetParam();
016: void MemPutParam();
017:
018: int main()
019: {
020:     int i;
021:     SetDefParam();
022:     MemGetParam();
023:     MemPutParam();
024:     MemGetParam();
025:     MemPutParam();
026:     read_char(str);
027:     read_int(i);
028:     read_string(str);
029:     write_message(msg);
030:     tag();
031:     if (i == 0) { return; }
032: }
```

バグ原因の約4割はコーディングミス



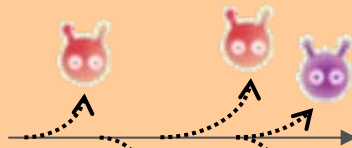
発生現象から原因追跡

```
001: /* All Right Reserved, Copyright © 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005:
006: void read_char(char *str);
007: void read_int(int *i);
008: void read_string(char *str);
009: void write_message(char *msg);
010: void tag();
011:
012: void SetDefParam();
013: void MemGetParam();
014: void MemPutParam();
015: void MemGetParam();
016: void MemPutParam();
017:
018: int main()
019: {
020:     int i;
021:     SetDefParam();
022:     MemGetParam();
023:     MemPutParam();
024:     MemGetParam();
025:     MemPutParam();
026:     read_char(str);
027:     read_int(i);
028:     read_string(str);
029:     write_message(msg);
030:     tag();
031:     if (i == 0) { return; }
032: }
```

[今後]

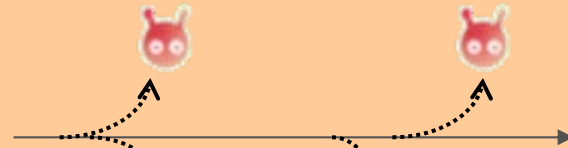
静的解析ツールを適用

```
001: /* All Right Reserved, Copyright © 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005:
006: void read_char(char *str);
007: void read_int(int *i);
008: void read_string(char *str);
009: void write_message(char *msg);
010: void tag();
011:
012: void SetDefParam();
013: void MemGetParam();
014: void MemPutParam();
015: void MemGetParam();
016: void MemPutParam();
017:
018: int main()
019: {
020:     int i;
021:     SetDefParam();
022:     MemGetParam();
023:     MemPutParam();
024:     MemGetParam();
025:     MemPutParam();
026:     read_char(str);
027:     read_int(i);
028:     read_string(str);
029:     write_message(msg);
030:     tag();
031:     if (i == 0) { return; }
032: }
```



コーディングミスを除去

```
001: /* All Right Reserved, Copyright © 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005:
006: void read_char(char *str);
007: void read_int(int *i);
008: void read_string(char *str);
009: void write_message(char *msg);
010: void tag();
011:
012: void SetDefParam();
013: void MemGetParam();
014: void MemPutParam();
015: void MemGetParam();
016: void MemPutParam();
017:
018: int main()
019: {
020:     int i;
021:     SetDefParam();
022:     MemGetParam();
023:     MemPutParam();
024:     MemGetParam();
025:     MemPutParam();
026:     read_char(str);
027:     read_int(i);
028:     read_string(str);
029:     write_message(msg);
030:     tag();
031:     if (i == 0) { return; }
032: }
```



```
001: /* All Right Reserved, Copyright © 富士通株式会社 */
002: #include "sample.h"
003:
004: #define MAX_SIZE 100
005:
006: void read_char(char *str);
007: void read_int(int *i);
008: void read_string(char *str);
009: void write_message(char *msg);
010: void tag();
011:
012: void SetDefParam();
013: void MemGetParam();
014: void MemPutParam();
015: void MemGetParam();
016: void MemPutParam();
017:
018: int main()
019: {
020:     int i;
021:     SetDefParam();
022:     MemGetParam();
023:     MemPutParam();
024:     MemGetParam();
025:     MemPutParam();
026:     read_char(str);
027:     read_int(i);
028:     read_string(str);
029:     write_message(msg);
030:     tag();
031:     if (i == 0) { return; }
032: }
```

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	
		信頼性	移植性 保守性	その他		
書き方の検証	文法的な誤り	—	—	—	PGRelief	コンパイル
	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	◎	—	—		
	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>

■ 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

```
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 struct {
007:     char ch;
008:     int i;
009: } tag_t;
010:
011: void SetInt(int n)
012: {
013:     tag_t tag;
014:     tag.i = n;
015: }
016:
017: void SetStr(char* str)
018: {
019:     tag_t tag;
020:     tag.ch = *str;
021: }
022:
023: void SetProc(int n)
024: {
025:     tag_t tag;
026:     tag.i = n;
027: }
028:
029: int main()
030: {
031:     tag_t tag;
032:     tag.i = 0;
033:     return 0;
034: }
```

PGRelief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

```
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 struct {
007:     char ch;
008:     int i;
009: } tag_t;
010:
011: void SetInt(int n)
012: {
013:     tag_t tag;
014:     tag.i = n;
015: }
016:
017: void SetStr(char* str)
018: {
019:     tag_t tag;
020:     tag.ch = *str;
021: }
022:
023: void SetProc(int n)
024: {
025:     tag_t tag;
026:     tag.i = n;
027: }
028:
029: int main()
030: {
031:     tag_t tag;
032:     tag.i = 0;
033:     return 0;
034: }
```

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。

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

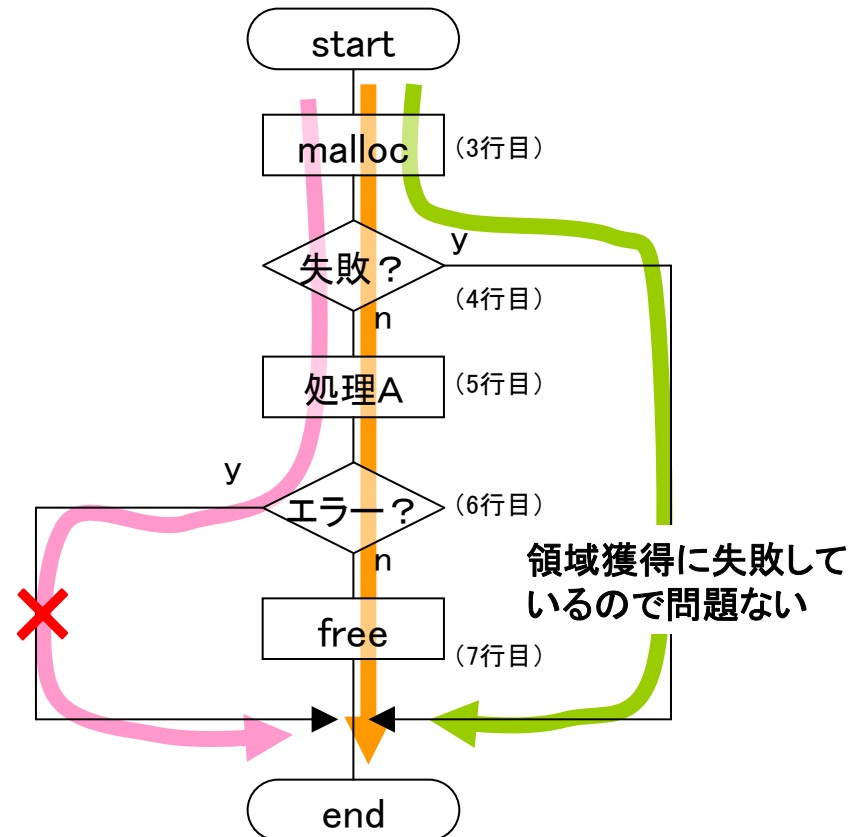
```
1 void func (void)
2 {
3   char *p = malloc (10);
4   if (p == NULL) { return; }
5   err = process ( );
6   if (err == ERR) { return; }
7   free (p);
8 }
```

領域獲得に成功しているのに解放していない



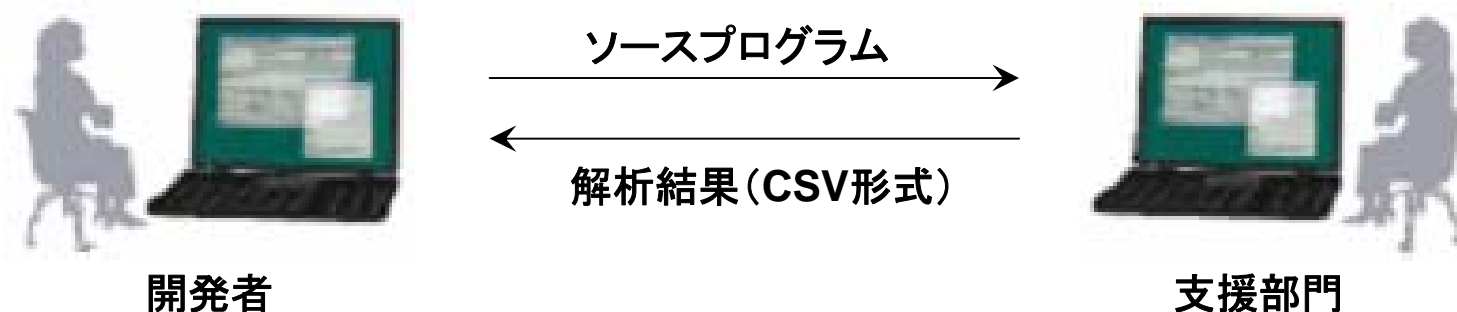
pgr0524 3行目の関数"malloc"で取得した資源が回収されていない可能性があります。

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。

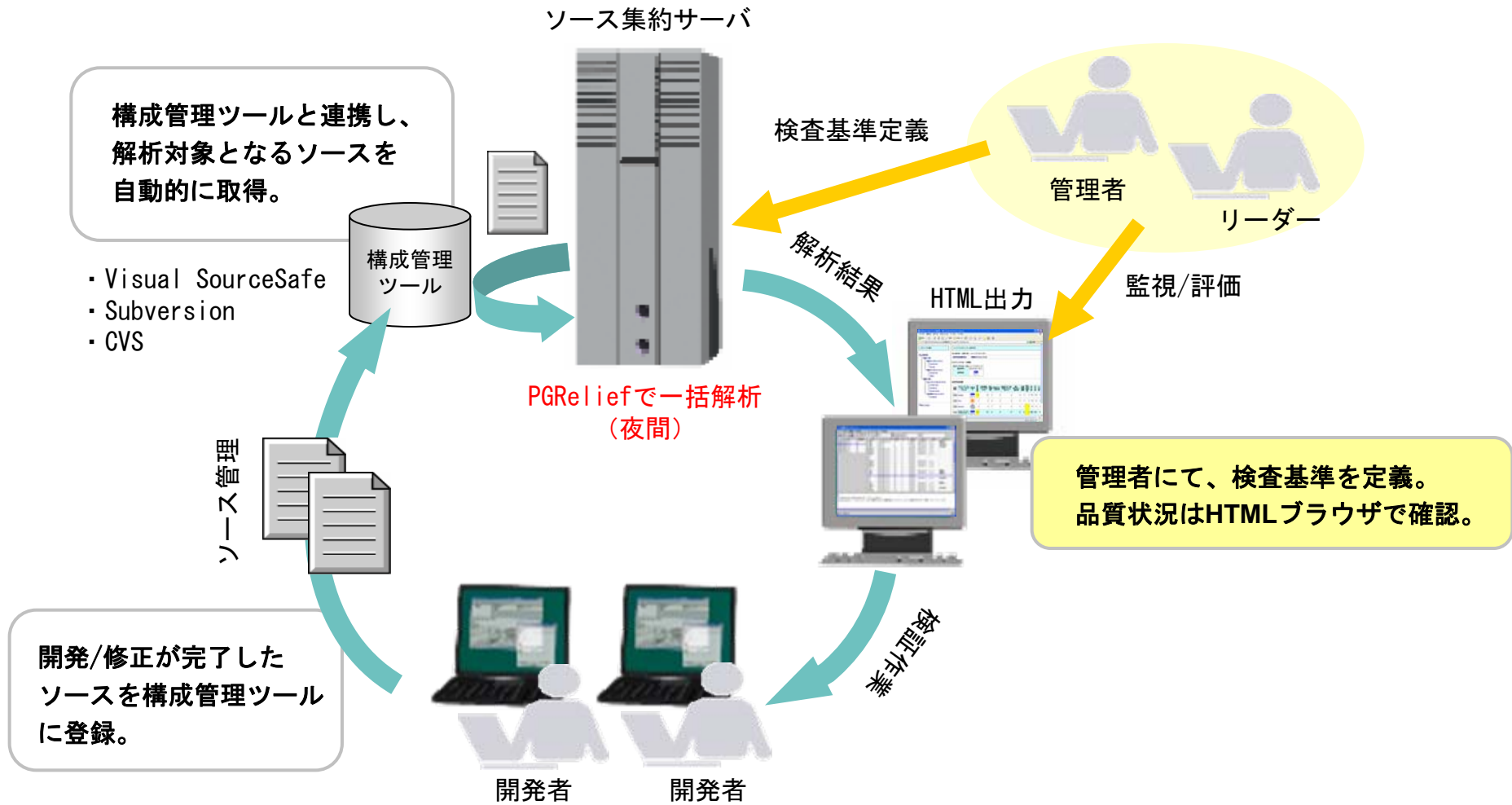
でも一般的には、支援部門が静的解析ツールを使用し、解析結果を開発者へフィードバックしている。



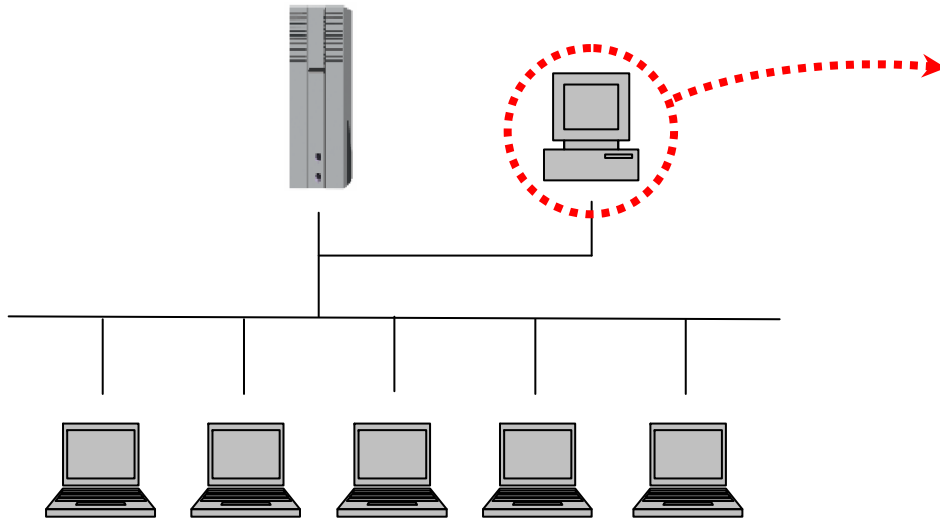
何故か？

- 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もらえない。

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



< 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)

```
C:\WINDOWS\system32\cmd.exe
C:\demo\demo2>"C:\Program Files\YPCRelief\YPCReliefQM\Ypgram.exe" -o "C:\demo\demo2\html_out" C:\demo\demo2\demo2.pg1
Sat Aug 04 17:44:19 2007 : /////////////// pgram 処理開始 ///////////////

Sat Aug 04 17:44:19 2007 : ===== プロジェクト構成定義ファイル/プロジェクト
構成定義一覧ファイルのロード =====
Sat Aug 04 17:44:19 2007 : "C:\demo\demo2\demo2.pg1" のロード中...
Sat Aug 04 17:44:19 2007 : "C:\demo\demo2\%a.pas" のロード中...
Sat Aug 04 17:44:19 2007 : "C:\demo\demo2\%b.pas" のロード中...

Sat Aug 04 17:44:19 2007 : ===== プロジェクトグループ/プロジェクト情報構築
=====
Sat Aug 04 17:44:19 2007 : "[demo2]" の構築中...
Sat Aug 04 17:44:19 2007 : "[demo2] - [a]" の構築中...
Sat Aug 04 17:44:19 2007 : "[demo2] - [a] - [src2]" の構築中...
Sat Aug 04 17:44:19 2007 : "[demo2] - [a] - [src1]" の構築中...
Sat Aug 04 17:44:19 2007 : "[demo2] - [b]" の構築中...
Sat Aug 04 17:44:19 2007 : "[demo2] - [b] - [src0]" の構築中...

Sat Aug 04 17:44:19 2007 : ===== プロジェクト解析 =====
Sat Aug 04 17:44:19 2007 : "[demo2] - [a] - [src1]" の解析中...
Sat Aug 04 17:44:19 2007 : "src1\sample.c" 単ファイル解析中...
```

(7. 品質状況の確認例)

品質 課題	品質 課題 名	品質 状態	実行 回数	修正 回数	未対応 回数	修正済 回数	未対応 回数	修正済 回数	色検 回数	色検 回数	実行 回数	実行 回数
YPP	sample.c	黄色	0	1	39	0	2	3	249	215	5	
YPP	html	黄色	0	0	0	0	0	0	0	0	901	217
YPP	sample.pas	黄色	0	4	3	0	4	3	0	2	1195	302
YPP	total (クライアント プロジェクト)	黄色	39	4	4	39	4	5	3	4	2345	764

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

コンパイルプロジェクト 品質状況

【第1開発部】 - 【開発2課】 - 【コンパイルプロジェクト】

・品質状況詳細を見る ・指標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 コンパイルプロジェクト
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	未対処指摘数	修正済指摘数	修正不要指摘数	未対処 MISRA-C 違反数	修正済 MISRA-C 違反数	修正不要 MISRA-C 違反数	危険超過数	要注意超過数	総行数	実行数	ファイル数
合計	compiler		30	0	1	39	0	2	3	2	249	215	5
合計	linker		0	0	0	0	0	0	0	0	901	217	9
合計	preprocess		0	4	3	0	4	3	0	2	1195	332	8
合計	total(コンパイルプロジェクト)		30	4	4	39	4	5	3	4	2345	764	22

品質遷移状況

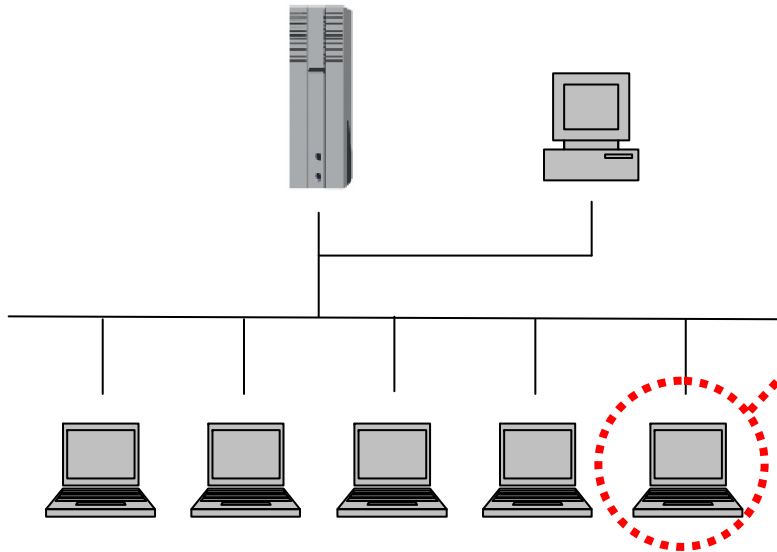
	2007/05/07	2007/05/08	2007/05/09	2007/05/10	2007/05/11
品質状況					
未対処指摘数	19	26	26	30	30
修正済指摘数	0	0	0	0	0
修正不要指摘数	1	1	1	1	1
未対処 MISRA-C 違反数	23	35	35	39	39
修正済 MISRA-C 違反数	0	0	0	0	0
修正不要 MISRA-C 違反数	2	2	2	2	2
危険超過数	0	0	0	3	3
要注意超過数	0	0	0	2	2
総行数	101	160	160	249	249
実行数	78	129	129	215	215
ファイル数	3	4	4	5	5

	2007/05/07	2007/05/08	2007/05/09	2007/05/10	2007/05/11
品質状況					
未対処指摘数	0	0	0	0	0
修正済指摘数	0	0	0	0	0
修正不要指摘数	0	0	0	0	0
未対処 MISRA-C 違反数	0	0	0	0	0
修正済 MISRA-C 違反数	0	0	0	0	0

linker

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

ページが表示されました



< 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

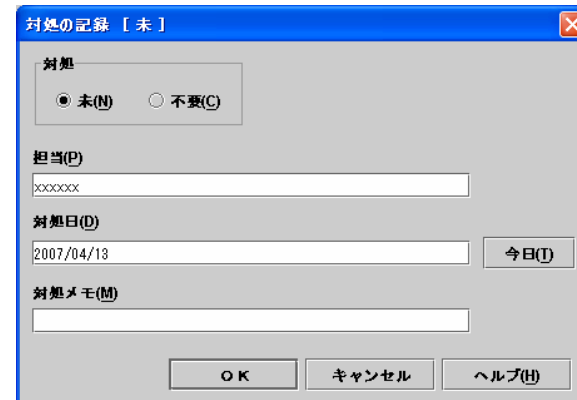
< 運用開始後 >

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）

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。

ファイル名 ^	総指摘数	対処	ファイル名	行番号 ^	グループ	指摘ID	対処日	担当	対処メモ
SAMPLEMAIN_01.C	64	不要	SampleMain_02.c	3	!	pgr0401	2007/03/31	xxxxxxx	問題なし
SAMPLEMAIN_02.C	64	不要	SetCnt.h	4	ε	pgr0705	2007/03/10	xxxxxxx	将来検討
SAMPLEMAIN_03.C	64	不要	SetCnt.h	4	ε	pgr0794	2007/03/10	xxxxxxx	将来検討
SAMPLEMAIN_04.C	64	修正済	SampleMain_02.c	15	b	pgr0638	2007/03/31	xxxxxxx	
SAMPLEMAIN_05.C	64	修正済	SampleMain_02.c	16	ε	pgr0794	2007/03/31	xxxxxxx	
		修正済	SampleMain_02.c	16	b	pgr0638	2007/03/31	xxxxxxx	
		未	SampleMain_02.c	22	ε	pgr0719			
		未	SampleMain_02.c	22	b	pgr0733			
		修正済	SampleMain_02.c	22	ε	pgr0705	2007/04/13	xxxxxxx	
		修正済	SampleMain_02.c	24	ε				
		修正済	SampleMain_02.c	25	ε				
		未	SampleMain_02.c	28	b				
		不要	SampleMain_02.c	28	b				
		不要	SampleMain_02.c	29	b				
		未	SampleMain_02.c	30	a				
		未	SampleMain_02.c	31	ε				
		未	SampleMain_02.c	32	a				
		不要	SampleMain_02.c	32	ε				
		修正済	SampleMain_02.c	32	a				
		不要	SampleMain_02.c	35	ε				
		修正済	SampleMain_02.c	35	a				

未
"F:#demo#sample_new#SampleMain_02.c" (30): a pgr0548
"memset(mem_buf, 0, sizeof(uint) * 100)"は領域"mem_buf"の範囲を超えてアクセスしてしまう

Project1 未 / 不要 / 修正済

対処の記録 [未]

対処

未(N) 不要(C)

担当(P)

xxxxxxx

対処日(D)

2007/04/13 今日(T)

対処メモ(M)

OK キャンセル ヘルプ(H)

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